FleXOR: Flexible garbling for XOR gates that beats free-XOR

Vladimir Kolesnikov*, Payman Mohassel, and Mike Rosulek**

[1] Bell Labs, kolesnikov@research.bell-labs.com
[2] University of Calgary, pmohasse@cpsc.ucalgary.ca
[3] Oregon State University, rosulekm@eecs.oregonstate.edu

Abstract. Most implementations of Yao's garbled circuit approach for 2-party secure computation use the *free-XOR* optimization of Kolesnikov & Schneider (ICALP 2008). We introduce an alternative technique called *flexible-XOR* (fleXOR) that generalizes free-XOR and offers several advantages. First, fleXOR can be instantiated under a weaker hardness assumption on the underlying cipher/hash function (related-key security only, compared to related-key and circular security required for free-XOR) while maintaining most of the performance improvements that free-XOR offers. Alternatively, even though XOR gates are not always "free" in our approach, we show that the other (non-XOR) gates can be optimized more heavily than what is possible when using free-XOR. For many circuits of cryptographic interest, this can yield a significantly (over 30%) smaller garbled circuit than any other known techniques (including free-XOR) or their combinations.

1 Introduction

This work proposes efficiency improvements of two-party Secure Function Evaluation (SFE). SFE allows two parties to evaluate any function on their respective inputs x and y, while maintaining privacy of both x and y. SFE of some useful functions today is borderline practical, and first uses of secure computation begin to crop up in industry. The main obstacle in SFE's wider adoption is the cost. Indeed, SFE of most of today's functions of interest is either completely out of reach of practicality, or carries costs sufficient to deter would-be adopters, who instead choose stronger trust models, entice users to give up their privacy with incentives, or use similar crypto-workarounds. We believe that truly practical efficiency is required for SFE to see use in real-life applications.

* Supported in part by the Intelligence Advanced Research Project Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract Number D11PC20194. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

** Supported by NSF award CCF-1149647.

Our results and motivation. We improve both the required assumptions and efficiency, albeit not both simultaneously, of a commonly used SFE tool, Garbled Circuit (GC).

On the practical side, our construction results in savings of GC size of over 30% (in garbled circuits typically analyzed in the literature) as compared to the state-of-the-art GC variant using the free-XOR technique of Kolesnikov & Schneider [15]. For a fundamental protocol, which has been studied and optimized for over three decades, this is a significant improvement. We emphasize that the fleXOR approach is more general than the specific instantiations we show, and we expect better optimizations to be discovered later on. At the same time, we prove that computing optimal instantiations (i.e. those minimizing the GC size) is NP-complete.

On the theoretical side, we aim to remove the Random Oracle (RO) requirement of the free-XOR technique without sacrificing efficiency. We weaken the RO assumption to that of correlation-robustness (CR) while retaining most of the performance improvements associated with free-XOR (only $10 - 20\%$ loss for analyzed circuits).[4] This choice is natural, motivated by several pragmatic considerations:

(1) Perhaps most importantly, today an efficient GC protocol will almost certainly use the OT extension of Ishai et al. [11]. Indeed, the orders of magnitude efficiency improvement brought by the IKNP OT extension transformed the field of secure computation. The OT extension, as well as its follow-up constructions, requires CR hash functions. Thus, our choice allows to avoid the introduction of any additional assumptions in most cases.

(2) Another important factor is the degree of analysis of the candidate implementations of the employed function. Cryptanalysts study at length related-key attacks for real-world block ciphers/primitives, but, to our knowledge, key circularity attacks are less researched.

Further, the question of understanding and reducing/eliminating the RO assumption associated with free-XOR is motivated by recent work. Choi et al. [5] shows that circular-correlation robustness is a sufficient condition for free-XOR. It also presents a black-box separation which demonstrates that CR is strictly weaker than circular-correlation robustness (which, in turn, is weaker than RO). Choi et al. [5] explicitly leave open the question: "is there a garbled-circuit variant where certain gates can be evaluated for free, without relying on assumptions beyond CPA-secure encryption?" Addressing this question, Applebaum [1] showed that free-XOR can be realized in the standard model under the learning parity with noise (LPN) assumption. While novel at the fundamental level, the efficiency of the protocol of [1] is far from practical.

Our work raises and addresses related questions: *Can the efficiency improvement of free XOR be extended? Can it be achieved under weaker assumptions?*

[4] In fact, there is no penalty at all for *formulas* (circuits with fan-out 1). That is, our approach matches the performance of free-XOR on formulas, but under the weaker correlation-robustness assumption.

Our metric: computation vs communication. In this work we focus on measuring performance by the size of the GC, a very clean and expressive metric. Since the associated computations are fast, we believe that in many (but not all, of course) practical scenarios communication complexity of our constructions will correlate with their total execution time. Indeed, in this work, we use aggressive (2-row) garbled row reduction (GRR2) due to Pinkas et al. [19], which involves computing polynomial interpolation. While more expensive than the standard PRF or hash function garbling, GRR2 nevertheless is a very efficient technique as evidenced by the performance experiments in [19]. GRR2 approach (denoted PRF-SS in the performance tables in [19]) is about 1x-3x times slower than the fastest experiment. However, note that a very fast 1Gbps network and a slow 2-core computer was used in [19]. Today, 1Gbps channel is still state-of-the-art, but computational power of a typical machine grew by factor of 4-6, mainly due to increased number of cores. Thus, we expect that today, the bottleneck of the [19] experiments would be in the network traffic, and not in the CPU load. This is even more likely to be so in the future, as historical hardware trends indicate faster advances in computational power than in network speeds.

At the same time, of course, specific use cases may dictate an extremely low-power CPU with an available fast network, which would imply different cost structure of our protocols. However, as argued above, today and in the expected future, communication performance is a good metric for our protocols.

1.1 Overview of Our Approach

In a garbled circuit, each wire receives a pair (A, B) of (bitstring) labels which conceptually encode TRUE and FALSE. Let us call $A \oplus B$ the **offset** of the wire. The idea behind the free XOR technique is to ensure that all wires have the same (secret) offset. Then the garbled gate can be evaluated by simply XOR-ing the wire labels.

FleXOR. With the idea of "wire offsets" in mind, consider the case where an XOR gate's input wires do not have the same wire offset. Intuitively, the free-XOR approach can be applied if we "translate" the incoming wire labels to bring them to the desired output offset. Namely, let the two input wires have wire labels $(A, A \oplus \Delta_1)$ and $(B, B \oplus \Delta_2)$, and suppose we would like the output wire labels to have offset Δ_3. We then select random "translated" wire values $\widetilde{A}, \widetilde{B}$. Let E be gate encryption function. Then we can garble this XOR gate with the following ciphertexts:

$$E_A(\widetilde{A}); \quad E_{A\oplus\Delta_1}(\widetilde{A} \oplus \Delta_3); \quad E_B(\widetilde{B}); \quad E_{B\oplus\Delta_2}(\widetilde{B} \oplus \Delta_3);$$

Now, the first two ciphertexts allow the evaluator to translate wire labels $(A, A\oplus \Delta_1)$ with offset Δ_1 into new ones $(\widetilde{A}, \widetilde{A} \oplus \Delta_3)$ of the desired offset Δ_3. Similarly the last two ciphertexts permit $(B, B \oplus \Delta_2) \rightsquigarrow (\widetilde{B}, \widetilde{B} \oplus \Delta_3)$. Now, these "translated" wire labels share the same offset Δ_3 and so the output labels $(\widetilde{A} \oplus \widetilde{B}, \widetilde{A} \oplus \widetilde{B} \oplus \Delta_3)$ can be obtained simply by XORing the "translated" labels.

So far we did not save anything: this method requires 4 ciphertexts to garble an XOR gate. However, we can reduce this cost with two simple observations:

- If we can arrange the wire label assignments so that $\varDelta_1 = \varDelta_3$, then the first two ciphertexts are not needed at all (the labels on this wire already have the correct offset). If $\varDelta_2 = \varDelta_3$, then the second two ciphertexts are not needed. Indeed, $\varDelta_1 = \varDelta_2 = \varDelta_3$, corresponds to the free-XOR case.
- Next, we can apply a standard garbled row-reduction technique (GRR) of [19]. The idea is that ciphertexts 1 & 3 above can always be set to the string 0^λ, implicitly setting $\widetilde{A} = D_A(0^\lambda)$ and $\widetilde{B} = D_B(0^\lambda)$, where D is the gate decryption function. Hence, ciphertexts 1 & 3 never need to be sent.

As a result, we obtain a method to garble XOR gates that requires 0, 1, or at most 2 ciphertexts total, depending on how many of $\{\varDelta_1, \varDelta_2, \varDelta_3\}$ are unique.[5] We call this method *flexible-XOR*, or **fleXOR** for short.

FleXOR application. We show how the fleXOR tool can be used to achieve the two goals motivating this work.

Consider grouping circuit wires into *equivalence classes*, where wires in the same equivalence class have the same offset. Since the arrangement of equivalence classes affects the cost of garbling each XOR gate, we are interested in assignments that minimize the total cost for all XOR gates.

If minimizing cost of XOR gates was the *only* constraint, then we could simply place all wires into a single equivalence class, and our construction in fact collapses to standard free-XOR. However, we consider additional constraints in class assignment, which result in the following improvements over the state-of-the-art GC (with free-XOR + GRR):

- **Performance improvement.** Recall, *row reduction* [19] is a technique for "compressing" a standard garbled gate from a size of 4 ciphertexts down to either 3 or 2. Free-XOR is compatible with the milder 3-ciphertext row reduction (which we call GRR3), but not with the more aggressive 2-ciphertext variant (GRR2). The problem is that gates garbled under GRR2 will have output wire labels with an unpredictable offset — it is not possible to force them to use the global wire offset $\varDelta$ used by free-XOR. In contrast, our fleXOR generalization does not force any specific wires to share the same offset hence there is no inherent incompatibility with using GRR2. Nevertheless it is necessary to put some constraints on the class assignment (a "safety" property that we define). We propose a heuristic algorithm for obtaining a safe assignment, and use it to obtain significant reduction in the GC size, in the experiments we run.

[5] Our high-level description does not indicate how to garble an XOR gate using just one ciphertext in the case that $\varDelta_1 = \varDelta_2 \neq \varDelta_3$. This is indeed possible using similar techniques (perform free XOR on the input wires, since they share a common offset, and then, with one ciphertext, adjust the result to $\varDelta_3$). However, our wire-ordering heuristics never produce XOR gates with this property, hence we do not consider this case throughout the writeup.

– **Weakened assumptions.** In the free-XOR world, the non-XOR gates are garbled by encrypting plaintexts $X, X \oplus \Delta$ using combinations of keys $Y, Y \oplus \Delta$. The appearance of a secret value Δ as both a key and plaintext requires a circularity assumption on the gate-level cipher [5]. With an appropriate constraint (i.e. monotonicity property) on wire equivalence classes, we can ensure that wire labels from the class indexed i are used as keys to encrypt wire labels only from a class indexed $j > i$. Under this additional constraint, our construction can be instantiated under a significantly weaker (related-key only, not circular) hardness assumption than free-XOR. At the same time, our experiments show only mild performance loss as compared to state-of-the-art algorithms needing circularity assumption.

Recall that fleXOR easily collapses to free-XOR when grouping all wires in the same class. We view this as an important feature of our scheme. In terms of size of garbled circuits, free-XOR performs better in some settings while the new fleXOR method performs better in others. By adopting and implementing fleXOR, one can always have available both options, and seamlessly choose the best method via appropriate choice of wire equivalence classes.

1.2 Organization of Paper

After discussing related work (Section 1.3) and preliminaries (Section 2), we set up the required technical details. In Section 3, we formalize the notion of gate cipher and show that it can be instantiated with RO and correlation-robust (CR) functions. In Section 4, we explicitly write our circuit garbling scheme in the recent "garbling schemes" convention [3], and provide a proof of security with a concrete reduction to the security of the underlying gate cipher. In Section 5 we explicitly integrate garbled row reductions from [19] into the garbling protocols and prove security via concrete reductions.

Once this set up is in place, in Section 6 we present two algorithms for assigning wire classes. One, achieving what we call monotone ordering, allows us to avoid circularity in key applications. The second, more performance-oriented, achieving what we call safe ordering, allows our garbling protocols to generate GC up to and over 30% smaller than currently best known.

In Section 7, we provide detailed performance comparison of both of our heuristic algorithms.

1.3 Related work

Garbled circuit is a general and an extremely efficient technique of secure computation, requiring only one round of interaction in the semi-honest model. Due to this generality and practicality, GC and related protocols have been receiving a lot of attention in the literature.

The basic GC is so simple and minimal that it has proven hard to improve. Most of the GC research considers its application to solving problems at hand, such as set intersection, auction design, etc. A much smaller number of papers deal with technical improvements to GC-based two-party SFE, such as OT extension [11, 14] or cut-and-choose improvements for malicious case [10, 16, 17].

Our work belongs to a third category, aiming to improve and understand the garbling scheme itself. Since the original paper of Yao over 30 years ago, only a few works fit into this category. Beaver et al. [2] introduced the point-and-permute idea, which allows the evaluator to decrypt just a single ciphertext in the garbled gate. Naor et al. [18] introduced 3-row garbled row reduction optimization. Kolesnikov and Schneider [15] introduced the popular free-XOR technique allowing XOR gates to be evaluated without cost. Pinkas et al. [19] introduced 2-row GRR and observed that GRR3 is compatible with free-XOR. Choi et al. and Appelbaum helped clarify the underlying assumptions for free-XOR, now seen as a natural part of GC. Choi et al. [5] weakened the free-XOR assumption, by defining a sufficient gate cipher property, circular security. Applebaum [1] showed how to implement free-XOR in the standard model (using the LPN assumption, and hence not competitive with today's standard GC).

In related but incomparable work, Kolesnikov and Kumaresan [13] obtained approximately 3x factor performance improvement over state-of-the-art GC by evaluating slices of information-theoretic GC of Kolesnikov [12]. Their protocol has linear number of rounds and is not secure against malicious evaluator. We also mention, but do not discuss in detail multi-party SFE such as [9, 8, 6].

Bellare et al. [3] introduced the *garbling schemes* abstraction, which we use here.

2 Preliminaries

2.1 Code-based Games

We use the convention of code-based games [4]: A game $\mathcal{G}$ starts by executing the Initialize procedure. Then the adversary $\mathcal{A}$ is invoked and allowed to query the procedures that comprise the game. When the adversary halts, the Finalize procedure is called with the output of the adversary. The output of the Finalize procedure is taken to be the outcome of the game, whose random variable we denote by $\mathcal{G}^{\mathcal{A}}(\lambda)$, where λ is the global security parameter.

2.2 Garbling Schemes

Bellare, Hoang, and Rogaway [3] introduce the notion of a garbling scheme as a cryptographic primitive. We refer the reader to their work for a complete treatment and give a brief summary here.[6] A garbling scheme consists of the following algorithms: Garble takes a circuit f as input and outputs (F, e, d) where F is a garbled circuit, e is encoding information, and d is decoding information. Encode takes an input x and encoding information e and outputs a garbled input X. Eval takes a garbled circuit F and garbled input X and outputs a garbled

[6] Their definitions apply to any kind of garbling, but we specify the notation for *circuit* garbling.

output Y. Finally, Decode takes a garbled output Y and decoding information d and outputs a plain circuit-output (or an error $\perp$).

Our work uses the $\mathsf{prv.sim}$ (privacy), $\mathsf{obv.sim}$ (obliviousness), and aut (authenticity) security definitions from [3], which we state below. In the $\mathsf{prv.sim}$ and $\mathsf{obv.sim}$ games, the Initialize procedure chooses $\beta \leftarrow \{0,1\}$, and the Finalize(β') procedure returns $\beta \overset{?}{=} \beta'$. In all three games, the adversary can make a single call to the Garble procedure, which is defined below. Additionally, the function Φ denotes the information about the circuit that is allowed to be leaked by the garbling scheme; the function $\mathcal{S}$ is a simulator, and G denotes a garbling scheme.

```
$\mathsf{prv.sim}_{G,\Phi,\mathcal{S}}$:

Garble$(f, x)$:
if $\beta = 0$
    $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
    $X \leftarrow \mathsf{Encode}(e, x)$
else $(F, X, d) \leftarrow \mathcal{S}(1^\lambda, f(x), \Phi(f))$
return $(F, X, d)$
```

```
$\mathsf{obv.sim}_{G,\Phi,\mathcal{S}}$:

Garble$(f, x)$:
if $\beta = 0$
    $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
    $X \leftarrow \mathsf{Encode}(e, x)$
else $(F, X) \leftarrow \mathcal{S}(1^\lambda, \Phi(f))$
return $(F, X)$
```

```
$\mathsf{aut}_G$:
Garble$(f, x)$:
$(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
$X \leftarrow \mathsf{Encode}(e, x)$
return $(F, X)$

Finalize$(Y)$:
return $\mathsf{Decode}(d, Y) \notin \{\perp, f(x)\}$
```

We then define the advantage of the adversary in the three security games:

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G,\Phi,\mathcal{S}}(\mathcal{A}, \lambda) := \left| \Pr[\mathsf{prv.sim}^{\mathcal{A}}_{G,\Phi,\mathcal{S}}(\lambda) = 1] - \frac{1}{2} \right| ;$$

$$\mathsf{Adv}^{\mathsf{obv.sim}}_{G,\Phi,\mathcal{S}}(\mathcal{A}, \lambda) := \left| \Pr[\mathsf{obv.sim}^{\mathcal{A}}_{G,\Phi,\mathcal{S}}(\lambda) = 1] - \frac{1}{2} \right| .$$

$$\mathsf{Adv}^{\mathsf{aut}}_{G}(\mathcal{A}, \lambda) := \Pr[\mathsf{aut}^{\mathcal{A}}_{G}(\lambda) = 1];$$

3 Our Gate-Level Cipher Abstraction

Yao's technique conceptually garbles each gate with "boxes locked via two keys." We adopt the approach used by [19] and elsewhere, in which gates are garbled as $H(w_i \| w_j \| T) \oplus w_k$, where w_i, w_j are wire labels on input wires, T is a tweak/nonce, w_k is a wire label of an output wire, and H is a key-derivation function. We now describe more specifically what property is needed of H.

3.1 Definitions

We define two security games formally. They are parameterized by a KDF $H : \{0,1\}^* \to \{0,1\}^{\lambda+1}$. Game $\mathsf{kdf.rk}_{H,n}$ includes the $\boxed{\text{boxed}}$ statement, and $\mathsf{kdf.circ}_{H,n}$ excludes the boxed statement.

```
Initialize:
    $\varDelta_1, \ldots, \varDelta_n \leftarrow \{0,1\}^\lambda$
    $\varDelta_0 := \varDelta_\infty := 0^\lambda$
    $\beta \leftarrow \{0,1\}$

Finalize($\beta'$):
    return $\beta' \overset{?}{=} \beta$

Fn($X, Y, a, b, c, T$):
    return $\bot$ if $T$ previously used in any Fn query
        or $\{a, b\} \subseteq \{0, \infty\}$ $\boxed{\text{or } c \le \max\{a, b\}}$
    if $\beta = 0$ then $Z := H(X \oplus \varDelta_a, Y \oplus \varDelta_b, T) \oplus (\varDelta_c \| 0)$
        else $Z \leftarrow \{0,1\}^{\lambda+1}$
    return $Z$
```

Briefly, the games proceed as follows. The challenger generates n random (secret) wire offsets $\{\varDelta_i\}_i$, where n is a parameter of the game. The values $\varDelta_0 := \varDelta_\infty := 0^\lambda$ are set as a convenience.

The adversary can then make queries of the form $H(X \oplus \varDelta_a, Y \oplus \varDelta_b, T) \oplus \varDelta_c$, provided that at least one of $\{\varDelta_a, \varDelta_b\}$ is unknown (i.e., $a, b \notin \{0, \infty\}$), and the tweak values T are never reused. The result of this expression should be indistinguishable from random.

In the $\mathsf{kdf.rk}$ variant of the game, there is an additional "monotonicity" restriction, that $c > \max\{a, b\}$, which prevents the adversary from invoking "key cycles" among the secret $\varDelta_i$ values. It is in this setting that having two values $\varDelta_0$ and $\varDelta_\infty$ is convenient. A query of the form $H(X, Y \oplus \varDelta_i, T)$ can be made via $a = 0$, $b = i$, $c = \infty$, so that the monotonicity condition is satisfied ($c = 0$, for example, would break monotonicity).

Definition 1. *Let $H : \{0,1\}^* \to \{0,1\}^{\lambda+1}$ be a KDF, $\mathcal{A}$ be a PPT adversary, and the games $\mathsf{kdf.rk}_{H,n}$, $\mathsf{kdf.circ}_{H,n}$ be defined as above. We then define the advantage of the adversary in these games as:*

$$\mathsf{Adv}^{\mathsf{kdf.rk}}_{H,n}(\mathcal{A}, \lambda) := \left| \Pr[\mathsf{kdf.rk}^{\mathcal{A}}_{H,n}(\lambda) = 1] - \frac{1}{2} \right|;$$
$$\mathsf{Adv}^{\mathsf{kdf.circ}}_{H,n}(\mathcal{A}, \lambda) := \left| \Pr[\mathsf{kdf.circ}^{\mathcal{A}}_{H,n}(\lambda) = 1] - \frac{1}{2} \right|$$

Single-key vs. Dual-key. In our main construction, we garble XOR gates using only one key (wire label) and non-XOR gates using two keys (wire labels). We let H^2 be a synonym for H, and define shorthand:

$$H^1(K, T) \overset{\text{def}}{=} H^2(K, K, T)[1..\lambda]$$

We take only the first λ bits of the output for H^1 because we do not need the 1 extra bit in our construction when using H^1 (the extra bit is used for the permute bit, which is easier to handle for XOR gates).

Since H^1 takes a shorter input than H^2, it is conceivable that H^1 could be implemented more efficiently than H^2 in practice (e.g., invoking a hash function with a smaller input and hence fewer iterations). However, this kind of optimization not the focus of our work.

3.2 Instantiation from a Random Oracle

Lemma 1. *Let $H : \{0,1\}^* \to \{0,1\}^{\lambda+1}$ be a random oracle. Then for all $\mathcal{A}$, we have $\mathsf{Adv}^{\mathsf{kdf.circ}}_{H,n}(\mathcal{A},\lambda) \leq 16n(q_A + q_C)^2/2^\lambda$, where q_A, q_C are the number of queries made to the random oracle (locally) and to the Fn procedure, respectively, by $\mathcal{A}$ (and n is the parameter of the security game).*

3.3 Instantiation from Correlation-Robustness

The free-XOR approach was formally proven secure in the RO model, and believed secure under some (unspecified) variant of correlation-robustness [11]. Choi et al [5] showed that the most natural variant of correlation-robustness (called *2-correlation-robust*) was in fact insufficient for free-XOR. Below we have translated their definition to the framework of code-based games. We then show that 2-correlation-robustness is sufficient for $\mathsf{kdf.rk}$ security.

Definition 2 (adapted from [5]). *Let $H : \{0,1\}^* \to \{0,1\}^{\lambda+1}$ be a hash function.*[7] *Define $\mathsf{Adv}^{\mathsf{2corr}}_{H}(\mathcal{A},\lambda) := |\Pr[\mathsf{2corr}^{\mathcal{A}}_{H}(\lambda) = 1] - \frac{1}{2}|$, where $\mathsf{2corr}_H$ is the game defined as follows:*

```
Initialize:
  $\Delta \leftarrow \{0,1\}^\lambda$
  $\beta \leftarrow \{0,1\}$

Finalize($\beta'$):
  return $\beta \overset{?}{=} \beta'$

Fn($X, Y, T$):
  return $\bot$ if this query previously made
  if $\beta = 0$ then $Z_1 := H(X \oplus \Delta, Y \oplus \Delta, T)$
                      $Z_2 := H(X \oplus \Delta, Y, T)$
                      $Z_3 := H(X, Y \oplus \Delta, T)$
  else $Z_1, Z_2, Z_3 \leftarrow \{0,1\}^\lambda$
  return $Z_1, Z_2, Z_3$
```

Lemma 2. *For all probabilistic polynomial-time $\mathcal{A}$, we have $\mathsf{Adv}^{\mathsf{kdf.rk}}_{\Sigma,n}(\mathcal{A},\lambda) \leq n \cdot \mathsf{Adv}^{\mathsf{2corr}}_{R}(\mathcal{A}',\lambda)$, where $\mathcal{A}'$ has comparable runtime to $\mathcal{A}$.*

4 Baseline Construction

We now present our "basic" fleXOR garbling scheme. It requires some auxiliary information about the circuit, defined below:

Definition 3. *A* **wire ordering** *for a boolean circuit C is a function $\mathcal{L}$ that assigns an integer to each wire in C. 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 that $\mathcal{L}$ is* **monotone** *if:*

1. *for each XOR gate, with input wires i & j and output wire k: $\mathcal{L}(k) \geq \max\{\mathcal{L}(i),\mathcal{L}(j)\}$, and*

[7] H may be drawn from a family of hash functions, but for simplicity we refer to H as a single function.

```
$\mathsf{Garble}(1^\lambda, C, \mathcal{L})$ :
for $\ell = 1$ to $|\mathcal{L}|$: $\Delta_\ell \leftarrow \{0,1\}^\lambda$
for each input bit $i$ corresponding to wire $j$ of $C$:
  $b_j \leftarrow \{0,1\}$
  $w_j^0 \leftarrow \{0,1\}^\lambda$;   $w_j^1 := w_j^0 \oplus \Delta_{\mathcal{L}(j)}$
  for $v \in \{0,1\}$: $e[i,v] := w_j^{v \oplus b_j} \| b_j$
for each gate $g$ in $C$, in topological order:
  let $i, j$ denote $g$'s input wires
  let $k$ denote $g$'s output wire
  if $g$ is an XOR gate:
    if $\mathcal{L}(i) \neq \mathcal{L}(k)$:
      $\widetilde{w}_i^0 \leftarrow \{0,1\}^\lambda$;   $\widetilde{w}_i^1 := \widetilde{w}_i^0 \oplus \Delta_{\mathcal{L}(k)}$
      for $b \in \{0,1\}$: $c_{0,b} := H^1(w_i^b, g\|0\|b) \oplus \widetilde{w}_i^b$
    else for $b \in \{0,1\}$: $\widetilde{w}_i^b := w_i^b$;   $c_{0,b} := \perp$
    if $\mathcal{L}(j) \neq \mathcal{L}(k)$:
      $\widetilde{w}_j^0 \leftarrow \{0,1\}^\lambda$;   $\widetilde{w}_j^1 := \widetilde{w}_j^0 \oplus \Delta_{\mathcal{L}(k)}$
      for $b \in \{0,1\}$: $c_{1,b} := H^1(w_j^b, g\|1\|b) \oplus \widetilde{w}_j^b$
    else for $b \in \{0,1\}$: $\widetilde{w}_j^b := w_j^b$;   $c_{1,b} := \perp$
    $w_k^0 := \widetilde{w}_i^0 \oplus \widetilde{w}_j^0$;   $w_k^1 := \widetilde{w}_i^1 \oplus \widetilde{w}_j^0$;
    $b_k := b_i \oplus b_j$
  else $g$ computes logic $G : \{0,1\}^2 \to \{0,1\}$:
    $b_k \leftarrow \{0,1\}$
    $w_k^0 \leftarrow \{0,1\}^\lambda$; $w_k^1 := w_k^0 \oplus \Delta_{\mathcal{L}(k)}$
    for $a, b \in \{0,1\}^2$:
      $v := b_k \oplus G(a \oplus b_i, b \oplus b_j)$
      $c_{a,b} = H^2(w_i^a, w_j^b, g\|a\|b) \oplus w_k^v \| v$
  $F[g] := (c_{00}, c_{01}, c_{10}, c_{11})$
for each output bit $i$ corresponding to wire $j$ of $C$:
  for $v \in \{0,1\}$: $d[i,v] := H^1(w_j^{v \oplus b_j}, \mathtt{out}\|j\|v)$
return $(F, e, d)$
```

```
$\mathsf{Encode}(e, x)$ :
for $i = 1$ to $|x|$: $X[i] := e[i, x_i]$
return $X$

$\mathsf{Eval}(F, X)$ :
for each input wire $i$ in $C$:
  $w_i^* \| b_i^* \leftarrow X[i]$
for each gate $g$ in $C$, in topological order:
  let $i, j$ denote $g$'s input wires
  let $k$ denote $g$'s output wire
  parse $F[g]$ as $(c_{00}, c_{01}, c_{10}, c_{11})$
  if $g$ is an XOR gate:
    if $c_{01} = \perp$ then $\widetilde{w}_i^* := w_i^*$
      else $\widetilde{w}_i^* := H^1(w_i^*, g\|0\|b_i^*) \oplus c_{0,b_i^*}$
    if $c_{11} = \perp$ then $\widetilde{w}_j^* := w_j^*$
      else $\widetilde{w}_j^* := H^1(w_j^*, g\|1\|b_j^*) \oplus c_{1,b_j^*}$
  $w_k^* := w_i^* \oplus w_j^*$;   $b_k^* := b_i^* \oplus b_j^*$
  else:
    $w_k^* \| b_k^* := H^2(w_i^*, w_j^*, g\|b_i^*\|b_j^*) \oplus c_{b_i^*, b_j^*}$
for each output bit $i$ in $C$:
  let $j$ be the corresponding wire
  $Y[i] := H^1(w_j^*, \mathtt{out}\|j\|b_j^*)$
return $Y$

$\mathsf{Decode}(Y, d)$ :
for $i = 1$ to $Y.len$:
  if $Y[i] = d[i,0]$ then $y_i = 0$
  elsif $Y[i] = d[i,1]$ then $y_i = 1$
  else return $\perp$
return $y$
```

Fig. 1. Our baseline garbling scheme

2. *for each non-XOR gate, with input wires i & j and output wire k:* $\mathcal{L}(k) > \max\{\mathcal{L}(i), \mathcal{L}(j)\}$.

We now give the complete description of our garbling scheme. Following [3], the scheme consists of 4 algorithms: Garble, Encode, Eval, Decode. We make one syntactic change, and allow Garble to accept as input auxiliary information $\mathcal{L}$, which is a wire ordering of the given circuit.

The scheme is described formally in Figure 1. It follows the typical Yao approach for garbling a circuit. Briefly, for each wire i, the garbler chooses two wire labels w_i^0, w_i^1 such that $w_i^0 \oplus w_i^1 = \Delta_{\mathcal{L}(i)}$. We use the point-and-permute bit optimization of [18], where a permute bit b_i is chosen so that $w_i^{b_i}$ encodes FALSE on wire i, and $w_i^{1 \oplus b_i}$ encodes TRUE. Non-XOR gates are garbled in the usual way.

XOR gates use the approach described in the introduction. Namely, suppose an XOR gate has input wires i, j and output wire k. If $\mathcal{L}(i) = \mathcal{L}(k)$, then no action is required for wire i in this gate (and no ciphertexts are included in the garbled circuit). Otherwise, we choose "adjusted" wire labels $\widetilde{w}_i^0, \widetilde{w}_i^1$ whose offset is the target value $\Delta_{\mathcal{L}(k)}$ and provide two ciphertexts that allow the evaluator to obtain $\widetilde{w}_i^b$ from w_i^b. The same logic applies for input wire j, and finally a "free XOR" is performed on these adjusted wire labels.

Theorem 1. *Let $G[H]$ denote our garbling scheme (Figure 1), where H is a KDF. Let Φ denote the side information function that leaks the circuit topology, distinction between XOR vs non-XOR gates (but not distinctions among non-XOR gates), and the wire ordering function $\mathcal{L}$ used. Then, for all probabilistic polynomial-time $\mathcal{A}$, there exists a polynomial-time simulator $\mathcal{S}$ such that:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G[H],\Phi,\mathcal{S}}(\mathcal{A},\lambda) \leq \mathsf{Adv}^{\mathsf{kdf.circ}}_{H,|\mathcal{L}|}(\mathcal{A}',\lambda)$$

where $\mathcal{A}'$ has runtime essentially the same as $\mathcal{A}$. Furthermore, when the wire ordering function $\mathcal{L}$ is **monotone**, *we have:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G[H],\Phi,\mathcal{S}}(\mathcal{A},\lambda) \leq \mathsf{Adv}^{\mathsf{kdf.rk}}_{H,|\mathcal{L}|}(\mathcal{A}',\lambda)$$

5 Incorporating Row Reductions

Row-reduction optimizations were introduced by Naor et al. [18] and later formalized and extended by Pinkas et al. [19]. They describe two flavors of row reduction, which we discuss and adapt to our fleXOR technique.

5.1 Optimization 1: Mild Row Reduction

In the first variant of row reduction, Naor et al. describe how to reduce standard 4-ciphertext garbled gates to 3 ciphertexts. Conceptually, this is done by fixing one of the ciphertexts to be the all-zeroes string. The idea is that if, say, c_{00} is known to always consist of all zeroes, then it does not actually need to be included in the garbled output.

For example, when garbling a non-XOR gate we see that ciphertext c_{00} will be zero if the appopriate output wire label (concatenated with its permute bit) is chosen to be $H^2(w_i^0, w_j^0, g\|00)$, which is the value that would be used to mask that wire label.

Hence, instead of choosing wire labels and permute bits uniformly, we choose one wire label to be an output of the KDF H and set the other label so that the two labels have the desired offset. We can use this idea with our XOR gates as well, following the ideas described in the introduction. Recall that to garble an XOR gate, we choose random "adjusted" wire labels for each input wire (whose offset requires adjusting). Instead of choosing these adjusted wire labels uniformly, we choose them to be the appropriate output of the KDF.

The formal description of this optimization is given in the full version. When garbling XOR gates, the ciphertexts c_{00}, c_{10} are always empty (implicitly set to all zeroes). Hence, XOR gates require 0, 1, or 2 ciphertexts. For non-XOR gates, the ciphertext c_{00} is always empty (implicitly set to all zeroes), so these gates require 3 ciphertexts.

That this optimization requires no additional properties of the wire ordering, and it achieves essentially identical security to our baseline construction:

Theorem 2. *Let $G^1[H]$ denote our "optimization #1" garbling scheme described above. Let Φ be as in Theorem 1. Then, for all probabilistic polynomial-time $\mathcal{A}$, there exists a polynomial-time simulator $\mathcal{S}$ such that:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G^1[H],\Phi,\mathcal{S}}(\mathcal{A},\lambda) \leq \mathsf{Adv}^{\mathsf{kdf.circ}}_{H,|\mathcal{L}|}(\mathcal{A}',\lambda)$$

where $\mathcal{A}'$ has runtime essentially the same as $\mathcal{A}$. Furthermore, when the wire ordering function $\mathcal{L}$ is **monotone***, we have:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G^1[H],\Phi,\mathcal{S}}(\mathcal{A},\lambda) \leq \mathsf{Adv}^{\mathsf{kdf.rk}}_{H,|\mathcal{L}|}(\mathcal{A}',\lambda)$$

5.2 Optimization 2: Aggressive Row Reduction

The second variant of row reduction reduces each garbled gate from 4 to 2 ciphertexts. Here we consider applying this optimization to the non-XOR gates in our scheme. This optimization has the effect of setting both output wire labels (and hence, their offset) implicitly. Superficially, this seems at odds with our approach, in which we always choose wire labels to have some desired offset.

However, suppose that g is a non-XOR gate with output wire k. If we process this gate before any other wire i with $\mathcal{L}(i) = \mathcal{L}(k)$, then we can indeed set the offset $\Delta_{\mathcal{L}(k)}$ implicitly based on the result of the row-reduction applied to this gate. If we process the gates in a topological order, one can capture this property by requiring that $\mathcal{L}(k) > \mathcal{L}(j)$ for every wire j that influences k (i.e. j has to be processed before k). We will also require that no other non-XOR gate in the circuit has output wire k' with $\mathcal{L}(k) = \mathcal{L}(k')$, though XOR gates can safely have this property.

The necessary properties on the wire ordering are summarized in the following definition:

Definition 4. *We say that $\mathcal{L}$ is* **safe** *if:*

1. *for each non-XOR gate g with output wire k, and each wire j that influences*[8] *g, we have $\mathcal{L}(k) > \mathcal{L}(j)$.*
2. *for each value ℓ, there is at most one non-XOR gate whose output wire k satisfies $\mathcal{L}(k) = \ell$.*

Note that a wire ordering may be any combination of safe/non-safe, monotone/non-monotone.

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

Assuming that $\mathcal{L}$ is safe, we can garble all non-XOR gates using only two ciphertexts, plus 4 additional bits. XOR gates still require 0, 1, or 2 ciphertexts as in the previous section.

[8] A wire j influences a wire k if there is a directed path in the circuit that contains wire j before wire k.

Our approach for row-reduction is the same as [19], but we give a short overview here in the interest of completeness. For simplicity, we assume that all non-XOR gates compute boolean-AND logic. Briefly, for each (a, b), we compute $V_{ab} = H^2(w_i^a, w_j^b, g\|a\|b)$. Hence, only one V_{ab} value is accessible to the evaluator. If the evaluator obtains V_{ab} with $(a, b) = (\bar{b}_i, \bar{b}_j)$, then the evaluator has TRUE on both input wires and hence this V_{ab} should allow the evaluator to obtain the "TRUE" output wire label $w_k^{1\oplus b_k}$. All other V_{ab} values should allow the evaluator to obtain the "FALSE" label $w_k^{b_k}$.

To make this work, let P be the degree-2 polynomial that passes through the 3 points of the form $(2a + b, V_{ab})$, for the (a, b) pairs which are supposed to yield $w_k^{b_k}$. Then let Q be the degree-2 polynomial that passes through the points $(4, P(4))$, $(5, P(5))$, and the point $(2a + b, V_{ab})$ for the "other" pair (a, b). The idea is that we can give the evaluator the values $P(4)$ and $P(5)$. When combined with his unique V_{ab} value, he can interpolate to obtain either the polynomial P or Q, depending on the output logic of the gate. Hence, we can set the two wire labels to be points on P and Q respectively, say, $P(-1)$ and $Q(-1)$.

The formal description of this optimization is given in the full version. We must also account for the permute bits, which require 4 extra bits. Overall, each AND-gate requires $2\lambda + 4$ bits, while XOR-gates still require 0, λ, or 2λ bits. We require the garbling procedure to process gates in a *safety-respecting* topological order, which ensures that Δ_ℓ gets set (while garbling an AND-gate) before it is used when later garbling an XOR gate.

Theorem 3. *Let $G^2[H]$ denote our "optimization #2" garbling scheme described above. Let Φ be as in Theorem 1. Then, for all probabilistic polynomial-time $\mathcal{A}$, there exists a polynomial-time simulator $\mathcal{S}$ such that:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G^2[H],\Phi,\mathcal{S}}(\mathcal{A}, \lambda) \leq (n + 1) \cdot \mathsf{Adv}^{\mathsf{kdf.circ}}_{H,|\mathcal{L}|}(\mathcal{A}', \lambda)$$

where $\mathcal{A}'$ has runtime essentially the same as $\mathcal{A}$. Furthermore, when the wire ordering function $\mathcal{L}$ is **monotone***, we have:*

$$\mathsf{Adv}^{\mathsf{prv.sim}}_{G^2[H],\Phi,\mathcal{S}}(\mathcal{A}, \lambda) \leq (n + 1) \cdot \mathsf{Adv}^{\mathsf{kdf.rk}}_{H,|\mathcal{L}|}(\mathcal{A}', \lambda)$$

5.3 GRR2-Salvaging

In general, it is not possible to combine fleXOR garbling with aggressive row reduction if the wire ordering is non-safe. Nevertheless, we observe that it is possible to garble *one* non-XOR gate in each $\mathcal{L}$-equivalence class using aggressive row reduction. Roughly speaking, for each value ℓ, we identify the topologically first non-XOR gates g whose output wire i satisfies $\mathcal{L}(i) = \ell$. We ensure that g is processed before any other such gates, garble it with GRR2, and use the result to implicitly set Δ_ℓ. The remaining gates in g's equivalence class can then be garbled using GRR3.

This approach slightly generalizes our construction in the previous section. It provides a modest reduction in size, which we discuss in Section 7.

6 Optimizing the Choice of Wire Orderings

We have identified two types of wire orderings for use with our fleXOR construction: *monotone* and *safe* ordering. In this section, we consider the problem of optimizing the choice of wire ordering: i.e., a safe/monotone wire ordering that minimizes the size of the fleXOR-garbled circuit. In particular, we need only consider the total size of garbled XOR gates. An XOR gate with input wires i and j and output wire k, requires two ciphertexts if $\mathcal{L}(i) \neq \mathcal{L}(k)$ and $\mathcal{L}(j) \neq \mathcal{L}(k)$, requires one ciphertext if only one of the inequalities holds, and is "free" (no ciphertexts) if $\mathcal{L}(i) = \mathcal{L}(j) = \mathcal{L}(k)$.

6.1 Monotone Orderings

We start by showing that the problem of finding an *optimal monotone ordering* of a circuit is NP-complete. In particular, we prove the following theorem in the full version, via a simple reduction to 3SAT.

Theorem 4. *The following problem is NP-complete: Given a circuit C and integer N, determine whether there is a monotone wire ordering of C for which garbling the XOR gates using the fleXOR scheme requires at most N ciphertexts.*

It is, however, easy to find at least *some* monotone wire ordering, using an elementary linear-time algorithm. First, assign each input wire i to $\mathcal{L}(i) = 1$. Then process the gates in topological order and assign to each output wire the minimum $\mathcal{L}$ allowed by the monotonicity condition. We mention this simple approach only because it can be computed on the fly at basically no expense, in the same pass that garbles the circuit. This may be important in memory-critical applications where circuits are processed via streaming.

In Figure 2, we propose a better heuristic for monotone orderings, inspired by the following observation. Note that it is only the non-XOR gates which necessarily increase the wire ordering number between input and output wires of a gate. Define the **non-XOR-depth** of a wire i in a circuit C as the maximum number of non-XOR gates among all directed paths from i to an output wire. The non-XOR-depth of every gate in a circuit can be computed via a simple dynamic programming approach. Then, we define a wire-ordering function $\mathcal{L}$ so that $\mathcal{L}(i) + \textsf{non-XOR-depth}(i)$ is constant for all wires i. Hence, wires closer to the outputs receive higher wire-ordering. This heuristic is in fact optimal, and results in *all* XOR gates free, when the circuit has fan-out 1 (i.e., the circuit encodes a *formula*). It is also not hard to prove that it minimizes the size of the range of the wire-ordering function hence (intuitively) increasing the likelihood of the input and output wires of an XOR gate being in the same class.

We further refine this heuristic by revisiting each XOR gate one more time, in topological order, and reducing the order of each output wire to maximum of orders of its input wires (if this is not already the case). If done in topological order, this does not affect the monotonicity of the ordering.

```
for every wire $i$:
    compute non-XOR-depth$[i]$
set $\Lambda = 1$ + num wires in circuit
for each wire $i$:
    set $\mathcal{L}[i] := \Lambda -$ non-XOR-depth$[i]$
for each XOR gate $g$ in topo. order:
    denote $g$'s inputs wires by $i, j$
    denote $g$'s output wire by $k$
    if $\mathcal{L}[k] > \max\{\mathcal{L}[i], \mathcal{L}[j]\}$
        set $\mathcal{L}[k] := \max\{\mathcal{L}[i], \mathcal{L}[j]\}$
```

Fig. 2. Monotone wire ordering heuristic

```
for each input wire $i$:
    set $\mathcal{L}[i] := 1$
set $count := 2$
for each gate $g$, in topo. order:
    denote $g$'s output wire by $k$
    if $g$ is an XOR gate:
        set $\mathcal{L}[k] := 1$
    else:
        set $\mathcal{L}[k] := count$
        $count := count + 1$
```

Fig. 3. Safe wire ordering heuristic

Proposition 5 *The algorithm of Figure 2 computes a monotone wire ordering in linear time.*

We implemented both heuristic algorithms for monotone orderings, and tested them on a wide range of circuits. In general, our second heuristic algorithm outperforms the elementary one by 20-40% (in terms of average cost per XOR gate).

6.2 Safe Orderings

The constraints for safe wire ordering are fairly strict, making it challenging to devise good heuristic algorithms that minimize the number ciphertexts needed to garble XOR gates. Nevertheless, we introduce a simple and intuitive algorithm that performs well in practice as demonstrated in our analysis in the following section.

Since the output wires of non-XOR gates must have distinct $\mathcal{L}$-values in a safe ordering, our idea is to assign such wires values incrementally, and in topological order, starting from 2. Then, for each XOR gate, we let the $\mathcal{L}$-value of its output wire be 1 (see Figure 3). The resulting ordering will always satisfy the definition of a safe ordering. In particular, if wire i influences a non-XOR gate with output wire j, then $\mathcal{L}(i) < \mathcal{L}(j)$, either by the topological constraint (when wire i emanates from a non-XOR gate), or because $\mathcal{L}(i) = 1 < \mathcal{L}(j)$ (when i emanates from an XOR gate).

Proposition 6 *The algorithm of Figure 3 computes a safe wire ordering in linear time.*

6.3 Other Constraints for Wire Orderings

Here we considered safe and monotone orderings separately, but we note that it is possible (and interesting) to consider their combination i.e. optimization problems for orderings that are both safe and monotone. We leave open the problem of designing good heuristics for this problem.

As mentioned earlier, using a trivial wire ordering (all wires assigned the same index) causes fleXOR construction to collapse to free-XOR.

			fleXOR		
circuit	GRR2	free-XOR	monotone	safe	best
DES	2.0 (2.0)	2.79 (0.0)	2.84 (0.93)	**1.89** (0.38)	1.89
AES	2.0 (2.0)	**0.64** (0.0)	0.76 (0.15)	0.72 (0.37)	0.64
SHA-1	2.0 (2.0)	1.82 (0.0)	2.02 (0.75)	**1.39** (0.45)	1.39
SHA-256	2.0 (2.0)	2.05 (0.0)	2.26 (0.76)	**1.56** (0.60)	1.56
Hamming distance	2.0 (2.0)	**0.50** (0.0)	0.67 (0.20)	**0.50** (0.20)	0.50
minimum in set	2.0 (2.0)	**0.87** (0.0)	1.01 (0.41)	**0.87** (0.41)	0.87
32×32 fast mult	2.0 (2.0)	**0.90** (0.0)	1.15 (0.36)	0.94 (0.49)	0.90
1024-bit millionaires	2.0 (2.0)	**1.00** (0.0)	1.08 (0.25)	**1.00** (0.50)	1.00

Fig. 4. Comparison of standard garbling (with GRR2 row reduction), free-XOR, and fleXOR instantiations. The main number in each cell shows average number of ciphertexts per gate; the number in the parentheses shows average number of ciphertexts per XOR gate only.

Most 2PC protocols based on garbled circuits require only what is provided by the "garbling schemes" abstraction of [3] which we use here. The fleXOR construction is thus automatically compatible with these protocols. However, some protocols [20, 16] "break the abstraction boundary" of garbling schemes and include optimizations that take advantage of specific properties of free-XOR. In particular, they only require that either the input wires or output wires all share a common offset (sometimes across several garbled circuits); they do not require anything of the internal wires. It is easy to include such a constraint on input/output wires in a fleXOR wire ordering, allowing fleXOR to be compatible with these protocols as well.

7 Performance Comparison

In this section we empirically evaluate the performance of our fleXOR approach against free-XOR and standard (GRR2) garbling. We obtained several circuits of interest [21, 7] and evaluated the performance of our garbling schemes on them. As outlined in the introduction, our primary metric is the size (number of ciphertexts) needed to garble a circuit. The results are summarized in Figure 4.

Eliminating the circularity assumption. As discussed earlier, fleXOR avoids the strong circular-security assumption of free-XOR, when instantiated with a monotone wire ordering. Weakening the assumption does come at a cost, since not all XOR gates are free as a result. Comparing the 2nd and 3rd colums in Figure 4 illustrates the cost savings of circularity. In general, we show that the circularity assumption can be eliminated with a typical increase in garbled circuit size of around 10% (and never more than 20% in our analysis).

We used the heuristic method of Figure 2 for finding good monotone wire orderings (it performed better than the elementary method, on all circuits we tried). The numbers for free-XOR and for fleXOR+monotone both reflect mild

(GRR3) row reduction for the non-XOR gates, except that we apply GRR2-salvaging (Section 5.3) for fleXOR. The gain from GRR2-salvaging varies considerably, but is sometimes noticeable. For example, the numbers in Figure 4 reflect a savings from GRR2-salvaging of 3976 ciphertexts for SHA256, but only 40 for the AES circuit.

Beating (and matching) free-XOR efficiency. As discussed earlier, fleXOR is compatible with aggressive (GRR2) row reduction when it is instantiated with a safe wire ordering. We used the heuristic of Figure 3 to compute good safe orderings for all circuits. The last column of Figure 4 shows the size of the resulting garbled circuits. We point out that the fleXOR-garbled circuit was larger than the free-XOR garbled circuit in only two cases: For the AES circuit (which contained a significantly higher proportion of XOR gates than any other circuit we obtained), the fleXOR garbling was 12% larger than free-XOR; for the fast multiplication circuit, fleXOR was 5% larger. Our best performance was from the DES circuit, whose fleXOR-garbled circuit was 32% smaller than free-XOR.

Again we emphasize that any implementation of fleXOR matches the performance of free-XOR when assigning all wires the same index in the wire ordering. Hence, any implementation of fleXOR would easily be able to be provide whichever of the two wire orderings — safe fleXOR or free-XOR — was preferable, on a per-circuit basis, to realize the column labeled “best” in Figure 4.

(Sub)Optimality. Finally, we emphasize that we did not attempt to find **optimal** orderings for any circuit (which is NP-hard in general), only “good enough” wire orderings found by our simple heuristics. Hence, fleXOR has potential to produce garbled circuits even smaller than the ones reflected in our empirical results here. It is also possible that the circuits themselves could be optimized for fleXOR, similar to how some circuits are currently optimized for free-XOR (i.e., to minimize the number of non-XOR gates).

References

1. B. Applebaum. Garbling XOR gates“for free” in the standard model. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 162–181. Springer, Mar. 2013.
2. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.
3. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, Oct. 2012.
4. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006.
5. S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the “free-XOR” technique. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Mar. 2012.

6. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith. Scalable multi-party computation with nearly optimal work and resilience. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 241–261. Springer, Aug. 2008.
7. W. Henecka and T. Schneider. Memory efficient secure function evaluation. https://code.google.com/p/me-sfe/.
8. M. Hirt, C. Lucas, and U. Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 203–219. Springer, Aug. 2013.
9. M. Hirt and D. Tschudi. Efficient general-adversary multi-party computation. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 181–200. Springer, Dec. 2013.
10. Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, Aug. 2013.
11. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Aug. 2003.
12. V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In B. K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 136–155. Springer, Dec. 2005.
13. V. Kolesnikov and R. Kumaresan. Improved secure two-party computation via information-theoretic garbled circuits. In I. Visconti and R. D. Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 205–221. Springer, Sept. 2012.
14. V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Aug. 2013.
15. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.
16. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Aug. 2013.
17. P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 36–53. Springer, Aug. 2013.
18. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139, New York, NY, USA, 1999. ACM.
19. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Dec. 2009.
20. A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, May 2011.
21. S. Tillich and N. Smart. Circuits of basic functions suitable for MPC and FHE. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.