Privacy-Free Garbled Circuits with Applications To Efficient Zero-Knowledge

(Full Version)

Tore Kasper Frederiksen[1], Jesper Buus Nielsen[1], and Claudio Orlandi[1,⋆]

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

Abstract In the last few years garbled circuits (GC) have been elevated from being merely a component in Yao's protocol for secure two-party computation, to a cryptographic primitive in its own right, following the growing number applications that use GCs. Zero-Knowledge (ZK) protocols is one of these examples: In a recent paper Jawurek *et al.* [JKO13] showed that GCs can be used to construct efficient ZK for unstructured languages. In this work we show that due to the property of this particular scenario (i.e., one of the party knows all the secret input bits, and therefore all intermediate values in the computation), we can construct more efficient garbled schemes specifically tailored to this goal. As a highlight of our result, in one of our constructions only *one encryption* per gate needs to be communicated and XOR gates never require any cryptographic operation. In addition to making a step forward towards more practical ZK, we believe that our contribution is also interesting from a conceptual point of view: in the terminology of Bellare *et al.* [BHR12] our garbling schemes achieve authenticity, but no privacy nor obliviousness, therefore representing the first *natural* separation between those notions.

[⋆] Partially supported by the European Research Commission Starting Grant 279447 and the Danish National Research Foundation and The National Science Foundation of China (grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation. Tore is supported by Danish Council for Independent Research Starting Grant 10-081612. The research leading to these results has received funding from the European Union Seventh Framework Programme ([FP7/2007-2013]) under grant agreement number ICT-609611 (PRACTICE).

Contents

1 Introduction

A garbled circuit (GC) is a cryptographic tool that allows to evaluate "encrypted" circuits on "encrypted" inputs. Garbled circuits were introduced by Yao in the 80's in the context of secure-two party computation [Yao86], and they owe their name to Beaver *et al.* [BMR90].

Since then, garbled circuits have been used in a number of different contexts such as two- and multi-party secure computation [Yao86, GMW87], verifiable outsourcing of computation [GGP10], key-dependent message security [BHHI10], efficient zero-knowledge [JKO13] etc. However, it is not until recently that a formal treatment of garbled circuits appeared in the literature. The first proof of security of Yao's celebrated protocol for two party computation, to the best of our knowledge, only appeared a few years ago in [LP09], and it is not until [BHR12] that garbled circuits were elevated from a technique to be used in other protocols, to a cryptographic primitive in their own right.

Different applications of GC often use different properties of the garbling scheme: In some applications we need GCs to protect the privacy of encrypted inputs, in others we need GCs to hide partial information about the encrypted function, while in yet others we ask GCs to ensure that even a malicious evaluator cannot tamper with the output of the GC. In their foundational work, Bellare *et al.* [BHR12] formally defined the different security properties that different applications require from GCs, showed separations between them, and showed that the original garbling scheme proposed by Yao satisfies all of the above properties. This raises a natural question:

Can we construct garbling schemes tailored to specific applications,
which are more efficient than Yao's original construction?

In this work we give the first such example, namely a garbling scheme which only satisfies *authenticity* (in the terminology of Bellare *et al.*) but not *privacy*: One of the main properties of Yao's garbling scheme is that the circuit evaluator cannot learn the values associated to the internal wires during the evaluation of the garbled circuit. This implies that the evaluation of each garbled gate must be *oblivious* (it must be the same for each input combination). In this work we give up on this property and we construct a scheme where the evaluator learns the values associated which each wire in the circuit, and explicitly uses this knowledge to perform *non-oblivious* garbled gate evaluation. This allows us to reduce significantly the size of a garbled circuit and the computational overhead for the circuit constructor. We show that this does not have any impact on *authenticity*, i.e., the only thing that a malicious evaluator can do with a garbled input and a garbled circuit is to use them in the intended way, that is to evaluate the garbled circuit on the garbled input and produce the (correct) garbled output.

Our new garbling schemes can be immediately plugged-in in Jawurek *et al.* [JKO13] efficient zero-knowledge protocol for non-algebraic languages, and therefore we believe that our results have both practical and conceptual value.

1.1 Previous Garbling Schemes

Since the introduction of GCs by Yao, a number of optimizations have been proposed to increase their efficiency. Some of the most significant optimizations include *point-and-permute* [Rog91, MNPS04] (which reduces the work of the circuit evaluator from 4 to 1 decryption per garbled gate) the *row-reduction technique* [NPS99, PSSW09] (which allows to reduce the number of ciphertexts per gate to be sent, by fixing some of them to be constant values), the *free-XOR* and *fleXOR* techniques [KS08, KMR14] technique (which allows to garble/evaluate XOR gates using none/less cryptographic operations). In [BHR12, BHKR13] efficient garbling schemes, which only uses one call to a block-cipher for each row in a garbled gate, are presented. Information theoretic garbling schemes can efficiently be constructed [IK02, Kol05, KK12] for low-depth circuits. All these techniques lead to very efficient garbling schemes that are used today in practical implementation of secure two-party computation. Our optimization is conceptually different from all of the above, as our scheme is not "general purpose" and does not satisfy privacy.

LEGO GCs [NO09, FJJBNO13] are different from Yao GCs as they allow to generate garbled gates independently of each other and then solder them at a later time into a functional garbled circuit, and have applications for secure two-computation in the presence of active corruptions.

The size of garbled input in Yao-style GCs grows linearly in the security parameter. In [AIKW13] a garbling scheme where the garbled input grows only by a constant factor is presented at the price of using public-key primitives (traditional GCs only use symmetric key operations). Traditional GCs only work on Boolean circuits, while [AIK11] presents a way of garbling arithmetic circuits directly.

All previously discussed garbling schemes are *one-time*, meaning that no security is guaranteed against an adversary that receives the garbling of two different inputs for the same garbled circuit. A recent line of work considers *reusable garbled circuits* [GKP$^+$12] and their (asymptotic) overhead [GGH$^+$13]. While the concept of reusable garbled circuits has numerous applications in establishing important theoretical feasibility result, their use of heavy crypto machinery makes them (still) far from being practical.

Finally, it is worth mentioning that garbling schemes exist also for non-circuit based computation, such as RAM programs [LO13, GHL$^+$14].

1.2 Our Contributions

We propose some novel garbling schemes which satisfy authenticity only and are more efficient than general purpose garbling schemes:

Privacy Free GRR1 with cheap XOR: In this garbling scheme we only send one ciphertext for each encrypted gate (both XOR and non-XOR). The circuit evaluator uses 3 calls to a *Key Derivation Function* (KDF) for each non-XOR gate, and none for each XOR gate (so from a computational point of view XOR gates are free). The scheme combines the row reduction technique with non-oblivious gate evaluation.

Privacy Free GRR2 with free-XOR: In this garbling scheme we send two ciphertexts for each encrypted non-XOR gate, and XOR gates are "for free". The circuit evaluator uses 3 calls to a KDF for each non-XOR gate (and none for XOR gates). The scheme is similar to GRR1, but using the free-XOR technique reduces the degrees of freedom we have in choosing the output keys and therefore require higher communication complexity for non-XOR gates.

Privacy Free fleXOR: In this garbling scheme we combine either our GRR1 or GRR2 scheme with the fleXOR technique of [KMR14]. The cost of non-XOR gates is unchanged from the previous scheme, i.e. one or two ciphertexts per gate respectively, but now the cost of XOR gate depends on the structure of the circuit: XOR gates require no cryptographic operations, while for communication, depending on the circuit structure, XOR gates require communication of $2, 1$ or 0 ciphertexts. Note that also our fleXOR, being tailored for privacy-free garbled circuits, performs better than the original.

Furthermore, we present a formal generalization of garbling schemes with gates with arbitrary fan-in and show how to construct each of our privacy-free schemes in such a setting. It turns our that all types of our privacy-free garbled gates yield even more significant improvements in computation (and in some settings also communication) over general garbled garbles when fan-in is larger than two.

1.3 Overview of Our Schemes

In a nutshell, our garbling scheme works as follows. Consider a NAND gate, with associate input keys L^0, L^1, R^0, R^1 for the left and right wire respectively, and output keys O^0, O^1. The circuit constructor needs to provide the evaluator with a cryptographic gadget that, on input L^a, R^b, outputs the corresponding output key $O^{a \bar{\wedge} b}$. Remember that our goal is not privacy, but only authenticity, meaning that the evaluator is allowed to learn a and b but even a corrupted evaluator should not learn $O^{1-(a \bar{\wedge} b)}$. In particular, this means that the evaluator should learn O^0 if and only if (iff) he holds both L^1, R^1. This can be ensured by encrypting O^0 under *both* L^1 and R^1.

On the other hand, it is enough that one of the inputs is 1 for the output to be 1, so it "should be enough" to hold L^0 or R^0 to learn O^1. In standard Yao GCs we do not want the evaluator to learn which of the three possible combinations of input keys he owns between (L^0, R^0), (L^0, R^1) and (L^1, R^0) (nor the output of the gate) and therefore we encrypt O^1 under all the three possibilities in the same way as we encrypt the 0 key. But if the evaluator is allowed to know which bits keys correspond to, we can simply encrypt O^1 separately under L^0 and R^0, thus saving an encryption.

Note that, using the row-reduction technique, we can instead derive O^0 as $O^0 = \mathsf{KDF}(L^1, R^1)$ and therefore we can remove one ciphertext from the garbled table. We now have two-choices:

- If we want to be compatible with the free-XOR technique the value O^1 is already determined by O^0 and the global difference Δ, and thus no more row-reduction is possible.
- Alternatively we can decide to give up on free-XOR and derive O^1 as $O^1 = \mathsf{KDF}(L^0)$, thus removing yet another ciphertext from the garbled table, that now contains only the ciphertext $C = O^1 \oplus \mathsf{KDF}(R^0)$.

"Almost" free-XOR. If we choose the second path, we need an efficient way of garbling the XOR gates: we do so by defining the output keys O^0 and O^1 respectively as $O^0 = L^0 \oplus R^0$ and $O^1 = L^0 \oplus R^1$. Of course it might be that at evaluation time the evaluator holds L^1 instead of L^0, and thus we provide him with an "advice" to compute the correct output key in this case. It turns out that it suffices to reveal the value $C = L^0 \oplus R^0 \oplus L^1 \oplus R^1$. Due to the symmetry of the XOR gate, now the evaluator can always derive the correct output key. Note that now XOR gates do not require any cryptographic operation but only k bits of communication, and therefore are "almost" for free.

The paranoid reader might now worry on whether revealing the XOR of all input keys affects the security of our scheme, and the impatient reader might not want to wait for the formal proof which appears later in the paper: Intuitively revealing C does not represent a problem because, if it did, then the free-XOR technique would be insecure as well: In (standard) free-XOR the value C is always 0, as $L^0 \oplus L^1 = R^0 \oplus R^1$, and therefore known to the adversary already.

Privacy free fleXOR. Finally we combine our technique with the recent fleXOR garbling scheme [KMR14]. A central concept in fleXOR is to look, for each wire, at the XOR between the two keys associated to that wire, or the *offset* of that wire. While in freeXOR the offset is a constant for the whole circuit (therefore fixing half of the keys in the circuit), in fleXOR wires are ordered in a way to maximize the number of offsets which are the same, while at the same time leaving the circuit garbler the ability to choose freely the output keys for the non-XOR gates.

The fleXOR wire ordering induces a partitioning of the wires for each XOR gates. In particular, each XOR gates is assigned a parameter t which denotes how many input wires have offset *different* than the output wire. Then a 0-XOR gate can be garbled exactly like in free-XOR, while for t-XORs (with $t > 0$) the garbler sends t ciphertexts to the evaluator, which are used to "adjust" the offsets of those input wires. In the privacy-free case, exploiting non-oblivious gate evaluation, we can simply reveal the XOR of the offsets instead, exactly like in our GRR1 scheme. So, while the original fleXOR requires the garbler and the evaluator to perform $2t$ and t calls respectively to the KDF, we do not require any cryptographic operations for fleXOR gates.

Garbling XORs. To conclude this technical introduction, we would like to present the reader with a recap of the different ways in which XOR gates are garbled in this paper. Like before, let L^0, L^1, R^0, R^1, and O^0, O^1 be the keys for the left, right and output wire, and let Δ_L, Δ_R and Δ_O be their differences, the offsets associated to the wires. Now, the "baseline" garbling of a XOR gate is done as follows: the garbler sets $O^0 = L^0 \oplus R^0$, then computes and send to the evaluator the following values:

$$C_L = \Delta_L \oplus \Delta_O \text{ and } C_R = \Delta_R \oplus \Delta_O$$

Now, on input keys L_a, R_b, the evaluator retrieves

$$O^{a \oplus b} = L^a \oplus R^b \oplus a \cdot C_L \oplus b \cdot C_R$$

The baseline garbling transmits 2 ciphertexts, but in most cases we can do better.

GRR1: In this case the garbler can freely choose both Δ_O, which is set to be equal to Δ_L (so that $O^1 = L^1 \oplus R^0$) and therefore we do not need to communicate C_L, saving one ciphertexts wrt the baseline.

free-XOR: Here it holds that $\Delta_L = \Delta_R = \Delta_O$, therefore both $C_L = C_R = 0$ and no ciphertexts need to be transfered.

fleXOR: a t-XOR gate is garbled like in the baseline garbling when $t = 2$, like in GRR1 when $t = 1$ and like free-XOR when $t = 0$.

1.4 Efficiency Improvements

Our garbling schemes offer different performances in terms of communication and computation overhead. It is natural to ask which one is the most efficient one. Like most interesting questions, the answer is not as simple as one might want, and to answer which garbling scheme offers the best performances one must define the price of communication vs. computation. The ultimate answer depends on the actual hardware setting (CPU, network) on which the protocol is to be run and can only be determined empirically.

	# of Gates		Private			**Privacy-free**			
Circuit	AND	XOR	GRR2	free-XOR	fleXOR	**GRR1**	**free-XOR**	**fleXOR**	Saving
DES	18124	1340	2.0	2.79	1.89	1.0	1.86	**0.96**	49%
AES	6800	25124	2.0	0.64	0.72	1.0	**0.43**	0.51	33%
SHA-1	37300	24166	2.0	1.82	1.39	1.0	1.21	**0.78**	44%
SHA-256	90825	42029	2.0	2.05	1.56	1.0	1.37	**0.87**	44%

Table 1. Comparison with other garbling schemes on some circuit examples from [ST12] in terms of communication complexity. The fleXOR scheme used is based on GRR1 and thus a "safe" topological ordering is assumed. The number in each cell shows the amortized number of ciphertext per gate that need to be sent. We ignore the inversion gates, as they can be pulled inside other kind of gates. The "Saving" column is computed against the previously best solution.

	# of Gates		Private			**Privacy-free**	
Circuit	AND	XOR	GRR2	free-XOR	fleXOR	**GRR1/free-XOR/fleXOR**	Saving
DES	18124	1340	4.0/1.0	3.72/0.93	3.78/0.96	**2.79/0.93**	25%/0%
AES	6800	25124	4.0/1.0	0.85/0.21	1.44/0.51	**0.64/0.21**	25%/0%
SHA-1	37300	24166	4.0/1.0	2.43/0.61	2.78/0.78	**1.82/0.61**	25%/0%
SHA-256	90825	42029	4.0/1.0	2.73/0.68	3.11/0.87	**2.05/0.68**	25%/0%

Table 2. Comparison with other garbling schemes on some circuit examples from [ST12] in terms of computational overhead. The fleXOR scheme used is based on a "safe" topological ordering. The number in each cell shows the amortized number of calls to a KDF per gate that the constructor/evaluator need to perform. (The evaluator always performs 1 KDF evaluation for non-free gates.) Note that we do not count the non cryptographic operations in this table (polynomial interpolation in GRR2, XOR of strings in all others). The "Saving" column is computed against the previously best solution.

In Table 1 and Table 2 we benchmark our garbling scheme against the best previous garbling schemes, on a number of circuits that we think are relevant for the zero-knowledge application that we have in mind e.g., proving "I know a secret x s.t., $y = \mathrm{SHA}(x)$" for a y known to both the prover and the verifier.

The circuits used are due to Smart and Tillich and are publicly available [ST12]. Note however that the numbers in our tables depend on the actual circuits being used, meaning that it might be possible to find different circuits that compute the same functions but that are more favorable to one or another garbling scheme. Finding such circuits requires non-trivial heuristics and manual work (e.g., [BP12]), as there are evidences that finding such circuits is computationally hard [Fin14, KMR14].

Still, no previous garbling scheme performs better than *all* of our proposed schemes, therefore while the actual saving factor might change, one of our schemes will always outperforms the rest.

2 Preliminaries and Definitions

To keep the paper self-contained, we include the definitions for garbling schemes from [BHR12, BHKR13] as well as a high-level description of the [JKO13] zero-knowledge protocol. In this section we define the garbling scheme and we refer the reader to Appendix A for a description of the zero-knowledge protocol.

2.1 Notation

Let $\mathbb{N} = \{1, 2, \dots\}$ be the natural numbers, excluding 0. We use $[x]$ for $x \in \mathbb{N}$ as a shorthand to denote the set $\{1, 2, \dots, x\}$ and $|\cdot|$ as a shorthand for the cardinality of a set or amount of bits in a string. If S is a set we use $x \in_R S$ to denote that x is a uniformly random sampled element from S. We let $\operatorname{poly}(\cdot)$ denote any polynomial of the argument.

Regarding variable names we let $k \in \mathbb{N}$ be the security parameter and call a function $\operatorname{negl} : \mathbb{N} \to \mathbb{R}^+$ negligible if for a big enough k it holds that $\operatorname{negl}(k) < 1/\operatorname{poly}(k)$. In general we use $\operatorname{negl}(\cdot)$ to denote any negligible functions.

We let $L \subset \{0,1\}^*$ be an arbitrary language in NP and M_L be the language verification function, i.e., for all $y \in L$ there exists a string $x \in \operatorname{poly}(|y|)$ s.t. $M_L(x, y) = \texttt{accept}$ and for all $y \notin L$ and $x \in \{0,1\}^*$ we have $M_L(x, y) = \texttt{reject}$.

2.2 Defining Our Garbling Scheme

We start by considering a plain description of a Boolean circuit with a single output bit, consisting of Boolean gates having arbitrary fan-in. This can be used to compute a Boolean function. The description is closely related to the ones in [BHR12, JKO13], but generalized to support gates with arbitrary fan-in along non-oblivious gate evaluation.

Let f be a description of such a circuit, taking $n \in \mathbb{N}$ bits as input and consisting of $q \in \mathbb{N}$ internal gates. We let $r = n+q$ be the number of wires in the circuit and specifically define $\mathsf{inputWires} = [n]$, $\mathsf{Wires} = [n+q]$, $\mathsf{outputWire} = n + q$ and $\mathsf{Gates} = [n + 1, n + q]$, where $\mathsf{inputWires}$ represent the set of input wires, $\mathsf{outputWire}$ represents the output wire, Gates represents the set of Boolean gates of arbitrary fan-in and Wires the set of all wires in the circuit.

Next we let I be a function mapping each element of Gates to an integer describing the fan-in of that gate, i.e., $I : \mathsf{Gates} \to \mathbb{N}$. We let W be a function mapping an element of Gates, along with an integer i (representing a gate's i'th input wire) to an element in Wires. When calling W on some $g \in \mathsf{Gates}$ we require that the i'th input wire is in $[I(g)]$, otherwise we return $\perp$. Thus, the signature for the method is $W : \mathsf{Gates} \times \mathbb{N} \to \{\mathsf{Wires} \backslash \mathsf{outputWire}\}^* \cup \{\perp\}$. We further require that $W(g, i) < W(g, i + 1) < g$ for all $g \in \mathsf{Gates}$ and $i \in [I(g) - 1]$ in order to avoid circularities in the circuit description.

Finally, we let G be a function taking as input an element of Gates along with an array of bits and returning a single bit or $\perp$. That is, $G : \mathsf{Gates} \times \{0,1\}^* \to \{0,1\} \cup \{\perp\}$. Specifically G is a description of the functionality of each gate in the circuit along with a short-circuit features such that $\perp$ is returned if the amount of elements in the binary input vector is not equal to the integer returned by I when queried on the same gate index. More formally $G\left(g, \{b_i\}_{i \in [I(g)]}\right) \in \{0,1\}$ for all $g \in \mathsf{Gates}$, $b_i \in \{0,1\}$ and $\perp$ otherwise. Sometimes we abuse notation and simply write $G(g, b)$ if $g \in \mathsf{Gates}$ and $b \in \{0,1\}^m$ when $I(g) = m$. We sometimes abuse notation and say $G(g,\cdot) = \mathrm{NAND}$ or $G(g,\cdot) = \mathrm{XOR}$ if the truth table of constructed from G is the truth table of a NAND, respectively, XOR gate.

Finally we combine all these functions and variables in f by letting $f = (n,q,I,W,G)$. However, we sometimes abuse notation and view f as a black box Boolean function, i.e., $f : \{0,1\}^n \to \{0,1\}$.

With this plain description of a Boolean circuit in hand we define a *verifiable* projective garbling scheme by a tuple

$$\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}, \mathsf{Ve})$$

such that:

- $\mathsf{Gb}(1^k, f) \to (F,e,d)$ is a randomized algorithm that takes as input a security parameter 1^k and a description of a Boolean function $(n,q,I,W,G) \leftarrow f$ under the constraint that $n = \mathrm{poly}(k)$, $n \geq k$ and $|f| = \mathrm{poly}(k)$. The function outputs a triple (F,e,d) representing a garbled circuit (F), input encoding information (e) and output decoding information (d).
- $\mathsf{En}(e,x) \to X$ is a deterministic function that uses the input encoding information e to map an input x to a *garbled input* X. We say a scheme is *projective* if $e = \left(\left\{X_i^0, X_i^1\right\}_{i\in[n]}\right)$ and the garbled input X is simply $\{X_i^{x_i}\}_{i\in[n]}$. In this paper we are only interested in projective schemes and therefore we do not use the En function explicitly.
- $\mathsf{Ev}(F,X,x) \to Z$ is a deterministic functionality that produces an encoded output Z by evaluating a garbled circuit F on an encoded input X. We assume that for fixed F, the evaluation can output at most two values Z^0 and Z^1.
- $\mathsf{De}(d,Z) \to z$ is a deterministic functionality that, using the string d, decodes the encoded output Z into a plaintext bit, z. We are only interested in whether $z = 1$ (i.e., the NP relation accepts), therefore we let $d = Z^1$ and $\mathsf{De}(d,Z')$ outputs $z = 1$ if $Z \overset{?}{=} Z^1$ and $z = 0$ otherwise.
- $\mathsf{ev}(f,x) \to b$ is a deterministic functionality that evaluates the plain function described by f on some input x, i.e., $\mathsf{ev}(f,x) = f(x)$.
- $\mathsf{Ve}(F,f,e) \to b$ is a deterministic functionality that on input garbled circuit F, a description of a Boolean function f and the input encoding information $e = \left\{X_i^0, X_i^1\right\}_{i\in[n]}$ outputs 1 if the garbled circuit F computes the functionality f. Otherwise the functionality outputs 0.

We now list a number of properties that we require from a garbling scheme and refer to [BHR12, JKO13] for a detailed explanation of these definitions.

The following definition says that a correct evaluation of a correct garbling gives the right output.

Definition 1 (Correctness). *Let $\mathcal{G}$ be a verifiable projective garbling scheme described as above. We say that $\mathcal{G}$ enjoys* correctness *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all $x \in \{0,1\}^n$ s.t. $f(x) = 1$ the following probability*

$$\Pr\left(\mathsf{Ev}\left(F, \{X_i^{x_i}\}_{i\in[n]}, x\right) \neq Z^1 : \left(F, \left\{X_i^0, X_i^1\right\}_{i\in[n]}, Z^1\right) \leftarrow \mathsf{Gb}\left(1^k, f\right)\right)$$

is negligible in k.

The following definition says that from a correct garbling of an input and a function outputting 0 on that input, you cannot find the decoding information for output 1, i.e., Z^1.

Definition 2 (Authenticity). *Let $\mathcal{G}$ be a verifiable projective garbling scheme described as above. We say that $\mathcal{G}$ enjoys* authenticity *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all inputs $x \in \{0,1\}^n$ s.t. $f(x) = 0$ and for any probabilistic polynomial time (PPT) $\mathcal{A}$, the following probability:*

$$\Pr\left(\mathcal{A}\left(f, x, F, \{X_i^{x_i}\}_{i\in[n]}\right) = Z^1 : \left(F, \left\{X_i^0, X_i^1\right\}_{i\in[n]}, Z^1\right) \leftarrow \mathsf{Gb}\left(1^k, f\right)\right)$$

is negligible in k.

The following definition says that for a malicious garbled circuit, if it passes the verification algorithm, then on all garbled inputs of x such that $f(x) = 1$, the evaluation algorithm outputs the same value Z^1, and hence leaks no information about x except that $f(x) = 1$. Furthermore, from such a garbled circuit one can extract this unique Z^1.

Definition 3 (Verifiability). *Let $\mathcal{G}$ be a verifiable projective garbling scheme described as above. We say that $\mathcal{G}$ enjoys* verifiability *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all $x, x' \in \{0,1\}^n$ with $x \neq x'$ and $f(x) = f(x') = 1$ and for all PPT $\mathcal{A}$ the probability:*

$$\Pr\left(\mathsf{Ev}\left(F, \{X_i^{x_i}\}_{i\in[n]}, x\right) \neq \mathsf{Ev}\left(F, \left\{X_i^{x'_i}\right\}_{i\in[n]}, x'\right) : \begin{array}{l} \mathsf{Ve}\left(F, f, \{X_i^0, X_i^1\}_{i\in[n]}\right) = 1 \\ \left(F, \{X_i^0, X_i^1\}_{i\in[n]}\right) \leftarrow \mathcal{A}\left(1^k, f\right) \end{array}\right)$$

is negligible in k. In addition, we require the existence of a expected polynomial time algorithm Ext *s.t., for all x satisfying $f(x) = 1$ the probability:*

$$\Pr\left(\mathrm{Ext}\left(F, \{X_i^0, X_i^1\}_{i\in[n]}, x\right) \neq \mathsf{Ev}\left(F, \{X_i^{x_i}\}_{i\in[n]}, x\right) : \begin{array}{l} \mathsf{Ve}\left(F, f, \{X_i^0, X_i^1\}_{i\in[n]}\right) = 1 \\ \left(F, \{X_i^0, X_i^1\}_{i\in[n]}\right) \leftarrow \mathcal{A}(1^k, f) \end{array}\right)$$

is negligible in k.

Intuitively, Definition 3 says that even a malicious constructor cannot create circuits that are successfully verifiable (i.e., make Ve output 1) and at the same time can output different values as a function of the evaluator's input x, as long as $f(x) = 1$. Jumping ahead, this is going to guarantee that the verifier cannot distinguish between different witnesses used by the prover.

Moreover, we require that the input of the testing function is enough to extract the secret in polynomial time. Note that this is trivial when it is easy to find an x s.t., $f(x) = 1$ but non-trivial otherwise. Jumping ahead, in the setting of zero-knowledge protocols this extra guarantee enables a simulator to extract the secret Z from the input of the malicious verifier to the oblivious transfer protocol, and it is therefore crucial to prove zero-knowledge protocols. Intuitively, this is because this requirement ensures that the verifier already knows the (unique) secret Z when he sends the garbled circuit to the prover, and therefore the verifier is not learning any information when he receives back the secret Z as the output of the circuit evaluation by the prover (given that the check passes).

Finally, combining these definitions we get a definition of a secure verifiable, projective and privacy-free garbling scheme.

Definition 4 (Privacy-free Garbling Scheme). *Let $\mathcal{G}$ be a verifiable projective garbling scheme described as above. If this scheme enjoys* correctness, authenticity *and* verifiability *in accordance with Def. 1, Def. 2 and Def. 3 respectively then $\mathcal{G}$ is a secure privacy-free garbling scheme.*

2.3 Key Derivation Function

We are going to use a "compressing" key derivation function $\mathsf{KDF} : \{0,1\}^* \to \{0,1\}^k$ mapping an arbitrary binary string to a pseudorandom string of k bits. The applications of the function will be of the form $K_o = \mathsf{KDF}(K_1, \ldots, K_c; id)$ for some $c \in \mathbb{N}$, where $K_i \in \{0,1\}^k$ is a wire key and $id \in \{0,1\}^*$ is a unique label or tweak.

We need a notion of security where the adversary cannot compute the output of the key derivation function except if he can do so trivially because he knows the entire input. Specifically we let keys be fresh uniformly random values, derived or linear combinations of other keys, and id be publicly known. We require that the adversary cannot guess a key derived from at least one uniformly random key, "uncompromised" derived key or linear combination of keys where at least one is "uncompromised". An uncompromised derived key is one that was derived from at least one uniformly random key, uncompromised derived key or linear combination where at least one key in the combination was uncompromised. We allow the adversary to compromise keys by leaking them and construct new keys through linear combinations or key derivations. Furthermore, we call a (potential) key compromised if the leaked keys allow to determine the key, in which case the adversary can trivially compute it. More precisely:

Definition 5 (Game KDF). *Let $\mathcal{A}$ be any PPT adversary and consider the following game*

Initialize: *Let $\mathsf{ID} \leftarrow \emptyset$ be a set of identifiers used by the adversary and let $\mathsf{LEAK} \leftarrow \emptyset$ be the set of identifiers that should be leaked.*

Query: *Let $\mathcal{A}$ make an arbitrary amount of calls, in any combination, to the following methods:*

- **Fresh key:** *If $\mathcal{A}$ outputs $(\texttt{fresh key}, id \notin \mathsf{ID})$, then sample $K_{id} \in_R \{0,1\}^k$ and store (id, K_{id}) and let $\mathsf{ID} \leftarrow \mathsf{ID} \cup \{id\}$.*
- **Linear:** *If $\mathcal{A}$ outputs $(\texttt{linear}, id_0 \notin \mathsf{ID}, id_1, \ldots, id_m)$ where $id_i \in \mathsf{ID}$ for all $i \in [m]$, then compute $K_{id_0} \leftarrow \bigoplus_{i=1}^m K_{id_i}$, store (id_0, K_{id_0}), and let $\mathsf{ID} \leftarrow \mathsf{ID} \cup \{id_0\}$.*
- **Derive:** *If $\mathcal{A}$ outputs $(\texttt{derive}, id_0 \notin \mathsf{ID}, id_1, \ldots, id_m)$ where $id_i \in \mathsf{ID}$ for all $i \in [m]$, then compute $K_{id_0} \leftarrow \mathsf{KDF}(K_{id_1}, \ldots, K_{id_m}; id_0)$, store (id_0, K_{id_0}) and let $\mathsf{ID} \leftarrow \mathsf{ID} \cup \{id_0\}$.*
- **Leak:** *If $\mathcal{A}$ outputs $(\texttt{leak}, id \in \mathsf{ID})$ set $\mathsf{LEAK} = \mathsf{LEAK} \cup \{id\}$.*

End: *When $\mathcal{A}$ outputs $(\texttt{end})$ then return the set $\{K_i\}_{i \in \mathsf{LEAK}}$ to $\mathcal{A}$.*

Guess: *When $\mathcal{A}$ outputs $(\texttt{guess}, id^*, K^*)$ for $id^* \in \mathsf{ID}$, then the adversary wins if $K^* = K_{id^*}$ and id^* was not compromised, i.e., if $id^* \notin \mathsf{COMP}$, see below.*

We define the set COMP of IDs of compromised keys iteratively as follows: Define a linear system LIN over formal variables $\mathtt{X}_{id}$ and $\mathtt{c}_{id}$ for $id \in \mathsf{ID}$. For each linear query $(\texttt{linear}, id_0, id_1, \ldots, id_m)$ add the equation $\oplus_{i=1}^m \mathtt{X}_{id_i} = \mathtt{X}_{id_0}$ to LIN. For each leakage command $(\texttt{leak}, id \in \mathsf{ID})$, add the equation $\mathtt{X}_{id} = \mathtt{c}_{id}$ to LIN. In the following we call an identifier id^* determined in LIN *if the linear system LIN allows to write $\mathtt{X}_{id^*}$ as a linear combination of the variables $\mathtt{c}_{id}$ for $id \in \mathsf{ID}$. We use $\mathrm{Det}(\mathsf{LIN})$ to denote the set of identifiers that are determined in LIN. We call id^** derivable in LIN *if there was a command $(\texttt{derive}, id^*, id_1, \ldots, id_m)$ and $id_i \in \mathrm{Det}(\mathsf{LIN})$ for each $i \in [m]$. We use $\mathrm{Der}(\mathsf{LIN})$ to denote the set of identifiers that are derivable in LIN. We define an extension $\mathsf{LIN}' = \mathrm{Ext}(\mathsf{LIN})$ by letting LIN' be LIN but with the equation $\mathtt{X}_{id^*} = \mathtt{c}_{id^*}$ added for each $id^* \in \mathrm{Der}(\mathsf{LIN})$. Define $\mathsf{LIN}_0 = \mathsf{LIN}$ and $\mathsf{LIN}_{i+1} = \mathrm{Ext}(\mathsf{LIN}_i)$. There are finitely many variable, so this has a fixed index j such that $\mathsf{LIN}_{j+1} = \mathrm{Ext}(\mathsf{LIN}_j)$. We let $\mathsf{COMP} = \mathsf{LIN}_j$.*

We use $\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k)$ to denote the probability that $\mathcal{A}$ wins the game. Using this game we define the notion of a secure key derivation function.

Definition 6 (Secure Key Derivation Function). *We say that a $\mathsf{KDF}(\cdot)$ is secure if the advantage of any PPT adversary $\mathcal{A}$ playing the KDF game is negligible in k, i.e.*

$$\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) \leq \mathrm{negl}(k)$$

for some negligible function $\mathrm{negl}(\cdot)$.

It can be proven using standard techniques that a random oracle is a secure KDF in the above sense. More precisely:

Theorem 1. *If $\mathsf{KDF}(\cdot)$ is modeled by a random oracle with k bits output then for any PPT $\mathcal{A}$ it holds that* $\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) \leq \mathrm{negl}(k)$ *for some negligible function* $\mathrm{negl}(\cdot)$.

For completeness a proof can be found in Appendix B.

We leave as future work the investigation of which exact computational assumptions are required for implementing our different garbling schemes: while it is clear that the freeXOR and fleXOR variant require strong notion of security (security under related-key attack and a flavour of circular security), it seems that the GRR1 variant could be instantiated using standard security notions.

3 Our Privacy-free Garbling Schemes

In this section we present our novel garbling schemes. Our schemes support gates with arbitrary fan-in, but as a warm-up we first present the garbling schemes for gates with fan-in 2 using GRR1 or GRR2 with free-XOR. Both allow to garble every Boolean gate with fan-in 2 using only 3 calls to the KDF for non-XOR gates and require no calls to the KDF for XOR gates.

Our first scheme has communication complexity of k bits per gate while our second garbling scheme is compatible with "free-XOR", but requires communication complexity of $2k$ bits for non-XOR gates.

Afterwards we present our two schemes for gates with arbitrary fan-in along with a scheme that supports the recent fleXOR approach [KMR14].

3.1 Warm-up

To simplify notation and give the intuition of our scheme we here only describe how to garble/evaluate a single NAND or XOR gate. We call the input keys to the left wire of a gate L^0, L^1, the input keys to the right wire R^0, R^1 and the output keys O^0, O^1. All these values are elements of $\{0,1\}^k$.

Again we point out that in contrast with general garbled circuits, in our case if the circuit evaluator has two keys L^a, R^b, he knows the corresponding bits a, b.

First consider a NAND gate with GRR1:

Garbling a GRR1 NAND Gate: Let $O^0 = \mathsf{KDF}\left(L^1, R^1\right)$ and $O^1 = \mathsf{KDF}\left(L^0\right)$. Compute $C = \mathsf{KDF}\left(R^0\right) \oplus O^1$ and output C.

Evaluating a GRR1 NAND Gate: To evaluate on input L^a, R^b, if $a = b = 1$ then output $O^0 = \mathsf{KDF}\left(L^1, R^1\right)$ otherwise, if $a = 0$ compute $O^1 = \mathsf{KDF}\left(L^0\right)$. Finally, if $b = 0$ compute $O^1 = C \oplus \mathsf{KDF}\left(R^0\right)$.

It should be clear that the scheme is correct. The intuition of authenticity is that if the evaluator only knows one input key for each wire, he can only learn one output key unless he can guess the output of KDF on an input he does not know. Thus the evaluator does not learn any input key that he did not have before, unless he can invert KDF. Next consider a XOR gate:

Garbling a GRR1 XOR Gate: Let $O^0 = L^0 \oplus R^0$ along with $O^1 = L^0 \oplus R^1$. Finally output $C = L^0 \oplus L^1 \oplus R^0 \oplus R^1$.

Evaluating a GRR1 XOR Gate: On input L^a, R^b if $a = 0$ then output $O^{a \oplus b} = L^a \oplus R^b$. Otherwise compute and return $O^{a \oplus b} = C \oplus L^a \oplus R^b$.

Again, it should be clear that the scheme is correct. The authenticity intuitively follows from the fact that the evaluator can only learn the XOR of two unknown keys which will not help decrypting the next gate.

Now consider how to achieve the same, while allowing support for free-XOR gates (and in turn GRR2). In this scheme there is a global difference Δ s.t., for all wires w in a garbled circuit, the key pair X_w^0, X_w^1 satisfies $X_w^0 \oplus X_w^1 = \Delta$.

Garbling a GRR2 NAND Gate: Let $O^0 = \mathsf{KDF}\left(L^1, R^1\right)$. This defines $O^1 = O^0 \oplus \Delta$ as well. Let $C_L = \mathsf{KDF}\left(L^0\right) \oplus O^1$ and $C_R = \mathsf{KDF}\left(R^0\right) \oplus O^1$. Finally output $\{C_L, C_R\}$.

Evaluating a GRR2 NAND Gate: To evaluate on input L^a, R^b, if $a = b = 1$ then output $O^0 = \mathsf{KDF}\left(L^1, R^1\right)$ otherwise, if $a = 0$ output $O^1 = \mathsf{KDF}\left(L^0\right) \oplus C_L$ otherwise output $O^1 = \mathsf{KDF}(R^0) \oplus C_R$.

Next consider a XOR gate:

Garbling a free-XOR Gate: Let $O^0 = L^0 \oplus R^0$. This defines $O^1 = O^0 \oplus \Delta$ as well. Output nothing.

Evaluating a free-XOR Gate: On input L^a, R^b, output $O^{a \oplus b} = L^a \oplus R^b$.

Again correctness should be clear and authenticity for NAND gates follow from the same argument as for GRR1 NAND gates, whereas authenticity follows from the security of free-XOR, i.e. that it is hard to learn Δ, unless one is given both keys on some wire.

		Garb.	Eval.	Size
GRR1	NAND	$m+1$	1	$k(m-1)$
	XOR	0	0	$k(m-1)$
Free-XOR	NAND	$m+1$	1	km
	XOR	0	0	0
FleXOR	NAND	$m+1$	1	$k(m-1)$
	t-XOR	0	0	kt

Table 3. Exact performances of our privacy-free garbling scheme. The "Garb." and "Eval." column state the number of calls to a KDF required for garbling and evaluation respectively, as a function of the gate fan-in m. The column "Size" states the number of bits added to the garbled circuit for each gate. We only report the fleXOR variant based on "Safe" wire ordering.

3.2 Generalization Intuition

We now consider how our approaches generalizes to gates with arbitrary fan-in.

NAND gates. Consider a NAND gate with fan-in m, call this gate g. Recall that for this gate the output bit $b_g = 0$ should occur exactly if all the input bits are equal to 1, $b_1 = b_2 = \ldots = b_m = 1$. This means that we can define the output key representing bit 0 directly from these: If we denote the key on input wire i by $X_i^{b_i}$, then the output 0-key is computed as

$$X_g^0 = \mathsf{KDF}\left(X_1^1, X_2^1, \ldots, X_m^1\right) \ .$$

Now, if we are not using a free-XOR scheme we define the 1-output key to be $X_g^1 = \mathsf{KDF}\left(X_1^0\right)$. Then the entries in the garbled computation table is as follows:

$$\left\{C_i = X_g^1 \oplus \mathsf{KDF}\left(X_i^0\right)\right\}_{i=2}^m \ .$$

When we are using a free-XOR scheme we have another entry in the garbled computation table since the output key X_{o}^1 needs to meet the constraint $X_g^1 = X_g^0 \oplus \varDelta$ and thus we cannot define it to simply be $\mathsf{KDF}\left(X_1^0\right)$. However, similarly to the scheme above that does not use free-XOR we use the KDF applied to the first input key (which we have not used to hide anything in the scheme above) to hide X_g^1. We let the rest of the table remain as before and thus the whole garbled computation table is computed as follows:

$$\left\{C_i = X_g^1 \oplus \mathsf{KDF}\left(X_i^0\right)\right\}_{i=1}^m \ .$$

We describe the evaluation. Call the input keys $X_1^{b'_1}, X_2^{b'_2}, \ldots, X_m^{b'_m}$. If $b'_i = 1$ for all $i \in [m]$ then the output is $X_g^0 = \mathsf{KDF}\left(X_1^1, X_2^1, \ldots, X_m^1\right)$. Otherwise find the first value of i for which $b'_i \neq 1$ and output $X_g^1 = C_i \oplus \mathsf{KDF}\left(X_i^0\right)$, except if $i = 1$ and we do not use a free-XOR garbling scheme, in which case the output is $X_g^1 = \mathsf{KDF}\left(X_1^0\right)$.

XOR gates. To garble XOR gates (when we are not using the free-XOR method), we define the output 0-key from information based on all the input 0-keys. Specifically as

$$X_g^0 = X_1^0 \oplus X_2^0 \oplus \cdots \oplus X_m^0 = \bigoplus_{i=1}^m X_i^0 \ .$$

In a similar manner we define the output 1-key from information based on the first input 1-key and all the other input 0-keys, that is

$$X_g^1 = X_1^1 \oplus X_2^0 \oplus \cdots \oplus X_{m-1}^0 \oplus X_m^0 = X_1^1 \oplus \left(\bigoplus_{i=2}^{m} X_i^0 \right) \ .$$

Let b_i, for all $i \in [m]$ be the input bits at evaluation time and $b_g = b_1 \oplus \ldots \oplus b_m$ be the output of that gate. It might be the case that the $b_1 \neq 1$ and that there are other j s.t., $b_j = 1$. So we let the garbled computation table consist of information which makes it possible for the evaluator to compute the right output key in any such situation. Specifically we define the table as the following set:

$$\left\{ C_i = X_i^0 \oplus X_i^1 \oplus X_1^0 \oplus X_1^1 \right\}_{i=2}^{m} \ .$$

It is clear that, for any $j \neq 1$

$$\left(\bigoplus_{i \in [m]} X_i^{b_i} \right) \oplus C_j = X_1^{b_1 \oplus 1} \oplus X_j^{b_j \oplus 1} \bigoplus_{j \neq i > 2} X_i^{b_i}$$

Thus by XORing all the C_i's for which $b_i = 1$ we obtain

$$\left(\bigoplus_{i \in [m]} X_i^{b_i} \right) \oplus \left(\bigoplus_{i : b_i = 1} C_i \right) = X_i^{b_1 \oplus \ldots \oplus b_m} \oplus \left(\bigoplus_{i : b_i = 0} X_i^0 \right) \oplus \left(\bigoplus_{i : b_i = 1} X_i^{1 \oplus 1} \right) = X_g^{b_g}$$

Other gates. It is easy to see that our garbling scheme can be applied also to few other kind of gates such as AND, (N)OR, XNOR etc., also in the case of high fan-in (by using a different partitioning of the inputs and relabelling the outputs) but it cannot be used in for generic, “unstructured” gates.

Using high fan-in gates. Note that our garbling scheme is favourable for gates with high fan-in, since the complexity shown in Table 3 (both in terms of communication and computational complexity) only grows linearly with the gate fan-in, while a straightforward use of standard garbled circuit leads in a exponential blow-up in the gate fan-in. Even when comparing the garbling of a gate with fan-in m to a circuit implementing the same functionality (e.g., a tree of fan-in 2 ANDs to implement an AND with fan-in m) our scheme is still favourable. Depending on the garbling scheme we can save a factor 2-3 in terms of computation for the garbler and communication. In addition, the evaluator has an overhead of $\log(m)$ when evaluating the circuit (versus a single call to the KDF in our case).

3.3 Formal specification

We describe our gate garbling schemes in the same notation as [BHR12], but with some changes in order to reflect that we only require privacy, only assume one bit output and that we support gates of arbitrary fan-in. The specification of the garbling scheme is given in Fig. 1 and the realizations for individual gate garbling is given in Fig. 2 and Fig. 3, depending on whether or not one uses free-XOR or GRR1.

To enhance understanding we describe each step of these procedures.

The Garbling Scheme. The first method, Gb, constructs a garbled circuit, F, along with information, e, to encode a binary string as garbled input to this garbled circuit and information, d, to check if the output of an evaluation of the garbled circuit has the semantic value 1. The method takes as input a security parameter 1^k and a description of the Boolean function to be computed, f. The format of the function description should be in accordance with the description given in Section 2.2, and thus can be viewed directly as a

```
$\mathsf{Gb}\left(1^k, f\right) \to (F, e, d)$
    1. Set $(n, q, I, W, G) \leftarrow f$ and $\left\{X_i^0, X_i^1\right\}_{i \in [n]} \leftarrow \mathsf{InKeys}(n, k)$.
    2. For each $g \in [n+1, n+q]$ set $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g,i)$ for all
       $i \in \{0,1\}^m$ and set $\left\{\left(X_g^0, X_g^1\right), P[g]\right\} \leftarrow \mathsf{Garb}\left(g, G', \left\{X_{W(g,i)}^0, X_{W(g,i)}^1\right\}_{i \in [m]}\right)$.
    3. Set $F \leftarrow (n, q, I, W, G, P)$, $e \leftarrow \left\{X_i^0, X_i^1\right\}_{i \in [n]}$ and $d \leftarrow X_{n+q}^1$.
    4. Finally return $(F, e, d)$.
$\mathsf{En}(e, x) \to X$
    1. Set $\left\{X_i^0, X_i^1\right\}_{i \in [n]} \leftarrow e$.
    2. Then set $X \leftarrow \{X_i^{x_i}\}_{i \in [n]}$ and return $X$.
$\mathsf{De}(d, Z) \to b$
    1. If $d = Z$ then output 1 otherwise output 0.
$\mathsf{Ev}(F, X, x) \to Z$
    1. Set $(n, q, I, W, G, P) \leftarrow F$ and for all $i \in [n]$ set $w_i = x_i$ and define $Q = \{w_i\}_{i \in [n]}$.
    2. For each $g \in [n+1, n+q]$ let $m = I(g)$ and add $w_g = G\left(g, \left\{w_{W(g,i)}\right\}_{i \in [m]}\right)$ to the set $Q$.
    3. Now for each $g \in [n+1, n+q]$ let $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g,i)$ and
       $w' \in \{0,1\}^m$ s.t. ${w'}_i = w_{W(g,i)}$ for all $i \in [m]$ and set $X_g \leftarrow \mathsf{Eval}\left(g, G', w', \left\{X_{W(g,i)}\right\}_{i \in [m]}, P[g]\right)$.
    4. Return $X_{n+q}$.
$\mathsf{ev}(f, x) \to b$
    1. Set $(n, q, I, W, G) \leftarrow f$ and for all $i \in [n]$ set $w_i = x_i$ and define $Q \leftarrow \{w_i\}_{i \in [n]}$.
    2. For each $g \in [n+1, n+q]$ let $m = I(g)$ and add $w_g = G\left(g, \left\{w_{W(g,i)}\right\}_{i \in [m]}\right)$ to the set $Q$.
    3. Finally return $w_{n+q}$.
$\mathsf{Ve}(F, f, e) \to b$
    1. Set $(n, q, I, W, G, P) \leftarrow F$, $(n', q', I', W', G') \leftarrow f$ and $\left\{X_i^0, X_i^1\right\}_{i \in [n]} \leftarrow e$.
    2. If $n \neq n'$, $q \neq q'$, $I \neq I'$, $W \neq W'$ or $G \neq G'$ output 0.
    3. For each $g \in [n+1, n+q]$ let $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g,i)$ for all
       $i \in \{0,1\}^m$ and set $\left\{\left(X_g^0, X_g^1\right), \bar{P}[g]\right\} \leftarrow \mathsf{Garb}\left(g, G', \left\{X_{W(g,i)}^0, X_{W(g,i)}^1\right\}_{i \in [m]}\right)$.
    4. If for any $g \in [n+1, n+q]$ we have $\bar{P}[g] \neq P[q]$ output 0, otherwise output 1.
```

Figure 1. Privacy-free Garbling

Boolean circuit. In step 1 the algorithm chooses two keys for each of the n input bits to f, in accordance with the specific type of garbling scheme used. These are the 0-, respectively, 1-input keys. Step 2 involves iteratively constructing each of the q garbled gates of the circuit, along with the two output keys needed for each of these gates. It is done by first using I to decide the fan-in of a given gate, then using G to find the specific functionality of the given gate. Finally the input keys for that gate (which have already been constructed) is loaded using W and all the information is passed to the gate garbling method Garb. In step 3 the garbled circuit, F, is set to include all the information of f along with the garbled computation table returned by Garb in the previous step for all the gates in the circuit. These tables are called P. Furthermore, the encoding information e is set to be the two keys for each input wire and the decoding information d is set to be the output 1-key of the final gate in the circuit. In the last step, the garbled circuit F, the input encoding information, e, and decoding information, d, is returned.

The second method, En, constructs an ordered set of input keys to a garbled circuit, X. It takes as input the encoding information e along with a binary string x of length n representing the input to the garbled circuit. In the first step the method parses e as n ordered pairs of keys. In step 2 the functionality returns an ordered subset of the keys. In particular if the i'th bit of x is 0 then the i'th element in the ordered set is the i'th 0-key, otherwise it is the i'th 1-key.

The third method, De, evaluates whether some value, Z, is equal to the output 1-key of a garbled circuit, d. It takes as input the decoding information of a garbled circuit, d, along with a potential output key, Z. The method only has one step which checks if $d = Z$ and returns 1 if that is true, otherwise it returns 0.

The fourth method, Ev, evaluates a garbled circuit, F, and returns the output key of the final gate as a result of this evaluation, Z. It takes as input a garbled circuit F, and an ordered set of input keys, X, along with a binary vector x where the i'th bit represents the semantic value of the i'th input key. In step 1 the method parses the information stored in the garbled circuit F and defines an ordered set of bits, Q, which represents the bits on each each wire in the garbled circuit. Initially this set only includes the bits of the input wires. Step 2 iteratively evaluates the garbled circuit one gate at a time. It first finds the fan-in of a given gate using I and then evaluates the gate in plain using the set Q along with the gate description G. After evaluating the gate in plain it updates Q to contain the output bit of the given gate. Thus at the end Q contains the expected bit on each wire given the garbled circuit F and the binary input x. In step 3 the method proceeds to evaluate each garbled gate iteratively. Again it uses I to learn the fan-in for a given gate, it uses G to decode the specific functionality of the gate and the elements of Q to find the semantic meaning of the keys supposed to be input to the garbled gate. Using this information, along with the garbled computation table of the gate, P, it calls Eval to evaluate the garbled gate and stores the output key which the method returns. Finally it returns the output key of the final gate in the garbled circuit.

The fifth method, ev, evaluates the Boolean functionality f in plain using a binary input vector x. It returns a bit being the value $f(x)$. In Step 1 it parses the functionality f and constructs a set Q which represents the bit on each wire in the circuit. Initially this set only contains the bits on the input wires, exactly as specified by x. In step 2 it iteratively evaluates each gate of the functionality. It does so by first learning the fan-in of the give gate using I and then using G with the given gate index and bits already stored in Q. It updates the set Q with the result. Finally it returns the result of evaluating the final gate in the circuit.

The sixth and last method, Ve, checks whether a garbled circuit, F, evaluates the same as some plain circuit, f, given both pairs of input keys for all wires of the garbled circuit, e. The method returns either accept or reject. It takes as input a garbled circuit F, a plain description of the circuit functionality f along with the ordered set of input keys, e. In the first step it parses the garbled circuit F and the plain function description f. Step 2 is a sanity check which verifies that the "meta" data of F and f is the same, i.e., same amount of input bits, n, the same amount of gates q, each with the same fan-in I, using the same wires, W, and computing the same functionality, G. If any of these checks fails the method outputs reject. Then step 3 iteratively constructs a new garbled circuit using Garb in the same manner as in Gb, based on the information in f. Finally in step 4 the method checks equality of each garbled computation table given in F with each of the tables generated in the previous step. If any are not equal then the method outputs reject, otherwise it outputs accept.

Gate Garbling. All of our garbling schemes have two methods: Garb and Eval. The first constructs a garbled gate, $\tilde{g}$, and two keys, $\left(X_g^0, X_g^1\right)$. It takes as input a nonce, g (gate ID), a function mapping a binary vector to a bit, G', along with a pair of input keys for each input wire to the gate. The second method reconstructs a single output key. It takes as input a nonce, g (gate ID), a function mapping a binary vector to a bit, G', a binary vector describing the bits on the input wires to the gate, w', an ordered set of input keys $\{X_i\}_{i\in[m]}$ along with an ordered set which is the garbled computation table $\tilde{g}$.[1]

3.4 Security

The scheme presented in Fig. 1 composed with Fig. 2 and Fig. 3 respectively are clearly correct. In fact, any correctly generated scheme evaluates to the correct output key with probability 1. From this it also follows that the scheme has verifiability, as we verify by regenerating the garbled gate, and hence a verified garbled

[1] Note that, as it is described, the running time of Eval depends on the particular input used. To prevent leakage of the input based on timing attacks, any implementation of Eval would need to take appropriate countermeasures, and ensure that the running time does not depend on the input used.

```
InKeys$(n,k) \rightarrow \left\{X_i^0, X_i^1\right\}_{i\in[n]}$
    1. For each $i \in [n]$ sample uniformly random $X_i^0, X_i^1 \in_R \{0,1\}^k$ and return the set $\left\{X_i^0, X_i^1\right\}_{i\in[n]}$.
Garb$\left(g, G', \left\{\left(X_i^0, X_i^1\right)\right\}_{i\in[m]}\right) \rightarrow \left\{\left(X_g^0, X_g^1\right), \tilde{g}\right\}$
    1. If $G'(\cdot) = \text{NAND}$ do as follows:
        (a) Let $X^0 = \mathsf{KDF}\left(X_1^1, X_2^1, \ldots, X_m^1; (\texttt{key}, g, 0)\right)$ and $X_g^1 = \mathsf{KDF}\left(X_1^0; (\texttt{key}, g, 1)\right)$.
        (b) Next let $C_i = X_g^1 \oplus \mathsf{KDF}\left(X_i^0; (\texttt{inte}, g, i)\right)$ for all $i \in [2,m]$ and set $\tilde{g} = \{C_i\}_{i=2}^m$.
        (c) Return $\left\{\left(X_g^0, X_g^1\right), \tilde{g}\right\}$.
    2. If instead $G'(\cdot) = \text{XOR}$ do as follows:
        (a) Let $X_g^0 = \bigoplus_{i=1}^m X_i^0$ and $X_g^1 = X_1^1 \oplus \left(\bigoplus_{i=2}^m X_i^0\right)$.
        (b) Next let $C_i = X_1^0 \oplus X_1^1 \oplus X_i^0 \oplus X_i^1$ for all $i \in [2,m]$ and set $\tilde{g} = \{C_i\}_{i=2}^m$.
        (c) Return $\left\{\left(X_g^0, X_g^1\right), \tilde{g}\right\}$.
Eval$\left(g, G', w', \{X_i\}_{i\in[m]}, \tilde{g}\right) \rightarrow \{X_g\}$
    1. If $G'(\cdot) = \text{NAND}$ do as follows:
        (a) If $w' = 1^m$ then set $X_g = \mathsf{KDF}\left(X_1, X_2, \ldots, X_m; (\texttt{key}, g, 0)\right)$. If instead $w'_1 = 0$ then set $X_g = \mathsf{KDF}\left(X_1; (\texttt{key}, g, 1)\right)$. Otherwise find the first $i \in [2,m]$ s.t. $w'_i = 0$, parse $\{C_i\}_{i=2}^m \leftarrow \tilde{g}$ and set $X_g = C_i \oplus \mathsf{KDF}\left(X_i; (\texttt{inte}, g, i)\right)$.
        (b) Return $X_g$.
    2. If instead $G'(\cdot) = \text{XOR}$ do as follows:
        (a) Parse $\{C_i\}_{i=2}^m \leftarrow \tilde{g}$.
        (b) Let $S$ be the set of $i \in \{2, m\}$ for which it is true that $w'_i = 1$.
        (c) Return $X_g = \left(\bigoplus_{i\in[m]} X_i\right) \oplus \left(\bigoplus_{i\in S} C_i\right)$.
```

Figure 2. Garbling - Without free-XOR

```
InKeys$(n,k) \rightarrow \left\{X_i^0, X_i^1\right\}_{i\in[n]}$
    1. Sample a uniformly random difference $\Delta \in \{0,1\}^k$.
    2. Then for each $i \in [n]$ sample uniformly random $X_i^0 \in_R \{0,1\}^k$ and return the set $\left\{X_i^0, X_i^0 \oplus \Delta\right\}_{i\in[n]}$.
Garb$\left(g, G', \left\{\left(X_i^0, X_i^1\right)\right\}_{i\in[m]}\right) \rightarrow \left\{\left(X_g^0, X_g^1\right), \tilde{g}\right\}$
    1. Set $\Delta = X_1^0 \oplus X_1^1$.
    2. If $G'(\cdot) = \text{NAND}$ do as follows:
        (a) Let $X_g^0 = \mathsf{KDF}\left(X_1^1, X_2^1, \ldots, X_m^1; (\texttt{key}, g, 0)\right)$ and $X_g^1 = X_g^0 \oplus \Delta$.
        (b) Next let $C_i = X_g^1 \oplus \mathsf{KDF}\left(X_i^0; (\texttt{inte}, g, i)\right)$ for all $i \in [m]$ and set $\tilde{g} = \{C_i\}_{i=1}^m$.
        (c) Return $\left\{\left(X_g^0, X_g^1\right), \tilde{g}\right\}$.
    3. If instead $G'() = \text{XOR}$ set $X_g^0 = \bigoplus_{i=1}^m X_i^0$, $X_g^1 = X_g^0 \oplus \Delta$ and return $\left\{\left(X_g^0, X_g^1\right), \perp\right\}$.
Eval$\left(g, G', w', \{X_i\}_{i\in[m]}, \tilde{g}\right) \rightarrow \{X_g\}$
    1. If $G'(\cdot) = \text{NAND}$ do as follows: If $w' = 1^m$ then set $X_g = \mathsf{KDF}\left(X_1, X_2, \ldots, X_m; (\texttt{key}, g, 0)\right)$. Otherwise find the first $i \in [m]$ s.t. $w_i{}' = 0$, parse $\{C_i\}_{i=1}^m \leftarrow \tilde{g}$ and compute and return $X_g = C_i \oplus \mathsf{KDF}\left(X_i; (\texttt{inte}, g, i)\right)$.
    2. If instead $G'(\cdot) = \text{XOR}$ return $X_g = \bigoplus_{i=1}^m X_i$.
```

Figure 3. Garbling - With free-XOR

gate is correctly generated. This takes care of the demands of Def. 1 and Def. 3 of a secure privacy-free garbling scheme, as defined in Def. 4. What remains is Def. 2: In the following we reduce this to the security of the KDF used.

Theorem 2. *If the* KDF *used in the garbling scheme of Fig. 1 composed with Fig. 2 is secure according to Def. 6, then the composed scheme enjoys authenticity according to Def. 2.*

Proof. For notational convenience we are going to focus on the case with fan-in 2. The proof idea generalizes immediately.

A NAND gate with input keys L^0, L^1 for the left wire and R^0, R^1 for the right wire and gate identifier g is garbled as follows:

$$\begin{aligned} O^1 &\leftarrow \mathsf{KDF}(L^0; (\mathtt{key}, g, 1)) \ , && (1)\\ O^0 &\leftarrow \mathsf{KDF}(L^1, R^1; (\mathtt{key}, g, 0)) \ , && (2)\\ A &\leftarrow \mathsf{KDF}(R^0; (\mathtt{inte}, g)) \ , && (3)\\ C &\leftarrow A \oplus O^1 (\text{with label } (\mathtt{garb}, g)) \ . && (4) \end{aligned}$$

The output keys are (O^0, O^1). The garbled gate is just C.

An XOR gate with input keys L^0, L^1 for the left wire and R^0, R^1 for the right wire and gate identifier g is garbled as follows:

$$\begin{aligned} O^0 &\leftarrow L^0 \oplus R^0 \ (\text{with label } (\mathtt{key}, g, 0)) \ , && (5)\\ O^1 &\leftarrow L^0 \oplus R^1 \ (\text{with label } (\mathtt{key}, g, 1)) \ , && (6)\\ C &\leftarrow L^0 \oplus L^1 \oplus R^0 \oplus R^1 \ (\text{with label } (\mathtt{garb}, g)) \ . && (7) \end{aligned}$$

The output keys are (O^0, O^1). The garbled gate is just C.

Besides this, the circuit garbling just consist of reusing the appropriate output keys as input keys to later gates. A garbled circuit F consists of, amongst other, a garbled gate for each of the q internal wires, $P = (C_{n+1}, \ldots, C_{n+q})$, in an order in which they can be evaluated. For each garbled gate C_i, let L_i^0 and L_i^1 be the corresponding keys on the left input wire, let R_i^0 and R_i^1 be the corresponding keys on the right input wire, and let O_i^0 and O_i^1 be the output keys.

We can assume without loss of generality that the last gate is the output gate. For a garbled input $X = \left\{\left(X_i^0, X_i^1\right)\right\}_{i=1}^n$ and a plaintext input $x \in \{0,1\}^n$, let $X^x = \{X_i^{x_i}\}_{i\in[n]}$ be the garbled version of x. For $i = n+1, \ldots, n+q$, let w_i be the bit we get by computing plaintext gate number i on the bits for its input wires, that is $w_i = G(i, \{W(i,1), W(i,2)\})$ in accordance with Fig. 1. This defines a *plaintext evaluation* $w = (w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+q})$. For $i = n+1, \ldots, n+q$, let $K_i = O_i^{w_i}$. This defines a *garbled evaluation* $K^x = (K_1, \ldots, K_n, K_{n+1}, \ldots, K_{n+q})$. The scheme is constructed such that from a correct garbled circuit F and X^x one can efficiently compute K^x, which in particular allows to compute $K_{n+q} = O_{n+q}^{f(x)}$. We have to prove that from a randomly generated P and X^x one cannot also efficiently compute $O_{n+q}^{1-f(x)}$. For this, it is sufficient to prove that one cannot efficiently compute $\left(i, O_i^{1-w_i}\right)$ for any $i \in [n+q]$ with non-negligible probability.

We do the proof by a simple reduction to the game KDF in Def. 5. It is easy to see that the garbling and the keys learned by the evaluator in the scheme can be computed by queries to the game KDF in such a way that all the keys $O_i^{1-w_i}$ are uncompromised. In more detail, the reduction runs as follows:

Input keys: For each $i \in [n]$ and $b \in \{0,1\}$, output $(\mathtt{fresh\ key}, (\mathtt{key}, i, b))$ to define a fresh random key $X_i^b \in_R \{0,1\}^k$. Then for each $i \in [n]$, output $(\mathtt{leak}, (\mathtt{key}, i, x_i))$ to add $X_i^{x_i}$ to the set of values to leak. Let $X^x = \{X_i^{x_i}\}_{i=1}^n$. Now for each input wire, both keys are defined in the game KDF.

Internal gates: Iteratively go through all the gates. Specifically for each $i \in [n+1, q]$ we do as follows, depending on whether or not gate i is a NAND or XOR gate:

NAND gate: Call the plaintext value on the left input wire $l_i = w_{W(i,1)}$, call the plaintext value on the right input wire $r_i = w_{W(i,2)}$, and call the plaintext value on the output wire w_i. Call the keys on these wires $\left(L_i^0, L_i^1\right)$, $\left(R_i^0, R_i^1\right)$ and $\left(O_i^0, O_i^1\right)$ respectively. Thus $\left(L_i^0, L_i^1\right) = \left(X_{W(i,1)}^0, X_{W(i,1)}^1\right)$, $\left(R_i^0, R_i^1\right) = \left(X_{W(i,2)}^0, X_{W(i,2)}^1\right)$ and $\left(O_i^0, O_i^1\right) = \left(X_i^0, X_i^1\right)$. The first four of these keys are defined in the game KDF and we are given $L_i^{l_i}$ and $R_i^{r_i}$ before our guess. We should define $\left(O_i^0, O_i^1\right)$ in the game and make sure we learn $O_i^{w_i}$ before our guess. We use $\mathtt{derive}$-commands to define $O_i^1 =$

$\mathsf{KDF}\left(L_i^0; (\mathtt{key}, i, 1)\right)$, $O_i^0 = \mathsf{KDF}\left(L_i^1, R_i^1; (\mathtt{key}, i, 0)\right)$, and $A_i = \mathsf{KDF}\left(R_i^0; (\mathtt{inte}, i)\right)$. Then we use a $\mathtt{linear}$-command to define $C_i = A_i \oplus O_i^1$ (with label $(\mathtt{garb}, i)$). Then we add C_i to the set of values to leak by outputting $(\mathtt{leak}, (\mathtt{garb}, i))$. This is a correct garbling, so when we are later given $L_i^{l_i}$ and $R_i^{r_i}$, we can use them to compute $O_i^{w_i}$ by computing the garbled gate on $(L_i^{l_i}, R_i^{r_i})$.

XOR gate: We proceed as for NAND gates, except for the specific commands issued: We use $\mathtt{linear}$-commands to define $O_i^0 = L_i^0 \oplus R_i^0$ (under identifier $(\mathtt{key}, i, 0)$), $O_i^1 = L_i^0 \oplus R_i^1$ (under identifier $(\mathtt{key}, i, 1)$) and $C_i = L_i^0 \oplus L_i^1 \oplus R_i^0 \oplus R_i^1$ (under identifier $(\mathtt{garb}, i)$). Then we add C_i to the set of values to leak by outputting $(\mathtt{leak}, (\mathtt{garb}, i))$. This is a correct garbling, so we later use it to compute $O_i^{w_i}$ by computing the garbled gate on $(L_i^{l_i}, R_i^{r_i})$.

End: After having handled all the gates, we issue the $\mathtt{end}$-command and learn the input keys $K_i = X_i^{x_i}$ for $i \in [n]$, along with the garbled gates C_i for $i \in [n+1; n+q]$. Using these we can evaluate the garbled circuit and thus learn the value $K_i = O_i^{w_i}$ for all $i \in [n+1; q]$. We then give $K^x = \{K_i, \ldots, K_{n+q}\}$ to the adversary.

Guess: If the adversary outputs $\left(i, O_i^{1-w_i}\right)$ for any $i \in [n+q]$, then we output $\left(\mathtt{guess}, (\mathtt{key}, i, 1 - w_i), O_i^{1-w_i}\right)$.

It is clear that we win the guessing game exactly when $(\mathtt{key}, i, 1 - w_i)$ is uncompromised and $O_i^{1-w_i}$ is the correct "other" key for wire i supplied by the adversary – we call $O_i^{w_i}$ the *known key* and we call $O_i^{1-w_i}$ the *other key.* We call a key O_i^b *compromised* if the label $(\mathtt{key}, i, b)$ is compromised as defined by the KDF game. We call gate C_i *compromised* if *the other key* $O_i^{1-w_i}$ is compromised as defined by the KDF game.

It is sufficient to prove that $(\mathtt{key}, i, 1-w_i)$ is uncompromised for all i. It is clear that whether $(\mathtt{key}, i, 1-w_i)$ is uncompromised does not depend on the strategy of the adversary, only the structure of the circuit, the nature of our garbling scheme and the input x. Hence, if for a fixed circuit and fixed input x some $(\mathtt{key}, i, 1 - w_i)$ is sometimes compromised, then it is always compromised. Hence, if any $(\mathtt{key}, i, 1 - w_i)$ can be compromised, then there exist a first gate j such that before executing the commands corresponding to gate j, no identifier $(\mathtt{key}, i, 1 - w_i)$ was compromised, and after executing the commands corresponding to gate j, some identifier $(\mathtt{key}, i, 1 - w_i)$ is compromised, where $i \leq j$. Consider this gate C_j. Furthermore, among the commands executed for gate j there is a first command that leads to a compromise of a gate. We call this command *patient zero.* We first show that patient zero is not a key derivation command. Then we show that it is not a linear command followed by a leak command. And then we are done.

Assume first that patient zero is a key derivation command. We use several times that a key derivation command, when it is the last command to have been executed, cannot compromise any other key than its output key. When patient zero is a key derivation command, then gate j must be a NAND gate, as there are no key derivation commands in XOR gates. Recall that we issue the key derivation commands (1), (2) and (3), as part of a NAND gate, and then we leak C_j. Assume that $l_j = 0$. In that case $O_j^1 = \mathsf{KDF}\left(L_j^0; (\mathtt{key}, j, 1)\right)$ is a known key and hence cannot be a compromised *other* key. We can also assume that L_j^1 is uncompromised (as it is an *other key* and we are at patient zero), and hence the *other* output key $O_j^0 = \mathsf{KDF}\left(L_j^1, R_j^1; (\mathtt{key}, j, 0)\right)$ will clearly be uncompromised after executing the command. Assume then that $r_j = 0$. In that case the other output key is again $O_j^0 = \mathsf{KDF}\left(L_j^1, R_j^1; (\mathtt{key}, j, 0)\right)$, and now R_j^1 is uncompromised. The command $A_j = \mathsf{KDF}\left(R_j^0; (\mathtt{inte}, j)\right)$ can therefore never be the patient zero compromising an output key, as A_j is not an output key.

Before we prove that patient zero cannot be a linear command we change the system that we analyze by replacing the processing of all NAND gates by the following commands: First we execute $(\mathtt{fresh\ key}, (\mathtt{key}, j, 0))$, $(\mathtt{fresh\ key}, (\mathtt{key}, j, 1))$ and $(\mathtt{fresh\ key}, (\mathtt{inte}, j))$ to define the values O_j^0, O_j^1 and A_j respectively. Then we compute $C_j = A_j \oplus O_j^1$, and leak C_j by issuing the commands $(\mathtt{linear}, (\mathtt{garb}, j), (\mathtt{inte}, j), (\mathtt{key}, j, 0))$ and $(\mathtt{leak}, (\mathtt{garb}, j))$ in that order. In addition we leak $O_j^{w_j}$. If $r_j = 0$ such that R_j^0 is a known key, then we also leak A_j. So, we essentially skip all key derivation commands and simulate their effect on the system by leaking the produced known keys. Since we could compute $O_j^{w_j}$ before the change, it was compromised before the change. It is also compromised after the change, as we now leak it. Similarly for A_j. Hence, the set of compromised identifiers is the same before and after the introduced changes, *at least right after the gate has been handled.* As a consequence, we have not changed whether or not some *other* key later gets compromised.[2] Furthermore, notice that since we have already showed that patient zero could not be a key derivation command this change does not affect the adversary's advantage. We therefore just have to prove that in the modified system, no *other* key gets compromised. Since there are no key derivation commands left, this is simple linear algebra.

Assume that patient zero is $C_j = A_j \oplus O_j^1$. Since A_j is a fresh key and only occurs in this equation, if A_j is uncompromised, adding this equation cannot change whether an output key is compromised or not.[3] Hence it must be the case that A_j is compromised. Since A_j is fresh and occurs in no other equation, this can only have happened because we leaked it earlier. Hence R_j^0 is a known key. So, $l_j = 0$ and hence $w_j = 1$. Therefore O_j^1 is a known key and hence already compromised. Hence $C_j = A_j \oplus O_j^1$ will compromise A_j, but since A_j occurs in no other equation, this does not further change the status of any variable. We can therefore assume in the following that we process all NAND gates, with index i, as follows: Call $(\texttt{fresh key}, (\texttt{key}, i, 0))$, $(\texttt{fresh key}, (\texttt{key}, i, 1))$ and $(\texttt{leak}, (\texttt{key}, i, w_i))$ to first define the key O_i^0, O_i^1 and then leak $O_i^{w_i}$. This does not change whether or not there will be a patient zero. We can even make further changes. We once and for all create a global key Δ through the call $(\texttt{fresh key}, \texttt{delta})$. Then we execute each AND gate as follows: Call $(\texttt{fresh key}, (\texttt{key}, i, 0))$, $(\texttt{linear}, (\texttt{key}, i, 1), (\texttt{key}, i, 0), \texttt{delta})$ and $(\texttt{leak}, (\texttt{key}, i, w_i))$ to define the key O_i^0 and O_i^1 respectively and leak $O_i^{w_i}$. Similarly we can create the input keys X_i^0 and $X_i^1 = X_i^0 \oplus \Delta$ by calling $(\texttt{fresh key}, (\texttt{key}, i, 0))$ and $(\texttt{linear}, (\texttt{key}, i, 1), (\texttt{key}, i, 0), \texttt{delta})$ respectively for $i \in [n]$. This will only *add* equations to the system, and hence if there was a patient zero in the system before the change there will also be a patient zero in the system after the change.

Assume then that patient zero is a linear command from an XOR gate, again with index j. We process such a gate as follows: Compute $O_j^0 \leftarrow L_j^0 \oplus R_j^0$ (with label $(\texttt{key}, j, 0)$), $O_j^1 \leftarrow L_j^0 \oplus R_j^1$ (with label $(\texttt{key}, j, 1)$) and $C_j \leftarrow L_j^0 \oplus L_j^1 \oplus R_j^0 \oplus R_j^1$ (with label $(\texttt{garb}, j)$) using the `linear` command, and leak C_j using the `leak` command. Notice that $L_j^0 \oplus L_j^1 \oplus R_j^0 \oplus R_j^1 = \Delta \oplus \Delta = 0$. Hence leaking C_j does not change the status of any key. We can therefore assume that we process XOR gates as follows: Compute $O_j^0 \leftarrow L_j^0 \oplus R_j^0$ and $O_j^1 \leftarrow L_j^0 \oplus R_j^1$ using the `linear` command.

After all the changes to the system, we now "garble" as follows: First call

$$\Delta \leftarrow (\texttt{fresh key}, \texttt{delta})$$

Then for each input key, $i \in [n]$, do:

$$\begin{aligned} X_i^0 &\leftarrow (\texttt{fresh key}, (\texttt{key}, i, 0)) \ , \\ X_i^1 &\leftarrow (\texttt{linear}, (\texttt{key}, i, 1), (\texttt{key}, i, 0), \texttt{delta}) \ , \\ X_i^{x_i} &\leftarrow (\texttt{leak}, (\texttt{key}, i, x_i)) \ . \end{aligned}$$

For each NAND gate, with index i, do:

$$\begin{aligned} O_i^0 &\leftarrow (\texttt{fresh key}, (\texttt{key}, i, 0)) \ , \\ O_i^1 &\leftarrow (\texttt{linear}, (\texttt{key}, i, 1), (\texttt{key}, i, 0), \texttt{delta}) \ , \\ O_i^{w_i} &\leftarrow (\texttt{leak}, (\texttt{key}, i, w_i)) \ . \end{aligned}$$

Finally, for each XOR gate, with index i, do:

$$\begin{aligned} O_i^0 &\leftarrow (\texttt{linear}, (\texttt{key}, i, 0), (\texttt{key}, l_i, 0), (\texttt{key}, r_i, 0)) \ , \\ O_i^1 &\leftarrow (\texttt{linear}, (\texttt{key}, i, 0), (\texttt{key}, l_i, 0), (\texttt{key}, r_i, 1)) \ , \\ O_i^{w_i} &\leftarrow (\texttt{leak}, (\texttt{key}, i, w_i)) \ . \end{aligned}$$

[2] Note that if eventually an *other* key gets compromised, then the introduced changes *will* have an effect. When we use key derivation commands, one compromised *other* key leads to many compromised *other* keys. When we use fresh key commands, a compromised *other* key might not have an avalanche effect. However, we are proving that the number of compromised *other* keys is 0, and hence using one system or the other is equally good.

[3] If O_j^1 is uncompromised then A_j goes from uncompromised to compromised, but A_j is not an output key, and clearly no other key than A_j can change status by this equation.

It is then fairly straight-forward to see that there are no compromised *other* key. In particular, it is trivial to see that if an other key would be compromised in this system, then the free-XOR scheme from [KS08] would trivially be insecure, as the system of equations created by the free-XOR scheme is a super set of the system created by the above commands. We therefore refer to [KS08] for the details of why the free-XOR trick is secure. □

Notice that can use a subset of this proof to prove security of our free-XOR privacy-free garbling scheme, since the free-XOR already implements the global difference Δ. Specifically we have the following theorem:

Theorem 3. *If the* KDF *used in the garbling scheme of Fig. 1 composed with Fig. 3 is secure according to Def. 6, then the composed scheme enjoys authenticity according to Def. 2.*

4 Privacy-free fleXOR

In [KMR14] Kolesnikov *et al.* introduced a generalization and optimization of the free-XOR approach which allows to weaken the security assumption needed for free-XOR and/or limit the amount of ciphertexts used to garble non-XOR gates. In their schemes (only considering fan-in 2 gates) non-XOR gates are constructed exactly as one would in a regular garbling scheme, but XOR gates are constructed differently and, depending on a wire ordering of the circuit, consists of either 0, 1 or 2 ciphertexts. When the garbling scheme used implements aggressive row reduction this yields an overall smaller size for most garbled circuits compared the size of garbled circuits constructed using the free-XOR approach.

Here we propose a variant of fleXOR which combines their ideas with non-oblivious gate evaluation, leading to a significant improvements in terms of computation complexity. Before we can describe our privacy-free fleXOR construction we need a few definitions. These are taken almost verbatim from [KMR14]. We assume familiarity with their construction and direct the reader to their paper if that is not the case.

Definition 7 (Wire Ordering). *A* wire ordering *for a Boolean circuit f is a function $\mathcal{L}$ that assigns an integer to each wire in f. Without loss of generality, we assume that $im(\mathcal{L}) = \{1, \ldots, L\}$ for some integer L, and we denote $|\mathcal{L}| = L$. We say a wire ordering $\mathcal{L}$ is* safe *if:*

- *For each non-XOR gate with output wire i, and each wire j where there exists a directed path in the circuit that contains wire j before wire i, we have $\mathcal{L}(i) > \mathcal{L}(j)$.*
- *For each value $\ell \in im(\mathcal{L})$, there is at most one non-XOR gate whose output wire i satisfies $\mathcal{L}(i) = \ell$.*

We say that a topological ordering of gates in a circuit f is safety-respecting *of $\mathcal{L}$ if for every non-XOR gate g with output wire i, g appears earlier in the ordering than any other gate g' with output wire i' satisfying $\mathcal{L}(i) = \mathcal{L}(i')$.*

Formal Description. We describe the privacy-free fleXOR protocol for gates of fan-in m in Fig. 4 and Fig. 5. Notice that the description in Fig. 4 is essentially the same as the one for the general privacy-free scheme we described in Fig. 1, except for the fact that we include the wire ordering $\mathcal{L}$ needed in order for the garbling scheme to know which Δ's should be used for which wires. Regarding the specificities of the garbling, described in Fig. 5, see that the garbling of NAND gates is exactly the same as in Fig. 2 and Fig. 3, depending on whether or not the wire ordering is safe. That is, the scheme first checks whether or not a gate is an XOR or NAND gate. If it is a NAND gate then the garbling is the same as in Fig. 2 if $\mathcal{L}$ is *safe*, and the same as in Fig. 3 if $\mathcal{L}$ is not safe.

Regarding XOR gates, we garble them essentially as in Fig. 2 but, since the offsets of the wires are chosen during the InKeys procedure, the Garb procedure can only define the 0-key corresponding to the output wire. Then, as in Fig. 2, the Garb procedure computes and outputs the XOR of the offsets between the inputs and output wire, but only for the wires that belong to the set T, that is those for which $\mathcal{L}(i) \neq \mathcal{L}(g)$, which means that the Δ used for the 1-key on wire i is different from the Δ used on the output wire of the gate g. This in turn means that we must associate a ciphertext in order "adjust" the key on wire i.

```
$\mathsf{Gb}\left(1^k, f, \mathcal{L}\right) \to (F, e, d)$
    1. Set $(n, q, I, W, G) \leftarrow f$ and $\left\{X_i^0, X_i^1, \{\Delta_i\}_{i \in [L]}\right\}_{i \in [n]} \leftarrow \mathsf{InKeys}(n, k, \mathcal{L})$.
    2. For each $g \in [n+1, n+q]$ set $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g, i)$ for all
       $i \in \{0,1\}^m$ and set $\left\{\left(X_g^0, X_g^1\right), P[g]\right\} \leftarrow \mathsf{Garb}\left(g, G', \mathcal{L}, \left\{X_{W(g,i)}^0, X_{W(g,i)}^1\right\}_{i \in [m]}, \Delta_{\mathcal{L}(g)}\right)$.
    3. Set $F \leftarrow (n, q, I, W, G, \mathcal{L}, P)$, $e \leftarrow \left\{X_i^0, X_i^1\right\}_{i \in [n]}$ and $d \leftarrow X_{n+q}^1$.
    4. Finally return $(F, e, d)$.
$\mathsf{En}(e, x) \to X$
  Like in Fig. 1.
$\mathsf{De}(d, Z) \to b$
  Like in Fig. 1.
$\mathsf{Ev}(F, X, x) \to Z$
    1. Set $(n, q, I, W, G, \mathcal{L}, P) \leftarrow F$ and for all $i \in [n]$ set $w_i = x_i$ and define $Q = \{w_i\}_{i \in [n]}$.
    2. For each $g \in [n+1, n+q]$ let $m = I(g)$ and add $w_g = G\left(g, \left\{w_{W(g,i)}\right\}_{i \in [m]}\right)$ to the set $Q$.
    3. Now for each $g \in [n+1, n+q]$ let $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g, i)$ and
       $w' \in \{0,1\}^m$ s.t. ${w'}_i = w_{W(g,i)}$ for all $i \in [m]$ and set $X_g \leftarrow \mathsf{Eval}\left(g, G', \mathcal{L}, w', \left\{X_{W(g,i)}\right\}_{i \in [m]}, P[g]\right)$.
    4. Return $X_{n+q}$.
$\mathsf{ev}(f, x) \to b$
  Like in Fig. 1.
$\mathsf{Ve}(F, f, e) \to b$
    1. Set $(n, q, I, W, G, \mathcal{L}, P) \leftarrow F$, $(n', q', I', W', G') \leftarrow f$ and $\left\{X_i^0, X_i^1\right\}_{i \in [n]} \leftarrow e$.
    2. If $n \neq n'$, $q \neq q'$, $I \neq I'$, $W \neq W'$ or $G \neq G'$ output 0.
    3. For each $g \in [n+1, n+q]$ let $m = I(g)$ and define $G' : \{0,1\}^m \to \{0,1\}$ s.t. $G'(i) = G(g, i)$ for all
       $i \in \{0,1\}^m$ and set $\left\{\left(X_g^0, X_g^1\right), \bar{P}[g]\right\} \leftarrow \mathsf{Garb}\left(g, G', \mathcal{L}, \left\{X_{W(g,i)}^0, X_{W(g,i)}^1\right\}_{i \in [m]}, \Delta_{\mathcal{L}(g)}\right)$.
    4. If for any $g \in [n+1, n+q]$ we have $\bar{P}[g] \neq P[q]$ output 0, otherwise output 1.
```

Figure 4. Privacy-free FleXOR Garbling

Regarding evaluation: for NAND gates the scheme again does the same as in Fig. 2 and Fig. 3 depending on whether or not the wire ordering is safe or not, respectively. For XOR gates the scheme first defines (in step a) the set of input wires for which $\mathcal{L}(i) \neq \mathcal{L}(g)$, T, and parses the garbled gate $\tilde{g}$ to its ciphertexts, $\{C_i\}_{i \in T}$. Then in step c the scheme identifies the subset $S \subset T$ of the input wires for which it is true that the input value for wire i is equal to 1 and finally, in step d it computes the output key by XORing all input keys and the adjustments for all the wires belonging to the set S.

Security. Like for our other privacy-free garbling schemes, correctness and verifiability follows relatively straightforwards from the constructions. The proof of authenticity follows from the one for the scheme in Fig. 2 (since the fleXOR variant is a generalization of the schemes described in Fig. 2 for which some input wires happen to the same offset as the output wire) and from the assumption on the wire ordering. We refer to [KMR14] for more details.

Acknowledgements. We would like to thank Payman Mohassel and Benny Pinkas for useful discussions, and the authors of [KMR14] for sharing with us an early copy of their manuscript and the result of their “safe ordering” heuristics that were used for compiling Table 1 and 2.

```
InKeys$(n, k, \mathcal{L}) \to \left\{ \left(X_i^0, X_i^1\right)_{i \in [n]}, \{\Delta_i\}_{i \in [L]} \right\}$
    1. For each $i \in [L]$ sample uniformly random differences $\Delta_i \in \{0,1\}^k$.
    2. Then for each $i \in [n]$ sample uniformly random $X_i^0 \in_R \{0,1\}^k$ and return the set
       $\left\{ \left(X_i^0, X_i^0 \oplus \Delta_{\mathcal{L}(i)}\right)_{i \in [n]}, \{\Delta_i\}_{i \in [L]} \right\}$.
Garb $\left(g, G', \mathcal{L}, \left\{ \left(X_i^0, X_i^1\right) \right\}_{i \in [m]}, \Delta_{\mathcal{L}(g)}\right) \to \left\{ \left(X_g^0, X_g^1\right), \tilde{g} \right\}$
    1. If $G'(\cdot) = \text{NAND}$ do garbling as described in Fig. 2 if $\mathcal{L}$ is *safe*, otherwise as described in Fig. 3.
    2. If instead $G'(\cdot) = \text{XOR}$ do as follows:
        (a) Let $T$ be the set of integers $i \in [m]$ for which $\mathcal{L}(i) \neq \mathcal{L}(g)$.
        (b) Let $X_g^0 = \bigoplus_{i=1}^{m} X_i^0$ and $X_g^1 = X_g^0 \oplus \Delta_{\mathcal{L}(g)}$.
        (c) Next let $C_i = \Delta_{\mathcal{L}(g)} \oplus \Delta_{\mathcal{L}(i)}$ for all $i \in T$ and set $\tilde{g} = \{C_i\}_{i \in T}$.
        (d) Return $\left\{ \left(X_g^0, X_g^1\right), \tilde{g} \right\}$.
Eval $\left(g, G', \mathcal{L}, w', \{X_i\}_{i \in [m]}, \tilde{g}\right) \to \{X_g\}$
    1. If $G'(\cdot) = \text{NAND}$ do evaluation as described in Fig. 2 if $\mathcal{L}$ is *safe*, otherwise as described in Fig. 3.
    2. If instead $G'(\cdot) = \text{XOR}$ do as follows:
        (a) Let $T$ be the set of integers $i \in [m]$ for which $\mathcal{L}(i) \neq \mathcal{L}(g)$ and parse $\{C_i\}_{i \in T} \leftarrow \tilde{g}$.
        (b) Parse $\{C_i\}_{i \in T} \leftarrow \tilde{g}$.
        (c) Let $S$ be the subset of $T$ for which it is true that ${w'}_i = 1$.
        (d) Return $X_g = \left(\bigoplus_{i \in [m]} X_i\right) \oplus \left(\bigoplus_{i \in S} C_i\right)$.
```

Figure 5. Garbling - Using fleXOR

References

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *FOCS*, pages 120–129, 2011. 4

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *CRYPTO (2)*, pages 166–184, 2013. 4

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010. 3

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. *IACR Cryptology ePrint Archive*, 2013:426, 2013. 3, 7

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012. Full version at `http://eprint.iacr.org/2012/265`. 1, 3, 7, 8, 13, 25

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990. 3

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the aes s-box. In *SEC*, pages 287–298, 2012. 7

[Fin14] Magnus Gausdal Find. On the complexity of computing two nonlinearity measures. In *CSR*, pages 167–175, 2014. 7

[FJJBNO13] Tore Frederiksen, Thomas P Jakobsen, Peter Sebastian Nordholt Jesper Buus Nielsen, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions (full version). Cryptology ePrint Archive, Report, 2013. `http://eprint.iacr.org/`. 4

[GGH$^+$13] Craig Gentry, Sergey Gorbunov, Shai Halevi, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. How to compress (reusable) garbled circuits. Cryptology ePrint Archive, Report 2013/687, 2013. `http://eprint.iacr.org/`. 4

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010. 3

[GHL$^+$14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *EUROCRYPT*, pages 405–422, 2014. 4

[GKP$^+$12] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. Cryptology ePrint Archive, Report 2012/733, 2012. `http://eprint.iacr.org/`. 4

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987. 3

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002. 3

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM Conference on Computer and Communications Security*, pages 955–966, 2013. 1, 3, 7, 8, 25

[KK12] Vladimir Kolesnikov and Ranjit Kumaresan. Improved secure two-party computation via information-theoretic garbled circuits. In *SCN*, pages 205–221, 2012. 3

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In *CRYPTO*, 2014. 3, 4, 5, 7, 11, 20, 21

[Kol05] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT*, pages 136–155, 2005. 3

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, pages 486–498, 2008. 3, 20

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs. In *EUROCRYPT*, pages 719–734, 2013. 4

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009. 3

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004. 3

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *TCC*, pages 368–386, 2009. 4

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999. 3

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009. 3

[Rog91] Phillip Rogaway. *The round complexity of secure protocols*. PhD thesis, Massachusetts Institute of Technology, 1991. 3

[ST12] Nigel Smart and Stefan Tillich. Circuits of basic functions suitable for mpc and fhe, 2012. `http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`. 6, 7

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986. 3

A Zero-Knowledge from Garbled Circuits.

In Figure 6 a sketch of the ZK protocol proposed by Jawurek *et al.* [JKO13] is shown. The protocol proceeds as follows: The prover (acting as the receiver in the OT) uses the bits of his witness x as choice bits in the OT while the verifier (acting as the sender in the OT) uses as input all the pairs of keys of the garbled circuits. The verifier also sends the garbled circuit F. Now if the prover uses a valid witness, he can evaluate the garbled circuit and compute the output key corresponding to the output bit 1. However, instead of disclosing this key at this stage, the prover commits to it and waits for the verifier to prove that she constructed the garbled circuit correctly (and acted honestly in the OT protocols as well). If this check goes through, the prover opens the commitment and the verifier accepts the proof if the commitment contains the key corresponding to the output bit 1. The main ideas behind the proof of security in [JKO13] are as follows: soundness (the verifier accepts only if the statement is true) is achieved thanks to the *authenticity* property of garbled circuits – using the terminology of Bellare *et al.* [BHR12]. At the same time the protocol is zero-knowledge (the verifier learns *only* that the statement is true) because the prover verifies that she generates the GC honestly before disclosing any information.

Using a "weak" KDF. Notice that the randomness used to generate the garbled circuit is completely revealed to the prover at the end of the protocol, thus the encryption used to garble the gates only needs to remain secure throughout the execution of the protocol (in contrast to most secure computation settings where the garbling scheme needs to remain secure long after the execution of the protocol is complete in order to ensure privacy of the input). Therefore one could imagine the following optimization: to garble the circuit, use a "weak" KDF such that it is reasonable to assume that the prover cannot break it in time $100t$, and let the verifier accept the proof iff the prover sends the commitment C before time t. This might allow to use shorter keys and faster KDFs, resulting in better computation and communication complexity.

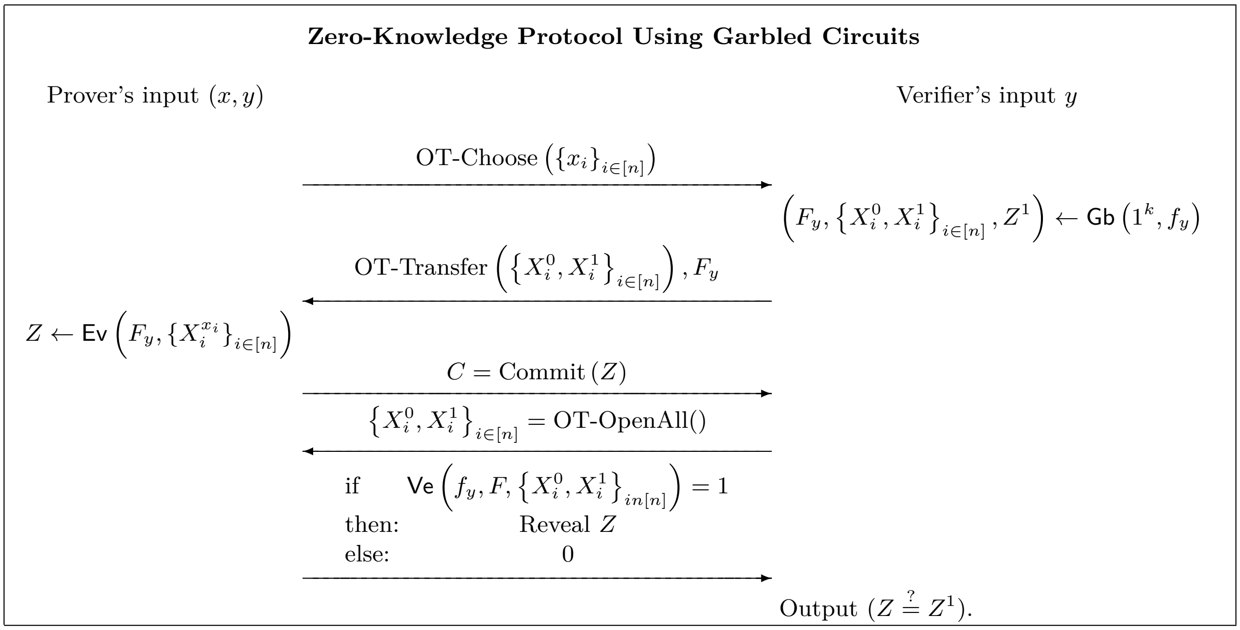
Zero-Knowledge Protocol Using Garbled Circuits
Prover's input (x, y)
Verifier's input y
OT-Choose $\left(\{x_i\}_{i\in[n]}\right)$
$\left(F_y, \left\{X_i^0, X_i^1\right\}_{i\in[n]}, Z^1\right) \leftarrow \mathsf{Gb}\left(1^k, f_y\right)$
OT-Transfer $\left(\left\{X_i^0, X_i^1\right\}_{i\in[n]}\right), F_y$
$Z \leftarrow \mathsf{Ev}\left(F_y, \{X_i^{x_i}\}_{i\in[n]}\right)$
$C = \text{Commit}\left(Z\right)$
$\left\{X_i^0, X_i^1\right\}_{i\in[n]} = \text{OT-OpenAll}()$
if $\mathsf{Ve}\left(f_y, F, \left\{X_i^0, X_i^1\right\}_{in[n]}\right) = 1$
then: Reveal Z
else: 0
Output $(Z \overset{?}{=} Z^1)$.

Figure 6. Informal Description of Jawurek *et al.* ZK from GC.

B Proof of Theorem 1

Proof. We do the proof by constructing several hybrid games and then show a polytime reduction between them. We then show that any PPT adversary can only win the last hybrid game with negligible probability and thus conclude that this must also be the case for the real game. First consider the following hybrids:

Definition 8 (Hybrid 1). *Let Hybrid 1 be the game defined in the same way as "game* KDF*" except that the command **Guess** is defined as follows: If* $\mathcal{A}$ *outputs* $(\texttt{guess}, id^* \in \mathsf{ID} \setminus \mathsf{COMP})$ *then sample a uniformly random bit* $b \leftarrow \{0,1\}$*. If* $b = 0$ *return a uniformly random* k *bits string* $K' \leftarrow \{0,1\}^k$*, otherwise return* K_{id^*}*.* $\mathcal{A}$ *then returns a bit* $c \in \{0,1\}$ *and wins if* $c = b$*. The advantage is the probability that* $\mathcal{A}$ *wins minus* $\frac{1}{2}$*.*

Definition 9 (Hybrid 2). *Let Hybrid 2 be the game defined in the same way as Hybrid 1 except that the command **Derive** does not exist.*

We let $\mathrm{H}^i_{\mathsf{KDF},\mathcal{A}}(1^k)$ denote the advantage of any PPT $\mathcal{A}$ playing Hybrid i. In particular notice that $\mathrm{H}^i_{\mathsf{KDF},\mathcal{A}}(1^k) = \frac{1}{2} - \Pr[\mathcal{A} \text{ wins }]$.

In the rest of the proof we say a key K_{id} is "fresh" if it has been constructed by the call $(\texttt{fresh key}, id)$. Similarly we say that a key K_{id_0} is "derived", respectively "linear" if it has been constructed by the call $(\texttt{derive}, id_0, id_1, \ldots, id_m)$, respectively $(\texttt{linear}, id_0, id_1, \ldots, id_m)$.

Before we start on the hybrid reductions consider the following lemmas:

Lemma 1. *Let* $\mathcal{A}^2$ *be any PPT adversary attacking Hybrid 2. Then for any* $id \in \mathsf{ID} \setminus \mathsf{COMP}$*, where* K_{id} *is a fresh key,* K_{id} *is indistinguishable from an uniformly random* k *bit string in the view of* $\mathcal{A}^2$*.*

Proof. First notice that since $id \notin \mathsf{COMP}$ and has been constructed by the call $(\texttt{fresh key}, id)$, then the value K_{id} will be a uniformly random sampled element which has not been given to $\mathcal{A}^2$. Thus anything that $\mathcal{A}^2$ can learn about K_{id} must necessarily be based on leaked information on keys constructed by calls to **Linear** which involves K_{id}. This follows since any other information the adversary can learn from the game will be independent of K_{id}. Next notice that any leaked linearly constructed key, say $K_{id'}$, depending on K_{id} is constructed as a linear combination of some keys and K_{id}. Then see that each of the keys are either fresh or linear keys, so if they are linear keys we can simply substitute their linear expressions. This can be done recursively. Thus we get that $K_{id'} = K_{id} \oplus \left(\bigoplus_{i\in[m]} K_{id_i}\right)$ for some integer $m \geq 1$. Next see that it must be the case that at least one $id_i \notin \mathsf{COMP}$, otherwise it would be the case that $id \in \mathsf{COMP}$. For each $id_i \notin \mathsf{COMP}$ we have that K_{id_i} is uniformly random in the view of $\mathcal{A}^2$. Furthermore, since any value XORed with a uniformly random value is uniformly random we have that $K_{id'}$ will be a one-time pad encryption of K_{id} under the key K_{id_i} (potentially XORed with other compromised or uncompromised keys). This will obviously be the case for all leaked linear constructed keys depending on K_{id}. However, if the leaked linear keys are different from the key equivalent to the one-time encryption key will also be different, otherwise, following a similar argument as above, K_{id} would be compromised.

Finally, since one-time encryptions are perfectly secure (leaks no information) when the key is unused and uniformly random, like above, then it will be impossible to use these to gain an advantage in distinguishing between K_{id} and another uniformly random string. □

Lemma 2. *Let* $\mathcal{A}^2$ *be any PPT adversary attacking Hybrid 2, then for any* $id_0 \in \mathsf{ID} \setminus \mathsf{COMP}$ *where* K_{id_0} *is constructed by the call* $(\texttt{linear}, id_0, id_1, \ldots, id_m)$ *and* $id_1, \ldots, id_m \in \mathsf{ID}$ *then* K_{id_0} *is indistinguishable from an uniformly random* k *bit string in the view of* $\mathcal{A}^2$*.*

Proof. First notice that since $id_0 \notin \mathsf{COMP}$ then there must be at least one $i \in [m]$ for which $id_i \notin \mathsf{COMP}$. Next notice that we can express the key K_{id_0} as a linear combination of keys constructed by calls to **fresh key** (by recursively expanding any key K_{id_i} constructed by a call to **linear**). If K_{id_i} has been constructed by the call $(\texttt{fresh key}, id_i)$ then we have by Lemma 1 that K_{id_i} is indistinguishable from a uniformly random k bit string for $\mathcal{A}^2$. Thus $K_{id_0} = K' \oplus K_{id_i}$ where K_{id_i} is indistinguishable from a uniformly random string and K' is some, perhaps known, string. From this we get that K_{id_0} is indistinguishable from a uniformly random string. However we must still show K_{id_0} remains indistinguishable from a uniformly random string no matter what following queries the adversary makes. Like in the proof of Lemma 1, the only way the adversary can gain information on K_{id_0} is through compromised linear queries depending on either K_{id_0} or the keys used in the linear construction of K_{id_0}. First consider the case of linear keys depending directly on K_{id_0}. Notice that any leaked linearly constructed key, say $K_{id'}$ depending on K_{id_0} can be expressed as a linear combination of some keys and K_{id_0}. Then see that each of these keys is either a fresh key or linear key. So for any linear keys we can simply substitute their linear expressions. Thus we get that $K_{id'} = K_{id_0} \oplus \left(\bigoplus_{i\in[m]} K_{id_i}\right)$ for some integer $m \geq 1$. Next see that it must be the case that at least one $id_i \notin \mathsf{COMP}$, otherwise it would be the case that $id_0 \in \mathsf{COMP}$. For each $id_i \notin \mathsf{COMP}$ we have that K_{id_i} is uniformly random sampled. Furthermore, since any value XORed with a uniformly random value is uniformly random we have that $K_{id'}$ will be a one-time pad encryption of K_{id_0} under the key K_{id_i} (potentially XORed with other compromised or uncompromised keys). This will obviously be the case for all leaked linear constructed keys depending on K_{id_0}. However, if they are different from the key equivalent to the one-time encryption key will also be different otherwise, following a similar argument as above, K_{id_0} would be compromised. Since one-time encryptions are perfectly hiding when the key is unused and uniformly random, like above, then it will be impossible to use these to gain an advantage in distinguishing between K_{id_0} and another uniformly random string.

Now consider the case of keys used in the linear combination of K_{id_0}. Notice that if there is only one uncompromised key in the linear combination defining K_{id_0} then any indistinguishably advantage in an uncompromised key of the linear combination of K_{id} can be directly used in distinguishing K_{id_0} from a random string. This is so since we could write $K_{id_0} = K' \oplus K_{id_i}$ for a known (or polytime computable string K') and thus efficiently compute the "adjusted" knowledge of K_{id_0} based on K_{id_i}. □

Lemma 3. *For any adversary* $\mathcal{A}^2$ *and some negligible function* $\mathrm{negl}(\cdot)$ *it holds that* $\mathrm{H}^2_{\mathsf{KDF},\mathcal{A}^2}(1^k) \leq \mathrm{negl}(k)$.

Proof. First notice that all keys in this game have been constructed by calls to either **Fresh key** or **Linear**. So when $\mathcal{A}^2$ calls $(\mathtt{guess}, id^*)$ then it must necessarily be the case that K_{id^*} is a fresh or linear key. Now since we are in Hybrid 2 we see that $\mathcal{A}^2$ cannot win the game with non-negligible probability following tLemma 1 and Lemma 2 since no matter what key he tries to guess it will indistinguishable from random in his view. □

Next we show a reduction between Hybrid 2 and Hybrid 1. First we notice that the only difference between Hybrid 1 and 2 is the possibility of the adversary to construct new keys with method **Derive**.

Lemma 4. *For any PPT adversary* $\mathcal{A}^1$ *there exists a PPT adversary* $\mathcal{A}^2$ *such that* $\mathrm{H}^1_{\mathsf{KDF},\mathcal{A}^1}(1^k) \leq \mathrm{H}^2_{\mathsf{KDF},\mathcal{A}^2}(1^k) - \mathrm{negl}(k)$ *for some negligible function* $\mathrm{negl}(\cdot)$ *when* $\mathsf{KDF}(\cdot)$ *is modeled as a random oracle.*

Proof. We fix any adversary $\mathcal{A}^1$ playing with Hybrid 1. We then construct a polytime adversary $\mathcal{A}^2$ playing with Hybrid 2, which runs $\mathcal{A}^1$ internally. We then argue that the advantage of $\mathcal{A}^2$ is at least the same as of $\mathcal{A}^1$, except with negligible difference. This means that if $\mathcal{A}^1$ can attack Hybrid 1 with non-negligible probability then there exists an adversary $\mathcal{A}^2$ that can attack hybrid 2 with non-negligible probability.

We let $\mathcal{A}^2$ play the role of the challenger in Hybrid 1 against $\mathcal{A}^1$. $\mathcal{A}^2$ starts by initializes an empty map D, such that queries of undefined elements result in $\perp$ and an empty list D'. It then passes on each query it gets from $\mathcal{A}^1$ to the Hybrid 2 game stores internally, and passes back to $\mathcal{A}^1$, the results it gets from Hybrid 2, except for the following cases:

Linear: If $\mathcal{A}^1$ outputs $(\mathtt{linear}, id_0 \notin \mathsf{ID}, id_1, \ldots, id_m)$ for $id_i \in \mathsf{ID}$, then update D such that $(id_1, \ldots, id_m) = \mathsf{D}(id_0)$, append id_0 to the end of D' and pass the call to hybrid 2.

Derive: If $\mathcal{A}^1$ outputs $(\mathtt{derive}, id_0 \notin \mathsf{ID}, id_1, \ldots, id_m)$ for $id_i \in \mathsf{ID}$, then update D such that $(id_1, \ldots, id_m) = \mathsf{D}(id_0)$, append id_0 to the end of D' and call $(\mathtt{fresh\ key}, id_0)$.

End: When $\mathcal{A}^1$ outputs $(\mathtt{end})$ define COMP as in the original game, based on the queries from $\mathcal{A}^1$ and initialize an empty set $\mathsf{K}' = \emptyset$. Now for each $id \in \mathsf{COMP}$ where $id \notin \mathsf{LEAK}$ we have $\mathcal{A}^2$ call $(\mathtt{leak}, id)$ and internally store the response from Hybrid 2. Then $\mathcal{A}^2$ calls $(\mathtt{end})$. Now let K to be the set of keys returned by Hybrid 2. Next, iterate through the list D', starting from the beginning, let id_0 be the current entry and set $(id_1, \dots, id_m) = \mathsf{D}(id_0)$. Now in each iteration if it holds, for all $i \in [m]$, that $id_i \in \mathsf{COMP}$ and K_{id_0} is a derived key, then, using the keys in the set K, define $K'_{id_0} = \mathsf{KDF}(K_{id_1}, \dots, K_{id_m}; id_0)$ and set $\mathsf{K}' = \mathsf{K}' \cup K'_{id_0}$. If instead K_{id_0} is a linear key, then, again using the keys in the set K, define $K'_{id_0} = \bigoplus_{i\in[m]} K_{id_i}$ and set $\mathsf{K}' = \mathsf{K}' \cup K'_{id_0}$. Furthermore, if $K_{id_0} \in \mathsf{K}$ then replace K_{id_0} with K'_{id_0} in the set K. Finally return the set K.

Guess: When $\mathcal{A}^1$ outputs $(\texttt{guess}, id^*)$ pass the call on to hybrid 2 and let K' be the output of Hybrid 2, then proceed as follows:
- If K_{id^*} is a fresh or derived key then return K'.
- If $\mathcal{A}^1$ has previously made the call $(\texttt{linear}, id^*, id_1, \dots, id_m)$ then for each $i \in [m]$ where $id_i \in \mathsf{COMP}$ and $K'_{id_i} \in \mathsf{K}'$ set $K' = K' \oplus K_{id_i} \oplus K'_{id_i}$.[4]

Finally, when receiving a bit c from $\mathcal{A}^1$ pass on the same bit to Hybrid 2.

Some notes are due regarding the simulation above. We use D' as a list of ID's, in chronological order, whose associated keys are either linear or derived and thus might need to be simulated by $\mathcal{A}^2$ (as they can be based on compromised derived keys). We then let $\mathcal{A}^2$ pass on calls to **Linear** and **Derive** to Hybrid 2, except that it adds the ID from such calls to the list D'. When **End** is finally called we let $\mathcal{A}^2$ leak all compromised keys from Hybrid 2, as these might be needed for "adjustments". In particular we let $\mathcal{A}^2$ potentially adjust (using XOR operations or replacement) each leaked key it gets back from Hybrid 2 such that it match what $\mathcal{A}^1$ would expect if playing with Hybrid 1. More specifically, if all keys used to construct a derived key have been compromised then the derived key returned to $\mathcal{A}^1$ has been constructed using the KDF, even though the key has been constructed using a call to **Fresh key** in Hybrid 2. The same goes for linearly constructed keys using a compromised derived key in its construction: Specifically by XOR'ing out the fresh key constructed by Hybrid 2 and then xor'ing in the derived keys, constructed using KDF by $\mathcal{A}^2$. Regarding the guess phase we let $\mathcal{A}^2$ make sure that the guess given by $\mathcal{A}^1$ is adjusted in the same manner to reflect the keys in Hybrid 2. This is in particular needed as derived keys are simulated using calls to **Fresh key** and the guess of $\mathcal{A}^1$ might be a linear key which is constructed from compromised derived keys.

We now proceed with the proof that the view of $\mathcal{A}^1$ playing with $\mathcal{A}^2$ and Hybrid 2 is computationally indistinguishable from the view of $\mathcal{A}^1$ playing with Hybrid 1.

First see that until the call to **End** the view of $\mathcal{A}^1$ is perfectly indistinguishable whether it is playing with $\mathcal{A}^2$ or Hybrid 1 since nothing is returned and the same calls are permitted. Furthermore, if $\mathcal{A}^1$ does not call **Derive** then the games will be perfectly indistinguishable, thus in the following we only consider games where $\mathcal{A}^1$ calls **Derive**. Now consider indistinguishability of the output $\mathcal{A}^1$ gets after calling **End**.

Start by noticing that for each $id \in \mathsf{LEAK}$ where $\mathcal{A}^1$ made the call $(\texttt{fresh key}, id)$ the key K_{id} will be perfectly indistinguishable whether $\mathcal{A}^1$ plays with $\mathcal{A}^2$ or Hybrid 1 as it will in both cases be uniformly random sampled. The same goes for each K_{id_0} with $id_0 \in \mathsf{LEAK}$ where $\mathcal{A}^1$ made the call $(\texttt{linear}, id_0, id_1, \dots, id_m)$ and for all $i \in [m]$ it was the case that id_i was constructed by a call to $(\texttt{fresh key}, id_i)$. This is again the case since $\mathcal{A}^2$ does exactly the same as Hybrid 1. Furthermore, we can extend the case for $id_0 \in \mathsf{LEAK}$ where $\mathcal{A}^1$ made the call $(\texttt{linear}, id_0, id_1, \dots, id_m)$ and for all $i \in [m]$ the key K_{id_i} is either a fresh key, or a linear key. This applies recursively.

Next consider indistinguishability of K_{id_0} of the first $id_0 \in \mathsf{D}'$ where $\mathcal{A}^1$ called $(\texttt{Derive}, id_0, id_1, \dots, id_m)$:

- If $id_1, \dots, id_m \in \mathsf{COMP}$ remember that we construct $K_{id_0} \leftarrow \mathsf{KDF}(K_{id_1}, \dots, K_{id_m}; id_0)$ and thus K_{id_0} is constructed exactly the same as in Hybrid 1. This construction is possible since $\mathcal{A}^1$ will know the values $K_{id_1}, \dots, K_{id_m}$ and thus can query the oracle himself.
- If $\exists i \in [m]$ s.t. $id_i \notin \mathsf{COMP}$ we simulated key K_{id_0} by asking Hybrid 1 to construct and leak a fresh key, i.e. a uniformly random key. Thus we must argue that a key sampled uniformly random is indistinguishable from a derived key when at least one of the keys used to construct the derived key is not compromised. To see this consider the following cases for an uncompromised key K_{id_i}:
  1. K_{id_i} was constructed by a call to **Fresh key**.
  2. K_{id_i} was constructed by a call to **Linear**. Assume w.l.o.g. that all keys used in the linear combination was constructed either by a call to **Fresh key** (if not we can repeatedly expand the keys in the linear combination defining K_{id_i} to get a single linear combination of keys made with **Fresh key**).

[4] The key K_{id_i} will always be known by $\mathcal{A}^2$ after **End** as it asks Hybrid 2 to leak it.

Now in the first case, K_{id_i} is indistinguishable from a uniformly random element following the proof of Lemma 1 and the observation that $\mathcal{A}^1$ at this point has not learned anything based on the calls to **Derive** where K_{id_i} has been used. The second case follows from the proof of Lemma 2 and again the observation that $\mathcal{A}^1$ at this point has not learned anything based on the calls to **Derive** where K_{id_i} has been used. Thus K_{id_i} is uniformly random in the view of $\mathcal{A}^1$. This means that for $\mathcal{A}^1$ to distinguish between a uniformly random sampled key and $\mathsf{KDF}(K_{id_1}, \ldots, K_{id_m})$ he must query exactly $\mathsf{KDF}(K_{id_1}, \ldots, K_{id_m}; id_0)$ to get an advantage, since $\mathsf{KDF}(\cdot)$ is a random oracle. However, K_{id_i} is k uniformly random bits in his view and he is bounded by polynomial time in k, thus his advantage can at most be $\mathrm{poly}(k)/2^k$, which is negligible in k. Thus the view induced on $\mathcal{A}^1$ by $\mathcal{A}^2$ is computationally indistinguishable from the view of $\mathcal{A}^1$ when playing with Hybrid 1.

Next we must argue that the view remains indistinguishable for the rest of the derived keys. Following the argument above, this remains true for each uncompromised derived key when the keys used in the derivation have, perhaps recursively, been constructed by calls to **Fresh key**. Thus the remaining case we must argue is when at least one uncompromised derived key has been used in the construction, either directly or as part of a linear combination: First notice that we just showed that the first uncompromised derived key is computationally indistinguishable from a uniformly random sampled key in the view of $\mathcal{A}^1$, thus the argument above goes through if such a key is used instead of a fresh key. The same remains true for linearly constructed keys consisting of at least one uncompromised derived key: Since the uncompromised derived key is computationally indistinguishable from a random key, a linear key where it is replaced with a fresh key will remain computationally indistinguishable from a uniformly random key.

Finally, see that the set of the compromised keys returned to $\mathcal{A}^1$ will be indistinguishable in both the game played with $\mathcal{A}^2$ and Hybrid 1 as they will either be sampled in exactly the same way (by the fact that we do appropriate adjustments). Then see that since $\mathcal{A}^1$ is bounded by $\mathrm{poly}(k)$ we can at most have $\mathrm{poly}(k)$ keys, each computationally indistinguishable from a uniformly random element, and thus the distinguishability advantage of the total view will be bounded by $\mathrm{poly}(k) \cdot \mathrm{negl}(k) = \mathrm{negl}(k)$.

Finally consider the view in regards to the guess by $\mathcal{A}^1$. By the previous arguments if $\mathcal{A}^1$ outputs id^* where K_{id^*} is a fresh or derived key then it will be computationally indistinguishable from a uniformly random string and thus the advantage of $\mathcal{A}^1$ will be the same as the advantage of $\mathcal{A}^2$. If instead K_{id^*} was constructed by the call $(\mathtt{linear}, id^*, id_1, \ldots, id_m)$ then before $\mathcal{A}^2$ passes the key it gets from Hybrid 2 back to $\mathcal{A}^1$ it adjusts it according to the compromised derived keys it might depend on. Thus the key returned to $\mathcal{A}^1$ will be computationally indistinguishable whether it comes from $\mathcal{A}^2$ or Hybrid 1 assuming each uncompromised derived key is indistinguishable form random, which is exactly the case as we previously showed. This in turn implies the advantage of $\mathcal{A}^2$ is the same as $\mathcal{A}^1$, except with negligible difference, as $\mathcal{A}^2$ inputs the same bit to the game as it received by $\mathcal{A}^1$.

Since the views are at most negligibly distinguishable the advantage of $\mathcal{A}^2$ must be the same as $\mathcal{A}^1$ with at most negligible difference. □

Lemma 5. *For any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{A}^1$ such that* $\mathrm{GUESS}_{\mathsf{KDF},\mathcal{A}}(1^k) \leq \mathrm{H}^1_{\mathsf{KDF},\mathcal{A}^1}(1^k) - \mathrm{negl}(k)$ *for some negligible function* $\mathrm{negl}(\cdot)$ *when* $\mathsf{KDF}(\cdot)$ *is modeled as a random oracle.*

Proof. We fix any adversary $\mathcal{A}$ attacking the KDF game. We then construct a polytime adversary $\mathcal{A}^1$ attacking Hybrid 1, which runs $\mathcal{A}$ internally and argue that the advantage of $\mathcal{A}^1$ is at least the same as of $\mathcal{A}$ except with negligible difference. We let $\mathcal{A}^1$ play the role of the challenger in the KDF game against $\mathcal{A}$. $\mathcal{A}^1$ passes on each query it gets from $\mathcal{A}$ to the Hybrid 1 game, stores internally, and passes back to $\mathcal{A}$, the result it gets from Hybrid 1, except for the case of **End**: When $\mathcal{A}$ outputs $(\mathtt{end}, K^*)$ call $(\mathtt{end})$ on Hybrid 1. Let K' be the key returned from Hybrid 1. If $K^* = K'$ then input $c = 1$ to Hybrid 1, otherwise input $c = 0$.

First notice that the view of $\mathcal{A}$ will be perfectly indistinguishable whether it is playing with $\mathcal{A}^1$ or the KDF game since the calls and values returned to $\mathcal{A}$ in both cases are constructed similarly. Thus we only need to argue that the advantage of $\mathcal{A}^1$ is the same as $\mathcal{A}$ playing with the KDF game except with negligible difference in k. See that if the bit b chosen by Hybrid 1 is 1 the winning probability of $\mathcal{A}^1$ and $\mathcal{A}$ is the same. To see this notice that if $K^* = K'$ then both $\mathcal{A}$ and $\mathcal{A}^1$ will win and if $K^* \neq K'$ then both $\mathcal{A}$ and $\mathcal{A}^1$ will loose. If instead $b = 0$ and $K^* = K'$ then $\mathcal{A}$ will win and $\mathcal{A}^1$ will loose. However, since K' in this case is uniformly random sampled $K^* = K'$ will only occur with probability 2^{-k}. If instead $K^* \neq K'$ (happening with probability $1 - 2^{-k}$) then $\mathcal{A}$ will loose and $\mathcal{A}^1$ will win. This will make $\mathcal{A}^1$ win with probability $(1 - 2^{-k})$ if $b = 0$ and probability $\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k)$ if $b = 1$. Since b is uniformly random sampled we get that $\mathcal{A}^1$ wins the game with probability $\frac{1}{2} \cdot (1 - 2^{-k} + \textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k))$. Thus giving $\mathcal{A}^1$ an advantage $\frac{1}{2}\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) - 2^{-k-1}$. Since 2^{-k-1} is negligible in k so is the total difference in advantage of $\mathcal{A}$ playing with $\mathcal{A}^1$, respectively the KDF game. □

Using Lemma 4 and Lemma 5 we get the following inequalities, for any adversary $\mathcal{A}$ and $\mathcal{A}^1$ and $\mathcal{A}^2$ defined as in the proofs, where $\mathrm{negl}(k)$ is some negligible function in k:

$$\begin{aligned} \textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) &\leq \mathrm{H}^1_{\mathsf{KDF},\mathcal{A}^1}(1^k) - \mathrm{negl}(k)\ , \\ \mathrm{H}^1_{\mathsf{KDF},\mathcal{A}^1}(1^k) &\leq \mathrm{H}^2_{\mathsf{KDF},\mathcal{A}^2}(1^k) - \mathrm{negl}(k)\ . \end{aligned}$$

Combining these we get

$$\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) \leq \mathrm{H}^2_{\mathsf{KDF},\mathcal{A}^2}(1^k) - \mathrm{negl}(k)\ .$$

Now we have from Lemma 3 that $\mathrm{H}^2_{\mathsf{KDF},\mathcal{A}^2}(1^k) \leq \mathrm{negl}(k)$ for any adversary $\mathcal{A}^2$, thus we get

$$\textsc{Guess}_{\mathsf{KDF},\mathcal{A}}(1^k) \leq \mathrm{negl}(k)\ .$$

This concludes the proof. □