MiniLEGO: Efficient Secure Two-Party Computation From General Assumptions

Tore Kasper Frederiksen[1], Thomas Pelle Jakobsen[1], Jesper Buus Nielsen[1], Peter Sebastian Nordholt[1], and Claudio Orlandi[1] *

Department of Computer Science, Aarhus University
{jot2re|tpj|jbn|psn|orlandi}@cs.au.dk

Abstract One of the main tools to construct secure two-party computation protocols are Yao garbled circuits. Using the cut-and-choose technique, one can get reasonably efficient Yao-based protocols with security against malicious adversaries. At TCC 2009, Nielsen and Orlandi [28] suggested to apply cut-and-choose at the gate level, while previously cut-and-choose was applied on the circuit as a whole. This idea allows for a speed up with practical significance (in the order of the logarithm of the size of the circuit) and has become known as the "LEGO" construction. Unfortunately the construction in [28] is based on a specific number-theoretic assumption and requires public-key operations per gate of the circuit. The main technical contribution of this work is a new XOR-homomorphic commitment scheme based on oblivious transfer, that we use to cope with the problem of connecting the gates in the LEGO construction. Our new protocol has the following advantages:

1. It maintains the efficiency of the LEGO cut-and-choose.
2. After a number of seed oblivious transfers linear in the security parameter, the construction uses only primitives from Minicrypt (i.e., private-key cryptography) per gate in the circuit (hence the name MiniLEGO).
3. MiniLEGO is compatible with all known optimization for Yao garbled gates (row reduction, free-XORs, point-and-permute).

1 Introduction

Secure two-party computation allows two parties to compute a function of their inputs while ensuring security properties such as the privacy of the inputs and the correctness of the outputs. The first protocol for secure two-party computation is Yao's garbled circuit [21, 32]. In recent years there has been a significant effort to bring secure computation into practice. These efforts resulted in terrific improvements in terms of algorithmic complexity, efficiency of implementations etc. (see e.g., [1, 2, 5, 7, 11–13, 15–20, 22–24, 27–31] and references therein). Perhaps the most interesting problem is

* Partially supported by the European Research Commission Starting Grant 279447 and the Danish National Research Foundation and The National Science Foundation of China (grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation. Tore and Thomas are supported by Danish Council for Independent Research Starting Grant 10-081612. Claudio is supported by The Danish Council for Independent Research (DFF) Grant 11-116416/FTP.

how to achieve protocols with security against malicious adversaries that are efficient enough to be used in practice.

In a nutshell, Yao's protocol works as follows: A constructs an encrypted version of the circuit to be computed (the "garbled circuit") and sends it to B who evaluates the encrypted circuit on encrypted inputs, thus learning nothing but the output of the computation. One of the main problems of this protocol is that if A is malicious she can encrypt a circuit different than the one B agreed on computing, with dramatic consequences for the correctness of the result and the privacy of B's input. One of the main tools to cope with this is the so called *cut-and-choose* technique: A prepares many copies of the encrypted circuit and B checks some of them for correctness. This induces a probability on the unopened circuits to be correct. Nielsen and Orlandi [28] presented a twist on this approach, known as LEGO: their approach consists of performing a cut-and-choose test at the gate level (instead of at circuit level), and allows to save a factor $O(\log(s))$ with s being the circuit size, in the computation and communication complexity w.r.t., "standard" cut-and-choose at the circuit level. However, the approach did not have a practical impact for the efficiency of Yao-based protocols, for several reasons:

1. *LEGO uses public-key primitives for each gate in the circuit:* Each gate has associated three commitments to its input/output keys. Those commitments are used for the "soldering" and need to be homomorphic. For this purpose LEGO uses Pedersen commitments. This is a drawback for the efficiency of the protocol (group operations, even in an elliptic curve, are orders of magnitude slower than symmetric primitives such as hash functions or private-key encryption). Moreover, it limits LEGO to discrete logarithm computational assumptions.
2. *LEGO is not compatible with known optimization for Yao's protocol:* Keys in LEGO are element of $\mathbb{Z}_p$ for some prime p, while using binary strings $\{0,1\}^t$ is more natural and standard. Therefore, it is not possible to use the "free-XOR" trick with LEGO, nor many of the others optimizations that are tailored for bit-string keys.
3. *LEGO has too many bricks:* there are many different kind of objects in LEGO (key-filters, not-two gates, etc.) that make the use of LEGO complex to understand and implement.

Contributions. In this paper, we present a generalization and a simplification of the LEGO approach. The main technical difference is to replace the Pedersen commitments with some XOR-homomorphic commitments based on oblivious transfer (OT) which we believe is of independent interest and might be used in other applications. We take this direction as OT can be efficiently extended (both with passive [14] and active security [9,27]), the price is only a few private-key operations per OT (together with a small number of "real" seed OTs that use public-key technology to bootstrap the process). Doing so allows us to:

1. Maintain LEGO's good complexity and achieve statistical security 2^{-k} when the replication factor is only $\rho = O(k/\log(s))$ against a replication factor of $\rho = O(k)$ for standard cut-and-choose such as the one in [22]
2. Implement a variant on LEGO whose security only relies on generic, symmetric primitives (except for the few seed OTs needed to bootstrap the OT extension).

3. Achieve a variant of LEGO that uses "standard" garbled gates (ANDs and free XORs), compatible with garbled gates optimization.

Whether our proposed protocol will be more efficient in practice than protocols with standard cut-and-choose [7, 18, 20, 29, 31] will only be decided by performing a serious comparison of similar implementations running on the same hardware-network configuration of our and other approaches. This is on-going work.

Technical Overview. The main idea of the protocol we present here is the same as in [28]: A prepares many garbled gates (NANDs in [28], while here we use ANDs) together with commitments to the input and output garbled keys. If A prepares a gate dishonestly we view it as a faulty gate, i.e., one that does not give the correct output on some inputs. B asks A to open a random subset of the AND gates and checks them for correctness. If the check goes through, B randomly permutes the unopened gates into buckets representing a redundant AND gate. He solders the gates within a given bucket together and then solders the buckets together to form a circuit that computes the function even in the presence of a minority of faulty gates within each bucket. As part of this soldering NOT gates can be injected thus the garbled AND gates and the soldering alone provides a universal set of Boolean gates.

As the gates have been generated independently, the output keys of the gates in one layer of the circuit cannot be directly fed as input to the next layer. Therefore, A reveals the XOR difference between the output keys in the first layer with the corresponding input keys in the second layer (using the XOR-homomorphic properties of the commitment scheme). This allows B to "align" the input keys of the gates in one layer with the output keys of the gates in the previous layer. He then evaluates all ρ garbled gates in a bucket on the same input key and take the output of the bucket to be any output key agreed upon by more than $\lfloor \rho/2 \rfloor$ of the replicated garbled gates it contains. The main intuition for the security of LEGO cut-and-choose is as follows: If A had sent B k faulty gates, B would detect this with probability $1 - 2^{-k}$. Therefore, if B accepts the test, with very high probability there are only a few faulty gates among the unopened ones. As all gates are permuted at random and placed in random buckets in the circuit, only very little redundancy is needed to correct for all faulty gates.

Because we use XOR homomorphic commitments, our construction can be instantiated with essentially any free-XOR compatible garbled gate scheme and is compatible with various state of the art optimizations (such as free-XOR, row-reduction, point-and-permute).

Organization. We start with preliminaries and background in Section 2. We then continue to go through the overall description of the secure-two party computation protocol in Section 3. This is followed by Section 4 where we describe the main technical contribution of this paper; the XOR homomorphic commitments.

2 Background

In this section we formalize our goal in the UC framework (refer to textbooks such as [8, 10] for definitions). We furthermore list the basic building blocks of our protocol and quickly review their individual constructions.

The Ideal Functionality. In Fig. 1 the ideal functionality for secure function evaluation is presented (taken almost verbatim from [28]). Note that the functionality is insecure in the sense that A can try to guess B's input bits, but if her guess is wrong B is told that A is cheating. This models a standard problem in Yao based protocols known as "selective failure attack", that can be solved by modifying the circuit to be evaluated. For instance, to evaluate a circuit $\mathcal{C}\left((a_i)_{i\in[\ell]},(b_i)_{i\in[\ell]}\right)$ securely one could instead evaluate $\mathcal{C}'\left((a_i)_{i\in[\ell]},(b_{i,j})_{i\in[\ell],j\in[k]}\right) = \mathcal{C}\left((a_i)_{i\in[\ell]},(\oplus_{i\in[k]}b_{i,j})_{j\in[\ell]}\right)$ i.e., B encodes his real input bit in the parity of a k bit-long string, and the modified circuit first reconstructs the real input and then evaluates the original circuit. Now, in order to guess one of B's real input bits A needs to guess correctly the k random bits, so she will fail with probability $1-2^{-k}$. As our construction allows to compute XOR gates for free, this only marginally increases the complexity. Better encodings can be used (See [20]) to reduce the size of the encoded input from $\ell \cdot k$ bits to $\max(4\ell, 8k)$ bits.

Circuit and inputs: On input $(\texttt{init}, A, k)$ from A and input $(\texttt{init}, B, k)$ from B where $A = (\mathbf{a}, \mathcal{C}_A)$, $B = (\mathbf{b}, \mathcal{C}_B)$ proceed as follows:

1. Let k be a statistical security parameter and let $\mathcal{C}_A$ and $\mathcal{C}_B$ be descriptions of Boolean circuits consisting of NOT, XOR and AND gates computing the corresponding Boolean functions f_A, respectively f_B.
2. Leak $\mathcal{C}_A$, $\mathcal{C}_B$ and k to the adversary.
3. If $\mathcal{C}_A \neq \mathcal{C}_B$, then the ideal functionality outputs `disagreement!` to both parties and terminates. Otherwise, let $\mathcal{C} = \mathcal{C}_A$ and parse $\mathcal{C}$ as $(\ell, \mathcal{C}')$, where $\ell \in \mathbb{N}$ and $\mathcal{C}'$ is a circuit with 2ℓ input wires and ℓ output wires. I.e., we potentially add blank wires to make sure that the size of A's input, B's input and the output are the same. Thus $\mathcal{C}'$ computes the Boolean function $f = f_A$.
4. Finally parse $\mathbf{a}$ as $\mathbf{a} \in \{0,1\}^\ell$ and $\mathbf{b} \in \{0,1\}^\ell$ and return $(\ell, \mathcal{C}')$ to both A and B.

Corrupt A: On input `(corrupt)` from A, let her be corrupt. She can then specify a set $\{(i, \beta_i)\}_{i\in I}$, where $I \subseteq \{1,\ldots,k\}$ and $\beta_i \in \{0,1\}$. If $\beta_i = b_i$ for $i \in I$, then output `correct!` to A. Otherwise, output `You were nicked!` to A and output `Alice cheats!` to B.

Evaluation: If both parties are honest or A was not caught above, then on input `(evaluate)` from both A and B the ideal functionality computes $z = f(a,b)$ and outputs z to A. The adversary decides the time of delivery.

Figure 1. The ideal functionality, $\mathcal{F}_{\mathsf{SFE}}$, for secure function evaluation for two parties

Building Blocks. We here review the main building blocks of our protocol.

Generic Free-XOR Yao Gate. Our protocol is compatible with every "free-XOR compatible" garbling schemes. In particular, it is possible to use very optimized garbling schemes. We now describe such a garbling scheme that combines the state of the art optimizations for Yao Gates i.e., free XOR [17], permutation bits [26], garbled row-reduction [26] in the same way as [1].

In particular this means that to garble a gate 4 evaluations of AES are needed, and a garbled gate consists of only 3 ciphertexts (therefore saving on communication complexity). The evaluation of the gate consists of a single AES evaluation.

– We have a (possibly randomized) algorithm $\mathsf{Yao}(L_0, R_0, \Delta, id)$ with id a unique gate identifier, a left input *zero-key* $L_0 \in \{0,1\}^t$, a right input zero-key $R_0 \in \{0,1\}^t$ and a global difference $\Delta \in \{0,1\}^t$ outputs a garbled gate gg and a output zero-key $O_0 \in \{0,1\}^t$.
– We have a (possibly randomized) algorithm $\mathsf{Eval}(gg, L', R')$ that on input a garbled gate gg, a left key $L' \in \{0,1\}^t$ and a right key $R' \in \{0,1\}^t$ outputs an output key $O' \in \{0,1\}^t \cup \{\perp\}$.
– We define the *one-keys* L_1,R_1,O_1 s.t. $L_0 \oplus L_1 = R_0 \oplus R_1 = O_0 \oplus O_1 = \Delta$.

The idea is that a garbled AND gate gg has a zero- and a one-key associated with each of its wires (left input, right input and output wire), and that these keys represent the bit values on those wires. E.g., if gg is a garbled AND gate generated as $(gg, O_0) \leftarrow \mathsf{Yao}(L_0, R_0, \Delta, id)$ then $\mathsf{Eval}(gg, L_a, R_b)$ for any $a, b \in \{0,1\}$ should output $O_{a \wedge b}$.

Note that if A samples Δ and a zero-key, say L_0, at random and give the key L_a to B then there is no way for B to infer the bit a from L_a. Furthermore, even if B learns a he cannot guess the key L_{1-a} with better probability than guessing Δ. For a garbling scheme to be secure we want that even if B learns gg and keys L_a and R_b for $a, b \in \{0,1\}$, and is able to evaluate $O_{a \wedge b} \leftarrow \mathsf{Eval}(gg, L_a, R_b)$, then he cannot guess L_{1-a}, R_{1-b} or $O_{1-a \wedge b}$ with better probability then guessing the random string Δ, even if he knows a and/or b.

Thus B can evaluate the garbled gate gg without knowing anymore about the output than he can infer from his knowledge of a and b. Furthermore, B cannot evaluate the gate on any other inputs. Thus if B sends back $O_{a \wedge b}$ to A, A can learn $a \wedge b$ (as she knows O_0 and Δ) and be confident that this is the correct result. We formalize this intuition about correctness and security of a garbled gate in Def. 1.

Definition 1. *We say that* $(\mathsf{Yao}, \mathsf{Eval})$ *is a* Yao free-XOR garbling scheme *if the following holds:*

Correctness: *Let* $(gg, O_0) \leftarrow \mathsf{Yao}(L_0, R_0, \Delta, id)$. *Then for all* $a, b \in \{0,1\}$, $\mathsf{Eval}(gg, L_a, R_b) = O_{a \wedge b}$, *with overwhelming probability over the choices of* L_0, R_0, Δ *and the random coins of* Yao *and* Eval.

Secrecy: *Consider the following indistinguishability under chosen input attack game for a stateful adversary* $\mathcal{A}$*: The adversary outputs two pairs of bit vectors* $\left(a_0^i, b_0^i\right)_{i \in [k]}, \left(a_1^i, b_1^i\right)_{i \in [k]} \in \{0,1\}^{2k}$. *The game picks a uniformly random challenge* $c \in_{\mathrm{R}} \{0,1\}$, *samples* $\Delta \in_{\mathrm{R}} \{0,1\}^t$ *and for* $i = 1, \ldots, k$ *it samples* $L^i, R^i \in_{\mathrm{R}} \{0,1\}^t$, *samples* $(gg^i, O_0^i) \leftarrow \mathsf{Yao}\left(L_0^i, R_0^i, \Delta, i\right)$ *and then inputs* $\left(gg^i, L^i_{a_c^i}, R^i_{b_c^i}\right)_{i \in [k]}$ *to* $\mathcal{A}$. *Finally* $\mathcal{A}$ *outputs a bit* $d \in \{0,1\}$ *and wins if* $d = c$. *We say that the scheme is IND-CIA if for all PPT* $\mathcal{A}$, $\mathcal{A}$ *wins the* IND-CIA *game with at most negligible advantage in* t.

In the full version [6] we describe an optimized garbling scheme that can be used with our protocol. See also [1].

Soldering. The idea for this component is the same as in [28], however, slightly changed to support a general gate garbling scheme. When a garbled gate gg^1 has the same zero-key (and therefore also one-key) associated to one of its wires, as is associated with one

of gg^2's wires, we say that the given wire of gg^1 is *soldered* to the given wire of gg^2. This is a useful concept when we want to build circuits of garbled gates. To see this consider a garbled gate gg^1 with its left input wire soldered to the output of gg^2, and its right input wire soldered to the output of gg^3. This means that if gg^2 and gg^3 has output zero-keys O_0^2 and O_0^3 respectively, then gg^1 has left and right zero-keys $L_0^1 = O_0^2$ and $R_0^1 = O_0^3$. Thus if we evaluate gg^2 and gg^3 on some input and obtain output keys O_a^2 and O_b^3 we can use this to further evaluate gg^1 on these outputs. The resulting output would be some output key $O_{a\wedge b}^1$.

Alternatively notice that if gg^1 has, e.g., left, input zero-key $L_0^1 = O_0^2 \oplus O_0^3$ then

$$O_a^2 \oplus O_b^3 = O_0^2 \oplus O_0^3 \oplus (a \oplus b)\Delta = L_0^1 \oplus (a \oplus b)\Delta = L_{a\oplus b}^1 \ .$$

In this case we say that the left input wire of gg^1 is soldered to the XOR of the output of gg^2 and gg^3. This is because by XOR'ing the outputs keys of gg^2 and gg^3 we get the left input key of gg^1 corresponding to XOR of the outputs of gg^2 and gg^3. This is also why we call the garbling *free-XOR*: we do not need to garble XOR gates, since this is handled by the soldering.

In our protocol we will first generate garbled gates where all zero-keys are picked independently, and then in a later stage we will *solder* the wires of the garbled gates to each other to form a garbled circuit. For this purpose, we introduce a function $\mathsf{Shift}\left(gg, L^d, R^d, O^d\right)$ that on input a garbled gate $(gg, O_0) \leftarrow \mathsf{Yao}\left(L_0, R_0, \Delta, id\right)$, and three *differences* $L^d, R^d, O^d \in \{0,1\}^t$, outputs a new *shifted gate* sgg. The shifted gate sgg is the gate gg modified to have have input zero-keys $\left(L_0 \oplus L^d\right)$ and $\left(R_0 \oplus R^d\right)$ and output zero-key $\left(O_0 \oplus O^d\right)$.

This can be implemented by letting Shift output the concatenation of its inputs i.e., $sgg = \left(gg, L^d, R^d, O^d\right)$ and let the evaluation of a shifted gate sgg be defined by:

$$\mathsf{ShiftEval}\left(sgg, \hat{L}, \hat{R}, \hat{O}\right) = \mathsf{Eval}\left(gg, \hat{L} \oplus L^d, \hat{R} \oplus R^d\right) \oplus O^d$$

where for all K we define $\perp \oplus K = \perp$. It is clear that a shifted gate is correct (with respect to the shifted zero-keys) if and only if a standard gate is correct, and clearly shifting a gate does not threaten its security property. A shifted gate can be shifted again: The Shift function will just update the values L^d, R^d, O^d accordingly.

Consider two garbled gates $\left(gg^1, O_0^1\right) \leftarrow \mathsf{Yao}\left(L_0^1, R_0^1, \Delta, 1\right)$ and $\left(gg^2, O_0^2\right) \leftarrow \mathsf{Yao}\left(L_0^2, R_0^2, \Delta, 2\right)$. The shifted gate $sgg^2 = \mathsf{Shift}\left(gg^2, \left(O_0^1 \oplus L_0^2\right), 0, 0\right)$ then becomes a garbled gate with left zero-key $L_0^2 \oplus \left(O_0^1 \oplus L_0^2\right) = O_0^1$. I.e. the output wire of gg^1 is now soldered to the left input wire of sgg^2.

Similarly we could have used the Shift function to solder the input of sgg^2 to the XOR of some other garbled gates.

If one wish to use NOT gates then these can be implemented as part of this shifting by a simply change in the the difference, i.e., to add a NOT gate to the soldering to the left wire of a gate we simply use $\neg L^d = L^d \oplus \Delta$ instead of just L^d.

To see this assume we want to put a NOT into the soldering between gg^1 and gg^2. In this case we would have $\neg L^{1^d} = L^{1^d} \oplus \Delta = O_0^1 \oplus L_0^2 \oplus \Delta$, i.e., $sgg^2 = \mathsf{Shift}\left(gg^2, \left(O_0^1 \oplus L_0^2 \oplus \Delta\right), 0, 0\right)$. Thus when the evaluator does

$$\mathsf{ShiftEval}\left(sgg^2, L^2, 0, 0\right) = \mathsf{Eval}\left(gg^2, L_a^2 \oplus (O_0^1 \oplus L_0^2 \oplus \Delta), R^2\right) \ .$$

If $a = 0$ we get the left input key for gg^2 is $L_0^2 \oplus (O_0^1 \oplus L_0^2 \oplus \Delta) = O_0^1 \oplus \Delta = O_1^1$ and similarly for $a = 1$ we get $L_1^2 \oplus (O_0^1 \oplus L_0^2 \oplus \Delta) = (L_0^2 \oplus \Delta) \oplus O_0^1 \oplus (L_0^2 \oplus \Delta) = O_0^1$. Thus we clearly see that the bit represented by the left input key (along with the key itself) for gg^2 has been flipped.

Initialization

On input $(\texttt{init}, ID, W)$ from the adversary, with $|ID| = \mu$, $|W| \leq \kappa$ and $W \subset ID$, output ID to both A and B and let $\mathcal{J} = \emptyset$. If A is honest, then $W = \emptyset$.

Commit

On input $(\texttt{commit}, j \in ID, m_j)$ with $m_j \in \{0,1\}^h$ from A, and where no value of the form $(j, \cdot)$ is stored, store (j, m_j). If $j \in ID \setminus W$, add $J = \{j\}$ to $\mathcal{J}$ and associate with J the equation $\mathtt{X}_j = m_j$. Then output $(\texttt{commit}, j)$ to B.

Open

On input $(\texttt{open}, J \subset ID)$ from A, where for all $j \in J$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{open}, J, \oplus_{j \in J} m_j)$ to B.
- If A is corrupted wait for A to input $(\texttt{corrupt-open}, J, m_J)$. Then add J to $\mathcal{J}$, associate the equation $\oplus_{j \in J} \mathtt{X}_j = m_J$ to J, and check that the equation system $\{\oplus_{j \in J} \mathtt{X}_j = m_J\}_{J \in \mathcal{J}}$ has a solution. If so, output $(\texttt{open}, J, m_J)$ to B. Otherwise, output `Alice cheats` to B and terminate.

Oblivious Opening

On input $(\texttt{OT-choose}, otid, b)$ with $b \in \{0,1\}$ from B output $(\texttt{OT-choose}, otid)$ to A. On input $(\texttt{OT-open}, otid, J_0, J_1)$ from A with $J_0, J_1 \subset ID$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored and $(\texttt{OT-choose}, otid, *)$ was input before by B do the following:

- If A is honest, output $(\texttt{OT-open}, otid, J_b, \oplus_{j \in J_b} m_j)$ to B (Note that B does not learn the set of ids J_{1-b}).
- If A is corrupted, wait for A to input $(\texttt{guess}, g)$ with $g \in \{0, 1, \perp\}$. If $g \in \{0,1\}$ and $g \neq b$ output `Alice cheats` to B and terminate. Otherwise, proceed to wait for A to input $(\texttt{corrupt-open}, J_0, J_1, m_{J_0}, m_{J_1})$. Add J_b to $\mathcal{J}$ and associate the equation $\oplus_{j \in J_b} \mathtt{X}_j = m_{J_b}$ to J_b. Check that the equation system still has a solution as described above. If so, output $(\texttt{OT-open}, J_b, m_{J_b})$ to B. Otherwise output `Alice cheats` to B.

OR Open

For up to ω Or-Openings, that must all occur before the first Oblivious-Opening, do the following: On input $(\texttt{OR-open}, J_0, J_1, a)$ from A, with $J_0, J_1 \subset ID, a \in \{0,1\}$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{OR-open}, J_0, J_1, \oplus_{j \in J_a} m_j)$ to B.
- If A is corrupted, and if $J_a \cap W \neq \emptyset$, wait for corrupt A to input $(\texttt{corrupt-open}, J_a, m_{J_a})$, add J_a to $\mathcal{J}$ and associate $\oplus_{j \in J_a} \mathtt{X}_j = m_{J_a}$ to J_a. Check if the equation system still has a solution as described above. If so, output $(\texttt{OR-open}, J_0, J_1, m_{J_a})$ to B. Otherwise output `Alice cheats` to B.

before the first Oblivious-Opening.

Figure 2. The ideal functionality, $\mathcal{F}_{\mathsf{COM}}$, for the commitment scheme used by π_{LEGO}.

Homomorphic commitments. To securely implement the soldering described above, we cannot simply have (potentially malicious) A send the differences needed to shift

the gates. Instead we will have A give homomorphic commitments to all zero-keys of each gate, and then have her open the differences of the committed keys. Therefore we need a homomorphic commitment scheme. In Fig. 2 we state the ideal functionality $\mathcal{F}_{\mathsf{COM}}$ for the homomorphic commitments that we implement in Section 4. As we are now going to use this functionality to implement $\mathcal{F}_{\mathsf{SFE}}$ we will briefly recap the features of this functionality.

The functionality allows A to commit to messages and to later reveal those messages. In addition the functionality allows to reveal the XOR of two or more committed messages to B (without revealing any extra information).

The functionality is "insecure", in the sense that A can choose a set of up to κ wildcard commitments where she can change her mind about the committed value at opening time. However, openings need to be consistent. More specifically, the $\mathcal{F}_{\mathsf{COM}}$ functionality stores a system of linear equations. Initially these equations simply specify that non-wildcard commitments must be opened to the value, they were commitments to. Every time A performs an opening involving wildcard commitments this defines a new linear equation, which is stored in the ideal functionality. For an opening of a wildcard commitment to be successful the set of linear equations stored in the ideal functionality must be consistent.

If the set of equations stored in the ideal functionality restricts the opening of a commitment in such a way that it can only be opened to *one* value, we say that the commitment is *fixed* to that value. Note, that all non-wildcard commitments are fixed, and a fixed wildcard commitment can essentially be viewed as a non-wildcard commitment.

As we are treating the commitments as an ideal functionality, we need to push into the ideal functionalities two extra commands (in a way similar to the commit-and-prove functionality in [3])): apart from the regular openings the functionality allows to open (the XOR of) committed messages in two alternative ways: In an *Oblivious-Opening*, B can choose between two sets of committed messages and learn the XOR of the messages in one of them. In an *Or-Opening* we allow A to open the XOR of one out of two sets of committed messages without revealing which one. For technical reasons there can only be a total of ω Or-Openings and all Or-Openings must be done before the first Oblivious-Opening. Also, note that there is a build-in selective failure attack in the Oblivious-Opening. However, this is not a problem as we will only use this type of opening to handle B's input where, as discussed above, the $\mathcal{F}_{\mathsf{SFE}}$ functionality already allows a selective failure attack.

Commitment from B to A. Additional to the $\mathcal{F}_{\mathsf{COM}}$ functionality we are going to use an extractable commitment Com. This commitment is used only once by B to commit to his challenge in the cut-and-choose phase and extraction is needed for simulation (to avoid selective opening issues). Since this commitment does not need to be homomorphic it can be easily implemented in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

3 The MiniLEGO Protocol

We now show how to use the ingredients outlined in the previous section in order to construct the MiniLEGO protocol.

We denote by $\mathcal{C}'$ the Boolean circuit to be evaluated. We assume $\mathcal{C}'$ to be composed of NOT, XOR and AND gates. The XOR gates are allowed to have unbounded fan-in while the AND gates have fan-in 2. With each AND gate in $\mathcal{C}'$ we associate a unique label and we let gates be the set of all these labels. A subset $\mathsf{inputGates} \subset \mathsf{gates}$ of size 2ℓ are specially marked as input gates. The AND gates in $\mathsf{inputGates}$ should be given the same bit on both input wires, so that the gate simply computes the identity function. A subset in $\mathsf{Ainputs} \subset \mathsf{inputGates}$ of size ℓ are taken to be A's inputs. The remaining ℓ gates in $\mathsf{Binputs} = \mathsf{inputGates} \setminus \mathsf{Ainputs}$ are B's inputs (for convenience assume that $\mathsf{Binputs} = [\ell]$). A has input bits $(a_1, \ldots, a_\ell)$, while B has input bits $(b_1, \ldots, b_\ell)$.

A subset $\mathsf{outputGates} \subset \mathsf{gates}$ of size ℓ are marked as output gates. The output of these gates are taken to be the output of the circuit. Note that this means that all output gates are AND gates. This is without loss of generality: a circuit with XOR gates as output gates can be modified to an equivalent circuit with AND gates as output gates by adding at most ℓ AND gates. The ℓ output bits are denoted $(z_1, \ldots, z_\ell)$.

The wiring of the circuit $\mathcal{C}'$ is described by two functions $\mathsf{lp}, \mathsf{rp} : \mathsf{gates} \setminus \mathsf{inputGates} \to 2^{\mathsf{gates} \cup \{\mathbb{1}\}}$. We call $\mathsf{lp}(j)$ the left parents of j (resp. $\mathsf{rp}(j)$ the right parents of j), and take the left (resp. right) input of j to be the XOR of the output bits of all gates in $\mathsf{lp}(j)$ (resp. $\mathsf{rp}(j)$). Thus, the XOR gates of $\mathcal{C}'$ are implicitly defined by the lp and rp functions. If the special symbol $\mathbb{1}$ is included in the set returned by lp, rp, this means that a NOT gate is inserted between gate j and its parent gate (i.e., the input is XORed with the constant 1). We assume that $\max(\mathsf{lp}(j) \cup \mathsf{rp}(j)) < j$ for all j.

Garbled Circuit. Let $\Gamma = 2\rho s$ for $s = |\mathsf{gates}|$ and some replication factor $\rho \in \mathbb{N}$. For our protocol A will construct Γ garbled gates. She constructs twice as many garbled gates as is needed to build the garbled circuit, because half the gates are going to be checked during the cut-and-choose phase. We choose to check exactly half for the sake of presentation but, as in [31], this could be changed to any fraction in order to optimize concrete efficiency.

Bucket Notation. In the protocol individual garbled gates are combined together in "buckets" of gates. We introduce here some convenient notation that allow us to address the gates in a bucket, the bucket of a gate etc. Let $\mathcal{B}$ be the family of ρ-to-1, ρ-wise independent functions from a set $U \subset [\Gamma]$ of size ρs to gates. For a function $\mathsf{BucketOf} \in \mathcal{B}$ let Bucket be the function that, for all $j \in \mathsf{gates}$ outputs the set $\{i \in U | \mathsf{BucketOf}(i) = j\}$. Let $\mathsf{BucketHead}(j)$ be the function that returns the "first" (in lexicographic order) element of $\mathsf{Bucket}(j)$.

There are $\Gamma' = 3\Gamma + 1$ keys in the protocol, because every constructed AND gate has a left, right and output key and in addition there is a global difference Δ. The key index is written as a superscript while subscripts are in $\{0, 1\}$ and describe the value carried by the key i.e., $K_b^i = K^i \oplus (b\Delta)$. Let id be a function that on input a key $K_0^j \in \{0,1\}^t$ returns a unique label for that key. We will sometimes abuse notation and write $\mathsf{id}\left(K_1^j\right)$ to denote the set $\left\{\mathsf{id}\left(K_0^j\right), \mathsf{id}\left(\Delta\right)\right\}$. This will simplify the notation when using the $\mathcal{F}_{\mathsf{COM}}$ functionality.

Protocol Specification. The protocol π_{LEGO} in Fig.s 3 and 4 progresses in six phases: **Setup**, **Garbling**, **Cut-and-choose**, **Soldering**, **Input** and **Evaluation**. Here we describe these phases one by one.

During **Setup**, A or B initialize the $\mathcal{F}_{\mathsf{COM}}$ functionality by calling $(\texttt{init}, ID, W)$ with $ID = \Gamma'$ and $|W| \leq k$. For the remainder of the protocol if $\mathcal{F}_{\mathsf{COM}}$ outputs `Alice cheats` , B will abort the protocol. Then A samples the global difference Δ and commits to it using the `commit` command in $\mathcal{F}_{\mathsf{COM}}$. B samples his challenge for the cut-and-choose phase and the BucketOf function as described above, and commits to both using the extractable commitment Com. B also "commits" to his input using the `OT-choose` command of the $\mathcal{F}_{\mathsf{COM}}$ functionality. These commitments of B's are needed to avoid selective opening issues in the cut-and-choose phase and reduce the security of the protocol to the IND-CIA game.

In **Garbling**, A constructs the candidate garbled gates $(gg^i)_{i\in[\Gamma]}$ and commits to the input/output zero-keys of each garbled gate using $\mathcal{F}_{\mathsf{COM}}$.

In **Cut-and-choose**, B reveals his challenge. The challenge consists of a set of indices $T \subset [\Gamma]$ of size ρs and a sequence of bits $(u_i, v_i)_{i\in T}$, indicating that B wants to test garbled gate gg^i on input (u_i, v_i). A opens the corresponding input and output keys for the test gates, allowing B to check for correctness. Note that B only tests one set of inputs for each gate – otherwise he will learn Δ.

In the remainder of the protocol the garbled gates that are not checked in **Cut-and-choose**, those with indices in $U = [\Gamma] \setminus T$, are used to build a garbled circuit according to the following fault tolerant circuit design: With each gate $j \in \mathsf{gates}$ we associate a *bucket* of ρ AND gates. To evaluate gate j we will evaluate each gate in the bucket of j on the inputs given to j. If more than $\lfloor \rho/2 \rfloor$ of the gates in the bucket agree on their output bit, we take this bit to be the output of j (otherwise the output is $\perp$). Clearly if there are more than $\lfloor \rho/2 \rfloor$ non-faulty gates in each bucket the output is correct.

To build such a garbled circuit the gates that were not checked $(gg^i)_{i\in U}$ are put into buckets using the BucketOf function. Then B uses the Shift function as described in Section 2 to solder the wires of the garbled gates. Since A may be malicious we cannot simply have her sent the XOR's of zero-keys that B needs for soldering. Instead A reveals the XOR's by opening the corresponding commitments to the zero-keys.

The garbled circuit is constructed in **Soldering** in three different soldering steps: For all $j \in \mathsf{gates}$ **Horizontal Soldering** solders all wires of all gates in $(gg^i)_{i\in\mathsf{Bucket}(j)}$ to the corresponding wires of $gg^{\mathsf{BucketHead}(j)}$. This allows to evaluate all the gates in the same bucket on the same input keys and get the same output keys. I.e., if A is honest, after the horizontal soldering all the gates in one bucket have exactly the same keys. For all $j \in \mathsf{gates}$ **Vertical Soldering** solders the left input wire of $gg^{\mathsf{BucketHead}(j)}$ to the XOR of the output wires of $\left(gg^{\mathsf{BucketHead}(i)}\right)_{i\in\mathsf{lp}(j)}$, and the right input wire of $gg^{\mathsf{BucketHead}(j)}$ to the XOR of the output wires of $\left(gg^{\mathsf{BucketHead}(i)}\right)_{i\in\mathsf{rp}(j)}$ (and we use the convention $O^{\mathbb{1}} = \Delta$ to deal with NOT gates – note that Δ can be seen as the 1 key of a special wire with zero-key equal to 0^t). Note that since **Horizontal Soldering** made all garbled gates in a bucket have the same input keys, this essentially means soldering *all* the gates in the bucket to the output wires of gates in $(\mathsf{Bucket}(i))_{i\in\mathsf{lp}(j)}$ and $(\mathsf{Bucket}(i))_{i\in\mathsf{rp}(j)}$. I.e., vertical soldering is "functional", in the sense that it ensures that the garbled circuit computes the right circuit, $\mathcal{C}'$. For all $j \in \mathsf{inputGates}$ **Input Soldering** simply solders the left and right input wire of garbled gates in $\mathsf{Bucket}(j)$ to each other, i.e., the gates in inputGates compute the identity function.

Setup Choose $\rho = O(k/\log(s))$ and $\Gamma = 2\rho s$ where $s = |\mathsf{gates}|$ in $\mathcal{C}'$. Let $\Gamma' = 3\Gamma + 1$ and proceed as follows:

1. A and B initialize a $\mathcal{F}_{\mathsf{COM}}$ functionality by having either of them call $(\texttt{init}, ID, W)$ with $|ID| = \Gamma'$ and $|W| \leq k$.
2. A samples $\Delta \in_R \{0,1\}^t$ and inputs $(\texttt{commit}, \mathsf{id}(\Delta), \Delta)$ to $\mathcal{F}_{\mathsf{COM}}$.
3. B samples a random $T \subset [\Gamma]$ of size ρs, and for all $i \in T$ samples $u_i, v_i \in_R \{0,1\}$. Let $U = [\Gamma] \setminus T$.
4. B samples $\mathsf{BucketOf} \in \mathcal{B}$ as described in Section 2.
5. B sends $C_T = \mathsf{Com}(T, (u_i, v_i)_{i \in T}, \mathsf{BucketOf}, r_T)$ to A.
6. For each $j \in \mathsf{Binputs}$, B inputs $(\texttt{OT-choose}, j, b_j)$ to $\mathcal{F}_{\mathsf{COM}}$.

Garbling

1. For all $i \in [\Gamma]$, A samples $L_0^i, R_0^i \in_R \{0,1\}^t$, computes $\left(gg^i, O_0^i\right) \leftarrow \mathsf{Yao}\left(L_0^i, R_0^i, \Delta, i\right)$ and sends $GG = \left(gg^i\right)_{i \in [\Gamma]}$ to B.
2. A inputs $\left(\texttt{commit}, \mathsf{id}\left(L_0^i\right), L_0^i\right)$, $\left(\texttt{commit}, \mathsf{id}\left(R_0^i\right), R_0^i\right)$ and $\left(\texttt{commit}, \mathsf{id}\left(O_0^i\right), O_0^i\right)$ to $\mathcal{F}_{\mathsf{COM}}$.

Cut-and-choose

1. B sends T, $(u_i, v_i)_{i \in T}$, $\mathsf{BucketOf}$ and randomness r_T to A.
2. If this is not a valid opening of C_T A aborts. Otherwise, for all $i \in T$ A inputs to $\mathcal{F}_{\mathsf{COM}}$ $\left(\texttt{open}, \mathsf{id}\left(L_{u_i}^i\right)\right)$, $\left(\texttt{open}, \mathsf{id}\left(R_{v_i}^i\right)\right)$, $\left(\texttt{open}, \mathsf{id}\left(O_{u_i \wedge v_i}^i\right)\right)$. Let $\hat{L}^i, \hat{R}^i, \hat{O}^i$ be the values output to B by $\mathcal{F}_{\mathsf{COM}}$.
3. B aborts if there is an $i \in T$ so that $\hat{O}^i \neq \mathsf{Eval}\left(gg^i, \hat{L}^i, \hat{R}^i\right)$.

Soldering

1. **Horizontal Soldering:** For all $j \in \mathsf{gates}$, let $h = \mathsf{BucketHead}(j)$: For all $i \neq h \in \mathsf{Bucket}(j)$ A inputs $\left(\texttt{open}, \left\{\mathsf{id}\left(L^h\right), \mathsf{id}\left(L^i\right)\right\}\right)$, $\left(\texttt{open}, \left\{\mathsf{id}\left(R^h\right), \mathsf{id}\left(R^i\right)\right\}\right)$, and $\left(\texttt{open}, \left\{\mathsf{id}\left(O^h\right), \mathsf{id}\left(O^i\right)\right\}\right)$ to $\mathcal{F}_{\mathsf{COM}}$. Let $L^{i\,d}, R^{i\,d}, O^{i\,d}$ be the keys output to B from $\mathcal{F}_{\mathsf{COM}}$ and $sgg^i = \mathsf{Shift}\left(gg^i, L^{i\,d}, R^{i\,d}, O^{i\,d}\right)$.
2. **Vertical Soldering:** For all $j \in \mathsf{gates} \setminus \mathsf{inputGates}$, let $h = \mathsf{BucketHead}(j)$: A inputs $\left(\texttt{open}, \left\{\mathsf{id}\left(L^h\right)\right\} \cup \left\{\mathsf{id}\left(O^{\mathsf{BucketHead}(i)}\right)\right\}_{i \in \mathsf{lp}(j)}\right)$ and $\left(\texttt{open}, \left\{\mathsf{id}\left(R^h\right)\right\} \cup \left\{\mathsf{id}\left(O^{\mathsf{BucketHead}(i)}\right)\right\}_{i \in \mathsf{rp}(j)}\right)$ to $\mathcal{F}_{\mathsf{COM}}$. Let $L^{h\,d}, R^{h\,d}$ be the keys output to B by $\mathcal{F}_{\mathsf{COM}}$ and $sgg^h = \mathsf{Shift}\left(gg^h, L^{h\,d}, R^{h\,d}, 0^t\right)$.
3. **Input Soldering:** For all $j \in \mathsf{inputGates}$, let $h = \mathsf{BucketHead}(j)$: A inputs $\left(\texttt{open}, \left\{\mathsf{id}\left(L^h\right), \mathsf{id}\left(R^h\right)\right\}\right)$ to $\mathcal{F}_{\mathsf{COM}}$. Let $R^{h\,d}$ be the key output to B by $\mathcal{F}_{\mathsf{COM}}$ and $sgg^h = \mathsf{Shift}\left(sgg^h, 0^t, R^{h\,d}, 0^t\right)$.

Figure 3. The Protocol π_{LEGO} implementing $\mathcal{F}_{\mathsf{SFE}}$ (Part 1).

In **Input**, for all $j \in \mathsf{Ainputs}$ A uses the Or-Opening of $\mathcal{F}_{\mathsf{COM}}$ to open the input key to the garbled gates in $\mathsf{Bucket}(j)$ corresponding to her input bit. For all $j \in \mathsf{Binputs}$ B also learns the input key to the garbled gates in $\mathsf{Bucket}(j)$ corresponding to his input bit, using the Oblivious-Opening.

Given the initial input keys in **Evaluation** B evaluates each bucket of garbled gates in the following way: He evaluates each gate in the bucket on the left and right input keys for that bucket. If a key appears more than $\lfloor \rho/2 \rfloor$ times as the output key of the

garbled gates in the bucket, he takes this to be the output key of the bucket. If no such key exists B aborts. Note that by the way we soldered the garbled circuit, this corresponds exactly to the fault tolerant circuit we described above. Finally B provides A with the output keys. Knowing Δ, A can decipher the output keys and obtain the output values.

Input

1. For all $j \in \mathsf{Ainputs}$ let $h = \mathsf{BucketHead}(j)$, A inputs $\left(\texttt{OR-open}, \mathsf{id}\left(L_0^h\right), \mathsf{id}\left(L_1^h\right), a_j\right)$ to $\mathcal{F}_{\mathsf{COM}}$.
2. For all $j \in \mathsf{Binputs}$ let $h = \mathsf{BucketHead}(j)$, A inputs $\left(\texttt{OT-open}, j, \mathsf{id}\left(L_0^h\right), \mathsf{id}\left(L_1^h\right)\right)$ to $\mathcal{F}_{\mathsf{COM}}$.
3. In both cases let $\hat{L}^h$ be the key output from $\mathcal{F}_{\mathsf{COM}}$ to B. B computes a list of candidate keys $\mathsf{Cand}_j = \left(\mathsf{ShiftEval}\left(sgg^i, \hat{L}^h, \hat{L}^h\right)\right)_{i \in \mathsf{Bucket}(j)}$. If any key appears more than $\rho/2$ times in Cand_j name it $\hat{O}^j$, otherwise B aborts.

Evaluate

1. For each gate $j \in \mathsf{gates} \setminus \mathsf{inputGates}$, B computes:
   (a) Left and right keys $\hat{L}^j = \bigoplus_{l \in \mathsf{lp}(j)} \hat{O}^l$ and $\hat{R}^j = \bigoplus_{l \in \mathsf{rp}(j)} \hat{O}^l$.
   (b) The list of output keys $\mathsf{Cand}_j = \left(\mathsf{ShiftEval}\left(sgg^i, \hat{L}^j, \hat{R}^j\right)\right)_{i \in \mathsf{Bucket}(j)}$.
   (c) If a key appears more than $\rho/2$ times in Cand_j name it $\hat{O}^j$ and proceed, otherwise abort.
2. For all $j \in \mathsf{outputGates}$, B sends $\hat{O}^j$ to A.
3. For all $j \in \mathsf{outputGates}$, A outputs $z_j = 0$ if $\hat{O}^j = O_0^{\mathsf{BucketHead}(j)}$, $z_j = 1$ if $\hat{O}^j = O_1^{\mathsf{BucketHead}(j)}$ and aborts otherwise.

Figure 4. The Protocol π_{LEGO} implementing $\mathcal{F}_{\mathsf{SFE}}$ (Part 2).

Theorem 1. *Let k be the security parameter, $\rho = O(k/\log(s))$. If $(\mathsf{Yao}, \mathsf{Eval})$ is an* IND-CIA *secure* Yao free-XOR garbling scheme *then the protocol π_{LEGO} in Fig.s 3 and 4 UC, active, static securely implements $\mathcal{F}_{\mathsf{SFE}}$ in the $(\mathcal{F}_{\mathsf{COM}})$-hybrid model (initialized with $(\mathtt{init}, ID, W)$ for $|ID| = 3\Gamma + 1$ and $|W| \leq k$).*

Analysis. We sketch the idea of the proof. Details are in the full version. First considering a corrupted B and then considering a corrupted A.

Corrupted B. B does not receive any output nor has any real way of cheating in the protocol (in the output phase, if B changes the output key in a way that makes A accept, then he must have guessed Δ, thus breaking the IND-CIA game). Essentially, we only need to argue that his view does not leak any information, thanks to the IND-CIA security of the garbling scheme. Note that in the protocol B starts by committing to his input and challenge for the cut-and-choose phase. This allows the simulator S to extract all this information at the beginning of the simulation (and provide input on behalf of corrupted B to the ideal functionality). Then we reduce the security of the protocol to the IND-CIA security of the garbling scheme: the simulator knows in fact T and U before it sends the gates to B, therefore S will place honestly constructed gates in T (for

which it knows the openings and therefore can easily simulate the cut-and-choose test – remember that the simulator fully controls $\mathcal{F}_{\mathsf{COM}}$) and the challenge garbled gates from the IND-CIA game in U: that is, the simulator produces a view such that distinguishing between a real and a simulated execution is equivalent to winning the IND-CIA game.

Corrupted A. Essentially, the proof of security boils down to proving correctness. By the design of the garbled circuit, correctness follows when there is more than $\lfloor \rho/2 \rfloor$ correct gates in each of the buckets.

The LEGO approach ensures that if A passes the cut-and-choose test, then with overwhelming probability there are at most k faulty gates left in U. Those faulty gates are then randomly assigned into buckets, and this means that with overwhelming probability each bucket will have a majority of correct gates. However, as opposed to [28] where all commitments were binding, here we have also k wildcard commitments to deal with. This is in problematic, as wildcard commitments can be opened to anything, and we need make sure that this does not break correctness.

To be more specific we say that a garbled gate gg^i is *faulty* if the commitments to its input and output zero-keys are fixed to values L_0^i, R_0^i and O_0^i respectively, and there exists some $a, b \in \{0,1\}$ so that $\mathsf{Eval}\left(gg^i, L_a^i, R_b^i\right)$ does not output $O_{a \wedge b}^i$ with overwhelming probability. If a gate gg^i has a wire where the commitment to the associated zero-key is not fixed, then we say that this wire is faulty, and gg^i has *faulty wiring*. We say that gg^i is *fault free* if it is neither faulty nor has faulty wiring. If a garbled gate gg^i is faulty, fault free or has faulty wiring, we say the same of any shifted gate sgg^i resulting from shifting gg^i.

Gates gg^i with faulty wiring are problematic for the cut-and-choose test: If $i \in T$ A can choose to let gg^i act as a fault free gate by opening the wildcard commitments consistently with the actual zero-keys used to generate gg^i. On the other hand, if $i \in U$ A can make sgg^i faulty by opening the commitment inconsistently in **Soldering**[1].

In the full version we show that, with overwhelming probability, there will be a majority of fault free gates in $\left(gg^i\right)_{i \in \mathsf{Bucket}(j)}$ for all $j \in \mathsf{gates}$. It is easy to verify that this means that after **Horizontal Soldering** all commitments to zero-keys are fixed. I.e., the commitment to the zero-key of a faulty wire will be fixed to open as one specific value. If this value is not consistent with the zero-keys used to generate the associated garbled gate, then that gate becomes faulty.

Note however, that for all $j \in \mathsf{gates}$ all fault free shifted gates $\left(sgg^i\right)_{i \in \mathsf{Bucket}(j)}$ resulting from **Horizontal Soldering** will have identical input and output keys, as required of the garbled circuit, even if some gates in $\left(gg^i\right)_{i \in \mathsf{Bucket}(j)}$ had faulty wiring. I.e., the effect of a garbled gate gg^i having faulty wiring is *at worst* that shifted gate sgg^i after **Soldering** is faulty. Since we use a $\mathcal{F}_{\mathsf{COM}}$ functionality with at most k wildcard commitments we still have at most k faulty gates in **Evaluation**. Since these faulty gates are placed in random buckets we can still guarantee correctness with overwhelming probability.

Note that the faulty wires are also the reason for gate replication on the input layer, to not let A change her or B's input by using the wildcard commitments.

[1] By *inconsistently* we mean inconsistent with the actual keys used for gg^i, not inconsistent with the equations stored in $\mathcal{F}_{\mathsf{COM}}$.

4 Commitments

In this section we present our novel construction of XOR-homomorphic commitment based solely on OT. We also describe how to transform this general construction of XOR-homomorphic commitments into the specific type we need in Fig.s 3 and 4.

The Ideal Functionality. We name our ideal functionality $\mathcal{F}_{\mathsf{WCOM}}$ and describe it in Fig. 5. The functionality allows A to commit to up to μ messages and to later reveal those messages. In addition the $\mathcal{F}_{\mathsf{WCOM}}$ allows to reveal the XOR of two or more committed messages to B (without revealing any extra information). In the context of Fig.s 3 and 4. this is the same as $\mathcal{F}_{\mathsf{COM}}$ except without the methods for **Oblivious Opening**, **OR open** and (which will become apparent later on) contains more wildcards than we can handle in Fig.s 3 and 4.

We formalize $\mathcal{F}_{\mathsf{WCOM}}$ in the following way: First we let the adversary specify a set of identifiers ID of size κ used to identify each of the μ commitments (for technical reasons ID will be a subset of $[2\mu]$). In addition the adversary gives a set $W \subset ID$, of size at most κ, to identify the wildcard commitments. $\mathcal{F}_{\mathsf{WCOM}}$ stores a set of linear equations on μ variables $(\mathtt{X}_i)_{i \in ID}$ (one for each commitment). Initially this set is empty. Each time A commits to a message m_j using a non-wildcard commitment (i.e., $j \in ID \setminus W$) $\mathcal{F}_{\mathsf{WCOM}}$ will store the equation $\mathtt{X}_j = m_j$. For wildcard commitments no such equation are stored when A makes the commitment. If A instructs $\mathcal{F}_{\mathsf{WCOM}}$ to open the XOR of the set of commitments $J \subset ID$ we let corrupted A input a message $m_J \in \{0,1\}^t$ she wants to open to. The functionality then adds the equation $\bigoplus_{j \in J} \mathtt{X}_j = m_J$ to the set of stored equations and checks that this set of equations has a solution, i.e., if there is an assignment of values in $\{0,1\}^t$ to each variable $\mathtt{X}_j$ such that all equations are satisfied. If so, the functionality permanently stores the equation $\bigoplus_{j \in J} \mathtt{X}_j = m_J$ and opens the XOR of the set of commitments J as m_J. Otherwise, $\mathcal{F}_{\mathsf{WCOM}}$ will output `Alice cheats` to B and terminate. Note that if $J \cap W = \emptyset$ then for all $j \in J$ the functionality has stored the equation $\mathtt{X}_j = m_j$, and therefore a corrupt A can only open the commitment successfully if $m_J = \bigoplus_{j \in J} m_j$. Note also that if, e.g., A has made commitments $i \in W$ and $j \in ID \setminus W$, and opens the XOR of commitments i and j as m' then for all later openings A can only successfully open the wildcard commitment i as $m'_i = m_j \oplus m'$. In these cases, when a commitment can only successfully be opened to one value, we say that the commitment is *fixed* to that value. Non-wildcard commitments are always fixed; when A opens the XOR of a wildcard commitments and non-wildcard commitments, a wildcard commitment can become fixed. When a wildcard commitment has been fixed it can essentially be viewed as a non-wildcard commitment.

Notice that the terminology can become a little confusing because of the wildcard commitments: when we say that A opens the XOR of some set of commitments $J \subset ID$ to a value m_J, then we cannot guarantee that $m_J = \bigoplus_{j \in J} m_j$, when $J \cap W \neq \emptyset$.

Building blocks. Here we give the building blocks from which we implement $\mathcal{F}_{\mathsf{WCOM}}$.
Oblivious Transfer. We use a $\binom{n}{u}$-Oblivious Transfer functionality with message strings of length 2μ. We denote this functionality $\binom{n}{u}$-$\mathcal{F}_{\mathrm{OT}}(2\mu)$. On input `start` from both A and B the $\binom{n}{u}$-$\mathcal{F}_{\mathrm{OT}}(2\mu)$ functionality picks n message strings $S_1, \ldots, S_n \in_{\mathrm{R}} \{0,1\}^\mu$,

Initialization: On input $(\texttt{init}, ID, W)$ with $|ID| = \mu$, $|W| \leq \kappa$ and $W \subset ID$ from the adversary output ID to both parties and let $\mathcal{J} = \emptyset$. If A is honest, then $W = \emptyset$.

Commit On input $(\texttt{commit}, j \in ID, m_j)$ with $m_j \in \{0,1\}^t$ from A, and where no value of the form $(j, \cdot)$ is stored, store (j, m_j). If $j \in ID \setminus W$, add $J = \{j\}$ to $\mathcal{J}$ and associate with J the equation $\mathtt{X}_j = m_j$. Then output $(\texttt{commit}, j)$ to B.

Open On input $(\texttt{open}, J \subset ID)$ from A, where for all $j \in J$ a pair (j, m_j) is stored do:

- If A is honest, output $(\texttt{open}, J, \oplus_{j \in J} m_j)$ to B.
- If A is corrupted wait for A to input $(\texttt{corrupt-open}, J, m_J)$. Then add J to $\mathcal{J}$, associate the equation $\oplus_{j \in J} \mathtt{X}_j = m_J$ to J, and check that the equation system $\{\oplus_{j \in J} \mathtt{X}_j = m_J\}_{J \in \mathcal{J}}$ has a solution[a]. If so, output $(\texttt{open}, J, m_J)$ to B. Otherwise, output `Alice cheats` to B and terminate.

[a] I.e., there should be an assignment of values to the wildcard commitments such that all stored openings can be explained by this assignment.

Figure 5. The ideal functionality, $\mathcal{F}_{\mathsf{WCOM}}$, for our basic commitment scheme consisting.

a uniformly random set $I \subseteq \{1, \ldots, n\}$ with $|I| = u$ and outputs $(S_i)_{i \in [n]}$ to A and $(I, (S_i)_{i \in I})$ to B. We can implement $\binom{n}{u}$-$\mathcal{F}_{\text{OT}}(2\mu)$ for any $2\mu = \text{poly}(k)$ using a $\mathcal{F}_{\text{OT}}$ functionality and a pseudo random generator (prg), where k is the security parameter, using the same technique as in [27]: Simply use the $\mathcal{F}_{\text{OT}}(k)$ functionality to send seeds to the prg and then use the prg to expand those seeds to 2μ bits. One can then construct a $\binom{n}{u}$-$\mathcal{F}_{\text{OT}}(2\mu)$ functionality from $\binom{2}{1}\mathcal{F}_{\text{OT}}(2\mu)$ e.g., as described in [25].

Error Correcting Codes. We also need an error correcting code (ECC), which encodes an t-bit string as an n-bit string with minimal distance at least d using some ϕ-bits of randomness. It should at the same time be a secret sharing scheme in that seeing u random positions of a random codeword does not leak information on the message. We denote this scheme by $\text{ssecc}^{t,n,d,u}$. We use $\text{enc}^{t,n,d,u}$ to denote the encoding function and we use $\text{dec}^{t,n,d,u}$ to denote the decoding function. Both should be PPT and we drop parameters for notational convenience. The code should have the following properties.

Error correction For all $m \in \{0,1\}^t$, $r \in \{0,1\}^\phi$ and error vectors $e \in \{0,1\}^n$ with $\text{hw}(e) < d/2$ it holds that $\text{dec}(\text{enc}(m; r) \oplus e) = m$, where hw is the Hamming weight in $\{0,1\}^n$. We assume that $\text{dec}(C) = \perp$ when C has distance more than $d/2$ to all codewords and we assume that there exists an efficient algorithm ncw (nearest codeword) such that $\text{ncw}(\text{enc}(m; r) \oplus e) = \text{enc}(m; r)$ when $\text{hw}(e) < d/2$.

Privacy There exists a PPT function xpl which can explain any codeword as being a codeword of any message to anyone who knows at most u positions of the codeword. Formally, for all $I \subset [n]$, $|I| = u$ and all $m, m' \in \{0,1\}^t$ the distributions D_0 and D_1 described below are statistically close. The distribution D_0 is generated as follows: sample $r \in_{\text{R}} \{0,1\}^\phi$, let $c = \text{enc}(m; r)$ and output $((c_i)_{i \in I}, m, r)$. The distribution D_1 is generated as follows: sample $r' \in_{\text{R}} \{0,1\}^\phi$, let $c = \text{enc}(m'; r')$, sample $r \leftarrow \text{xpl}(I, m', r', m)$ and output $((c_i)_{i \in I}, m, r)$.

Linearity For all $m, m' \in \{0,1\}^t$ and $r, r' \in \{0,1\}^\phi$ it holds that $\text{enc}(m; r) \oplus \text{enc}(m'; r') = \text{enc}(m \oplus m'; r \oplus r')$.

Note that **Error correction** implies that the minimal distance is at least d, i.e., for all $m \neq m' \in \{0,1\}^t$ and $r, r' \in \{0,1\}^\phi$, $c = \mathrm{enc}(m; r)$ and $c' = \mathrm{enc}(m'; r')$ it holds that $\mathrm{ham}(c, c') \geq d$ where ham is the Hamming distance.

We further require of the parameters of ssecc that: $n = \Theta(k)$, $u = \Theta(n)$ and $d = \Theta(n)$. I.e., both the privacy and minimum distance of ssecc must be a constant fraction of the length of codewords, and the code should have constant rate. Codes that satisfy the desired properties can be found in [4].

Protocol Specification. Here we describe the ideas behind the protocol π_{WCOM} implementing $\mathcal{F}_{\mathsf{WCOM}}$ described in Fig. 6.

Let $v \in \{0,1\}^n$ and $I = \{i_1, i_2, \ldots, i_u\} \subseteq [n]$. We define the function $w_I : \{0,1\}^n \to \{0,1\}^u$ so that $w_I(v) = (v_{i_1}, v_{i_2}, \ldots, v_{i_u}) \in \{0,1\}^u$, i.e., $w_I(v)$ is the u-bit string consisting of the u bits in v indexed by I.

In the protocol a commitment to a message m is a one-time pad of m with some key T. Clearly this is hiding but not binding. To make the commitment binding we allow the receiver of the commitment (B) to learn $w_I(m)$ for some secret set $I \subseteq [n]$. We denote $w_I(m)$ the *watch bits* of the commitment. To open the commitment to m A sends m' to B and B checks if $w_I(m') = w_I(m)$. If this is not the case B rejects the opening.

The watch bits give some degree of binding since A can only open the commitment to some message $m' \neq m$ if $w_I(m') = w_I(m)$. I.e., if $u = |I|$ is large enough A can only hope to change a few bits of m without getting caught. On the other hand the watch bits clearly compromises the hiding property of the commitment. To avoid this we use the code ssecc to encode the message m and commit to the encoded m instead. I.e., a commitment to m becomes $\mathrm{enc}(m; r) \oplus T$. By privacy of ssecc m is now hidden.

The encoding additionally strengthens the binding of the commitment: codewords c and c' encoding to two different messages m and m' must be different in many bit positions. Thus for A to open a commitment to m to m' none of these positions must be in the watch bits.

More precisely let $d = 2w + 1$ be the minimum distance of ssecc for some $w < \frac{n}{2}$. Suppose a corrupt A gives the commitment, $c \oplus T$. Note that when A is corrupt c does not have to be a codeword. In that case we have that $c = \mathrm{ncw}(c) \oplus e$ for some *error vector* $e \in \{0,1\}^n$, and we say the commitment has $\mathrm{hw}(e)$ *errors*.

Regardless of the number of errors, consider what it takes for A to be able to open this commitment to two different messages m' and m'', with codewords c' and c'' respectively: for any two different codewords c' and c'' one of them has distance at least w to c, say c'. In other words c' has at least w bit positions different from c. If A tries to open the commitment to m', B only accepts the opening if none of these bit positions are in his u watch bits for the commitment. Thus for any commitment (possibly with errors) the probability that a cheating A can open the commitment to two different messages m' and m'' is at most

$$\binom{n-u}{w}\binom{n}{w}^{-1} = \prod_{i=0}^{w-1} \frac{n-u-i}{s-i} \prod_{i=0}^{w-1} \frac{w-i}{n-i} = \prod_{i=0}^{w-1} \frac{n-u-i}{n-i} = \prod_{i=0}^{w-1} 1 - \frac{u}{n-i}\,.$$

Assume that $w = w'n$ and $u = u'n$ for constants $0 < u', w' < 1$, then the probability of A breaking the binding property is at most

$$\prod_{i=0}^{w-1}\left(1-\frac{u}{n-i}\right) \leq \prod_{i=0}^{w-1}\left(1-\frac{u'n}{n}\right) = (1-u')^{w'n} \ .$$

Thus with k being the security parameter, $n = \Theta(k)$ and for any positive constants u', w' with $0 < u', w' < 1$, A will have negligible probability of breaking binding.

Notice, that while c' will have distance at least w to c it could be that c'' is much closer to c. E.g., c'' could be the nearest codeword to c. In this case, we have that, if the commitment has *very few* errors, a cheating A *could* open the commitment to m'' with noticeable probability (say, if the number of errors were constant in k). This is not a problem since such a "slightly wrong" commitment can be seen as a commitment to the message m'' encoded by the nearest codeword c''.

To get XOR-homomorphic commitments, more work has to be done. The problem being that the XOR of several commitments with errors, may become a commitment that breaks binding, even if the individual commitments only have a few errors. Consider a number of commitments made with non-codewords c_i with nearest codewords c_i'. The XOR the non-codewords $c = \bigoplus_i c_i$ may then be very far from the XOR of their nearest codewords $c' = \bigoplus_i c_i'$. In fact c might be so far away from c' that it gets very close to some other codeword c''. Hence the XOR of the commitments can be opened to a message different from the XOR of the message associated with the individual commitments. This would break the binding property.

To deal with this problem, the protocol initialization π_{WCOM} starts by letting A commit to 2μ random messages. We then do a cut-and-choose test to check that half of these commitments can be opened correctly. If A passes the test we have that, with overwhelming probability, the remaining commitments only have a few errors. Additionally, those errors must be isolated to a few common bit positions. Thus the result of XOR'ing these commitments will at most have a few errors, namely in these few positions.

Thus if A passes the cut-and-choose test we use the un-tested random commitments to implement the actual commitments. The resulting commitment will have exactly the same errors as the random commitment (if any).

Theorem 2. *Let k be the security parameter and use a code with $n = \Theta(k)$, $u = \Theta(n)$, $d = \Theta(n)$ and $k < d/2$ as, e.g., given by [4]. Then the protocol in Fig. 6 UC, active, static securely implements $\mathcal{F}_{\mathsf{WCOM}}$ in the $\binom{n}{u}$-$\mathcal{F}_{\mathrm{OT}}(2\mu)$-hybrid model when initialized on $(\mathtt{init}, ID, W)$ with $|ID| = \mu$ and $|W| \leq nk + k$.*

Analysis. Simulating when no party is corrupted or both parties are corrupted is straight forward. Simulating when B is corrupted is also quite simple, and can be done using standard techniques from simulation in secure multi-party computation based on secret sharing. Thus we will only sketch the proof for corrupted B, and focus on the case of corrupted A.

Corrupted B. The simulator commits to 0^t in all commitments. When asked to open such a commitment U_j to a given $m_j \in \{0,1\}^t$ it uses the efficient algorithm xpl

Setup To set up the scheme A and B run the following.

1. A and B run a $\binom{n}{u}$-$\mathcal{F}_{\text{OT}}(2\mu)$ functionality and get as output $(R_i)_{i\in[n]}$ and $(I,(R_i)_{i\in I})$ respectively where $R_1,\ldots,R_n \in_{\text{R}} \{0,1\}^{2\mu}$ and I is a uniformly random subset of $[n]$ with $|I|=u$.
2. A lets $R \in \{0,1\}^{n\times 2\mu}$ be the matrix with R_i as the i'th row and lets $T_j \in \{0,1\}^n$ be the j'th column of R[a].
3. A for $j=1,\ldots,2\mu$, samples $x_j \in_{\text{R}} \{0,1\}^t$, $r_j \in_{\text{R}} \{0,1\}^\phi$ and sends the commitment $c_j = (\text{enc}(x_j;r_j)\oplus T_j, j)$. Let $c_j=(U_j,j)$ the value received by B.
4. B sends a uniformly random subset $C \subset [2\mu]$. This also defines $ID=\bar{C}$.
5. For $j\in C$, A opens c_j by sending $o_j=(x_j,r_j,j)$.
6. For $j\in C$, B receives (x'_j,r'_j,j) and checks that $w_I\left(\text{enc}\left(x'_j;r'_j\right)\right) = w_I(U_j)\oplus w_I(T_j)$, if not B terminates the protocol.

Commit To commit to m_j for $j\in ID$ A sends (y_j,j) to B where y_j is the *correction value* $y_j = x_j \oplus m_j$.

Open To open the XOR of commitments $J\subset ID$ the parties do the following.

1. For $j\in J$, let $c_j=(\text{enc}(x_j;r_j)\oplus T_j,j)$ be the commitments sent in initialization and y_j the value sent during commitment. A computes the opening of $\bigoplus_{j\in J} m_j$ as $o_J = \left(\bigoplus_{j\in J} x_j, \bigoplus_{j\in J} r_j, J\right)$, and sends it to B.
2. If an opening of J was done previously, B uses the previous m_J, otherwise he proceeds as follows: Let $c_j=(U_j,j)$ be the commitments received during **Setup**. B accepts $o_J=(x_J,r_J,J)$ iff

$$w_I(\text{enc}(x_J;r_J)) = w_I\left(\bigoplus_{j\in J} U_j\right) \oplus w_I\left(\bigoplus_{j\in J} T_j\right) \text{, where}$$

$$w_I\left(\bigoplus_{j\in J} U_j\right) = \bigoplus_{j\in J} w_I(U_j) \text{ and } w_I\left(\bigoplus_{j\in J} T_j\right) = \bigoplus_{j\in J} w_I(T_j)\ .$$

If B accepts he outputs $x_J \oplus y_J$, where $y_J = \bigoplus_{j\in J} y_j$. Otherwise, B rejects the opening and terminates the protocol.

[a] Notice B can use $(R_i)_{i\in I}$ to compute $w_I(T_j)$ for all $j\in[2\mu]$.

Figure 6. The protocol π_{WCOM} implementing $\mathcal{F}_{\mathsf{WCOM}}$.

to explain the commitment as $U_j = \text{enc}(x_j;r_j)\oplus T_j$ for $x_j = y_j \oplus m_j$. The only non-trivial detail is that if the simulator is asked to open a commitment, where the value of the opening follows from previous openings (i.e., using some linear equation), it computes the opening as a linear combination of the previous simulated openings. As an example, if the simulator opened U_j as $U_j=\text{enc}(x_j;r_j)\oplus T_j$ and opened U_i as $U_i=\text{enc}(y_i;r_i)\oplus T_i$. Then it will open $U_j\oplus U_i$ as $U_j=\text{enc}(x_j\oplus x_i; r_j\oplus r_i)\oplus T_j\oplus T_i$.

Corrupted A. Intuition of the proof when A is corrupted is that the cut-and-choose test will catch A if there are many indices i for which there exists a commitment that has an error in position i. This is because if the errors of the commitments are very spread out, with high probability, many of them will be in the watch bits positions. As mentioned above, this means that almost all errors must be isolated in a few positions. Therefore XOR's of commitments will also have errors only in these position, so the XOR's will

also be close to their "correct" codeword. The formal proof is complicated by the fact that a few commitments with many errors, or errors outside isolated few positions, may pass the cut-and-choose. These commitments will be the wildcards. It can be shown that not even a commitment with many errors can be opened to two different values, as it would give a codeword encoding a non-zero value which is 0 in all the watch bits, which happens with negligible probability by the watch bits being random and the minimal distance high. This translates into it being impossible to make any combination of openings of linear equations yielding inconsistent outputs.

Completing the Construction. There is a gap between the ideal functionality $\mathcal{F}_{\mathsf{WCOM}}$ that we just implemented and the functionality $\mathcal{F}_{\mathsf{COM}}$ used in the protocol π_{LEGO}. The gap can be closed using standard techniques, as sketched now. There are many more details on this in the full version. We can reduce the number of wildcard commitments by opening random pairs of commitments and discarding one of the commitments. This fixes any wildcard commitment not lucky enough to be paired with another wildcard commitment. We can implement **Oblivious Opening** by sending both openings through an oblivious transfer: note that we allow selective errors in the ideal functionality, so it is not an issue that the adversary can send one correct and one incorrect opening. Finally, we can implement **OR Open** using a standard technique where the committer commits to many pairs of values, each pair being a random permutation of the values in the two commitments of which he wants to open one. Then for each pair he is randomly challenged by the receiver to either uses the XOR homomorphism to show that the correct two messages were committed, or to open one of the two commitments to the claimed value.

References

1. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012.
2. A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.
3. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
4. H. Chen and R. Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.
5. I. Damgård, M. Keller, E. Larraia, C. Miles, and N. P. Smart. Implementing AES via an actively/covertly secure dishonest-majority mpc protocol. In *SCN*, pages 241–263, 2012.
6. T. Frederiksen, T. P. Jakobsen, P. S. N. Jesper Buus Nielsen, and C. Orlandi. Minilego: Efficient secure two-party computation from general assumptions (full version). Cryptology ePrint Archive, Report, 2013. `http://eprint.iacr.org/`.
7. T. K. Frederiksen and J. B. Nielsen. Fast and maliciously secure two-party computation using the gpu. Cryptology ePrint Archive, Report 2013/046, 2013. `http://eprint.iacr.org/`.
8. O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

9. D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. OT-combiners via secure computation. In *TCC*, pages 393–411, 2008.
10. C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, 2010.
11. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462, 2010.
12. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.
13. Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy*, pages 272–284, 2012.
14. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.
15. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.
16. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.
17. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, pages 486–498, 2008.
18. B. Kreuter, shelat abhi, and C. Shen. Billion-gate secure computation with malicious adversaries. USENIX Security. Available at Cryptology ePrint Archive, Report 2012/179, 2012. `http://eprint.iacr.org/`.
19. Y. Lindell, E. Oxman, and B. Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *CRYPTO*, pages 259–276, 2011.
20. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.
21. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
22. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.
23. Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN*, pages 2–20, 2008.
24. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
25. M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 573–590. Springer, 1999.
26. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.
27. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.
28. J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *TCC*, pages 368–386, 2009.
29. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.
30. B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *ASIACRYPT*, pages 119–136, 2004.
31. shelat abhi and C.-H. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.
32. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.