Protected Hardware Implementation of WAGE

Yunsi Fei[1] and Guang Gong[2] and Cheng Gongye[1] and Kalikinkar Mandal[3] and Raghvendra Rohit[2] and Tianhong Xu[1] and Yunjie Yi[2] and Nusa Zidaric[2]

[1] Department of Electrical and Computer Engineering, Northeastern University, 360 Huntington Ave., Boston, MA, 02115, USA
[2] Department of Electrical and Computer Engineering, University of Waterloo, Ontario, N2L 3G1, Canada
[3] Faculty of Computer Science, University of New Brunswick, Fredericton, E3B 5A3, Canada

Abstract. WAGE is a hardware-oriented authenticated cipher, which has the smallest hardware cost (for 128-bit security level) among the round 2 candidates of the NIST lightweight cryptography (LWC) competition. Recently, we, as the same group of authors, have provided analysis of the security of WAGE against the correlation power analysis (CPA) on ARM Cortex-M4F microcontroller, proposed the first optimized masking scheme of WAGE in the t-strong non-interference (SNI) security model, and given a brief evaluation on the hardware performance of WAGE. In this work, we present a detailed side-channel protected implementation of WAGE in the t-SNI security model. We provide optimized gate count in terms of algebraic normal form (ANF) for WGP S-boxes and evaluate the hardware performance of protected WAGE for 1, 2, and 3-order security and provide a comparison with other NIST LWC round 2 candidates. For instance, for the 1-order protection, the protected WAGE consumes an area of about 11.2 kGE in STMicro 65 nm technology.

Keywords: Authenticated encryption, WAGE, Side-channel attack, Masking scheme

1 Introduction

Side-channel analysis is a class of attacks that exploit the implementation and physical execution of a cryptographic algorithm to extract secret information through the power consumption [28] or electro-magnetic emanations [22,3]. Starting from the seminal work of Kocher *et al.* [28], there has been an active research on evaluating the security of ciphers against differential power analysis (DPA) and its variant correlation power analysis (CPA) [12]. In general, a DPA attack aims to recover the secret information by analyzing the differences in power consumption for varying input data, while a CPA attack focuses on the correlation factor between the hamming weight of handled (unknown) data and power samples. Several standardized encryption algorithms and hash functions such as DES, AES, KECCAK and ASCON have been analyzed against

such attacks and countermeasures of side-channel attacks have been proposed [5,4,29,11,35,26,23,24].

An effective countermeasure against side-channel attacks exploiting the power consumption is *masking*. Any linear operation over the shares can be masked linearly, however processing nonlinear operations such as AND and/or S-box is complex. The design of efficient secure masking schemes for nonlinear operations is a challenging task. Ishai, Sahai and Wagner (ISW) [27] have initiated the study of securely computing a circuit consisting of XOR, AND and NOT gates where the AND gates are replaced by secure AND gadgets. From the security point of view, the ISW construction is resistant to the t-order probing attack when the number of shares $n \geq 2t + 1$, i.e., evaluating the leakage on a set of at most t out of n points does not reveal information about a sensitive variable x. Barthe *et al.* [8] redefined the ISW security and introduced a stronger security notion, called t-strong non interference (t-SNI) security under the ISW probing model, which minimizes the number of shares to $n = t + 1$ (i.e., almost half) and its compositional security definition guarantees the t-SNI security in a large construction by securely composing t-SNI secure gadgets. Several other techniques for side-channel attacks have been proposed, including Threshold implementation [30], Consolidated Masking Scheme [31], Domain Oriented Masking (DOM) [25] and Unified Masking Approach [24]. In [20], De Cnudde *et al.* presented an AES hardware implementation using $t+1$ shares in the presence of glitches. The countermeasures on the secure evaluation of the AES S-box have been investigated in the literature extensively, e.g., [32,14,17,33,15,16,19,34] based on finite field computations, randomized lookup table, and customized gate-level implementations. In particular, for the randomized lookup table, a first-order countermeasure for S-boxes was first proposed by Chari *et al.* in [15], and later on, in [16], Coron generalized the randomized lookup table countermeasure [16]. In the follow-up work [19], Coron *et al.* proposed a construction of a randomized lookup table countermeasure that is t-SNI secure.

Our contribution. We have provided analysis of the security of WAGE against the correlation power analysis (CPA) on ARM Cortex-M4F microcontroller, proposed the first optimized masking scheme of WAGE in the t-strong non-interference (SNI) security model, and given a brief evaluation on the hardware performance of WAGE in [21]. In this work, we present detailed side-channel implementation results of WAGE in the t-SNI security model. To achieve the area optimized masked S-boxes, we exploit the internal structures of the SB and WGP S-boxes. For SB S-box, we exploit its iterative construction and apply the common share multiplication technique to optimize the area. We obtain an optimized circuit implementation of WGP with help of the VHDL and GAP tools and provide optimized gate counts in terms of algebraic normal form (ANF) for WGP S-boxes. The hardware implementation results of WGP for different approaches are presented. The optimized WGP S-box is implemented using 313 XOR, 172 AND and 66 NOT gates. Our hardware architecture for the masked WAGE is parallel. The protected hardware implementation is built on top of the

original WAGE hardware presented in [1]. We implement the round function for the t-order ($t = 1, 2, 3$) masked WAGE in STMicro 65 nm and TSMC 65 nm technologies and provide area results for WGP and SB S-boxes and the WAGE authenticated encryption (AE) scheme in Table 4. For instance, our smallest implementation of WAGE AE has a cost of 11.2 kGE for the 1-order protection in STMicro 65 nm technology. We provide a comparison [4] of WAGE AE with the currently known available first-order protected implementations of the NIST LWC round 2 candidates in Table 1.

In this paper, we also provide the ANF expressions for each component of WGP (Appendix A.1), and the circuit implementing WGP using AND, XOR, and NOT gates (Appendix A.2).

Table 1: Comparison of WAGE with NIST LWC round 2 candidates for the 1-order protection. Only the results for the round based implementation of primary members is listed.

Algorithm	Ref.	Impl. type	Technology	Synthesis	Area [GE]
WAGE AE	**Section 4**	Masking	STMicro 65 nm TSMC 65 nm	Physical	**11177** **12711**
ASCON	[26]	Threshold	UMC 90 nm	Physical	28610
SKINNY-AEAD	[10]	DOM	UMC 90 nm IBM 130 nm	- -	20534 18817
GIFT-COFB	[7]	Threshold	STMicro 90 nm	Logic	13131
SUNDAE-GIFT	[13]	Threshold	TSMC 90 nm	Logic	13297

2 Preliminaries

Notations. Let $\mathbb{F}_2 = \{0, 1\}$ be the Galois field, and $\mathbb{F}_{2^7}$ be an extension field where each element is a tuple of 7 bits. $\mathbb{F}_2^m$ is a vector space of dimension m. $\oplus$ and $\cdot$ denote the bitwise XOR and bitwise AND operations, respectively. Double square brackets $[[x]] = (x^1, x^2, \cdots, x^n)$ denotes the additive shares of $x = \oplus_{i=1}^{n} x^i$. $r \xleftarrow{\$} \mathbb{F}_p$ denotes the element r is chosen from $\mathbb{F}_p$ uniformly at random.

2.1 Description of WAGE

We provide a description of WAGE, following the same notations from [1,6]. The WAGE authenticated encryption is built upon the WAGE permutation in the unified sponge duplex mode where the WAGE permutation is a 111-round of an iterative permutation with a state width of 259 bits over an extension field $\mathbb{F}_{2^7}$. The core components of the permutation, described in detail below, include two different S-boxes (WGP and SB), a linear feedback function defined over $\mathbb{F}_{2^7}$,

[4] A fair comparison is difficult due to different types of side-channel implementations and ASIC libraries.

five word-wise XORs, and 111 pairs of 7-bit round constant (rc_1, rc_0). Figure 1 provides an overview of the round function of the WAGE permutation.

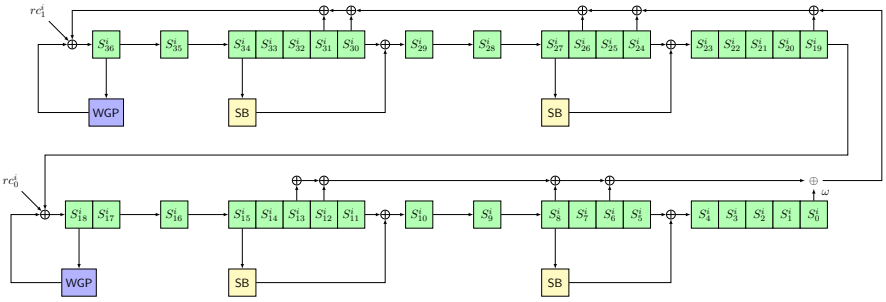

Fig. 1: An overview of the state update function of WAGE [6].

Nonlinear components of WAGE. WAGE uses two distinct 7-bit S-boxes, namely WGP and SB where WGP is defined over a finite field $\mathbb{F}_{2^7}$ and SB is constructed iteratively at the bit-level from quadratic functions. We now provide a brief description of WGP and SB.

Welch-Gong permutation (WGP). The WGPerm, denoted by WGP7, is defined over $\mathbb{F}_{2^7}$ which is given by

$$\mathsf{WGP7}(x) = x + (x+1)^{33} + (x+1)^{39} + (x+1)^{41} + (x+1)^{104},\ x \in \mathbb{F}_{2^7}$$

where $\mathbb{F}_{2^7}$ is defined by the primitive polynomial $x^7+x^3+x^2+x+1$. WGP is constructed from WGP7 by applying decimation $d = 13$ as $\mathsf{WGP}(x) = \mathsf{WGP7}(x^{13})$.

SB S-box. The 7-bit S-box SB is constructed in an iterative way using the nonlinear transformation Q and the bit permutation P which are given by

$$Q(x_0, x_1, \cdots, x_5, x_6) = (x_0 \oplus (x_2 \quad x_3), x_1, x_2, \overline{x}_3 \oplus (x_5 \quad x_6), x_4, \overline{x}_5 \oplus (x_2 \quad x_4), x_6)$$
$$P(x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (x_6, x_3, x_0, x_4, x_2, x_5, x_1).$$

The construction of SB is given by

$$\begin{aligned}(x_0, x_1, x_2, x_3, x_4, x_5, x_6) &\leftarrow R^5(x_0, x_1, x_2, x_3, x_4, x_5, x_6)\\ (x_0, x_1, x_2, x_3, x_4, x_5, x_6) &\leftarrow Q(x_0, x_1, x_2, x_3, x_4, x_5, x_6)\\ x_0 &\leftarrow x_0 \oplus 1; x_2 \leftarrow x_2 \oplus 1\end{aligned}$$

where the round R is a composition of Q and P, i.e., $R = P \circ Q$.

State update function of WAGE. The 259-bit state of WAGE consists of 37 7-bit words and is denoted by $\mathbf{S} = (S_{36}, \cdots, S_0)$ where each S_i is of 7 bits.The state update function of WAGE, denoted by WAGE_STATEUPDATE, takes as inputs the current state S and a pair of round constants (rc_1, rc_0), and updates the state with the following three steps:

1. **Computing linear feedback: $fb \leftarrow \mathbf{FB}(\mathbf{S})$.** The following primitive polynomial of degree 37 over $\mathbb{F}_{2^7}$ is used as a feedback function

$$\ell(y) = y^{37} + y^{31} + y^{30} + y^{26} + y^{24} + y^{19} + y^{13} + y^{12} + y^8 + y^6 + \omega$$

where ω is a root $x^7 + x^3 + x^2 + x + 1$, which is a primitive polynomial defining $\mathbb{F}_{2^7}$. The feedback computation is given by

$$fb = S_{31} \oplus S_{30} \oplus S_{26} \oplus S_{24} \oplus S_{19} \oplus S_{13} \oplus S_{12} \oplus S_8 \oplus S_6 \oplus (\omega \otimes S_0).$$

For an input $x \in \mathbb{F}_{2^7}$, the multiplier ω maps x to $\omega \otimes x$, i.e., $x \mapsto \omega \otimes x$. The ANF representation of it is given by

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6) \otimes \omega \rightarrow (x_6, x_0 \oplus x_6, x_1 \oplus x_6, x_2 \oplus x_6, x_3, x_4, x_5).$$

2. **Updating intermediate words and adding round constants: $(\mathbf{S}, fb) \leftarrow \mathbf{IWRC}(\mathbf{S}, fb, rc_0, rc_1)$.**

$$\begin{aligned} S_5 &\leftarrow S_5 \ \oplus \mathsf{SB}(S_8) \\ S_{11} &\leftarrow S_{11} \oplus \mathsf{SB}(S_{15}) \\ S_{19} &\leftarrow S_{19} \oplus \mathsf{WGP}(S_{18}) \oplus rc_0 \\ S_{24} &\leftarrow S_{24} \oplus \mathsf{SB}(S_{27}) \\ S_{30} &\leftarrow S_{30} \oplus \mathsf{SB}(S_{34}) \\ fb &\leftarrow fb \oplus \mathsf{WGP}(S_{36}) \oplus rc_1. \end{aligned}$$

3. **Shifting register contents and update the last word: $\mathbf{S} \leftarrow \mathbf{Shift}(\mathbf{S}, fb)$.**

$$\begin{aligned} S_j &\leftarrow S_{j+1}, 0 \le j \le 35 \\ S_{36} &\leftarrow fb. \end{aligned}$$

On an input state $\mathbf{S}$, the output of the WAGE permutation is obtained by applying the state update function 111 times. Note that only the IWRC transformation performs the nonlinear operations and the others are linear operations.

2.2 Adversarial Model

We consider an adversarial model in which an attacker can probe up to t intermediate variables in the circuit, known as the t-probing (ISW) model [27] or t-non interference (NI) model [9] where the number of shares for each secret variable is $n \geq 2t + 1$. The t-probing security is provided in Definition 1. The security of t-strong non interference (t-SNI) was introduced in [8] (see Definition 3). Intuitively, a t-SNI gadget information-theoretically hides dependencies between each of its inputs and its outputs, even in the presence of internal probes [8]. Note that combining t-probing (or t-NI) secure gadgets does not necessarily results in a t-probing secure algorithm [18]. We consider the standard t-SNI security for WAGE as the number of shares is only $n = t + 1$, instead of $n = 2t + 1$ in the t-NI security and it provides an assurance on the t-SNI security of the entire scheme when t-SNI secure gadgets are composed securely.

Definition 1 (t-probing Security). *[27] An algorithm $\mathcal{C}$ is t-probing secure if the values taken by at most t intermediate variables of $\mathcal{C}$ during its execution do not leak any information about secrets.*

Definition 2 (t-NI Security). *[8] Let $\mathcal{G}$ be a gadget accepting $(x_i)_{1\leq i\leq n}$ as input and outputting $(y_i)_{1\leq i\leq n}$. We call the gadget $\mathcal{G}$ is t-*non interference *(t-NI) (also known as t-threshold probing) secure if for any set of $\ell \leq t$ intermediate variables, there exists a subset I of input indices with $|I|\leq \ell$ such that ℓ intermediate variables can be perfectly simulated from $x_{|I} = (x_i)_{i\in I}$.*

Definition 3 (t-SNI Security). *[8] Let $\mathcal{G}$ be a gadget accepting $(x_i)_{1\leq i\leq n}$ as input and outputting $(y_i)_{1\leq i\leq n}$. We call the gadget $\mathcal{G}$ is t-*strongly non-interference *(t-SNI) secure if for any set of $\ell \leq t$ intermediate variables and any subset of output indices O such that $\ell + |O|\leq t$, there exists a subset I of input indices with $|I|\leq \ell$ such that the ℓ intermediate variables and output variables $y_{|O}$ can be perfectly simulated from $x_{|I}$.*

2.3 Masking Schemes for Side-channel Countermeasures

Masking is an effective countermeasure against side-channel attacks such as power analysis on cryptographic algorithms. In a masking scheme, a variable x containing sensitive information is protected by masking it with a random value r as $x' = x \oplus r$, i.e., $x = x' \oplus r$, meaning the sensitive variable x is shared between variables r and x'. In an n-order masking, each sensitive variable x is shared among n variables x^i as $x = x^1 \oplus x^2 \oplus \cdots \oplus x^n$. We denote the n shares of x by $[[x]] = (x^1, x^2, \cdots, x^n)$ such that $x = \oplus_{i=1}^{n} x^i$. For $[[x]] = (x^1, x^2, \cdots, x^n)$ and $[[y]] = (y^1, y^2, \cdots, y^n)$, $[[x]] \oplus [[y]] = (x^1 \oplus y^1, x^2 \oplus y^2, \cdots, x^n \oplus y^n) = [[x \oplus y]]$. For a binary variable x with shares $[[x]]$, it is easy to compute $[[\bar{x}]]$ from $[[x]]$ as $\bar{x} = \bar{x}^1 \oplus x^2 \oplus \cdots \oplus x^n$. The nonlinear operations such as AND and S-box are most complex operations when sensitive variables are additively shared.

3 The Masking Scheme for WAGE

High-level description. We construct a t-SNI secure masking scheme of WAGE where the number of shares $n = t + 1$. In doing so, the state of WAGE, denoted by $\mathbf{S}$, is split into n state shares such that $\mathbf{S} = \mathbf{S}^1 \oplus \mathbf{S}^2 \oplus \cdots \oplus \mathbf{S}^n$ where $\mathbf{S}^i = (S^i_{36}, \cdots, S^i_0)$ is the i-th share of the state $\mathbf{S}$. In our masked WAGE, we update the shared states according to the round function so that at the end of 111 rounds, the state of the WAGE permutation can be constructed from the n state shares. The operations involved in the round function of WAGE are the computation of the linear feedback function, computing WGP and SB and updating intermediate words, and shifting operation where all the operations in the round function on the shared states are performed independently and in parallel, except for WGP and SB. As the transformation ω-multiplier is linear, the feedback computation is linear and can be performed in parallel on the n shared states, i.e., for each shared state, the corresponding feedback is computed

as $fb^i \leftarrow \mathsf{FB}(\mathbf{S}^i), i \in \{1, \cdots, n\}$. To optimize the hardware area for SB with the SNI security, we exploit the iterative construction of SB and apply the common share technique for two AND operations in the Q transformation. For the t-SNI secure WGP, we use the randomized lookup table from [19], and develop a new optimized gate-level implementation of WGP in which we replace the AND gates by t-SNI secure AND gadgets of [8].

The evaluation of the masked SB, denoted by SecSB, is performed in a single clock cycle although our technique of the masked SB is iterative. The gate-level implementation of the masked WGP, denoted by SecWGP, is also computed in one clock cycle. On the other hand, the randomized lookup table approach for secure WGP and SB S-boxes takes at least 128 cycles, and was used for the software implementation. The masked S-boxes are computed as $[[S_j]] \leftarrow \mathsf{SecSB}([[S_j]]), j \in \{8, 15, 27, 34\}$ and $[[S_j]] \leftarrow \mathsf{SecWGP}([[S_j]]), j \in \{18, 36\}$ where $S_j = \oplus_{i=1}^{n} S_j^i$. This operation can be performed parallelly on the shared states, i.e., $\mathbf{S}^i \leftarrow \mathsf{Shift}(\mathbf{S}^i), i \in \{1, \cdots, n\}$.

Protected WAGE algorithm. Our masked WAGE is designed to provide a t-order protection against side-channel attacks such as power analysis. Our hardware architecture for the masked WAGE is parallel and designed to be low-latency. Algorithm 1 describes the pseudocode of the masked WAGE permutation. In each round of the masked WAGE, the state is shared among n state shares ($\mathbf{S}^i$) where the feedback computations in Lines 5-7, updating intermediate words in Lines 10-14 and shift operations in Lines 20-22 that are linear are computed in parallel. Our architecture uses corresponding masked WGP and SB in Lines 8-9 for the S-box operations, which are evaluated in parallel. Note that the pair of round constants at each round are added to only one share (say $\mathbf{S}^1$) in Lines 15-16. A high-level overview of the architecture of the masked WAGE permutation for the first-order protection is shown in Figure 2. For a low-latency implementation of the masked WAGE, the circuit level implementation of the masked WGP is used as it can be computed in one clock cycle. For software implementations, to avoid bit level operations, we use the randomized lookup table for the secure evaluation of both WGP and SB. For the details about the constructions of SecWGP and SecSB and the security of the scheme, we refer the reader to the full paper [21].

Complexity. We now provide the amount of random bits required for the masked WAGE permutation in terms of the number of shares n. The randomness amount can be computed by calculating the unit operations of different gadgets. The RefreshMask, CommonMult and SecMult gadgets consume $\frac{7n(n-1)}{2}$, $\frac{3n(n-1)}{2}$ and $\frac{n(n-1)}{2}$ bits, respectively. Thus, the total number of random bits for SecSB is $33n(n-1)$ bits (asymptotically $O(n^2)$). According to [19], the number of random bits for SecWGP with the randomized lookup table is $\frac{64n(n-1)(2n-1)}{3}$ (asymptotically $O(n^3)$). On the other hand, the number of random bits for the gate-level implementation of SecWGP is $87n(n-1)$ bits. For each round of WAGE with masked WGP implemented using gate-level, the amount of bits is $(33n(n-1)\times 4 + 87n(n-1)\times 2) = 255n(n-1)$, thus for evaluating the WAGE permutation

Algorithm 1 The Masked WAGE

Input: $[[\mathbf{S}]] = (\mathbf{S}^1, \mathbf{S}^2, \cdots, \mathbf{S}^n)$
Output: $[[\mathbf{S}]]$ where $\mathbf{S} \leftarrow \text{WAGE\_STATEUPDATE}^{111}(\mathbf{S})$
procedure MASKED_WAGE( )
  for $i = 0$ to 110 **do**
    for $j = 1$ to n **do**
      $fb^j \leftarrow \mathsf{FB}(\mathbf{S}^j)$
    end for
    $[[S_j]] \leftarrow \mathsf{SecSB}([[S_j]]), j \in \{8, 15, 27, 34\}$
    $[[S_j]] \leftarrow \mathsf{SecWGP}([[S_j]]), j \in \{18, 36\}$
    $[[S_5]] \leftarrow [[S_5]] \oplus [[S_8]]$
    $[[S_{11}]] \leftarrow [[S_{11}]] \oplus [[S_{15}]]$
    $[[S_{19}]] \leftarrow [[S_{19}]] \oplus [[S_{18})]]$
    $[[S_{24}]] \leftarrow [[S_{24}]] \oplus [[S_{27}]]$
    $[[S_{30}]] \leftarrow [[S_{30}]] \oplus [[S_{34}]]$
    $S^1_{19} \leftarrow S^1_{19} \oplus rc_0$
    $fb^1 \leftarrow fb^1 \oplus tmp^1 \oplus rc_1$
    for $j = 2$ to n **do**
      $fb^j \leftarrow fb^j \oplus tmp^j$
    end for
    for $j = 1$ to n **do**
      $\mathbf{S}^j \leftarrow \mathsf{Shift}(\mathbf{S}^j, fb^j)$
    end for
  end for
  return $[[\mathbf{S}]] = (\mathbf{S}^1, \mathbf{S}^2, \cdots, \mathbf{S}^n)$ s.t. $\mathbf{S} = \bigoplus_{i=1}^{n} \mathbf{S}^i$
end procedure

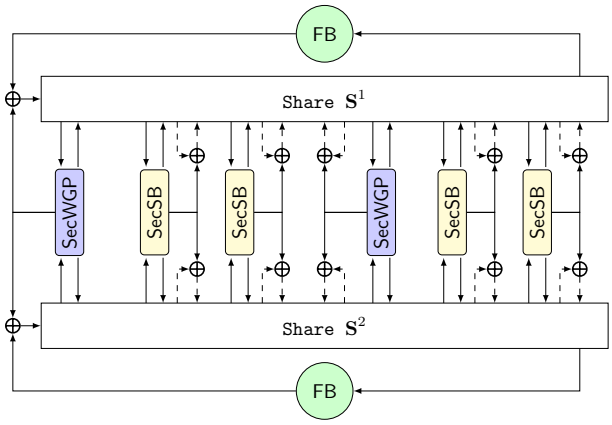

Fig. 2: Schematic of the masked WAGE permutation for 1-order protection.

the total number of random bits is $255 \times 111 \times n(n-1)$. The (asymptotic) time complexity of the masked WAGE is $O(n^2)$ when SecWGP is implemented using a Boolean circuit.

4 Hardware Implementation Results

In this section, we provide the implementation results of the protected WAGE for different masking orders in hardware. Our simulations were done in Mentor Graphics ModelSim SE v10.7c and logic synthesis was performed with Synopsys Design Compiler version P-2019.03 (using the `compile_ultra` command). For the physical synthesis Cadence Encounter v14.13 was used. We used two 65 nm ASIC cell libraries: ST Microelectronics 65 nm and TSMC 65 nm.

Protected parallel WAGE architecture. The protected hardware implementation is built on top of the original WAGE hardware presented in [1]. The datapath was modified for a given number of shares n. This includes the LFSR feedback XOR gates and the multiplication by the constant ω, the XOR gates and multiplexers needed for the nonlinear inputs from the two WGP permutations and the four SB Sboxes, and multiplexers needed to support the mode. A section of the WAGE LFSR with stages S_9 to S_4 is shown in Figure 3. Every stage holds n elements of the finite field $\mathbb{F}_{2^7}$, shown on top of one another. The $S_9 \rightarrow S_4$ section of the WAGE LFSR contains the D_0/O_0 port and multiplexers needed to support the mode ($\texttt{Amux}_0$ for absorbing and $\texttt{RLmux}_0$ for replacing and loading). The connections, XOR gates, multiplexers, and input/output ports are now also $n \times 7$ bits wide.

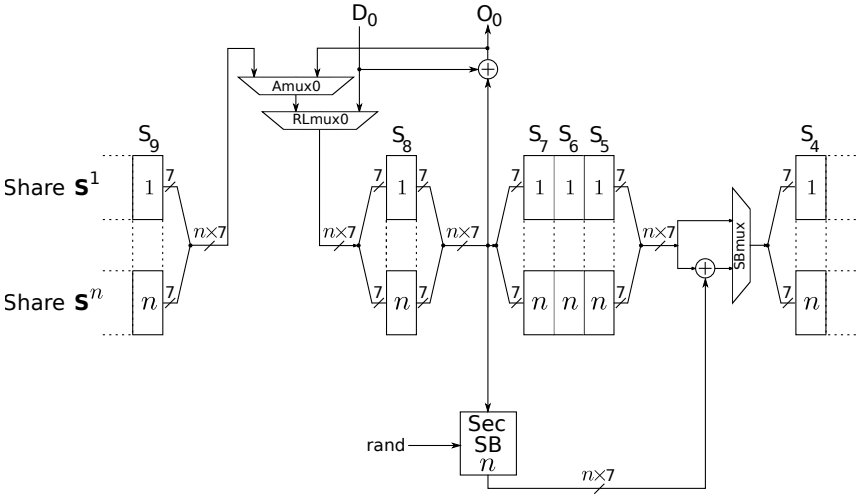

Fig. 3: Parallel architecture of the protected WAGE.

The n shares of stage S_8 are used as an input to the SB, and its n output shares are added to n shares of stage S_5, requiring an $n \times 7$ XOR gate and a

multiplexer SBmux, needed to switch between the WAGE permutation and the loading and tag extraction phase of WAGE. For simplicity, the control signals for the multiplexers are omitted from Figure 3.

The unprotected WAGE LFSR has a post-PAR area of 2120 GE (ST Micro 65nm). For protected implementations, the area grows to 4256 GE for $n = 2$, 6471 GE for $n = 3$ and 8648 GE for $n = 4$. The nonlinear components (WGP and SB), and the LFSR for generating the constants, are not included in the above area numbers.

As shown in Figure 3, the SB has extra inputs for the random bits, in addition to the $n \times 7$ bits input and output. The random bits are needed for SecMult multiplication gadgets (Algorithm 3) and for common share CS_SecMult multiplication gadgets (Algorithm 5). The number of random bits needed depends on the number of shares n. The design flow of WGP is summarized below.

Design flow of WGP. The design flow for the protected implementation of a WGP module was as follows. First, an algebraic normal form for every component of the WGP output was obtained and implemented using custom GAP packages [**?**,36]. The ANF expressions for each component of the output are listed in Appendix A.1. This circuit was run through synthesis tools restricted to use only AND, OR, XOR, and NOT gates (AOXI[5]). The OR gates were then manually replaced by AND and NOT gates and further optimized. The final circuit has 172 AND, 313 XOR, 66 NOT gates (AXI), and post-PAR area of 759 GE. The circuit is given in Tables 5-9 in Appendix A.2. The detailed step by step results are given in Table 2.

Table 2: WGP hardware area for different steps (STMicro 65 nm): constant array (any gates available in the ASIC library), ANF (Algebraic Normal Form: AND, XOR), AOIX (AND, OR, XOR, INV), and AXI (AND, XOR, NOT).

Implementation Approach	Area [GE]	Number of gates AND	OR	XOR	INV	Comment
Constant array	258	–				–
ANF	958	1132	–	439	–	App. A.1
AOIX	825	146	30	310	39	–
AXI	759	172	–	313	66	Tables 5-9

Implementation results. Our preliminary implementation results are provided in Tables 3 and 4. Table 3 shows the comparison of two SecMult multiplication gadgets (Algorithm 3) with a common share CS_SecMult multiplication gadget (Algorithm 5). The two multiplication gadgets in 2x SecMult were implemented with a common input. In Table 4, we include a detailed break down of area and combinational delay (resp. clock period) for $n = 2, 3, 4$ shares (resp. $t = 1, 2, 3$-order protection). For both technologies, the middle column indicates

[5] as VHDL is not case-sensitive, INV is used instead of NOT to avoid confusion with number of shares n

the area overhead compared to the unprotected module. To accurately show the scaling with n, all implementations assume an environment capable of providing random bits. Further details are omitted for brevity.

Figure 4 is showing the breakdown of hardware implementation area (STMicro 65 nm post-PAR results) by major components of WAGE. For unprotected WAGE, the LFSR including the multiplexers needed to support the mode has the biggest area contribution. As the number of shares n increases, the WGP becomes the largest component of WAGE.

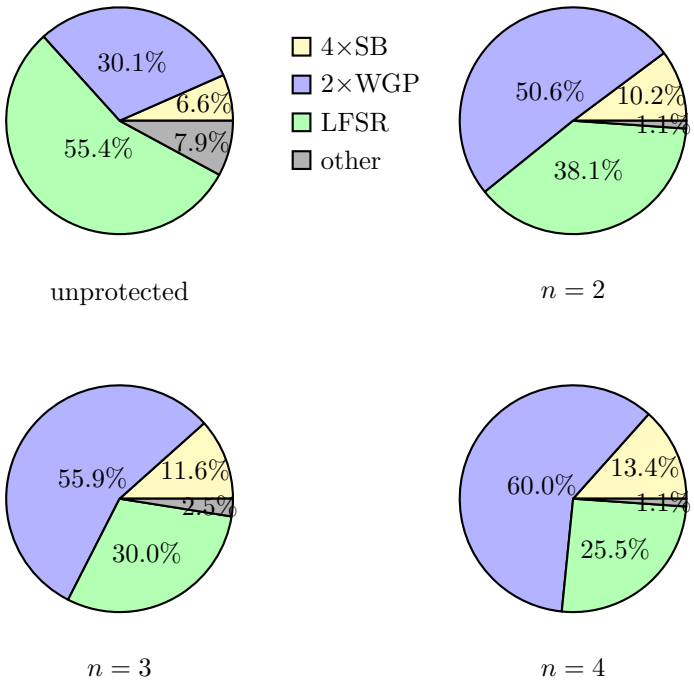

Fig. 4: Hardware implementation area breakdown by components

Table 3: Area comparison [GE] of two SecMult multiplication gadgets with a common share CS_SecMult multiplication gadget.

	STMicro 65 nm			TSMC 65 nm		
	$n = 2$	$n = 3$	$n = 4$	$n = 2$	$n = 3$	$n = 4$
2x SecMult	24	59	119	24	59	130
CS_SecMult	12	69	98	13	69	105

Table 4: Implementation results of S-boxes (WGP and SB) and the WAGE AE in ASIC. Italic denotes the use of the common share multiplication gadget.

Algorithm	STMicro 65 nm			TSMC 65 nm		
	Area [GE]	Area overhead	Delay [ns]	Area [GE]	Area overhead	Delay [ns]
WGP						
Constant array [1,2]	258	-	1.4	270	-	0.9
Unprotected AXI	759 ‡	-	1.9	804	-	1.3
$n = 2$	2830	3.7	2.3	3090	3.8	1.9
$n = 3$	6030	2.1	3.1	6580	8.1	2.1
$n = 4$	10200	1.7	3.7	11400	14.1	2.4
SB						
Unprotected AXI	63	-	0.9	70	-	0.8
$n = 2$	285	4.5	1.7	307	4.3	1.4
	285	*4.5*	*1.9*	*323*	*4.6*	*1.4*
$n = 3$	626	9.9	2.2	677	9.6	1.5
	715	*11.3*	*2.3*	*829*	*11.8*	*1.5*
$n = 4$	1140	18.1	2.3	1200	17.1	1.9
	1275	*20.2*	*2.2*	*1280*	*18.2*	*2.1*
	Area [GE]	Area overhead	Clk. period [ns]	Area [GE]	Area overhead	Clk. period [ns]
WAGE AE						
Constant array [1,2]	2900	-	1.1	3290	-	0.9
Unprotected AXI	3830	-	1.9	4430	-	1.8
$n = 2$	11177	2.9	3.6	12714	*2.9*	2.9
	11177	*2.9*	*2.9*	*12711*	*2.9*	*2.9*
$n = 3$	21566	5.6	5.0	23912	5.4	4.9
	21953	*5.7*	*3.9*	*24174*	*5.5*	*3.4*
$n = 4$	33985	8.9	5.2	*38818*	*8.7*	*4.9*
	34238	*8.9*	*4.6*	*39067*	*8.8*	*4.4*

‡ When restricted to AND, OR, XOR, NOT gates. For arbitrary gates, it drops to 577GE.

5 Conclusions and Future Work

In this paper, we presented the first high-order masking scheme of WAGE and proved its t-SNI security in the ISW probing model to resist against side-channel attacks. We designed the hardware of the masked WAGE in ASIC using STMicro 65 nm and TSMC 65 nm technologies for the 1, 2, and 3-order security and reported the detailed performance results along with a comparison with the other NIST LWC round 2 candidates.

As a future work, we will explore an iterative implementation of SecSB and SecWGP which would allow a smaller number of random bits needed per single

clock cycle. We will investigate tradeoffs among the throughout, hardware area, and the amount of randomness available per single clock cycle.

Acknowledgements. Hardware implementations in this work are based on the original WAGE hardware implementations from [1]. Authors would like to thank Dr. Mark Aagaard for the help with synthesis tools.

References

1. Mark D. Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, Raghvendra Rohit, and Nusa Zidaric. Wage: An authenticated cipher, 2019. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/wage-spec-round2.pdf.
2. Mark D. Aagaard, Marat Sattarov, and Nuša Zidarič. Hardware design and analysis of the ACE and WAGE ciphers. NIST LWC Workshop 2019. Also available at https://arxiv.org/abs/1909.12338.
3. Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side—channel(s). In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
4. Mehdi-Laurent Akkar, Régis Bevan, and Louis Goubin. Two power analysis attacks against one-mask methods. In *International Workshop on Fast Software Encryption*, pages 332–347. Springer, 2004.
5. Mehdi-Laurent Akkar and Christophe Giraud. An implementation of des and aes, secure against some attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 309–318. Springer, 2001.
6. Riham AlTawy, Guang Gong, Kalikinkar Mandal, and Raghvendra Rohit. Wage: An authenticated encryption with a twist. *IACR Transactions on Symmetric Cryptology*, 2020(S1):132–159, Jun. 2020.
7. Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift-cofb. Cryptology ePrint Archive, Report 2020/738, 2020. https://eprint.iacr.org/2020/738.
8. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS 16, page 116129, New York, NY, USA, 2016. Association for Computing Machinery.
9. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 457–485, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
10. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. Skinny-aead and skinny-hash. *IACR Transactions on Symmetric Cryptology*, pages 88–131, 2020.
11. Olivier Benoît and Thomas Peyrin. Side-channel analysis of six sha-3 candidates. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 140–157. Springer, 2010.

12. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.
13. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Energy analysis of lightweight aead circuits. Cryptology ePrint Archive, Report 2020/607, 2020. https://eprint.iacr.org/2020/607.
14. Claude Carlet, Louis Goubin, Emmanuel Prouff, Michael Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *Fast Software Encryption*, pages 366–384, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
15. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
16. Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 441–458, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
17. Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 498–514, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
18. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption*, pages 410–424, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
19. Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):40–72, Feb. 2018.
20. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking aes with d+1 shares in hardware. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS 16, page 43, New York, NY, USA, 2016. Association for Computing Machinery.
21. Yunsi Fei, Guang Gong, Cheng Gongye, Kalikinkar Mandal, Raghvendra Rohit, Tianhong Xu, Yunjie Yi, and Nusa Zidaric. Correlation power analysis and higher-order masking implementation of wage, 2020. In submission at Selected Areas in Cryptography 2020.
22. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 251–261, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
23. Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.
24. Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptographic Engineering*, 8(2):109–124, 2018.
25. Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS 16, page 3, New York, NY, USA, 2016. Association for Computing Machinery.

26. H. Gro, E. Wenger, C. Dobraunig, and C. Ehrenhfer. Suit up! – made-to-measure hardware implementations of ascon. In *2015 Euromicro Conference on Digital System Design*, pages 645–652, 2015.
27. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
28. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.
29. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.
30. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Proceedings of the 8th International Conference on Information and Communications Security*, ICICS06, page 529545, Berlin, Heidelberg, 2006. Springer-Verlag.
31. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 764–783, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
32. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
33. Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of fse 2012. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 417–434. Springer, 2013.
34. Pascal Sasdrich, Begl Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to aes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):300–326, Mar. 2020.
35. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 282–296, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
36. Zidaric, Nusa. Automated design space exploration and datapath synthesis for finite field arithmetic with applications to lightweight cryptography, 2020.

A Different forms of the WGP

This section gives different forms for the $z = \mathsf{WGP}(x)$ used in WAGE. The decimation exponent used is $d = 13$. Finite field $\mathbb{F}_{2^7}$ is represented using the polynomial basis given by the root ω of the defining polynomial $x^7+x^3+x^2+x+1$, namely $\mathrm{PB} = \{1, \omega, \ldots, \omega^6\}$.

A.1 WGP ANF

$$
\begin{aligned}
&z_0 \\
&= x_0x_1x_2x_3x_4x_5 + x_0x_1x_2x_3x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_5 \\
&+ x_0x_1x_2x_3x_6 + x_0x_1x_2x_4x_5 + x_0x_1x_2x_5x_6 + x_0x_1x_3x_4x_6 + x_0x_1x_3x_5x_6 + x_0x_1x_4x_5x_6 \\
&+ x_0x_2x_3x_4x_5 + x_0x_2x_3x_5x_6 + x_1x_2x_3x_4x_6 + x_1x_2x_3x_5x_6 + x_1x_3x_4x_5x_6 + x_2x_3x_4x_5x_6 \\
&+ x_0x_1x_2x_4 + x_0x_1x_2x_5 + x_0x_1x_2x_6 + x_0x_1x_3x_6 + x_0x_1x_4x_5 + x_0x_1x_5x_6 + x_0x_2x_3x_5 + x_0x_2x_5x_6 \\
&+ x_0x_3x_4x_6 + x_1x_2x_3x_5 + x_1x_2x_3x_6 + x_1x_2x_4x_6 + x_1x_3x_4x_5 + x_1x_3x_4x_6 + x_2x_3x_5x_6 + x_3x_4x_5x_6 \\
&+ x_0x_1x_2 + x_0x_1x_3 + x_0x_1x_4 + x_0x_2x_3 + x_0x_2x_4 + x_0x_2x_5 + x_0x_2x_6 + x_0x_3x_5 + x_0x_3x_6 + x_0x_4x_5 \\
&+ x_0x_4x_6 + x_0x_5x_6 + x_1x_2x_4 + x_1x_3x_4 + x_1x_3x_5 + x_1x_3x_6 + x_1x_4x_6 + x_1x_5x_6 + x_2x_3x_5 + x_2x_4x_5 \\
&+ x_2x_4x_6 + x_2x_5x_6 + x_0x_2 + x_0x_4 + x_1x_2 + x_1x_3 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_6 + x_3x_6 + x_4x_6 \\
&+ x_5x_6 + x_0 + x_2 + x_3 + x_4
\end{aligned}
$$

$$
\begin{aligned}
&z_1 \\
&= x_0x_1x_2x_3x_4x_6 + x_0x_1x_2x_3x_5x_6 + x_0x_1x_2x_4x_5x_6 + x_0x_1x_3x_4x_5x_6 + x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_5 \\
&+ x_0x_1x_2x_3x_6 + x_0x_1x_2x_4x_5 + x_0x_1x_2x_4x_6 + x_0x_1x_2x_5x_6 + x_0x_1x_3x_4x_5 + x_0x_1x_3x_5x_6 \\
&+ x_0x_1x_4x_5x_6 + x_0x_2x_3x_4x_6 + x_0x_2x_3x_5x_6 + x_0x_2x_4x_5x_6 + x_1x_2x_3x_4x_5 + x_1x_3x_4x_5x_6 + x_0x_1x_2x_4 \\
&+ x_0x_1x_2x_6 + x_0x_1x_3x_4 + x_0x_1x_3x_5 + x_0x_1x_3x_6 + x_0x_1x_4x_5 + x_0x_1x_4x_6 + x_0x_1x_5x_6 + x_0x_2x_3x_4 \\
&+ x_0x_2x_3x_5 + x_0x_2x_3x_6 + x_0x_2x_4x_5 + x_0x_2x_4x_6 + x_0x_4x_5x_6 + x_1x_2x_3x_6 + x_1x_2x_4x_6 + x_1x_3x_4x_5 \\
&+ x_2x_3x_4x_5 + x_2x_3x_4x_6 + x_3x_4x_5x_6 + x_0x_1x_5 + x_0x_2x_3 + x_0x_3x_5 + x_0x_5x_6 + x_1x_2x_6 + x_1x_3x_6 \\
&+ x_1x_4x_5 + x_1x_4x_6 + x_1x_5x_6 + x_2x_3x_4 + x_2x_3x_5 + x_3x_4x_5 + x_3x_4x_6 + x_3x_5x_6 + x_0x_2 + x_0x_3 \\
&+ x_0x_5 + x_1x_2 + x_1x_3 + x_1x_4 + x_1x_5 + x_1x_6 + x_4x_5 + x_4x_6 + x_2 + x_3 + x_4
\end{aligned}
$$

$$
\begin{aligned}
&z_2 \\
&= x_0x_1x_2x_4x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_5 + x_0x_1x_2x_3x_6 + x_0x_1x_2x_4x_6 \\
&+ x_0x_1x_3x_4x_6 + x_0x_1x_3x_5x_6 + x_0x_2x_3x_4x_5 + x_0x_3x_4x_5x_6 + x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_6 + x_0x_1x_2x_4 \\
&+ x_0x_1x_2x_5 + x_0x_1x_2x_6 + x_0x_1x_3x_4 + x_0x_1x_3x_5 + x_0x_1x_4x_5 + x_0x_2x_3x_4 + x_0x_2x_3x_5 + x_0x_2x_4x_5 \\
&+ x_0x_2x_4x_6 + x_0x_3x_4x_6 + x_1x_2x_3x_4 + x_1x_2x_3x_6 + x_1x_2x_4x_5 + x_1x_2x_5x_6 + x_1x_3x_4x_6 + x_1x_3x_5x_6 \\
&+ x_1x_4x_5x_6 + x_2x_3x_4x_6 + x_3x_4x_5x_6 + x_0x_1x_2 + x_0x_1x_4 + x_0x_1x_5 + x_0x_1x_6 + x_0x_2x_3 + x_0x_2x_5 \\
&+ x_0x_3x_5 + x_0x_3x_6 + x_0x_4x_6 + x_0x_5x_6 + x_1x_2x_3 + x_1x_2x_6 + x_1x_3x_4 + x_1x_3x_6 + x_0x_2 + x_0x_3 \\
&+ x_0x_4 + x_0x_5 + x_0x_6 + x_1x_2 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5 + x_2x_6 + x_3x_4 + x_3x_5 + x_3x_6 \\
&+ x_4x_6 + x_5x_6 + x_1 + x_2 + x_3 + x_6
\end{aligned}
$$

$$
\begin{aligned}
&z_3 \\
&= x_0x_1x_2x_3x_4x_5 + x_0x_1x_2x_3x_5x_6 + x_0x_1x_3x_4x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_4x_5 \\
&+ x_0x_1x_2x_4x_6 + x_0x_1x_3x_4x_5 + x_0x_1x_3x_5x_6 + x_0x_1x_4x_5x_6 + x_0x_2x_3x_4x_5 + x_0x_2x_3x_4x_6 \\
&+ x_0x_2x_3x_5x_6 + x_1x_2x_3x_4x_5 + x_1x_2x_3x_4x_6 + x_1x_2x_3x_5x_6 + x_1x_2x_4x_5x_6 + x_2x_3x_4x_5x_6 + x_0x_1x_2x_4 \\
&+ x_0x_1x_2x_5 + x_0x_1x_2x_6 + x_0x_1x_3x_4 + x_0x_1x_3x_6 + x_0x_1x_4x_6 + x_0x_2x_3x_5 + x_0x_2x_3x_6 + x_0x_2x_4x_5 \\
&+ x_0x_2x_4x_6 + x_0x_3x_4x_6 + x_0x_4x_5x_6 + x_1x_2x_3x_4 + x_1x_2x_5x_6 + x_1x_3x_5x_6 + x_2x_3x_4x_5 + x_2x_3x_4x_6 \\
&+ x_2x_4x_5x_6 + x_3x_4x_5x_6 + x_0x_1x_2 + x_0x_1x_4 + x_0x_1x_5 + x_0x_2x_5 + x_0x_2x_6 + x_0x_3x_4 + x_0x_3x_6 \\
&+ x_0x_4x_6 + x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_6 + x_1x_3x_5 + x_1x_4x_6 + x_2x_3x_4 + x_2x_3x_5 + x_3x_4x_5 + x_3x_4x_6 \\
&+ x_0x_1 + x_0x_2 + x_0x_4 + x_1x_2 + x_1x_5 + x_1x_6 + x_2x_4 + x_2x_5 + x_2x_6 + x_3x_4 + x_4x_6 + x_1 + x_2 + x_3 \\
&+ x_5
\end{aligned}
$$

$$
\begin{aligned}
&z_4 \\
&= x_0x_1x_2x_3x_4x_6 + x_0x_1x_2x_4x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_6 \\
&+ x_0x_1x_2x_4x_6 + x_0x_1x_2x_5x_6 + x_0x_1x_3x_4x_6 + x_0x_2x_3x_4x_5 + x_0x_3x_4x_5x_6 + x_1x_2x_3x_4x_6 \\
&+ x_1x_2x_4x_5x_6 + x_0x_1x_2x_3 + x_0x_1x_2x_5 + x_0x_1x_2x_6 + x_0x_1x_3x_4 + x_0x_1x_3x_5 + x_0x_1x_4x_6 + x_0x_1x_5x_6 \\
&+ x_0x_2x_3x_4 + x_0x_2x_3x_5 + x_0x_2x_3x_6 + x_0x_2x_4x_5 + x_0x_2x_4x_6 + x_0x_3x_4x_5 + x_0x_3x_4x_6 + x_0x_4x_5x_6 \\
&+ x_1x_2x_3x_5 + x_1x_2x_4x_5 + x_1x_4x_5x_6 + x_2x_4x_5x_6 + x_0x_1x_4 + x_0x_1x_5 + x_0x_1x_6 + x_0x_2x_4 + x_0x_2x_5 \\
&+ x_0x_2x_6 + x_0x_3x_4 + x_0x_5x_6 + x_1x_2x_4 + x_1x_2x_5 + x_1x_2x_6 + x_1x_3x_5 + x_2x_3x_6 + x_2x_4x_5 + x_3x_5x_6 \\
&+ x_0x_1 + x_0x_5 + x_0x_6 + x_2x_3 + x_2x_5 + x_2x_6 + x_3x_4 + x_3x_6 + x_4x_5 + x_1 + x_2 + x_4
\end{aligned}
$$

$$
\begin{aligned}
&z_5 \\
&= x_0x_1x_2x_3x_4x_5 + x_0x_1x_2x_3x_5x_6 + x_0x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_6 + x_0x_1x_2x_4x_6 \\
&+ x_0x_1x_3x_4x_5 + x_0x_1x_3x_4x_6 + x_0x_1x_3x_5x_6 + x_0x_1x_4x_5x_6 + x_0x_2x_3x_4x_5 + x_0x_2x_3x_4x_6 \\
&+ x_0x_3x_4x_5x_6 + x_1x_2x_3x_4x_5 + x_2x_3x_4x_5x_6 + x_0x_1x_2x_6 + x_0x_1x_3x_6 + x_0x_1x_4x_6 + x_0x_1x_5x_6 \\
&+ x_0x_2x_3x_5 + x_0x_2x_5x_6 + x_0x_3x_5x_6 + x_1x_2x_5x_6 + x_1x_4x_5x_6 + x_2x_3x_4x_5 + x_2x_3x_5x_6 + x_2x_4x_5x_6 \\
&+ x_0x_1x_3 + x_0x_1x_5 + x_0x_1x_6 + x_0x_2x_3 + x_0x_2x_5 + x_0x_2x_6 + x_0x_4x_5 + x_0x_4x_6 + x_1x_3x_4 + x_1x_3x_6 \\
&+ x_1x_4x_6 + x_1x_5x_6 + x_2x_3x_6 + x_2x_4x_5 + x_2x_4x_6 + x_3x_4x_5 + x_3x_4x_6 + x_4x_5x_6 + x_0x_2 + x_0x_4 \\
&+ x_0x_5 + x_1x_2 + x_1x_3 + x_1x_5 + x_1x_6 + x_2x_4 + x_2x_6 + x_3x_4 + x_3x_5 + x_5x_6 + x_2 + x_4 + x_5 + x_6
\end{aligned}
$$

$$
\begin{aligned}
&z_6 \\
&= x_0x_1x_2x_3x_4x_6 + x_0x_1x_2x_4x_5x_6 + x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_6 + x_0x_1x_2x_4x_5 \\
&+ x_0x_1x_2x_5x_6 + x_0x_1x_3x_5x_6 + x_0x_1x_4x_5x_6 + x_0x_2x_3x_4x_5 + x_0x_2x_3x_5x_6 + x_1x_2x_4x_5x_6 \\
&+ x_2x_3x_4x_5x_6 + x_0x_1x_2x_4 + x_0x_1x_2x_5 + x_0x_1x_2x_6 + x_0x_1x_3x_5 + x_0x_1x_5x_6 + x_0x_2x_4x_5 \\
&+ x_0x_3x_4x_6 + x_0x_3x_5x_6 + x_0x_4x_5x_6 + x_1x_2x_3x_6 + x_1x_2x_4x_6 + x_1x_2x_5x_6 + x_1x_3x_5x_6 + x_1x_4x_5x_6 \\
&+ x_2x_3x_4x_5 + x_2x_3x_5x_6 + x_2x_4x_5x_6 + x_0x_1x_5 + x_0x_2x_4 + x_0x_2x_6 + x_0x_3x_5 + x_0x_3x_6 + x_0x_4x_5 \\
&+ x_0x_4x_6 + x_1x_2x_3 + x_1x_2x_5 + x_1x_2x_6 + x_1x_3x_4 + x_1x_3x_5 + x_1x_3x_6 + x_1x_5x_6 + x_2x_3x_4 + x_2x_3x_6 \\
&+ x_2x_4x_5 + x_3x_4x_6 + x_0x_1 + x_0x_5 + x_2x_6 + x_3x_6 + x_5x_6 + x_1 + x_3
\end{aligned}
$$

A.2 WGP AXI form

Tables 5-9 show the WGP AXI (AND, XOR, INV), formatted as number of inputs, number of outputs, wire number for inputs i_1 and i_2, wire number for the output o, and gate type. The input and output components of $z = \mathsf{WGP}(x)$ are wired as follows:

$$
\begin{array}{cccccccc}
x_0 = 1020 & x_1 = 1021 & x_2 = 1022 & x_3 = 1023 & x_4 = 1024 & x_5 = 1025 & x_6 = 1026 \\
z_0 = 1027 & z_1 = 1028 & z_2 = 1029 & z_3 = 1030 & z_4 = 1031 & z_5 = 1032 & z_6 = 1033
\end{array}
$$

Table 5: WGP AXI (AND, XOR, INV)

#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate
2	1	502	503	1033	XOR	1	1	1045		567	NOT	1	1	634		1050	NOT
2	1	504	505	503	XOR	1	1	569		1043	NOT	2	1	1049	1050	1051	AND
2	1	506	507	505	XOR	1	1	570		1044	NOT	2	1	635	636	607	XOR
2	1	508	509	507	XOR	2	1	1043	1044	1045	AND	2	1	579	568	636	XOR
2	1	510	511	509	XOR	1	1	1048		570	NOT	2	1	637	551	635	XOR
2	1	512	513	511	XOR	1	1	571		1046	NOT	2	1	638	639	603	XOR
2	1	514	515	513	XOR	1	1	572		1047	NOT	2	1	640	641	601	XOR
2	1	516	517	515	XOR	2	1	1046	1047	1048	AND	2	1	642	643	641	XOR
2	1	518	519	517	XOR	2	1	573	574	572	AND	2	1	644	645	643	XOR
2	1	520	521	519	XOR	2	1	575	576	571	AND	2	1	646	647	645	XOR
2	1	522	523	521	XOR	1	1	573		576	NOT	2	1	648	649	647	XOR
2	1	524	525	523	XOR	2	1	1023	536	575	AND	2	1	573	1024	648	AND
2	1	526	527	522	XOR	2	1	577	578	569	AND	2	1	650	651	644	XOR
2	1	528	529	520	XOR	2	1	1021	573	577	XOR	2	1	652	653	651	XOR
2	1	530	531	529	XOR	2	1	579	580	508	XOR	2	1	654	655	595	XOR
2	1	532	533	531	AND	2	1	581	582	506	XOR	2	1	656	657	655	XOR
1	1	1036		532	NOT	2	1	583	584	582	XOR	2	1	658	659	593	AND
1	1	534		1034	NOT	2	1	585	586	581	XOR	2	1	542	660	659	XOR
1	1	535		1035	NOT	2	1	587	588	504	AND	2	1	573	661	658	XOR
2	1	1034	1035	1036	AND	2	1	589	590	502	XOR	1	1	1054		661	NOT
2	1	536	537	534	AND	2	1	591	592	1032	XOR	1	1	662		1052	NOT
2	1	538	539	528	XOR	2	1	593	594	592	XOR	1	1	663		1053	NOT
2	1	540	541	518	AND	2	1	595	596	594	XOR	2	1	1052	1053	1054	AND
1	1	1039		541	NOT	2	1	597	598	596	XOR	1	1	1057		663	NOT
1	1	1021		1037	NOT	2	1	599	600	598	XOR	1	1	664		1055	NOT
1	1	542		1038	NOT	2	1	601	602	600	XOR	1	1	665		1056	NOT
2	1	1037	1038	1039	AND	2	1	603	604	602	XOR	2	1	1055	1056	1057	AND
2	1	543	544	540	AND	2	1	605	606	604	XOR	2	1	1026	666	665	AND
1	1	1042		543	NOT	2	1	607	608	606	XOR	2	1	667	537	664	AND
1	1	1025		1040	NOT	2	1	609	610	608	XOR	2	1	1025	668	667	AND
1	1	545		1041	NOT	2	1	611	612	610	XOR	1	1	669		668	NOT
2	1	1040	1041	1042	AND	2	1	613	614	612	XOR	2	1	573	669	662	AND
2	1	546	547	516	XOR	2	1	615	616	614	XOR	2	1	670	671	591	XOR
2	1	548	549	514	XOR	2	1	617	618	616	XOR	2	1	672	673	1031	XOR
2	1	550	551	549	XOR	2	1	524	619	615	XOR	2	1	674	675	673	XOR
2	1	552	553	548	XOR	2	1	620	621	524	XOR	2	1	676	677	675	XOR
2	1	554	555	512	XOR	2	1	622	623	613	XOR	2	1	678	679	677	XOR
2	1	556	557	510	XOR	2	1	624	625	623	XOR	2	1	680	681	679	XOR
2	1	558	559	557	XOR	2	1	626	627	622	XOR	2	1	682	683	681	XOR
2	1	560	561	559	AND	2	1	628	629	609	XOR	2	1	684	685	683	XOR
2	1	562	563	561	XOR	2	1	630	1026	629	AND	2	1	686	687	685	XOR
2	1	564	565	560	XOR	2	1	631	632	628	AND	2	1	688	689	687	XOR
2	1	1026	566	558	AND	1	1	1051		632	NOT	2	1	690	691	689	XOR
2	1	567	568	556	XOR	1	1	633		1049	NOT	2	1	692	693	691	XOR

Table 6: WGP AXI (AND, XOR, INV) - continued

#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate
2	1	694	695	693	XOR	1	1	743		742	NOT	2	1	807	808	631	XOR
2	1	696	697	695	XOR	2	1	744	745	672	XOR	1	1	550		807	NOT
2	1	698	699	697	XOR	2	1	746	747	745	XOR	2	1	809	810	778	XOR
2	1	619	700	696	XOR	2	1	748	749	747	AND	2	1	811	812	754	XOR
2	1	701	585	619	XOR	2	1	1021	542	744	XOR	2	1	813	545	812	XOR
2	1	702	703	694	XOR	2	1	750	751	1030	XOR	1	1	746		545	NOT
2	1	625	621	703	XOR	2	1	752	753	751	XOR	2	1	1020	1021	746	AND
2	1	704	633	690	XOR	2	1	754	755	753	XOR	2	1	814	815	811	AND
2	1	1024	1022	633	XOR	2	1	756	757	755	XOR	1	1	1069		815	NOT
2	1	630	1024	704	AND	2	1	758	759	757	XOR	1	1	816		1067	NOT
2	1	705	706	688	XOR	2	1	760	761	759	XOR	1	1	562		1068	NOT
2	1	707	708	706	XOR	2	1	762	639	761	XOR	2	1	1067	1068	1069	AND
2	1	551	539	705	XOR	2	1	763	764	639	XOR	2	1	817	818	750	XOR
2	1	709	710	684	XOR	2	1	765	766	760	XOR	2	1	1025	670	818	XOR
2	1	711	712	710	AND	2	1	767	768	766	AND	1	1	819		817	NOT
2	1	713	714	712	XOR	2	1	769	770	767	XOR	2	1	820	821	1029	XOR
2	1	715	1020	713	AND	2	1	771	586	770	XOR	2	1	822	823	821	XOR
2	1	716	717	711	XOR	1	1	739		586	NOT	2	1	824	825	823	XOR
1	1	718		716	NOT	2	1	772	773	771	AND	2	1	826	827	825	XOR
2	1	719	720	682	XOR	2	1	774	775	773	XOR	2	1	828	829	827	XOR
2	1	553	721	720	XOR	2	1	588	776	769	AND	2	1	830	831	829	XOR
2	1	552	587	719	XOR	1	1	777		776	NOT	2	1	832	833	831	XOR
2	1	722	723	678	XOR	2	1	778	779	758	XOR	2	1	834	835	833	XOR
2	1	724	725	676	XOR	2	1	780	781	779	XOR	2	1	836	837	835	XOR
2	1	555	526	725	XOR	2	1	782	783	781	XOR	1	1	1072		837	NOT
1	1	726	526		NOT	2	1	784	785	783	XOR	1	1	838		1070	NOT
2	1	727	670	555	XOR	2	1	786	787	785	XOR	1	1	839		1071	NOT
2	1	544	728	724	XOR	2	1	788	789	787	XOR	2	1	1070	1071	1072	AND
1	1	1060		728	NOT	2	1	790	791	789	XOR	2	1	670	537	839	AND
1	1	729		1058	NOT	2	1	709	792	791	XOR	2	1	840	841	838	AND
1	1	554		1059	NOT	2	1	1023	793	709	AND	2	1	842	1026	840	AND
2	1	1058	1059	1060	AND	2	1	552	794	790	XOR	2	1	730	749	842	AND
2	1	730	731	554	XOR	2	1	795	796	788	XOR	1	1	671		749	NOT
1	1	1063		544	NOT	2	1	797	798	796	XOR	2	1	802	843	836	XOR
1	1	666		1061	NOT	2	1	546	699	795	XOR	2	1	844	845	843	XOR
1	1	732		1062	NOT	2	1	624	799	699	XOR	2	1	846	847	845	XOR
2	1	1061	1062	1063	AND	2	1	800	701	786	AND	2	1	794	848	847	XOR
2	1	733	734	674	XOR	1	1	530		701	NOT	2	1	849	850	846	XOR
2	1	735	736	734	AND	2	1	587	1026	530	AND	1	1	1075		850	NOT
1	1	1066		736	NOT	1	1	801		800	NOT	1	1	646		1073	NOT
1	1	737		1064	NOT	2	1	802	803	784	XOR	1	1	605		1074	NOT
1	1	738		1065	NOT	2	1	804	805	782	XOR	2	1	1073	1074	1075	AND
2	1	1064	1065	1066	AND	2	1	539	707	805	XOR	2	1	743	717	605	XOR
2	1	739	740	735	XOR	2	1	583	806	804	XOR	2	1	851	718	646	XOR
2	1	741	742	733	AND	2	1	631	649	780	XOR	2	1	1025	806	718	AND

Table 7: WGP AXI (AND, XOR, INV) - continued

#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate
2	1	737	777	851	XOR	2	1	893	894	820	XOR	2	1	580	934	797	XOR
2	1	852	853	844	XOR	2	1	895	896	894	XOR	2	1	895	1023	580	AND
2	1	810	854	853	XOR	2	1	660	732	893	XOR	2	1	935	653	854	XOR
1	1	855		810	NOT	2	1	897	898	1028	XOR	2	1	936	937	915	XOR
2	1	856	857	855	XOR	2	1	899	900	898	XOR	2	1	721	808	937	XOR
2	1	858	585	857	XOR	2	1	901	902	900	XOR	2	1	1024	660	808	AND
2	1	546	809	852	XOR	2	1	903	904	902	XOR	2	1	1026	660	721	AND
2	1	566	1021	809	XOR	2	1	905	906	904	XOR	2	1	652	550	936	XOR
1	1	859		566	NOT	2	1	907	908	906	XOR	2	1	938	939	874	XOR
2	1	860	715	546	XOR	2	1	762	909	908	XOR	2	1	1020	1023	938	AND
2	1	861	568	802	XOR	2	1	910	911	909	XOR	2	1	650	940	912	XOR
2	1	715	1024	861	AND	2	1	912	913	911	XOR	2	1	892	1024	940	AND
2	1	862	863	834	XOR	2	1	874	914	913	XOR	2	1	941	942	910	XOR
2	1	525	723	863	XOR	2	1	915	916	914	XOR	2	1	943	944	942	XOR
2	1	864	865	525	XOR	2	1	917	918	916	XOR	2	1	535	585	941	XOR
1	1	806		864	NOT	2	1	919	920	918	XOR	1	1	630		535	NOT
2	1	866	867	862	XOR	2	1	921	922	920	XOR	2	1	717	654	762	XOR
2	1	726	562	867	AND	2	1	832	923	922	XOR	2	1	1025	765	654	AND
2	1	868	869	726	XOR	2	1	924	738	832	XOR	2	1	1024	626	765	AND
2	1	870	871	866	AND	1	1	1081		738	NOT	2	1	1024	868	717	AND
2	1	872	873	871	XOR	1	1	578		1079	NOT	2	1	945	946	907	XOR
2	1	657	714	870	XOR	1	1	925		1080	NOT	2	1	680	947	946	AND
2	1	874	875	830	XOR	2	1	1079	1080	1081	AND	1	1	948		947	NOT
2	1	876	877	828	XOR	1	1	707		924	NOT	2	1	1022	539	948	AND
2	1	878	621	877	XOR	2	1	1026	538	707	AND	2	1	1026	627	539	AND
2	1	1024	652	621	AND	2	1	801	803	921	XOR	2	1	764	533	680	XOR
1	1	1078		878	NOT	2	1	698	740	803	XOR	2	1	1024	625	764	AND
1	1	879		1076	NOT	2	1	926	865	698	XOR	2	1	1020	814	625	AND
1	1	880		1077	NOT	1	1	1084		865	NOT	2	1	949	564	945	AND
2	1	1076	1077	1078	AND	1	1	666		1082	NOT	2	1	950	951	905	XOR
2	1	881	882	880	XOR	1	1	925		1083	NOT	2	1	590	868	951	XOR
2	1	883	708	879	XOR	2	1	1082	1083	1084	AND	2	1	777	775	950	XOR
2	1	587	1021	708	AND	1	1	538		925	NOT	1	1	1087		775	NOT
2	1	884	583	876	XOR	1	1	1025		666	NOT	1	1	599		1085	NOT
2	1	729	722	826	XOR	2	1	551	1020	926	AND	1	1	638		1086	NOT
2	1	885	886	722	XOR	2	1	883	620	801	XOR	2	1	1085	1086	1087	AND
2	1	1026	1020	885	AND	2	1	1024	551	883	AND	2	1	806	1022	599	AND
2	1	649	752	729	XOR	2	1	927	928	919	XOR	2	1	620	1021	777	AND
2	1	1022	1025	752	AND	2	1	539	527	928	XOR	2	1	952	640	903	XOR
2	1	887	888	649	XOR	2	1	929	930	927	XOR	2	1	814	574	640	XOR
2	1	1023	1024	887	AND	2	1	931	723	917	XOR	2	1	1021	1024	952	AND
2	1	889	890	824	XOR	2	1	932	765	723	XOR	2	1	953	954	901	XOR
2	1	625	618	890	XOR	2	1	630	1021	932	AND	2	1	955	891	954	XOR
2	1	630	891	889	XOR	2	1	854	933	931	AND	2	1	873	956	953	XOR
2	1	793	892	822	XOR	1	1	797		933	NOT	2	1	957	958	899	XOR

Table 8: WGP AXI (AND, XOR, INV) - continued

#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate
2	1	892	872	958	AND	2	1	1025	551	579	AND	1	1	552		935	NOT
2	1	748	562	957	AND	2	1	1023	888	551	AND	2	1	1026	884	590	AND
2	1	1024	1025	748	AND	2	1	1021	550	881	AND	2	1	583	1022	998	AND
2	1	959	960	897	XOR	2	1	626	929	991	XOR	2	1	686	774	975	XOR
2	1	819	563	960	XOR	2	1	1024	553	929	AND	2	1	565	869	774	XOR
2	1	961	962	959	AND	2	1	650	553	798	XOR	2	1	656	873	565	XOR
2	1	963	731	962	XOR	2	1	1024	814	650	AND	2	1	1021	538	873	AND
1	1	1090		731	NOT	2	1	1021	1026	814	AND	2	1	1024	896	538	AND
1	1	537		1088	NOT	2	1	618	692	980	XOR	2	1	620	1026	656	AND
1	1	841		1089	NOT	2	1	849	792	692	XOR	2	1	657	563	686	XOR
2	1	1088	1089	1090	AND	2	1	1096		792	NOT	2	1	1025	868	563	AND
2	1	727	730	961	XOR	1	1	816		1094	NOT	2	1	1021	806	657	AND
2	1	1093		730	NOT	1	1	732		1095	NOT	2	1	1020	550	806	AND
1	1	537		1091	NOT	2	1	1094	1095	1096	AND	2	1	1024	741	550	AND
1	1	768		1092	NOT	1	1	944		849	NOT	2	1	1001	1002	973	XOR
2	1	1091	1092	1093	AND	2	1	1026	939	944	AND	2	1	891	763	1002	XOR
1	1	1026		537	NOT	2	1	799	943	618	XOR	2	1	868	955	763	XOR
2	1	964	965	1027	XOR	2	1	1023	896	943	AND	2	1	1026	626	955	AND
2	1	966	967	965	XOR	2	1	1025	896	799	AND	2	1	1020	574	626	AND
2	1	923	968	967	XOR	2	1	702	993	978	XOR	2	1	1021	627	891	AND
2	1	969	970	968	XOR	2	1	611	547	993	XOR	2	1	1025	793	627	AND
2	1	756	971	970	XOR	2	1	994	995	547	XOR	2	1	637	1003	1001	XOR
2	1	875	972	971	XOR	2	1	996	624	995	XOR	2	1	1004	1005	1003	AND
2	1	973	974	972	XOR	2	1	1026	896	624	AND	1	1	638		1005	NOT
2	1	975	976	974	XOR	2	1	653	794	996	XOR	2	1	562	589	638	XOR
2	1	772	977	976	XOR	2	1	1020	741	794	AND	1	1	1105		589	NOT
2	1	978	979	977	XOR	2	1	1026	574	653	AND	1	1	816		1103	NOT
2	1	980	981	979	XOR	2	1	627	997	994	XOR	1	1	533		1104	NOT
2	1	798	982	981	XOR	2	1	630	652	997	XOR	2	1	1103	1104	1105	AND
2	1	983	984	982	XOR	2	1	860	848	611	XOR	1	1	1108		562	NOT
2	1	985	986	984	XOR	1	1	584		848	NOT	1	1	578		1106	NOT
2	1	700	856	986	XOR	2	1	1024	574	584	AND	1	1	533		1107	NOT
2	1	987	930	856	XOR	2	1	1026	793	860	AND	2	1	1106	1107	1108	AND
2	1	1025	550	930	AND	2	1	538	858	702	XOR	1	1	637		533	NOT
2	1	896	1021	987	AND	2	1	1021	793	858	AND	1	1	1023		578	NOT
2	1	988	806	700	XOR	2	1	998	999	772	XOR	2	1	588	956	1004	XOR
2	1	715	1025	988	AND	2	1	590	714	999	XOR	2	1	1024	583	956	AND
2	1	527	617	985	XOR	1	1	1099		714	NOT	2	1	1026	553	583	AND
2	1	989	884	617	XOR	1	1	935		1097	NOT	2	1	1025	574	553	AND
2	1	799	1026	989	AND	1	1	1000		1098	NOT	1	1	737		588	NOT
2	1	990	882	527	XOR	2	1	1097	1098	1099	AND	2	1	1020	652	637	AND
2	1	1026	715	882	AND	2	1	1102		1000	NOT	2	1	892	1026	652	AND
2	1	552	1024	990	AND	1	1	816		1100	NOT	2	1	1006	573	875	XOR
2	1	991	992	983	XOR	1	1	578		1101	NOT	2	1	1025	1026	573	AND
2	1	881	579	992	XOR	2	1	1101	1100	1102	AND	2	1	819	741	1006	XOR

Table 9: WGP AXI (AND, XOR, INV) - continued

#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate	#	#	i_1	i_2	o	gate
2	1	1023	1026	741	AND	2	1	1023	739	727	AND	2	1	1023	868	743	AND
2	1	1024	1026	819	AND	2	1	1024	568	739	AND	2	1	1020	552	868	AND
2	1	642	1007	756	XOR	2	1	1025	552	568	AND	2	1	1026	1008	552	AND
1	1	597		1007	NOT	2	1	816	859	923	XOR	2	1	1016	1017	964	XOR
2	1	793	949	597	XOR	2	1	1023	1014	859	XOR	2	1	1018	1019	1017	XOR
2	1	732	896	949	XOR	2	1	564	634	966	XOR	2	1	574	872	1019	AND
2	1	1022	1020	896	AND	2	1	1015	587	634	XOR	1	1	1123		872	NOT
1	1	1008		732	NOT	2	1	895	1025	587	AND	1	1	536		1121	NOT
2	1	895	892	642	XOR	2	1	1026	895	1015	AND	1	1	740		1122	NOT
2	1	1025	1021	892	AND	2	1	1022	1024	895	AND	2	1	1121	1122	1123	AND
2	1	1009	1010	969	XOR	2	1	743	768	564	XOR	1	1	884		740	NOT
2	1	886	1020	1010	XOR	2	1	1114		768	NOT	2	1	1022	630	884	AND
2	1	1022	1023	886	AND	1	1	816		1112	NOT	2	1	1023	939	630	AND
2	1	1011	813	1009	XOR	1	1	869		1113	NOT	2	1	1025	1020	939	AND
1	1	1111		813	NOT	2	1	1112	1113	1114	AND	1	1	1021		536	NOT
1	1	671		1109	NOT	1	1	1117		869	NOT	2	1	1023	1021	574	AND
1	1	669		1110	NOT	1	1	1014		1115	NOT	2	1	888	1025	1018	AND
2	1	1109	1110	1111	AND	1	1	585		1116	NOT	2	1	1022	1026	888	AND
2	1	1012	963	669	XOR	2	1	1115	1116	1117	AND	2	1	934	841	1016	XOR
2	1	1025	743	963	AND	1	1	1120		585	NOT	1	1	670		841	NOT
2	1	670	1025	1012	AND	1	1	536		1118	NOT	2	1	793	715	670	AND
2	1	1026	737	671	AND	1	1	542		1119	NOT	2	1	1023	1008	715	AND
2	1	620	1020	737	AND	2	1	1118	1118	1120	AND	2	1	1022	1021	1008	AND
2	1	587	1023	620	AND	1	1	939		542	NOT	2	1	1024	1020	793	AND
2	1	888	1013	1011	AND	1	1	1022		1014	NOT	2	1	1022	660	934	AND
1	1	727		1013	NOT	1	1	1024		816	NOT	2	1	1025	1023	660	AND

B Basic Masking Gadgets

Refresh mask and Common share multiplication. We provide the pseudocodes for algorithms RefreshMask and SecMult from [8], and CommonMult and CommonShare from [17] for the ease of completeness and quick references. Note that in Algorithm 5, in Lines 5-6, the multiplications for $a = (a^i)_{1\leq i\leq n}$ and the common shares of $b = (b^i)_{1\leq i\leq n}$ and $c = (c^i)_{1\leq i\leq n}$ are computed only once which results in reducing the area of the gadget.

Algorithm 2 Refresh Mask [8]

Input: $(a^1, a^2, \cdots, a^n)$ s.t. $a = a^1 \oplus a^2 \oplus \cdots \oplus a^n, a^i \in \mathbb{F}_2^k$
Output: $(c^1, c^2, \cdots, c^n)$ s.t. $a = c^1 \oplus c^2 \oplus \cdots \oplus c^n, a^i \in \mathbb{F}_2^k$
procedure RefreshMask$(a^1, a^2, \cdots, a^n)$
    for $i = 1$ to n **do**
        $c^i \leftarrow a^i$
    end for
    for $i = 1$ to $n - 1$ **do**
        for $j = i + 1$ to n **do**
            $r \leftarrow^{\$} \mathbb{F}_2^k$
            $c^i = c^i \oplus r$
            $c^j = c^j \oplus r$
        end for
    end for
    return $(c^1, c^2, \cdots, c^n)$
end procedure

t-SNI secure randomized table countermeasure of S-boxes. For the sake of completeness, we rewrite the t-SNI secure randomize lookup table algorithm from [19], and take WGP as an example to describe the algorithm.

Algorithm 3 Multiplication Gadget (t-SNI) [8]

Inputs: $(x^1, x^2, \cdots, x^n)$ and $(y^1, y^2, \cdots, y^n), x^i, y^i \in \mathbb{F}_2$
Output: $(z^1, z^2, \cdots, z^n)$
procedure SecMult(x, y)
    for $i = 1$ to n **do**
        $z^i \leftarrow x^i y^i$
    end for
    for $i = 1$ to n **do**
        for $j = i + 1$ to n **do**
            $r \leftarrow^{\$} \mathbb{F}_2$
            $z^i \leftarrow z^i \oplus r$
            $t \leftarrow x^i y^j$
            $r \leftarrow r \oplus t$
            $t \leftarrow x^j y^i$
            $r \leftarrow r \oplus t$
            $z^j \leftarrow z^j \oplus r$
        end for
    end for
end procedure

Algorithm 4 Common Share [17]

Input: $(a^1, a^2, \cdots, a^n)$ and $(b^1, b^2, \cdots, b^n)$ s.t. $\bigoplus_{i=1}^{n} a^i = a$ and $\bigoplus_{i=1}^{n} b^i = b$
Output: $(c^1, c^2, \cdots, c^n)$ and $(d^1, d^2, \cdots, d^n)$ s.t. $\bigoplus_{i=1}^{n} c^i = \bigoplus_{i=1}^{n} a^i$ and $\bigoplus_{i=1}^{n} d^i = \bigoplus_{i=1}^{n} b^i$
procedure CommonShare$(a^1, a^2, \cdots, a^n)$
    for $i = 1$ to $\frac{n}{2}$ **do**
        $r \leftarrow \mathbb{F}_2$
        $c^i \leftarrow r$; $c^{\frac{n}{2}+i} \leftarrow a^i \oplus a^{\frac{n}{2}+i} \oplus r$
        $d^i \leftarrow r$; $d^{\frac{n}{2}+i} \leftarrow b^i \oplus b^{\frac{n}{2}+i} \oplus r$
    end for
    return $(c^1, c^2, \cdots, c^n)$ and $(d^1, d^2, \cdots, d^n)$
end procedure

Algorithm 5 Common Share Multiplication Gadget (t-SNI) [17]

Input: $(a^1, a^2, \cdots, a^n)$, $(b^1, b^2, \cdots, b^n)$ and $(c^1, c^2, \cdots, c^n)$
Output: $(d^1, d^2, \cdots, d^n)$ and $(e^1, e^2, \cdots, e^n)$ s.t. $d = a \cdot b$ and $e = a \cdot b$
procedure $\mathsf{CommonMult}((a^1, a^2, \cdots, a^n), (b^1, b^2, \cdots, b^n), (c^1, c^2, \cdots, c^n))$
    $(b^i)_{1\le i\le n}, (c^i)_{1\le i\le n} \leftarrow \mathsf{CommonShare}((b^i)_{1\le i\le n}, (c^i)_{1\le i\le n})$
    $(d^i)_{1\le i\le n} \leftarrow \mathsf{SecMul}((a^i)_{1\le i\le n}, (b^i)_{1\le i\le n})$
    $(e^i)_{1\le i\le n} \leftarrow \mathsf{SecMul}((a^i)_{1\le i\le n}, (c^i)_{1\le i\le n})$
    return $(d^1, d^2, \cdots, d^n), (e^1, e^2, \cdots, e^n)$
end procedure

Algorithm 6 Randomized lookup table computation of $y = \mathsf{WGP}(x)$ (t-SNI) [19]

Input: Input shares $(x^1, x^2, \cdots, x^n)$ s.t. $x = x^1 \oplus x^2 \oplus \cdots \oplus x^n, x^i \in \mathbb{F}_2^7$
Output: Output shares $(y^1, y^2, \cdots, y^n)$ s.t. $y = \mathsf{WGP}(x) = y^1 \oplus y^2 \oplus \cdots \oplus y^n$
procedure $\mathsf{SecWGP}(\ )$
    for $u = 0$ to 127 **do**
        $T(u) \leftarrow (\mathsf{WGP}(u), 0, \cdots, 0)$ ▷ n-tuple
    end for
    for $i = 1$ to $n - 1$ **do**
        for $u = 0$ to 127 **do**
            for $j = 1$ to i **do**
                $T'(u)[j] \leftarrow T(u \oplus x^i)[j]$
            end for
        end for
        for $u = 0$ to 127 **do**
            $T(u) \leftarrow (T'(u)[1], T'(u)[2], \cdots T'(u)[i], 0, \cdots, 0)$
            $T(u) \leftarrow \mathsf{RefreshMasks}_{i+1}(T(u))$
        end for
    end for
    $(y^1, y^2, \cdots, y^n) \leftarrow \mathsf{RefreshMasks}_n(T(x^n))$
    return $(y^1, y^2, \cdots, y^n)$
end procedure
procedure $\textsc{RefreshMasks}_i\ (\ )$ ▷ $\mathsf{RefreshMasks}_i()$
    Input: $(z^1, \cdots, z^i)$ s.t. $z = z^1 \oplus z^2 \oplus \cdots \oplus z^i$
    Output: $(z^1, \cdots, z^i)$ s.t. $z = z^1 \oplus z^2 \oplus \cdots \oplus z^i$
    for $j = 2$ to i **do**
        $t \leftarrow \{0, 1\}^7$ ▷ Randomly generate a 7-bit number
        $z^1 \leftarrow z^1 \oplus t$
        $z^j \leftarrow z^j \oplus t$
    end for
    return $(z^1, \cdots, z^i)$
end procedure