TinyLEGO: An Interactive Garbling Scheme for Maliciously Secure Two-party Computation

Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti ***

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

Abstract. This paper reports on a number of conceptual and technical contributions to the currently very lively field of two-party computation (2PC) based on garbled circuits. Our main contributions are as follows:

1. We propose a notion of an *interactive garbling scheme*, where the garbled circuit is generated as an interactive protocol between the garbler and the evaluator. The garbled circuit is correct and privacy preserving even if one of the two parties was acting maliciously during garbling. The security notion is game based.
2. We show that an interactive garbling scheme combined with a Universally Composable (UC) secure oblivious transfer protocol can be used in a black-box manner to implement two-party computation (2PC) UC securely against a static and malicious adversary. The protocol abstracts many recent protocols for implementing 2PC from garbled circuits and will allow future designers of interactive garbling schemes to prove security with the simple game based definitions, as opposed to directly proving UC security for each new scheme.
3. We propose a new instantiation of interactive garbling by designing a new protocol in the LEGO family of protocols for efficient garbling against a malicious adversary. The new protocol is based on several new technical contributions and many optimizations, including a highly efficient UC commitment scheme. A theoretical evaluation of the efficiency shows that the instantiation is one to two orders of magnitude faster than the previously most efficient LEGO protocol and that it in general compares favorably to all existing garbling-based 2PC protocols for malicious adversaries.

Keywords: Secure Computation, XOR-Homomorphic Commitments, Garbled Circuits, Interactive Garbling Scheme, Oblivious Transfer, Universal Composability, Standard Assumptions, Large Circuits.

1 Introduction

Secure two-party computation (2PC) is the area of cryptography concerned with two mutually distrusting parties who wish to securely compute an arbitrary function with private output based on their respective private inputs. A bit more formally Alice has the input x, Bob the input y, and they wish to compute the function $z \leftarrow f(x, y)$ without Bob learning anything about y and without Alice learning anything about x.

This area was introduced in 1982 by Andrew Yao [Yao82, Yao86], specifically for the *semi-honest* case, where both parties are assumed to follow the prescribed protocol and only try to compromise security by extracting information from their own views of the protocol execution. Yao showed how to construct a protocol preventing this using a technique referred to as the *garbled circuit approach*. This approach involves having one party (*the constructor*), say Alice, encrypt, or "garble", a Boolean circuit computing the desired functionality. This is done by choosing two random keys for each wire in the circuit, one representing a value of 0 and another representing a value of 1. Each gate in the garbled circuit is then constructed such that Bob

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

** Partially supported by Danish Council for Independent Research via DFF Starting Grant 10-081612. Partially supported by the European Research Commission Starting Grant 279447.

(*the evaluator*), given exactly one key for each input wire, can compute exactly one key for the output wire, namely the key corresponding to the bit that the gate is supposed to output (for example, the logical AND of the two input bits). Alice sends the garbled circuit to Bob and makes sure that for each input wire, Bob only learns one of the wire's keys. For Alice's own input, she simply sends these keys to Bob. Using an oblivious transfer (OT) protocol, Bob also learns one key for each input wire corresponding to his own input, without Alice learning Bob's input. Now, given one key for each input wire, Bob can then evaluate the whole garbled circuit, gate by gate, while at the same time, he cannot learn which bits flow on the wires. When he reaches the output keys he simply sends these back to Alice, who uses some auxiliary information to learn which bits the keys encode. See [LP09] for a thorough description of Yao's approach.

A major reason why Yao's approach is only secure against a semi-honest adversary is that Bob cannot be sure that the garbled circuit he receives from Alice has been garbled correctly. One way to cope with this, and achieve *malicious* security, where a corrupt party might deviate from the prescribed protocol in an arbitrary manner, is with a *cut-and-choose* approach: Instead of sending one circuit, Alice sends several independently garbled versions of the circuit to Bob. Bob then randomly selects some of these, called the *check circuits*, which are then opened, allowing Bob to verify that they do indeed correspond to the correct function f. If this is the case, he knows that the remaining circuits, called *evaluation circuits*, contain a majority of correct circuits, except with negligible probability in the amount of circuits constructed.

However, doing cut-and-choose on several garbled circuits introduces some other issues that have to be dealt with in order to obtain malicious security. One of these issues is the *selective failure attack* [MF06, KS06]: Since Alice will supply Bob with the keys in correspondence with his input bits through an OT, Alice can simply input garbage for one of the keys, e.g, the 0-key for the first bit of his input. If Bob aborts the protocol, because he cannot evaluate a garbled circuit when one of the keys is garbage, Alice will know that the first bit of his input is 0. On the other hand, if he does not abort the protocol then his first bit must be 1.

Solutions to the selective failure attack and several other attacks specific for cut-and-choose of garbled circuits, along with several paths for optimizations has led to a plethora of work on such protocols including, but not limited to, [LP07, PSSW09, LP11, SS11, HEKM11, KSS12, Bra13, FN13, HKE13, Lin13, MR13, SS13, HMSG13, RT13, FJN14, AMPR14].

Still, garbled circuits can be used without the cut-and-choose paradigm in order to achieve full malicious security. For example by using it as a sub-protocol in the approach of "MPC-in-the-head" [IKOS07, IPS08, LOP11] where the parties run a semi-honestly secure two-party protocol, but within this they emulate at least three parties that run a virtual second protocol, secure against a malicious minority. This results in a final protocol maliciously secure against a dishonest majority.

If one is willing to accept a weaker kind of "malicious" security, then it is possible to achieve significant improvements by using the "dual execution" approach. With this approach each party *both* constructs *and* evaluates a single garbled circuit [MF06, HKE12, MR13].

LEGO Considering a garbled circuit as a modular construction, consisting of many connected garbled gates, has led to a new approach to cut-and-choose called *LEGO*. In this approach, cut-and-choose is not done on several circuits, but rather on individual and independent garbled gates. The idea is that if none of the garbled gates that are checked are incorrect, then, with overwhelming probability, at most a few of the remaining garbled gates are maliciously constructed. The remaining gates are then shuffled and *soldered* into fault tolerant *buckets* computing a specific Boolean functionality, such as AND. The fault tolerance comes as the buckets are constructed to output the majority of the output of its individual gates. Thus, since only a few malicious gates remain after the cut-and-choose step, the probability that a majority of these are combined in the same bucket is overwhelmingly small, even for buckets consisting of only a few garbled gates. These buckets can then be soldered together to form an entire garbled circuit which will compute the correct output with overwhelming probability. This "gate-level" approach to cut-and-choose makes it possible to achieve an asymptotic increase in efficiency of the logarithm of the size of the circuit to compute, compared to the protocols based on cut-and-choose of whole garbled circuits.

The LEGO approach was introduced by Nielsen and Orlandi in [NO09]. In that paper the soldering of garbled gates was based on homomorphic commitments, making it possible to obliviously "transform" the key on one wire to the key with similar semantics (whether it represents the bit 0 or 1) on another wire.

Specifically the additively homomorphic Pedersen commitments were used. Unfortunately, these commitments require heavy computational operations in the form of exponentiations of elements in a group. Furthermore, as the key commitments worked on group elements this also required the keys of the garbled gates to be group elements under certain constraints. Unfortunately, this ruined the possibility to use several optimizations of garbled gates which requires the keys to be random bitstrings.

In [FJN+13] the authors introduced an XOR-homomorphic commitment scheme based on OT and error correcting codes. Using this scheme they constructed a new LEGO protocol, called MiniLEGO, which eliminated the need of group exponentiations for each commitment. The usage of XOR-homomorphic commitments (XHC) on bit-strings also eliminated all the constraints previously needed on the gate keys, and thus their protocol works with most gate garbling optimizations. Unfortunately, the error correcting code used to construct the XHC introduced a rather large concrete increase in the communication complexity of each garbled gate. So while MiniLEGO asymptotically performs better than non-LEGO protocols, for practical parameters and circuit sizes the protocol is not competitive.

Finally, it should be noted that recent results [LR14, HKK+14] combine the idea of cut-and-choose of garbled circuits and the LEGO approach to achieve protocols asymptotically more efficient in the amortized (batched) setting than any protocol based on cut-and-choose of garbled circuits. Ignoring the details, their idea is to construct many garbled circuits computing the same functionality, do cut-and-choose to check some fraction of these, and then put the remaining circuits into slots (buckets using LEGO lingo). When the parties wish to do a secure computation it then suffices to use a single slot of circuits. We stress that these protocols only apply to the batched setting, where one is interested in computing the same function many times. In this work we look at the single function evaluation setting, which is more general than the batched setting.

1.1 Contributions

We introduce a new maliciously and static secure 2PC protocol based on the LEGO paradigm, which is more efficient than the MiniLEGO protocol of [FJN+13]. Our contribution includes:

Framework for Interactive Garbling We start by introducing a generic framework for *interactive garbling schemes* in Section 3. This work is much inspired by the work of [BHR12]. However, here we give definitions in an interactive setting. Next, we show that our notion of an interactive garbling scheme suffices for UC-secure 2PC in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

Improvement of the additive commitment scheme of [CDD+14] In Section 4.3 we introduce a modified version of the "basic" non-homomorphic commitment scheme of [CDD+14]. First of all, our scheme directly improves on the communication complexity of each commitment by the length of the message, and furthermore add support for XOR-homomorphic operations on the commitments. Secondly, we propose an amortized opening mechanism that significantly reduces communication complexity and has computational complexity independent of the number of openings sent. Finally, we require the minimum distance of the underlying error correctioncode used by the scheme to be $s+4$ for security 2^{-s} as opposed to $2s+1$ in [CDD+14]. The two former optimizations along with XOR-homomorphism are achieved using a linear combination check on all the messages opened to instead of opening to all messages individually. We believe these optimizations have independent interest in any setting where many commitments are required (additively homomorphic or not), *e.g.* in [AHMR14].

Advancement over previous protocols We then show how to implement an interactive garbling scheme using the before-mentioned modified commitment scheme. Using a mixture of the techniques used in [NO09] and [FJN+13] to create buckets of garbled gates, we reduce the communication complexity (dependent on the circuit size) of one to two orders of magnitude compared to [FJN+13]. One of our optimizations include only requiring a single "correct" gate in each bucket (as opposed to a majority), which is made possible using what we call *wire authenticators*.[1] We also remove the need of a random

[1] For the readers familiar with the LEGO protocol in [NO09], these are very similar to the *key check* gadgets. However, our wire authenticators are a bit simpler and we only require about half the amount of them, compared to the key check gadgets.

oracle, and instead rely on the weaker assumption of a circular-correlation robust hash function. Finally, we compare favorably to current state-of-the-art protocols for realistic circuit sizes. For example, for a circuit with at least 953,021 AND gates and 40-bit statistical security we reduce communication complexity with a factor of 0.69. Our protocol is described in Section 5 and the performance comparisons are presented in Section 6.

2 Preliminaries and Notation

We assume Alice is the party constructing the garbled gates and call her the *constructor*. Likewise, we assume Bob is the party evaluating the garbled gates and call him the *evaluator*. Furthermore, we say that the functionality they wish to compute is $z \leftarrow f(x, y)$, where Alice gives input x, Bob gives input y and only one party, Alice, receives the output z. We denote the bit-length of x as $|x| = n_{\mathsf{A}}$, the bit-length of y as $|y| = n_{\mathsf{B}}$ and let $n = n_{\mathsf{A}} + n_{\mathsf{B}}$. We will denote the bit-length of the output z as $|z| = m$. We denote the Boolean circuit computing the functionality $f(\cdot, \cdot)$ by C and assume without loss of generality (w.l.o.g.) that it is composed of NOT, XOR and AND gates. The XOR gates are allowed to have unlimited fan-in, while the AND gates are restricted to fan-in 2, and NOT gates have fan-in 1. All gates are allowed to have unlimited fan-out. Furthermore, we assume that the first n_{A} input wires are for Alice's input and the following n_{B} input wires are for Bob's input.

We define the *semantic* value of a wire-key of a garbled gate to be the bit it represents. We will use K_j^b to denote the j'th wire key representing bit b. Sometimes, when the context allows it, we will let $L_g^{b_l}$, $R_g^{b_r}$, and $O_g^{b_o}$ denote the left, right, and output key respectively for garbled gate g representing the bits b_l, b_r and b_o respectively. When the bit represented by a key is unknown we simply omit the superscript, e.g. K_j or L_g.

In this work circuits are handled in a similar fashion as to [FJN+13], but we adopt the notation of [BHR12] with some minor syntactic modifications which makes it possible to handle NOT and XOR gates implicitly. Thus, a circuit is a 5-tuple $C = (n, m, q, \mathsf{lp}, \mathsf{rp})$ where $n \geq 2$ is the number of inputs, $m \geq 1$ is the number of outputs and q is the number of AND gates. Thus $w = n + q$ is the total number of wires in the circuit. We let $\mathsf{Wires} = \{1, \ldots, w\}$, $\mathsf{Inputs} = \{1, \ldots, n\}$, $\mathsf{Gates} = \{n + 1, \ldots, w\}$ and $\mathsf{Outputs} = \{w - m + 1, \ldots, w\}$. The maps $\mathsf{lp}, \mathsf{rp} : \mathsf{Gates} \rightarrow \{\{\mathsf{Wires} \setminus \mathsf{Outputs}\} \cup \{\mathbb{1}\}\}^*$ define the topology of the circuit, mapping from gates to their respective left and right input wire. We also require that for all $g \in \mathsf{Gates}$ and $\forall l \in \mathsf{lp}(g) \forall r \in \mathsf{rp}(g)$ it holds that $l \leq r < g$. We say that the set $\mathsf{lp}(g)$ (resp. $\mathsf{rp}(g)$) is the left (right) parents of gate g and we let the left (right) input bit of gate g be $\bigoplus_{j \in \mathsf{lp}(g)} K_j^b$. In this way all XOR gates of C are implicitly defined by lp, rp. The special symbol $\mathbb{1}$ denotes an "implicit" key with semantic values 1. It is used in order to support NOT gates, by the simple observation that a NOT gate is logically equivalent to an XOR where one of the inputs is the constant 1.

We use k to denote the computational security parameter and s to represent the statistical security parameter. Technically, this means that for any fixed s and any polynomial time bounded adversary, the advantage of the adversary is $2^{-s} + \mathsf{negl}(k)$ for a negligible function negl. I.e., the advantage of any adversary goes to 2^{-s} faster than any inverse polynomial in the computational security parameter. If $s = \Omega(k)$ then the advantage is negligible.

We will use as shorthand $[n] = \{1, 2, \ldots, n\}$ and $e \in_R S$ to mean; sample an element e uniformly at random from the set S. We write $y \leftarrow P(x)$ to mean; perform the (potentially randomized) procedure P on input x and store the output in variable y. We sometimes (when the semantic meaning is clear) use subscript to denote an index of a vector, i.e. x_i denotes the i'th bits of a vector x. An overview of the various variables and parameters along with their meaning is given in Appendix I.

3 A Generic Framework for 2PC from Interactive Garbling

We introduce the notion of an interactive garbling scheme and show that it allows to implement UC secure 2PC in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. Our notion extends the notion of a garbling scheme from [BHR12] to allow the garbling algorithm to be a two-party protocol. We assume that the reader is familiar with the syntax,

notational conventions and security notions from [BHR12]. In terms of [BHR12] the below definitions are for the side information function $\Phi_{\mathsf{circ}}(f) = f$, i.e., the evaluator is allowed to learn f, and we only formalize obliviousness, not privacy, as we in our protocol will return the garbled output to the circuit constructor. It is easy to generalize the notions to general side-information functions and to capture privacy.

Syntax An *interactive garbling scheme* consists of a seven-tuple $\mathcal{G}_\pi = (\mathsf{Gb}_\pi, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}, \mathsf{Op}, \mathsf{Ve})$. The first component, called the *garbling protocol*, is a two party protocol. The remaining components are deterministic algorithms. All components are poly-time. The evaluation function ev takes two inputs, a function description f and an input x for f. A string f, the *original function*, by definition describes a function $\mathsf{ev}(f, \cdot) : \{0,1\}^n \to \{0,1\}^m$, which is the function we want to garble. We will often not distinguish between the description of the function and the function, i.e., we write $f(x)$ to mean $\mathsf{ev}(f, x)$. We assume that the input length $f.n$ and the output length $f.m$ can be computed in linear time from f. The *garbling protocol* Gb_π is executed two parties, the *constructor* C and the *evaluator* E. To be concrete, we assume it is a protocol in the UC framework. We assume that the parties send no messages directly to each other, instead all communication is through ideal functionalities. This is without loss of generality, as we can always introduce an ideal functionality for communication. The input to both parties is $\left(1^k, f\right)$, where $k \in \mathbb{N}$ is the security parameter and f is a *function description*. The output to C is (F, e, d, o), where F is the *garbled function*, e is the *input encoding function*, d is the *output decoding function*, and o is the *opening function*. The output to E is a garbled function $F \in \{0,1\}^*$ and a *verification function* v. The *input encoding algorithm* takes as the input encoding function e and an input $x \in \{0,1\}^n$ and outputs an *encoded input* $X \leftarrow \mathsf{En}(e, x)$. The garbled function F maps each encoded input X to an encoded output $Z \leftarrow \mathsf{Ev}(F, X)$. The *output decoding algorithm* maps a garbled output Z to the final output $z \leftarrow \mathsf{De}(d, Z)$.

The algorithms Op and Ve are extra compared to [BHR12]. They are only needed if the scheme has input opening, as we need in this work. The *input opening algorithm* Op takes as input o, a bitstring x and a subset $I \subseteq [n]$. It returns the *opening information* O. The *input verification algorithm* Ve takes as input v, garbled input X, opening information O and a set $((i, x_i))_{i \in I}$ for some set $I \subseteq [n]$ where all x_i being bits. It outputs true ($\top$) or false ($\bot$). Intuitively, it uses v to judge whether X has been used in an encoding consistent with opening information O where bit number i is x_i for $i \in I$.

Below we informally describe the defining security properties of our notion of an interactive garbling. For a formal treatment of these security properties we direct the reader to Appendix C, in particular Figure 16 and Figure 17.

Correctness We define *correctness* (property name: corr) as in [BHR12], except that now the material is generated interactively. We also add the requirement that the opening and verification algorithms must be correct.

Obliviousness We define *obliviousness* (property name: obl.ind.act) as in [BHR12], i.e., by seeing a garbling and an encoding of one of two inputs x_0 and x_1, the evaluator cannot guess which input was used. As an addition we let the adversary $\mathcal{B}$ play the role of E in the garbling protocol. We allow $\mathcal{B}$ to deviate from the garbling protocol. This gives a notion of malicious security. One can define a relaxed notion by requiring that $\mathcal{B}$ only gets to see the randomness of E, but we do not need this notion in this work. We also consider only one function, as $\Phi_{\mathsf{circ}}(f_0) = \Phi_{\mathsf{circ}}(f_1)$ implies that $f_0 = f_1$. We extend the notion to require obliviousness to hold also in the presence of the opening information. This can only be the case, of course, if the two challenge inputs agree on the opened positions.

Authenticity We define authenticity as in [BHR12], to mean that the evaluator given a garbling and an encoded input can compute the unique output encoding that will be accepted by the constructor, except that we again let the adversary participate maliciously in the garbling protocol. In [BHR12] it was sufficient to require that only the unique correct garbled output can be returned. However, when the scheme is interactive and the adversary participates in the garbling protocol, we also need to require that the generated circuit is correct, as it does not make sense to reason about the correct garbled output, if the output itself is not correct. This means authenticity is extended to include also robustness of the garbling protocol against a corrupted evaluator.

Knowledge of F We also need that even a cheating evaluator knows F (property name: knof). The reason why we need this is that knowing F means that the evaluator can compute and hence knows the correct $Z \leftarrow \mathsf{Ev}(F, X)$. This in turn means that if the scheme has authenticity, then the evaluator knows that all $Z' \neq Z$ will be rejected by the constructor and $Z' = Z$ will be accepted. Hence, whether or not the constructor rejects a given Z' cannot be used to leak any information on the input of the constructor.

Robustness against the Constructor We also define a notion of *correctness against the constructor* (property name: rob.con). We ask that even if C is malicious in the garbling protocol, the produced material computes correctly. To define this we need that the constructor knows an explicit input encoding function and an explicit output decoding function.

Uniqueness of Output Encoding We also need to require that all outputs have a unique encoding, even if the constructor is cheating during Gb_π. We call this *uniqueness of output encoding* (property name: unqoe). The reason for this requirement is that if there were several alternative encodings, then the particular encoding of the output might conceivably be used to signal information extra to the output that is encoded.

Uniqueness of Input Encoding Similarly, we require that there are unique input encodings (property name: unqie) even when the constructor is cheating. We only require uniqueness for encodings which make the opening check and the garbled evaluation succeed.

Token Commitment We finally need a notion of *token commitment* (property name: tok.com). It essentially just says that the opening and verification algorithms are correct even if the constructor is cheating, i.e., if a token for an opened position is claimed to be a token for the bit b, then this is indeed the case.

Projective Schemes We say that e is projective if e is of the form $\left(X_1^0, X_1^1, , \ldots, X_n^0, X_n^1\right)$ and o is of the form $\left(O_1^0, O_1^1, \ldots, O_n^0, O_n^1\right)$. We call X_j^b the *b-token for input wire j* and O_j^b the *opening information* for the b-token for input wire j. Each token is from $\{0,1\}^\kappa$, where κ is called the *token length*. For a scheme to be called *projective* we further require that En, Op, Ve and En^{-1} are *the projective encoding algorithm*, *the projective opening algorithm*, *the projective verification algorithm*, and *the projective de-encoder*, as defined now: If e is such that $X_i^0 = X_i^1$ or $X_i^b \notin \{0,1\}^\kappa$ for any i or b, then $\mathsf{En}(e, x) = \bot$. Otherwise on input $x \in \{0,1\}^n$ and e as above $(X_1^{x_1}, \ldots, X_n^{x_n}) \leftarrow \mathsf{En}(e, x)$ and $\{(i, O_i^{x_i})\}_{i \in I} \leftarrow \mathsf{Op}(o, x, I)$. As for verification, we require that the verification algorithm verifies the openings individually, i.e., there exist a poly-time algorithm Ve taking as input v and a single X_i, a single O_i, an index i and a bit x_i and outputs true or false such that $\mathsf{Ve}\left(v, (X_1, \ldots, X_n), \{(i, O_i)\}_{i \in I}, \{(i, x_i)\}_{i \in I}\right) = \bigwedge_{i \in I} \mathsf{Ve}\left(v, X_i, O_i, i, x_i\right)$. We describe the projective de-encoder En^{-1}. If it is not possible to parse e on the form $\left(X_1^0, X_1^1, \ldots, X_n^0, X_n^1\right)$, where $X_i^b \in \{0,1\}^\kappa$ and $X_i^0 \neq X_i^1$ for all i and b, then output $\mathsf{En}^{-1}(e, X) = \bot$. If it is not possible to parse X on the form $(X_1, \ldots, X_n)$, where $X_i \in \{0,1\}^\kappa$ and $X_i \in \left\{X_i^0, X_i^1\right\}$ for all i, then also output $\bot$. Otherwise, let x be the unique $x \in \{0,1\}^n$ such that $X \leftarrow \mathsf{En}(e, x)$ and let $x \leftarrow \mathsf{En}^{-1}(e, X)$. If a scheme is of the above form we say that it has the property proj.

3.1 Interactive Garbling with Constructor getting the Output

We now describe our generic protocol π_{IGCO}, see Figure 1. It abstracts the protocols in [NO09] and [FJN+13] where it is the constructor which gets the output, and also captures our new interactive garbling scheme. This means that the evaluator never sees decoding information, and hence only obliviousness is needed, as opposed to privacy, to get privacy against a corrupted evaluator. The privacy against a corrupt constructor follows from robustness and uniqueness of output encoding, which guarantees that the encoded output returned by the evaluator does not leak anything but the output.

Our protocol uses a notion of OT which we call *delayed OT*. It is a one-out-of-two OT of κ'-bit strings where $\kappa' = \kappa + o$, where o is the size of an opening and it runs in two phases, as follows: First the receiver inputs a choice bit c. In response to this the sender receives the string chosen. If later the sender inputs

Setup: We denote the two parties of the protocol by A and B. The parties agree on k and f. The parties also agree on n_{A} and n_{B} such that $f.n = n_{\mathsf{A}} + n_{\mathsf{B}}$. We assume that the parties have access to n_{B} copies of the ideal functionality for delayed OT for κ-bit strings. We denote them by $\mathcal{F}_{\mathsf{DOT}}^1, \ldots, \mathcal{F}_{\mathsf{DOT}}^{n_{\mathsf{B}}}$. It is B that inputs the selection bits.

Input B, I: Denote the input of B by $y \in \{0,1\}^{n_{\mathsf{B}}}$. For $i = 1, \ldots, n_{\mathsf{B}}$, B inputs y_i to $\mathcal{F}_{\mathsf{DOT}}^i$ and A waits for output chosen from $\mathcal{F}_{\mathsf{DOT}}^i$.

Garbling: Run Gb_π with A playing C and B playing E. Each party inputs $\left(1^k, f\right)$. The output to A is (F, e, d, o) and the output to B is (F, v).

Input A: Denote the input of A by $x \in \{0,1\}^{n_{\mathsf{A}}}$. A parses e as $\left(X_1^0, X_1^1, \ldots, X_{n_{\mathsf{A}}}^0, X_{n_{\mathsf{A}}}^1, Y_1^0, Y_1^1, \ldots, Y_{n_{\mathsf{B}}}^0, Y_{n_{\mathsf{B}}}^1\right)$ and parse o as $\left(\cdot, \ldots, \cdot, O_1^0, O_1^1, \ldots, O_{n_{\mathsf{B}}}^0, O_{n_{\mathsf{B}}}^1\right)$. Then A sends $\left(X_1^{x_1}, \ldots, X_{n_{\mathsf{A}}}^{x_{n_{\mathsf{A}}}}\right)$ to B.

Input B, II: For $i = 1, \ldots, n_{\mathsf{B}}$, A inputs $\left(\left(Y_i^0, O_i^0\right), \left(Y_i^1, O_i^1\right)\right)$ to $\mathcal{F}_{\mathsf{DOT}}^i$ and B waits for output $(Y_i^{y_i}, O_i^{y_i})$ from $\mathcal{F}_{\mathsf{DOT}}^i$. Then B lets $X = \left(X_1^{x_1}, \ldots, X_{n_{\mathsf{A}}}^{x_{n_{\mathsf{A}}}}, Y_1^{y_1}, \ldots, Y_{n_{\mathsf{B}}}^{y_{n_{\mathsf{B}}}}\right)$ and $O = \left(O_1^{y_1}, \ldots, O_{n_{\mathsf{B}}}^{y_{n_{\mathsf{B}}}}\right)$. If $\mathsf{Ve}\left(v, X, O, \{(i, y_i)\}_{i=n_{\mathsf{A}}+1}^{n_{\mathsf{A}}+n_{\mathsf{B}}}\right) = \bot$, then B outputs abort and terminates.

Evaluation: B computes $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \bot$, then B outputs abort and terminates.

Output: B sends Z to A. If $\mathsf{De}(d, Z) = \bot$, then A outputs abort. Otherwise, A outputs $z \leftarrow \mathsf{De}(d, Z)$.

Fig. 1. Generic Protocol π_{IGCO} for Interactive Garbling with Constructor Output.

$(m_0, m_1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$, then the receiver receives m_c. Delayed OT can be based on normal OT by first transferring uniformly random pads and then later use these to one-time-pad the messages to be transferred.

The ideal functionality $\mathcal{F}_{\mathsf{SFE}}^f$ which our protocol realizes is given in Figure 21 in Appendix F. It is the standard functionality for 2PC except that it only gives output to A. However, if one wish that B should receive the output then this can be done using the standard technique of [LP07]. The ideal functionality has also been designed as to not prevent A from mounting a selective failure attack, which is needed to achieve a full malicious secure protocol – A can make a guess at some input bits of B and if she guesses correct, then she will be told, and the attack goes unnoticed. If she guesses incorrect, B is informed of the attack. This reflects that garbling allows such attacks if not dealt with explicitly. However, such selective attacks can be mitigated easily and efficiently using off-the-shelf constructions, such as the ones in [LP07, SS13]. This is done by a small extension of the function to compute. Which technique is best depends on context, so we consider it cleaner to not make a choice and instead analyze the protocol allowing selective errors.

In Theorem 1 we show that π_{IGCO} UC securely realizes $\mathcal{F}_{\mathsf{SFE}}^f$ against static and malicious corruption of any number of parties. The notion of an extended interactive garbling scheme is formally defined in Appendix C.

Theorem 1. *If $\mathcal{G}_\pi$ is an extended interactive garbling scheme and has the properties proj, corr, obl.ind.act, aut.act, knof, rob.con, unqoe, unqie, and tok.com, then π_{IGCO} UC securely realizes $\mathcal{F}_{\mathsf{SFE}}^f$ against any static and malicious corruption of any number of parties.*

The case of no corruptions follows easily using the properties proj, corr and obl.ind.act, using a subset of the proof arguments below. If both parties are corrupted, there is nothing to show.

If A is corrupted, the simulator uses uniqueness of input encoding to extract the input of A from the tokens sent and inputs it to the ideal functionality. In the protocol the token commitment property is used by B to ensure that a cheating A inputs the right tokens to the OTs used by B to pick his input. However, for each input wire of B where A inputs tokens (X^0, X^1) to the OT, B only gets to see X^b where b is his input bit for that wire. Therefore he can only check whether X^b is correct. Hence, in the simulation, the simulator should also abort iff X^b is incorrect, but b is not known to the simulator. Therefore it proceeds as follows. It extracts X^0 and X^1 from the OT (simulated by the simulator). If they are both incorrect, then clearly X^b is incorrect, and it aborts. If they are both correct, then it does not abort. If there is exactly one correct, X^c say, then the simulator makes a guess to the ideal functionality that this input bit of B is $b = c$. This will abort exactly when $b \neq c$, so if the ideal functionality aborts, the simulation is perfect. Assume then that all the needed guesses are correct. In that case, the ideal functionality returns the output z. From robustness against the constructor and uniqueness of output encoding, the simulator of corrupted A can then from z simulate the output encoding Z that the real world B would send to A and send this Z in the simulation too.

If B is corrupted, then the simulator extracts his input from the choice bits in the OT and gives it to the ideal functionality. Then it runs the protocol with a dummy input for A which is undetectable by B by obliviousness. Authenticity and knowledge of F guarantees that B, when he returns an output encoding to the constructor, will know in advance if it is accepted or rejected. This prevents the abort probability from depending on A's input. We prove the theorem in formally in Appendix C.

4 Building Blocks

Now that we've seen that an interactive garbling scheme (as defined in Section 3) is sufficient for UC secure 2PC we turn our attention to instantiating such a scheme. In this section we will introduce the building blocks we will use to accomplish this. We start by briefly describing how gate garbling is done in our scheme. Afterwards we introduce the concept of a wire authenticator which will be used together with the garbled gates to build our buckets. Finally, we describe our optimized XOR-homomorphic commitment scheme which is based on [CDD+14].

4.1 Garbling

Both MiniLEGO and our protocol make use of an optimized garbling scheme. To be concrete we use a modified version of the garbling scheme of [ZRE14], whereas MiniLEGO used a modified version of the scheme of [PSSW09] (although MiniLEGO would also work fine with the scheme of [ZRE14]). We call our modified version of [ZRE14] $\mathsf{HGarb} = (\mathsf{HGb}, \mathsf{HEn}, \mathsf{HDe}, \mathsf{HEv}, \mathsf{Hev})$. The modifications are merely syntactical and are necessary as the original version samples its own keys (it is not given as an input to the function) and takes an entire circuit description as input. In our setting a deterministic scheme is needed that garbles and evaluates individual gates, take keys as input and supports soldering of gates.[2]

In short the support for soldering means that we need to add some auxiliary information to each garbled gate. This auxiliary information is simply the values with which each of its wire keys have been shifted and is therefore just stored as auxiliary values in a garbled gate. As an example, if one learns the key K_{id} and has previously learned the shifting value $S_{id} = K_{id} \oplus K_{id'}$, then clearly one can compute $K_{id'} = S_{id} \oplus K_{id}$.

The scheme of [ZRE14] implements the *free-XOR* optimization [KS08] and *permutation bits* [Rog91] and uses either a circular-correlation-robust hash function [CKKZ12] or a Davies-Meyer type construction in the ideal random permutation model [BHKR13] for security. We describe the full details of our modified version HGarb in Figure 4 in Appendix A and direct the reader to Figure 2 of [ZRE14] for a presentation of the original scheme.

Gate Garbling Recall that using the free-XOR optimization, a global offset Δ is selected such that $K_j^{b_j} = K_j^0 \oplus (b_j \cdot \Delta)$ for all j. Our gate garbling scheme consists of the three methods $\mathsf{HGb}\left(L^0, R^0, \Delta, id\right) \to \left(G_{id}, O^0\right)$, $\mathsf{HEv}\left(G_{id}, L, R\right) \to O$, and $\mathsf{SGa}\left(G_{id}, L, R, O\right) \to \widetilde{G}_{id}$ where the last method is used to support the notion of "shifting" keys. Within a garbled gate we associate an initially 0^k-string with each wire. This string can then be updated through calls to SGa in order to shift the value of the 0-key on each wire of the gate. This is needed in the soldering phase in order to make it possible to shift the independently generated wire keys.

The HGb method is used to construct garbled AND gates, denoted (G_{id}, O^0). It does so using a 0-key for the left and right wires (L^0, R^0), the global difference Δ along with a unique gate id. Besides the gate's id and the ciphertexts, the method also adds three 0^k-strings to the garbled gate which are placeholders for the shifting information used if/when a gate is to be shifted.

The HEv method is used to evaluate such an AND gate G_{id} by taking a left- and right-key, L, R and outputting the output-key O. It does so by "shifting" the keys in case the wires of the gate have been soldered to other wires. Specifically by computing the XOR of the keys with the auxiliary information stored in the gate, and thus "XOR-out" the keys of the wires it has been soldered to.

[2] For the uninitiated reader see Appendix G for a quick recap of MiniLEGO, the free-XOR optimization and the concept of solderings.

4.2 Wire Authenticators

To increase performance we suggest a refinement in the way buckets of garbled gates are created. The idea is to solder together roughly half the number of gates that MiniLEGO required, while also soldering onto the buckets a number of authenticated wires. The performance gain comes from authenticated wires being less costly to produce and solder onto the buckets than gates are.

A wire authenticator is a gadget that takes as input a value and outputs accept if this value is a specific key (either a 0- or a 1-key) associated with the authenticator, otherwise it outputs reject. This means that once a key is floating on the wire an authenticator associated with this wire will either accept or reject this key. We instantiate our wire authenticators by adding the hash digests of both the 0- and 1-key for each wire, in random order. The crux is then that these are constructed in the beginning of the protocol, before any cut-and-choose steps or bucketing occurs, and thus makes it possible for Bob to discover (with high probability) if a given gate outputs a non-expected key.

For convenience we introduce three macro methods for both authentication, shifting and verifying wire keys in Figure 5 in Appendix A. In short this is the methods $\mathsf{Auth}\left(K^0_{id}, \Delta, id\right) \rightarrow H_{id}$, $\mathsf{Ver}\left(H_{id}, K_{id}, id\right) \rightarrow \top/\bot$ and $\mathsf{SAuth}\left(H_{id}, S\right) \rightarrow \widetilde{H}_{id}$. The first method constructs a piece of information H_{id}, which can be used by the second method to verify that a candidate K_{id} is either K^0_{id} or $K^0_{id} \oplus \Delta$. More specifically this reflects that the first method constructs an authenticator on the two possible keys on a given wire. The second method then uses the authenticators to verify that a candidate key is in fact one of the keys authenticated to, but does not leak whether it is the 0- or 1-key. The third method is used as short-hand for soldering authenticated wires onto regular ones. This works exactly the same as the SGa method does for gates.

4.3 ECC Commitments

The last of our building blocks is our method for constructing commitments to all keys. As previously mentioned these commitments need to support XOR-homomorphic operations for the soldering of wires to work. Our construction takes its inspiration from the "basic" scheme of [CDD+14], but with a number of tweaks and optimizations. Also, the basic construction of [CDD+14] does not support XOR-homomorphic commitments in itself if the committer is malicious. In order to fix this we employ a linear combination check (LCC from now on) after Alice has committed to all wire keys. This has the effect that whatever cheating strategy Alice might follow, after the check she is forced to open consistent values, except with probability 2^{-s}.

We use the term ECC commitments because at the heart of the construction lies a $[\Gamma, k, d]$ binary linear error correcting code $\mathcal{C}$ (ECC from now on). The basic scheme of [CDD+14] works by the committer first encoding her message using the ECC and then produces an additive sharing of each entry of this codeword. When committing, she sends the shares for each entry XOR-padded with the output of a PRG evaluated on a random secret seed. The protocol is set up such that the receiver only holds one of each pair of seeds for each entry (using an $\mathcal{F}_{\mathsf{OT}}$ functionality). Once it is time to open the commitment, the committer simply sends both shares of each entry to the receiver who checks that they match the shares he learned in the commitment phase. The usage of seeds expanded by a PRG makes it possible to suffice with only Γ one-out-of-two OTs of uniformly random elements in $\{0,1\}^k$, in order to construct $\mathrm{poly}(k)$ commitments.

Intuitively, the scheme is binding because the committer does not know which seeds the receiver holds, thus she is unaware of which shares he holds after committing to the messages. Since the code has minimum distance d she has to guess which seeds he holds, for at least d entries, in order to introduce an inconsistency without getting caught. Therefore she will get caught with probability $(1 - 2^{-d})$, if she tries to open to a different message than committed to. The hiding property of the scheme follows from the fact that the receiver only knows one share of each entry of the codeword, until the commitment is opened. In [CDD+14], for statistical security s, they require a code with minimum distance $2s + 1$, because the simulator needs to be able to decode any commitment at commitment time. We improve on this restriction using the aforementioned LCC. Here we achieve security s using a code with only minimum distance $s + 4$ which significantly improves the communication complexity as the required codeword length of all commitments is reduced. The need for the extra 4 in the distance is explained in detail in Appendix E.

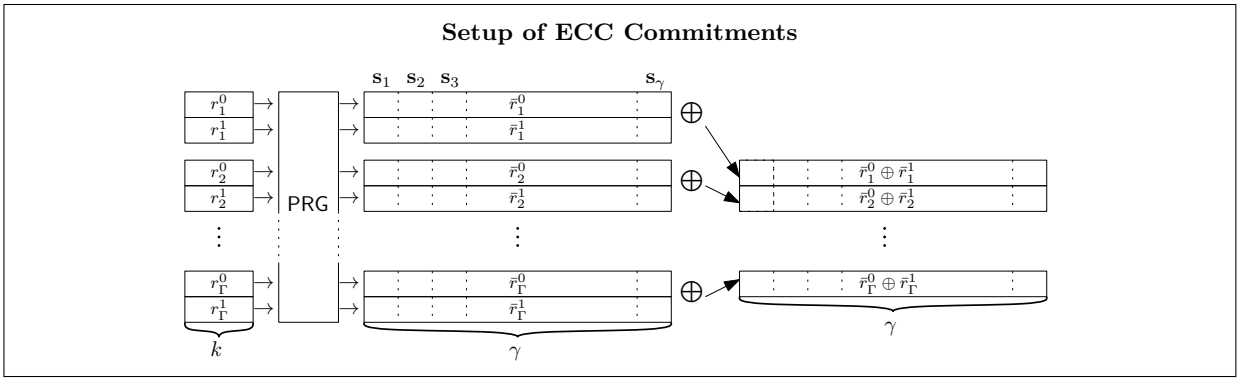

Fig. 2. Illustration of the setup phase of the "basic construction" of [CDD+14] for committing to γ messages.

Intuitively, our LCC works by challenging the committer on many random linear combinations of the commitments, so if she has introduced any errors she will get challenged on at least one of these errors with high probability. To furthermore improve on performance we also use the LCC for amortizing the openings of many commitments at the same time. The motivation for this is that the openings are much larger than the underlying messages (for standard parameters 4-7x larger), so to increase the efficiency of our scheme we let the committer send the required messages directly and afterwards have the receiver challenge her to open random linear combinations of these instead. The amount of linear combinations required only depends on s, which has great impact on communication complexity if a large amount of commitments are required. The intuition for the security is the same as for the above and greatly reduces the number of openings the committer has to send to the receiver in our setting. The details of the check is described in full in Figure 15 in Appendix B.

A final optimization over [CDD+14] is the observation that most of the values we wish to commit to needs to be (pseudo-)randomly chosen. By using the code in systematic form we can simply let these messages be defined to be the XOR of the output of the PRG on the seeds, thus saving us k bits of communication for each of these commitments. In particular all input keys match this criteria. However, as the garbling scheme we use has the property that the output keys of a gate are constructed from the input keys, the committer must send corrections to the commitments of the output keys once these has been determined. Summing up; for all output keys the committer sends masked sharings of the k first codeword entries, while for *all* keys she sends masked sharings of the $\Gamma - k$ parity check-bits. For the full details of these optimizations and the entire construction we direct the reader to Appendix B. In Figure 11 therein the procedures for correcting output keys and parity check-bits are described as **Key Correct** and **Codeword Correct**, respectively.

5 Interactive Garbling

In this section we present our implementation of an interactive garbling scheme. We start from the *non-interactive* garbling scheme HGarb, described in Section 4.1 and lift this up to the interactive setting. We recall that the goal of such a protocol is for the participants C and E to mutually agree on a garbled circuit. In the end of the protocol it must be the case that C outputs (F, e, d, o) while E outputs (F, v). We denote our realization of an interactive garbling scheme $\mathsf{IGarb}_\pi = (\mathsf{IGb}_\pi, \mathsf{IEn}, \mathsf{IDe}, \mathsf{IEv}, \mathsf{Iev}, \mathsf{IOp}, \mathsf{IVe})$. IGb_π is the garbling protocol and it is described in the $\mathcal{F}_{\mathsf{OT}}$-,$\mathcal{F}_{\mathsf{COM}}$-hybrid model.[3] The remaining six algorithms $(\mathsf{IEn}, \mathsf{IDe}, \mathsf{IEv}, \mathsf{Iev}, \mathsf{IOp}, \mathsf{IVe})$ are based on the underlying algorithms of HGarb and the procedures of our ECC commitments.

[3] $\mathcal{F}_{\mathsf{COM}}$ should not be mistaken with the previously described ECC commitments. See Figure 22 and Figure 23 in Appendix F for descriptions of these functionalities.

5.1 Protocol Description

In a nutshell, our protocol consists of doing cut-and-choose of independently garbled gates, which are then soldered together into fault tolerant buckets, which are again soldered together into a fault tolerant circuit. Robustness is guaranteed by ensuring a combined majority of correct gates and correct wire authenticators in each bucket, under the constraint that there must be at least one correct gate in each bucket. As wire authenticators are lighter than gates, in terms of communication, we get a significant reduction in communication over MiniLEGO where buckets only consisted of gates.

High Level Description To be a bit more specific, but still informal, we describe the interactive garbling protocol IGarb_π in the following. For the full details we direct the reader to Appendix A.

Setup Alice and Bob use the ECC commitments described in Section 4.3 to setup commitments to sufficiently many 0-keys along with a global offset Δ and wire authenticators. Next, Bob commits to his challenges for the following cut-and-choose phase, and a specification of how the gates and authenticators are to be soldered into buckets.

Gate Construction Alice uses the keys on the wires to garble gates with the HGb method and sends these to Bob. Next Alice sends "corrections" to her commitments (the **Key Correct** and **Codeword Correct** steps of the ECC commitments) as she now knows the output keys. The parties then complete a LCC to ensure that the commitments are indeed of valid codewords.

Cut-and-Choose Bob opens the commitments of his challenges for both the gates and authenticators; the gates (authenticators) selected for checking are called the check gates (authenticators) and the remaining are called the evaluation gates (authenticators). Furthermore, the challenges also include a choice of input bits which the gates (authenticators) should be evaluated on (checking all possible input combinations would reveal Δ to Bob and thus break the privacy of the protocol). Based on Bob's choice, Alice uses the LCC to open to random linear combinations of the wires used by the check gates (authenticators). Bob then verifies that they are valid openings and that the opened values correspond to the learned keys.

Soldering Bob opens to his chosen bucketing functions and thus which evaluation gates and authenticators should be soldered together into buckets and how these buckets should be soldered together into a complete circuit. More specifically, one gate in a bucket is selected as the *head gate*, then the soldering consists of the following three types:

- **Bucket Soldering** For each bucket Alice solders the left-, right- and output-wire of the head gate onto the left-, right-, and output-wire of each other gate in a bucket.
- **Topological Soldering** For each bucket the left and right parents of the bucket's output keys are soldered onto the left, respectively right input wire of the head of the bucket. Remember that when a gate's input wire have more than one parent, the input is defined to be an XOR gate applied to the output of all the parents. Furthermore, Alice solders the required authenticators onto the output wire of the head gate.
- **Input Authentication** For each input wires of the circuit a bucket consisting purely of authenticators is associated. In this bucket a head authentication wire is selected which Alice solders the additional wire authenticators of the bucket onto.

For details on the above bucketing, see Section A.2. We stress that Alice sends the required solderings directly and then opens to random subsets of these using the aforementioned linear combination check instead of opening each soldering individually.

Output All the data is put together to form a tuple in correspondence with the definition of an interactive garbling scheme.

The algorithms for encoding and decoding wire keys follow directly from HGarb and the procedures for selecting and verifying openings follow from the ECC commitments. Finally the evaluation is carried out by evaluating the buckets in topological order. If the gates of a bucket do not agree on a distinct output key, then the authenticators of the bucket are also evaluated on each of the potential output keys. The key which is output and accepted by the most gates and authenticators of the bucket is defined as the output key. By

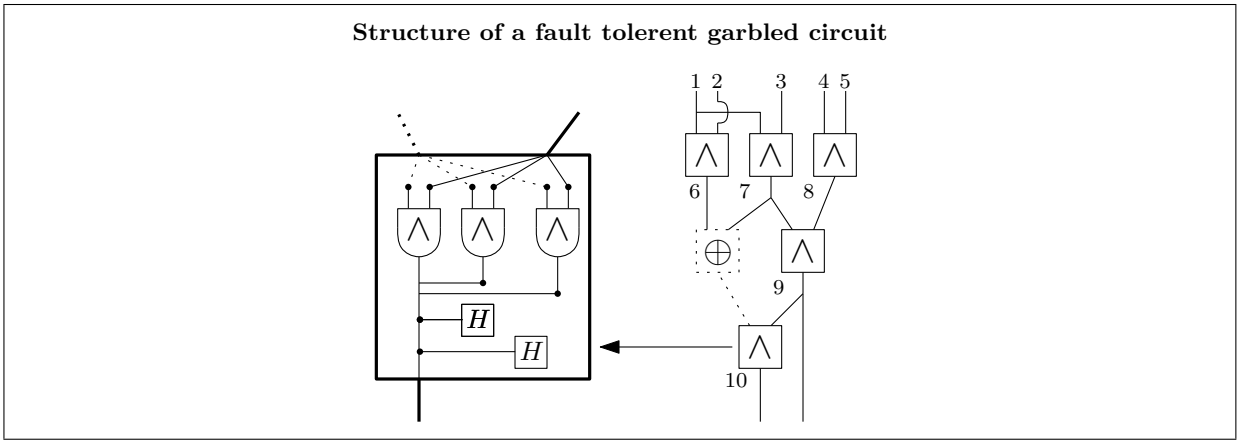

Fig. 3. Illustration of the wirings of a fault tolerent garbled circuit. The right hand side of the image shows a garbled circuit consisting of 5 garbled AND buckets where one bucket has its left input being the XOR of the output of two earlier buckets. The left hand side shows an AND bucket with 3 garbled gates and 2 authenticators. Furthermore a possible enumeration of the wires is shown. Notice that a small black-filled circle is used to illustrate solderings of wires.

the security analysis in Appendix D we have that the above will always output the single correct key for each bucket, except with probability 2^{-s}.

6 Performance Comparison

The computational complexity of TinyLEGO (and other LEGO-based protocols [NO09, FJN+13]) is $O(q \cdot s/\log q)$. This is asymptotically better than other recent two-party protocols [Bra13, Lin13, HKE13, AMPR14, FJN14] which achieve at best $O(q \cdot s)$. On the other hand TinyLEGO has more overhead per gate due to bucketing, wire authenticators and solderings. The natural questions are therefore (1) how do we perform compared to other LEGO-based protocols and (2) which concrete circuit sizes are required for our protocol to outperform other 2PC protocols not based on LEGO.

Ideally we would compare protocols based on prototype implementations, but instead we give a concrete efficiency count. This allows others to do the same and do reasonable comparison to our protocol. Experience from implementations of protocols based on Yao's garbling [KSS12, FN13, FJN14] show that with realistic circuits the communication overhead often becomes a major bottleneck. Especially the communication from the constructor (Alice) to the evaluator (Bob). So in our analysis we focus solely on this and ignore the overhead that does not depend on the circuit size. Comparing this way only makes sense for large circuits where the fraction of input and output wires compared to the total number of wires is small. This is the case for many real world circuits.

With the recent half-gate technique [ZRE14] each garbled gate is represented using as little as $2k$ bits while still being compatible with the free-XOR technique [KS08]. Prior to [Lin13, HKE13, Bra13] the most efficient non-LEGO protocols required sending $3.1s$ copies of the garbled circuits [SS11], resulting in a total communication overhead of $6.2qks$ bits. Recent protocols [Lin13, HKE13, Bra13, AMPR14, FJN14] only require Alice to down to s garbled circuits to Bob, yielding instead a total of down to $2qks$ bits.

In some cases such as [AMPR14, FJN14] the random seed checking optimization [GMS08] can be used to make the communication overhead of check circuits independent of the circuit size, at the price of additional computation. This means a communication overhead of $c \cdot 2qks$ for some fraction $c < 1$. Standard values (avoiding excessive local computation) is $c = 1/2$ [LP11, Lin13, FJN14] or $c = 3/5$ [SS11]. However the random seed checking optimization is only known to work in the random oracle model. Hence, the smallest

communication overhead of non-LEGO protocols is $2qks$ in the standard model and qks (or even less) in the random oracle model.[4]

Contrary to the other protocols, the communication overhead of a single AND gate in TinyLEGO (and other LEGO protocols) decreases as the circuit size grows. The concrete counting of communication overhead for TinyLEGO and MiniLEGO can be found in Appendix H.

Table 1 shows the amount of data that Alice must send to Bob for various circuit sizes and security levels. In the table qks refers to the minimal communication overhead achieved by non-LEGO protocols in the random oracle model so far, *e.g.*, with a protocol such as [FJN14] using random seed checking. $2qks$ reflects current best non-LEGO protocols in the standard model, *e.g.*, [Lin13]. Table 1 also shows communication overhead for MiniLEGO and TinyLEGO. For each value of k, s, and circuit size q, the parameters of MiniLEGO (β') and TinyLEGO (β, α, p_g, p_a) have been chosen[5] so as to minimize the overall communication overhead while still guaranteeing security except with probability 2^{-s}.

s	Protocol	Circuit size q								
		10^3	10^4	10^5	$5 \cdot 10^5$	10^6	$5 \cdot 10^6$	10^7	10^8	10^9
40	$2qks$ [Lin13]	0.020	0.095	0.95	4.77	9.5	48	95	954	9,537
40	qks [FJN14]	0.005	0.048	0.48	2.38	4.8	24	48	477	4,768
40	MiniLEGO	1.089	8.467	60.47	302.37	604.8	1,814	3,628	36,275	362,754
40	TinyLEGO	0.015	0.106	0.83	3.49	6.5	31	60	547	4,976
60	$2qks$ [Lin13]	0.014	0.143	1.43	7.15	14.3	72	143	1,431	14,305
60	qks[FJN14]	0.007	0.072	0.72	3.58	7.2	36	72	715	7,153
60	MiniLEGO	2.686	19.700	161.18	626.77	1,253.5	4,477	8,953	89,532	895,321
60	TinyLEGO	0.037	0.175	1.31	5.81	10.8	48	93	776	7,164
80	$2qks$ [Lin13]	0.019	0.191	1.91	9.53	19.1	95	191	1,907	19,073
80	qks [FJN14]	0.010	0.095	0.95	4.77	9.5	48	95	954	9,537
80	MiniLEGO	4.507	30.837	260.93	1,067.40	2,134.8	8,302	16,603	166,035	$1.2 \cdot 10^6$
80	TinyLEGO	0.251	0.411	2.01	8.19	15.8	69	132	1,129	10,531

Table 1. Amount of gibits (*i.e.*, 2^{30} bits) that Alice must send to Bob for $k = 128$.

As expected we clearly outperform MiniLEGO, with a factor of one to two orders of magnitude. This is due to our more efficient ECC commitments yielding smaller constants and because we do not need a majority of garbled gates in each bucket. The circuit size where TinyLEGO outperforms the non-LEGO protocols depends on whether or not random seed checking is used. If not, this happens at some point between circuits of size 10^4 and 10^5 for $s = 40$ and $s = 60$. For $s = 80$ it happens for circuit sizes between 10^5 and 10^6. With random seed checking a circuit size of more than a billion gates is needed before TinyLEGO is on par with non-LEGO protocols.

Once again we stress that this is only a rough indicator of performance. Many factors are not taken into account here, including cases where local computation is the bottleneck and circuits where a considerable fraction of the wires are input and output wires. In the latter case, however, we expect TinyLEGO to compare well with existing protocols.

To give a more precise idea of when TinyLEGO performs better than recent protocols in the standard model such as [Lin13] (without random seed checking) Table 2 shows, for $k = 128$ and different values of s and some selected parameters α, β, p_a, p_g, the minimal circuit size q where our protocol outperforms [Lin13]

[4] Because all gates in TinyLEGO are garbled using the same global difference Δ, we cannot immediately use the [GMS08] optimization.

[5] This paper does not give a method for finding the provably optimal parameters. Instead, we searched for good parameters using a script. See Appendix H for details.

with respect to communication overhead. As before, the parameters α, β, p_a, p_g are simply the best that we were able to find. Again we see that bigger circuits yield better relative performance of TinyLEGO.

s	α	β	p_a	p_g	q	[Lin13]	TinyLEGO
40	3	4	0.10	0.10	953,021	9.09	6.23 (0.69)
40	3	4	0.20	0.15	452,622	4.32	3.16 (0.73)
40	3	4	0.65	0.15	320,320	3.05	2.88 (0.94)
40	4	5	0.25	0.20	21,293	0.20	0.20 (1.00)
60	4	5	0.02	0.05	8,501,426	121.61	79.34 (0.65)
60	4	5	0.05	0.05	5,289,299	75.66	49.75 (0.66)
60	4	5	0.20	0.25	593,941	8.50	6.83 (0.80)
60	5	6	0.25	0.10	157,297	2.25	1.99 (0.88)
60	7	6	0.10	0.20	49,730	0.71	0.71 (1.00)
80	5	6	0.10	0.05	6,603,497	125.95	88.47 (0.70)
80	7	6	0.02	0.10	2,120,537	40.45	31.67 (0.78)
80	6	7	0.10	0.15	324,250	6.18	5.52 (0.89)
80	8	7	0.05	0.20	123,127	2.35	2.33 (0.99)

Table 2. Communication overhead (GB) of TinyLEGO compared to other recent 2PC protocols, *e.g.* [Lin13], in the standard model for various parameters. The numbers in parentheses are the relative communication overhead of TinyLEGO.

We see that TinyLEGO is indeed competitive for realistic circuit sizes. For instance, for 40-bit statistical security our bandwidth becomes slightly better than [Lin13] at only 21,293 gates (0.2 GB) and at 953,021 gates our bandwidth (6.23 GB) is only 69% of [Lin13]. For 80-bit security, we outperform [Lin13] slightly at 123,127 gates (2.35 GB) and achieve 70% bandwidth at 6,6 million gates (126 GB).

Overall, our efficiency count suggests that TinyLEGO is definitely an improvement in the family of LEGO protocols [NO09, FJN$^+$13]. In addition it is among the most efficient 2PC protocols depending on circuit size and which optimizations can be applied.

References

[AHMR14] Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. How to efficiently evaluate ram programs with malicious security. Cryptology ePrint Archive, Report 2014/759, 2014.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2014.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 478–492. IEEE Computer Society, 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.

[Bra13] Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 441–463. Springer, 2013.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[CDD+14] Ignacio Cascudo, Ivan Damgård, Bernardo David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. *IACR Cryptology ePrint Archive*, 2014:829, 2014.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-xor" technique. In Ronald Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012.

[FJN+13] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2013.

[FJN14] Tore Kasper Frederiksen, Thomas P. Jakobsen, and Jesper Buus Nielsen. Faster maliciously secure two-party computation using the GPU. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 358–379. Springer, 2014.

[FN13] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the GPU. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 339–356. Springer, 2013.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 272–284. IEEE Computer Society, 2012.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2013.

[HKK+14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 458–475. Springer, 2014.

[HMSG13] Nathaniel Husted, Steven Myers, Abhi Shelat, and Paul Grubbs. GPU and CPU parallelization of honest-but-curious secure two-party computation. In Charles N. Payne Jr., editor, *Annual Computer Security Applications Conference, ACSAC '13, New Orleans, LA, USA, December 9-13, 2013*, pages 169–178. ACM, 2013.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30. ACM, 2007.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[KS06] Mehmet S. Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *Proceedings of 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 285–300. USENIX Association, 2012.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 259–276. Springer, 2011.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 476–494. Springer, 2014.

[MF06] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

[MR13] Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2013.

[MZ06] Robert H. Morelos-Zaragoza. *The Art of Error Correcting Coding*. John Wiley & Sons, 2nd edition, 2006.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer, 2009.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[Rog91] Phillip Rogaway. *The round complexity of secure protocols*. PhD thesis, Massachusetts Institute of Technology, 1991.

[RT13] Samuel Ranellucci and Alain Tapp. Secure two-party computation via leaky generalized oblivious transfer. *IACR Cryptology ePrint Archive*, 2013:99, 2013.

[SS06] Rudolf Schürer and Wolfgang Ch. Schmid. Mint: A database for optimal net parameters. In *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 457–469. Springer, 2006.

[SS10] Rudolf Schürer and Wolfgang Ch. Schmid. Mint - architecture and applications of the (t, m, s)-net and OOA database. *Mathematics and Computers in Simulation*, 80(6):1124–1132, 2010.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 386–405. Springer, 2011.

[SS13] Abhi Shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 523–534. ACM, 2013.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.

[ZRE14] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. *IACR Cryptology ePrint Archive*, 2014:756, 2014.

A Protocol Details

In this section we present in detail our interactive garbling scheme $\mathsf{IGarb}_\pi = (\mathsf{IGb}_\pi, \mathsf{IEn}, \mathsf{IDe}, \mathsf{IEv}, \mathsf{Iev}, \mathsf{IOp}, \mathsf{IVe})$. The dominant work of the scheme is performed in the garbling protocol IGb_π which is presented in its entirety in Figure 6, Figure 7, Figure 8, and Figure 9. In Figure 10 the remaining six algorithms are presented. In Section A.1 the parameters used in the scheme are presented and in Section A.2 the details of our bucketing mechanism is described in full. As the production and evaluation of gates and wire authenticators are an integral part of our protocol we include the full details of these in Figure 4 and Figure 5, respectively.

HGarb

Let $\mathcal{H}(\cdot,\cdot)$ denote a circular correlation robust hash function for natural keys giving k bits of output.[a] Call the least significant bit of every key the *permutation bit* and define a function $\mathsf{lsb} : \{0,1\}^k \to \{0,1\}$ which extracts exactly this bit. We write $p_L \leftarrow \mathsf{lsb}(L^0)$, $p_R \leftarrow \mathsf{lsb}(R^0)$ and $p_O \leftarrow \mathsf{lsb}(O^0)$ and assume that $\mathsf{lsb}(\Delta) = 1$.

$\mathsf{HGb}\left(L^0, R^0, \Delta, id\right) \to \left(G_{id}, O^0\right)$:

For any key in the following we have that $K^b = K^0 \oplus (b \cdot \Delta)$. Also $L^0, R^0, \Delta \in \{0,1\}^k$ and we let $\mathtt{G}$ and $\mathtt{E}$ be two distinct literals and id be a unique and numeric identifier of fixed length.[b] Then do as follows:

1. Compute:

$$T_G = \mathcal{H}\left(L^0, id\|\mathtt{G}\right) \oplus \mathcal{H}\left(L^1, id\|\mathtt{G}\right) \oplus (p_R \cdot \Delta)$$
$$G = \mathcal{H}\left(L^0, id\|\mathtt{G}\right) \oplus (p_L \cdot T_G)$$
$$T_E = \mathcal{H}\left(R^0, id\|\mathtt{E}\right) \oplus \mathcal{H}\left(R^1, id\|\mathtt{E}\right) \oplus L^0$$
$$E = \mathcal{H}\left(R^0, id\|\mathtt{E}\right) \oplus \left(p_R \cdot \left(T_E \oplus L^0\right)\right)$$

2. Then define $O^0 = G \oplus E$.
3. Output a garbled gate $\left(G_{id}, O^0\right) \leftarrow \left(\left(id, T_G, T_E, 0^k, 0^k, 0^k\right), O^0\right)$.

$\mathsf{HEv}\left(G_{id}, L, R\right) \to O$:

1. First parse $\left(id, T_G, T_E, S^L_{id}, S^R_{id}, S^O_{id}\right) \leftarrow G_{id}$.
2. Next compute $p'_L \leftarrow \mathsf{lsb}\left(L \oplus S^L_{id}\right)$ and $p'_R \leftarrow \mathsf{lsb}\left(R \oplus S^R_{id}\right)$, let $i = 2p'_L + p'_R$.
3. Compute

$$G = \mathcal{H}\left(L \oplus S^L_{id}, id\|\mathtt{G}\right) \oplus \left(p'_L \cdot T_G\right)$$
$$E = \mathcal{H}\left(R \oplus S^R_{id}, id\|\mathtt{E}\right) \oplus \left(p'_R \cdot \left(T_E \oplus L \oplus S^L_{id}\right)\right)$$

4. Let $O = G \oplus E$ and output $O = S^O_{id} \oplus O$.

$\mathsf{SGa}\left(G_{id}, L, R, O\right) \to \widetilde{G}_{id}$:

Parse $\left(id, T_G, T_E, S^L_{id}, S^R_{id}, S^O_{id}\right) \leftarrow G_{id}$ and output $\widetilde{G}^{id} = \left(id, T_G, T_E, S^L_{id} \oplus L, S^R_{id} \oplus R, S^O_{id} \oplus O\right)$.

[a] See Appendix D for a definition of this notion.

[b] These literals are included to ensure that T_G and T_E are always constructed using unique identifiers. This is merely one way of ensuring this.

Fig. 4. Individual garbling and evaluation of AND gates.

A.1 Parameters and replication

In the following Alice will play the role of C and Bob will play the role of E. We will let $\ell_g = \frac{1}{1-p_g-\delta_g}$ be the replication factor of gates, where p_g is the expected fraction of gates we sacrifice in the cut-and-choose step and δ_g is the fraction of extra gates we garble to ensure the actual number of remaining gates is not lower than expected. In addition we also need to consider the bucket size needed for the protocol which we denote

Authenticated Wires

Let $\mathcal{H}(\cdot,\cdot)$ denote a circular correlation robust hash function for natural keys giving k bits of output. Let $\mathtt{A}$ be a distinct literal.[a]

$\mathsf{Auth}\left(K^0_{id}, \Delta, id\right) \to H_{id}$:

1. Compute
$$H^0_{id} \leftarrow \mathcal{H}\left(K^0_{id}, id\|\mathtt{A}\right), \quad H^1_{id} \leftarrow \mathcal{H}\left(K^0_{id} \oplus \Delta, id\|\mathtt{A}\right) .$$
2. View H^0_{id} and H^1_{id} as binary strings and output $H_{id} = \left(H^0_{id}, H^1_{id}, 0^k\right)$ if $H^0_{id} \leq H^1_{id}$, otherwise output $H_{id} = \left(H^1_{id}, H^0_{id}, 0^k\right)$.

$\mathsf{SAuth}\left(H_{id}, S\right) \to \widetilde{H}_{id}$:

1. Parse $\left(H^a_{id}, H^b_{id}, S_{id}\right) \leftarrow H_{id}$ and output $\widetilde{H}_{id} \leftarrow \left(H^a_{id}, H^b_{id}, S \oplus S_{id}\right)$.

$\mathsf{Ver}\left(H_{id}, K_{id}, id\right) \to \top/\bot$:

1. Parse $\left(H^a_{id}, H^b_{id}, S_{id}\right) \leftarrow H_{id}$. If $\mathcal{H}(K_{id} \oplus S_{id}, id\|\mathtt{A}) = H^a_{id}$ or $\mathcal{H}(K_{id} \oplus S_{id}, id\|\mathtt{A}) = H^b_{id}$ output $\top$, otherwise output $\bot$.

[a] Here simply used to pad the input size of the hash function.

Fig. 5. Methods for achieving authenticated wires.

by β. For a circuit $C = (n, m, q, \mathsf{lp}, \mathsf{rp})$ let $w = n + q$ and therefore Alice needs to garble $Q = q\beta\ell_g$ gates in total.

Each of these Q gates require three wires each. This is because we garble all AND gates individually, meaning they all have a left, right and output wire. In addition to these gate wires, we also need to produce the required wire authenticators. We will need $2(\beta + 1)$ wire authenticators pr. input gate of the circuit and we denote by α the number of wire authenticators we need for each bucket. We ensure the quality of these authenticators by performing a cut-and-choose test in a similar manner as we do for the gates. Therefore we let $\ell_a = \frac{1}{1-p_a-\delta_a}$ be the replication factor of the wire authenticators, where p_a is the expected fraction we check and δ_a is the fraction of extra wires we produce to ensure the number of non-checked wire authenticators is not lower than expected. Thus we need to produce $A = (q\alpha + n(2\beta+1))\ell_a$ wire authenticators. In summary we need to produce commitments to $W = 3Q + A$ wires.

For convenience we let $\mathsf{GWires} = [W]$ be the indices of these wires and furthermore we let $\mathsf{GateWires} = [3Q]$ and $\mathsf{AWires} = \{3Q + 1, \ldots, 3Q + A\}$. As gates are indexed by their output wire we also let $\mathsf{GGates} \subset \mathsf{GateWires}$ denote the indices of garbled gates, meaning that $|\mathsf{GGates}| = Q$. In the protocol, Bob will sample a subset $\mathsf{CheckGates} \subset_R \mathsf{GGates}$ where each of the gates in GGates is included with probability p_g. These gates will be the ones sacrificed during the cut-and-choose step. To illustrate why the extra fraction δ_g is needed lets assume that we did not include it and let $Q' = q\beta\frac{1}{1-p_g}$. Then we let G be the amount of gates not chosen for checking and we see that $\mathbb{E}[G] = Q'(1 - p_g) = q\beta\frac{1}{1-p_g}(1 - p_g) = q\beta$ which is exactly what is needed to create q buckets each of size β. However, it is clear that if we check even a single gate more than expected (which is quite likely) then we do not have enough gates to build q buckets each consisting of β gates. This is the reason for us including the extra "slack" fraction δ_g, which means we produce a little extra, but which ensures us that at least $q\beta$ gates are left after the cut-and-choose step except with negligible probability. We handle this slack explicitly in Lemma 1. Therefore after the cut-and choose phase there will be enough gates left for creating buckets of size β for each gate of the circuit C.

In an analogous manner Bob will also sample $\mathsf{CheckWires} \subset_R \mathsf{AWires}$ where each wire is included in $\mathsf{CheckWires}$ with probability p_a. For the exact same reason as above we produce a little extra, decided by δ_a, to ensure we have at least $q\alpha + n(2\beta + 1)$ wire authenticators left after the cut-and-choose step.

Finally in order to ensure privacy for Alice we produce some additional wires which are "consumed" by the **Linear Combination Check** of the ECC commitments. By Theorem 2 in Appendix E each check needs to consist of at least $7.3(s + 3)$ random linear combinations to guarantee security 2^{-s}. Because of the nature of the first codeword check we need to sacrifice a commitment for each of the opened linear combinations in order to guarantee privacy for Alice. In the following two checks this is not necessary because Bob at this

point already knows the individual keys. Therefore, we end up needing to produce $\gamma = W + 1 + 7.3(s+3)$ wires in total, where the 1 is due to also producing a commitment to the global difference Δ.

As already mentioned, since we check gates (wires) independently at random we need to guarantee that after the cut-and-choose phase enough gates and wires remain to successfully build the fault tolerant garbled circuit. We therefore introduced the variables δ_g and δ_a which represents the additional fraction of gates (wires) we need to produce for this situation not to occur except with exponentially small probability in the security parameter. The calculations of the value of δ_g and δ_a are captured in the following lemma.

Lemma 1 (Tail Bounds). *Let* $R_g = |\mathsf{GGates} \setminus \mathsf{CheckGates}|$ *and* $R_a = |\mathsf{AWires} \setminus \mathsf{CheckWires}|$ *denote the random variables representing the number of remaining gates and wire authenticators after the cut-and-choose step of Figure 6. Then*

$$\Pr[R_g \leq q\beta] \leq e^{-2\delta_g^2 Q} \quad \textit{and}$$
$$\Pr[R_a \leq (q\alpha + n(2\beta+1))] \leq e^{-2\delta_a^2 A}$$

where $Q = q\beta \cdot \frac{1}{1-p_g-\delta_g}$ *and* $A = (q\alpha + n(2\beta+1)) \cdot \frac{1}{1-p_a-\delta_a}$.

Proof. We look at the two statements individually. Since a gate is selected for checking with probability p_g in the protocol, we keep a gate for evaluation with probability $1-p_g$. We now observe that R_g is in fact a sum of identically distributed independent Bernoulli trials with success probability $1-p_g$. We can thus apply the Hoeffding bound [Hoe63] yielding

$$\begin{aligned}\Pr[R_g \leq q\beta] &= \Pr[R_g \leq ((1-p_g) - \delta_g) \cdot Q] \\ &\leq e^{-2\delta_g^2 Q}\end{aligned}$$

By the exact same reason we see that

$$\begin{aligned}\Pr[R_a \leq (q\alpha + n(2\beta+1))] &= \Pr[R_a \leq ((1-p_a) - \delta_a) \cdot A] \\ &\leq e^{-2\delta_a^2 A}\end{aligned}$$

which proves the statement. □

A.2 Bucketing

In the protocol, individual garbled gates are combined together into buckets. We here introduce some convenient notation that allows us to describe this precisely. For each gate in the circuit C one garbled gate is defined to be the head gate corresponding to this gate. A bucket is then constructed by soldering the wires of the $\beta - 1$ other gates in the bucket onto the wires of the head gate.

To be more precise, once Bob has decided on the set $\mathsf{CheckGates}$ he will sample a random subset $\mathsf{EvalGates} \subseteq_R \mathsf{GGates} \setminus \mathsf{CheckGates}$ of size exactly $q\beta$. He then lets $\mathcal{B}$ be the family of all injective β-to-1 functions from $\mathsf{EvalGates}$ to $\mathsf{Gates} = \{n+1, \ldots, w\}$. Now for a function $\mathsf{BucketOf} \in \mathcal{B}$ we let $\mathsf{Bucket}_h = \{g \in \mathsf{EvalGates} \mid \mathsf{BucketOf}(g) = h\}$. For all $h \in \mathsf{Gates}$ we define the head gate of Bucket_h to be the gate with lowest lexicographical index. For convenience we let $\mathsf{HeadGates}$ be the set of these head gate indices. Finally we assume that given $\mathsf{BucketOf}$, it is easy to identify the domain of the function, meaning that $\mathsf{EvalGates}$ is assumed to be directly identified from the description of $\mathsf{BucketOf}$.

Analogously we also need to specify how the wire authenticators are to be combined with the buckets. Again Bob will sample a random subset $\mathsf{AuthWires} \subseteq_R \mathsf{AWires} \setminus \mathsf{CheckWires}$ of size exactly $n(2\beta+1)+q\alpha$. We then let $\mathcal{V}$ be the family of all injective functions from $\mathsf{AuthWires}$ to $[w]$ where for the image subset $\mathsf{Inputs} = [n] \subset [w]$ the functions are $(2\beta+1)$-to-1 and for the remaining image elements $\{n+1, \ldots, w\}$ the functions are α-to-1. As in the above for a function $\mathsf{AuthOf} \in \mathcal{V}$ we let $\mathsf{Auth}_i = \{a \in \mathsf{AuthWires} \mid \mathsf{AuthOf}(a) = i\}$. For all $i \in [n]$ we define the head authenticator of Auth_i to be the wire authenticator with lowest lexicographical index. For convenience we let $\mathsf{HeadAuths}$ be the set of these head authenticator indices. Also in this case we assume that the domain $\mathsf{AuthWires}$ is efficiently determined from AuthOf.

In the protocol, once Alice has learned the bucketing functions AuthOf and BucketOf, both parties reenumerate the gate and wire authenticators indices in the following way:

1. Reenumerate the indices of HeadGates with indices from Gates such that the lowest index of HeadGates gets replaced with the lowest index of Gates, second lowest with second lowest, and so on.
2. Reenumerate the indices of HeadAuths with indices from Inputs such that the lowest index of HeadAuths gets replaced with the lowest index of Inputs, second lowest with second lowest, and so on.

We now see that the functions lp and rp of C are well-defined to work on all indices of HeadGates while preserving the semantics of the circuit. Also a function $\mathsf{AuthOf} \in \mathcal{V}$ now specifies which authentication wires are soldered into "buckets" of size $2\beta+1$ and which wires are to be soldered onto the head gates' output wires.

For convenience we finally define a function WiresOf that maps an index of a garbled gate to three wires, which will represent the left-, right-, and output-wire respectively associated with that gate. We let WiresOf be deterministic and known to all parties and assume that when gate indices are reenumerated WiresOf is updated accordingly by all parties.

Protocol IGb_π in the $\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{COM}}$-hybrid model

Common Input: $C = (n, m, q, \mathsf{lp}, \mathsf{rp}), p_g, p_a, \beta, \alpha = \beta - 1$.

Setup:

1. Let $Q = q\beta\frac{1}{1-p_g-\delta_g}$, $A = (q\alpha + n(2\beta+1))\frac{1}{1-p_a-\delta_a}$ and $W = 3Q + A$.[a] Alice and Bob initialize $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{COM}}$ and execute the **Setup** step of Figure 11 using a linear ECC with minimum distance $d = s+4$ to commit to $\gamma = W + 1 + 7.3(s+3)$ wires. Let $\left\{\boldsymbol{s}_j\right\}_{j\in[\gamma]}$ be the output of Alice and $\left(\left\{\boldsymbol{w}_j\right\}_{j\in[\gamma]}, \boldsymbol{b}\right)$ be the output of Bob. For all $j \in [\gamma]$, Alice parses $\left(\boldsymbol{s}_j^0, \boldsymbol{s}_j^1\right) \leftarrow \boldsymbol{s}_j$ and denote by $\boldsymbol{k}_j^b$ the first k bits of $\boldsymbol{s}_j^b$. Let $K_j^0 = \boldsymbol{k}_j^0 \oplus \boldsymbol{k}_j^1$ and as a special case defines $\mathsf{lsb}\left(\boldsymbol{k}_{W+1}^0\right) := 0$ and $\mathsf{lsb}\left(\boldsymbol{k}_{W+1}^1\right) := 1$ regardless of the output of **Setup** and let $\Delta = \boldsymbol{k}_{W+1}^0 \oplus \boldsymbol{k}_{W+1}^1$.
2. As described in Section A.2, Bob samples $\mathsf{CheckGates} \subset_R \mathsf{GGates}$ and $\mathsf{CheckWires} \subset_R \mathsf{AWires}$ where each gate (wire authenticator) is included with probability p_g (p_a). If $|\mathsf{GGates} \setminus \mathsf{CheckGates}| < q\beta$ or $|\mathsf{AWires} \setminus \mathsf{CheckWires}| < q\alpha + n(2\beta+1)$, Bob outputs $\perp$ and terminates.
3. Next, Bob samples $\mathsf{BucketOf} \in_R \mathcal{B}$ and $\mathsf{AuthOf} \in_R \mathcal{V}$ and for each $g \in \mathsf{CheckGates}$ and $j \in \mathsf{CheckWires}$, he samples three bits $a_g, b_g, c_j \in_R \{0,1\}$. He then sends $(\mathsf{commit}, \mathsf{sid}, 1, \{(a_g, b_g, g)\}, \{(c_j, j)\})$ and $(\mathsf{commit}, \mathsf{sid}, 2, (\mathsf{BucketOf}, \mathsf{AuthOf}))$ to $\mathcal{F}_{\mathsf{COM}}$ which in turn sends $(\mathsf{commit}, \mathsf{sid}, 1)$ and $(\mathsf{commit}, \mathsf{sid}, 2)$ to both parties.

Gate Construction:

1. For all $g \in \mathsf{GGates}$, let $(l_g, r_g, o_g) \leftarrow \mathsf{WiresOf}(g)$ and denote $(L_g^0, R_g^0) \leftarrow (K_{l_g}^0, K_{r_g}^0)$. Alice computes $\left\{G_g, O_g^0\right\} \leftarrow \mathsf{HGb}\left(L_g^0, R_g^0, \Delta, g\right)$. Also for all $j \in \mathsf{AWires}$, Alice computes $H_j \leftarrow \mathsf{Auth}(K_j^0, \Delta, j)$ and then sends $\left\{G_g\right\}, \left\{H_j\right\}$ to Bob. If $\left|\left\{G_g\right\}\right| \neq Q$ or if $\left|\left\{H_j\right\}\right| \neq A$, Bob outputs $\perp$ and terminates.
2. Alice and Bob then execute the **Key Correct** step of Figure 12 where Alice has input $\left(\left\{\boldsymbol{s}_g\right\}_{g\in\mathsf{GGates}}, \left\{O_g^0\right\}_{g\in\mathsf{GGates}}, \mathsf{GGates}\right)$ and Bob has input $\left(\left\{\boldsymbol{w}_g\right\}_{g\in\mathsf{GGates}}, \boldsymbol{b}, \mathsf{GGates}\right)$.[b] Let $\left(\left\{\boldsymbol{s}_g\right\}_{g\in\mathsf{GGates}}\right) = \left(\left\{\boldsymbol{k}_g^0, \boldsymbol{k}_g^0 \oplus O_g^0\right\}_{g\in\mathsf{GGates}}\right)$ be the updated shares of Alice and $\left\{\boldsymbol{w}_g\right\}_{g\in\mathsf{GGates}}$ be the updated watch-bits of Bob, returned by the **Key Correct** call.

[a] See Section A.1 for details on these parameters.

[b] Recall that gates are indexed by their output wire.

Fig. 6. Interactive garbling protocol IGb_π– part 1.

Protocol IGb_π in the $\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{COM}}$-hybrid model

For clarity we will write $\left(L_g^0, R_g^0, O_g^0\right)$ to mean $\left(K_{l_g}^0, K_{r_g}^0, K_{o_g}^0\right)$.

Commitment Correction:

1. Alice and Bob execute the **Codeword Correct** step of Figure 12 where Alice has input $\left(\left\{\boldsymbol{s}_j\right\}_{j\in[\gamma]}, [\gamma]\right)$ and Bob has input $\left(\left\{\boldsymbol{w}_j\right\}_{j\in[\gamma]}, \boldsymbol{b}, [\gamma]\right)$. Let $\left\{(\boldsymbol{k}_j, \boldsymbol{c}_j)\right\}_{j\in[\gamma]}$ be the updated shares of Alice and $\left\{\boldsymbol{w}_j\right\}_{j\in[\gamma]}$ be the updated watch-bits of Bob using the **Codeword Correct** procedure.
2. Alice and Bob now execute the **Linear Combination Check** of Figure 15 with common input $\left(\{\{(j,0)\}\}_{j\in[W+1]}, \mathsf{true}\right)$, *i.e.* each set only contains a single pair. Also, Alice has private input $\left(\left\{(\boldsymbol{k}_j, \boldsymbol{c}_j)\right\}_{j\in[W+1]}, \boldsymbol{k}_{W+1}, \boldsymbol{c}_{W+1}\right)$ and Bob has private input $\left(\left\{\boldsymbol{w}_j\right\}_{j\in[W+1]}, \emptyset, \boldsymbol{b}, \boldsymbol{w}_{W+1}\right)$. If the check outputs $\perp$, Bob outputs $\perp$ and terminates.

Cut-and-Choose:

1. Bob sends $(\mathsf{open}, \mathsf{sid}, 1)$ to $\mathcal{F}_{\mathsf{COM}}$ which in turn sends $(\mathsf{open}, \mathsf{sid}, 1, \{(a_g, b_g, g)\}, \{(c_j, j)\})$ to Alice. For all g, if $g \notin \mathsf{GGates}$ or $a_g, b_g \notin \{0,1\}$, Alice outputs $\perp$ and terminates. Similarly for all j, if $j \notin \mathsf{AuthWires}$ or $c_j \notin \{0,1\}$, Alice outputs $\perp$ and terminates.
2. Let $S = O_{\mathsf{A}} = W_{\mathsf{B}} = R_{\mathsf{B}} = \emptyset$. For all $g \in \mathsf{CheckGates}$ and $j \in \mathsf{CheckWires}$, Alice and Bob do the following:
   - Alice sends $\left(L_g^{a_g}, R_g^{b_g}, g\right)$ and $\left(K_j^{c_j}, j\right)$ to Bob. Let L_g, R_g, K_j be the keys Bob receives. If $\mathsf{Ver}\left(K_j, H_j\right) \to \perp$, Bob outputs $\perp$ and terminates. Else he computes $O_g \leftarrow \mathsf{HEv}\left(G_g, L_g, R_g\right)$. Then both parties update the set $S := S \cup \left\{\left\{\left(l_g, a_g\right)\right\}, \left\{\left(r_g, b_g\right)\right\}, \left\{\left(o_g, a_g \wedge b_g\right)\right\}, \{(j, c_j)\}\right\}$. Also Alice updates $O_{\mathsf{A}} := O_{\mathsf{A}} \cup \left\{\left(\boldsymbol{k}_{l_g}, \boldsymbol{c}_{l_g}\right), \left(\boldsymbol{k}_{r_g}, \boldsymbol{c}_{r_g}\right), \left(\boldsymbol{k}_{o_g}, \boldsymbol{c}_{o_g}\right), \left(\boldsymbol{k}_j, \boldsymbol{c}_j\right)\right\}$ and Bob updates $W_{\mathsf{B}} := W_{\mathsf{B}} \cup \left\{\boldsymbol{w}_{l_g}, \boldsymbol{w}_{r_g}, \boldsymbol{w}_{o_g}, \boldsymbol{w}_j\right\}$ and $R_{\mathsf{B}} := R_{\mathsf{B}} \cup \left\{L_g, R_g, O_g, K_j\right\}$.
3. Alice and Bob now execute the **Linear Combination Check** of Figure 15 with common input (S, false). Also, Alice has private input $(O_{\mathsf{A}}, \boldsymbol{k}_{W+1}, \boldsymbol{c}_{W+1})$ and Bob has private input $(W_{\mathsf{B}}, R_{\mathsf{B}}, \boldsymbol{b}, \boldsymbol{w}_{W+1})$. If the check outputs $\perp$, Bob outputs $\perp$ and terminates.

Fig. 7. Interactive garbling protocol IGb_π– part 2.

Protocol IGb_π in the $\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{COM}}$-hybrid model

In the following, for all $g \in \mathsf{EvalGates}$ we let $h \leftarrow \mathsf{BucketOf}(g)$ and $\left(l_g, r_g, o_g\right) \leftarrow \mathsf{WiresOf}(g)$. For clarity we will also write $\left(L_g^0, R_g^0, O_g^0\right)$ to mean $\left(K_{l_g}^0, K_{r_g}^0, K_{o_g}^0\right)$.

Bucketing setup:

1. Bob sends $(\mathsf{open}, \mathsf{sid}, 2)$ to $\mathcal{F}_{\mathsf{COM}}$ which in turn sends $(\mathsf{open}, \mathsf{sid}, 2, (\mathsf{BucketOf}, \mathsf{AuthOf}))$ to Alice. If $\mathsf{BucketOf} \notin \mathcal{B}$ or $\mathsf{AuthOf} \notin \mathcal{V}$, Alice outputs $\perp$ and terminates.
2. Both parties then perform the reenumeration steps described in Section A.2. Also Alice and Bob reset the sets $S := O_{\mathsf{A}} := W_{\mathsf{B}} := R_{\mathsf{B}} := \emptyset$.

Bucket Soldering: For all $h \in \mathsf{HeadGates}$, all $g \in \mathsf{Bucket}_h$ where $g \neq h$ and all $a \in \mathsf{AuthOf}(h)$:

1. Alice sends to Bob:
$$\widetilde{S}_g^L = L_g^0 \oplus L_h^0, \quad \widetilde{S}_g^R = R_g^0 \oplus R_h^0, \quad \widetilde{S}_g^O = O_g^0 \oplus O_h^0 \quad \widetilde{S}_a = K_a^0 \oplus O_h^0 \ .$$
2. Bob updates $\widetilde{G}^g \leftarrow \mathsf{SGa}\left(G^g, \widetilde{S}_g^L, \widetilde{S}_g^R, \widetilde{S}_g^O\right)$ and $\widetilde{H}_a \leftarrow \mathsf{SAuth}(H_a, \widetilde{S}_h^a)$.
3. Both parties update the set $S := S \cup \left\{ \left\{ \left(l_g, 0\right), (l_h, 0) \right\}, \left\{ \left(r_g, 0\right), (r_h, 0) \right\}, \left\{ \left(o_g, 0\right), (o_h, 0) \right\}, \{(a, 0), (o_h, 0)\} \right\}$. Also Alice updates $O_{\mathsf{A}} := O_{\mathsf{A}} \cup \left\{ \left(\boldsymbol{k}_{l_g}, \boldsymbol{c}_{l_g}\right), \left(\boldsymbol{k}_{l_h}, \boldsymbol{c}_{l_h}\right), \left(\boldsymbol{k}_{r_g}, \boldsymbol{c}_{r_g}\right), \left(\boldsymbol{k}_{r_h}, \boldsymbol{c}_{r_h}\right), \left(\boldsymbol{k}_{o_g}, \boldsymbol{c}_{o_g}\right), \left(\boldsymbol{k}_{o_h}, \boldsymbol{c}_{o_h}\right), (\boldsymbol{k}_a, \boldsymbol{c}_a) \right\}$ and Bob updates $W_{\mathsf{B}} := W_{\mathsf{B}} \cup \left\{ \boldsymbol{w}_{l_g}, \boldsymbol{w}_{l_h}, \boldsymbol{w}_{r_g}, \boldsymbol{w}_{r_h}, \boldsymbol{w}_{o_g}, \boldsymbol{w}_{o_h}, \boldsymbol{w}_a \right\}$ and $R_{\mathsf{B}} := R_{\mathsf{B}} \cup \left\{ \widetilde{S}_g^L, \widetilde{S}_g^R, \widetilde{S}_g^O, \widetilde{S}_a \right\}$.

Topological Soldering For all $h \in \mathsf{HeadGates}$:

1. Let $L = \bigoplus_{u \in \mathsf{lp}(h)} O_u^0$, $R = \bigoplus_{v \in \mathsf{rp}(h)} O_v^0$. In case either $\mathsf{lp}(h) \in [n]$ or $\mathsf{rp}(h) \in [n]$ the wire K_v^0, respectively K_u^0 is used instead.
2. Alice sends to Bob:
$$\widetilde{S}_h^L = L_h^0 \oplus L, \quad \widetilde{S}_h^R = R_h^0 \oplus R \ .$$
3. For all $g \in \mathsf{Bucket}_h$, Bob updates $\widetilde{G}^g \leftarrow \mathsf{SGa}\left(\widetilde{G}^g, \widetilde{S}_h^L, \widetilde{S}_h^R, 0^k\right)$.
4. Alice and Bob define two sets of pairs, $A_h = \{(o_u, 0)\}_{u \in \mathsf{lp}(h)} \cup \{(l_h, 0)\}$ and $B_h = \{(o_v, 0)\}_{v \in \mathsf{rp}(h)} \cup \{(r_h, 0)\}$ and update the set $S := S \cup \{A_h, B_h\}$. Also, Alice updates $O_{\mathsf{A}} := O_{\mathsf{A}} \cup \{(\boldsymbol{k}_{o_u}, \boldsymbol{c}_{o_u})\}_{u \in \mathsf{lp}(h)} \cup \{(\boldsymbol{k}_{o_v}, \boldsymbol{c}_{o_v})\}_{v \in \mathsf{rp}(h)} \cup \left\{ \left(\boldsymbol{k}_{l_h}, \boldsymbol{c}_{l_h}\right), \left(\boldsymbol{k}_{r_h}, \boldsymbol{c}_{r_h}\right) \right\}$ and Bob updates $W_{\mathsf{B}} := W_{\mathsf{B}} \cup \{\boldsymbol{w}_{o_u}\}_{u \in \mathsf{lp}(h)} \cup \{\boldsymbol{w}_{o_v}\}_{v \in \mathsf{rp}(h)} \cup \left\{ \boldsymbol{w}_{l_h}, \boldsymbol{w}_{r_h} \right\}$ and $R_{\mathsf{B}} := R_{\mathsf{B}} \cup \left\{ \widetilde{S}_h^L, \widetilde{S}_h^R \right\}$.

Fig. 8. Interactive garbling protocol IGb_π– part 3.

Protocol IGb_π in the $\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{COM}}$-hybrid model

In the following, we let $\left(l_g, r_g, o_g\right) \leftarrow \mathsf{WiresOf}(g)$. For clarity we will also write $\left(L_g^0, R_g^0, O_g^0\right)$ to mean $\left(K_{l_g}^0, K_{r_g}^0, K_{o_g}^0\right)$.

Input Authentication For all $i \in \{\mathsf{HeadAuths} \cap [n]\}$ and all $a \in \mathsf{Auth}_i$ where $a \neq i$:

1. Alice sends to Bob:
$$\widetilde{S}_a = K_a^0 \oplus K_i^0 \ .$$
2. Bob updates $\widetilde{H}_a \leftarrow \mathsf{SAuth}\left(H_a, \widetilde{S}_a\right)$.
3. Both parties update the set $S := S \cup \{\{(a,0),(i,0)\}\}$. Also, Alice updates $O_{\mathsf{A}} := O_{\mathsf{A}} \cup \{(\boldsymbol{k}_a, \boldsymbol{c}_a), (\boldsymbol{k}_i, \boldsymbol{c}_i)\}$ and Bob updates $W_{\mathsf{B}} := W_{\mathsf{B}} \cup \{\boldsymbol{w}_a, \boldsymbol{w}_i\}$ and $R_{\mathsf{B}} := R_{\mathsf{B}} \cup \left\{\widetilde{S}_a\right\}$.
4. Alice and Bob now execute the **Linear Combination Check** of Figure 15 with common input (S, false). Also, Alice has private input $(O_{\mathsf{A}}, \boldsymbol{k}_{W+1}, \boldsymbol{c}_{W+1})$ and Bob has private input $(W_{\mathsf{B}}, R_{\mathsf{B}}, \boldsymbol{b}, \boldsymbol{w}_{W+1})$. If the check outputs $\perp$, Bob outputs $\perp$ and terminates.

Output:

1. Alice defines $e = \left(K_1^0, K_1^1, \ldots, K_n^0, K_n^1\right)$, $o = \left(\mathcal{O}_{K_1^0}, \mathcal{O}_{K_1^1}, \ldots, \mathcal{O}_{K_n^0}, \mathcal{O}_{K_n^1}\right)$ and $d = \left(O_{w-m+1}^0, O_{w-m+1}^1, \ldots, O_w^0, O_w^1\right)$.
2. Bob lets $v = ((\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n), \boldsymbol{w}_{W+1}, \boldsymbol{b})$, *i.e.* Bob's watch bits for the n head authenticators in $\{\mathsf{HeadAuths} \cap [n]\}$ along with the choice bits $\boldsymbol{b}$ Bob used in the **Setup** step of the ECC commitments.
3. Finally Alice and Bob define
$$F = \left(n, m, q, (\mathsf{Ip}, \mathsf{rp}, \mathsf{BucketOf}, \mathsf{AuthOf}), \left\{\widetilde{G}_g\right\}_{g \in \mathsf{EvalGates}}, \left\{\widetilde{H}_j\right\}_{j \in \mathsf{AuthWires}}\right)$$
to be the produced garbled circuit. Alice and Bob define their output to be (F, e, d, o) and (F, v), respectively.

Fig. 9. Interactive garbling protocol IGb_π– part 4.

Algorithms for the interactive garbling scheme IGarb_π

In the following let $\mathsf{GateScore}(O_g)$ be a function that returns the number of gates in a bucket that outputs O_g on the two input keys. Likewise let $\mathsf{AuthScore}(K_j)$ be a function that returns the number of authenticators of the current bucket that outputs $\top$ on input K_j.

$\mathsf{IEv}\,(F, X) \to Z/\bot$:

1. $\left(n, m, q, (\mathsf{Ip}, \mathsf{rp}, \mathsf{BucketOf}, \mathsf{AuthOf}), \left\{\widetilde{G}_g\right\}_{g\in\mathsf{EvalGates}}, \left\{\widetilde{H}_a\right\}_{a\in\mathsf{AuthWires}}\right) \leftarrow F.$
2. $(X_1, \ldots, X_n) \leftarrow X$ and $w = n + q$.
3. For all $i \in [n]$, if for any i we have that $\mathsf{AuthScore}(X_i) < \beta + 1$ output $\bot$.[a]
4. For $h = n + 1$ to w, let $L_h = \bigoplus_{l\in\mathsf{Ip}(h)} X_l$ and $R_h = \bigoplus_{r\in\mathsf{rp}(h)} X_r$ and do:
   (a) For all $g \in \mathsf{Bucket}_h$, compute $O_g \leftarrow \mathsf{HEv}(\widetilde{G}_g, L_h, R_h)$ and let $\mathsf{Cand} = \left\{O_g\right\}$.
   (b) If $|\mathsf{Cand}| = 1$, let $X_h = O_h$.
   (c) Else let $\mathsf{MAJ} = \left\{O_g \in \mathsf{Cand} \mid \mathsf{GateScore}(O_g) + \mathsf{AuthScore}(O_g) > \lfloor(\beta + \alpha)/2\rfloor\right\}$. If $|\mathsf{MAJ}| \neq 1$ output $\bot$. Else set X_h to be the singleton output key in MAJ.
5. Output $Z = (X_{w-m+1}, X_{w-m+2}, \ldots, X_w)$.

$\mathsf{IEn}(e, x) \to X/\bot$:

1. Parse $\left(X_1^0, X_1^1, \ldots, X_n^0, X_n^1\right) \leftarrow e$.
2. For any $i \in [n]$ and any $b \in \{0, 1\}$, if $X_i^0 = X_i^1$ or $X_i^b \notin \{0, 1\}^k$ output $\bot$.
3. Otherwise output $X = (X_1^{x_1}, \ldots, X_n^{x_n})$.

$\mathsf{IDe}(d, Z) \to z/\bot$:

1. Parse $\left(Z_{w-m+1}^0, Z_{w-m+1}^1, \ldots, Z_w^0, Z_w^1\right) \leftarrow d$ and $(Z_{w-m+1}, \ldots, Z_w) \leftarrow Z$.
2. For $i = 1$ to m:
   (a) If $Z_{w-m+i} = Z_{w-m+i}^0$ let $z_i = 0$.
   (b) If $Z_{w-m+i} = Z_{w-m+i}^1$ let $z_i = 1$.
   (c) Else output $\bot$.
3. Output $z \leftarrow (z_1, \ldots, z_m)$.

$\mathsf{IOp}(o, x, I) \to O/\bot$:

1. Parse $\left(\mathcal{O}_{K_1^0}, \mathcal{O}_{K_1^1}, \ldots, \mathcal{O}_{K_n^0}, \mathcal{O}_{K_n^1}\right) \leftarrow o$.
2. If $I \not\subseteq [n]$ or $x \notin \{0, 1\}^n$ output $\bot$.
3. Otherwise output $O = \left\{\left(i, \mathcal{O}_{K_i^{x_i}}\right)\right\}_{i\in I}$.

$\mathsf{IVe}(v, X_i, O_i, i, x_i) \to \top/\bot$:

1. Parse $((\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n), \boldsymbol{w}_\Delta, \boldsymbol{b}) \leftarrow v$.
2. If $i \notin [n]$ output $\bot$.
3. Parse $K_i^{x_i} \leftarrow X_i$ and $\mathcal{O}_{K_i^{x_i}} \leftarrow O_i$.
4. If $\mathsf{ComVer}\left(\mathcal{O}_{K_i^{x_i}}, \boldsymbol{b}, \boldsymbol{w}_i, \boldsymbol{w}_\Delta\right) = K_i^{x_i}$ output $\top$. Else output $\bot$.

[a] Here each key X_i is evaluated on the $2\beta + 1$ wire authenticators that have been created for each input gate of C.

Fig. 10. Algorithms for the interactive garbling scheme IGarb_π.

B ECC commitments Details

In this section we present the details of our ECC commitments. The full setup specification is described in Figure 11 and Figure 12 while methods for opening and checking a commitment (or an XOR of commitments) is described in Figure 13. Informally the setup and committing part of the scheme can be described as follows, assuming Alice is the committer and Bob the receiver. We let $\mathcal{C} = [\Gamma, k, d]$ be a binary, systematic, linear ECC. The construction of the ECC commitments consists of three phases; first a generic **Setup** step where Alice and Bob only need to know the amount of wires γ they wish to commit to. After the **Setup** step, for any $j \in [\gamma]$ and $i \in [\Gamma]$, Alice has the shares $\boldsymbol{s}_j^0, \boldsymbol{s}_j^1 \in \{0,1\}^\Gamma$ and Bob knows the entry $w_j^i = \boldsymbol{s}_j^{b_i}[i]$, where b_i is his choice-bits for the $\mathcal{F}_{\mathsf{OT}}$. We call $\boldsymbol{w}_j = \left(w_j^1, \ldots, w_j^\Gamma\right)$ the watch-bits of Bob and their role is to enable Bob to verify openings sent from Alice.

Protocol for setup and corrections of ECC commitments in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model

Alice and Bob have common input $\gamma \in \mathbb{N}$, the number of values Alice wants to commit to. We also assume that they agree on the variables Γ, k, s and a binary, systematic and linear ECC $\mathcal{C} = [\Gamma, k, d]$.

Setup:

1. Alice samples Γ pairs of uniformly random seeds of k bits each. Call these pairs $\left(r_i^0, r_i^1\right)$ for $i \in [\Gamma]$, such that $r_i^b \in_R \{0,1\}^k$ for $b \in \{0,1\}$.
2. Bob now selects Γ random bits, call these $b_i \in \{0,1\}$ for $i \in [\Gamma]$.
3. Alice and Bob perform Γ calls to $\mathcal{F}_{\mathsf{OT}}$ where Alice inputs $\left(\mathsf{sid}, i, \mathsf{sender}, r_i^0, r_i^1\right)$ and Bob inputs $(\mathsf{sid}, i, \mathsf{receiver}, b_i)$ and receives $r_i^{b_i}$ for $i \in [\Gamma]$.
4. Alice and Bob now use a PRG to extend each of the seeds they know into strings $\bar{r}_i^0$ and $\bar{r}_i^1$ of length γ. That is, $\bar{r}_i^0, \bar{r}_i^1 \in \{0,1\}^\gamma$ for $i \in [\Gamma]$. Thus Alice knows all of these strings and Bob knows only half, exactly the strings $\bar{r}_i^{b_i}$ for $i \in [\Gamma]$.
5. Now define the matrices $S^0, S^1 \in \{0,1\}^{\Gamma \times \gamma}$ such that the i'th row of S^b is $\bar{r}_i^b$ for $i \in [\Gamma]$ and $b \in \{0,1\}$. Define a column vector of these matrices as $\boldsymbol{s}_j^0$, respectively $\boldsymbol{s}_j^1$ for $j \in [\gamma]$.
6. For $j \in [\gamma]$, Alice lets $\boldsymbol{s}_j = \left(\boldsymbol{s}_j^0, \boldsymbol{s}_j^1\right)$. In a similar manner Bob lets $\boldsymbol{w}_j = \left(w_j^1, \ldots, w_j^\Gamma\right)$ and $\boldsymbol{b} = (b_1, \ldots, b_\Gamma)$ where $w_j^i = \boldsymbol{s}_j^{b_i}[i]$.
7. Define the output of Alice to be $\left\{\boldsymbol{s}_j\right\}_{j \in [\gamma]}$ and the output of Bob to be $\left(\left\{\boldsymbol{w}_j\right\}_{j \in [\gamma]}, \boldsymbol{b}\right)$.

Fig. 11. Protocol for setup and corrections of ECC commitments – part 1.

Secondly, a **Key Correct** step is carried out, in which Alice can choose to commit to some specific keys (the rest of the keys will remain pseudorandom). The reason for this step is that we wish to support the use of garbling schemes where the output key is derived by a function of the input keys. Since all keys are determined given the seeds for expansion, Alice cannot fully influence the keys output by the **Setup** step. Therefore the **Key Correct** step serves as a correction step for the gates' output keys in our protocol.

Finally, a **Codeword Correct** step is performed in which Alice is able to correct the last $\Gamma - k$ bits of all the expanded keys. This is to enable Alice to commit to messages with structure, in particular codewords of the code $\mathcal{C}$. The reason for having this as a separated step is the same as above, namely that the keys output by the **Setup** string cannot be fully controlled. The conclusion is that after the three steps have been carried out Alice holds shares that add up to a codeword of $\mathcal{C}$, and for some messages she has full control over what is encoded.

The linear combination check is described in Figure 15. After the three steps above have been completed, Bob is able to challenge Alice on a particular subset of the commitments. As previously mentioned, the high-level idea is, for the subset in question, that Alice constructs an opening to a random linear combination of the commitments and Bob verifies the validity using his watch-bits.

Protocol for setup and corrections of ECC commitments in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model

Alice and Bob have common input $\gamma \in \mathbb{N}$, the number of values Alice wants to commit to. We also assume that they agree on the variables Γ, k, s and a binary, systematic and linear ECC $\mathcal{C} = [\Gamma, k, d]$.

Key Correct:

On input $\left(\left\{\boldsymbol{s}_g\right\}_{g\in G}, \{\boldsymbol{u}_g\}_{g\in G}, G\right)$ for some $G \subseteq [\gamma]$ from Alice and input $\left(\left\{\boldsymbol{w}_g\right\}_{g\in G}, \boldsymbol{b}, G\right)$ from Bob do the following:

1. For $g \in G$, parse $(\boldsymbol{s}_g^0, \boldsymbol{s}_g^1) \leftarrow \boldsymbol{s}_g$. Let $\boldsymbol{k}_g^0$ and $\boldsymbol{k}_g^1$ denote the first k entries of $\boldsymbol{s}_g^0$ and $\boldsymbol{s}_g^1$, respectively.
2. Alice computes the set
$$\bar{G} = \left\{\bar{\boldsymbol{u}}_g = \boldsymbol{k}_g^0 \oplus \boldsymbol{k}_g^1 \oplus \boldsymbol{u}_g\right\}_{g\in G} ,$$
and sends this to Bob. In addition, for all $g \in G$, Alice updates her share $\boldsymbol{k}_g^1 := \boldsymbol{k}_g^0 \oplus \boldsymbol{u}_g$. If $|\bar{G}| \neq |G|$, Bob outputs $\perp$ and terminates.
3. For all $g \in G$, parse $\left(w_g^1, \ldots, w_g^\Gamma\right) \leftarrow \boldsymbol{w}_g$ and $(b_1, \ldots, b_\Gamma) \leftarrow \boldsymbol{b}$. For all $i \in [k]$ where $b_i = 1$, Bob updates his watch-bit $w_g^i := w_g^i \oplus \bar{\boldsymbol{u}}_g[i]$.
4. Define the output of Alice to be $\left(\left\{\boldsymbol{s}_g\right\}_{g\in G}\right) = \left(\left\{\boldsymbol{k}_g^0, \boldsymbol{k}_g^0 \oplus \boldsymbol{u}_g\right\}_{g\in G}\right)$ and the output of Bob to be the updated watch-bits $\left\{\boldsymbol{w}_g\right\}_{g\in G}$.

Codeword Correct:

On input $\left(\left\{\boldsymbol{s}_j\right\}_{j\in[\gamma]}, [\gamma]\right)$ from Alice and input $\left(\left\{\boldsymbol{w}_j\right\}_{j\in[\gamma]}, \boldsymbol{b}, [\gamma]\right)$ from Bob do the following:

1. For all $j \in [\gamma]$, parse $(\boldsymbol{s}_j^0, \boldsymbol{s}_j^1) \leftarrow \boldsymbol{s}_j$. Let $\boldsymbol{k}_j^0$ and $\boldsymbol{k}_j^1$ denote the first k entries of $\boldsymbol{s}_j^0$ and $\boldsymbol{s}_j^1$, respectively.
2. For all $j \in [\gamma]$, let $\boldsymbol{t}_j \leftarrow \mathcal{C}\left(\boldsymbol{k}_j^0 \oplus \boldsymbol{k}_j^1\right)$, *i.e.* the codewords of the XOR of each of pair of shares. As the code is systematic we have that the first k entries of $\boldsymbol{t}_j$ are equal to $\boldsymbol{k}_j^0 \oplus \boldsymbol{k}_j^1$. As the code has size Γ we denote by $\boldsymbol{v}_j$ the last $\Gamma - k$ bits of $\boldsymbol{t}_j$.
3. Alice lets C^0, respectively C^1 denote the $\{0,1\}^{(\Gamma-k)\times\gamma}$ matrix formed by viewing the last $\Gamma - k$ entires of $\boldsymbol{s}_j^0$, respective $\boldsymbol{s}_j^0 \oplus \boldsymbol{t}_j$ as its j'th column vector for $j \in [\gamma]$. In other words the matrices C^0 and C^1 are shares of the check-bits of each of the γ codewords. We denote these columns $\boldsymbol{c}_j^0$ and $\boldsymbol{c}_j^1$ respectively.
4. For all $j \in \gamma$ Alice lets $\bar{\boldsymbol{c}}_j$ be the last $\Gamma - k$ bits of $\boldsymbol{s}_j^0 \oplus \boldsymbol{s}_j^1 \oplus \boldsymbol{t}_j$. She then sends the set $\left\{\bar{\boldsymbol{c}}_j\right\}_{j\in[\gamma]}$ to Bob.
5. For all $j \in [\gamma]$ and $l \in [\Gamma - k]$ where $b_{k+l} = 1$, Bob updates $w_j^{k+l} := w_j^{k+l} \oplus \bar{\boldsymbol{c}}_j[l]$, thus defining his watch-bits of C^0 and C^1.
6. Define the output of Alice to be $\left\{\left(\left(\boldsymbol{k}_j^0, \boldsymbol{k}_j^1\right), \left(\boldsymbol{c}_j^0, \boldsymbol{c}_j^1\right)\right)\right\}_{j\in[\gamma]} = \left\{\left(\boldsymbol{k}_j, \boldsymbol{c}_j\right)\right\}_{j\in[\gamma]}$ and the output of Bob to be the updated watch-bits $\left\{\boldsymbol{w}_j\right\}_{j\in[\gamma]}$.

Fig. 12. Protocol for setup and corrections of ECC commitments – part 2.

In detail, the common input to the linear combination check is a set of sets $E = \{U_j\}_{j\in[t]}$ where $U_j = \{(u_l, v_l)\}_{l\in[e_j]}$ and where each tuple of the inner sets represent a wire. The value u_l is the index of the wire and v_l denotes if this is supposed to be a commitment to a 0- or 1-key. The counter e_j represents how many wires this commitment is an XOR of, meaning in the first two invocations of the linear combination check in our garbling protocol $e_j = 1$ for all j. However when checking the solderings this is not the case, since each of these consist of the XOR of at least two wires. The reason for this slightly heavy notation is to make the check versatile enough for our different use cases.

Jumping ahead to how the check is performed by our protocol, in the first invocation of our check procedure, $t = W + 1$, the total amount of "real" wires produced. As mentioned above all sets will be singletons $\{(u_1, v_1)\}$, meaning $e_j = 1$ for all j. In this case all the values v_l will be set to 0, as the goal here is to check each wire commitment (including the commitment to the Δ wire) individually. We also include a flag $\mathsf{bool} \in \{\mathsf{false}, \mathsf{true}\}$ which if set means that the opened linear combinations must be masked by a random value. This is because at this point in the protocol Bob is not allowed to learn linear combinations of wires

Methods for producing and verifying openings of ECC commitments

$\mathsf{ComOpen}\left(\left\{\boldsymbol{k}_j, \boldsymbol{c}_j, z_j\right\}_{j\in J}, \boldsymbol{k}_\Delta, \boldsymbol{c}_\Delta\right) \to \mathcal{O}_J$**:**

1. For $j \in J$, parse $\left(\boldsymbol{k}_j^0, \boldsymbol{k}_j^1\right) \leftarrow \boldsymbol{k}_j$, $\left(\boldsymbol{c}_j^0, \boldsymbol{c}_j^1\right) \leftarrow \boldsymbol{c}_j$. Also $\left(\boldsymbol{k}_\Delta^0, \boldsymbol{k}_\Delta^1\right) \leftarrow \boldsymbol{k}_\Delta$ and $\left(\boldsymbol{c}_\Delta^0, \boldsymbol{c}_\Delta^1\right) \leftarrow \boldsymbol{c}_\Delta$.
2. To produce an opening to $\bigoplus_{j\in J} K_j^{z_j}$, compute

$$\boldsymbol{d}^0 = \bigoplus_{j\in J}\left(\boldsymbol{k}_j^0 \oplus \left(z_j \cdot \boldsymbol{k}_\Delta^0\right)\right) \ , \ \boldsymbol{d}^1 = \bigoplus_{j\in J}\left(\boldsymbol{k}_j^1 \oplus \left(z_j \cdot \boldsymbol{k}_\Delta^1\right)\right)$$
$$\boldsymbol{e}^0 = \bigoplus_{j\in J}\left(\boldsymbol{c}_j^0 \oplus \left(z_j \cdot \boldsymbol{c}_\Delta^0\right)\right) \ , \ \boldsymbol{e}^1 = \bigoplus_{j\in J}\left(\boldsymbol{c}_j^1 \oplus \left(z_j \cdot \boldsymbol{c}_\Delta^1\right)\right)$$

3. Then let $\mathcal{O}_J \leftarrow \left(\boldsymbol{d}^0, \boldsymbol{d}^1, \boldsymbol{e}^0, \boldsymbol{e}^1, \{z_j\}_{j\in J}\right)$ and output $\mathcal{O}_J$.

$\mathsf{ComVer}\left(\mathcal{O}_J, \boldsymbol{b}, \{\boldsymbol{w}_j\}_{j\in J}, \boldsymbol{w}_\Delta\right) \to K_J/\perp$**:**

1. Parse $\left(\boldsymbol{d}^0, \boldsymbol{d}^1, \boldsymbol{e}^0, \boldsymbol{e}^1, \{z_j\}_{j\in J}\right) \leftarrow \mathcal{O}_J$.
2. Parse $(b_1, \ldots, b_\Gamma) \leftarrow \boldsymbol{b}$.
3. For all $j \in J$, parse $\left(w_j^1, \ldots, w_j^\Gamma\right) \leftarrow \boldsymbol{w}_j$. Also $\left(w_\Delta^1, \ldots, w_\Delta^\Gamma\right) \leftarrow \boldsymbol{w}_\Delta$.
4. For $i \in [k]$ and $l \in [\Gamma - k]$, verify that

$$\boldsymbol{d}^{b_i}[i] = \bigoplus_{j\in J}\left(w_j^i \oplus \left(z_j \cdot w_\Delta^i\right)\right) \ , \ \boldsymbol{e}^{b_{k+l}}[l] = \bigoplus_{j\in J}\left(w_j^{k+l} \oplus \left(z_j \cdot w_\Delta^{k+l}\right)\right)$$

5. Compute $\boldsymbol{f} \leftarrow \mathcal{C}\left(\boldsymbol{d}^0 \oplus \boldsymbol{d}^1\right)$ and verify if the last $\Gamma - k$ bits of $\boldsymbol{f}$ are equal to $\boldsymbol{e}_0 \oplus \boldsymbol{e}_1$. If all of the verifications are true let $K_J = \boldsymbol{d}^0 \oplus \boldsymbol{d}^1$ and output K_J. Else output $\perp$.

Fig. 13. Methods for producing and verifying openings of ECC commitments.

Illustration of an ECC Commitment

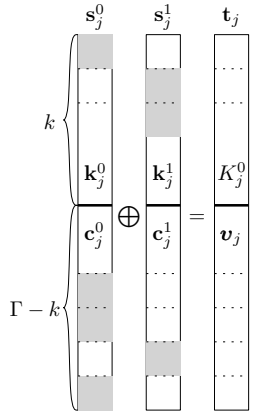

Fig. 14. Illustration of an ECC Commitment, after all key and codeword corrections have been done. The grey squares represent Bob's watchbits.

Protocol for Linear Combination Check

Alice and Bob have common input (E, bool) where $E = \{U_j\}_{j\in[t]} = \left\{\{(u_l, v_l)\}_{l\in[e_j]}\right\}_{j\in[t]}$ where t is the number of wires (or XOR of wires) to check, $e_j, u_l \in [W+1]$, $v_l \in \{0,1\}$ and $\mathsf{bool} \in \{\mathsf{false}, \mathsf{true}\}$. Thus E is a set consisting of t sets, U_j, and each U_j is a set consisting of e_j pairs in which one element is a wire index and the other a bit. In addition Alice has private input $(A, \boldsymbol{k}_\Delta, \boldsymbol{c}_\Delta)$ and Bob has private input $\left(B, \left\{K_j\right\}_{j\in[t]}, \boldsymbol{b}, \boldsymbol{w}_\Delta\right)$. First parse $\left\{\left(\boldsymbol{k}_{u_l}, \boldsymbol{c}_{u_l}\right)\right\}_{j\in[t]:(u_l,v_l)\in U_j} \leftarrow A$ and $\left\{\boldsymbol{w}_{u_l}\right\}_{j\in[t]:(u_l,v_l)\in U_j} \leftarrow B$. If $\mathsf{bool} = \mathsf{true}$, then let $r = 7.3(s+3)$ and $A := A \cup \{(\boldsymbol{k}_{W+1+i}, \boldsymbol{c}_{W+1+i})\}_{i\in[r]}$ and $B := B \cup \{\boldsymbol{w}_{W+1+i}\}_{i\in[r]}$.[a]

1. Bob samples a uniformly random binary matrix, V, of dimension $r \times t$ and sends this to Alice.
   For all $i \in [r]$, let $D_i = \{(u_l, v_l)\}_{j\in[t]\wedge V_{i,j}=1:(u_l,v_l)\in U_j}$ and if $\mathsf{bool} = \mathsf{true}$ let $D_i := D_i \cup \{(W+1+i, 0)\}$. Then
   - Alice computes:
   $$\mathcal{O}_i \leftarrow \mathsf{ComOpen}\left(\left\{\left(\boldsymbol{k}_{u_l}, \boldsymbol{c}_{u_l}, v_l\right)\right\}_{(u_l,v_l)\in D_i}, \boldsymbol{k}_\Delta, \boldsymbol{c}_\Delta\right)$$
   and sends this to Bob.
   - Bob then computes:
   $$K_i \leftarrow \mathsf{ComVer}\left(\mathcal{O}_i, \boldsymbol{b}, \left\{\boldsymbol{w}_{u_l}\right\}_{(u_l,v_l)\in D_i}, \boldsymbol{w}_\Delta\right)$$
   $$K_i' = \bigoplus_{j\in[t]:V_{i,j}=1} K_j$$
   - If $\mathsf{bool} = \mathsf{true}$ Bob checks that $K_i \neq \perp$. If $\mathsf{bool} = \mathsf{false}$ he also checks that $K_i = K_i'$. If any of the checks fail, Bob outputs $\perp$.
2. If none of the r checks failed, Bob outputs $\top$.

[a] These extra r wires are used to blind the opened to linear combinations with a random one-time value. However this is only necessary for the first LCC check performed, so we parameterize this operation with bool.

Fig. 15. Protocol for Linear Combination Check.

that are to be used later in the protocol. In the following two invocations he already knows the individual keys, so there is no issue in revealing the linear combination in these cases.

The next invocation of the check is in the cut-and-choose step, where the tuples again will be singletons, but the values v_1 will correspond to the input challenge bits chosen by Bob. The final invocation is for the soldering step of our protocol and here the sets are not singletons, but will contain a tuple for each wire that's included in the specific soldering. Again, because solderings are constructed as the XOR of the 0-keys all values v_l are set to 0 here.

C Interactive Garbling Scheme is sufficient for UC-secure 2PC

In this section we state the formal security properties of an interactive garbling scheme in Figure 16 and Figure 17. Next we show that the generic protocol π_{IGCO} described in Figure 1 UC-implements the functionality $\mathcal{F}_{\mathsf{SFE}}$ of Figure 21 in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. This is captured by Lemma 2 and Lemma 3 below.

Defining Security. In defining security we will require the existence of some auxiliary algorithms. For clarity we will consider them part of an extended scheme. An *extended interactive garbling scheme* has the form $\mathcal{G}_\pi = (\mathsf{Gb}_\pi, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}, \mathsf{Op}, \mathsf{Ve}, \mathsf{Ex}_\mathsf{C}, \mathsf{Ex}_\mathsf{E}, \mathsf{De}^{-1}, \mathsf{En}^{-1})$. Here Ex_C is a deterministic poly-time algorithm called the *constructor extractor*. After a run of Gb_π between C and E, where C might deviate from the protocol, it is applied to the view of C, i.e., inputs $(1^k, f)$ of C plus the messages sent to the ideal functionalities of Gb_π by C and the messages sent to C by the ideal functionalities. It outputs $(\hat{e}, \hat{d})$. The intuition is that $\hat{e}$ is a well-formed encoding function and that $\hat{d}$ is a well-formed decoding function. We call $\hat{e}$ the *implicit input encoding function* and we call $\hat{d}$ the *implicit output decoding function*. The reason is that we will sometimes need that even a cheating constructor knows well-defined encoding and decoding functions. The *evaluator extractor* Ex_E works the same way but is applied to the view of a possibly cheating E and it outputs an *implicit garbled function* $\hat{F}$. As we discuss later, we sometimes need that even a cheating evaluator knows a well-defined garbled function. The deterministic poly-time algorithm En^{-1} is called the *de-encoder*. It takes as input the encoding function e and an encoded input X. It outputs an input x, which is supposed to be the x encoded by X. It is used to guarantee that even a malicious constructor has a well-defined input. The deterministic poly-time algorithm De^{-1} is called the *de-decoder*. It takes as input the decoding function d and an output z. It outputs an encoded input Z. For now, simply think of it as the inverse of the decoding algorithm.

We will now define security notions of an extended scheme. Each security notion is defined via a game, $\mathrm{Game}^{\mathcal{A}}_{\mathcal{G}_\pi}$, between an extended scheme $\mathcal{G}_\pi$ and an adversary $\mathcal{A}$. If the game outputs $\top$ it means that $\mathcal{A}$ won. If the game outputs $\bot$ it means that $\mathcal{A}$ lost. If the name of the game defining property prop contains the sub-string Ind, then we say that the game is indistinguishability based, and we define the advantage as follows $\mathbf{Adv}^{\mathrm{prop}}_{\mathcal{G}_\pi}(\mathcal{A}, k) = 2\Pr[\mathrm{Game}^{\mathcal{A}}_{\mathcal{G}_\pi}(1^k) = \top] - 1$. Otherwise, we define the advantage as $\mathbf{Adv}^{\mathrm{prop}}_{\mathcal{G}_\pi}(\mathcal{A}, k) = \Pr[\mathrm{Game}^{\mathcal{A}}_{\mathcal{G}_\pi}(1^k) = \top]$. In both cases we say that $\mathcal{G}_\pi$ has the property prop if it holds for all PPT adversaries $\mathcal{A}$ that $\mathbf{Adv}^{\mathrm{prop}}_{\mathcal{G}_\pi}(\mathcal{A}, k)$ is negligible in k.

Game $\mathrm{OblIndAct}^{\mathcal{B}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** obl.ind.act.

1. Run $\mathcal{B}(1^k)$ to produce (f, x_0, x_1, I).
2. If $\{x_0, x_1\} \not\subseteq \{0,1\}^{f.n}$ or $I \not\subseteq [f.n]$ or $x_{0,i} \neq x_{1,i}$ for $i \in I$, then output $\bot$.
3. Sample uniformly random $b \in_R \{0,1\}$.
4. Run Gb_π between $\mathsf{C}\left(1^k, f\right)$ and $\mathcal{B}$. If C outputs $\bot$, output $\bot$. Otherwise, denote the output of C by (F, e, d, o).
5. Compute $X \leftarrow \mathsf{En}(e, x_b)$.
6. Compute $O \leftarrow \mathsf{Op}(o, x_b, I)$.
7. Give (X, O) as input to $\mathcal{B}$ and run $\mathcal{B}$ to get an output b'.
8. If $b' = b$, then output $\top$, otherwise output $\bot$.

Game $\mathrm{Corr}^{\mathcal{A}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** corr.

1. Run $\mathcal{A}(1^k)$ to produce (f, x, I).
2. If $x \notin \{0,1\}^{f.n}$ or $I \not\subseteq [f.n]$, then output $\bot$.
3. Run Gb_π between $\mathsf{C}\left(1^k, f\right)$ and $\mathsf{E}\left(1^k, f\right)$. If any party outputs $\bot$, then output $\top$. Otherwise, denote the output of C by (F, e, d, o) and the output of E by (F', v). If $F' \neq F$, then output $\top$.
4. Compute $X \leftarrow \mathsf{En}(e, x)$.
5. Compute $O \leftarrow \mathsf{Op}(o, x, I)$.
6. If $\mathsf{Ve}\left(v, O, X, \{(i, x_i)\}_{i \in I}\right) = \bot$, output $\top$.
7. Compute $Z \leftarrow \mathsf{Ev}(F, X)$.
8. If $\mathsf{De}(d, Z) \neq f(x)$, then output $\top$, otherwise output $\bot$.

Game $\mathrm{AutAct}^{\mathcal{B}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** aut.act.

1. Run $\mathcal{B}(1^k)$ to produce (f, x, I).
2. If $x \notin \{0,1\}^{f.n}$ or $I \not\subseteq [f.n]$, then output $\bot$.
3. Run Gb_π between $\mathsf{C}\left(1^k, f\right)$ and $\mathcal{B}$. If C outputs $\bot$, output $\bot$. Otherwise, denote the output of C by (F, e, d, o).
4. Compute $X \leftarrow \mathsf{En}(e, x)$.
5. Compute $O \leftarrow \mathsf{Op}(o, x, I)$.
6. Compute $Z \leftarrow \mathsf{Ev}(F, X)$.
7. If $\mathsf{De}(d, Z) \neq f(x)$, then output $\top$.
8. Give (X, O) as input to $\mathcal{B}$ and run $\mathcal{B}$ to get an output Z'.
9. If $\mathsf{De}(d, Z') \neq \bot$ and $Z' \neq Z$, then output $\top$, otherwise output $\bot$.

Game $\mathrm{TokCom}^{\mathcal{A}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** tok.com.

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}\left(1^k, f\right)$. If E outputs $\bot$, output $\bot$. Otherwise E outputs some (F, v).
3. Run Exc on $\mathcal{A}$ to produce $\hat{e}$ and $\hat{d}$.
4. Run $\mathcal{A}$ to produce X, O, I and $\{(i, x'_i)\}_{i \in I}$. If $I \not\subseteq [f.n]$, then output $\bot$.
5. Compute $x \leftarrow \mathsf{En}^{-1}(\hat{e}, X)$. If $x = \bot$, then output $\bot$.
6. If $\mathsf{Ve}\left(v, X, O, \{(i, x'_i)\}\right)_{i \in I}) = \top$ and there exists $i \in I$ such that $x_i \neq x'_i$, then output $\top$, otherwise output $\bot$.

Fig. 16. Security games for interactive garbling scheme – part 1

Game $\mathrm{KnoF}^{\mathcal{B}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** knof.

1. Run $\mathcal{B}(1^k)$ to produce f.
2. Run Gb_π between $\mathsf{C}\left(1^k, f\right)$ and $\mathcal{B}$. If C outputs $\perp$, output $\perp$. Otherwise, denote the output of C by (F, e, d, o).
3. Run Ex_E on $\mathcal{B}$ to compute $\hat{F}$.
4. If $\hat{F} \neq F$, then output $\top$, otherwise output $\perp$.

Game $\mathrm{RobCon}^{\mathcal{A}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** rob.con.

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}\left(1^k, f\right)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F, v).
3. Run Ex_C on $\mathcal{A}$ to produce $\hat{e}$ and $\hat{d}$.
4. Run $\mathcal{A}$ to produce x. If $x \notin \{0,1\}^{f.n}$, then output $\perp$.
5. If $f(x) \neq \mathsf{De}(\hat{d}, \mathsf{Ev}(F, \mathsf{En}(\hat{e}, x)))$, then output $\top$, otherwise output $\perp$.

Game $\mathrm{UnqIE}^{\mathcal{A}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** unqie.

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}\left(1^k, f\right)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F, v).
3. Run Ex_C on $\mathcal{A}$ to produce $\hat{e}$ and $\hat{d}$.
4. Run $\mathcal{A}$ to produce X, O, I and $\{(i, x'_i)\}_{i \in I}$. If $I \not\subseteq [f.n]$, then output $\perp$.
5. If $\mathsf{En}(\hat{e}, \mathsf{En}^{-1}(\hat{e}, X)) \neq X$ and $\mathsf{Ve}\left(v, X, O, \{(i, x'_i)\}\right)_{i \in I}) = \top$ and $\mathsf{Ev}(F, X) \neq \perp$, then output $\top$, otherwise output $\perp$.

Game $\mathrm{UnqOE}^{\mathcal{A}}_{\mathcal{G}_\pi}\left(1^k\right)$. **Property** unqoe.

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}\left(1^k, f\right)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F, v).
3. Run Ex_C on $\mathcal{A}$ to produce $\hat{e}$ and $\hat{d}$.
4. Run $\mathcal{A}$ to produce X.
5. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \perp$, then output $\perp$.
6. If $\mathsf{De}^{-1}(\hat{d}, \mathsf{De}(\hat{d}, Z)) \neq Z$, then output $\top$, otherwise output $\perp$.

Fig. 17. Security games for interactive garbling scheme – part 2

Lemma 2. *If $\mathcal{G}_\pi$ is a projective extended interactive garbling scheme and has the properties, obl.ind.act, aut.act and knof, then π_{IGCO} UC securely realizes $\mathcal{F}_{\mathsf{SFE}}^f$ against a static and malicious corruption of B.*

Proof. If B is corrupted and A is honest, then the simulator $\mathcal{T}$ proceeds as follows. Use $\mathcal{B}$ to denote the adversary controlling B. We can assume without loss of generality that this is the UC environment.

1. The simulator $\mathcal{T}$ runs a copy of the protocol and lets it interact with $\mathcal{B}$ as in the real world. In particular, it simulates the ideal functionalities to $\mathcal{B}$ by running them honestly.
2. Since $\mathcal{T}$ does not know the input of A it instead uses the *dummy input* $x' = 0^{n_\mathsf{A}}$ for A.
3. In **Input B, I**, $\mathcal{T}$ inspects the OT's to learn the choice bits $y_1, \ldots, y_{n_\mathsf{B}}$ of $\mathcal{B}$ and defines $y = y_1 \cdots y_{n_\mathsf{B}}$.
4. Now $\mathcal{T}$ runs according to the protocol, until it receives some Z' from $\mathcal{B}$.
5. Apply the algorithm Ex_E to compute from the communication of $\mathcal{B}$ a garbled function F'. If $Z' \neq \mathsf{Ev}(F', X)$, then $\mathcal{T}$ inputs abort to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of B.
6. Then $\mathcal{T}$ inputs y to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of B. As a result $\mathcal{F}_{\mathsf{SFE}}$ outputs $z \leftarrow f(x, y)$ on behalf of A.

We show that the simulation is indistinguishable from the real-world to $\mathcal{B}$ via a hybrid argument.

Consider the first hybrid where we replace Step 2 by this:

2^1. We let $\mathcal{T}$ cheat and inspect $\mathcal{F}_{\mathsf{SFE}}^f$ to learn x. Then it finishes as in the simulation except that it uses the real input x of A instead of the dummy input x'.

It is straightforward to show that if $\mathcal{B}$ can guess with probability p whether it is in the simulation or the hybrid, then we can use $\mathcal{B}$ to win $\mathrm{OblIndAct}_{\mathcal{G}_\pi}^{\mathcal{B}}(1^k)$ with advantage p. We output (f, x_0, x_1, I), where $x_0 = xy$ and $x_1 = x'y$ (where $x' = 0^{n_\mathsf{A}}$) and $I = \{n_\mathsf{A} + 1, \ldots, n_\mathsf{A} + n_\mathsf{B}\}$. Then we let $\mathcal{B}$ participate in Gb_π in $\mathrm{OblIndAct}_{\mathcal{G}_\pi}^{\mathcal{B}}(1^k)$. If C aborts the run of Gb_π we let A terminate with output abort as it would in the protocol and then terminate the simulation and make a random guess in the reduction. Otherwise we are given (X, O) and we parse X as $\left(X_1, \ldots, X_{n_\mathsf{A}}, Y_1^{y_1}, \ldots, Y_{n_\mathsf{B}}^{y_{n_\mathsf{B}}}\right)$ and we parse O as $\left(O_1^{y_1}, \ldots, O_{n_\mathsf{B}}^{y_{n_\mathsf{B}}}\right)$. Then we let $\mathcal{T}$ send $(X_1, \ldots, X_{n_\mathsf{A}})$ to $\mathcal{B}$ in **Input A** and in **Input B, II** we input $(Y_i^{y_i}, O_i^{y_i}, Y_i^{y_i}, O_i^{y_i})$ to $\mathcal{F}_{\mathsf{DOT}}^i$. Then finish the execution by running as in the simulation and then output whatever bit $\mathcal{B}$ outputs.

Consider then the second hybrid where we replace Step 5 by this:

5^1. Let (F, e, d, o) denote the output to C in the run of Gb_π. If $Z' \neq \mathsf{Ev}(F, X)$, then $\mathcal{T}$ inputs abort to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of B.

By the security property, knof, the first and second hybrids are indistinguishable to $\mathcal{B}$. The reduction is trivial. Consider then the third hybrid where we replace Step 5 and Step 6 by this:

5^2. Let (F, e, d, o) denote the output to C in the run of Gb_π. If $\mathsf{De}(d, Z') = \bot$, then $\mathcal{T}$ inputs abort to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of B.

6^1. Input y to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of B. As a result $\mathcal{F}_{\mathsf{SFE}}$ computes $z \leftarrow f(x, y)$. Then cheat and replace z by $z \leftarrow \mathsf{De}(d, Z')$, i.e., let $\mathcal{F}_{\mathsf{SFE}}$ output $z \leftarrow \mathsf{De}(d, Z')$ on behalf of A.

At this point the values (F, e, d, o) and Z' are generated exactly as in game AutAct. It therefore follows from the property aut.act that the probability that $\mathsf{De}(d, Z') = \bot$ is negligibly close to the probability that $Z' \neq \mathsf{Ev}(F, X)$, and hence the change to Step 5 is indistinguishable to $\mathcal{B}$. When $Z' = \mathsf{Ev}(F, X)$ it also follows from the property aut.act that the probability that $\mathsf{De}(d, Z') \neq f(x, y)$ is negligible and hence the change to Step 6 is indistinguishable.

Note then that the values (F, e, d, o) and Z' now are computed in the same way in the third hybrid and in the generic protocol. Furthermore, the output of $\mathcal{F}_{\mathsf{SFE}}$ is patched to be $\mathsf{De}(d, Z')$ and in the protocol A also outputs $\mathsf{De}(d, Z')$. Hence the third hybrid is perfectly indistinguishable from the protocol in the view of $\mathcal{B}$. This concludes the proof. □

Lemma 3. *If $\mathcal{G}_\pi$ is a projective extended interactive garbling scheme and has the properties, rob.con, unqoe, unqie and tok.com, then π_{IGCO} UC securely realizes $\mathcal{F}_{\mathsf{SFE}}^f$ against a static and malicious corruption of A.*

Proof. If A is corrupted and B is honest, then the simulator $\mathcal{S}$ proceeds as follows. Use $\mathcal{A}$ to denote the adversary controlling A. We can assume without loss of generality that this is the UC environment.

1. In **Garbling**, $\mathcal{S}$ simulates the ideal functionalities of π_{IGCO} to $\mathcal{A}$ by running them honestly, and it participates honestly in π_{IGCO} on behalf of B unless another behaviour is specified below.
2. If B aborts in the garbling, then abort A (i.e., input abort to the ideal functionality on behalf of A). Otherwise, let (F, v) denote the output to B.
3. Then $\mathcal{S}$ applies Ex_{C} to the communication of $\mathcal{A}$ in Gb_{π} to extract $\hat{e}$ and $\hat{d}$. It parses $\hat{e}$ as $\left(\hat{X}_1^0, \hat{X}_1^1, \ldots, \hat{X}_{n_{\mathsf{A}}}^0, \hat{X}_{n_{\mathsf{A}}}^1, \hat{Y}_1^0, \hat{Y}_1^1, \ldots, \hat{Y}_{n_{\mathsf{B}}}^0, \hat{Y}_{n_{\mathsf{B}}}^1\right)$.
4. Let $(X_1, \ldots, X_{n_{\mathsf{A}}})$ be the value sent by $\mathcal{A}$ in **Input A**. We call an index i *bad* if $X_i \notin \left\{\hat{X}_i^0, \hat{X}_i^1\right\}$ or $\hat{X}_i^0 = \hat{X}_i^1$. If there is a bad index, then let $x = 0^{n_{\mathsf{A}}}$. Otherwise, let each x_i be the unique bit such that $X_i = \hat{X}_i^{x_i}$, and then let $x = x_1 \cdots x_{n_{\mathsf{A}}}$. Notice that $(X_1, \ldots, X_{n_A}) = \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_{\mathsf{A}}}^{x_{n_{\mathsf{A}}}}\right)$ if there are no bad indices.
5. In **Input B, II**, $\mathcal{S}$ inspects the OT's to learn the messages $\left(\left(Y_i^0, O_i^0\right), \left(Y_i^1, O_i^1\right)\right)$ input to $\mathcal{F}_{\mathsf{DOT}}^i$ by $\mathcal{A}$. We call (i, b) a *faulty position* if $Y_i^b \neq \hat{Y}_i^b$ or $\mathsf{Ve}\left(Y_i^b, O_i^b, i, b\right) = \bot$. We call an index i *double faulty* if $(i, 0)$ and $(i, 1)$ are both faulty. We call an index i *correct* if neither $(i, 0)$ nor $(i, 1)$ is faulty. We call an index i *single faulty* if it is not double faulty nor correct. If there is a double faulty index i, then abort A (i.e., input abort to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A) and terminate the simulated protocol. Otherwise, let I be the set of indices that are single faulty, and for each $i \in I$ let β_i be the unique bit for which (i, β_i) is not faulty. If $|I| > 0$, then input $\{(i, \beta_i)\}_{i \in I}$ to $\mathcal{F}_{\mathsf{SFE}}^f$. If the output is abort, then terminate the simulated protocol. Otherwise, define a *dummy input* y' for B by letting $y'_i = \beta_i$ for $i \in I$ and $y'_i = 0$ for $i \notin I$. Then let $X = \left(X_1, \ldots, X_{n_{\mathsf{A}}}, Y_1^{y'_1}, \ldots, Y_{n_{\mathsf{B}}}^{y'_{n_{\mathsf{B}}}}\right)$ and $O = \left(O_1^{y'_1}, \ldots, O_{n_{\mathsf{B}}}^{y'_{n_{\mathsf{B}}}}\right)$. If $\mathsf{Ve}\left(v, X, O, \{(i, y'_i)\}_{i=1}^{n_{\mathsf{B}}}\right) = \bot$, then abort A (i.e., input abort to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A) and terminate the simulated protocol.
6. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \bot$, then abort A. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z \leftarrow f(x, y)$. Then compute $Z' \leftarrow \mathsf{De}^{-1}(\hat{d}, z)$ and send Z' to $\mathcal{A}$ as if coming from B.

We show that the simulation and the protocol are indistinguishable to $\mathcal{A}$ using a hybrid argument. Define a first hybrid where we replace Step 5 by this:

5^1. In **Input B, II**, $\mathcal{S}$ inspects the OT's to learn the messages $\left(\left(Y_i^0, O_i^0\right), \left(Y_i^1, O_i^1\right)\right)$ input to $\mathcal{F}_{\mathsf{DOT}}^i$ by $\mathcal{A}$. It then inspects $\mathcal{F}_{\mathsf{SFE}}^f$ to get the real value y of the input of B, as given by the environment. Define y' by letting $y' = y$. Then run as in the simulation, but with input y' for B, i.e., let $X = \left(X_1, \ldots, X_{n_{\mathsf{A}}}, Y_1^{y'_1}, \ldots, Y_{n_{\mathsf{B}}}^{y'_{n_{\mathsf{B}}}}\right)$ and $O = \left(O_1^{y'_1}, \ldots, O_{n_{\mathsf{B}}}^{y'_{n_{\mathsf{B}}}}\right)$, and if $\mathsf{Ve}\left(v, X, O, \{(i, y'_i)\}_{i=1}^{n_{\mathsf{B}}}\right) = \bot$, then abort A and terminate the simulated protocol. If any (i, y'_i) is faulty, then send "$\mathsf{you\ are\ in\ a\ hybrid}$" to the adversary if this was not already done.

The simulation and the hybrid are indistinguishable to the adversary. To see this, first observe the fact that if some (i, y'_i) is faulty in the hybrid, it will abort except with negligible probability. To see this, notice that if (i, y'_i) is faulty, then $Y_i^{y'_i} \neq \hat{Y}_i^{y'_i}$ or $\mathsf{Ve}\left(Y_i^{y'_i}, O_i^{y'_i}, i, y'_i\right) = \bot$. If $\mathsf{Ve}\left(Y_i^{y'_i}, O_i^{y'_i}, i, y'_i\right) = \bot$, then $\mathsf{Ve}\left(v, X, O, \{(i, y'_i)\}_{i=1}^{n_{\mathsf{B}}}\right) = \bot$ by definition and the hybrid aborts. Therefore, assume that (i, y'_i) is faulty and that $\mathsf{Ve}\left(Y_i^{y'_i}, O_i^{y'_i}, i, y'_i\right) = \top$. Then a simple reduction to the property tok.com shows that $Y_i^{y'_i} \in \left\{\hat{Y}_i^0, \hat{Y}_i^1\right\}$ (or $\mathsf{En}^{-1}(\hat{e}, X) = \bot$) and that $Y_i^{y'_i} \neq \hat{Y}_i^{1-y'_i}$, or $\mathsf{Ve}\left(\hat{Y}_i^{1-y'_i}, O_i^{y'_i}, i, y'_i\right) = \top$ could be used to win in Step 6 of $\mathrm{TokCom}_{\mathcal{G}_\pi}^{\mathcal{A}}$. Hence $Y_i^{y'_i} = \hat{Y}_i^{y'_i}$, contradicting that (i, y'_i) is faulty. Then observe that there are only four changes between the simulation and the hybrid. First of all, in the simulation we explicitly abort if there is a double faulty index. This we no longer do in the hybrid. This makes no difference, however, by the above fact. Second, in the simulation we abort if $\beta_i \neq y_i$. It follows from the above fact that we do the same in the

hybrid, as $(i, 1-\beta_i)$ is a faulty position. Third, in the hybrid we use a different y'. This is indistinguishable, as the view of the adversary does not depend on y' at all when the protocol aborts, as argued above, and when the protocol does not abort, then the reply in both distributions is $Z \leftarrow \mathsf{De}^{-1}(\hat{d}, z)$ which is identically distributed in the simulation and the hybrid. Finally we send the string "you are in a hybrid" to the adversary at the end if any (i, y'_i) is faulty, but by the above fact we have already aborted at this point if any (i, y'_i) is faulty, so this change is indistinguishable.

Then make the following change to Step 5:

5^2. Run as Step 5^1, but with this addition at the end: if the protocol did not abort and $\mathsf{En}^{-1}(\hat{e}, X) \neq xy'$ and $\mathsf{Ev}(F, X) \neq \perp$, then send "you are in a hybrid" to the adversary if this was not already done.

Notice that at the point where the addition is made it holds that if the protocol did not abort, then $\mathsf{Ve}(v, X, O, \{(i, y'_i)\}_{i=1}^{n_\mathsf{B}}) = \top$ (or we would have aborted in Step 5^1). Furthermore, we only send the string if $\mathsf{Ev}(F, X) \neq \perp$. Below we argue that when we send the string then it also holds that $\mathsf{En}(\hat{e}, \mathsf{En}^{-1}(\hat{e}, X)) \neq X$. I.e., if we send that string, then $\mathsf{En}(\hat{e}, \mathsf{En}^{-1}(\hat{e}, X)) \neq X$ and $\mathsf{Ve}(v, X, O, \{(i, y'_i)\}_{i=1}^{n_\mathsf{B}}) = \top$ and $\mathsf{Ev}(F, X) \neq \perp$. So, a simple reduction lets us win the game for property unqie with the probability with which "you are in a hybrid" is sent to the adversary in 5^2 Hence it is sent with negligible probability and the hybrids are indistinguishable. We prove that $\mathsf{En}(\hat{e}, \mathsf{En}^{-1}(\hat{e}, X)) \neq X$ by a case analysis. Assume first that there is a bad index. If there is a bad index, then $\mathsf{En}(\hat{e}, xy') \neq X$ by definition of En being the projective encoding algorithm. So, since $\mathsf{En}(\hat{e}, \perp) = \perp \neq X$ and by construction $\mathsf{En}^{-1}(\hat{e}, X) \in \{\perp, xy'\}$, it follows that $\mathsf{En}(\hat{e}, \mathsf{En}^{-1}(\hat{e}, X)) \neq X$ if there is a bad index. Assume then that there is no bad index. If we send the string in Step 5^2, then there are no faulty positions (i, y'_i) either, as we would have sent the string already in Step 5^1 When there are no bad index and no faulty positions (i, y'_i), then $(X_1, \ldots, X_{n_A}) = \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}}\right)$ and all $Y_i^{y'_i} = \hat{Y}_i^{y'_i}$, so $X = \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}}, \hat{Y}_1^{y'_1}, \ldots, \hat{Y}_{n_\mathsf{B}}^{y'_{n_\mathsf{B}}}\right) = \mathsf{En}(\hat{e}, xy')$. This contradicts that $\mathsf{En}^{-1}(\hat{e}, X) \neq xy'$.

Then define the next hybrid where we replace Step 4 with this:

4^1. Let $(X_1, \ldots, X_{n_\mathsf{A}})$ be the value sent by $\mathcal{A}$ in **Input A**. We call an index i *bad* if $X_i \notin \left\{\hat{X}_i^0, \hat{X}_i^1\right\}$ or $\hat{X}_i^0 = \hat{X}_i^1$. If there is a bad index and $\mathsf{Ev}(F, X) = \perp$, then abort A. If there is a bad index and $\mathsf{Ev}(F, X) \neq \perp$, then let $x = 0^{n_\mathsf{A}}$. Otherwise, let each x_i be the unique bit such that $X_i = \hat{X}_i^{x_i}$, and then let $x = x_1 \cdots x_{n_\mathsf{A}}$. If $(X_1, \ldots, X_{n_A}) \neq \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}}\right)$, then send "you are in a hybrid" to the adversary, if this was not already done.

As for the change *If there is a bad index and $\mathsf{Ev}(F, X) = \perp$, then abort A*, notice that we would abort in Step 6 anyway when $\mathsf{Ev}(F, X) = \perp$, so this changes nothing in the view of the adversary. For the change *If $(X_1, \ldots, X_{n_A}) \neq \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_A}^{x_{n_A}}\right)$, then send "you are in a hybrid" if not already done* notice that $(X_1, \ldots, X_{n_A}) = \left(\hat{X}_1^{x_1}, \ldots, \hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}}\right)$ if there is no bad index, so if we send the string, there is a bad index. Furthermore, if there is a bad index and $\mathsf{Ev}(F, X) = \perp$, then we aborted. So, if we send the string, then there is a bad index and $\mathsf{Ev}(F, X) \neq \perp$. When there is a bad index, then $\mathsf{En}^{-1}(\hat{e}, X) = \perp \neq xy'$ and so since $\mathsf{Ev}(F, X) \neq \perp$, we would have sent the string in Step 5^2 anyway, so this changes nothing.

Then define the next hybrid where we replace Step 6 with this:

6^1. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \perp$, then abort. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z \leftarrow f(x, y)$. If $\mathsf{De}(\hat{d}, Z) \neq z$, then send "you are in a hybrid" to $\mathcal{A}$, if not already done. Then compute $Z' \leftarrow \mathsf{De}^{-1}(\hat{d}, z)$ and send Z' to $\mathcal{A}$.

If we send the string "you are in a hybrid", then it was not sent by the previous changes and hence $X = \mathsf{En}(\hat{e}, xy)$. At the point where we send the string we know that $\mathsf{Ev}(F, X) \neq \perp$ and $z \leftarrow f(x, y)$. So, when we send the string we know that $\mathsf{Ev}(F, \mathsf{En}(\hat{e}, xy)) \neq \perp$ and yet $\mathsf{De}(\hat{d}, \mathsf{Ev}(F, \mathsf{En}(\hat{e}, xy))) \neq f(x, y)$. A simple reduction to rob.con therefore shows that the hybrids are indistinguishable.

Then define a new hybrid where we replace Step 6 with this:

6^2. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \bot$, then abort. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z \leftarrow f(x, y)$. If $\mathsf{De}(\hat{d}, Z) \neq z$, then send "you are in a hybrid" to $\mathcal{A}$, if not already done. Then send Z to $\mathcal{A}$.

We argue that the new hybrid is no easier to distinguish from the simulation than the previous hybrid. Note first that the change makes a difference only if $Z \neq Z' = \mathsf{De}^{-1}(\hat{d}, z)$. Note then that if $\mathsf{De}(\hat{d}, Z) \neq z$, then in both the new hybrid and the previous hybrid we send "you are in a hybrid" to $\mathcal{A}$, which allows $\mathcal{A}$ to perfectly distinguish from the simulation, where no such strings are sent, so sending $Z' \neq Z$ will not make it easier to distinguish from the simulation. Hence, the difference makes the new hybrid easier to distinguish from the simulation only if it is both the case that $Z \neq \mathsf{De}^{-1}(\hat{d}, z)$ and $\mathsf{De}(\hat{d}, Z) = z$. Putting these two together we get that $Z \neq \mathsf{De}^{-1}(\hat{d}, \mathsf{De}(\hat{d}, Z))$. The claim therefore follows from a trivial reduction to unqoe.

Now consider a hybrid, where we do not send the strings "you are in a hybrid" at any of the places where we do so in the previous hybrid. Since the previous hybrid is indistinguishable from the simulation where we do not send such strings, we conclude that it is sent with negligible probability. Hence not sending it is indistinguishable. Putting the current changes together and dropping all code only needed for sending the strings "you are in a hybrid" we see that the new hybrid looks as follows:

1. In **Garbling**, $\mathcal{S}$ simulates the ideal functionalities of π_{IGCO} to $\mathcal{A}$ by running them honestly, and it participates honestly in π_{IGCO} on behalf of B unless another behaviour is specified below.
2. If B aborts in the garbling, then abort A (i.e., input abort to the ideal functionality on behalf of A). Otherwise, let (F, v) denote the output to B.
3. Then $\mathcal{S}$ applies Ex_{C} to the communication of $\mathcal{A}$ in Gb_π to extract $\hat{e}$ and $\hat{d}$. It parses $\hat{e}$ as $\left(\hat{X}_1^0, \hat{X}_1^1, \ldots, \hat{X}_{n_\mathsf{A}}^0, \hat{X}_{n_\mathsf{A}}^1, \hat{Y}_1^0, \hat{Y}_1^1, \ldots, \hat{Y}_{n_\mathsf{B}}^0, \hat{Y}_{n_\mathsf{B}}^1\right)$.

4^2. Let $(X_1, \ldots, X_{n_\mathsf{A}})$ be the value sent by $\mathcal{A}$ in **Input A**. We call a index i *bad* if $X_i \notin \left\{\hat{X}_i^0, \hat{X}_i^1\right\}$ or $\hat{X}_i^0 = \hat{X}_i^1$. If there is a bad index and $\mathsf{Ev}(F, X) = \bot$, then abort A. If there is a bad index and $\mathsf{Ev}(F, X) \neq \bot$, then let $x = 0^{n_\mathsf{A}}$. Otherwise, let each x_i be the unique bit such that $X_i = \hat{X}_i^{x_i}$, and then let $x = x_1 \cdots x_{n_\mathsf{A}}$.

5^3. In **Input B, II**, $\mathcal{S}$ inspects the OT's to learn the messages $\left(\left(Y_i^0, O_i^0\right), \left(Y_i^1, O_i^1\right)\right)$ input to $\mathcal{F}_{\mathsf{DOT}}^i$ by $\mathcal{A}$. It then inspects $\mathcal{F}_{\mathsf{SFE}}^f$ to get the real value y of the input of B, as given by the environment. Define y' by letting $y' = y$. Then run as in the simulation, but with input y' for B, i.e., let $X = \left(X_1, \ldots, X_{n_\mathsf{A}}, Y_1^{y'_1}, \ldots, Y_{n_\mathsf{B}}^{y'_{n_\mathsf{B}}}\right)$ and $O = \left(O_1^{y'_1}, \ldots, O_{n_\mathsf{B}}^{y'_{n_\mathsf{B}}}\right)$, and if $\mathsf{Ve}\left(v, X, O, \{(i, y'_i)\}_{i=1}^{n_\mathsf{B}}\right) = \bot$, then abort A and terminate the simulated protocol.

6^3. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \bot$, then abort. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z \leftarrow f(x, y)$. Then send Z to $\mathcal{A}$.

In Step 6^3 we can drop the code *Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back* $z \leftarrow f(x, y)$ as it has no effect at this point. We can also drop the code *If there is a bad index and $\mathsf{Ev}(F, X) = \bot$, then abort A* in Step 4^2 as we would abort in Step 6 anyway when $\mathsf{Ev}(F, X) = \bot$. After that all the code of Step 4^2 used to define x can be dropped, as x is not used later on anymore. After that we can drop Step 3 as $\hat{e}$ and $\hat{d}$ are no longer used. In Step 5^3 we have that $y' = y$, so we can replace y' by y in all places. With these changes we arrive at this hybrid.

1. In **Garbling**, $\mathcal{S}$ simulates the ideal functionalities of π_{IGCO} to $\mathcal{A}$ by running them honestly, and it participates honestly in π_{IGCO} on behalf of B unless another behaviour is specified below.
2. If B aborts in the garbling, then abort A (i.e., input abort to the ideal functionality on behalf of A). Otherwise, let (F, v) denote the output to B.

3^1.

4^3. Let $(X_1, \ldots, X_{n_\mathsf{A}})$ be the value sent by $\mathcal{A}$ in **Input A**.

5^4. In **Input B, II**, $\mathcal{S}$ inspects the OT's to learn the messages $\left(\left(Y_i^0, O_i^0\right), \left(Y_i^1, O_i^1\right)\right)$ input to $\mathcal{F}_{\mathsf{DOT}}^i$ by $\mathcal{A}$. We then inspect $\mathcal{F}_{\mathsf{SFE}}^f$ to get the real value y of the input of B, as given by the environment. Then let

$X = \left(X_1, \ldots, X_{n_\mathsf{A}}, Y_1^{y_1}, \ldots, Y_{n_\mathsf{B}}^{y_{n_\mathsf{B}}}\right)$ and $O = \left(O_1^{y_1}, \ldots, O_{n_\mathsf{B}}^{y_{n_\mathsf{B}}}\right)$, and if $\mathsf{Ve}(v, X, O, \{(i, y_i)\}_{i=1}^{n_\mathsf{B}}) = \perp$, then abort A and terminate the simulated protocol.

6^4. Compute $Z \leftarrow \mathsf{Ev}(F, X)$. If $Z = \perp$, then abort. Otherwise send Z to $\mathcal{A}$.

It can be seen that by now all the values received by $\mathcal{A}$ are distributed exactly as in the protocol. This concludes the proof. □

D Proof of IGarb_π being an Interactive Garbling Scheme

In order to securely plug in our interactive garbling scheme IGarb_π into the generic protocol of Section 3, we need to show that it satisfies the properties of Figure 16 and Figure 17 which define an interactive garbling scheme. We prove this formally in Lemma 4, Lemma 5, Lemma 6, Lemma 8, Lemma 7, Lemma 11, Lemma 9, Lemma 12, and Lemma 10.

For ease of presentation we show our protocol secure for a restricted set of parameters, namely the case where $\alpha = \beta - 1$. We stress that our protocol can be shown secure for other combinations of α and β, but for sake of concreteness we have single out this case as we found it to perform well in terms of overall performance, relative to the security it provided.[6]

Before we continue recall the indistinguishability definition of obliviousness of a garbling scheme from [BHR12] which we include in Figure 18 and the definition of a secure PRG as shown in Figure 19. Also for completeness we include the definition of a circular correlation robust hash function for natural keys in Definition 1 as defined by [ZRE14], which in turn is based on the work of [CKKZ12]. This is simply Definition 1 and 3 of [ZRE14] written as one.

1. Run $\mathcal{B}(1^k)$ to produce (f, x_0, x_1).
2. If $\{x_0, x_1\} \not\subseteq \{0,1\}^{f.n}$, then return $\bot$.
3. Sample uniformly random $b \in \{0,1\}$.
4. Let $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and compute $X \leftarrow \mathsf{En}(e, x_b)$.
5. Let $b' \leftarrow \mathcal{B}(F, X)$.
6. If $b' = b$, then output $\top$, otherwise output $\bot$.

Fig. 18. The game $\mathrm{OblInd}^{\mathcal{B}}_{\mathcal{G}}(1^k)$ from [BHR12] with the modification that $f = f_0 = f_1$. This can be enforced in their setting by simply using the side-information function $\Phi = \Phi_{\mathrm{circ}}$.

1. Run $\mathcal{P}(1^k)$ to produce c. If c is not polynomial in k output $\bot$.
2. Sample a uniformly random bit b.
3. If $b = 0$ then sample a uniformly random seed $s \in \{0,1\}^k$ and use PRG to expand this seed to a string of c bits and return this to $\mathcal{P}$. If instead $b = 1$ sample a uniformly random string of c bits and return this to $\mathcal{P}$.
4. In the end let $\mathcal{P}$ outputs a guess b'. If $b' = b$ output $\top$, otherwise output $\bot$.

Fig. 19. Game $\mathrm{PRG}^{\mathcal{P}}_{\mathsf{PRG}}(1^k)$. **Property** prg.

Definition 1 (Circular correlation robustness). *Given a hash function $\mathcal{H}$ with k-bit output, define two oracles $\mathsf{Circ}_\Delta(x,i,b) = \mathcal{H}(x \oplus \Delta, i) \oplus b \cdot \Delta$ and $\mathsf{Rand}(x,i,b)$ which always outputs a uniformly random k-bit string. Say that a sequence of oracle queries of the form (x,i,b) is legal if the same value of (x,i) is never queried for different values of b. Then $\mathcal{H}$ is circular correlation robust if for all PPT adversaries $\mathcal{A}$, making legal queries,*

$$\left| \Pr_{\Delta}[\mathcal{A}^{\mathsf{Circ}_\Delta}(1^k) = 1] - \Pr_{\mathsf{Rand}}[\mathcal{A}^{\mathsf{Rand}}(1^k) = 1] \right| \text{ is negligible.}$$

Furthermore, we have that the function $\mathcal{H}$ is circular correlation robust for natural keys, if $\mathcal{A}$'s queries (x,i,b) is restricted in the following way;

1. *For the q'th query, we have that $i = q$.*
2. *$b \in \{0,1\}$.*
3. *That x is naturally derived, meaning it is obtained from one of the following:*

[6] However for some parameters this choice is not optimal, see Appendix H for details.

– $x \in_R \{0,1\}^k$.
– $x = x_1 \oplus x_2$, *where* x_1 *and* x_2 *are naturally derived.*
– $x \leftarrow \mathcal{H}(x_1, i)$, *where* x_1 *is naturally derived and* $i \in \mathbb{Z}$.
– $x \leftarrow \mathcal{O}(x_1, i, b)$, *where* x_1 *is naturally derived. Here* $\mathcal{O}$ *is either* Rand *or* Circ_Δ.

Finally define a gate garbling procedure $\mathsf{HGb}'(L^0, R^0, \Delta, id) \to \{G_{id}, O^0\}$ to be the same as procedure HGb from Figure 4, except that step 3 is as follows:

3. Output a garbled gate $\{G_{id}, O^0\} \leftarrow \{(id, T_G, T_E), O^0\}$.

Thus the only difference between HGb and HGb' is that the gates output by HGb' does not include "shifting information".

Lemma 4 (Obliviousness). *If* $\mathsf{HGarb} = (\mathsf{HGb}, \mathsf{HEn}, \mathsf{HDe}, \mathsf{HEv}, \mathsf{Hev})$ *is a obl.ind-secure garbling scheme and* $\mathcal{H}$ *is a circular correlation robust hash function for natural keys then the scheme* IGarb_π *is obl.ind.act-secure in the* $\mathcal{F}_{OT}$*-* $\mathcal{F}_{COM}$*-hybrid model assuming that* PRG *is prg-secure.*

Proof. It has been proven in [ZRE14] that a free-XOR (non-interactive) garbling scheme using $\mathsf{HGb}'(\cdot)$ to garble individual AND gates is obl.ind secure against malicious adversaries, assuming that $\mathcal{H}$ is a correlation robust hash function for natural keys.[7] We remark that for sake of exposition that our interactive garbling scheme IGarb_π does not satisfy the condition that for the q'th query, we have that $i = q$. However this is *w.l.o.g.* as a simple reenumeration of indices would ensure this.

Let $\mathrm{OblInd}^b_{\mathsf{HGarb}}(1^k)$ denote the game $\mathrm{OblInd}_{\mathsf{HGarb}}(1^k)$ when the bit sampled by the game is b, similarly let $\mathrm{OblIndAct}^c_{\mathsf{IGarb}_\pi}(1^k)$ denote the game $\mathrm{OblIndAct}_{\mathsf{IGarb}_\pi}(1^k)$ when the bit sampled by the game is c. Since we assume that HGarb is obl.ind-secure then it must hold that for any PPT $\mathcal{S}$ playing the games, we have $\mathrm{OblInd}^{\mathcal{S},0}_{\mathsf{HGarb}}(1^k) \approx^c \mathrm{OblInd}^{\mathcal{S},1}_{\mathsf{HGarb}}(1^k)$. In a similar manner we notice that it is enough for us to prove that $\mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathrm{OblIndAct}^{\mathcal{B},1}_{\mathsf{IGarb}_\pi}(1^k)$ for any PPT $\mathcal{B}$, assuming the PRG used is prg-secure.

We start by defining the game $\mathsf{H}^{\mathcal{B},b}(1^k)$ to be the same game as $\mathrm{OblIndAct}^{\mathcal{B},b}_{\mathsf{IGarb}_\pi}(1^k)$ except that it does not execute $\mathsf{C}(1^k, f)$, but instead $\mathsf{C}'(1^k, f)$. The only difference between C and C' is that in C', during the **Setup** phase of the ECC commitments, the bits $\{b_i\}_{i\in[\Gamma]}$ given by $\mathcal{B}$ as input to $\mathcal{F}_{\mathsf{OT}}$ are extracted by C'. Furthermore, based on these bits, the values $\bar{r}^i_{1-b_i} \in \{0,1\}^\gamma$ are chosen uniformly at random instead of as the output of a PRG.

Because of the prior observation we see that it is enough to prove

$$\mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},0}(1^k) \approx^c \mathsf{H}^{\mathcal{B},1}(1^k) \approx^c \mathrm{OblIndAct}^{\mathcal{B},1}_{\mathsf{IGarb}_\pi}(1^k) \ .$$

We prove each of these relations one at a time.

$\mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},0}(1^k)$. Consider the following reduction $\mathsf{R}^{\mathcal{B}}(1^k)$ participating in the $\mathrm{PRG}^{\mathsf{R}^{\mathcal{B}}}_{\mathsf{PRG}}(1^k)$ game, which we will use to show that $\mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},0}(1^k)$. We use $\mathsf{R}^{\mathcal{B},b}$ to denote $\mathsf{R}^{\mathcal{B}}$ when the bit sampled by $\mathrm{PRG}^b_{\mathsf{PRG}}$ is b. Now $\mathsf{R}^{\mathcal{B},b}$ runs the game $\mathrm{OblIndAct}^{\mathcal{B},b}_{\mathsf{IGarb}_\pi}(1^k)$ as the game is defined, except that it extracts the input bits to $\mathcal{F}_{\mathsf{OT}}$ given by $\mathcal{B}$ during the execution of the **Setup** phase of the ECC commitments in IGb_π. Denote these $\{b_i\}_{i\in[\Gamma]}$, it then uses $\mathrm{PRG}^b_{\mathsf{PRG}}$ to generate the bitstrings $\{\bar{r}^i_{1-b_i}\}_{i\in[\Gamma]}$ which it will use in the rest of the game. In particular this means when playing $\mathrm{PRG}^{\mathsf{R}^{\mathcal{B}},0}_{\mathsf{PRG}}(1^k)$, then the PRG is used to derive $\bar{r}^i_{1-b_i}$, if instead playing $\mathrm{PRG}^{\mathsf{R}^{\mathcal{B}},1}_{\mathsf{PRG}}(1^k)$ then a uniformly random string is used in place of $\bar{r}^i_{1-b_i}$. From this notice that $\mathsf{R}^{\mathcal{B},0}(1^k) \approx^p \mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k)$ and that $\mathsf{R}^{\mathcal{B},1}(1^k) \approx^p \mathsf{H}^{\mathcal{B},0}(1^k)$, since the OT functionality is ideal and the strings $r^i_{1-b_i}$ are never used anywhere else.

Now see that because efficient transformations maintain indistinguishability we get that $\mathsf{R}^{\mathcal{B},0}(1^k) \approx^c \mathsf{R}^{\mathcal{B},1}(1^k)$ since $\mathsf{R}^{\mathcal{B},b}(1^k)$ runs in PPT and because we assume $\mathrm{PRG}^0_{\mathsf{PRG}}(1^k) \approx^c \mathrm{PRG}^1_{\mathsf{PRG}}(1^k)$. Thus the relation follows from:

$$\mathrm{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^p \mathsf{R}^{\mathcal{B},0}(1^k) \approx^c \mathsf{R}^{\mathcal{B},1}(1^k) \approx^p \mathsf{H}^{\mathcal{B},0}(1^k) \ .$$

[7] Actually the authors show security for obl. sim, but [BHR12] shows that obl. sim implies obl.ind in this setting.

$\text{OblIndAct}^{\mathcal{B},1}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},1}(1^k)$. We see this follows directly from the proof of $\text{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},0}(1^k)$.

$\mathsf{H}^{\mathcal{B},0}(1^k) \approx^c \mathsf{H}^{\mathcal{B},1}(1^k)$. Consider the following reduction $\mathsf{S}^{\mathcal{B},b}(1^k)$ participating in the $\text{OblInd}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$ game. $\mathsf{S}^{\mathcal{B},b}$ works as follows:

For ease of notation we let (f, x_0, x_1, I) denote the input $\mathcal{B}$ gives to $\mathsf{S}^{\mathcal{B},b}$ and let $\left(\hat{f}, \hat{x}_0, \hat{x}_1\right)$ denote the input $\mathsf{S}^{\mathcal{B},b}$ will give to $\text{OblInd}^b_{\mathsf{HGarb}}$. First let $\mathsf{S}^{\mathcal{B},b}$ learn (f, x_0, x_1, I) from $\mathcal{B}$ at the beginning of the game. If $\{x_0, x_1\} \not\subseteq \{0,1\}^{f.n}$ or $I \not\subseteq [f.n]$ or $x_{0,i} \neq x_{1,i}$ for $i \in I$, then output $\perp$. Now $\mathsf{S}^{\mathcal{B},b}$ runs the game $\text{OblIndAct}^{\mathcal{B},b}_{\mathsf{IGarb}_\pi}(1^k)$ playing the role of an honest C , except it extracts the messages of all the commitments $\mathcal{B}$ makes to $\mathcal{F}_{\mathsf{COM}}$ in **Setup** and all the input bits he gives to $\mathcal{F}_{\mathsf{OT}}$ in **Setup** of the ECC commitments, call these $\left\{\bar{b}_i\right\}_{i\in[I]}$. Thus $\mathsf{S}^{\mathcal{B},b}$ will know exactly what the cut-and-choose challenges will be and which bits $\mathcal{B}$ will want to use to verify the garbled gates and wire authenticators selected for checking. For $g \in \mathsf{CheckGates}$ and $j \in \mathsf{CheckWires}$ this is the set $\left\{\{(a_g, b_g, g)\}, \{(c_j, j)\}, \mathsf{BucketOf}, \mathsf{AuthOf}\right\}$. Then $\mathsf{S}^{\mathcal{B},b}$ defines $\hat{f}$ as a function with only a single layer, consisting entirely of AND gates. In particular $\hat{f}$ will contain $|\mathsf{GGates}|$ AND gates. Next $\mathsf{S}^{\mathcal{B},b}$ evaluates which bit will be on each of the wires in the circuit representing f, when evaluated on x_0, respectively x_1. For each wire in f that is an input wire to an AND gate or a circuit input wire we find the indices of the garbled gates in the bucket representing this AND gate that has this wire as one of its inputs. There will be β such gates for each of these wires. These gate are uniquely defined by $\mathsf{BucketOf}$, which is known to the simulator at this point. The simulator then sets the $2|\mathsf{EvalGates}|$ bits of $\hat{x}_0$, respectively $\hat{x}_1$, in accordance with the values expected on each wire. Then for each $g \in \mathsf{CheckGates}$ the simulator sets $\hat{x}_0[2g] = \hat{x}_1[2g] = a_g$ and $\hat{x}_0[2g+1] = \hat{x}_1[2g+1] = b_g$. Notice that $|\hat{x}_0| = |\hat{x}_1| = 2|\mathsf{GGates}|$ since each gate has 2 bits input. For the rest of the entries in $\hat{x}_0$ and $\hat{x}_1$ it always chooses the 0-bit (these are the "slack" entries and will just be discarded).

Next $\mathsf{S}^{\mathcal{B},b}$ sends $(\hat{f}, \hat{x}_0, \hat{x}_1)$ to $\text{OblInd}^b_{\mathsf{HGarb}}$ and receives back $(\hat{F}, \hat{X}_b)$. $\mathsf{S}^{\mathcal{B},b}$ then runs $\mathsf{HEv}(\hat{F}, \hat{X}_b)$ and thus learns a key for each wire (which will be exactly the key for each wire in correspondence to what $\mathcal{B}$ expects when it is put together to a fault tolerant garbled circuit with respect to the $\mathsf{BucketOf}$ function). Name the set of all these keys $\left\{\hat{K}_j\right\}_{j\in\mathsf{GateWires}}$. Now $\mathsf{S}^{\mathcal{B},b}$ runs the rest of IGb_π with $\mathcal{B}$ as an honest C would except for the following steps:

1. In **Gate Construction** step 1 instead of actually garbling the gates $\mathsf{S}^{\mathcal{B},b}$ "extracts" the garbled gates from $\hat{F}$. Parse each of these gates as $\{G_g\}_{g\in\mathsf{GGates}} = \{(g, T^g_G, T^g_E)\}_{g\in\mathsf{GGates}}$. For $g \in \mathsf{GGates}$ it then defines the gates $\left\{G'_g\right\} = \left\{\left(g, T^g_G, T^g_E, 0^k, 0^k, 0^k\right)\right\}$. For $j \in [\gamma]$ the simulator parses $\boldsymbol{s}_j \to (\boldsymbol{s}^0_j, \boldsymbol{s}^1_j)$ and denotes by $\boldsymbol{k}^{b_j}_j$ the first k bits of $\boldsymbol{s}^{b_j}_j$. Furthermore for $j \in \mathsf{AWires} \setminus \mathsf{AuthWires}$, $\mathsf{S}^{\mathcal{B},b}$ sets the key $\hat{K}_j = \boldsymbol{k}^0_j \oplus \boldsymbol{k}^1_j$ and for $j \in \mathsf{AuthWires}$ it uses the value $\hat{K}_j$ it learned from $\text{OblInd}^b_{\mathsf{HGarb}}$ or the value it computed by running Ev. It then computes for $j \in \mathsf{AWires}$, $H^0_j \leftarrow \mathcal{H}\left(\hat{K}_j, j\right)$ and samples $H^1_j \in_R \{0,1\}^k$. As in the Auth procedure if $H^0_j \leq H^1_j$ it sets $H_j \leftarrow \left(H^0_j, H^1_j, 0^k\right)$, otherwise it sets $H_j \leftarrow \left(H^1_j, H^0_j, 0^k\right)$.
   Now, for $i \in \mathsf{CheckGates} \cup \mathsf{EvalGates}$ let $(l_i, r_i, o_i) \leftarrow \mathsf{WiresOf}(i)$ and update the variables as follows:
$$\boldsymbol{k}^{1-\bar{b}_{l_i}}_{l_i} := \boldsymbol{k}^0_{l_i} \oplus \boldsymbol{k}^1_{l_i} \oplus \hat{K}_{l_i}$$
$$\boldsymbol{k}^{1-\bar{b}_{r_i}}_{r_i} := \boldsymbol{k}^0_{r_i} \oplus \boldsymbol{k}^1_{r_i} \oplus \hat{K}_{r_i} \ .$$
   The simulator furthermore updates $\boldsymbol{k}^{1-\bar{b}_{W+1}}_{W+1} := \boldsymbol{k}^{1-\bar{b}_{W+1}}_{W+1} \oplus \boldsymbol{k}^0_{W+1} \oplus \boldsymbol{k}^1_{W+1}$ (recall that K_{W+1} is used as the global difference Δ, and this procedure makes this the all zero string). It also updates the first k bits of each $\boldsymbol{s}^{1-\bar{b}_{l_i}}_{l_i}$ and $\boldsymbol{s}^{1-\bar{b}_{r_i}}_{r_i}$ (along with $\boldsymbol{s}^{1-\bar{b}_{W+1}}_{W+1}$) so they are consistent with $\boldsymbol{k}^{1-\bar{b}_{l_i}}_{l_i}$ and $\boldsymbol{k}^{1-\bar{b}_{r_i}}_{r_i}$ (and $\boldsymbol{k}^{1-\bar{b}_{W+1}}_{W+1}$). Finally, $\mathsf{S}^{\mathcal{B},b}$ then sends $\left\{G'_g\right\}_{j\in\mathsf{GGates}}, \{H_j\}_{j\in\mathsf{AWires}}$ to $\mathcal{B}$.
2. In **Bucket Soldering** for all $h \in \mathsf{HeadGates}$, all $g \in \mathsf{Bucket}_h$ where $g \neq h$ and all $a \in \mathsf{AuthOf}(h)$ the simulator sends to $\mathcal{B}$:
$$\widetilde{S}^L_g = L_g \oplus L_h, \quad \widetilde{S}^R_g = R_g \oplus R_h, \quad \widetilde{S}^O_g = O_g \oplus O_h \quad \widetilde{S}^a_h = \hat{K}_a \oplus O_h \ .$$

Where the keys L_g, R_g, O_g, L_h, R_h, O_h and $\hat{K}_a$ are the keys $\mathsf{S}^{\mathcal{B},b}$ got from $\text{OblInd}^b_{\mathsf{HGarb}}$ and learned from calling HEv on $\hat{F}$. It follows the rest of the step as an honest C would.

3. In **Topological Soldering** for all $h \in \mathsf{HeadGates}$ it sets $L = \bigoplus_{u \in \mathsf{lp}(h)} O_u$, $R = \bigoplus_{v \in \mathsf{rp}(h)} O_v$, where the keys O_u and O_v are the keys $\mathsf{S}^{\mathcal{B},b}$ learned from calling HEv on $\hat{F}$. In case either $\mathsf{lp}(h) \in [n]$ or $\mathsf{rp}(h) \in [n]$ the wire $\hat{K}^0_v$, respectively $\hat{K}^0_u$ is used instead. The simulator then sends to $\mathcal{B}$:

$$\widetilde{S}^L_h = L_h \oplus L, \quad \widetilde{S}^R_h = R_h \oplus R \ .$$

   Where the keys L_h, R_h, O_h are the keys $\mathsf{S}^{\mathcal{B},b}$ got from $\text{OblInd}^b_{\mathsf{HGarb}}$ and learned from calling HEv on $\hat{F}$. It follows the rest of the step as an honest C would.
4. In **Input Authentication** for all $i \in \{\mathsf{HeadAuths} \cap [n]\}$ and all $a \in \mathsf{Auth}_i$ where $a \neq i$ the simulator sends to $\mathcal{B}$:

$$\widetilde{S}_a = \hat{K}_a \oplus \hat{K}_i \ .$$

   Where the keys $\hat{K}_a$ and $\hat{K}_i$ are the keys $\mathsf{S}^{\mathcal{B},b}$ got from $\text{OblInd}^b_{\mathsf{HGarb}}$ and learned from calling HEv on $\hat{F}$. It follows the rest of the step as an honest C would.
5. In **Output** step 1, $\mathsf{S}^{\mathcal{B},b}$ sets e, o and d to be uniformly random sampled bit strings of sufficient length.

$\mathsf{S}^{\mathcal{B},b}$ then sets X to be the subset of $\hat{X}_b$ it got from $\text{OblInd}^b_{\mathsf{HGarb}}$ in correspondence with the wires that are supposed to be the input wires of f. It furthermore sets O to be the updated shares of the input wires in I, that is corresponding to $\mathcal{B}$'s part of the x_0, respectively x_1.[8] Finally, $\mathsf{S}^{\mathcal{B},b}$ gives (X, O) to $\mathcal{B}$.

Now notice that $\mathsf{S}^{\mathcal{B},b}(1^k) \approx^c \mathsf{H}^{\mathcal{B},b}(1^k)$ for any $b \in \{0,1\}$. To see this we argue that everything sent by $\mathsf{S}^{\mathcal{B},b}$ to $\mathcal{B}$ is computationally indistinguishable from what is sent in $\mathsf{H}^{\mathcal{B},b}(1^k)$:

1. In **Gate Construction** step 1 notice that $\mathsf{S}^{\mathcal{B},b}(1^k)$ picks the garbled gates using the $\text{OblInd}^b_{\mathsf{HGarb}}$ game, where the garbled computation tables are constructed exactly in the same way as in $\mathsf{H}^{\mathcal{B},b}(1^k)$, except that we augment these with three 0^k-strings. It also simulates the authenticator pairs, H_j. In each of these pairs one of the entries is the output of a circular correlation robust hash function on a uniformly random key, and the other is simply a uniformly random value. Since in $\mathsf{H}^{\mathcal{B},b}(1^k)$, all H_j are produced using the Auth method, which in turn invokes $\mathcal{H}$ on K^0_j and $K^0_j \oplus \Delta$ we see that all invocations can be modeled as queries to the Circ_Δ oracle (with $b = 0$) as defined in Definition 1. In the simulation, only one of the authenticators is produced this way, while the other is a uniformly sampled string. However by the correlation robustness property of the hash function $\mathcal{H}$ the two distributions are indistinguishable to $\mathcal{B}$. It is also eminent to see that all queries would satisfy the "natural" property, except for the condition that for the q'th query, we have that $i = q$, but we have already addressed this earlier in the proof.
   Since $\mathcal{B}$ will only learn the key which is the output of the oracle, the other part of the authenticator pair will be distributed similarly in this simulation and in the $\mathsf{H}^{\mathcal{B},b}$ game.[9] Furthermore, notice that for $j \in \mathsf{AuthWires}$, $\mathcal{B}$ will eventually learn either K^0_j or $K^0_j \oplus \Delta$, through the evaluation of the garbled circuit, and that exactly one of the values hashed in the authenticator will be this key since its value comes from $\text{OblInd}^b_{\mathsf{HGarb}}$. For $j \in \mathsf{AWires} \backslash \mathsf{AuthWires}$ only K^0_j will be accepted, but this can be made to look like K^1_j since $\mathcal{B}$ will only learns the key through the opening of an ECC commitment, and the index which should be Δ is the 0-string.
   The corrections $\mathsf{S}^{\mathcal{B},b}$ makes to $\boldsymbol{s}_j$ will only be visible to $\mathcal{B}$ through the **Linear Combination Check** executions, but here each opening of shares based on shares which are not uniformly random distributed will be XORed with a share of a key that is uniformly random distributed (in particular a key with index greater than $W + 1$).
2. In **Soldering** step 3,4 and 5 the shifting values will be distributed similarly in $\mathsf{S}^{\mathcal{B},b}$ and $\mathsf{H}^{\mathcal{B},b}$ as in both cases they will be the XOR of keys constructed similarly. The difference is that in $\mathsf{S}^{\mathcal{B},b}$ some of the

[8] Notice that the bits supposed to be on these wires will be the same for both x_0 and x_1 in correspondence with OblIndAct.

[9] Any information on the "other key" on a wire will yield a direct advantage in guessing Δ, and thus correspond to an advantage in winning the game, no matter if we are in the simulation or not.

keys might be 1-keys. However $\text{OblInd}^b_{\mathsf{HGarb}}$ ensures that an XOR combinations of distinct wire keys are indistinguishable whether or not the keys are 0,1 or a combination, because of the security of the free-XOR approach. Furthermore, we notice that $K^0_j \oplus K^0_{j+1} = K^0_j \oplus \Delta \oplus K^0_{j+1} \oplus \Delta = K^1_j \oplus K^1_{j+1}$, so we see that the solderings will be the same in the real protocol and the simulation since the keys to be soldered together will always represent the same bit.

3. In **Output** notice that none of these values are given to $\mathcal{B}$.

Finally (X, O) is constructed in the same way in both $\mathsf{S}^{\mathcal{B},b}(1^k)$ and $\mathsf{H}^{\mathcal{B},b}(1^k)$. Thus we conclude that the view of the values sent to $\mathcal{B}$ in these hybrids are computationally indistinguishable. This concludes the proof that $\mathsf{S}^{\mathcal{B},b}(1^k) \approx^c \mathsf{H}^{\mathcal{B},b}(1^k)$.

Now see that because efficient transformations maintain indistinguishability we get that $\mathsf{S}^{\mathcal{B},0}(1^k) \approx^c \mathsf{S}^{\mathcal{B},1}(1^k)$ since $\mathsf{S}^{\mathcal{B},b}(1^k)$ runs in PPT and because we assume that $\text{OblInd}^0_{\mathsf{HGarb}}(1^k) \approx^c \text{OblInd}^1_{\mathsf{HGarb}}(1^k)$. Thus the statement follows from the following observation:

$$\mathsf{H}^{\mathcal{B},0}(1^k) \approx^c \mathsf{S}^{\mathcal{B},0}(1^k) \approx^c \mathsf{S}^{\mathcal{B},1}(1^k) \approx^c \mathsf{H}^{\mathcal{B},1}(1^k) \ .$$

Finally see that we are done since we have showed

$$\text{OblIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}_\pi}(1^k) \approx^c \mathsf{H}^{\mathcal{B},0}(1^k) \approx^c \mathsf{H}^{\mathcal{B},1}(1^k) \approx^c \text{OblIndAct}^{\mathcal{B},1}_{\mathsf{IGarb}_\pi}(1^k) \ .$$

□

Before proving our scheme has the aut.act property, recall the definition of Authenticity in a non-interactive garbling scheme in Figure 20.

1. Run $\mathcal{B}(1^k)$ to produce (f, x).
2. If $x \not\subseteq \{0,1\}^{f.n}$, then return $\perp$.
3. Let $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and compute $X \leftarrow \mathsf{En}(e, x)$.
4. Let $Z \leftarrow \mathcal{B}(F, X)$.
5. If $\mathsf{De}(d, Z) \neq \perp$ and $Z \neq \mathsf{Ev}(F, X)$ output $\top$, otherwise output $\perp$.

Fig. 20. The game $\text{Aut}^{\mathcal{B}}_{\mathcal{G}}(1^k)$ from [BHR12].

Lemma 5 (Authenticity). *If* $\mathsf{HGarb} = (\mathsf{HGb}, \mathsf{HEn}, \mathsf{HDe}, \mathsf{HEv}, \mathsf{Hev})$ *is an* aut*-secure garbling scheme and* $\mathcal{H}$ *is a circular correlation robust hash function for natural keys then the scheme* $\mathsf{IGarb}_\pi = (\mathsf{IGb}_\pi, \mathsf{IEn}, \mathsf{IDe}, \mathsf{IEv}, \mathsf{Iev}, \mathsf{IOp}, \mathsf{IVe})$ *is aut.act-secure in the* $\mathcal{F}_{OT}$*-* $\mathcal{F}_{COM}$*-hybrid model assuming that PRG is prg-secure.*

Proof. Considering the game $\text{AutAct}^{\mathcal{B}}_{\mathsf{IGarb}_\pi}(1^k)$ for the aut.act property we first argue that $\mathcal{B}$ cannot win the game by being malicious in IGb_π such that $\mathsf{IDe}(d, Z) \neq f(x)$, thus concluding he can only win the game by finding Z' such that $\mathsf{IDe}(d, Z') \neq \perp$ with $Z' \neq Z$. We will then prove that if he can find such a Z' with non-negligible probability in 1^k then we can use him to win the $\text{Aut}_{\mathsf{HGarb}}(1^k)$ game with non-negligible probability, under the assumption that PRG is prg-secure. In particular we will construct a simulator $\mathcal{S}^{\mathcal{B}}$ that plays the $\text{Aut}^{\mathcal{S}^{\mathcal{B}}}_{\mathsf{HGarb}}(1^k)$ game in Figure 20, to construct the garbled gates used in IGb_π, then argue that whether or not a $\mathcal{B}$ playing the $\text{AutAct}^{\mathcal{B}}_{\mathsf{IGarb}_\pi}(1^k)$ game is communicating with the simulator or the real game, what it learns will be computationally indistinguishable under the assumption that PRG is prg-secure. The simulator then uses the output of $\mathcal{B}$ to win the $\text{Aut}^{\mathcal{S}^{\mathcal{B}}}_{\mathsf{HGarb}}(1^k)$ game if $\mathcal{B}$ wins the $\text{AutAct}^{\mathcal{B}}_{\mathsf{IGarb}_\pi}(1^k)$ game.

To first show that $\mathsf{IDe}(d, Z) \neq f(x)$ is not possible, notice that the only way $\mathcal{B}$ could cause this to happen is to act maliciously during the execution of IGb_π. However, see that $\mathcal{B}$ only gets to give the following input to this protocol:

1. The Γ random bits in **Setup** of the ECC commitments.

2. The commitments, and subsequently openings, to the cut-and-choose challenges $\Big(\{(a_g, b_g, g)\}_{g\in\mathsf{CheckGates}}, \{(c_j, j)\}_{j\in\mathsf{CheckWires}}\Big)$ and $(\mathsf{BucketOf}, \mathsf{AuthOf})$ in **Setup**.
3. The random matrix V used during the calls to **Linear Combination Check** in the ECC commitments.

Notice all the elements are randomly sampled and made only to protect Bob against a malicious Alice. In particular in the first case, the choices are never learned by Alice because of the security of $\mathcal{F}_{\mathsf{OT}}$ and thus cannot have any influence. In the second case we first notice that if the protocol finishes then by the correctness of $\mathcal{F}_{\mathsf{COM}}$, it must mean that $\mathcal{B}$ committed (and later opened) to valid messages. In both the second and third case the values are simply selections for random checks and Alice terminates the protocols if the choices are not sane. Also notice that all the possible random choices, as long as they are well-formed, will not influence the correctness of the protocol. Thus it is clearly not possible for $\mathcal{B}$ to influence the execution in such a way that the garbling is incorrect. In particular this means that for the output (F, e, d, o) to C it will always be the case that $\mathsf{IDe}(d, \mathsf{IEv}(F, \mathsf{IEn}'(e, x))) = f(x)$.

Now define the hybrid game $\mathsf{G}^{\mathcal{B},b}(1^k)$ which is the same game as $\mathrm{AutAct}^{\mathcal{B},b}_{\mathsf{IGarb}_\pi}(1^k)$ except that it does not execute $\mathsf{C}(1^k, f)$, but instead $\mathsf{C}'(1^k, f)$. The only difference between C and C' is that in C', during the **Setup** phase of the ECC commitments, the bits $\{b_i\}_{i\in[I]}$ given by $\mathcal{B}$ as input to $\mathcal{F}_{\mathsf{OT}}$ are extracted by C'. Furthermore, based on these bits, the values $\bar{r}^i_{1-b_i} \in \{0,1\}^\gamma$ are chosen uniformly at random instead of through the PRG.

Now recall the proof of obl.ind.act security in Lemma 4 where exactly the same algorithm C' is introduced and used to show that there is computational indistinguishability between using C' instead of C under the assumption that PRG is prg-secure. Using this observation we see that it is enough to show that we can construct a simulator $\mathcal{S}^{\mathcal{B},b}$ which makes use of an adversary $\mathcal{B}$ attacking $\mathsf{G}^{\mathcal{B},b}(1^k)$ with non-negligible advantage to win the $\mathrm{Aut}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$ game with non-negligible advantage.

Again recalling the proof of obl.ind.act security in Lemma 4, remember the simulator $\mathsf{S}^{\mathcal{B},b}(1^k)$, where b is the bit used internally in the game $\mathrm{OblInd}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$. We create $\mathsf{S}^{\mathcal{B},b}$ in exactly the same way except that it is playing the game $\mathrm{Aut}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$ instead of $\mathrm{OblInd}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$ and therefore, instead of getting both the value x_0 and x_1 from $\mathcal{B}$, it gets a single value x. Notice that when x_0 and x_1 are used in $\mathsf{S}^{\mathcal{B},b}$ they are used in the same manner, so $\mathsf{G}^{\mathcal{B},b}(1^k)$ will simply do the same, but only for the single value x. From this it is easy to see, following the indistinguishability proof of $\mathsf{S}^{\mathcal{B},b}$ and $\mathsf{H}^{\mathcal{B},b}$, that whatever $\mathcal{B}$ sees will be computationally indistinguishable whether or not he is playing with the actual hybrid game $\mathsf{G}^{\mathcal{B},b}(1^k)$ or $\mathrm{AutAct}^{\mathcal{B},b}_{\mathsf{IGarb}_\pi}(1^k)$. Now when $\mathcal{B}$ sends Z' to $\mathcal{S}^{\mathcal{B},b}$, the simulator finds the first key entry in Z' that differs with Z. It then XORs these two keys together to learn Δ'. It then XORs this with the first key entry in the X it got from $\mathrm{Aut}^b_{\mathsf{HGarb}}$ to learn a new string X'. It then inputs this to $\mathrm{Aut}^b_{\mathsf{HGarb}}$. It is now easy to see that if $\mathcal{B}$ can win the hybrid game $\mathsf{G}^{\mathcal{B},b}(1^k)$ with non-negligible probability then $\mathcal{S}^{\mathcal{B},b}$ also wins the $\mathrm{Aut}^{\mathcal{S}^{\mathcal{B}},b}_{\mathsf{HGarb}}(1^k)$ game with non-negligible probability since the simulation and the hybrid game are indistinguishable and that the keys are constructed using $\mathrm{Aut}^b_{\mathsf{HGarb}}$. As HGarb is assumed aut-secure this proves the claim. □

Lemma 6 (knof). *The scheme IGarb_π has the knof property.*

Proof. In order to prove that our scheme satisfies the knof property we need to specify the Ex_{E} extractor. We first assume that $\mathcal{B}$ is in a state where it first received an input 1^k, then output some function f and finally ran an instance of IGb_π playing the role of E against an honest C. Also assume that the output of C is $(F, e, d, o) \neq \perp$ (else there is nothing to show). We now make the following observations:

1. As C did not output $\perp$ during the execution of IGb_π, by the correctness of $\mathcal{F}_{\mathsf{COM}}$, it must mean that $\mathcal{B}$ committed (and later opened to) valid cut-and-choose challenges $\Big(\{(a_g, b_g, g)\}_{g\in\mathsf{CheckGates}}, \{(c_j, j)\}_{j\in\mathsf{CheckWires}}\Big)$ and bucket mapping functions $(\mathsf{BucketOf}, \mathsf{AuthOf})$ and thus these are part of his view.
2. The next thing to note is that C also sent $\{G_g\}_{g\in\mathsf{GGates}}$ to $\mathcal{B}$ and that $\mathsf{EvalGates} \subset \mathsf{GGates} \setminus \mathsf{CheckGates}$ is of size $q\beta$ (else C would have output $\perp$). Recall that $\mathsf{EvalGates}$ is directly identified from $\mathsf{BucketOf}$.

3. Similarly we see that C also sent $\{H_j\}_{j\in\mathsf{AWires}}$ to $\mathcal{B}$ and that $\mathsf{AuthWires} \subset \mathsf{AWires} \setminus \mathsf{CheckWires}$ is of size $q\alpha + n(2\beta+1)$ (else C would have output $\perp$). Recall that $\mathsf{AuthWires}$ is directly identified from AuthOf.
4. It is also clear that C sent all solderings specified by $\mathsf{BucketOf}$ and AuthOf to $\mathcal{B}$, as it is honest.

Finally notice that given $\mathsf{BucketOf}$ the reenumeration described in Section A.2 is completely deterministic. It now follows by the above observations that all information for computing $\hat{F}$ is in the view of $\mathcal{B}$, when C does not output $\perp$. In particular, $\{G_g\}_{g\in\mathsf{EvalGates}}$, $\{H_j\}_{j\in\mathsf{AWires}}$, the solderings, $\mathsf{BucketOf}$ and AuthOf completely define $\left\{\widetilde{G}_g\right\}_{g\in\mathsf{EvalGates}}$ and $\left\{\widetilde{H}_j\right\}_{j\in\mathsf{AuthWires}}$.

Ex_E therefore simply extracts the above information from $\mathcal{B}$'s view and lets its output be $\hat{F} = \Big(n, m, q, (\mathsf{Ip}, \mathsf{rp}, \mathsf{BucketOf}, \mathsf{AuthOf}), \left\{\widetilde{G}_g\right\}_{g\in\mathsf{EvalGates}}, \left\{\widetilde{H}_j\right\}_{j\in\mathsf{AuthWires}}\Big)$. By the above it is now clear to see that indeed $F = \hat{F}$ if C does not output $\perp$ in the execution of IGb_π. We thus conclude that the scheme IGarb_π has the knof property. □

Lemma 7 (corr). *The scheme IGarb_π has the corr property.*

Proof. Consider the game $\mathrm{Corr}^{\mathcal{A}}_{\mathsf{IGarb}_\pi}(1^k)$ where an adversary $\mathcal{A}$ inputs (f, x, I) to $\mathrm{Corr}_{\mathsf{IGarb}_\pi}$. We assume that $x \in \{0,1\}^{f.n}$ and $I \subseteq [f.n]$ as else there is nothing to prove. $\mathrm{Corr}_{\mathsf{IGarb}_\pi}$ now interprets f as a binary circuit C on the form specified in Section 2, under the constraint that the last $|I|$ bits of the input wires correspond to the indices of I. In other words the indices of I are treated as the inputs of E. It then runs $\mathsf{C}(1^k, f)$ and $\mathsf{E}(1^k, f)$ as specified by IGb_π.

Then by the linearity of the ECC code $\mathcal{C}$, correctness of HGarb, $\mathcal{F}_{\mathsf{OT}}$, and $\mathcal{F}_{\mathsf{COM}}$, the projective algorithms $\mathsf{IEn}, \mathsf{IOp}, \mathsf{IVe}$ do not output $\perp$ and neither does IEv. The requirement that $\mathsf{IDe}(d, Z) = f(x)$ follows clearly by the fact that $\mathsf{BucketOf} \in \mathcal{B}$ and $\mathsf{AuthOf} \in \mathcal{V}$ (which holds since both parties ran IGb_π honestly) and we thus conclude that IGarb_π has the corr property. □

Lemma 8 (Projectiveness). *The scheme IGarb_π is projective.*

Proof. It is clear that the produced e and o of the protocol IGb_π are of the required form. The verification algorithm IVe defined in Figure 10 clearly also verifies openings individually so it also satisfies the requirement. Finally, the projective algorithms IEn and IOp clearly coincide with the requirements as well. We thus conclude that the scheme IGarb_π has the proj property. □

The proofs of the remaining properties, unqie, rob.con, unqoe and tok.com requires an extractor Ex_C, which we will now define. Furthermore, the proof of unqie and rob.con requires the two helper lemmas, Lemma 13 and Lemma 14. We will prove the helper lemmas in the very end of this appendix.

$\mathsf{Ex}_\mathsf{C}(\mathcal{A})$. Let $\mathcal{A}$ be an adversary playing the role of C in an execution of IGb_π. Assume that it is in a state where it first received an input 1^k, then output some function f and finally ran an instance of IGb_π playing against an honest E. Also assume that the output of E is $(F, v) \neq \perp$. As E does not output $\perp$, it must be the case that $\mathcal{A}$ correctly answers the linear combination checks of the protocol. By viewing the challenge matrix V as consisting of rows of random puzzles as defined in Appendix E we can construct Ex_C using the extractor S defined in Theorem 2. For clarity the solution to such a puzzle is the opening to the XOR of the commitments defined by the puzzle. Since there are $7.3(s+3)$ rows in each V used in IGb_π the condition for Theorem 2 is satisfied. Also since the ECC $\mathcal{C}$ has distance $d = s + 4$ we have that $\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k, r) \leq 2^{-4}2^{-s}$ as required by the theorem.

By invoking the theorem we therefore get that if Alice (playing the role of R in the theorem) can successfully solve these random puzzles, we can run the poly-time S as defined in Theorem 2 on her view to extract all the basis solutions $(\hat{\sigma}^1, \ldots, \hat{\sigma}^t)$ except with probability 2^{-s}. We now notice that each of these basis solutions $\hat{\sigma}^i$ are in fact a valid opening to wire i.

Since in the first linear combination check the random puzzles involve all produced wires we therefore end up extracting $(\hat{e}^1, \ldots, \hat{e}^\gamma)$ which are the openings to all individual keys $\hat{K}^0_i$ for $i = 1, \ldots, \gamma$.

As the extractor $\mathsf{Ex_C}$ knows $\mathsf{BucketOf}$ and $\mathsf{WiresOf}$, it reenumerates the wires according to the procedure described in Section A.2. Finally it defines $\hat{e} = \left(\left(\hat{K}_1, \hat{K}_1 \oplus \hat{K}_{W+1}\right), \ldots, \left(\hat{K}_n, \hat{K}_n \oplus \hat{K}_{W+1}\right)\right)$ and $\hat{d} = \left((\hat{O}_{w-m+1}, \hat{O}_{w-m+1} \oplus \hat{O}_{W+1}), \ldots, (\hat{O}_w, \hat{O}_w \oplus \hat{O}_{W+1})\right)$ and outputs these.

Lemma 9 (unqie). IGarb_π *has the property unqie.*

Proof. First let $\hat{e}$ be the extracted produced by $\mathsf{Ex_C}(\mathcal{A})$ after a has participated in an execution of IGb_π. Next let X, O, I and $\{(i, x_i')\}$ be the output of $\mathcal{A}$. We start by assuming that $\mathsf{IEn}(\hat{e}, \mathsf{IEn}^{-1}(\hat{e}, X)) \neq X$ as else there is nothing to prove. First notice that $\mathsf{IEn}(\hat{e}, \mathsf{IEn}^{-1}(\hat{e}, X)) \neq X$ can only occur if there exists an $i \in [n]$ such that $\hat{K}_i^0 \neq X_i \neq \hat{K}_i^0 \oplus \Delta$. We then split the proof into two cases, depending if $i \in I$ or not. For the case where $i \in I$ we see that by Lemma 13 $\mathsf{IVe}\,(v, X, O, \{(i, x_i')\}) = \perp$ except with prob. 2^{-s}, because this would either imply that $\mathcal{A}$ guesses at least $d = \Omega(s)$ of C's watch bits or the extraction of $\hat{e}$ failed in the first place.

For the second case, i.e. $i \notin I$, we claim that $\mathsf{IEv}(F, X) = \perp$ if $\mathsf{IEn}(\hat{e}, \mathsf{IEn}^{-1}(\hat{e}, X)) \neq X$ except with prob. 2^{-s}. To see this assume for contradiction this is not the case. Then recall that in the **Input Authentication** step of IGb_π, $\mathcal{A}$ solders $2\beta+1$ wire authenticators onto $\hat{K}_i^0$ except with probability 2^{-s}, again due to Lemma 13. Next see that in order for $\mathsf{IEv}(F, X) \neq \perp$, then in step 3 of IEv X_i needs to be accepted by at least $\beta + 1$ of these wire authenticators. As we have that $\hat{K}_i^0 \neq X_i \neq \hat{K}_i^0 \oplus \Delta$ this can only happen if at least $\beta + 1$ of the wire authenticators are corrupt. However the probability of this occurring is bounded in the analysis of Lemma 14 which is bounded by 2^{-s} for appropriate values of β.

□

Lemma 10 (rob.con). *If* IGarb_π *has the corr property, then it also has the rob.con property except with probability at most*

$$q \cdot \left(\prod_{i=\beta}^{1} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) + \sum_{l=2}^{\beta} \prod_{i=\beta}^{l} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha+2-l} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) \right) + 2^{-s}$$

Proof. By definition of $\hat{e} = (K_1^0, K_1^1, \ldots, K_n^0, K_n^1)$ and $\hat{d} = (\hat{O}_{w-m+1}^0, \hat{O}_{w-m+1}^1, \ldots, \hat{O}_w^0, \hat{O}_w^1)$ we see from inspection of $\mathrm{RobCon}^{\mathcal{A}}_{\mathsf{IGarb}_\pi}(1^k)$ in Figure 17 that it is sufficient to prove that

$$\mathsf{IEv}(F, (\hat{K}_1^{x_1}, \ldots, \hat{K}_n^{x_n})) = (\hat{O}_{w-m+1}^{z_1}, \ldots, \hat{O}_w^{z_m}) \ .$$

where $z \leftarrow f(x)$ except with negl. probability in k.

We let (F, v) be the output of E after running IGb_π with $\mathcal{A}$. We then run $\mathsf{Ex_C}(\mathcal{A})$ to extract the encoding and decoding information $\hat{e}, \hat{d}$. Also we let x be the output of $\mathcal{A}$ and we assume that $x \in \{0,1\}^{f.n}$, since else there is nothing to show. By definition of $\mathsf{Ex_C}$ we have that $\hat{e} = (\hat{K}_1^0, \hat{K}_1^1, \ldots, \hat{K}_n^0, \hat{K}_n^1)$ and $\hat{d} = (\hat{O}_{w-m+1}^0, \hat{O}_{w-m+1}^1, \ldots, \hat{O}_w^0, \hat{O}_w^1)$ except with probability 2^{-s}.

By Lemma 14 we also have that the correct key is always output by a bucket (assuming it is given correct input keys) except with probability at most

$$q \cdot \left(\prod_{i=\beta}^{1} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) + \sum_{l=2}^{\beta} \prod_{i=\beta}^{l} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha+2-l} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) \right)$$

By Lemma 13 we also have that the solderings used to build F are correct, except with probability 2^{-s}. The above combined with the scheme having the corr property proves the claim. □

Lemma 11 (unqoe). *If* IGarb_π *has the unqie and rob.con properties, then the scheme has the unqoe property as well.*

Proof. Let $\hat{e}$ and $\hat{d}$ be the output of the extractor $\mathsf{Ex}_\mathsf{C}(\mathcal{A})$. Next, assume X is output by $\mathcal{A}$ in step 4 and that $\mathsf{IEv}(F, X) \to Z \neq \perp$. Since we are now in the same situation as in the game $\mathrm{UnqIE}^{\mathcal{A}}_{\mathsf{IGarb}_\pi}(1^k)$ a simple reduction with $I = \emptyset$ shows that $\mathsf{IEn}\left(\hat{e}, \mathsf{IEn}^{-1}\left(\hat{e}, X\right)\right) = X$, except with negl. probability in k since the scheme has the unqie property.

What remains to be shown is that $\mathsf{IDe}^{-1}\left(\hat{d}, \mathsf{IDe}\left(\hat{d}, Z\right)\right) = Z$, except with negl. probability in k. Because $\mathsf{IEn}\left(\hat{e}, \mathsf{IEn}^{-1}\left(\hat{e}, X\right)\right) = X$ we can view $x \leftarrow \mathsf{IEn}^{-1}(\hat{e}, X)$ as the output of $\mathcal{A}$. This means that we are in the same situation as in the game $\mathrm{RobCon}^{\mathcal{A}}_{\mathsf{IGarb}_\pi}(1^k)$ and thus it can be shown by a simple reduction that $f(x) = \mathsf{IDe}\left(\hat{d}, \mathsf{IEv}\left(F, \mathsf{IEn}\left(\hat{e}, x\right)\right)\right) = \mathsf{IDe}\left(\hat{d}, \mathsf{IEv}\left(F, X\right)\right) = \mathsf{IDe}\left(\hat{d}, Z\right)$ since the scheme has the rob.con property. In particular we have that $\mathsf{IDe}\left(\hat{d}, Z\right) \neq \perp$. Then it follows from the description of IDe in Figure 10 that $\mathsf{IDe}^{-1}\left(\hat{d}, \mathsf{IDe}\left(\hat{d}, Z\right)\right) = Z$. We thus conclude that if IGarb_π has the unqie and rob.con properties it also has the unqoe property. □

Lemma 12. IGarb_π *has property tok.com.*

Proof. Based on the output of the extractor $\mathsf{Ex}_\mathsf{C}(\mathcal{A})$ to extract $\hat{e}$, we argue that it is not possible for $\mathcal{A}$ to construct seemingly correct (in accordance with an execution of IGb_π and $\hat{e}$) values X, O, I and $\{(i, x'_i)\}$ such that $\mathsf{IEn}^{-1}(\hat{e}, X) \to x$ with $x'_i \neq x_i \neq \perp$ for some $i \in I$.

Let $\hat{e}$ and $\hat{d}$ be the output of $\mathsf{Ex}_\mathsf{C}(\mathcal{A})$. We then wish to show that it is not possible for $\mathcal{A}$ to produce X, O, I and $\{(i, x'_i)\}_{i \in I}$ with $I \subseteq [f.n]$ such that $x \leftarrow \mathsf{IEn}^{-1}(\hat{e}, x)$ with $x \neq \perp$, yet such that $\mathsf{IVe}(v, X, O, \{(i, x'_i)\}) = \top$ while there exists $i \in I$ such that $x_i \neq x'_i$. By the correctness of IEn this means that she can find X consistent with the values in $\hat{e}$ and also find an O such that IVe accepts a change of the semantic value of at least one key. Now since $\mathsf{Ex}_\mathsf{C}(\mathcal{A})$ extracted $\hat{e}$ through the ECC commitments we have from Lemma 13 that $\mathcal{A}$ cannot construct any opening to a value different from what was actually committed to except with probability 2^{-s}. This means that the only way she can succeed is to either open a commitment to a 0-key when the opening was supposed to be for a 1-key or vice versa. The opening to a 1-key will contain two indices, one for the 0-key and one for Δ, and the opening to a 0-key only one index. Furthermore, since we assume $\mathsf{lsb}(\Delta) = 1$ the message in these two commitments will be distinct. Thus this will also only be possible with probability 2^{-d}. Since we assume that $d = s + 4$ we get that this is negligible which concludes the proof.

□

We now prove the helper lemmas used in Lemma 10.

Lemma 13 (Binding of commitments). *In an execution of the interactive protocol* IGb_π*, using a binary and symmetric ECC* $[\Gamma, k, d]$ *where* $d = s + 4$*, after the first linear combination check,* $\mathcal{A}$ *can only open any one commitment to a single message except with probability* 2^{-s}*.*

Moreover, when opening to the XOR of any subset of these messages, $\mathcal{A}$ *can only open to the XOR of the fixed underlying messages except with probability* 2^{-s}*.*

Proof. As already argued in in the description of $\mathsf{Ex}_\mathsf{C}(\mathcal{A})$, we have that $\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k, r) \leq 2^{-4}2^{-s}$ as the code has minimum distance $d = s + 4$ and C asks $\mathcal{A}$ $7.3(s + 3)$ puzzles in each linear combination check. Both statements then follow from Theorem 2, as this gives us that we can extract a unique commitment to all individual wires except with probability 2^{-s} in each linear combination check execution.

□

Lemma 14 (Probability of corrupt bucket). *For any F output by IGb_π, we have that the* IEv *algorithm of Figure 10 always outputs the correct key for any bucket except with probability at most*

$$q \cdot \left(\prod_{i=\beta}^{1} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) + \sum_{l=2}^{\beta} \prod_{i=\beta}^{l} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha+2-l} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) \right)$$

Proof. Before discussing IGb_π, consider an experiment where a potentially malicious $\mathcal{A}$ playing the role of C is given two buttons at the start of the protocol for one arbitrary bucket, a corrupt gate button and corrupt authenticator button. The outcome of pressing each button will be either success, failure or neither, where success will mean a gate (wire) of the bucket will be arbitrarily defined by $\mathcal{A}$ as long as it has the correct form, failure will mean that a gate (wire) is selected for checking and does not pass the check and neither will mean that a gate (wire) is bad and is either not selected for checking and falls into another bucket than the one we look at, or it is selected for checking but passes the check. Furthermore we allow $\mathcal{A}$ to learn the outcome of pressing a button, before deciding on pressing either button once more. Also if the outcome is success Bob will not learn that the button was pressed. If at any time the result of either buttons is failure the experiment ends and Bob learns that $\mathcal{A}$ pressed the button.

Assuming, without loss of generality, that $\mathcal{A}$ starts by constructing all the bad gate (wires) then it should be clear that the above experiment is sufficient to model $\mathcal{A}$'s ability to corrupt gates or wires in an execution of IGb_π. In fact it gives her strictly more power as she can adaptively change her strategy based on the outcome of the current result of pressing the button. This means she has no additional risk of getting caught, once she deems her corruption strategy succeeded. This is in contrast to an execution of IGb_π where $\mathcal{A}$ must decide a priori which gates and wires to corrupt and after this she is committed to her choice (as she sends these objects to Bob). The event of failure will model the probability that $\mathcal{A}$ gets caught in either of the cut-and-choose checks performed by Bob in IGb_π. It is sufficient to only look at the experiment for a single bucket, because in IGb_π all buckets have the same probability of becoming corrupted and therefore we can use a simple union bound to bound the probability of any bucket becoming corrupted (we will specify what a corrupt bucket means later in the proof).

Before continuing we need to calculate the probability of the events success and failure when pressing the buttons such that they correctly model an execution of IGb_π. We first consider the gate button. Recall that a gate is chosen for checking in the cut-and-choose step of IGb_π with probability p_g. If a gate is corrupted and selected for checking we see that it will get caught with at least probability $\frac{1}{4}$, since it is checked on one random input out of four possible. Thus the probability of catching a corrupt gate is at least $p_g \cdot \frac{1}{4}$. For the same reason a gate is not chosen for check with probability $(1-p_g)$ and the probability a gate ends up in the bucket in question is therefore at most $(1-p_g) \cdot \frac{i}{q\beta}$ where i is the number of non-corrupt gate slots left in the bucket.[10] The decrease caused by i needs to be taken into account because each time a corrupt gate ends up in the bucket it takes up a slot, so there is less probability for corrupting the following gates as there are less free slots in the bucket.

As we only consider whether pressing the button results in success or failure we normalize these two outcomes as complementary events. We see that $p_g \cdot \frac{1}{4} = p_g \cdot \frac{q\beta}{4q\beta}$ and $(1-p_g) \cdot \frac{i}{q\beta} = (1-p_g) \cdot \frac{4i}{4q\beta}$. Multiplying both expressions by $4q\beta$ and dividing the success probability with the sum of the success and failure probability we see that the relative probability of success becomes

$$\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \tag{1}$$

where i is the number of non-corrupt gate slots in the bucket.

[10] This is an upper bound since there is a slight probability a corrupt gate (wire) will not be part of $\mathsf{EvalGates}$ ($\mathsf{AuthWires}$). Also we do not consider consider non-corrupt gates (wires) taking up any slots.

In an analogously way we determine the relative success probability for the authenticator button. The only difference is that here a corrupt authenticator is caught with probability at least $\frac{1}{2}$, because there are only two possible values as opposed to four the gates. By the same procedure as above we have that the relative probability of success becomes

$$\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \tag{2}$$

where j is the number of non-corrupt authenticator slots for the bucket.

First recall that we are in a setting where $\alpha = \beta - 1$. Let E_1 be the event that all β gates of the bucket become corrupt, E_2 the event that $\beta - 1$ gates and one authenticators become corrupt, E_3 the event that $\beta - 2$ gates and two authenticators become corrupt and so on until E_β which denotes the event that one gate and $\beta - 1$ authenticators become corrupt. We now claim that when evaluating F using IEv, all buckets will always output the correct key if none of the above events occur.

The reason for the bucket always outputting the correct key is that if the above cases are ruled out, there is always a correct majority when considering both authenticators and gates. Notice that we require that at least one gate be correct, since else we cannot guarantee that the correct key is part of the candidate output keys. Thus $\Pr[\text{corrupt bucket}] = \Pr[E_1 \vee E_2 \vee \cdots \vee E_\beta] \leq \sum_{b=1}^{\beta} \Pr[E_b]$ by the union bound.

We now turn to the calculation of this probability. From (1) and (2), the observation that the probabilities of bad gates and authenticators ending up in the same bucket are independent and by a union bound, we can conclude that there are no corrupt buckets after $\mathcal{A}$ has participated in the above mentioned experiment except with probability at most

$$\begin{aligned}
q \cdot \sum_{b=1}^{\beta} \Pr[E_b] = q \cdot \Bigg(& \prod_{i=\beta}^{1} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) + \\
& \prod_{i=\beta}^{2} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) + \\
& \prod_{i=\beta}^{3} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha-1} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) + \\
& \prod_{i=\beta}^{\beta} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{1} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) \Bigg) \\
= q \cdot \Bigg(& \prod_{i=\beta}^{1} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) + \\
& \sum_{l=2}^{\beta} \prod_{i=\beta}^{l} \left(\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \right) \cdot \prod_{j=\alpha}^{\alpha+2-l} \left(\frac{(1-p_a)2j}{p_a q\alpha + (1-p_a)2j} \right) \Bigg)
\end{aligned}$$

As we already argued that in the experiment $\mathcal{A}$ is given more power than in an execution of IGb_π the statement follows. □

E Hard-core Puzzles and Solutions

In this section we prove a theorem which is best seen as a variant of the hard-core bit theorem. The hard core of the hard-core bit theorem say that if you consider a secret value $x \in \{0,1\}^\ell$ and you are given an algorithm R which for uniformly random challenges $\pi \in \{0,1\}^\ell$ can return you the inner product $\sigma = \langle x, \pi \rangle$ with a probability which is noticeably better than random guessing, then you can compute x using a polynomial number of queries to R. We are going to generalize this to linear functions returning longer outputs than the single bit returned by the inner product. We phrase this in terms of linear puzzles. This is not a dramatic generalization, as it can be seen as several applications of the one-bit hard-core bit theorem using the same challenge π for several instances, which is known to work. However, after generalizing to longer replies, we complicate things by allowing that each challenge has many correct answers. I.e., for each challenge $\pi \in \{0,1\}^\ell$, we will allow that there exists several values σ such that σ is a correct reply to π. In the hard-core bit theorem setting there is only one correct answer to π, namely the inner product. Clearly, if any σ is a correct answer, then we cannot extract x from an algorithm R returning correct answers, as R can easily be constructed without knowing x. It turns out, however, that if we make the assumption that it is hard to find two distinct correct replies to some chosen π, then we can still prove a hard-core bit like result. The reduction is considerably complicated by the fact that the reduction cannot test whether a reply is correct. To make the reduction work, we therefore set it up such that *either* a hard-core bit theorem like reduction works *or* among the replies returned by R there is a large fraction of distinct correct answers to the same challenge. Then we can make a guess at where the collision is sitting and make a reduction to the hardness of computing different correct replies to the same challenge. The formal details follow now.

A *puzzle system* $\Pi = (\mathsf{Gen}, \mathsf{Ver})$ consists of two algorithms. A randomised algorithm Gen called the *puzzle generator*. It takes as input the security parameter k. It outputs $(\mathsf{pk}, \mathsf{vk})$, where pk is the *puzzle key* and vk is the *verification key*. And a deterministic algorithm Ver. It takes as input vk, a *puzzle identifier* $\pi \in \{0,1\}^\ell$ and a *potential solution* $\sigma \in \{0,1\}^*$. Here ℓ is an integer fixed by the system, specifying the length of the puzzles. It outputs $\bot$ or $\top$. If $\mathsf{Ver}(\mathsf{pk}, \pi, \sigma) = \top$, then we say that σ is a *solution* π. We say that $(\mathsf{Gen}, \mathsf{Ver})$ is $\oplus$-*homomorphic* if $\mathsf{Ver}(\mathsf{vk}, \pi_1, \sigma_1) = \top$ and $\mathsf{Ver}(\mathsf{vk}, \pi_2, \sigma_2) = \top$ implies that $\mathsf{Ver}(\mathsf{vk}, \pi_1 \oplus \pi_2, \sigma_1 \oplus \sigma_2) = \top$. We only consider $\oplus$-homomorphic puzzle systems below.

For $i = 1, \ldots, \ell$, let $e_i = 0^{i-1}10^{\ell-i}$ such that $e_1, \ldots, e_\ell$ is the canonical basis for $\{0,1\}^\ell$. We call $e_1, \ldots, e_\ell$ the basis puzzles. Note that given a solution to all basis puzzles in an $\oplus$-homomorphic puzzle system, one can efficiently compute the solution to any other puzzle in the system. This gives rise to a notion of solver, which is an algorithm that can compute solutions to all basis puzzles, as defined now. We use $\mathsf{Sol}(\mathsf{vk})$ to denote the set of all vectors $(\sigma_1, \ldots, \sigma_\ell)$ for which $\wedge_{i=1}^{\ell} \mathsf{Ver}(\mathsf{vk}, e_i, \sigma_i) = \top$. A *solver* is a randomized algorithm S. It takes as input pk and it outputs $(\sigma_1, \ldots, \sigma_\ell)$. We use $\mathbf{Adv}^{\mathrm{sol}}_{\Pi,S}(k)$ to denote the probability that S solves Π, i.e., the probability that $S(\mathsf{pk}) \in \mathsf{Sol}(\mathsf{vk})$ when $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$.

We use $\mathsf{Dou}(\mathsf{vk})$ to denote the set of (i, σ, σ') for which $\mathsf{Ver}(\mathsf{vk}, \pi_i, \sigma) = \top$ and $\mathsf{Ver}(\mathsf{vk}, \pi_i, \sigma') = \top$ and $\sigma' \neq \sigma$. A *double solver* is a randomized algorithm D. It takes as input pk and it outputs (i, σ, σ'). We use $\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k)$ to denote the probability that D solves a puzzle in two different ways, i.e., the probability that $D(\mathsf{pk}) \in \mathsf{Dou}(\mathsf{vk})$ when $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$.

We finally introduce the notion of a random solver, which is an algorithm which solves a number of random puzzles. Let r be an integer, the number of given random puzzles. We use $\mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_r)$ to denote the set of all vectors $(\sigma_1, \ldots, \sigma_r)$ for which $\wedge_{i=1}^{r} \mathsf{Ver}(\mathsf{vk}, \pi_i, \sigma_i) = \top$. A *random solver* is a randomized algorithm R. It takes as input pk and $\pi_1, \ldots, \pi_r$. It outputs $(\sigma_1, \ldots, \sigma_r)$. We use $\mathbf{Adv}^{\mathrm{ran}}_{\Pi,R}(k, r)$ to denote the probability that R solves r random puzzles, i.e., the probability that $R(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_r)$ when $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$ and each π_i is sampled uniformly at random from $\{0,1\}^\ell$.

We now show that if there exists a poly-time random solver, then there also exists a poly-time solver, unless it is easy to find double solutions.

Lemma 15. *For all $0 < \alpha < \frac{1}{2}$ and for all $0 < \beta < \frac{1}{4}$ and all poly-time R and integers r there exists poly-time S and poly-time D such that*

$$\mathbf{Adv}^{sol}_{\Pi,S}(k) \geq \mathbf{Adv}^{ran}_{\Pi,R}(k, r) - \epsilon^{-1}\mathbf{Adv}^{dou}_{\Pi,D}(k) - 2^{-s-3} \ ,$$

where $s+3 = (-\log_2((\frac{1}{2}+\alpha)^{(\frac{1}{2}-\beta)}))r$ *and* $\epsilon = \min\left(\beta, \frac{4\alpha-1}{\frac{1}{2}+\alpha}\right) - 2^{-2s}$.

Proof. The proof will have the following form. From any random solver R we construct S which can take R in a state where it just successfully answered r random challenges and extract from R using a polynomial number of runs of R a correct solution to all basis puzzles. There will be a small probability 2^{-s-1} that R answers r random puzzles correctly and that S cannot extract, which explains the 2^{-s-1} occurring on the RHS of the bound.[11] The reduction can also fail if R can compute distinct correct solutions to the same puzzle, but then S can do the same, with probability ϵ, which explains the middle term occurring on the RHS of the bound.

So, assume we sample $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(k)$ and uniformly random $(\pi_1, \ldots, \pi_r)$ and $(\sigma_1, \ldots, \sigma_r) \leftarrow R(\pi_1, \ldots, \pi_r)$. It is sufficient to prove that under the assumption that $(\sigma_1, \ldots, \sigma_r) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_r)$ we can compute all solutions, except with probability $\epsilon^{-1}\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k) - 2^{-s-1}$.

We first argue that it happens with probability at most 2^{-s-1} that a random run with puzzles $(\pi_1, \ldots, \pi_r)$ is accepting and does not have the property that there exists a $(\frac{1}{2}+\alpha)$-fraction of positions i such that if one reruns R on input $(\pi_1, \ldots, \pi_i, \pi'_{i+1}, \ldots, \pi'_r)$ (for uniformly random π'_j for $j = i+1, \ldots, r$), then R returns an elements from $\mathsf{Sol}(\mathsf{vk}, (\pi_1, \ldots, \pi_i, \pi'_{i+1}, \ldots, \pi'_r))$ with probability at least $\frac{3}{4}+\beta$.

To be more precise, given any $\pi = (\mathsf{pk}, \pi_1, \ldots, \pi_r)$ for verification key vk, we call position i bad if $\Pr[R(\mathsf{pk}, \pi_1, \ldots, \pi_i, \pi'_{i+1}, \ldots, \pi'_r) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_i, \pi'_{i+1}, \ldots, \pi'_r)] \leq \frac{3}{4}+\beta$ when each π'_i is sampled uniformly at random from $\{0,1\}^\ell$. We say $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_I(\mathsf{vk})$ if it both holds that there are at least I bad positions in $(\mathsf{pk}, \pi_1, \ldots, \pi_r)$ and yet $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_r)$. The claim is that

$$\Pr[(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{r(\frac{1}{2}-\alpha)}(\mathsf{vk})] \leq 2^{-s} \ ,$$

when $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$ and each π_i is sampled uniformly at random from $\{0,1\}^\ell$.

We first analyse the probability that $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{I+1}(\mathsf{vk})$ conditioned on $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_I(\mathsf{vk})$. Notice that for any I and $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_I(\mathsf{vk})$ it follows that there is an I'th bad position. Consider any I and any $(\mathsf{pk}, \pi_1, \ldots, \pi_j) \in \mathsf{Bad}_I(\mathsf{vk})$, where $j \leq r$ is the I'th bad position. To have that $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j+1}, \ldots, \pi'_r) \in \mathsf{Bad}_{I+1}(\mathsf{vk})$ we must get a bad position for some $j' > j$. This means that $r > j$. Since position j is bad it follows that for fixed $(\mathsf{pk}, \pi_1, \ldots, \pi_j)$ and a uniformly random π'_{j+1} it holds that $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j+1}) \in \mathsf{Bad}_I(\mathsf{vk})$ with probability at most $\frac{1}{2}+\alpha$, as $\mathsf{Bad}_I(\mathsf{vk}) \subset \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_j, \pi'_{j+1})$ and $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j+1}) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_j, \pi'_{j+1})$ with probability at most $\frac{1}{2}+\alpha$, by definition of position j being bad. Since $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j-1}) \in \mathsf{Bad}_{I+1}(\mathsf{vk})$ implies that $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j+1}) \in \mathsf{Bad}_I(\mathsf{vk})$ it follows for all $j' > j$ that the probability that $(\mathsf{pk}, \pi_1, \ldots, \pi_j, \pi'_{j+1}, \ldots, \pi_{j'}) \in \mathsf{Bad}_{I+1}(\mathsf{vk})$ is less than $\frac{1}{2}+\alpha$. Therefore

$$\Pr[(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{I+1}(\mathsf{vk}) | (\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_I(\mathsf{vk})] \leq \frac{1}{2} + \alpha \ .$$

Since $y \in \mathsf{Bad}_{I+1}(\mathsf{vk})$ implies that $y \in \mathsf{Bad}_I(\mathsf{vk})$ it follows by induction that

$$\Pr[(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{r(\frac{1}{2}-\beta)}(\mathsf{vk}) | (\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_0(\mathsf{vk})] \leq (\frac{1}{2} + \alpha)^{r(\frac{1}{2}-\beta)} \ .$$

Since $\Pr[y \in \mathsf{Bad}_0(\mathsf{vk})] = 1$, it follows that

$$\begin{aligned} &\Pr[(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{r(\frac{1}{2}-\beta)}(\mathsf{vk})] = \\ &\Pr[(\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_{r(\frac{1}{2}-\beta)}(\mathsf{vk}) | (\mathsf{pk}, \pi_1, \ldots, \pi_r) \in \mathsf{Bad}_0(\mathsf{vk})] \ . \end{aligned}$$

We can therefore continue the analysis under the assumption that the puzzles in $(\mathsf{pk}, \pi_1, \ldots, \pi_r) \notin \mathsf{Bad}_{r(\frac{1}{2}-\beta)}(\mathsf{vk})$ and prove that in that case we can compute all solutions, except with probability $\epsilon^{-1}\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k)$.

Consider the following algorithm $B_{(\mathsf{pk}, \pi_1, \ldots, \pi_r), i}(\hat{\pi})$. It takes as input $(\mathsf{pk}, \pi_1, \ldots, \pi_r)$ and $i \in \{1, \ldots, r\}$ along with a challenge $\hat{\pi}$ to which it will try to compute a solution. It samples uniformly random $(\pi'_{i+1}, \ldots, \pi'_r)$, samples

$$(\sigma_1, \ldots, \sigma_r) \leftarrow R(\mathsf{pk}, \pi_1, \ldots, \pi_i, \pi'_{i+1}, \ldots, \pi'_r)$$

[11]

$$(\bar{\sigma}_1, \ldots, \bar{\sigma}_r) \leftarrow R(\mathsf{pk}, \pi_1, \ldots, \pi_i, \pi'_{i+1} \oplus \hat{\pi}, \ldots, \pi'_r) \ ,$$

and outputs $\sigma_{i+1} \oplus \bar{\sigma}_{i+1}$. Consider now any $(\mathsf{pk}, \pi_1, \ldots, \pi_r)$ and assume that the i is such that position i in $(\mathsf{pk}, \pi_1, \ldots, \pi_r)$ is not bad. Then it holds that

$$\Pr[\mathsf{Ve}(\mathsf{vk}, \hat{\pi}, B_{(\mathsf{pk},\pi_1,\ldots,\pi_r),i}(\hat{\pi}) = \top] \geq \frac{1}{2} + 2\beta \ . \tag{3}$$

To see this, note when i is not bad, then σ_{i+1} is a solution to π'_{i+1} with probability at least $\frac{3}{4} + \beta$ and $\bar{\sigma}_{i+1}$ is a solution to $\pi'_{i+1} \oplus \hat{\pi}$ with probability at least $\frac{3}{4} + \beta$. Hence, by a union bound, the probability that both are correct is at least $1 - (\frac{1}{4} - \beta) - (\frac{1}{4} - \beta)$. Then apply that the puzzle is $\oplus$-homomorphic.

We would like to amplify the above algorithm such that when i is a good position, then it computes a correct solution except with negligible probability. To this end, notice that in the bound in (3) the probability is over the random values sampled by the algorithm. We can therefore run the algorithm a number of times until we have a majority of correct solutions except with negligible probability. However, we cannot just take majority at this point, as the correct solutions might not be identical. We therefore adopt a slightly more complicated strategy. Consider the following algorithm $F_{(\mathsf{pk},\pi_1,\ldots,\pi_r)}(\hat{\pi})$. Pick γ such that if you flip γ independent coins which all come out head with probability at least $\frac{1}{2} + 2\beta$ then there will be a fraction of $\frac{1}{2} + \beta$ heads except with probability $2^{-2s}/(r\ell)$. Then for $i = 1, \ldots, r$, sample $\sigma_i \leftarrow B_{(\mathsf{pk},\pi_1,\ldots,\pi_r),i}(\hat{\pi})$ until having γ samples. If there is a value occurring more than $(\frac{1}{2} - \beta)\gamma$ times among these γ samples, then let σ_i be that that value, otherwise let $\sigma_i = \perp$. Then, if there is a value σ occurring $(\frac{1}{2} - \alpha)r$ times in the list $\sigma_1, \ldots, \sigma_r$, then output that value. Otherwise $\perp$.

We first argue that for all runs of F, if it does not output $\perp$, then the output is correct except with probability $2^{-2s}/\ell$. We have chosen γ such that when i is good, then there are at least $(\frac{1}{2} + \beta)\gamma$ correct samples except with probability $2^{-2s}/r\ell$. Since we produce such pools of γ samples r times, a union bound allows us to ignore the event that for a run on some good i there are not $(\frac{1}{2} + \beta)\gamma$ correct samples. It then holds for all good i that any value occurring more than $(\frac{1}{2} - \beta)\gamma$ times is correct. Therefore, when i is good, then σ_i is correct when $\sigma_i \neq \perp$. Furthermore, if a value σ occurs more than $(\frac{1}{2} - \alpha)r$ times in the list, then it was output by at least one run of F on a good position i, as there are $(\frac{1}{2} + \alpha)r$ good positions.

The algorithm S then works as follows: For $j = 1, \ldots, \ell$, compute the solution $\sigma_j \leftarrow F(e_j, \mathsf{pk}, \pi_1, \ldots, \pi_r)$. By a union bound, it holds for all σ_j that if $\sigma_j \neq \perp$, then σ_j is correct, except with probability 2^{-2s}. It is therefore sufficient to prove that the probability that $\sigma_j \neq \perp$ is at most $\min\left(\beta, \frac{4\alpha-1}{\frac{1}{2}+\alpha}\right)$.

We break the analysis up into two cases.

1. Among the r values σ_i at least $2(\frac{1}{2} - \alpha)r$ of them outputs $\perp$.
2. Among the r values σ_i less than $2(\frac{1}{2} - \alpha)r$ of them outputs $\perp$.

We first analyse Case 1. Since we are analysing under the assumption that there are at most $(\frac{1}{2} - \alpha)r$ bad positions, we know that among the positions i for which $\sigma_i = \perp$, at least half of them are good. So, if we sample a random i for which $\sigma_i = \perp$, then i is good with probability at least i. Assume then that i is good. We are analysing under the assumption that among the γ runs of $B_{(\mathsf{pk},\pi_1,\ldots,\pi_r),i}(\hat{\pi})$ at most $(\frac{1}{2} - \beta)\gamma$ are bad. That means that if we pick a random of these outputs, call it $\sigma_{i,1}$ then it is good with probability at least $\frac{1}{2} + \beta$. Assume that it is good. Since $\sigma_i = \perp$ no sample occurs more than $(\frac{1}{2} - \beta)\gamma$ times, so in particular $\sigma_{i,1}$ occurs at most $(\frac{1}{2} - \beta)\gamma$ times. That means that if we sample a uniform other output among the samples that are *different* from $\sigma_{i,1}$, then we get another *good* sample with probability at least $\frac{(\frac{1}{2}+\beta)\gamma - (\frac{1}{2}-\beta)\gamma}{\gamma - (\frac{1}{2}-\beta)\gamma} = \frac{2\beta}{\frac{1}{2}+\beta}$. This gives two distinct opening with probability at least $\frac{1}{2}(\frac{1}{2} + \beta)\frac{2\beta}{\frac{1}{2}+\beta} = \beta$.

We then analyse Case 2. In this case, at least $2\alpha r$ of the values σ_i are different from $\perp$. Of these values at most $(\frac{1}{2} - \alpha)r$ are bad. So, there are at least $2\alpha r - (\frac{1}{2} - \alpha)r = (3\alpha - \frac{1}{2})r$ good position different from $\perp$. Sample a random position and assume that σ_i is correct. This happens with probability at least $(3\alpha - \frac{1}{2})$. Now sample a random σ_j for which $\sigma_j \neq \perp$ and $\sigma_j \neq \sigma_i$. As σ_i occurs at most $(\frac{1}{2} - \alpha)r$ times the second value will be correct too with probability at least $\frac{(3\alpha - \frac{1}{2}) - (\frac{1}{2} - \alpha)r}{r - (\frac{1}{2} - \alpha)r} = \frac{4\alpha - 1}{\frac{1}{2} + \alpha}$

□

We now define a game played between a random solver R which has to answer r random puzzles and a solver which has to solve all basis puzzles. The random solver wins if it can answer the r puzzles and S cannot solve all basis puzzles. The solver is given the state of R after an accepting run.

Sample $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$ and then sample $\pi_i \in \{0,1\}^\ell$ uniformly at random for $i = 1, \ldots, r$. Then sample $(\sigma_1, \ldots, \sigma_r) \leftarrow R(\mathsf{pk}, \pi_1, \ldots, \pi_r)$ and let ρ denote the randomness used by R. Then sample $(\hat{\sigma}_1, \ldots, \hat{\sigma}_\ell) \leftarrow S(\mathsf{pk}, \pi_1, \ldots, \pi_r, \rho)$. Define $\mathbf{Adv}^{\mathrm{sol,ran}}_{\Pi,R,S}(k,r)$ to be the probability that $(\sigma_1, \ldots, \sigma_r) \in \mathsf{Sol}(\mathsf{vk}, \pi_1, \ldots, \pi_r)$ and $(\hat{\sigma}_1, \ldots, \hat{\sigma}_\ell) \notin \mathsf{Sol}(\mathsf{vk})$, *i.e.* the probability that $(\sigma_1, \ldots, \sigma_r)$ is a solution to the r random puzzles and $(\hat{\sigma}_1, \ldots, \hat{\sigma}_\ell)$ is not a solution to all basis puzzles.

Theorem 2. *Assume that* $\mathbf{Adv}^{dou}_{\Pi,D}(k,r) \leq 2^{-4}2^{-s}$ *for all poly-time* D*. Assume that* $s \geq 3$*. Let* $r = 7.3(s+3)$*. Then for all poly-time* R *there exists poly-time* S *such that*

$$\mathbf{Adv}^{sol,ran}_{\Pi,R,S}(k,r) \leq 2^{-s} \ .$$

Proof. Note that the S constructed in the above lemma has the required form. Then let $\beta = \frac{1}{8}$ and set $\alpha = \frac{17}{62}$. Then $r = (-\log_2((\frac{47}{62})^{\frac{3}{8}}))^{-1}(s+3) < 7.3s + 22$ and $\epsilon = 2^{-3} - 2^{-2s}$ and therefore $\epsilon^{-1}\mathbf{Adv}^{\mathrm{dou}}_{\Pi,D}(k) - 2^{-s-3} \leq (2^{-3} - 2^{-2s})^{-1}2^{-4}2^{-s} + 2^{-s-3} \leq 2^{-s}$, where the last inequality is true for all $s \geq 3$ □

F Ideal Functionalities

In this section we define the ideal functionalities we need in our protocol.

Functionality $\mathcal{F}_{\mathsf{SFE}}^{f}$

Setup: We denote the two parties of the protocol by A and B. The parties agree on k and f. The parties also agree on n_{A} and n_{B} such that $f.n = n_{\mathsf{A}} + n_{\mathsf{B}}$.
Input A: The ideal functionality takes exactly one input $x \in \{0,1\}^{n_{\mathsf{A}}}$ from A.
Input B: The ideal functionality takes exactly one input $y \in \{0,1\}^{n_{\mathsf{B}}}$ from B.
Abort: If any corrupted party inputs abort, then output abort to the other party and terminate.
Corrupt A: On input ($\mathsf{corrupt}$) from A, let her be corrupt. She can then specify a set $\{(i, \beta_i)\}_{i \in I}$, where $I \subseteq \{1, \ldots, n_{\mathsf{B}}\}$ and $\beta_i \in \{0,1\}$. If $\beta_i = y_i$ for $i \in I$, then output $\mathsf{correct!}$ to A. Otherwise, output abort to both parties and terminate.
Evaluation: If both parties gave input, then on input $\mathsf{evaluate}$ from the adversary, compute $z \leftarrow f(x, y)$ and output z to A.

Fig. 21. Ideal Functionality $\mathcal{F}_{\mathsf{SFE}}^{f}$.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Initialization
Both P_1 and P_2 input (init).
Transfer
- Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, b)$ where $b \in \{0,1\}$ from P_1, if no tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, \cdot)$ is stored, then store the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, b)$ and output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{bit\ received})$ to P_2. Otherwise replace the stored tuple with $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, \perp)$.
- Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{sender}, m_0, m_1)$ where $m_0, m_1 \in \{0,1\}^k$ from P_2, if a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, b)$ is stored where none of the elements is the symbol $\perp$, then output $(\mathsf{sid}, \mathsf{ssid}, m_b)$, otherwise output nothing. Finally, update the tuple such that what is now stored is $(\mathsf{sid}, \mathsf{ssid}, \mathsf{receiver}, \perp)$.

Corruption
A corrupted player can input abort at any time, in which case $\perp$ is output to the honest player and the functionality no longer responds to input.

Fig. 22. Ideal Functionality $\mathcal{F}_{\mathsf{OT}}$.

Functionality $\mathcal{F}_{\mathsf{COM}}$

Initialization

Both P_1 and P_2 input (init).

Input

- Upon receiving the command $(\mathsf{commit}, \mathsf{sid}, \mathsf{ssid}, m)$ from either P_1 or P_2, if $(\mathsf{commit}, \mathsf{sid}, \mathsf{ssid}, \cdot)$ has not been received before, then the functionality stores $(\mathsf{sid}, \mathsf{ssid}, m, \text{name})$ where name $= \mathsf{P}_1$ if P_1 is the party sending the command and name $= \mathsf{P}_2$ otherwise. The functionality then outputs $(\mathsf{commit}, \mathsf{sid}, \mathsf{ssid})$ to both parties.

Output

Upon receiving $(\mathsf{open}, \mathsf{sid}, \mathsf{ssid})$ from P_1 if a commitment $(\mathsf{sid}, \mathsf{ssid}, \cdot, \mathsf{P}_1)$ is stored then the functionality outputs $(\mathsf{open}, \mathsf{sid}, \mathsf{ssid}, m)$ to P_2. Similarly upon receiving $(\mathsf{open}, \mathsf{sid}, \mathsf{ssid})$ from P_2 if a commitment $(\mathsf{sid}, \mathsf{ssid}, \cdot, \cdot, \mathsf{P}_2)$ is stored then the functionality outputs $(\mathsf{open}, \mathsf{sid}, \mathsf{ssid}, m)$ to P_1.

Fig. 23. The Ideal Functionality $\mathcal{F}_{\mathsf{COM}}$.

G MiniLEGO Recap

The MiniLEGO protocol described in [FJN+13] assumes access to a XOR-Homomorphic Commitment (XHC) functionality. The authors describe a possible realization based on OT and an Error Correcting Code (ECC).

Free-XOR Both MiniLEGO and our protocol have the constraint that the XOR of both the 0- and 1-key on any wire of any garbled gate in the circuit yields the same value, Δ, which we call the *global difference*. This constraint comes from the free-XOR optimization and makes it possible to avoid garbling XOR gates, since such a gate can be computed locally by simply letting the output 0-key be defined as the XOR of the two input 0-keys. That is, for any j being a wire index we have that $K_j^1 = K_j^0 \oplus \Delta$ and in turn that for any gate g:

$$L_g^a \oplus R_g^b = \left(L_g^0 \oplus R_g^0\right) \oplus (a \oplus b)\Delta \ = O_g^{a \oplus b}.$$

We notice that this also makes it possible to do free XOR gates by simply adding an extra wire to the circuit with a static 1-key. When we wish to compute the negation of any wire, we simply XOR it with the static wire containing the 1-key.

Soldering The free-XOR optimization furthermore makes it easy to solder wires of gates together which is needed in a LEGO protocol. More specifically what we mean when we say that we solder two wires together is that we release an auxiliary value, called the *soldering*, that can transform the key representing bit b on one wire into the key representing bit b on another. More concretely, assume we wish to solder the output wire of gate g to the left input wire of gate $g+1$. To do so we release the value $S_{g+1}^L = O_g^0 \oplus L_{g+1}^0$. When gate g outputs the key representing the bit b then it is easy to learn the left b-key for gate $g+1$. Specifically it can be computed as follows:

$$L_{g+1}^b = O_g^b \oplus S_{g+1}^L = \left(O_g^0 \oplus (b \cdot \Delta)\right) \oplus O_g^0 \oplus L_{g+1}^0 = L_{g+1}^0 \oplus (b \cdot \Delta) \ .$$

This obviously generalizes when one wishes to solder together several different wire, e.g. if we wish to solder the output wire of gate g to the left input wire of gate $g+1$, $g+2$ and $g+3$, it is enough to release the values:

$$S_{g+1}^L = O_g^0 \oplus L_{g+1}^0, \quad S_{g+2}^L = O_g^0 \oplus L_{g+2}^0, \quad S_{g+3}^L = O_g^0 \oplus L_{g+3}^0 \ .$$

It is also easy to "inject" XOR gates into the soldering: Say we wish to compute the XOR of the output of gate g and $g+1$ and solder the result to the left wire of gate $g+2$ we simply release the value $S_{g+2}^L = \left(O_g^0 \oplus O_{g+1}^0\right) \oplus L_{g+2}^0$. In general we simply let the soldering be the XOR of the 0-keys of the wires we wish to XOR together and the 0-key of the wire we wish to solder onto.

Informal MiniLEGO Description The MiniLEGO protocol can informally (when abstracting away some details) be described by the following steps:

Setup The XHC functionality is initialized and Bob makes commitments to a randomly chosen cut-and-choose challenge, along with a random function for combining the gates into buckets and these buckets into the circuit they wish to compute. The cut-and-choose challenge consists of a partitioning of all the garbled gates into two equally sized sets, one as a set of check gates and the other as a set of evaluation gates. The check gates are used to verify that Alice has constructed a large amount of the gates honestly, and the evaluation gates are combined into buckets to form the fault tolerant circuit. Furthermore, the challenge contains two random bits for each of the check gates which need to be evaluated on as part of the opening.[12] Bob then uses a special OT method of the XHC functionality to commit to the bits of to the circuit and Alice makes an XHC to the global difference, Δ.

[12] Notice that checking a gate on more than one input combination would leak the global difference, which completely compromises the security.

Garbling Alice picks a global difference Δ and constructs the garbled AND gates using Δ, by picking random 0-keys for the left and right input wires of each gate. She further constructs XHC to the 0-keys (for both the left, right and output wire) for each of the garbled gates. She then sends each of the garbled gates to Bob.

Cut-and-choose Bob sends the opening to the commitment of his cut-and-choose challenge. In response Alice opens the commitments to *one* of the left-, right- and output-keys of each of the garbled gate Bob chose to check. In particular, whether she opens to the 0-key or the 0-key XOR Δ, i.e. the 1-key, depends on Bob's choice of bits for each specific garbled gate. Thus if Bob's choice of random bits for a particular gate g is a and b, Alice will open $L_g^0 \oplus a \cdot \Delta$, $R_g^0 \oplus b \cdot \Delta$ and $O_g^0 \oplus (a \wedge b) \cdot \Delta$. Using each pair of input keys, Bob evaluates each check gate and verifies that the output key resulting form the evaluation is the same as Alice opened to.

Soldering Based on Bob's choice of how the gates should be combined into buckets, Alice uses the XHC to open the XOR of appropriate 0-keys. This will enable Bob to solder the keys of the individual garbled gates together to form buckets, and then solder the buckets together to form a fault tolerant garbled circuit. In particular he solders the wires of each of the gates in a bucket to the lexicographically first gate in the bucket. In this way a bucket only contains one wire for the left, right and output of the gate it is supposed to compute. The buckets are then soldered together, using the wires of their lexicographically first gates, to construct a fault tolerant circuit.

Input Alice uses a special command of the XHC scheme to open either the 0- or 1-key of her input wires to the circuit, in correspondence with her plain input bits. For each of Bob's input bits he will receive the keys in correspondence with his input bits, which he committed to through the special OT functionality in the XHC scheme.

Evaluate Using the input keys Bob evaluates the entire garbled circuit, taking the majority of outputs within a given bucket to be the output of that bucket. In the end he sends the keys of the output wires of the circuit to Alice, who then outputs the bits they correspond to, or aborts if any of the keys are incorrect.

To avoid confusion we stress that the construction for realizing the XHC in MiniLEGO is not the same as the optimized commitment scheme presented in this work, although our construction also relies on an ECC and OT. We now give some more details on how the above soldering process is performed in MiniLEGO and our protocol.

H Detailed Efficiency Count

We count concretely the amount of bits that Alice sends to Bob during the protocol. We ignore any terms that do not depend on the circuit size.

H.1 Counting TinyLEGO Communication

For each original gate Alice must send $\beta/(1-p_g-\delta_g)$ garbled gates. Using the recent optimized garbled row reduction [ZRE14] each gate consists of $2k$ bits. In addition, for each gate Alice also sends k bits for correcting the output wire and she must do three codeword corrections to turn the left, right, and output keys into codes. For each original gate, Alice also needs to prepare $\alpha/(1-p_a-\delta_a)$ wire authenticators, each authenticator involving sending $2k$-bit hash values and one codeword correction. The communication required for codeword correction depends on the code used, which in turn depends on the security parameter. We recall that due to the linear combination checks we need a code with minimum distance $d \geq s+4$ for security 2^{-s}, see Theorem 2 in Appendix E for details. For $k = 128$ and $s = 40, 60, 80$ one can use the shortened BCH codes $[317, 128, 45]$, $[398, 128, 65]$, $[437, 128, 85]$, respectively. These codes were found using the BCH encoder/decoder program available at the website of [MZ06]. Other codes found in the MinT database [SS06, SS10] yield even less communication, *e.g.* $[279, 128, 44]$, $[358, 128, 64]$. In our count, we use the most communication efficient codes we could find. In the cut-and-choose phase, using linear combinations, expected βp_g gates are checked and for each Alice sends $2k$ bits. Similarly, in the cut-and-choose of authenticators, Alice checks αp_a authenticators, sending k bits for each. Finally, soldering requires Alice to send $3(\beta-1)k + \alpha k + 2k$ bits to Bob for each original gate.

We let $X(k, s)$ denote the amount of extra bits that the code requires for k and s and from the above we get $\beta(3k + 3X(k, s))/(1-p_g-\delta_g) + \alpha(2k + X(k, s))/(1-p_a-\delta_a) + \beta p_g 2k + \alpha p_a k + 3(\beta-1)k + \alpha k + 2k$ as the total amount of bits sent pr. original gate. Using the best codes we could find we have $X(128, 40) = 151$, $X(128, 60) = 230$, and $X(128, 80) = 309$.

H.2 TinyLEGO concrete parameters

This section shows suggested parameters for various values of statistical security s and circuit size q while keeping $k = 128$ fixed. These values are not necessarily optimal, but simply found by running a script searching over a subset of reasonable parameters. The table shows the best parameters found, *i.e.*, the parameters that result in the least communication overhead (the least amount of data that Alice must send to Bob, ignoring overhead that does not depend on circuit size).

For each s and q, $q_{\min}$ is the minimal circuit size that achieves statistical security s with α, β, p_a, and p_g. The last two columns show bits send per gate and total amount of gibits (*i.e.*, 2^{30} bits) send from Alice to Bob. We remark that some of these parameters do not follow the form of $\alpha = \beta - 1$ which is what is assumed in Appendix D. However by tweaking the rule for evaluation one can make these parameters work as well.

H.3 Counting MiniLEGO Communication

We here give a description of how we calculated the communication in the MiniLEGO protocol. We omit all data sent that does not directly depend on the circuit size q. In the following we let $\mathcal{C}' = [n', k, d]$ be the ECC code proposed in [FJN+13] and ψ be the length of the randomness used for each encoding. First let $\Gamma = 2\beta' q$ and $\Gamma' = 3\Gamma + 1$ as described in the protocol where β' is the required bucket size for the majority buckets. Following Figure 7, 9 and 11 of [FJN+13] we see that in the setup phase Alice sends to Bob $2\Gamma' n' + \Gamma'(k + \psi) + \Gamma' k + \Gamma'(k + \psi)$ bits. The first term is the setup of the commitments, the second, also in setup, is the expected number of Bob's challenges that needs to be opened to, the third is the actual commitments sent and the last term is the openings to the XORs defined by the pairing π. In the garbling step Alice sends $2\Gamma k + 3\Gamma k$ bits where the first term is the two cipher texts representing the gate (we assume for a fair comparison that they use [ZRE14] for garbling as well) and the second is the commitments to the

s	q	$q_{\min}$	α	β	p_a	p_g	bits/gate	total gibits
40	1,000	866	8	7	3/20	1/4	16,463	0.015
40	10,000	9,166	5	6	1/10	3/20	11,425	0.11
40	100,000	92,678	4	5	1/20	1/10	8,859	0.83
40	200,000	165,232	4	5	1/20	1/20	8,540	1.6
40	400,000	375,640	3	4	3/20	1/5	7,696	2.9
40	600,000	514,297	3	4	3/20	3/20	7,395	4.1
40	800,000	733,960	3	4	1/5	1/10	7,231	5.4
40	1,000,000	953,022	3	4	1/10	1/10	7,021	6.5
40	2,000,000	1,827,303	3	4	3/20	1/20	6,869	12.8
40	4,000,000	2,580,996	3	4	1/20	1/20	6,678	24.9
40	6,000,000	4,059,866	3	4	1/50	1/20	6,626	37.0
40	8,000,000	6,809,984	3	4	1/20	1/50	6,537	48.7
40	10,000,000	9,128,099	3	4	1/50	1/50	6,486	60.4
40	100,000,000	90,711,906	3	3	1/10	3/20	5,874	547.1
60	1,000	983	8	7	2/5	7/10	39,878	0.037
60	10,000	8,690	8	7	1/10	1/4	18,775	0.18
60	100,000	99,138	5	6	1/4	3/20	14,101	1.31
60	200,000	180,603	5	6	3/20	1/10	13,133	2.4
60	400,000	378,840	5	6	3/20	1/20	12,663	4.7
60	600,000	514,154	5	6	1/20	1/20	12,292	6.9
60	800,000	754,273	4	5	1/5	1/5	11,830	8.8
60	1,000,000	954,771	4	5	1/4	3/20	11,555	10.8
60	2,000,000	1,782,449	4	5	3/20	1/10	10,777	20.1
60	4,000,000	3,861,677	4	5	3/20	1/20	10,394	38.7
60	6,000,000	5,289,299	4	5	1/20	1/20	10,100	56.4
60	8,000,000	5,289,299	4	5	1/20	1/20	10,100	75.3
60	10,000,000	8,501,436	4	5	1/50	1/20	10,021	93.3
60	100,000,000	96,824,114	3	4	1/10	1/10	8,328	775.6
80	1,000	1,000	8	7	3/4	19/20	269,372	0.25
80	10,000	9,013	8	7	7/20	7/10	44,106	0.41
80	100,000	81,135	8	7	1/10	1/4	21,527	2.0
80	200,000	171,508	8	7	1/20	3/20	19,537	3.6
80	400,000	394,799	6	7	3/20	1/10	17,857	6.7
80	600,000	457,184	6	7	1/10	1/10	17,592	9.8
80	800,000	644,339	6	7	1/20	1/10	17,348	12.9
80	1,000,000	881,197	6	7	1/10	1/20	16,952	15.8
80	2,000,000	1,926,399	5	6	3/20	3/20	15,742	29.3
80	4,000,000	3,356,174	5	6	1/10	1/10	14,928	55.6
80	6,000,000	4,754,583	5	6	1/20	1/10	14,728	82.3
80	8,000,000	6,603,497	5	6	1/10	1/20	14,386	107.2
80	10,000,000	8,227,316	5	6	1/20	1/20	14,188	132.1
80	100,000,000	94,939,000	4	5	1/20	1/10	12,123	1,129.0

Table 3. For $k = 128$ and a given statistical security s and circuit size q, the figure shows the best found parameters α, β, p_a, and p_g with respect to communication overhead. The last two columns show the resulting amount of bits per gate and the total amount of gibits (*i.e.*, 2^{30} bits) sent from Alice to Bob, ignoring communication that does not depend on circuit size.

three wires of each gate. In the Cut-and-Choose step Alice sends to Bob the openings to the three requested keys which amounts to $3\frac{\Gamma}{2}(k+\psi)$ bits. Finally for the soldering step Alice sends to Bob $3(\beta'-1)q(k+\psi)$ bits for the horizontal solderings and $2q(k+\psi)$ bits for the vertical solderings.

As the randomness used in the code $\mathcal{C}'$ is used to obtain privacy, it is needed that $k \leq \psi$, so we can assume $k = \psi$. With this in mind we see that the expression is

$$\begin{aligned}
&2\Gamma' n' + \Gamma'(k+\psi) + \Gamma' k + \Gamma'(k+\psi) + 2\Gamma k + 3\Gamma k \\
&\quad + 3\frac{\Gamma}{2}(k+\psi) + 3(\beta'-1)q(k+\psi) + 2q(k+\psi) \\
&\quad = 2\Gamma' n' + 5\Gamma' k + 8\Gamma k + 6(\beta'-1)qk + 4qk \\
&\quad = (6\Gamma+2)n' + (15\Gamma+5)k + 8\Gamma k + 6(\beta'-1)qk + 4qk \\
&\quad = (6\Gamma+2)n' + 23\Gamma k + 6(\beta'-1)qk + 4qk + 5k \\
&\quad = (12\beta' q + 2)n' + 46\beta' qk + 6(\beta'-1)qk + 4qk + 5k
\end{aligned}$$

It is very hard to give a precise calculation of the parameters of the code in [CC06], as everything is phrased asymptotically: there is a security parameter and the length of the key is given as $O(k)$, the number of positions one can correct in a code word should be $O(k)$, the number of inspected positions in the watch list is $O(k)$ and the security is given as $2^{-O(k)}$ and so on. Finding the best setting of these parameters for a specific code seems to be a hard optimization problem. However, it seems that setting all parameters to be the same k which is also the key length will not give an estimate which is far away from optimal, so this is what we will do.

A quick estimate of the codes in [CC06] proposed by the authors of [FJN+13] shows that the best setting would be to use a code over the field $GF(16)$ for which to get distance $2k$ and room for a key of length k, the length of the code would be around $10k$ field elements. To get a binary code the best choice seems to be to encode a single bit in each position. Concatenation seems to give worse parameter. This gives a binary code of length $40k$. The authors propose to inspect k positions at random using a watch list mechanism. Note that they might as well inspect entire field elements, as this makes no difference for the privacy of the code. This catches an attempt to open to not the closest code word except with probability $(9/10)^k$, as the distance to that code word is at least k field elements and we can inspect k out of $10k$ field elements as part of the watch list mechanism. If we solve $(9/10)^k = 2^{-s}$ we see that $k > 6.5s$ for a total bit length of a code word of more than $6.5 \cdot 40s = 260s$.

With this in mind we derive the approximated codes $[10400, 128, 81]$, $[15600, 128, 121]$ and $[20800, 128, 161]$ which we use for our comparison. In our count we use the security analysis of TinyLEGO to derive the minimum value of β' which is needed for MiniLEGO. The reasoning for this is that the majority principle is the same for both protocols, but that TinyLEGO takes both wire authenticators and gates into consideration. This analysis however only makes sense if MiniLEGO was modified so that each gate was chosen for check with probability $\frac{1}{2}$. Therefore we assume that this is the case and ignore the issues of calculating slack in the MiniLEGO case, this is clearly of no disadvantage for MiniLEGO in the comparison.

I Overview of Variables and Parameters

A list of variable names and their meaning is given in Table 4.

Symbol	Meaning
s	Statistical security parameter.
k	Computational security parameter.
C	The plain description of the Boolean circuit to compute.
f	The function computed by C.
x	A bit string representing the constructor's (Alice's) input to the circuit.
y	A bit string representing the evaluator's (Bob's) input to the circuit.
z	The circuit output destined for the constructor.
q	Amount of AND gates in C.
n_{C}	Amount of input bits to the circuit from the constructor, defined as $n_{\mathsf{C}} = \|x\|$.
n_{E}	Amount of input bits to the circuit from the evaluator, defined as $n_{\mathsf{E}} = \|y\|$.
n	Total amount of input bits to the circuit, defined as $n = n_{\mathsf{C}} + n_{\mathsf{E}}$.
m	Total amount of output bits from the circuit, defined as $m = \|z\|$.
w	Amount of wires in C, defined as $w = n + q$.
$\mathcal{C}$	The linear error correction code used in the protocol.
Γ	The codeword length of $\mathcal{C}$.
d	The minimum distance of the codewords of $\mathcal{C}$, defined as $d = s + 4$.
G,E,A	Unique literals used to ensure uniqueness of hashed indices.
p_g	Fraction of garbled gates that should be checked.
δ_g	Fraction of garbled gates we need to get sufficient "slack".
p_a	Fraction of authentication wires that should be checked.
δ_a	Fraction of authentication wires we need to get sufficient "slack".
β	The amount of gates in each bucket.
α	The amount of authenticators for each bucket.
ℓ_g	The gate replication factor, defined as $\ell_g = \frac{1}{1-p_g-\delta_g}$.
ℓ_a	The authentication wire replication factor, defined as $\ell_a = \frac{1}{1-p_a-\delta_a}$.
Q	The total amount of garbled gates we need to construct, defined to be $Q = q\beta\ell_g$.
A	The total amount of authenticators we need to construct, defined to be $A = (q\alpha + n(2\beta+1))\ell_a$.
W	The amount of wires considered in a protocol execution, defined to be $W = 3Q + A$.
γ	The total amount of commitments required, defined as $\gamma = W + 1 + 7.3(s+3)$.
Δ	The global difference on all wires, defined as the key K_{W+1}.
BucketOf	A β-to-1 map from garbled gates to buckets.
AuthOf	A α-to-1 map from authenticators to buckets.
WiresOf	A public map from gate indices to left,right and output wire indices.

Table 4. Overview of variables along with their meaning.