Fast Evaluation of S-boxes with Garbled Circuits

Erik Pohle, Aysajan Abidin, Bart Preneel

***Abstract*—Garbling schemes are vital primitives for privacy-preserving protocols and secure two-party computation. This paper presents a projective garbling scheme that assigns 2^n values to wires in a circuit comprising XOR and unary projection gates. A generalization of FreeXOR allows the XOR of wires with 2^n values to be very efficient. We then analyze the performance of our scheme by evaluating substitution-permutation ciphers. Using our proposal, we measure high-speed evaluation of the ciphers with a moderately increased cost in garbling and bandwidth. Theoretical analysis suggests that for evaluating the nine examined ciphers, one can expect a 4- to 70-fold improvement in evaluation performance with, at most, a 4-fold increase in garbling cost and, at most, an 8-fold increase in communication cost compared to state-of-the-art garbling schemes. In an offline/online setting, such as secure function evaluation as a service, the circuit garbling and communication to the evaluator can proceed before the input phase. Thus, our scheme offers a fast online phase. Furthermore, we present efficient Boolean circuits for the S-boxes of TWINE and Midori64 ciphers. To our knowledge, our formulas give the smallest number of AND gates for the S-boxes of these two ciphers.**

I. Introduction

Privacy-preserving protocols enable collaborative computation on sensitive data while protecting the privacy of the sensitive data. Successful implementations in a two-party scenario include privacy-preserving genome analysis [1], email spam filtering [2], image processing [3] and machine learning [4]. The formalization of such two-party computation is called Secure Function Evaluation (SFE). Here the two parties, namely, Alice and Bob, want to compute a public function $f(x, y)$, where x is the input of Alice and y is the input of Bob, without revealing their input to each other. Yao's garbled circuit protocol [5] has become a practical solution for SFE. Moreover, garbling schemes (derived from the original garbled circuit construction) have also been identified as a useful cryptographic primitive. Most of the previous works focus on projective garbling schemes that assign two values to a wire, 0 and 1, such as the garbling scheme Half-Gates by Zahur et al. [6] or the work by Rosulek and Roy [7].

This paper considers garbling schemes in the offline/online setting. The offline phase performs function-dependent preprocessing. Concretely, the garbler garbles the circuit computing f and transmits the garbled gates to the evaluator but withholds the wire labels for the input layer. Once the input data of the garbler and the evaluator is available, the parties engage to obtain the appropriate wire labels for their respective inputs. Then, the evaluator evaluates the garbled circuit. The offline phase can be performed ahead of time and even batched to allow for optimal use of hardware and bandwidth if multiple function evaluations are expected. Hence, the online time, i.e., the time from having obtained the respective inputs to the evaluated output of the garbled circuit, is essential in this setting. This offline/online setting enables an efficient SFE as a service where the SFE service providers agree on a set of useful functions. The offline phase is run when the system is under low load and pre-processing results are stored. This way, the user of the service benefits from improved online times.

In this work, we examine a projective garbling scheme that assigns 2^n values to a wire. As a consequence, each wire in the circuit carries the semantics of an n-bit string. We generalize the encoding of FreeXOR by Kolesnikov and Schneider [8] to obtain a scheme where bitwise-XOR between n-bit strings is free. Our scheme allows fast evaluation of highly non-linear functions with n input bits at a moderate additional garbling and bandwidth cost in the offline phase. We demonstrate this trade-off by implementing several symmetric-key primitives of a certain structure (cf. Sect. VI). Further, the presented scheme may exhibit useful trade-offs for programs with many small, secret table lookups.

An Internet of Things (IoT) to Cloud scenario is a particular application of our garbling scheme. In this context, the focus is on encrypting data at the source, specifically on the IoT devices, employing a Substitution-Permutation Network (SPN) cipher and efficient distributed decryption in the Cloud prior to privacy-preserving computation on the data. This approach would facilitate end-to-end secure data collection and processing. Our proposed garbling scheme aims to balance the security demands of IoT-to-Cloud paradigms and the practical constraints of resource-constrained IoT devices, ultimately contributing to a more resilient and privacy-preserving IoT-to-Cloud ecosystem.

While the new garbling scheme assumes semi-honest adversaries, i.e. neither the garbler nor the evaluator may deviate from the protocol, several general approaches exist to make a garbled circuit protocol secure in the presence of active adversaries, which are allowed to deviate arbitrarily from the protocol. Prominent examples are based on cut-and-choose [9], [10], [11], [12], on zero-knowledge proofs [13], [14] and authenticated garbling [15], [16], [17]. Moreover, semi-honest garbling schemes can be compiled into actively secure three party protocols in the honest majority setting [18].

1) Technical Overview.: The core ideas of the scheme are summarized as follows. We encode an n-bit string with bits $x_1, x_2, \dots x_n$ into a wire label as

$$W \oplus x_1 R_1 \oplus x_2 R_2 \oplus \cdots \oplus x_n R_n \ ,$$

where W is a random label. We call R_i the wire label offsets that are randomly chosen by the garbler but fixed for all encodings in the circuit (see Definition 1 for details) and if $x_i = 1$, $x_i R_i$ is R_i, otherwise it is the zero string.

COSIC KU Leuven, Belgium.

This work is supported by the Flemish Government through FWO SBO project MOZAIK S003321N.

For $n = 1$, this is the encoding of FreeXOR. We define two types of gates, XOR and projection gates. XOR gates compute the bitwise-XOR of two n-bit strings, require little garbling and evaluation work and are non-interactive, making them practically free. A projection gate computes any n-bit to m-bit function on a wire value by using Yao's garbled table lookup, i.e., encrypting output wire labels using the respective input wire label as key. We apply standard garbled row reduction [19] and point-and-permute techniques [20]. For a projection gate, the garbler's work is 2^n calls to the encryption primitive, and $2^n - 1$ ciphertexts have to be sent to the evaluator. However, the evaluator only makes a single call to the encryption primitive, independent of the "size" n of the projection gate. This makes the scheme attractive in the pre-processed garbled circuit model since any non-linear n-bit functionality is evaluated with one call to the cryptographic primitive.

2) Contributions.: We present a projective garbling scheme that assigns n-bit strings to each wire and in which XOR gates are free. The specific encoding of an n-bit string in a label allows seamless integration into existing garbling schemes that assign two values per wire. Following the spirit of modular proofs, we identify necessary properties of the cryptographic primitive that is used to encrypt the truth table. Subsequently, we obtain a generalization of tweakable circular correlation robustness (TCCR, first defined by Choi et al. [21]), which we call n-TCCR, for hash functions (denoted by $\mathcal{H}$).

We apply the garbling scheme to compute a number of selected symmetric-key primitives that follow the SPN architecture. For these, we show a significant improvement in evaluation work in the online phase over the state-of-the-art schemes Half-Gates[6] and ThreeHalves [7] that is traded off with moderate additional garbling work and/or communication cost in the offline phase. Table I shows the estimated evaluation improvement based on calls to $\mathcal{H}$, which is complemented by a practical implementation in Sect. VI that shows that this evaluation improvement translates into practice (see Table VII). We obtain evaluation times for, e.g., AES as low as 0.016 ms.

Furthermore, to facilitate implementation, we give Boolean circuits for the S-boxes of TWINE [22] and Midori64 [23], which is also used in MANTIS [24] and CRAFT [25], using only AND and XOR gates. To the best of our knowledge, our Boolean circuits give the smallest number of AND gates for these two ciphers, namely, 6 AND gates for TWINE's 4-bit S-box and 4 AND gates for the Midori64 Sb_0 S-box. Details can be found in Appendix A.

3) Organisation.: The rest of the paper is organized as follows. In Sect. II, we review related work on garbling schemes. Section III gives more details on previous work which we build upon. We present our scheme in Sect. IV and prove its security in Sect. V. A comparison of the state of the art and our scheme for nine (lightweight) symmetric primitives is given in Sect. VI. Finally, we conclude the paper in Sect. VII.

Table I: Evaluation work improvement for selected symmetric primitives over ZRE15 [6]. Garbling and communication trade-off is listed in Table VI.

Primitive	Evaluation Improvement	Primitive	Evaluation Improvement
AES-128 [26]	$\times 26.23$	Piccolo-128	$\times 4.55$
CRAFT [25]	$\times 5.71$	SKINNY-64-64 [24]	$\times 7.11$
Fides-80 [27]	$\times 15.45$	SKINNY-64-128	$\times 4.68$
Fides-96	$\times 65.05$	TWINE-80 [22]	$\times 9.81$
MANTIS [24]	$\times 4.31$	TWINE-128	$\times 9.05$
Midori64 [23]	$\times 5.33$	WAGE [28]	$\times 72.87$
Piccolo-80 [29]	$\times 4.71$		

Table II: Comparison of pre-processed lookup table (LUT) approaches in MPC protocols. The depth of the circuit is denoted by d. Total communication is denoted in kilobytes.

Scheme	Round Complexity	Total Comm. in kB (4-bit LUT)	(8-bit LUT)
OTTT [38] (from [33, Table IV])	$\mathcal{O}(d)$	≈ 6	≈ 262
MiniMAC AES [36]	$\mathcal{O}(d)$	-	≈ 700
Dessouky et al. [33, Table IV]	$\mathcal{O}(d)$	0.039	0.288
This work	$\mathcal{O}(1)$	0.240	4.080

II. Related Work

Recent improvements on Yao's garbled circuit protocol in the passive security setting focus on lowering bandwidth requirements, e.g., [30], [31]. In the line of work [19], [20], [8] leading to the state-of-the-art schemes Half-Gates [6] and ThreeHalves [7], AND gates only require 2κ bits and $\approx 1.5\kappa$ bits, respectively, where κ is a security parameter, to be sent, while XOR gates are free. Recently, Acharya et al. [32] propose an approach to garbling where the garbled gate is no longer composed of ciphertexts from individual rows in the truth table, focusing only on binary gates.

While computation with binary values is mainly expressed in Boolean circuits with binary gates, gates with more inputs than two or more outputs than one have been studied as well. Dessouky et al. [33] define those gates as lookup tables and show how they can be evaluated in the passive security case in the Goldreich-Micali-Wigderson protocol [34]. Damgård et al. [35], [36] design a table lookup for two-party secure computation and Keller et al. [37] extend it to the multi-party case based on secret-sharing. The basis for the aforementioned constructions is the one-time truth-table protocol OTTT by Ishai et al. [38]. Table II compares the (estimated) communication cost of these approaches for a 4-bit and 8-bit lookup table, respectively. AES as a function has been studied explicitly by Durak and Guajardo [39], SKINNY and Photon were studied by Abidin et al. [40]. However, both works are in the arithmetic setting.

In the garbled circuit domain, Fairplay [41] and TASTY [42] already compute larger gates. Huang et al. [43] focus on an 8-bit to 8-bit AES S-box gate. But unlike our scheme, these works consider multiple wires instead of multiple values *per* wire. They also do not provide any security proof for the larger gates. Computing a gate with multiple input wires

Table III: Comparison of multi-input, multi-output gates in garbling schemes. We note the cost for a n-to-m-bit gate.

Scheme	Garbling Work	Circuit Size (κ-bit strings)	Evaluation Work
Fairplay [41]	$2^n \cdot m \cdot 2$ SHA-1	$2^n \cdot m$	$m \cdot 2$ SHA-1
TASTY [42]	$2^n \cdot m$ SHA-256	$(2^n - 1) \cdot m$	1 SHA-256
Huang et al. [43]	$2^n \cdot m$ SHA-1	$2^n \cdot m$	4 SHA-1
This work	2^n AES-NI	$2^n - 1$	1 AES-NI

Table IV: Notation.

κ	Security parameter
$\mathbf{v}$	Bold letters denote vectors
$\{0,1\}^l$	The set of bit-vectors of length l
W_α^β	The wire label of wire α that encodes the value β
$A \oplus B$	Bitwise XOR for $A, B \in \{0,1\}^l$
$A\|\|B$	Bit-vector concatenation for $A \in \{0,1\}^l, B \in \{0,1\}^{l'}$
$\{A, B\}\|\|C$	Bit-vector concatenation extended to sets, i.e., $\{A\|\|C, B\|\|C\}$
$x' \leftarrow x$	Assignment of value x to x'
$x \leftarrow\$ \{a, b, c, \dots\}$	Uniform sampling from the set $\{a, b, c, \dots\}$

necessitates more generic hash function constructions that operate on inputs longer than one block. Practical performance improvements are due to the use of AES-NI instructions in permutation-based constructions like [44], [45]. It is unclear how these constructions extend to the multi-input case in the context of garbled circuits. Our scheme uses a cryptographic primitive with fixed-length input, enabling the use of AES-NI instructions. An overview of garbling and evaluation work, circuit size and the hash function construction for a generic n-to-m-bit gate is given in Table III. Heath and Kolesnikov [46] construct a garbling gadget that computes a one-hot outer-product of two bit-vectors which can be used to select one entry from a truth-table based on an index known by the evaluator. However, their approach doesn't generalize to secret access to arbitrary truth-tables.

Since the work of Ball et al. [47] is conceptually very close to ours, we discuss it in detail in Sect. III-A. The main difference is that their scheme uses arithmetic circuits, where addition modulo an integer is free, while our proposal sticks to a bit representation where XOR is free.

III. Background

We start with the arithmetic circuit scheme by Ball et al. [47] in Sect. III-A and detail the security model by Bellare, Hoang and Rogaway (BHR) [48] in Sect. III-B. Table IV lists the notation used throughout the paper.

A. Garbled Circuits for Bounded Integers

Ball et al. [47] propose a scheme based on garbled circuits that assigns integers $x \in \mathbb{Z}_m$ to each wire in the circuit. In this representation, addition (in $\mathbb{Z}_m$) is free in the same sense as FreeXOR. We briefly describe their scheme as our scheme is similar but represents n-bit strings per wire instead of numbers in $\mathbb{Z}_m$.

The wire encoding of $x \in \mathbb{Z}_m$ is $W_i^x = W_i^0 + x \odot \Delta_m$, where $W_i^0, \Delta_m \in \mathbb{Z}_m^{\lambda_m}$, $\lambda_m = \lceil \frac{\kappa}{\log m} \rceil$. Addition is component-wise in the ring $\mathbb{Z}_m$. Here $\odot$ denotes a scalar multiplication. For each m, Δ_m is a secret, random vector known by the garbler.

The scheme mainly offers two types of gates, addition and unary projection. For addition of wires a and b with output wire c, let W_a^0, W_b^0 be the two input wire labels of zero, then the garbler computes $W_c^0 = W_a^0 + W_b^0$ as the output zero label. The evaluator, given W_a^x and W_b^y for evaluation, computes

$$W_c^{x+y} = W_a^x + W_b^y = \underbrace{W_a^0 + W_b^0}_{W_c^0} + (x+y) \odot \Delta_m \ .$$

Addition incurs neither transmitted ciphertexts nor invocations of the encryption primitive. Let $\phi : \mathbb{Z}_n \mapsto \mathbb{Z}_m$ be an arbitrary function. The projection gate Proj_ϕ computes the operation $\phi(x)$, $x \in \mathbb{Z}_n, \phi(x) \in \mathbb{Z}_m$. Let G be the garbled table, then the garbler fills G for every $x \in \mathbb{Z}_n$ as follows:

$$G[x+r] = H(W_a^0 + x \odot \Delta_n) + W_c^0 + \phi(x) \odot \Delta_m = H(W_a^x) + W_c^{\phi(x)} \ ,$$

where r is the secret cyclic shift offset. We can reduce the number of ciphertexts per projection gate to $n-1$ by applying garbled row reduction. The zero label is obtained when $r = -x$, $W_c^0 = -H(W_a^{-r}) - \phi(-r) \odot \Delta_m$, from the encryption above. This, analogous to the binary case, fixes the ciphertext of the first *garbled* row to 0^{λ_m}.

Again, one element of the label can be used as a pointer and replace the shift r during garbling if Δ_n is chosen appropriately. With this, the evaluator only has to decrypt the ciphertext the pointer indicates.

B. Security Model by Bellare, Hoang and Rogaway

Bellare, Hoang and Rogaway [48] define a security model for garbling schemes that formalizes the principle of circuit garbling as a cryptographic primitive. Many recent garbling schemes were proven secure in their model, e.g., [6], [49], [50], [7]. As we will use the same model, we give a brief overview.

A garbling scheme is a tuple of Garble, Encode, Eval and Decode algorithms:

- Garble: Transforms the input circuit f into the tuple (GC, e, d) where GC is the garbled circuit, e is the input encoding information (e.g., all semantic labels for input wires) and d is the decoding information.
- Encode: Encodes a given input x using the semantic labels e and returns a garbled input X, e.g., the input label with semantic x.
- Eval: Evaluates the garbled circuit GC using the input wire labels $\{W_i\}_{i \in \mathsf{Inputs}}$ and returns the output wire labels $\{W_i\}_{i \in \mathsf{Outputs}}$.
- Decode: Decodes the output wire labels $\{W_i\}_{i \in \mathsf{Outputs}}$ using the decoding information d and returns the plaintext output $y \in \{0,1\}^m$ or $\perp$ if the output wire labels are invalid.

The garbling scheme must produce correct circuit evaluations for any circuit f and inputs $x \in \{0,1\}^n$. Let GC, e, d be the outputs of $\mathsf{Garble}(f)$, and X_i the output of $\mathsf{Encode}(x_i, e)$ for $i \in \mathsf{Inputs}$ then

$$\mathsf{Decode}(\mathsf{Eval}(GC, \{X_i\}_{i \in \mathsf{Inputs}}), d) = f(x) \ ,$$

where $f(x)$ denotes the circuit evaluation in the clear.

Bellare et al. define two notions of secrecy. In the privacy notion, given (GC, X, d), a party cannot learn any information besides what is revealed from the final output y and the side-information function Φ. In our case, $\Phi = \Phi_{\text{topo}^*}$ where only the circuit topology and the XOR gates are revealed but the function computed by projection gates remains hidden to the evaluator[1]. The privacy property can be achieved by giving a simulator $\mathcal{S}$ for the Garble function that only receives the output y and Φ. In the code-based game in Fig. 1, the garbling scheme is prv.sim secure if for every polynomial-time adversary $\mathcal{A}$ there is a polynomial-time simulator $\mathcal{S}$ such that $\mathsf{Adv}(\text{prv.sim})$ is negligible, where

$$\mathsf{Adv}(\text{prv.sim}) = \left|\Pr[\mathcal{A} \text{ wins prv.sim}] - \frac{1}{2}\right| = \left|\Pr[b = b'] - \frac{1}{2}\right| .$$

Intuitively, if the output of the simulator is indistinguishable from the output of Garble and Encode on a circuit and input chosen by the adversary, the scheme is prv.sim secure. In the notion of obliviousness, the adversary does not learn the decoding function. So given (GC, X), a party cannot learn any information besides the side-information Φ. The advantage is defined analogously to $\mathsf{Adv}(\text{prv.sim})$.

function GARBLE(f, x)
$b \leftarrow\$ \{0,1\}$
if b = 1 **then**
$(GC, e, d) \leftarrow \mathsf{Garble}(f)$
$X \leftarrow \mathsf{Encode}(x, e)$
else
$y \leftarrow f(x)$
$(GC, X, d) \leftarrow \mathcal{S}(1^k, y, \Phi(f))$
return (GC, X, d)

(a) Game prv.sim$_{\Phi,\mathcal{S}}$.

function GARBLE(f, x)
$b \leftarrow\$ \{0,1\}$
if b = 1 **then**
$(GC, e, d) \leftarrow \mathsf{Garble}(f)$
$X \leftarrow \mathsf{Encode}(x, e)$
else
$(GC, X) \leftarrow \mathcal{S}(1^k, \Phi(f))$
return (GC, X)

(b) Game obv.sim$_{\Phi,\mathcal{S}}$.

Figure 1: For every circuit f and input x of the adversary's choice, the respective game function is called and the adversary outputs a choice b' given (GC, X, d) (resp. (GC, X)). The adversary wins if $b = b'$.

IV. The Scheme

In Sect. IV-A, we first describe the notation for a circuit comprising XOR gates and projection gates. Then, we detail how the garbler encodes n-bit strings and transforms them into wire labels. Next, in Sect. IV-B, we show how XOR gates are garbled and evaluated, followed by a description of how projection gates are garbled and evaluated. Section IV-C describes higher-level gadgets that can be obtained from the aforementioned gates. In Sect. IV-D, all concepts are pieced together to describe the garbling, evaluation and decoding function. We also describe how input is handled. The complete garbling scheme Π is given in Fig. 2. We start with some general notations. Let $\mathsf{lsb}_n(W)$ be the n least significant bits[2] of the bit-vector $W \in \{0,1\}^k$. With k we denote the wire label length. We use a hash function $\mathcal{H} : \{0,1\}^k \times \{0,1\}^\tau \mapsto \{0,1\}^k$ that accepts a k-bit input, a τ-bit tweak and outputs k bits. Further properties of $\mathcal{H}$ are presented in Sect. V-A.

A. Circuit Definition

We define a circuit with p-bit input and q gates. The function computed by the circuit is denoted by f. Let the wire index be $1, \ldots, p, p+1, \ldots, p+q$, where the input wires have index $1, \ldots, p$ and the output wire of the i-th gate has index $p + i$. We denote the set of input wire indices as Inputs, and the set of output wire indices as Outputs. We associate a bit-length $\ell(i)$ to each wire i. Let $\bar{n}$ denote the maximum bit-length of wires used in f, then we use bit strings of length $k = \kappa + \bar{n}$ as wire labels. Let Gates be a topologically sorted list of gates $\mathsf{G}_1, \ldots, \mathsf{G}_q$. We distinguish two types of gates: XOR and projection gates. XOR gates accept two wires of the same bit-length n as input and output a wire with bit-length n. The unary projection gate accepts one n-bit wire and outputs one m-bit wire.

Definition 1 (Wire Label Offsets)**.** For each bit-length n $(1 \le n \le \bar{n})$ that is used in f, a wire label offset is a bit-vector of length $k = \kappa + n$ with κ random bits and n fixed bits. The garbler draws the matrix $\mathbf{M} \in \{0,1\}^{\kappa \times n}$ at random and appends fixed bits to each column-vector to form $\mathbf{R}_n = \begin{pmatrix} \mathbf{M} \\ \mathbf{I}_n \end{pmatrix}$, where $\mathbf{I}_n \in \{0,1\}^{n \times n}$ is the identity matrix. The column vector R_i in $\mathbf{R}_n$ is the i-th wire label offset. We denote the distribution from which $\mathbf{R}_n$ is sampled $\mathcal{R}_n$, i.e., $\mathbf{R}_n \leftarrow\$ \mathcal{R}_n$.

The matrix $\mathbf{R}_n$ is used throughout the whole circuit for all wires of bit-length n. We use the last n bits of the label to fix distinct values to allow point-and-permute. The inner product of $x \cdot \mathbf{R}_n$ is defined as $x_1 R_1 \oplus \ldots \oplus x_n R_n$.

Definition 2 (Wire Label Encoding)**.** The encoding W_i^x of an n-bit string $x \in \{0,1\}^n$ on a wire with index i is defined as $W_i^x = W_i^{0^n} \oplus x \cdot \mathbf{R}_n$.

Note, this yields a unique encoding for all x and $\mathbf{R}$ even if the random part $\mathbf{M}$ is linearly dependent in the columns because the lower n bits of $x \cdot \mathbf{R}$ are always unique due to $\mathbf{I}_n$.

Intuitively, there are n distinct offsets R, one for each encoded bit. The offset applied to a wire label that encodes x is the linear combination of R values.

B. Gates

For an XOR gate with n-bit input wires a and b, and output wire c, the garbler generates the output wire label $W_c^x \leftarrow W_a^{0^n} \oplus W_b^{0^n} \oplus x \cdot \mathbf{R}_n$ where $x \in \{0,1\}^n$. No ciphertext is sent.

To evaluate an XOR gate, let W_a and W_b be the wire labels that the evaluator obtained as input labels for the XOR gate. The output label is then computed as $W_c \leftarrow W_a \oplus W_b$.

A projection gate Proj_ϕ computes the unary projection $\phi : \{0,1\}^n \mapsto \{0,1\}^m$, a n-to-m-bit function. Let a be the input wire index to the projection gate and c be the index of the output wire, the garbler first draws the output wire label for 0 at random: $W_c^{0^m} \leftarrow\$ \{0,1\}^k$ and then generates

[1] In [48] Φ_{topo} is defined as completely gate hiding, we therefore denote the slightly weaker notion topo*.

[2] The exact location of the n bits in W is not important for the scheme as long as it is consistently used by both parties.

function $\textsc{GenR}(\bar{n})$
  for $i \in \{1,\ldots,\bar{n}\}$ **do**
    $\mathbf{R}_i \leftarrow\!\!\$\ \mathcal{R}_i$
  return $\mathbf{R}_1,\ldots,\mathbf{R}_{\bar{n}}$

function $\textsc{Garble}(f)$
  $\mathbf{R}_1,\ldots,\mathbf{R}_{\bar{n}} \leftarrow \textsc{GenR}(\bar{n})$
  for $i \in \mathsf{Inputs}$ **do**
    $W_i^{0^{\ell(i)}} \leftarrow\!\!\$\ \{0,1\}^k$
  for $\mathsf{G}_i \in \mathsf{Gates}$ **do**
    if $\mathsf{G}_i = XOR$ **then** ▷ G_i with n-bit input wires a,b
      $W_i^{0^n} \leftarrow W_a^{0^n} \oplus W_b^{0^n}$
    else ▷ G_i with n-bit input wire a
      $W_i^{0^m} \leftarrow\!\!\$\ \{0,1\}^k$ ▷ and $\phi : \{0,1\}^n \mapsto \{0,1\}^m$
      for $x \in \{0,1\}^n$ **do**
        $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathcal{H}(W_a^x, i) \oplus W_i^{0^n} \oplus \phi(x) \cdot \mathbf{R}_n$
  $d \leftarrow \{\mathsf{lsb}_{\ell(i)}(W_i^{0^{\ell(i)}})\}_{i\in\mathsf{Outputs}}$
  $e \leftarrow \{W_i^{0^{\ell(i)}}\}_{i\in\mathsf{Inputs}}, \mathbf{R}_1,\ldots,\mathbf{R}_{\bar{n}}$
  return GC, e, d

function $\textsc{Encode}(e, \{x_i\}_{i\in\mathsf{Inputs}})$
  for $i \in \mathsf{Inputs}$ **do**
    $X_i \leftarrow W_i^{0^{\ell(i)}} \oplus x_i \cdot \mathbf{R}_{\ell(i)}$
  return $\{X_i\}_{i\in\mathsf{Inputs}}$

function $\textsc{Eval}(GC, \{W_i\}_{i\in\mathsf{Inputs}})$
  for $\mathsf{G}_i \in \mathsf{Gates}$ **do**
    if $\mathsf{G}_i = XOR$ **then** ▷ G_i with input wires a, b
      $W_i \leftarrow W_a \oplus W_b$
    else ▷ G_i with n-bit input wire a
      $W_i \leftarrow \mathcal{H}(W_a, i) \oplus GC[i, \mathsf{lsb}_n(W_a)]$
  return $\{W_i\}_{i\in\mathsf{Outputs}}$

function $\textsc{Decode}(\{W_i\}_{i\in\mathsf{Outputs}})$
  for $i \in \mathsf{Outputs}$ **do**
    $y_i \leftarrow d_i \oplus \mathsf{lsb}_{\ell(i)}(W_i)$
  return y

Figure 2: The new garbling scheme Π comprises a garble, evaluation, encoding and decoding function.

2^n ciphertexts for each $x \in \{0,1\}^n$ and stores the result in the garbled table at the position indicated by the pointer bits, i.e., $GC[c, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathcal{H}(W_a^x, c) \oplus W_c^{\phi(x)}$. We apply the row-reduction technique and reduce the number of ciphertexts that need to be sent by one. Let a be the input wire index to the projection gate Proj_ϕ and c be the index of the output wire. Then, the garbler chooses the output wire label for 0^m as

$$W_c^{0^m} = \mathcal{H}(W_a^{\mathsf{lsb}_n(W_a^{0^n})}, c) \oplus \phi(\mathsf{lsb}_n(W_a^{0^n})) \cdot \mathbf{R}_n$$

and computes the remaining ciphertexts as described above. Since the first ciphertext (where $x = \mathsf{lsb}_n(W_a^{0^n})$) is always 0^k, it does not need to be sent. The number of rows sent to the evaluator is therefore $2^n - 1$.

For evaluation, let W_a be the wire label that the evaluator obtained as input to the projection gate. The output label W_c is computed by

$$W_c \leftarrow GC[c, \mathsf{lsb}_n(W_a)] \oplus \mathcal{H}(W_a, c) \ ,$$

where the position of the ciphertext to evaluate is indicated by the pointer bits of the input wire label. The first ciphertext is set to 0: $GC[c, 0^n] = 0^m$.

C. Circuit Constructions

Below, we give useful gadgets comprised of XOR and projection gates.

Wire Composition. We can compose an n-bit wire a with an m-bit wire b resulting in a $(n+m)$-bit wire c. The composition construction computes the functionality $f : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^{n+m}$ defined by $f(x,y) = x||y$. The composition is then $W_c \leftarrow \mathsf{Proj}_s(W_a) \oplus \mathsf{Proj}_{s'}(W_b)$, where $s : \{0,1\}^n \mapsto \{0,1\}^{n+m}$ is defined as $s(x) = x||0^m$ and $s' : \{0,1\}^m \mapsto \{0,1\}^{n+m}$ is given as $s'(y) = 0^n||y$. Wire composition costs $2^n + 2^m$ ciphertexts to garble and two ciphertexts to evaluate.
Note that the construction is not limited to two arguments. It is efficient to compose many wires together at once instead of cascading or using a tree-based approach[3]. E.g., to compose four 1-bit wires a, b, c, d , we may use

$$\mathsf{Proj}_{s_a}(W_a) \oplus \mathsf{Proj}_{s_b}(W_b) \oplus \mathsf{Proj}_{s_c}(W_c) \oplus \mathsf{Proj}_{s_d}(W_d) \ ,$$

where $s_a(x) = 000||x$, $s_b(x) = 00||x||0$, $s_c(x) = 0||x||00$, $s_d = x||000$. This costs $4 \cdot 2^1 = 8$ ciphertexts (or 4 with row reduction) instead of $4 \cdot 2^1 + 2 \cdot 2^2 = 16$ (resp. 10) ciphertexts.

Wire Decomposition. Likewise, we can decompose, i.e., split, a $2n$-bit wire into two n-bit wires. Let W_a be a $2n$-bit wire, then the decomposition construction computes $f : \{0,1\}^{2n} \mapsto \{0,1\}^n$ defined as $f(x_1||\ldots||x_{2n}) = x_1||\ldots||x_n$ and $f' : \{0,1\}^{2n} \mapsto \{0,1\}^n$ as $f'(x_1||\ldots||x_{2n}) = x_{n+1}||\ldots||x_{2n}$ via two projection gates. Note that this time, a tree-like decomposition, e.g., from 4-bit to 2-bit to 1-bit, is more efficient than constructing four projections from 4-bit to 1-bit. The latter costs $4 \cdot 2^4 = 64$ ciphertexts (60 with row reduction) while the former costs $2 \cdot 2^4 + 4 \cdot 2^2 = 48$ (resp. 42) ciphertexts.

Constants. In the garbling scheme, we can encode public constants or constants known only to the garbler at no cost. Let $x \in \{0,1\}^n$ be the constant for the n-bit wire a, then the garbler chooses $W_a^{0^n} \leftarrow x \cdot \mathbf{R}_n$. This fixes the label W_a^x to 0^k. No ciphertext is sent to the evaluator. Likewise, the evaluator uses $W_a = 0^k$ for further evaluation.

D. Garbling Scheme

We now describe the complete garbling scheme (see Fig. 2).

Garble. The garbler chooses $\bar{n}$ matrices of offset values (see Definition 1). For each input bit i, a wire label W_i^0 is chosen uniformly at random. The garbling process applies

[3]Following the example, the tree-based approach first composes $a||b$ and $c||d$ resulting in 2-bit wires. Then $ab||cd$ is composed.

the operations for projection and XOR gates as described in Sect. IV-B gate-by-gate in topological order. In the end, the garbling routine outputs the ciphertexts, input wire values, offsets and decoding information.

Encoding and Oblivious Transfer. The garbler encodes their own input by picking the respective wire label. In Yao's protocol, the evaluator obtains the appropriate wire labels that correspond to its input via oblivious transfer (OT) [51]. Using OT extensions [38], [52] speeds this up in practice. To obtain the correct label for an n-bit wire, one could simply perform a 1-out-of-2^n OT. Naor and Pinkas [53] show how to reduce this to n 1-out-of-2 OTs by introducing additional pseudorandom function (PRF) evaluations. However, using the FreeXOR property of our scheme, we can instead perform only n 1-out-of-2 OTs (as in a garbling scheme with 2 wire labels). For each input bit b_i at position i, the sender sends

$$\begin{array}{ll} W_i^{0^n} & \text{if } b_i = 0\,, \\ W_i^{0^n} \oplus R_i & \text{if } b_i = 1\,, \end{array}$$

where R_i is the i-th column vector in $\mathbf{R}_n$. To obtain the wire label for the n-bit wire, we XOR the obtained labels together at no additional cost. Note that $W_i^{0^n}$ is a fresh random wire label for each bit i of the input.

Evaluation and Decoding. Once the evaluator obtains the garbled inputs, it computes the garbled output of each gate accordingly (see Sect. IV-B). Having computed the garbled output, the evaluator may either share the wire labels with the garbler or directly use the decoding information $d_i = \mathsf{lsb}_n(W_i^{0^n})$ for output wire $i \in \mathsf{Outputs}$ in the decoding function to obtain the output bits in the clear. Let us briefly look at why this decoding scheme is correct. Let i be an output wire. Since we fixed $\mathsf{lsb}_n(y \cdot \mathbf{R}_n) = y \cdot \mathbf{I}_n = y$ by construction of the offset values, for any value $y \in \{0,1\}^n$, we have $\mathsf{lsb}_n(W_i^y) = \mathsf{lsb}_n(W_i^{0^n}) \oplus y$. As $d_i = \mathsf{lsb}_n(W_i^{0^n})$, the decoding is correct

$$d_i \oplus \mathsf{lsb}_n(W_i^x) = \mathsf{lsb}_n(W_i^{0^n}) \oplus \mathsf{lsb}_n(W_i^{0^n}) \oplus \mathsf{lsb}_n(y \cdot \mathbf{R}_n) = y\,.$$

V. Security

Using the BHR security model (see Sect. III-B) we show that if a hash function satisfies the properties of n-TCCR security defined in Sect. V-A below, our scheme is prv.sim (Sect. V-B) and obv.sim (Sect. V-C) secure. We sketch how to achieve authenticity in Sect. V-D.

A. (n-)TCCR Security

We revisit the tweakable circular correlation robustness (TCCR) definition by Guo et al. [44] adapted to our notation.

Definition 3 (TCCR Security [44])**.** A TCCR (tweakable circular correlation robust) hash function $\mathcal{H}$ is a function $\{0,1\}^k \times \{0,1\}^\tau \mapsto \{0,1\}^k$ that accepts a message m and a tweak t. In the TCCR security game, the distinguisher $\mathcal{D}_{\text{TCCR}}$ is given one of the two oracles with signature $\{0,1\}^k \times \{0,1\}^\tau \times \{0,1\} \mapsto \{0,1\}^k$

- (Real) $\mathcal{O}_R(m,t,b) = \mathcal{H}(m \oplus R, t) \oplus bR$
- (Ideal) $\mathsf{Rand}(m,t,b)$ is a random function.

with the goal to decide which is the oracle given to it. The distinguisher doesn't know the secret value $R \in \{0,1\}^k$, $R \leftarrow\!\$\ \mathcal{R}_{\text{TCCR}}$ and is only allowed to make legal queries. An illegal query is $(m,t,1-b)$ if (m,t,b) has been queried before. We define the advantage as

$$\begin{aligned} &\mathsf{Adv}_{\mathcal{R}_{\text{TCCR}}}(\mathcal{D}_{\text{TCCR}}) \\ &\quad = \left| \Pr[\mathcal{D}_{\text{TCCR}}^{\mathsf{Rand}}(1^\kappa) = 0] - \Pr_{R \leftarrow \mathcal{R}_{\text{TCCR}}}[\mathcal{D}_{\text{TCCR}}^{\mathcal{O}_R}(1^\kappa) = 0] \right|, \end{aligned}$$

where $\mathcal{D}^{\mathcal{O}}$ signifies that the distinguisher has access to oracle $\mathcal{O}$. We call $\mathcal{H}$ TCCR secure if $\mathsf{Adv}_{\mathcal{R}}(\mathcal{D}_{\text{TCCR}})$ is negligible in the security parameter κ.

Note that the advantage of $\mathcal{D}_{\text{TCCR}}$ depends on the distribution of the secret value $\mathcal{R}$. Next, we define n-TCCR security, a generalized TCCR notion incorporating n secret offsets.

Definition 4 (n-TCCR Security)**.** A n-TCCR hash function $\mathcal{H}$ is a function $\{0,1\}^k \times \{0,1\}^\tau \mapsto \{0,1\}^k$ that accepts a message m and a tweak t. In the n-TCCR security game, the distinguisher $\mathcal{D}_{\text{n-TCCR}}$ is given one of the two oracles with signature $\{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^k$

- (Real) $\mathcal{O}_{\mathbf{R}}(m,t,\mathbf{a},\mathbf{b}) = \mathcal{H}(m \oplus \mathbf{a} \cdot \mathbf{R}, t) \oplus \mathbf{b} \cdot \mathbf{R}$
- (Ideal) $\mathsf{Rand}(m,t,\mathbf{a},\mathbf{b})$ is a random function

with the goal to decide which is the oracle given to it. We interpret $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$ as binary vectors, $\mathbf{R} = (R_1, \ldots, R_n)$, $\mathbf{R} \in \{0,1\}^{k \times n} \leftarrow\!\$\ \mathcal{R}_n$ and $R_i \in \{0,1\}^k$, $1 \le i \le n$. The expression $\mathbf{a} \cdot \mathbf{R} = a_1 R_1 \oplus \cdots \oplus a_n R_n$ is the linear combination of offsets defined by $\mathbf{a}$. The distinguisher doesn't know the secret value $\mathbf{R}$ and is only allowed to make legal queries. An illegal query is $\mathbf{a} = 0$ or $(m,t,\mathbf{a},\mathbf{b}')$ if $(m,t,\mathbf{a},\mathbf{b})$ has been queried before for $\mathbf{b} \neq \mathbf{b}'$.
We define the advantage as

$$\begin{aligned} &\mathsf{Adv}_{\mathcal{R}_n}(\mathcal{D}_{\text{n-TCCR}}) \\ &\quad = \left| \Pr[\mathcal{D}_{\text{n-TCCR}}^{\mathsf{Rand}}(1^\kappa) = 0] - \Pr_{\mathbf{R} \leftarrow \mathcal{R}_n}[\mathcal{D}_{\text{n-TCCR}}^{\mathcal{O}_{\mathbf{R}}}(1^\kappa) = 0] \right|. \end{aligned}$$

We call $\mathcal{H}$ n-TCCR secure if $\mathsf{Adv}_{\mathcal{R}_n}(\mathcal{D}_{\text{n-TCCR}})$ is negligible in κ.

B. Privacy

The prv.sim definition states that given the garbled circuit GC, all the labels of the garbled input X and the decoding information d, no information is revealed about the input except from what can be deduced from the output y.

Theorem 1. *Given a $\bar{n}$-TCCR secure hash function $\mathcal{H}$ and $\bar{n} \ll \kappa$, the garbling scheme Π is prv.sim secure.*

Proof. We define a simulator $\mathcal{S}$ (see Fig. 3) and show through a series of hybrids that the output of $\mathcal{S}$ is indistinguishable for an adversary from the output of Garble. We require $\bar{n} \ll \kappa$, i.e., the largest bit length $\bar{n}$ used in a wire in the circuit is small compared to the security parameter κ, to ensure that for any adversarially chosen circuit, both the garbling scheme and the simulator run in polynomial time. When evaluating a garbled circuit, let the assignment of active labels to the

1: **function** $\mathcal{S}(f, y)$
2: **for** $i \in \mathsf{Inputs}$ **do**
3: $W_i^{0^{\ell(i)}} \leftarrow\!\!\$\ \{0,1\}^k$
4: $X_i \leftarrow W_i^{0^{\ell(i)}}$
5: **for** $\mathsf{G}_i \in \mathsf{Gates}$ **do**
6: **if** $\mathsf{G}_i = XOR$ **then** ▷ G_i with n-bit input wires a, b
7: $W_i^{0^n} \leftarrow W_a^{0^n} \oplus W_b^{0^n}$
8: **else** ▷ G_i with n-bit input wire a and output ▷ size m of the function G_i realizes
9: $W_i^{0^m} \leftarrow\!\!\$\ \{0,1\}^k$
10: $GC[i, \mathsf{lsb}_n(W_a^{0^n})] \leftarrow \mathcal{H}(W_a^{0^n}, i) \oplus W_i^{0^m}$
11: **for** $x \neq 0^n \in \{0,1\}^n$ **do**
12: $GC[i, \mathsf{lsb}_n(W_a^{0^n}) \oplus x]$ $\leftarrow \mathsf{Rand}_n(W_a^{0^n}, i, x, 0^m) \oplus W_i^{0^m}$
13: **for** $i \in \mathsf{Outputs}$ **do**
14: $d_i \leftarrow \mathsf{lsb}_n(W_i^{0^n}) \oplus y_i$
15: **return** GC, X, d

Figure 3: Simulator $\mathcal{S}$.

wires be called the active path, i.e., for input wires, the active labels are retrieved via OT, for gate outputs, the active wires are retrieved by decrypting the row denoted by the point-and-permute bits.

The idea of the simulator is to produce a garbled circuit with a fixed active path. The simulator chooses the wire labels such that

- the garbled input X that is handed to the adversary corresponds to 0^p;
- the active label on each gate's output wire that the adversary obtains if they choose to evaluate the circuit with X is W^{0^n} (see Line 10 in Fig. 3).

The simulator adapts the decoding information s.t. if the garbled output is W^{0^n}, the expected output y is decoded.

$\mathcal{S} \approx \mathcal{G}_1$**.** Hybrid $\mathcal{G}_1$ (see Fig. 4) describes the simulator from the perspective of the evaluator. Let x be the input that the adversary chooses in the game. We view x as a black box as it is unknown. Suppose we evaluated the circuit on x in plaintext. We denote v_i as the active value on wire i. Instead of fixing the active path on labels W^{0^n}, we fix it on W^{v_i}.
The output values GC, d and the outputs of $\mathcal{S}$ are identically distributed as W^{0^n} and W^{v_i} are both distributed uniformly at random. Further, the change of input arguments,

$$\mathsf{Rand}_n(W_a^{0^n}, i, x, 0^m) \approx \mathsf{Rand}(W_a^{v_a}, i, v_a \oplus x, \phi(v_a \oplus x)) \ ,$$

does not change the distribution since all inputs $(x, 0^m)$, $\forall x \in \{0,1\}^n \neq 0^n$ and $(v_a \oplus x, \phi(v_a \oplus x))$, $\forall x \in \{0,1\}^n \neq v_a$, respectively, are unique and therefore amount to fresh randomness from the oracle, irrespective of ϕ.

$\mathcal{G}_1 \approx \mathcal{G}_2$**.** In hybrid $\mathcal{G}_2$, we replace Rand by the real construction $\mathcal{H}(m \oplus \mathbf{a} \cdot \mathbf{R}, t) \oplus \mathbf{b} \cdot \mathbf{R}$. This change is indistinguishable for the adversary by the definition of the n-TCCR secure function $\mathcal{H}$ (see Definition 4)

$$\mathsf{Rand}(W_a^{v_a}, i, v_a \oplus x, \phi(v_a \oplus x)) \approx \mathcal{H}(W_a^{v_a} \oplus (v_a \oplus x) \cdot \mathbf{R}, i) \oplus \phi(v_a \oplus x) \cdot \mathbf{R} \, .$$

$\mathcal{G}_2 \approx \mathcal{G}_3$**.** In hybrid $\mathcal{G}_3$ (see Fig. 5), we no longer compute the wire values v_i explicitly from the black-box input x. We fix an encoding for v_i, namely $v_i = 0^n$.
For the input wires, note that $x_i = v_i$ by definition of EVALWIRES, so $X_i \leftarrow W_i^{x_i}$ instead of $W_i^{v_i}$.
Further, the ciphertext indexing $GC[i, \cdot]$ (Line 14 in Fig. 5) is identical after the re-write. In $\mathcal{G}_2$,

$$\mathsf{lsb}_n(W_a^{v_a}) \oplus x = \mathsf{lsb}_n(W_a^{0^n}) \oplus x \, ,$$

and in $\mathcal{G}_3$,

$$\mathsf{lsb}_n(W_a^x) = \mathsf{lsb}_n(W_a^{0^n}) \oplus \mathsf{lsb}_n(x \cdot \mathbf{R}_n) = \mathsf{lsb}_n(W_a^{0^n}) \oplus x$$

by definition of $\mathbf{R}_n$. In the output of all gates G_i, we now maintain the invariant with $x \in \{0,1\}^n$

$$W_i^{v_i} \text{ becomes } W_i^{0^n} \, ,$$
$$W_i^{v_i} \oplus (v_i \oplus x) \cdot \mathbf{R}_{\ell(i)} \text{ becomes } W_i^{0^n} \oplus x \cdot \mathbf{R}_{\ell(i)} \, .$$

1: **function** EVALWIRES(f, x)
2: **for** $i \in \mathsf{Inputs}$ **do**
3: $v_i \leftarrow x_i$
4: **for** $\mathsf{G}_i \in \mathsf{Gates}$ **do**
5: $a, b \leftarrow in(\mathsf{G}_i)$
6: **if** $\mathsf{G}_i = XOR$ **then** ▷ G_i with n-bit input wires a, b
7: $v_i \leftarrow v_a \oplus v_b$
8: **else** ▷ G_i with n-bit input wire a
9: $v_i \leftarrow \phi(v_a)$ ▷ and $\phi : \{0,1\}^n \mapsto \{0,1\}^m$
10: **return** v

11: **function** $\mathcal{G}_1(f, x)$
12: $v \leftarrow \mathsf{EvalWires}(f, x)$
13: **for** $i \in \mathsf{Inputs}$ **do**
14: $W_i^{\boxed{v_i}} \leftarrow\!\!\$\ \{0,1\}^k$
15: $X_i \leftarrow W_i^{\boxed{v_i}}$
16: **for** $\mathsf{G}_i \in \mathsf{Gates}$ **do**
17: **if** $\mathsf{G}_i = XOR$ **then** ▷ G_i with n-bit input wires a, b
18: $W_i^{\boxed{v_i}} \leftarrow W_a^{\boxed{v_a}} \oplus W_b^{\boxed{v_b}}$
19: **else** ▷ G_i with n-bit input wire a
20: $W_i^{\boxed{v_i}} \leftarrow\!\!\$\ \{0,1\}^k$ ▷ and $\phi : \{0,1\}^n \mapsto \{0,1\}^m$
21: $GC[i, \mathsf{lsb}_n(W_a^{\boxed{v_a}})] \leftarrow \mathcal{H}(W_a^{\boxed{v_a}}, i) \oplus W_i^{\boxed{v_i}}$
22: **for** $x \neq \boxed{v_a} \in \{0,1\}^n$ **do**
23: $GC[i, \mathsf{lsb}_n(\boxed{W_a^{v_a}}) \oplus x]$
24: $\leftarrow \mathsf{Rand}(W_a^{\boxed{v_a}}, i, \boxed{v_a \oplus x}, \boxed{\phi(v_a \oplus x)}) \oplus W_i^{\boxed{v_i}}$
25: **for** $i \in \mathsf{Outputs}$ **do**
26: $d_i \leftarrow \mathsf{lsb}_n(W_i^{\boxed{v_i}}) \oplus y_i$
27: **return** GC, X, d

Figure 4: Hybrid $\mathcal{G}_1$. The simulator from the perspective of the evaluator where x is a black box value. Values in a box $\boxed{v_i}$ highlight the difference between $\mathcal{S}$ and $\mathcal{G}_1$.

And for the decoding information, first note that for $i \in$ Outputs $v_i = y_i$, thus in $\mathcal{G}_2$ (We denote $\ell(i) = n$),

$$d_i \oplus \mathsf{lsb}_n(W_i^{v_i}) = \mathsf{lsb}_n(W_i^{v_i}) \oplus \mathsf{lsb}_n(W_i^{v_i}) \oplus y_i = y_i$$

and in $\mathcal{G}_3$:

$$\begin{aligned} d_i \oplus \mathsf{lsb}_n(W_i^{y_i}) &= \mathsf{lsb}_n(W_i^{0^n}) \oplus \mathsf{lsb}_n(W_i^{y_i}) \\ &= \mathsf{lsb}_n(W_i^{0^n}) \oplus \mathsf{lsb}_n(W_i^{0^n}) \oplus y_i \\ &= y_i \,. \end{aligned}$$

The decoding information in $\mathcal{G}_2$ and $\mathcal{G}_3$ yield correct results when used with their respective garbled inputs. $d_{\mathcal{G}_2}$ and $d_{\mathcal{G}_3}$ are both uniformly distributed as $\mathsf{lsb}_{\ell(i)}(W_i^{vi})$ resp. $\mathsf{lsb}_{\ell(i)}(W_i^{0^{\ell(i)}})$ are distributed at random. So $d_{\mathcal{G}_2}$ and $d_{\mathcal{G}_3}$ remain indistinguishable.

We conclude the proof by noting that $\mathcal{G}_3$ and Garble yield *identical* outputs in the prv.sim game. This can easily be seen when the exceptional case for $x = 0^n$ (Line 21 in Fig. 5) in the projection gates part is incorporated into the loop and the computation of d is re-written, $\mathcal{G}_3$ is a description of the Garble function. □

1: **function** $\mathcal{G}_3(f, x)$
2: $\mathbf{R}_1, \ldots, \mathbf{R}_{\bar{n}} \leftarrow GenR(\bar{n})$
3: **for** $i \in$ Inputs **do**
4: $W_i^{\boxed{0^{\ell(i)}}} \leftarrow\!\$\ \{0,1\}^k$
5: $X_i \leftarrow W_i^{\boxed{x_i}}$
6: **for** $\mathsf{G}_i \in$ Gates **do**
7: $a, b \leftarrow in(\mathsf{G}_i)$
8: **if** $\mathsf{G}_i = XOR$ **then** ▷ G_i with n-bit input wires a, b
9: $W_i^{\boxed{0^n}} \leftarrow W_a^{\boxed{0^n}} \oplus W_b^{\boxed{0^n}}$
10: **else** ▷ G_i with n-bit input wire a ▷ and $\phi : \{0,1\}^n \mapsto \{0,1\}^m$
11: $W_i^{\boxed{0^n}} \leftarrow\!\$\ \{0,1\}^k$
12: $GC[i, \mathsf{lsb}_n(W_a^{\boxed{0^n}})] \leftarrow \mathcal{H}(W_a^{\boxed{0^n}}, i) \oplus W_i^{\boxed{\phi(0^n)}}$
13: **for** $x \neq 0^n \in \{0,1\}^n$ **do**
14: $GC[i, \boxed{\mathsf{lsb}_n(W_a^x)}]$
$\leftarrow \mathcal{H}(\boxed{W_a^x}, i) \oplus \phi(\boxed{x}) \cdot \mathbf{R}_n \oplus W_i^{\boxed{0^m}}$
15: **for** $i \in$ Outputs **do**
16: $d_i \leftarrow \mathsf{lsb}_n(W_i^{\boxed{0^n}})$
17: **return** GC, X, d

Figure 5: Hybrid $\mathcal{G}_3$. We fix the encoding of $W_i^{v_i}$ to $W_i^{0^n}$. Values in a box $\boxed{0^n}$ highlight the difference between $\mathcal{G}_2$ and $\mathcal{G}_3$.

C. Obliviousness

The notion of obv.sim expresses that the adversary cannot learn any information given the garbled circuit GC and all input wire labels X. Unlike the privacy notion, the adversary does not have access to the decoding information d.

Theorem 2. *Given a $\bar{n}$-TCCR secure hash function $\mathcal{H}$ and $\bar{n} \ll \kappa$, the garbling scheme Π is obv.sim secure.*

Proof. Let $\mathcal{S}_{\text{auth}}$ be $\mathcal{S}$ from Fig. 3 with the lines 13-14 removed. Then we note that the computation of GC and X doesn't depend on y, neither in $\mathcal{S}_{\text{auth}}$ nor in one of the hybrids $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$. We can thus use the same reasoning as for prv.sim security, omitting parts that correspond to y or d. □

D. Authenticity

Authenticity states that an adversary cannot forge wire labels that are not obtained through evaluating the garbled circuit. Clearly, the presented scheme does not satisfy this property as any wire label is decoded to output bits. If authenticity is desired, we modify the decoding information d to list hashes of all output wire labels and associations to their semantic meaning. As in [48], the decoding function checks if the presented wire is indeed in the list d.

VI. Evaluation of SPN Primitives

In the following, we discuss how SPN primitives with a specific structure can be implemented with our new garbling scheme and how this improves over the state-of-the-art. Note that we don't intend to compare the performance of the primitives among each other in MPC protocols. Instead, we focus on how each primitive can be accelerated. Consequently, we will not consider other traditional or MPC-friendly primitives. We compare the state-of-the-art garbling schemes Half-Gates by Zahur, Rosulek and Evans [6], which we abbreviate ZRE15, as well as the work of Rosulek and Roy [7], abbreviated RR21. Both schemes support free XOR gates and AND gates on wires holding one bit.

In SPN-based primitives, a state is updated with a round function consisting of a substitution layer, a permutation layer, a round constant and/or (round) key addition layer. SPNs are commonly used to construct block ciphers and pseudo-random permutations used, e.g., in hash or MAC functions.

We show an efficient circuit representation with projection gates for primitives that satisfy the following conditions for state and round function parts.

- **State.** The state is (conceptually) split into n-bit cells.
- **Substitution Layer.** The substitution layer consists of S-boxes that are applied to each cell.
- **Permutation Layer.** The permutation layer can be described by a permutation on the cells and/or by a mixing matrix which encodes a fixed matrix multiplication with the state. In this paper, we focus on primitives with a binary mixing matrix.
- **Round Constant/(Round) Key Addition Layer.** The round constant or (round) key is XORed cell-wise.

With this structure, we set $\bar{n} = n$ and implement a single cell as n-bit wire. Each S-box in the substitution layer is replaced with an n-bit projection gate computing the same functionality. The permutation layer and the addition layer are expressible with XOR gates only.

We identified nine SPN primitives in the literature that fulfill the conditions. Since the studied primitives have at most 8-bit cells, we set $\bar{n} = 8$.

A. Implementation Details

For the projection gates implementation, we assume that the input block is already setup in n-bit wires where n denotes the cell size in bits. This doesn't incur additional cost since the input phase using OT can already share wire labels with the desired wire label offset, as detailed in Sect. IV-D. For the implementation using AND gates, only the S-box costs AND gates in the data path of the primitive. We selected implementations for the S-boxes with the lowest number of AND gates since their cost dominates in Half-Gates and RR21. Table V details the number of projection and AND gates for each primitive.

For the 4-bit S-box used in TWINE-80 and TWINE-128 and for the 4-bit S-box used in Midori64, MANTIS and CRAFT, we found new circuits using the smallest number of AND gates so far, reducing the number of AND gates for the TWINE S-box from 7 to 6 and for the Midori64 S-box from 8 to 4. We used the heuristic optimization tool LIGHTER by Jean et al. [54] operating on a customized cost metric, for more details see Appendix A.

For the key, we assume individual key bits to be available in 1-bit wires as this eases key scheduling in many cases. Note that the cost to transform the (round) key bits into n-bit wires is taken into account. In scenarios where one party knows the complete key, e.g., to offer blind symmetric encryption or decryption where the encryption or decryption is performed without learning the message and ciphertext, the key schedule does not need to be computed within the garbling scheme. Instead, if the garbler knows the key, they can compute the key schedule separately and insert the round keys as secret constants. Similarly, if the evaluator knows the key, they may receive the wire labels for round keys via OT instead.

If the key is shared among the players using a linear secret-sharing scheme, for instance as $k = k_G \oplus k_E$ where k_G is the garbler's share and k_E is the evaluator's share, the key schedule can be computed outside of the garbling scheme by each player on their share instead for ciphers with a linear key schedule, e.g., for Piccolo, Midori, SKINNY, MANTIS and CRAFT. The resulting round key shares can then be treated as input and are recombined using only linear operations saving any gates specified in the key schedule column for the cipher. However, the gate counts presented here compute the entire key schedule of the primitive which is required in the distributed encryption/decryption scenario.

B. Performance

The gate counts from Table V can be turned into calls to $\mathcal{H}$ and sent ciphertexts. In ZRE15, each AND gate costs 4 calls to $\mathcal{H}$ for garbling, 2 ciphertexts are sent, and 2 calls to $\mathcal{H}$ for evaluation. In RR21, each AND gate costs 6 calls to $\mathcal{H}$ for garbling, 1.5 ciphertexts are sent, and 3 calls to $\mathcal{H}$ for evaluation.

Table VI lists all studied primitives with the corresponding trade-off in garbling and communication cost, and evaluation improvement measured in the number of calls to $\mathcal{H}$ and in the number of ciphertexts, respectively. We found three primitives in five configurations in total where our scheme improves in

Table V: Detailed gate counts for setup, key schedule and data path of the selected symmetric primitives. The top entry denotes the number of AND gates while the bottom entry denotes the number of projection gates.

† Gate counts obtained from Mandal et al. [55].

Primitive	Setup	Key Schedule	Data Path
AES-128 [26]		1280 AND 128 1-bit + 49 8-bit	5120 AND 320 8-bit
CRAFT [25]		192 1-bit	1920 AND 480 4-bit
Fides-80 [27]	160 1-bit		320 AND 32 5-bit
Fides-96	192 1-bit		1088 AND 32 6-bit
MANTIS [24]		192 1-bit	896 AND 224 4-bit
Midori64 [23]		128 1-bit	1024 AND 256 4-bit
Piccolo-80 [29]		80 1-bit	1600 AND 600 4-bit
Piccolo-128		128 1-bit	1984 AND 744 4-bit
SKINNY-64-128		128 1-bit + 280 4-bit	2304 AND 576 4-bit
TWINE-80 [22]		432 AND 80 1-bit + 70 4-bit	1728 AND 288 4-bit
TWINE-128		630 AND 128 1-bit + 104 4-bit	1728 AND 288 4-bit
WAGE [28]	259 1-bit		37745 AND† 777 7-bit

Table VI: Estimated performance difference for selected symmetric ciphers. The notation $\times x$ denotes an improvement by factor x in the category with respect to the base scheme, i.e., $x > 1$ is an improvement, $x < 1$ is degradation.

Base Scheme	Primitive	Garble	Send	Eval
ZRE15 [6] RR21 [7]	AES-128 [26]	$\times 0.28$ $\times 0.42$	$\times 0.14$ $\times 0.10$	$\times 26.23$ $\times 39.34$
ZRE15 RR21	CRAFT [25]	$\times 0.95$ $\times 1.43$	$\times 0.52$ $\times 0.39$	$\times 5.71$ $\times 8.57$
ZRE15 RR21	Fides-80 [27]	$\times 1.23$ $\times 1.84$	$\times 0.64$ $\times 0.48$	$\times 15.45$ $\times 23.18$
ZRE15 RR21	Fides-96	$\times 2.10$ $\times 3.15$	$\times 1.07$ $\times 0.81$	$\times 50.26$ $\times 75.39$
ZRE15 RR21	MANTIS [24]	$\times 0.90$ $\times 1.35$	$\times 0.50$ $\times 0.38$	$\times 4.31$ $\times 6.46$
ZRE15 RR21	Midori64 [23]	$\times 0.94$ $\times 1.41$	$\times 0.52$ $\times 0.39$	$\times 5.33$ $\times 8.00$
ZRE15 RR21	Piccolo-80 [29]	$\times 0.66$ $\times 0.98$	$\times 0.35$ $\times 0.26$	$\times 4.71$ $\times 7.06$
ZRE15 RR21	Piccolo-128	$\times 0.65$ $\times 0.98$	$\times 0.35$ $\times 0.26$	$\times 4.55$ $\times 6.83$
ZRE15 RR21	SKINNY-64-128	$\times 0.66$ $\times 0.99$	$\times 0.36$ $\times 0.27$	$\times 4.68$ $\times 7.02$
ZRE15 RR21	TWINE-80 [22]	$\times 1.46$ $\times 2.19$	$\times 0.79$ $\times 0.59$	$\times 9.81$ $\times 14.71$
ZRE15 RR21	TWINE-128	$\times 1.44$ $\times 2.16$	$\times 0.78$ $\times 0.59$	$\times 9.05$ $\times 13.58$
ZRE15 RR21	WAGE [28]	$\times 1.51$ $\times 2.27$	$\times 0.76$ $\times 0.57$	$\times 72.87$ $\times 109.30$

Table VII: Performance benchmark results for some SPN-ciphers comparing garbling and evaluation time as well as the circuit size. All reported numbers are amortized from 500 (for SKINNY-128-*) and 1000 parallel primitive calls averaged over 10 repetitions.

Base Scheme	Primitive	Garble in ms	Circuit size in KB	Eval in ms
ZRE15 [6]		0.767	204.93	0.722
RR21 [7]	AES-128 [26]	0.436	156.51	0.305
This work		0.928	1242.55	0.016
ZRE15		0.093	32.84	0.070
RR21	MANTIS_7 [24]	0.083	30.96	0.077
This work		0.133	76.36	0.040
ZRE15		0.283	73.80	0.194
RR21	SKINNY-64-128	0.153	59.13	0.096
This work		0.289	139.30	0.026
ZRE15		0.343	81.99	0.246
RR21	SKINNY-64-192	0.166	67.11	0.118
This work		0.321	154.69	0.041
ZRE15		0.595	163.98	0.440
RR21	SKINNY-128-128	0.346	126.66	0.279
This work		2.281	2613.28	0.015
ZRE15		0.803	196.74	0.594
RR21	SKINNY-128-256	0.442	154.71	0.348
This work		2.563	3135.62	0.028
ZRE15		1.107	229.51	0.819
RR21	SKINNY-128-384	0.558	182.77	0.472
This work		2.841	3658.00	0.041
ZRE15		0.202	75.52	0.168
RR21	TWINE-128 [22]	0.136	60.40	0.081
This work		0.191	108.21	0.059
ZRE15		0.187	68.80	0.153
RR21	TWINE-80	0.128	53.99	0.074
This work		0.199	99.04	0.045

both garbling and evaluation cost over both reference garbling schemes. In the remaining primitives and cases, projection gates trade off higher garbling and communication cost for faster evaluation performance. Note that for most primitives, the evaluation improvement is much higher than the additional communication cost. E.g., for Midori64, at a cost of slightly more garbling work ($\approx 6\%$ more) and less than twice the number of sent ciphertexts, we improve the evaluation work by a factor of five. We detail the implementation approach with projection gates for the ciphers in Appendix B.

Next, we experimentally compared the performance of four primitives in nine configurations in ZRE15, RR21 and our scheme. RR21 has been implemented by Hamacher et al. [56] in the MOTION framework while ZRE15 and our scheme have been implemented in MP-SPDZ [57]. Table VII lists the garbling and evaluation time, and the circuit size. We achieve a considerable speed-up in evaluation time of, e.g., factor 20 to 45 for AES. In general the expected trade-off of faster evaluation and larger circuit size is immediate for all implemented ciphers. However, we observed differences in garbling and evaluation time between ZRE15 and RR21 executions of the same circuit which cannot be explained by the differing number of hash function calls. We believe the observations are due to the implementation in the two MPC frameworks which have differing overhead.

Besides oblivious computation of SPN primitives, statements where a prover proves knowledge of a key k to a pair x, y s.t. $AES_k(x) = y$ are highly relevant. Garbling schemes have been used to construct efficient interactive zero-knowledge protocols that prove statements over "unstructured" languages expressible in Boolean circuits [58], [59]. Using our garbling scheme, proving statements involving SPN primitives would be much faster since proving equates to evaluating the garbled circuit. This is traded-off with a larger proof size.

VII. Conclusion

We presented a garbling scheme that encodes n-bit strings per wire. It generalizes the idea of FreeXOR and integrates seamlessly into state-of-the-art schemes with FreeXOR on the 1-bit wire level. Projection gates can be used to convert strings from n- to m-bit or to compute arbitrary n- to m-bit functions, while XOR is free. We prove the scheme secure under the assumption of a n-TCCR secure hash function, a generalization of TCCR security.

For an important application in two-party secure function evaluation, the evaluation of symmetric primitives, we show that substitution-permutation network primitives with certain structure can be efficiently implemented in our scheme. Compared to AND gate-based circuits, we show a high-speed evaluation that is traded off with moderate additional garbling or communication cost. In scenarios where the garbling scheme runs in an offline/online setting, we shift the garbling work and garbled circuit transfer to the evaluator into the pre-processing phase and thus obtain a high-speed online phase. We obtained a considerable performance improvement, a 4- to 72-times faster online phase, for nine primitives in literature when taking hash function calls as a metric. Implementation of some ciphers shows that this evaluation performance improvement translates into practical applications.

Appendix A
Formulas for S-Boxes of TWINE and Midori64

We use the heuristic optimization tool LIGHTER by Jean et al. [54] operating on a customized cost metric. We restrict the tool to use only NOT, AND and XOR gates with the associated costs of 0.01, 1 and 0.01, respectively. These costs describe our setting where NOT and XOR gates are practically free, i.e., very low cost, and AND gates are expensive, i.e., high cost[4]. The tool then searches an implementation with low total cost following a heuristic. This approach reduces the number of AND gates for the TWINE S-box from 7 AND gates (algebraic normal form) to 6 AND gates (see Fig. 6a). For the Midori64 S-box, the number of AND gates is reduced from 8 AND gates (formula given in the specification [23]) to 4 AND gates (see Fig. 6b).

Appendix B
Implementation of SPN Primitives

In the following, we give a more detailed explanation of the implementation from Tables V and VI for each primitive.

[4]Essentially, this implies that we prefer implementations using 99 NOT or XOR gates in addition to x AND gates to an implementation using $x + 1$ AND gates.

$$
\begin{aligned}
a &\leftarrow x_2 \oplus x_3 \\
b &\leftarrow x_3 \oplus (\neg x_0 \wedge x_1) \\
c &\leftarrow \neg x_0 \oplus a \oplus (x_1 \wedge a \wedge b) \\
d &\leftarrow a \oplus (b \wedge c) \\
e &\leftarrow b \oplus c \\
f &\leftarrow c \oplus d \\
x'_3 &\leftarrow x_1 \oplus e \\
x'_2 &\leftarrow e \oplus (d \wedge x'_3) \\
x'_0 &\leftarrow c \oplus (f \wedge x'_2) \\
x'_1 &\leftarrow f \oplus x'_3
\end{aligned}
$$

(a) The 4-bit S-box of TWINE [22] computed using 6 AND gates.

$$
\begin{aligned}
a &\leftarrow \neg(x_0 \oplus x_2) \\
b &\leftarrow x_0 \oplus (a \wedge x_3) \\
c &\leftarrow x_1 \oplus b \\
d &\leftarrow \neg x_3 \oplus (a \wedge b) \\
x'_0 &\leftarrow b \oplus (c \wedge d) \\
e &\leftarrow d \oplus x'_0 \\
x'_1 &\leftarrow a \oplus d \\
x'_2 &\leftarrow c \oplus e \\
x'_3 &\leftarrow e \oplus (x'_0 \wedge x'_2)
\end{aligned}
$$

(b) The 4-bit S-box Sb_0 of Midori64 computed using 4 AND gates.

Figure 6: Implementation formulas for the TWINE and Midori64 S-boxes. The input bits are x_0 through x_3, the output bits are x'_0 through x'_3.

A. AES

The key schedule of AES-128 applies 4 S-boxes per round to the state. All remaining key schedule operations can be expressed using XOR gates. The AES S-box can be computed with 32 AND gates, as described by Boyar and Peralta [60]. In the data path, 16 S-boxes are applied per round. The ShiftRows, MixColumns and AddRoundKey steps can be expressed with XOR gates. AES-128 defines 10 rounds.

For an implementation using projection gates, we first compose the key into 8-bit wires. Then, the key schedule can be computed by replacing the S-box with a single 8-bit projection gate computing the same functionality. For the data path, we replace S-boxes with 8-bit projection gates. The mixing step in AES cannot be described with a binary matrix alone but we re-write the MixColumns step as

$$
\begin{pmatrix} 2311 \\ 1231 \\ 1123 \\ 3112 \end{pmatrix}
\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} =
$$

$$
\begin{pmatrix} 0111 \\ 1011 \\ 1101 \\ 1110 \end{pmatrix}
\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix} \oplus
\begin{pmatrix} 1100 \\ 0110 \\ 0011 \\ 1001 \end{pmatrix}
\begin{pmatrix} f(s_0) & f(s_4) & f(s_8) & f(s_{12}) \\ f(s_1) & f(s_5) & f(s_9) & f(s_{13}) \\ f(s_2) & f(s_6) & f(s_{10}) & f(s_{14}) \\ f(s_3) & f(s_7) & f(s_{11}) & f(s_{15}) \end{pmatrix}
$$

where $s_0, \dots s_{15}$ are the 8-bit cells of the state and $f(s) = 2s$ computes the finite field doubling in $\mathbb{GF}(2^8)$ defined for AES. Therefore, we compute a round of AES with $2 \cdot 16$ 8-bit projection gates. This yields a correct result, since $s \oplus f(s) = 3s$ in $\mathbb{GF}(2^8)$.

B. CRAFT

The key and tweak bits are first composed into 4-bit wires. The remaining key schedule is linear w.r.t. 4-bit wires.

The data path is linear except for the 16 S-boxes that are applied in each of the 30 rounds. CRAFT uses the Midori Sb_0 S-box which can be computed with 4 AND gates (see Fig. 6b), or one 4-bit projection gate.

C. Fides

The internal state of Fides is a 4×8 grid of 5-bit and 6-bit cells for Fides-80 and Fides-96, respectively. We can compute the 5-bit S-box with 10 AND gates (see Fig. 7), or one 5-bit projection gate. The 6-bit S-box may be computed with 34 AND gates expressing each output bit in algebraic normal form. This approach doesn't aim to optimise the number of AND gates used. However, we count common terms from different output bits only once since they can be shared as intermediate results. In our garbling scheme, the S-box is expressed in one 6-bit projection gate.

$$
\begin{aligned}
a &\leftarrow x_0 \wedge x_2 \\
b &\leftarrow x_1 \wedge x_4 \quad x'_0 \leftarrow \neg(x_0 \oplus x_3 \oplus b \oplus a) \\
c &\leftarrow x_2 \wedge x_3 \quad x'_1 \leftarrow x_4 \oplus c \oplus d \oplus e \oplus (x_0 \wedge x_1) \\
d &\leftarrow x_0 \wedge x_4 \quad x'_2 \leftarrow x_3 \oplus x_4 \oplus a \oplus d \oplus f \oplus (x_3 \wedge x_4) \\
e &\leftarrow x_2 \wedge x_4 \quad x'_3 \leftarrow x_1 \oplus x_4 \oplus a \oplus c \oplus f \oplus (x_1 \wedge x_3) \\
f &\leftarrow x_1 \wedge x_2 \quad x'_4 \leftarrow x_1 \oplus x_2 \oplus x_3 \oplus b \oplus e \oplus f \oplus (x_0 \wedge x_3)
\end{aligned}
$$

Figure 7: The 5-bit S-box of Fides [27] can be computed with 10 AND gates. Input bits are $x_0, \dots, x_4$, output bits are $x'_0, \dots, x'_4$.

D. MANTIS

The key $k = k_0 || k_1$ is expanded as defined in [24]:

$$k_0 || (k_0 >>> 1) \oplus (k_0 >> 63) || k_1 \,.$$

Afterwards we compose the required 4-bit wires for the expanded key costing 192 1-bit projection gates. MANTIS uses the Midori Sb_0 S-box, which can be computed with 4 AND gates (see Fig. 6b), or one 4-bit projection gate.

E. Midori64

The key bits are first composed into 4-bit wires. The key schedule can then be computed using XOR gates between the 4-bit wires.

In the data path, all steps except for the S-box can be computed with XOR gates alone. The 4-bit S-box Sb_0 can be computed with 4 AND gates (see Fig. 6b), or one 4-bit projection gate.

F. Piccolo

The key schedule for Piccolo-80 and Piccolo-128 can be computed using only XOR gates after the key bits are composed to 4-bit wires.

Piccolo's data path applies the 16-bit function F two times per round to half of the state. This function F is composed of a parallel application of 4 4-bit S-boxes, followed by a mixing matrix multiplication, followed by another parallel application of 4 4-bit S-boxes.

$$F(s_0, s_1, s_2, s_3) = \mathcal{S}\left(\begin{pmatrix} 2311 \\ 1231 \\ 1123 \\ 3112 \end{pmatrix} \mathcal{S}\left(\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}\right)\right) \,,$$

where the function $\mathcal{S}$ applies the 4-bit S-box S element-wise

$$\mathcal{S}\left(\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}\right) = \begin{pmatrix} S(s_0) \\ S(s_1) \\ S(s_2) \\ S(s_3) \end{pmatrix}.$$

The mixing matrix encodes multiplications with elements in the finite field $\mathbb{GF}(2^4)$ with the irreducible polynomial $x^4 + x + 1$. Clearly, Piccolo doesn't have the property of a binary mixing matrix. However, we can still provide an implementation with projection gates at additional cost.

We re-write the function F as

$$F'(s_0, s_1, s_2, s_3) = \mathcal{S}\left(\begin{pmatrix} 0111 \\ 1011 \\ 1101 \\ 1110 \end{pmatrix}\begin{pmatrix} f(s_0) \\ f(s_1) \\ f(s_2) \\ f(s_3) \end{pmatrix} \oplus \begin{pmatrix} 1100 \\ 0110 \\ 0011 \\ 1001 \end{pmatrix}\begin{pmatrix} g(s_0) \\ g(s_1) \\ g(s_2) \\ g(s_3) \end{pmatrix}\right)$$

where $f(s) = S(s)$ and $g(s) = 2S(s)$. Subsequently, we compute f, g and the remaining S-box layer $\mathcal{S}$ via 4-bit projection gates. Therefore, F' can be computed with $4+4+4 = 12$ 4-bit projection gates. This re-writing is correct because $f(s) \oplus g(s) = 3S(s)$ w.r.t $\mathbb{GF}(2^4)$.

G. SKINNY

The SKINNY cipher family comprises three tweakey sizes, 64, 128 and 192 bit, of which we include the size 128-bit here. The key schedule for SKINNY-64-128 also includes the application of a linear feedback shift register (LFSR) to 8 per round. This LFSR is implemented with a 4-bit projection gate.

The SKINNY data path contains 16 4-bit S-boxes per round. Each S-box is implemented with 4 AND gates using the formula from [24], or one 4-bit projection gate.

H. TWINE

The key bits are first composed into 4-bit wires. The key schedule is linear except for 2 and 3 S-box computations per round for TWINE-80 and TWINE-128, respectively. In total, the key schedule comprises 35 rounds with S-box computation for both TWINE-80 and TWINE-128.

The data path is the same for TWINE-80 and TWINE-128 and contains 8 S-boxes per round in 36 rounds. The S-box can be computed with 6 AND gates (see Fig. 6a), or one 4-bit projection gate.

I. WAGE

The internal state of the WAGE permutation is represented as 37 7-bit cells. We load the initial state by computing the 7-bit wire composition for all bits.

We write s_i to denote the i-th 7-bit cell and s'_i to denote the updated i-th 7-bit cell. The internal state is updated 111 times in the following procedure:

$$\begin{aligned}
\mathsf{fb} \leftarrow\ & \mathsf{WGP}(s_{36}) \oplus s_{31} \oplus s_{30} \oplus s_{26} \oplus s_{24} \oplus s_{19} \oplus s_{13} \oplus s_{12} \\
& \oplus s_8 \oplus s_6 \oplus \mathsf{Dbl}(s_0) \\
s_5 \leftarrow\ & s_5 \oplus \mathsf{SB}(s_8) \\
s_{11} \leftarrow\ & s_{11} \oplus \mathsf{SB}(s_{15}) \\
s_{19} \leftarrow\ & s_{19} \oplus \mathsf{WGP}(s_{18}) \oplus rc_0 \\
s_{24} \leftarrow\ & s_{24} \oplus \mathsf{SB}(s_{27}) \\
s_{30} \leftarrow\ & s_{30} \oplus \mathsf{SB}(s_{34}) \\
s'_j \leftarrow\ & s_{j+1},\ 0 \le j \le 35 \\
s_{36} \leftarrow\ & \mathsf{fb}\ .
\end{aligned}$$

The 7-bit functions WGP, Dbl and SB denote a Welch-Gong permutation, finite field doubling and a lightweight 7-bit S-box. All three are implemented using a 7-bit projection gate. Further, rc_0 is a round-dependent constant.

REFERENCES

[1] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, "Deriving genomic diagnoses without revealing patient genomes," *Science*, vol. 357, no. 6352, pp. 692–695, 2017.

[2] T. Gupta, H. Fingler, L. Alvisi, and M. Walfish, "Pretzel: Email encryption and provider-supplied functions are compatible," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 169–182.

[3] D. Chen, W. Chen, J. Chen, P. Zheng, and J. Huang, "Edge detection and image segmentation on encrypted image with homomorphic encryption and garbled circuit," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2018, pp. 1–6.

[4] H.-J. Kim, H.-I. Kim, and J.-W. Chang, "A privacy-preserving kNN classification algorithm using yao's garbled circuit on cloud computing," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 766–769.

[5] A. C.-C. Yao, "How to Generate and Exchange Secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, ser. SFCS '86. USA: IEEE Computer Society, 1986, p. 162–167.

[6] S. Zahur, M. Rosulek, and D. Evans, "Two Halves Make a Whole," in *Advances in Cryptology - EUROCRYPT 2015*. Berlin, Heidelberg: Springer, 2015, pp. 220–250.

[7] M. Rosulek and L. Roy, "Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits," in *Advances in Cryptology – CRYPTO 2021*. Springer, 2021, pp. 94–124.

[8] V. Kolesnikov and T. Schneider, "Improved Garbled Circuit: Free XOR Gates and Applications," in *Automata, Languages and Programming*. Berlin, Heidelberg: Springer, 2008, pp. 486–498.

[9] J. B. Nielsen and C. Orlandi, "LEGO for two-party secure computation," in *Theory of Cryptography*. Berlin, Heidelberg: Springer, 2009, pp. 368–386.

[10] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," *Journal of Cryptology*, vol. 25, no. 4, pp. 680–722, 2012.

[11] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *Advances in Cryptology – CRYPTO 2013*. Berlin, Heidelberg: Springer, 2013, pp. 18–35.

[12] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff, "Amortizing garbled circuits," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer, 2014, pp. 458–475.

[13] S. Jarecki and V. Shmatikov, "Efficient Two-Party Secure Computation on Committed Inputs," in *Advances in Cryptology - EUROCRYPT 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 97–114.

[14] A. shelat and C. hao Shen, "Two-output secure computation with malicious adversaries," in *Advances in Cryptology – EUROCRYPT 2011*, K. G. Paterson, Ed. Berlin, Heidelberg: Springer, 2011, pp. 386–405.

[15] X. Wang, S. Ranellucci, and J. Katz, "Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 21–37.

[16] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang, "Optimizing Authenticated Garbling for Faster Secure Two-Party Computation," in *Advances in Cryptology–CRYPTO 2018*. Springer, 2018, pp. 365–391.

[17] S. Dittmer, Y. Ishai, S. Lu, and R. Ostrovsky, "Authenticated Garbling from Simple Correlations," in *Advances in Cryptology - CRYPTO 2022*, ser. Lecture Notes in Computer Science, Y. Dodis and T. Shrimpton, Eds., vol. 13510. Springer, 2022, pp. 57–87.

[18] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: The garbled circuit approach," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 591–602.

[19] D. Beaver, S. Micali, and P. Rogaway, "The Round Complexity of Secure Protocols," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 503–513.

[20] M. Naor, B. Pinkas, and R. Sumner, "Privacy Preserving Auctions and Mechanism Design," in *Proceedings of the 1st ACM Conference on Electronic Commerce*, ser. EC '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 129–139.

[21] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou, "On the security of the "Free-XOR" technique," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, vol. 7194. Springer, 2012, pp. 39–53.

[22] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "Twine: A lightweight block cipher for multiple platforms," in *International Conference on Selected Areas in Cryptography*. Springer, 2012, pp. 339–354.

[23] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A block cipher for low energy," in *ASIACRYPT (2)*. Springer, 2015, pp. 411–436.

[24] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS," in *Advances in Cryptology – CRYPTO 2016*, M. Robshaw and J. Katz, Eds. Berlin, Heidelberg: Springer, 2016, pp. 123–153.

[25] C. Beierle, G. Leander, A. Moradi, and S. Rasoolzadeh, "Craft: lightweight tweakable block cipher with efficient protection against dfa attacks," *IACR Transactions on Symmetric Cryptology*, vol. 2019, no. 1, pp. 5–45, 2019.

[26] National Institute of Standards and Technology, "Specification for the ADVANCED ENCRYPTION STANDARD (AES)," Federal Information Processing Standards Publications 197, 2001.

[27] B. Bilgin, A. Bogdanov, M. Knežević, F. Mendel, and Q. Wang, "Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 142–158.

[28] R. AlTawy, G. Gong, K. Mandal, and R. Rohit, "Wage: An authenticated encryption with a twist," *IACR Transactions on Symmetric Cryptology*, pp. 132–159, 2020.

[29] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2011, pp. 342–357.

[30] B. Pinkas, T. Schneider, N. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer, 2009, pp. 250–267.

[31] V. Kolesnikov, P. Mohassel, and M. Rosulek, "FleXOR: Flexible garbling for XOR gates that beats Free-XOR," in *Advances in Cryptology – CRYPTO 2014*. Berlin, Heidelberg: Springer, 2014, pp. 440–457.

[32] A. Acharya, T. Ashur, E. Cohen, C. Hazay, and A. Yanai, "A New Approach to Garbled Circuits," in *Applied Cryptography and Network Security - 21st International Conference, ACNS 2023*, ser. Lecture Notes in Computer Science, M. Tibouchi and X. Wang, Eds., vol. 13906. Springer, 2023, pp. 611–641.

[33] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables." in *24. Network and Distributed System Security Symposium (NDSS'17)*. Internet Society, 2017.

[34] O. Goldreich, S. Micali, and A. Wigderson, "How to Play ANY Mental Game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229.

[35] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci, "The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds. Springer, 2017, pp. 167–187.

[36] I. Damgård and R. Zakarias, "Fast Oblivious AES A Dedicated Application of the MiniMac Protocol," in *Progress in Cryptology – AFRICACRYPT 2016*, D. Pointcheval, A. Nitaj, and T. Rachidi, Eds. Cham: Springer, 2016, pp. 245–264.

[37] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek, "Faster Secure Multi-party Computation of AES and DES Using Lookup Tables," in *Applied Cryptography and Network Security*. Springer, 2017.

[38] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 600–620.

[39] F. B. Durak and J. Guajardo, "Improving the Efficiency of AES Protocols in Multi-Party Computation," in *Financial Cryptography and Data Security*, N. Borisov and C. Diaz, Eds. Berlin, Heidelberg: Springer, 2021, pp. 229–248.

[40] A. Abidin, E. Pohle, and B. Preneel, "Arithmetic Circuit Implementations of S-boxes for SKINNY and PHOTON in MPC," in *To appear in Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security*, 2023.

[41] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay – Secure Two-Party Computation System," in *Proceedings of the 13th USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004.

[42] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: Tool for Automating Secure Two-Party Computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 451–462.

[43] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security Symposium*, vol. 201, 2011, pp. 331–335.

[44] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and secure multiparty computation from fixed-key block ciphers," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 825–841.

[45] Y. L. Chen and S. Tessaro, "Better security-efficiency trade-offs in permutation-based two-party computation," in *Advances in Cryptology – ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds. Cham: Springer, 2021, pp. 275–304.

[46] D. Heath and V. Kolesnikov, "One hot garbling," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 574–593.

[47] M. Ball, T. Malkin, and M. Rosulek, "Garbling gadgets for boolean and arithmetic circuits," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 565–577.

[48] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 784–796.

[49] C. Kempka, R. Kikuchi, and K. Suzuki, "How to circumvent the two-ciphertext lower bound for linear garbling schemes," in *Advances in Cryptology – ASIACRYPT 2016*, J. H. Cheon and T. Takagi, Eds. Berlin, Heidelberg: Springer, 2016, pp. 967–997.

[50] C. Guo, J. Katz, X. Wang, C. Weng, and Y. Yu, "Better concrete security for half-gates garbling (in the multi-instance setting)," in *Advances in Cryptology – CRYPTO 2020*. Springer, 2020, pp. 793–822.

[51] M. O. Rabin, "How To Exchange Secrets with Oblivious Transfer," Cryptology ePrint Archive, Report 2005/187, 2005. [Online]. Available: https://eprint.iacr.org/2005/187

[52] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More Efficient Oblivious Transfer and Extensions for Faster Secure Computation," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 535–548.

[53] M. Naor and B. Pinkas, "Oblivious Transfer and Polynomial Evaluation," in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, J. S. Vitter, L. L. Larmore, and F. T. Leighton, Eds. ACM, 1999, pp. 245–254.

[54] J. Jean, T. Peyrin, S. M. Sim, and J. Tourteaux, "Optimizing implementations of lightweight building blocks," *IACR Trans. Symmetric Cryptol.*, vol. 2017, Issue 4, pp. 130–168, 2017.

[55] K. Mandal and G. Gong, "Can LWC and PEC be Friends?: Evaluating Lightweight Ciphers in Privacy-enhancing Cryptography," in *Fourth Lightweight Cryptography Workshop*. NIST, 2020. [Online]. Available: https://csrc.nist.gov/Events/2020/lightweight-cryptography-workshop-2020

[56] K. Hamacher, T. Kussel, T. Schneider, and O. Tkachenko, "PEA: Practical Private Epistasis Analysis Using MPC," in *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part III*. Springer, 2022, pp. 320–339.

[57] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[58] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 955–966.

[59] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi, "Privacy-Free Garbled Circuits with Applications to Efficient Zero-Knowledge," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer, 2015, pp. 191–219.

[60] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *International Symposium on Experimental Algorithms*. Springer, 2010, pp. 178–189.