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 the notion of an *interactive garbling scheme*, where the garbled circuit is generated through 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 any probabilistic polynomial time 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 an 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 optimizations, for example making it possible to get distinct output to both parties with minimal overhead. The scheme makes black-box usage of a XOR-homomorphic commitment scheme, an authentic, private and oblivious garbling scheme and a 2-correlation-robust and collision-resistant hash function.

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 f with private output based on their respective private inputs. We say that A has the input x, B the input y, and they wish to learn the output $(z_{\mathsf{A}},z_{\mathsf{B}}) \leftarrow f(x,y)$ without B learning anything about x and without A learning anything about y. Here z_{A} and z_{B} denotes the private output of A and B, respectively.

This area was introduced in 1982 by Andrew Yao [Yao82, Yao86], specifically for the *semi-honest* case, where all parties are assumed to follow the protocol and only try to compromise security by analyzing their own views of the protocol execution. Yao showed how to prevent this using a technique referred to as the *garbled circuit approach*. This approach entails one party (*the constructor*), say A, encrypt, or "garble", a Boolean circuit f computing the desired function. This is achieved 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 of f is then garbled such that B (*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

⋆ 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.

of the two input bits). A sends the garbled circuit to B and, using an oblivious transfer (OT) protocol, B also learns one key for each input wire corresponding to his own input, without A learning which one. For the input wires corresponding to A's input, she sends the keys directly to B along with some auxiliary information for decoding the output wires of the circuit. Now, given the input keys, B will evaluate the garbled circuit, without knowing which bits flow on the wires. Finally, when he reaches the output keys he uses the auxiliary information to learn which bits the keys encode. See [LP09] for a thorough description of Yao's approach.

If one considers a *malicious* adversary, where a corrupt party might deviate from the protocol in an arbitrary manner, then Yao's approach is no longer secure. One of the major issues is that B cannot be sure that the garbled circuit he receives from A has been garbled correctly. To cope with this issue, the *cut-and-choose* approach can be used: Instead of sending a single circuit, A sends several independently garbled versions of the circuit to B. B then randomly selects a subset of these, called the *check circuits*, which are opened by A, allowing B to verify that they correspond to the correct function f. If this is the case, he is guaranteed that a majority of the remaining circuits, called *evaluation circuits*, are garbled correctly.

However, the cut-and-choose approach introduces other issues that have to be dealt with in order to obtain malicious security, *e.g.* ensuring consistent inputs in all evaluation circuits. Another prominent issue when considering malicious adversaries is the *selective failure attack* [MF06, KS06]: Because A supplies B with keys in correspondence with his input bits through an OT, A is free to input garbage for one of the keys, e.g, the 0-key for the first bit of B's input. If B now aborts the protocol, A will know that the first bit of his input is 0 as he cannot evaluate a garbled circuit when one of the input keys is garbage. On the other hand, if no abort occurs then A learns that his first bit must be 1.

Solutions to the above attack and several other attacks on the cut-and-choose approach, along with several optimizations have led to a plethora of work on cut-and-choose protocols including, but not limited to, [LP07, PSSW09, LP11, sS11, HEKM11, KSS12, Bra13, FN13, HKE13, Lin13, MR13, sS13, HMSG13, RT13, FJN14, AMPR14].

Related Work. 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 maliciously constructed 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 additive 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. One such optimization is the celebrated "free-XOR" optimization [KS08] which makes it possible to construct and evaluate XOR gates for "free" (free meaning that no cryptographic operations or communication is needed).

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 on bitstrings 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 commitments introduced a rather large concrete increase in the communication complexity of each garbled gate. So while MiniLEGO asymptotically performs better than circuit cut-and-choose protocols, for practical parameters and circuit sizes the protocol is not competitive. In practice the XOR-homomorphic commitments can be approximated to be at least 40 times the size of the message committed to. This has the effect that the asymptotic saving LEGO achieves only becomes substantial at impractically large circuits. Thus for realistic circuits the MiniLEGO protocol induces too much overhead compared to the fastest protocols for cut-and-choose of garbled circuits.

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.

Motivation. Efficient protocols for maliciously secure two-party computation based on garbled circuits are often extremely complex. The corresponding proofs of security are no better and often hinges on subtle interconnected (seemingly ad-hoc) elements of the entire protocol in order to go through. Due to this complexity it is also highly non-trivial to modify these protocols and reason about what new security guarantees hold. One would usually have to go through the time-consuming task of reproving the modified construction in order to have full confidence in a design change. This leaves a lot to be desired in terms of flexibility if one wants to trim a protocol for a particular application. As an example, if the full power of the original protocol is not needed it can be a daunting task to identify which elements can be safely left out without losing all security guarantees.

In this work we take a step towards solving this issue by introducing a new abstraction in the area of garbled circuits. Much inspired by the work of Bellare *et al.* [BHR12] we present similar definitions of security, however in an interactive setting considering a malicious adversary. Next, we show that our proposed security notions for an interactive garbling scheme imply UC-secure 2PC in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model with both parties receiving output. By abstractly defining each distinctive security property separately (as opposed to all in one using an ideal functionality) the primitive is modular and can be weakened (or strengthened) for a particular application without having to reprove all security properties from scratch.

Our Contribution. We present a new abstraction for achieving efficient malicious and static secure 2PC based on garbled circuits. Our contributions includes:

- We introduce a fully generic framework for interactive garbling and define notions of security in an interactive setting considering a malicious adversary. We then show that our notion of an interactive garbling scheme suffices for UC-secure 2PC in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model with both parties receiving output.
- Next we show how to realize such an interactive garbling scheme. Our instantiation is based on the LEGO techniques of [NO09, FJN$^+$13, LR14] along with several optimizations. 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] Another is the possibility of distinct output to *both* parties without the need of circuit augmentation or any other add-ons.
- Finally, we instantiate our scheme with the commitment scheme of [FJNT15] and give a detailed comparison with current state-of-the-art protocols and see that our construction compares favorably. As expected the LEGO technique becomes more competitive as the size of the circuit being garbled increases. For example, for a circuit with at least 514,297 AND gates and 40-bit statistical security we reduce communication with 29% compared to previous protocols in the same setting. The details of our comparison can be found in Section 7.

[1] For the readers familiar with the LEGO protocol of [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.

2 Preliminaries

Notation. We will use as shorthand $[n]=\{1,2,...,n\}$ and $[i;n]=\{i,i+1,i+2,...,n\}$ for $i\leq n$. We write $e\in_R S$ 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 use $\|$ to denote concatenation of vectors. 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 bit of a vector x. 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 $2^{-s}+\text{negl}(k)$ 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. For two ensembles $X=\{X_{k,z}\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and $Y=\{Y_{k,z}\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ of binary random variables we say these are *indistinguishable*, denoted by $X\overset{c}{\approx}Y$, if for all z it holds that $|Pr[X_{k,z}=1]-Pr[Y_{k,z}=1]|\leq\text{negl}(k)$. Finally an overview of the various variables and parameters along with their meaning is given in Section 8.

Garbling Schemes. We assume A is the party constructing the garbled gates and call her the *constructor*. Likewise, we assume B is the party evaluating the garbled gates and call him the *evaluator*. Furthermore, we say that the functionality they wish to compute is $(z_{\mathsf{A}},z_{\mathsf{B}})\leftarrow f(x,y)$, where A gives input x, B gives input y, A, receives the output z_{A} and B receives the output z_{B}. We assume (w.l.o.g.) that f is described only using 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. 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_{A} as $|z_{\mathsf{A}}|=m_{\mathsf{A}}$, the bit-length of z_{B} as $|z_{\mathsf{B}}|=m_{\mathsf{B}}$ and $m=m_{\mathsf{A}}+m_{\mathsf{B}}$. Furthermore, we assume that the first n_{A} input wires are for A's input and the following n_{B} input wires are for B's input.[2] Similarly, the first m_{A} output wires are for A's output and the following m_{B} output wires are for B's output.

We define the *semantic* value of a wire-key of a garbled gate to be the bit it represents. We will use X_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. X_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 make it possible to handle NOT and XOR gates implicitly. Thus, a circuit is a 7-tuple $f=(n_{\mathsf{A}},n_{\mathsf{B}},m_{\mathsf{A}},m_{\mathsf{B}},q,\mathsf{lp},\mathsf{rp})$ where $n=n_{\mathsf{A}}+n_{\mathsf{B}}$ and $n\geq 2$ is the number of inputs, $m=m_{\mathsf{A}}+m_{\mathsf{B}}$ and $m\geq 1$ 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, as XOR and NOT gates are handled implicitly as described below. We let $\mathsf{Wires}=\{1,...,w\}$, $\mathsf{Inputs}=\{1,...,n\}$, $\mathsf{Gates}=\{n+1,...,w\}$ and $\mathsf{Outputs}=\{w-m+1,...,w\}$. 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 key of gate g be $\bigoplus_{j\in\mathsf{lp}(g)}O_j^{b_j}$. In this way all XOR and NOT gates of f are 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 define a garbling scheme to be a 5 tuple of poly-time algorithms $\mathcal{G}=(\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})$. Gb denotes a randomized algorithm, taking as input a security parameter and a function description f, while producing as output a triple consisting of a garbled circuit F, input encoding information e along with output decoding information d. That is $\mathsf{Gb}(1^k,f)\rightarrow(F,e,d)$. The function En can then be used to construct the garbled input X when given e and $x\|y$. That is, $\mathsf{En}(e,x\|y)\rightarrow X$. The garbled input X can then be evaluated by the garbled circuit F using the function Ev, yielding the garbled output Z. That is, $\mathsf{Ev}(F,X)\rightarrow Z$. The garbled output Z can then be decoded to the plain output z using the decoding information d and the function De. That is, $\mathsf{De}(d,Z)\rightarrow z$. Finally, it is possible to evaluate the plain function f using the plain input $x\|y$ to the plain output

[2] For ease of presentation we restrict our attention to circuits with fan-out 1 input-wires only. This is not a major restriction as one can always augment the circuit with identity gates on the input layer. Each of these gates then takes one input wire as input and is allowed unlimited fan-out.

z using the algorithm ev. That is, $\mathsf{ev}(f,x\|y)\rightarrow z$, or, abusing notation $f(x\|y)=f(x,y)=z=z_\mathsf{A}\|z_\mathsf{B}=(z_\mathsf{A},z_\mathsf{B})$. For convenience we define implicit functions f_A and f_B such that $f_\mathsf{A}(x\|y)=f_\mathsf{A}(x,y)=z_\mathsf{A}$ and $f_\mathsf{B}(x\|y)=f_\mathsf{B}(x,y)=z_\mathsf{B}$.

We have some further requirements on the output of the algorithms. It must be the case that $|F|,|e|$ and $|d|$ only depend on k,n,m and $|f|$. We also have the *length* condition; if $n=n'$, $m=m'$ and $|f|=|f'|$ when $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f)$, $(F',e',d')\leftarrow\mathsf{Gb}(1^k,f')$ then it must hold that $|F|=|F'|$, $|e|=|e'|$ and $|d|=|d'|$. Finally we have the *correctness* requirement that if $f\in\{0,1\}^*$, $k\in\mathbb{N}$, $x\|y\in\{0,1\}^n$ and $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f)$ then it must hold that $\mathsf{De}(d,\mathsf{Ev}(F,\mathsf{En}(e,x\|y)))=\mathsf{ev}(f,x\|y)$.

We require both secrecy and authenticity of the garbling scheme we use. Bellare *et al.* [BHR12] discusses two types of secrecy, *privacy* and *obliviousness*. For privacy the demand is that a party learning (F,X,d) does not learn anything besides some allowed leakage and the output z (for example by computing $\mathsf{De}(d,\mathsf{Ev}(F,X))$). The interpretation of this notion is that the semantic values on the internal wires remain private towards the party who has garbled the circuit. The allowed leakage is captured through a side-information function $\mathbf{\Phi}$, which is queried on the plain function f and returns the allowed leakage when a party is in possession of (F,X,d) (when $F\leftarrow\mathsf{Gb}(1^k,f)$). In the case of obliviousness we assume the evaluating party does not know the plain function f nor the decoding information d, and thus is only in possession of (F,X). We then wish that he does not learn anything about $f,x\|y$ or z, and thus only learns what is permitted by $\mathbf{\Phi}$. The interpretation of this notion is that everything about the function and inputs remain secret to the evaluator. Like obliviousness we define the notion of authenticity towards a party which is only given F and X. We wish that he is not able to construct a garbled output $Z^*\neq\mathsf{Ev}(F,X)$ such that $\mathsf{De}(d,Z^*)\neq\perp$. The interpretation of this notion is that one cannot construct permissible garbled output different from what is dictated by X and F. We define these notions formally in the indistinguishability based games in Fig. 1.

We let the advantage of a PPT adversary $\mathcal{A}$ playing game G using the garbling scheme $\mathcal{G}$ with security parameter k and potentially an auxiliary function tuple η be denoted by $\mathbf{Adv}_\mathcal{G}^{\mathsf{G},\eta}(\mathcal{A},k)$. For the games in Fig. 1 the advantages are defined as follows:

$$\mathbf{Adv}_\mathcal{G}^{\text{prv.ind},\mathbf{\Phi}}(\mathcal{A},k)=2\Pr[\text{PrvInd}_{\mathcal{G},\mathbf{\Phi}}^\mathcal{A}(1^k)=\top]-1\ ,$$
$$\mathbf{Adv}_\mathcal{G}^{\text{obl.ind},\mathbf{\Phi}}(\mathcal{A},k)=2\Pr[\text{OblInd}_{\mathcal{G},\mathbf{\Phi}}^\mathcal{A}(1^k)=\top]-1\ ,\mathbf{Adv}_\mathcal{G}^{\text{aut},\mathbf{\Phi}}(\mathcal{A},k)=\Pr[\text{Aut}_\mathcal{G}^\mathcal{A}(1^k)=\top]\ .$$

Game $\text{PrvInd}_{\mathcal{G},\mathbf{\Phi}}^\mathcal{A}(1^k)$. **Property** prv.ind.

1. Run $\mathcal{A}(1^k)$ to produce (f_0,f_1,x_0,x_1).
2. If $\{x_0,x_1\}\not\subseteq\{0,1\}^{f_0.n}$, $\mathbf{\Phi}(f_0)\neq\mathbf{\Phi}(f_1)$ or $\mathsf{ev}(f_0,x_0)\neq\mathsf{ev}(f_1,x_1)$ then output $\perp$.
3. Sample uniformly random bit $b\in_R\{0,1\}$.
4. Run $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f_b)$.
5. Compute $X\leftarrow\mathsf{En}(e,x_b)$.
6. Let $b'\leftarrow\mathcal{A}(F,X,d)$.
7. If $b'=b$, then output $\top$, otherwise output $\perp$.

Game $\text{OblInd}_{\mathcal{G},\mathbf{\Phi}}^\mathcal{A}(1^k)$. **Property** obl.ind.

1. Run $\mathcal{A}(1^k)$ to produce (f_0,f_1,x_0,x_1).
2. If $\{x_0,x_1\}\not\subseteq\{0,1\}^{f_0.n}$ or $\mathbf{\Phi}(f_0)\neq\mathbf{\Phi}(f_1)$ then output $\perp$.
3. Sample uniformly random bit $b\leftarrow\{0,1\}$.
4. Run $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f_b)$.
5. Compute $X\leftarrow\mathsf{En}(e,x_b)$.
6. Let $b'\leftarrow\mathcal{A}(F,X)$.
7. If $b'=b$, then output $\top$, otherwise output $\perp$.

Game $\text{Aut}_\mathcal{G}^\mathcal{A}(1^k)$. **Property** aut.

1. Run $\mathcal{A}(1^k)$ to produce (f,x).
2. If $x\notin\{0,1\}^{f.n}$ then output $\perp$.
3. Run $(F,e,d)\leftarrow\mathsf{Gb}(1^k,f)$.
4. Compute $X\leftarrow\mathsf{En}(e,x)$.
5. Let $Z\leftarrow\mathcal{A}(F,X)$.
6. If $\mathsf{De}(d,Z)\neq\perp$ *and* $Z\neq\mathsf{Ev}(F,X)$ then output $\top$, otherwise output $\perp$.

Fig. 1. Security games for garbling schemes

We also use the notion of a projective garbling scheme as introduced by Bellare *et al.* [BHR12]. Informally speaking a garbling scheme is *projective* if it is possible to parse the encoding information e as $\left(X_1^0,X_1^1,\ldots,X_n^0,X_n^1\right)$, two for each of the n input wires. We generalize this notion to the output wires as well, and thus define a garbling scheme to be *output projective* if there are exactly two possible tokens associated with each of the m circuit output wires, where one represents the 0-bit and the other the 1-bit of the wire. Specifically, for $z\in\{0,1\}^m$ we have a unique set $\left(Z_1^0,Z_1^1,Z_2^0,Z_2^1,\ldots,Z_m^0,Z_m^1\right)$ and $\mathsf{Ev}(F,X)\to(Z_1^{z_1},Z_2^{z_2},\ldots,Z_m^{z_m})$. More formally we demand that for all f with $x,x'\in\{0,1\}^n$, $k\in\mathbb{N}$ and $i,j\in[m]$ where $z\leftarrow\mathsf{ev}(f,x),z'\leftarrow\mathsf{ev}(f,x')$, $(F,e,d)\leftarrow\mathsf{Gb}\left(1^k,f\right)$, $X=\mathsf{En}(e,x)$, and $X'=\mathsf{En}(e,x')$ that $\mathsf{Ev}(F,X)\to Z$ and $\mathsf{Ev}(F,X')\to Z'$ we have that $Z=(Z_1,Z_2,\ldots,Z_m)$ and $Z'=(Z_1',Z_2',\ldots,Z_m')$ are m element vectors, $|Z_i|=|Z_i'|=|Z_j|=|Z_j'|$ and $Z_i=Z_i'$ iff $z_i=z_i'$.

We say a scheme has *projective coding* if both e and d are projective as defined above. As a consequence of this the encoding and decoding algorithms, En and De contain subalgorithms $\overline{\mathsf{En}}$ and $\overline{\mathsf{De}}$, respectively, working on individual elements. A bit more formally we can define the algorithms as follows:

$\mathsf{En}(e,x)\to X$**:**
1. Parse $(e_1,\ldots,e_n)\leftarrow e$ and $(x_1,\ldots,x_n)\leftarrow x$.
2. For $i\in[n]$ let $X_i=\overline{\mathsf{En}}(e_i,x_i)$.
3. Set $(X_1,\ldots,X_n)\to X$ and return X.

$\mathsf{De}(d,Z)\to z$**:**
1. Parse $(d_1,\ldots,d_m)\leftarrow d$ and $(Z_1,\ldots,Z_m)\leftarrow Z$.
2. For $i\in[m]$ let $z_i=\overline{\mathsf{De}}(d_i,Z_i)$.
3. Set $(z_1,\ldots,z_m)\to z$ and return z.

3 Interactive Garbling Schemes

We introduce the notion of a (projective) interactive garbling scheme. Our notion extends the notion of a projective garbling scheme from [BHR12] to allow the garbling algorithm to be a two-party protocol. In Section 2 we introduced a simplified version of the syntax, notational conventions and security notions from [BHR12]. In terms of [BHR12] the definitions below are for the leakage function $\mathbf{\Phi}_{\mathsf{xor}}(f)$, *i.e.*, the evaluator is allowed to learn the topology of f and which gates are XOR gates. Furthermore, we work with a unified notion of *secrecy* that captures both privacy and obliviousness in the two party setting where both parties are supposed to learn some private output.

Syntax. An *interactive garbling scheme* consists of a six-tuple $\mathcal{G}_\pi=(\mathsf{Gb}_\pi,\mathsf{En}_\pi,\mathsf{De}_\pi,\mathsf{Ev}_\pi,\mathsf{ev}_\pi,\mathsf{Ve}_\pi)$. The first component, called the *garbling protocol*, is a two party protocol. The remaining components are deterministic algorithms. All components are poly-time (in k). The evaluation function ev_π takes two inputs, a function description f and an input x for f. A string f, the *original function*, by definition describes a function $\mathsf{ev}_\pi(f,\cdot):\{0,1\}^n\to\{0,1\}^m$, 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 $\mathsf{ev}_\pi(f,x)$. We assume that the input lengths $f.n_{\mathsf{A}}$, $f.n_{\mathsf{B}}$, $f.n=f.n_{\mathsf{A}}+f.n_{\mathsf{B}}$ and the output length $f.m_{\mathsf{A}}$, $f.m_{\mathsf{B}}$, $f.m=f.m_{\mathsf{A}}+f.m_{\mathsf{B}}$ can be computed in linear time from f. The *garbling protocol* Gb_π is executed between two parties, the *constructor* C (played by A) and the *evaluator* E (played by B). 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 of C is (F,e,d), where F is the *garbled function*, e is the *input encoding function*, and d is the *output decoding function.* The output of E is a garbled function $F\in\{0,1\}^*$ and a *verification function* v.

We let Ev_π be the *garbled evaluation function*, working like the evaluation function in a regular garbling scheme. That is, it takes as input F and X where F has been constructed using Gb_π and X using the encoding algorithm En_π on e and input x. Ev_π outputs a *garbled output* Z which can then be used in De_π along with d to restore the *plain output* z.

The algorithm Ve_π is extra compared to [BHR12]. This *verification algorithm* uses the verification function v to verify that a wire token encodes a particular bit. Like the algorithms $\overline{\mathsf{En}}$ and $\overline{\mathsf{De}}$ defined for *projective coding* in Section 2, this algorithm works element wise. A bit more specifically we require that v can be parsed into individual elements v_i. Ve_π then takes as input a verification element v_i, a garbled value X_i, and a bit x_i. It then outputs true ($\top$) or false ($\perp$). Intuitively, it uses v_i to judge whether X_i has been constructed consistently with the bit x_i. We capture the security requirements of Ve_π via the tok.com property in Fig. 3 in the following.

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 $\mathcal{G}_\pi = (\mathsf{Gb}_\pi, \mathsf{En}_\pi, \mathsf{De}_\pi, \mathsf{Ev}_\pi, \mathsf{ev}_\pi, \mathsf{Ve}_\pi, \mathsf{Ex}_\mathsf{C}, \mathsf{Ex}_\mathsf{E}, \mathsf{De}_\pi^{-1}, \mathsf{En}_\pi^{-1}, \mathsf{EqV})$. 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_π^{-1} is called the *de-encoder*. It takes as input the encoding function e_i and an encoded input X_i. It outputs an input x_i, which is supposed to be the x_i encoded by X_i. It is used to guarantee that even a malicious constructor has a well-defined input. The deterministic poly-time algorithm De_π^{-1} is called the *de-decoder*. It takes as input the decoding function d_j and an output z_j. It outputs an encoded input Z_j. 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 $\perp$ 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.

Below we informally describe the security properties of our notion of an interactive garbling scheme. These properties are formally captured in Fig. 2 and Fig. 3.

Projective Schemes. We require that the scheme has *projective coding* as defined in Section 2, meaning that the encoding and decoding functions can be semantically tokenized into individual bitstrings of equal length. As a consequence of this the above-mentioned "reverting" algorithms En_π^{-1} and De_π^{-1} are directly defined as they simply select the corresponding semantic value and output token, respectively. We now describe the de-encoder En_π^{-1} and de-decoder De_π^{-1} in more detail. Let $(e_1, e_2, \ldots, e_n) \leftarrow e$. For any $i \in [n]$, on input X_i and e_i we let $\mathsf{En}_\pi^{-1}(e_i, X_i) = \perp$ if e_i cannot be parsed as (X_i^0, X_i^1), it can be parsed this way but $X_i^0 = X_i^1$ or $|X_i^0| \neq |X_i^1|$ or if there does not exist a $x_i \in \{0,1\}$ such that $X_i \leftarrow \mathsf{En}_\pi(e_i, x_i)$. Otherwise let x_i be this value from $\{0,1\}$ and return x_i. Likewise let $(d_1, d_2, \ldots, d_m) \leftarrow d$. For any $j \in [m]$, on input z_j and d_j let $\mathsf{De}_\pi^{-1}(d_j, z_j) = \perp$ if d_j cannot be parsed as (d_j^0, d_j^1), it can be parsed this way but $d_i^0 = d_i^1$ or if $z_j \notin \{0,1\}$. Otherwise let Z_j be the unique value such that $z_j \leftarrow \mathsf{De}_\pi(d_j, Z_j)$ and let $\mathsf{De}_\pi^{-1}(d_j, z_j) \rightarrow Z_j$. If a scheme satisfies the above we say that it has the property proj.

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 verification algorithm must be correct.

Secrecy. We define *secrecy* (property name: sec.ind.act) like obliviousness in [BHR12] for the first m_A output bits and like privacy in [BHR12] for the last m_B output bits. That is, we require that for the first m_A output bits, by seeing a garbling and an encoding of one of two inputs x_0 and x_1 along with decoding information for the last m_B output bits, the evaluator cannot guess which input was used, under the constraint that $f_\mathsf{B}(x_0) = f_\mathsf{B}(x_1)$. As an addition we let the adversary $\mathcal{B}$ play the role of E in the garbling protocol. We furthermore 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.

Authenticity. We define *authenticity* (property name: aut.act) 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

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

1. Run $\mathcal{A}(1^k)$ to produce (f,x).
2. If $x \notin \{0,1\}^n$ then output $\perp$.
3. Run Gb_π between $\mathsf{C}(1^k,f)$ and $\mathsf{E}(1^k,f)$. If any party outputs $\perp$, then output $\top$. Otherwise, denote the output of C by (F,e,d) and the output of E by (F',v) and let $(v_1,\ldots,v_{n_\mathsf{B}},v_{n_\mathsf{B}+1},\ldots,v_{n_\mathsf{B}+m_\mathsf{B}}) \leftarrow v$, $(e_1,\ldots,e_n) \leftarrow e$ and $(d_1,\ldots,d_m) \leftarrow d$. If $F' \neq F$, then output $\top$.
4. Compute $X_i \leftarrow \mathsf{En}_\pi(e_i,x_i)$ and let $X=(X_1,\ldots,X_n)$.
5. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z \leftarrow \mathsf{ev}_\pi(f,x)$ and let $Z=(Z_1,\ldots,Z_m)$.
6. If $\mathsf{Ve}_\pi(v_i,X_{n_\mathsf{A}+i},x_{n_\mathsf{A}+i})=\perp$ for any $i\in[n_\mathsf{B}]$ then output $\top$.
7. If $\mathsf{Ve}_\pi(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j},z_{m_\mathsf{A}+j})=\perp$ for any $j\in[m_\mathsf{B}]$ then output $\top$.
8. If $\mathsf{De}_\pi(d_j,Z_j)\neq z_j$ for any $j\in[m]$ then output $\top$ otherwise output $\perp$.

Game $\mathrm{SecIndAct}^{\mathcal{B}}_{\mathcal{G}_\pi}(1^k)$. **Property** sec.ind.act.

1. Run $\mathcal{B}(1^k)$ to produce (f_0,f_1,x_0,x_1).
2. If any of the following are true, then output $\perp$.
   (a) $x_0,x_1 \notin \{0,1\}^n$.
   (b) $x_{0,i} \neq x_{1,i}$ for $i \in [n_\mathsf{A}+1;n]$.
   (c) $\mathbf{\Phi}_{\mathsf{xor}}(f_0) \neq \mathbf{\Phi}_{\mathsf{xor}}(f_1)$.
   (d) $z_{0,j} \neq z_{1,j}$ for any $j \in [m_\mathsf{A}+1;m]$ when we let $z_0=\mathsf{ev}_\pi(f_0,x_0)$ and $z_1=\mathsf{ev}_\pi(f_1,x_1)$.
3. Sample uniformly random $b \in_R \{0,1\}$.
4. Run Gb_π between $\mathsf{C}(1^k,f_b)$ and $\mathcal{B}$. If C outputs $\perp$, output $\perp$. Otherwise, denote the output of C by (F,e,d) where $(e_1,\ldots,e_n) \leftarrow e$ and $(d_1,\ldots,d_m) \leftarrow d$.
5. Compute $X_i \leftarrow \mathsf{En}_\pi(e_i,\ x_{b,i})$ and let $X=(X_1,\ldots,X_n)$.
6. Give X and $d_\mathsf{B}=(d_{m_\mathsf{A}+1},\ldots,d_m)$ as input to $\mathcal{B}$ and run $\mathcal{B}$ to get an output b'.
7. If $b'=b$, then output $\top$, otherwise output $\perp$.

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

1. Run $\mathcal{B}(1^k)$ to produce (f,x).
2. If $x \notin \{0,1\}^n$ then output $\perp$.
3. Run Gb_π between $\mathsf{C}(1^k,f)$ and $\mathcal{B}$. If C outputs $\perp$, output $\perp$. Otherwise, denote the output of C by (F,e,d) where $(e_1,\ldots,e_n) \leftarrow e$ and $(d_1,\ldots,d_m) \leftarrow d$.
4. Compute $X_i \leftarrow \mathsf{En}_\pi(e_i,\ x_i)$ and let $X=(X_1,\ldots,X_n)$.
5. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,\ X)$ and let $Z=(Z_1,\ldots,Z_m)$.
6. Let $z_j \leftarrow \mathsf{De}_\pi(d_j,\ Z_j)$ for $j \in [m]$ and $z=(z_1,\ldots,z_m)$. If $z \neq \mathsf{ev}_\pi(f,x)$, then output $\top$.
7. Give X as input to $\mathcal{B}$ and run $\mathcal{B}$ to get an output $Z'_\mathsf{A} \leftarrow \left(Z'_1,\ldots,Z'_{m_\mathsf{A}}\right)$.
8. If $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j}) \neq \perp$ and $Z'_{\mathsf{A},j} \neq Z_j$ for any $j \in [m_\mathsf{A}]$ output $\top$, otherwise output $\perp$.

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

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

Fig. 2. Security games for interactive garbling scheme – part 1.

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 $Z \leftarrow \mathsf{Ev}_\pi(F,X)$. 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.

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

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}(1^k,f)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F,v).
3. Run Ex_C on $\mathcal{A}$ to extract $\hat{e}=(\hat{e}_1,\ldots,\hat{e}_n)$ and $\hat{d}=\big(\hat{d}_1,\ldots,\hat{d}_m\big)$.
4. Run $\mathcal{A}$ to produce $x=(x_1,\ldots,x_n)$.
5. Let $\hat{X}_i \leftarrow \mathsf{En}_\pi(\hat{e}_i,x_i)$ for $i\in[n]$ and $\hat{Z}\leftarrow \mathsf{Ev}_\pi(F,\hat{X})$ where $\hat{X}=\big(\hat{X}_1,\ldots,\hat{X}_n\big)$.
6. Let $z_j \leftarrow \mathsf{De}_\pi(\hat{d}_j,\hat{Z}_j)$ for $j\in[m]$ and $z=(z_1,\ldots,z_m)$. If $\mathsf{ev}_\pi(f,x)\neq z$, then output $\top$, otherwise output $\perp$.

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

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}(1^k,f)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F,v) where $(v_1,\ldots,v_{n_\mathsf{B}},v_{n_\mathsf{B}+1},\ldots,v_{n_\mathsf{B}+m_\mathsf{B}})\leftarrow v$.
3. Run Ex_C on $\mathcal{A}$ to extract $\hat{e}=(\hat{e}_1,\ldots,\hat{e}_n)$ and $\hat{d}=\big(\hat{d}_1,\ldots,\hat{d}_m\big)$.
4. Run $\mathcal{A}$ to produce $X=(X_1,\ldots,X_n)$.
5. If $\mathsf{Ev}_\pi(F,X)\neq\perp$ and $\mathsf{En}_\pi(\hat{e}_i,\mathsf{En}_\pi^{-1}(\hat{e}_i,X_i))\neq X_i$ for any $i\in[n]$ then output $\top$, otherwise output $\perp$.

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

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}(1^k,f)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F,v) where $(v_1,\ldots,v_{n_\mathsf{B}},v_{n_\mathsf{B}+1},\ldots,v_{n_\mathsf{B}+m_\mathsf{B}})\leftarrow v$.
3. Run Ex_C on $\mathcal{A}$ to extract $\hat{e}=(\hat{e}_1,\ldots,\hat{e}_n)$ and $\hat{d}=\big(\hat{d}_1,\ldots,\hat{d}_m\big)$.
4. Run $\mathcal{A}$ to produce $X=(X_1,\ldots,X_n)$.
5. Compute $Z\leftarrow\mathsf{Ev}_\pi(F,X)$. If $Z=\perp$, then output $\perp$. If $\mathsf{De}_\pi^{-1}(\hat{d}_j,\mathsf{De}_\pi(\hat{d}_j,Z_j))\neq Z_j$ for any $j\in[m]$, then output $\top$, otherwise output $\perp$.

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

1. Run $\mathcal{A}(1^k)$ to produce f.
2. Run Gb_π between $\mathcal{A}$ and $\mathsf{E}(1^k,f)$. If E outputs $\perp$, output $\perp$. Otherwise E outputs some (F,v) where $(v_1,\ldots,v_{n_\mathsf{B}},v_{n_\mathsf{B}+1},\ldots,v_{n_\mathsf{B}+m_\mathsf{B}})\leftarrow v$.
3. Run Ex_C on $\mathcal{A}$ to extract $\hat{e}=(\hat{e}_1,\ldots,\hat{e}_n)$ and $\hat{d}=\big(\hat{d}_1,\ldots,\hat{d}_m\big)$.
4. Run $\mathcal{A}$ to produce $X=(X_1,\ldots,X_n)$, $x'=(x'_1,\ldots,x'_n)$, and $z'=(z'_1,\ldots,z'_m)$.
5. Compute $Z\leftarrow\mathsf{Ev}_\pi(F,X)$ and $z_j\leftarrow\mathsf{De}_\pi(\hat{d}_j,Z_j)$ for $j\in[m]$ and let $x_i\leftarrow\mathsf{En}_\pi^{-1}(\hat{e}_i,X_i)$ for $i\in[n]$. If any $z_j=\perp$ or $x_i=\perp$ output $\perp$.
6. If there is an $i\in[n_\mathsf{B}]$ such that $\mathsf{Ve}_\pi(v_i,X_{n_\mathsf{A}+i},x'_{n_\mathsf{A}+i})=\top$ and $x_{n_\mathsf{A}+i}\neq x'_{n_\mathsf{A}+i}$ or there is an $j\in[m_\mathsf{B}]$ such that $\mathsf{Ve}_\pi(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j},z'_{m_\mathsf{A}+j})=\top$ where $z_{m_\mathsf{A}+j}\neq z'_{m_\mathsf{A}+j}$ then output $\top$, otherwise output $\perp$.

Fig. 3. Security games for interactive garbling scheme – part 2.

Uniqueness of Input Encoding. We also need that there are unique input encodings (property name: unqie) even when the constructor is cheating. We only require uniqueness for encodings which make the garbled evaluation succeed. We need this to ensure that the values the evaluator learns are valid 0- or 1-keys. This is true for the output of each gate, if it is true for the input. Therefore we need this property to start this "induction".

Uniqueness of Output Encoding. Similarly we 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.

Token Commitment. Finally we need a notion of *token commitment* (property name: tok.com). It essentially just says that the verification algorithm is 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.

4 Interactive Garbling Scheme implies UC-secure 2PC

The ideal functionality $\mathcal{F}^f_{\mathsf{SFE}}$ which our protocol realizes is given in Fig. 4. It has been designed 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.

Setup: We denote the two parties of the protocol by A and B. The parties agree on k and f and as shorthand we let $n_{\mathsf{A}} = f.n_{\mathsf{A}}$, $n_{\mathsf{B}} = f.n_{\mathsf{B}}$, $m_{\mathsf{A}} = f.m_{\mathsf{A}}$, $m_{\mathsf{B}} = f.m_{\mathsf{B}}$, and $m = m_{\mathsf{A}} + m_{\mathsf{B}}$.
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 corrupt from A before evaluation, 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 correct! to A. Otherwise, output abort to both parties and terminate.
Evaluation: If both parties gave input, then on input evaluate from B, compute $z_{\mathsf{B}} \leftarrow f_{\mathsf{B}}(x,y)$ and $z_{\mathsf{A}} \leftarrow f_{\mathsf{A}}(x,y)$. Then output z_{B} to B and wait. If B inputs deliver send z_{A} to A and terminate. If instead receiving abort from B and B is corrupt, output abort to A and terminate.

Fig. 4. Ideal Functionality $\mathcal{F}^f_{\mathsf{SFE}}$.

We present our generic protocol π_{IGCO} for UC-realizing $\mathcal{F}^f_{\mathsf{SFE}}$ in Fig. 5. It abstracts and generalizes the protocols in [NO09] and [FJN+13] using our new notion of an interactive garbling scheme. However unlike the previous protocols, π_{IGCO} allows both parties to learn distinct outputs which is also reflected in the description of $\mathcal{F}^f_{\mathsf{SFE}}$.

The protocol is phrased in the $\mathcal{F}_{\mathsf{DOT}}$-hybrid model, a notion of OT which we call *delayed OT*. It is a one-out-of-two OT of κ-bit strings 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 $(m_0,m_1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$, 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 parties first run a setup phase in which the function f and security parameter k are determined. Next B sends its input y to $\mathcal{F}_{\mathsf{DOT}}$ where-after the parties run an interactive garbling Gb_π of f with A playing the role of C and B playing the role of E. Thus the output of A from Gb_π is (F,e,d) and the output of B is (F,v). After learning the input encoding function e, A encodes its input x and sends this to B. It also sends the decoding information d_{B} that is associated with B's designated output. Next it sends the input encodings for B's input to $\mathcal{F}_{\mathsf{DOT}}$ which delivers the input keys to B in correspondence with the earlier choice. Using the verification information v, B checks that it received consistent input keys for its input y. B then evaluates the encoded function F on the encoded input X to obtain an encoded output Z. Using the decoding information d_{B}, B decodes its output to z_{B} which he verifies using the verification information v. If everything checks out he sends back the encoded output Z_{A} to A which uses the decoding information d to obtain her final output. Theorem 1 shows that the protocol π_{IGCO} UC-realizes the functionality $\mathcal{F}^f_{\mathsf{SFE}}$ in the $\mathcal{F}_{\mathsf{DOT}}$-hybrid model.

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

Proof. The case of no corruptions follows easily using the properties proj, corr and sec.ind.act, using a subset of the proof arguments below. If both parties are corrupted, there is nothing to show. The statement then follows directly from Lemma 1 and Lemma 2. □

Lemma 1. *If $\mathcal{G}_\pi$ is an extended interactive garbling scheme and has the properties, sec.ind.act, aut.act, knof, and proj, then π_{IGCO} UC-securely realizes $\mathcal{F}^f_{\mathsf{SFE}}$ against any PPT static and malicious adversary corrupting B.*

Proof. If B is corrupted and A is honest, let $\mathcal{B}$ denote the adversary controlling B. We can assume without loss of generality that this is the UC environment. We prove security through a series of hybrids based on the

Setup: We denote the two parties of the protocol by A and B. The parties agree on k and f and as shorthand we let $n_{\mathsf{A}} = f.n_{\mathsf{A}}$, $n_{\mathsf{B}} = f.n_{\mathsf{B}}$, $n = n_{\mathsf{A}} + n_{\mathsf{B}}$, $m_{\mathsf{A}} = f.m_{\mathsf{A}}$, $m_{\mathsf{B}} = f.m_{\mathsf{B}}$, and $m = m_{\mathsf{A}} + m_{\mathsf{B}}$. 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 $\mathcal{F}^1_{\mathsf{DOT}},\ldots,\mathcal{F}^{n_{\mathsf{B}}}_{\mathsf{DOT}}$. 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 $\mathcal{F}^i_{\mathsf{DOT}}$ and A waits for output chosen from $\mathcal{F}^i_{\mathsf{DOT}}$.

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) and the output to B is (F,v). Furthermore, A sends $d_{\mathsf{B}} = (d_{m_{\mathsf{A}}+1},\ldots,d_m)$ to B.

Input A: Denote the input of A by $x \in \{0,1\}^{n_{\mathsf{A}}}$. A parses e as $\left(e^{\mathsf{A}}_1,\ldots,e^{\mathsf{A}}_{n_{\mathsf{A}}},e^{\mathsf{B}}_1,\ldots,e^{\mathsf{B}}_{n_{\mathsf{B}}}\right)$. Then A lets $X^{x_i}_i \leftarrow \mathsf{En}_\pi(e^{\mathsf{A}}_i, x_i)$ for $i \in [n_{\mathsf{A}}]$ and sends $\left(X^{x_1}_1,\ldots,X^{x_{n_{\mathsf{A}}}}_{n_{\mathsf{A}}}\right)$ to B.

Input B, II: For $i = 1,\ldots,n_{\mathsf{B}}$, A lets $Y^b_i \leftarrow \mathsf{En}_\pi(e^{\mathsf{B}}_i, b)$ for $b \in \{0,1\}$ and inputs $\left(Y^0_i, Y^1_i\right)$ to $\mathcal{F}^i_{\mathsf{DOT}}$ and B waits for output $(Y^{y_i}_i)$ from $\mathcal{F}^i_{\mathsf{DOT}}$. Then B lets $X = \left(X^{x_1}_1,\ldots,X^{x_{n_{\mathsf{A}}}}_{n_{\mathsf{A}}}, Y^{y_1}_1,\ldots,Y^{y_{n_{\mathsf{B}}}}_{n_{\mathsf{B}}}\right)$ and $Y = \left(Y^{y_1}_1,\ldots,Y^{y_{n_{\mathsf{B}}}}_{n_{\mathsf{B}}}\right)$. If $\mathsf{Ve}_\pi(v_i, Y_i, y_i) = \perp$ for any $i \in [n_{\mathsf{B}}]$ then B outputs abort and terminates.

Evaluation: B computes $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j}, Z_{m_{\mathsf{A}}+j})$ for $j \in [m_{\mathsf{B}}]$ and outputs abort if $Z = \perp$ or $z_{\mathsf{B},j} = \perp$. Furthermore, if $\mathsf{Ve}_\pi(v_{n_{\mathsf{B}}+j}, Z_{m_{\mathsf{A}}+j}, z_{\mathsf{B},j}) = \perp$ for any $j \in [m_{\mathsf{B}}]$ then B outputs abort and terminates.

Output: B sends $Z_{\mathsf{A}} = (Z_1,\ldots,Z_{m_{\mathsf{A}}})$ to A and outputs $z_{\mathsf{B}} \leftarrow (z_{\mathsf{B},1},\ldots,z_{\mathsf{B},m_{\mathsf{B}}})$. A computes $z_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi(d_j, Z_{\mathsf{A},j})$ for $j \in [m_{\mathsf{A}}]$. If any $z_{\mathsf{A},j} = \perp$ then A outputs abort. Otherwise, A outputs $z_{\mathsf{A}} \leftarrow (z_{\mathsf{A},1},\ldots,z_{\mathsf{A},m_{\mathsf{A}}})$.

Fig. 5. Generic Protocol π_{IGCO} for 2PC using an Interactive Garbling Scheme in the $\mathcal{F}_{\mathsf{DOT}}$-hybrid model.

properties sec.ind.act, aut.act, knof, and proj such that if an adversary can distinguish between a pair of the hybrids, then he can break at least one of the properties.

The first pair of hybrids are induced by the bit flipped by the secrecy game $\mathrm{SecIndAct}^{\mathcal{B}}_{\mathcal{G}_\pi}(1^k)$. Specifically the bit flipped in the game will define one of two possible simulators as follows, denoting the simulator as $\mathcal{T}$ in general and $\mathcal{T}^b$ where the bit flipped in the game is b.

The simulator 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. If $\mathcal{B}$ ever inputs abort or an honest A would input abort, then $\mathcal{T}$ inputs abort to $\mathcal{F}^f_{\mathsf{SFE}}$. The simulation runs with the following modifications from the protocol:

1. In **Input B, I**, inspect the OTs to learn the choice bits $y_1,\ldots,y_{n_{\mathsf{B}}}$ of $\mathcal{B}$ and define $y = y_1 \cdots y_{n_{\mathsf{B}}}$.
2. Input y to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of B, along with the command $\mathsf{evaluate}$ and receive back $z_{\mathsf{B}} \leftarrow f_{\mathsf{B}}(x,y)$.
3. Let $x' = 0^{n_{\mathsf{A}}}$ denote a *dummy* input and let $z' = f(x',y)$. Now define f' as f, but if an output $z_{\mathsf{B},j} \neq z'_j$ for $j \in [m_{\mathsf{A}}+1;m]$ then replace the j'th output AND gate with a NAND gate. Notice this makes $f'_{\mathsf{B}}(x',y) = f_{\mathsf{B}}(x,y)$. $\mathcal{T}$ then instantiates the game $\mathrm{SecIndAct}^{\mathcal{B}}_{\mathcal{G}_\pi}(1^k)$ on input $(f, f', x\|y, x'\|y)$, playing the role of the adversary, where x is the real input of A which it gets by cheating and looking into $\mathcal{F}^f_{\mathsf{SFE}}$. During the execution of Gb_π we have $\mathcal{T}$ relay $\mathcal{B}$'s input in the simulation directly into the game execution. If the game outputs $\perp$ then $\mathcal{T}$ inputs abort to $\mathcal{F}^f_{\mathsf{SFE}}$. At the end of the game $\mathcal{T}$ knows (F,v,X,d_{B}) and $\mathcal{B}$ has learned (F,v) as part of the execution of Gb_π. The simulator then sends d_{B} to $\mathcal{B}$ as an honest A would.
4. In **Input A**, parse $(X_1,\ldots,X_n) \leftarrow X$ and send $(X_1,\ldots,X_{n_{\mathsf{A}}})$ to $\mathcal{B}$.
5. In **Input B, II**, simulate $\mathcal{F}_{\mathsf{DOT}}$, and return $Y^{y_i}_i = X_{n_{\mathsf{A}}+i}$ for $i \in [m_{\mathsf{B}}]$ to $\mathcal{B}$.
6. Continue the protocol as an honest A until receiving $Z'_{\mathsf{A}} = \left(Z'_{\mathsf{A},1},\ldots,Z'_{\mathsf{A},m_{\mathsf{A}}}\right)$ from $\mathcal{B}$ in the **Output** step.
7. Apply the algorithm Ex_{E} to compute from the communication of $\mathcal{B}$ a garbled function F' and let $Z' \leftarrow \mathsf{Ev}_\pi(F',X)$. If $Z'_{\mathsf{A},j} \neq Z'_j$ for any $j \in [m_{\mathsf{A}}]$, then input abort to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of B.
8. Finally input $\mathsf{deliver}$ to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of $\mathcal{B}$ which results in the functionality outputting $z_{\mathsf{A}} \leftarrow f_{\mathsf{A}}(x,y)$ to A.

It is easy to see that if the SecIndAct game samples bit $b = 1$, then the simulator, $\mathcal{T}^1$ induced by this game is well-defined. That is, it does not do anything it is not allowed to. In more detail, Step 3 is simply computed as follows:

3. Let $x' = 0^{n_{\mathsf{A}}}$ denote a *dummy* input and let $z' \leftarrow f(x',y)$. Now define f' as f, but if an output $z_{\mathsf{B},j} \neq z'_j$ for $j \in [m_{\mathsf{A}}+1;m]$ then replace the j'th output AND gate with a NAND gate. Notice this makes $f'_{\mathsf{B}}(x',y) = f_{\mathsf{B}}(x,y)$. Then run the protocol Gb_π as an honest constructor C with $\mathcal{B}$. If C would ever abort then input abort to $\mathcal{F}^f_{\mathsf{SFE}}$. Let the output of C in this garbling be (F,e,d). Then compute $X_i \leftarrow \mathsf{En}_\pi(e_i, (x'\|y)_i)$ and let $X = (X_1,\ldots,X_n)$.

However, if the bit sampled is 0, then the simulator is cheating. We will now remove this cheating through a series of hybrids, where the properties sec.ind.act, aut.act and knof ensures that each pair of hybrids are indistinguishable. Finally, we argue that the last hybrid induces a view to $\mathcal{B}$ that is indistinguishable from a real world execution.

First notice that it is clear that no adversary can distinguish between playing with $\mathcal{T}^0$ or $\mathcal{T}^1$ except with negligible probability. This follows from everything in the simulation not coming from the game SecIndAct is constructed in exactly the same manner independent of the bit flipped by SecIndAct. Thus, if an adversary could distinguish with non-negligible probability, then he could also win the game with non-negligible probability.

In the following we will take our departure in the hybrid where the bit flipped by SecIndAct is 0 (which we call the first hybrid). A bit more specifically this means that Step 3 can be described as follows for $\mathcal{T}^0$:

3^1. Cheat and inspect $\mathcal{F}^f_{\mathsf{SFE}}$ to learn x. Then run the protocol Gb_π as an honest constructor C with $\mathcal{B}$. If C would ever abort then input abort to $\mathcal{F}^f_{\mathsf{SFE}}$. Let the output of C in this garbling be (F,e,d). Then compute $X_i \leftarrow \mathsf{En}_\pi(e_i,(x\|y)_i)$ and let $X=(X_1,\ldots,X_n)$.

Now consider the second hybrid where we replace Step 7 by the following:

7^1. Let $Z \leftarrow \mathsf{Ev}_\pi(F,X)$. If $Z'_{\mathsf{A},j} \neq Z_j$ for any $j \in [m_\mathsf{A}]$, then input abort to $\mathcal{F}^f_{\mathsf{SFE}}$ 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 7 and Step 8 by this:

7^2. Let $Z \leftarrow \mathsf{Ev}_\pi(F,X)$. If $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j})=\perp$ for any $j \in [m_\mathsf{A}]$, then input abort to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of B.

8^1. Cheat and replace $z_{\mathsf{A},j}$ stored in $\mathcal{F}^f_{\mathsf{SFE}}$ by $z'_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$. Then input $\mathsf{deliver}$ to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of $\mathcal{B}$ which results in the functionality outputting $z'_\mathsf{A} = \left(z'_{\mathsf{A},1},z'_{\mathsf{A},2},\ldots,z'_{\mathsf{A},m_\mathsf{A}}\right)$ to A.

At this point the values (F,e,d) and Z'_A are generated exactly as in game AutAct. It therefore follows from the property aut.act that the probability of $\mathsf{De}_\pi(d_j,Z_j)=\perp \neq z_j$ when $(z_1,\ldots,z_m) \leftarrow \mathsf{ev}_\pi(f,x\|y)$ is negligible. From this we can also conclude that for any $j \in [m_\mathsf{A}]$ the probability of $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j})=\perp$ is negligibly close to the probability that $Z'_{\mathsf{A},j} \neq Z_j$. Hence the change to Step 7 is indistinguishable to $\mathcal{B}$. When $Z'_{\mathsf{A},j}=Z_j$ for $j \in [m_\mathsf{A}]$ it again follows from the property aut.act that the probability of $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j}) \neq z_{\mathsf{A},j}$ for $j \in [m_\mathsf{A}]$ is negligible and hence the change to Step 8 is indistinguishable.

Finally see that the values (F,e,d) and Z'_A are now computed in the same way in the last hybrid and in the generic protocol. Furthermore, the output of $\mathcal{F}^f_{\mathsf{SFE}}$ is patched to be $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$ and in the protocol A also outputs $\mathsf{De}_\pi(d_j,Z'_{\mathsf{A},j})$. Hence the last hybrid is indistinguishable from the protocol in the view of $\mathcal{B}$.

This concludes the proof. □

Lemma 2. *If $\mathcal{G}_\pi$ is an extended interactive garbling scheme and has the properties, rob.con, unqie, unqoe, tok.com, and proj, then π_{IGCO} UC-securely realizes $\mathcal{F}^f_{\mathsf{SFE}}$ against a static and malicious corruption of A.*

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

1. In **Garbling**, simulate an honest E in the execution of Gb_π with $\mathcal{A}$ unless another behavior is specified below.
2. If an honest E would abort in the execution of Gb_π, then abort $\mathcal{A}$ (*i.e.*, input abort to the ideal functionality on behalf of $\mathcal{A}$). Otherwise, let (F,v) denote the output to B. Also denote by $d_\mathsf{B}=(d_{m_\mathsf{A}+1},\ldots,d_m)$ the decoding information received from $\mathcal{A}$.
3. Apply Ex_C to the communication of $\mathcal{A}$ in Gb_π to extract $\hat{e}$ and $\hat{d}$. Then parse $\hat{e}$ as $\left(\hat{e}^\mathsf{A}_1,\ldots,\hat{e}^\mathsf{A}_{n_\mathsf{A}},\hat{e}^\mathsf{B}_1,\ldots,\hat{e}^\mathsf{B}_{n_\mathsf{B}}\right)$ and $\hat{d}$ as $\left(\hat{d}^\mathsf{A}_1,\ldots,\hat{d}^\mathsf{A}_{m_\mathsf{A}},\hat{d}^\mathsf{B}_1,\ldots,\hat{d}^\mathsf{B}_{m_\mathsf{B}}\right)$. For $b \in \{0,1\}$ we denote $\hat{X}^b_i=\mathsf{En}_\pi(e^\mathsf{A}_i,b)$ for $i \in [n_\mathsf{A}]$ and $\hat{Y}^b_i=\mathsf{En}_\pi(e^\mathsf{B}_i,b)$ for $i \in [n_\mathsf{B}]$.
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 $X_i \notin \left\{\hat{X}^0_i,\hat{X}^1_i\right\}$ or $\hat{X}^0_i=\hat{X}^1_i$. If there is a bad index, then let $x=0^{n_\mathsf{A}}$. Otherwise, let each x_i be the unique bit such that $X_i=\hat{X}^{x_i}_i$, and set $x=(x_1,\ldots,x_{n_\mathsf{A}})$. Notice that $(X_1,\ldots,X_{n_A})=(\hat{X}^{x_1}_1,\ldots,\hat{X}^{x_{n_\mathsf{A}}}_{n_\mathsf{A}})$ if there are no bad indices.
5. In **Input B, II**, inspect the OTs to learn the messages $\left(Y^0_i,Y^1_i\right)$ input to $\mathcal{F}^i_{\mathsf{DOT}}$ by $\mathcal{A}$. We call (i,b) a *faulty position* if $Y^b_i \neq \hat{Y}^b_i$. 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 $\mathcal{F}^f_{\mathsf{SFE}}$ 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 γ_i be the unique bit for which (i,γ_i) is not faulty. If $|I| > 0$, then input $\{(i,\gamma_i)\}_{i\in I}$ to $\mathcal{F}^f_{\mathsf{SFE}}$. If the output is abort, then terminate the simulated protocol. Otherwise, define a *dummy input* y' for B by letting $y'_i = \gamma_i$ for $i \in I$ and $y'_i = 0$ for $i \notin I$. Then let $X = (X_1,\ldots,X_{n_\mathsf{A}},Y_1^{y'_1},\ldots,Y_{n_\mathsf{B}}^{y'_{n_\mathsf{B}}})$. If $\mathsf{Ve}_\pi(v_i,Y_i^{y'_i},y'_i) = \bot$ for any $i \in [n_\mathsf{B}]$, then abort A (*i.e.*, input abort to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of the corrupted A) and terminate the simulated protocol.

6. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z'_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(\hat{d}^\mathsf{B}_j,Z_{m_\mathsf{A}+j})$ for $j \in [m_\mathsf{B}]$. If $Z = \bot$, $z'_{\mathsf{B},j} = \bot$ or $\mathsf{Ve}_\pi\big(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j}, z'_{\mathsf{B},j}\big) = \bot$ for any $j \in [m_\mathsf{B}]$, then abort A. Otherwise, input x to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of the corrupted A and receive back $z_\mathsf{A} \leftarrow f_\mathsf{A}(x,y)$. Let $Z'_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi^{-1}(\hat{d}_j,z_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$ and send $Z'_\mathsf{A} = (Z'_{\mathsf{A},1},\ldots,Z'_{\mathsf{A},m_\mathsf{A}})$ 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**, inspect the OTs to learn the messages $\big(Y_i^0,Y_i^1\big)$ input to $\mathcal{F}^i_{\mathsf{DOT}}$ by $\mathcal{A}$. Then cheat and inspect $\mathcal{F}^f_{\mathsf{SFE}}$ 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 $X = (X_1,\ldots,X_{n_\mathsf{A}},Y_1^{y'_1},\ldots,Y_{n_\mathsf{B}}^{y'_{n_\mathsf{B}}})$, and if $\mathsf{Ev}_\pi(F,X) \to Z \neq \bot$ and $\mathsf{Ve}_\pi(v_i,Y_i^{y'_i},y'_i) = \bot$ for any $i \in [n_\mathsf{B}]$ then abort A and terminate the simulated protocol. If any (i,y'_i) is faulty, then send "$\mathsf{hybrid!}$" to the adversary if this was not already done.

The simulation and the hybrid are indistinguishable to the adversary. As a stepping stone towards showing this we show 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 $Y_i^{y'_i} \neq \hat{Y}_i^{y'_i}$. If $\mathsf{Ev}_\pi(F,X) \to Z = \bot$ or $\mathsf{Ve}_\pi(v_i,Y_i^{y'_i},y'_i) = \bot$ for any $i \in [n_\mathsf{B}]$ the hybrid clearly aborts. Therefore, assume that (i,y'_i) is faulty and that the algorithms do not output $\bot$. Then a simple reduction to the property unqie shows that $Y_i^{y'_i} \in \Big\{\hat{Y}_i^0,\hat{Y}_i^1\Big\}$, as else $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{B}_i,Y_i^{y'_i}) = \bot$ and this break unqie. Furthermore a reduction to the property tok.com shows that $Y_i^{y'_i} \neq \hat{Y}_i^{1-y'_i}$, since if this was not the case then $\mathsf{Ve}_\pi(v_i,Y_i^{y'_i},1-y'_i) = \top$. 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 $\gamma_i \neq y_i$. It follows from the above fact that we do the same in the hybrid, as $(i,1-\gamma_i)$ 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 $(Z'_{\mathsf{A},1},\ldots,Z'_{\mathsf{A},m_\mathsf{A}})$ where $Z'_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi^{-1}(\hat{d}_j,z_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$ which is identically distributed in the simulation and the hybrid. Finally we send the string "$\mathsf{hybrid!}$" 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 $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{A}_l,X_l) \neq x_l$ for any $l \in [n_\mathsf{A}]$ or $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{B}_i,Y_i) \neq y'_i$ for any $i \in [n_\mathsf{B}]$, then send "$\mathsf{hybrid!}$" to the adversary if this was not already done.

Clearly if the above change makes a difference it must be the case that "$\mathsf{hybrid!}$" has not previously been sent to the adversary. We will show by a simple case analysis that the probability with which "$\mathsf{hybrid!}$" is sent to the adversary in $\mathsf{5}^2$ is negligible as $\mathcal{G}_\pi$ has the unqie property. Assume first that there is a bad index l. Then $\mathsf{En}_\pi(\hat{e}^\mathsf{A}_l,x_l) \neq X_l$ by definition of En_π being the projective encoding algorithm. So, since $\mathsf{En}_\pi(\hat{e}^\mathsf{A},\bot) = \bot$ and by construction $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{A}_l,X_l) = \bot$, it follows that $\mathsf{En}_\pi(\hat{e}^\mathsf{A}_l,\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{A}_l,X_l)) = \bot \neq X_l$ if l is a bad index. Notice that at the point where the addition is made we have $\mathsf{Ev}_\pi(F,X) \neq \bot$, since the protocol did not abort. By a simple reduction to unqie it follows that "$\mathsf{hybrid!}$" is sent with negligible probability if there is a bad index l. Assume then that there is no bad index. If we send "$\mathsf{hybrid!}$" in Step 5^2, then there are no faulty positions (i,y'_i) either, as we would have sent "$\mathsf{hybrid!}$" already in Step 5^1. When there are no bad indicis and no faulty positions (i,y'_i), then $(X_1,\ldots,X_{n_A}) = (\hat{X}_1^{x_1},\ldots,\hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}})$ and all $Y_i^{y'_i} = \hat{Y}_i^{y'_i}$. This contradicts that $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{A}_l,X_l) \neq x_l$ for any $l \in [n_\mathsf{A}]$ or $\mathsf{En}_\pi^{-1}(\hat{e}^\mathsf{B}_i,Y_i) \neq y'_i$ for any $i \in [n_\mathsf{B}]$. We therefore conclude that the hybrids are indistinguishable.

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 $\mathsf{Ev}_\pi(F,X) = \bot$, then abort A. If there is a bad index and $\mathsf{Ev}_\pi(F,X) \neq \bot$, 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 set $x = x_1\|\cdots\|x_{n_\mathsf{A}}$. If $(X_1,\ldots,X_{n_A}) \neq (\hat{X}_1^{x_1},\ldots,\hat{X}_{n_\mathsf{A}}^{x_{n_\mathsf{A}}})$, then send "hybrid!" to the adversary, if this was not already done.

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

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

6^1. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j},Z_{m_\mathsf{A}+j})$ for $j \in [m_\mathsf{B}]$. If $Z = \bot$, $z_{\mathsf{B},j} = \bot$ or $\mathsf{Ve}_\pi\big(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j}, z_{\mathsf{B},j}\big) = \bot$ for any $j \in [m_\mathsf{B}]$, then abort A. If $\mathsf{De}_\pi(\hat{d}_j^\mathsf{B},Z_{m_\mathsf{A}+j}) \neq z_{\mathsf{B},j}$ for any $j \in [m_\mathsf{B}]$, then send "hybrid!" to $\mathcal{A}$, if not already done. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z_\mathsf{A} \leftarrow f_\mathsf{A}(x,y)$. Let $Z'_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi^{-1}(\hat{d}_j,z_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$ and send $Z'_\mathsf{A} = (Z'_{\mathsf{A},1},\ldots,Z'_{\mathsf{A},m_\mathsf{A}})$ to $\mathcal{A}$ as if coming from B.

It is clear that if we do not send the string "hybrid!" in the above step then the change is indistinguishable from Step 6. We therefore argue that "hybrid!" is sent with negligible probability. Assume "hybrid!" is sent to $\mathcal{A}$, then clearly A was not aborted and therefore $\mathsf{Ev}_\pi(F,X) \to Z \neq \bot$, $\mathsf{De}_\pi(d_{\mathsf{B},j},Z_{m_\mathsf{A}+j}) \to z_{\mathsf{B},j} \neq \bot$ and $\mathsf{Ve}_\pi(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j},z_{\mathsf{B},j}) = \top$ for all $j \in [m_\mathsf{B}]$. As "hybrid!" is sent there is also a $j \in [m_\mathsf{B}]$ such that $\mathsf{De}_\pi(\hat{d}_j^\mathsf{B},Z_{m_\mathsf{A}+j}) \neq z_{\mathsf{B},j}$, but by a reduction to tok.com this only occurs with negligible probability. We thus conclude that the hybrids are indistinguishable.

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

6^2. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j},Z_{m_\mathsf{A}+j})$ for $j \in [m_\mathsf{B}]$. If $Z = \bot$, $z_{\mathsf{B},j} = \bot$ or $\mathsf{Ve}_\pi\big(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j}, z_{\mathsf{B},j}\big) = \bot$ for any $j \in [m_\mathsf{B}]$, then abort A. If $\mathsf{De}_\pi(\hat{d}_j^\mathsf{B},Z_{m_\mathsf{A}+j}) \neq z_{\mathsf{B},j}$ for any $j \in [m_\mathsf{B}]$, then send "hybrid!" to $\mathcal{A}$, if not already done. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z_\mathsf{A} \leftarrow f_\mathsf{A}(x,y)$. If $\mathsf{De}_\pi(\hat{d}_j^\mathsf{A},Z_j) \neq z_{\mathsf{A},j}$ for any $j \in [m_\mathsf{A}]$, then send "hybrid!" to $\mathcal{A}$, if not already done. Finally let $Z'_{\mathsf{A},j} \leftarrow \mathsf{De}_\pi^{-1}(\hat{d}_j,z_{\mathsf{A},j})$ for $j \in [m_\mathsf{A}]$ and send $Z'_\mathsf{A} = \Big(Z'_{\mathsf{A},1},\ldots,Z'_{\mathsf{A},m_\mathsf{A}}\Big)$ to $\mathcal{A}$ as if coming from B.

If we send the string "hybrid!", then it was not sent by the previous changes and hence $X_l = \mathsf{En}_\pi(\hat{e}_l^\mathsf{A},x_l)$ for $l \in [n_\mathsf{A}]$ and $Y_i^{y_i} = \mathsf{En}_\pi(\hat{e}_i^\mathsf{B},y_i)$ for $i \in [n_\mathsf{B}]$. At the point when we send "hybrid!" we know that $\mathsf{Ev}_\pi(F,X) \neq \bot$ and $z_\mathsf{A} \leftarrow f_\mathsf{A}(x,y)$ and yet there exists a $j \in [m_\mathsf{A}]$ such that $\mathsf{De}_\pi(\hat{d}_j^\mathsf{A},Z_j) \neq z_{\mathsf{A},j}$. 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:

6^3. Compute $Z \leftarrow \mathsf{Ev}_\pi(F,X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j},Z_{m_\mathsf{A}+j})$ for $j \in [m_\mathsf{B}]$. If $Z = \bot$, $z_{\mathsf{B},j} = \bot$ or $\mathsf{Ve}_\pi\big(v_{n_\mathsf{B}+j},Z_{m_\mathsf{A}+j}, z_{\mathsf{B},j}\big) = \bot$ for any $j \in [m_\mathsf{B}]$, then abort A. If $\mathsf{De}_\pi(\hat{d}_j^\mathsf{B},Z_{m_\mathsf{A}+j}) \neq z_{\mathsf{B},j}$ for any $j \in [m_\mathsf{B}]$, then send "hybrid!" to $\mathcal{A}$, if not already done. Otherwise, input x to $\mathcal{F}_{\mathsf{SFE}}^f$ on behalf of the corrupted A and receive back $z_\mathsf{A} \leftarrow f_\mathsf{A}(x,y)$. If $\mathsf{De}_\pi(\hat{d}_j^\mathsf{A},Z_j) \neq z_{\mathsf{A},j}$ for any $j \in [m_\mathsf{A}]$, then send "hybrid!" to $\mathcal{A}$, if not already done. Finally let $Z_\mathsf{A} = (Z_1,\ldots,Z_{m_\mathsf{A}})$ and send Z_A to $\mathcal{A}$ as if coming from B.

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 $Z_\mathsf{A} \neq Z'_\mathsf{A}$. Note then that if there is a $j \in [m_\mathsf{A}]$ such that $\mathsf{De}_\pi(\hat{d}_j^\mathsf{A},Z_j) \neq z_{\mathsf{A},j}$, then in both the new hybrid and the previous hybrid we send "hybrid!" to $\mathcal{A}$, which allows $\mathcal{A}$ to perfectly distinguish from the simulation, where no such strings are sent, so sending $Z_\mathsf{A} \neq Z'_\mathsf{A}$ 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 there exists a $j \in [m_{\mathsf{A}}]$ such that both $\mathsf{De}_\pi(\hat{d}^{\mathsf{A}}_j, Z_j) = z_{\mathsf{A},j}$ and $\mathsf{De}^{-1}_\pi(\hat{d}^{\mathsf{A}}_j, z_{\mathsf{A},j}) \neq Z_j$. Putting these two together we get that $\mathsf{De}^{-1}_\pi(\hat{d}, \mathsf{De}_\pi(\hat{d}, Z_j)) \neq Z_j$. The claim therefore follows from a trivial reduction to unqoe.

Now consider a hybrid, where we do not send the strings "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 string "hybrid!" we see that the new hybrid looks as follows:

1. In **Garbling**, simulate an honest E in the execution of Gb_π with $\mathcal{A}$ unless another behavior is specified below.
2. If an honest E would abort in the execution of Gb_π, then abort $\mathcal{A}$ (*i.e.*, input abort to the ideal functionality on behalf of $\mathcal{A}$). Otherwise, let (F, v) denote the output to B. Also denote by $d_{\mathsf{B}} = (d_{m_{\mathsf{A}}+1}, \ldots, d_m)$ the decoding information received from $\mathcal{A}$.
3. Apply Ex_{C} to the communication of $\mathcal{A}$ in Gb_π to extract $\hat{e}$ and $\hat{d}$. Then parse $\hat{e}$ as $\left(\hat{e}^{\mathsf{A}}_1, \ldots, \hat{e}^{\mathsf{A}}_{n_{\mathsf{A}}}, \hat{e}^{\mathsf{B}}_1, \ldots, \hat{e}^{\mathsf{B}}_{n_{\mathsf{B}}}\right)$ and $\hat{d}$ as $\left(\hat{d}^{\mathsf{A}}_1, \ldots, \hat{d}^{\mathsf{A}}_{m_{\mathsf{A}}}, \hat{d}^{\mathsf{B}}_1, \ldots, \hat{d}^{\mathsf{B}}_{m_{\mathsf{B}}}\right)$. For $b \in \{0,1\}$ we denote $\hat{X}^b_i = \mathsf{En}_\pi(e^{\mathsf{A}}_i, b)$ for $i \in [n_{\mathsf{A}}]$ and $\hat{Y}^b_i = \mathsf{En}_\pi(e^{\mathsf{B}}_i, b)$ for $i \in [n_{\mathsf{B}}]$.

4^2. 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 $X_i \notin \left\{\hat{X}^0_i, \hat{X}^1_i\right\}$ or $\hat{X}^0_i = \hat{X}^1_i$. If there is a bad index and $\mathsf{Ev}_\pi(F, X) = \perp$, then abort A. If there is a bad index and $\mathsf{Ev}_\pi(F, X) \neq \perp$, then let $x = 0^{n_{\mathsf{A}}}$. Otherwise, let each x_i be the unique bit such that $X_i = \hat{X}^{x_i}_i$, and set $x = x_1 \| \cdots \| x_{n_{\mathsf{A}}}$.

5^3. In **Input B, II**, inspect the OTs to learn the messages $\left(Y^0_i, Y^1_i\right)$ input to $\mathcal{F}^i_{\mathsf{DOT}}$ by $\mathcal{A}$. Then cheat and inspect $\mathcal{F}^f_{\mathsf{SFE}}$ 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 $X = (X_1, \ldots, X_{n_{\mathsf{A}}}, Y^{y'_1}_1, \ldots, Y^{y'_{n_{\mathsf{B}}}}_{n_{\mathsf{B}}})$, and if $\mathsf{Ev}_\pi(F, X) \to Z \neq \perp$ and $\mathsf{Ve}_\pi(v_i, Y^{y'_i}_i, y'_i) = \perp$ for any $i \in [n_{\mathsf{B}}]$ then abort A and terminate the simulated protocol.

6^4. Compute $Z \leftarrow \mathsf{Ev}_\pi(F, X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j}, Z_{m_{\mathsf{A}}+j})$ for $j \in [m_{\mathsf{B}}]$. If $Z = \perp$, $z_{\mathsf{B},j} = \perp$ or $\mathsf{Ve}_\pi\left(v_{n_{\mathsf{B}}+j}, Z_{m_{\mathsf{A}}+j}, z_{\mathsf{B},j}\right) = \perp$ for any $j \in [m_{\mathsf{B}}]$, then abort A. Otherwise, input x to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of the corrupted A and receive back $z_{\mathsf{A}} \leftarrow f_{\mathsf{A}}(x, y)$. Finally let $Z_{\mathsf{A}} = (Z_1, \ldots, Z_{m_{\mathsf{A}}})$ and send Z_{A} to $\mathcal{A}$ as if coming from B.

In Step 6^4 we can drop the code *Otherwise, input x to $\mathcal{F}^f_{\mathsf{SFE}}$ on behalf of the corrupted A and receive back $z_{\mathsf{A}} \leftarrow f_{\mathsf{A}}(x, y)$* as it has no effect at this point. In Step 4^2 we can also drop the code *If there is a bad index and $\mathsf{Ev}_\pi(F, X) = \perp$, then abort A* as we would abort in Step 6 anyway when $\mathsf{Ev}_\pi(F, X) = \perp$. After that all the code of Step 4^2 used to define x can be dropped, as x is not used later on anymore. Due to this we can also 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**, simulate an honest E in the execution of Gb_π with $\mathcal{A}$ unless another behavior is specified below.
2. If an honest E would abort in the execution of Gb_π, then abort $\mathcal{A}$ (*i.e.*, input abort to the ideal functionality on behalf of $\mathcal{A}$). Otherwise, let (F, v) denote the output to B. Also denote by $d_{\mathsf{B}} = (d_{m_{\mathsf{A}}+1}, \ldots, d_m)$ the decoding information received from $\mathcal{A}$.

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**, inspect the OTs to learn the messages $\left(Y^0_i, Y^1_i\right)$ input to $\mathcal{F}^i_{\mathsf{DOT}}$ by $\mathcal{A}$. Then cheat and inspect $\mathcal{F}^f_{\mathsf{SFE}}$ to get the real value y of the input of B, as given by the environment. Then let $X = (X_1, \ldots, X_{n_{\mathsf{A}}}, Y^{y_1}_1, \ldots, Y^{y_{n_{\mathsf{B}}}}_{n_{\mathsf{B}}})$, and if $\mathsf{Ev}_\pi(F, X) \to Z \neq \perp$ and $\mathsf{Ve}_\pi(v_i, Y^{y_i}_i, y_i) = \perp$ for any $i \in [n_{\mathsf{B}}]$ then abort A and terminate the simulated protocol.

6^5. Compute $Z \leftarrow \mathsf{Ev}_\pi(F, X)$ and $z_{\mathsf{B},j} \leftarrow \mathsf{De}_\pi(d_{\mathsf{B},j}, Z_{m_{\mathsf{A}}+j})$ for $j \in [m_{\mathsf{B}}]$. If $Z = \perp$, $z_{\mathsf{B},j} = \perp$ or $\mathsf{Ve}_\pi\left(v_{n_{\mathsf{B}}+j}, Z_{m_{\mathsf{A}}+j}, z_{\mathsf{B},j}\right) = \perp$ for any $j \in [m_{\mathsf{B}}]$, then abort A. Finally let $Z_{\mathsf{A}} = (Z_1, \ldots, Z_{m_{\mathsf{A}}})$ and send Z_{A} to $\mathcal{A}$ as if coming from B.

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. $\square$

5 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 introducing the abstract notion of a *Key-Size Preserving Free-XOR Gate Garbling Scheme* with *projective coding* and show how this supports soldering of gates. Next we use a correlation robust and collision resistant hash function to construct a gadget we call a wire authenticator. Together the garbled gates and the wire authenticators will be used to build the final garbled circuit. As in the previous LEGO protocol, in order to glue these objects together we also require commitments that allow for XOR-homomorphic operations.

Free-XOR Gate Garbling. We will take our departure in any projective coding garbling scheme which obeys some further constraints. We capture these constrains in the following definition.

Definition 1 (Key-Size Preserving Free-XOR Gate Garbling Scheme). *We say that a* projective coding *garbling scheme* $\mathcal{G}$ *is a* Key-Size Preserving Free-XOR Gate Garbling Scheme *if for all* $f \rightarrow (n_A, n_B, m_A, m_B, q, \mathsf{lp}, \mathsf{rp})$ *with* $k \in \mathbb{N}$ *where* $(F,e,d) \leftarrow \mathsf{Gb}(1^k, f)$ *it is possible to efficiently and uniquely parse* $F \rightarrow (\gamma, \delta_{n+1}, \delta_{n+2}, \dots, \delta_w)$. *Furthermore the following must hold:*

1. *There exists an efficient procedure* $\mathsf{GEv}(\delta_g, X_l^a, X_r^b, \gamma_g) \rightarrow X_g^{a\wedge b}$ *for* $g \in \mathsf{Gates}$ *and* $a, b \in \{0,1\}$ *where* $X_l^0 = \bigoplus_{i \in \mathsf{lp}(g)} X_i^0$, $X_r^0 = \bigoplus_{i \in \mathsf{rp}(g)} X_i^0$ *and* $(X_1^0, X_1^1, \dots, X_n^0, X_n^1) \leftarrow e$. *Finally we require that for all wires* $j \in \mathsf{Wires}$ *of the garbled gates it holds that* $X_j^0 \oplus X_j^1 = \Delta$ *where* Δ *must be a single bitstring uniquely defined from an execution of* $\mathsf{Gb}(1^k, f)$.
2. *For* $j \in [m]$ *and* $b \in \{0,1\}$ *we have* $Z_j^b = X_{w-m+j}^b$.
3. *We have that* $|X_g^a| = |X_{g+1}^b| = \kappa$ *for* $a, b \in \{0,1\}$ *and* $g \in [w-1]$.

On an intuitive level the above definition states three specific requirements of our underlaying garbling scheme:

Gate Garbling: Each AND gate given by the topology of the plain function description, f, will have a one-to-one correspondence to garbled gate δ_g in the garbled circuit, F. In a similar manner there will be a one-to-one correspondence between each 0- respectively 1-bit on each wire in the plain function description f and a key in the garbled circuit.

Key-Size Preservation: All of the keys associated with the garbled circuit will have equal size. That is, there will be two equally sized tokens associated with each wire defined by the topology of f. Furthermore, this size will be constant over all wires.

Free-XOR Garbling: Computation of XOR gates is defined implicitly by the left, respectively right parent functions from f, and are handled without the need of storing specific data since the keys given by computing an arbitrary fan-in XOR gate are defined by simply XOR'ing the input keys together. In effect of this we need that for any pair of wire keys, X_j^0, X_j^1, for $j \in \mathsf{Wires}$ it must hold that $X_j^0 \oplus X_j^1 = \Delta$.

We note that the above definition only makes sense if both the topology and which gates compute XOR are leaked (as in any gate garbling scheme). For convenience we define $\mathbf{\Phi}_{\mathsf{xor}}$ to be the leakage function leaking the topology and positions of XOR gates of the circuit. That is, letting $f \rightarrow (n_{\mathsf{A}}, n_{\mathsf{B}}, m_{\mathsf{A}}, m_{\mathsf{B}}, q, \mathsf{lp}, \mathsf{rp})$ we have $\mathbf{\Phi}_{\mathsf{xor}}(f) = (n_{\mathsf{A}}, n_{\mathsf{B}}, m_{\mathsf{A}}, m_{\mathsf{B}}, q, \mathsf{lp}, \mathsf{rp})$. For technical reasons we also require that $\mathsf{lsb}(\Delta) = 1$, for example, as done in garbling schemes using *permutation bits* [Rog91, NPS99].

Soldering. As in [NO09, FJN+13], our scheme requires the ability to solder wires together, a concept made possible due to the free-XOR optimization. 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 δ_g to the left input wire of gate δ_{g+1}. To do so we release the value $S_{g+1}^L = X_g^0 \oplus X_{l_{g+1}}^0$. When gate δ_g outputs the key representing the bit b then it is easy to learn the left b-key for gate δ_{g+1}. Specifically it can be computed as follows:

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

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

$$S_{g+1}^L = X_g^0 \oplus X_{l_{g+1}}^0, \quad S_{g+2}^L = X_g^0 \oplus X_{l_{g+2}}^0, \quad S_{g+3}^L = X_g^0 \oplus X_{l_{g+3}}^0 \ .$$

It is also easy to "inject" XOR gates into the soldering: Say we wish to compute the XOR of the output of gate δ_g and δ_{g+1} and solder the result to the left wire of gate δ_{g+2} we simply release the value $S^L_{g+2} = \left(X^0_g \oplus X^0_{g+1}\right) \oplus X^0_{l_{g+2}}$. In general we 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. In Fig. 6 we sum up how to augment any Key-Size Preserving Free-XOR Gate Garbling Scheme to support solderings.

$\mathsf{SGb}\left(\bar{\gamma}_g, \bar{S}^L_g, \bar{S}^R_g, \bar{S}^O_g\right) \to \widetilde{\gamma}_g$:
1. Parse $\left(\gamma_g, S^L_g, S^R_g, S^O_g\right) \leftarrow \bar{\gamma}_g$.
2. Set $\widetilde{\gamma}_g \leftarrow \left(\gamma_g, \bar{S}^L_g \oplus S^L_g, \bar{S}^R_g \oplus S^R_g, \bar{S}^O_g \oplus S^O_g\right)$ and output $\widetilde{\gamma}_g$.

$\mathsf{SEv}\left(\delta_g, X^a_l, X^b_r, \widetilde{\gamma}_g\right) \to X^{a\wedge b}_g$:
1. Parse $\left(\gamma_g, S^L_g, S^R_g, S^O_g\right) \leftarrow \widetilde{\gamma}_g$.
2. $X^{a\wedge b}_g \leftarrow \mathsf{GEv}\left(\delta_g, X^a_l \oplus S^L_g, X^b_r \oplus S^R_g, \gamma_g\right)$.
3. Output $X^{a\wedge b}_g \oplus S^O_g$.

Fig. 6. The interface provided by a Key-Size Preserving Free-XOR Gate Garbling Scheme augmented with support for solderings. The remaining algorithms are the same as for the standard notion of a projective garbling scheme.

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. We also use the wire authenticators to guarantee validity of the keys B receives from A before evaluating the circuit. The mentioned performance gain comes from the fact that wire authenticators are less costly to produce and send than garbled gates are.

A wire authenticator is a gadget that takes as input a value and outputs accept if this value is 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 enables B to notice (with high probability) if a given key is a mismatch for the current wire. This idea was independently used in [KMRR15] for a different context to reduce the potential leakage using the dual-execution protocol of [MF06].

Let $\mathcal{H}(\cdot,\cdot)$ denote a correlation-robust and collision-resistant hash function with k'-bit output length.

$\mathsf{Aut}\left(X^0_j, X^1_j\right) \to H_j$:
1. Compute
$$H^0_j \leftarrow \mathcal{H}\left(X^0_j\right), \quad H^1_j \leftarrow \mathcal{H}\left(X^1_j\right) .$$
2. View H^0_j and H^1_j as binary strings and output $H_j = \left(H^0_j, H^1_j, 0^\kappa\right)$ if $H^0_j \leq H^1_j$, otherwise output $H_j = \left(H^1_j, H^0_j, 0^\kappa\right)$.

$\mathsf{SAut}\left(H_j, \bar{S}_j\right) \to \widetilde{H}_j$:
1. Parse $\left(H^a_j, H^b_j, S_j\right) \leftarrow H_j$ and output $\widetilde{H}_j \leftarrow \left(H^a_j, H^b_j, \bar{S}_j \oplus S_j\right)$.

$\mathsf{Ver}\left(\widetilde{H}_j, X_j\right) \to \top/\bot$:
1. Parse $\left(H^a_j, H^b_j, S_j\right) \leftarrow \widetilde{H}_j$.
2. If $\mathcal{H}(X_j \oplus S_j) \in \left\{H^a_j, H^b_j\right\}$ output $\top$, otherwise output $\bot$.

Fig. 7. Constructing Authenticated Wires using a Correlation Robust Hash Function.

We show how to implement a wire authenticator using a correlation robust and collision resistant hash function in Fig. 7. For convenience we also introduce methods for authentication, soldering and verifying values in this figure. In short, the Aut method constructs a piece of information H_j, which can be used by Ver to verify that a candidate X_j is one of two values. More specifically this reflects that the first method in the figure constructs an authenticator on the two possible keys on a given wire. The second method in the figure is used as short-hand for soldering authenticated wires onto regular ones. Finally the third method in the figure 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.

Implementation wise, the method $\mathsf{Aut}(X_j^0,X_j^1)$ constructs hash digests of two keys and lets the largest (when viewing the bits of the digest as the binary representation of an integer) of the two resulting bit strings be the first authenticated value. The two digests are then stored in H_j. In the same manner as the garbled gates supporting solderings we use the method SAut to solder wire authenticators onto other wires of the circuit. Finally the method $\mathsf{Ver}(\widetilde{H}_j,X_j)$ constructs a hash digest of X_j and then verifies that it matches either the first entry or second entry of $\widetilde{H}_j$. Because the output of the hash function is pseudorandom and the digests are always sorted this does not leak whether $X_j=X_j^0$ or $X_j=X_j^1$. Using the wire authenticators as described above enables B to check if a given key is valid for a given wire, but cannot be used to verify if a key represents a specific value.

XOR-Homomorphic Commitments. Our final building block is a UC-secure commitment scheme that allows for XOR-homomorphic operations on committed values. The requirement for a XOR-homomorphic commitment scheme is tied to the concept of solderings which our protocol makes heavy use of. Therefore besides the standard operations such as commit and open, we require that it is also possible to open to the XOR of committed values. As we will commit to all wires of the garbled circuit, the scheme must also support commitments to values of the same domain as the keys used by the garbling scheme. The ideal functionality $\mathcal{F}_{\mathsf{HCOM}}$ in Fig. 8 describes in detail what is needed for our construction. We will often abuse notation and commit to multiple messages in one go using the notation $(\mathsf{commit},\mathsf{sid},(\mathsf{cid}_1,\boldsymbol{m}_1),(\mathsf{cid}_2,\boldsymbol{m}_2),\ldots,(\mathsf{cid}_l,\boldsymbol{m}_l))$ to mean committing to l messages individually. In addition we also require a traditional non-homomorphic commitment scheme as part of our protocol, which is captured by the ideal functionality $\mathcal{F}_{\mathsf{COM}}$. We do not give a box for this functionality as it is identical to $\mathcal{F}_{\mathsf{HCOM}}$, except that it does not allow for homomorphic operations on committed values. It is for sake of flexibility that we separate the two requirements as this might lead to more efficient realizations of our interactive garbling scheme. There is however nothing that prevents using the same scheme to implement both functionalities. In Section 7 we go into detail on how the functionality $\mathcal{F}_{\mathsf{HCOM}}$ can be efficiently instantiated.

Init: Upon receiving a message $(\mathsf{init},\mathsf{sid},\mathsf{len})$ from both parties A and B, store the message length len.
Commit: Upon receiving a message $(\mathsf{commit},\mathsf{sid},\mathsf{cid},\boldsymbol{m})$ from A where $\boldsymbol{m}\in\{0,1\}^{\mathsf{len}}$, store the tuple $(\mathsf{sid},\mathsf{cid},\boldsymbol{m}_{\mathsf{cid}})$. Then send $(\mathsf{committed},\mathsf{sid},\mathsf{cid})$ to A and $(\mathsf{receipt},\mathsf{sid},\mathsf{cid})$ to B.
Open: Upon receiving a message $\left(\mathsf{open},\mathsf{sid},\{c\}_{c\in C}\right)$ from A, if for all $c\in C$, a tuple $(\mathsf{sid},c,\boldsymbol{m}_c)$ was previously recorded, send $\left(\mathsf{opened},\mathsf{sid},\{c\}_{c\in C},\bigoplus_{c\in C}\boldsymbol{m}_c\right)$ to B. Otherwise, ignore.

Fig. 8. Ideal Functionality $\mathcal{F}_{\mathsf{HCOM}}$.

6 Instantiation of an Interactive Garbling Scheme

In this section we present our implementation of an interactive garbling scheme. We start from any projective coding, key-size preserving free-xor gate garbling scheme $\mathcal{G}=(\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})$, 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) while E outputs (F,v). We denote our realization of an interactive garbling scheme $\mathsf{IGarb}=(\mathsf{IGb},\mathsf{IEn},\mathsf{IDe},\mathsf{IEv},\mathsf{Iev},\mathsf{IVe})$. IGb is the garbling protocol and it is described in the $(\mathcal{F}_{\mathsf{HCOM}}\text{-},\mathcal{F}_{\mathsf{COM}})$-hybrid model. The remaining five algorithms are based more or less directly on the underlying algorithms of $\mathcal{G}$.

In a nutshell, our protocol consists of doing cut-and-choose of independently garbled gates and wire authenticators, 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 for each bucket. In other words, if the gates of any bucket disagree on the output key (after soldering) then the attached wire authenticators are invoked and the key which is output/accepted by a majority of both gates and wires will be chosen as output key. As wire authenticators are lighter than gates, in terms of computation and communication, we get a significant increase in performance over MiniLEGO where buckets only consisted of gates. We start by giving an informal description of the elements of our garbling scheme, while in Section 6.1 we show the full details of our scheme and prove that it meets the requirements of an interactive garbling scheme.

Setup B starts by committing, using the commitment scheme $\mathcal{F}_{\mathsf{COM}}$, to his challenges for the cut-and-choose phase, and a specification of how the gates and authenticators are to be soldered together into buckets.

Garbling Next, using Gb and $\mathcal{H}$, A produces sufficiently many garble gates and wire authenticators and sends these to B. Next A commits to all the 0-keys of the garbled gates and wire authenticators, including the global difference Δ, using $\mathcal{F}_{\mathsf{HCOM}}$. She also commits to $2s$ additional random values which will be used for leaking the least significant bits associated to B's designated input and output wires. After this, B opens his cut-and-choose 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.[3] A opens to the chosen values and B evaluates the gates and authenticators. If any discrepancy is found he aborts the protocol.

Soldering Next B 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, A 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. Furthermore, A solders the required authenticators onto the output wire of the head gate.

Topological Soldering For each bucket, the left and right parents' 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 has more than one parent, the input is defined to be an XOR gate applied to the output of all the parents.

Input Authentication For each input wire of the circuit, A solders onto these a number of wire authenticators. These authenticators ensure that the input keys B receives are in fact valid, since otherwise these authenticators will reject them.

Input/Output Verification Finally A uses $\mathcal{F}_{\mathsf{HCOM}}$ and the $2s$ extra commitments to efficiently and securely leak the least significant bits of the 0-key of each of B's input and output wires. This is done by A simply sending the bits, and then B challenging her to open to linear combinations of these. The $2s$ commitments are used as blinding values and are thus "sacrificed" in this process.

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 $\mathcal{G}$. The verification algorithm uses the previously mentioned least significant bits of the 0-keys to determine if a key corresponds to a specific bit. This enables B to verify his received input and decode his designated output. 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.

6.1 Protocol Details

In this section we present in detail our interactive garbling scheme $\mathsf{IGarb} = (\mathsf{IGb}, \mathsf{IEn}, \mathsf{IDe}, \mathsf{IEv}, \mathsf{Iev}, \mathsf{IVe})$. The dominant work of the scheme is performed in the garbling protocol IGb which is presented in its entirety in Fig. 10, Fig. 11, and Fig. 12. In Fig. 13 the remaining algorithms are presented.

[3] Checking all possible input combinations would reveal Δ to B and thus break the privacy of the protocol. This is so since if B learns both X_j^0 and X_j^1 for any wire j, he also learns $\Delta = X_j^0 \oplus X_j^1$. He could then use this knowledge to evaluate gates on multiple inputs, and hence learn *e.g.* A's input.

Structure of a fault tolerent garbled circuit

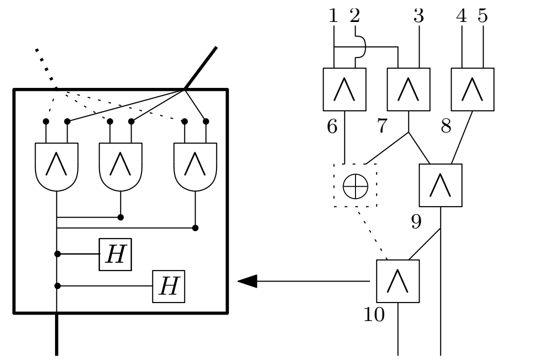

Fig. 9. Illustration of the wirings of a fault tolerent garbled circuit. The right hand side of the image shows a garbled circuit consisting of 5 garbled AND buckets where one bucket has its left input being the XOR of the output of two earlier buckets. The left hand side shows an AND bucket with 3 garbled 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.

Setup(pp):

1. On input $(f=(n_{\mathsf{A}},n_{\mathsf{B}},m_{\mathsf{A}},m_{\mathsf{B}},q,\mathsf{lp},\mathsf{rp}),k,s,p_g,p_a,\beta,\alpha,\lambda)\leftarrow\mathsf{pp}$, let $Q=q\beta\frac{1}{1-p_g-\epsilon_g}$, $A=(q\alpha+n\lambda)\frac{1}{1-p_a-\epsilon_a}$. For all $g\in[Q],j\in[A]$ and $l\in[2s]$ denote by l_g,r_g,o_g,a_j,b_l and τ unique identifiers that both parties agree on. A and B also initialize the functionality $\mathcal{F}_{\mathsf{HCOM}}$ by sending $(\mathsf{init},\mathsf{sid},\mathsf{A},\mathsf{B},\kappa)$ to it, where κ is the token length of $\mathcal{G}$. In future interaction with $\mathcal{F}_{\mathsf{HCOM}}$ we omit the party identifiers for ease of notation.
2. B samples $C_g\subset[Q]$ and $C_a\subset[A]$ where each element is included with probability p_g and p_a, respectively. Let $E_g=[Q]\setminus C_g$ and $E_a=[A]\setminus C_a$. If $|E_g|<q\beta$ or $|E_a|<q\alpha+n\lambda$, B outputs $\perp$ and aborts.
3. Next, B samples $\mathsf{Bof}\in_R\mathcal{B}(E_g)$ and $\mathsf{AWof}\in_R\mathcal{W}(E_a)$ and for each $g\in C_g$ and $j\in C_a$, he samples $\eta_g,\rho_g,\sigma_j\in_R\{0,1\}$. He then sends $\left(\mathsf{commit},\mathsf{sid},1,(\eta_g,\rho_g,o_g)_{g\in C_g},(\sigma_j,a_j)_{j\in C_a}\right)$ and $(\mathsf{commit},\mathsf{sid},2,(\mathsf{Bof},\mathsf{AWof}))$ to $\mathcal{F}_{\mathsf{COM}}$.

Garble:

1. A lets f' denote a function description for a circuit with Q parallel AND gates in a single input layer. She then runs $(F',e',d')\leftarrow\mathsf{Gb}(1^k,f')$. Parse $(\gamma,\delta_{2Q+1},\delta_{2Q+2},\ldots,\delta_{3Q})\leftarrow F$. For all $g\in[Q]$ we associate to gate δ_{2Q+g} the identifier o_g and to the left and right input wire l_g and r_g, respectively. Also we associate the identifier τ with the global difference Δ. For convenience we write (L_g^0,R_g^0,O_g^0) to denote the left, right and output 0-key of gate δ_{o_g}.
2. For all $j\in[A]$ and $l\in[2s]$, A samples random values $A_j,B_l\in_R\{0,1\}^\kappa$ and we again associate to these values the identifiers a_j and b_l, respectively. For all $g\in[Q],j\in[A]$ and $l\in[2s]$ she then sends
$$\left(\mathsf{commit},\mathsf{sid},(l_g,L_g^0),(r_g,R_g^0),(o_g,O_g^0),(a_j,A_j),(b_l,B_l),(\tau,\Delta)\right)$$
to $\mathcal{F}_{\mathsf{HCOM}}$ which sends a receipt of this to B.
3. For all $j\in[A]$, A computes $H_{a_j}\leftarrow\mathsf{Aut}(A_j,A_j\oplus\Delta)$. Then for all $g\in[Q]$ and all $j\in[A]$, A sends $(\delta_{o_g},\gamma_{o_g})$ and H_{a_j} to B.
4. After receiving the garbled gates and wire authenticators, B sends $(\mathsf{open},\mathsf{sid},1)$ to $\mathcal{F}_{\mathsf{COM}}$, which sends $\left(\mathsf{opened},\mathsf{sid},1,(\eta_g,\rho_g,o_g)_{g\in C_g},(\sigma_j,a_j)_{j\in C_a}\right)$ to A. For all $g\in C_g$ and $j\in C_a$, she sends
$$(\mathsf{open},\mathsf{sid},(\{l_g\}\cup\eta_g\cdot\{\tau\}),(\{r_g\}\cup\rho_g\cdot\{\tau\}),(\{o_g\}\cup(\eta_g\wedge\rho_g)\cdot\{\tau\}),(\{a_j\}\cup\sigma_j\cdot\{\tau\}))^a$$
to $\mathcal{F}_{\mathsf{HCOM}}$ which in turn sends $L_g^{\eta_g},R_g^{\rho_g},O_g^{\eta_g\wedge\rho_g}$ and $A_j^{\sigma_j}$ to B.
5. For all $g\in C_g$ and $j\in C_a$, B checks that $\mathsf{GEv}(\delta_{o_g},L_g^{\eta_g},R_g^{\rho_g},\gamma_{o_g})=O_g^{\eta_g\wedge\rho_g}$ and $\mathsf{Ver}(H_{a_j},A_j^{\sigma_j})=\top$. If any of the checks fail he outputs $\perp$ and aborts.

[a] Here the value of *e.g.* η_g decides if the index τ is included in the opening or not.

Fig. 10. Interactive garbling protocol IGb in the $(\mathcal{F}_{\mathsf{HCOM}},\mathcal{F}_{\mathsf{COM}})$-hybrid model – part 1.

Parameters and Replication. In the following A will play the role of C and B will play the role of E. We will let $\ell_g = \frac{1}{1-p_g-\epsilon_g}$ be the replication factor of gates, where p_g is the expected fraction of gates we sacrifice in the cut-and-choose step and ϵ_g is the fraction of extra gates we garble to ensure the actual number of remaining gates is not lower than expected. Therefore if one wishes to end up with T garble gates after the cut-and-choose step, one needs to garbled $T\ell_g$ in total. In addition we also need to consider the bucket size needed for the protocol which we denote by β. For a circuit $f = (n_\mathsf{A}, n_\mathsf{B}, m_\mathsf{A}, m_\mathsf{B}, q, \mathsf{lp}, \mathsf{rp})$ let $w = n+q$ and therefore A needs to garble $Q = q\beta\ell_g$ AND gates in total.

Solder:

1. B sends $(\mathsf{open}, \mathsf{sid}, 2)$ to $\mathcal{F}_{\mathsf{COM}}$ which in turn sends $(\mathsf{open}, \mathsf{sid}, 2, (\mathsf{Bof}, \mathsf{AWof}))$ to A which means for all $g \in \mathsf{Gates}$ she learns Bu_g and thus also the set $\mathsf{HeadGates}$, and for all $j \in \mathsf{Wires}$ she also learns Au_j.
2. For all $h \in \mathsf{HeadGates}$, both parties let $h_t = \mathsf{Bof}(h)$ denote the "topological" index of h and store this information as $T = \{(h, h_t)\}_{h \in \mathsf{HeadGates}}$. Also for convenience we define
$$\mathsf{OUT} = \{h \in \mathsf{HeadGates} \mid h_t \in \mathsf{Outputs}\}$$
$$\mathsf{LINP} = \{h \in \mathsf{HeadGates} \mid \mathsf{lp}(h_t) \in \mathsf{Inputs}\}, \quad \mathsf{RINP} = \{h \in \mathsf{HeadGates} \mid \mathsf{rp}(h_t) \in \mathsf{Inputs}\}$$
3. **Bucket Soldering:** For all $h \in \mathsf{HeadGates}$, all $g \in \mathsf{Bu}_{h_t}$ where $g \neq h$ and all $j \in \mathsf{Au}_{h_t}$, A sends $(\mathsf{open}, \mathsf{sid}, (\{l_g, l_h\}), (\{r_g, r_h\}), (\{o_g, o_h\}), (\{a_j, o_h\}))$ to $\mathcal{F}_{\mathsf{HCOM}}$.
4. **Topological Soldering:** For all $h \in \mathsf{HeadGates}$, let
$$\mathsf{LP}_h = \left\{\overline{h} \in \mathsf{HeadGates} \mid \overline{h}_t \in \mathsf{lp}(h_t)\right\}, \quad \mathsf{RP}_h = \left\{\overline{h} \in \mathsf{HeadGates} \mid \overline{h}_t \in \mathsf{rp}(h_t)\right\}.$$
Let $L = \{l_h\} \cup \{o_l\}_{l \in \mathsf{LP}_h}$ and $R = \{r_h\} \cup \{o_r\}_{r \in \mathsf{RP}_h}$. If $\mathbb{1} \in \mathsf{lp}(h_t)$ let $L := L \cup \{\tau\}$ and if $\mathbb{1} \in \mathsf{rp}(h_t)$ let $R := R \cup \{\tau\}$. A then sends $(\mathsf{open}, \mathsf{sid}, L, R)$ to $\mathcal{F}_{\mathsf{HCOM}}$.
5. **Input Authentication:** For all $h^l \in \mathsf{LINP}$, all $h^r \in \mathsf{RINP}$, all $a_l \in \mathsf{Au}_{\mathsf{lp}(h^l_t)}$ and all $a_r \in \mathsf{Au}_{\mathsf{rp}(h^r_t)}$, A sends $(\mathsf{open}, \mathsf{sid}, (\{a_l, l_{h^l}\}), (\{a_r, r_{h^r}\}))$ to $\mathcal{F}_{\mathsf{HCOM}}$.
6. B lets $\widetilde{S}^L_g, \widetilde{S}^R_g, \widetilde{S}^O_g$ and $\widetilde{S}^h_{a_j}$ be the received bucket solderings, $\widetilde{S}^L_h$ and $\widetilde{S}^R_h$ be the received topological solderings and $\widetilde{S}^h_{a_l}$ and $\widetilde{S}^h_{a_r}$ be the received input authentication solderings from $\mathcal{F}_{\mathsf{HCOM}}$.
7. For all $h \in \mathsf{HeadGates}$, all $g \in \mathsf{Bu}_{h_t}$ where $g \neq h$, all $j \in \mathsf{Au}_{h_t}$, all $h_l \in \mathsf{LINP}$, all $h_r \in \mathsf{RINP}$, all $a_l \in \mathsf{Au}_{\mathsf{lp}(h_l)}$ and all $a_r \in \mathsf{Au}_{\mathsf{rp}(h_r)}$, B computes
$$\widetilde{\gamma}_{o_g} \leftarrow \mathsf{SGb}\left(\gamma_{o_g}, \widetilde{S}^L_g, \widetilde{S}^R_g, \widetilde{S}^O_g\right), \quad \widetilde{\gamma}_{o_h} \leftarrow \mathsf{SGb}\left(\gamma_{o_h}, \widetilde{S}^L_h, \widetilde{S}^R_h, 0^\kappa\right)$$
$$\widetilde{H}_{a_j} \leftarrow \mathsf{SAut}\left(H_{a_j}, \widetilde{S}^h_{a_j}\right), \quad \widetilde{H}_{a_l} \leftarrow \mathsf{SAut}\left(H_{a_l}, \widetilde{S}^h_{a_l}\right), \quad \widetilde{H}_{a_r} \leftarrow \mathsf{SAut}\left(H_{a_r}, \widetilde{S}^h_{a_r}\right)$$

Fig. 11. Interactive garbling protocol IGb in the $(\mathcal{F}_{\mathsf{HCOM}}, \mathcal{F}_{\mathsf{COM}})$-hybrid model – part 2.

Each of these Q gates requires 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 denote by α the number of wire authenticators used for each bucket of the circuit and by λ the number of wire authenticators used for each input wire. 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 $\ell_a = \frac{1}{1-p_a-\epsilon_a}$ be the replication factor of the wire authenticators, where p_a is the expected fraction we check and ϵ_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 $A = (q\alpha + n\lambda)\ell_a$ wire authenticators. Recall that in the protocol, A will leak to B the least significant bits of each 0-key associated to his input and output wires. In order to verify the validity of these leaked bits, B will challenge A with a consistency check that ensures she sent the correct bits. This check requires sacrificing $2s$ committed random values to guarantee that no more than the least significant bits are leaked. In summary we need to produce commitments to $W = 3Q + A + 1 + 2s$ wires as we also need a commitment to the global difference Δ.

For all $g \in [Q], j \in [A]$ and $l \in [2s]$ we associate the unique identifiers l_g, r_g, o_g, a_j, b_l and τ unique identifiers that both parties agree on. For a gate δ_{o_g} we associate l_g, r_g, o_g to be the indices of the left, right and output 0-keys. a_j is the indices of the authenticated wires, b_l is the indices of the $2s$ previously mentioned random commitments and τ is the index of Δ. In the protocol, B will sample a subset $C_g \subset [Q]$ where each of the gates in $[Q]$ is included with probability p_g. These gates will be the ones sacrificed during the cut-and-choose step and we let $E_g = [Q] \setminus C_g$. To illustrate why the extra fraction ϵ_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 ϵ_g, which means we produce a little extra, to ensure 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 3. 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 f. In an analogous manner B will also sample $C_a \subset [A]$ where each wire is included in C_a with probability p_a and we let $E_a = [A] \setminus C_a$. For the exact same reason as above we produce a little extra, decided by ϵ_a, to ensure we have at least $q\alpha + n\lambda$ wire authenticators left after the cut-and-choose step.

VerLeak:

1. Let $\mathsf{BLEAK} = \{n_\mathsf{A}+1, n_\mathsf{A}+2, \ldots, n\} \cup \{w - m_\mathsf{B}+1, w-m_\mathsf{B}+2, \ldots, w\}$. Recall that $T = \{(h, h_t)\}_{h \in \mathsf{HeadGates}}$ determined which topological gate (of f) h is the head gate of. Then define the sets
$$\mathsf{O} = \{h \in \mathsf{OUT} \mid h_t \in \mathsf{BLEAK}\}$$
$$\mathsf{L} = \{h \in \mathsf{LINP} \mid \mathsf{lp}(h_t) \in \mathsf{BLEAK}\}, \quad \mathsf{R} = \{h \in \mathsf{RINP} \mid \mathsf{rp}(h_t) \in \mathsf{BLEAK}\} .$$
For all $v \in [s]$ and all $h_l \in \mathsf{L}$, $h_r \in \mathsf{R}$ and $h_o \in \mathsf{O}$, A sends $\left(p_{b_v}, p_{b_{s+v}}, p_{l_{h_l}}, p_{r_{h_r}}, p_{o_{h_o}}\right) = \left(\mathsf{lsb}(B_v), \mathsf{lsb}(B_{s+v}), \mathsf{lsb}(L^0_{h_l}), \mathsf{lsb}(R^0_{h_r}), \mathsf{lsb}(O^0_{h_o})\right)$ to B where b_l are the identifiers defined in **Garble** for $l \in [2s]$.
2. After receiving the above bits, B samples four uniformly random binary matrices, $\boldsymbol{V}^b \in_R \{0,1\}^{(s+1)\times s}, \boldsymbol{V}^L \in_R \{0,1\}^{|\mathsf{L}|\times s}, \boldsymbol{V}^R \in_R \{0,1\}^{|\mathsf{R}|\times s}, \boldsymbol{V}^O \in_R \{0,1\}^{|\mathsf{O}|\times s}$ and sends these to A.
3. Recall that τ is the identifier defined for the global difference Δ. Then for all $v \in [s]$ Alice lets
$$D^0_v = \bigcup_{u \in [s]} \left(\boldsymbol{V}^b_{u,v} \cdot \{b_u\}\right) \cup \boldsymbol{V}^b_{s+1,v} \cdot \{\tau\} \cup \{b_{s+v}\}$$
$$D^1_v = \bigcup_{l \in [|\mathsf{L}|], r \in [|\mathsf{R}|], o \in [|\mathsf{O}|]} \left(\boldsymbol{V}^L_{l,v} \cdot \{l_{h_l}\} \cup \boldsymbol{V}^R_{r,v} \cdot \{r_{h_r}\} \cup \boldsymbol{V}^O_{o,v} \cdot \{o_{h_o}\}\right) \cup \{b_v\}$$
and sends $\left(\mathsf{open}, \mathsf{sid}, D^0_v, D^1_v\right)$ to $\mathcal{F}_\mathsf{HCOM}$.
4. Upon receiving the values $\widetilde{S}_{D^0_v}$ and $\widetilde{S}_{D^1_v}$ from $\mathcal{F}_\mathsf{HCOM}$, B lets $p_\tau = 1$ and for all $v \in [s]$ verifies that
$$\mathsf{lsb}(\widetilde{S}_{D^0_v}) = \bigoplus_{j \in D^0_v} p_j, \quad \mathsf{lsb}(\widetilde{S}_{D^1_v}) = \bigoplus_{i \in D^1_v} p_i .$$
If any of the s checks fail, B outputs $\perp$ and aborts. Else B defines $v = \left(p_{l_{h_l}}, p_{r_{h_r}}, p_{o_{h_o}}\right)_{h_l \in \mathsf{L}, h_r \in \mathsf{R}, h_o \in \mathsf{O}}$ sorted in ascending typological order.

Output:

1. Recall that for a gate h we let l_h and r_h denote the identifiers of the left and right input wire, respectively. Then for all $h^l \in \mathsf{LINP}$, all $h^r \in \mathsf{RINP}$ and all $i \in \mathsf{Inputs}$, if $\mathsf{lp}(h^l_t) = i$ then A lets $e_i = e'_{l_{h^l}}$ or if $\mathsf{rp}(h^r_t) = i$ she lets $e_i = e'_{r_{h^r}}$.[a] She then defines $e = (e_1, e_2, \ldots, e_n)$.
2. For all $h \in \mathsf{OUT}$ and all $o \in [m]$ where $h_t = o$, A lets $d_o = d'_h$. She then defines $d = (d_1, d_2, \ldots, d_m)$.
3. A and B finally define
$$F = \left(f, T, \mathsf{HeadGates}, \left\{\left\{\left(\delta_{o_g}, \widetilde{\gamma}_{o_g}\right)\right\}_{g \in \mathsf{Bu}_{h_t}}\right\}_{h_t \in \mathsf{Gates}}, \left\{\left\{\widetilde{H}_{a_i}\right\}_{i \in \mathsf{Au}_j}\right\}_{j \in \mathsf{Wires}}\right)$$
to be the produced garbled circuit. A and B define their output to be (F, e, d) and (F, v), respectively.

[a] Recall that we only consider circuits f where input wires have fan-out 1. Hence these assignments are unique and well defined.

Fig. 12. Interactive garbling protocol IGb in the $\mathcal{F}_\mathsf{HCOM}$-hybrid model – part 3.

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 ϵ_g and ϵ_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 approach to calculate the value of ϵ_g and ϵ_a is captured in Lemma 3.

Lemma 3 (Tail Bounds). *Let $R_g = |E_g|$ and $R_a = |E_a|$ denote the random variables representing the number of remaining gates and wire authenticators after the cut-and-choose steps of* IGb. *Then*

$$\Pr[R_g \le q\beta] \le e^{-2\epsilon_g^2 Q} \quad \textit{and}$$
$$\Pr[R_a \le q\alpha + n\lambda] \le e^{-2\epsilon_a^2 A}$$

where $Q = q\beta \cdot \frac{1}{1-p_g-\epsilon_g}$ and $A = (q\alpha + n\lambda) \cdot \frac{1}{1-p_a-\epsilon_a}$.

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

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

By the exact same reason we see that

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

which proves the statement. ☐

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 f, one garbled gate is selected as representing this gate. We call this special garbled gate the head gate. A bucket is then constructed by soldering the wires of $\beta-1$ randomly selected gates onto the wires of the head gate.

To be more precise, once B has decided on the set E_g he lets $\mathcal{B}(E_g)$ be the family of all surjective β-to-1 functions from E_g to Gates.[4] For any function $\mathsf{Bof} \in \mathcal{B}(E_g)$ we let $\overline{E}_g \subseteq E_g$ denote the domain of the function, notice $|\overline{E}_g| = q\beta$. Then for all $g \in \mathsf{Gates}$ we define the set $\mathsf{Bu}_g = \{g' \in \overline{E}_g \mid \mathsf{Bof}(g') = g\}$ and let the head gate of each bucket be the gate h such that for all $g \in \mathsf{Bu}_i : o_h \le o_{g'}$, *i.e.*, h is the gate with lowest lexicographical index in Bu_i. For convenience we let $\mathsf{HeadGates}$ be the set of these head gate indices. Finally we assume that given Bof, it is easy to identify the domain of the function, meaning that $|\overline{E}_g|$ is assumed to be directly identified from the description of Bof.

Analogously we also need to specify how the wire authenticators are to be combined with the buckets. Again when B has determined E_a he lets $\mathcal{W}(E_a)$ be the family of all surjective functions from E_a to Wires. We furthermore require that for the images of the function in Inputs the functions are λ-to-1 and for the remaining images in Gates the functions are α-to-1. As in the above for a function $\mathsf{AWof} \in \mathcal{W}(E_a)$ we let $\overline{E}_a \subseteq E_a$ denote the domain of the function and notice $|\overline{E}_a| = q\alpha + n\lambda$. Again, the elements of E_a that AWof is undefined will simply be discarded. For all $j \in \mathsf{Wires}$ we define the set $\mathsf{Au}_j = \{a \in \overline{E}_a \mid \mathsf{AWof}(a) = j\}$. This means that for all $j \in \mathsf{Inputs} : |\mathsf{Au}_j| = \lambda$ and for all $j' \in \mathsf{Gates} : |\mathsf{Au}_{j'}| = \alpha$. Also in this case we assume that the domain $\overline{E}_a$ is efficiently determined from AWof.

[4] As E_g contains more elements than $q\beta$ with high probability there may be some elements of E_g that the functions are not defined for. These are simply left unused by our protocol.

Algorithms for the interactive garbling scheme IGarb

In the following let $\mathsf{GateScore}(O, L, R, G)$ be a function that returns the number of gates $\delta_i \in G$ where $\mathsf{SEv}(\delta_i,L,R,\widetilde{\gamma}_i)=O$. Likewise let $\mathsf{AuthScore}(X,H)$ be a function that returns the number of authenticators $H_i \in H$ where $\mathsf{Ver}(H_i,X)=\top$.

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

1. Parse F as $\left(f,T,\mathsf{HeadGates},\left\{\left\{\left(\delta_{o_g},\widetilde{\gamma}_{o_g}\right)\right\}_{g\in \mathsf{Bu}_{h_t}}\right\}_{h_t\in \mathsf{Gates}},\left\{\left\{\widetilde{H}_{a_i}\right\}_{i\in \mathsf{Au}_j}\right\}_{j\in \mathsf{Wires}}\right)$.
2. Parse $(X_1,\ldots,X_n)\leftarrow X$, $(n_{\mathsf{A}},n_{\mathsf{B}},m_{\mathsf{A}},m_{\mathsf{B}},q,\mathsf{lp},\mathsf{rp})\leftarrow f$ and set $w=n+q$.
3. For all $i\in[n]$, check that $\mathsf{AuthScore}\left(X_i,\left\{\widetilde{H}_{a_l}\right\}_{l\in \mathsf{Au}_i}\right)>\lambda/2$. If any of these checks fail output $\bot$.
4. Parse $\{(h,h_t)\}_{h\in \mathsf{HeadGates}}\leftarrow T$.
   For all $h_t\in \mathsf{Gates}$, let $L_h=\bigoplus_{l\in \mathsf{lp}(h_t)}X_l$ and $R_h=\bigoplus_{r\in \mathsf{rp}(h_t)}X_r$ and do:
   (a) For all $g\in \mathsf{Bu}_{h_t}$, compute $O_g\leftarrow \mathsf{SEv}(\delta_{o_g},L_h,R_h,\widetilde{\gamma}_{o_g})$.
   (b) Let $\mathsf{Cand}=\left\{O_g\right\}_{g\in \mathsf{Bu}_{h_t}}$. If $|\mathsf{Cand}|=1$, let $X_{h_t}=O_h$. Else let
   $$\mathsf{MAJ}=\left\{O_g\in \mathsf{Cand}\,|\ \mathsf{GateScore}\left(O_g,L_h,R_h,\left\{\left(\delta_{o_g},\widetilde{\gamma}_{o_g}\right)\right\}_{g\in \mathsf{Bu}_{h_t}}\right)+\mathsf{AuthScore}\left(O_g,\left\{\widetilde{H}_{a_i}\right\}_{i\in \mathsf{Au}_{h_t}}\right)>(\beta+\alpha)/2\right\}.$$
   If $|\mathsf{MAJ}|\neq 1$ output $\bot$. Else set X_{h_t} to be the singleton output key in MAJ.
5. Output $Z=(X_{w-m+1},X_{w-m+2},\ldots,X_w)$.

$\mathsf{IEn}(e_i,x_i)\to \underline{X_i}/\bot$:

1. Output $\overline{\mathsf{En}}(e_i,x_i)$.

$\mathsf{IDe}(d_j,Z_j)\to \underline{z_j}/\bot$:

1. Output $\overline{\mathsf{De}}(d_j,Z_j)$.

$\mathsf{IVe}(v_i,X_i,b_i)\to \top/\bot$:

1. If $\mathsf{lsb}(X_i)=v_i$ and $b_i=0$ output $\top$.
2. Else if $\mathsf{lsb}(X_i)=\overline{v_i}$ and $b_i=1$ output $\top$.
3. Else output $\bot$.

Fig. 13. Algorithms for the interactive garbling scheme IGarb.

VerLeak. As previously mentioned π_{IGCO} allows both parties to learn distinct output as part of the computation. In our interactive garbling scheme this feature is achieved using the VerLeak procedure described in Fig. 12. The outcome of this phase is that B learns the least significant bits (lsbs) of the 0-keys of the wires that he is allowed to decode, *i.e.* his designated input and output wires. Using these bits he can later verify if a given key is either a 0- or a 1-key for the wire in question. In addition he is also convinced that $\mathsf{lsb}(\Delta)=1$, as then the lsb of the 0-key is always different from the lsb of the 1-key. The technique for securely leaking the lsbs is inspired by the consistency check of [FJNT15]. In essence the step proceeds by A first sending the lsbs directly to B which afterwards challenges the validity of these proclaimed bits. This is done by challenging A to open random linear combinations of the 0-keys in question using $\mathcal{F}_{\mathsf{HCOM}}$ while B checks that these match the same linear combinations of the previously received lsbs. However as this would leak information about the entire 0-keys the linear combinations are blinded by a random value for which B only knows the lsb of.

In more detail, two things are verified as part of the phase VerLeak. Firstly the lsbs of the random values are leaked to B. This check also guarantees that $\mathsf{lsb}(\Delta)=1$ as Δ is included in a linear combination with probability $1/2$. This is captured in the v'th linear combination D_v^0 of Fig. 12 where $v\in[s]$. Second the lsbs of the 0-keys for B's input and output are checked in the v'th linear combination D_v^1 blinded by one of the previously mentioned random values. Since B only knows the lsb of these random values it is guaranteed that he does not learn anything besides the validity of the proclaimed lsbs. Following the analysis of [FJNT15] we have that the above checks ensure that A sends the correct lsbs of the 0-keys and $\mathsf{lsb}(\Delta)=1$ except with probability 2^{-s}.

6.2 Proof that IGarb is an Interactive Garbling Scheme

We now show that our interactive garbling scheme IGarb satisfies the security properties defined in Fig. 2 and Fig. 3 of Section 3.

For ease of presentation we show our scheme 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 singled out this case as we found it to perform well in terms of overall performance, relative to the security it provided.[5]

Before we continue recall the definition of a (2-)correlation robust hash function. This definition is taken almost verbatim from [IKNP03]:

Definition 2 (Correlation robustness). *A hash function $\mathcal{H}$ with κ-bit output is said to be* correlation robust *(denoted by the property* cor*) if for probabilistic polynomial time bounded adversary (in κ) denoted by $\mathcal{A}$ it holds that*

$$|\Pr[\mathcal{A}(X_1,...,X_m,\mathcal{H}(X_1\oplus\Delta),...,\mathcal{H}(X_m\oplus\Delta))=1]-\Pr[\mathcal{A}(U_1,...,U_{2m})=1]|\leq \text{negl}(\kappa) ,$$

where, $\Delta,X_1,...,X_m,U_1,...,U_m\in_R\{0,1\}^\kappa$ and $U_{m+1},...,U_{2m}\in_R\{0,1\}^{k'}$.

Lemma 4 (corr). *The scheme* IGarb *has the corr property.*

Proof. Consider the game $\text{Corr}^{\mathcal{A}}_{\mathsf{IGarb}}(1^\kappa)$ where an adversary $\mathcal{A}$ inputs (f,x) to $\text{Corr}_{\mathsf{IGarb}}$. We assume that $x\in\{0,1\}^{f.n}$ as else there is nothing to prove. It then runs $\mathsf{C}(1^\kappa,f)$ and $\mathsf{E}(1^\kappa,f)$ as specified by IGb.

Then by the correctness of the garbling scheme $\mathcal{G}$, correctness of $\mathcal{F}_{\mathsf{HCOM}}$, $\mathcal{F}_{\mathsf{COM}}$, and $\mathcal{F}_{\mathsf{OT}}$ and the fact that Bof and AWof were chosen correctly the encoded output Z decodes to the correct output as well. ☐

Lemma 5 (sec.ind.act). *If $\mathcal{G}=(\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})$ is a Key-Size Preserving Free-XOR Gate Garbling Scheme which is obl.ind and prv.ind secure and $\mathcal{H}$ is correlation robust then the scheme $\mathsf{IGarb}=(\mathsf{IGb},\mathsf{IEn},\mathsf{IDe},\mathsf{IEv},\mathsf{Iev},\mathsf{IVe})$ is sec.ind.act-secure in the ($\mathcal{F}_{\mathsf{HCOM}}$-$\mathcal{F}_{\mathsf{COM}}$)-hybrid model.*

Proof. Let $\text{OblInd}^{\mathcal{S},b_o}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$, respectively $\text{PrvInd}^{\mathcal{S},b_p}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ denote the security games $\text{OblInd}^{\mathcal{S}}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$, respectively $\text{PrvInd}^{\mathcal{S}}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ when the bits sampled by the games are b_o, respectively b_p. Similarly we let $\text{SecIndAct}^{\mathcal{B},b}_{\mathsf{IGarb}}(1^\kappa)$ denote the game $\text{SecIndAct}^{\mathcal{B}}_{\mathsf{IGarb}}(1^\kappa)$ when the bit sampled by the game is b. Since we assume that $\mathcal{G}$ is obl.ind- and prv.ind-secure it must hold that for any PPT $\mathcal{S}$ playing the games, we have $\text{OblInd}^{\mathcal{S},0}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)\overset{c}{\approx}\text{OblInd}^{\mathcal{S},1}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ and similarly $\text{PrvInd}^{\mathcal{S},0}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)\overset{c}{\approx}\text{PrvInd}^{\mathcal{S},1}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$. What we wish to prove is therefore that $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{IGarb}}(1^\kappa)\overset{c}{\approx}\text{SecIndAct}^{\mathcal{B},1}_{\mathsf{IGarb}}(1^\kappa)$ for any PPT $\mathcal{B}$.

Consider the following simulator $\mathcal{S}^b$ participating in the $\text{PrvInd}^{\mathcal{S}^b,b}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ game while simulating towards $\mathcal{B}$ the game $\text{SecIndAct}^{b}_{\mathsf{IGarb}}$. For ease of notation we let (f_0,f_1,x_0,x_1) denote the input $\mathcal{B}$ gives to $\mathcal{S}^b$ and let $\left(\hat{f}_0^{\mathcal{S}},\hat{f}_1^{\mathcal{S}},\hat{x}_0^{\mathcal{S}},\hat{x}_1^{\mathcal{S}}\right)$ denote the input $\mathcal{S}^b$ will give to $\text{PrvInd}^{\mathcal{S}^b,b}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$. First let $\mathcal{S}^b$ learn (f_0,f_1,x_0,x_1) from $\mathcal{B}$ at the beginning of the game. If $\{x_0,x_1\}\not\subseteq\{0,1\}^{f_0.n}$, $\mathbf{\Phi}_{\mathsf{xor}}(f_0)\neq\mathbf{\Phi}_{\mathsf{xor}}(f_1)$ or $x_{0,f_0.n_{\mathsf{A}}+i}\neq x_{1,f_1.n_{\mathsf{A}}+i}$ for $i\in[n_{\mathsf{B}}]$, then output $\perp$. Next compute $z_0\leftarrow\mathsf{ev}(f_0,x_0)$ and $z_1\leftarrow\mathsf{ev}(f_1,x_1)$. If $z_{0,j}\neq z_{1,j}$ for any $j\in[m_{\mathsf{A}}+1;m]$ then output $\perp$.

Now $\mathcal{S}^b$ runs the protocol IGb playing the role of an honest C, except it extracts the messages of all the commitments $\mathcal{B}$ makes to $\mathcal{F}_{\mathsf{COM}}$. Thus after the **Setup** phase of IGb, $\mathcal{S}^b$ 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, which is the set $\left\{\{(\eta_g,\rho_g,g)\}_{g\in C_g},\{(\sigma_j,j)\}_{j\in C_a},\mathsf{Bof},\mathsf{AWof}\right\}$. Now, instead of doing Step 1 of **Garble** $\mathcal{S}^b$ will use $\text{PrvInd}^{\mathcal{S}^b,b}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ to compute the garbled gates. It defines $\hat{f}_0^{\mathcal{S}}$ and $\hat{f}_1^{\mathcal{S}}$ as functions with only a single layer of AND gates. Next $\mathcal{S}^b$ evaluates which bit will be on each of the wires in the circuit representing f_0, respectively f_1 when evaluated on x_0, respectively x_1.

[5] However for some parameters this choice is not optimal, see Section 7.1 for details.

For each wire in f_0 that is an input wire to an AND gate or a circuit input wire find the indices of the garbled gates in the bucket representing this gate that has this wire as one of its inputs. There will be β such gates for each of these wires. These gates are uniquely defined by Bof, which is known to the simulator at this point. The simulator then sets the $2|\overline{E}_g|$ bits of $\hat{x}_0^{\mathcal{S}}$, respectively $\hat{x}_1^{\mathcal{S}}$, in accordance with the values expected on each wire when evaluating $f_0(x_0)$, respectively $f_1(x_1)$. However, for $\hat{f}_1^{\mathcal{S}}$ we make the change that when $z_{0,j} \neq z_{1,j}$ for $j \in [1;m_{\mathsf{A}}]$ we switch the AND gates in the bucket computing the j'th output bit with NAND gates. Notice that this is clearly possible since we assume only the topology and XOR gates are leaked on f_b. That is, we transform a function $f_1^{\mathcal{S}}$ with $\mathbf{\Phi}_{\mathsf{xor}}(f_1^{\mathcal{S}}) = \mathbf{\Phi}_{\mathsf{xor}}(f_1)$ to $\hat{f}_1^{\mathcal{S}}$.

Next, for each $g \in C_g$ the simulator sets $\hat{x}_0^{\mathcal{S}}[l_g] = \hat{x}_1^{\mathcal{S}}[l_g] = \eta_g$ and $\hat{x}_0^{\mathcal{S}}[r_g] = \hat{x}_1^{\mathcal{S}}[r_g] = \rho_g$. Notice that $|\hat{x}_0^{\mathcal{S}}| = |\hat{x}_1^{\mathcal{S}}| = 2Q$ since each gate has 2 bits input. For the rest of the entries in $\hat{x}_0^{\mathcal{S}}$ and $\hat{x}_1^{\mathcal{S}}$ it always chooses the 0-bit (these are the "slack" entries caused by the ϵ_g fraction and will just be discarded).

Then $\mathcal{S}^b$ sends $(\hat{f}_0^{\mathcal{S}}, \hat{f}_1^{\mathcal{S}}, \hat{x}_0^{\mathcal{S}}, \hat{x}_1^{\mathcal{S}})$ to $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^b,b}$ and receives back $(\hat{F}_b^{\mathcal{S}}, \hat{X}_b^{\mathcal{S}}, d_b^{\mathcal{S}})$ where b is the challenge bit picked by $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^b,b}$. Notice that $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^b,b}$ does not output $\perp$ since we have that $\hat{f}_0^{\mathcal{S}}(\hat{x}_0^{\mathcal{S}}) = \hat{f}_1^{\mathcal{S}}(\hat{x}_1^{\mathcal{S}})$.

The simulator then runs $\mathsf{Ev}(\hat{F}_b^{\mathcal{S}}, \hat{X}_b^{\mathcal{S}})$ 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 Bof function). Now $\mathcal{S}^b$ runs the rest of IGb with $\mathcal{B}$ as an honest C would, simulating commitment calls to $\mathcal{F}_{\mathsf{HCOM}}$. Furthermore, in **Garble**, Step 3 and 4 are replaced with the following:

3. $\mathcal{S}^b$ "extracts" the garbled gates from $\hat{F}_b^{\mathcal{S}}$. That is, it parses $(\gamma, \delta_{2Q+1}, \ldots, \delta_{3Q}) \leftarrow \hat{F}_b^{\mathcal{S}}$. For $g \in [Q]$ it then defines the gates $\{\gamma'_g\} = \{(\gamma_g, 0^\kappa, 0^\kappa, 0^\kappa)\}$. For $j \in [A]$, $\mathcal{S}^b$ picks two values uniformly at random $A_j, H_j^1 \in \{0,1\}^\kappa$ and computes $H_j^0 \leftarrow \mathcal{H}(A_j)$. It then defines $H_{a_j} = (H_j^0, H_j^1, 0^\kappa)$ if $H_j^0 \leq H_j^1$ and $H_{a_j} = (H_j^1, H_j^0, 0^\kappa)$ otherwise. Then for $g \in [Q]$ and $j \in [A]$ the simulator sends $(\delta_{o_g}, \gamma_{o_g})$ and H_{a_j} to $\mathcal{B}$.
4. After $\mathcal{B}$ sends $(\mathsf{open}, \mathsf{sid}, 1)$ to $\mathcal{F}_{\mathsf{COM}}$, for all $g \in C_g$ and $j \in C_a$, $\mathcal{S}^b$ simulates openings from $\mathcal{F}_{\mathsf{HCOM}}$ by sending $X_g^{\eta_g}, X_g^{\rho_g}, A_j$ to $\mathcal{B}$. These are the keys the simulator received from the $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^{\mathcal{B}},b}$ game, the computation of $\mathsf{Ev}(\hat{F}_b^{\mathcal{S}}, \hat{X}_b^{\mathcal{S}})$ and the random values A_j for $j \in [A]$ it picked in Step 3 for the wire authenticators.

Similarly to Step 4 above, for the solderings $\mathcal{S}^b$ simulates the $\mathcal{F}_{\mathsf{HCOM}}$ functionality by using the values it got from the $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^b,b}$ game, the computation $\mathsf{Ev}(\hat{F}_b^{\mathcal{S}}, \hat{X}_b^{\mathcal{S}})$ and the random values A_j for $j \in [A]$ it picked in the new **Garble** Step 3 for the authenticators. That is, for any given gate wire g the simulator knows exactly one key, either key 0 or key 1, depending on what $\mathcal{B}$ should learn during evaluation of $f_b(x_b)$. If the keys of two wires to be soldered together have semantic value 0, then this happens as in $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},b}$. If they instead have semantic value 1, then it is also as in $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},b}$ since the 1-key of a wire will be the 0-key XOR'ed with $\varDelta$. So even though the simulator does not know the $\varDelta$ used in the $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^b,b}$ game the solderings will be done correctly. Finally, notice that we will never have to solder together two gate wires where we know one 0-key and one 1-key. For the authenticators we look at the wire it will be soldered to. For authenticator j soldered to wire g we let the soldering be the key we know on wire g XOR'ed with A_j.

In **VerLeak** Step 1, for $l \in [2s]$ the simulator picks $B_l \in_R \{0,1\}^\kappa$. It then sends the least significant bit of the keys it knows as it is supposed to in $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},b}$. However, if the key it knows carries the semantic value 1, the simulator flips the lsb bit before sending it to $\mathcal{B}$.

In Step 3 the simulator emulates $\mathcal{F}_{\mathsf{HCOM}}$ by sending the values D_v^0, D_v^1 to $\mathcal{B}$ for $v \in [s]$. Here D_v^0 and D_v^1 are computed as defined by the matrices $\mathbf{V}^b, \mathbf{V}^L, \mathbf{V}^R, \mathbf{V}^O$ given by $\mathcal{B}$ using the single key per wire the simulator knows (no matter if it is a 0- or a 1-key). However, if the index τ is included in a linear combination, the simulator flips the least significant bit of the computed value before sending it to B. The rest of the protocol $\mathcal{S}^b$ carries out like an honest C would.

We now show that anything sent in the simulation above is indistinguishable from what is sent in $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},b}$ and thus conclude that any advantage in winning the game $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},b}$ translates directly into an advantage in winning the underlaying $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^{b_p},b_p}$ or $\mathrm{OblInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^{b_o},b_o}$ game. We do this through a hybrid argument: First define the hybrids induced by $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}^{b_p},b_p}(1^\kappa)$ in the simulation above as $\mathsf{H}^{1,b_p}(1^\kappa)$. Now define the pair of hybrids $\mathsf{H}^{2,b_p}(1^\kappa)$ which are exactly like $\mathsf{H}^{1,b_p}(1^\kappa)$, but where the simulator cheats and looks into the game $\mathrm{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{b_p}$ to learn $\varDelta$. It then sets $H_j^1 = \mathcal{H}(A_j \oplus \varDelta)$ for $j \in [A]$, which it uses in **Garble** Step 3 to construct the authenticators exactly like in $\text{SecIndAct}^{\mathcal{B},b_p}_{\mathsf{lGarb}}$. Furthermore, for the solderings of authenticator j to wire g we let the soldering be the key we know on wire g XOR'ed with A_j if its semantic value is 0, and XOR'ed with $A_j \oplus \Delta$ otherwise.

We now argue that $\mathsf{H}^{2,b_p}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,b_p}(1^\kappa)$ for $b_p \in \{0,1\}$ by the assumption that $\mathcal{H}$ is correlation robust. We notice that the only difference between hybrid H^{2,b_p} and H^{1,b_p} is the way we construct the authenticators. To see that the hybrids are indistinguishable notice that in H^{1,b_p} all authenticators consists of the values $H^0_j = \mathcal{H}(A_j)$ and $H^1_j \in_R \{0,1\}^{k'}$ where $A_j \in_R \{0,1\}^\kappa$. In hybrid H^{2,b_p} on the other hand $H^0_j = \mathcal{H}(A_j)$ and $H^1_j = \mathcal{H}(A_j \oplus \Delta)$ where $A_j, \Delta \in_R \in \{0,1\}^\kappa$. Thus, distinguishing between H^{1,b_p} and H^{2,b_p} implies the ability to distinguish between the two cases in the correlation robustness definition. Finally see that there is one more difference between the hybrids: The soldering of authentications onto circuit wires. In H^{1,b_p} it is always done onto A_j and thus might contain Δ as a term. In H^{2,b_p} it is done with $A_j \oplus \Delta$ in case the semantic value of the key soldered with is 1 (in this case these soldering are exactly like in the real protocol). However, since A_j is uniformly random sampled then the soldering will also be uniformly random sampled, no matter if Δ is a part term of it. We therefore conclude that since $\mathcal{H}$ is correlation robust we have $\mathsf{H}^{2,b_p}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,b_p}(1^\kappa)$.

Next notice that $\mathsf{H}^{1,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,1}(1^\kappa)$ since the only variation is based on the bit picked in $\text{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$. However, we have assumed that $\mathcal{G}$ has the prv.ind property.

We now argue that $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,0}(1^\kappa)$:

1. In **Garble** in $\mathsf{H}^{2,0}$ notice that $\mathcal{S}^0$ picks the garbled gates using the $\text{PrvInd}^0_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ game, where the garbled circuit is constructed exactly the same way as in the $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$ game, that is, using function f_0 transformed into $\hat{f}^{\mathcal{S}}_0$, a circuit containing a single layer of AND gates. Furthermore, the openings to the commitments are exactly to values $\mathcal{B}$ would expect in accordance with the way they have been constructed in the $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$ game.
2. We already argued, during the description of the first hybrid, that the solderings sent for the garbled gates, in **Solder**, will be exactly like in $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$. Furthermore, this will also be the case for the authenticators since we have $A'_j = A_j \oplus \Delta$, where A_j is picked uniformly at random both in $\mathsf{H}^{2,0}$ and the $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$ game.
3. In **VerLeak** we first see that since in $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$ we have $\mathsf{lsb}(\Delta) = 1$ when we flip the bits of the keys in Step 1 in $\mathsf{H}^{2,0}$ if they represent 1-keys, then these bits will be distributed like in the real $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$ game because the keys are constructed from Gb in both cases. Now in Step 3 we notice that the check of the values $\mathsf{lsb}(D^0_v), \mathsf{lsb}(D^1_v)$ for $v \in [s]$ will also be distributed the same way since we flip the least significant bit of the keys with semantic value 1 in $\mathsf{H}^{2,0}$ so they will match what they are supposed to for 0-keys in $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$. In regards to the other bits of D^0_v and D^1_v we notice that they will always be one-time padded with B_{s+v}, respectively B_v which is uniformly random in both $\mathsf{H}^{2,0}$ and $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}$.

From the above discussion we conclude that $\text{SecIndAct}^{\mathcal{B},0}_{\mathsf{lGarb}}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,0}(1^\kappa)$.

Next notice that $\mathsf{H}^{2,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,1}(1^\kappa)$ since the only variation is based on the bit picked in $\text{PrvInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ and we have assumed that $\mathcal{G}$ has the prv.ind property.

Now to show that $\mathsf{H}^{2,1}(1^\kappa) \overset{c}{\approx} \text{SecIndAct}^{\mathcal{B},1}_{\mathsf{lGarb}}(1^\kappa)$ we introduce a new hybrid $\mathsf{H}^{3,b_o}(1^\kappa)$. The purpose of this hybrid is to change the NAND gates in $\hat{f}^{\mathcal{S}}_1$ from $\mathsf{H}^{2,1}$ and $\mathsf{H}^{1,1}$ back to AND gates. We do this using the $\text{OblInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ game. Hybrid H^{3,b_o} basically works as $\mathsf{H}^{2,1}(1^\kappa)$. However, when reaching Step 1 in **Garble** define $\hat{f}^3_1$ and $\hat{x}^3_1$ as $\hat{f}^{\mathcal{S}}_1$ and $\hat{x}^{\mathcal{S}}_1$ from $\mathsf{H}^{2,1}$, that is, based on f_1. However, we also let $\hat{f}^3_0$ be based on f_1 and compute the "^" version using the same method as in $\mathsf{H}^{2,0}$, thus *without* changing any AND gates to NAND gates. That is, $\hat{f}^3_0$ is a function with only a single layer of AND gates. We evaluate $f_1(x_1)$ to learn which bit will be on each of the wires in the circuit representing f_1 when evaluated x_1. For each wire in f_1 that is an input wire to an AND gate we find the indices of the garbled gates in the bucket representing this gate that has this wire as one of its inputs. The simulator then sets the $2|\overline{E}_g|$ bits of $\hat{x}_1$ in accordance with the values expected on each wire when evaluating $f_1(x_1)$. It sets the rest of the bits of $\hat{x}^3_0$ like in $\mathsf{H}^{2,1}$. Then give $(\hat{f}^3_0, \hat{f}^3_1, \hat{x}^3_0, \hat{x}^3_1)$ as input to the $\text{OblInd}^{b_o}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ game. Notice that in H^{2,b_p} the input is $(\hat{f}^{\mathcal{S}}_0, \hat{f}^{\mathcal{S}}_1 = \hat{f}^3_1, \hat{x}^{\mathcal{S}}_0, \hat{x}^{\mathcal{S}}_1 = \hat{x}^3_1)$ to the $\text{PrvInd}^{b_p}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ game. Furthermore, cheat and extract $\hat{d}_{b_o}$ from $\text{OblInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$. For the rest of the protocol it proceeds as hybrid $\mathsf{H}^{2,1}$.

Now notice that this means that setting $\hat{Z}_{b_o} \leftarrow \mathsf{IEv}(\hat{F}_{b_o}, \hat{X}_{b_o})$ in hybrid H^{3,b_o} we have that $\hat{z}_{b_o,j} \leftarrow \mathsf{IDe}(\hat{d}_{b_o,j}, \hat{Z}_{b_o,j})$ for $j \in [m]$. Furthermore, by the way we construct $\hat{f}_0^3$, $\hat{f}_1^3$ and $\hat{x}_0^3$, $\hat{x}_1^3$ in this hybrid we have that $\hat{z}_{0,j} = \hat{z}_{1,j}$ for $j \in [m_{\mathsf{A}}+1;m]$, however, it might be the case that $\hat{z}_{0,j} \neq \hat{z}_{1,j}$ for $j \in [m_{\mathsf{A}}]$.

Now see that it is clearly the case that $\mathsf{H}^{3,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,1}(1^\kappa)$ since we assume the obl.ind property and the result of the $\mathrm{OblInd}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}$ gate is the only point of variability in the two hybrids.

We now argue that $\mathsf{H}^{3,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,1}(1^\kappa)$. This follows somewhat trivially since in both hybrids the garbled circuits are constructed using Gb and evaluated on the same input. The rest of both the hybrids proceed similarly.

Finally we must argue that $\mathsf{H}^{3,0}(1^\kappa) \overset{c}{\approx} \mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}(1^\kappa)$. First see that both $\mathsf{H}^{3,0}$ and $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}$ are based on the same function f_1. Thus, hybrid $\mathsf{H}^{3,0}$ is in fact similar to hybrid $\mathsf{H}^{2,0}$ but where $\hat{f}_0^{\mathcal{S}} = \hat{f}_0^3$. This basically means that hybrid $\mathsf{H}^{3,0}$ is the same as $\mathsf{H}^{2,0}$, only using a different input function, that is, f_0 in $\mathsf{H}^{2,0}$ versus f_1 in $\mathsf{H}^{3,0}$. Thus $\mathsf{H}^{3,0}(1^\kappa) \overset{c}{\approx} \mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}(1^\kappa)$ follows from the same argument that $\mathsf{H}^{2,0}(1^\kappa) \overset{c}{\approx} \mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},0}(1^\kappa)$.

Now see that because efficient transformations maintain indistinguishability we get that $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},0}(1^\kappa) \overset{c}{\approx} \mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}(1^\kappa)$ from the following observation:

$$\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,0}(1^\kappa) \overset{c}{\approx} \mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}(1^\kappa)\ . \tag{1}$$

Here $\mathsf{H}^{2,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,0}(1^\kappa)$ and $\mathsf{H}^{2,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,1}(1^\kappa)$ follows from cor, $\mathsf{H}^{1,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{1,1}(1^\kappa)$ follows from prv.ind, $\mathsf{H}^{3,0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,1}(1^\kappa)$ follows from obl.ind, $\mathsf{H}^{2,1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,1}(1^\kappa)$ follows trivially by direct construction and the bulk of the above proof consists of show that $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,0}(1^\kappa)$, and by similar construction $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},1}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{3,0}(1^\kappa)$.

□

Lemma 6 (aut.act). *If $\mathcal{G} = (\mathsf{Gb},\mathsf{En},\mathsf{De},\mathsf{Ev},\mathsf{ev})$ is a Key-Size Preserving Free-XOR Gate Garbling Scheme which is* aut *secure and $\mathcal{H}$ is a correlation-robust hash function, then the scheme $\mathsf{IGarb} = (\mathsf{IGb},\mathsf{IEn},\mathsf{IDe},\mathsf{IEv},\mathsf{Iev},\mathsf{IVe})$ is aut.act-secure in the ($\mathcal{F}_{\mathsf{HCOM}}$-,$\mathcal{F}_{\mathsf{COM}}$)-hybrid model.*

Proof. Considering the game $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$ for the aut.act property we first argue that $\mathcal{B}$ cannot win the game by being malicious in IGb such that $z \neq \mathsf{Iev}(f,x)$ where $z = (z_1,z_2,\ldots,z_m)$ and $z_j \leftarrow \mathsf{IDe}(d_j,Z_j)$ for $j \in [m]$. We therefore conclude that he can only win the game by finding Z'_{A} such that there exists $j \in [m_{\mathsf{A}}]$ where $\mathsf{IDe}(d_j,Z'_{\mathsf{A},j}) \neq \perp$ with $Z'_{\mathsf{A},j} \neq Z_j$. We will then prove that if he can find such a Z'_{A} with non-negligible probability in κ then we can use him to win the $\mathrm{Aut}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}(1^\kappa)$ game with non-negligible probability, under the assumption that $\mathcal{H}$ is a correlation-robust hash function (in the following we assume $\mathbf{\Phi}_{\mathsf{xor}}$ to always be implied and omit the subscript). In particular we will construct a simulator $\mathcal{S}^{\mathcal{B}}$ that plays the $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$ game in Fig. 1, to construct the garbled gates used in IGb, then argue that whether or not a $\mathcal{B}$ playing the $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$ game is communicating with the simulator or the real game, what it learns will be computationally indistinguishable under the assumption that $\mathcal{H}$ is a correlation-robust hash function. The simulator then uses the output of $\mathcal{B}$ to win the $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$ game if $\mathcal{B}$ wins the $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$ game.

To first show that $\mathsf{IDe}(d_j,Z_j) \neq z_j$ for $(z_1,\ldots,z_m) \leftarrow \mathsf{Iev}(f,x)$ with $j \in [m]$ is not possible, notice that the only way $\mathcal{B}$ could cause this to happen is to act maliciously during the execution of IGb. However, see that $\mathcal{B}$ only gets to give the following input to this protocol:

1. The commitments, and subsequently openings, to the cut-and-choose challenges $\left(\{(\eta_g,\rho_g,o_g)\}_{g \in C_g},\ \{(\sigma_j,j)\}_{j \in C_a}\right)$ and the bucketing functions $(\mathsf{Bof},\mathsf{AWof})$ in **Setup**.
2. The random matrices used for determining the consistency checks of the **VerLeak** step.

Notice all the elements are randomly sampled and made only to protect B against a malicious A. In particular since the protocol does not abort then by the correctness of $\mathcal{F}_{\mathsf{COM}}$, it must mean that $\mathcal{B}$ committed (and later opened) to valid messages. In the second and third case the values are simply used for random checks and A terminates the protocol 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) to C it will always be the case that $z = \mathsf{Iev}(f,x)$ where $z = (z_1,z_2,...,z_m)$ and $z_j \leftarrow \mathsf{IDe}(d_j,\mathsf{IEv}(F,X)_j)$ for $j \in [m]$.

We now turn to the construction of $\mathcal{S}^{\mathcal{B}}$. Consider the simulator $\mathcal{S}^b$ for $b \in \{0,1\}$ defined in the proof of Lemma 5 which made use of the $\mathrm{PrvInd}_{\mathcal{G}}^{\mathcal{S}^b}$ game to simulate an execution of IGb. Our simulator $\mathcal{S}^{\mathcal{B}}$ will follow the same strategy, but instead of $\mathrm{PrvInd}_{\mathcal{G}}$ it interact with the game $\mathrm{Aut}_{\mathcal{G}}$. Since in the game $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}$ the adversary only provides a single function f we adapt the description of $\mathcal{S}^b$ to this setting and denote our modified simulator $\mathcal{S}^{\mathcal{B}}$. In short the simulator is simply adapted to the single case setting and constructs a single function $\hat{f}^{\mathcal{S}}$ (for sending to the $\mathrm{Aut}_{\mathcal{G}}$ game to get the garbled gates) and we construct it like $\mathcal{S}^b$ constructs $\hat{f}_0^{\mathcal{S}}$.

In more detail $\mathcal{S}^{\mathcal{B}}$ defines $\hat{f}^{\mathcal{S}}$ as a function with only a single layer of AND gates. It then evaluates $f(x)$ to learn which bit is expected to flow on each wire of f. For each wire in f that is an input wire to an AND gate or a circuit input wire find the indices of the garbled gates in the bucket representing this gate that has this wire as one of its inputs. There will be β such gates for each of these wires. These gates are uniquely defined by Bof, which is known to the simulator at this point. We denote by $\hat{x}^{\mathcal{S}}$ the input that is to be sent along with $\hat{f}^{\mathcal{S}}$ to $\mathrm{Aut}_{\mathcal{G}}$ by $\mathcal{S}^{\mathcal{B}}$. The simulator sets the $2|\overline{E}_g|$ bits of $\hat{x}^{\mathcal{S}}$, in accordance with the values expected on each wire when evaluating $f(x)$.

Next, for each $g \in C_g$ the simulator sets $\hat{x}^{\mathcal{S}}[l_g] = \eta_g$ and $\hat{x}^{\mathcal{S}}[r_g] = \rho_g$. For the rest of the entries in $\hat{x}^{\mathcal{S}}$ it always chooses the 0-bit (these are the "slack" entries caused by the ϵ_g fraction and will just be discarded). Then $\mathcal{S}$ sends $(\hat{f}^{\mathcal{S}},\hat{x}^{\mathcal{S}})$ to $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}$ and receives back $(\hat{F}^{\mathcal{S}},\hat{X}^{\mathcal{S}})$. The remaining steps are the same as in $\mathcal{S}^b$, but adapted to the setting of only one function.

We now define the hybrid $\mathsf{G}^1(1^\kappa)$ which is induced by $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$ as explained above. We then define a new hybrid $\mathsf{G}^2(1^\kappa)$ to be exactly like $\mathsf{G}^1(1^\kappa)$, but where the simulator cheats and looks into the game $\mathrm{Aut}_{\mathcal{G},\mathbf{\Phi}_{\mathsf{xor}}}^{\mathcal{S}}(1^\kappa)$ to learn Δ. It then sets $A'_j = A_j \oplus \Delta$ for $j \in [A]$, which it uses in **Garble** Step 3 to construct the authenticators exactly like in $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$. By the same argument as in the proof of Lemma 5 we have that $\mathsf{G}^1(1^\kappa) \overset{c}{\approx} \mathsf{G}^2(1^\kappa)$ by correlation-robustness of $\mathcal{H}$. It is also the case that $\mathsf{G}^2(1^\kappa) \overset{c}{\approx} \mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$. This follows from the arguments already given for $\mathrm{SecIndAct}_{\mathsf{IGarb}}^{\mathcal{B},0}(1^\kappa) \overset{c}{\approx} \mathsf{H}^{2,0}(1^\kappa)$. In particular we have that $\mathsf{G}^1(1^\kappa) \overset{c}{\approx} \mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$ as efficient transformations maintain indistinguishability.

By the above it follows that $\mathcal{B}$ cannot tell whether it is playing $\mathsf{G}^1(1^\kappa)$ or the actual game $\mathrm{AutAct}_{\mathsf{IGarb}}^{\mathcal{B}}(1^\kappa)$. This means if he can win in the actual game he can also win in the hybrid. We now show how winning in the hybrid translates directly into winning the $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$ game.

When receiving Z'_{A} from $\mathcal{B}$ which wins in the hybrid $\mathsf{G}^1(1^\kappa)$, meaning there is a $j \in [m_{\mathsf{A}}]$ such that $Z_j \neq Z'_{\mathsf{A},j}$ the simulator forwards $Z'_{\mathsf{A},j}$ to $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$. It is now easy to see that if $\mathcal{B}$ can win the hybrid game $\mathsf{G}^{\mathcal{B}}(1^\kappa)$ with non-negligible probability then $\mathcal{S}^{\mathcal{B}}$ also wins the $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$ game with non-negligible probability since the keys are constructed using $\mathrm{Aut}_{\mathcal{G}}^{\mathcal{S}^{\mathcal{B}}}(1^\kappa)$. As $\mathcal{G}$ is assumed aut-secure we therefore conclude that IGarb is aut.act-secure as well. ☐

Lemma 7 (knof). *The scheme* IGarb *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^κ, then output some function f and finally ran an instance of IGb playing the role of E against an honest C. Also assume that the output of C is $(F,e,d) \neq \perp$ (else there is nothing to show). We now make the following observations:

1. As C did not output $\perp$ during the execution of IGb, by the correctness of $\mathcal{F}_{\mathsf{COM}}$, it must mean that $\mathcal{B}$ committed (and later opened to) valid cut-and-choose challenges $(\eta_g,\rho_g,o_g)_{g \in C_g},(\sigma_j,a_j)_{j \in C_a}$ and bucket mapping functions $(\mathsf{Bof},\mathsf{AWof})$ and thus these are part of his view.
2. The next thing to note is that for all $g \in [Q]$ and all $j \in [A]$ we have that C sent $(\delta_{o_g},\gamma_{o_g})$ and H_{a_j} to B.
3. It is also clear that C sent all solderings specified by Bof and AWof to $\mathcal{B}$, as it is honest.

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, the garbled gates, the wire authenticators, the solderings, Bof and AWof completely define $\left\{\left\{\left(\delta_{o_g},\widetilde{\gamma}_{o_g}\right)\right\}_{g \in \mathsf{Bu}_{h_t}}\right\}_{h_t \in \mathsf{Gates}}$ and $\left\{\left\{\widetilde{H}_{a_i}\right\}_{i \in \mathsf{Au}_j}\right\}_{j \in \mathsf{Wires}}$.

Ex_E therefore simply extracts the above information from $\mathcal{B}$'s view and lets its output be

$$\hat{F}=\left(f,T,\mathsf{HeadGates},\left\{\left\{\left(\delta_{o_g},\widetilde{\gamma}_{o_g}\right)\right\}_{g\in\mathsf{Bu}_{h_t}}\right\}_{h_t\in\mathsf{Gates}},\left\{\left\{\widetilde{H}_{a_i}\right\}_{i\in\mathsf{Au}_j}\right\}_{j\in\mathsf{Wires}}\right).$$

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. We thus conclude that the scheme IGarb has the knof property. ☐

The proofs of the remaining properties, tok.com, unqie, unqoe, and rob.con, require an extractor Ex_C, which we will now define. We will also show that IGarb satisfies the projectiveness property. Also, the proof of rob.con and unqie requires a helper lemma Lemma 13. We will prove this lemma in the end of this appendix. Before continuing with the rest of the properties we describe the extractor Ex_C.

$\mathsf{Ex}_\mathsf{C}(\mathcal{A})$. Let $\mathcal{A}$ be an adversary playing the role of C in an execution of IGb. Assume that it is in a state where it first received an input 1^κ, then output some function f and finally ran an instance of IGb playing against an honest E. Also assume that the output of E is $(F,v)\neq\perp$. Since $\mathcal{F}_\mathsf{HCOM}$ is a UC-secure commitment scheme there exists a simulator $\mathcal{S}$ that can extract all values committed to in the protocol, including all wires keys and Δ. As E chose Bof it also knows which wires correspond to the input and output wires of F.

Using the extracted wire keys and knowledge of Bof, Ex_C proceeds as follows. For all $h^l\in\mathsf{LINP}$, all $h^r\in\mathsf{RINP}$ and all $i\in\mathsf{Inputs}$, if $\mathsf{lp}(\mathsf{Bof}(h^l))=i$ let $\hat{e}_i=\left(L^0_{h_l},L^0_{h_l}\oplus\Delta\right)$ or if $\mathsf{rp}(\mathsf{Bof}(h^r))=i$ let $\hat{e}_i=\left(R^0_{h_r},R^0_{h_r}\oplus\Delta\right)$. Then define $\hat{e}=(\hat{e}_1,...,\hat{e}_n)$. Analogously for all $h\in\mathsf{OUT}$ and all $o\in[m]$ where $\mathsf{Bof}(h)=o$, Ex_C lets $\hat{d}_o=\left(O^0_h,O^0_h\oplus\Delta\right)$. Then define $\hat{d}=\left(\hat{d}_1,\hat{d}_2,...,\hat{d}_m\right)$ and output $\left(\hat{e},\hat{d}\right)$.[6]

Lemma 8 (proj). *The scheme* IGarb *has the proj property.*

Proof. It follows from the underlying garbling scheme $\mathcal{G}$ having projective coding that the produced e and d of Gb_π are of the required form and that IEn and IDe work for individual elements as well. As the projective de-encoder IEn^{-1} and IDe^{-1} have already been defined in Section 3 for schemes with projective coding we conclude that the scheme IGarb has the proj property. ☐

Before continuing with the proofs of the remaining properties we need to define what it means for a garbled gate or a wire authenticator to be "corrupt". We specify this in the following definition.

Definition 3 (Corrupt GGate/AWire). *After an execution of* Gb_π *we have that:*

- *A garbled gate (δ_g,γ_g) with left and right input wire index l_g,r_g and output wire index o_g is* corrupt *if for any $a,b\in\{0,1\}$ we have $\mathsf{GEv}(\delta_g,L^a_g,R^b_g,\gamma_g)\neq O^{a\wedge b}_g$. Here L^0_g,R^0_g and O^0_g are the values sent to $\mathcal{F}_\mathsf{HCOM}$ for index l_g,r_g and o_g and $L^1_g=L^0_g\oplus\Delta,R^1_g=R^0_g\oplus\Delta$ and $O^1_g=O^0_g\oplus\Delta$ where Δ is the value sent for the index τ.*
- *A wire authenticator H_{a_j} is* corrupt *if for $(H^0_{a_j},H^1_{a_j})\leftarrow H_{a_j}$ we have $\{H^0_{a_j},H^1_{a_j}\}\neq\{\mathcal{H}(A_j),\mathcal{H}(A_j\oplus\Delta)\}$ where A_j is the value sent to $\mathcal{F}_\mathsf{HCOM}$ for index a_j and Δ is the value sent for the index τ.*

We say a garbled gate or wire authenticator is correct *if it is not corrupt. Furthermore we say a bucket is corrupt if it consists of only corrupt gates or if a combined majority of its gates and wire authenticators is corrupt. Again we say a bucket is correct if it is not corrupt.*

The above definition loosely says that a garbled gate or wire authenticator is corrupt if it is not consistent with the keys committed to using $\mathcal{F}_\mathsf{HCOM}$. We now continue with the proofs of the remaining properties.

[6] We here assume a concrete form of $\hat{e}$ and $\hat{d}$. This is not necessarily the same form as the one of e and d defined by a concrete scheme $\mathcal{G}$. However we assume that given the information included in $\hat{e}$ and $\hat{d}$ one can always convert to the correct form if necessary when using En and De of $\mathcal{G}$ (which IEn and IDe does).

Lemma 9 (rob.con). *If* IGarb *has the corr property, then it also has the rob.con property except with probability at most*

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

Proof. Run $\mathcal{A}$ to produce a f, then run IGb with $\mathcal{A}$ playing the role of C and denote the output of the evaluator (F,v). Let $\hat{e}=\left(\hat{X}_1^0,\hat{X}_1^1,...,\hat{X}_n^0,\hat{X}_n^1\right)$ and $\hat{d}=\left(\hat{d}_1,\hat{d}_2,...,\hat{d}_m\right)$ be the output of $\mathsf{Ex}_{\mathsf{C}}(\mathcal{A})$ and let $x=(x_1,...,x_n)$ be the output of $\mathcal{A}$. Furthermore we parse $\left(\hat{Z}_j^0,\hat{Z}_j^1\right)\leftarrow\hat{d}_j$ for all $j\in[m]$. We see from inspection of $\mathrm{RobCon}_{\mathsf{IGarb}}^{\mathcal{A}}(1^\kappa)$ in Fig. 3 that it is sufficient to prove that

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

where $z\leftarrow\mathsf{Iev}(f,x)$.

By Lemma 13 and the observation that the probabilities of corrupt gates and authenticators ending up in the same bucket are independent, using a union bound we have that when evaluating F using IEv, all buckets will always output the correct key except with probability at most

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

The reason for the bucket always outputting the correct key is that 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. Since the scheme has the corr property it follows that indeed $\mathsf{IEv}(F,(\hat{X}_1^{x_1},...,\hat{X}_n^{x_n}))=(\hat{Z}_{w-m+1}^{z_1},...,\hat{Z}_w^{z_m})$ which concludes the proof. ☐

Lemma 10 (unqie). *If* $\mathcal{H}$ *is a collision-resistant hash function, then* IGarb *has the property unqie except with probability at most*

$$n\cdot\sum_{v=1}^{\lceil\frac{\lambda}{2}\rceil}\prod_{l=\lambda}^{v}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)$$

Proof. Run $\mathcal{A}$ to produce a f, then run IGb with $\mathcal{A}$ playing the role of C and denote the output of the evaluator (F,v). Let $\hat{e}=\left(\hat{X}_1^0,\hat{X}_1^1,...,\hat{X}_n^0,\hat{X}_n^1\right)$ and $\hat{d}=\left(\hat{d}_1,\hat{d}_2,...,\hat{d}_m\right)$ be the output of $\mathsf{Ex}_{\mathsf{C}}(\mathcal{A})$ and let X be the output of $\mathcal{A}$. We start by assuming $\mathsf{IEn}(\hat{e}_i,\mathsf{IEn}^{-1}(\hat{e}_i,X_i))\neq X_i$ for at least one $i\in[n]$ as else there is nothing to prove. Notice that this can only occur if $\hat{X}_i^0\neq X_i\neq\hat{X}_i^0\oplus\Delta$. We shown that in this case $\mathsf{IEv}(F,X)=\perp$ except with bounded probability which for properly chosen λ and α will be negligible. As the protocol execution does not abort, we have for each $i\in[n]$ that $\mathcal{A}$ correctly instructs $\mathcal{F}_{\mathsf{HCOM}}$ to open to solderings of λ wire authenticators onto $\hat{X}_i^0$. In order for $\mathsf{IEv}(F,X)\neq\perp$, then in Step 3 of IEv X_i needs to be accepted by a majority of the input wire authenticators. As we have $X_i\notin\left\{\hat{X}_i^0,\hat{X}_i^0\oplus\Delta\right\}$ this can only happen if at least a majority of the wire authenticators are corrupt. However by Lemma 13, $\mathcal{H}$ being collision-resistant and the union bound, the probability of this occurring is at most

$$n\cdot\sum_{v=1}^{\lceil\frac{\lambda}{2}\rceil}\prod_{l=\lambda}^{v}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)$$

which concludes the proof. ☐

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

Proof. Run $\mathcal{A}$ to produce a f, then run IGb with $\mathcal{A}$ playing the role of C and denote the output of the evaluator (F,v). Let $\hat{e}=\left(\hat{X}_1^0,\hat{X}_1^1,\ldots,\hat{X}_n^0,\hat{X}_n^1\right)$ and $\hat{d}=\left(\hat{d}_1,\hat{d}_2,\ldots,\hat{d}_m\right)$ be the output of $\mathsf{Ex}_{\mathsf{C}}(\mathcal{A})$ and let X be the output of $\mathcal{A}$. We start by assuming that $\mathsf{IEv}(F,X)\rightarrow Z\neq\perp$ as else there is nothing to prove. Thus, by a simple reduction to the unqie property we have that $\mathsf{IEn}(\hat{e}_i,\mathsf{IEn}^{-1}(\hat{e}_i,X_i))=X_i$ for all $i\in[n]$. Then by a reduction to rob.con we have that $\mathsf{IDe}(\hat{d}_j,Z_j)\rightarrow z_j\neq\perp$ for all $j\in[m]$. By definition of IDe^{-1} in Lemma 8 it now clearly follows that $\mathsf{IDe}^{-1}(\hat{d}_j,\mathsf{IDe}(\hat{d}_j,Z_j))=\mathsf{IDe}^{-1}(\hat{d}_j,z_j))=Z_j$ for all $j\in[m]$ which concludes the proof. □

Lemma 12. IGarb *has property tok.com except with probability* 2^{-s}*.*

Proof. Run $\mathcal{A}$ to produce a f, then run IGb with $\mathcal{A}$ playing the role of C and denote the output of the evaluator (F,v). Let $\hat{e}=\left(\hat{X}_1^0,\hat{X}_1^1,\ldots,\hat{X}_n^0,\hat{X}_n^1\right)$ and $\hat{d}=\left(\hat{d}_1,\hat{d}_2,\ldots,\hat{d}_m\right)$ be the output of $\mathsf{Ex}_{\mathsf{C}}(\mathcal{A})$ and let X,x',z' be the output of $\mathcal{A}$. As the execution of IGb did not abort we know that $\mathcal{A}$ answered satisfactory in the **VerLeak** phase. This can be seen as A successfully answering $\hat{s}={}^{s}/_{\log_2(|\{0,1\}|)}=s$ linear combinations and by the proof of Theorem 1 in [FJNT15], it now follows that the least significant bit of the committed global difference Δ with index τ is 1 and that v consists of least significant bits of the committed 0-keys corresponding to B's input and output wires except with probability 2^{-s}.

Following the description of the game let $Z\leftarrow\mathsf{IEv}(F,X)$, $z_j\leftarrow\mathsf{IDe}(\hat{d}_j,Z_j)$, and $x_i\leftarrow\mathsf{IEn}^{-1}(\hat{e}_i,X_i)$ for $i\in[n]$. We assume none of these values equal $\perp$ as else there is nothing to prove. Now assume that there exists an $i\in[n_{\mathsf{B}}]$ such that $\mathsf{IVe}(v_i,X_{n_{\mathsf{A}}+i},x'_{n_{\mathsf{A}}+i})=\top$ and $x_{n_{\mathsf{A}}+i}\neq x'_{n_{\mathsf{A}}+i}$ or there exists a $j\in[m_{\mathsf{B}}]$ such that $\mathsf{IVe}(v_{n_{\mathsf{B}}+j},Z_{m_{\mathsf{A}}+j},z'_{m_{\mathsf{A}}+j})=\top$ where $z_{m_{\mathsf{A}}+j}\neq z'_{m_{\mathsf{A}}+j}$. From the description of IVe this can only happen if v_i or $v_{n_{\mathsf{B}}+j}$ is not the least significant bit of the corresponding 0-key. However by a similar analysis as in Theorem 1 of [FJNT15] this only occurs with probability at most 2^{-s}. □

We now state and prove the "bucketing" lemma used in the proofs of rob.con and unqie. Informally the lemma provides bounds on the probability that any bucket is corrupt and on the probability that a majority of the wire authenticators on any input wire is corrupt.

Lemma 13 (Probability of corrupt bucketing). *For randomly generated F output by* IGb *where $\alpha=\beta-1$, for any bucket is correct except with probability at most*

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

Furthermore, for any input wire, if $\mathcal{H}$ is a collision-resistant hash function then the probability that a majority of the λ wire authenticators end up being corrupt is at most

$$\sum_{v=1}^{\lceil\frac{\lambda}{2}\rceil}\prod_{l=\lambda}^{v}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)$$

Proof. First consider the following game against an adversary $\mathcal{A}$:

- $\mathcal{A}$ is given two buttons he can press, one corrupt gate button and one corrupt authenticator button for an arbitrary bucket as defined in Gb_π.
- The outcome of pressing either button will be either "success", "failure" or "nothing" where "success" will mean a gate (wire authenticator) of the bucket will be arbitrarily corrupted by $\mathcal{A}$ as long as it has the correct form, "failure" will mean that she looses the game immediately and "nothing" will have no effect whatsoever. We let $\mathcal{A}$ learn the outcome of pressing each button immediately after pressing.

– We say $\mathcal{A}$ wins the game if at any time after pushing the above buttons the bucket becomes corrupt.

It should be clear that the above game is sufficient to model $\mathcal{A}$'s ability to corrupt a bucket in an execution of IGb. In fact it gives her strictly more power as in the above game she can adaptively change her strategy based on the outcome of the current result of pressing a button. This means she has no additional risk of getting caught, once she deems her corruption strategy has succeeded. This is in contrast to a real execution of IGb 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 B). The event of "failure" will model the probability that $\mathcal{A}$ gets caught in either of the cut-and-choose checks performed by B in IGb and "nothing" will model that a gate (authenticator) is corrupt, but is not selected for checking and falls into another bucket than the one we look at, it is selected for checking and does not end up in any bucket (discarded) or it is selected for checking but passes the check (discarded). It is sufficient to only look at the game for a single bucket, because in IGb 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 therefore conclude that $\mathcal{A}$ can win the above game with strictly higher probability than corrupting any bucket in an execution of IGb. We therefore continue the proof by bounding the probability of winning the above game.

We need to calculate the probability of the events success and failure when pressing the above buttons such that they correctly model an execution of IGb. We first consider the gate button. Recall that a gate is chosen for checking in the cut-and-choose step of IGb with probability p_g. If a gate is corrupted and selected for checking we see that it will get caught with probability at least $\frac{1}{4}$, since it is checked on one random input out of the 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.[7] 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

$$\frac{(1-p_g)4i}{p_g q\beta + (1-p_g)4i} \tag{2}$$

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

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 for the gates. This can be seen from a simple reduction to $\mathcal{H}$ being a collision-resistant hash function, since if $\mathcal{A}$ can successfully cheat in the cut-and-choose Ver check with noticeable probability greater than $\frac{1}{2}$ then she can find a collision for $\mathcal{H}$ with noticeable probability as well. By the same procedure as above we therefore have that the relative probability of success becomes

$$\frac{(1-p_a)2j}{p_a(q\alpha+n\lambda)+(1-p_a)2j} \tag{3}$$

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. 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.

[7] This is an upper bound since there is a slight probability a corrupt gate (wire) will not be part of $\overline{E}_g$ ($\overline{E}_a$), the domains of Bof and AWof. Also we do not consider non-corrupt gates (wires) taking up any slots.

We now turn to the calculation of this probability. From (2) and (3), we conclude a bucket is left "uncorrupt" after $\mathcal{A}$ has participated in the above mentioned experiment except with probability at most

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

Using the same type of experiment for an arbitrary input wire where $\mathcal{A}$ is given a corrupt authenticator button only. Using the same reasoning as above we see that the probability of a majority of corrupt authenticators for any input wire is at most

$$
\begin{aligned}
& \prod_{l=\lambda}^{1}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)+\\
& \prod_{l=\lambda}^{2}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)+\\
& \vdots\\
& \prod_{l=\lambda}^{\lceil\frac{\lambda}{2}\rceil}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)\\
= & \sum_{v=1}^{\lceil\frac{\lambda}{2}\rceil}\prod_{l=\lambda}^{v}\left(\frac{(1-p_a)2l}{p_a(q\alpha+n\lambda)+(1-p_a)2l}\right)
\end{aligned}
$$

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

7 Performance Comparison

The communication complexity of TinyLEGO (and other LEGO-based protocols [NO09, FJN+13]) is $O(qks/\log q)$. This is asymptotically better than other recent maliciously secure two-party protocols based on garbled circuits [Bra13, Lin13, HKE13, AMPR14, FJN14] which achieve at best $O(qks)$. 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.

We base our performance comparison on efficiency counts based on bits needed to be communicated. This allows others to do the same and do reasonable comparison to our protocol. We have chosen this benchmark since experience from implementations of protocols based on Yao's garbling [KSS12, FN13, FJN14] show that with realistic circuits the communication overhead in general becomes the major bottleneck. Especially the communication from the constructor (A) to the evaluator (B). 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 advent of the half-gate garbling scheme [ZRE15] each garbled gate is represented using as little as 2κ bits while still being compatible with the free-XOR technique [KS08]. With minor modification this scheme also fulfills the requirements for a key-size preserving free-XOR garbling scheme. In this efficiency count we will therefore instantiate our protocol using this garbling scheme together with the recent commitment scheme of [FJNT15]. Using half gate garbling most of the values we need to commit to are simply random 0-keys (and Δ), thus we can exploit that [FJNT15] is more efficient when committing to random values. In order to do a fair comparison with MiniLEGO we consider this protocol instantiated with these primitives as well.

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 $6.2q\kappa s$ bits. Recent protocols [Lin13, HKE13, Bra13, AMPR14, FJN14] only require A to send down to s garbled circuits to B, yielding instead a total of down to $2q\kappa s$ 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 $c\cdot 2q\kappa s$ 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 $2q\kappa s$ in the standard model and $q\kappa s$ (or even less) in the random oracle model.[8]

Table 1 shows the amount of data that A must send to B for various circuit sizes and security levels. In the table $q\kappa s$ 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. $2q\kappa s$ reflects current best non-LEGO protocols in the standard model, *e.g.*, [Lin13]. Table 1 also shows communication overhead for MiniLEGO and TinyLEGO. We fix the computational security parameter to $k=127$ and therefore have $\kappa=128$ due to the point-and-permute optimization used. We also set the digest size of $\mathcal{H}$ used in TinyLEGO to $k'=80$. This is following the reasoning in [Lin13] on how to choose the digest size for the encoded translation tables therein. For each value of s and circuit size q, the parameters of MiniLEGO (β') and TinyLEGO (β,α,p_g,p_a) have been chosen so as to minimize the overall communication overhead while still guaranteeing security except with probability 2^{-s}.[9]

As expected we outperform MiniLEGO on all circuit sizes. This is due to our optimizations of how buckets are constructed and the flexible way of choosing the cut-and-choose check fraction. 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 $5\cdot 10^5$. 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 $\kappa=128$, $k'=80$, different values of s and some selected parameters α, β, p_a, p_g, the minimal circuit size q where our protocol outperforms [Lin13] 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.

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

[9] This paper does not give a method for finding the provably optimal parameters. Instead, we searched for good parameters using a script.

s	Protocol	Circuit size q								
		10^3	10^4	10^5	$5\cdot10^5$	10^6	$5\cdot10^6$	10^7	10^8	10^9
40	$2q\kappa s$ [Lin13]	0.010	0.095	0.95	4.77	9.5	48	95	954	9,537
40	$q\kappa s$ [FJN14]	0.005	0.048	0.48	2.38	4.8	24	48	477	4,768
40	MiniLEGO	0.019	0.151	1.08	5.39	10.8	32	64	641	6,418
40	TinyLEGO	0.015	0.104	0.80	3.42	6.3	30	58	534	4,799
60	$2q\kappa s$ [Lin13]	0.014	0.143	1.43	7.15	14.3	72	143	1,431	14,305
60	$q\kappa s$[FJN14]	0.007	0.072	0.72	3.58	7.2	36	72	715	7,153
60	MiniLEGO	0.040	0.290	2.37	9.19	18.4	65	131	1,310	13,096
60	TinyLEGO	0.026	0.175	1.30	5.68	10.7	47	91	762	6,990
80	$2q\kappa s$ [Lin13]	0.019	0.191	1.91	9.53	19.1	95	191	1,907	19,073
80	$q\kappa s$ [FJN14]	0.010	0.095	0.95	4.77	9.5	48	95	954	9,537
80	MiniLEGO	0.059	0.403	3.41	13.92	27.8	108	216	2,163	15,415
80	TinyLEGO	0.055	0.253	2.02	8.13	15.6	68	131	1,120	10,381

Table 1. Amount of gibibits (*i.e.*, 2^{30} bits) that A must send to B for $\kappa = 128$ (ignoring data not dependent on circuit size).

s	α	β	p_a	p_g	q	[Lin13]	TinyLEGO
40	3	4	0.02	0.05	4,059,866	38.72	24.16 (0.62)
40	3	4	0.10	0.10	953,021	9.09	6.02 (0.66)
40	3	4	0.15	0.15	514,297	4.90	3.46 (0.71)
40	3	4	0.5	0.15	337,670	3.22	2.68 (0.83)
40	4	5	0.15	0.20	27,335	0.26	0.25 (0.95)
60	4	5	0.02	0.05	8,501,436	121.61	77.58 (0.64)
60	4	5	0.05	0.05	5,289,299	75.66	48.66 (0.64)
60	4	5	0.20	0.25	593,941	8.50	6.84 (0.81)
60	5	6	0.25	0.10	157,297	2.25	1.95 (0.87)
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	87.43 (0.69)
80	7	6	0.02	0.10	2,120,537	40.45	31.39 (0.78)
80	6	7	0.10	0.15	324,250	6.18	5.50 (0.89)
80	8	7	0.05	0.20	123,127	2.35	2.33 (0.99)

Table 2. Gibibits (2^{30} bits) sent from A to B in TinyLEGO compared to [Lin13] for various parameters. Data independent of circuit size is ignored. The numbers in parentheses are the relative communication overhead of TinyLEGO compared to [Lin13].

We conclude 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 27,335 gates (0.21 gibibit) and at 953,021 gates our bandwidth (6.02 gibibit) is only 66% of [Lin13]. For 80-bit security, we outperform [Lin13] slightly at 123,127 gates (2.33 gibibit) and achieve 70% bandwidth at 6,6 million gates (126 gibibit).

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

7.1 Counting TinyLEGO communication

We here elaborate on how the numbers in Section 7 were computed. We ignore any terms that do not depend on the circuit size. In **Garble** for each original gate of f, A must send $\beta/(1-p_g-\epsilon_g)$ garbled gates and commit to three values for each of these. In addition, for each gate A also needs to construct and send $\alpha/(1-p_a-\epsilon_a)$ wire authenticators, where each authenticator involves sending $2k'$-bits and committing to a random value.

In the cut-and-choose step, we expect $\frac{\beta}{(1-p_g-\epsilon_g)}p_g$ gates to be checked and for each A opens to three committed values. Similarly, in the cut-and-choose of authenticators, we expect B to check $\frac{\alpha}{(1-p_a-\epsilon_a)}p_a$ authenticators, having A open one committed value. Finally, soldering requires A to open $3(\beta-1)+\alpha+2$ commitments for each original gate. Summing up we see that A in total sends

$$\frac{\beta g+3c}{(1-p_g-\epsilon_g)}+\frac{\alpha(2k'+c)}{(1-p_a-\epsilon_a)}+3op_g\frac{\beta}{(1-p_g-\epsilon_g)}+op_a\frac{\alpha}{(1-p_a-\epsilon_a)}+3o(\beta-1)+o\alpha+2o$$

bits pr. original gate of f where g is the size of a garbled gate, c is the cost in bits of committing to a value and o is the cost of opening a value in bits using $\mathcal{F}_{\mathsf{HCOM}}$.

On the Instantiating Primitives. As already mentioned we use the half-gate garbling scheme of [ZRE15] and the homomorphic commitment scheme of [FJNT15] to instantiate IGarb. Following this and relating to the calculation above we therefore have $g=2\kappa$ for the garbled gates. For the commitments $c=\Gamma$ if the commitment is a chosen value where Γ here is the length of the error correcting code used. However $c=\Gamma-\kappa$ if the commitment is a random value. As we are using half-gates we see that the two input 0-keys of a garbled gate are chosen at random in our protocol, while the output 0-key is computed as a function of the input keys. Therefore we conclude that A sends $2(\Gamma-k)+\Gamma$ bits to commit to the wires of each garbled gate. The length of the code depends on the statistical security parameter used. We see that for statistical security s, we need a linear code with minimum distance $d\geq s$. For $\kappa=128$ and $s=40,60,80$ one can use binary codes with parameters [262,128,40], [345,128,60], and [428,128,81], respectively. The first two codes were found in the MinT database [SS06, SS10] and the last one using the BCH encoder/decoder program available at the website of [MZ06]. Finally we note that all the openings of the homomorphic commitments in our protocol can be done using the batch-opening technique of [FJNT15] and thus require κ bits of communication each.

8 Overview of Variables and Parameters

A list of variable names and their meaning is given in Table 3.

Symbol	Meaning
s	Statistical security parameter.
k	Computational security parameter.
k'	The number of output bits of the hash function $\mathcal{H}$.
κ	The bit-length of the keys of the garbling scheme $\mathcal{G}$.
f	The plain description of the Boolean circuit to compute.
x	A bit string representing the constructor's (A's) input to the circuit.
y	A bit string representing the evaluator's (B's) input to the circuit.
z_A	The circuit output destined for A.
z_B	The circuit output destined for B.
z	The output of the circuit, defined as $z = z_\mathsf{A} \\| z_\mathsf{B}$.
q	Amount of AND gates in f.
n_A	Amount of input bits to the circuit from A, defined as $n_\mathsf{A} = \lvert x \rvert$.
n_B	Amount of input bits to the circuit from B, defined as $n_\mathsf{B} = \lvert y \rvert$.
n	Total amount of input bits to the circuit, defined as $n = n_\mathsf{A} + n_\mathsf{B}$.
m_A	Amount of output bits of the circuit for A, defined as $m_\mathsf{A} = \lvert z_\mathsf{A} \rvert$.
m_B	Amount of output bits of the circuit for B, defined as $m_\mathsf{B} = \lvert z_\mathsf{B} \rvert$.
m	Total amount of output bits from the circuit, defined as $m = z_\mathsf{A} + z_\mathsf{B}$.
w	Amount of wires in f, defined as $w = n + q$.
p_g	Fraction of garbled gates that should be checked.
ϵ_g	Fraction of garbled gates we need to get sufficient "slack".
p_a	Fraction of authentication wires that should be checked.
ϵ_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.
λ	The amount of authenticators used for each input wire.
ℓ_g	The gate replication factor, defined as $\ell_g = \frac{1}{1-p_g-\epsilon_g}$.
ℓ_a	The authentication wire replication factor, defined as $\ell_a = \frac{1}{1-p_a-\epsilon_a}$.
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 $A = (q\alpha + n\lambda)\ell_a$.
W	The amount of wires considered in a protocol execution, defined to be $W = 3Q + A + 1 + 2s$.
$\mathit{\Delta}$	The global difference on all wires, has index τ.
Wires	All the wires of the circuit f, in particular $\mathsf{Wires} = \{1,...,w\}$.
Inputs	Subset of the wires of f, in particular $\mathsf{Inputs} = \{1,...,n\}$.
Gates	Subset of the wires of f, in particular $\mathsf{Gates} = \{n+1,...,w\}$.
$\mathsf{Outputs}$	Subset of the wires of f, in particular $\mathsf{Outputs} = \{w-m+1,...,w\}$.
Bof	A β-to-1 map from garbled gates to buckets which represent gates of f.
Bu_g	The bucket corresponding to the circuit gate $g \in \mathsf{Gates}$.
AWof	A α-to-1 map from authenticators to buckets.
Au_j	The authenticators associated to the circuit wire $j \in \mathsf{Wires}$.
$\mathsf{HeadGates}$	The set of head gates of the buckets defined by Bof. Thus $\lvert\mathsf{HeadGates}\rvert = q$.
LINP	The set of head gates which has as left input wire a circuit input wire.
RINP	The set of head gates which has as right input wire a circuit input wire.
BLEAK	The set of indices for which B is allowed to decode a corresponding key, in particular $\mathsf{BLEAK} = \{n_\mathsf{A}+1, n_\mathsf{A}+2, ..., n\} \cup \{w-m_\mathsf{B}+1, w-m_\mathsf{B}+2, ..., w\}$.

Table 3. Overview of variables along with their meaning.

References

[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, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Germany.

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

[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, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 441–463, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle 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, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany.

[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, *SCN 14*, volume 8642 of *LNCS*, pages 358–379, Amalfi, Italy, September 3–5, 2014. Springer, Berlin, Germany.

[FJNT15] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. *IACR Cryptology ePrint Archive*, 2015:694, 2015.

[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, *ACNS 13*, volume 7954 of *LNCS*, pages 339–356, Banff, AB, Canada, June 25–28, 2013. Springer, Berlin, Germany.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 289–306, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[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*, 2011.

[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, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[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, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 458–475, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

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

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

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany.

[KMRR15] Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2PC. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259, Warsaw, Poland, March 23–25, 2015. Springer, Berlin, Germany.

[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, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Berlin, Germany.

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

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany.

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

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany.

[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, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473, New York, NY, USA, April 24–26, 2006. Springer, Berlin, Germany.

[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, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 36–53, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

[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, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Berlin, Germany, March 15–17, 2009.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Germany.

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

[RT13] Samuel Ranellucci and Alain Tapp. Constant-round secure two-party computation from a linear number of oblivious transfer. Cryptology ePrint Archive, Report 2013/099, 2013. `http://eprint.iacr.org/2013/099`.

[SS06] Rudolf Schürer and WolfgangCh. Schmid. Mint: A database for optimal net parameters. In Harald Niederreiter and Denis Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 457–469. Springer Berlin Heidelberg, 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, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

[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 CCS 13*, pages 523–534, Berlin, Germany, November 4–8, 2013. ACM Press.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

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

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Germany.