J. Cryptol. (2009) 22: 161–188
DOI: 10.1007/s00145-008-9036-8

Journal of
CRYPTOLOGY

A Proof of Security of Yao's Protocol for Two-Party Computation

Yehuda Lindell[*]
Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel
lindell@cs.biu.ac.il

Benny Pinkas[†]
Department of Computer Science, Haifa University, Haifa 31905, Israel
benny@pinkas.net

Communicated by Dan Boneh.

Online publication 9 December 2008

Abstract. In the mid 1980s, Yao presented a constant-round protocol for securely computing any two-party functionality in the presence of semi-honest adversaries (FOCS 1986). In this paper, we provide a complete description of Yao's protocol, along with a rigorous proof of security. Despite the importance of Yao's protocol to the theory of cryptography and in particular to the field of secure computation, to the best of our knowledge, this is the first time that an explicit proof of security has been published.

Key words. Secure two-party computation, Semi-honest adversaries, Yao's two-party protocol, Proofs of security

1. Introduction

In the setting of two-party computation, two parties with respective private inputs x and y wish to jointly compute a functionality $f(x,y)=(f_1(x,y),f_2(x,y))$ such that the first party receives $f_1(x,y)$ and the second party receives $f_2(x,y)$. This functionality may be probabilistic, in which case $f(x,y)$ is a random variable. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*) and that the output is distributed according to the prescribed functionality (*correctness*). The definition of security that has become standard today [1,4,10,11] blends these two conditions. In this paper, we consider the problem of achieving security in the presence of *semi-honest* (or passive) adversaries who follow the protocol specification but attempt to learn additional information by analyzing the transcript of messages received during the execution.

The first general solution for the problem of secure two-party computation in the presence of semi-honest adversaries was presented by Yao [15]. Later, solutions were

[*] Most of this work was carried out while at IBM T.J. Watson Research, New York.

[†] Most of this work was carried out while at HP Labs, New Jersey.

provided for the multi-party and malicious adversarial cases by Goldreich et al. [9]. These ground-breaking results essentially began the field of secure multiparty computation and served as the basis for countless papers. In addition to its fundamental theoretic contribution, Yao's protocol is remarkably efficient in that it has only a *constant number of rounds* and uses one oblivious transfer per input bit only (with no additional oblivious transfers in the rest of the computation). Unfortunately, to the best of our knowledge, a full proof of security of Yao's protocol has never been published. Our motivation for publishing such a proof is twofold. First, Yao's result is central to the field of secure computation. This is true both because of its historic importance as the first general solution to the two-party problem, and because many later results have relied on it in their constructions. As such, having a rigorous proof of the result is paramount. Second, the current situation is very frustrating for those who wish to study secure multiparty computation but are unable to find a complete presentation of one of the most basic results in the field. We hope to correct this situation in this paper.

Yao's Protocol [15] Let f be a polynomial-time functionality (assume for now that it is deterministic), and let x and y be the parties' respective inputs. The first step is to view the function f as a boolean circuit C. In order to describe Yao's protocol, it is helpful to first recall how such a circuit is computed. Let x and y be the parties' inputs. Then, the circuit $C(x, y)$ is computed gate-by-gate, from the input wires to the output wires. Once the incoming wires to a gate g have obtained values $\alpha, \beta \in \{0, 1\}$, it is possible to give the outgoing wires of the gate the value $g(\alpha, \beta)$. The output of the circuit is given by the values obtained in the output wires of the circuit. Thus, essentially, computing a circuit involves allocating appropriate zero-one values to the *wires* of the circuit. In the description below, we refer to four different types of wires in a circuit: circuit-input wires (that receive the input values x and y), circuit-output wires (that carry the value $C(x, y)$), gate-input wires (that enter some gate g), and gate-output wires (that leave some gate g).

We now present a high-level description of Yao's protocol. The construction is actually a "compiler" that takes any polynomial-time functionality f, or actually a circuit C that computes f, and constructs a protocol for securely computing f in the presence of semi-honest adversaries. In a secure protocol, the only value learned by a party should be its output. Therefore, the values that are allocated to all wires that are not circuit-output should not be learned by either party (these values may reveal information about the other party's input that could not be otherwise learned from the output). The basic idea behind Yao's protocol is to provide a method of computing a circuit so that values obtained on all wires other than circuit-output wires are never revealed. For every wire in the circuit, two random values are specified such that one value represents 0 and the other represents 1. For example, let w be the label of some wire. Then, two values k_w^0 and k_w^1 are chosen, where k_w^σ represents the bit σ. An important observation here is that even if one of the parties knows the value k_w^σ obtained by the wire w, this does not help it to determine if $\sigma = 0$ or $\sigma = 1$ (because both k_w^0 and k_w^1 are identically distributed). Of course, the difficulty with such an idea is that it seems to make computation of the circuit impossible. That is, let g be a gate with incoming wires w_1 and w_2 and output wire w_3. Then, given two random values k_1^σ and k_2^τ, it does not seem possible to compute the gate because σ and τ are unknown. We therefore need a method of computing

Table 1. Garbled OR gate

Input wire w_1	Input wire w_2	Output wire w_3	Garbled computation table
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	k_3^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

the value of the output wire of a gate (also a random value k_3^0 or k_3^1), given the value of the two input wires to that gate. In short, this method involves providing "garbled computation tables" that map the random input values to random output values. However, this mapping should have the property that given two input values, it is only possible to learn the output value that corresponds to the output of the gate (the other output value must be kept secret). This is accomplished by viewing the four possible inputs to the gate $k_1^0, k_1^1, k_2^0, k_2^1$ as encryption keys. Then, the output values k_3^0 and k_3^1, which are also keys, are encrypted under the appropriate keys from the incoming wires. For example, let g be an OR gate. Then, the key k_3^1 is encrypted under the pairs of keys associated with the values $(1, 1)$, $(1, 0)$, and $(0, 1)$. In contrast, the key k_3^0 is encrypted under the pair of keys associated with $(0, 0)$. See Table 1.

Notice that given the input wire keys k_1^α and k_2^β corresponding to α and β, and the four table values (found in the fourth column of Table 1), it is possible to decrypt and obtain the output wire key $k_3^{g(\alpha,\beta)}$. Furthermore, as required above, this is the only value that can be obtained (the other keys on the input wires are not known, and so only a single table value can be decrypted). In other words, it is possible to compute the output key $k_3^{g(\alpha,\beta)}$ of a gate, and only that key, without learning anything about the real values α, β or $g(\alpha, \beta)$. (We note that the values of the table are randomly ordered so that a key's position does not reveal anything about the value that it is associated with. Despite this random ordering, the specific construction is such that given a pair of input wire keys, it is possible to locate the table entry that is encrypted by those keys.)

So far we have described how to construct a single garbled gate. A garbled circuit consists of garbled gates along with "output decryption tables". These tables map the random values on *circuit-output wires* back to their corresponding real values. That is, for a circuit-output wire w, the pairs $(0, k_w^0)$ and $(1, k_w^1)$ are provided. Then, after obtaining the key k_w^γ on a circuit-output wire, it is possible to determine the actual output bit by comparing the key to the values in the output decryption table.[1] Notice that given the keys associated with inputs x and y, it is possible to (obliviously) compute the entire circuit gate-by-gate. Then, having obtained the keys on the circuit-output wires, these can be "decrypted" providing the result $C(x, y)$.

The above construction can be described metaphorically using "locked boxes". The basic idea, as above, is that every wire is allocated two padlock keys; one key is associated with the bit zero, and the other with the bit one. Then, for each gate four doubly-locked boxes are provided, where each box is associated with a row in the truth

[1] Alternatively, in the output gates it is possible to directly encrypt 0 or 1 instead of k_w^0 or k_w^1, respectively.

table computing the gate (i.e., one box is associated with inputs $(0, 0)$, another with $(0, 1)$, and so on). The four boxes are locked so that each pair of keys (one from each input wire) opens exactly one box. Furthermore, in each box a single key relating to the output wire of the gate is stored. This key is chosen so that it correctly associates the input bits to the output bit of the gate. (For example, if the keys that open the box are associated with 0 and 1 and the gate computes the AND function, then the key inside the box is the key associated with 0 in the output wire.) The first important observation is that given the set of keys that are associated with the parties' inputs, it is possible to "compute the circuit" by opening the locked boxes one at a time (for each gate, only one box will open). The process concludes at the output-gate boxes, which can contain the actual output rather than a key. The second important observation is that the computation of the circuit reveals absolutely no information beyond the output itself. This is due to the fact that the keys are not labeled, and so it is impossible to know if a given key is associated with zero or with one. This all holds under the assumption that the keys associated with the circuit-input wires are obtained in an "oblivious manner" that does not reveal the association with the parties' inputs. Furthermore, we must assume that only a single set of keys is provided (and so in each gate only a single box can be opened). Of course, in the actual garbled-circuit construction, double-encryption replaces doubly-locked boxes, and decryption keys replace physical padlock keys.

We now proceed to informally describe Yao's protocol. In this protocol, one of the parties, henceforth the sender, constructs a garbled circuit and sends it to the other party, henceforth the receiver. The sender and receiver then interact so that the receiver obtains the input-wire keys that are associated with the inputs x and y (this interaction is described below). Given these keys, the receiver then computes the circuit as described, obtains the output, and concludes the protocol. This description only shows how the receiver obtains its output, while ignoring the output of the sender. However, the receiver's output can include the sender's output in *encrypted form* (where only the sender knows the decryption key). Then, the receiver can just forward the sender its output at the end of the computation. Since the sender's output is encrypted, the receiver learns nothing more than its own output, as required.

It remains for us to describe how the receiver obtains the keys for the circuit-input wires. Here we differentiate between the inputs of the sender and the inputs of the receiver. Regarding the sender, it simply sends the receiver the values that correspond to its input. That is, if its ith input bit is 0 and the wire w_i receives this input, then the sender just hands the receiver the string k_i^0. Notice that since all of the keys are identically distributed, the receiver can learn nothing about the sender's input from these keys. Regarding the receiver, this is more problematic. The sender cannot hand it all of the keys pertaining to its input (i.e., both the 0 and 1 keys on the receiver's input wires), because this would enable the receiver to compute more than just its output. (For a given input x of the sender, this would enable the receiver to compute $C(x, \tilde{y})$ for every $\tilde{y}$. This is much more information than a single value $C(x, y)$.) On the other hand, the receiver cannot openly tell the sender which keys to send it, because then the sender would learn the receiver's input. The solution to this is to use a 1-out-of-2 oblivious transfer protocol [6,13]. In such a protocol, a sender inputs two values x_0 and x_1 (in this case, k_w^0 and k_w^1 for some circuit-input wire w), and a receiver inputs a bit σ (in this case, corresponding to its appropriate input bit). The outcome of the protocol is that the receiver obtains the value x_σ (in this case, the key k_w^σ). Furthermore, the receiver learns nothing about the other value $x_{1-\sigma}$, and the sender learns nothing about the receiver's input σ. By having the receiver obtain its keys in this way, we obtain that (a) the sender learns nothing of the receiver's input value, and (b) the receiver obtains only a single set of keys and so can compute the circuit on only a single value, as required. This completes our high-level description of Yao's protocol.

Related Work Sketches of Yao's protocol have appeared in a number of places; see, for example, [2,7,12]. In addition, an extension of Yao's protocol to the multiparty case was presented in [3], with a full proof in [14]. This work also contains an *implicit* description (and proof) of Yao's protocol. We remark also that a full proof of [9] has recently appeared in [7].

2. Definitions

We denote the length of the inputs and the security parameter by n. We say that a function $\mu(\cdot)$ is negligible in n (or just negligible) if for every positive polynomial $p(\cdot)$ and all sufficiently large n's, it holds that $\mu(n) < 1/p(n)$. Let S be an infinite set, and let $X = \{X_s\}_{s\in S}$ and $Y = \{Y_s\}_{s\in S}$ be distribution ensembles. We say that X and Y are computationally indistinguishable, denoted $X \overset{\text{c}}{\equiv} Y$, if for every nonuniform probabilistic polynomial-time distinguisher D and all sufficiently large $s \in S$, $|\Pr[D(X_s) = 1] - \Pr[D(Y_s) = 1]|$ is negligible in $|s|$. Finally, for a probabilistic machine M, we denote by $a \leftarrow M(x)$ the event of obtaining a by invoking M on input x and a uniformly chosen random tape.

2.1. *Secure Two-Party Protocols for Semi-Honest Adversaries*

The model that we consider here is that of two-party computation in the presence of *static semi-honest* adversaries. Such an adversary controls one of the parties (statically, and so at the onset of the computation) and follows the protocol specification exactly. However, it may try to learn more information than allowed by looking at the transcript of messages that it received. Since we only consider static semi-honest adversaries here, we will sometimes omit the qualification that security is with respect to such adversaries only. The definitions presented here are according to Goldreich in [7].

Two-Party Computation A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0,1\}^n$, the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$, and the second party (with input y) wishes to obtain $f_2(x, y)$. We often denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the oblivious transfer functionality is specified by $((z_0, z_1), \sigma) \mapsto (\lambda, z_\sigma)$, where λ denotes the empty string. When the functionality f is probabilistic, we sometimes use the notation $f(x, y, r)$, where r is a uniformly chosen random tape used for computing f.

Privacy by Simulation Intuitively, a protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output only. This is formalized according to the simulation paradigm. Loosely speaking, we require that a party's *view* in a protocol execution be simulatable given only its input and output.[2] This then implies that the parties learn nothing from the protocol *execution* itself, as desired.

Definition of Security We begin with the following notation:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality, and let π be a two-party protocol for computing f.
- The view of the ith party ($i \in \{1, 2\}$) during an execution of π on (x, y) is denoted $\mathsf{view}_i^\pi(x, y)$ and equals $(x, r^i, m_1^i, \dots, m_t^i)$, where r^i equals the contents of the ith party's *internal* random tape, and m_j^i represents the jth message that it received.
- The output of the ith party during an execution of π on (x, y) is denoted $\mathsf{output}_i^\pi(x, y)$ and can be computed from its own view of the execution. Denote $\mathsf{output}^\pi(x, y) = (\mathsf{output}_1^\pi(x, y), \mathsf{output}_2^\pi(x, y))$.

Definition 1 (Security w.r.t. semi-honest behavior). Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that

$$\big\{\big(S_1\big(x, f_1(x, y)\big), f(x, y)\big)\big\}_{x,y\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \big\{\big(\mathsf{view}_1^\pi(x, y), \mathsf{output}^\pi(x, y)\big)\big\}_{x,y\in\{0,1\}^*} \quad (1)$$

$$\big\{\big(S_2\big(y, f_2(x, y)\big), f(x, y)\big)\big\}_{x,y\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \big\{\big(\mathsf{view}_2^\pi(x, y), \mathsf{output}^\pi(x, y)\big)\big\}_{x,y\in\{0,1\}^*} \quad (2)$$

where $|x| = |y|$.

Equations (1) and (2) state that the view of a party can be simulated by a probabilistic polynomial-time algorithm given access to the party's *input and output only*. We emphasize that the adversary here is semi-honest, and therefore the view is exactly according to the protocol definition. We note that it is not enough for the simulator S_i to generate a string indistinguishable from $\mathsf{view}_i^\pi(x, y)$. Rather, the *joint distribution* of the simulator's output and the functionality output $f(x, y)$ must be indistinguishable from $(\mathsf{view}_i^\pi(x, y), \mathsf{output}^\pi(x, y))$. This is necessary for probabilistic functionalities; see [4, 7] for a full discussion.

A Simpler Formulation for Deterministic Functionalities In the case that the functionality f is deterministic, a simpler definition can be used. Specifically, we do not need to

[2] A different definition of security for multiparty computation compares the output of a real protocol execution to the output of an *ideal* computation involving an incorruptible trusted third party. This trusted party receives the parties' inputs, computes the functionality on these inputs and returns to each their respective output. Loosely speaking, a protocol is secure if any real-model adversary can be converted into an ideal-model adversary such that the output distributions are computationally indistinguishable. We remark that in the case of semi-honest adversaries, this definition is equivalent to the (simpler) simulation-based definition presented here; see [7].

consider the joint distribution of the simulator's output with the protocol output. Rather, we separately require that

$$\left\{\mathsf{output}^{\pi}(x,y)\right\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{f(x,y)\right\}_{x,y\in\{0,1\}^*}$$

and in addition, that there exist S_1 and S_2 such that

$$\left\{S_1\left(x, f_1(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_1^{\pi}(x,y)\right\}_{x,y\in\{0,1\}^*} \tag{3}$$

$$\left\{S_2\left(y, f_2(x,y)\right)\right\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_2^{\pi}(x,y)\right\}_{x,y\in\{0,1\}^*} \tag{4}$$

The reason that this suffices is that when f is deterministic, $\mathsf{output}^{\pi}(x,y)$ must equal $f(x,y)$. Furthermore, the distinguisher for the ensembles can compute $f(x,y)$ by itself (because it is given x and y which are the indices of the ensemble). See [7, Sect. 7.2.2] for more discussion.

Deterministic Same-Output Functionalities We say that a functionality $f=(f_1,f_2)$ is same-output if $f_1=f_2$. In our presentation, we will show how to securely compute *deterministic same output functionalities* only. This suffices for obtaining secure protocols for arbitrary probabilistic functionalities.

In order to see this, first note that given a protocol for securely computing any deterministic functionality, it is possible to construct a secure protocol for computing any probabilistic functionality as follows. Let $f=(f_1,f_2)$ be a probabilistic functionality. Then, define a deterministic functionality $f'((x,r),(y,s))=f(x,y,r\oplus s)$ and assume that we have a secure protocol π' for computing f'. Now, the following is a secure protocol π for computing f. Upon respective inputs $x,y\in\{0,1\}^n$, parties P_1 and P_2 choose uniformly distributed strings $r\in_R\{0,1\}^{q(n)}$ and $s\in_R\{0,1\}^{q(n)}$, respectively, where $q(n)$ is an upper bound on the number of random bits used to compute f. They then invoke the protocol π' for securely computing f' in order to both obtain $f'((x,r),(y,s))=f(x,y,r\oplus s)$. The fact that this yields a secure protocol for computing f was formally proved in [7, Sect. 7.3]. Note that the size of the circuit computing f' is of the same order as the size of the circuit computing f. The only difference is that the circuit for f' has $|r|$ additional exclusive-or gates, where $|r|$ is the length of f's random tape.

So far we have shown that it suffices to consider deterministic functionalities. Next, we show that the restriction to same-output functionalities is also not a limitation. That is, as above it is possible to construct a secure protocol for computing arbitrary functionalities from a secure protocol for computing same-output functionalities. In particular, let $f=(f_1,f_2)$ be an arbitrary functionality and define the same-output functionality f' as follows: $f'((x,r),(y,s))=(f_1(x,y)\oplus r\|f_2(x,y)\oplus s)$, where $a\|b$ denotes the concatenation of a with b. Now, given a secure protocol π' for computing the same-output functionality f', it is possible to securely compute the functionality $f=(f_1,f_2)$. As above, upon respective inputs $x,y\in\{0,1\}^n$, parties P_1 and P_2 choose uniformly distributed strings $r\in_R\{0,1\}^{q(n)}$ and $s\in_R\{0,1\}^{q(n)}$, respectively, where $q(n)$ is an upper bound on the output length of f on inputs of length n. They then invoke the protocol π' for securely computing f' in order to both obtain $f'((x,r),(y,s))$; denote the first half of this output by v and the second half by w. Then, upon receiving (v, w), party P_1 computes $v \oplus r$ and obtains $f_1(x, y)$. Likewise, upon receiving (v, w), party P_2 computes $w \oplus s$ and obtains $f_2(x, y)$. It is easy to see that the resulting protocol securely computes f. This is due to the fact that r completely obscures $f_1(x, y)$ from P_2 and likewise s completely obscures $f_2(x, y)$ from P_1. Thus, neither party learns more than its own input. (In fact, the strings $f_1(x, y) \oplus r$ and $f_2(x, y) \oplus s$ are uniformly distributed and so are easily simulated.) As above, the size of the circuit computing f' is of the same order as the size of the circuit computing f. The only difference is that f' has one additional exclusive-or gate for every circuit-output wire.

Since it suffices to consider deterministic same-output functions only, we will present Yao's protocol for this simpler case. The generalization to arbitrary probabilistic functionalities will then be derived by corollary from the above arguments.

3. Tools

3.1. *"Special" Private-Key Encryption*

Our construction uses a private-key encryption scheme that has indistinguishable encryptions for multiple messages. Informally speaking, this means that for every two (known) vectors of messages $\overline{x}$ and $\overline{y}$, no polynomial-time adversary can distinguish an encryption of the vector $\overline{x}$ from an encryption of the vector $\overline{y}$. We stress that according to our construction of Yao's garbled circuit, the encryption scheme must be secure for *multiple messages*. Therefore one-time pads cannot be used. In our proof of security, we will actually use an encryption scheme that is secure under chosen-plaintext attacks (strictly speaking this is not necessary, but it does simplify the presentation). We refer the reader to [7, Chap. 5] for formal definitions of secure encryption.

We will require an additional property from the encryption scheme that we use. Loosely speaking, we require that an encryption under one key will fall in the range of an encryption under another key with negligible probability. We also require that given the key k, it is possible to efficiently verify if a given ciphertext is in the range of k. (These two requirements are very easily satisfied, as demonstrated below.) The reason that we require these additional properties is to enable the receiver to *correctly* compute the garbled circuit. Recall that in every gate, the receiver is given two random keys that enable it to decrypt and obtain the random key for the gate-output wire; see Table 1. A problem that immediately arises here is how can the receiver know which value is the intended decryption. (Notice that it may be the case that all strings can be decrypted.) By imposing the requirement that encryptions under one key will almost never be valid encryptions under another key, and requiring that this can also be efficiently verified, it will hold that only one of the values will be valid (except with negligible probability). The receiver will then take the (single) correctly-decrypted value as the key for the gate-output wire.

We now formally define the requirements on the encryption scheme:

Definition 2. Let (G, E, D) be a private-key encryption scheme and denote the range of a key in the scheme by $\mathsf{Range}_n(k) \overset{\text{def}}{=} \{E_k(x)\}_{x \in \{0,1\}^n}$. Then,

1. We say that (G, E, D) has an $\mathsf{elusive\ range}$ if for every probabilistic polynomial-time machine A, every polynomial $p(\cdot)$, and all sufficiently large n,

$$\Pr_{k \leftarrow G(1^n)}\left[A\left(1^n\right) \in \mathsf{Range}_n(k)\right] < \frac{1}{p(n)}$$

2. We say that (G, E, D) has an $\mathsf{efficiently\ verifiable\ range}$ if there exists a probabilistic polynomial-time machine M such that $M(1^n, k, c) = 1$ if and only if $c \in \mathsf{Range}_n(k)$.

By convention, for every $c \notin \mathsf{Range}_n(k)$, we have that $D_k(c) = \perp$.

Notice that the requirements for an "elusive range" are quite weak. In particular, the machine A is *oblivious* in that it is given no information on k and no examples of ciphertexts within $\mathsf{Range}_n(k)$. Thus, A must "hit" the range with no help whatsoever.

We now show that it is easy to construct encryption schemes with the above properties. Let $\mathcal{F} = \{f_k\}$ be a family of pseudorandom functions [8], where $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ for $k \in \{0, 1\}^n$. Then, define

$$E_k(x) = \left\langle r, f_k(r) \oplus x0^n \right\rangle$$

where $x \in \{0, 1\}^n$, $r \in_R \{0, 1\}^n$, and $x0^n$ denotes the concatenation of x and 0^n.[3] The fact that this encryption scheme has indistinguishable encryptions under chosen-plaintext attacks is well known. Regarding our additional requirements:

1. *Elusive range:* Notice that if a truly random function f_{rand} was used instead of f_k, then the probability that a value c output by the machine A is in the range of $\langle r, f_{\text{rand}}(r) \oplus x0^n \rangle$ is negligible. This follows from the fact that obtaining such a c involves finding a value r and then predicting the last n bits of $f_{\text{rand}}(r)$ (notice that these last n bits are fully revealed in $f_{\text{rand}}(r) \oplus x0^n$). Since f_{rand} is random, this prediction can succeed with probability at most 2^{-n}. Now, by the assumption that f_k is *pseudorandom,* it follows that a polynomial-time machine A will also succeed in generating such a c with at most negligible probability. Otherwise, such an A could be used to distinguish f_k from a random function.
2. *Efficiently verifiable range:* Given k and $c = \langle r, s \rangle$, it is possible to compute $f_k(r)$ and verify that the last n bits of $f_k(r)$ equal the last n bits of s. If yes, then it follows that $c \in \mathsf{Range}_n(k)$, and if not, then $c \notin \mathsf{Range}_n(k)$.

We stress that there are many possible ways to ensure correctness in the decryption of a gate. For example, as described in [12], explicit (and randomly permuted) indices may be used instead.[4]

[3] In fact, the string of 0's can have any super-logarithmic length. We set it to be of length n for simplicity.

[4] We chose this method somewhat arbitrarily. We feel some preference due to the fact that the gate description and circuit construction is the simplest this way. As we will see, however, some price is paid in the proof of correctness.

Double-Encryption Security In Yao's protocol, the private-key encryption scheme is used in order to double-encrypt values. As we have described, the protocol works by double-encrypting four values, where each double encryption uses a different combination of the keys associated with the input wires. Intuitively, given only two keys, it is possible to decrypt only one of the values. However, formally, this must be proven. We define a double-encryption experiment here and prove that any encryption scheme that is secure under chosen plaintext attacks is secure for double-encryption here. We remark that the experiment does not look very natural. However, it is exactly what is needed in our proof of security. Let (G, E, D) be a private-key encryption scheme and assume without loss of generality that $G(1^n)$ returns a string of length n (i.e., the length of a key generated with security parameter 1^n is exactly n). We denote $\overline{E}(k_0, k_1, m) = E_{k_0}(E_{k_1}(m))$. The experiment definition is as follows.

$\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(n, \sigma)$

1. The adversary $\mathcal{A}$ is invoked upon input 1^n and outputs two keys k_0 and k_1 of length n and two triples of messages (x_0, y_0, z_0) and (x_1, y_1, z_1), where all messages are of the same length.
2. Two keys $k'_0, k'_1 \leftarrow G(1^n)$ are chosen for the encryption scheme.
3. $\mathcal{A}$ is given the challenge ciphertext $\langle \overline{E}(k_0, k'_1, x_\sigma), \overline{E}(k'_0, k_1, y_\sigma), \overline{E}(k'_0, k'_1, z_\sigma)\rangle$ as well as oracle access to $\overline{E}(\cdot, k'_1, \cdot)$ and $\overline{E}(k'_0, \cdot, \cdot)$.[5]
4. $\mathcal{A}$ outputs a bit b, and this is taken as the output of the experiment.

Security under double encryption simply means that the adversary outputs 1 when $\sigma = 0$ with almost the same probability as it outputs 1 when $\sigma = 1$.

Definition 3. An encryption scheme (G, E, D) is secure under chosen double encryption if for every nonuniform probabilistic polynomial-time machine $\mathcal{A}$, every polynomial $p(\cdot)$, and all sufficiently large n,

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(n, 1) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(n, 0) = 1\right]\right| < \frac{1}{p(n)}$$

We now show that any encryption scheme that is secure (i.e., has indistinguishable encryptions) under chosen plaintext attacks is secure under chosen double-encryption. We remark that all security here is in the nonuniform model (and so we assume security under chosen plaintext attacks for nonuniform adversaries). It is well known that under chosen plaintext attacks, security for a single message implies security for multiple messages, see [7, Sect. 5.4], and we will thus assume this in our proof. For the sake of completeness, we define the chosen-plaintext experiment for the case of multiple mes-

[5] Note that in the ciphertexts that $\mathcal{A}$ receives, at least one of the keys used is unknown to $\mathcal{A}$. In addition, the oracle access here means that $\mathcal{A}$ can provide any k and m to $\overline{E}(\cdot, k'_1, \cdot)$ and receive back $\overline{E}(k, k'_1, m)$; likewise for $\overline{E}(k'_0, \cdot, \cdot)$.

sages. In fact, we consider only the case of two messages, because this suffices for our proof later.

$\mathsf{Expt}^{\mathsf{cpa}}_{\mathcal{A}}(n,\sigma)$

1. A key $k \leftarrow G(1^n)$ is chosen, and the adversary $\mathcal{A}$ is invoked with input 1^n and oracle access to $E_k(\cdot)$. The adversary $\mathcal{A}$ outputs two pairs of messages (x_0, y_0) and (x_1, y_1).
2. The challenge ciphertexts $c_1 = E_k(x_\sigma)$ and $c_2 = E_k(y_\sigma)$ are computed.
3. $\mathcal{A}$ is given the pair (c_1, c_2) and continued oracle access to $E_k(\cdot)$
4. $\mathcal{A}$ outputs a bit b, and this is taken as the output of the experiment.

The definition of security under chosen plaintext attacks is analogous to Definition 3, except that $\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}$ is replaced with $\mathsf{Expt}^{\mathsf{cpa}}_{\mathcal{A}}$. We are now ready to state the lemma.

Lemma 4. *Let (G, E, D) be a private-key encryption scheme that has indistinguishable encryptions under chosen plaintext attacks in the presence of nonuniform adversaries. Then (G, E, D) is secure under chosen double encryption.*

Proof sketch. We do not provide a full proof of this lemma but rather a detailed proof sketch only. The full proof can be derived from this sketch in a straightforward manner. In order to prove this lemma, we define a modified experiment, denoted $\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}(n,\sigma)$, which is exactly the same as $\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(\sigma)$, except that the y part of the challenge ciphertext does not depend on σ. That is, the challenge ciphertext equals $\langle \overline{E}(k_0, k_1', x_\sigma), \overline{E}(k_0', k_1, y_0), \overline{E}(k_0', k_1', z_\sigma)\rangle$; note that x_σ and z_σ are encrypted as before, but y_0 is always encrypted (even if $\sigma = 1$). Clearly,

$$\Pr\big[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(n,0)=1\big] = \Pr\big[\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}(n,0)=1\big] \tag{5}$$

because in both cases, the encrypted values are x_0, y_0, and z_0. We will prove that for every nonuniform probabilistic polynomial-time adversary and for some negligible function $\mu(\cdot)$, the following two equations hold:

$$\big|\Pr\big[\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}(n,0)=1\big] - \Pr\big[\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}(n,1)=1\big]\big| < \mu(n) \tag{6}$$

$$\big|\Pr\big[\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}(n,1)=1\big] - \Pr\big[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}}(n,1)=1\big]\big| < \mu(n) \tag{7}$$

Combining (5) to (7), we obtain that (G, E, D) is secure under chosen double encryption. We will prove (6); (7) is proven in an analogous way.

We begin by modifying $\mathsf{Expt}^{\mathsf{mod}}_{\mathcal{A}}$ in the following way. First, we claim that indistinguishability holds even if the adversary $\mathcal{A}$ can choose k_0' by itself. We can therefore let $\mathcal{A}$ choose k_0, k_1, and k_0'. Given that this is the case, we need not generate $\overline{E}(k_0', k_1, y_0)$ as part of the challenge ciphertext (because given k_0' and k_1, $\mathcal{A}$ can compute it by itself). For the same reason, we can remove the oracle $\overline{E}(k_0', \cdot, \cdot)$ from the experiment. We therefore need only to prove that (6) holds for the further modified experiment $\mathsf{Expt}^{\mathsf{mod}'}_{\mathcal{A}}$ defined as follows.

$\mathbf{Expt}_{\mathcal{A}}^{\mathsf{mod}'}(\boldsymbol{n}, \boldsymbol{\sigma})$

1. The adversary $\mathcal{A}$ is invoked upon input 1^n and outputs three keys k_0, k_1, and k_0' of length n and two pairs of messages (x_0, z_0) and (x_1, z_1), where all messages are of the same length.
2. A key $k_1' \leftarrow G(1^n)$ is chosen for the encryption scheme.
3. $\mathcal{A}$ is given $\langle \overline{E}(k_0, k_1', x_\sigma), \overline{E}(k_0', k_1', z_\sigma)\rangle$ and oracle access to $\overline{E}(\cdot, k_1', \cdot)$.
4. $\mathcal{A}$ outputs a bit b, and this is taken as the output of the experiment.

From what we have stated above, if we prove that the analogue of (6) holds for $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{mod}'}$, then (6) itself clearly also holds. However, $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{mod}'}$ is now almost identical to $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}}$. The only differences are:

1. In $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{mod}'}$ the challenge ciphertext is first encrypted with k_1' (the secret key) and then with k_0 or k_0', whereas in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}}$ the challenge ciphertext is encrypted with the secret key only. However, this clearly does not matter because the adversary knows k_0 and k_0' and so can compute this itself.
2. In $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{mod}'}$ the oracle given to the adversary is $\overline{E}(\cdot, k_1', \cdot)$, whereas in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{cpa}}$ it is $E_k(\cdot)$. However, since k and k_1' play the same role as the secretly-chosen key, it is clear that given oracle $E_{k_1'}(\cdot)$, it is possible to efficiently emulate the oracle $E(\cdot, k_1', \cdot)$. Therefore, this also makes no difference.

We conclude that (6) follows from the security of (G, E, D) under chosen plaintext attacks. As we have stated, (7) is proven in an analogous way, and thus we obtain that (G, E, D) is also secure under chosen double encryption. This concludes the proof sketch. □

3.2. *Oblivious Transfer*

As we have mentioned, the 1-out-of-2 oblivious transfer functionality is defined by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where λ denotes the empty string. For the sake of self-containment, we will briefly describe the oblivious transfer protocol of [6] that is secure in the presence of semi-honest adversaries. Our description will be for the case that $x_0, x_1 \in \{0, 1\}$; when considering semi-honest adversaries, the general case can be obtained by running the single-bit protocol many times in parallel.

Protocol 1 (Oblivious transfer [6]).

- **Inputs:** *P_1 has $x_0, x_1 \in \{0, 1\}$, and P_2 has $\sigma \in \{0, 1\}$.*
- **The protocol:**
    1. *P_1 randomly chooses a permutation-trapdoor pair (f, t) from a family of enhanced trapdoor permutations.*[6] *P_1 sends f (but not the trapdoor t) to P_2.*
    2. *P_2 chooses a random v_σ in the domain of f and computes $w_\sigma = f(v_\sigma)$. In addition, P_2 chooses a random $w_{1-\sigma}$ in the domain of f, using the "enhanced" sampling algorithm (see Footnote 6). P_2 sends (w_0, w_1) to P_1.*

[6] Informally speaking, an enhanced trapdoor permutation has the property that it is possible to sample from the range, so that given the coins used for sampling, it is still hard to invert the value. See [7, Appendix C.1] for more details.

3. *P_1 uses the trapdoor t and computes $v_0 = f^{-1}(w_0)$ and $v_1 = f^{-1}(w_1)$. Then, it computes $b_0 = B(v_0) \oplus x_0$ and $b_1 = B(v_1) \oplus x_1$, where B is a hard-core bit of f. Finally, P_1 sends (b_0, b_1) to P_2.*
4. *P_1 computes $x_\sigma = B(v_\sigma) \oplus b_\sigma$ and outputs x_σ.*

The proof to the following theorem can be found in [7, Sect. 7.3.2].

Theorem 5. *Assuming that (f, t) are chosen from a family of enhanced trapdoor permutations, Protocol 1 securely computes the 1-out-of-2 oblivious transfer functionality in the presence of static semi-honest adversaries.*

Recall that since the oblivious transfer functionality is deterministic, it suffices to use the simplified definition of (3) and (4). Thus, it is guaranteed that there exist simulators, denoted S_1^{OT} and S_2^{OT}, that generate the appropriate views of parties P_1 and P_2, respectively. We remark that simulator S_1^{OT} receives P_1's input (x_0, x_1) and outputs a full view of P_1 that includes the input (x_0, x_1), a random tape, and the incoming messages that P_1 expects to see in a real execution (of course, this view output by S_1^{OT} is only computationally indistinguishable from a real view). Notice that P_1 has no output in the oblivious transfer functionality, and so S_1^{OT} receives only P_1's input. The simulator S_2^{OT} receives P_2's input σ and output x_σ and outputs a view, as described above.

4. Yao's Two-Party Protocol

We are now ready to describe the protocol. We begin by formally describing how the garbled circuit is constructed. Then, we describe the protocol and prove its security.

4.1. *The Garbled Circuit Construction*

In this section, we describe the garbled circuit construction. Let C be a boolean circuit that receives two inputs $x, y \in \{0,1\}^n$ and outputs $C(x, y) \in \{0,1\}^n$ (for simplicity, we assume that the input length, output length, and the security parameter are all of the same length n). We also assume that C has the property that if a circuit-output wire comes from a gate g, then gate g has no wires that are input to other gates.[7] (Likewise, if a circuit-input wire is itself also a circuit-output, then it is not input into any gate.)

We begin by describing the construction of a single garbled gate g in C. The circuit C is boolean, and therefore any gate is represented by a function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$. Now, let the two input wires to g be labeled w_1 and w_2, and let the output wire from g be labeled w_3. Furthermore, let $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$ be six keys obtained by independently invoking the key-generation algorithm $G(1^n)$; for simplicity, assume that these keys are also of length n. Intuitively, we wish to be able to compute $k_3^{g(\alpha,\beta)}$ from k_1^α and k_2^β, without revealing any of the other three values $k_3^{g(1-\alpha,\beta)}, k_3^{g(\alpha,1-\beta)}, k_3^{g(1-\alpha,1-\beta)}$.

[7] This requirement is due to our labeling of gates described below; see Footnote 8. We note that this assumption on C increases the number of gates by at most n.

The gate g is defined by the following four values:

$$c_{0,0}=E_{k_1^0}\big(E_{k_2^0}\big(k_3^{g(0,0)}\big)\big)$$

$$c_{0,1}=E_{k_1^0}\big(E_{k_2^1}\big(k_3^{g(0,1)}\big)\big)$$

$$c_{1,0}=E_{k_1^1}\big(E_{k_2^0}\big(k_3^{g(1,0)}\big)\big)$$

$$c_{1,1}=E_{k_1^1}\big(E_{k_2^1}\big(k_3^{g(1,1)}\big)\big)$$

where E is from a private key encryption scheme (G,E,D) that has indistinguishable encryptions under chosen plaintext attacks and has an elusive efficiently verifiable range; see Sect. 3.1. The actual gate is defined by a *random permutation* of the above values, denoted as c_0,c_1,c_2,c_3; from here on we call them the garbled table of gate g. Notice that given k_1^α and k_2^β and the values c_0,c_1,c_2,c_3, it is possible to compute the output of the gate $k_3^{g(\alpha,\beta)}$ as follows. For every i, compute $D_{k_2^\beta}(D_{k_1^\alpha}(c_i))$. If more than one decryption returns a non-$\perp$ value, then output abort. Otherwise, define k_3^γ to be the only non-$\perp$ value that is obtained. (Notice that if only a single non-$\perp$ value is obtained, then this will be $k_3^{g(\alpha,\beta)}$ because it is encrypted under the given keys k_1^α and k_2^β. Later we will show that except with negligible probability, only one non-$\perp$ value is indeed obtained.)

We are now ready to show how to construct the entire garbled circuit. Let m be the number of *wires* in the circuit C, and let $w_1,\dots,w_m$ be labels of these wires. These labels are all chosen uniquely with the following exception: if w_i and w_j are both output wires from the same gate g, then $w_i=w_j$ (this occurs if the fan-out of g is greater than one). Likewise, if an input bit enters more than one gate, then all circuit-input wires associated with this bit will have the same label.[8] Next, for every label w_i, choose two independent keys $k_i^0,k_i^1\leftarrow G(1^n)$; we stress that all of these keys are chosen independently of the others. Now, given these keys, the four garbled values of each gate are computed as described above, and the results are permuted randomly. Finally, the output or decryption tables of the garbled circuit are computed. These tables simply consist of the values $(0,k_i^0)$ and $(1,k_i^1)$, where w_i is a *circuit-output wire*. (Alternatively, output gates can just compute 0 or 1 directly. That is, in an output gate, one can define $c_{\alpha,\beta}=E_{k_1^\alpha}(E_{k_2^\beta}(g(\alpha,\beta)))$ for every $\alpha,\beta\in\{0,1\}$.)

The entire garbled circuit of C, denoted $G(C)$, consists of the garbled table for each gate and the output tables. We note that the structure of C is given, and the garbled version of C is simply defined by specifying the output tables and the garbled table that belongs to each gate. This completes the description of the garbled circuit.

[8] This choice of labeling is not essential, and it is possible to provide unique labels for all wires. However, in such a case, the table of a gate with fan-out greater than one will have to be redefined so that the keys of all of the wires leaving the gate are encrypted. We chose this labeling because it seems to make for a simpler gate definition. We note, however, that due to this choice, we must assume that if a gate g has an output wire exiting from it, then it does not have another wire that exits it and enters another gate. As we have mentioned, this increases the number of gates by at most n.

Correctness We now claim that the above garbled circuit enables correct computation of the function. That is, given the appropriate input strings and the garbled table for each gate, it is possible to obtain the correct output. It is at this point that we use the "special" properties of the encryption scheme described in Sect. 3.1.

Claim 6 (Correctness). *Let $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_n$ be two n-bit inputs for C. Furthermore, let $w_{\mathrm{in}_1}, \ldots, w_{\mathrm{in}_n}$ be the labels of the circuit-input wires corresponding to x, and let $w_{\mathrm{in}_{n+1}}, \ldots, w_{\mathrm{in}_{2n}}$ be the labels of the circuit-input wires corresponding to y. Finally, assume that the encryption scheme used to construct $G(C)$ has an elusive and efficiently verifiable range. Then, given the garbled circuit $G(C)$ and the strings $k_{\mathrm{in}_1}^{x_1}, \ldots, k_{\mathrm{in}_n}^{x_n}, k_{\mathrm{in}_{n+1}}^{y_1}, \ldots, k_{\mathrm{in}_{2n}}^{y_n}$, it is possible to compute $C(x, y)$, except with negligible probability.*

Proof. We begin by showing that every gate can be "decrypted" correctly. Specifically, let g be a gate with incoming wires w_1, w_2 and outgoing wire w_3. Then, we show that for every $\alpha, \beta \in \{0, 1\}$, given k_1^α and k_2^β and the garbled table of g, it is possible to determine $k_3^{g(\alpha,\beta)}$, except with negligible probability. More formally, let c_0, c_1, c_2, c_3 be the garbled table of gate g. We wish to find c_i such that $c_i = E_{k_1^\alpha}(E_{k_2^\beta}(k_3^{g(\alpha,\beta)}))$. We claim that except with negligible probability, there exists a *single* i such that $c_i \in \mathsf{Range}_n(k_1^\alpha)$ and $D_{k_1^\alpha}(c_i) \in \mathsf{Range}_n(k_2^\beta)$. In other words, at most one of the values decrypts correctly (from here on we use this informal term to mean what is formally described above).

This follows from the fact that the encryption scheme has an elusive range. Specifically, recall that the gate was constructed by first choosing independent values for the gate-input and gate-output wires $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$. Next, the values c_0, c_1, c_2, and c_3 are computed. Now, assume that there are (at least) two values c such that for both of them, $c \in \mathsf{Range}(k_1^\alpha)$ and $D_{k_1^\alpha}(c) \in \mathsf{Range}_n(k_2^\beta)$; denote these two values c_i and c_j. Without loss of generality, assume also that $c_i = E_{k_1^\alpha}(E_{k_2^\beta}(k_3^{g(\alpha,\beta)}))$; i.e., assume that c_i should be correctly decrypted. There are two cases regarding c_j:

1. $c_j = E_{k_1^\alpha}(E_{k_2^{1-\beta}}(z))$ *for* $z \in \{k_3^0, k_3^1\}$:

   By our assumption regarding c_j, it follows that $c_j \in \mathsf{Range}(k_1^\alpha)$ and $D_{k_1^\alpha}(c_j) \in \mathsf{Range}_n(k_2^\beta)$. This means that $E_{k_2^{1-\beta}}(z) \in \mathsf{Range}_n(k_2^\beta)$. Next, as mentioned above, recall that $k_2^{1-\beta}, k_3^0$, and k_3^1 are all uniform and independent of k_2^β. Therefore, we can define a machine A that chooses two random keys $k', k'' \leftarrow G(1^n)$ and outputs $c = E_{k'}(k'')$. The probability that $c \in \mathsf{Range}(k)$ for $k \leftarrow G(1^n)$ *equals* the probability that $E_{k_2^{1-\beta}}(z) \in \mathsf{Range}_n(k_2^\beta)$ (recall that $z \in \{k_3^0, k_3^1\}$). Since the encryption scheme (G, E, D) has an elusive range, we conclude that the probability that $c \in \mathsf{Range}_n(k)$ is negligible. Therefore, the probability that $E_{k_2^{1-\beta}}(z) \in \mathsf{Range}_n(k_2^\beta)$ is also negligible. This concludes this case.
2. $c_j = E_{k_1^{1-\alpha}}(z)$ *for* $z = E_{k'}(k'')$ *where* $k' \in \{k_2^0, k_2^1\}$ *and* $k'' \in \{k_3^0, k_3^1\}$:

   In this case, we have that $E_{k_1^{1-\alpha}}(z) \in \mathsf{Range}_n(k_1^\alpha)$. Using the same arguments as above and noticing once again that $k_1^{1-\alpha}, k'$, and k'' are all independent of k_1^α, we have that this case occurs also with at most negligible probability.

Now, given that in every gate at most one c_i decrypts correctly, we prove the claim. In order to do this, we define that the key k is correct for wire w_i if $k = k_i^\alpha$, where $\alpha \in \{0, 1\}$ is the value obtained on wire w_i when computing the un-garbled circuit C on inputs (x, y). By induction on the circuit, starting from the bottom and working up, we show that for every wire, the correct key for the wire is obtained. This holds for the circuit-input wires by the fact that the keys $k_{\mathrm{in}_1}^{x_1}, \ldots, k_{\mathrm{in}_n}^{x_n}, k_{\mathrm{in}_{n+1}}^{y_1}, \ldots, k_{\mathrm{in}_{2n}}^{y_n}$ are given and is the base case of the induction. Assume that it is true for a gate g with gate-input wires w_i and w_j and let k_i^α and k_j^β be the respective keys held for these wires. Then, by the decryption procedure, it follows that the value $k_\ell^{g(\alpha,\beta)} = D_{k_j^\beta}(D_{k_i^\alpha}(c_{\alpha,\beta}))$ is obtained, where w_ℓ is the output wire of the gate.[9] Furthermore, by the arguments shown above, this is the only value that is decrypted correctly. Therefore, the correct key for the output wire of gate g is also obtained. This concludes the inductive step.

It follows that the correct keys of the output wires of the circuit are obtained, except with negligible probability. That is, the keys obtained for the circuit-output wires all correspond to the output value $C(x, y)$. Therefore, the value obtained after using the output tables is exactly $C(x, y)$, as required. □

Removing the Error Probability The above construction allows for a negligible probability of error. This is due to two possible events: (a) in some gate, more than one value decrypts correctly, or (b) in some gate, the correct value does not decrypt correctly. As we have mentioned in footnote 9, this second event can occur if the encryption scheme has decryption errors. This problem can be removed by using a scheme *without* decryption errors (this is not a limitation because decryption errors can always be removed [5]).

Regarding the first event causing error, this can be overcome in one of two ways. First, when constructing the circuit, it is possible to check that an error does not occur. Then, if an error has occurred, it is possible to reconstruct the garbled circuit again, repeating until no errors occur. (For this to work, we need to assume that the machine that verifies if a value is in the range of a key runs in deterministic polynomial-time, as is the case in our construction. Alternatively, it suffices to assume that it has only a one-sided error and never returns 1 when a value is not in the range.) The problem with this approach is that the construction of the circuit now runs in expected, and not strict, polynomial-time. Another approach is to use explicit randomly permuted indices, meaning that the decrypted values in the gates reveal exactly which item in the next table is to be opened. This approach was described in [12].

4.2. *Yao's Two-Party Protocol*

As we have seen above, given the keys that correspond to the correct input, it is possible to obtain the correct output from the garbled circuit. Thus, the protocol proceeds by party P_1 constructing the garbled circuit and giving it to P_2. Furthermore, P_1 hands P_2 the keys that correspond to $x = x_1 \cdots x_n$. In addition, P_2 must obtain the keys that correspond to its input $y = y_1 \cdots y_n$. However, this must be done carefully, ensuring the following:

[9] This holds if there are no decryption errors (i.e., if for every k and every x, $D_k(E_k(x)) = x$). If there *is* a negligible error in the decryption, then we will inherit a negligible error probability here.

1. P_1 should not learn anything about P_2's input string y.
2. P_2 should obtain the keys corresponding to y and no others. (Otherwise, P_2 could compute $C(x, y)$ and $C(x, y')$ for $y' \neq y$, in contradiction to the requirement that $C(x, y)$ and nothing else is learned.)

The above two problems are solved by having P_1 and P_2 run 1-out-of-2 oblivious transfer protocols [6,13]. That is, for every bit of P_2's input, the parties run an oblivious transfer protocol where P_1's input is (k^0_{n+i}, k^1_{n+i}) and P_2's input is y_i. In this way, P_2 obtains the keys $k^{y_1}_{n+1}, \ldots, k^{y_n}_{2n}$ and only these keys. In addition, P_1 learns nothing about y. We are now ready to formally describe the protocol.

Protocol 2 (Yao's two-party protocol).

- **Inputs:** *P_1 has $x \in \{0, 1\}^n$, and P_2 has $y \in \{0, 1\}^n$.*
- **Auxiliary input:** *A boolean circuit C such that for every $x, y \in \{0, 1\}^n$, it holds that $C(x, y) = f(x, y)$, where $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}^n$. We require that C is such that if a circuit-output wire leaves some gate g, then gate g has no other wires leading from it into other gates (i.e., no circuit-output wire is also a gate-input wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates.*
- **The protocol:**
  1. *P_1 constructs the garbled circuit $G(C)$, as described in Sect. 4.1, and sends it to P_2.*
  2. *Let $w_1, \ldots, w_n$ be the circuit-input wires corresponding to x, and let $w_{n+1}, \ldots, w_{2n}$ be the circuit-input wires corresponding to y. Then,*
     (a) *P_1 sends P_2 the strings $k^{x_1}_1, \ldots, k^{x_n}_n$.*
     (b) *For every i, P_1 and P_2 execute a 1-out-of-2 oblivious transfer protocol in which P_1's input equals (k^0_{n+i}, k^1_{n+i}) and P_2's input equals y_i.*
     The above oblivious transfers can all be run in parallel.
  3. *Following the above, P_2 has obtained the garbled circuit and $2n$ keys corresponding to the $2n$ input wires to C. Party P_2 then computes the circuit, as described in Sect. 4.1, obtaining $f(x, y)$. P_2 then sends $f(x, y)$ to P_1, and they both output this value.*

We now provide a formal proof that Protocol 2 securely computes the functionality f. Our proof could be simplified by relying on a composition theorem, such as that found in [7, Sect. 7.3.1]. However, for the sake of self-containment, we provide a direct proof of the security of the protocol.

Theorem 7. *Let f be a deterministic same-output functionality. Furthermore, assume that the oblivious transfer protocol is secure in the presence of static semi-honest adversaries and that the encryption scheme has indistinguishable encryptions under chosen plaintext attacks and has an elusive and efficiently verifiable range. Then, Protocol 2 securely computes f in the presence of static semi-honest adversaries.*

Proof. Intuitively, since the oblivious transfer protocol is secure, party P_2 receives exactly one key per circuit-input wire. Then, by the security of the encryption scheme, it is only able to decrypt one value in each gate. Furthermore, it has no idea if the value obtained in this decryption corresponds to 0 or to 1. Therefore, it learns nothing from this computation, except for the output itself. We now formally prove this. Recall that since we consider deterministic functionalities, we can use the simpler formulation of security as stated in (3) and (4). We prove separately the cases where P_1 is corrupted and where P_2 is corrupted.

A Simplifying Convention In the proof below, we will use the simulators S_1^{OT} and S_2^{OT} that exist for the oblivious transfer functionality in order to generate views for the corrupted parties. In general, a view is represented as the party's input followed by its random tape and concluding with the series of incoming messages. In order to simplify the presentation, we will present the view of a party in a different order. Specifically, we will write the view of a party in Protocol 2—*excluding the oblivious transfers*—in the usual way. However, the view of the party in the oblivious transfers is written in full where it appears in the protocol transcript. That is, instead of splitting the view in the oblivious transfers into input, random-tape, and incoming messages, the input and random-tape are written together with the incoming messages. This clearly makes no difference and is just to simplify notation (the standard way of writing the view of a party can be received by a trivial transformation of the view that we write below).

Case 1—P_1 is corrupted

Notice that P_1's view in an execution of π consists only of its view in the oblivious transfer protocols and a single message that it receives from P_2 at the end (that is supposedly the output). By the security of the oblivious transfer protocol, P_1's view in the oblivious transfer executions can be generated without knowing P_2's input. Furthermore, by the correctness of the construction of the garbled circuit (Claim 6), party P_2 obtains the correct output $f(x,y)$, except with negligible probability. Therefore, the message that P_1 receives from P_2 at the end of a real protocol execution equals $f(x,y)$, except with negligible probability. A simulator that is given $(x, f(x,y))$ can therefore simulate the complete view of P_1 by first simulating its view in the oblivious transfers and then writing $f(x,y)$ at the end. The formal proof of this follows a rather standard hybrid argument.

We begin by describing the simulator S_1: Upon input $(x, f(x,y))$, simulator S_1 uniformly chooses a random-tape r_C for P_1 and generates the garbled circuit that P_1 would generate with randomness r_C. Then, let $k_{n+1}^0, k_{n+1}^1, \ldots, k_{2n}^0, k_{2n}^1$ be the keys that correspond to P_2's input in the constructed garbled circuit, and let S_1^{OT} be the simulator that is guaranteed to exist for party P_1 in the oblivious transfer protocol. For every $i = 1, \ldots, n$, simulator S_1 invokes the simulator S_1^{OT} upon input (k_{n+i}^0, k_{n+i}^1) in order to obtain P_1's view in the ith oblivious transfer (since P_1 has no output from the oblivious transfer, the simulator is invoked with its input only). Recall that the view generated by S_1^{OT} is made up of the input (in this case (k_{n+i}^0, k_{n+i}^1)), a random tape, and a transcript of messages received. As we have mentioned, we will place the entire view of the party in the oblivious transfers together with the message transcript. In addition, S_1 writes the output $f(x,y)$ that P_1 expects to receive at the end of the execution from P_2. We have that S_1 outputs

$$\big(x, r_C, S_1^{\mathrm{OT}}\big(k_{n+1}^0, k_{n+1}^1\big), \ldots, S_1^{\mathrm{OT}}\big(k_{2n}^0, k_{2n}^1\big), f(x,y)\big) \tag{8}$$

This concludes the description of S_1. We now prove that

$$\bigl\{S_1\bigl(x, f(x,y)\bigr)\bigr\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \bigl\{\mathsf{view}_1^{\pi}(x,y)\bigr\}_{x,y\in\{0,1\}^*}$$

where $S_1(x, f(x,y))$ is as shown in (8), and π denotes Protocol 2. We first prove a hybrid argument over the simulated views for the oblivious transfers. That is, we define a hybrid distribution H_i in which the first i oblivious transfers are simulated and the last $n-i$ are real. Formally, let $H_i(x,y,r_C)$ denote the distribution

$$\bigl\{\bigl(x, r_C, S_1^{\text{OT}}\bigl(k_{n+1}^0, k_{n+1}^1\bigr), \ldots, S_1^{\text{OT}}\bigl(k_{n+i}^0, k_{n+i}^1\bigr), R_1^{\text{OT}}\bigl(\bigl(k_{n+i+1}^0, k_{n+i+1}^1\bigr), y_{i+1}\bigr), \ldots,$$
$$R_1^{\text{OT}}\bigl(\bigl(k_{2n}^0, k_{2n}^1\bigr), y_n\bigr), f(x,y)\bigr)\bigr\}$$

where $R_1^{\text{OT}}((k_{n+j}^0, k_{n+j}^1), y_j)$ denotes the *real transcript* from $\mathsf{view}_1^{\text{OT}}((k_{n+j}^0, k_{n+j}^1), y_j)$. Notice that the keys k_{n+j}^0, k_{n+j}^1 here are as defined by the garbled circuit, when generated with the random tape r_C. Notice also that when r_C is uniformly chosen, $H_n(x,y,r_C)$ equals the distribution that appears in (8); i.e., it equals $S_1(x, f(x,y))$. Furthermore, $H_0(x,y,r_C)$ is almost the same as $\mathsf{view}_1^{\pi}(x,y)$; the only difference is that the last component of H_0 equals $f(x,y)$, whereas the last component of $\mathsf{view}_1^{\pi}(x,y)$ is the message that P_2 would send P_1 in the last message of the protocol. For simplicity, from here on we will assume that x, y, r_C are all of the same length and, in particular, are of length n.

We now prove that $\{H_0(x,y,r_C)\} \overset{\text{c}}{\equiv} \{H_n(x,y,r_C)\}$. By contradiction, assume that there exists a probabilistic polynomial-time distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n's (and $x, y, r_C \in \{0,1\}^n$),

$$\bigl|\Pr\bigl[D\bigl(H_0(x,y,r_C)\bigr)=1\bigr] - \Pr\bigl[D\bigl(H_n(x,y,r_C)\bigr)=1\bigr]\bigr| > \frac{1}{p(n)}$$

It follows that there exists i such that for infinitely many x, y, r_C,

$$\bigl|\Pr\bigl[D\bigl(H_i(x,y,r_C)\bigr)=1\bigr] - \Pr\bigl[D\bigl(H_{i+1}(x,y,r_C)\bigr)=1\bigr]\bigr| > \frac{1}{np(n)}$$

We now use D to contradict the security of the oblivious transfer protocol. First, notice that the only difference between $H_i(x,y,r_C)$ and $H_{i+1}(x,y,r_C)$ is that the random-tape and transcript of the $(i+1)$th oblivious transfer are according to $\mathsf{view}_1^{\text{OT}}((k_{n+i+1}^0, k_{n+i+1}^1), y_{i+1})$ in H_i and according to $S_1^{\text{OT}}(k_{n+i+1}^0, k_{n+i+1}^1)$ in H_{i+1}. Furthermore, given x, y, r_C, i, and a view v (which is either $\mathsf{view}_1^{\text{OT}}((k_{n+i+1}^0, k_{n+i+1}^1), y_{i+1})$ or $S_1^{\text{OT}}(k_{n+i+1}^0, k_{n+i+1}^1)$), it is possible to construct a distribution H such that if v is from $\mathsf{view}_1^{\text{OT}}$, then $H = H_i(x,y,r_C)$ and if v is from S_1^{OT}, then $H = H_{i+1}(x,y,r_C)$. It therefore follows that for infinitely many inputs, it is possible to distinguish the view of P_1 in a real oblivious transfer execution from its simulated view with the same probability that it is possible to distinguish $H_i(x,y,r_C)$ from $H_{i+1}(x,y,r_C)$. However, this contradicts the security of the oblivious transfer protocol. We therefore conclude that $\{H_0(x,y,r_C)\} \overset{\text{c}}{\equiv} \{H_n(x,y,r_C)\}$. (We remark that the distinguisher that we construct here is *nonuniform* because it needs to have x, y, r_C, and i. For this reason, we defined nonuniform indistinguishability; see the beginning of Sect. 2.) Until now, we have shown that

$$\left\{S_1\big(x, f(x,y)\big)\right\} \overset{c}{\equiv} \left\{\big(x, r_C, R_1^{\mathrm{OT}}\big(\big(k_n^0, k_n^1\big), y_1\big), \ldots, R_1^{\mathrm{OT}}\big(\big(k_{2n}^0, k_{2n}^1\big), y_n\big), f(x,y)\big)\right\} \quad (9)$$

However, this does not quite suffice because we cannot just assume that P_2 sends the correct value $f(x, y)$ to P_1 in a real execution (notice that in the right-hand distribution in (9), the last message received by P_1 in the view is $f(x, y)$). We now show that

$$\begin{aligned}&\left\{\big(x, r_C, R_1^{\mathrm{OT}}\big(\big(k_n^0, k_n^1\big), y_1\big), \ldots, R_1^{\mathrm{OT}}\big(\big(k_{2n}^0, k_{2n}^1\big), y_n\big), f(x,y)\big)\right\}\\ &\quad \overset{c}{\equiv} \left\{\big(x, r_C, R_1^{\mathrm{OT}}\big(\big(k_n^0, k_n^1\big), y_1\big), \ldots, R_1^{\mathrm{OT}}\big(\big(k_{2n}^0, k_{2n}^1\big), y_n\big), \mathsf{msg}_3(P_2 \to P_1)\big)\right\} \quad (10)\end{aligned}$$

where $\mathsf{msg}_3(P_2 \to P_1)$ denotes the message that P_2 sends to P_1 in Step 3 of the protocol. Notice that the only difference between these distributions is whether the last component equals $f(x, y)$ or the message sent by P_2 to P_1 in Step 3. Recall that this message sent by P_2 is exactly the output that it obtains from the garbled circuit. Now, by Claim 6, the output obtained by P_2 from the garbled circuit when P_1 sends it the keys corresponding to x and it receives the keys corresponding to y from the oblivious transfers, equals $f(x, y)$, except with negligible probability. By the security of the oblivious transfer protocol, we have that P_2 receives the keys corresponding to y, except with negligible probability. (This follows immediately from the correctness condition, which is implied by the definition of security.) Therefore, $\mathsf{msg}_3(2 \to 1) = f(x, y)$, except with negligible probability, and (10) follows. Notice now that

$$\left\{\big(x, r_C, R_1^{\mathrm{OT}}\big(\big(k_n^0, k_n^1\big), y_1\big), \ldots, R_1^{\mathrm{OT}}\big(\big(k_{2n}^0, k_{2n}^1\big), y_n\big), \mathsf{msg}_3(2 \to 1)\big)\right\} \equiv \left\{\mathsf{view}_1^{\pi}(x, y)\right\} \quad (11)$$

and so by combining (9) to (11) the proof of this case is concluded.

Case 2—P_2 is corrupted

In this case, we construct a simulator S_2 that is given input $(y, f(x, y))$ and generates the view of P_2 in Protocol 2. Notice that P_2 expects to receive a garbled circuit, and so S_2 must generate such a circuit. Furthermore, this circuit must be such that P_2 would obtain $f(x, y)$ when computing the circuit according to the protocol instructions. Of course, S_2 cannot just honestly generate the circuit, because it does not know x. (Without knowing x, it would not know which of the keys $k_1^0, k_1^1, \ldots, k_n^0, k_n^1$ to hand to P_2.) It therefore generates a "fake" garbled circuit that always evaluates to $f(x, y)$, irrespective of which keys are used. This is achieved by using gate tables in which all four entries encrypt the same key, and therefore the values of the input wires do not affect the value of the output wire. The crux of the proof is in showing that this circuit is indistinguishable from the real garbled circuit that P_2 receives in a real execution. In order to show this, we use a hybrid argument. We first show that P_2's view in a *real* execution of the protocol is indistinguishable from a hybrid distribution $H_{\mathrm{OT}}(x, y)$ in which the real oblivious transfers are replaced with simulated ones. Next, we consider a series of hybrids $H_i(x, y)$ in which one gate at a time is replaced in the real garbled circuit. The hybrid distributions are such that $H_0(x, y)$ contains a real garbled circuit (and therefore equals $H_{\mathrm{OT}}(x,y)$). In contrast, distribution $H_{|C|}(x,y)$ contains the same fake circuit constructed by S_2 (and, as we will see, therefore equals $S_2(y,f(x,y))$). By a standard hybrid argument, it follows that a distinguisher between $H_0(x,y)$ and $H_{|C|}(x,y)$ can be used to distinguish between two successive hybrids. However, the security of the encryption scheme that is used for generating the gate tables ensures that neighboring hybrids are computationally indistinguishable. We conclude that $H_0(x,y)$ is indistinguishable from $H_{|C|}(x,y)$, and so $\{S_2(y,f(x,y))\} \overset{\mathrm{c}}{\equiv} \{\mathsf{view}_2^{\pi}(x,y)\}$.

We now formally describe S_2. Simulator S_2 begins by constructing a fake garbled circuit, denoted $\tilde{G}(C)$. This is accomplished as follows. For every wire w_i in the circuit C, simulator S_2 chooses two random keys k_i and k_i'. Next, the gates are computed: let g be a gate with input wires w_i, w_j and output wire w_ℓ. Then, g contains encryptions of the single key k_ℓ under *all four combinations* of the keys k_i, k_i', k_j, k_j' that are associated with the input wires to g (in contrast, the key k_ℓ' is not encrypted at all). That is, S_2 computes the following values:

$$c_{0,0} = E_{k_i}\big(E_{k_j}(k_\ell)\big)$$
$$c_{0,1} = E_{k_i}\big(E_{k_j'}(k_\ell)\big)$$
$$c_{1,0} = E_{k_i'}\big(E_{k_j}(k_\ell)\big)$$
$$c_{1,1} = E_{k_i'}\big(E_{k_j'}(k_\ell)\big)$$

and writes them in random order. This is carried out for all of the gates of the circuit. It remains to describe how the output decryption tables are constructed. Denote the n-bit output $f(x,y)$ by $z_1 \cdots z_n$ (recall that this is part of S_2's input), and denote the circuit-output wires by $w_{m-n+1},\dots,w_m$. In addition, for every $i=1,\dots,n$, let k_{m-n+i} be the (single) key encrypted in the gate from which wire w_{m-n+i} left, and let k_{m-n+i}' be the other key (as described above). Then, the output decryption table for wire w_{m-n+i} is given by $[(0,k_{m-n+i}),(1,k_{m-n+i}')]$ if $z_i=0$ and $[(0,k_{m-n+i}'),(1,k_{m-n+i})]$ if $z_i=1$. This completes the description of the construction of the fake garbled circuit $\tilde{G}(C)$. (Notice that the keys $k_{m-n+1},\dots,k_m$ decrypt to $z_1\cdots z_n = f(x,y)$ exactly.)

Next, S_2 generates the view of P_2 in the phase where it obtains the keys. First, in the simulated view, it sets the keys that P_2 receives from P_1 in Step 2 of Protocol 2 to be $k_1,\dots,k_n$. (Recall that $w_1,\dots,w_n$ are the circuit-input wires associated with P_1's input x and that the keys for these wires are $k_1,k_1',\dots,k_n,k_n'$. Here, S_2 takes the keys $k_1,\dots,k_n$. However, it could have taken $k_1',\dots,k_n'$ or any other combination, and this would make no difference.) Next, let S_2^{OT} be the simulator that is guaranteed to exist for the oblivious transfer protocol. Then, for every $i=1,\dots,n$, simulator S_2 invokes the simulator S_2^{OT} upon input (y_i,k_{n+i}) in order to obtain P_2's view in the ith oblivious transfer. (Here y_i and k_{n+i} are P_2's respective input and output in the ith oblivious transfer. As above, we use the keys $k_{n+1},\dots,k_{2n}$ associated with the input wires for y. However, this choice is arbitrary, and we could have used $k_{n+1}',\dots,k_{2n}'$ or any other combination.) Recall that the view generated by S_2^{OT} is made up of the input (in this case y_i), a random tape, and a transcript of messages received. Recall also that by convention, we place the entire view in the oblivious transfer (including the random-tape) together.

We therefore have that S_2 outputs

$$\big(y, \tilde{G}(C), k_1, \ldots, k_n, S_2^{\text{OT}}(y_1, k_{n+1}), \ldots, S_2^{\text{OT}}(y_n, k_{2n})\big)$$

This concludes the description of S_2. We now prove that

$$\big\{S_2\big(y, f(x,y)\big)\big\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \big\{\mathsf{view}_2^{\pi}(x,y)\big\}_{x,y\in\{0,1\}^*}$$

First, observe that

$$\begin{aligned}&\big\{\mathsf{view}_2^{\pi}(x,y)\big\}\\ &\quad\equiv \big\{\big(y, G(C), k_1^{x_1}, \ldots, k_n^{x_n}, R_2^{\text{OT}}\big(\big(k_{n+1}^0, k_{n+1}^1\big), y_1\big), \ldots, R_2^{\text{OT}}\big(\big(k_{2n}^0, k_{2n}^1\big), y_n\big)\big)\big\}\end{aligned}$$

where $R_2^{\text{OT}}((k_{n+i}^0, k_{n+i}^1), y_i)$ denotes the *real transcript* from $\mathsf{view}_2^{\text{OT}}((k_{n+i}^0, k_{n+i}^1), y_i)$. We also denote the hybrid distribution where the real oblivious transfers are replaced by simulated ones by

$$H_{\text{OT}}(x,y) = \big(y, G(C), k_1^{x_1}, \ldots, k_n^{x_n}, S_2^{\text{OT}}\big(y_1, k_{n+1}^{y_1}\big), \ldots, S_2^{\text{OT}}\big(y_n, k_{2n}^{y_n}\big)\big)$$

We stress that in the hybrid distribution H_{OT}, the garbled circuit $G(C)$ that appears is the real one and not the fake one. We first claim that

$$\big\{H_{\text{OT}}(x,y)\big\}_{x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \big\{\mathsf{view}_2^{\pi}(x,y)\big\}_{x,y\in\{0,1\}^*} \tag{12}$$

The only difference between the distributions in (12) is due to the fact that simulated views of the oblivious transfers are provided instead of real ones. Indistinguishability therefore follows from the security of the oblivious transfer protocol. The formal proof of this is almost identical to the case that P_1 is corrupted and is therefore omitted.

Next, we consider a series of hybrid experiments $H_i(x,y)$ in which one gate at a time is replaced in the real garbled circuit $G(C)$ until the result is the fake garbled circuit $\tilde{G}(C)$. Before we do this, we consider an alternative way of constructing the fake garbled circuit $\tilde{G}(C)$. This alternative construction uses knowledge of both inputs x and y but results in exactly the same fake garbled circuit as that constructed by S_2 that is given only y and $f(x,y)$. (This is therefore just a mental experiment or a different description of S_2. Nevertheless, it is helpful in describing the proof.)

The alternative construction works by first traversing the circuit from the circuit-input wires to the circuit-output wires and labeling all keys as active or inactive. Intuitively, a key is active if it is used in order to compute the garbled circuit upon input (x,y); otherwise it is inactive. Formally, a key k_a^{α} that is associated with wire w_a is active if when computing the nongarbled circuit C on input (x,y), the bit that is obtained on wire w_a equals α. As expected, an inactive key is just any key that is not active. Now, the alternative construction of $\tilde{G}(C)$ works by first constructing the real garbled circuit $G(C)$. Next, using knowledge of both x and y, all keys in $G(C)$ are labeled active or inactive (given x and y, it is possible to compute $C(x,y)$ and obtain the real values on each wire). Finally, $\tilde{G}(C)$ is obtained by replacing each gate g as follows: Let w_a be the wire that exits gate g. Then, recompute g by encrypting the *active key* on wire w_a with all four combinations of the (active and inactive) keys that are on the wires that enter g. This completes the alternative construction. We claim that the circuit obtained in this alternative construction is identically distributed to the circuit constructed by $S_2(x, f(x, y))$. First, in both constructions, all gates contain encryptions of a single key only. Second, in both constructions, the order of the ciphertexts in each gate is random. Finally, in both constructions, the output decryption tables yield the same result (i.e., exactly $f(x, y)$). This last observation is due to the fact that in the alternative construction, the output decryption table decrypts active keys to $f(x, y)$, and these active keys are the only ones encrypted in the gates from which the circuit-output wires exit. Likewise, in the circuit $\tilde{G}(C)$, the only keys encrypted in the gates from which the circuit-output wires exit are the keys that decrypt to $f(x, y)$.

Before proceeding, we order the gates $g_1, \ldots, g_{|C|}$ of the circuit C as follows: if the input wires to a gate g_ℓ come from gates g_i and g_j, then $i < \ell$ and $j < \ell$; this is called a topological sort of the circuit. We are now ready to define the hybrid experiment $H_i(x, y)$.

Hybrid experiment $H_i(x, y)$. In this experiment the view of P_2 in the oblivious transfers is generated in exactly the same way as in $H_{\rm OT}(x, y)$. However, the garbled circuit is constructed differently. As in the alternative construction of $\tilde{G}(C)$, the first step is to construct the real garbled circuit $G(C)$ and then use x and y in order to label all keys in $G(C)$ as active or inactive. Next, the first i gates $g_1, \ldots, g_i$ are modified as in the alternative construction. That is, let w_a be the wire that exits gate g_j for $1 \leq j \leq i$. Then, recompute g_j by encrypting the *active key* on wire w_a with all four combinations of the (active and inactive) keys that are on the wires that enter g_j. The remaining gates $g_{i+1}, \ldots, g_{|C|}$ are left unmodified and are therefore as in the real garbled circuit $G(C)$.

We claim that the distribution $\{H_0(x, y)\}$ equals $\{H_{\rm OT}(x, y)\}$. This follows from the fact that the only difference is that in $H_0(x, y)$ the keys are labeled active or inactive. However, since nothing is done with this labeling, there is no difference in the resulting distribution. Next, notice that in $H_{|C|}(x, y)$, the circuit that appears in the distribution is exactly the fake garbled circuit $\tilde{G}(C)$ as constructed by S_2. This follows immediately from the fact that in $H_{|C|}$ all gates are replaced, and so the circuit obtained is exactly that of the full alternative construction described above.

We wish to show that $\{H_0(x, y)\} \stackrel{c}{\equiv} \{H_{|C|}(x, y)\}$. Intuitively, this follows from the indistinguishability of encryptions. Specifically, the only difference between H_0 and $H_{|C|}$ is that the circuit in H_0 is made up of gates that contain encryptions of active and inactive keys, whereas the circuit in $H_{|C|}$ is made up of gates that contain encryptions of active keys only. Since only active keys are seen by P_2 during the computation of the garbled circuit, the difference between H_0 and $H_{|C|}$ cannot be detected.

We prove that $\{H_0(x, y)\} \stackrel{c}{\equiv} \{H_{|C|}(x, y)\}$ using a hybrid argument. That is, assume that there exists a nonuniform probabilistic polynomial-time distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n (and values $x, y \in \{0, 1\}^n$), $|\Pr[D(H_0(x, y)) = 1] - \Pr[D(H_{|C|}(x, y)) = 1]| > 1/p(n)$. Then, it follows that there exists i such that $|\Pr[D(H_{i-1}(x, y)) = 1] - \Pr[D(H_i(x, y)) = 1]| > 1/|C|p(n)$. We use D and x, y, i in order to construct a nonuniform probabilistic polynomial-time distinguisher $\mathcal{A}_E$ for the encryption scheme (G, E, D). The high-level idea here is for $\mathcal{A}_E$ to receive some ciphertexts from which it will construct a partially real and partially fake garbled circuit $G'(C)$. However, the construction will be such that if the ciphertexts received were of one "type," then the resulting circuit is according to $H_{i-1}(x, y)$. However, if the ciphertexts received were of another "type," then the resulting circuit is according to $H_i(x, y)$. In this way, the ability to successfully distinguish $H_{i-1}(x, y)$ from $H_i(x, y)$ yields the ability to distinguish ciphertexts, in contradiction to the security of the encryption scheme. We now formally prove the above intuition, using Lemma 4 that states that (G, E, D) is secure under chosen double encryption.

A Concrete Case First, let us consider the concrete case where g_i is an OR gate and where wires w_a and w_b enter g_i, and wire w_c exits g_i. Furthermore, assume that the wires w_a and w_b enter gate g_i and no other gate. Finally, assume that when the inputs to the circuit are x and y, the wire w_a obtains the bit 0, and the wire w_b obtains the bit 1. Then, it follows that the keys k_a^0 and k_b^1 are active, and the keys $\boldsymbol{k_a^1}$ and $\boldsymbol{k_b^0}$ are inactive (we mark the inactive keys in bold in order to distinguish them from the active ones). Likewise, the key k_c^1 is active (because $g_i(0, 1) = 0 \vee 1 = 1$), and the key $\boldsymbol{k_c^0}$ is inactive. The difference between a real garbled gate g_i and a fake garbled gate g_i is with respect to the encrypted values. Specifically, the real garbled OR gate g_i contains the following values:

$$E_{k_a^0}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^0)\big), \quad E_{k_a^0}\big(E_{k_b^1}(k_c^1)\big), \quad \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^1)\big), \quad \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(E_{k_b^1}(k_c^1)\big) \tag{13}$$

In contrast, the fake garbled OR gate g_i contains the following values, which are all encryptions of the active value k_c^1 (recall that the input to g_i equals 0 and 1, and so the output is 1):

$$E_{k_a^0}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^1)\big), \quad E_{k_a^0}\big(E_{k_b^1}(k_c^1)\big), \quad \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^1)\big), \quad \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(E_{k_b^1}(k_c^1)\big) \tag{14}$$

Thus, in this *concrete* case, the indistinguishability between the gates depends on the indistinguishability of a single encryption (of k_c^0 versus k_c^1) under the inactive key $\boldsymbol{k_b^0}$. (In other cases, the indistinguishability may depend on both inactive keys $\boldsymbol{k_a^1}$ and $\boldsymbol{k_b^0}$ and may depend on more than one encryption under a key; see the general case below.) It is not difficult to show here that indistinguishability follows directly from the chosen-plaintext security of the encryption scheme E with key $\boldsymbol{k_b^0}$. Nevertheless, it follows immediately from the definition of $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{double}}$ and security under chosen double encryption (see Definition 3 in Sect. 3.1). Specifically, we construct a nonuniform probabilistic polynomial-time machine $\mathcal{A}_E$ for $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}$ such that

$$\left|\Pr\left[\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 1) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 0) = 1\right]\right| \geq \frac{1}{|C| p(n)}$$

Upon input 1^n, machine $\mathcal{A}_E$ outputs keys $k_a^0, k_b^1 \leftarrow G(1^n)$ and message triples (k_c^0, k_c^1, k_c^1) and (k_c^1, k_c^1, k_c^1). By the experiment $\mathsf{Expt}^{\mathsf{double}}$, two keys are chosen. For the sake of consistency, we denote them $\boldsymbol{k_a^1}$ and $\boldsymbol{k_b^0}$. Then, machine $\mathcal{A}_E$ receives either the ciphertexts

$$\big\langle E_{k_a^0}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^0)\big), \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(E_{k_b^1}(k_c^1)\big), \boldsymbol{E}_{\boldsymbol{k_a^1}}\big(\boldsymbol{E}_{\boldsymbol{k_b^0}}(k_c^1)\big)\big\rangle \tag{15}$$

or the ciphertexts

$$\left\langle E_{k_a^0}\left(\boldsymbol{E}_{\boldsymbol{k_b^0}}\left(k_c^1\right)\right), \boldsymbol{E}_{\boldsymbol{k_a^1}}\left(E_{k_b^1}\left(k_c^1\right)\right), \boldsymbol{E}_{\boldsymbol{k_a^1}}\left(\boldsymbol{E}_{\boldsymbol{k_b^0}}\left(k_c^1\right)\right)\right\rangle \tag{16}$$

depending on whether $\sigma = 0$ (the first case) or $\sigma = 1$ (the second case); note that the only difference is that in the first case the first plaintext is k_c^0 and in the second case the first plaintext is k_c^1. Denote the ciphertexts received by $\mathcal{A}_E$ by (c_1, c_2, c_3). Now, $\mathcal{A}_E$ first computes the value $c = E_{k_a^0}(E_{k_b^1}(k_c^1))$; it can do this by itself because it knows both k_a^0 and k_b^1, as well as k_c^1. Next, given c, $\mathcal{A}_E$ generates the tuple $\langle c_1, c, c_3, c_2\rangle$. The important point to notice here is that if $\mathcal{A}_E$ received the ciphertexts in (15), then $\langle c_1, c, c_3, c_2\rangle$ is identical to the ciphertexts in (13). On the other hand, if $\mathcal{A}_E$ received the ciphertexts in (16), then $\langle c_1, c, c_3, c_2\rangle$ is identical to the ciphertexts in (14). Therefore, if it is possible to distinguish between the gates in (13) and (14) with nonnegligible probability, then $\mathcal{A}_E$ can succeed in $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}$ with nonnegligible probability, in contradiction to the security of the encryption scheme.

This does not yet suffice because we must still show how $\mathcal{A}_E$ can generate the rest of the H_{i-1} or H_i distributions. Notice that $\mathcal{A}_E$ knows the active keys that enter g_i (because it chooses them itself) but does not know the inactive keys. We therefore show that the distributions can be constructed without knowledge of the inactive keys $\boldsymbol{k_a^1}$ and $\boldsymbol{k_b^0}$. In order to show this, we distinguish between two cases:

1. *Case 1—w_b is a circuit-input wire:* In this case, the keys associated with wire w_b do not appear in any gates g_j for $j < i$. However, keys that are associated with circuit-input wires do appear in the distributions H_{i-1} and H_i: the keys $k_i^{x_i}$ appear directly, and the keys $k_{n+i}^{y_i}$ are used to generate the view of P_2 in the oblivious transfers. Nevertheless, notice that the keys used here are all *active*. Therefore, $\mathcal{A}_E$ can construct the distributions, as required. We note that $\mathcal{A}_E$ uses the keys k_c^0 and k_c^1 that it receives in its experiment in order to construct the gates into which wire w_c enters.
2. *Case 2—w_b is not a circuit-input wire:* In this case, the keys associated with wire w_b can appear only in the gate g_j from which w_b exits. However, by our ordering of the gates, $j < \ell$. Therefore, in both H_{i-1} and H_i, gate g_j contains encryptions of the *active* key k_b^0 only. It follows that $\mathcal{A}_E$ can construct the rest of the distribution, as required. (Again, as above, $\mathcal{A}_E$ uses the keys k_c^0 and k_c^1 in this construction.)

Now, as we have shown above, if $\mathcal{A}_E$ participates in $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 0)$, then the gate g_i is constructed as for a real garbled circuit. In contrast, if $\mathcal{A}_E$ participates in $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 1)$, then the gate g_i is constructed as for a fake garbled circuit. The only dependence between the gate g_i and the rest of the distribution H_{i-1} or H_i is with respect to the keys k_c^0 and k_c^1; however, these are known to $\mathcal{A}_E$ and used appropriately. We therefore conclude that if $\mathcal{A}_E$ participates in $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 0)$, then it generates a distribution H that equals $H_{i-1}(x, y)$. In contrast, if it participates in $\mathsf{Expt}_{\mathcal{A}_E}^{\mathsf{double}}(n, 1)$, then it generates a distribution H that equals $H_i(x, y)$. Distinguisher $\mathcal{A}_E$ concludes by running machine D on the distribution H and outputting whatever D does. By the contradicting assumption, machine D distinguishes $H_{i-1}(x, y)$ from $H_i(x, y)$ with probability

$1/|C|p(n)$. That is, we have that for infinitely many n,

$$\begin{aligned}&\bigl|\Pr\bigl[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}_E}(n,0)=1\bigr]-\Pr\bigl[\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}_E}(n,1)=1\bigr]\bigr|\\&\quad=\bigl|\Pr\bigl[D\bigl(H_{i-1}(x,y)\bigr)=1\bigr]-\Pr\bigl[D\bigl(H_i(x,y)\bigr)=1\bigr]\bigr|>\frac{1}{|C|p(n)}\end{aligned}$$

in contradiction to the security of the encryption scheme. It follows that $\{H_0(x,y)\}\stackrel{\mathrm{c}}{\equiv}\{H_{|C|}(x,y)\}$. Having proven the argument with respect to a concrete case, we now move to the general case.

The General Case Let g_i be an arbitrary gate, let w_a and w_b be the wires entering g_i, and let w_c be the wire that exits g_i. Furthermore, let α and β be the respective values obtained on w_a and w_b in $C(x,y)$. Note that this means that k_a^α and k_b^β are active, and $\boldsymbol{k_a^{1-\alpha}}$ and $\boldsymbol{k_b^{1-\beta}}$ are inactive. Then, the real garbled gate g_i contains the following values (in random order):

$$\begin{aligned}&E_{k_a^\alpha}\bigl(E_{k_b^\beta}\bigl(k_c^{g_i(\alpha,\beta)}\bigr)\bigr),\quad E_{k_a^\alpha}\bigl(\boldsymbol{E}_{\boldsymbol{k_b^{1-\beta}}}\bigl(k_c^{g_i(\alpha,1-\beta)}\bigr)\bigr),\\&\boldsymbol{E}_{\boldsymbol{k_a^{1-\alpha}}}\bigl(E_{k_b^\beta}\bigl(k_c^{g_i(1-\alpha,\beta)}\bigr)\bigr),\quad \boldsymbol{E}_{\boldsymbol{k_a^{1-\alpha}}}\bigl(\boldsymbol{E}_{\boldsymbol{k_b^{1-\beta}}}\bigl(k_c^{g_i(1-\alpha,1-\beta)}\bigr)\bigr)\end{aligned}\tag{17}$$

In contrast, the fake garbled gate g_i contains the following values, which are all encryptions of the active value $k_c^{g_i(\alpha,\beta)}$:

$$\begin{aligned}&E_{k_a^\alpha}\bigl(E_{k_b^\beta}\bigl(k_c^{g_i(\alpha,\beta)}\bigr)\bigr),\quad E_{k_a^\alpha}\bigl(\boldsymbol{E}_{\boldsymbol{k_b^{1-\beta}}}\bigl(k_c^{g_i(\alpha,\beta)}\bigr)\bigr),\\&\boldsymbol{E}_{\boldsymbol{k_a^{1-\alpha}}}\bigl(E_{k_b^\beta}\bigl(k_c^{g_i(\alpha,\beta)}\bigr)\bigr),\quad \boldsymbol{E}_{\boldsymbol{k_a^{1-\alpha}}}\bigl(\boldsymbol{E}_{\boldsymbol{k_b^{1-\beta}}}\bigl(k_c^{g_i(\alpha,\beta)}\bigr)\bigr)\end{aligned}\tag{18}$$

Thus, the indistinguishability between the gates depends on the indistinguishability of encryptions under the inactive keys $\boldsymbol{k_a^{1-\alpha}}$ and $\boldsymbol{k_b^{1-\beta}}$. As above, we use $\mathsf{Expt}^{\mathsf{double}}$ and security under chosen double encryption. The gate is generated in exactly the same way here as in the concrete case. Now, in the restricted case that both wires w_a and w_b enter the gate g_i only, it is possible to proceed in the same way as in the concrete case above. However, in the more general case, wires w_a and w_b may enter multiple gates $g^a_{i_1},\ldots,g^a_{i_j}$ and $g^b_{i_1},\ldots,g^b_{i_\ell}$, respectively. In this case, $\mathcal{A}_E$ *cannot* construct the rest of the circuit given only the active keys, because the inactive keys $\boldsymbol{k_a^{1-\alpha}}$ and $\boldsymbol{k_b^{1-\beta}}$ are used in more than one gate. (We stress that in order to prove the indistinguishability of the neighboring hybrid H_{i-1} and H_i, it is crucial that $\mathcal{A}_E$ is not given these inactive keys. Therefore, it cannot construct these other gates itself.) This is solved by using the *special chosen-plaintext attack* of $\mathsf{Expt}^{\mathsf{double}}$. Recall that in $\mathsf{Expt}^{\mathsf{double}}$, the adversary has access to oracles $\overline{E}(\cdot,k_1',\cdot)$ and $\overline{E}(k_0',\cdot,\cdot)$, where $k_0'=\boldsymbol{k_a^{1-\alpha}}$ and $k_1'=\boldsymbol{k_b^{1-\beta}}$. Here, this means that the adversary can ask for encryptions under these inactive keys, as needed for constructing all of the other gates $g^a_{i_1},\ldots,g^a_{i_j}$ and $g^b_{i_1},\ldots,g^b_{i_\ell}$ that use them. Once again, we have that in $\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}_E}(n,0)$ the distribution generated by $\mathcal{A}_E$ is exactly that of $H_{i-1}(x,y)$, whereas in $\mathsf{Expt}^{\mathsf{double}}_{\mathcal{A}_E}(n,1)$ the distribution generated by $\mathcal{A}_E$ is exactly that of $H_i(x,y)$. Therefore, as above, we conclude that $H_{i-1}(x,y)$ is indistinguishable from $H_i(x,y)$, and so $\{H_0(x,y)\} \stackrel{\mathrm{c}}{\equiv} \{H_{|C|}(x,y)\}$.

Concluding the Proof Having proven that $\{H_0(x,y)\} \stackrel{\mathrm{c}}{\equiv} \{H_{|C|}(x,y)\}$, we obtain that

$$\begin{aligned}
&\bigl\{\bigl(y, \tilde{G}(C), k_1^{x_1}, \ldots, k_n^{x_n}, S_2^{\mathrm{OT}}\bigl(y_1, k_{n+1}^{y_1}\bigr), \ldots, S_2^{\mathrm{OT}}\bigl(y_n, k_{2n}^{y_n}\bigr)\bigr)\bigr\} \\
&\quad \stackrel{\mathrm{c}}{\equiv} \bigl\{\bigl(y, G(C), k_1^{x_1}, \ldots, k_n^{x_n}, S_2^{\mathrm{OT}}\bigl(y_1, k_{n+1}^{y_1}\bigr), \ldots, S_2^{\mathrm{OT}}\bigl(y_n, k_{2n}^{y_n}\bigr)\bigr)\bigr\} \\
&\quad = \bigl\{H_{\mathrm{OT}}(x,y)\bigr\} \qquad (19)
\end{aligned}$$

Notice that the first distribution in (19) looks almost the same as the distribution $\{S_2(y, f(x,y))\}$. The only difference is that in $S_2(y, f(x,y))$ the keys $k_1, \ldots, k_n, k_{n+1}, \ldots, k_{2n}$ are used instead of the keys $k_1^{x_1}, \ldots, k_n^{x_n}, k_{n+1}^{y_1}, \ldots, k_{2n}^{y_n}$. That is, the keys that S_2 takes for the circuit-input wires have no correlation to the actual input (unlike in a real execution). However, in the fake garbled circuit $\tilde{G}(C)$, there is no difference between k_i and k_i' because all combinations of keys are used to encrypt the same (active) key. Thus, the distribution over the keys $k_1, \ldots, k_n, k_{n+1}, \ldots, k_{2n}$ and $k_1^{x_1}, \ldots, k_n^{x_n}, k_{n+1}^{y_1}, \ldots, k_{2n}^{y_n}$ are identical in the fake garbled-circuit construction. (Notice that there is not even a distinction between k_i^0 and k_i^1 in the fake garbled circuit. Nevertheless, one could define $k_i^0 = k_i$ and $k_i^1 = k_i'$, and the result would still be that the distributions are identical.) We therefore obtain that the first distribution in (19) is actually identical to the distribution $\{S_2(y, f(x,y))\}$, and so

$$\bigl\{S_2\bigl(y, f(x,y)\bigr)\bigr\}_{x,y\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \bigl\{H_{\mathrm{OT}}(x,y)\bigr\}_{x,y\in\{0,1\}^*}$$

Recalling that, by (12), $\{H_{\mathrm{OT}}(x,y)\} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}_2^{\pi}(x,y)\}$, we conclude that

$$\bigl\{S_2\bigl(y, f(x,y)\bigr)\bigr\}_{x,y\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \bigl\{\mathsf{view}_2^{\pi}(x,y)\bigr\}_{x,y\in\{0,1\}^*}$$

as required. □

By Theorem 5 it is possible to securely compute the oblivious transfer functionality assuming the existence of enhanced trapdoor permutations. Furthermore, secure encryption schemes as required in Theorem 7 can be constructed from one-way functions and so also from enhanced trapdoor permutations. Finally, recall that given a secure protocol for deterministic same-output functionalities, it is possible to obtain a secure protocol for arbitrary probabilistic functionalities. Combining these facts with Theorem 7, we obtain the following corollary:

Corollary 8. *Let $f = (f_1, f_2)$ be a probabilistic functionality. Then, assuming the existence of enhanced trapdoor permutations, there exists a constant-round protocol that securely computes f in the presence of static semi-honest adversaries.*

Acknowledgements

We would like to thank Yael Tauman Kalai and the anonymous referees for many helpful comments on the write-up.

References

[1] D. Beaver, Foundations of secure interactive computing, in *CRYPTO'91*. LNCS, vol. 576 (Springer, Berlin, 1991), pp. 377–391

[2] D. Beaver, Correlated pseudorandomness and the complexity of private computations, in 28*th STOC* (1996), pp. 479–488

[3] D. Beaver, S. Micali, P. Rogaway, The round complexity of secure protocols, in 22*nd STOC* (1990), pp. 503–513

[4] R. Canetti, Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **13**(1), 143–202 (2000)

[5] C. Dwork, M. Naor, O. Reingold, Immunizing encryption schemes from decryption errors, in *Eurocrypt 2004*. LNCS, vol. 3027 (Springer, Berlin, 2004), pp. 342–360

[6] S. Even, O. Goldreich, A. Lempel, A randomized protocol for signing contracts. *Commun. ACM* **28**(6), 637–647 (1985)

[7] O. Goldreich, *Foundations of Cryptography; vol. 2: Basic Applications* (Cambridge University Press, Cambridge, 2004)

[8] O. Goldreich, S. Goldwasser, S. Micali, How to construct random functions. *J. ACM* **33**(4), 792–807 (1986)

[9] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game—a completeness theorem for protocols with honest majority, in 19*th STOC* (1987), pp. 218–229. For details, see [7]

[10] S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, in *CRYPTO'90*. LNCS, vol. 537 (Springer, Berlin, 1990), pp. 77–93

[11] S. Micali, P. Rogaway, Secure computation, in *CRYPTO'91*. LNCS, vol. 576 (Springer, Berlin, 1991), pp. 392–404

[12] M. Naor, B. Pinkas, R. Sumner, Privacy preserving auctions and mechanism design, in *The 1st ACM Conference on Electronic Commerce* (1999), pp. 129–139

[13] M. Rabin, How to exchange secrets by oblivious transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981

[14] P. Rogaway, The round complexity of secure protocols. MIT PhD thesis, June 1991

[15] A. Yao, How to generate and exchange secrets, in 27*th FOCS* (1986), pp. 162–167