IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 26 August 2020; revised 31 March 2021; accepted 8 April 2021.
Date of publication 14 April 2021; date of current version 7 June 2022.

Digital Object Identifier 10.1109/TETC.2021.3073163

Hardware Constructions for Error Detection in Lightweight Welch-Gong (WG)-Oriented Streamcipher WAGE Benchmarked on FPGA

JASMIN KAUR, (Student Member, IEEE), AUSMITA SARKER, (Student Member, IEEE), MEHRAN MOZAFFARI KERMANI, (Senior Member, IEEE), AND REZA AZARDERAKHSH, (Member, IEEE)

Jasmin Kaur, Ausmita Sarker, and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA
Reza Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA
CORRESPONDING AUTHOR: MEHRAN MOZAFFARI-KERMANI (mehran2@usf.edu)

ABSTRACT Providing an acceptable level of security at low cost becomes a challenge in embedded systems that have limited resources, e.g., Internet of Things application devices. Lightweight cryptography aims to overcome this challenge by adopting security measures which are well suited for resource-constrained usage models. As we move towards standardizing the approach for lightweight cryptography, several cipher implementations were proposed to NIST, out of which WAGE is one of the algorithms advanced to the next round. WAGE is a 259-bit lightweight stream cipher that derives its cryptographic properties from the WG stream cipher and is designed to provide Authenticated Encryption with Associated Data (AEAD) in hardware implementation. In this article, we present error detection schemes for the nonlinear sub-blocks of WAGE cipher for secure implementations of the cipher on hardware. The proposed signature-based error detection schemes are platform oblivious. Derivations for signature and interleaved signature schemes for both logic- and LUT-based implementations are presented for the 7-bit S-Box and Welch- Gong permutation (WGP) of WAGE. The presented schemes are evaluated for error coverage and are benchmarked on field-programmable gate array (FPGA) which show acceptable overheads and practical error coverage.

INDEX TERMS Concurrent error detection (CED), field-programmable gate array (FPGA), multi-bit upsets (MBU)

I. INTRODUCTION

Lightweight cryptography utilizes symmetric-key cryptography to provide acceptable level of security to highly resource-constrained applications, e.g., Internet of Things devices, sensor networks, RFID tags, and constrained communication nodes. Stream ciphers are symmetric ciphers that function on a data stream bit by bit and are often used in applications where the plaintext input length is unknown or is continuous. These ciphers are especially beneficial to resource-constrained applications for their simplicity, low area, and power consumption. However, cryptanalysis has shown that most of the available stream ciphers are weak against algebraic and side-channel attacks. In the research work of [1], differential fault analysis (DFA) using random faults was successfully mounted on stream ciphers. Therefore, lightweight stream ciphers with good cryptographic properties are required for practical applications.

WAGE [2] is a 259-bit lightweight stream cipher, based on the initialization step of the WG stream cipher [3], [4]. It works in a unified duplex sponge operation [5]–[7] to provide efficient AEAD for hardware implementations of the cipher. The WAGE-AEAD algorithm consists of two parts: Authenticated encryption and verified decryption algorithms. We note that WAGE-AEAD shows acceptable security margins (128-bit security for a 128-bit key and nonce), however its vulnerability to active fault analysis attacks needs to be thwarted for lightweight cryptography. For example, attackers can exploit statistical fault attacks (SFA) [8], where the ciphertext is manipulated through fault injection, to recover the key even with authenticated encryption. This motivates us to take measures to make the WAGE implementations more secure against fault attacks as well as natural VLSI defects.

Error detection schemes [9]–[11] detect the cryptographic hardware implementations against errors. These approaches can detect either natural (caused due to manufacturing defects) or maliciously injected faults (biased or adjacent multi-bit upsets, i.e., multi-bit upsets (MBU), by intelligent adversaries). For example, in authenticated encryption schemes, both the authentication and encryption capabilities could be compromised due to natural faults, reducing their reliability. Parity-based error detection has been utilized in lightweight ciphers, due to its high error coverage and low overheads in hardware implementations.Limited work is performed on error detection of lightweight cryptography. The work of [12] explores error detection on concurrent error detection (CED) on transient faults, whereas the work of [13] computes CED on Speck and Simon. This paper, for the first time, proposes error detection schemes for the nonlinear sub-blocks of WAGE.The contributions of this paper are as follows:

- To the best of authors' knowledge, this is the first work that proposes logic-gate and look-up table (LUT) based error detection schemes for the hardware implementations of WAGE cipher. We implement the proposed schemes in the nonlinear layer of WAGE and aim to harden the architecture of WAGE against error injection.
- The presented single/interleaved signature schemes are tailored to detect single-event upsets (SEU) and adjacent MBU, respectively. Such schemes do not undermine the performance and implementation metrics of the original design impractically, providinghigh fault coverage with acceptable overheads.
- Using closed formulations, we also present the logic gate variants of the nonlinear sub-blocks of WAGE. Through the variants and the proposed error detection schemes, we aim to detect errors in VLSI implementations with low overhead for resource-constrained applications.
- The proposed schemes are benchmarked on field-programmable gate array (FPGA) hardware platform to confirm the achieved objectives. Since the proposed approaches are platform oblivious, we expect similar overheads on the application-specific integrated circuit (ASIC) hardware platform. Error coverage simulations have also been performed to show the high error coverage of the proposed schemes.

The paper is organized as follows: In Section II, we discuss preliminaries describing the functionality of WAGE In Section III, the proposed error detection schemes are explained. In Section IV, we present the error simulations and FPGA implementations, along with conclusion in Section V.

II. PRELIMINARIES

WAGE [2] is a 259-bit iterative permutation cipher designed as a tweaked version of the initialization phase of the WG [3]. It is divided into two parts: The authenticated encryption algorithm (*WAGE-E*) and the verified decryption algorithm (*WAGE-D*). Both of these are symmetrical in application, and each mode takes in a k-bit sized key K, a public message number N (nonce) of n-bits, a block header associated data (AD) along with an m-bit plaintext M (encryption mode) or a ciphertext C (decryption mode). The output of *WAGE-E* is an authenticated ciphertext C and an authentication tag of size t-bits. *WAGE-D* outputs the plaintext M if the associated tag is verified or an error symbol $\perp$ otherwise. The AEAD algorithm processes an r-bit data per call of WAGE which is parameterized by the input key K.

The round function of WAGE consists of two nonlinear 7-bit WGPs, four nonlinear 7-bit S-Boxes, and a linear feedback shift register. The internal state IS of WAGE permutation ($IS = (IS_{36}, \ldots, IS_0)$) consists of 37 stages. It operates over the finite field $\mathbb{F}_{2^7}$, defined using the primitive polynomial $f(x) = x^7 + x^3 + x^2 + x + 1$. Polynomial basis $Pb = \{1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, \omega^6\}$ is used to represent the elements of the finite field, where an element $\alpha \in \mathbb{F}_{2^7}$, with vector representation as $[\alpha]_{PB} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)$ is given by: $\alpha = \sum_{i=0}^{6} \alpha_i \omega^i, \alpha_i \in \mathbb{F}_2$.

The WAGE LFSR is defined by the following feedback polynomial: $g(y) = y^{37} + y^{31} + y^{30} + y^{26} + y^{24} + y^{19} + y^{13} + y^{12} + y^8 + y^6 + \omega$, and the linear feedback lfb is computed as: $lfb = IS_{31} \oplus IS_{30} \oplus IS_{26} \oplus IS_{24} \oplus IS_{19} \oplus IS_{13} \oplus IS_{12} \oplus IS_8 \oplus IS_6 \oplus (\omega \oplus IS_0)$.

The decimated Welch-Gong permutation (WGP) over $\mathbb{F}_{2^7}$ is defined as: $WGP(x^d) = x^d + (x^d + 1)^{33} + (x^d + 1)^{39} + (x^d + 1)^{41} + (x^d + 1)^{104}, x \in \mathbb{F}_{2^7}$, where the decimation d is given as $\gcd(d, 2^m - 1) = 1$, and uses the value $d = 13$.

The iterations of the lightweight 7-bit S-Box with input vector m such that $m = (m_0, m_1, m_2, m_3, m_4, m_5, m_6)$, are mathematically defined as

$$(m_0, m_1, \ldots, m_6) \leftarrow R^5(m_0, m_1, \ldots, m_6)$$

$$(m_0, m_1, \ldots, m_6) \leftarrow Q(m_0, m_1, \ldots, m_6),$$

followed by updating the values of m_0 and m_2 as $m_0 \leftarrow m_0 \oplus 1$ and $m_2 \leftarrow m_2 \oplus 1$. Here, $R = P \circ Q$ denotes one round of the S-Box, where $P(m_0, m_1, m_2, m_3, m_4, m_5, m_6) = (m_6, m_3, m_0, m_4, m_2, m_5, m_1)$, and $Q(m_0, m_1, m_2, m_3, m_4, m_5, m_6) = (m_0 \oplus (m_2 \wedge m_3), m_1, m_2, \bar{m_3} \oplus (m_5 \wedge m_6), m_4, \bar{m_5} \oplus (m_2 \wedge m_4), m_6)$.

Throughout the paper, the symbols $\oplus$, $\vee$, $\wedge$ and the *bar* represent bit-wise XOR, OR, AND, and NOT operations, respectively.

III. PROPOSED ERROR DETECTION ARCHITECTURES

In this section, the efficient and overhead-aware error detection schemes for the nonlinear layer of WAGE (Figure 1) are presented. In the hardware implementations, two variants of the nonlinear layer of WAGE can be utilized: logic-based or LUT-based. Our contribution to the error detection schemes is in two parts, i.e., for both logic gate-based and LUT-based variants of the nonlinear layer of WAGE. The former is better for ASIC implementations of the cipher as register units are costly in ASIC, while the latter is better for the FPGA-

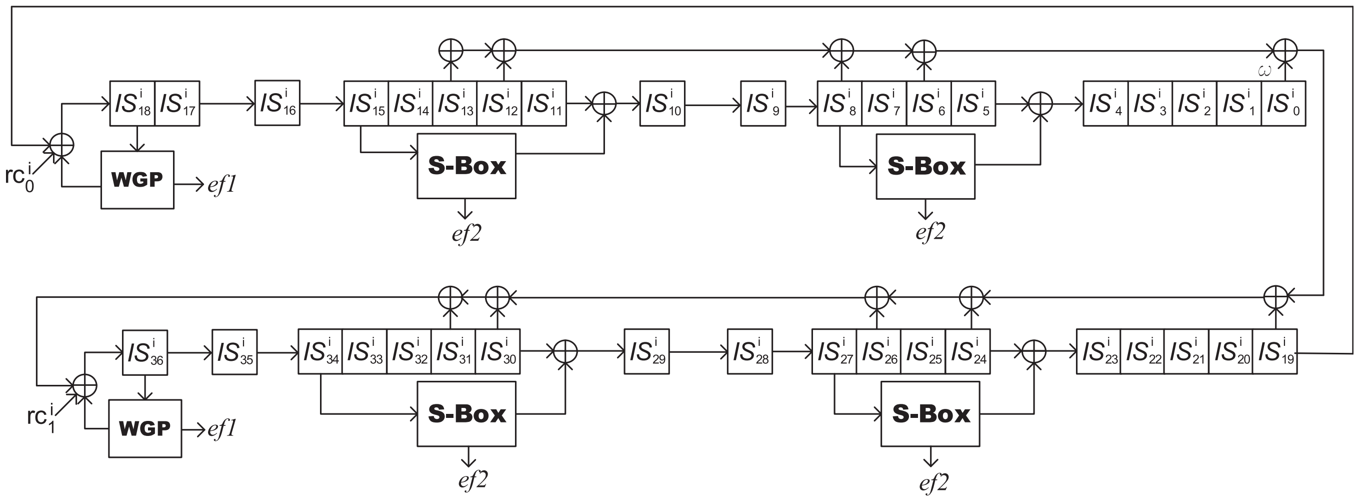

FIGURE 1. **The proposed architecture-oblivious error detection scheme depicted in the ith round of WAGE. The flags $ef1$ and $ef2$ show the error indication flags of the proposed schemes for the WGP and the S-Box, respectively.**

TABLE 1. **LUT representation of the WGP of WAGE in hexadecimal form.**

00	12	0A	4B	66	0C	48	73	79	3E	61	51	01	15	17	0E
7E	33	68	36	42	35	37	5E	53	4C	3F	54	58	6E	56	2A
1D	25	6D	65	5B	71	2F	20	06	18	29	3A	0D	7A	6C	1B
19	43	70	41	49	22	77	60	4F	45	55	02	63	47	75	2D
40	46	7D	5C	7C	59	26	0B	09	03	57	5D	27	78	30	2E
44	52	3B	08	67	2C	05	6B	2B	1A	21	38	07	0F	4A	11
50	6A	28	31	10	4D	5F	72	39	16	5A	13	04	3C	34	1F
76	1E	14	23	1C	32	4E	7B	24	74	7F	3D	69	64	62	6F

based implementations where memory units such as LUTs can be utilized. Signatures, e.g., interleaved or single/multiple parity bits, represent stored data efficiently. In our proposed CED schemes for the WGP and S-Box of WAGE, the signatures are computed via modulo-2 addition using the input and output vectors of the said components. For one-bit signature, the output bits are modulo-2 added with each other (one-bit) while the interleaved signatures are computed by separately performing the modulo-2 addition of odd bits and even bits of the output vector. Both the error flags $ef1$ and $ef2$ in Figure 1 have either one or two outputs for one bit signature and interleaved signatures, respectively. The predicted parity uses the input vectors and is either stored within a LUT or computed using logic equations, while the actual parity is computed using modulo-2 addition of the interleaved bits of the output vectors. The two parities are then compared with each other to generate the error flags which give an output of $'1'$ if a bit-error is detected or an output of $'0'$ otherwise. Detailed equations for logic-gate based implementation of the S-box and WGP are presented in this section.

A. PROPOSED SIGNATURE-BASED SCHEMES FOR WGP

The hexadecimal representation of WGP of WAGE is given in Table 1. WGP takes its cryptographic properties from the WG permutation and transformations [4]. A decimated WGP is used to provide high nonlinearity and low differential uniformity. The finite-field $\mathbb{F}_{2^7}$ is chosen for the WGP for low hardware overhead and for low software complexity. The formulations are based on the input ($\mu = (\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6)$), and output ($\nu = (\nu_0, \nu_1, \nu_2, \nu_3, \nu_4, \nu_5, \nu_6)$) vectors of WGP. For example, for the input value $(01)_{16} = (0000001)_2$ to the WGP, the corresponding output value of $(12)_{16} = (0010010)_2$ will have one-bit parity as 0 while its interleaved parities will be (1,1). The one-bit signature $(\hat{\rho_0})$ and interleaved signatures $(\hat{\rho_1}, \hat{\rho_2})$ of the WGP for the LUT-based approach are shown in Tables 2 and 3, and are stored along with the LUT of WGP.

B. PROPOSED SIGNATURE-BASED SCHEMES FOR THE S-BOX

Similar to WGP, the hexadecimal form of the S-Box of WAGE is shown in Table 4, which is computed as explained in Section II. The nonlinear transformations Q of the S-Box were chosen such that they differed across all 7! (5040) bit permutations P and had hardware efficiency. This ensured that the S-Box had good nonlinearity and differential uniformity with small hardware cost.

For the input $m = (m_0, m_1, m_2, m_3, m_4, m_5, m_6)$ and output $n = (n_0, n_1, n_2, n_3, n_4, n_5, n_6)$, the equations for the logic-gate based implementation of S-Box, along with signature and interleaved signature bits for the LUT-based schemes

TABLE 2. Signature $(\hat{\rho}_0)$ of the WGP of WAGE.

0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	0	0	1	1	0	1	0	1	1	1	0	1
0	1	1	0	1	0	1	1	0	0	1	0	1	1	0	0
1	1	1	0	1	0	0	0	1	1	0	1	0	0	1	0
1	1	0	0	1	0	1	1	0	0	1	1	0	0	0	0
0	1	1	1	1	1	0	1	0	1	0	1	1	0	1	0
0	0	0	1	1	0	0	0	0	1	0	1	1	0	1	1
1	0	0	1	1	1	0	0	0	0	1	1	0	1	1	0

TABLE 5. Signature $(\hat{p}_0)$ of the 7-bit S-Box of WAGE.

0	1	1	0	0	1	1	1	0	1	1	0	1	0	0	0
1	1	1	0	1	1	1	0	0	1	1	0	1	0	0	0
1	0	1	1	1	1	0	1	0	0	1	1	0	1	1	0
0	0	1	0	0	0	0	0	0	1	1	1	0	1	0	0
0	1	0	1	1	0	0	1	0	1	0	0	0	1	0	0
0	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0
0	1	0	1	1	0	1	0	0	0	1	0	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0

are derived. Tables 5 and 6 show the one-bit signature $(\hat{p_0})$ and interleaved signatures $(\hat{p_1}, \hat{p_2})$ in row-major, corresponding to each 7-bit element of the S-Box for the LUT-based approach. For the LUT-based approach, the predicted signatures of the S-Box are computed and stored as explained for WGP above.

C. DERIVED FORMULATIONS

For the the input vectors $(\mu = (\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6))$ and $(m = (m_0, m_1, m_2, m_3, m_4, m_5, m_6))$, the following notation will be used in the equations for the sake of brevity. These are shown in the table below, where as seen, we use either numbers or numbers with bars to show the inputs of inverted versions:

$\mu_0/m_0 = 0$	$\bar{\mu_0}/\bar{m_0} = \bar{0}$	$\mu_1/m_1 = 1$	$\bar{\mu_1}/\bar{m_1} = \bar{1}$
$\mu_2/m_2 = 2$	$\bar{\mu_2}/\bar{m_2} = \bar{2}$	$\mu_3/m_3 = 3$	$\bar{\mu_3}/\bar{m_3} = \bar{3}$
$\mu_4/m_4 = 4$	$\bar{\mu_4}/\bar{m_4} = \bar{4}$	$\mu_5/m_5 = 5$	$\bar{\mu_5}/\bar{m_5} = \bar{5}$
$\mu_6/m_6 = 6$		$\bar{\mu_6}/\bar{m_6} = \bar{6}$	

WGP Derivations. For the the input $(\mu = (\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6))$ and output $(\nu = (\nu_0, \nu_1, \nu_2, \nu_3, \nu_4, \nu_5, \nu_6))$ vectors of the WGP, the low complexity equations for the logic-gate based variants, along with signature and interleaved signature for the LUT-based approach are derived as follows:

$$\begin{aligned}\nu_0 = {} & (245)(3 \oplus 0) \vee (35)(\bar{0}(6 \oplus 1)(6 \oplus 4)) \vee (\bar{0}\bar{4}) \\ & \times (\bar{1}\bar{6}(2 \oplus 3) \vee 12\bar{3}6) \vee (\bar{2}4\bar{5}6)(0 \oplus 1) \vee (\bar{0}\bar{1}\bar{2}\bar{3}5) \\ & \times (4 \vee 6) \vee (\bar{3}\bar{5}\bar{6})(0 \oplus 4) \vee (12)(034 \vee \bar{0}5\bar{6}) \vee (23\bar{5}) \\ & \times (\bar{0} \vee 16) \vee (\bar{3}\bar{5})(\bar{1}4\bar{6} \vee 0\bar{2}6) \vee (\bar{4}5)(0\bar{1}\bar{2} \vee 013\bar{6}) \\ & \vee (\bar{3})(\bar{4}(0 \oplus 1)(0 \oplus 5) \vee 0145)\end{aligned}$$

$$\begin{aligned}\nu_1 = {} & (0\bar{5}\bar{6})((4 \oplus 1)(4 \oplus 3) \vee 12\bar{4}) \vee (\bar{0}456)(2 \oplus \bar{3}) \vee (02) \\ & \times (3 \oplus 4)(3 \oplus 6) \vee 1(6 \oplus 2)(6 \oplus 3)(6 \oplus 4) \vee \bar{1}(2 \oplus 0) \\ & \times (2 \oplus 3)(2 \oplus 4) \vee (\bar{0}1\bar{3}4).(6 \vee 5) \vee (13)(45\bar{6} \vee 02) \\ & \vee (\bar{4}5)(\bar{1}6(\bar{0}3 \vee 2) \vee \bar{0}1\bar{2} \vee 023) \vee (\bar{0}\bar{1}\bar{5})(246 \vee \bar{2}3\bar{4}) \\ & \vee (\bar{3}5)(\bar{0}2\bar{6} \vee 016 \vee 0\bar{2}\bar{6}(\bar{4} \vee \bar{1}) \vee (\bar{2}4\bar{5})(\bar{1}\bar{3}\bar{6} \vee 136)\end{aligned}$$

TABLE 3. Interleaved-signature $(\hat{\rho}_1, \hat{\rho}_2)$ of the WGP of WAGE.

0,0	1,1	0,0	0,0	0,0	1,1	1,1	1,0	1,0	0,1	0,1	1,0	1,0	1,0	1,1	1,0
1,1	0,0	1,0	0,0	1,1	1,1	1,0	1,0	1,1	0,1	1,1	1,0	0,1	0,1	1,1	0,1
1,1	0,1	1,0	1,1	1,0	1,1	0,1	0,1	1,1	1,1	1,0	1,1	0,1	0,1	0,0	0,0
0,1	0,1	0,1	0,0	0,1	0,0	0,0	1,1	1,0	1,0	0,0	0,1	0,0	1,1	0,1	0,0
1,0	0,1	0,0	1,1	1,0	1,1	1,0	1,0	1,1	1,1	0,1	0,1	0,0	0,0	1,1	1,1
0,0	0,1	0,1	0,1	1,0	1,0	0,0	0,1	1,1	1,0	1,1	1,0	0,1	0,0	1,0	0,0
0,0	1,1	0,0	0,1	1,0	1,1	0,0	0,0	0,0	0,1	0,0	0,1	1,0	0,0	0,1	1,0
1,0	0,0	0,0	1,0	0,1	1,0	0,0	1,1	1,1	1,1	0,1	1,0	0,0	0,1	1,0	1,1

TABLE 4. LUT representation of the 7-bit S-Box of WAGE in hexadecimal form.

2E	1C	6D	2B	35	07	7F	3B	28	08	0B	5F	31	11	1B	4D
6E	54	0D	09	1F	45	75	53	6A	5D	61	00	04	78	06	1E
37	6F	2F	49	64	34	7D	19	39	33	43	57	60	62	13	05
77	47	4F	4B	1D	2D	24	48	74	58	25	5E	5A	76	41	42
27	3E	6C	01	2C	3C	4E	1A	21	2A	0A	55	3A	38	18	7E
0C	63	67	56	50	7C	32	7A	68	02	6B	17	7B	59	71	0F
30	10	22	3D	40	69	52	14	36	44	46	03	16	65	66	72
12	0E	29	4A	4C	70	15	26	79	51	23	3F	73	5B	20	5C

TABLE 6. Interleaved-signature ($\hat{p_1}, \hat{p_2}$) of the 7-bit S-Box of WAGE.

1,1	0,1	1,0	1,1	1,1	0,1	0,1	0,1	0,0	0,1	1,0	0,0	0,1	0,0	0,0	1,1
0,1	1,0	0,1	1,1	1,0	1,0	0,1	1,1	1,1	0,1	0,1	0,0	1,0	0,0	1,1	0,0
1,0	1,1	0,1	0,1	0,1	0,1	0,0	0,1	0,0	0,0	0,1	0,1	1,1	1,0	0,1	0,0
0,0	1,1	1,0	0,0	1,1	0,0	1,1	1,1	1,1	0,1	0,1	1,0	0,0	1,0	0,0	1,1
0,0	0,1	0,0	1,0	1,0	0,0	0,0	1,0	1,1	0,1	0,0	0,0	1,1	1,0	1,1	1,1
1,1	0,0	1,0	1,1	0,0	1,0	1,0	0,1	1,0	0,1	0,1	1,1	1,1	1,1	1,1	0,0
1,1	1,0	0,0	1,0	1,0	0,0	0,1	0,0	0,0	0,0	0,1	1,1	0,1	1,1	0,0	0,0
1,1	1,0	1,0	1,0	0,1	0,1	1,0	1,0	1,0	1,0	1,0	1,1	1,0	1,0	0,1	1,1

$$\begin{aligned}
\nu_2 = {} & (026)(\bar{1}35 \vee \bar{4}\bar{5}) \vee (5)(\bar{0}24\bar{6} \vee 0\bar{2}\bar{4}6) \vee (\bar{2}\bar{4}\bar{5})(6 \oplus 0) \\
& \times (6 \oplus 1) \vee (\bar{2}\bar{5})(\bar{0}(3 \oplus 1)(3 \oplus 4) \vee 0\bar{1}\bar{3}4) \vee (\bar{3}5) \\
& \times (\bar{1}(0 \oplus 4)(0 \oplus 6) \vee 0146) \vee (\bar{4}5)(\bar{1}36 \vee 12\bar{6} \vee 013) \\
& \vee (\bar{2}3)(5\bar{6}(0 \vee \bar{1}4) \vee 6(1 \vee 4\bar{5})) \vee (01\bar{3})(2\bar{5} \vee \bar{2}4\bar{6}) \\
& \vee (\bar{0}\bar{1}2)(3 \oplus 6) \vee (\bar{3}\bar{5}\bar{6})(\bar{0}2\bar{4} \vee 1\bar{2})
\end{aligned}$$

$$\begin{aligned}
\nu_3 = {} & (\bar{2}5)(3 \oplus \bar{4})(3 \oplus \bar{6}) \vee (\bar{2}\bar{3}4)(6 \oplus 1)(6 \oplus 5) \vee (\bar{4}5\bar{6}) \\
& \times (2 \oplus 1)(2 \oplus 3) \vee (56)(1 \oplus \bar{2})(1 \oplus \bar{3})(1 \oplus \bar{4}) \vee (0\bar{1}\bar{2}) \\
& \times (4 \oplus 3)(4 \oplus 5)(4 \oplus 6) \vee (\bar{4}\bar{5}6)(\bar{0}\bar{2}3 \vee 01\bar{3}) \\
& \vee \bar{4}(\bar{0}\bar{2}3\bar{5}6 \vee 2(0 \oplus \bar{3})(0 \oplus \bar{5})(0 \oplus \bar{6})) \vee (\bar{5}\bar{6})(3 \oplus 1) \\
& \times (3 \oplus 2)(3 \oplus 4) \vee (\bar{5}6)(01\bar{3}\bar{4} \vee \bar{1}23) \vee (14) \\
& \times (02\bar{3}5 \vee \bar{0}\bar{2}\bar{5}\bar{6}) \vee (46)(1 \oplus 2)(3 \oplus 5)(1 \oplus \bar{3}) \vee (0\bar{5}) \\
& \times (1 \oplus 2)(3 \oplus 4)(3 \oplus 6) \vee (\bar{0}\bar{1}5\bar{6})(4 \oplus 2)(4 \oplus 3) \\
& \vee (\bar{0}1\bar{2}\bar{4})(3 \oplus \bar{5})(3 \oplus \bar{6}) \vee (0\bar{1}45)(6 \oplus 2).(6 \oplus 3) \\
& \vee (\bar{0}13\bar{6})(2 \oplus 4)(2 \oplus 5) \vee (0\bar{1}6)(2 \oplus 3)(2 \oplus 4)(2 \oplus \bar{5}) \\
& \vee (234)(0 \oplus \bar{1})(0 \oplus 5)(0 \oplus 6) \vee (23\bar{6})(4 \oplus 0)(4 \oplus 1) \\
& \times (4 \oplus 5)
\end{aligned}$$

$$\begin{aligned}
\nu_4 = {} & (\bar{0}\bar{1}25)(3 \oplus 4)(3 \oplus 6) \vee (0\bar{1}\bar{2}\bar{3})(6 \oplus 4)(6 \oplus 5) \vee (24) \\
& \times (5 \oplus 3)(5 \oplus 6) \vee (136)(4 \oplus \bar{5})(0 \vee 2) \vee (012\bar{4}6) \\
& \times (5 \oplus \bar{3}) \vee (\bar{2}5)(0\bar{1}\bar{4} \vee 14\bar{6}) \vee (\bar{0}\bar{2}5\bar{6})(1 \oplus 3)(1 \oplus 4) \\
& \vee (\bar{2}4\bar{5}6)(0 \oplus \bar{1}) \vee (13\bar{6})(\bar{2}4\bar{5} \vee 2\bar{4}) \vee (\bar{0}\bar{4})(1 \oplus 2) \\
& \times (1 \oplus 5)(1 \oplus 6) \vee (\bar{1}3\bar{5})(6 \oplus 0)(6 \oplus 4) \\
& \vee (1236).(4 \oplus \bar{5}) \vee (1\bar{3}\bar{4})(6 \oplus 0)(6 \oplus 2) \vee (24\bar{6}) \\
& \times (0 \oplus 3)(0 \oplus 5) \vee (\bar{1}\bar{2}4).(\bar{3}\bar{5}\bar{6} \vee 356) \vee (12\bar{3}\bar{5}) \\
& \times (4 \oplus \bar{6})
\end{aligned}$$

$$\begin{aligned}
\nu_5 = {} & (\bar{0}\bar{1}\bar{3}\bar{4})(\bar{2}5 \vee 6) \vee (\bar{1}23)(0\bar{5} \vee \bar{0}\bar{4}\bar{6}) \vee (\bar{4}\bar{5}6)(\bar{1}\bar{2} \vee 0\bar{3}) \\
& \vee (123)(05\bar{6} \vee \bar{0}4\bar{5}) \vee (\bar{4}5)(\bar{0}136 \vee 0\bar{1}2\bar{3}\bar{6}) \vee (2\bar{3}\bar{4}\bar{5}) \\
& \times (0 \oplus \bar{1}) \vee (\bar{0}\bar{3}4\bar{6})(2 \oplus \bar{5}) \vee (035\bar{6})(2 \oplus \bar{4}) \vee (12\bar{3}6) \\
& \times (0 \vee \bar{5}) \vee (1\bar{2}36)(0 \oplus 4) \vee (3\bar{5}\bar{6})((1 \oplus 0)(1 \oplus 4)) \\
& \vee (\bar{1}24\bar{5})(3 \oplus \bar{6}) \vee (45)(012 \vee \bar{2}36 \vee 0\bar{2}\bar{3} \vee \bar{0}\bar{1}3) \\
& \vee (\bar{3}45)(1 \oplus 6)
\end{aligned}$$

$$\begin{aligned}
\nu_6 = {} & (\bar{0}\bar{2}\bar{4}5)(1 \oplus 6) \vee (0\bar{2}\bar{5})(3\bar{4}\bar{6} \vee \bar{3}46) \vee (24\bar{6}) \\
& \times (\bar{0}\bar{3}5 \vee 13\bar{5}) \vee (\bar{2}\bar{3}45)(1 \oplus 6) \vee (0\bar{1}24)(3 \oplus \bar{6}) \\
& \vee 3(0\bar{1}\bar{2}\bar{4} \vee \bar{0}124) \vee (\bar{0}1)(4\bar{5}\bar{6} \vee \bar{3}\bar{4}6 \vee \bar{2}\bar{3}\bar{5} \vee 2\bar{4}\bar{5}) \\
& \vee (456)(13 \vee 02) \vee (0\bar{4}5)(1 \oplus \bar{6}) \vee (3\bar{6}) \\
& \times (0\bar{1}\bar{5} \vee \bar{0}2\bar{4} \vee 2\bar{4}5 \vee \bar{0}\bar{1}\bar{2}) \vee (\bar{0}\bar{1}\bar{5})(\bar{2}34 \vee 2\bar{3}6)
\end{aligned}$$

$$\begin{aligned}
\hat{\rho}_0 = {} & (0135)(2 \oplus 4) \vee (\bar{0}\bar{2}3)(4\bar{5} \vee \bar{1}\bar{4}) \vee (\bar{3}5) \\
& \times (\bar{1}2(0 \oplus 4) \vee \bar{0}1\bar{2}4) \vee (\bar{4}5)(\bar{1}\bar{2}3 \vee 016 \vee \bar{0}2\bar{3}\bar{6}) \\
& \vee (4\bar{5}\bar{6})(1\bar{2} \vee 0\bar{3}) \vee (0\bar{1}\bar{2}\bar{3})(4 \oplus \bar{5}) \vee (1)(5 \oplus 0) \\
& \times (5 \oplus 2)(5 \oplus 4)(5 \oplus 6) \vee (12)(\bar{5}(\bar{0}\bar{4} \vee \bar{3}\bar{6}) \vee 345\bar{6}) \\
& \vee (\bar{4}6)(1(\bar{0}\bar{3}\bar{5} \vee 0\bar{2}3) \vee 23(\bar{0} \vee \bar{1})) \vee (0\bar{1}2)(\bar{3}6 \vee 34\bar{6}) \\
& \vee (\bar{0}\bar{1}4)(56 \vee 3\bar{5})
\end{aligned}$$

$$\begin{aligned}
\hat{\rho}_1 = {} & (6 \oplus 0)(6 \oplus 1)(6 \oplus 2)(6 \oplus 3)(6 \oplus 4)(6 \oplus 5) \vee (\bar{5}) \\
& \times (2 \oplus \bar{3})(4 \oplus \bar{6}).(2 \oplus 4) \vee (01\bar{2}5)(6 \oplus 3)(6 \oplus 4) \\
& \vee (\bar{1}\bar{3}\bar{6})(0 \oplus 2)(0 \oplus 5) \vee (1\bar{4}).(0256 \vee \bar{0}\bar{2}\bar{6}) \vee (\bar{0}\bar{3}\bar{4}) \\
& \times (2 \oplus 1)(2 \oplus 5)(2 \oplus 6) \vee (\bar{1}\bar{3}56)(0 \oplus 2).(0 \oplus 4) \\
& \vee (012\bar{3}\bar{5})(4 \oplus \bar{6}) \vee (\bar{0}13\bar{5}6)(2 \oplus \bar{4}) \vee (0\bar{2}3) \\
& \times (\bar{1}4\bar{5} \vee \bar{4}56) \vee (1 \oplus \bar{2})(1 \oplus \bar{6})(3 \oplus 4)(3 \oplus 5) \\
& \vee (1\bar{2}\bar{3}4\bar{5}\bar{6}) \vee (\bar{1}45)(\bar{2}6 \vee 3\bar{6}) \vee (\bar{1}2)(4 \oplus 3)(4 \oplus 5) \\
& \vee (036)(1 \oplus \bar{4})(1 \oplus \bar{5})
\end{aligned}$$

$$\begin{aligned}\hat{\rho}_2 = {} & (124\bar{5})(3 \oplus \bar{6}) \vee (3\bar{4}\bar{5}6)(0 \oplus \bar{1}) \vee (\bar{1}3)(0 \oplus 2)(0 \oplus 4)\\ & \vee (345\bar{6}).(0 \oplus 2) \vee (\bar{3}45)(026 \vee \bar{0}\bar{2}\bar{6}) \vee (\bar{2}6)(\bar{3}\bar{5} \vee 1\bar{4})\\ & \vee (2\bar{4}\bar{6})(\bar{0}1\bar{3} \vee 03) \vee (025)(\bar{1}3\bar{4} \vee 146) \vee (\bar{0}156)\\ & \times (3 \oplus 4) \vee (\bar{3}\bar{4})(\bar{0}1\bar{5} \vee 0\bar{1}6) \vee (\bar{2}3)(1 \oplus 0)(1 \oplus 5)\\ & \vee (\bar{0}\bar{1}2\bar{5})(\bar{6} \vee 4) \vee (\bar{1}3\bar{6})(2\bar{5} \vee \bar{0}5).\end{aligned}$$

S-Box Derivations.

For the input ($m = (m_0, m_1, m_2, m_3, m_4, m_5, m_6)$) and output ($n = (n_0, n_1, n_2, n_3, n_4, n_5, n_6)$) vectors of the 7-bit S-Box of WAGE, the low complexity equations for the logic-gate based variants, along with signature and interleaved signature for the LUT-based approach are derived as follows:

$$\begin{aligned}n_0 = {} & (\bar{1}\bar{2}5)(3 \oplus \bar{6}) \vee (\bar{0}26)(1 \oplus \bar{5}) \vee (0\bar{1}2)(3 \oplus 6) \vee (5)\\ & \times (\bar{1}2\bar{3}(0 \oplus 4) \vee \bar{0}1\bar{2}3\bar{4}) \vee (16)(0 \oplus \bar{3})(0 \oplus \bar{4}) \vee (\bar{0}12)\\ & \times (3 \oplus \bar{4}) \vee (\bar{6})(1 \oplus \bar{4})(1 \oplus 2)(1 \oplus 3) \vee (14\bar{5})(0 \oplus 3)\\ & \vee (2\bar{5})(0 \oplus \bar{4}) \vee (05)(2\bar{3}\bar{4}6 \vee 1\bar{2}3\bar{6}) \vee (13)(0\bar{5}6 \vee 2\bar{5})\end{aligned}$$

$$\begin{aligned}n_1 = {} & (01\bar{4}5)(2 \oplus \bar{3}) \vee (\bar{2}5)(0134 \vee \bar{0}\bar{1}\bar{3}) \vee (\bar{2}\bar{5})(0 \oplus 1)\\ & \vee (046)(2 \oplus 3) \vee (\bar{6})((0 \oplus \bar{2})(0 \oplus \bar{3}) \vee 23\bar{4} \vee \bar{0}\bar{2}\bar{5})\\ & \vee (\bar{4}\bar{5}\bar{6})(\bar{0} \vee \bar{2}) \vee (\bar{3}5\bar{6})(0 \oplus 4) \vee (0\bar{1})(\bar{3}\bar{5}6 \vee 25\bar{6})\\ & \vee (4\bar{5}6)(\bar{0}23 \vee 1\bar{3})\end{aligned}$$

$$\begin{aligned}n_2 = {} & (\bar{1}4\bar{6})(0 \oplus \bar{2})(0 \oplus \bar{3}) \vee (1\bar{2}\bar{4}6)(0 \oplus 3) \vee (\bar{1}2\bar{4}6)(0 \oplus \bar{5})\\ & \vee (\bar{0}345)(1 \oplus 2)(1 \oplus 6) \vee (\bar{0}1\bar{3}4\bar{5})(2 \oplus 6) \vee (1\bar{4})\\ & \times (\bar{5}\bar{6} \vee 236) \vee (014\bar{6} \vee \bar{1}46)(3 \oplus 2)(3 \oplus 5) \vee (\bar{5})\\ & \times (6 \oplus 1)(6 \oplus 2)(6 \oplus 3)(6 \oplus 4) \vee (\bar{2}5)(1 \oplus \bar{4})(1 \oplus 3)\\ & \times (1 \oplus 6) \vee (\bar{1}4\bar{6})(2 \oplus 3) \vee (046)(2 \oplus 3)(2 \oplus 5)\\ & \vee (46)(5 \oplus 2)(5 \oplus 3) \vee (046)(1 \oplus \bar{2})(1 \oplus \bar{3})(1 \oplus \bar{5})\end{aligned}$$

$$\begin{aligned}n_3 = {} & (12\bar{3}6)(0 \oplus 4) \vee (3\bar{4}\bar{5}\bar{6})(0 \oplus \bar{2}) \vee (\bar{5}6)(0 \oplus \bar{4})(0 \oplus 2)\\ & \times (0 \oplus 3) \vee (2\bar{4})(1 \oplus \bar{5})(1 \oplus \bar{6}) \vee (56)(1 \oplus 3)(1 \oplus 4)\\ & \vee 6(0 \oplus \bar{2})(0 \oplus \bar{3})(0 \oplus \bar{4}) \vee (\bar{1}\bar{2})(0 \oplus \bar{4}) \vee (\bar{0}2\bar{5}\bar{6})\\ & \times (14 \vee \bar{1}\bar{3}) \vee (0\bar{1}3)(\bar{4}5\bar{6} \vee 4\bar{5}) \vee (12\bar{3})(\bar{4}5 \vee 4\bar{5}\bar{6})\\ & \vee (\bar{0}\bar{3}5)(\bar{4} \vee \bar{2}) \vee (46)(0\bar{1} \vee \bar{1}23) \vee (\bar{1}\bar{2})(\bar{4}\bar{5}6 \vee 5\bar{6})\end{aligned}$$

$$\begin{aligned}n_4 = {} & (26)(0 \oplus \bar{5})((0 \oplus \bar{1})(0 \oplus \bar{4}) \vee (0 \oplus 1)(0 \oplus 4)) \vee (\bar{0}2\bar{6})\\ & \times (1 \oplus 4) \vee (3 \oplus \bar{5})((12\bar{4}6) \vee (\bar{1}46)) \vee (\bar{5} \vee \bar{6})(\bar{1}\bar{2}\bar{3})\\ & \vee (\bar{0}\bar{3}) \vee (356)(\bar{0}\bar{2} \vee 0\bar{1}) \vee (01\bar{2})(3(\bar{5} \vee \bar{6}) \vee \bar{3}56)\\ & \vee (\bar{3}\bar{6})(\bar{1}\bar{4} \vee 124)\end{aligned}$$

$$\begin{aligned}n_5 = {} & (3456)(0 \oplus 2) \vee (\bar{0}\bar{5})(2 \oplus 1)(2 \oplus 3)(2 \oplus 6) \vee (\bar{3}\bar{4}\bar{5})\\ & \times (12 \vee \bar{1}\bar{2}\bar{6}) \vee (4)(5 \oplus 1)(5 \oplus 2)(5 \oplus 3) \vee (03\bar{6})\\ & \times (4 \oplus 5) \vee (\bar{0}\bar{1}\bar{4})(2 \oplus 5)(2 \oplus 6) \vee (\bar{0}\bar{1}4)(\bar{2}\bar{3}6 \vee 235)\\ & \vee (01\bar{2}\bar{6})(3 \vee 5) \vee (5)(3 \oplus 0)(3 \oplus 2)(3 \oplus 6) \vee (56)\\ & \times (13\bar{4} \vee \bar{1}24) \vee (\bar{0}1\bar{3}\bar{4})(\bar{5} \vee \bar{6}) \vee (0\bar{1}\bar{4})(\bar{5}6 \vee 25)\\ & \vee (2\bar{3})(\bar{0}1\bar{4} \vee 0\bar{1}5)\end{aligned}$$

$$\begin{aligned}n_6 = {} & (\bar{0}\bar{2})(1 \oplus 4 \vee 5) \vee (\bar{1}2\bar{4}\bar{5}6)(0 \oplus 3) \vee (0\bar{2}\bar{5})(1 \oplus \bar{4})\\ & \times (1 \oplus \bar{6}) \vee (03\bar{4}56) \vee (\bar{4}5)(2 \oplus 6) \vee (15\bar{6})(\bar{0}3 \vee 02\bar{3})\\ & \vee (\bar{0}\bar{3})(\bar{1}5 \vee 1\bar{4}) \vee (023)(\bar{1}4 \vee 1\bar{5}) \vee (\bar{0}2\bar{3}4\bar{5})\end{aligned}$$

$$\begin{aligned}\hat{p_0} = {} & (45\bar{6})(0 \oplus \bar{1})(0 \oplus \bar{2})(0 \oplus \bar{3}) \vee (\bar{3}4\bar{6})(1 \oplus 2)(1 \oplus 5)\\ & \vee (2\bar{3}\bar{5})(\bar{0}\bar{1}\bar{6} \vee 014) \vee (\bar{4}5)(3 \oplus 0)(3 \oplus 1)(3 \oplus 2)\\ & \times (3 \oplus 6) \vee (13)(5 \oplus 2)(5 \oplus 4)(5 \oplus 6) \vee (\bar{1}\bar{3}\bar{5})\\ & \times ((0 \oplus \bar{4})(0 \oplus 2)(0 \oplus 6)) \vee (\bar{1}\bar{5}6)(2\bar{3}4 \vee \bar{2}\bar{4}) \vee (\bar{2}\bar{5})\\ & \times (1 \oplus 0)(1 \oplus 3)(1 \oplus 4)(1 \oplus 6) \vee (\bar{3}5)(0 \oplus \bar{2})(0 \oplus \bar{4})\\ & \times (0 \oplus \bar{6}) \vee (\bar{0}1\bar{2}46)(3 \oplus 5) \vee (\bar{4}\bar{5}6)(01\bar{3} \vee 23)\\ & \vee (2\bar{4}5\bar{6}) \vee (014\bar{6})(3 \oplus 5) \vee (\bar{2}5)(0 \oplus \bar{6})(0 \oplus 1)\\ & \times (0 \oplus 3)(0 \oplus 4) \vee (\bar{2}\bar{3}56)(1 \oplus 4) \vee (\bar{0}3)(4 \oplus 1)\\ & \times (4 \oplus 5)(4 \oplus 6) \vee (\bar{5})(4 \oplus 0)(4 \oplus 2)(4 \oplus 3)(4 \oplus \bar{6})\end{aligned}$$

$$\begin{aligned}\hat{p_1} = {} & (0\bar{4}56)(2 \vee 1) \vee (\bar{1}4)(\bar{3}\bar{5}(2 \oplus \bar{6}) \vee \bar{2}356) \vee (2\bar{4})\\ & \times (3\bar{5}\bar{6} \vee 01) \vee (0\bar{2}\bar{3})(1 \oplus 5)(1 \oplus 6) \vee (125)(3 \oplus \bar{6})\\ & \vee (\bar{0}\bar{3}\bar{5}\bar{6})(2 \oplus \bar{4}) \vee 6((5 \oplus 1)(5 \oplus 3)(5 \oplus 4) \vee 1\bar{3}\bar{4}\bar{5})\\ & \vee (\bar{0}\bar{2})(\bar{1}\bar{4}5\bar{6} \vee 134\bar{5}) \vee (24)(\bar{1}3\bar{6} \vee 1\bar{3}5) \vee (02\bar{3}\bar{6})\\ & \times (\bar{4} \vee 5) \vee \bar{1}(0\bar{2}34 \vee \bar{0}2\bar{3}6) \vee (03\bar{5})(\bar{1}\bar{6} \vee 24)\end{aligned}$$

$$\begin{aligned}\hat{p_2} = {} & ((014\bar{5}) \vee (\bar{0}14\bar{6}))(2 \oplus 3) \vee (0235)(1 \oplus \bar{4}) \vee (0\bar{1}3\bar{5}) \\ & \times (2 \oplus \bar{6}) \vee (\bar{1}\bar{4}\bar{5}6)(\bar{2} \vee 3) \vee (01\bar{2}\bar{3}45\bar{6}) \vee (\bar{0}\bar{2}) \\ & \times (1\bar{4}5 \vee \bar{3}6) \vee (\bar{0}\bar{6})(\bar{2}4\bar{5} \vee \bar{1}25) \vee (\bar{1}) \\ & \times ((2 \oplus 3)(2 \oplus 4)(2 \oplus 5)(2 \oplus 6)) \vee (1\bar{4}\bar{5})(0\bar{3}\bar{6} \vee \bar{0}26) \\ & \vee (\bar{0}\bar{1}\bar{2}\bar{3})(\bar{5} \vee 4) \vee (5)(\bar{1}2\bar{3}6 \vee 1\bar{2}3\bar{4}) \vee (256) \\ & \times (\bar{0}14 \vee 03\bar{4}) \vee (3\bar{6})(\bar{0}2\bar{4} \vee 0\bar{1}4).\end{aligned}$$

IV. ERROR COVERAGE AND FPGA BENCHMARK

This section presents the error coverage and overhead benchmark of the proposed error detection schemes. Often, in fault attacks, the intelligent adversaries repeatedly compare the faulty and non-faulty outputs to get the sub-keys and eventually the secret key. In the work of [1], DFA on WG-l ciphers (where $l = 7, 8, 16, 29$) refers to the different key length bits) were implemented and verified. Here, the six randomly placed faults are injected into the *IS*, and the secret key is recovered by comparing faulty while non-faulty keystreams once the locations and values of the injected faults are determined. As discussed in [8], SFAs are preferred for the AEAD ciphers where the nonce is mixed with the key in the initialization phase. Here, the attacker relies on biased fault models to manipulate a bit or a byte of the *IS*. The generated faulty ciphertexts are then compared to the non-faulty ciphertexts, through which the secret key can be recovered. Even with added security measures, since WAGE uses both AEAD and WG-7 transformation, it could potentially be vulnerable to SFAs and DFAs.

A. FAULT MODEL

Our proposed error detection schemes are designed to detect a number of faults due to the nature of the proposed signature-based schemes such as stuck-at faults, single-event upsets, single-byte double-bit upset, single-byte triple bit upset (SBTBU), other single byte (OSB) faults, and MBU. Specifically, SEUs and SBTBUs are detected fully through parities. Generally, single-bit stuck-at models are considered for fault attacks (malicious fault injections). As injecting single-stuck at faults becomes costly (due to technological constraints), therefore multi-bit stuck-at faults are often preferred by adversaries. A stuck-at-fault model (both single and multiple) replicates both natural and malicious faults and therefore is chosen as the base model for error simulations performed. Commonly, we notice SEU and MBU more often and widespread, but adjacent MBU could also occur. However, in this work, we focus on SEU and MBU, for simulating our error detection schemes. For the error coverage simulations, we have considered a permanent fault model (both the single and multiple stuck-at faults) to cover natural and malicious (biased) fault injections at the RTL level.

TABLE 7. Error coverage of permanent faults using the proposed schemes for WAGE derived through our simulations.

WAGE Architecture of WGP and S-Box	Type of Fault	Injected Faults	Detected Faults	Error Coverage
w/ One-bit scheme	SEU	25,000	24,603	98.41%
	MBU		24,605	98.42%
w/Interleaved scheme	SEU	25,000	24776	99.10%
	MBU		24,994	99.98%

B. ERROR SIMULATION AND COVERAGE

The error coverage of the proposed schemes for both permanent and transient faults is evaluated using VHDL simulations. As interleaved parities help in detection of random and adjacent MBU (which are more realistic to consider for the attackers than the costly single faults) as well as a high percentage of the SBTBUs and OSBs, we leveraged this nature of interleaved parity and injected multi-bit faults for our error simulation. To simulate permanent fault injection, a faulty module is placed at the output of the non-linear S-Box and WGP, which forces the outgoing bits to either 1 or 0. As the simulation is run, the error flags of both S-Box and WGP components are monitored to determine the error coverage of the presented schemes.

The simulations were performed for 25,000 SEU (one-bit) and MBUs for both the presented schemes, in all modules of S-Boxes and WGP, the results for which are shown in Table 7. As our focus is stuck-at faults, we injected permanent stuck-at one faults for 25,000 cases (multi-bit upset) and our error coverage is 99.98 percent for interleaved parity. On the contrary, after injecting SEU and simulating for 25,000 cases, our error coverage is 98.10 percent for interleaved scheme. The high error coverage of the presented schemes makes mounting fault attacks more difficult due to a high error coverage. One can harden the comparators (i.e., error flags in our schemes), which will make them immune to faults.

C. FPGA IMPLEMENTATION

The overhead benchmarks of the proposed schemes, implemented for nonlinear S-Box and WGP of WAGE cipher, are presented in Table 8. The FPGA implementation has been performed on the device (xc7a200tfbv484-3) of Xilinx Artix-7 FPGA family. Xilinx Vivado version 2018.3 has been used for performance and implementation metrics derivations. The results of our implementations for the logic-gate based and LUT-based variants have been implemented using VHDL by modifying the RTL code[1] provided in [2]. For the LUT-based variant of our proposed schemes, the memory units (LUTs) presented in FPGAs can be utilized without any additional

[1]The RTL codes can be provided upon request by emailing any of the authors. The codes contain the formulations provided in the paper and the numbers shown Table 8 are after adding the registers and optimizations.

TABLE 8. FPGA implementation results for the nonlinear 7-bit S-Box and WGP of WAGE cipher with the proposed error detection schemes on Artix-7 FPGA Device xc7a200tfbv484-3.

WAGE Architecture		Area (*Slice*)	Power (*mW*)	Delay (*ns*)	Thr'put (*Gbps*)	Efficiency (*Mbps/Slices*)
Logic gate -based	Original	743	150	8.890	14.398	19.378
	w/ One-bit	819 (10.22%)	152 (1.32%)	11.582 (30.28%)	11.055 (23.22%)	13.498 (30.34%)
	w/ Interleaved	938 (26.24%)	152 (1.32%)	11.645 (30.99%)	10.992 (23.65%)	11.719 (39.52%)
LUT - based	Original	336	151	11.586	11.048	32.880
	w/ One-bit	384 (14.28%)	152 (0.66%)	11.863 (2.39%)	10.790 (2.34%)	28.099 (14.54%)
	w/ Interleaved	385 (14.58%)	152 (0.66%)	14.076 (21.49%)	9.093 (17.70%)	23.618 (28.17%)

circuitry, hence providing us with overheads similar to the original design implementation at the cost of slightly higher delay overhead. However, the area and delay overheads for logic implementation are higher due to combinational logic but with negligible power overhead. While our schemes are the first work in the field to detect errors on stream cipher WAGE, for qualitative comparison, let us compare the overheads by considering existing work on block ciphers. The authors in [12] propose fault detection schemes for block cipher SIMON, which incur 30.137 and 3.330 percent area and delay overhead, respectively. Moreover, the area and throughput overhead in the methods presented in [13] are 14.33 percent for the parity prediction scheme of ChaCha, a well known stream cipher. Such research work on classical cryptography proves our error detection schemes achieve acceptable overhead with more than 99 percent error coverage.

Our presented schemes can easily be implemented on other FPGA families as well as ASIC designs with similar overheads. The proposed approaches are also designed to provide VLSIerror detection through using logic gate-based implementation as they do not require costly memory units suited for ASIC implementations. From the tabulated results, it is shown that the proposed schemes provide high error coverage while incurring acceptable overhead costs making hardware implementations of WAGE more reliable.

V. CONCLUSION

This paper proposes error detection schemes for the nonlinear sub-blocks of the lightweight stream cipher WAGE for the first time. The signature based error detection schemes are derived and implemented using both logic gate-based and LUT-based variants for the nonlinear S-Box and WGP operations of WAGE. The derived one-bit signature and interleaved signature are capable of detecting both single-event upsets and multi-bit upsets, hence providing measures against both permanent and maliciously injected faults. The simulations are done in Vivado and the design is implemented using Xilinx Artix-7 FPGA family. Simulation and implementation results show that the schemes have acceptable overheads with high error coverage for resource-constrained applications. We would like to emphasize that efforts in this paper would be a step-forward towards considering error detection capabilities as a design factor.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their useful comments throughout the review process. This work was supported by the U.S. Federal Agency Award from the U.S. Department of Commerce, National Institute of Standards and Technology (NIST) under Grant 60NANB20D013.

REFERENCES

[1] M. A. Orumiehchiha, S. Rostami, E. Shakour, and J. Pieprzyk, "A differential fault attack on the WG family of stream ciphers," *J. Cryptogr. Eng.*, vol. 10, pp. 189–195, Mar. 2020.
[2] M. Aagaard, R. AlTawy, G. Gong, K. Mandal, R. Rohit, and N. Zidaric, "WAGE: An authenticated encryption algorithm," Sep. 2019. [Online] Available: https://uwaterloo.ca/communications-security-lab/lwc/wage.
[3] Y. Nawaz and G. Gong, "WG: A family of stream ciphers with designed randomness properties," *Inf. Sci.*, vol. 178, no. 7, pp. 1903–1916, Apr. 2008.
[4] G. Gong and A. M. Youssef, "Cryptographic properties of the Welch-Gong transformation sequence generators," *IEEE Trans. Inf. Theory*, vol. 48, no. 11, pp. 2837–2846, Nov. 2002.
[5] R. Altawy, R. Rohit, M. He, K. Mandal, G. Yang, and G. Gong, "sLiSCP: Simeck-based permutations for lightweight sponge cryptographic primitives," in*Proc. Sel. Areas Cryptogr.*, 2017, pp. 129–150.
[6] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "On the security of the keyed sponge construction," in *Proc. Symmetric Key Encryption Workshop*, 2011, pp. 1–15.
[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications," in *Proc. Sel. Areas Cryptogr.*, 2011, pp. 320–337.
[8] C. Dobraunig, M. Eichlseder, T. Korak, V. Lomnï¿ú, and F. Mendel, "Statistical fault attacks on nonce-based authenticated encryption schemes," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2016, pp. 369–395.
[9] A. Sarker, M. Mozaffari Kermani, and R. Azarderakhsh, "Error detection architectures for ring polynomial multiplication and modular reduction of Ring-LWE in $Z = pZ[x]/x^n + 1$ benchmarked on ASIC," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 1403–1407, Mar. 2021.
[10] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, "Security analysis of concurrent error detection against differential fault analysis," *J. Cryptogr. Eng.*, vol. 5, no. 3, pp. 153–169, Sep. 2015.
[11] A. Sarker, M. Mozaffari Kermani, and R. Azarderakhsh, "Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 738–741, Mar. 2019.
[12] S. Bauer, S. Rass, and P. Schartner, "Generic parity-based concurrent error detection for lightweight ARX ciphers," *IEEE Access*, vol. 8, pp. 142016–142025, 2020.
[13] P. Ahir, M. Mozaffari-Kermani, and R. Azarderakhsh, "Lightweight architectures for reliable and fault detection simon and speck cryptographic algorithms on FPGA," *ACM Trans. Embedded Comput. Syst.*, vol. 16, pp. 1–17, 2017.