Fast Evaluation of S-boxes with Garbled Circuits

Erik Pohle, Aysajan Abidin, and Bart Preneel

imec-COSIC KU Leuven, Belgium
`firstname.lastname@esat.kuleuven.be`

Abstract. Garbling schemes, a formalization of Yao's garbled circuit protocol, are useful cryptographic primitives both in privacy-preserving protocols and for secure two-party computation. In projective garbling schemes, n values are assigned to each wire in the circuit. Current state-of-the-art schemes project two values.
More concretely, we present 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 moderate increased cost in garbling and bandwidth. Theoretical analysis suggests that for evaluating the nine examined ciphers, one can expect a 4- to 70-fold increase in evaluation with at most a 4-fold increase in garbling cost and, at most, an 8-fold increase in communication cost when 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 computation formulas for the S-boxes of TWINE and Midori64 in Boolean circuits. To our knowledge, our formulas give the smallest number of AND gates for the S-boxes of these two ciphers.

Keywords: garbled circuits, garbling scheme, substitution-permutation cipher, S-box, multi-party computation

1 Introduction

Privacy-preserving protocols combine the benefits of analysis on shared data with privacy rights of the data owners. In a two-party scenario, privacy-preserving genome analysis [29], email spam filtering [23], image processing [13] and machine learning [34] have been successfully implemented, just to name a few. Formalization of such two-party computation is called Secure Function Evaluation (SFE). Here the two parties, 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 [50] 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 in itself. 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. [51] or the recent work by Rosulek and Roy [46].

In this paper, we consider garbling schemes in the offline/online setting. Here, the offline phase performs function-dependent pre-processing. 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 solely to obtain the appropriate wire labels for their respective inputs. Then, the evaluator evaluates the garbled circuit. Clearly, 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. Therefore, the online time, i.e., the time from having obtained the respective inputs to the evaluated output of the garbled circuit, is important 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 mainly 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 number, i.e., an integer $0 \leq i < 2^n$. We generalize the encoding of FreeXOR by Kolesnikov and Schneider [36] to obtain a scheme where bitwise-XOR between n-bit numbers 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.

While the new garbling scheme assumes semi-honest adversaries, i.e. neither garbler nor 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 [43,38,26,27] and on zero-knowledge proofs [30,47]. Moreover, semi-honest garbling schemes can be compiled into actively secure three party protocols in the honest majority setting [40].

Technical Overview. The core ideas of the scheme are summarized as follows. We encode an n-bit number expressed in 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. 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 numbers, 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 and point-and-permute techniques. 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.

Contributions. We present a projective garbling scheme that assigns n-bit numbers to each wire and in which XOR gates are free. The specific encoding of an n-bit number 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), which we call n-TCCR, for hash functions, first defined by Choi et al. [15]. We prove n-TCCR to be equivalent to the TCCR notion with a modified distribution of the secret randomness. Thus, any TCCR secure hash function construction can be used in our scheme.

We apply the garbling scheme to compute a number of selected symmetric primitives that follow the substitution-permutation network architecture. For these, we show a significant improvement in evaluation work in the online phase that is traded off with moderate additional garbling work and/or communication cost in the offline phase. Table 1 shows the evaluation improvement, which is complemented by detailed analysis of the trade-off in Tables 4 and 5.

Furthermore, we give implementation formulas for the S-boxes of TWINE [49] and Midori64 [5], which is also used in MANTIS [7] and CRAFT [8], using only AND and XOR gates. To the best of our knowledge, our formulas 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 Sect. 6.2 and Fig. 7.

Table 1: Evaluation work improvement for selected symmetric primitives. Garbling and communication trade-off is listed in Table 5.

Primitive	Evaluation Improvement
TWINE-80 [49]	×9.81
TWINE-128	×9.05
Fides-80 [11]	×15.45
Fides-96	×65.05
WAGE [1]	×72.87
AES-128 [42]	×26.23
Piccolo-80 [48]	×4.71
Piccolo-128	×4.55
Midori64 [5]	×5.33
SKINNY-64-64 [7]	×7.11
SKINNY-64-128	×4.68
MANTIS [7]	×4.31
CRAFT [8]	×5.71

Organisation. The rest of the paper is organized as follows. In Sect. 2, we review related work on garbling schemes. Sect. 3 gives more details on previous work which we build upon. We present our scheme in Sect. 4 and prove its security in Sect. 5. A comparison of the state of the art and our scheme for general circuits and nine (lightweight) symmetric primitives in particular is given in Sect. 6. Finally, we conclude the paper in Sect. 7.

2 Related Work

Recent improvements on Yao's garbled circuit protocol focus on lowering bandwidth requirements. In the state-of-the-art scheme Half-Gates (Zahur et al. [51]), AND gates are broken into two unary gates and thus only require 2κ bits, where κ is a security parameter, to be sent, while XOR gates are free due to FreeXOR (Kolesnikov and Schneider [36]). The proposal by Rosulek and Roy [46] reduces the bandwidth per AND gate to 1.5κ bits by splitting wire labels into halves and treating the halves separately. This lower bandwidth is traded off with two additional cryptographic primitive calls when garbling and one additional primitive call when evaluating.

Another technique by Pinkas et al. [44] to reduce the number of ciphertexts that are sent per garbled gate is polynomial interpolation where XOR gates are no longer free. FleXOR (Kolesnikov et al. [35]) generalizes FreeXOR where the number of ciphertexts of any gate (XOR or non-XOR) depends on its position in the circuit. Here the number of ciphertexts per gate and if XOR gates are free depends on the circuit topology.

Security proofs for garbled circuits have become more modular (see e.g., [15,10,9]) where necessary properties of the encryption function have been explicitly defined. This allows separate study of efficient instantiations as in [9,20,22,14].

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. [17] define those gates as lookup tables and show how they can be evaluated in the passive security case in the Goldreich-Micali-Wigderson protocol [19]. Damgård et al. [16] design a table lookup for two-party secure computation and Keller et al. [32] extend it to the multi-party case based on secret-sharing. AES as a function has been studied explicitly by Durak and Guajardo [18] but in the arithmetic setting. Another protocol to mention is the one-time truth-table protocol by Ishai et al. [28].

In the garbled circuit domain, Huang et al. [25] compute larger gates, in their case an 8-bit to 8-bit AES S-box gate. But unlike our scheme, they use 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 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 [22,14]. 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. Recently, Heath and Kolesnikov [24] construct a garbling gadget that computes a so-called one-hot outer-product of two bit-vectors. The resulting matrix can be used to select one entry from a truth-table based on an index known by the evaluator. The authors show how to mask the index for certain functions but their approach doesn't generalize to arbitrary truth-tables.

Since the work of Ball et al. [4] is conceptually very close to ours, we discuss it in detail in Sect. 3.2. 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.

3 Background

We start with a brief overview of relevant improvements of the original garbled circuit protocol in Sect. 3.1. The arithmetic circuit scheme by Ball et al. [4] is detailed in Sect. 3.2 while the security model by Bellare, Hoang and Rogaway (BHR) [10] is presented in Sect. 3.3. Table 2 lists the notation used throughout the paper.

3.1 Yao's Garbled Circuits and Improvements

The garbled circuit protocol of Yao [50] provides a solution for two-player secure computation with a constant communication round complexity. In the protocol, the function to compute is represented as a Boolean circuit. One party, the circuit garbler, projects two random labels onto each wire in the circuit. The semantic meaning of true is associated to one label, the semantic of false to the other. A row in the truth table of each gate is encrypted such that given wire labels

Table 2: 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 $A, B \in \{0,1\}^l$
$A \vert\vert B$	Bit-vector concatenation for $A \in \{0,1\}^l, B \in \{0,1\}^{l'}$
$\{A, B\} \vert\vert C$	Bit-vector concatenation extended to sets, i.e., $\{A \vert\vert C, B \vert\vert C\}$
$x' \leftarrow x$	Assignment of value x to x'
$x \overset{\$}{\leftarrow} \{a, b, c, \dots\}$	Uniform sampling from the set $\{a, b, c, \dots\}$

of the input to the gate, the output wire label that corresponds to the output value, can be decrypted. Let $W^0 \in \{0,1\}^\kappa$ denote the label with semantic 0 and $W^1 \in \{0,1\}^\kappa$ with semantic 1, then a row of the gate's truth table is encrypted as

$$\mathbb{E}_{W^x, W^y}(W^{g(x,y)}),\ x, y \in \{0,1\}\ ,$$

where $\mathbb{E}_{k_1,k_2}$ is an appropriate encryption function with keys k_1, k_2. The gate outputs the truth value $g(x,y) \in \{0,1\}$ for the input $x, y \in \{0,1\}$. The garbler randomly permutes the order of the ciphertexts to prevent leakage due to the row's position.

Once all gates are encrypted, the garbled circuit is sent to the second party, the evaluator, along with the wire labels that correspond to the garbler's input. The evaluator obtains the labels for its input by engaging in a 1-out-of-2 oblivious transfer (OT) per input bit, where the evaluator poses as receiver and the garbler as the sender. Thus the evaluator only learns the wire label corresponding to its input while the garbler does not learn the evaluator's choice. The evaluator then evaluates the circuit by decrypting the truth tables using the wire labels. In the end, the evaluator obtains the semantic meaning of the output wire labels using the decoding information made public by the garbler. For a formal definition as well as proof of security, we refer the reader to the work of Lindell and Pinkas [37].

Beaver et al. [6] introduced the point-and-permute technique. The garbler chooses the labels per wire to include a pointer bit, e.g., the least significant bit, such that the pointer bit of W^1 is the complement of the pointer bit of W^0. These bits then indicate the position of the ciphertext that can be decrypted correctly. For example, let the least significant bit of the first input wire be 1 and that of the second wire be 0. Then the evaluator only needs to decrypt the second ciphertext of the gate. As the pointer bit is disconnected from the semantic meaning of the wire, no information about the association is leaked and the truth table permutation is automatically applied. This optimization reduces the number of decryptions from four to one per gate for the evaluator.

The so-called garbled row reduction technique introduced by Naor et al. [41] reduces the number of ciphertexts that need to be sent by one. One ciphertext, typically the first of the *garbled* table, is fixed to a constant and therefore does not need to be transmitted to the evaluator. If the encryption function is realized as a hash function construction $\mathbb{E}_{W^x,W^y}(W^z) = H(W^x||W^y) \oplus W^z$, then this means that $H(W^x||W^y) \oplus W^z = 0^\kappa$ (and thus W^z is chosen as $H(W^x||W^y)$) when the pointer bits of W^x and W^y are both 0.

Kolesnikov and Schneider [36] introduce the FreeXOR optimization, where the wire label encoding throughout the circuit has the invariant

$$W^1 = W^0 \oplus R\,,$$

where $R \in \{0,1\}^\kappa$ is a secret random offset chosen by the garbler. If R is used for every wire in the circuit, then XOR gates can be expressed without sending any ciphertexts and without involving the encryption function. For an XOR gate with input wires a and b, and output wire c, the garbler chooses $W_c^0 = W_a^0 \oplus W_b^0$. Let W_a^x and W_b^y be the wire labels obtained by the evaluator. Then

$$\begin{aligned} W_c^{x\oplus y} &= W_a^x \oplus W_b^y \\ &= W_a^0 \oplus xR \oplus W_b^0 \oplus yR \\ &= \underbrace{W_a^0 \oplus W_b^0}_{W_c^0} \oplus (x \oplus y)R \ \ , \end{aligned}$$

which satisfies the invariant and preserves the expected semantic association. For FreeXOR, the bit times label scalar multiplication xR results in R if $x = 1$, or else 0^κ. The correlation between the two labels requires more specific properties of the encryption function. More details can be found in the security proof of Choi et al. [15] and Guo et al. [21].

All the improvements described above as well as many others (e.g., [25,51,46,3]) only result in garbling schemes that project two values onto a wire.

3.2 Garbled Circuits for Bounded Integers

Ball et al. [4] 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 very similar but represents n-bit numbers 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 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, where

$$W_c^0 = -H(W_a^{-r}) - \phi(-r) \odot \Delta_m$$

obtained from the encryption above when $r = -x$. 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.

3.3 Security Model by Bellare, Hoang and Rogaway

Bellare, Hoang and Rogaway [10] 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., [51,33,21]. As we will also prove our proposed scheme secure using the same model, we give a brief overview.

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

- Garble: Transforms the input circuit f into the tuple (GC, X, d) where GC is the garbled circuit, X are *all* semantic labels for input wires and d is the decoding information.
- 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.

We leave out the input encoding, i.e., letting the evaluator choose the appropriate labels from X, since this is typically handled by OT (extensions) (see Sect. 4.3). The garbling scheme must produce correct circuit evaluations for any circuit f and inputs $x \in \{0,1\}^n$. Let GC, X, d be the outputs of $\mathsf{Garble}(f)$, then

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

where $f(x)$ denotes the circuit evaluation in the clear and $W_i^{x_i}$ is the appropriate label for input value x_i selected from X.

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, $\Phi = \Phi_{\text{circ}}$ where the whole circuit is revealed. This means that both parties know the function that is computed. The privacy property can be achieved by giving a simulator $\mathcal{S}$ for the Garble function. 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 Adv(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 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 $\mathbf{Adv}^{\text{prv.sim}}$.

```
function GARBLE$(f, x)$                                  ▷ Game prv.sim $\Phi$, $\mathcal{S}$
    $b \xleftarrow{\$} \{0,1\}$
    if b = 1 then
        $(GC, X, d) \leftarrow \mathsf{Garble}(f)$
    else
        $y \leftarrow f(x)$
        $(GC, X, d) \leftarrow \mathcal{S}(1^k, y, \Phi(f))$
    return $(GC, X, d)$
```

```
function GARBLE$(f, x)$                                  ▷ Game obv.sim $\Phi$, $\mathcal{S}$
    $b \xleftarrow{\$} \{0,1\}$
    if b = 1 then
        $(GC, X, d) \leftarrow \mathsf{Garble}(f)$
    else
        $(GC, X) \leftarrow \mathcal{S}(1^k, \Phi(f))$
    return $(GC, X)$
```

Fig. 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 $b = b'$.

4 The Scheme

A garbling scheme comprises a garbling, evaluation and decoding function. In Sect. 4.1, we first describe the notation for a circuit comprising XOR gates and projection gates. Then, we detail how the garbler encodes n-bit numbers and transforms them into wire labels. Next, we show how XOR gates are garbled and evaluated, followed by a description how projection gates are garbled and evaluated. Sect. 4.2 describes higher-level building blocks that can be obtained from the aforementioned gates. In Sect. 4.3 all concepts are pieced together to describe the garbling, evaluation and decoding function. We also describe how input is handled. The complete garbling scheme $\mathit{\Pi}$ is given in Fig. 2.

We start with some general notations. Let $\mathsf{lsb}_n(W)$ be the n least significant bits[1] 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. 5.1.

4.1 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 $\mathsf{Outputs}$. 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 ($1 \leq n \leq \bar{n}$) 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 appends fixed bits to each column-vector to form $\mathbf{R}_n$*

$$\mathbf{R}_n = \begin{pmatrix} \mathbf{M} \\ \mathbf{I}_n \end{pmatrix} ,$$

where $\mathbf{I}_n \in \{0,1\}^{n \times n}$ is the identity matrix. 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., $\mathbf{R}_n \overset{\$}{\leftarrow} \mathcal{R}_n$.

[1] 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 $i \in \{1, \dots, \bar{n}\}$ do
        $\mathbf{R}_i \xleftarrow{\$} \mathcal{R}_i$
    return $\mathbf{R}_1, \dots, \mathbf{R}_{\bar{n}}$
function $\textsc{Garble}(f)$
    $\mathbf{R}_1, \dots, \mathbf{R}_{\bar{n}} \leftarrow \textsc{GenR}(\bar{n})$
    for $i \in \mathsf{Inputs}$ do
        $W_i^{0^{\ell(i)}} \xleftarrow{\$} \{0,1\}^k$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two $n$-bit input wires to $\mathsf{G}_i$
            $W_i^{0^n} \leftarrow W_a^{0^n} \oplus W_b^{0^n}$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $W_i^{0^m} \xleftarrow{\$} \{0,1\}^k$
            for $x \in \{0,1\}^n$ do
                $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathcal{H}(W_a^x, i) \oplus W_i^{0^n} \oplus \phi(x) \cdot \mathbf{R}_n$
    for $i \in \mathsf{Outputs}$ do
        $d_i \leftarrow \mathsf{lsb}_{\ell(i)}(W_i^{0^{\ell(i)}})$
    $X \leftarrow \{W_i^x | x \in \{0,1\}^{\ell(i)}\}_{i \in \mathsf{Inputs}}$
    return $GC, X, d$

function $\textsc{Eval}(GC, \{W_i\}_{i \in \mathsf{Inputs}})$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be the two input wires to $\mathsf{G}_i$
            $W_i \leftarrow W_a \oplus W_b$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            $W_i \leftarrow \mathcal{H}(W_a, i) \oplus GC[i, \mathsf{lsb}_n(W_a)]$
    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 $y$
```

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

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.

Example 1. The wire label offsets for $n = 1$ are $\mathbf{R}_1 = (R_1)$ with $R_1 \overset{\$}{\leftarrow} \{0,1\}^{k-1}||1$. For $n = 2$, the matrix has two columns $\mathbf{R}_2 = (R_1, R_2)$ where the first column is $R_1 \overset{\$}{\leftarrow} \{0,1\}^{k-2}||10$ and the second column is $R_2 \overset{\$}{\leftarrow} \{0,1\}^{k-2}||01$. ◁

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 number $x \in \{0,1\}^n$ on a wire with index i is defined as*

$$W_i^x = W_i^{0^n} \oplus x \cdot \mathbf{R}_n \ .$$

Note that this yields an unique encoding for all values of 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 $\mathbf{I}_n$.

Example 2. In the case $n = 1$, the wire labels of wire i are

$$\begin{aligned} &W_i^0 \ , \\ &W_i^1 = W_i^0 \oplus R_1 \ , \end{aligned}$$

for 0 and 1, respectively. Note that this is the FreeXOR scheme.

For $n = 2$, the four wire labels are encoded as

$$\begin{aligned} &W_i^{00} \ , \\ &W_i^{01} = W_i^{00} \oplus R_1 \ , \\ &W_i^{10} = W_i^{00} \oplus R_2 \ , \\ &W_i^{11} = W_i^{00} \oplus R_1 \oplus R_2 \ . \end{aligned}$$

◁

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.

Definition 3 (XOR Gate Garbling). *For an XOR gate with n-bit input wires a and b, and output wire c, the garbler generates the output wire label*

$$W_c^x \leftarrow \underbrace{W_a^{0^n} \oplus W_b^{0^n}}_{W_c^{0^n}} \oplus x \cdot \mathbf{R}_n \ \ ,$$

where $x \in \{0,1\}^n$; no ciphertext is sent.

Definition 4 (XOR Gate Evaluation). *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 computed as*

$$W_c \leftarrow W_a \oplus W_b \ .$$

Observation. Garbling and evaluating an XOR gate results in the correct wire label encoding. We obtain

$$\begin{aligned} W_c^{x\oplus y} &= \underbrace{W_a^{0^n} \oplus W_b^{0^n}}_{W_c^{0^n}} \oplus x \cdot \mathbf{R}_n \oplus\ y \cdot \mathbf{R}_n \\ &= W_c^{0^n} \oplus x \cdot \mathbf{R}_n \oplus y \cdot \mathbf{R}_n \\ &= W_c^{0^n} \oplus (x_1 \oplus y_1)R_1 \oplus \ldots \oplus (x_n \oplus y_n)R_n \\ &= W_c^{0^n} \oplus (x \oplus y) \cdot \mathbf{R}_n \ , \end{aligned}$$

which satisfies the wire label encoding from Definition 2 for an encoding of $x \oplus y$.

Definition 5 (Projection Gate Garbling). *Let a be the input wire index to the projection gate Proj_ϕ that computes the unary projection $\phi : \{0,1\}^n \mapsto \{0,1\}^m$, and c be the index of the output wire. The garbler first draws the output wire label for 0 at random: $W_c^{0^m} \overset{\$}{\leftarrow} \{0,1\}^k$ and then generates 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:*

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

Definition 6 (Projection Gate 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.

Observation. Garbling and evaluating a projection gate results in the correct wire label encoding (Definition 2). Evaluating the projection gate with input label W_a^x returns the correct wire label $W_c^{\phi(x)}$ due to

$$GC[c, \mathsf{lsb}_n(W_a^x)] \oplus \mathcal{H}(W_a^x, c) = \mathcal{H}(W_a^x, c) \oplus W_c^{\phi(x)} \oplus \mathcal{H}(W_a^x, c) = W_c^{\phi(x)} \ .$$

We can apply the row-reduction technique and reduce the number of ciphertexts that need to be sent by one.

Definition 7 (Garbled Row Reduction for Projection Gates). *Let a be the input wire index to the projection gate Proj_ϕ that computes the unary projection $\phi : \{0,1\}^n \mapsto \{0,1\}^m$, 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 in Definition 5.

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 $2^n - 1$.

The evaluator computes the garbled output wire as stated in Definition 6, setting $GC[c, 0^n] = 0^m$ implicitly.

4.2 Circuit Constructs

In the following, we give useful constructs 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 more efficient to compose many wires together at once instead of cascading or using a tree-based approach[2]. 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 $f(x_1||\dots||x_{2n}) = x_1||\dots||x_n$ and $f' : \{0,1\}^{2n} \mapsto \{0,1\}^n$ as $f'(x_1||\dots||x_{2n}) = x_{n+1}||\dots||x_{2n}$ 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.

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

4.3 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 the operations for projection and XOR gates as described in Definitions 3 and 5 gate-by-gate in topological order. In the end, the garbling routine outputs the ciphertexts, input wire values and decoding information.

Evaluation and Decoding. Once the evaluator obtains the garbled inputs, it computes the garbled output of each gate accordingly (see Definitions 4 and 6). 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

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

Oblivious Transfer. In Yao's protocol, the evaluator obtains the appropriate wire labels that correspond to its input via oblivious transfer (OT) [45]. Using OT extensions [28,2] 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. However, using the FreeXOR property of our scheme, we can instead perform 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.

Example 3. For instance, let $n = 2$ and the input be $b_1 b_0$, the evaluator receives

$$W_a = \begin{cases} W_a^{00} & \text{if } b_0 = 0 \\ W_a^{01} & \text{if } b_0 = 1 \end{cases}$$

$$W_b = \begin{cases} W_b^{00} & \text{if } b_1 = 0 \\ W_b^{10} & \text{if } b_1 = 1 \end{cases}$$

via 2 1-out-of-2 OTs. The final input wire label is then $W_a \oplus W_b$. ◁

We can further reduce the transmission size of the OT by following the idea for correlated OT (C-OT) inputs by Asharov et al. [2]. This effectively halves the number of values sent.

5 Security

Using the BHR security model (see Sect. 3.3) we show that if a hash function satisfies the properties of n-TCCR security defined in Sect. 5.1 below, our scheme is prv.sim (Sect. 5.2) and obv.sim (Sect. 5.3) secure. We sketch how to achieve authenticity (Sect. 5.4), prove the equivalence of n-TCCR and TCCR security (Sect. 5.5) and give a concrete construction for the hash function (Sect. 5.6).

5.1 (n-)TCCR Security

We revisit the tweakable circular correlation robustness (TCCR) definition by Guo et al. [22] adapted to our notation.

Definition 8 (TCCR Security [22]). *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 $\mathcal{D}_{TCCR}$ 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 $R \in \{0,1\}^k$, $R \overset{\$}{\leftarrow} \mathcal{R}_{\mathrm{TCCR}}$ 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

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

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

Note that the advantage of $\mathcal{D}_{\mathrm{TCCR}}$ 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 9 (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 $\mathcal{D}_{n\text{-}TCCR}$ 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, $\mathbf{R} = (R_1, \dots, R_n)$, $\mathbf{R} \in \{0,1\}^{k \times n} \overset{\$}{\leftarrow} \mathcal{R}_n$ and $R_i \in \{0,1\}^k$, $1 \le i \le n$. 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

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

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

5.2 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 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}.

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 $W^{v_i} = W^{0^n} \oplus v_i \cdot \mathbf{R}_n$ are both distributed uniformly at random because W^{0^n} is.

```
function $\mathcal{S}(f, y)$
    $\mathbf{R}_1, \ldots, \mathbf{R}_{\bar{n}} \leftarrow GenR(\bar{n})$
    for $i \in \mathsf{Inputs}$ do
        $W_i^{0^{\ell(i)}} \xleftarrow{\$} \{0,1\}^k$
        $X_i \leftarrow W_i^{0^{\ell(i)}}$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two $n$-bit input wires to $\mathsf{G}_i$
            $W_i^{0^n} \leftarrow W_a^{0^n} \oplus W_b^{0^n}$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $W_i^{0^m} \xleftarrow{\$} \{0,1\}^k$
            $W_i^{0^m} \leftarrow W_i^{0^m} \oplus \phi(0^n) \cdot \mathbf{R}_n$
            for $x \in \{0,1\}^n$ do
                $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathsf{Rand}_n(W_a^{0^n}, i, x, \phi(x)) \oplus W_i^{0^m}$
    for $i \in \mathsf{Outputs}$ do
        $d_i \leftarrow \mathsf{lsb}_n(W_i^{0^n}) \oplus y_i$
    return $GC, X, d$
```

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

$\mathcal{G}_1 \approx \mathcal{G}_2$. In hybrid $\mathcal{G}_2$ (see Fig. 5), 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 9).

$$\begin{aligned}\mathsf{Rand}(W_a^{0^n}, i, x, \phi(x)) &\approx \mathcal{H}(W_a^{0^n} \oplus x \cdot \mathbf{R}, i) \oplus \phi(x) \cdot \mathbf{R} \\ &= \mathcal{H}(W_a^x, i) \oplus \phi(x) \cdot \mathbf{R}_n\,.\end{aligned}$$

$\mathcal{G}_2 \approx \mathcal{G}_3$. In hybrid $\mathcal{G}_3$ (see Fig. 6), 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 $W_i^{v_i}$.

In the output of all gates G_i, we now maintain the invariant with $x \in \{0,1\}^n$

$$\begin{aligned}W_i^{v_i} &\text{ becomes } W_i^{0^n}\,, \\ W_i^{v_i \oplus x} &\text{ becomes } W_i^{0^n} \oplus x \cdot \mathbf{R}_n\,.\end{aligned}$$

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

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

and in $\mathcal{G}_3$:

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

```
function $\textsc{EvalWires}(f, x)$
    for $i \in \mathsf{Inputs}$ do
        $v_i \leftarrow x_i$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        $a, b \leftarrow in(\mathsf{G}_i)$
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two input wires to $\mathsf{G}_i$
            $v_i \leftarrow v_a \oplus v_b$
        else
            Let $a$ be the input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $v_i \leftarrow \phi(v_a)$
    return $v$
function $\mathcal{G}_1(f, x)$
    $\mathbf{R}_1, \ldots, \mathbf{R}_{\bar{n}} \leftarrow GenR(\bar{n})$
    $v \leftarrow \mathsf{EvalWires}(f, x)$
    for $i \in \mathsf{Inputs}$ do
        $W_i^{\boxed{v_i}} \xleftarrow{\$} \{0,1\}^k$
        $X_i \leftarrow W_i^{\boxed{v_i}}$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two $n$-bit input wires to $\mathsf{G}_i$
            $W_i^{\boxed{v_i}} \leftarrow W_a^{\boxed{v_a}} \oplus W_b^{\boxed{v_b}}$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $W_i^{\boxed{v_i}} \xleftarrow{\$} \{0,1\}^k$
            $W_i^{\boxed{v_i}} \leftarrow W_i^{\boxed{v_i}} \oplus \phi(\boxed{v_a}) \cdot \mathbf{R}_n$
            for $x \in \{0,1\}^n$ do
                $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathsf{Rand}(W_a^{0^n}, i, x, \phi(x)) \oplus W_i^{\boxed{v_i}}$
    for $i \in \mathsf{Outputs}$ do
        $d_i \leftarrow \mathsf{lsb}_n(W_i^{\boxed{v_i}}) \oplus y_i$
    return $GC, X, d$
```

Fig. 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 $\mathcal{G}_1$.

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. □

5.3 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. Note that unlike the

```
function $\mathcal{G}_2(f, x)$
    $\mathbf{R}_1, \ldots, \mathbf{R}_{\bar{n}} \leftarrow GenR(\bar{n})$
    $v \leftarrow \mathsf{EvalWires}(f, x)$
    for $i \in \mathsf{Inputs}$ do
        $W_i^{v_i} \xleftarrow{\$} \{0,1\}^k$
        $X_i \leftarrow W_i^{v_i}$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two $n$-bit input wires to $\mathsf{G}_i$
            $W_i^{v_i} \leftarrow W_a^{v_a} \oplus W_b^{v_b}$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $W_i^{v_i} \xleftarrow{\$} \{0,1\}^k$
            $W_i^{v_i} \leftarrow W_i^{v_i} \oplus \phi(v_a) \cdot \mathbf{R}_n$
            for $x \in \{0,1\}^n$ do
                $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \boxed{\mathcal{H}(W_a^x, i)} \oplus W_i^{v_i} \boxed{\oplus\phi(x) \cdot \mathbf{R}_n}$
    for $i \in \mathsf{Outputs}$ do
        $d_i \leftarrow \mathsf{lsb}_n(W_i^{v_i}) \oplus y_i$
    return $GC, X, d$
```

Fig. 5: Hybrid $\mathcal{G}_2$. We replaced Rand by the Circ construction. Values in a box $\boxed{v_i}$ highlight the difference between $\mathcal{G}_1$ and $\mathcal{G}_2$.

```
function $\mathcal{G}_3(f, x)$
    $\mathbf{R}_1, \ldots, \mathbf{R}_{\bar{n}} \leftarrow GenR(\bar{n})$

    for $i \in \mathsf{Inputs}$ do
        $W_i^{\boxed{0^{\ell(i)}}} \xleftarrow{\$} \{0,1\}^k$
        $X_i \leftarrow W_i^{\boxed{x_i}}$
    for $\mathsf{G}_i \in \mathsf{Gates}$ do
        $a, b \leftarrow in(\mathsf{G}_i)$
        if $\mathsf{G}_i = XOR$ then
            Let $a$, $b$ be two $n$-bit input wires to $\mathsf{G}_i$
            $W_i^{\boxed{0^n}} \leftarrow W_a^{\boxed{0^n}} \oplus W_b^{\boxed{0^n}}$
        else
            Let $a$ be the $n$-bit input wire to $\mathsf{G}_i$
            Let $\phi : \{0,1\}^n \mapsto \{0,1\}^m$ be the function $\mathsf{G}_i$ realizes
            $W_i^{\boxed{0^n}} \xleftarrow{\$} \{0,1\}^k$

            for $x \in \{0,1\}^n$ do
                $GC[i, \mathsf{lsb}_n(W_a^x)] \leftarrow \mathcal{H}(W_a^x, i) \oplus W_i^{\boxed{0^m}} \oplus \phi(x) \cdot \mathbf{R}_n$
    for $i \in \mathsf{Outputs}$ do
        $d_i \leftarrow \mathsf{lsb}_n(W_i^{\boxed{0^n}})$
    return $GC, X, d$
```

Fig. 6: Hybrid $\mathcal{G}_3$. We fix the encoding of $W_i^{v_i}$ to $W_i^{0^n}$. Values in a box $\boxed{v_i}$ highlight the difference between $\mathcal{G}_2$ and $\mathcal{G}_3$.

privacy notion, the adversary doesn't 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 17-18 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. □

5.4 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 [10], the decoding function checks if the presented wire is indeed in the list d.

5.5 Equivalence of TCCR and n-TCCR Security

In this section, we prove that the TCCR notion and n-TCCR security are equivalent. This is stated in Theorem 3.

We recall that $\mathcal{R}_n$ is the distribution of column vectors from Definition 1, i.e.,

$$\begin{pmatrix} \mathbf{M} \\ \mathbf{I}_n \end{pmatrix} ,$$

where $\mathbf{M} \in \{0,1\}^{\kappa \times n}$ is a random matrix and $\mathbf{I}_n \in \{0,1\}^{n \times n}$ is the identity matrix. Further, let $\mathcal{U}_{\kappa,n}$ be the distribution of $\kappa + n$-bit strings with κ uniform random bits, and n fixed bits which cannot be 0^n.

Theorem 3. *The TCCR game with $R \xleftarrow{\$} \mathcal{U}_{\kappa,n}$ and the n-TCCR game with $\mathbf{R} \xleftarrow{\$} \mathcal{R}_n$ are equivalent.*

Proof. We show the equivalence by constructing (1) a distinguisher that breaks the n-TCCR game assuming a distinguisher for TCCR with non-negligible advantage, and (2) a distinguisher that breaks the TCCR game assuming a distinguisher for n-TCCR with non-negligible advantage.

1. n-TCCR $\implies$ TCCR. Given a distinguisher $\mathcal{D}_{\text{TCCR}}$ with advantage $\mathsf{Adv}_{\mathcal{U}_{\kappa,n}}(\mathcal{D}_{\text{TCCR}})$, one can construct a distinguisher $\mathcal{D}$ for the n-TCCR game with

$$\mathsf{Adv}_{\mathcal{R}_n}(\mathcal{D}) \geq \mathsf{Adv}_{\mathcal{U}_{\kappa,n}}(\mathcal{D}_{\text{TCCR}}) .$$

Let $\mathbf{x} \in \{0,1\}^n$ be the n fixed bits in $\mathcal{U}_{\kappa,n}$ of the TCCR distinguisher. For every query of $\mathcal{D}_{\text{TCCR}}$ of the form m, t, b, we return

$$\begin{cases} \mathcal{O}_{\mathbf{R}}(m, t, \mathbf{x}, \mathbf{0}) & \text{if b} = 0 \\ \mathcal{O}_{\mathbf{R}}(m, t, \mathbf{x}, \mathbf{x}) & \text{if b} = 1\,. \end{cases}$$

The output choice of $\mathcal{D}$ is the output choice of the underlying $\mathcal{D}_{\text{TCCR}}$. If $\mathcal{O}_{\mathbf{R}}$ is the ideal oracle, we also simulate an ideal oracle for $\mathcal{D}_{\text{TCCR}}$. If the oracle is the real oracle, the distinguisher sees

$$\begin{cases} \mathcal{H}(m \oplus \mathbf{x} \cdot \mathbf{R}, t) & \text{if b} = 0 \\ \mathcal{H}(m \oplus \mathbf{x} \cdot \mathbf{R}, t) \oplus \mathbf{x} \cdot \mathbf{R} & \text{if b} = 1\,. \end{cases}$$

This distribution is identical to $\mathcal{H}(m \oplus R, t) \oplus bR$, $R \xleftarrow{\$} \mathcal{U}_{\kappa,n}$ since $\mathbf{x} \cdot \mathbf{R} \sim R$. The first κ bits of $\mathbf{x} \cdot \mathbf{R}$ and R are uniform, while the last n bits are equal by construction $\mathbf{x} \cdot \mathbf{I}_n = \mathbf{x}$. We therefore provide a perfect simulation and win the n-TCCR game with the same advantage as the TCCR distinguisher (for $R \xleftarrow{\$} \mathcal{U}_{\kappa,n}$).

2. TCCR $\implies$ *n*-TCCR Given a distinguisher $\mathcal{D}_{\text{n-TCCR}}$ with advantage $\mathsf{Adv}_{\mathcal{R}_n}(\mathcal{D}_{\text{n-TCCR}})$, one can construct a distinguisher $\mathcal{D}$ for the TCCR game with advantage

$$\mathsf{Adv}_{\mathcal{U}_{\kappa,n}}(\mathcal{D}) \geq \mathsf{Adv}_{\mathcal{R}_n}(\mathcal{D}_{\text{n-TCCR}})\,.$$

We construct $\mathcal{D}$ which has access to $\mathcal{O}_R(m, t, b)$. Choose $\mathbf{R}$ uniform at random from $\mathcal{R}_n$ and let X be the fixed n bits in $\mathcal{U}_{\kappa,n}$ prepended with 0^κ to obtain a $\kappa + n$-bit string. Then answer every query m, t, $\mathbf{a}$, $\mathbf{b}$ of $\mathcal{D}_{\text{n-TCCR}}$ with

$$\mathcal{O}_R(m \oplus \mathbf{a} \cdot \mathbf{R} \oplus X, t, \mathbb{1}(\mathbf{b})) \oplus \mathbf{b} \cdot \mathbf{R} \oplus \mathbb{1}(\mathbf{b})X\,,$$

where $\mathbb{1}(\mathbf{b})$ returns 0 if $\mathbf{b} = 0$, otherwise it returns 1. The output of our construction $\mathcal{D}$ is the output of the underlying $\mathcal{D}_{\text{n-TCCR}}$. If $\mathcal{O}_R$ is the ideal oracle, $\mathcal{D}_{\text{n-TCCR}}$ sees a random function since $\mathsf{Rand} \oplus \mathbf{b} \cdot \mathbf{R} \oplus \mathbb{1}(\mathbf{b})X \sim \mathsf{Rand}$. If the oracle is the real construction, $\mathcal{D}_{\text{n-TCCR}}$ sees

$$\mathcal{H}(m \oplus \mathbf{a} \cdot \mathbf{R} \oplus R \oplus X, t) \oplus \mathbb{1}(\mathbf{b})R \oplus \mathbf{b} \cdot \mathbf{R} \oplus \mathbb{1}(\mathbf{b})X\,.$$

Now note that $\mathbf{a} \cdot \mathbf{R} \oplus R \oplus X$ is identically distributed as $\mathbf{a} \cdot \mathbf{R}$. The first κ bits are uniform random, the last n bits are $\mathbf{a} \cdot \mathbf{I_n}$ (X cancels the fixed bits of R). Further, $\mathbb{1}(\mathbf{b})R \oplus \mathbf{b} \cdot \mathbf{R} \oplus \mathbb{1}(\mathbf{b})X$ is identically distributed as $\mathbf{b} \cdot \mathbf{R}$. For $\mathbf{b} = 0$, all is zero. For $\mathbf{b} \neq 0$, $R \oplus \mathbf{b} \cdot \mathbf{R} \oplus X \sim \mathbf{b} \cdot \mathbf{R}$ for the same reasons as above.

Clearly, we perfectly simulated the n-TCCR game for $\mathcal{D}_{\text{n-TCCR}}$ and therefore win the TCCR game with the same advantage. □

5.6 Instantiating $\mathcal{H}$

As a consequence of Theorem 3, any TCCR secure hash function is also $\bar{n}$-TCCR secure (with the slight security degradation due to R being sampled from $\mathcal{U}_{\kappa,n}$ instead of uniformly) and thus can be used as instantiation for $\mathcal{H}$.

To exemplify, we give the construction that uses one permutation call given by Chen and Tessaro [14, Theorem 2] with its $\bar{n}$-TCCR security bound. Let $k = \kappa + \bar{n}$ be the length of the labels, $\otimes$ be a non-linear operation between k-bit strings, and $\pi \in \mathrm{Perm}(k)$ be a permutation on k-bit strings. Then

$$\mathcal{H}_\pi(m, t) = \pi(m \otimes t) \oplus m \otimes t\,.$$

For any distinguisher $\mathcal{D}$ making at most q construction queries, and at most p queries to $\pi^{\pm}$, and for input tweaks $t \in \{0,1\}^k \setminus \{0^k, 0^{k-1}1\}$, the advantage is bounded by

$$\mathsf{Adv}^{\mathcal{H}_\pi}_{\mathcal{U}_{\kappa,\bar{n}}}(\mathcal{D}) \leq \frac{2qp}{2^\kappa} + \frac{q^2}{2^{\kappa+1}} + \frac{q^2}{2^{k+1}}\,.$$

6 Applications

In the following, we discuss applications where our new garbling scheme improves over the state-of-the-art. In Sect. 6.1, some general properties of algorithms are listed that, if fulfilled, result in better evaluation performance. These properties are not far-fetched as plenty of desirable functionalities show them. We illustrate this with a more detailed application to substitution permutation network-based block ciphers and permutations in Sect. 6.2.

We compare the state-of-the-art garbling schemes Half-Gates by Zahur, Rosulek and Evans. [51], which we abbreviate ZRE15, as well as the work of Rosulek and Roy [46], abbreviated RR21. Both schemes support free XOR gates and AND gates on wires holding one bit.

6.1 General

Our presented garbling scheme does support the same circuits as ZRE15 and RR21. On the one hand, projection gates can be used to compute the functionality of n parallel AND gates[3] which is prohibitively expensive, however. On the other hand, the generalization of FreeXOR for $n > 1$ in our scheme is orthogonal to garbling schemes working with wires representing a single bit and using FreeXOR, such as ZRE15 and RR21. Indeed, our garbling scheme can be used in addition to ZRE15/RR21 or other garbling gadgets, e.g., [24].

The new garbling scheme becomes interesting when the functions to be implemented have clearly separated non-linear parts and linear parts. We can then study the non-linear part individually and compare its implementation cost when using AND gates to the cost of a single projection gate. For instance, if a 4-bit function is implemented with at least 4 or 8 AND gates, then employing one 4-bit projection gate yields better garbling or communication performance, respectively, in a comparison with ZRE15. Our garbling scheme offers faster evaluation performance in any case, even if the functionality can be computed with

[3] Given two n-bit wires, we first compute a wire composition and obtain one $2n$-bit wire. A subsequent projection gate computes the $2n$ to n-bit function $f(x_1||\dots||x_n||y_1||\dots||y_n) = x_1 \wedge y_1||\dots||x_n \wedge y_n$.

a single AND gate. Table 3 lists this consideration for 3- to 8-bit functions with respect to ZRE15 and RR21.

Table 3: Overview of the least number of AND gates with which the n-bit to n-bit function needs to be expressed for our scheme to perform more efficiently. The column garbling improvement lists this number regarding garbling performance, the column communication improvement lists this number regarding sent ciphertexts. Note that our scheme always outperforms ZRE15 and RR21 when considering only evaluation performance, i.e., for any n-bit functionality that costs at least 1 AND gate to compute.

n-bit functionality	Garbling improvement		Communication improvement	
	ZRE15 [51]	RR21 [46]	ZRE [51]	RR21 [46]
3	$\geq$ 2 AND	$\geq$ 2 AND	$\geq$ 4 AND	$\geq$ 5 AND
4	$\geq$ 4 AND	$\geq$ 3 AND	$\geq$ 8 AND	$\geq$ 10 AND
5	$\geq$ 8 AND	$\geq$ 6 AND	$\geq$ 16 AND	$\geq$ 21 AND
6	$\geq$ 16 AND	$\geq$ 11 AND	$\geq$ 32 AND	$\geq$ 42 AND
7	$\geq$ 32 AND	$\geq$ 22 AND	$\geq$ 64 AND	$\geq$ 85 AND
8	$\geq$ 64 AND	$\geq$ 43 AND	$\geq$ 128 AND	$\geq$ 170 AND

6.2 SPN-based Symmetric Primitives

The structure of SPN-based primitives lends itself to an efficient representation as circuit with XOR and projection gates. In these 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 have been studied widely and are commonly used to construct block ciphers and pseudo-random permutations used, e.g., in hash or MAC functions.

The following conditions for state and round function parts are sufficient to ensure a well-performing circuit representation with projection gates.

- **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. The mixing matrix is binary.
- **Round Constant/(Round) Key Addition Layer.** The round constant or (round) key is XORed cell-wise.

With this structure, we 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.

Indeed, we identified many SPN primitives in literature that fulfil the conditions and counted the number of AND gates and projection gates, respectively, for their implementation. Table 4 lists the AND gate count and the projection gate count for data path and key schedule (if applicable) of the primitives. Next, we describe the implementation of the data path in more detail, followed by the key schedule.

Data Path. 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. 4.3.

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.

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 implementations using the smallest number of AND gates so far. We used the heuristic optimization tool LIGHTER by Jean et al. [31] operating on a customized cost metric. We restricted 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. 7a). For the Midori64 S-box, the number of AND gates is reduced from 8 AND gates (formula given in the specification [5]) to 4 AND gates (see Fig. 7b).

Key Schedule 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

[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.

Table 4: 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. [39].

Primitive	Setup	Key Schedule	Data Path
TWINE-80		432 AND	1728 AND
		80 1-bit + 70 4-bit	288 4-bit
TWINE-128		630 AND	1728 AND
		128 1-bit + 104 4-bit	288 4-bit
Fides-80			320 AND
	160 1-bit		32 5-bit
Fides-96			1088 AND
	192 1-bit		32 6-bit
WAGE			37745 AND†
	259 1-bit		777 7-bit
AES-128		1280 AND	5120 AND
		128 1-bit + 49 8-bit	320 8-bit
Piccolo-80			1600 AND
		80 1-bit	600 4-bit
Piccolo-128			1984 AND
		128 1-bit	744 4-bit
Midori64			1024 AND
		128 1-bit	256 4-bit
SKINNY-64-64			2048 AND
		64 1-bit	512 4-bit
SKINNY-64-128			2304 AND
		128 1-bit + 280 4-bit	576 4-bit
MANTIS			896 AND
		192 1-bit	224 4-bit
CRAFT			1920 AND
		192 1-bit	480 4-bit

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

(a) The 4-bit S-box of TWINE [49] 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.

Fig. 7: 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.

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.

Performance. The gate counts from Table 4 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 5 lists the symmetric 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 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 A.

Table 5: 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 [51]	TWINE-80 [49]	×1.46	×0.79	×9.81
RR21 [46]		×2.19	×0.59	×14.71
ZRE15	TWINE-128	×1.44	×0.78	×9.05
RR21		×2.16	×0.59	×13.58
ZRE15	Fides-80 [11]	×1.23	×0.64	×15.45
RR21		×1.84	×0.48	×23.18
ZRE15	Fides-96	×2.10	×1.07	×50.26
RR21		×3.15	×0.81	×75.39
ZRE15	WAGE [1]	×1.51	×0.76	×72.87
RR21		×2.27	×0.57	×109.30
ZRE15	AES-128 [42]	×0.28	×0.14	×26.23
RR21		×0.42	×0.10	×39.34
ZRE15	Piccolo-80 [48]	×0.66	×0.35	×4.71
RR21		×0.98	×0.26	×7.06
ZRE15	Piccolo-128	×0.65	×0.35	×4.55
RR21		×0.98	×0.26	×6.83
ZRE15	Midori64 [5]	×0.94	×0.52	×5.33
RR21		×1.41	×0.39	×8.00
ZRE15	SKINNY-64-64 [7]	×0.98	×0.53	×7.11
RR21		×1.48	×0.40	×10.67
ZRE15	SKINNY-64-128	×0.66	×0.36	×4.68
RR21		×0.99	×0.27	×7.02
ZRE15	MANTIS [7]	×0.90	×0.50	×4.31
RR21		×1.35	×0.38	×6.46
ZRE15	CRAFT [8]	×0.95	×0.52	×5.71
RR21		×1.43	×0.39	×8.57

7 Conclusion

We presented a garbling scheme that encodes n-bit numbers 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 numbers from n- to m-bit, or to compute arbitrary n- to m-bit functions, while XOR is free. We prove the scheme secure under assumption of a n-TCCR secure hash function, a generalization of TCCR security. Further, we show equivalence of the two notions which allows any TCCR-secure hash function construction to be used in our scheme as well.

Circuits that have separated non-linear and linear parts can be efficiently expressed using projection gates and XOR gates. Our scheme offers high-speed evaluation that is traded off with moderate additional garbling and/or communication cost when compared to AND gate-based circuits. 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. Furthermore, for an important application in two-party secure function evaluation, the evaluation of symmetric primitives, we show that substitution-permutation network primitives can be efficiently implemented in our scheme. We obtained a considerable performance improvement, a 4- to 72-times faster online phase, for nine primitives in literature.

References

1. AlTawy, R., Gong, G., Mandal, K., Rohit, R.: Wage: An authenticated encryption with a twist. IACR Transactions on Symmetric Cryptology pp. 132–159 (2020). https://doi.org/10.46586/tosc.v2020.iS1.132-159
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. p. 535–548. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516738
3. Ashur, T., Cohen, E., Hazay, C., Yanai, A.: A new framework for garbled circuits. Cryptology ePrint Archive, Report 2021/739 (2021), `https://eprint.iacr.org/2021/739`
4. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for boolean and arithmetic circuits. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 565–577. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978410
5. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: ASIACRYPT (2). pp. 411–436. Springer (2015). https://doi.org/10.1007/978-3-662-48800-3_17
6. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing. p. 503–513. STOC '90, Association for Computing Machinery, New York, NY, USA (1990). https://doi.org/10.1145/100216.100287
7. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 123–153. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_5

8. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: Craft: lightweight tweakable block cipher with efficient protection against dfa attacks. IACR Transactions on Symmetric Cryptology **2019**(1), 5–45 (2019). https://doi.org/10.13154/tosc.v2019.i1.5-45
9. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy. p. 478–492. SP '13, IEEE Computer Society, USA (2013). https://doi.org/10.1109/SP.2013.39
10. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. p. 784–796. CCS '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2382196.2382279
11. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 142–158. Springer (2013). https://doi.org/10.1007/978-3-642-40349-1_9
12. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: International Symposium on Experimental Algorithms. pp. 178–189. Springer (2010). https://doi.org/10.1007/978-3-642-13193-6_16
13. Chen, D., Chen, W., Chen, J., Zheng, P., Huang, J.: Edge detection and image segmentation on encrypted image with homomorphic encryption and garbled circuit. In: 2018 IEEE International Conference on Multimedia and Expo (ICME). pp. 1–6. IEEE (2018). https://doi.org/10.1109/ICME.2018.8486551
14. Chen, Y.L., Tessaro, S.: Better security-efficiency trade-offs in permutation-based two-party computation. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 275–304. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_10
15. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.S.: On the security of the "Free-XOR" technique. In: Theory of Cryptography. Lecture Notes in Computer Science, vol. 7194, pp. 39–53. Springer (2012). https://doi.org/10.1007/978-3-642-28914-9_3
16. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 167–187. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-63688-7_6
17. Dessouky, G., Koushanfar, F., Sadeghi, A.R., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: 24. Network and Distributed System Security Symposium (NDSS'17). Internet Society (2017). https://doi.org/10.14722/ndss.2017.23097
18. Durak, F.B., Guajardo, J.: Improving the efficiency of AES protocols in multi-party computation. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 229–248. Springer Berlin Heidelberg, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_11
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. p. 218–229. STOC '87, Association for Computing Machinery, New York, NY, USA (1987). https://doi.org/10.1145/28395.28420
20. Gueron, S., Lindell, Y., Nof, A., Pinkas, B.: Fast garbling of circuits under standard assumptions. Journal of Cryptology **31**, 798–844 (2018). https://doi.org/10.1007/s00145-017-9271-y

21. Guo, C., Katz, J., Wang, X., Weng, C., Yu, Y.: Better concrete security for half-gates garbling (in the multi-instance setting). In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 793–822. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-56880-1_28
22. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 825–841. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00016
23. Gupta, T., Fingler, H., Alvisi, L., Walfish, M.: Pretzel: Email encryption and provider-supplied functions are compatible. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 169–182 (2017). https://doi.org/10.1145/3098822.3098835
24. Heath, D., Kolesnikov, V.: One hot garbling. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 574–593. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484764
25. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security Symposium. vol. 201, pp. 331–335 (2011)
26. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 18–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_2
27. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 458–475. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_26
28. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Theory of Cryptography Conference. pp. 600–620. Springer (2013). https://doi.org/10.1007/978-3-642-36594-2_34
29. Jagadeesh, K.A., Wu, D.J., Birgmeier, J.A., Boneh, D., Bejerano, G.: Deriving genomic diagnoses without revealing patient genomes. Science **357**(6352), 692–695 (2017). https://doi.org/10.1126/science.aam9710
30. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4515, pp. 97–114. Springer (2007). https://doi.org/10.1007/978-3-540-72540-4_6
31. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. IACR Trans. Symmetric Cryptol. **2017, Issue 4**, 130–168 (2017). https://doi.org/10.13154/tosc.v2017.i4.130-168
32. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of AES and DES using lookup tables. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) Applied Cryptography and Network Security. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-61204-1_12
33. Kempka, C., Kikuchi, R., Suzuki, K.: How to circumvent the two-ciphertext lower bound for linear garbling schemes. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 967–997. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_32

34. Kim, H.J., Kim, H.I., Chang, J.W.: A privacy-preserving kNN classification algorithm using yao's garbled circuit on cloud computing. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). pp. 766–769. IEEE (2017). https://doi.org/10.1109/CLOUD.2017.110
35. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: Flexible garbling for XOR gates that beats Free-XOR. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 440–457. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_25
36. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) Automata, Languages and Programming. pp. 486–498. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
37. Lindell, Y., Pinkas, B.: A proof of yao's protocol for secure two-party computation. Journal of Cryptology **22**(2), 161–188 (2009). https://doi.org/10.1007/s00145-008-9036-8
38. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. Journal of Cryptology **25**(4), 680–722 (2012). https://doi.org/10.1007/s00145-011-9107-0
39. Mandal, K., Gong, G.: Can LWC and PEC be friends?: Evaluating lightweight ciphers in privacy-enhancing cryptography. In: Fourth Lightweight Cryptography Workshop. NIST (2020), https://csrc.nist.gov/Events/2020/lightweight-cryptography-workshop-2020
40. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 591–602. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813705
41. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce. p. 129–139. EC '99, Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/336992.337028
42. National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publications 197 (2001)
43. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) Theory of Cryptography. pp. 368–386. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_22
44. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. pp. 250–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15
45. Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005), https://eprint.iacr.org/2005/187
46. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 94–124. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-84242-0_5
47. shelat, A., hao Shen, C.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT

2011. pp. 386–405. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_22

48. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: International workshop on cryptographic hardware and embedded systems. pp. 342–357. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_23
49. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A lightweight block cipher for multiple platforms. In: International Conference on Selected Areas in Cryptography. pp. 339–354. Springer (2012). https://doi.org/10.1007/978-3-642-35999-6_22
50. Yao, A.C.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science. p. 162–167. SFCS '86, IEEE Computer Society, USA (1986). https://doi.org/10.1109/SFCS.1986.25
51. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 220–250. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8

A Appendix

In the followin, we give a more detailed explanation of the implementation from Tables 4 and 5 for each primitive.

A.1 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. 7a), or one 4-bit projection gate.

A.2 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. 8), 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.

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

Fig. 8: The 5-bit S-box of Fides [11] can be computed with 10 AND gates. Input bits are $x_0, \ldots, x_4$, output bits are $x'_0, \ldots, x'_4$.

A.3 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:

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

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.

A.4 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 [12]. 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 $\mathbb{GF}(2^8)$.

A.5 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.

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

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

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

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 $\mathbb{GF}(2^4)$.

A.6 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. 7b), or one 4-bit projection gate.

A.7 SKINNY

The SKINNY cipher family comprises three tweakey sizes, 64, 128 and 192 bit, of which we include the sizes 64 and 128 bit here. The key schedule for SKINNY-64-64 is a permutation on 4-bit wires, thus costing only the initial wire composition. 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 implementable with 4 AND gates using the formula from the specification [7], or one 4-bit projection gate.

A.8 MANTIS

The key $k = k_0 || k_1$ is expanded as defined in [7]:

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

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. 7b), or one 4-bit projection gate.

A.9 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. 7b), or one 4-bit projection gate.