ON SECURE TWO-PARTY COMPUTATION

TOOMAS KRIPS

Research Seminar in Cryptography
Supervisor: Helger Lipmaa
University of Tartu
Fall 2013

1. Introduction

1.1. **Secure Two-Party Computation.** In essence, secure multiparty computation is the solution to the problem how to compute on private inputs from several parties when we do not trust other parties and we don't have a central authority who can get inputs, perform computations and give outputs to those that need them. Here we will talk about secure two-party computation, where Alice, holding x_a and Bob, holding x_b, wish to learn the outcome of $f(x_a, x_b)$ for some function f without revealing anything else. The original example used by Yao, when he proposed the problem in 1982 in [1] is the millionaire problem, where two millionaires want to find out who is richer without the other millionaire learning how much money they have, but there are also more practical examples. Yao proposed the *garbled circuits* solution — we construct a circuit that evaluates the function f where gates and wires have been replaced with cryptographic primitives. This report aims to give a short overview of two recent approaches in secure two-party computation — one is based on the [2] and the other is the LEGO protocol, proposed by [3] and improved by [4]. First, we will describe how garbled circuits are usually done.

2. Garbled circuits

2.1. **Oblivious Transfer.** We will be needing a protocol called Oblivious Transfer, or OT for short. In this protocol, Alice holds two messages, m_0 and m_1 and Bob holds a bit b. As a result of this protocol, Bob should learn m_b but nothing about the other message m_{1-b} and Alice should learn nothing about b.

2.2. **Garbled circuits.** We will now describe garbled circuits. In [5], a garbling scheme is described as an algorithm *Garble* that takes in a function $f : \{0,1\}^m \to \{0,1\}^n$ and outputs a triple of functions (e, d, F) where $f = d \circ F \circ e$— it decomposes f into three functions. Roughly speaking, F is the garbled version of f, e is an encoding function that takes the inputs $x \in \{0,1\}^m$ of f and maps them to the inputs of F and d is the decoding function that takes the outputs of F and maps them to the outputs of f. Evaluating F shouldn't, broadly speaking, give too much information about the input x. Note that we can think of a garbled circuit as a protocol on its own that might or might not have certain

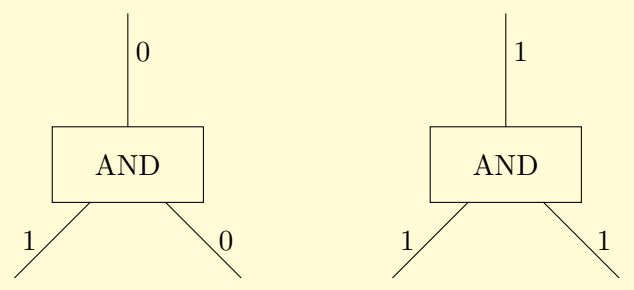

FIGURE 1. AND-gate evaluated in two possible ways

properties. Later on, when we talk about a certain article, we require that a garbling scheme has certain properties in order for it to be usable in a certain setting.

More specifically, garbling schemes are often achieved by using *projective garbled circuits,* which we will now describe and which will be central throughout this report. We will speak of boxes, keys and the contents of the boxes here to make the text easier to picture, but by boxes we mean encryptions, by the keys we mean the decryption keys and by the contents of the boxes we mean the encrypted information. When we say that a box Cneeds to be opened by keys α and β to get the content t inside, then we mean that $c = Dec_\alpha(Dec_\beta(C))$.

To put the idea of garbled circuits shortly, Alice makes lots of boxes so that every box contains a key that can be used to unlock other boxes. Each gate consists of four boxes, only one of which can be unlocked with the keys that Bob has gotten before. Alice sends all the boxes to Bob and tells him how they are connected. Now Bob gets some keys from Alice using Oblivious Transfer and starts unlocking as much as he can. In the end, if everybody acts as they are supposed to act, Bob should be able to unlock the boxes that either contain the right answers or that contain the encryptions of the right answers that Bob can then give Alice, who can then decrypt and see the answer. Here e is a function that maps the input the first keys that Bob gets from Alice using oblivious transfer, d is the function that Alice uses to decrypt the value that Bob told her and F is everything between those two — i.e. the process of Bob unlocking boxes.

We will give now a more detailed description of how garbled circuits are usually done. An ordinary circuit consists of wires and gates. When the input wires get input bits, then we can evaluate the wires — for example, if wire W_1 comes out of an AND-gate into which go wires W_2 and W_3, and wires W_2 and W_3 happen to hold values 1 and 0, then the wire W_1 will hold the value 0. Wires can potentially carry two values - 0 and 1. Figure 1 is a diagram of an AND-gate being evaluated in two possible ways.

The garbled circuit is similar in a manner — for every wire W^α, two values W_0^α and W_1^α are generated and while the circuit is evaluated, then the wire will hold either of those two values, signifying which bit the wire is carrying. We signify the value of the wire W^α by $W^{\alpha'}$ — $W^{\alpha'}$ is the value that Bob will know at the point when the wire has been evaluated.

If we evaluated the non-garbled circuit with some inputs $b_1, \ldots, b_k$, and the wire W^α carried the bit b, then if the garbled version of the same circuit is evaluated with the same inputs $b_1, \ldots, b_k$, then the wire W^α should be carrying the value W^α_b.

Suppose now that we have an AND-gate gg with left input wire L and right input wire R and output wire O. The gate consists of four ciphertexts (or boxes), $C_{0,0}, C_{0,1}, C_{1,0}$ and $C_{1,1}$. These boxes are permuted by Alice so that Bob does not know which box is which. The values carried by the wires W^α_0 and W^α_1 are actually different keys that can open some of the boxes — or to be more precise, to open the box $C_{i,j}$, one needs to know keys L_i and R_j. The box $C_{a,b}$ contains the key $O_{a\wedge b}$ so that the wire O will take the value $O_{a\wedge b}$

Input wires will be evaluated first. If an input wire W is supposed to be evaluated by Alice with some bit a, then she will just send Bob W_a without telling Bob what a is. If the input is supposed to come from Bob, Bob will get W_b from Alice by Oblivious Choice.

Figure 2 depicts a diagram of a garbled AND-gate that is being evaluated in two possible ways. Note that what is going on is exactly the same thing that is going on at Diagram 1 with the same values being involved. We give two possible evaluations of the same gate, but in order to picture Alice permuting the boxes in a random manner, the order in different evaluations is different.

At the end, there may be two different possibilities that we may want from the protocol — whether Bob should learn the final result $f(x_a, x_b)$ or not.

If Bob should not learn the answer, then when he opens the last boxes the values he gets from them should be just values of output wires from these gates: $O^{1'}, \ldots, O^{k'}$. Bob cannot tell whether $O^{i'} = O^i_0$ or $O^{i'} = O^i_1$ for any i and thus has to show the result to Alice, who will know whether $O^{i'} = O^i_0$ or $O^{i'} = O^i_1$ since, after all, she prepared them.

If Bob should learn the answer, then when he opens the last boxes the values he gets from them are not $O^0_{b_0}, \ldots, O^k_{b_k}$ but just the bits $b_0, \ldots, b_k$.

3. Security problems and how to solve them

There was a young lady called Alice
Who said to the Buckingham Palace:
"Let the crown be an x,
Put it in my box,
I'll find its worth, honest, no malice!"

The method described here is secure when all parties diligently follow the protocol (although at some point they may wish to look at the outputs of the other party and ponder whether this tells them anything). This is known as the semi-honest model — all the parties follow the protocol. The problem is what happens if some parties do not follow the protocol in order to learn more than they should. This is called the malicious model.

3.1. **Constructing different circuits.**

One of the main risks is that Alice might construct a different garbled circuit with the same shape that doesn't evaluate the function f that we want but some other function g. This other function g may give her additional knowledge about Bob's input. For example, it might be that $g(x_a, x_b) = x_b$. Note that when the topology (i.e. how gates are connected to each other) of the garbled circuit is different

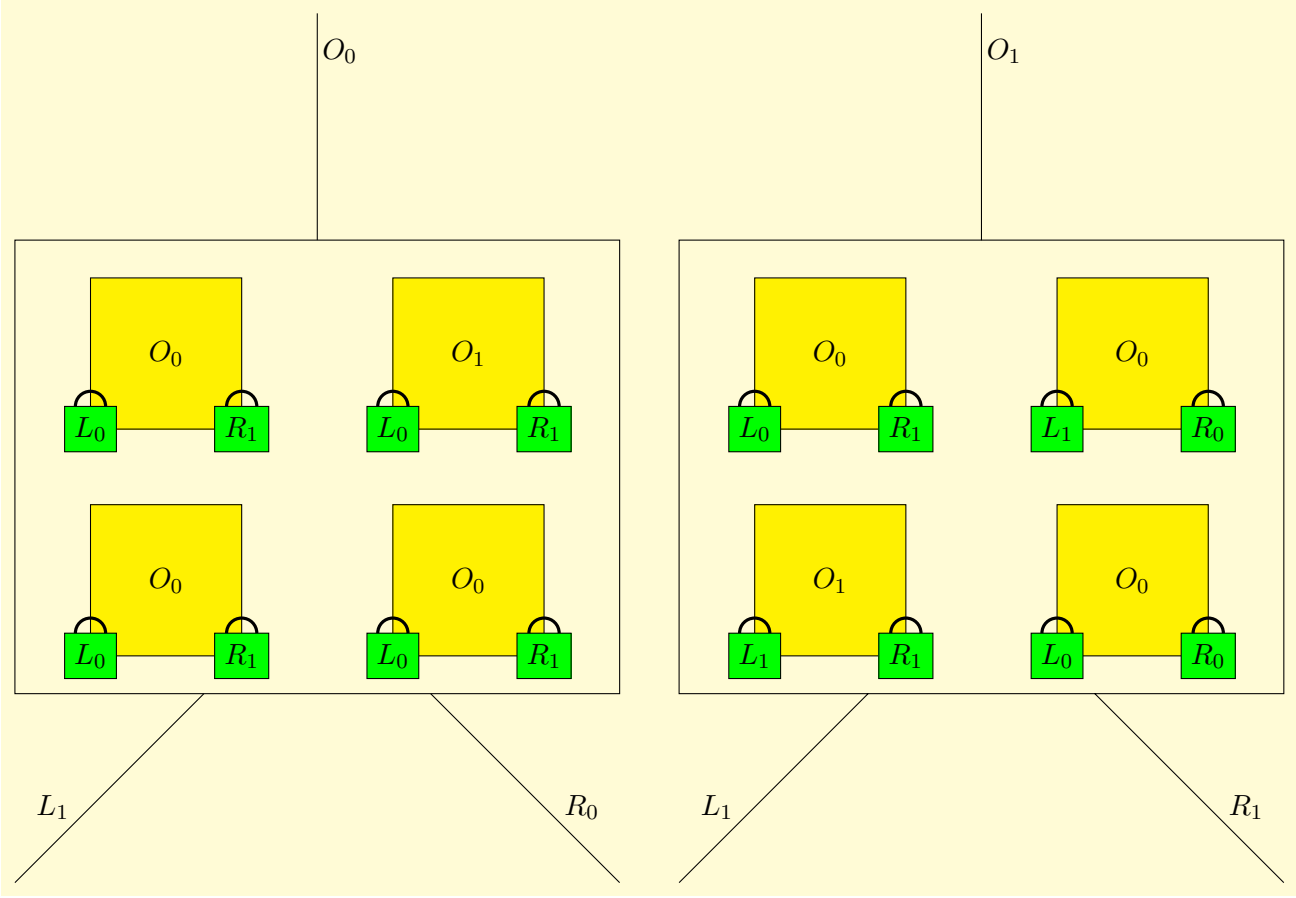

FIGURE 2. Garbled AND-gate evaluated in two possible ways

from the circuit of f, then Bob can detect that something is wrong without any extra procedures, but we can not rely on that. Observe the Figure 3 — to Bob, this garbled gate is indistinguishable from an AND-gate, since he can only open one of the boxes and cannot tell anything about the contents of the other boxes, yet it encodes a gate that outputs the input from the left wire. Note that Alice may replace any binary gate with any other binary gate as she pleases and in the model that we have described so far, Bob has no possibility of finding out that she did so.

3.2. **Selective failure.** Imagine that the circuit that Alice gives Bob is proper, but when Bob wants to perform OT with bit b on some key-pair $\{K_0, K_1\}$, she guesses that $b = 0$ and lets him perform OT on key-pair $\{K_0, K^*\}$ where K^* is a randomly chosen string that can not be used to open anything. If she was right in guessing that $b = 0$, then Bob gets the same key as he would have gotten anyway and the process terminates and Alice receives the correct answer Z as an output. However, if she was wrong and $b = 1$, then Bob cannot

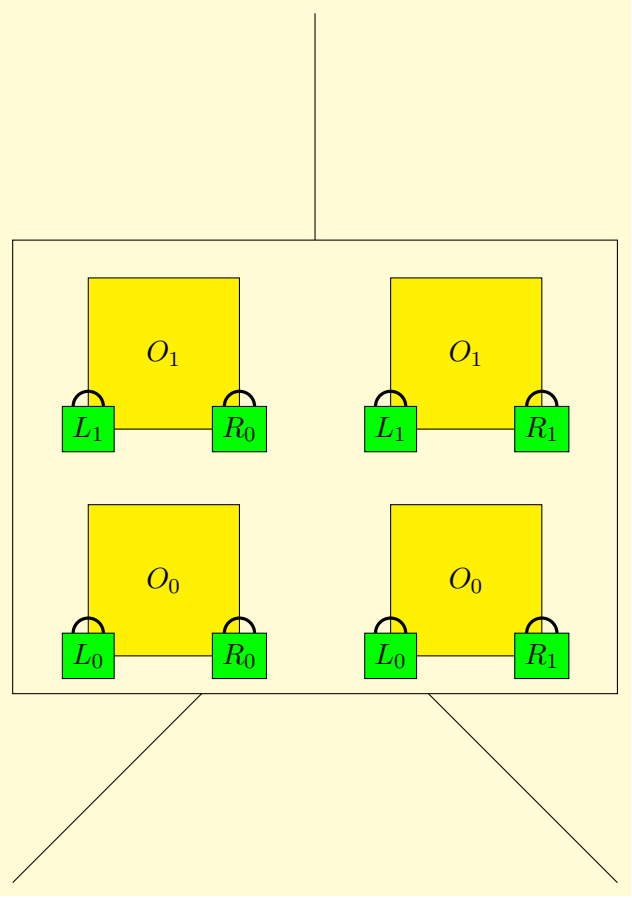

FIGURE 3. Garbled gate that outputs the left input but is indistinguishable from an AND-gate

finish evaluating the garbled circuit and must terminate the process before reaching the end. Thus from observing whether Bob sends her Z or not, Alice will learn b. In the same manner, she can guess several input bits of Bob — if she guesses all of them, the process terminates and Bob is able to send her the final result Z, if she is wrong, then Bob is not able to reach Z.

Alice can also modify some inner gates of the circuit in a similar manner — some boxes of a gate will hold a correct key for the next step, but the other boxes will hold garbage — thus Alice can learn which box did Bob open. These attacks are called the *selective failure*.

3.3. **Cut-n-choose.** The main technique how to guard against Alice encoding the wrong circuit is called *cut-n-choose*. Given the function f, Alice will prepare m circuits $\mathcal{C}_1, \ldots, \mathcal{C}_m$ that are all supposed to encode f. Bob randomly picks some i from $\{1, \ldots, m\}$ Then the circuit $\mathcal{C}_i$ will be left unopened but all the other circuits will be opened. Here opening

means that for a given circuit $\mathcal{C}_j$ that will be opened, Alice gives Bob all the key pairs $\{K_0^i, K_1^i\}_{i\in\{1,\dots,n\}}$. Recall that in the usual case, Bob got only the keys $\{K_{w_i}^i\}_{i\in\{1,\dots,n\}}$—i.e. Bob got only one key from each pair. Since Bob now holds all the key pairs, Bob can open all the boxes and check that everything is as it is supposed to be. If Alice produces one faulty circuit, then the probability that she will not be caught is $\frac{1}{m}$.

Another possibility for cut-n-choose is checking only some v gates and evaluating the remaining $m-v$ gates to determine the answer by setting it to be the the majority of the outputs of those $m-v$ gates, if it exists. In that case, some extra cryptographic machinery is needed to force the parties to use same input for all the $m-v$ circuits that are evaluated. Alice must corrupt at least $\lceil\frac{m-v}{2}\rceil$ circuits to succeed and the chance that Bob will pick none of them for checking is $(\frac{m-\lceil\frac{m-v}{2}\rceil}{m})^v$. When we set v equal to $\frac{m}{2}$, then the probability of success of Alice is about $\left(\frac{3}{4}\right)^{\frac{m}{2}}$.

4. (Mini)LEGO

In 2009, Nielsen and Orlandi proposed a garbling scheme where the *cut-n-choose* procedure is more efficient, which they called LEGO [3]. In 2013, Frederiksen, Jakobsen,Nielsen, Nordholt and Orlandi proposed an improvement to this, which they called MiniLEGO [4]. We will give a short overview of the MiniLEGO procedure, as it is more simple, but will still not describe it on a detailed level, since it is nevertheless rather complicated.

The basic idea of MiniLEGO is performing the *cut-n-choose* on gate level, not on circuit level. This helps to save a factor of $O(\log(s))$ in communication and computation where s is the circuit size — while in the "standard" cut-n-choose methods, such as in [6], a replication factor of $O(k)$ is needed to achieve the security of 2^{-k}, in LEGO and MiniLEGO, only $O(k/\log(s))$ is needed. The only gates used in the MiniLEGO protocol are AND-gates, XOR-gates and NOT-gates. In many applications of garbled circuits, there are ways how to make the evaluation of XOR-gates "free" — i.e. we do not need cryptography to implement them. This is called the free-XOR trick and we will describe later how it is achieved here. It turns out that NOT-gates may be injected while soldering AND-gates together, which means that we only have to worry about the AND-gates.

Let us call the circuit of the function f we evaluate $\mathcal{C}$. The circuit $\mathcal{C}$ will have a set of gates that we will denote with G.

Alice will create more AND-gates than is necessary for one circuit. Bob will choose some gates and some inputs for these gates to check whether they are correct. The unopened gates are grouped into *buckets* of gates and with each gate $j \in G$ we associate a bucket of ρ gates. When evaluating that gate j, we evaluate the gates in the associated bucket and if more that $\lfloor\rho/2\rfloor$ of gates in that buckets agree on some bit, that bit will be considered the output bit of j, otherwise $\perp$ will be considered to be the output of j. This means that Alice must change many gates to change what function the circuit computes and thus has a greater chance of getting caught. We will not discuss here the details how the gates in the buckets can vote on the general value of the bucket in a secure way.

We require that for every gate $L_0 \oplus L_1 = R_0 \oplus R_1 = O_0 \oplus O_1 = \Delta$, where Δ is called the global difference — Δ is the same over all the gates and secret from Bob. When Bob

tests a gate, he will provide a pair of bits (u_i, v_i) for every gate i that he wants to test. Alice will open the corresponding input and output keys for the tested gates — but note that since she only opens one key for every tested wire, Δ is not leaked.

Alice first builds the gates, Bob checks some, then sorts them into buckets, and now *solders* these buckets of gates together to make a circuit. We will say that the output wire O^2 of the gate gg_2 is soldered to the left input wire L^1 of a gate gg_1, when $L_0^1 = O_0^2$ or that the output wire O^3 of the gate gg_3 is soldered to the right input wire R^1 of a gate gg_1, when $R_0^1 = O_0^3$. Note that defining the zero-keys L_0, R_0 and O_0 of a gate defines the whole gate, since Δ is globally defined.

Of course, when the gates are generated independently, then these equalities will not hold at first and we have to do some shifting and modifying these gates when we are building a circuit from the gates to make the zero-keys match (and to make the one-keys match, but that is not very difficult, since they are defined by the zero-gates). We will not go into detail about this. However, note that Alice will generate these zero-keys along with the other keys. She will have to commit to these keys using some homomorphic commitment scheme so that later she can tell Bob the various differences between the zero-keys when the circuit is being built so that the gates can be shifted by the necessary amount.

We shall finally describe how we get the free-XOR trick— when a gate gg_1 has, for example left input zero-key so that $L_0^1 = O_0^2 \oplus O_0^3$, then

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

and hence we do not need to garble XOR-gates, since this is handled by the soldering.

5. Zero-Knowledge via Secure Two-Party Computation

5.1. **Zero-Knowledge.** A zero-knowledge protocol is a protocol, often interactive, where the prover P (sometimes called Peggy) convinces the verifier V (sometimes called Victor) in the correctness of some statement without the verifier learning anything else than the trueness of the statement. Zero-knowledge also denotes the *zero-knowledge property*. More formally, for both parties, we require the existence of some simulator that does not see the secret values of the other party but can not be distinguished from the other party.

To put it figuratively, imagine that I have a twin brother who knows the answer to life, universe and everything and that I don't. Now, when my brother talks to a stranger and the stranger is not sure which one of us he is talking to, then my brother can be sure that his secret is safe — if the stranger could somehow cheat the secret out of him, he would have a way of telling us apart. In this case, I would be a simulator for my twin brother.

In this report, we take the statement to be the truth of some NP problem y. NP problems are yes/no- problems that are easy to verify in the case that the answer is "yes". More specifically, there exists some value or string w, called *witness* such that knowing this value, a polynomial-time program can verify that the answer is indeed "yes". For many problems in NP, there are no good methods to solve them.

The article [2] uses garbled circuits to achieve zero-knowledge. Starting from now on until the end of the report, we will talk about that article.

Let the length of w be n and let the bits of w be $w_1, w_2, \ldots, w_n$.
P sends F_{OT} the message (i, w_i) for every $i \in \{1, \ldots, n\}$
V generates a garbled circuit GC along with the pairs of possible input keys $\{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}$ and the correct result Z.
V sends $\{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}$ to the F_{OT}.
F_{OT} sends P the keys $K_{w_i}^i$ for every $i \in \{1, \ldots, n\}$.
V sends P the garbled circuit GC.
P evaluates the circuit and sends the result Z' to V.
V outputs *accept* iff $Z = Z'$.

FIGURE 4. The passively secure case

We will construct a witness checker using garbled circuits. Denote the witness-checker function by $f$and let $f(w) = 1$ if w is a valid witness and $f(w) = 0$ if not. Here Victor will now play the part of Alice and Peggy the part of Bob. Victor will construct a garbled circuit that computes f. Note that the circuit will just output only one bit and that here we are mainly interested that Victor, the circuit constructor, should learn this bit. As we remember, there must be two possible final outputs — O_0 and O_1. When Peggy has a valid witness and the circuit was honestly built, then she can get O_1 and send it to Victor so when Victor sees O_1 then he knows that Peggy knows a valid witness. We shall refer to O_1 as Z from now on. We stress again that Victor can tell the validity of Peggys witness from just seeing the final key Z— that is enough.

Note that our case differs from regular two-party computation also by that Victor hasn't got any secrets. This means that we do not need *cut-n-choose* at all. When Peggy has finished evaluating the circuit and has gotten the output Z', she will commit to it, after which Victor will open the circuit so that Peggy can check that everything is as it was supposed to be. If everything is fine, Peggy sends Z' to Victor, who outputs *accept* if $Z = Z'$.

5.2. **Passively secure model.** First we will talk about the case where the verifier is semi-honest but prover may be malicious. In order to achieve honest-verifier zero-knowledge, we require that the garbling scheme has the following properties

(1) Correctness: When P evaluates the function on a value x where $f(x) = 1$ then she should be able to produce the correct result Z.
(2) Soundness: No malicious evaluator can guess the secret Z without knowing a x that satisfies $f(x) = 1$.

The protocol for the passively secure case is depicted on Figure 4. Trusted Third Parties were used for several cryptographic primitives in this protocol. In the passive model we will use the trusted third party F_{OT} for oblivious transfer. Note that we assume that the steps of the will be followed in the order as described in Figure 4. This is not difficult to ensure, but will add unnecessary details to the picture and make understanding take more time.

The described protocol is secure in the case of an actively corrupted prover or a passively corrupted verifier. We will give a proof sketch. We will use simulators that do not know the secret information of the other party but have other powers to compensate for the lack of that knowledge. We will show that the corrupted party will not be able to distinguish between the case where they interact with a simulator who does not know the secret and the case when they interact with the other party who knows this secret information — thus we get the zero-knowledge property — the corrupted party will learn nothing about the secrets of the other party.

Suppose that the verifier is passively corrupted. We will use a simulator that will extract Z from the algorithm that generates the garbled circuit, the pairs of keys and Z. Since when P is honest, P will input a w that is a valid witness of y and since our protocol has the correctness property, the result of the real world would have been Z anyway so there is no way to distinguish between the real life and the simulation.

Suppose that the prover is actively corrupted — i.e. she potentially does not input a valid witness and does something with the garbled circuit that she was not supposed to do. Since the verifier does not have any secrets, her only possible goal is convincing the verifier that she knows a valid witness when she does not. Let her invalid witness be w^*. We will use a simulator that will extract w^* from the F_{OT} functionality, will send P the garbled circuit and the necessary keys, and in the end, when receiving Z', will output *accept* if w^* is a valid witness for y, not when $Z = Z'$. The prover cannot distinguish whether she is interacting with the simulator or in the real world due to the Soundness property and thus can not fool the system.

6. Actively secure model

We will now consider the model where both the prover and the verifier can be actively corrupted — i.e. one of them will not follow the protocol. There are some extra procedures that we will require.

6.1. **Verifying algorithm.** We require an extra procedure $Ve(GC, \{K_0^i, K_1^i\}_{i\in\{1,\dots,n\}}, f)$ that essentially checks that the garbled circuit indeed computes the function f. Note that this procedure is pretty easy, we just use a specific notation here to make writing things down easier. One should not take this as a black box that will be later replaced with powerful and efficient cryptography, although there may be some trick to make the checking faster. It is just opening the circuit in the same manner as we described in the *cut-n-choose* and seeing that it does what it is supposed to do.

6.2. **Extractor.** For technical purposes, we need a polynomial-time function Ext that gets as input the garbled circuit GC and the input keys $\{K_0^i, K_1^i\}_{i\in\{1,\dots,n\}}$ and outputs Z. This will be used by the simulator for showing zero-knowledge — the simulator will not know the witness but can still output Z. Note that we have to require that Ext is polynomial-time — if it had more time, for example, an exponential amount of time, then it could solve the problem y by using brute force and could get some valid witness w' which may make some people uncomfortable — the verifier should not learn any witness for y from this process.

Another point is that the number of possible witnesses could be small, so maybe the valid witness that was found using Ext happens to be w, which means that the simulator would, in a way, know w.

Note that the job of the extractor function is not very difficult or mysterious — it has all the keys and knows their indices and can thus open all the boxes and for each gate and also tell which of the four boxes is $C_{0,0}$, which $C_{0,1}$, which $C_{1,0}$ and which $C_{1,1}$. Hence the extractor can also open the last boxes and figure out which of these values correspond to 1 and take that as the Z. This is linear in the number of gates.

6.3. **Requirements and protocol.** In order for the garbling scheme to have the actively-secure zero-knowledge property, we require from it both the Correctness and Soundness properties, as we did in the passive model, but we also want the following property.

Verifiability: It is not possible to construct such a circuit and such input that two valid witnesses x and y would result in different Z'_x and Z'_y and that the procedure Ve would accept that garbled circuits and input. Also, we require the existence of a polynomial functionExt that, given GC and $\{K_0^i, K_1^i\}_{i\in\{1,\dots,n\}}$, can produce Z in polynomial time.

We will also want to use a flavour of the oblivious transfer scheme called Weak Committing Oblivious Transfer, or F_{COT} for short — it preserves all the message pairs $\{K_0^i, K_1^i\}$ sent to it and when it gets the message *open-all*, it will send these message pairs to the other party. Here this F_{COT} is depicted as a trusted third party, but various efficient real-world Oblivious Transfer schemes can rather naturally have this *open-all* property. We will also use F_{COM} as a trusted third party that helps to use commitments. The protocol for the active security case is depicted on Figure 5. Again, we omitted the parts of the protocol that were there to ensure that everything happens in the right order for the sake of simplicity.

In essence, Peggy gets the garbled circuits from Victor and the input keys using Weak Commmitting Oblivious Choice. She evaluates the circuit, if that succeeds, she commits to the value, and lets Victor open everything and show that there was no foul play involved — if there is, then Peggy doesn't show her result to Victor. It might happen that she is not able to evaluate the circuit — this might happen in the case of the selective failure attack. However, she will then commit to $\perp$ and still ask Victor to open everything, and, since then it can be seen from the opened circuit that foul play was involved, Peggy will quit. Victor will not know whether Peggy quit because she couldn't evaluate the protocol or because she saw that his circuit was incorrect, since he has not seen any output of Peggy and that she will always quit when foul play is involved.

Somewhat more formally, if the garbling scheme has the correctness property, the soundness property and the verifiability property, then the protocol described securely implements the zero-knowledge functionality in the presence of an actively corrupted prover or a actively corrupted verifier.

We will give a proof sketch.

The case of the actively corrupted prover is similar to the semi-honest case.

If the verifier is actively corrupted, then the simulator S will act in the following way. S extracts $\{K_0^i, K_1^i\}_{i\in\{1,\dots,n\}}$ from F_{COT} and gets GC from V. Then S uses the Ext function

Let the length of w be n and let the bits of w be $w_1, w_2, \ldots, w_n$.
P sends F_{COT} the message (i, w_i) for every $i \in \{1, \ldots, n\}$.
V generates a garbled circuit GC along with the pairs of possible input keys $\{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}$ and the correct result Z.
V sends $\{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}$ to the F_{COT}.
F_{COT} sends P the keys $K_{w_i}^i$ for every $i \in \{1, \ldots, n\}$.
V sends P the garbled circuit GC.
P evaluates the circuit and calculates Z'. If the evaluation fails, set Z' to $\perp$. Then she commits to Z' by sending Z' to a Trusted Third Party F_{COM}.
F_{COT} gets command *open-all* and sends $\{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}$ to P.
P runs $Ve(GC, \{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}, f_y)$. If the output is *reject*, she aborts the protocol. If it is *accept*, F_{COM} sends Z' to V.
V checks whether $Z = Z'$ and outputs *accept* or *reject*.

FIGURE 5. The actively secure case

to compute Z'. S also checks, whether $Ve(GC, \{K_0^i, K_1^i\}_{i\in\{1,\ldots,n\}}, f_y)$ accepts or rejects. If it accepts, then S sends the Z' that S got from Ext to V, otherwise, S aborts the protocol after F_{COT} gets the message *open-all*. Note that S can not abort earlier since that could potentially leak the information that V is interacting with a simulator, not with Peggy. If V sent such a garbled circuit and inputs that the algorithm Ve rejects, then the protocol will be aborted in both the real world and in the simulation. If V sent such a garbled circuit and inputs that the algorithm Ve accepts, then Ext will output the same result Z' as the prover in the real world would have, thanks to the verifiability property.

7. Results and Conclusion

7.1. **Results of** [2]**.** Authors of [2] ran experiments on a machine that had Intel(R) Core(TM) i7-2600 CPU 3.40GHz and 16GBytes of RAM.The experiments were averaged over 100 runs. They consider this to be a proof-of-concept work and prototypical in nature, however, the results are promising.

They performed a baseline case of Honest-Verifier Zero Knowledge of AES — i.e P proves that she knows a private key k so that $AES_k(x) = c$ where c is a publicly known cipher text and x is a publicly known plaintext in the passively secure case, which took 1404 ± 6 milliseconds. Running the actively secure case of AES took only 1667 ± 6 milliseconds — the cost of going from passive security to active security gives a slowdown that is less than 20% — this is still in the same order of magnitude, while in the case of the best garbling schemes that perform cut-n-choose, such as MiniLEGO, the actively secure case takes at least several times more time than the passively secure case.

Take note that [2] describes a scheme that can use different variants of protocols, e.g. different garbling schemes or commitment schemes as building blocks and thus the efficiency

estimations also depend on the efficient of the building blocks. The concrete garbling scheme used by [2], for example, uses the free-XOR trick.

We can estimate the complexity as the sum of complexity of the garbling scheme in the passively secure case plus the complexity of the commitment scheme plus the complexity of the Weak Committing Oblivious Transfer. The commitment scheme is used only once and we can probably ignore its cost. The cost of the oblivious transfer can be estimated to be approximately linearly dependent on the size of the input. Since the size of the circuit will probably depend on the size of the input superlinearly, we may also ignore the oblivious transfer to give a ballpark estimate on the asymptotic of the protocol. The cost of the garbling scheme in the passively secure case is linearly dependent on the number of gates, and we can use the free-XOR trick. Thus we may estimate the approximate cost of the algorithm with $O(s)$ where s is the number of non-XOR gates in the protocol.

7.2. **Conclusion.** Garbled Circuits offer a way how to perform secure two-party computation. It is fairly straightforward to do in a semi-honest setting, but usually, doing it in the active security model requires more resources. While the asymptotic complexity of some previous solutions such as [6] are $O(s \cdot k)$ where s is the number of non-XOR gates and k is the statistical security parameter, when using asymmetric primitives, the asymptotic complexity for MiniLEGO is $O(s \cdot k/\log(s))$. However, MiniLEGO requires some extra machinery, such as homomorphic commitments and error-correcting codes.

When we want to prove the correctness of some NP-problem, then we do not need much complicated machinery since we can do without the cut-n-choose. The solution by [2] is fairly simple, the only things we need in addition to the baseline passive-security garbled circuits are one use of a commitment scheme and the requirement that the oblivious transfer ought to have the extra *open-all* property that many oblivious transfer schemes already naturally have. As a result, the asymptotic estimate is about $O(s)$— assuming that the building-block protocols used are secure enough. However, this is generally not comparable with more universal garbling schemes since it can be used only in a specific setting.

References

[1] A. C.-C. Yao, Protocols for Secure Computations (Extended Abstract), FOCS 1982: 160-164
[2] M.Jawurek, F.Kerschbaum, C.Orlandi, Zero-Knowledge Using Garbled Circuits: How To Prove Non-Algebraic Statements Efficiently, Cryptology ePrint Archive, Report 2013/073 2013
[3] J.B. Nielsen, C. Orlandi, LEGO for Two-Party Secure Computation, Cryptology ePrint Archive, Report 2008/427, 2008
[4] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt and C. Orlandi, MiniLEGO: Efficient Secure Two-Party Computation From General Assumptions, Cryptology ePrint Archive, Report 2013/155, 2013
[5] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. ACM Conference on Computer and Communications Security, 784796, 2012.
[6] Y.Lindell, B. Pinkas, Secure two-party computation via cut-and-choose oblivious transfer, TCC, 329346, 2011