Garbling Gadgets for Boolean and Arithmetic Circuits

Marshall Ball
Columbia Uninversity
New York, NY
marshall@cs.columbia.edu

Tal Malkin
Columbia University
New York, NY
tal@cs.columbia.edu

Mike Rosulek
Oregon State University
Corvallis, Oregon
rosulekm@eecs.
oregonstate.edu

ABSTRACT

We present simple, practical, and powerful new techniques for garbled circuits. These techniques result in significant concrete and asymptotic improvements over the state of the art, for several natural kinds of computations.

For arithmetic circuits over the integers, our construction results in garbled circuits with *free* addition, weighted threshold gates with cost independent of fan-in, and exponentiation by a fixed exponent with cost independent of the exponent. For boolean circuits, our construction gives an *exponential* improvement over the state of the art for threshold gates (including AND/OR gates) of high fan-in.

Our construction can be efficiently instantiated with practical symmetric-key primitives (e.g., AES), and is proven secure under similar assumptions to that of the Free-XOR garbling scheme (Kolesnikov & Schneider, ICALP 2008). We give an extensive comparison between our scheme and state-of-the-art garbling schemes applied to boolean circuits.

1. INTRODUCTION

Garbled circuits were famously introduced by Yao in the 1980s [20]. Since that time they have become an invaluable technique for both practical and theoretical cryptographic constructions. Most notably, garbled circuits form the conceptual core for the most practical approaches to secure two-party computation. In these protocols, the garbled circuits are the major performance bottleneck both in computation and communication.

A considerable amount of work [5, 16, 14, 18, 6, 13, 21, 11] has been dedicated to reducing the cost of garbled circuits since Yao's seminal construction. The current state of the art provides highly efficient garbling for boolean circuits expressed using XOR, NOT, & AND gates. Concretely, using the most recent *half gates* construction of Zahur, Rosulek and Evans [21], XOR & NOT involve *no* computation or communication, while fan-in-2 AND gates require 4 AES calls to garble, 2 AES calls to evaluate, and 256 bits to communicate. Implementations like JustGarble [6], which take advantage of hardware-accelerated AES, can garble circuits at a rate of 100s of millions AND gates per second on consumer hardware.

CCS'16, October 24 - 28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4139-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2976749.2978410

Despite this success story, garbled circuits remain tied deeply to boolean circuits. Many computations of interest are cumbersome and expensive to express as boolean circuits. As two specific examples (which our contributions address directly):

- Threshold computations with very high fan-in (for example, the kinds of computations that might be found in a neural-network-based classifier) do not have small boolean circuits.
- Arithmetic computations (over the integers or in a ring mod m) are poorly suited to boolean circuits, especially when compared to other techniques for secure computation that are based on secret sharing. In particular, secret-sharing-based secure computation protocols allow additions *for free*, whereas addition in a boolean circuit requires non-free AND gates (even ignoring a possible modular reduction step).

Our work aims to address these shortcomings of boolean circuits and directly construct garbled circuit techniques supporting these advanced kinds of computations.

1.1 Our Contributions

We show a practical garbling scheme that can be used to natively garble both boolean circuits and arithmetic circuits (with arithmetic over a large modulus), applying insights and techniques from one domain to the other. Our approach gives significant concrete & asymptotic improvements over the state of the art. In particular, our most extreme improvements are for the following kinds of computations (below λ is a security parameter, e.g., 128 bits):

Linear operations in arithmetic circuits. Our scheme supports addition and multiplication by a public constant *for free,* over the integers. Other costs in the scheme (i.e., size of wire labels & cost of other gates) depend only *polylogarithmically* on the maximum size of integers in the computation.

In this way, our construction combines the best aspects of the two main paradigms for secure computation: free addition (beyond addition mod 2) as in secret-sharing-based MPC, and constant-round protocols from garbled circuits.

Current garbled circuit techniques would represent integers in binary and incur $O(\lambda\,\ell)$ cost to add two ℓ-bit numbers.

Other arithmetic operations. Our scheme supports exponentiation (by a fixed/public power) with cost independent of the choice of exponent, and weighted threshold gates with cost that is independent of the fan-in.

High fan-in boolean threshold gates. For gates of fan-in b, our construction requires $O(\lambda \log^3 b)$ bits, or only $O(\lambda \log^2 b)$ bits in the special case of AND/OR gates. Current techniques are exponentially worse, requiring $\Theta(\lambda b)$ bits even for AND/OR gates.

On the other hand, our scheme does least well on comparison gates, where we are polynomially slower. We give more in-depth comparisons between our scheme and existing techniques in Sections 7 and 8. We also explore the case of natural linear algebraic operations over the integers (e.g., matrix multiplication) and show that our techniques give close to an order of magnitude improvement.

Finally, we use our construction to circumvent the lower bound of [21]. They proved that any garbling scheme using "known techniques" requires 2λ bits to garble a single AND-gate, while we show an instantiation of our construction that garbles a single AND-gate using only λ bits. This instantiation is of theoretical interest, but does not lead to improvements for general larger circuits.

1.2 Techniques

Our results build on a simple and powerful combination of techniques that were introduced in previous works in several contexts. In particular, our garbling scheme is based on a natural generalization of the Free-XOR technique of Kolesnikov & Schneider [14], allowing for free addition mod larger m, rather than just mod 2. This generalization was shown before, e.g. by Malkin, Pastro and shelat [15], who used it to obtain free addition in arithmetic (mod m) circuits. We observe that this technique is in fact useful not only for mod-m addition, but for any operation that depends only on the Hamming weight of its inputs (namely, symmetric operation): such operations can be garbled by first applying free addition to get the Hamming weight, then applying a *projection* gadget which garbles a unary mapping of each sum to the correct output. This projection gadget can be viewed as the trivial extension of Yao's garbled gate, applied to unary gates over mod m inputs (similar gadgets have been used before, at least implicitly, e.g. in [14, 13]).

In Section 4 we formally describe these simple components as a garbling scheme for what we call "mixed moduli simple circuits", which are circuits that allow only modular addition (under many moduli) and projections. As we show in Section 5, these simple components already provide savings over the state of the art, even for boolean circuits, through simple ways to represent boolean gadgets as mixed moduli circuits. One example is a boolean fan-in-b AND gate, which has output equal 1 if and only if the sum of its inputs equals b, and thus can be represented as a projection of sum. This representation can be viewed as an extension of the one by Nielsen and Orlandi [17], who (in a different context) represent a fan-in-2 NAND gate by adding the two inputs over the integers and then giving a gadget that checks whether or not the result equals 2.

While the above ideas directly handle symmetric operations like boolean AND and threshold gates, as well as arithmetic addition, the cost grows prohibitively high as the modulus (or the fan-in) grows, and other operations such as comparison or modular multiplication are also highly inefficient. We address this in Section 6 by showing how to express those more complex operations with high moduli more efficiently as mixed moduli circuits. In particular, we represent large-modulus values using the Chinese Remainder Theorem (CRT), as well as another primorial mixed-radix representation, together with several other optimizations (as we will explain). We note that CRT-based representations have also been used in many other contexts, with the most relevant one being by Applebaum, Ishai, and Kushilevitz [2] who (in addition to their more efficient main result) outline a garbling scheme for arithmetic circuits, relying on first encoding the inputs via CRT encoding, and then applying standard boolean garbled circuit techniques.

In Section 7 we discuss how to use our gadgets for better garbling of boolean and arithmetic circuits, provide asymptotic and concrete comparison with standard garbled circuit techniques, and discuss a linear algebraic application scenario. Finally in Section 8 we give a more in-depth comparison to the related work discussed above, and to other 2PC techniques.

2. PRELIMINARIES

Logarithms are taken to be base 2, unless otherwise noted. We take p_i to denote the i-th prime. Let $\mathbb{Z}$ denote the integers, and $\mathbb{N}$ the natural numbers. Additionally, $\mathbb{Z}_m$ denotes the ring of integers modulo $m \in \mathbb{N}$. We use $[x]_m$ to denote the residue of x mod m. In some cases it is convenient to define $<$ on $\mathbb{Z}_m$. For concreteness, if $m = 2k + 1$ for some k, then consider $\mathbb{Z}_m$ as $\{-k, \dots, -1, 0, 1, \dots, k\}$ and let $x < y$ if $y - x > 0$ over $\mathbb{Z}$. Otherwise if $m = 2k$, then consider $\mathbb{Z}_m$ as $\{-k + 1, \dots, -1, 0, 1, \dots, k\}$, defining order identically.

2.1 Garbled Circuits

We use the *garbling schemes* abstraction and security definitions of Bellare, Hoang and Rogaway [7]. Roughly speaking, a garbling scheme consists of the following algorithms:

- Gb: given input a circuit f, generates garbled circuit F, encoding information e, and decoding information d
- En: given a circuit-input x, encoding information e, generates garbled input X
- Ev: given garbled circuit F and garbled input X, generates garbled output Y
- De: given garbled output Y and decoding information d, generates plaintext circuit-output y

Bellare et al. identify 3 natural security properties for a garbling scheme, which we summarize below. For more details, we refer the reader to [7]:

- Privacy (prv.sim security): intuitively, the distribution of values (F, X, d) — generated as above — leaks no more than $f(x)$. More specifically, there exists a simulator that can simulate the joint distribution of (F, X, d) given just f and $f(x)$.
- Obliviousness (obv.sim security): the values (F, X) alone (i.e., without d) leak nothing about x. That is, there exists a simulator that can simulate (F, X) given just f.

- Authenticity (aut security): Given (F, X), it is infeasible for an adversary to generate $\tilde{Y} \neq \mathsf{Ev}(F, X)$ such that $\mathsf{De}(d, Y) \neq \perp$.

3. BACKGROUND ON GARBLED CIRCUITS

We give a brief and self-contained overview of standard garbled circuit constructions and optimizations. Readers familiar with garbled circuits may safely skip this section.

3.1 Yao's Classical Construction

In (the modern interpretation of) Yao's scheme [20], the garbler chooses two random *wire labels* W^0 and W^1 for each wire, where W^x is a bit-string encoding the truth value x. Then for each (boolean, fan-in 2) gate, the garbler generates a *garbled gate* consisting of 4 ciphertexts. If a gate has input wires indexed i and j, output wire index k, and functionality $g : \{0,1\}^2 \to \{0,1\}$, then the 4 ciphertexts are:

$$E_{W_i^0, W_j^0}(W_k^{g(0,0)}),\ E_{W_i^0, W_j^1}(W_k^{g(0,1)}),$$
$$E_{W_i^1, W_j^0}(W_k^{g(1,0)}),\ E_{W_i^1, W_j^1}(W_k^{g(1,1)})$$

where $E_k(m)$ is a suitable encryption scheme. Intuitively, the wire labels encoding (a, b) on the input wires are used as keys to encrypt the wire label encoding $g(a, b)$ on the output wire.

An evaluator evaluates the garbled circuit by holding one wire label per wire. Hence, she can decrypt only one ciphertext per gate, and learn one label for the output wire.

We point out that Yao's classical scheme can be trivially extended to support garbling of non-boolean circuits of any fan-in. In particular, for m-ary wires we choose m different wire labels on each wire. Then to garble a fan-in-k gate, we include m^k ciphertexts (one for each entry in the gate's truth table). While this trivial extension is obviously not efficient, we will rely on this observation for unary gates (fan-in 1) as a component in some of our gadgets.

3.2 Standard Elementary Optimizations

Arranging the 4 ciphertexts in order of truth values leaks information, so in the classical scheme these ciphertexts are given in random order. The evaluator performs trial decryption on each one, and we use an encryption scheme that makes it obvious when decrypting the "correct" ciphertext.

A better approach is to use the **point-and-permute** optimization of Beaver, Micali and Rogaway [5]. A random "**color bit**" is appended to each wire label, so that W_i^0 and W_i^1 have opposite color bits. Because the association between colors and truth values is random, it is safe to arrange the 4 ciphertexts according to color bits of the input wire labels (i.e., the first ciphertext should be the one that uses two keys having both 0 color bits, regardless of what truth value they represent).

Using point-and-permute, the evaluator need only decrypt one ciphertext — the one indicated by the color bits of the input wire labels. Hence, it is possible to use a simple one-time encryption scheme $E_{k_1,k_2}(m) = H(g; k_1 \| k_2) \oplus m$, where g is the index of the gate and H is a key derivation function or random oracle.

The number of ciphertexts can also be reduced from 4 to 3 by the following **row reduction** trick of Naor, Pinkas and Sumner [16]. Instead of choosing the output wire labels W_k^0 and W_k^1 at random, we choose one of them so that the first of the 4 ciphertexts is always the all-zeroes string. For example, if the first ciphertext for a gate is $E_{W_i^0, W_j^0}(W_k^1) = H(g; W_i^0 \| W_j^0) \oplus W_k^1$, then we choose W_k^1 not uniformly, but as $H(g; W_i^0 \| W_j^0)$. Since the first ciphertext is all zeroes, it need not be sent, and only 3 ciphertexts are required. Note that this method constrains the selection of one of W_k^0, W_k^1. A more sophisticated approach, constraining the selection of both labels, can further reduce the garbled gate to 2 ciphertexts, as shown by Pinkas et al. [18] (see also a simpler construction in [11]).

3.3 Free-XOR

Arguably the optimization to garbled circuits with the highest practical impact is the Free-XOR optimization of Kolesnikov & Schneider [14]. When using Free-XOR, wire labels are chosen so that $W_i^0 \oplus W_i^1 = \Delta$, where $\Delta \in \{0,1\}^\lambda$ is a secret value that is common to the entire circuit. In other words, the wire label that encodes x can be written as $W_i^x = W_i^0 \oplus x\Delta$. The result of this choice of wire labels is that $(W_i^x \oplus W_j^y) = (W_i^0 \oplus W_j^0) \oplus (x \oplus y)\Delta$; that is, simply XORing two wire labels that encode x & y results in a wire label encoding $x \oplus y$, if we take $W_k^0 = W_i^0 \oplus W_j^0$ to be the "false" wire label of the output wire. As a consequence, garbled values can be XOR'ed without any cryptographic operations by the evaluator or any garbled-gate information in the garbled circuit.

To support point-and-permute, consider appending an additional bit to both Δ and each wire label to represent the color bits. Suppose this last bit of Δ is 1, and we extend the relation $W_i^0 \oplus W_i^1 = \Delta$ to include these color bits, then on every wire the two wire labels W_i^0 and W_i^1 will still have *opposite and random* color bits. This is all that is required for point-and-permute.

The Free-XOR optimization is easily compatible with the row reduction trick above, allowing for 3 ciphertexts per AND gate and 0 per XOR. It is not compatible with the 2-row reduction of [18], since that technique constrains the selection of both wire labels, and thus does not allow to maintain the required Δ relation. However, the half-gates technique of [21] provides a way to achieve a 2 ciphertext AND that is compatible with free-XOR.

4. OUR BUILDING BLOCKS

4.1 Generalizing Free-XOR & Point-Permute

Our starting point is a natural generalization of Free-XOR which permits free addition mod m for any fixed m (collapsing to Free-XOR when $m = 2$). This generalization was also used by [15]. In this section, and throughout the rest of this paper, we interpret wire labels as vectors of $\mathbb{Z}_m$-elements. We use bold-face symbols to denote wire labels ($\mathbb{Z}_m$-vectors).

Each wire carries a logical value in $\mathbb{Z}_m$. The wire label encoding $x \in \mathbb{Z}_m$ is $\boldsymbol{W}_i^x = \boldsymbol{W}_i^0 + x\boldsymbol{\Delta}_m$, where now addition refers to component-wise addition in $\mathbb{Z}_m$. The value $\boldsymbol{\Delta}_m$ is a random vector of $\mathbb{Z}_m$-elements that is global to the circuit. Our construction will involve wires with different moduli, and we use a different $\boldsymbol{\Delta}_m$ for each modulus m (but all wires with associated modulus m will share the same $\boldsymbol{\Delta}_m$).

Like Free-XOR, this generalization supports several computations on garbled values for free:

- **Addition mod** m**:** We can add garbled values mod m

for free, since $(\boldsymbol{W}_i^x + \boldsymbol{W}_j^y) = (\boldsymbol{W}_i^0 + \boldsymbol{W}_j^0) + (x+y)\boldsymbol{\Delta}_m$ (where additions are mod m).

- **Multiplication by a public/constant c mod m,** provided that c is coprime to m: This becomes nontrivial only when generalizing beyond $m = 2$. Indeed, let $c \in \mathbb{Z}_m$ be a known constant, then $c\boldsymbol{W}_i^x = c(\boldsymbol{W}_i^0 + x\boldsymbol{\Delta}_m) = c\boldsymbol{W}_i^0 + (cx)\boldsymbol{\Delta}_m$, where the operations are component-wise mod m. We require c to be coprime to m for technical reasons in the security proof — intuitively, multiplying $\boldsymbol{W}_i^0$ by c preserves its uniform distribution.

We can similarly generalize the point-permute optimization. As described in Section 3, imagine appending an extra **color "digit"** (now a $\mathbb{Z}_m$ element rather than a single bit) to each wire label and a $1 \in \mathbb{Z}_m$ digit to $\boldsymbol{\Delta}_m$ (any other digit value that is coprime to m would also work). Let $\tau_m(\boldsymbol{W})$ denote the last component of such a wire label, then we have

$$\tau_m(\boldsymbol{W}_j^x) = \tau_m(\boldsymbol{W}_j^0) + x \cdot \tau_m(\boldsymbol{\Delta}_m) = \tau_m(\boldsymbol{W}_j^0) + x.$$

In other words the m possible wire labels for this wire are assigned a *random cyclic shift* of the set $\mathbb{Z}_m$ of possible colors, with the cyclic shift amount being determined by the random value $\tau_m(\boldsymbol{W}_j^0)$. This turns out to be sufficient to prove security of this generalization of point-permute. In short, seeing the color of a single label $\boldsymbol{W}_i^x$ on a wire leaks no information about its truth value x.

Length of the wire labels.

Let λ denote the global security parameter, and define $\lambda_m = \lceil \lambda / \log m \rceil$. Suppose wire labels mod m have λ_m components, each from $\mathbb{Z}_m$. Then the length of wire labels when written as strings is at least λ bits, which is important for security.

When accounting for the point-permute optimization, we add an extra component to wire labels. In the end, wire labels mod m are elements of $\mathbb{Z}_m^{\lambda_m+1}$. Their length in bits is therefore at most $\lambda + 2\log m$ bits.

Instead of starting with λ-bit wire labels and *adding* a few bits for the "color digit," one could alternatively think of all wire labels having exactly λ bits (including color digits), but the color digits slightly *degrading the effective security parameter* by $\log m$ bits. For instance, in practice one would typically use AES-128 to implement garbled circuits. Then it is convenient if all wire labels are exactly 128 bits. One would require AES to provide security when the last $\log m$ bits of the key are known. In our final constructions, we never suggest a modulus m larger than, say, 256. So in practice our construction would degrade the AES security by only 8 bits. We note that all implementations of garbled circuits take this approach, but for the case of simple point-permute where the degradation of security is only 1 bit.

4.2 Garbling Mixed Moduli Simple Circuits

Next, we construct a simple and practical garbling scheme for a special subclass of circuits. In the following sections we will show how to efficiently express more general computations with this subclass.

A **mixed-modulus simple circuit** is a circuit (directed acyclic graph) where each wire has an associated modulus (i.e., the wire can carry values from $\mathbb{Z}_m$ for its preferred m). In addition to standard input/output gates, the circuit is allowed to have the following types of internal gates:

- An addition-mod-m gate (of unbounded fan-in) is allowed if all the wires touching the gate are mod-m wires.
- A unary gate that multiplies by a constant mod m is allowed if both the input and output wire are mod m, and if the constant is coprime to m.
- Arbitrary *unary* "projection" gates: if the input wire is mod-m and the output wire is mod-n, then the gate can apply an arbitrary function $\varphi : \mathbb{Z}_m \to \mathbb{Z}_n$. We refer to a gate with this functionality as Proj_φ.

In our construction, the first two types of gates are "free", and the third type uses at most $m-1$ ciphertexts (the "-1" follows from the row-reduction technique). The main idea follows the discussion above. For each modulus m that appears in the circuit, we choose a global value $\boldsymbol{\Delta}_m$ (interpreted as a vector in $\mathbb{Z}_m^{\lambda_m+1}$), and use the generalized free-XOR method of choosing wire labels. That is, $\boldsymbol{W}_i^0$ is random and $\boldsymbol{W}_i^x = \boldsymbol{W}_i^0 + x\boldsymbol{\Delta}_m$. Addition mod m and multiplication by a (coprime) constant can be garbled for free, as described above.

A projection gate Proj_φ can be garbled using m ciphertexts, where each wire label $\boldsymbol{W}_i^x$ is used to encrypt the associated payload $\boldsymbol{W}_j^{\varphi(x)}$. These m ciphertexts can be ordered by the color digits of the input wire labels, namely by $x+\tau$, where $\tau = \tau_m(\boldsymbol{W}_i^0)$. Thus, the garbled gate consists of the following m ciphertexts (one for each x):

$$\begin{aligned}\hat{\boldsymbol{G}}_{x+\tau} &= H(g, \boldsymbol{W}_i^0 + x\boldsymbol{\Delta}_m) + \boldsymbol{W}_j^0 + \varphi(x)\boldsymbol{\Delta}_n \\ &= H(g, \boldsymbol{W}_i^x) + \boldsymbol{W}_j^{\varphi(x)},\end{aligned}$$

where H is a hash/key derivation function (see below) and g is the gate index (used as a tweak in the hash function). The outer vector addition (as well as the value of $\varphi(x)$) is over $\mathbb{Z}_n$, while the inner vector addition (as well as the values of x, τ) are over $\mathbb{Z}_m$. The evaluator will decrypt only one of these ciphertexts, specifically the one whose subscript is the color digit of the input wire label she holds.

Using the row reduction trick described above [16], we can remove one of the ciphertexts (getting $m-1$) by setting $\hat{\boldsymbol{G}}_0 = \boldsymbol{0}$. To do this, choose $\boldsymbol{W}_j^0 = -H(g, \boldsymbol{W}_i^{-\tau}) - \varphi(-\tau)\boldsymbol{\Delta}_n$.

We could also remove 2 ciphertexts (getting $m-2$) using the approach of [18, 11], but at the cost of having the output wire labels no longer satisfy the $\boldsymbol{\Delta}_m$-correlation property.

The scheme is presented formally in Figure 1.

4.3 Security

Here we prove security of the basic garbling scheme we just presented. We point out that the security of our constructions in later sections will follow from the fact that we express the desired functionality as a mixed-modulus simple circuit, to be garbled using our basic scheme.

The original free-XOR construction of [14] was proven secure in the random oracle model. Choi et al. [9] later proved that the security depends only on a specific property of the oracle that they called *circular correlation-robustness*. Let H be a hash function and define an oracle $\mathcal{O}_\Delta^H(g, X, Y, a, b, c) = H(g, X \oplus a\Delta, Y \oplus b\Delta) \oplus c\Delta$. Here X, Y, Δ are string of length λ, while a, b, c are bits, and g is an arbitrary string.

Then H is circular-correlation-robust if the outputs of this oracle appear random, to adversaries that query on distinct g values with a, b not both zero, when Δ is chosen uniformly.

We use the corresponding generalization to multiple $\boldsymbol{\Delta}$

```
procedure $\textsc{Gb}(1^\lambda, f)$
    for $m \in f$.wiredomains do
        $\lambda_m \leftarrow \left\lceil \frac{\lambda}{\log m} \right\rceil$
        $\boldsymbol{\Delta}_m \overset{u}{\leftarrow} \mathbb{Z}_m^{\lambda_m} \| 1$
    for $i \in f$.inputs do
        $\mathbb{Z}_m \leftarrow i$.domain
        $\boldsymbol{W}_i^0 \overset{u}{\leftarrow} \mathbb{Z}_m^{\lambda_m+1}$
    $\hat{e} \leftarrow (\boldsymbol{W}_1^0, \ldots, \boldsymbol{W}_q^0, \boldsymbol{\Delta}_{m_1}, \ldots, \boldsymbol{\Delta}_{m_r})$
    for $g \leftarrow f.\mathsf{gates}_{\mathsf{topo}}$ do
        $\boldsymbol{a} \leftarrow g$.inputs
        $\mathbb{Z}_m \leftarrow g$.domain
        if $g$ is $\mathsf{Add}_m$-gate then
            $\boldsymbol{W}_g^0 \leftarrow \sum_{i=1}^{b} \boldsymbol{W}_{a_i}^0$
        else if $g$ is $\mathsf{Mult}_c$-gate then
            $\boldsymbol{W}_g^0 \leftarrow c \cdot \boldsymbol{W}_{a_1}^0$
        else if $g$ is $\mathsf{Proj}_\varphi$ gate then
            $\mathbb{Z}_n \leftarrow g$.range
            $\tau \leftarrow \tau(\boldsymbol{W}_{a_1}^0)$
            $\boldsymbol{W}_g^0 \leftarrow -H(g, \boldsymbol{W}_{a_1}^0 - \tau\boldsymbol{\Delta}_m) - \varphi(-\tau)\boldsymbol{\Delta}_n$
            for $x \in \mathbb{Z}_m$ do
                $\hat{\boldsymbol{G}}_{x+\tau}^g \leftarrow H\left(g, \boldsymbol{W}_{a_1}^0 + x \cdot \boldsymbol{\Delta}_m\right) + \boldsymbol{W}_g^0 + \varphi(x)\boldsymbol{\Delta}_n$
            $\hat{\boldsymbol{G}}^g \leftarrow (\hat{\boldsymbol{G}}_1^g, \ldots, \hat{\boldsymbol{G}}_{k-1}^g)$
    $\hat{F} \leftarrow (\hat{G}^1, \ldots, \hat{G}^{|\mathsf{gates.proj}|})$
    for $i \in f$.outputs do
        $\mathbb{Z}_m \leftarrow i$.domain,
        for $k \in \mathbb{Z}_m$ do
            $d_i^k \leftarrow H(\mathsf{out}\|i\|k, \boldsymbol{W}_i^0 + k\boldsymbol{\Delta}_m)$
        $d_i \leftarrow (d_i^0, \ldots, d_i^{m-1})$
    $\hat{d} \leftarrow (d_1, \ldots, d_\ell)$
    return $(\hat{F}, \hat{e}, \hat{d})$

procedure $\textsc{En}(\hat{e}, \hat{x})$
    for $x_i \in \hat{x}$ do
        $\mathbb{Z}_m \leftarrow i$.domain
        $X_i \leftarrow \boldsymbol{W}_i^0 + x_i \cdot \boldsymbol{\Delta}_m$
    return $\hat{X} \leftarrow (X_1, \ldots, X_q)$
procedure $\textsc{De}(\hat{d}, \hat{Y})$
    for $d_i \in \hat{d}$ do
        $\mathbb{Z}_m \leftarrow i$.domain
        $(h_1, \ldots, h_m) \leftarrow d_i$
        for $k \in \mathbb{Z}_m$ do
            if $H(\mathsf{out}\|i\|k, Y_i) = h_k$ then
                $y_i \leftarrow k$
        if $y_i$ unassigned then return $\perp$
    return $\hat{y} \leftarrow (y_1, \ldots, y_\ell)$
procedure $\textsc{Ev}(\hat{F}, \hat{\boldsymbol{X}})$
    for $i \in f$.inputs do
        $\boldsymbol{W}_i \leftarrow X_i$
    for $g \leftarrow f.\mathsf{gates}_{\mathsf{topo}}$ do
        $\boldsymbol{a} \leftarrow g$.inputs
        $\mathbb{Z}_m \leftarrow g$.domain
        if $g$ is $\mathsf{Add}_m$-gate then
            $\boldsymbol{W}_g \leftarrow \sum_{i=1}^{b} \boldsymbol{W}_{a_i}$
        else if $g$ is $\mathsf{Mult}_c$-gate then
            $\boldsymbol{W}_g \leftarrow c \cdot \boldsymbol{W}_{a_1}$
        else if $g$ is $\mathsf{Proj}$ gate then
            $\hat{\tau} \leftarrow \tau(\boldsymbol{W}_{a_1})$
            $\boldsymbol{W}_g \leftarrow \hat{\boldsymbol{G}}_{\hat{\tau}}^g - H(g, \boldsymbol{W}_{a_1})$
    for $i \in f$.outputs do
        $Y_i \leftarrow \boldsymbol{W}_i$
    return $\hat{Y} \leftarrow (Y_1, \ldots, Y_\ell)$
```

Figure 1: Our garbling scheme for mixed moduli simple circuits

values (one for each modulus). We define the following oracle:

$$\mathcal{O}^H_{\boldsymbol{\Delta}_{m_1},\ldots,\boldsymbol{\Delta}_{m_n}}(g, i, j, \boldsymbol{X}, a, c) = H(g, \boldsymbol{X} + a\boldsymbol{\Delta}_{m_i}) + c\boldsymbol{\Delta}_{m_j}$$

We assume that the inner addition is mod m_i and the outer addition is mod m_j. The parameter a is interpreted as a value mod m_i and c as a value mod m_j. We also assume that the inputs and outputs of H can be interpreted as vectors of $\mathbb{Z}_m$-elements for appropriate m whenever needed.

Definition 1. *We say that H is* ***mixed-modulus circular correlation robust*** *if for all polynomial time adversaries that query their oracle on distinct g values and $a \neq 0$, the oracle $\mathcal{O}^H_{\boldsymbol{\Delta}_{m_1},\ldots,\boldsymbol{\Delta}_{m_n}}$ (with $\boldsymbol{\Delta}_{m_i}$ values chosen uniformly) is indistinguishable from a random function.*

As usual, this assumption can be abstracted away if one is willing to use the random oracle model.

Theorem 1. *The scheme in Figure 1 satisfies the* prv.sim, obv.sim, *and* aut *security definitions (Section 2.1), when the function H is mixed-modulus circular correlation robust (Definition 1).*

Proof Sketch. The proofs follow very closely the security proof for Free-XOR in [9] and a similar proof in [13].

We start out with (F, X, d) generated via $(F, e, d) \leftarrow \mathsf{Gb}(f)$ and $X \leftarrow \mathsf{En}(e, x)$. The main idea is to first perform a conceptual shift to this hybrid. The garbling procedure is normally written from the garbler's point of view, maintaining the "false" wire labels $\boldsymbol{W}$ for each wire. We can instead compute for every wire i in the circuit the value v_i that is on that wire when the circuit's input is x. Note that this hybrid still requires knowing the circuit input x — only in later hybrids can we argue that x is not used. Then we can identify for each wire which wire label $\boldsymbol{W}_i^*$ will be visible to the evaluator. Finally, we replace every reference to a wire label $\boldsymbol{W}_i^z$ with $\boldsymbol{W}_i^* + (z - v_i)\boldsymbol{\Delta}_m$.

After this conceptual shift, we see that all garbled gate ciphertexts (and decoding information d) can be written in the following form:

$$H(g, \boldsymbol{W}_i^* + a\boldsymbol{\Delta}_m) + c\boldsymbol{\Delta}_{m'} + \boldsymbol{Q}$$

where $\boldsymbol{W}_i^*$ and Q are known to the evaluator ($\boldsymbol{Q}$ is either another visible wire label, or an empty string in the case of the decoding information). Because of the shift to the evaluator's point of view, whenever $a = 0$, we also have $c = 0$. Hence all of these ciphertexts can be computed without the $\boldsymbol{\Delta}_m$ values, and only oracle access to the mixed-modulus-correlation-robustness oracle of Definition 1. Hence all ciphertexts, apart from the ones where $a = 0$ above, can be replaced with uniformly chosen values.

After doing such a replacement, we see that the v_i values (the truth values on each wire of the circuit) are no longer needed. They were used to compute a and c values in the above. The only other place there were used is to determine which cyclic shift of the decoding information d to apply on each output wire. But the v_i values for output wires are simply the circuit output $f(x)$. Hence, the final result is a simulator that depends only on $f(x)$.

For $\mathsf{obv.sim}$ security, we simply observe that in the above simulator, $f(x)$ is used only to compute d. So when d is not given, the simulator does not require $f(x)$.

For aut security, we observe that all other entries in d (apart from the ones obtainable by $\mathsf{Ev}(F, X)$) are chosen uniformly by the simulator. Hence, the probability that the evaluator guesses any other output wire label correctly is $1/2^{\lambda}$. $\square$

5. SIMPLE IMPROVEMENTS TO GARBLED SYMMETRIC BOOLEAN GATES

In the previous section we saw a gadget extending Free-XOR to allow free addition-mod-m for $m > 2$ (assuming all wires have mod-m labels). Perhaps surprisingly, this turns out to be useful not only for (arithmetic) modular addition, but even for *boolean* computations. For example, consider an AND gate $x_1 \wedge \cdots \wedge x_b$ with fan-in b. We observe that this gate can be expressed as $\chi_b(\sum_i x_i)$ where $\chi_b(n) = 1$ if $n = b$ and $\chi_b(n) = 0$ otherwise.[1] In the terminology above, we have expressed an AND as a projection of a sum.

If the input wires to this AND gate have mod-$(b + 1)$ wire labels, then the addition is free (and will not wrap around). So the total cost of the AND gate is that of the $\mathbb{Z}_{b+1}$ projection gate, which is b ciphertexts.

By comparison, the best-known way to garble a fan-in-b AND gate using existing boolean techniques is to make a tree of $b - 1$ binary AND gates and use the construction of [21], costing 2 ciphertexts per AND gate for a total of $2b - 2$ ciphertexts. Thus, our basic building blocks already give a constant size improvement (from $2b - 2$ to b).

It is also clear that any t-out-of-b threshold gate can be garbled at the same cost (b ciphertexts) by substituting χ_b for an appropriate projection function. Most generally, we can garble at a cost of b ciphertexts any *symmetric* fan-in-b Boolean gate (a symmetric gate is one whose output depends only on the Hamming weight of its inputs).

While these improvements are already significant, we later show how to do *exponentially better* for threshold gates in the case of very high fan-in.

Circumventing a lower bound.

For $b = 2$, this gives us a boolean AND gate at a cost of 2 ciphertexts, which matches the half-gates construction of [21]. The half-gates construction is compatible with Free-XOR (in our terminology, it uses wire labels that are $\mathbf{\Delta}_2$-correlated), while ours requires input wire labels to be mod-3 (the output labels in our construction can use any modulus).

Looking more closely at our construction, we express an AND as a projection of a sum. As explained above, the nominal cost of the projection is $b+1$ ciphertexts, which we reduce to b using the simple row-reduction idea. However, as we pointed out, we can further reduce it to $b-1$ using the more involved row reduction technique of [18], at the price of having *both* output wire labels constrained, and thus not satisfying the required $\mathbf{\Delta}$ correlation.

Consider now a *single* boolean AND gate (thus, correlation of the output wire labels is unimportant). We can garble this AND gate with just a *single* ciphertext. This construction is not very useful in itself, since it does not compose even with itself (although it may give some savings for boolean circuits of a certain structure, that doesn't require all wires to have a $\mathbf{\Delta}$ correlation). However, it is of theoretical interest, since it circumvents a lower bound. Specifically, [21] show not only a 2-ciphertext upper bound for garbled AND gates, but also that the cost of 2 ciphertexts is *optimal* in a model they define, capturing all previously known garbling schemes.

The reason our construction is able to circumvent the lower bound is that their model includes the implicit assumption that there is a *single color bit* per wire label. Our construction breaks the bound by using a generalized color digit from $\mathbb{Z}_3$ instead of from $\mathbb{Z}_2$. This underscores the power of our point-permute generalization, whose main technical difference from Free-XOR is the use of generalized color digits for wire labels.

6. GARBLING ARITHMETIC GATES OVER LARGE MODULI

Our basic construction (Figure 1) can in principle be used to represent any boolean circuit. However, while it supports free addition mod m, the cost of the non-free projection gates is linear in the modulus m, which is clearly impractical for large values of m. We are interested in large values of m both for boolean circuits with massive fan-in (say $m \sim 1000$), and even more so for arithmetic circuits (with, e.g., $m \sim 2^{64}$). Indeed, one of the most natural uses of arithmetic circuits is to carry out computations over the *integers*, by choosing a modulus that is larger than any intermediate value in the computation. In this section we suggest new representations and gadgets that allow for radically more efficient garbling over large moduli. Despite the arithmetic nature of the gadgets, we will show applications for both boolean and arithmetic circuits.

Rather than choosing a prime modulus, we choose a composite **primorial modulus** $P_k = 2 \cdot 3 \cdots p_k$, the product of the first k primes. Then values can be represented in terms of smaller moduli $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \mathbb{Z}_{p_k}$ using the Chinese Remainder Theorem (CRT). Define the *residue representation* of x as:

$$[\![x]\!]_{\mathsf{crt}} := ([x]_2, [x]_3, \ldots, [x]_{p_k})$$

In terms of mixed-modulus circuits, we represent $[\![x]\!]_{\mathsf{crt}}$ with a bundle of wires having wire-moduli 2, 3, and so on.

Bounds on parameter sizes.

Suppose we wish to support arithmetic over the integers, and we require a modulus at least as large as some bound Z on the possible intermediate values in the computation. We have the following facts from number theory:

LEMMA 1. *Let k be the smallest integer such that $\prod_{i \le k} p_i > Z$ (where p_i denotes the ith prime). Then, asymptotically:*

$$k = O(\log Z / \log\log Z); \qquad p_k = O(k \log k) = \Theta(\log Z);$$

$$\sum_{i=1}^{k} p_i = O(\log^2 Z / \log\log Z); \qquad \sum_{i=1}^{k} \lceil \log_2 p_i \rceil = \Theta(\log Z).$$

[1]As previously mentioned, this can be viewed as generalizing the representation of fan-in-2 NAND gates in [17].

Concrete example parameters are given below:

Z	k	p_k	$\sum_{i=1}^{k} p_i$	$\sum_{i=1}^{k} \lceil \log_2 p_i \rceil$
2^8	5	11	28	13
2^{16}	7	17	58	22
2^{32}	10	29	129	37
2^{64}	16	53	381	72
2^{128}	27	103	1264	147

Our constructions in this section have cost either $O(\sum_i p_i) = O(\log^2 Z)$ or $O(k \sum_i p_i) = O(\log^3 Z)$ ciphertexts (each of length λ bits). To represent a single value $[\![x]\!]_{\mathsf{crt}}$ requires a bundle of $k = O(\log Z)$ wires and hence that many wire labels (each λ bits long). This is comparable to representing the number x in binary for a standard boolean garbled circuit, which would also require $\log Z$ wires/labels.

6.1 Basic Arithmetic

Addition & multiplication by a constant.

To add $[\![x]\!]_{\mathsf{crt}}$ and $[\![y]\!]_{\mathsf{crt}}$ (modulo the primorial composite), one simply adds their CRT residues component-wise. The cost is free in our garbling scheme.

Similarly, to multiply $[\![x]\!]_{\mathsf{crt}}$ by a constant c, one simply multiplies by c within each individual CRT residue. For residues p where c is coprime to p, the operation is free in our garbling scheme. For residues where c is not coprime to p, this only happens when $c \equiv 0 \pmod{p}$ by our choice of prime CRT moduli. Hence the result of the multiplication-by-c will be zero (and since c is public, it is known that the result will be zero). For these CRT residues, we instead use a *global* wire label representing $[0]_p$ (this is common to the whole circuit and sent as part of the garbling procedure - independent of the number of gates). Overall the cost of multiplying by *any* constant c is free.

Exponentiation.

To raise $[\![x]\!]_{\mathsf{crt}}$ to the power n (modulo the primorial composite), for a public constant n, it again suffices to do so within each CRT residue. This can be done with a simple projection gate $\phi_p(x) = [x^n]_p$ within each modulus p. The cost of a mod-p projection gate is $p-1$ ciphertexts, so the total cost of exponentiation is $\sum_i (p_i - 1)$ ciphertexts. For the use case of arithmetic over integers bounded by Z, the cost is $O(\log^2 Z)$ ciphertexts.

This construction works also when the exponent is secret but known only to the circuit garbler. This is because our projection gates hide the actual choice of projection function.

Remainder mod p_i.

Suppose we wish to transform $[\![x]\!]_{\mathsf{crt}}$ into $[\![x \bmod p_i]\!]_{\mathsf{crt}}$ for some p_i that is among the primes in our CRT representation. Note that the value $[x]_{p_i}$ already exists within $[\![x]\!]_{\mathsf{crt}}$. All that is needed is to "copy" that value to the other residues. This is achieved by using an identity projection gate $[x]_{p_i} \mapsto [x]_{p_j}$ for each other prime p_j. The cost is $p_i - 1$ ciphertexts for each projection, for a total of $(k-1)(p_i-1)$. As a special case, when the remainder is mod 2, the total cost is $k-1$ ciphertexts.

General Multiplication.

To multiply two (private) values $[\![x]\!]_{\mathsf{crt}}$ and $[\![y]\!]_{\mathsf{crt}}$, we again simply multiply their residues component-wise.

A naïve way to multiply two values mod p would be to generate a truth table of all p^2 combinations, resulting in p^2 ciphertexts. However, we can take advantage of the fact that each p is prime and instead garble a multiplication with only $O(p)$ ciphertexts (with small constants). For example, Malkin, Pastro & shelat [15] suggest such a way based on a generalization of [21] (who constructed a low-cost multiplication gate over $\mathbb{Z}_2$).

Here we suggest the following alternative approach (which conveniently scales well with high fan-in). First, let us blatantly ignore the case where $0 \in \{x, y\}$. Doing so, we may write

$$x \cdot y = g^{\mathrm{dlog}_g(x) + \mathrm{dlog}_g(y)}$$

where g is any primitive root mod p. The addition in the exponent is mod $p-1$.

Hence, our approach for multiplication is to first use a projection gate to map $\mathbb{Z}_p$ values to their discrete logs (in $\mathbb{Z}_{p-1}$). The cost is $p-1$ ciphertexts for each input. While the discrete logarithm problem is of course difficult in general, we are only asking for a lookup table of discrete logarithms to be precomputed for very small p (e.g., $p \leq 103$ for all of our proposed instantiations). The discrete logs can then be added mod $p-1$ for free, and finally the result promoted to the final product using a projection gate $z \mapsto g^z \pmod{p}$. The final projection gate uses $p-2$ ciphertexts, but to handle zeroes we will use a slightly different projection.

To handle the case where one of the multiplicands may be zero mod p, we write:

$$x \cdot y = \begin{cases} 0 & \text{if } \mathsf{OR}(x=0, y=0) \\ g^{\mathrm{dlog}_g(x) + \mathrm{dlog}_g(y)} & \text{else} \end{cases}$$

To compute the comparisons $x = 0$ and $y = 0$ requires $2p-2$ total ciphertexts. We arrange for these comparisons to have output wire with modulus 2. That way, we can compute their OR for only 2 ciphertexts. The two dlog projections require $2p-2$ total ciphertexts.

The final operation is

$$f(z, b) = \begin{cases} g^z & \text{if } b = 0 \\ 0 & \text{else} \end{cases}.$$

We can garble this operation with the standard Yao truth-table approach, using $2(p-1)-1$ ciphertexts if we use a row-reduction trick. The total cost of the entire multiplication mod p is $6p-5$ ciphertexts. For the entire CRT representation, the cost of multiplication is $\sum_i (6p_i - 5)$ ciphertexts.

For the use case of arithmetic over integers bounded by Z, the cost is $O(\log^2 Z)$ ciphertexts.

In the special case where one of the multiplicands y is a (secret) value known to the garbler, the cost can be reduced. The idea is to garble a projection gate $[x]_p \mapsto [xy]_p$ within each residue. The total cost is $\sum_i (p_i - 1)$ ciphertexts. The asymptotic cost is the same as a general multiplication, but the concrete cost is roughly 6 times better.

6.2 Equality Tests & Exact Weighted Threshold

To test whether $[\![x]\!]_{\mathsf{crt}} = [\![y]\!]_{\mathsf{crt}}$, we observe that

$$[\![x]\!]_{\mathsf{crt}} = [\![y]\!]_{\mathsf{crt}} \iff \mathsf{AND}([x-y]_{p_1} = 0, \ldots, [x-y]_{p_k} = 0).$$

The subtractions mod each p_i are free. We can test whether $z \equiv 0 \pmod p$ using a simple projection gate. The cost of such a projection gate is $p-1$ ciphertexts. Note that the output of this projection gate can be any modulus, and we choose the output modulus to be $k+1$ where k is the number of primes in the CRT representation. That way, we can garble the final AND gate using k ciphertexts using the construction described in Section 5.

The total cost to garble this equality test is therefore $k + \sum_i (p_i - 1) = \sum_i p_i$ ciphertexts. However, this assumes an output of a single mod-2 wire. To use the output of the equality test in other gadgets, the output would have to be represented as $[\![x]\!]_{\mathsf{crt}}$ — that is, as a bundle of wires with distinct moduli. This simply requires an identity projection $\mathbb{Z}_2 \to \mathbb{Z}_{p_i}$ for each prime p_i. The cost is 1 ciphertext per projection, bringing the total cost of a *composable* equality test to $k + \sum_i p_i = \sum_i (p_i + 1)$ ciphertexts, where k is the number of CRT moduli.

For the use case of arithmetic over integers bounded by Z, the cost is $O(\log^2 Z)$ ciphertexts.

Application to exact-weighted-threshold gates.

An *exact weighted threshold* gate refers to the following kind of computation:

$$\mathsf{Th}_{t,c_1,\ldots,c_b}(x_1,\ldots,x_b) = \begin{cases} 1 & \text{if } t = \sum_i c_i x_i \\ 0 & \text{otherwise} \end{cases}$$

For example, an AND gate is an exact weighted threshold, corresponding to the case where $c_1 = \cdots = c_b = 1$ and $t = b$ and where the x_i's are bits.

We can garble these kinds of gates efficiently in our scheme when the inputs are in our CRT residue encoding. The computation consists of a weighted sum followed by equality test. The equality test requires $\sum_i p_i$ ciphertexts. When the weights c_i are public, the weighted sum is free, so there is no additional cost.

In the case of boolean circuits (where x_i's and c_i's are bits), this construction gives exponential improvements over the state of the art for AND/OR gates. To use this construction in a boolean circuit, the inputs must still be CRT-encoded, and the CRT encoding must be spacious enough to not overflow during the addition step. The maximum possible sum is equal to the fan-in b, hence we must have k primes where $\prod_{i=1}^{k} p_i > b$. Compared to the boolean case, each bit will have k wire labels rather than just one, but the cost of the AND/OR gate will be asymptotically $O(\log^2 b)$ ciphertexts rather than $2b-2$ ciphertexts (via a tree of binary AND gates).

6.3 Comparisons & Weighted Threshold Gates

While the CRT representation is effective for arithmetic operations, it does not lend itself to simple comparisons. In this section we discuss how to compare $[\![x]\!]_{\mathsf{crt}}$ and $[\![y]\!]_{\mathsf{crt}}$. Our approach has several steps.

Our high-level idea is to convert the CRT representation into a **positional** number system. Specifically, we use a **primorial mixed-radix (PMR)** system. This number system is defined as $[\![x]\!]_{\mathsf{pmr}} := (d_k, \ldots, d_1) \in \mathbb{Z}_{p_k} \times \cdots \times \mathbb{Z}_{p_1}$, where:

$$d_i = \left\lfloor \frac{x}{p_1 p_2 \cdots p_{i-1}} \right\rfloor \pmod{p_i}$$

Whereas binary has a 1s-digit, 2s-digit, 4s-digit, 8s-digit, and more generally a 2^i-digit, PMR has a 1s-digit, 2s-digit, 6s-digit, 30s-digit, and more generally a $\prod_{j<i} p_j$ digit.[2]

Once a number is converted into PMR form, a comparison can be done easily, as we show.

First steps.

We will first show how to efficiently compute $\Big[\lfloor x/p \rfloor\Big]_q$, given $[x]_p, [x]_q$ and using the operations we have previously discussed (namely, free modular additions and projections). A running example corresponding to $p = 3, q = 5$ is given in Figure 2.

The idea is to consider the function $\delta(x) = [x]_p - [x]_q$, where we interpret the terms $[x]_p$ and $[x]_q$ as integers (from $\{0,\ldots,p-1\}$ and $\{0,\ldots,q-1\}$, respectively), and the subtraction also over the integers. δ has the property that it is piecewise constant, with a constant "run" ending each time x is a multiple of p or of q. In Figure 2, the "runs" are shown divided by vertical lines. In particular, if $\lfloor x/p \rfloor \neq \lfloor x'/p \rfloor$, then x and x' will be in different "runs" of δ.

With p and q relatively prime, there are $p+q-1$ runs. Furthermore, each run gives a distinct output of δ mod $p+q-1$. In other words, if $\lfloor x/p \rfloor \neq \lfloor x'/p \rfloor$ then $\delta(x) \not\equiv \delta(x') \pmod{p+q-1}$. This implies that we can write

$$\Big[\lfloor x/p \rfloor\Big]_q = \varphi\left(\big[[x]_p - [x]_q\big]_{p+q-1}\right)$$

for a suitable projection φ. For the example in Figure 2, we can obtain $[\lfloor x/3 \rfloor]_5$ as $\varphi([[x]_3 - [x]_5]_7)$ where φ is the projection $0 \mapsto 0; 1 \mapsto 3; 2 \mapsto 1; 3 \mapsto 3; 4 \mapsto 1; 5 \mapsto 4; 6 \mapsto 2$.

The cost to compute this in a garbled circuit is the cost of Proj_φ ($p+q-2$ ciphertexts) plus the cost to project $[x]_p$ and $[x]_q$ to $\mathbb{Z}_{p+q-1}$ (another $p+q-2$ ciphertexts). The subtraction mod $p+q-1$ is free. The total cost is $2p+2q-4$ ciphertexts.

Full conversion.

Now we show how to use the previous gadget to convert $[\![x]\!]_{\mathsf{crt}}$ to $[\![x]\!]_{\mathsf{pmr}}$. Define:

$$x_{i,j} := \left[\left\lfloor \frac{x}{p_1 \cdots p_i} \right\rfloor\right]_{p_j}$$

To compute $[\![x]\!]_{\mathsf{pmr}}$ it suffices to compute $x_{i,i+1}$ for all i. We proceed recursively. In the base case, $x_{0,j} = [x]_{p_j}$, which is given as part of $[\![x]\!]_{\mathsf{crt}}$. For the recursive step, we use the identity $\lfloor \lfloor a/b \rfloor / c \rfloor = \lfloor a/bc \rfloor$ and observe that

$$x_{i,j} = \left[\left\lfloor \frac{x}{p_1 \cdots p_i} \right\rfloor\right]_{p_j} = \left[\left\lfloor \frac{\left\lfloor \frac{x}{p_1 \cdots p_{i-1}} \right\rfloor}{p_i} \right\rfloor\right]_{p_j}$$

which is just the previous gadget applied to $x_{i-1,i}$ and $x_{i-1,j}$.

Inductively, the total cost to obtain $[\![x]\!]_{\mathsf{pmr}}$ is the cost to

[2]In fact, one can obtain the binary number system by setting every $p_i = 2$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$[x]_3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$[x]_5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$[x]_3 - [x]_5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2
$[[x]_3 - [x]_5]_7$	0	0	0	4	4	2	6	6	6	3	1	1	5	5	5
$[\lfloor x/3 \rfloor]_5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

Figure 2: An example of the gadget computing $[\lfloor x/p \rfloor]_q$.

apply the gadget for all pairs p_i, p_j with $i < j$:

$$\sum_{1 \le i < j \le k} (2p_i + 2p_j + 4) = 2(k-1)\sum_{i=1}^{k} p_i + 4\binom{k}{2}$$

Comparing values via PMR representation.

Suppose we wish to determine whether $[\![x]\!]_{\mathsf{crt}} < [\![y]\!]_{\mathsf{crt}}$ (note the CRT representation, not PMR). This is equivalent to the comparison $[\![x-y]\!]_{\mathsf{crt}} < 0$. Suppose our CRT representation uses one more prime modulus than is necessary. Then by assumption, all intermediate values in the circuit are at most $p_1 \cdots p_{k-1}$ in absolute value.

Then if $x - y$ is positive, the most significant digit of $[\![x-y]\!]_{\mathsf{pmr}}$ will be zero. If $x - y$ is negative, the sum will wrap around mod $p_1 \cdots p_k$ and be a number larger than $p_1 \cdots p_{k-1}$. Hence the most significant digit of $[\![x-y]\!]_{\mathsf{pmr}}$ will be nonzero.

Hence, to compare $[\![x]\!]_{\mathsf{crt}}$ to $[\![y]\!]_{\mathsf{crt}}$, we obtain $[\![x-y]\!]_{\mathsf{crt}}$ for free, convert to $[\![x-y]\!]_{\mathsf{pmr}}$ at the above cost, then do a simple projection on the most significant digit of $[\![x-y]\!]_{\mathsf{pmr}}$, costing p_k ciphertexts.

If we desire to have the result of the comparison as a CRT-encoded wire bundle, an additional k ciphertexts are required, as was the case for exact thresholds. Overall, the total cost will be

$$2(k-1)\sum_{i=1}^{k} p_i + 2k^2 - k + p_k$$

For the use case of arithmetic over integers bounded by Z, the cost of a comparison is $O(\log^3 Z)$ ciphertexts.

Application to weighted threshold gates.

A *weighted threshold gate* refers to the following kind of computation:

$$\mathsf{Th}_{t,c_1,\ldots,c_b}(x_1,\ldots,x_b) = \begin{cases} 1 & \text{if } t > \sum_i c_i x_i \\ 0 & \text{otherwise} \end{cases}$$

Since a weighted threshold gate is simply a weighted sum followed by a comparison, the cost of such a threshold gate is simply the cost of a comparison as described above. The weighted sum is free if the weights are public.

7. USING OUR SCHEME AND COMPARISON TO STANDARD GARBLING

We have now introduced all of the low-level gates that are supported compactly by our garbling scheme. In this section we discuss the costs involved when using our scheme in the context of a secure computation protocol. In particular, we focus on the "hidden" costs involved in: (1) transferring garbled inputs via oblivious transfers (2) ensuring that all of our low-level gate gadgets interoperate within a common circuit. We discuss these costs in the context of a natural application scenario, and provide a comprehensive comparison of our scheme to standard state-of-the-art garbling techniques.

Our focus is on the *communication* costs: size (number of ciphertexts) of the garbled circuits, and costs of OTs. With the use of hardware-accelerated AES instructions, current 2PC applications of garbled circuits are usually network-bound,[3] so communication cost reflects the dominant bottleneck.

7.1 Transferring Wire Labels via OT

In Yao's protocol paradigm, the circuit evaluator obtains her garbled input via oblivious transfer (OT). For boolean circuits, there are two possible wire labels per input wire, and 1-out-of-2 OT is the natural way for the evaluator to obtain the one wire label of her choice on each wire.

In practice, OT instances are realized via *OT extension protocols* [4, 12]. The main idea behind OT extension is that after performing only λ so-called *base OTs* on random strings, the parties can obtain a large number $N \gg \lambda$ of *effective* OTs using only cheap symmetric-key operations. Only the base OTs require expensive public-key operations. When discussing the cost of OT, we ignore the fixed cost of the base OTs. The marginal cost incurred by each 1-out-of-2 OT instance (for OT of λ-bit messages) is 2λ bits. However, when using OT to transfer wire labels in particular, it is possible to use an optimization similar to row-reduction for garbled circuits. One can allow the OT protocol itself to randomly choose one of the two possible wire labels. This reduces the marginal cost of each OT to just λ bits (cf. [3]).

In our scheme, we consider circuits whose wires carry values in $\mathbb{Z}_p$ for some prime p. There are p possible wire labels for each such wire. When these wires are the input wires of a circuit, we must provide a way for the receiver to obtain appropriate garbled input. The naive way to do this is using 1-out-of-p OTs, at the cost of $(p-1)\lambda$ bits. However, we suggest the following superior approach which takes advantage of the specific form of the strings.

Consider a mod-p input wire w to the circuit. Set $\ell = \lceil \log p \rceil$ and write w in binary as $w = \sum_{j=0}^{\ell-1} w_j 2^j$. Our idea is to use ℓ 1-out-of-2 OTs to obtain garbled inputs encoding the bits of w. While these w_j inputs are bits, we require them to be represented as mod-p wire labels. Then the computation $w = \sum_{j=0}^{\ell-1} w_j 2^j$ can be done for free within the circuit, as the values 2^j are public. We note that the free addition will be mod p, but by construction the result of the weighted sum is $w < p$.

[3]As a concrete example, the garbling scheme of Zahur et al. [21] reduced garbled circuit size by 33% but doubled the number of AES calls for the evaluator (compared to prior work). The changes were still a significant net improvement.

The total cost of these OTs (using the OT-row-reduction optimization described above) is only ℓ ciphertexts. For an input value $[\![x]\!]_{\mathsf{crt}}$ represented by k primes, the total cost of all OTs under this method is $\sum_i \lceil \log p_i \rceil$.

7.2 Arithmetic Circuits

Consider the setting of garbling an arithmetic circuit involving operations over the integers. Let $\lfloor \frac{Z-1}{2} \rfloor$ be an upper bound on the absolute value of intermediate values within the circuit. Then our scheme should be instantiated using CRT representations with k primes, where $\prod_{i<k} p_k > Z$ to avoid any wrap-arounds. We make a distinction between the *logical* values of the arithmetic circuit (i.e., values in $\{-\lfloor \frac{Z-1}{2} \rfloor, -Z+1, \ldots, \lfloor \frac{Z-1}{2} \rfloor\}$) and the *physical* (encoding) values in our mixed-modulus simple circuit (i.e., values mod p for some prime p).

In Figure 3 we summarize the cost of various arithmetic gates in our scheme, including concrete costs for $Z \in \{2^{16}, 2^{32}, 2^{64}\}$. The numbers in the table reflect gates whose (logical) input and output wires are all $[\![x]\!]_{\mathsf{crt}}$ bundles (e.g., not just single boolean wires in the case of comparisons). A single logical value in the circuit is encoded by $k = O(\log Z / \log\log Z)$ wire labels. The total OT cost for a logical circuit input is $\sum_i \lceil \log p_i \rceil = O(\log Z)$.

The figure also includes a comparison to the "standard boolean garbled circuit" approach. To obtain these numbers, we consider directly converting the arithmetic circuit into a boolean circuit, and representing its logical values as binary integers of length $\log Z$ bits. In particular, this means that the *outputs* of the multiplication/exponentiation gates, not the *inputs*, are taken to be $\log Z$ bits long. To generate optimized boolean subcircuits for these operations, we first used Cryptol [10] to convert the input/output specification into an unoptimized circuit. We then used the ABC [8] and Yosys [19] circuit synthesis tools to create an optimized Verilog sequential circuit. Yosys was configured to treat XOR and NOT gates as free and otherwise minimize the circuit. The numbers in the table reflect the cost to garble each such subcircuit using the state-of-the art *half-gates* garbling scheme [21]. For multiplication- and exponentiation-by-constant, we chose arbitrary constants to obtain the subcircuits.

We see that, with the exception of comparison gates, our scheme results in less cost in almost all dimensions. Both the size of the garbled circuit and the memory requirement (to store garbled values) are smaller in our construction. The cost of OTs is slightly higher (12-37%). Additionally, we emphasize that linear operations (addition and multiplication by a constant) are free in our scheme.

7.3 Boolean Circuits with High Fan-In Gates

Our construction also gives improvements for Boolean circuits, specifically when gates have high fan in. In this setting, we consider Boolean circuits consisting of AND, OR, XOR, and threshold gates. Let b be an upper bound on the fan-in of any non-XOR gate in a circuit.

Our approach is to encode boolean values in a CRT representation with k primes, where $P_k := \prod_{i\le k} p_i > b$. This suffices for us to use the boolean AND/OR and threshold gates described in Sections 6.2 & 6.3. The costs are summarized in Figure 4, and they reflect *composable* gates whose logical outputs are also in the CRT representation.

A side-effect of using a CRT representation to encode single bits is that we no longer have XOR for free. Rather, we have addition mod P_k for free. However, since $2 \mid P_k$, the cost of XOR is indeed low. To compute the XOR of values $x_1, \ldots, x_n$, we add them mod P_k (for free), and then perform the transformation $[\![\sum_i x_i]\!]_{\mathsf{crt}} \mapsto [\![\sum_i x_i \bmod 2]\!]_{\mathsf{crt}}$ using the method in 6.1. The total cost of the final transformation is $k-1$ ciphertexts, regardless of the fan-in of this XOR gate.

As above, the figure also contains a comparison to the standard boolean garbled circuit paradigm. The corresponding numbers reflect the cost of garbling the best available boolean circuit using the half-gates construction. The numbers for threshold gates are for a majority gate (whereas the numbers for our scheme are for any threshold gate).

For circuits of this kind, our cost for OTs and for XOR gates is certainly higher. However, our exponential improvement for AND/OR/threshold gates is striking even for the modest values of fan-in that we consider.

7.4 Application Scenario

We now focus our comparison to a specific application. Suppose Alice and Bob have private vectors $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$, respectively, and they would like to privately compute the inner product $\sum_i a_i b_i$, in the presence of semi-honest adversaries (i.e., using Yao's protocol). The entries of these vectors are 32-bit nonnegative integers (for example, these matrices could be a 32-bit fixed-point representation of real numbers), and so the inner product may contain 64-bit values. Such a computation is representative of a natural class of elementary linear-algebraic computations — for example, a matrix multiplication consists of many such inner products.

The computation consists of (1) an OT, (2) a multiplication gate, and (3) an addition gate, for each component of the parties' input vectors (of course there are n multiplications and $n-1$ additions, but we assume n is large enough that the difference of 1 addition is insignificant).

- **Using our scheme,** the parties would choose a CRT encoding large enough to avoid overflow — i.e., so that $\prod_{i\le k} p_i > 2^{64}$. In this case, $k = 16$. Alice can garble the simple arithmetic circuit using the approach outlined in Section 6.1. Since Alice knows one argument of each of the multiplication gates, the cost to garble each multiplication gate is $\sum_{i=1}^{16}(p_i - 1) = 365$ ciphertexts. Additions are free, and the cost of OT per input element is 74 ciphertexts.
- **Using the standard boolean approach,** the parties must generate a boolean circuit that expresses the arithmetic computation. For each component of the vectors, the circuit includes a 32-bit $\times$ 32-bit multiplication and 64-bit addition circuit. Using Figure 3, we see that the cost of an addition subcircuit is 126 ciphertexts, and the cost of a multiplication-by-constant is at least 3744 ciphertexts. Note that in the desired functionality, the garbled circuit must hide Alice's argument to the multiplication gate, so it is perhaps rather optimistic to use the cost of a multiplication-*by-constant* gate in our calculation. The OT cost per input element is 32 ciphertexts.

Overall, our total protocol cost per vector-component is $365 + 74 = 439$ ciphertexts compared to $3744 + 126 + 32 = 3902$ for the boolean case — an improvement of 88%.

This simple example demonstrates that our construction

	scheme	asymptotic cost	concrete cost $Z = 2^{16}$	$Z = 2^{32}$	$Z = 2^{64}$
fan-in-2 addition gate	us	0	0	0	0
	standard	$\log Z$	30	62	126
multiplication-by-constant gate	us	0	0	0	0
	standard	$\log^{1.585} Z$	232	758	3744
exponentiation-by-constant gate	us	$\log^2 Z/\log\log Z$	51	119	365
	standard	$\log^2 Z$	44	1864	8496
fan-in-2 multiplication gate	us	$\log^2 Z/\log\log Z$	313	724	2206
	standard	$\log^{1.585} Z$	330	1200	4494
comparison gate	us	$\log^3 Z/\log\log Z$	804	2541	11979
	standard	$\log Z$	32	64	128
OT cost per input integer value	us	$\log Z$	22	37	72
	standard	$\log Z$	16	32	64
total wire label size per integer value	us	$\log Z/\log\log Z$	7	10	16
	standard	$\log Z$	16	32	64

Figure 3: Cost of various operations garbling an arithmetic circuit, where $\lfloor \frac{Z-1}{2} \rfloor$ is an upper bound on the magnitude of any intermediate value in the computation. Costs are in # of ciphertexts (multiples of λ bits). "Standard" scheme refers to encoding values in binary and using the half-gates garbling scheme [21] on the best available boolean circuit.

	scheme	asymptotic cost	concrete cost $b = 10$	$b = 100$	$b = 1000$
arbitrary fan-in XOR gate	us	$\log b$	2	3	4
	standard	0	0	0	0
fan-in-$\leq b$ AND/OR gate	us	$\log^2 b/\log\log b$	13	21	33
	standard	b	18	198	1998
fan-in-$\leq b$ threshold/majority gate	us	$\log^3 b/\log\log b$	60	137	280
	standard	$b \log b$	36	948	30082
OT cost per input bit	us	$\log b$	6	9	13
	standard	1	1	1	1
total wire label size per logical bit	us	$\log b/\log\log b$	3	4	5
	standard	1	1	1	1

Figure 4: Cost of various operations garbling a boolean circuit consisting of high fan-in gates. b is an upper bound on the fan-in of any AND/OR/threshold gate in the circuit. Costs are in # of ciphertexts (multiples of λ bits). "Standard" scheme refers to using the half-gates garbling scheme [21] on the best available boolean circuit.

provides significant concrete improvement for secure computations of arithmetic operations. In larger and more realistic computations each input values is likely to be used many times, so the cost of the garbled circuit itself easily dominates the cost of the OTs. We note that our garbled circuit cost in this example is 365 vs $3744 + 12 = 3870$, a factor 10 difference. This example used the fact that Alice knew one of the inputs to every multiplication gate. Even when that is not the case, our construction is more efficient by a non-trivial factor.

8. COMPARISON TO OTHER SECURE COMPUTATION TECHNIQUES

Above we have extensively compared our scheme with the state of the art for standard garbling of boolean circuits, and its application to arithmetic circuits (when those are first converted to boolean). In this section we compare our results to other approaches to 2PC (the most natural application of garbled circuits) and other relevant work on garbled circuits & randomized encodings.

Secret-sharing-based 2PC.

A completely different paradigm for 2PC uses secret sharing, and natively supports addition and multiplication operations over a prime modulus. In this paradigm, addition is free and (depending on the required security) multiplication can be performed by exchanging a constant number of $GF(p)$-elements.

Garbled-circuit-based and secret-sharing-based 2PC are fundamentally different, so direct comparisons are difficult. We simply point out the similarities and differences. Certainly secret-sharing-based 2PC is and will likely always be better for some cases. Our focus here is on improving the garbled circuit paradigm for computations that are currently expensive/cumbersome in that paradigm, thus significantly closing the gap with secret-sharing-based 2PC in these cases.

Our scheme allows addition for free, just as in secret-sharing protocols. We require a particular type of modulus which makes our scheme likely only useful for computations over the integers (where any modulus large enough to prevent overflows works), whereas secret-sharing can use any prime modulus. Our other non-free operations are certainly more expensive than in a secret-sharing protocol, but we highlight that the garbled circuit paradigm results in a *constant-round* protocol while multiplications require interaction in a secret-sharing protocol. Hence, our work can be thought of combining some of the best features of both worlds for arithmetic computations (free addition, but constant-round, easily mixing boolean/arithmetic paradigms).

Garbled Arithmetic Circuits & Arithmetic Model.

Like us, Applebaum, Ishai, and Kushilevitz [2] also describe a scheme for directly garbling arithmetic circuits over the integers, and like us, their scheme supports additions for free.

Their construction requires a particular assumption, namely LWE over the integers, whereas our construction uses a simpler correlation-robust hash function assumption. As a practical consequence, our scheme can be instantiated using concrete primitives like AES to give much lower overhead (than LWE) in the constant factors and also to take advantage of hardware-accelerated AES that has already been extremely successful for traditional garbled circuits.

We also point out that our scheme uses the conventional paradigm for garbled circuits, making our scheme easier to integrate into existing systems, and combine with other garbling schemes (e.g., garble part of a computation using our scheme, and other parts using standard boolean garbling).

Finally, we note that [2] also outline a construction for garbling arithmetic circuits based on one-way functions, and relying on CRT encoding. The CRT encoding use as a technical tool is thus similar in our construction and the one they outline. However, their construction uses the CRT encoding simply to get the integer input into more efficient eventual bit-wise representation (with bit-wise representation of the input modulo each of the CRT components); after this, standard boolean garbled circuit techniques (e.g., Yao) are applied. As a result, that scheme is more efficient than directly transforming the circuit from arithmetic to boolean, but less efficient than either their main LWE construction or our constructions here.

Applebaum, Avron & Brzuska [1] define an *arithmetic model* for cryptography, in which primitives work by manipulating field elements in a black-box way. In this work they prove some lower bounds related to garbled circuits & randomized encodings. While our construction takes advantage of certain algebraic structures, it does not fall within this arithmetic model. An important feature of the arithmetic model is that the construction is *oblivious* to the choice of underlying field, whereas we rely on specific choices of the underlying algebraic structure. Furthermore, their model focuses on information-theoretic constructions, or at least constructions that can be constructed in a black-box way from an arbitrary field. This does not capture our use of generic cryptographic assumptions like correlation-robust hash functions.

Fully Homomorphic Encryption.

Fully homomorphic encryption can be used for secure computation in a natural way. One party sends an encryption of his input $E(x)$, while the other party uses the homomorphic properties to compute $E(f(x, y))$ (here we assume semi-honest parties). While we cannot compete with such a protocol in terms of its asymptotically optimal communication overhead, our construction requires only symmetric-key operations and has small concrete constants.

Other Related Work.

We mention the recent work of Malkin, Pastro & shelat [15], which bears some similarities with our work in terms of high level themes explored, though the goals and directions they take and the concrete results obtained are quite different from ours.

First, as we already pointed out, they also use the free-XOR generalization to a larger modulus, yielding the same basic (free) addition as us. However, their multiplication is linear in the modulus size, and thus they can only apply this to small fields (they suggest fields of size up to 2^7). In contrast, we can handle moduli that are orders of magnitude larger, with CRT based polylogarithmic multiplication (and other gadgets).

In fact, garbling arithmetic gates is only a small part of their work, and their main focus is boolean circuits. There, similarly to us, they consider direct garbling of more complex gates, with higher fan-in, rather than the standard fan-in 2 gates. Specifically, they provide a garbling scheme for gates computing low-degree polynomials with many terms. However, their scheme does not yield any improvement when applied to the type of boolean gadgets that we consider here, e.g., a high fan-in AND gate. On the technical level, they do not use their generalized free-XOR technique in the boolean domain (it's only used for an arithmetic circuits over the given field). In contrast, for us, the free-XOR generalization is a major insight we use in obtaining improvements in the boolean domain. It would be interesting to explore whether a combination of their techniques and ours can yield even more significant improvements for specific, useful circuits.

Finally, they bypass the [21] lower bound of 2 ciphertexts per AND gate, by directly garbling a composition of several binary gates together, while [21] only consider a gate-by-gate garbling model (for binary gates). We bypass the lower bound in a very different way, breaking it even for a single binary AND gate. As explained above, we do so by exposing an implicit assumption embedded in the model of [21], namely that there's a single color bit.

9. CONCLUSIONS

We have introduced new techniques for garbled circuits, based on a generalization of Free-XOR that yields free addition mod m. Starting with rather simple building blocks, we show how to construct gadgets for garbling boolean and arithmetic gates with significantly lower cost (both asymptotically and concretely) than state-of-the-art garbling techniques. In particular, we can garble arithmetic circuits over the integers with free addition and multiplication by a constant, and we can garble boolean circuits of high fan-in exponentially better than standard techniques.

Acknowledgements

We are grateful to Brent Carmer for help synthesizing the boolean circuits considered in Figures 3 and 4. This work was initiated while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467. The first two authors are supported in part by the Defense Advanced Research Project Agency (DARPA) and Army Research Office (ARO) under Contract #W911NF-15-C-0236, and NSF grants #CNS-1445424 and #CCF-1423306. Part of the work was done while the first author was visiting IDC Herzliya's FACT center and supported by NSF-BSF Cyber Security and Privacy grant #2014/632. The third author is supported by NSF award #1149647. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency, Army Research Office, the National Science Foundation, or the U.S. Government.

10. REFERENCES

[1] Applebaum, B., Avron, J., and Brzuska, C. Arithmetic cryptography: Extended abstract. In *ITCS 2015* (Jan. 2015), T. Roughgarden, Ed., ACM, pp. 143–151.

[2] Applebaum, B., Ishai, Y., and Kushilevitz, E. How to garble arithmetic circuits. In *52nd FOCS* (Oct. 2011), R. Ostrovsky, Ed., IEEE Computer Society Press, pp. 120–129.

[3] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 13* (Nov. 2013), A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds., ACM Press, pp. 535–548.

[4] Beaver, D. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC* (May 1996), ACM Press, pp. 479–488.

[5] Beaver, D., Micali, S., and Rogaway, P. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC* (May 1990), ACM Press, pp. 503–513.

[6] Bellare, M., Hoang, V. T., Keelveedhi, S., and Rogaway, P. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy* (May 2013), IEEE Computer Society Press, pp. 478–492.

[7] Bellare, M., Hoang, V. T., and Rogaway, P. Foundations of garbled circuits. In *ACM CCS 12* (Oct. 2012), T. Yu, G. Danezis, and V. D. Gligor, Eds., ACM Press, pp. 784–796.

[8] Berkeley Verification and Synthesis Research Center. ABC: a system for sequential synthesis and verification. http://www.eecs.berkeley.edu/~alanmi/abc/.

[9] Choi, S. G., Katz, J., Kumaresan, R., and Zhou, H.-S. On the security of the "free-XOR" technique. In *TCC 2012* (Mar. 2012), R. Cramer, Ed., vol. 7194 of *LNCS*, Springer, Heidelberg, pp. 39–53.

[10] Galois, Inc. Cryptol: The language of cryptography. http://www.cryptol.net/.

[11] Gueron, S., Lindell, Y., Nof, A., and Pinkas, B. Fast garbling of circuits under standard assumptions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015* (2015), I. Ray, N. Li, and C. Kruegel, Eds., ACM, pp. 567–578.

[12] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. In *CRYPTO 2003* (Aug. 2003), D. Boneh, Ed., vol. 2729 of *LNCS*, Springer, Heidelberg, pp. 145–161.

[13] Kolesnikov, V., Mohassel, P., and Rosulek, M. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In *CRYPTO 2014, Part II* (Aug. 2014), J. A. Garay and R. Gennaro, Eds., vol. 8617 of *LNCS*, Springer, Heidelberg, pp. 440–457.

[14] Kolesnikov, V., and Schneider, T. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*. Springer, 2008, pp. 486–498.

[15] Malkin, T., Pastro, V., and shelat, a. An algebraic approach to garbling. Unpublished manuscript. See https://simons.berkeley.edu/talks/tal-malkin-2015-06-10, 2016.

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

[17] Nielsen, J. B., and Orlandi, C. LEGO for two-party secure computation. In *TCC 2009* (Mar. 2009), O. Reingold, Ed., vol. 5444 of *LNCS*, Springer, Heidelberg, pp. 368–386.

[18] Pinkas, B., Schneider, T., Smart, N. P., and Williams, S. C. Secure two-party computation is practical. In *ASIACRYPT 2009* (Dec. 2009), M. Matsui, Ed., vol. 5912 of *LNCS*, Springer, Heidelberg, pp. 250–267.

[19] Wolf, C. Yosys open synthesis suite. http://www.clifford.at/yosys/.

[20] Yao, A. C.-C. How to generate and exchange secrets (extended abstract). In *27th FOCS* (Oct. 1986), IEEE Computer Society Press, pp. 162–167.

[21] Zahur, S., Rosulek, M., and Evans, D. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT 2015, Part II* (Apr. 2015), E. Oswald and M. Fischlin, Eds., vol. 9057 of *LNCS*, Springer, Heidelberg, pp. 220–250.