LEGO for Two-Party Secure Computation

Jesper Buus Nielsen and Claudio Orlandi

BRICS, Department of Computer Science, Aarhus University
{jbn,orlandi}@cs.au.dk

Abstract. This paper continues the recent line of work of making Yao's garbled circuit approach to two-party computation secure against an active adversary. We propose a new cut-and-choose based approach called LEGO (Large Efficient Garbled-circuit Optimization): It is specifically aimed at large circuits. Asymptotically it obtains a factor $\log |\mathcal{C}|$ improvement in computation and communication over previous cut-and-choose based solutions, where $|\mathcal{C}|$ is the size of the circuit being computed. The protocol is universally composable (UC) in the OT-hybrid model against a static, active adversary.

1 Introduction

In secure two-party computation we have two parties, Alice and Bob, that want to compute a function $f(\cdot,\cdot)$ of their inputs a, b, and learn the result $y = f(a, b)$, without any party learning any other information.

Yao [Yao82, Yao86] was the first to present a solution to this problem. His protocol, presented and proved in [LP04], is only secure against a passive adversary. We give a novel approach to making Yao's idea secure against active adversaries.

In Yao's protocol Alice constructs a garbled circuit and sends it to Bob: a malicious Alice can send Bob a circuit that does not compute the agreed function, as a consequence the computation loses both privacy and correctness. In our protocol instead Alice and Bob both participate in the circuit construction. The main idea of our protocol is to have Alice prepare and send a bunch of garbled NAND gates (together with some other components) to Bob. Bob selects a random subset of the gates and Alice provides Bob with the keys to test them. If they all work correctly he assumes that at most a small fraction of the remaining gates are malfunctioning. Bob shuffles the remaining gates to put the faulty gates in random positions and connects them into a circuit that computes the desired function even in the presence of a few random faults — the scrambled NAND gates are designed such that Bob can, with a limited help from Alice, connect the gates as he likes. Then the circuit is evaluated by Bob as in Yao's protocol: Bob gets his keys running oblivious transfers (OT) with Alice and he evaluates the circuit.

Related Work: In the last years many solutions have been proposed to achieve two-party computation secure against malicious adversaries.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-00457-5_36

O. Reingold (Ed.): TCC 2009, LNCS 5444, pp. 368–386, 2009.

In [LP07, LPS08], Alice sends s copies of the Yao's garbled circuit to be computed. Bob checks half of them and computes on the remaining circuits. A similar approach was suggested in [MF06]. Due to the circuit replication, they need to introduce some machinery in order to force the parties to provide the same inputs to every circuit, resulting in an overhead of s^2 commitments per input wire for a total of $O(s\kappa|\mathcal{C}| + s^2\kappa|\mathcal{I}|)$, where $|\mathcal{C}|$ is the size of the circuit, $|\mathcal{I}|$ is the size of the input and κ is the length of a hash value. To optimize the cut-and-choose construction, Woodruff [Woo07] proposed a way of proving input consistency using expander graphs: using this construction it is possible to get rid of the dependency on the input size and therefore achieving complexity of $O(s\kappa|\mathcal{C}|)$. More concretely, the protocol in [LP07, LPS08] requires s copies of a circuit of size $|\mathcal{C}| + |D|$, where D is an input decoder, added to the circuit to deal with so-called selective failures. They propose a basic version of D with size $O(s|\mathcal{I}|)$ and a more advanced with size $O(s + |\mathcal{I}|)$. However, because of the $s^2|\mathcal{I}|$ commitments, their optimized encoding gives them just a benefit in the number of OT required. With our construction, we can fully exploit their encoding. In fact we need just to replicate $s/\log(|\mathcal{C}|)$ times a circuit of size $O(|\mathcal{C}|+ s + |\mathcal{I}|) = O(|\mathcal{C}|)$, which gives our protocol a complexity of $O((s/\log(|\mathcal{C}|))\kappa|\mathcal{C}|)$, i.e., our replication factor is reduced by the logarithm of the circuit size. The improvement in replication factor from s to $s/\log(|\mathcal{C}|)$ comes from doing cut-and-choose on individual gates instead of doing it on entire circuits.

Another approach to making Yao's idea actively secure is to use generic zero-knowledge proofs to force good behavior [GMW86]. In theory this can be done with just a constant overhead in communication [NN01].

Other related works include: Considering UC security, in [JS07] a solution for two party computation on committed inputs is presented. This construction uses public-key encryption together with efficient zero-knowledge proofs, in order to prove that the circuit was built correctly. Their asymptotic complexity is $O(\kappa'|\mathcal{C}|)$, where κ' is the length of factorization-based public-key cryptosystems. In our protocol the parameter κ can be chosen to be much smaller i.e., the required size for hashing and elliptic curves cryptography.

In [IPS08] a protocol for UC secure two-party computation in the OT-hybrid model is presented with communication complexity $O(|\mathcal{C}|) + \text{poly}(s, d, \log|\mathcal{C}|)$, where s is the security parameter and d is the depth of $\mathcal{C}$. The hidden constant factor and the term $\text{poly}(s, d, \log|\mathcal{C}|)$, however, makes the comparison between the protocols unclear. Moreover, their protocol has non-constant round complexity as opposed to ours. Even the situation for very small or alternatively very large circuits is not clear *a priori* without trying to optimize and implement both approaches.

Our Contribution: Our scheme is based on three assumptions. We need a UC secure OT scheme. In addition we assume a finite group and an element g of prime order p such that the discrete logarithm problem is hard in $\langle g\rangle$, for instance the group of points over an elliptic curve. Finally we need a function $H : \mathbb{Z}_p \to \mathbb{Z}_p$ which is collision resistant and which is correlation resistant according to Def. 1.

Definition 1. *Given $F : \mathbb{Z}_p \to \mathbb{Z}_p$, let $\mathcal{O}^F$ be the following oracle: It samples a uniformly random $\Delta \in \mathbb{Z}_p$ and stores Δ. Whenever it is queried on $c \in \{0,1,2\}$, it samples a uniformly random $K_0 \in \mathbb{Z}_p$, lets $K_1 = K_0 + \Delta \bmod p$, $K_2 = K_0 + 2\Delta \bmod p$ and returns K_c and $F(K_d)$ for $d \in \{0,1,2\} \setminus \{c\}$. We call H* correlation resistant *if no poly-time adversary can distinguish $\mathcal{O}^H$ from $\mathcal{O}^R$, where R is a uniformly random function from $\mathbb{Z}_p$ to $\mathbb{Z}_p$.*

It is clear that a random function is correlation resistant and it seems reasonable to assume that practical hash functions satisfy this property. The notion of correlation resistance is closely related to the notion of correlation robustness in [IKNP03].

Theorem 1. *If H is collision resistant and correlation resistant according to Def. 1 with output length at most κ, the DL problem is hard in $\langle g \rangle$, elements of $\langle g \rangle$ can be represented with κ bits and 2^{-s} is negligible, then our protocol securely evaluates any function $y = f(a,b)$ computed by a poly-sized Boolean circuit $\mathcal{C}$. The protocol is UC secure against a static, active adversary in the OT-hybrid model. The round complexity is constant, the communication complexity is $O(\kappa s|\mathcal{C}|/\log|\mathcal{C}|)$. Regarding computational complexity, $O(s|\mathcal{C}|/\log|\mathcal{C}|)$ exponentiations in $\langle g \rangle$ are performed and the number of OT calls is $O(|\mathcal{I}| + s)$.*

The version of our protocol presented in this short version of the paper is less efficient than need be, to allow clearer presentation and analysis. In the full version [NO08] some (constant factor) efficiency improvements are presented.

2 Ideal Functionalities

The ideal functionality we implement is described in Fig. 1 (see Fig. 2 for notation). It is "insecure" in the sense that it allows Alice to guess Bob's input bits. This can be solved in a black-box manner by replacing $\mathcal{C}$ with a circuit computing a function of an encoded version of Bob's input. The randomized

The ideal functionality $\mathcal{F}_{\text{SCE}}$ is parametrized by a circuit $\mathcal{C}$ and runs as follows:

Inputs: Alice inputs $\{x_w\}_{w \in \mathcal{I}_A}$ to specify an input bit x_w for each of her wires w, and Bob inputs $\{x_w\}_{w \in \mathcal{I}_B}$.

Abort: If a party is corrupted and inputs `abort!`, then $\mathcal{F}_{\text{SCE}}$ outputs `abort!` to the other party and terminates.

Evaluation: If no party inputs `abort!`, then $\mathcal{F}_{\text{SCE}}$ computes $\{y_w\}_{w \in \mathcal{O}} = \mathcal{C}(\{x_w\}_{w \in \mathcal{I}})$ and outputs $\{y_w\}_{w \in \mathcal{O}}$ to Alice.

Guess: If Alice is corrupted, she can after Bob inputs $\{x_w\}_{w \in \mathcal{I}_B}$ give an input $W \subseteq \mathcal{I}_B$ and $\{\beta_w\}_{w \in W}$. If $\beta_w = x_w$ for all $w \in W$, then $\mathcal{F}_{\text{SCE}}$ outputs `correct!` to Alice and continues as above. Otherwise, it outputs `You were busted!` to Alice, outputs `Alice cheats!` to Bob and then terminates.

Fig. 1. Our ideal functionality for secure circuit evaluation

encoding is such that any s bits of a codeword are uniformly random and independent. The security follows from the fact that Alice cannot guess s or more bits with probability more than 2^{-s}, and if she guesses less bits she will learn no information. One method for this is given in [LP07]. The extra number of gates used is $O(|\mathcal{I}_B| + s)$, where $|\mathcal{I}_B|$ is the length of Bob's input and s is a security parameter.

We implement $\mathcal{F}_{\text{SCE}}$ in the OT-hybrid model with an ideal functionality $\mathcal{F}_{\text{OT}}$. We also assume an ideal functionality $\mathcal{F}_{\text{ZK}}$ for zero-knowledge proof of knowledge. This can be implemented by s calls to the OT functionality: The prover offers a reply to challenge $e = 0$ and $e = 1$ and the verifier chooses and verifies one of the replies. The simulator reads the verifier's challenge and sets the corresponding message to be a simulated reply. The extractor can get both replies and compute the witness if both are correct. Repeating s times gives a soundness error of 2^{-s}. We use in total just 2 calls to the proof of knowledge functionality, giving an overhead of $2s$ calls to $\mathcal{F}_{\text{OT}}$.

3 LEGO Circuits

We start by describing our variation of Yao circuits. It is designed to allow Alice to generate garbled gates independently and later let Bob connect the gates in any order. In the description we use the notation in Fig. 2.

We work with garbled circuits where each wire can carry a value $c \in \{0, 1, 2\}$. Input wires to the circuit and output values from the circuit only carry values $c \in \{0, 1\}$ — the value $c = 2$ is only carried by certain internal wires. For a wire with name w we use $\mathcal{V}(w) \in \{0, 1, 2\}$ to denote the *value* carried by the wire. The values on input wires are specified by Alice and Bob.

LEGO circuits consist of wires and so-called bricks. Wires are essentially just names $w \in \{0, 1\}^*$ to which we will associate certain values, in particular a zero-key K_0 and a commitment $[K_0]$. We write $\mathcal{Z}(w) = K_0$ to denote the zero-key associated to wire w and we write $\mathcal{COM}(w) = [K_0]$ to denote the associated commitment to K_0. Alice knows K_0 and $[K_0]$ while Bob knows only $[K_0]$ from the beginning. During the evaluation of the garbled circuit Bob will learn a key $K_c \in \{K_0, K_1, K_2\}$ for each wire w — here $K_c = K_0 + c\Delta \bmod p$. We think of this as the wire w carrying the value $c \in \{0, 1, 2\}$. Bob does not know the value of K_0 and therefore he does not know the value of c.

Bricks are special garbled circuits allowing Bob to compute on keys K_c. As an example we sketch the not-two brick. It has one input wire w_I and one output wire w_O — these are just unique names from $\{0, 1\}^*$. Let $I_0 = \mathcal{Z}(w_I)$ and $O_0 = \mathcal{Z}(w_O)$ be the associated zero-keys. If Bob knows $I_x \in \{I_0, I_1, I_2\}$ then the not-two brick will allow Bob to compute $O_z \in \{O_0, O_1\}$ with $z = \text{nt}(x)$. The not-two brick does not leak either x or z to Bob. We also have an addition brick with two input wires (with zero-keys L_0 and R_0) and one output wire (with zero-key S_0): from $L_x \in \{L_0, L_1\}$ and $R_y \in \{R_0, R_1\}$ it allows Bob to compute $S_{x+y} \in \{S_0, S_1, S_2\}$. Finally we have a key-filter brick with one associated zero-key K_0. Given a possibly large set of keys $\mathcal{K} \subset \mathbb{Z}_p$ it allows to compute $\mathcal{K} \cap \{K_0, K_1\}$. It

- s is the security parameter.
- $\mathcal{C}$ is a Boolean circuit specifying the function to be computed. It consists of NAND gates only.
- $|\mathcal{C}|$ is the number of NAND gates in $\mathcal{C}$.
- $\mathcal{I}, \mathcal{O}$ are the names of input wires respectively output wires in $\mathcal{C}$. $\mathcal{I}_A \subset \mathcal{I}$ are Alice's input wires and $\mathcal{I}_B = \mathcal{I} \setminus \mathcal{I}_A$ are Bob's input wires.
- $\{y_w\}_{w\in\mathcal{O}} = \mathcal{C}\{x_w\}_{w\in\mathcal{I}}$ says that if input wires $w \in \mathcal{I}$ are assigned the bits x_w and $\mathcal{C}$ is evaluated on them, then the output wires $w \in \mathcal{O}$ will hold the bits y_w.
- p is a large prime.
- $[x]$ is a Pedersen commitment to $x \in \mathbb{Z}_p$. It is computed as $[x] = g^x h^r$ for uniformly random $r \in \mathbb{Z}_p$. Here g is a group generator of order p, and $h \in_R \langle g \rangle$ is chosen by Bob. The prime p and the group is picked such that no poly-time algorithm can solve the DL problem in $\langle g \rangle$ with probability better than 2^{-s}.
- If $[x] = g^x h^r$ and $[y] = g^y h^s$ then $[x] \boxplus [y] = [x][y] = g^{x+y} h^{r+s}$ is a commitment to $x + y \bmod p$, and $[x] \boxminus [y] = [x][y]^{-1} = g^{x-y} h^{r-s}$ is a commitment to $x - y \bmod p$.
- $\Delta \in \mathbb{Z}_p$ is the *global difference*, chosen uniformly at random by Alice and unknown by Bob; $[\Delta]$ is a uniformly random commitment to it. Our use of Δ is inspired by [KS08].
- $K_0 \in \mathbb{Z}_p$ denotes a so-called *zero-key*. After Δ is fixed it defines the *one-key* $K_1 = K_0 + \Delta \bmod p$ and the *two-key* $K_2 = K_0 + 2\Delta \bmod p$. The key K_c will be Bob's representation of the "plaintext" $c \in \{0, 1, 2\}$.
- $[K_0]$ is a commitment to a zero-key, always produced by Alice. From $[K_0]$ and $[\Delta]$ Bob can compute commitments $[K_1] = [K_0] \boxplus [\Delta]$ and $[K_2] = [K_1] \boxplus [\Delta]$.
- $\Sigma \in \mathbb{Z}_p$ denotes a so-called *shifting value*: It is a difference $\Sigma = K'_0 - K_0 \bmod p$ between two zero-keys K_0 and K'_0. Note that $K_c + \Sigma = K'_c$ for $c = 0, 1, 2$.
- $H : \mathbb{Z}_p \to \mathbb{Z}_p$ is a hash function which is collision resistant and correlation resistant according to Def. 1. It is picked such that no poly-time adversary can distinguish with probability better than 2^{-s} in Def. 1.
- For $K, K' \in \mathbb{Z}_p$ we let $E_K(K') = H(K) + K' \bmod p$ and think of it as a deterministic encryption of K' using K. We let $D_K(C) = C - H(K) \bmod p = K'$.
- We define $\mathrm{nt} : \{0, 1, 2\} \to \{0, 1\}$ by $\mathrm{nt}(0) = \mathrm{nt}(1) = 1$ and $\mathrm{nt}(2) = 0$.
- We define $\bar{\wedge} : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$ by $a \bar{\wedge} b = 0$ iff $a = 1$ and $b = 1$. Note that if $a, b \in \{0, 1\}$, then $\mathrm{nt}(a + b) = a \bar{\wedge} b$.
- $\mathcal{K} : \{0, 1\}^* \to 2^{\mathbb{Z}_p}$ maps wire names $w \in \{0, 1\}^*$ to subsets of keys.

Fig. 2. Notation, explained further in main text

is among other things used to ensure that Alice sends valid keys for her input wires.

In the evaluation of the circuit Alice will send $K_a \in \{K_0, K_1\}$ to represent her input $a \in \{0, 1\}$ for each of her input wires w with zero-key K_0. Bob uses a key-filter brick to ensure that $K_a \in \{K_0, K_1\}$. For each of Bob's input wires Alice offers K_0 and K_1 in an OT and Bob inputs b to get the key K_b representing input $b \in \{0, 1\}$. Then the circuit $\mathcal{C}$ is evaluated by Bob on these keys. Each NAND gate in $\mathcal{C}$ is implemented by an addition brick followed by a not-two brick. When Bob knows $K_c = K_0 + c\Delta \bmod p$ for an output wire he sends K_c to Alice who computes $c \in \{0, 1\}$ using K_0 and Δ.

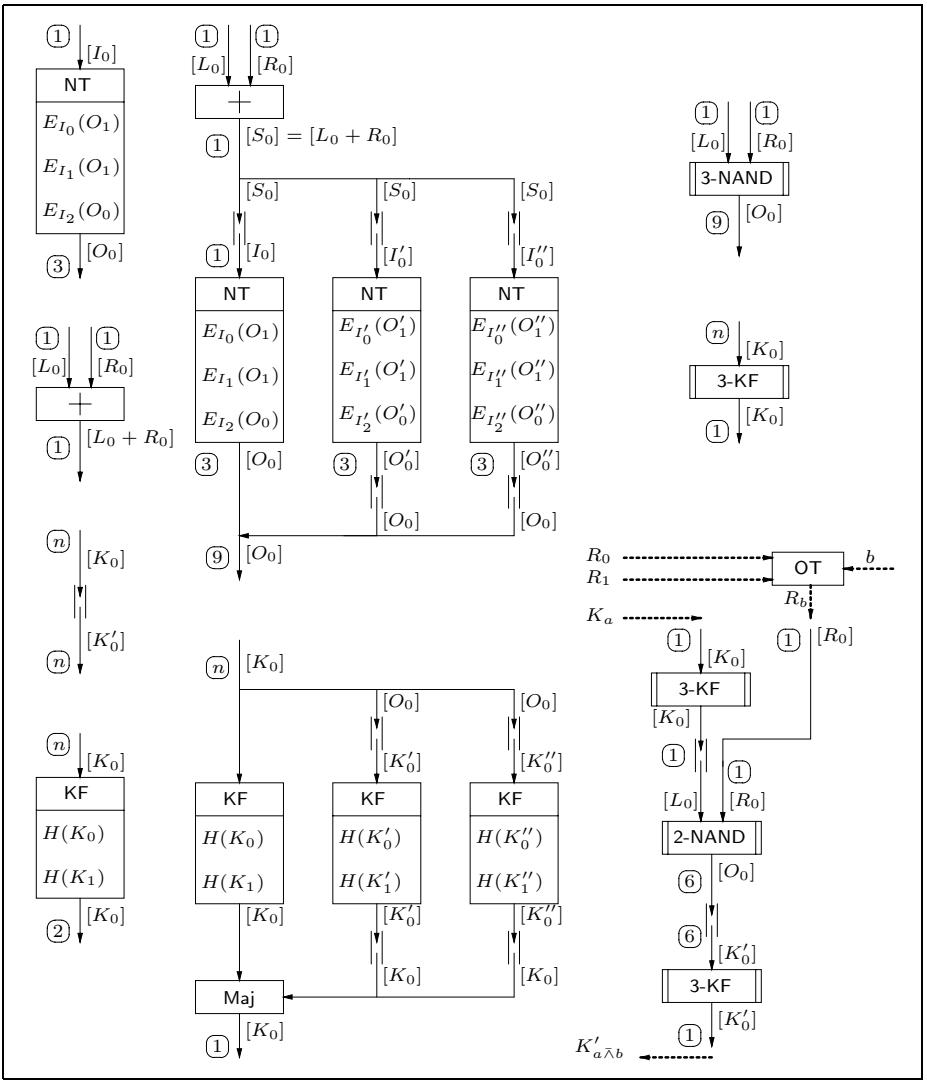

Fig. 3. Graphical notation, explained in main text

The above did not use the associated commitments $[K_0]$. These are used to allow Bob to connect the bricks as he desires, even though they were generated independently. A wire w with zero-key K_0 is connected to a wire w' with zero-key K'_0 by Alice opening $[K'_0] \boxminus [K_0]$ to Bob to let him learn $\Sigma = K'_0 - K_0 \bmod p$. Given K_c for w, Bob can then compute $K'_c = K_c + \Sigma \bmod p$ for w'. The commitments are use to prevent Alice from sending a wrong value of Σ.

Alice can, however, cheat when she generates the bricks. To deal with this she produces more bricks than needed and a random subset of them is selected by Bob for testing. In the test Bob selects a random input for the brick and Alice sends the appropriate keys by opening the associated commitments. If the

selected bricks pass the test Bob will be ensured that most of the remaining bricks will also run correctly. To deal with a small number of incorrect bricks Bob will replicate each of the bricks in the circuit design.

We now give the details of how bricks are generated, evaluated, connected and replicated.

Potential Keys: Above we said that Bob learns $K_c \in \{K_0, K_1, K_2\}$ for each wire. In fact, at some points Bob might hold more than one potential key for a wire. To ease notation we introduce a function $\mathcal{K} : \{0,1\}^* \to 2^{\mathbb{Z}_p}$ mapping a wire name w to the set of potential keys held by Bob.

NT Bricks: An *NT brick* (Fig. 3 top, left) with input wire w_I and output wire w_O contains two commitments $[I_0]$ and $[O_0]$, where $\mathcal{COM}(w_I) \stackrel{\text{def}}{=} [I_0]$ and $\mathcal{COM}(w_O) \stackrel{\text{def}}{=} [O_0]$. Besides it contains three encryptions $C_0 = E_{I_0}(O_1), C_1 = E_{I_1}(O_1), C_2 = E_{I_2}(O_0)$. The zero-keys $I_0, O_0 \in \mathbb{Z}_p$ are chosen uniformly at random by Alice, and Alice computes and sends $(w_I, w_O, [I_0], \{C_0, C_1, C_2\}, [O_0])$ to Bob. The three ciphertexts are sent in a randomly permuted order, so that Bob does not know which ciphertext encrypts which key.

Given potential keys $\mathcal{K}(w_I)$ for the input wire w_I, Bob computes potential keys for the output wire as follows:

$$\mathcal{K}(w_O) \stackrel{\text{def}}{=} \bigcup_{K \in \mathcal{K}(w_I)} \{D_K(C_0), D_K(C_1), D_K(C_2)\} \ .$$

If $K = I_c$ for $I_c \in \{I_0, I_1, I_2\}$, then it follows from $C_c \in \{C_0, C_1, C_2\}$ that $O_{\text{nt}(c)} \in \{D_K(C_0), D_K(C_1), D_K(C_2)\}$. Therefore

$$I_c \in \mathcal{K}(w_I) \quad \Rightarrow \quad O_{\text{nt}(c)} \in \mathcal{K}(w_O) \ .$$

Note that $|\mathcal{K}(w_O)| \leq 3|\mathcal{K}(w_I)|$. Our particular use of NT bricks will at all times ensure that $|\mathcal{K}(w_I)| = 1$, and thus $|\mathcal{K}(w_O)| \leq 3$. This is depicted by the numbers in circles next to the wires in Fig. 3.

Addition Bricks: An *addition brick* (Fig. 3 left, second to the top) has two input wires w_L and w_R and one output wire w_S. It contains two commitments $[L_0]$ and $[R_0]$, where $\mathcal{COM}(w_L) \stackrel{\text{def}}{=} [L_0]$, $\mathcal{COM}(w_R) \stackrel{\text{def}}{=} [R_0]$. We let $[S_0] = [L_0] \boxplus [R_0]$ and let $\mathcal{COM}(w_S) \stackrel{\text{def}}{=} [S_0]$. I.e., the zero-key S_0 for the output wire is $S_0 = L_0 + R_0 \bmod p$. Alice picks $L_0, R_0 \in \mathbb{Z}_p$ uniformly at random and sends $(w_L, w_R, w_S, [L_0], [R_0])$ to Bob.

Given potential keys $\mathcal{K}(w_L)$ and $\mathcal{K}(w_R)$ for the input wires, Bob computes potential keys for the output wire as follows:

$$\mathcal{K}(w_S) \stackrel{\text{def}}{=} \mathcal{K}(w_L) + \mathcal{K}(w_R) = \{L + R \bmod p | L \in \mathcal{K}(w_L) \wedge R \in \mathcal{K}(w_R)\} \ .$$

Note that $L_x + R_y \equiv_p (L_0 + x\Delta) + (R_0 + y\Delta) \equiv_p (L_0 + R_0) + (x+y)\Delta \equiv_p S_{x+y}$. Therefore, if $x, y \in \{0,1\}$, then

$$L_x \in \mathcal{K}(w_L) \wedge R_y \in \mathcal{K}(w_R) \quad \Rightarrow \quad S_{x+y} \in \mathcal{K}(w_S) \ .$$

Our particular use of addition bricks will at all times ensure that $|\mathcal{K}(w_L)| = 1$ and $|\mathcal{K}(w_R)| = 1$, and thus $|\mathcal{K}(w_S)| = 1$. This is depicted by the numbers in circles next to the wires in Fig. 3.

Shifts: The next component is the *shift* (Fig. 3 left, second to the bottom). Assume that Bob just evaluated a brick, like an NT brick, to get potential keys $\mathcal{K}(w)$ for its output wire w, and let $[K_0] = \mathcal{COM}(w)$ be the commitment associated to w. Bob needs to use $\mathcal{K}(w)$ as input for the next brick in the LEGO circuit. The next brick was, however, generated independently of the brick producing the potential keys $\mathcal{K}(w)$. This means that it has an input wire $w' \neq w$ and another associated commitment $[K'_0] = \mathcal{COM}(w')$. This is handled as follows: After Bob announces the position of the bricks in the LEGO circuit, Alice will open the commitment $[K'_0] \boxminus [K_0]$ to let Bob learn the *shifting value* $\Sigma = K'_0 - K_0 \bmod p$ for each pair of output wire w and input wire w', where w is to feed w'. We denote such a connection by $w \Longrightarrow w'$. Given Σ, Bob can compute

$$\mathcal{K}(w') \stackrel{\text{def}}{=} \mathcal{K}(w) + \Sigma = \{K + \Sigma \bmod p | K \in \mathcal{K}(w)\} \ .$$

Note that $K_c + \Sigma \equiv_p (K_0 + c\Delta) + (K'_0 - K_0) \equiv_p K'_0 + c\Delta \equiv_p K'_c$. Therefore

$$K_c \in \mathcal{K}(w) \quad \Rightarrow \quad K'_c \in \mathcal{K}(w') \ ,$$

unless Alice opens $[K'_0] \boxminus [K_0]$ to $\Sigma \neq K'_0 - K_0 \bmod p$, which would involve breaking the computational binding of the commitment scheme. We make this precise in the formal analysis. Clearly $|\mathcal{K}(w')| = |\mathcal{K}(w)|$.

KF Bricks: Already with the above components one can evaluate a NAND circuit $\mathcal{C}$, using one addition brick and one NT brick to securely evaluate each NAND gate in $\mathcal{C}$. The maximal size of potential-key sets, however, grows by a factor 3 after each NT gate and squares after each addition gate. We deal with this using the key-filter brick.

A *KF brick* (Fig. 3 left, bottom) has one input wire w_I and one output wire w_O. It consists of one commitment $[K_0]$ and $\mathcal{COM}(w_I) \stackrel{\text{def}}{=} [K_0]$ and $\mathcal{COM}(w_O) \stackrel{\text{def}}{=} [K_0]$. It also contains two hash values $T_0 = H(K_0)$ and $T_1 = H(K_1)$. Alice chooses K_0 uniformly at random and sends $([K_0], \{T_0, T_1\})$ to Bob, with T_0 and T_1 in a uniformly random order.

When Bob has potential-key set $\mathcal{K}(w_I)$ for the input wire he computes potential keys for w_O as follows

$$\mathcal{K}(w_O) \stackrel{\text{def}}{=} \{K \in \mathcal{K}(w_I) | H(K) \in \{T_0, T_1\}\} \ .$$

It clearly holds for $b \in \{0, 1\}$ that $K_b \in \mathcal{K}(w_I) \Rightarrow K_b \in \mathcal{K}(w_O)$. It is also clear that when Alice knows K_0 and K_1 such that $T_0 = H(K_0)$ and $T_1 = H(K_1)$, then unless the collision resistance of the hash function is broken, $\mathcal{K}(w_O) \subseteq \{K_0, K_1\}$. Therefore, except with negligible probability,

$$\mathcal{K}(w_O) = \mathcal{K}(w_I) \cap \{K_0, K_1\} \ .$$

NAND Composites: Take an addition brick $Add = (w_L, w_R, w_S, [L_0], [R_0])$ and an NT brick $NT = (w_I, w_O, [I_0], \{C_0, C_1, C_2\}, [O_0])$. We call $(Add, w_S \Longrightarrow w_I, NT)$ a NAND brick. I.e., Bob is given $\Sigma = I_0 - S_0 \bmod p$ so that he can shift the potential output keys from the *Add* brick to the NT brick. If for $x, y \in \{0,1\}$ we have $L_x \in \mathcal{K}(w_L)$ and $R_y \in \mathcal{K}(w_R)$, then $S_{x+y} \in \mathcal{K}(w_S)$ and therefore $O_{\mathrm{nt}(x+y)} \in \mathcal{K}(w_O)$. I.e.,

$$L_x \in \mathcal{K}(w_L) \wedge R_y \in \mathcal{K}(w_R) \quad \Rightarrow \quad O_{x \barwedge y} \in \mathcal{K}(w_O) \ .$$

The above describes our components and how they work when they are correct. We now proceed to describe how we deal with faulty components using replication. We start with the NT brick.

Consider the NT brick $NT = (w_I, w_O, [I_0], \{C_0, C_1, C_2\}, [O_0])$ connected to an addition brick above. If the encryptions of NT are not correct, it might not output the correct O_z. To deal with this, we use a fresh NT brick $NT' = (w'_I, w'_O, [I'_0], \{C'_0, C'_1, C'_2\}, [O'_0])$ and we add a new fresh wire name ν' to the circuit, with $\mathcal{COM}(\nu') = [O_0]$ and $\mathcal{Z}(\nu') = O_0$. We then connect the output wire of *Add* to the input wire of NT' and the output wire w'_O of NT' to ν' by adding $(w_S \Longrightarrow w'_I, NT', w'_O \Longrightarrow \nu')$ to the circuit design.

If for $x, y \in \{0,1\}$ we have $L_x \in \mathcal{K}(w_L)$ and $R_y \in \mathcal{K}(w_R)$, then $S_{x+y} \in \mathcal{K}(w_S)$ and therefore (by $w_S \Longrightarrow w'_I$) $I'_{x+y} \in \mathcal{K}(w'_I)$. So, if NT' is correct, then $O'_{\mathrm{nt}(x+y)} \in \mathcal{K}(w'_O)$ and thus (by $w'_O \Longrightarrow \nu'$) $O_{\mathrm{nt}(x+y)} \in \mathcal{K}(\nu')$. We already argued that if NT is correct, then $O_{\mathrm{nt}(x+y)} \in \mathcal{K}(w_O)$. This implies that if NT is correct *or* NT' is correct, then $O_{\mathrm{nt}(x+y)} \in \mathcal{K}(w_O) \cup \mathcal{K}(\nu')$. We can therefore add a fresh wire u to the circuit and let Bob compute $\mathcal{K}(u) = \mathcal{K}(w_O) \cup \mathcal{K}(\nu')$. We picture this as the wires w_O and ν' being joined in Fig. 3.

When using a total of ℓ NT bricks we call the structure an *ℓ-NAND composite* or *ℓ-NANDC*, as depicted in the top, center of Fig. 3. When we use a NANDC in a larger construction we depict it as the right, top graphic. It is clear that if at least one of the ℓ NT bricks is correct, then for $x, y \in \{0,1\}$ we have

$$L_x \in \mathcal{K}(w_L) \wedge R_y \in \mathcal{K}(w_R) \quad \Rightarrow \quad O_{x \barwedge y} \in \mathcal{K}(u) \ .$$

In our use of ℓ-NANDC's we always ensure that $|\mathcal{K}(w_L)| = |\mathcal{K}(w_R)| = 1$, which clearly implies that $|\mathcal{K}(u)| \leq 3\ell$.

KF Composites: Also KF bricks can be incorrect, e.g. by T_0 and T_1 being random values. If both are incorrect it is not a real problem as Bob will end up with $\mathcal{K}(w_O) = \emptyset$ in all cases. Alice might, however, create a brick where $T_0 = H(K_0)$ and T_1 is random. In that case Bob will get $\mathcal{K}(w_O) = \{K_0\}$ if $K_0 \in \mathcal{K}(w_I)$ and $\mathcal{K}(w_O) = \emptyset$ if $K_1 \in \mathcal{K}(w_I)$. In the first case he completes the protocol, but in the second case he has to terminate. Alice detects the termination by not receiving her keys, which allows her to learn the bit on the (possibly) internal wire w_O. To avoid this *leakage via conditional failure* we replicate key filters.

The simple observation is that if we run several KF bricks on the same potential keys, then a correct key will be contained in the output of all correct filters.

If we use $2\ell+1$ filters under the assumption that that at most ℓ are faulty, this allows us to pick the correct keys by majority voting.

To ease the presentation, we describe just the case $\ell = 1$. Consider three KF bricks $KF = (w_I, w_O, [K_0], \{T_0, T_1\})$, $KF' = (w'_I, w'_O, [K'_0], \{T'_0, T'_1\})$, $KF' = (w''_I, w''_O, [K''_0], \{T''_0, T''_1\})$, add fresh wires ν', ν'', v with $\mathcal{COM}(\nu') \stackrel{\text{def}}{=} [K_0]$, $\mathcal{COM}(\nu'') \stackrel{\text{def}}{=} [K_0]$ and $\mathcal{COM}(v) \stackrel{\text{def}}{=} [K_0]$, and add the shifts $w_I \Longrightarrow w'_I$, $w_I \Longrightarrow w''_I$, $w'_O \Longrightarrow \nu'$, $w'_O \Longrightarrow \nu''$. Let $\mathcal{K}(v)$ consist of the K which are found in at least two of the sets $\mathcal{K}(w_O), \mathcal{K}(\nu'), \mathcal{K}(\nu'')$. We write

$$\mathcal{K}(v) \stackrel{\text{def}}{=} \text{Maj}(\mathcal{K}(w_O), \mathcal{K}(\nu'), \mathcal{K}(\nu'')) \ .$$

If $K_c \in \mathcal{K}(w_I)$ and KF (resp KF', KF'') is correct, then $K_c \in \mathcal{K}(w_O)$ (resp. $\mathcal{K}(\nu'), \mathcal{K}(\nu'')$). So, if a majority of the KF bricks are correct, then $K_c \in \mathcal{K}(w_I) \Rightarrow K_c \in \mathcal{K}(v)$. Furthermore, since a correct KF brick only output keys in $\{K_0, K_1\}$, it follows that $\mathcal{K}(v) \subseteq \{K_0, K_1\}$. So, except with negligible probability

$$\mathcal{K}(v) = \mathcal{K}(w_I) \cap \{K_0, K_1\} \ .$$

We depict the KFC in Fig. 3 (center, bottom). When using a KFC in a larger construction we depict it as the right, second from the top graphic.

Overall Architecture: To evaluate a NAND circuit $\mathcal{C}$ with replication parameter ℓ we place one $(\ell+1)$-NANDC on each NAND gate G in $\mathcal{C}$ and attach a $(2\ell+1)$-KFC to the output wire of the NANDC — we connect them using the appropriate shift $u \Longrightarrow w_I$. We consider the two input wires w_L and w_R of the NANDC as the input wires of G and we consider the output wire v of the KFC as the output wire of G. We then use shifts to connect output wires of gates to input wires of other gates according to the architecture of $\mathcal{C}$.

In the evaluation Bob will learn input keys K_{x_w} for his input wires via an OT of K_0 and K_1. By letting Alice send the opening of $[K_{x_w}]$ in the OT, Bob can verify that K_{x_w} is correct. For Alice's input wires, she will simply send K_{x_w} to give input x_w on wire w. To prevent her from sending an incorrect key, $K' \notin \{K_0, K_1\}$ we add an extra $(2\ell+1)$-KFC and send K' through this filter before feeding it unto w. This ensures that $K' \in \{K_0, K_1\}$, if accepted.

In Fig. 3 (bottom, right) we depict this construction for $\ell = 1$ and $\mathcal{C}$ consisting of one NAND gate where Alice provides the left input and Bob provides the right input.

It is easy to see that if for each $(\ell+1)$-NANDC at most ℓ NT bricks are incorrect and for each $(2\ell+1)$-KFC at most ℓ KF bricks are incorrect, then the LEGO circuit will compute the correct result, in the following sense: If $\mathcal{Z}(w) + x_w\Delta \bmod p \in \mathcal{K}(w)$ for all input wires $w \in \mathcal{I}$ and $x_w \in \{0,1\}$, then $\mathcal{Z}(w) + y_w\Delta \bmod p \in \mathcal{K}(w)$ for all output wires $w \in \mathcal{O}$ and $\{y_w\}_{w\in\mathcal{O}} = \mathcal{C}(\{x_w\}_{w\in\mathcal{I}})$.

If in addition $\mathcal{K}(w)$ is a singleton set for all $w \in \mathcal{I}$, then $\mathcal{K}(w)$ will be a singleton set for all $w \in \mathcal{O}$, except with negligible probability. As detailed in the analysis, this follows from the use of KFC's and the fact that Bob could not compute both keys K_0 and K_1 for a wire even if he tried.

4 The Protocol

The overall protocol is given in Fig. 4. It depends on the NAND circuit $\mathcal{C}$ and a replication factor $\ell \in \mathbb{N}$.

Bricks Production: Details of how individual bricks are produced and which values are sent to Bob were given in Section 3. Alice will produce $P_{NT} = (1 + \tau_{NT})(\ell + 1)|\mathcal{C}|$ NT bricks and Bob will select a uniformly random subset of size $\tau_{NT}(\ell + 1)|\mathcal{C}|$ for testing. This leaves $(\ell + 1)|\mathcal{C}|$ NT bricks, which is sufficient to construct a $(\ell + 1)$-NANDC for each gate in $\mathcal{C}$. She generates $P_{KF} = (1 + \tau_{KF})(2\ell+1)(|\mathcal{C}|+|\mathcal{I}_A|)$ KF bricks and Bob tests a random subset of size $\tau_{KF}(2\ell+1)(|\mathcal{C}|+|\mathcal{I}_A|)$, leaving enough to construct a $(2\ell+1)$-KFC for each gate and each of Alice's input wires.

After sending the components, and *before* the tests, Alice proves to Bob that she knows the openings of all commitments in the bricks, as follows: Let $[K_0^{(1)}], \ldots, [K_0^{(n)}]$ be the set of all commitments contained in the bricks. Bob picks a uniformly random challenge $e \in_R \mathbb{Z}_p^n$ and sends e to Alice. Bob computes $[\sum_{i=1}^n e_i K_0^{(i)}] = \boxplus_{i=1}^n [K_0^{(i)}]^{e_i}$ and Alice computes the opening of $[\sum_{i=1}^n e_i K_0^{(i)}]$. Then Alice uses $\mathcal{F}_{\text{ZK}}$ to prove that she knows this opening. Bob terminates if the proof fails.

Addition bricks are by definition correct, as Bob lets $[S_0] = [L_0] \boxplus [R_0]$. For NT bricks and KF bricks Alice can cheat by not providing correct encryptions respectively hash values, which is the reason for the following tests.

NT Bricks: A test of $NT = (w_I, w_O, [I_0], [O_0], \{C_0, C_1, C_2\})$ proceeds as follows:

1. Alice sends a value telling Bob the correct order C_0, C_1, C_2 of $\{C_0, C_1, C_2\}$.
2. Bob computes $[O_1] = [O_0] \boxplus [\Delta]$, $[I_1] = [I_0] \boxplus [\Delta]$, $[I_2] = [I_1] \boxplus [\Delta]$ and sends challenge $e \in_R \{0, 1, 2\}$ to Alice.
3. Alice opens $[I_e]$ and $[O_{\text{nt}(e)}]$ to let Bob learn I_e and $O_{\text{nt}(e)}$.
4. Bob accepts iff the openings are correct and $E_{I_e}(O_{\text{nt}(e)}) = C_e$.

It is clear that if Alice can answer all three challenges correctly and cannot break the computational binding of the commitment scheme, then NT is a correct NT brick, i.e., it would produce the correct output key in $\{O_0, O_1\}$ on all input keys in $\{I_0, I_1, I_2\}$, where I_0 is the key in $[I_0]$ and and O_0 is the key in $[O_0]$. For now, we informally define *the key in* $[I_0], [O_0]$ as the value Alice can open $[I_0], [O_0]$ to. The formal analysis is, of course, more crisp.

KF Bricks: A test of $KF = (w_I, w_O, [K_0], \{T_0, T_1\})$ proceeds as follows:

1. Alice sends a bit telling Bob the correct order T_0, T_1 of $\{T_0, T_1\}$.
2. Bob computes $[K_1] = [K_0] \boxplus [\Delta]$ and sends a one bit challenge $e \in_R \{0, 1\}$ to Alice.
3. Alice opens $[K_e]$ to let Bob learn K_e.
4. Bob accepts iff the opening is correct and $T_e = H(K_e)$.

It is clear that if Alice can answer both challenges, then she can open $[K_0]$ to K_0 such that $H(K_0) = T_0$ and $H(K_0 + \Delta) = T_1$, or she can break the computational binding of the commitment scheme. Therefore KF is correct except with negligible probability. I.e., it would work correctly on both K_0 and K_1, where K_0 is the key in $[K_0]$.

5 Analysis

In the full version of this paper [NO08] we prove Theorem 1. Here we sketch the proof, but assuming that H is a random oracle. By a hybrid argument the random oracle can be replaced by a collision resistant function which is correlation resistant according to Def. 1.

5.1 Corrupted Bob

We first consider the case where Alice is honest and Bob is corrupted. We model H as a non-programmable and non-extractable random oracle.

The UC simulator $\mathcal{S}$ runs Alice's part of the protocol towards Bob completely honestly, except that it uses $x_w = 0$ for each of her input wires. If Bob sends keys in **Result announcement** which makes Alice reject, then $\mathcal{S}$ inputs `abort!` to $\mathcal{F}_{\text{SCE}}$ on behalf of Bob. Otherwise, $\mathcal{S}$ for each of Bob's wires takes $x_w \in \{0, 1\}$ to be the input of Bob to the OT associated to wire $w \in \mathcal{I}_B$ and inputs these x_w to $\mathcal{F}_{\text{SCE}}$ on behalf of Bob. That completes the description of $\mathcal{S}$.

It remains to argue that the views of the environment in the simulation and in the protocol are indistinguishable. When H is random they are in fact statistically close. There are the following differences between the simulation and the protocol:

1. In the simulation $x_w = 0$ for each of Alice's wires. In the protocol the x_w's have the values specified by the environment.
2. In the protocol Alice's output to the environment is computed from the values sent by Bob. In the simulation it is the value output by $\mathcal{F}_{\text{SCE}}$.

To handle the first difference we argue that Bob's view is statistically independent of Alice's input. To handle the second difference we argue that if Alice does not abort she outputs the same y_w values in the two settings, and we argue that she aborts with the same probability in the two settings.

It is straight-forward to verify that when H is a uniformly random function, then the LEGO circuit leaks no information on Alice's input to Bob unless he queries H on two different points $P, P' \in \{K_0, K_0 + \Delta, K_0 + 2\Delta\}$ for some zero-key K_0 of the bricks sent by Alice, and that until he makes such queries, Δ is uniformly random in the view of Bob. It follows from an easy application of the birthday bound that when Δ is uniformly random and independent of the view of Bob, the probability that he makes such queries is less than 2^{-s}, because of our choice of p.

Setup: We assume Alice and Bob agree on a generator g of order p. Bob picks uniformly random $h \in_R \langle g \rangle$ and sends it to Alice. This defines the commitment scheme $[K] = g^K h^r$. Alice picks a global difference $\Delta \in_R \mathbb{Z}_p$ and sends a commitment $[\Delta]$ to Bob and uses $\mathcal{F}_{\text{ZK}}$ to prove knowledge of the opening of $[\Delta]$.

Bricks production: Alice produces P_{NT} uniformly random and independent NT bricks, P_{KF} uniformly random and independent KF bricks and $P_{Add} = |\mathcal{C}|$ uniformly random addition bricks, and sends all these bricks to Bob. All bricks are produced using Δ.

Test: Bob selects some NT bricks and KF bricks for testing. For each such brick he picks a random input and Alice provides Bob with the keys needed to run on those inputs. Bob terminates if any test fails.

Bricks shuffling: For each NAND gate G in $\mathcal{C}$ Bob picks: a random unused addition brick, $\ell + 1$ random unused NT bricks and $2\ell + 1$ random unused KF bricks and assigns them to G. For each of Alice's input wires he picks $2\ell + 1$ random unused KF bricks and assigns them to w. He announce the assignment to Alice.

Bricks connection: The positions of the bricks chosen by Bob define a number of brick output wires w feeding unto brick input wires w'. For each such connection Alice and Bob add $w \Longrightarrow w'$ to the circuit by Alice opening $\mathcal{COM}(w') \boxminus \mathcal{COM}(w)$ to Bob. Bob terminates if any opening is incorrect.

Bob's input: For each input wire $w \in \mathcal{I}_B$ (with $\mathcal{Z}(w) = K_0$ and $\mathcal{COM}(w) = [K_0]$) Bob has an input $x_w \in \{0,1\}$. Alice and Bob run an OT where Alice inputs messages $m_0^{(w)} = (K_0, r_0)$ and $m_1^{(w)} = (K_1, r_1)$ and Bob inputs the selection bit x_w — here (K_0, r_0) is the opening of $[K_0]$ and (K_1, r_1) is the opening of $[K_1] = [K_0] \boxplus [\Delta]$. Bob terminates if $m_{w_x}^{(w)}$ is not an opening of $[K_{w_x}]$. Otherwise he lets $\mathcal{K}(w) \stackrel{\text{def}}{=} \{K_{x_w}\}$ for $w \in \mathcal{I}_B$.

Alice's input: For each input wire $w \in \mathcal{I}_A$ (with $\mathcal{Z}(w) = K_0$ and $\mathcal{COM}(w) = [K_0]$) Alice has an input $x_w \in \{0,1\}$. She sends K_{x_w} to Bob and Bob lets $\mathcal{K}(w) = \{K_{x_w}\}$. Bob evaluates the KFC for w on $\mathcal{K}(w)$ to get $\mathcal{K}(w')$. Since $|\mathcal{K}(w)| = 1$ and $\mathcal{K}(w') = \mathcal{K}(w) \cap \{K_0, K_1\}$, it follows that $|\mathcal{K}(w')| \leq 1$ and $\mathcal{K}(w') \subset \{K_0, K_1\}$. If $|\mathcal{K}(w')| = 1$ for all of Alice's wires w, Bob adopts the sets $\mathcal{K}(w')$ for $w \in \mathcal{I}_A$, otherwise he terminates.

Garbled evaluation: If Bob did not yet terminate, he holds a singleton set $\mathcal{K}(w) \subset \{K_0, K_1\}$ for all input wires $w \in \mathcal{I}$, where $K_0 = \mathcal{Z}(w)$. Now Bob computes singleton sets $\mathcal{K}(w) \subset \{K_0, K_1\}$ for all output wires $w \in \mathcal{O}$, where $K_0 = \mathcal{Z}(w)$. Details of how this is done were given in Section 3.

Result announcement: For each output wire w, Bob sends $(w, \mathcal{K}(w))$ to Alice. Alice terminates if any set $\mathcal{K}(w)$ is not a singleton. Otherwise she picks $K \in \mathcal{K}(w)$ and tries to write K as $K = \mathcal{Z}(w) + y_w \Delta \bmod p$ for $y_w \in \{0,1\}$. If this fails, she terminates the protocol; Otherwise, she adopts y_w as the output bit for wire w.

Fig. 4. The LEGO protocol

Now note that from the keys $K_{x_w}, w \in \mathcal{I}_A$ sent by Alice and the keys $K_{x_w}, w \in \mathcal{I}_B$ Bob chooses in the OT's, he can compute K_{y_w} for each $w \in \mathcal{O}$, where $\{y_w\}_{w \in \mathcal{O}} = \mathcal{C}(\{x_w\}_{w \in \mathcal{I}})$. So, Bob *knows* the correct output keys K_{y_w}. If Bob sends K_{y_w} for all $w \in \mathcal{O}$, then Alice accepts and outputs the correct values y_w.

If Bob sends $K' \neq K_{y_w}$ for some wire, then Alice rejects unless $K' = K_{1-y_w}$. But then, if $K' = K_{1-y_w}$, Bob could query H on two different points $K_{y_w}, K' \in \{K_0, K_0+\Delta, K_0+2\Delta\}$, and we already argued that this happens with probability less than 2^{-s}. So, the probability that Alice aborts is statistically close to the probability that Bob sends some $K' \neq K_{y_w}$.

5.2 Corrupted Alice

We then consider the case where Bob is honest and Alice is corrupted.

The simulator: The simulator runs Bob honestly in **Setup**, **Bricks production**, **Test**, **Bricks shuffling**, **Bricks connection**, **Bob's input** and **Alice's input**, except that:

1. It inspects Alice's input to $\mathcal{F}_{\text{ZK}}$ in **Setup**. If the input to $\mathcal{F}_{\text{ZK}}$ is not an opening of $[\Delta]$, it inputs `abort!` to $\mathcal{F}_{\text{SCE}}$ on behalf of Alice. Otherwise it records Δ.
2. For each of Bob's input wires $w \in \mathcal{I}_B$ it records both of Alice's inputs $m_0^{(w)}$ and $m_1^{(w)}$ to $\mathcal{F}_{\text{OT}}$ in **Bob's input**. For $c = 0, 1$, if $m_c^{(w)}$ is a valid opening of the $[K_c]$ defined in **Bob's input**, it defines $V_c^{(w)} \stackrel{\text{def}}{=} K_c$. If $m_c^{(w)}$ is not an opening of $[K_c]$, it defines $V_c^{(w)} \stackrel{\text{def}}{=} \perp$.

Plain abort: If Alice (formally the adversary) makes Bob abort in any of the steps run so far, e.g. by sending a key for one of Alice's input wires which is not accepted by the corresponding KFC, then $\mathcal{S}$ inputs `abort!` to $\mathcal{F}_{\text{SCE}}$ to make it output `abort!` to the environment, exactly as Bob would have done in the protocol. The only step in which $\mathcal{S}$ cannot compute whether Bob would have aborted is in **Bob's input**, as $\mathcal{S}$ does not know the inputs $\{x_w\}_{w \in \mathcal{I}_w}$ of Bob — these were input to $\mathcal{F}_{\text{SCE}}$ by the environment. This is handled as follows.

Handling conditional failures: If $V_0^{(w)} = V_1^{(w)} = \perp$ for some $w \in \mathcal{I}_B$, then $\mathcal{S}$ inputs `abort!` to $\mathcal{F}_{\text{SCE}}$ on behalf of Alice. Note that Bob would always abort in the protocol in this case, as he would always receive an incorrect opening for the wire w. If not already terminated, then $\mathcal{S}$ computes $W = \{w \in \mathcal{I}_B | V_0^{(w)} = \perp \vee V_1^{(w)} = \perp\}$ and for $w \in W$ sets $\beta_w \in \{0, 1\}$ to be the value where $V_{\beta_w}^{(w)} \neq \perp$. It then inputs $(W, \{\beta_w\}_{w \in W})$ to $\mathcal{F}_{\text{SCE}}$ on behalf of Alice in **Guess**. Note that $\mathcal{F}_{\text{SCE}}$ aborts iff $\beta_w \neq x_w$ for some $w \in W$. By construction of the protocol and definition of the $V_c^{(w)}$, Bob would have aborted in the protocol iff $V_{x_w}^{(w)} \neq K_{x_w}$, if he was running with the inputs $\{x_w\}_{w \in \mathcal{I}_B}$ input to $\mathcal{F}_{\text{SCE}}$ by the environment. Therefore $\mathcal{S}$ makes $\mathcal{F}_{\text{SCE}}$ abort iff Bob would have aborted in the protocol. So, until now the simulation is perfect.

Extracting inputs: If $\mathcal{S}$ did not yet make $\mathcal{F}_{\text{SCE}}$ abort, it must now give an input to $\mathcal{F}_{\text{SCE}}$ on behalf of Alice. Note that if $\mathcal{S}$ did not yet make $\mathcal{F}_{\text{SCE}}$ abort, then it knows a key $K^{(w)}$ for each $w \in \mathcal{I}_A$, namely the keys obtained by running the KFC's on the keys sent by Alice in **Alice's input**. The simulator uses Δ to

compute $K_{-1}^{(w)} = K_{x_w}^{(w)} - \Delta$ and $K_{+1}^{(w)} = K_{x_w}^{(w)} + \Delta$.[1] Then it runs the KFC for input wire w on $\mathcal{K} = \{K_{-1}^{(w)}, K^{(w)}, K_{+1}^{(w)}\}$. If the output from the KFC does not consist of two keys K and K' where $K - K' \in \{\Delta, -\Delta\}$ or $K^{(w)} \notin \{K, K'\}$, then $\mathcal{S}$ makes $\mathcal{F}_{\text{SCE}}$ abort. Otherwise it names and orders the two keys as $K_0^{(w)}, K_1^{(w)}$ such that $K_1^{(w)} = K_0^{(w)} + \Delta$ and computes $x_w \in \{0,1\}$ such that $K^{(w)} = K_{x_w}^{(w)}$. It inputs $\{x_w\}_{w \in \mathcal{I}_A}$ to $\mathcal{F}_{\text{SCE}}$ and gets back $\{y_w\}_{w \in \mathcal{O}} = \mathcal{C}(\{x_w\}_{w \in \mathcal{I}_A} \cup \{x_w\}_{w \in \mathcal{I}_B})$.

Evaluation: By now $\mathcal{S}$ has a key $K^{(w)}$ for each $w \in \mathcal{I}_A$. It lets $\mathcal{K}(w) = \{K^{(w)}\}$. For each $w \in \mathcal{I}_B$ it lets $\mathcal{K}(w) = \{V_0^{(w)}\}$ if $V_0^{(w)} \neq \perp$ and $\mathcal{K}(w) = \{V_1^{(w)}\}$ if $V_0^{(w)} = \perp$. Note that at this point $V_1^{(w)} \neq \perp$ when $V_0^{(w)} = \perp$. It then evaluates the LEGO circuit on these $\mathcal{K}(w)$. If the evaluation fails, then it makes $\mathcal{F}_{\text{SCE}}$ abort. Otherwise it computed $\mathcal{K}(w) = \{K^{(w)}\}$ for $w \in \mathcal{O}$. It computes $\mathcal{K} = \{K_{-1}^{(w)}, K^{(w)}, K_{+1}^{(w)}\}$ as above and runs the KFC from the gate computing $K^{(w)}$ on $\mathcal{K}$. If the output does not consist of two keys which can be named and ordered such that $K_1^{(w)} = K_0^{(w)} + \Delta$ it makes $\mathcal{F}_{\text{SCE}}$ abort. Otherwise, it sends $\{K_{y_w}^{(w)}\}_{w \in \mathcal{O}}$ to Alice, where the y_w are the values received from $\mathcal{F}_{\text{SCE}}$.

That completes the description of the simulator.

Analysis: We argued that the simulation is perfect up until the evaluation. What remains is to argue that the simulator aborts in the evaluation with the same probability as Bob would in the protocol and that when it does not abort, then the keys $\{K_{y_w}^{(w)}\}_{w \in \mathcal{O}}$ sent to Alice have the same distribution in the protocol and the simulation. This boils down to arguing that the LEGO circuit is correct if Bob does not abort before the evaluation phase.

Defining *good* and *bad*: We start the analysis by dividing bricks into *good* and *bad* and use this to define good and bad composites.

Consider the entire state of an execution after Alice sent all bricks to Bob and before Bob sends the challenge $e \in \mathbb{Z}_p^n$. Define q to be the probability that Bob accepts Alice's proof for $[K_e] \stackrel{\text{def}}{=} [\sum_{i=1}^n e_i K_0^{(i)}]$, when run from this fixed state (formally we fix the random tape of the environment and the adversary). If $q \leq 2/p$ we define all bricks to be *bad*. Otherwise we extract the openings of all commitments and use these to define the goodness of the bricks, as described now.

For a fixed state of Alice she will for a fixed $e \in E \stackrel{\text{def}}{=} \mathbb{Z}_p^n$ input a correct opening of $[K_e]$ to $\mathcal{F}_{\text{ZK}}$ with probability 0 or 1. Let E_{g} denote the $e \in E$ where the probability is 1. Then $q = |E_{\text{g}}|/|E|$, and we have a poly-time oracle for computing $K_e \stackrel{\text{def}}{=} \sum_{i=1}^n e_i K_0^{(i)}$ for $e \in E_{\text{g}}$: Given any $e \in E$ we can run the execution from the fixed state until Alice inputs to $\mathcal{F}_{\text{ZK}}$. If $e \in E_{\text{g}}$, this gives us K_e. Let n denote the number of commitments $[K_i]$. It is clear that if we get K_{e^i} for n linear independent values $e^i \in \mathbb{Z}_p^n$, then we can efficiently solve for openings of all $[K_i]$. We get the e^i by querying the oracle on several uniformly

[1] Both additions are modulo p. Below we will continue not explicitly mentioning the mod p when computing on elements from $\mathbb{Z}_p$.

random $e \in E$. At any point, let $\{e^i\}$ denote the e^i on which we got an opening of $[K_{e^i}]$. We continue querying as long as $\text{span}\{e^i\} \neq \mathbb{Z}_p^n$. If we query on a fresh uniformly random $e \in E$, then the probability that $e \in \text{span}\{e^i\}$ is at most $1/p$. The probability that $e \in E_{\text{g}}$ is q. Therefore the probability that $e \in E_{\text{g}} \setminus \text{span}\{e^i\}$ is at least $q - 1/p \geq q/2$. It follows that the expected number of queries to get $\text{span}\{e^i\} = \mathbb{Z}_p^n$ is at most $n(q/2)^{-1} = 2nq^{-1}$. The expected running time is therefore $\text{poly}(s)q^{-1}$. We only need to run the above extraction when Bob accepts the proof in the simulation, which he does with probability q. Therefore the expected running time of running the simulation once and extracting if Bob accepts is $q\,\text{poly}(s)q^{-1} = \text{poly}(\kappa)$.

After having extracted openings of all commitments we let $\mathcal{Z}(w) = K_0$ for all wires, where $[K_0] = \mathcal{COM}(w)$ and K_0 is the key extracted from $[K_0]$.

We then define a KF brick $(w_I, w_O, [K_0], \{T_0, T_1\})$ to be *good* if there is an ordering T_0, T_1 of $\{T_0, T_1\}$ such that $T_0 = H(K_0)$ and $T_1 = H(K_0 + \Delta)$. We define an NT brick $(w_L, w_R, w_O, [L_0], [R_0], [O_0], \{C_0, C_1, C_2\})$ to be *good* if there is an ordering C_0, C_1, C_2 of $\{C_0, C_1, C_2\}$ such that $C_0 = H(I_0) + O_0 + \Delta$, $C_1 = H(I_0 + \Delta) + O_0 + \Delta$ and $C_2 = H(I_0 + 2\Delta) + O_0$. All other KF and NT bricks are defined to be *bad*.

We already now note that Alice cannot later open shifting values incorrectly: Assume that Alice opens $[K_0'] \boxminus [K_0]$ to $\Sigma \neq K_0' - K_0$ as part of implementing $w \Longrightarrow w'$ (here $[K_0] = \mathcal{COM}(w), K_0 = \mathcal{Z}(w), [K_0'] = \mathcal{COM}(w'), K_0' = \mathcal{Z}(w')$). We can then use the opening of $[K_0']$ to K_0', the opening of $[K_0]$ to K_0 and the opening of $[K_0'] \boxminus [K_0]$ to Σ to compute an opening of e.g. $[K_0]$ to two different values. Since all three openings originate from Alice[2] and were computed in expected poly-time, we broke the computational binding of the commitment scheme. We also note the easy fact that if a KF brick is defined to be bad, then there exists at least one challenge $e \in \{0, 1\}$ on which Alice cannot make Bob accept a test, and if an NT brick is defined to be bad, then there exists at least one challenge $e \in \{0, 1, 2\}$ on which Alice cannot make Bob accept a test.[3]

We call a composite *bad* if it consists of more than ℓ bad composites. Otherwise it is *good*. An NANDC consists of $\ell+1$ bricks and a KFC consists of $2\ell+1$ bricks. As a consequence they will work correctly if they are good. It then follows from Section 3 that a LEGO circuit consisting of only good composites will compute correct $\mathcal{K}(w)$ for all $w \in \mathcal{O}$.

We first analyze the probability that any NANDC is bad. Let $P = (1+\tau)(\ell+1)|\mathcal{C}|$ be the number of NT bricks produced by Alice, let B be the number of NT bricks being defined as bad, let $T = \tau(\ell+1)|\mathcal{C}|$ be the number of NT bricks being tested, and let $U = (\ell+1)|\mathcal{C}|$ be the number of NT bricks used in the LEGO circuit.

Split each good NT brick into three green balls and split each bad NT brick into two green balls and a red ball. A ball represents one of the values a specific brick can be tested on. Each bad brick has at least one value which will catch

[2] The two first ones from extraction and the last because she sent it.

[3] In both case, unless she breaks the computational binding of the commitment scheme.

Alice if Bob tests on that value — this is the red ball. There are $3P$ balls and B are red. We analyze the game where Bob chooses T balls at random and accepts if they are all green. This upper bounds his probability of accepting in the protocol, except for a negligible amount coming from Alice's possible breaking of the commitment scheme. The probability that Bob accepts in the game is upper bounded by $\left(\frac{3P-B}{3P}\right)^T$. The probability that a given NANDC is bad is upper bounded by $\left(\frac{B}{U}\right)^{\ell+1}$, so by a union bound, the probability that Bob accepts and there is at least one bad NANDC is upper bounded by

$$\left(\frac{3P-B}{3P}\right)^T |\mathcal{C}| \left(\frac{B}{U}\right)^{\ell+1} = |\mathcal{C}| \left(\left(\frac{3P-B}{3P}\right)^{\tau|\mathcal{C}|} \frac{B}{U}\right)^{\ell+1} . \tag{1}$$

This is maximal in B when $(3P-B)^{\tau|\mathcal{C}|} B$ is maximal in B, which it is when $B = 3P(1+\tau|\mathcal{C}|)^{-1} = 3(1+\tau)(\ell+1)|\mathcal{C}|(1+\tau|\mathcal{C}|)^{-1} \approx 3\frac{1+\tau}{\tau}(\ell+1) = 3(1+\tau^{-1})(\ell+1)$. We use $B = 3(1+\tau^{-1})(\ell+1)$. Then $B/U = 3(1+\tau^{-1})|\mathcal{C}|^{-1}$, and $(3P-B)(3P)^{-1} = 1 - B(3P)^{-1} = 1-(1+\tau^{-1})(1+\tau)|\mathcal{C}|^{-1} = 1-\tau^{-1}|\mathcal{C}|^{-1}$. So, $\left(\frac{3P-B}{3P}\right)^{\tau|\mathcal{C}|} = (1-\tau^{-1}|\mathcal{C}|^{-1})^{\tau|\mathcal{C}|} \approx e^{-1}$. Plugging this into (1), we get an error probability of $|\mathcal{C}|^{-\ell}\left(\frac{3(\tau+1)}{e\tau}\right)^{\ell+1}$. Isolating for a probability of 2^{-s} that there is a bad NT composite we get

$$\ell \approx \frac{s + 0.1423 + \log\frac{\tau+1}{\tau}}{\log|\mathcal{C}| - 0.1423 - \log\frac{\tau+1}{\tau}} = O(s/\log|\mathcal{C}|) .$$

We round up this value to the nearest integer to get the replication factor for the protocol. A more careful analysis without the approximations shows that this actually gives a probability of $O(2^{-s})$ that Bob accepts and yet there is a NANDC being defined as bad.

Correctness: In the full version [NO08] we give a similar analysis for KFCs and recommend optimal values of τ for a wide range of circuit sizes and desired security levels. Here it suffices to note that for $\ell = O(s/\log|\mathcal{C}|)$ also the probability that Bob accepts and there is a bad KFC is $O(2^{-s})$. Therefore, if Bob accepts (in the protocol or simulation) then all composites are good, except with negligible probability. As described in Section 3, this implies that Bob will be able to evaluate the LEGO circuit in both settings, or at least abort with negligible probability. Furthermore, in the protocol the keys sent to Alice are $\{K_{y_w}^{(w)}\}_{w\in\mathcal{O}}$, where $K_0^{(w)}$ is the key she can open $\mathcal{COM}(w)$ to (by the extraction argument) and $\{y_w\}_{w\in\mathcal{O}} = \mathcal{C}(\{x_w\}_{w\in\mathcal{I}_A} \cup \{x_w\}_{w\in\mathcal{I}_B})$ for Bob's inputs $\{x_w\}_{w\in\mathcal{I}_B}$ and inputs $\{x_w\}_{w\in\mathcal{I}_A}$ defined by the keys $\{K_{x_w}^{(w)}\}_{w\in\mathcal{I}_A}$ sent by Alice and the keys $K_0^{(w)}$ she can open the commitment of the input KFCs to. By construction of the simulator the keys sent to Alice are $\{K_{y_w}^{(w)}\}_{w\in\mathcal{O}}$, where $K_0^{(w)}$ is the key she can open $\mathcal{COM}(w)$ to and $\{y_w\}_{w\in\mathcal{O}} = \mathcal{C}(\{x_w\}_{w\in\mathcal{I}_A} \cup \{x_w\}_{w\in\mathcal{I}_B})$ for Bob's inputs $\{x_w\}_{w\in\mathcal{I}_B}$ (held by the $\mathcal{F}_{\text{SCE}}$) and the inputs $\{x_w\}_{w\in\mathcal{I}_A}$ extracted from the keys sent by Alice. So, it suffices to argue that the extraction produces

the correct values of x_w for $w \in \mathcal{I}_A$: Since Bob did not abort, the KFC for w produced a unique output. Since the KFC is correct, the output is of the form $K^{(w)} = K^{(0)} + c\Delta$, defined relative to the $K^{(0)}$ Alice can open $\mathcal{COM}(w)$ to and $c \in \{0,1\}$. Therefore $\{K_0^{(w)}, K_1^{(w)}\} \subset \{K_{-1}^{(w)}, K^{(w)}, K_{+1}^{(w)}\}$. Therefore the KFC outputs $\mathcal{K} \cap \{K_0^{(w)}, K_1^{(w)}\} = \{K_0^{(w)}, K_1^{(w)}\}$. By construction this leads to $\mathcal{S}$ computing the $x_w \in \{0,1\}$ for which $K^{(w)} = K_0^{(w)} + x_w\Delta$.

Acknowledgments

We thank the TCC reviewers for their suggestions on improving the presentation and in particular Yuval Ishai who provided valuable feedback during our writing of the conference version.

References

[GMW86] Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: FOCS (1986)

[IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)

[IPS08] Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)

[JS07] Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)

[KS08] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)

[LP04] Lindell, Y., Pinkas, B.: A proof of Yao's protocol for secure two-party computation. Electronic Colloquium on Computational Complexity (2004)

[LP07] Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)

[LPS08] Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)

[MF06] Mohassel, P., Franklin, M.K.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)

[NN01] Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: STOC (2001)

[NO08] Nielsen, J.B., Orlandi, C.: Lego for two party secure computation. Cryptology ePrint Archive, Report 2008/427 (2008), http://eprint.iacr.org/

[Woo07] Woodruff, D.P.: Revisiting the efficiency of malicious two-party computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer, Heidelberg (2007)

[Yao82] Yao, A.C.: Protocols for secure computations (extended abstract). In: FOCS (1982)

[Yao86] Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS (1986)