Efficient Garbling from a Fixed-Key Blockcipher

Mihir Bellare[1] Viet Tung Hoang[2] Sriram Keelveedhi[1] Phillip Rogaway[2]

[1] *Dept. of Computer Science and Engineering, University of California, San Diego, USA.*
Email: {mihir, skeelvee}@eng.ucsd.edu
[2] *Dept. of Computer Science, University of California, Davis, USA.*
Email: {hoangvt, rogaway}@cs.ucdavis.edu

***Abstract*—We advocate schemes based on fixed-key AES as the best route to highly efficient circuit-garbling. We provide such schemes making only one AES call per garbled-gate evaluation. On the theoretical side, we justify the security of these methods in the random-permutation model, where parties have access to a public random permutation. On the practical side, we provide the JustGarble system, which implements our schemes. JustGarble evaluates moderate-sized garbled-circuits at an amortized cost of 23.2 cycles per gate (7.25 nsec), far faster than any prior reported results.**

***Keywords*-Garbled circuits; garbling schemes; multiparty computation; random-permutation model; timing study; Yao's protocol.**

I. Introduction

A garbled circuit (GC) is like an ordinary circuit except that each wire carries a string-valued token instead of the bit it represents. The idea dates to A. Yao, who explained the approach in talks about 2-party SFE (secure function evaluation) in the 1980's [19, 41, 42]. Long the centerpiece of multiparty computation (MPC) protocols, GCs now enjoy diverse applications. Beyond this, some GC-based protocols have become practical. Beginning with Fairplay [32], a bit of a cottage industry has emerged to improve the efficiency and practicality of GC-based MPC [2, 7, 9, 10, 12, 17, 22–24, 28, 29, 31, 36, 37]. Such works target the efficiency of GCs themselves, the way in which GCs are used in higher-level protocols, the software architecture for MPC systems, and the manner in which a user specifies a desired functionality.

This paper shows how to construct and evaluate GCs at unprecedented speeds. Our gains come from two main sources. On the cryptographic side, we describe garbling schemes that evaluate a gate using a single call to a fixed permutation, which can be instantiated by fixed-key AES. On the systems side, we exploit more efficient representations of circuits.

Many approaches are known to propagate tokens across a gate. Yao's original idea was based on a specific public-key encryption scheme; the original exposition describes the use of eight public keys per garbled gate [1, 20]. A more modern description by Naor, Pinkas, and Sumner [35] suggests token propagation using two calls to a pseudorandom function. Lindell and Pinkas [30] proved security for a 2-party protocol in which tokens are propagated using the composition of semantically secure symmetric encryption schemes with an "elusive" and "efficiently verifiable" range. Implementation-oriented work by Lindell, Pinkas, and Smart [31] does token-propagation based on a single call to a cryptographic hash function—the customary choice in later MPC systems.

The advent of AES-NI support (AES new instructions) has made it natural to turn from hash functions to blockciphers for token propagation, and AES256 was the primitive used by Kreuter, Shelat, and Shen [29]. But we contend that the starting point best suited for exploiting AES-NI is not a blockcipher but a *cryptographic permutation*, which can be realized by fixed-key AES: $\mathrm{AES}_{\mathsf{c}}(\cdot)$ with c a fixed, non-secret key. An encryption key can be setup and, after, one has a pipeline into which 128-bit blocks can be fed.

To capitalize on this possibility we seek garbling schemes in the random-permutation model (RPM) [39], meaning that all parties, adversary included, can access a single, fixed, random permutation, as well as its inverse. We aim for high efficiency (a single call to the permutation to evaluate a garbled gate), proven security, and the ability to incorporate existing optimizations, including free xor [28] and garbled row reduction [37].

Our starting point is the recent work of Bellare, Hoang, and Rogaway (BHR) [5]. Traditionally, circuit garbling was seen as an MPC-enabling technique, not an actual primitive. BHR advocates a different point of view, one that sees garbling schemes as a stand-alone cryptographic object. One way to build these objects, BHR explain, is to start from what they term a *dual-key cipher* (DKC). The present work shows that suitable DKCs can be built using a single call to a fixed-key blockcipher. More specifically, we introduce a notion of a *σ-derived* DKC and then prove security of various (reasonably standard) garbling schemes under specified assumptions on the function σ. By instantiating σ in different RPM-based ways one obtains schemes that meet both our efficiency and security aims. Let us explain our main contributions in a bit more detail.

1. Garbling in the RPM. We begin by precisely specifying three garbling schemes: Ga, GaX, and GaXR. The first

1081-6011/13 $26.00 © 2013 IEEE
DOI 10.1109/SP.2013.39

	$\mathbb{E}^{\pi}(A, B, T, X) =$	$k/8$	Ga			GaX			GaXR		
			T_{E}	T_{G}	S_P	T_{E}	T_{G}	S_P	T_{E}	T_{G}	S_P
A1	$\pi(K) \oplus K \oplus X$, with $K \leftarrow A \oplus B \oplus T$	16	50.3	218	64.0	—	—	—	—	—	—
A2	$\pi(K) \oplus K \oplus X$, with $K \leftarrow 2A \oplus 4B \oplus T$	16	52.1	221	64.0	**23.2**	**55.6**	11.5	23.9	56.4	8.64
A3	$\pi(K \parallel T)[1\!:\!k] \oplus K \oplus X$, with $K \leftarrow A \oplus B$	10	93.7	242	40.0	—	—	—	—	—	—
A4	$\pi(K \parallel T)[1\!:\!k] \oplus K \oplus X$, with $K \leftarrow 2A \oplus 4B$	10	97.9	246	40.0	34.2	62.7	7.20	35.0	63.3	**5.40**

Figure 1. **Efficiency of permutation-based garbling**. Data is from the JustGarble system, garbling a moderate-size circuit (a 36.5K gate AES circuit; 82% xor gates). Columns labeled T_{E} and T_{G} give the time to evaluate and garble using the specified protocol, measured in **cycles per gate** (**cpg**). Multiply by 0.3124 to get nanoseconds per gate on our test platform. Columns labeled S_P give the size of the garbled tables, measured in **bytes per gate** (**bpg**). Column $k/8$ is the token length, in **bytes**. This is the length of of A, B, and X. The permutation π is always $\mathrm{AES}_{\mathsf{c}}(\cdot)$. Insecure possibilities are dashed.

is based on the Garble1 scheme of BHR [5], which, in turn, closely follows NPS [35]. The scheme include the *point-and-permute* technique [38], which hijacks one bit of each token so that the agent evaluating the GC knows which "row" of the garbled gate to decrypt. GaX augments Ga with the *free-xor* technique [28], wherein XOR gates can be computed by xoring their incoming token. The savings can be large, as many circuits are rich in XOR gates, or can be refactored so. Finally, GaXR augments GaX with *garbled row reduction* [37], which reduces the size of a GC by arranging that one of the four rows of each garbled gate need not be stored: tokens are selected so as to make this ciphertext a constant.

In each of the three schemes the underlying primitive is a dual-key cipher (DKC). This is a deterministic function $\mathbb{E}\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^k \to \{0,1\}^k$ taking keys A, B, a tweak T, and a plaintext X, returning a ciphertext $\mathbb{E}(A, B, T, X)$. All schemes (Ga, GaX, and GaXR) use at most four calls to $\mathbb{E}$ to garble a gate and at most one call to evaluate a gate. We must efficiently and securely construct the needed DKC.

Our DKC constructions are in the RPM; the DKC has oracle access to a random permutation $\pi\colon \{0,1\}^\ell \to \{0,1\}^\ell$. (An important challenge for security is that the adversary has access not only to π but also to π^{-1}.) This is the sole source of cryptographic hardness available. Our implementations set $\pi = \mathrm{AES}_{\mathsf{c}}(\cdot)$ for a fixed key c. Fig. 1 shows four constructions, with A1/A3 suitable for Ga and A2/A4 suitable for all three schemes. All of our DKC constructions employ a single call to π. We postpone a description of what $2A$ and $4B$ actually mean except to indicate that these are simple operations, a couple of shifts or the like, but *not* integer multiplication.

To validate the security of our schemes instantiated with our DKC constructions, a natural first thought is to prove security of the schemes in the random-oracle (RO) model (ROM) [6] and then show that the constructions of Fig. 1 are indifferentiable from ROs [15, 16, 33]. However, attacks show that the constructions are *not* indifferentiable from ROs. We have preferred them to constructions that are indifferentiable from ROs because the latter are less efficient. The performance gains we have achieved must accordingly be backed up by dedicated proofs.

Rather than provide many *ad hoc* proofs, we provide a unified framework that defines a class of DKCs we call σ-derived. All our instantiations fall in this class. We give conditions on σ sufficient to guarantee the security of Ga, GaX, and GaXR, all in the RPM. Our results use concrete security, giving formulas that bound an adversary's maximal advantage as a function of the effort it expends.

2. VULNERABILITIES IN EXISTING CONSTRUCTIONS. It is common in this area to start from a basic, proven scheme, and then implement an instantiation, enhancement, or variant that is not itself proven. In particular, while there are proofs for some schemes that use the free-xor method [14, 28], ours are the first proofs for schemes that simultaneously use both free xor and garbled row reduction.

Absence of proof can belie presence of error. We consider $\mathbb{E}^H(A,B,T,X) = H(A[1\!:\!k-1] \parallel T) \oplus H(B[1\!:\!k-1] \parallel T) \oplus X$ for a cryptographic hash function H. This DKC was suggested for Fairplay [32], but claimed to work [28] with free xor [28]. We will later show that this not to be the case. Note that other authors have gone so far as to implement MPC using this DKC [37]; the construction has only been considered undesirable because it is less efficient than alternatives, not because its security was in doubt. Our view is that it is not possible to look at a DKC and reliably ascertain if it will work in a complex security protocol; assurance here requires proofs.

3. THE JUSTGARBLE SYSTEM. Prior implementation work has viewed MPC as the goal, with garbling implemented as a component. Our JustGarble system takes a different view, divorcing garbling from MPC to deliver a system whose goal is *just* optimized garbling. This reflects and follows the view of BHR [5]. JustGarble aims to be a general-purpose tool for use not only in MPC, but also beyond.

JustGarble implements Ga, GaX, and GaXR with the DKCs of Fig. 1 and the DKCs' permutation instantiated with fixed-key AES. Among the system-level optimizations and choices in JustGarble, the most prominent is programmatically realizing the mathematical conventions of BHR for representing circuits. The combination of faster DKCs and a simple representation of circuits results in impressive performance gains over previous implementations.

We have carried out a number of timing studies using JustGarble. The main one on which we report is described in Fig. 1. We built an AES128 circuit, a standard test case for this domain, and looked at the time to evaluate the circuit, T_{E}; the time to garble the circuit, T_{G}; and the size of the garbled tables of the circuit, S_P. Breaking with tradition for this domain, we prefer to give running times in cycles per gate (cpg), a measure that's at least a little more robust than time per gate or total time. Similarly, we report on circuit size in units of bytes per gate (bpg).

Fig. 1 highlights the best evaluation time, 23.2 cpg, and the best garbling time, 55.6 cpg. (As our processor runs at 3.201 GHz, this translates to 7.25 nsec/gate for evaluating the GC and 17.4 nsec/gate for garbling it.) The smallest garbled tables are also highlighted, 5.40 bpg. Garbled circuits themselves, which include more than garbled tables, are always 8 bpg larger.

As a point of reference, Huang, Evans, Katz, and Malka (HEKM) evaluate a similar AES circuit in around 2 μsec per gate [23, Section 7: 0.06 sec, online, about 30K gates]. They indicate 10 μsec per gate for very large circuits. Kreuter, Shelat, and Shen (KSS) [29], using a DKC based on AES256 and implemented with AES-NI processor support, report constructing a 31 Kgate AES-128 circuit in 80 msec, so 2.5 μsec per gate. These times are more than two orders of magnitude off of what we report. While such a comparison is in some ways unfair—as we have explained, HKEM and KSS build systems for MPC, not garbling schemes—the time discrepancy is vast, and prior MPC work has routinely maintained that circuit garbling and evaluation *are* key components of the total work done (and have thus been the locus of prior optimizations). We note that the HEKM and KSS figures are times spent on garbling and evaluation alone; they don't include time spent on, say, oblivious transfer or network overhead.

We obtain performance gains over previous implementations even if we drop into JustGarble one of the previously designed, comparatively slow DKCs. The main reason for this is our extremely simple representation of garbled circuit. Gates are not objects that communicate by sending messages, for example; they are indexes into an array. There is no queue of gates ready to be evaluated; gates are topologically ordered, so one just evaluates them in numerical order. We call the representation format we use SCD, for Simple Circuit Description. Its simplicity helps ensure that most of the work in garbling a circuit or evaluating a GC is actual cryptographic work, not overhead related to procedure invocation, message passing, bookkeeping, or the like.

We emphasize that JustGarble knows nothing of MPC, oblivious transfer, compiling programs into circuits, or any of the other tasks associated to making a useful higher-level protocol. JustGarble is a building block. If offers but two services: garble a circuit already built by other means, and evaluate a GC on a garbled input.

II. Preliminaries

We adopt the definitions of BHR [5] lifted to the random-permutation model (RPM).

Notation. We write Σ for $\{0,1\}$. We routinely ignore the distinction between strings and more structured objects encoded by them, implicitly employing simple and fixed encoding schemes. We write $a \twoheadleftarrow A$ to sample a from distribution A. If A is a finite set, it has the uniform distribution.

Circuits. A *circuit*, as defined in BHR [5], is a 6-tuple $f = (n, m, q, A, B, G)$ where $n \geq 2$ is the number of *inputs*, $m \geq 1$ is the number of *outputs*, $q \geq 1$ is the number of *gates*, and $n + q$ be the number of *wires*. We let $\mathrm{Inputs} = [1 \mathinner{..} n]$, $\mathrm{Wires} = [1 \mathinner{..} n+q]$, $\mathrm{OutputWires} = [n+q-m+1 \mathinner{..} n+q]$, and $\mathrm{Gates} = [n+1 \mathinner{..} n+q]$. Then $A\colon \mathrm{Gates} \to \mathrm{Wires}\backslash\mathrm{OutputWires}$ is a function to identify each gate's *first* incoming wire and $B\colon \mathrm{Gates} \to \mathrm{Wires}\backslash\mathrm{OutputWires}$ is a function to identify each gate's *second* incoming wire. Finally $G:\ \mathrm{Gates} \times \{0,1\}^2 \to \{0,1\}$ is a function that determines the *functionality* of each gate. We require $A(g) < B(g) < g$ for all $g \in \mathrm{Gates}$.

The conventions above embody all of the following. Gates have two inputs, arbitrary functionality, and arbitrary fan-out. The wires are numbered 1 to $n+q$. Every non-input wire is the outgoing wire of some gate. The ith bit of input is presented along wire i. The ith bit of output is collected off wire $n+q-m+i$. The outgoing wire of each gate serves as the name of that gate. Output wires may not be input wires and may not be incoming wires to gates. No output wire may be twice used in the output. Requiring $A(g) < B(g) < g$ ensures that the directed graph corresponding to f is acyclic, and that no wire twice feeds a gate; the numbering of gates comprises a topological sort.

Syntax. An (RPM-based) *garbling scheme* is a tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. The first algorithm, Gb, is probabilistic, while all the rest are deterministic. Algorithm Gb has access to an oracle, as does Ev, and we write Gb^{π} and Ev^{π} to denote these algorithms given oracle π. Algorithm Gb^{π} transforms a pair of strings $(1^k, f)$ to a triple of strings (F, e, d). These strings name functions $\mathsf{En}(e, \cdot)\colon \Sigma^n \to \Sigma^*$ and $\mathsf{De}(d, \cdot)\colon \Sigma^* \to \Sigma^m \cup \{\bot\}$ and $\mathsf{Ev}^{\pi}(F, \cdot)\colon \Sigma^* \times \Sigma^* \to \Sigma^*$, where $n = f.n$ and $m = f.m$ are the first and second components of $f = (n, m, q, A, B, G)$. String f itself names a function $\mathsf{ev}(f, \cdot)\colon \Sigma^* \times \Sigma^* \to \Sigma^*$. We call k, f, F, e, d and π the *security parameter*, *initial circuit*, *garbled circuit*, *token list*, *decoding data*, and *random permutation*, respectively.

Throughout this work we will only be concerned with what BHR call projective, circuit-garbling schemes. This assumption was built into some of the names above, as when calling F a "garbled circuit" (instead of a "garbled function"). Function ev will always be the canonical circuit-evaluation function: $\mathsf{ev}(f, x)$ is the m-bit result one gets by

```
proc GARBLE($f_0, f_1, x_0, x_1$)        Game $\mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}$
if $\Phi(f_0) \neq \Phi(f_1)$ then return $\perp$
if $\{x_0, x_1\} \not\subseteq \Sigma^{f_0.n}$ then return $\perp$
if $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$ then return $\perp$
$(F, e, d) \leftarrow \mathsf{Gb}^{\pi}(1^k, f_b)$;   $X \leftarrow \mathsf{En}(e, x_b)$
return $(F, X, d)$
```

Figure 2. **Game for defining the prv security of garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$.** INITIALIZE() samples $b \twoheadleftarrow \{0,1\}$ and FINALIZE(b') returns $(b = b')$.

feeding $x \in \Sigma^n$ to circuit $f = (n, m, q, A, B, G)$. Dealing exclusively with projective schemes, e will always encode a list of strings $(e_1, \ldots, e_{2n})$ and $\mathsf{En}(e, x_1 x_2 \cdots x_n)$ $(x_i \in \Sigma)$ will be $X = (e_{1+x_1}, e_{3+x_2}, \ldots, e_{2n-1+x_n})$.

SIDE INFORMATION. We parameterize privacy by a "knob" that measures what we allow to be revealed. The *side-information function* Φ maps f to some information about it, $\Phi(f)$. Two side-information functions will be of special interest to us. The first, Φ_{topo}, already appeared in BHR. It maps a circuit $f = (n, m, q, A, B, G)$ to its underlying topological circuit $\Phi_{\mathrm{topo}}(f) = (n, m, q, A, B)$. The second, Φ_{xor}, is new. It maps a circuit $f = (n, m, q, A, B, G)$ to something that obscures the functionality of each non-XOR gate. Formally, function Φ_{xor} maps $f = (n, m, q, A, B, G)$ to the circuit $\Phi_{\mathrm{xor}}(f) = (n, m, q, A, B, G')$ where $G'_g =$ XOR if $G_g =$ XOR and, arbitrarily, $G'_g =$ AND otherwise.

SECURITY. Given a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, security parameter $k \in \mathbb{N}$, side-information function Φ, and length-preserving permutation $\pi \colon \Sigma^* \to \Sigma^*$, Fig. 2 specifies the game $\mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}$ used to define privacy. For an adversary $\mathcal{A}$, let

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.rpm},\Phi}(\mathcal{A}, k) = 2 \Pr\left[\pi \twoheadleftarrow \mathrm{Perm} \colon\ \mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}^{\mathcal{A}}\right] - 1$$

be the probability, normalized to $[0,1]$, that the FINALIZE procedure of game $\mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}$ returns 1 (i.e., the adversary correctly predicts b) when adversary $\mathcal{A}$, running with oracles π, π^{-1} and provided an input of 1^k, interacts with the specified game, making a single call to GARBLE. Here Perm is the set of all length-preserving permutations and a random sample π from Perm, restricted to strings of length $\ell \in \mathbb{N}$, is a uniformly random permutation on Σ^{ℓ}.

Informally, we will say that $\mathcal{G}$ is prv secure over Φ, in the RPM, if $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.rpm},\,\Phi}(\mathcal{A}, k)$ is "small" for any "reasonable" $\mathcal{A}$ and k. Insofar as we are working in the RPM, we will not need cryptographic assumptions in order to specify just how small $\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.rpm},\,\Phi}(\mathcal{A}, k)$ is: it will be a function of k and the total number of queries, q, it makes to its π and π^{-1} oracles.

We comment that our security definition allows Gb and Ev to depend on π but not its inverse. This choice was made simply because we have no occasion, in schemes, to use π^{-1}. On the other hand, it is essential that the *adversary* has access to both π and π^{-1}.

IND VERSUS SIM. The definition above formalizes security using the indistinguishability (ind) style definition of BHR [5]. BHR also give a simulation-style definition and show the two equivalent for side-information functions having a property they called *efficient invertibility*. We revisit this equivalence in the RPM. The complicating issue is that in an idealized model like the RPM there are two possibilities for the simulator, namely to program or not program the ideal primitive, here the random permutation. Somewhat curiously, the proofs of BHR [5] show that, for efficiently-invertible side-information functions, both are equivalent to ind and thus to each other. The side-information function Φ_{topo} was shown in BHR to be efficiently invertible. We show in Appendix A that Φ_{xor} is as well. As a consequence, our ind-based definition is equivalent to the sim-based definition in the strong model where the simulator does not program the random permutation.

ANALOGS. One can easily give a random-oracle model (ROM) definition to complement RPM one. Let Func be the set of all functions $\pi \colon \Sigma^* \times \mathbb{N} \to \Sigma^*$ with $|\pi(x, \ell)| = \ell$. Give this a distribution by asserting that a random $\pi \twoheadleftarrow$ Func (x, ℓ) to ℓ uniformly random bits. Then let

$$\mathbf{Adv}_{\mathcal{G}}^{\mathrm{prv.rom},\Phi}(\mathcal{A}, k) = 2 \Pr\left[\pi \twoheadleftarrow \mathrm{Func} \colon\ \mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}^{\mathcal{A}}\right] - 1$$

be the probability, normalized to $[0,1]$, that the FINALIZE procedure of game $\mathrm{Prv}_{\mathcal{G},\Phi,k,\pi}$ returns 1 when adversary $\mathcal{A}$, running with oracle π and given input 1^k, interacts with the specified game, making a single call to GARBLE. The only difference between the ROM and RPM definitions is that in the RPM setting, the adversary gets the random permutation and its inverse, while in the ROM setting, it's a random function and there's no inverse to give.

One can analogously give other idealized-model definitions, the most important being the ideal-cipher model (ICM). And one can of course give a standard-model definition, simply by dropping all mention of π and its inverse.

DUAL-KEY CIPHERS. Again following BHR, we will describe our garbling schemes in terms of a *dual-key cipher* (DKC). Now, however, these objects will be provided oracles. Letting Ω be a set of functions π from Σ^* to Σ^*, an (oracle-) DKC is a function $\mathbb{E} \colon \Omega \times \Sigma^k \times \Sigma^k \times \Sigma^{\tau} \times \Sigma^k \to \Sigma^k$ that associates to $\pi \in \Omega$ and $A, B \in \Sigma^k$ and $T \in \Sigma^{\tau}$ some permutation $\mathbb{E}^{\pi}(A, B, T, \cdot) \colon \Sigma^k \to \Sigma^k$.

In this paper we won't need to develop any DKC security notions: we shall be using the syntax of DKCs only to make the descriptions of our protocols more clear.

GARBLING SCHEMES Ga, GaX, GaXR. The scheme we call Ga is based on an oracle DKC $\mathbb{E}^{\pi} \colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\tau} \times \{0,1\}^k \to \{0,1\}^k$ whose inverse is denoted $\mathbb{D}$. We associate to $\mathbb{E}$ the RPM-model garbling scheme $\mathrm{Ga}[\mathbb{E}]$ of Fig. 3. Wires carry k-bit tokens (strings) the last bit of each is the token's *type*.

```
proc $\mathsf{Gb}^\pi(1^k, f)$                                   Ga
$(n, m, q, A', B', G) \leftarrow f$

for $i \leftarrow 1$ to $n + q$ do
   $t \twoheadleftarrow \{0,1\}$
   $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}t,\ \ X_i^1 \twoheadleftarrow \{0,1\}^{k-1}\bar{t}$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A'(g),\ \ b \leftarrow B'(g)$
   for $i \leftarrow 0$ to $1,\ \ j \leftarrow 0$ to $1$ do
      $A \leftarrow X_a^i,\ \ \mathtt{a} \leftarrow \mathrm{lsb}(A)$
      $B \leftarrow X_b^j,\ \ \mathtt{b} \leftarrow \mathrm{lsb}(B)$
      $P[g, \mathtt{a}, \mathtt{b}] \leftarrow \mathbb{E}^\pi(A, B, g, X_g^{G_g(i,j)})$

$F \leftarrow (n, m, q, A', B', P)$
$e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
$d \leftarrow \big(\mathrm{lsb}(X_{n+q-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0)\big)$
return $(F, e, d)$
```

```
proc $\mathsf{Gb}^\pi(1^k, f)$                                   GaX
$(n, m, q, A', B', G) \leftarrow f$

$R \twoheadleftarrow \{0,1\}^{k-1}1$
for $i \leftarrow 1$ to $n$ do
   $t \twoheadleftarrow \{0,1\}$
   $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}t,\ \ X_i^1 \leftarrow X_i^0 \oplus R$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A'(g),\ \ b \leftarrow B'(g),\ \ G'_g \leftarrow \mathrm{XOR}$
   if $G_g = \mathrm{XOR}$ then
      $X_g^0 \leftarrow X_a^0 \oplus X_b^0,\ \ X_g^1 \leftarrow X_g^0 \oplus R$
   else
      $G'_g \leftarrow \mathrm{AND}$
      $X_g^0 \twoheadleftarrow \{0,1\}^k,\ \ X_g^1 \leftarrow X_g^0 \oplus R$
      for $i \leftarrow 0$ to $1,\ \ j \leftarrow 0$ to $1$ do
         $A \leftarrow X_a^i,\ \ \mathtt{a} \leftarrow \mathrm{lsb}(A)$
         $B \leftarrow X_b^j,\ \ \mathtt{b} \leftarrow \mathrm{lsb}(B)$
         $P[g, \mathtt{a}, \mathtt{b}] \leftarrow \mathbb{E}^\pi(A, B, g, X_g^{G_g(i,j)})$

$F \leftarrow (n, m, q, A', B', G', P)$
$e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
$d \leftarrow \big(\mathrm{lsb}(X_{n+q-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0)\big)$
return $(F, e, d)$
```

```
proc $\mathsf{Gb}^\pi(1^k, f)$                                   GaXR
$(n, m, q, A', B', G) \leftarrow f$

$R \twoheadleftarrow \{0,1\}^{k-1}1$
for $i \leftarrow 1$ to $n$ do
   $t \twoheadleftarrow \{0,1\}$
   $X_i^0 \twoheadleftarrow \{0,1\}^{k-1}t,\ \ X_i^1 \leftarrow X_i^0 \oplus R$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A'(g),\ \ b \leftarrow B'(g),\ \ G'_g \leftarrow \mathrm{XOR}$
   if $G_g = \mathrm{XOR}$ then
      $X_g^0 \leftarrow X_a^0 \oplus X_b^0,\ \ X_g^1 \leftarrow X_g^0 \oplus R$
   else
      for $\mathtt{a} \leftarrow 0$ to $1,\ \ \mathtt{b} \leftarrow 0$ to $1$ do
         $i \leftarrow \mathtt{a} \oplus \mathrm{lsb}(X_a^0),\ \ A \leftarrow X_a^i$
         $j \leftarrow \mathtt{b} \oplus \mathrm{lsb}(X_b^0),\ \ B \leftarrow X_b^j$
         $r \leftarrow G_g(i,j),\ \ G'_g \leftarrow \mathrm{AND}$
         if $\mathtt{a} = 0$ and $\mathtt{b} = 0$ then
          $X_g^r \leftarrow \mathbb{E}^\pi(A, B, T, 0^k)$
          $X_g^{\overline{r}} \leftarrow X_g^r \oplus R$
         else $P[g,\mathtt{a},\mathtt{b}] \leftarrow \mathbb{E}^\pi(A,B,g,X_g^{G_g(i,j)})$

$F \leftarrow (n, m, q, A', B', G', P)$
$e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
$d \leftarrow \big(\mathrm{lsb}(X_{n+q-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0)\big)$
return $(F, e, d)$
```

```
proc $\mathsf{Ev}^\pi(F, X)$                                     Ga
$(n, m, q, A, B, P) \leftarrow F$
$(X_1, \ldots, X_n) \leftarrow X$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A(g),\ \ b \leftarrow B(g)$
   $\mathtt{a} \leftarrow \mathrm{lsb}(X_a), \mathtt{b} \leftarrow \mathrm{lsb}(X_b)$
   $X_g \leftarrow \mathbb{D}^\pi(X_a, X_b, g, P[g, \mathtt{a}, \mathtt{b}])$

return $(X_{n+q-m+1}, \ldots, X_{n+q})$
```

```
proc $\mathsf{Ev}^\pi(F, X)$                                     GaX
$(n, m, q, A', B', P) \leftarrow F$
$(X_1, \ldots, X_n) \leftarrow X$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A(g),\ \ b \leftarrow B(g)$
   $\mathtt{a} \leftarrow \mathrm{lsb}(X_a),\ \ \mathtt{b} \leftarrow \mathrm{lsb}(X_b)$
   if $G'_g = \mathrm{XOR}$ then $X_g \leftarrow X_a \oplus X_b$
   else $X_g \leftarrow \mathbb{D}^\pi(X_a, X_b, g, P[g, \mathtt{a}, \mathtt{b}])$

return $(X_{n+q-m+1}, \ldots, X_{n+q})$
```

```
proc $\mathsf{Ev}^\pi(F, X)$                                     GaXR
$(n, m, q, A, B, G', P) \leftarrow F$
$(X_1, \ldots, X_n) \leftarrow X$

for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A(g),\ \ b \leftarrow B(g)$
   $\mathtt{a} \leftarrow \mathrm{lsb}(X_a),\ \ \mathtt{b} \leftarrow \mathrm{lsb}(X_b)$
   if $G'_g = \mathrm{XOR}$ then $X_g \leftarrow X_a \oplus X_b$
   elsif $\mathtt{a} = 0$ and $\mathtt{b} = 0$ then
      $X_g \leftarrow \mathbb{E}^\pi(X_a, X_b, g, 0^k)$
   else $X_g \leftarrow \mathbb{D}^\pi(X_a, X_b, g, P[g, \mathtt{a}, \mathtt{b}])$
return $(X_{n+q-m+1}, \ldots, X_{n+q})$
```

```
proc $\mathsf{En}(e, x)$                     Ga, GaX, GaXR
$(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
$x_1 \cdots x_n \leftarrow x$
$X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$
return $X$
```

```
proc $\mathsf{De}(d, Y)$                     Ga, GaX, GaXR
$(d_1, \ldots, d_m) \leftarrow d$
$(Y_1, \ldots, Y_m) \leftarrow Y$
for $i \leftarrow 1$ to $m$ do $y_i \leftarrow \mathrm{lsb}(Y_i) \oplus d_i$
return $y \leftarrow y_1 \cdots y_m$
```

```
proc $\mathsf{ev}(f, x)$                     Ga, GaX, GaXR
$(n, m, q, A, B, G) \leftarrow f$
$x_1 \cdots x_n \leftarrow x$
for $g \leftarrow n+1$ to $n + q$ do
   $a \leftarrow A(g),\ \ b \leftarrow B(g)$
   $x_g \leftarrow G_g(x_a, x_b)$
return $x_{n+q-m+1} \cdots x_{n+q}$
```

Figure 3. **Garbling schemes of this paper.** Schemes Ga, GaX, and GaXR have the same En, De, and ev procedures, but their own Gb and Ev procedures For a bit t, let $\{0,1\}^{k-1}t$ denote the set of k-bit strings whose last bit is t, and $\bar{t}$ the complement bit of t.

To garble a circuit, we begin selecting two tokens for each wire, one of each type. One of these will represent 0—the token is said to have *semantics* of 0—while the other will represent 1. The variable X_i^b names the token of wire i with semantics of b. Thus the token list e will map $x = x_1 \cdots x_n \in \{0,1\}^n$ to $X = (X_1^{x_1}, \ldots, X_n^{x_n})$. For each wire i we select random tokens of opposite type, making the association between a token's type and its semantics random. We then compute q garbled tables, one for each gate g. Table $P[g, \cdot, \cdot]$ has four rows, row $\mathtt{a}, \mathtt{b}$ used when the left incoming token is of type $\mathtt{a}$ and the right incoming token is of type $\mathtt{b}$. The token that gets encrypted for this row is the one for the outgoing-wire with the correct semantics. Given incoming tokens X_a and X_b we use their types to determine which row of the garbled table to decrypt. The description of the decoding data d is a bit vector; the ith component is the last bit of the token of semantics 0 on the ith output wire. Garbling scheme GaX augments what we have described with the free-xor technique. Scheme GaXR additionally incorporates the row-reduction technique.

III. Instantiation Overview

We discuss some of the challenges, and choices we make in response, with regard to garbling in the RPM.

The DKC $\mathbb{E}^H(A,B,T,X) = H(A\|B\|T) \oplus X$ is a natural starting point, where H is a hash function. Our constructions can be seen as realizations of this approach, but based on a fixed-key blockcipher. Kreuter, Shelat, and Shen [29] had already considered $H(A\|B\|T) = \mathrm{AES256}_{A\|B}(T)$ where $|A| = |B| = |T| = 128$. Fixed-key AES provides a primitive π with only a third the number of input bits as AES256.

One possibility is to build H from π in a manner that will render H indifferentiable from a RO [16, 33]. However, known constructions with this property will not be as efficient as we would like. We aim to use a Davies-Meyer type construction [34, 40], which applies the permutation only once. Such constructions are *not* indifferentiable from ROs [15, 16], necessitating considerable caution.

For simplicity we start by ignoring the tweak and considering the garbling of one-gate circuits. We present several natural constructions and show that they fail. We then present our constructions, and finally explain how to incorporate tweaks so as to handle circuits with an arbitrary number of gates.

Instantiating Ga. Consider instantiating the DKC of Ga from a permutation π by $\mathbb{E}^\pi(A,B,T,X) = \pi(A \oplus B) \oplus X$. The resulting scheme can be trivially broken, as follows. Suppose that we garble an AND gate, as illustrated on the top-left corner of Fig. 4, and suppose the adversary is given the garbled table and tokens A and C. First, it opens the third row to obtain token X. Next, let V be the ciphertext in the last row. Then the adversary can obtain token $D = \pi^{-1}(V \oplus X) \oplus A$. Likewise, it can obtain token B. Now the adversary can open every row of the garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary asks $(f_0, f_1, 00, 00)$ to Garble where f_0 is an AND gate and f_1 is a gate that always outputs 0. Following the idea above, the adversary open every row of the garbled table. If each row encrypts the same token then it outputs 1; otherwise, it outputs 0.

The attack arises because the adversary can invert $\pi(A \oplus D)$ to get D. To break this invertibility we employ the Davies-Meyer construction $\rho(K) = \pi(K) \oplus K$ to obtain the instantiation

$$\mathbb{E}^\pi(A,B,T,X) = \rho(A \oplus B) \oplus X \ . \qquad (1)$$

We shall see in Theorem 1 that instantiation (1) indeed makes Ga secure, once the tweaks are appropriately introduced.

Instantiating GaX. Yet instantiation (1) doesn't work for scheme GaX, even if the circuit remains a single gate. Here is an attack. Again we garble an AND gate. The illustration is given at the bottom-left corner of Fig. 4. Suppose the adversary is given the garbled table and tokens A and B. It first xors the ciphertexts in the second and third rows and obtains the string R. It then can open every row of the garbled table. Now all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary queries $(f_0, f_1, 00, 01)$ where f_0 is an AND gate and f_1 is a gate such that $f_1(a,b) = a$ for all $a, b \in \{0,1\}$. Following the idea above, the adversary can open every row of the garbled table, regardless of the challenge bit. If there are three rows that encrypt the same token then it outputs 0; otherwise, it outputs 1.

The attack above arises because of a "symmetry" between tokens of the first and second incoming wires, leading to the use of $\rho(A \oplus B)$ twice to mask tokens of the output wire. One possible way to break this symmetry is to apply some simple operation to the token of the second incoming wire before using it. For example, consider the instantiation

$$\mathbb{E}^\pi(A,B,T,X) = \rho(A \oplus 2B) \oplus X, \qquad (2)$$

where doubling ($B \mapsto 2B$) is multiplying in $\mathrm{GF}(2^k)$ by the group element $\mathsf{x} = 0^{k-2}10$. The attack above is thwarted, because the ciphertext in the third row is $\rho(A \oplus 2B) \oplus X$ while that in the second row is now $\rho(A \oplus 2B \oplus 3R) \oplus X \oplus R$, where $3R$ means multiplying R by the group element $\mathsf{x}+1 = 0^{k-2}11$ in $\mathrm{GF}(2^k)$.

Still, instantiation (2) can be broken as follows. See the illustration on the top-right corner of Fig. 4. Garble an OR gate. Suppose the adversary is given the garbled table and tokens A and B. First it opens the third row to obtain token X. Let V be the ciphertext in the first row. Query $V \oplus A \oplus 2B \oplus X$ to π^{-1}, and let K be the answer. Then, the adversary can obtain $R = K \oplus A \oplus 2B$. It can now open every row of the garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security. The adversary queries $(f_0, f_1, 00, 01)$ where f_0 is an OR gate and f_1 is an AND gate. Following the idea above, the adversary can open every row of the garbled table, regardless of the challenge bit. Using the decoding data, the adversary can determine the semantics of the tokens on the output wire. If there are three rows that encrypt the token of semantics 1 then it outputs 1; otherwise, it outputs 0.

To thwart the attack above one can apply the multiplication in $\mathrm{GF}(2^k)$ to the first incoming token as well; for example, we can use the instantiation

$$\mathbb{E}^\pi(A,B,T,X) = \rho(2A \oplus 4B) \oplus X \qquad (3)$$

where $4B$ means applying the doubling operation to B twice, that is, multiplying B by the group element $\mathsf{x}^2 = 0^{k-3}100$ in $\mathrm{GF}(2^k)$. The ciphertext in the first row will be $\pi(2A \oplus 4B \oplus 2R) \oplus 2A \oplus 4B \oplus X \oplus 3R$. Since $R \twoheadleftarrow \{0,1\}^{k-1}1$ is secret, the attack fails. We shall see in Theorems 1 and 2 that instantiation (3) indeed makes both Ga and GaX secure, after the gate-number tweak is appropriately introduced.

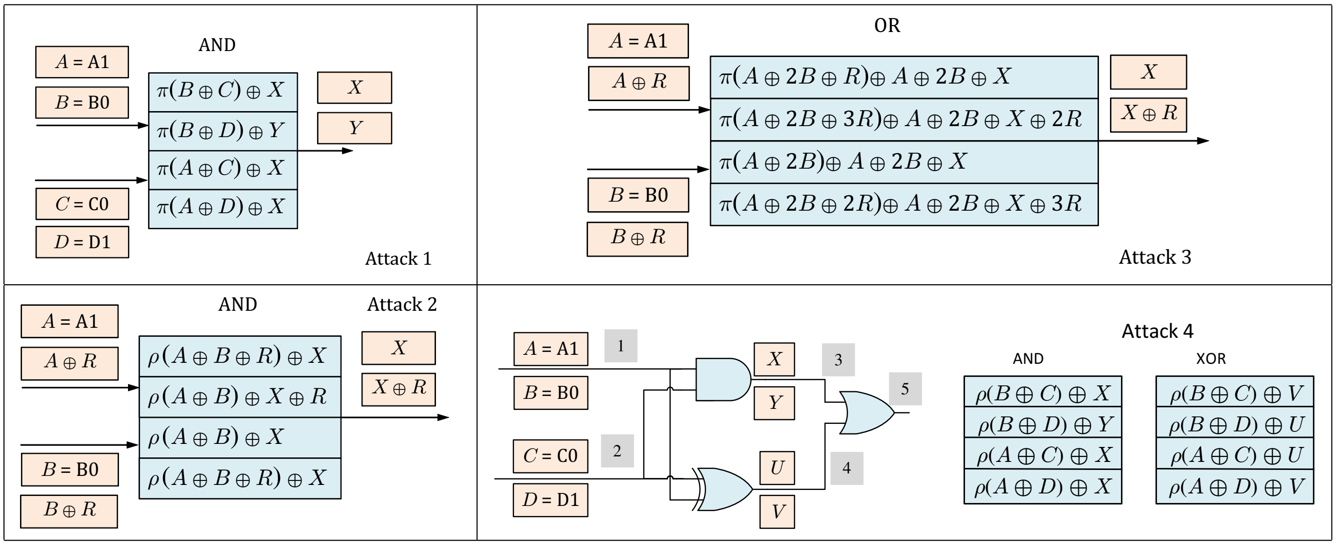
AND
A = A1
B = B0
$\pi(B \oplus C) \oplus X$
$\pi(B \oplus D) \oplus Y$
$\pi(A \oplus C) \oplus X$
$\pi(A \oplus D) \oplus X$
X
Y
C = C0
D = D1
Attack 1
OR
A = A1
$A \oplus R$
$\pi(A \oplus 2B \oplus R) \oplus A \oplus 2B \oplus X$
$\pi(A \oplus 2B \oplus 3R) \oplus A \oplus 2B \oplus X \oplus 2R$
$\pi(A \oplus 2B) \oplus A \oplus 2B \oplus X$
$\pi(A \oplus 2B \oplus 2R) \oplus A \oplus 2B \oplus X \oplus 3R$
X
$X \oplus R$
B = B0
$B \oplus R$
Attack 3
AND
Attack 2
A = A1
$A \oplus R$
$\rho(A \oplus B \oplus R) \oplus X$
$\rho(A \oplus B) \oplus X \oplus R$
$\rho(A \oplus B) \oplus X$
$\rho(A \oplus B \oplus R) \oplus X$
X
$X \oplus R$
B = B0
$B \oplus R$
A = A1
B = B0
C = C0
D = D1
1
2
3
4
5
X
Y
U
V
Attack 4
AND
XOR
$\rho(B \oplus C) \oplus X$
$\rho(B \oplus D) \oplus Y$
$\rho(A \oplus C) \oplus X$
$\rho(A \oplus D) \oplus X$
$\rho(B \oplus C) \oplus V$
$\rho(B \oplus D) \oplus U$
$\rho(A \oplus C) \oplus U$
$\rho(A \oplus D) \oplus V$

Figure 4. **Attacks on DKC instantiations.** Top-left: $\mathbb{E}^\pi(A, B, T, X) = \pi(A \oplus B) \oplus X$ for scheme Ga. Bottom-left: $\mathbb{E}^\pi(A, B, T, X) = \rho(A \oplus B) \oplus X$ for scheme GaX, with $\rho(K) = \pi(K) \oplus K$. Top-right: $\mathbb{E}^\pi(A, B, T, X) = \rho(A \oplus 2B) \oplus X$ for scheme GaX. Bottom-right: $\mathbb{E}^\pi(A, B, T, X) = \rho(A \oplus B) \oplus X$ for scheme Ga. The doubling here is multiplying in $\mathrm{GF}(2^k)$ by $\mathtt{x} = 0^{k-2}10$. In each wire, the top and bottom tokens have semantics 0 and 1 respectively.

THE NEED FOR THE TWEAK. Suppose now that one uses instantiation (1) for scheme Ga, but in a circuit of multiple gates. This leads to a new attack. Garble the circuit f illustrated at the bottom-right of Fig. 4. Suppose the adversary is given the garbled tables and tokens A and D. (In the illustration, only the garbled tables of the first two gates are shown.) It first opens the last rows in the first two tables to get tokens X and V. Next, it xors the ciphertexts in the third rows of the two first tables, and then xors the resulting string with X to get U. Likewise, the adversary can obtain Y. It now can open every row of the last garbled table, and all security is lost.

We can translate the idea to an attack of advantage 1 on prv security, in which the adversary queries $(f, f, 01, 11)$ to obtain (F, X, d). Following the idea above, regardless of the challenge bit, the adversary can open every row of the last garbled table. Using d, the adversary can determine the semantics of the tokens on the output wire. There is only one row of the last garbled table that encrypts the token of semantics 0. The token on wire 3 used as a key for this row must have semantics 0. The adversary then can determine the semantics of tokens on wire 3. Now evaluate F on X. If the token obtained on wire 3 during the evaluation has semantics 0 then output 0. Otherwise, output 1.

The attack above arises if the circuit contains two gates that have the same pair of incoming wires. We therefore introduce the tweak-based variants $\mathbb{E}^\pi(A, B, T, X) = \rho(A \oplus B \oplus T) \oplus X$ and $\mathbb{E}^\pi(A, B, T, X) = \rho(2A \oplus 4B \oplus T) \oplus X$ of instantiations (1) and (3), respectively, with the tweak being the gate index. We shall see in Theorems 1 and 2 that these tweak-based instantiations indeed make Ga secure, and the second one makes GaX secure.

Alternatively, for scheme Ga, one can avoid using tweaks by demanding that no two gates have the same pair of incoming wires. However, this condition is not sufficient when the free-xor trick is used, because one can arrange for distinct wires to carry the *same* pair of tokens. For example, consider the circuit in Fig. 5. Wires 6 and 7 there have the same pair of tokens. This kind of subtle degeneracy serves to emphasize the need for proofs.

OTHER WAYS TO DOUBLE. Besides the multiplication in $\mathrm{GF}(2^k)$ (named D1 below) doubling may have several other interpretations, setting $2A$ to any of the following:

D1:	$(A \ll 1) \oplus (A[1] \cdot \mathrm{const})$	*Finite field multiply*
D2:	$A \ll 1$	*Logical left shift*
D3:	$A \gg 1$	*Logical right shift*
D4:	$A \lll 1$	*Circular left shift*
D5:	$A \ggg 1$	*Circular right shift*
D6:	$(A[1 : \lfloor k/2 \rfloor] \ll 1) \parallel (A[\lfloor k/2 \rfloor + 1 : k] \ll 1)$	*SIMD left*
D7:	$(A[1 : \lfloor k/2 \rfloor] \gg 1) \parallel (A[\lfloor k/2 \rfloor + 1 : k] \gg 1)$	*SIMD right*

We will later show that all of these methods "work" for the schemes in this paper, although the security bounds differ by a constant. In particular, we will identify a sufficient condition for the doubling map and a real number r associated to it, this number showing up in our bounds. The reason for attending to these different doubling methods is that "true" doubling has the best security bound, but its implementation is a bit slower than alternatives with slightly inferior bounds.

AN INSECURITY ISSUE IN PRIOR WORKS. Besides proposing the free-xor trick, Kolesnikov and Schneider (KS) [28] propose two instantiations of a DKC, suggesting to set

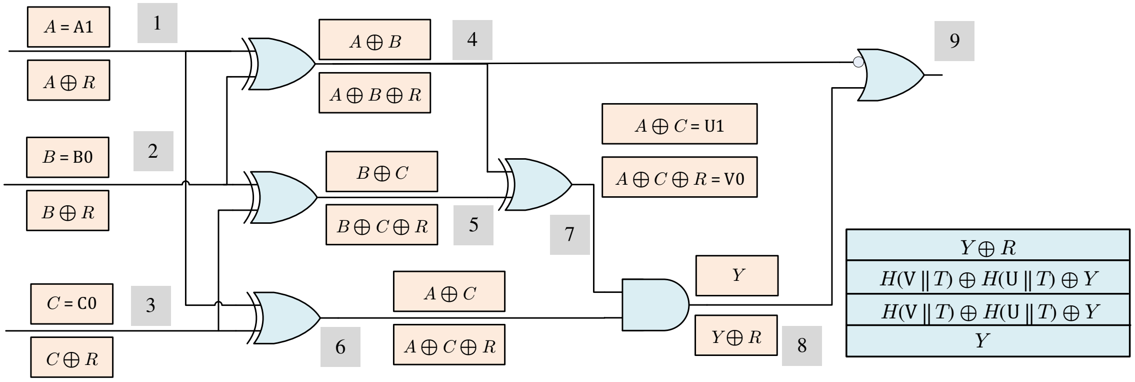

Figure 5. **An attack on GaX with DKC $\mathbb{E}(A, B, T, X) = H(A[1\colon k-1] \parallel T) \oplus H(B[1\colon k-1] \parallel T) \oplus X$.** In each wire, the top token has semantics 0, the bottom one has semantics 1. The table on the right is the garbled table of gate 8. Gate 9 negates the bit on wire 4, then ORs it with the bit on wire 8.

$\mathbb{E}^T_{A,B}(X)$ as either

$$H(A[1\!:\!k{-}1] \parallel B[1\!:\!k{-}1] \parallel T) \oplus X \quad \text{or} \qquad (4)$$

$$H(A[1\!:\!k{-}1] \parallel T) \oplus H(B[1\colon k-1] \parallel T) \oplus X \qquad (5)$$

where $H\colon \{0,1\}^* \to \{0,1\}^k$ is a hash function, to be modeled as a random oracle. KS effectively show that GaX, built on top of instantiation (4), leads to a secure two-party SFE protocol. They claim that one can use instantiation (5) as well. Pinkas, Schneider, Smart, and Williams (PSSW) [37] implement both instantiations; their garbling schemes are variants of Ga/GaX/GaXR, where each DKC's tweak is a nonce instead of the gate index. Subsequent works [22, 23, 27] use only (4) because of efficiency issues, but the authors apparently continue to believe that (5) works fine; see, for example, [13, p. 5] and [27, p. 7].

We now show that an adversary can completely break GaX if the DKC is instantiated by (5). Our attack also applies to the GaX/GaXR variants of PSSW based on (5). The key idea of the attack is that, as mentioned previously, when one uses free-xor trick, different wires in the circuit can be forced to share the same pair of tokens. Observe that if $A = B$ then instantiation (5) sends the plaintext in the clear, as $H(A[1\!:\!k{-}1] \parallel T) \oplus H(B[1\colon k-1] \parallel T) \oplus X = X$. Suppose that we garble the circuit f in Fig. 5. Wires 6 and 7 have the same pair of tokens. As shown in the garbled table of gate 8, we send both Y and $Y \oplus R$ in the clear, and there is no security whatsoever.

To translate the above to an attack of advantage 1 on prv security, the adversary queries $(f, f, 000, 100)$ to obtain (F, X, d). Following the idea above, the adversary obtains all tokens and opens every row of every garbled table. Using d, it can determine the semantics of the tokens on the output wire. There is only one row of the garbled table of gate 9 that encrypts the token of semantics 0. The token on wire 4 used as a key for this row must have semantics 1. The adversary therefore can determine the semantics of the tokens on wire 4. Now evaluate F on X. If the token obtained on wire 4 has semantics 0 then output 1, otherwise output 0.

IV. Security of Ga, GaX and GaXR

We will justify the security of our schemes in a common framework. We define a class of DKCs that we call σ-derived. Under various conditions on the map σ, we prove security for our schemes.

σ-derived DKCs. Let $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$ be a function. We say that $\mathbb{E}$ is σ-derived DKC if $\mathbb{E}^\pi(A, B, T, X) = (\pi(K) \oplus K)[1\colon k] \oplus X$ for $K = \sigma(A, B, T)$ and the function σ satisfies the following two conditions:

(i) $\sigma(A \oplus A^*, B \oplus B^*, T \oplus T^*) = \sigma(A, B, T) \oplus \sigma(A^*, B^*, T^*)$ for every $A, A^*, B, B^* \in \{0,1\}^k$ and $T, T^* \in \{0,1\}^\tau$, and

(ii) $\sigma(0^k, 0^k, T) \neq 0^\ell$ unless $T = 0^\tau$.

The *injectivity indicator* of σ is a number $\delta \in \{0,1\}$; it is 0 if and only if σ is tweak-wise injective, that is, $\sigma(A, B, T) \neq \sigma(A^*, B^*, T^*)$ whenever $T \neq T^*$. The *regularity* of σ is the smallest $r \in \mathbb{Z}^+$ such that

(iii) $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(x, 0^k, 0^\tau) = s] \leq r/2^k$ and also $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(0^k, x, 0^\tau) = s] \leq r/2^k$ for every string $s \in \{0,1\}^\ell$.

The *strong regularity* of σ is the smallest $r \in \mathbb{Z}^+$ such that (iii) is satisfied and

(iv) $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(a{\cdot}x, b{\cdot}x, 0^\tau) \oplus x0^{\ell-k} = s] \leq r/2^k$ and $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ \sigma(x, x, 0^\tau) = s] \leq r/2^k$ for every string $s \in \{0,1\}^\ell$ and every $(a, b) \in \{0,1\}^2$, where $0 \cdot x = 0^{|x|}$ and $1 \cdot x = x$.

Each of our DKC instantiations is a σ-derived DKC; the regularity, strong regularity, and injectivity indicator of its σ are shown in Fig. 6. This claim can be verified by a simple but tedious analysis. For example, consider scheme A2 with the doubling method D2. Its function σ is $\sigma(A, B, T) = 2A \oplus 4B \oplus T$, satisfying both (i) and (ii), and the injectivity indicator of this σ is 1. The regularity is 4, because $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ x \ll 1 = s] \leq 2/2^k$ and $\Pr[x \twoheadleftarrow \{0,1\}^k\colon\ x \ll 2 = s] \leq 4/2^k$ for every

DKC	A1	A2				A3	A4			
doubling	—	D1	D2, D3	D4, D5	D6, D7	—	D1	D2, D3	D4, D5	D6, D7
regularity	1	1	4	1	16	1	1	4	1	16
strong regularity	—	1	4	4	16	—	1	4	4	16
injectivity indicator	1	1				0	0			

Figure 6. **Parameters for DKC instantiations.** The strong regularity of A1 and A3 is huge ($\delta = 2^k$); the corresponding entries are dashed.

string $s \in \{0,1\}^k$. To verify that the strong regularity is also 4, suppose that one wants to show that, say $\Pr[x \twoheadleftarrow \{0,1\}^k \colon\ (x \lll 1) \oplus x = s] \le 4/2^k$ for every string $s \in \{0,1\}^k$. Let $x = x_1 \cdots x_k$. Note that function $f(x) = (x \lll 1) \oplus x$ returns

$$(x_1 \oplus x_2) \parallel (x_2 \oplus x_3) \parallel \cdots \parallel (x_{k-1} \oplus x_k) \parallel x_k,$$

and thus it is a permutation on $\{0,1\}^k$. Since $x \twoheadleftarrow \{0,1\}^k$, it follows that $f(x)$ is also uniformly distributed over $\{0,1\}^k$. Hence the chance that $f(x) = s$ is at most $1/2^k$.

SECURITY OF GA. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small regularity, then Ga[$\mathbb{E}$] is prv-secure over Φ_{topo}.

Theorem 1: Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma\colon \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the regularity and injectivity indicators of σ, respectively. Then $\mathbf{Adv}^{\text{prv.rpm},\,\Phi_{\text{topo}}}_{\text{Ga}[\mathbb{E}]}(\mathcal{A}, k) \le (6Qq + 15q^2)/2^\ell + (30rQ + 84rq)/2^k + \delta(42rQq + 69rq^2)/2^k$.

In the advantage formula above, we use the injectivity indicator δ to "safeguard" the term $(Qq + q^2)/2^k$. For the DKC instantiation A3, our implementation uses $k = 80$, and in practice, q may go up to 2^{32}, say, as in recent works [23, 29]. The presence of the term $(Qq + q^2)/2^k$ for A3 would result in a poor bound. Fortunately, this term vanishes, because $\delta = 0$ for A3. The advantage for A3 is about $(Qq + q^2)/2^\ell + (Q + q)/2^k$, which is satisfactory for $\ell = 128$ and $k = 80$. In the DKC instantiation A1, for example, $\delta = 1$, but there we'll use $k = \ell = 128$, and the advantage becomes about $(Qq + q^2)/2^\ell$, which is very good.

To obtain the desirable bound above, the proof for Theorem 1, given in Appendix B, is complex. Without care the advantage formula for $\mathbb{E} = \text{A3}$, for example, might easily include the term $Qq/2^k$ (without the guard of δ), which results in a poor bound for the choice $k = 80$.

SECURITY OF GAX. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small strong regularity, then GaX[$\mathbb{E}$] is prv-secure over Φ_{xor}. See Appendix C of the full version [4] for the omitted proof.

Theorem 2: Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the strong regularity and injectivity indicators of σ, respectively. Then $\mathbf{Adv}^{\text{prv.rpm},\,\Phi_{\text{xor}}}_{\text{GaX}[\mathbb{E}]}(\mathcal{A}, k) \le (6qQ + 15q^2)/2^\ell + (36rQ + 108rq)/2^k + \delta(48rQq + 84rq^2)/2^k$.

SECURITY OF GAXR. The following says that if $\mathbb{E}$ is σ-derived and its σ has a small strong regularity, then GaXR[$\mathbb{E}$] is prv-secure over Φ_{xor}. See Appendix D of the full version [4] for the omitted proof.

Theorem 3: Let $\mathcal{A}$ be an adversary that outputs circuits of at most q gates and makes at most Q queries to π and π^{-1}. Let $\mathbb{E}$ be a σ-derived DKC, where $\sigma : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \to \{0,1\}^\ell$, and let r and δ be the strong regularity and injectivity indicators of σ, respectively. Then $\mathbf{Adv}^{\text{prv.rpm},\,\Phi_{\text{xor}}}_{\text{GaXR}[\mathbb{E}]}(\mathcal{A}, k) \le (10qQ + 20q^2)/2^\ell + (36rQ + 123rq)/2^k + \delta(48rQq + 94rq^2)/2^k$.

DISCUSSION. The first use of the free-xor technique was justified in the ROM [28] but some subsequent works have been able to justify the use of garbling schemes within the standard model [3, 14]. We have not investigated whether GaX or GaXR can proven secure in the standard model.

V. JUSTGARBLE AND ITS PERFORMANCE

We have built a system, JustGarble, to realize the ideas described so far. The high speeds it achieves come from use of a fixed-key blockcipher and various implementation optimizations. We explore these factors here.

ARCHITECTURE. JustGarble starts with the idea (already advocated in BHR [5]) that garbling should be decoupled from MPC, oblivious transfer, and the compilation of programs into circuits. The separation of concerns facilitates construction of an efficient tool, but it also necessitates caution when comparing reported speeds.

To facilitate speed and interoperability, JustGarble uses a circuit representation that is simple and easy to work with: SCD, for Simple Circuit Description. SCD closely follows the formulation of circuits from BHR [5] recalled in Section II. An SCD file starts with values n, m, q, followed by arrays A, B, and G. If G is absent the file represents a topological circuit. For cross-language and cross-platform compatibility, values are encoded with MessagePack [18].

JustGarble consists of modules for building circuits, garbling them, and evaluating garbled circuits; see Fig. 7. The BUILD module can be used to construct circuits, working at the level of individual gates or collections of them. Constructed circuits are written to SCD files. The

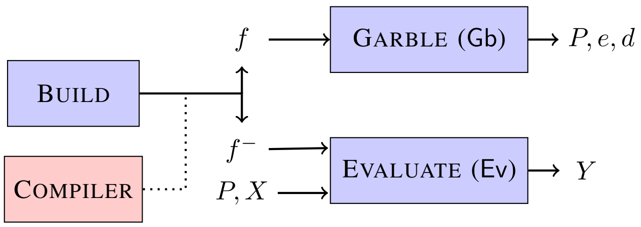

Figure 7. **The JustGarble framework.** The BUILD module or an external compiler can be used to generate a circuit f described in the SCD format, which is provided to the GARBLE module to get garbled tables P, token list e, and decoding data d. The EVALUATE module takes as input a circuit topology f^- also described in the SCD format, along with garbled tables P, and a token list X, and outputs a token list Y.

GARBLE module realizes the Gb algorithm of Ga, GaX, or GaXR. It can use any of the DKCs specified in this paper. GARBLE takes in an SCD-described circuit $f = (n, m, q, A, B, G)$ and produces the garbled tables P that comprise the final component of the associated garbled circuit $F = (n, m, q, A, B, P)$. The EVALUATE module takes in a topological circuit $f^- = (n, m, q, A, B)$, the garbled tables P needed to complete this, and a garbled input X. It produces the garbled output Y. JustGarble also includes simple routines (not shown in Fig. 7) to realize De, which maps the garbled output Y to the corresponding output y with the help of d.

The garbling module does not use the operating system to generate the pseudorandom bits needed for tokens; such a choice would not be cryptographically secure. Instead, pseudorandom bits are also generated by fixed-key AES, now operating in counter mode. At present, we use a different AES key than that employed for the random permutation underlying the selected DKC. We have verified that it would also work, cryptographically, to employ the same key for these conceptually distinct tasks. But there would be a small quantitative security loss, and the proofs would need to deal with this complication. With GaX-A2, the measured time savings from using the same permutation is at most 0.3 cpg.

JustGarble utilizes hardware AES support through AES-NI [21]. The system is written in C and employs compiler intrinsics to access SSE4 [25] instructions and 128-bit registers, which hold and manipulate the tokens.

JustGarble is entirely open-source and freely available for download [26].

We did test AES without NI support. As an example, Ga-A1 garbling and evaluation speeds are about 5.2 times slower in software (1147 cpg and 263 cpg), while GaX-A2 speeds are about 4.5 times slower (239 cpg and 65.2 cpg).

EXPERIMENTAL METHODOLOGY. We ran our experiments on an x86-64 Intel Core i7-970 processor clocked at 3.201 GHz with a 12MB L3 cache. Tests were compiled with gcc version 4.6, optimization level `-O3`, with support for SSE4 and AES-NI instructions through the `-sse4` and `-maes` flags. The tests were run in isolation, with processor frequency scaling turned off. We used the `rdtsc` instruction to count cycles.

We ran tests in batches of 1000 runs each, noting the median of the times recorded in the runs. This process was repeated for 1000 batches, and the final time reported is the mean of the batch medians. The cache was warm during the tests from initial runs. The standard deviation of the batch medians does not exceed 0.25 cpg in any of the experiments.

AES-CIRCUIT BENCHMARKS. We measure garbling and evaluation speeds on a circuit computing $\mathrm{AES128}_K(X)$ (hereafter simply AES) for a particular key K. This corresponds to a GC-based SFE of AES where the first party holds K and prepares a circuit for the second party, who holds X and wants to compute $\mathrm{AES}_K(X)$. We choose this setting because it has been used as a benchmark in prior work [22, 23, 29, 32], and hence helps compare our system with existing ones.

We build the AES circuit as described in HEKM [23]. The key is first expanded into 1280 bits. Conceptually, this is done locally by the party holding the key. We use a different S-box circuit [11] than HEKM, which results in a smaller AES circuit. This is not significant; as we measure speed in cycles per gate, small differences in circuit size are unlikely to have a noticeable effect on speed as long as the fraction of xor gates is little changed. Overall, our AES circuit has 36,480 gates, of which 29,820 (82%) are xor.

The evaluation and garbling speeds of $\mathrm{A1}, \mathrm{A2}, \mathrm{A3}$, and $\mathrm{A4}$ are listed in Fig. 1. For A2 we use doubling method D7; for A4, we use D3. These choices will be explained shortly. The fastest among our constructions, GaX with A2, evaluates the AES circuit at 23.2 cpb (7.25 ns/gate) and garbles it at 55.6 cpg (17.4 ns/gate). Overall, this comes to 637 μs for garbling the AES circuit and 264 μs for evaluating it.

Schemes A3 and A4 are a little slower than A1 and A2. Part of the speed difference may be due to JustGarble being better optimized for 128-bit tokens. Beyond this, there are memory-alignment overheads in dealing with 10-byte tokens: SSE4 instructions can have higher read and write latencies when data is not 16-byte aligned [25].

The sizes S_P we report in Fig. 1 measure only the contribution from the garbled tables: $S_P = |P|/8q$. Focusing on this value is justifiable because, in MPC applications, the other components of the GC, its topology, will be known and need not be communicated. Regardless, the size of the GC that JustGarble makes will always be $S_F = S_P + 8$ bytes, as gates are represented as four-byte numbers and we need to record two of these per gate—one for each of arrays A and B. Here we ignore the space to store n, m, q.

For the DKC A2, we implement doubling in many ways; see the definition for methods D1–D7 in Section III. We find D6 and D7 the fastest, followed by D2 and D3, then D4 and

Primitive	$\mathbb{E}(A,B,T,X) =$	Ga		GaX		GaXR	
		T_{E}	T_{G}	T_{E}	T_{G}	T_{E}	T_{G}
Permutation	$\pi(K) \oplus K \oplus X$, with $K \leftarrow 2A \oplus 4B \oplus T$	52.1	221	23.2	55.6	23.9	56.4
Blockcipher	$E(K,T) \oplus X$, with $K \leftarrow A \| B$	256	991	60.1	172	58.7	171
Hash function	$H(K \parallel T)[1:k] \oplus X$, with $K \leftarrow A \| B$	875	3460	161	566	160	568

Figure 8. **Permutation-based, blockcipher-based, and hash-based garbling**. The T_{E} (time to evaluate) and T_{G} (time to garble) values are in mean **cycles per gate (cpg)** using the subject AES circuit. The first method, A2, is based on a permutation $\pi\colon \{0,1\}^k \to \{0,1\}^k$. The permutation chosen is fixed-key AES128. The second method, from KSS [29], uses a blockcipher $E\colon \{0,1\}^{2k} \times \{0,1\}^k \to \{0,1\}^k$. The selected blockcipher is AES256. The last method, employed in [23], builds a DKC from a hash $H\colon \{0,1\}^* \to \{0,1\}^k$. The hash function chosen is SHA-1.

D5, and finally D1. The speed of D6 and D7 (SIMD shift) is due to the availability of a matching SSE4 instruction. The speed difference between the fastest and slowest doubling methods is $\Delta T_E \approx 7$ cpg and $\Delta T_G \approx 11$ cpg. We find this significant enough to trade a small quantity in the security bound, which is why we select A2 with D7 doubling. For the DKC A4, which uses 10-byte tokens, similar experiments lead us to select the doubling scheme D3.

LARGER CIRCUITS. The size of the garbled table for each non-xor gate ranges from 30 bytes (GaXR with A3, A4) to 64 bytes (GaX with A1, A2). This means that even circuits with hundreds of thousands of gates can fit in the processor's L3 cache during evaluation. However, if the circuit is too big to fit entirely in the cache, per-gate garbling and evaluation times will increase.

To understand the performance of JustGarble on circuits larger than the cache size, we measured garbling and evaluation times of the modular exponentiation (MEXP) ("RSA circuits") and edit distance (EDT) circuits of KSS with various input sizes. We used GaX with A2 (henceforth GaX-A2); see Fig. 9. The MEXP-ℓ circuit takes inputs a and b and returns $a^b \bmod c$ for $c = 1^8 0^{\ell-9} 1$. The EDT-m circuit takes as inputs two m-bit strings and returns their edit distance as a $(\lg m)$-bit integer. We obtained these circuits by patching the KSS compiler to produce outputs in SCD format. The garbling and evaluation times (in cycles per gate) are higher than the measured values for the AES circuit due to higher latencies involved in reading data directly from main memory. However, JustGarble is still several times faster than KSS report. Taking RSA-32 as an example, KSS report a garbling time of 4.53 seconds, which translates to 6546 cpg, while JustGarble uses 91.6 cpg, a 70x speedup.

At present, JustGarble cannot handle circuits that are too big to fit in main memory. An obvious direction for future work is extending JustGarble with a streaming mode of operation that can garble and evaluate large circuits by keeping only a small portion in memory at any given point.

COMPARISONS. JustGarble garbles and evaluates moderately-sized circuits about two orders of magnitude faster than what recent MPC implementations of HKEM and KSS report [23, 29]. For evaluating an AES circuit, the best previously-reported figure comes from KSS [29], garbling the circuit in 80 ms. The fastest among our own constructions, GaX using A2, does the job in 638 μs. We note that both systems use AES-NI and SSE4 instructions and the free-xor optimization, and that, in both cases, the reported times are for garbling alone, excluding other operations and network overhead. One reason JustGarble performs better is that it spends less time on non-cryptographic operations, by which we mean all operations other than the DKC computations. Moreover, using a fixed-key DKC like A2 results in a sizable gain in performance, in spite of the large percentage of xor gates (82%) in the AES circuit. We measured the contributions of both of these factors as below.

JustGarble spends about 23% and 43% of its time on non-cryptographic operations when GaXR-A2 does garbling and garbled-circuit evaluation, respectively. In contrast, KSS measure AES256 (with AES-NI) overhead at 225 cycles per invocation but report an overall GaXR garbling time of over 6000 cpg, suggesting that close to 95% of the garbling time is non-cryptographic overhead. The reduced overhead is likely connected to our simple representation of circuits, one consequence of which is the absence of a need to maintain a queue of ready gates. A downside of this simple circuit representation is that, unlike HEKM and KSS, JustGarble cannot handle circuits that do not fit in memory.

To measure the contribution of the DKC itself we implemented within JustGarble the blockcipher-based DKC from KSS and the hash-function based DKC from HEKM; see Fig. 8. Let us focus on GaXR, as free-xor and garbled-row reduction are both employed in the MPC systems of KSS and HEKM. Comparing the first and second rows, the DKC-attributable speedup we get by using a permutation instead of a blockcipher is 2.5-fold improvement in evaluation time and 3-fold improvement in garbling time. Comparing the first and the third rows, the DKC-attributable speedup we get by using a permutation instead of a cryptographic hash function is 6.7-fold improvement in evaluation time and 10-fold improvement in garbling time. One may conclude that the improved DKCs play a large role in our performance gains—a factor of about 2.5 to 10—yet more mileage is obtained through other aspects of JustGarble.

Circuit	Gates	Xor gates	T_{E}	T_{G}
MEXP-16	0.21 M	0.14 M	44.1	91.6
MEXP-32	1.75 M	1.15 M	45.3	96.3
MEXP-64	14.3 M	9.31 M	44.6	95.8
EDT-255	15.5 M	9.11 M	48.4	101.3

Figure 9. **Larger circuits.** Evaluation times (T_{E}) and garbling times (T_{G}) are in median **cycles per gate** using GaX-A2. The modular exponentiation (MEXP) and edit distance (EDT) circuits are described in text. Gate counts are in **millions of gates** (1M = 1 million gates).

Acknowledgments

Many thanks to the NSF, who sponsored this work under CNS 0904380, CNS 1116800, and CNS 1228828/90.

References

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[2] M. Abdalla, X. Boyen, C. Chevalier, and D. Pointcheval. Distributed public-key cryptography from weak secrets. In *PKC 2009*, volume 5443 of *LNCS*, pages 139–159. Springer, Mar. 2009.

[3] B. Applebaum. Garbling XOR gates "for free" in the standard model. In *TCC 2013*, volume 7785 of LNCS, pages 162–181, Springer, 2013.

[4] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. ePrint Archive, 2013. Full version of this paper.

[5] M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Computer and Communications Security (CCS'12)*. ACM, 2012. Full version as ePrint Archive, Report 2012/265, May, 2012.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993

[7] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM CCS 08*, pages 257–266. ACM Press, Oct. 2008.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Assche. Keccak specifications. Submission to NIST, 2009.

[9] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Feb. 2009.

[10] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. Sepia: privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240, 2010.

[11] D. Canright. A very compact S-box for AES. *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 441–455, 2005.

[12] S. Choi, K. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *CT-RSA*, pages 416–432, 2012.

[13] S. Choi, J. Katz, R. Kumaresan, and H. Zhou. On the security of the "free-XOR" technique. Cryptology ePrint Archive, Report 2011/510, Sep 2011.

[14] S. Choi, J. Katz, R. Kumaresan, and H. Zhou. On the security of the "free-xor" technique. In *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Mar. 2012.

[15] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: how to construct a hash function. In *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Aug. 2005.

[16] Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging Merkle-Damgård for practical applications. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 371–388. Springer, Apr. 2009.

[17] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell. SCAPI: the secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012.

[18] S. Furuhashi. The MessagePack format. http://msgpack.org.

[19] O. Goldreich. Cryptography and cryptographic protocols. Manuscript, June 9 2001.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[21] S. Gueron. Advanced encryption standard (AES) instructions set. *Intel Corporation*, 25, 2008.

[22] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM CCS 10*, pages 451–462. ACM Press, Oct. 2010.

[23] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[24] Y. Huang, C. Shen, D. Evans, J. Katz, and A. Shelat. Efficient secure computation with garbled circuits. In *ICISS*, volume 7093 of *Lecture Notes in Computer Science*, pages 28–48. Springer, 2011.

[25] Intel. Intel SSE4 manual. http://software.intel.com/file/17971/

[26] JustGarble source. http://cseweb.ucsd.edu/groups/justgarble

[27] K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *FC 2010*, volume 6052 of *LNCS*, pages 207–221. Springer, Jan. 2010.

[28] V. Kolesnikov and T. Schneider. Improved garbled circuit: free XOR gates and applications. In *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.

[29] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21th USENIX Security Symposium (USENIX 2012)*, 2012. Full version as Cryptology ePrint Archive, Report 2012/179.

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

[31] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN 08*, volume 5229 of *LNCS*, pages 2–20. Springer, Sept. 2008.

[32] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 20–20. USENIX Association, 2004.

[33] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Feb. 2004.

[34] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[35] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[36] J. Nielsen and C. Orlandi. LEGO for two-party secure computation. *Theory of Cryptography*, pages 368–386, 2009.

[37] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Dec. 2009.

[38] P. Rogaway. The round complexity of secure protocols. MIT Ph.D. Thesis, 1991.

[39] P. Rogaway and J. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *CRYPTO 2008*, volume 5157 of *LNCS*, pages 433–450. Springer, Aug. 2008.

[40] R. Winternitz. A secure one-way hash function built from DES. *Proceedings of the IEEE Symposium on Information Security and Privacy*, pages 88–90. IEEE Press, 1984.

[41] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[42] A. Yao. Protocols for secure computations. In *Foundations of Computer Science, 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

Appendix A. Invertibility of $\Phi_{\rm xor}$

We recall the notion of *efficient invertibility* of BHR [5]. Let Φ be a side-information function. An algorithm M is called a (Φ, ev)-inverter if on input (ϕ, y), where $\phi = \Phi(f')$ and $y = \mathsf{ev}(f', x')$ for some f' and $x \in \{0,1\}^{f'.n}$, it returns an (f, x) satisfying $\Phi(f) = \phi$ and $\mathsf{ev}(f, x) = y$. We say that (Φ, ev) is efficiently invertible if there is a polynomial-time (Φ, ev)-inverter.

Proposition 1: There exists a cubic-time $(\Phi_{\rm xor}, \mathsf{ev}_{\sf circ})$-inverter.

Proof of Proposition 1: We specify a cubic-time $(\Phi_{\rm xor}, \mathsf{ev}_{\sf circ})$-inverter $M_{\rm xor}$ as follows. Let $\textsc{Gauss}(S)$ be the algorithm that takes as input a system S of linear equations in GF(2), uses Gaussian elimination to solve it, and then lets each free variable be 0. The inverter $M_{\rm xor}$ gets as input $\phi = (n, m, q, A, B, G')$ and a string $y \in \{0,1\}^m$, and proceeds as follows.

```
proc $M_{\rm xor}(\phi, y)$
$(n, m, q, A, B, G') \leftarrow \phi$,  $y_1 \cdots y_m \leftarrow y$,  $S \leftarrow \emptyset$
for $g \in \{n+1, \ldots, n+q\}$ do
   $a \leftarrow A(g)$,  $b \leftarrow B(g)$
   if $G'_g$ = XOR then
      if $g \le n+q-m$ then $S \leftarrow S \cup \{x_a \oplus x_b \oplus x_g = 0\}$
      else $S \leftarrow S \cup \{x_a \oplus x_b = y_{g-(n+q-m)}\}$
$(x_1, \ldots, x_{n+q-m}) \leftarrow \textsc{Gauss}(S)$
for $(g, i, j) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ do
   if $G'_g$ = XOR then $G_g \leftarrow$ XOR else $G_g(i,j) \leftarrow x_g$
$f \leftarrow (n, m, q, A, B, G)$,  $x \leftarrow x_1 \cdots x_n$
return $(f, x)$
```

We then have (f, x) as desired. The system S has $q+n-m$ variables, and at most q equations. Hence the running time of $\textsc{Gauss}(S)$ is at most $O\big((q+n)^3\big)$, and so is $M_{\rm xor}$'s running time. ■

Appendix B. Proof of Theorem 1

In our code, a procedure with the keyword "private" is local to the caller, and thus cannot be invoked by the adversary. It can be viewed as a function-like macro in the C/C++ programming language. That is, it still has read/write access to the variables of the caller, even if these variables are not its parameters. Consider games G_0–G_2 in Fig. 10. They share the same code for procedure $\textsc{Garble}$, but each has a different implementation of a local procedure $\textsc{GarbleRow}$. The adversary $\mathcal{A}$ makes queries to procedures Π and Π^{-1} to access an ideal permutation π, which is implemented lazily. Without loss of generality, assume that $q + Q \le 2^{k-2}/r$; otherwise the theorem is trivially true.

We reformulate game $\mathrm{Prv}_{\mathrm{Ga}, \Phi_{\rm topo}, k, \pi}$ as game G_0. Recall that in the scheme Ga, each wire i carries tokens X_i^0 and X_i^1 with semantics 0 and 1 respectively. If wire i ends up having value (semantics) v_i in the computation $v \leftarrow \mathsf{ev}(f_c, x_c)$, where c is the challenge bit, then token $X_i^{v_i}$ becomes visible to $\mathcal{A}$ while $X_i^{\overline{v}_i}$ stays invisible. Game G_0 makes this explicit. It picks for each wire i a "visible" token and an "invisible" one. Each garbled row that can be opened by visible tokens will be built directly in $\textsc{Garble}$. To construct

```
proc $\text{GARBLE}(f_0, f_1, x_0, x_1)$
$(n, m, q, A', B', G) \leftarrow f_c$
for $i \leftarrow 1$ to $n+q$ do
    $v_i \leftarrow \mathsf{ev}(f_c, x_c, i),\ t_i \twoheadleftarrow \{0,1\},\ X_i^{v_i} \twoheadleftarrow \{0,1\}^{k-1} t_i,\ X_i^{\overline{v}_i} \twoheadleftarrow \{0,1\}^{k-1} \overline{t}_i$
for $g \leftarrow n+1$ to $n+q,\ i \leftarrow 0$ to $1,\ j \leftarrow 0$ to $1$ do
    $a \leftarrow A'(g),\ b \leftarrow B'(g)$
    $A \leftarrow X_a^i,\ B \leftarrow X_b^j,\ \mathtt{a} \leftarrow \text{lsb}(A),\ \mathtt{b} \leftarrow \text{lsb}(B),\ K \leftarrow \sigma(A, B, g)$
    if $i = v_a$ and $j = v_b$ then $P[g, \mathtt{a}, \mathtt{b}] \leftarrow (\Pi(K) \oplus K)[1\colon k] \oplus X_g^{v_g}$ else $P[g, \mathtt{a}, \mathtt{b}] \leftarrow \text{GARBLEROW}()$
$F \leftarrow (n, m, q, A', B', P),\ X \leftarrow (X_1^{v_1}, \ldots, X_n^{v_n})$
$d \leftarrow \big(\text{lsb}(X_{n+q-m+1}^0), \ldots, \text{lsb}(X_{n+q}^0)\big)$
return $(F, X, d)$
```

```
private proc $\text{GARBLEROW}()$
$S \twoheadleftarrow \{0,1\}^\ell$
if $K \in \text{Dom}(\pi)$ or $S \oplus K \in \text{Ran}(\pi)$ then
    $bad \leftarrow \mathsf{true}$
    $S \leftarrow \Pi(K) \oplus K$        $\longleftarrow$ Use in game $\text{G}_0$
$Y \leftarrow S[1\colon k] \oplus X_g^{G_g(i,j)},\ \pi[K] \leftarrow S \oplus K$
return $Y$
```

```
proc $\Pi(u)$              Game $\text{G}_0$ / Game $\text{G}_1$
if $u \notin \text{Dom}(\pi)$ then $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \text{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \notin \text{Ran}(\pi)$ then
    $u \twoheadleftarrow \{0,1\}^\ell \backslash \text{Dom}(\pi),\ \pi[u] \leftarrow v$
return $\pi^{-1}[v]$
```

```
private proc $\text{GARBLEROW}()$
$S \twoheadleftarrow \{0,1\}^\ell,\ Y \leftarrow S[1\colon k] \oplus X_g^{G_g(i,j)}$
$\text{BadDom} \leftarrow \text{BadDom} \cup \{K\}$
$\text{BadRan} \leftarrow \text{BadRan} \cup \{K \oplus S\}$
return $Y$
```

```
proc $\Pi(u)$              Game $\text{G}_2$
if $u \in \text{BadDom}$ then $bad \leftarrow \mathsf{true}$
if $u \notin \text{Dom}(\pi)$ then $\pi[u] \twoheadleftarrow \{0,1\}^\ell \backslash \text{Ran}(\pi)$
return $\pi[u]$

proc $\Pi^{-1}(v)$
if $v \in \text{BadRan}$ then $bad \leftarrow \mathsf{true}$
if $v \notin \text{Ran}(\pi)$ then
    $u \twoheadleftarrow \{0,1\}^\ell \backslash \text{Dom}(\pi),\ \pi[u] \leftarrow v$
return $\pi^{-1}[v]$
```

Figure 10. **Games for the proof of Theorem 1.** Each set is initialized to be $\emptyset$. Initially, procedure INITIALIZE() samples the challenge bit $c \twoheadleftarrow \{0,1\}$.

each other garbled row, we invoke the "private" procedure GARBLEROW, which inherits all variables of GARBLE.

We explain the game chain up until the terminal game. $\triangleright \text{G}_0 \to \text{G}_1$: the two games are identical until either game sets bad. In these games, we sample a uniformly random string S and want to set $\pi(K)$ to $K \oplus S$. This may cause inconsistency if $\pi(K)$ or $\pi^{-1}(K \oplus S)$ is already defined, triggering bad. In this case, G_0 resets S to the consistent value, but game G_1 does nothing. Hence in game G_1, a point $v \in \text{Ran}(\pi)$ may have several preimages, and in that case $\pi^{-1}[v]$ means an arbitrary preimage.

We now bound the chance that G_1 sets bad. Consider the ith invocation of GARBLEROW. It triggers bad to true if its string K falls into $\text{Dom}(\pi)$ or $S \oplus K$ falls into $\text{Ran}(\pi)$, with $S \twoheadleftarrow \{0,1\}^\ell$. Since $|\text{Ran}(\pi)| \le (Q+q+i-1)$, the latter happens with probability at most $(Q+i+q-1)/2^\ell$. Let $K = \sigma(A, B, g)$. We claim that the chance that $K \in \text{Dom}(\pi)$ is at most $6r/2^k + N_i r(2\delta+1)/2^k$, where N_i is the size of $\text{Dom}(\pi) \cap \big\{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\big\}$, which is at most $|\text{Dom}(\pi)| \le Q+q+i-1$. By the union bound, the chance that G_1 sets bad is at most

$$\begin{aligned} &\sum_{i=1}^{3q} \frac{Q+q+i-1}{2^\ell} + \frac{6r}{2^k} + \frac{rN_i(2\delta+1)}{2^k} \\ \le\ & \frac{3qQ + 7.5q^2}{2^\ell} + \frac{3rQ + 30rq}{2^k} + \frac{\delta(9rqQ + 22.5rq^2)}{2^k}. \end{aligned}$$

If $\delta = 1$ the last inequality is obvious, as $N_i \le Q+q+i-1$. For the case $\delta = 0$, note that for each string s there is at most one value g such that $s \in \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$. Hence when we sum up the numbers N_i, because the invocations of GARBLEROW use each tweak value at most three times, we count each point in $\text{Dom}(\pi)$ at most three times, so the sum is at most $3|\text{Dom}(\pi)| \le 3(Q+4q)$.

We now justify the claim above. Consider the moment that procedure GARBLE makes the ith call to GARBLEROW. Let D_1 be the set of points in $\text{Dom}(\pi)$ created by adversarial queries before its querying GARBLE, and let D_2 be the set of points in $\text{Dom}(\pi)$ created by procedure GARBLE so far. Then $D_1 \cup D_2 = \text{Dom}(\pi)$. Recall that $K = \sigma(A, B, g) = \sigma(A, 0^k, 0^\tau) \oplus \sigma(0^k, B, 0^\tau) \oplus \sigma(0^k, 0^k, g)$. Because $A \twoheadleftarrow \{0,1\}^k$ and r is the regularity of σ, it follows that $\Pr[\sigma(A, 0^k, 0^\tau) = s] \le r/2^k$ for any string $s \in \{0,1\}^\ell$. Since A is independent of B and all points in D_1, the chance that $K \in D_1$ is at most $rN_i/2^k$.

What remains is to show that $\Pr[K \in D_2] \le 6r/2^k + 2N_i r\delta/2^k$. Consider an arbitrary point $K^* \in D_2$. Let $K^* = \sigma(A^*, B^*, g^*)$. If $A \equiv A^*$ and $B \equiv B^*$ then K and K^* belong to different gates, and thus $g \ne g^*$. Hence $K \oplus K^* = \sigma(A, B, g) \oplus \sigma(A, B, g^*) = \sigma(0^k, 0^k, g \oplus g^*) \ne 0^\ell$. Otherwise, wlog, suppose that $A[1\colon k-1]$ is independent of B, A^*, and B^*. For any string $s \in \{0,1\}^\ell$, as r is the regularity of σ, there are at most r strings x such that $\sigma(x, 0^k, 0^\tau) = s$. Given B, A^*, and B^*, because each but the last bit of A is still uniformly random, the conditional probability that A falls into one of the r strings above is at most $2r/2^k$, and thus the conditional probability that $\sigma(A, 0^k, 0^\tau) = s$ is at most $2r/2^k$. Hence $\Pr[K = K^*] \le 2r/2^k$. Moreover, if the injectivity indicator $\delta = 0$ and $g \ne g^*$ then $K \ne K^*$. In other words, $\Pr[K = K^*] \le 2r/2^k$ if $g = g^*$, and $\Pr[K = K^*] \le 2r\delta/2^k$ otherwise. Summing up, $\Pr[K \in D_2] \le 6r/2^k + 2N_i r\delta/2^k$, because there are most three elements of D_2 using the tweak g.

$\triangleright \mathrm{G}_1 \to \mathrm{G}_2$: in game G_1 we write $\pi[K] \leftarrow S \oplus K$, but game G_2 omits this step. In addition, we maintain two sets, BadDom and BadRan, each of which are initialized to the empty set. Each call to GARBLEROW will add K to BadDom and $S \oplus K$ to BadRan. The two games are identical until G_2 sets *bad*, that is, when $\mathcal{A}$ happens to query $\Pi(u)$ with $u \in \mathrm{BadDom}$, or $\Pi^{-1}(v)$ with $v \in \mathrm{BadRan}$. Since G_2 samples S uniformly at random, and doesn't store it in π, the output of GARBLEROW() is uniformly random, independent of the token that S masks.

We now bound the chance that G_2 sets *bad*. Consider an arbitrary point $K \in \mathrm{BadDom}$. It has a corresponding point $K \oplus S \in \mathrm{BadRan}$. Let $K = \sigma(A, B, g)$. Either A or B must be invisible. Wlog, suppose that A is invisible. Condition on the output of GARBLE. Initially, as each but the last bit of A is still uniformly random and the regularity of σ is r, the conditional probability that $K = s$ is at most $2r/2^k$ for any string $s \in \{0,1\}^\ell$. Consider a query u to Π. Each prior query to Π or Π^{-1} removes at most one value of K. Since there are at most $q + Q$ queries to Π and Π^{-1} (procedure GARBLE only queries Π for q rows that can be opened by visible tokens), the chance that u hits K is at most

$$\frac{2r/2^k}{1 - 2(Q+q)r/2^k} = \frac{2r}{2^k - 2r(Q+q)} \le 4r/2^k,$$

where the last inequality is due to the assumption $Q + q \le 2^{k-2}/r$. By the union bound, the chance that $u \in \mathrm{BadDom}$ is at most $12rq/2^k$. But if the injectivity indicator $\delta = 0$ then there is at most one possible value of g such that $u \in \{\sigma(x, y, g) \mid x, y \in \{0,1\}^k\}$, and, consequently, $\Pr[u \in \mathrm{BadDom}] \le 12r/2^k$ because each tweak value is used at most three times in BadDom. Hence in general, $\Pr[u \in \mathrm{BadDom}] \le 12r(q\delta + 1)/2^k$. Likewise, for each query v to Π^{-1}, the chance that $v \in \mathrm{BadRan}$ is at most $12r(q\delta{+}1)/2^k$. By the union bound, the chance that game G_2 sets *bad* is at most

$$\begin{aligned} &12r(Q+q)(q\delta+1)/2^k \\ &= \quad (12rQ + 12rq)/2^k + \delta(12rQq + 12rq^2)/2^k \ . \end{aligned}$$

```
proc $\textsc{Garble}(f_0, f_1, x_0, x_1)$
$(n, m, q, A', B') \leftarrow \Phi_{\mathrm{topo}}(f_0)$
$v_{q+n-m+1} \cdots v_{q+n} \leftarrow \mathsf{ev}(f_0, x_0)$
for $i \leftarrow 1$ to $n + q$ do
   $t_i \twoheadleftarrow \{0,1\}$,  $V_i \twoheadleftarrow \{0,1\}^{k-1} t_i$,  $I_i \twoheadleftarrow \{0,1\}^{k-1} \bar{t}_i$
for $i \leftarrow n + q - m + 1$ to $n + q$ do
   $X_i^{v_i} \leftarrow V_i$,  $X_i^{\bar{v}_i} \leftarrow I_i$
for $g \leftarrow n + 1$ to $n + q$ do
   $a \leftarrow A'(g)$,   $b \leftarrow B'(g)$
   for $(A, B) \in \{V_a, I_a\} \times \{V_b, I_b\}$ do
      $\mathtt{a} \leftarrow \mathrm{lsb}(A)$,  $\mathtt{b} \leftarrow \mathrm{lsb}(B)$,  $K \leftarrow \sigma(A, B, g)$
      if $A = V_a$ and $B = V_b$ then
         $Y \leftarrow (\Pi(K) \oplus K)[1\colon k] \oplus V_g$
      else $Y \twoheadleftarrow \{0,1\}^k$
      $P[g, \mathtt{a}, \mathtt{b}] \leftarrow Y$
$F \leftarrow (n, m, q, A', B', P)$,  $X \leftarrow (V_1, \ldots, V_n)$
$d \leftarrow \big(\mathrm{lsb}(X^0_{n+q-m+1}), \ldots, \mathrm{lsb}(X^0_{n+q})\big)$
return $(F, X, d)$
```

Figure 11. **Rewritten game G_2 of the proof of Theorem 1.** This game depends solely on the topological circuit $f^- = \Phi_{\mathrm{topo}}(f_0) = \Phi_{\mathrm{topo}}(f_1)$ and the output $v = \mathsf{ev}(f_0, x_0) = \mathsf{ev}(f_1, x_1)$. Procedures Π and Π^{-1} lazily implement a random permutation and its inverse, respectively.

ANALYSIS OF GAME G_2. The output of game G_2 is independent of the challenge bit c. Hence $\Pr[\mathrm{G}_2^{\mathcal{A}}(k)] = 1/2$. To justify this, from a topological circuit f^- and the final output $v = \mathsf{ev}(f_c, x_c)$, which is independent of c, we can rewrite the code of procedure GARBLE of game G_2 as shown in Fig. 11. There, we refer to the visible token of wire i as V_i, and its invisible counterpart as I_i, omitting the semantics of these tokens. Each garbled row is an independent, uniformly random string, except for rows that can be opened by visible tokens. Summing up,

$$\begin{aligned} &\mathbf{Adv}^{\mathrm{prv.rpm},\,\Phi_{\mathrm{topo}}}_{\mathrm{Ga}[\mathbb{E}]}(\mathcal{A}, k) \\ &\quad = \quad 2(\Pr[\mathrm{G}_0^{\mathcal{A}}(k)] - \Pr[\mathrm{G}_2^{\mathcal{A}}(k)]) \\ &\quad \le \frac{6qQ + 15q^2}{2^\ell} + \frac{30rQ + 84rq}{2^k} + \frac{\delta(42rQq + 69rq^2)}{2^k}, \end{aligned}$$

concluding the proof.