WAGE: An Authenticated Cipher

Submission to the NIST LWC Competition

SUBMITTERS/DESIGNERS:
Mark Aagaard, Riham AlTawy, Guang Gong,
Kalikinkar Mandal*, Raghvendra Rohit, and Nusa Zidaric

*Corresponding submitter:
Email: kmandal@uwaterloo.ca
Tel: +1-519-888-4567 x45650

COMMUNICATION SECURITY LAB
Department of Electrical and Computer Engineering
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, CANADA

http://comsec.uwaterloo.ca/

March 29, 2019

Contents

Chapter 1

Introduction

WAGE is a 259-bit lightweight permutation based on the Welch-Gong (WG) stream cipher [20, 21]. It is designed to achieve an efficient hardware implementation for Authenticated Encryption with Associated Data (henceforth "AEAD"), while providing sufficient security margins. To accomplish this, the WAGE components and mode of operation are adopted from well known and analyzed cryptographic primitives. The design of WAGE, its security properties, and features are described as follows.

- WAGE **nonlinear layer:** WG permutation over $\mathbb{F}_{2^7}$ and a new 7-bit Sbox. The WG cipher, including the WG permutation, is a well-studied cryptographic primitive and has low hardware cost.
- WAGE **linear layer:** An LFSR with low hardware cost and good resistance against differential and linear cryptanalysis.
- WAGE **security:** Simple analysis and good bounds for security using automated tools such as CryptoSMT solver [15] and Gurobi [1].
- **Functionality:** Authenticated Encryption with Associated Data.
- WAGE **mode of operation:** Unified sponge duplex mode [3] that has a stronger keyed initialization and finalization phase.
- **Security claims:** Offers 128-bit security. Accepts a 128-bit key and nonce.
- **Hardware performance:** Efficient in hardware. Achieves a throughput of 606.92 Mbps at 2990 GE for 65 *nm* CMOS.
- **Microcontroller performance:** WAGE is implemented on three different microcontroller platforms, namely ATmega128, MSP430F2370, and LM3S9D96 (Cotex M3). The best throughput for the permutation is achieved on LM3S9D96, which is 286.78 Kbps

1.1 Notation

The following notation will be used throughout the document.

Notation	Description
$X \odot Y, X \oplus Y, X\|\|Y$	Bitwise AND, XOR and concatenation of X and Y
$X \otimes Y$	Finite field multiplication of X and Y
S	259 bit state of WAGE
S_j, $S_{j,k}$	stage j of state S and k-th bit of stage S_j, where $j \in \{0, \ldots, 36\}$ and $k \in \{0, \ldots, 6\}$
S_r, S_c	r-bit rate part and c-bit capacity part of S ($r = 64, c = 195$)
$\mathbb{F}_{2^7}$	Finite field $\mathbb{F}_{2^7}$
f, ω	Defining polynomial for $\mathbb{F}_{2^7}$ and its root, i.e., $f(\omega) = 0$
ℓ	LFSR feedback polynomial
WGP	Welch-Gong permutation over $\mathbb{F}_{2^7}$
SB	7-bit Sbox
rc_1^i, rc_0^i	7-bit round constants
K, N, T	key, nonce and tag
k, n, t	length of key, nonce and tag in bits ($k = n = t = 128$)
AD, M, C	associated data, plaintext and ciphertext (in blocks AD_i, M_i, C_i)
ℓ_X	length of X in words where $X \in \{AD, M, C\}$
K_j, N_j	word j of key and nonce, $j = 0, 1$
$\widehat{K}_j, \widehat{N}_j$	7-bit tuple of key and nonce, $j = 0, \ldots, 17$
WAGE-$\mathcal{AE}$	WAGE authenticated encryption scheme
WAGE-$\mathcal{E}$	WAGE encryption
WAGE-$\mathcal{D}$	WAGE decryption

1.2 Outline

The rest of the document is organized as follows. In Chapter 2, we present the complete specification of the WAGE. We summarize the security claims of our submission in Chapter 3. In Chapter 4, we present the rationale of our design by justifying the choice of each component and its respective parameters and provide the detailed security analysis in Chapter 5. The details of our hardware implementations and performance results in ASIC CMOS 65 nm and FPGA are provided in Chapter 6. In Chapter 7, we discuss the efficiency of WAGE on microcontroller implementations. Finally, we conclude with references and test vectors in Appendix A.

Chapter 2

Specification of WAGE

2.1 WAGE AEAD Algorithm

WAGE is an iterative permutation with a state size of 259 bits inspired by the initialization phase of the Welch-Gong (WG) cipher [20, 21]. It operates in a unified duplex sponge mode [3] to offer authenticated encryption with associated data (AEAD) functionality. The AEAD algorithm ($\mathsf{WAGE}\text{-}\mathcal{AE}\text{-}k$) processes an r-bit data per call of WAGE and is parameterized by the secret key size k. The AEAD algorithm $\mathsf{WAGE}\text{-}\mathcal{AE}\text{-}k$ consists of two algorithms, namely an authenticated encryption algorithm $\mathsf{WAGE}\text{-}\mathcal{E}$ and a verified decryption algorithm $\mathsf{WAGE}\text{-}\mathcal{D}$.

Encryption. The authenticated encryption algorithm $\mathsf{WAGE}\text{-}\mathcal{E}$ takes as input a secret key K of length k bits, a public message number N (nonce) of size n bits, a block header AD (a.k.a, associated data) and a message M. The output of $\mathsf{WAGE}\text{-}\mathcal{E}$ is an authenticated ciphertext C of the same length as M, and an authentication tag T of size t bits. Mathematically, $\mathsf{WAGE}\text{-}\mathcal{E}$ is defined as

$$\mathsf{WAGE}\text{-}\mathcal{E} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^t$$

with

$$\mathsf{WAGE}\text{-}\mathcal{E}(K, N, AD, M) = (C, T).$$

Decryption. The decryption and verification algorithm takes as input the secret key K, nonce N, associated data AD, ciphertext C and tag T, and outputs the plaintext M of same length as C if the verification of tag is correct or $\perp$ if the tag verification fails. More formally,

$$\mathsf{WAGE}\text{-}\mathcal{D} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{\{0,1\}^* \cup \perp\}$$

where

$$\mathsf{WAGE}\text{-}\mathcal{D}(K, N, AD, C, T) \in \{M, \perp\}.$$

2.2 Recommended Parameter Set

In Table 2.2, we list the recommended parameter set for WAGE-$\mathcal{AE}$-128. The length of each parameter is given in bits and d denotes the amount of allowed data (including both AD and M) before a re-keying is required.

Table 2.1: Recommended parameter set for WAGE-$\mathcal{AE}$-128

Functionality	Algorithm	r	k	n	t	$\log_2(d)$
AEAD	WAGE-$\mathcal{AE}$-128	64	128	128	128	64

2.3 Description of the **WAGE** Permutation

WAGE is an iterative permutation and its round function is constructed by tweaking the initialization phase of the WG cipher over $\mathbb{F}_{2^7}$ where an additional Welch-Gong permutation (WGP) and four 7-bit Sboxes (SB) are added to achieve faster confusion and diffusion. We opt for a design based on a combination of an LFSR with WGP and SB, which provides a good trade-off between security and hardware efficiency. The core components of the round function are an LFSR, two WGPs and four SBs, which are described below in detail.

2.3.1 Underlying finite field

WAGE operates over the finite field $\mathbb{F}_{2^7}$, defined using the primitive polynomial $f(x) = x^7 + x^3 + x^2 + x + 1$. Elements of the finite field $\mathbb{F}_{2^7}$ are represented using the polynomial basis $\mathsf{PB} = \{1, \omega, \ldots, \omega^6\}$, and an element $a \in \mathbb{F}_{2^7}$ is given by

$$a = \sum_{i=0}^{6} a_i \omega^i, a_i \in \mathbb{F}_2$$

and its vector representation is

$$[a]_{\mathsf{PB}} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6).$$

To represent a 7-bit finite field element as a byte, a 0 is appended on the left. For unambiguity, we include the conversion to binary as an intermediate step:

$$[a]_{\mathsf{PB}} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6) \to [a]_b = (0, a_0, a_1, a_2, a_3, a_4, a_5, a_6) \to [a]_{hex} = (h_1, h_0)$$

Table 2.2 shows some examples of the conversion to HEX:

2.3.2 The LFSR

The internal state S of the permutation is composed of 37 stages and given by $S = (S_{36}, \cdots, S_1, S_0)$, where each S_j holds an element from the finite field $\mathbb{F}_{2^7}$

Table 2.2: Examples of conversion of the field elements to HEX

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	16^1	16^0
$a \in \mathbb{F}_{2^7}$	0	a_0	a_1	a_2	a_3	a_4	a_5	a_6	h_1	h_0
1	0	1	0	0	0	0	0	0	4	0
ω	0	0	1	0	0	0	0	0	2	0
$1 + \omega$	0	1	1	0	0	0	0	0	6	0
$1 + \omega^6$	0	1	0	0	0	0	0	1	4	1

represented using the PB, i.e., $S_j = (S_{j,0}, S_{j,1}, S_{j,2}, S_{j,3}, S_{j,4}, S_{j,5}, S_{j,6})$. The WAGE LFSR is defined by the feedback polynomial

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

which is primitive over $\mathbb{F}_{2^7}$. The linear feedback fb is computed as follows:

$$fb = S_{31} \oplus S_{30} \oplus S_{26} \oplus S_{24} \oplus S_{19} \oplus S_{13} \oplus S_{12} \oplus S_{8} \oplus S_{6} \oplus (\omega \otimes S_0).$$

2.3.3 The nonlinear components

In this subsection, we provide the details of the WGP and SB.

The Welch-Gong Permutation (WGP). The cryptographic properties of the WG permutation and transformation have been widely investigated in the literature [13]. We use a decimated WGP with low differential uniformity and high nonlinearity. Using the decimation $d = 13$, the differential uniformity for WGP is 6, and its nonlinearity is 42. The WGP7 over $\mathbb{F}_{2^7}$ is defined as

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

A decimated WG permutation with decimation d such that $\gcd(d, 2^m - 1) = 1$ is defined as

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

We use the decimation $d = 13$ and denote it by $\mathsf{WGP}(x) = \mathsf{WGP7}(x^{13})$. The maximum algebraic degree of its components is 6. An Sbox representation of WGP is given in Table 2.3 in a row-major order. The 7-bit finite field elements are represented in hex using the technique provided in Table 2.2.

SBox (SB). We construct a lightweight 7-bit Sbox in an iterative way. Let the input be $x = (x_0, x_1, x_2, x_3, x_4, x_5, x_6)$. The nonlinear transformation Q is given by

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

The bit permutation P is given by

$$P(x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (x_6, x_3, x_0, x_4, x_2, x_5, x_1).$$

Table 2.3: Hex representation of WGP

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

One-round R of the Sbox SB is obtained by composing the nonlinear transformation Q and the bit permutation P, and is given by $R = Q \circ P$ where

$$R(x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (x_6, \overline{x}_3 \oplus (x_5 \wedge x_6), x_0 \oplus (x_2 \wedge x_3), x_4, x_2, \overline{x}_5 \oplus (x_2 \wedge x_4), x_1).$$

The 7-bit Sbox SB is constructed by iterating the function R 5 times, followed by applying Q once, and then complementing the 0th and 2nd components. Mathematically,

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

SB has a differential uniformity of 8 and a nonlinearity of 44. The maximum algebraic degree of its components is 6.

Although SB is defined bit-wise, the interpretation of the 7 bits is identical to the interpretation of the coefficients of the finite field element represented in polynomial basis. The hex representation of SB is provided in Table 2.4 and the conversion to hex is the same as that of WGP.

Table 2.4: Hex representation of SB

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

2.3.4 Description of the core permutation

The WAGE permutation is a 259-bit permutation consisting of a 37-stage NLFSR defined over $\mathbb{F}_{2^7}$. It is based on the initialization phase of the WG cipher and utilizes

5 additional Sboxes to update the internal state. At the i-th iteration, the internal state is denoted by $S^i = (S^i_{36}, S^i_{35}, \cdots, S^i_1, S^i_0)$. The round function that updates 6 stages of the register nonlinearly is viewed as

- Updating with initialization of the WG cipher:

  $$S^{i+1}_{36} \leftarrow \mathsf{WGP}(S^i_{36}) \oplus S^i_{31} \oplus S^i_{30} \oplus S^i_{26} \oplus S^i_{24} \oplus S^i_{19} \oplus S^i_{13} \oplus S^i_{12} \oplus S^i_{8} \oplus S^i_{6} \oplus (\omega \otimes S^i_0)$$

- Updating one stage with WGP:

  $$S^{i+1}_{18} \leftarrow S^i_{19} \oplus \mathsf{WGP}(S^i_{18})$$

- Updating four stages with SB:

  $$S^{i+1}_{4} \leftarrow S^i_{5} \ \oplus \mathsf{SB}(S^i_{8})$$

  $$S^{i+1}_{10} \leftarrow S^i_{11} \oplus \mathsf{SB}(S^i_{15})$$

  $$S^{i+1}_{23} \leftarrow S^i_{24} \oplus \mathsf{SB}(S^i_{8})$$

  $$S^{i+1}_{29} \leftarrow S^i_{30} \oplus \mathsf{SB}(S^i_{15}).$$

A schematic diagram of the round function is presented in Figure 2.1. A pair of 7-bit round constants (rc_1, rc_0) is XORed with the pair of stages $(36, 18)$ to destroy similarity among state updates. On an input S^0, an output of the permutation is obtained by applying the round function, denoted by WAGE-StateUpdate, 111 times, i.e., $S^{111} \leftarrow \mathsf{WAGE\text{-}StateUpdate}^{111}(S^0)$. An algorithmic description of WAGE is provided in Algorithm 1.

Algorithm 1 WAGE permutation

```
Input : $S^0 = (S^0_{36}, S^0_{35}, \cdots, S^0_1, S^0_0)$
Output : $S^{111} = (S^{111}_{36}, S^{111}_{35}, \cdots, S^{111}_1, S^{111}_0)$

for $i = 0$ to 110 do:
        $S^{i+1} \leftarrow \mathsf{WAGE\text{-}StateUpdate}(S^i, rc^i_1, rc^i_0)$
return $S^{111}$

Function WAGE-StateUpdate($S^i$):
        $fb = S^i_{31} \oplus S^i_{30} \oplus S^i_{26} \oplus S^i_{24} \oplus S^i_{19} \oplus S^i_{13} \oplus S^i_{12} \oplus S^i_{8} \oplus S^i_{6} \oplus (\omega \otimes S^i_0)$
        $S^{i+1}_{4} \leftarrow S^i_{5} \oplus \mathsf{SB}(S^i_{8})$
        $S^{i+1}_{10} \leftarrow S^i_{11} \oplus \mathsf{SB}(S^i_{15})$
        $S^{i+1}_{18} \leftarrow S^i_{19} \oplus \mathsf{WGP}(S^i_{18}) \oplus rc^i_0$
        $S^{i+1}_{23} \leftarrow S^i_{24} \oplus \mathsf{SB}(S^i_{8})$
        $S^{i+1}_{29} \leftarrow S^i_{30} \oplus \mathsf{SB}(S^i_{15})$
        $S^{i+1}_{36} \leftarrow fb \oplus \mathsf{WGP}(S^i_{36}) \oplus rc^i_1$
        $S^{i+1}_{j} \leftarrow S^i_{j+1}, j \in \{0, \cdots, 36\} \backslash \{4, 10, 18, 23, 29, 36\}$
        return $S^{i+1}$
```

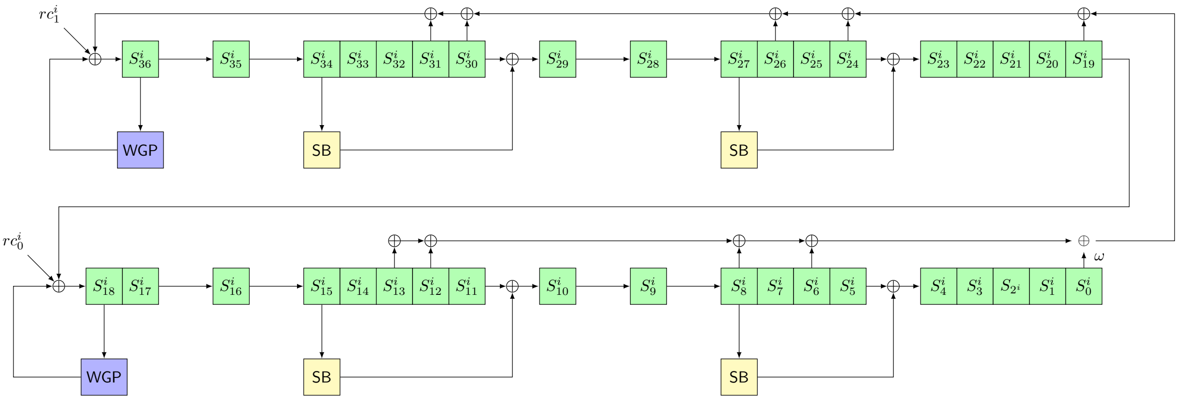


Figure 2.1: The i-th round of the WAGE permutation

2.3.5 Round constants

We use two 7-bit round constants at each round of WAGE. The round constants are listed in Table 2.5. The interpretation of the hex values of round constants in terms of polynomial basis is the same as for SB, and hence details are omitted.

Table 2.5: Round constants of WAGE

Round i	**Round constant** (rc_1^i, rc_0^i)									
0 - 9	(3f, 7f)	(0f, 1f)	(03, 07)	(40, 01)	(10, 20)	(04, 08)	(41, 02)	(30, 60)	(0c, 18)	(43, 06)
10 - 19	(50, 21)	(14, 28)	(45, 0a)	(71, 62)	(3c, 78)	(4f, 1e)	(13, 27)	(44, 09)	(51, 22)	(34, 68)
20 - 29	(4d, 1a)	(66, 73)	(5c, 39)	(57, 2e)	(15, 2b)	(65, 4a)	(79, 72)	(3e, 7c)	(2f, 5f)	(0b, 17)
30 - 39	(42, 05)	(70, 61)	(1c, 38)	(47, 0e)	(11, 23)	(24, 48)	(49, 12)	(32, 64)	(6c, 59)	(5b, 36)
40 - 49	(56, 2d)	(35, 6b)	(6d, 5a)	(7b, 76)	(5e, 3d)	(37, 6f)	(0d, 1b)	(63, 46)	(58, 31)	(16, 2c)
50 - 59	(25, 4b)	(69, 52)	(74, 3a)	(6e, 5d)	(3b, 77)	(4e, 1d)	(33, 67)	(4c, 19)	(53, 26)	(54, 29)
60 - 69	(55, 2a)	(75, 6a)	(7d, 7a)	(7f, 7e)	(1f, 3f)	(07, 0f)	(01, 03)	(20, 40)	(08, 10)	(02, 04)
70 - 79	(60, 41)	(18, 30)	(06, 0c)	(21, 43)	(28, 50)	(0a, 14)	(62, 45)	(78, 71)	(1e, 3c)	(27, 4f)
80 - 89	(09, 13)	(22, 44)	(68, 51)	(1a, 34)	(66, 4d)	(39, 73)	(2e, 5c)	(2b, 57)	(4a, 15)	(72, 65)
90 - 99	(7c, 79)	(5f, 3e)	(17, 2f)	(05, 0b)	(61, 42)	(38, 70)	(0e, 1c)	(23, 47)	(48, 11)	(12, 24)
100 - 109	(64, 49)	(59, 32)	(36, 6c)	(2d, 5b)	(6b, 56)	(5a, 35)	(76, 6d)	(3d, 7b)	(6f, 5e)	(1b, 37)
110	(46, 0d)									

2.4 WAGE-$\mathcal{AE}$-128Algorithm

WAGE uses the unified sponge duplex mode to provide the AEAD functionality [3]. A WAGE instance is parametrized by a key of length k, denoted as WAGE-$\mathcal{AE}$-k. Algorithm 2 presents a high-level overview of WAGE-$\mathcal{AE}$-128. The encryption (WAGE-$\mathcal{E}$) and decryption (WAGE-$\mathcal{D}$) WAGE-$\mathcal{AE}$-128 are shown in Figure 2.2. In what follows, we first illustrate the rate and the capacity part of the state, and then the padding rule. We then describe each phase of WAGE-$\mathcal{E}$ and WAGE-$\mathcal{D}$.

2.4.1 Rate and capacity part of state

The internal state S of WAGE is divided into two parts, namely the rate part S_r and the capacity part S_c. The 0-th bit of stage S_{36}, i.e., $S_{36,0}$, and all bits of stages $S_{35}, S_{34}, S_{28}, S_{27}$, $S_{18}, S_{16}, S_{15}, S_9$ and S_8 constitute S_r (shaded orange in Figure 2.3), while all remaining bits in the state constitute S_c. The rationale for the choice of

Algorithm 2 WAGE-$\mathcal{AE}$-128 algorithm

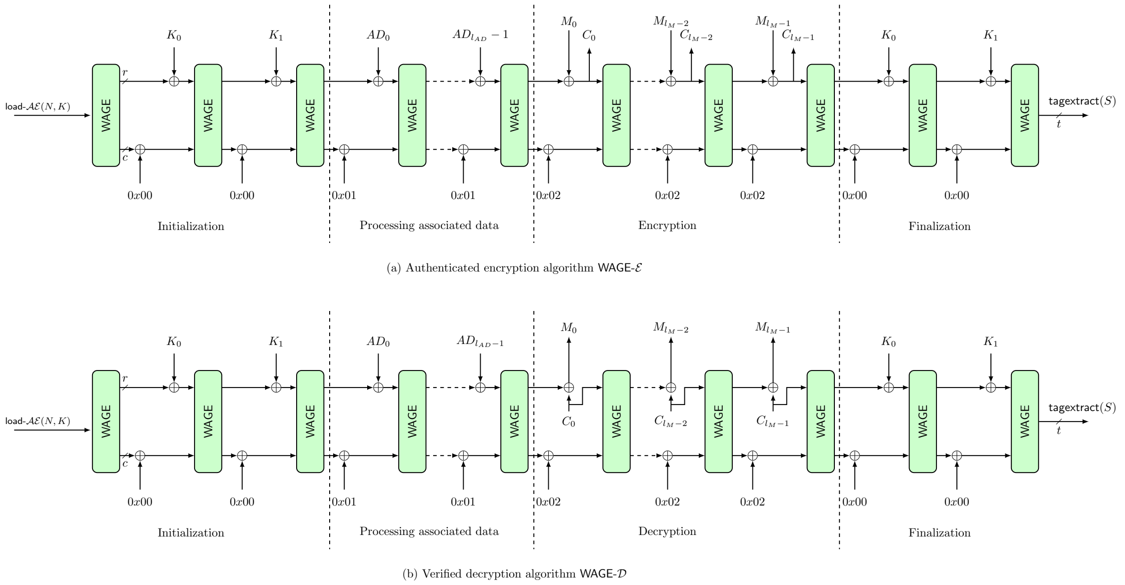
```
Authenticated encryption WAGE-$\mathcal{E}(K, N, AD, M)$:
    $S \leftarrow$ Initialization$(N, K)$
    if $|AD| \neq 0$ then:
        $S \leftarrow$ Processing-Associated-Data$(S, AD)$
    $(S, C) \leftarrow$ Encyption$(S, M)$
    $T \leftarrow$ Finalization$(S, K)$
    return $(C, T)$

Initialization$(N, K)$:
    $S \leftarrow$ load-$\mathcal{AE}(N, K)$
    $S \leftarrow$ WAGE$(S)$
    for $i = 0$ to $1$ do:
        $S \leftarrow (S_r \oplus K_i, S_c)$
        $S \leftarrow$ WAGE$(S)$
    return $S$

Processing-Associated-Data$(S, AD)$:
    $(AD_0||\cdots||AD_{\ell_{AD}-1}) \leftarrow$ pad$_\mathsf{r}(AD)$
    for $i = 0$ to $\ell_{AD} - 1$ do:
        $S \leftarrow (S_r \oplus AD_i, S_c \oplus 0^{c-7}||1||0^6)$
        $S \leftarrow$ WAGE$(S)$
    return $S$

Encryption$(S, M)$:
    $(M_0||\cdots||M_{\ell_M-1}) \leftarrow$ pad$_\mathsf{r}(M)$
    for $i = 0$ to $\ell_M - 1$ do:
        $C_i \leftarrow M_i \oplus S_r$
        $S \leftarrow (C_i, S_c \oplus 0^{c-7}||0||1||0^5)$
        $S \leftarrow$ WAGE$(S)$
    $C_{\ell_M-1} \leftarrow$ trunc-msb$(C_{\ell_M-1}, |M| \bmod r)$
    $C \leftarrow (C_0, C_1, \ldots, C_{\ell_M-1})$
    return $(S, C)$

pad$_\mathsf{r}(X)$:
    $X \leftarrow X||10^{r-1-(|X| \bmod r)}$
    return $X$

trunc-lsb$(X, n)$:
    return $(x_{r-n}, x_{r-n+1}, \ldots, x_{r-1})$

Verified decryption WAGE-$\mathcal{D}(K, N, AD, C, T)$:
    $S \leftarrow$ Initialization$(N, K)$
    if $|AD| \neq 0$ then:
        $S \leftarrow$ Processing-Associated-Data$(S, AD)$
    $(S, M) \leftarrow$ Decyption$(S, C)$
    $T' \leftarrow$ Finalization$(S, K)$
    if $T' \neq T$ then:
        return $\perp$
    else:
        return $M$

Decryption$(S, C)$:
    $(C_0||\cdots||C_{\ell_C-1}) \leftarrow$ pad$_\mathsf{r}(C)$
    for $i = 0$ to $\ell_C - 2$ do:
        $M_i \leftarrow C_i \oplus S_r$
        $S \leftarrow (C_i, S_c \oplus 0^{c-7}||0||1||0^5)$
        $S \leftarrow$ WAGE$(S)$
    $M_{\ell_C-1} \leftarrow S_r \oplus C_{\ell_C-1}$
    $C_{\ell_C-1} \leftarrow$ trunc-msb$(C_{\ell_C-1}, |C| \bmod r)||$trunc-lsb$(M_{\ell_C-1}, r - |C| \bmod r))$
    $M_{\ell_C-1} \leftarrow$ trunc-msb$(M_{\ell_C-1}, |C| \bmod r)$
    $M \leftarrow (M_0, M_1, \ldots, M_{\ell_C-1})$
    $S \leftarrow$ WAGE$(C_{\ell_C-1}, S_c \oplus 0^{c-7}||0||1||0^5)$
    return $(S, M)$

Finalization$(S, K)$:
    for $i = 0$ to $1$ do:
        $S \leftarrow$ WAGE$(S_r \oplus K_i, S_c)$
    $T \leftarrow$ tagextract$(S)$
    return $T$

trunc-msb$(X, n)$:
    if $n = 0$ then:
        return $\phi$
    else:
        return $(x_0, x_1, \ldots, x_{n-1})$
```

the S_r positions is explained in Section 4.7. The rate part S_r of the state is used for both absorbing and squeezing.

For example, the 64-bit bits of a message block are absorbed into the S_r as follows:

$$\begin{array}{llll@{\qquad}llll}
S_{36} & \leftarrow & (m_{63}, 0, \ldots, 0) & \backsim D_9 & S_{18} & \leftarrow & (m_{28}, \ldots, m_{34}) & \backsim D_4 \\
S_{35} & \leftarrow & (m_{56}, \ldots, m_{62}) & \backsim D_8 & S_{16} & \leftarrow & (m_{21}, \ldots, m_{27}) & \backsim D_3 \\
S_{34} & \leftarrow & (m_{49}, \ldots, m_{55}) & \backsim D_7 & S_{15} & \leftarrow & (m_{14}, \ldots, m_{20}) & \backsim D_2 \\
S_{28} & \leftarrow & (m_{42}, \ldots, m_{48}) & \backsim D_6 & S_{9} & \leftarrow & (m_{7}, \ldots, m_{13}) & \backsim D_1 \\
S_{27} & \leftarrow & (m_{35}, \ldots, m_{41}) & \backsim D_5 & S_{8} & \leftarrow & (m_{0}, \ldots, m_{6}) & \backsim D_0
\end{array}$$

The tuples above are labeled with D_k, $k = 0, \ldots, 9$, that are used as data inputs to S_r; they carry the associated data bits, the message bits during encryption, the ciphertext bits during decryption, and the key bits during the initialization and



(a) Authenticated encryption algorithm WAGE-$\mathcal{E}$

(b) Verified decryption algorithm WAGE-$\mathcal{D}$

Figure 2.2: Schematic diagram of the WAGE-$\mathcal{AE}$-128 algorithm

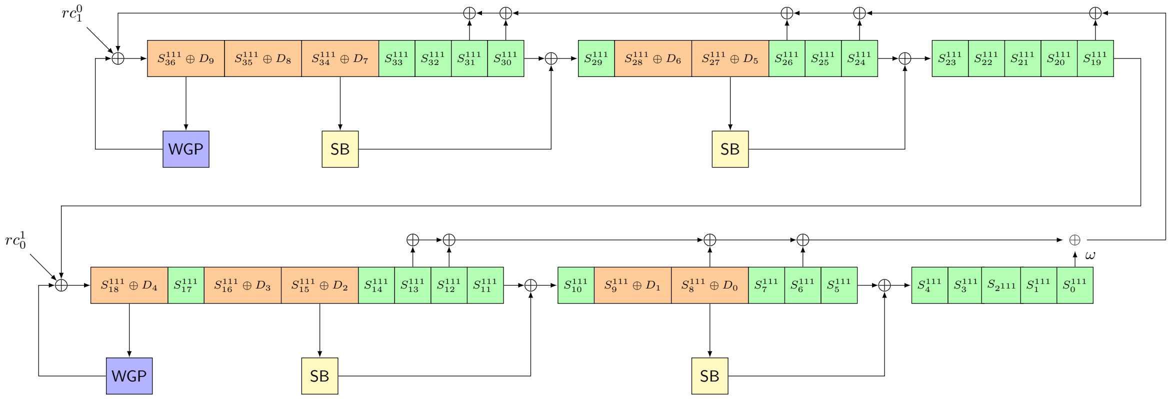


Figure 2.3: Schematic diagram of absorbing and squeezing phase of WAGE-$\mathcal{AE}$-128.

finalization phases. Figure 2.3 shows the D_k XORed to the appropriate stages S_j^{111}, $j \in \{36, 35, 34, 28, 27, 18, 16, 15, 9, 8\}$, shaded orange. The two domain separator bits ds_1 and ds_0 are XORed to the first two bits of S_c, namely $S_{0,1}^{111}$ and $S_{0,0}^{111}$ respectively.

2.4.2 Padding

Padding is necessary when the length of the processed data is not a multiple of the rate r value. Since the key size is a multiple of r, we get two key blocks K_0 and K_1, so no padding is needed. Afterwards, the padding rule (10*), denoting a single 1 followed by required number of 0's, is applied to the message M so that its length after padding is a multiple of r. The resulting padded message is divided into ℓ_M r-bit blocks $M_0\| \cdots \|M_{\ell_M-1}$. A similar procedure is carried out on the associated data AD which results in ℓ_{AD} r-bit blocks $AD_0\| \cdots \|AD_{\ell_{AD}-1}$. In the case where no associated data is present, no processing is necessary. We summarize the padding rules for the message and associated data below.

$$\begin{aligned}
\mathsf{pad_r}(M) &\leftarrow M\|1\|0^{r-1-(|M| \bmod r)} \\
\mathsf{pad_r}(AD) &\leftarrow \begin{cases} AD\|1\|0^{r-1-(|AD| \bmod r)} & \text{if } |AD|>0 \\ \phi & \text{if } |AD|=0 \end{cases}
\end{aligned}$$

Note that in case of AD or M whose length is a multiple of r, an additional r-bit padded block is appended to AD or M to distinguish between the processing of partial and complete blocks.

2.4.3 Loading key and nonce

The state is loaded with a 128-bit nonce $N = (n_0, \dots, n_{127})$ and 128-bit key $K = (k_0, \dots, k_{127})$. The remaining three bits of S are set to zero. Both the nonce and the key are divided into 7-bit tuples as follows:

- for $0 \le i \le 8$, $\widehat{N}_i = (n_{7i}, \dots, n_{7i+6})$ and $\widehat{K}_i = (k_{7i}, \dots, k_{7i+6})$
- for $9 \le i \le 17$, $\widehat{N}_i = (n_{7i+1}, \dots, n_{7i+7})$ and $\widehat{K}_i = (k_{7i+1}, \dots, k_{7i+7})$
- $\widehat{K}^*_{18} = (k_{63}, k_{127}, n_{63}, n_{127}, 0, 0, 0)$

The state S is initialized as follows:

$$\begin{aligned}
S_{36},S_{35},S_{34},S_{33},S_{32},S_{31},S_{30},S_{29},S_{28} &\leftarrow \widehat{N}_{16},\widehat{N}_{14},\widehat{N}_{12},\widehat{N}_{10},\widehat{N}_{8},\widehat{N}_{6},\widehat{N}_{4},\widehat{N}_{2},\widehat{N}_{0} \\
S_{27},S_{26},S_{25},S_{24},S_{23},S_{22},S_{21},S_{20},S_{19} &\leftarrow \widehat{K}_{17},\widehat{K}_{15},\widehat{K}_{13},\widehat{K}_{11},\hat{K}_{9},\widehat{K}_{7},\widehat{K}_{5},\widehat{K}_{3},\widehat{K}_{1} \\
S_{18},S_{17} &\leftarrow \widehat{K}^*_{18},\widehat{N}_{15} \\
S_{16},S_{15},S_{14},S_{13},S_{12},S_{11},S_{10},S_{9} &\leftarrow \widehat{N}_{17},\widehat{N}_{13},\widehat{N}_{11},\widehat{N}_{9}\ ,\widehat{N}_{7},\widehat{N}_{5},\widehat{N}_{3},\widehat{N}_{1} \\
S_8,\ S_7,\ S_6,\ S_5,\ S_4,\ S_3,\ S_2,\ S_1,\ S_0 &\leftarrow \widehat{K}_{16},\widehat{K}_{14},\widehat{K}_{12},\widehat{K}_{10},\widehat{K}_{8},\widehat{K}_{6},\widehat{K}_{4},\widehat{K}_{2},\widehat{K}_{0}
\end{aligned}$$

This loading scheme is further discussed in Section 4.7. We use $\mathsf{load}\text{-}\mathcal{AE}(N, K)$ to denote the process of loading the state with nonce N and key K in the positions described above.

2.4.4 Initialization

The goal of this phase is to initialize the state S with the public nonce N and the key K. The state is first loaded using $\mathsf{load}\text{-}\mathcal{AE}(N, K)$ as described above, and then the two key blocks K_0 and K_1, with $K = K_0||K_1$, are absorbed into the state, with the WAGE permutation applied each time. The steps of the initialization are described as follows.

$$\begin{aligned}
S &\leftarrow \mathsf{WAGE}(\mathsf{load}\text{-}\mathcal{AE}(N, K)) \\
S &\leftarrow \mathsf{WAGE}(S_r \oplus K_0, S_c) \\
S &\leftarrow \mathsf{WAGE}(S_r \oplus K_1, S_c)
\end{aligned}$$

2.4.5 Processing associated data

If there is associated data, then for each absorbed block of AD, a domain separator bit is XORed to the current value of $S_{0,0}$. Then the WAGE permutation is applied to the whole state. This phase is defined in Algorithm 2.

$$S \leftarrow \mathsf{WAGE}(S_r \oplus AD_i, S_c \oplus 0^{c-7}||1||0^6),\ i = 0, \ldots, \ell_{AD} - 1$$

2.4.6 Encryption

This phase is similar to the processing of associated data, however, the domain separator bit is XORed to the current value of $S_{0,1}$. In addition, each message block M_i, $i = 0, \ldots, \ell_M - 1$, is XORed to S_r part of the internal state as described in Section 2.4.1, which gives the corresponding ciphertext block C_i, which is extracted from the S_r part of the state as well. After that, the WAGE permutation is applied to the internal state S.

$$\begin{array}{rl} C_i & \leftarrow S_r \oplus M_i, \\ S & \leftarrow \mathsf{WAGE}(C_i, S_c \oplus 0^{c-7}||0||1||0^5),\ i = 0, \cdots, \ell_M - 1 \end{array}$$

To minimize communication overhead, the last ciphertext block is truncated so that its length is equal to that of the last unpadded message block. The details of this phase are given in Algorithm 2.

2.4.7 Finalization

After the extraction of the last ciphertext block, the domain separator is reset to zero. First, the two 64-bit key blocks $K = K_0||K_1$ are absorbed into the state, with the WAGE permutation applied each time. Then, the tag is extracted from the S positions used for loading the nonce during load-$\mathcal{AE}(N, K)$. The finalization steps are mentioned below and illustrated in Algorithm 2.

$$\begin{array}{rl} S & \leftarrow \mathsf{WAGE}((S_r \oplus K_i), S_c),\ i = 0, 1 \\ T & \leftarrow \mathsf{tagextract}(S). \end{array}$$

The function tagextract(S) extracts the 128-bit tag $T = \widehat{T}_0||\widehat{T}_1|| \ldots ||\widehat{T}_{17}||\widehat{T}^*_{18}$ from the S positions that were used to load the 7-bit tuples of the nonce N during load-$\mathcal{AE}(N, K)$, namely stages S_{36} ,..., S_{28} and $S_{18} \ldots S_9$. The 7-bit $\widehat{T}_i$ tuples are given by:

$$\begin{array}{rcl} \widehat{T}_{16},\widehat{T}_{14},\widehat{T}_{12},\widehat{T}_{10},\widehat{T}_8,\widehat{T}_6,\widehat{T}_4, \widehat{T}_2, \widehat{T}_0 & \leftarrow & S_{36},S_{35},S_{34},S_{33},S_{32},S_{31},S_{30},S_{29},S_{28} \\ \widehat{T}_{15},\widehat{T}_{13},\widehat{T}_{11},\widehat{T}_9,\widehat{T}_7,\widehat{T}_5, \widehat{T}_3, \widehat{T}_1 & \leftarrow & S_{16}, S_{15},S_{14},S_{13},S_{12},S_{11},S_{10},S_9 \\ \widehat{T}^*_{18},\widehat{T}_{17} & \leftarrow & S_{18},S_{17} \end{array}$$

where

$$\begin{array}{l} \widehat{T}_i = (t_{7i}, \ldots, t_{7i+6}),\ \text{for } 0 \leq i \leq 17,\ \text{and} \\ \widehat{T}^*_{18} = (-, -, t_{126}, t_{127}, -, -, -). \end{array}$$

Note that for $\widehat{T}^*_{18}$, only the second two bits of stage S_{18} are used, the remaining stage bits are discarded, as indicated by the $-$ sign.

2.4.8 Decryption

The decryption procedure is symmetrical to encryption and illustrated in Algorithm 2.

Chapter 3

Security Claims

WAGE is designed to provide authenticated encryption with associated data functionality. We assume a nonce-respecting adversary and do not claim security in the event of nonce reuse. If the verification procedure fails, the decrypted ciphertext and the new tag should not be given as output. Moreover, we do not claim security for the reduced-round versions of WAGE-$\mathcal{AE}$-128. The security claims of WAGE-$\mathcal{AE}$-128 are summarized in Table 3.1. Note that the security for integrity in Table 3.1 includes the integrity of plaintext, associated data and nonce.

Table 3.1: Security claims of WAGE-$\mathcal{AE}$-128 (in bits)

Confidentiality	**Integrity**	**Authenticity**	**Data limit**
128	128	128	64

Chapter 4

Design Rationale

W$\mathcal{AG}\mathcal{E}$ is a hardware-oriented $\mathcal{AE}$ scheme. Our design philosophy for the WAGE permutation is to reuse and adopt the initialization phase of the well-studied WG cipher. More specifically, we use the initialization phase of the WG cipher over $\mathbb{F}_{2^7}$. Feedback shift registers (FSR) are widely used as basic building blocks in many cryptographic designs, due to their simple architecture and efficient implementations. We choose a design for a lightweight permutation based on word-oriented shift registers and substitution boxes (Sboxes).

Our parameter selection was aimed at reducing the hardware implementation cost. First, we exhaustively collected pre place-and-route (pre-PAR) synthesis results for the CMOS 65 nm area of the WGP for $\mathbb{F}_{2^m}$, $m \in \{5, 7, 8, 10, 11, 13, 14, 16\}$, and all polynomial bases, to find the balance between security and hardware implementation area. Once the field was set, we searched for the Sboxes based on their hardware cost, differential uniformity and nonlinearity, and exhaustively searched for symmetric feedback polynomials with a low number of nonzero terms, and with good security properties.

4.1 Mode of Operation

WAGE adopts the sLiSCP sponge mode [3] as its mode of operation. The adopted mode is a slight variation of well the analyzed traditional sponge duplex mode [4] and offers the following features.

- Provable security bounds when instantiated with an ideal permutation [5, 14].
- No key scheduling is required.
- Inverse free as the permutation is always evaluated in the forward direction.
- Encryption and decryption functionalities are identical and can be implemented with the same hardware circuit (only r-bit MUXs are required to replace the rate part of state).
- The length of processed data is not required beforehand.
- Strong keyed initialization and finalization phases, where the key is absorbed in the state using the XORs of the rate part. This ensures that key recovery

is hard, even if the internal state is recovered. Universal forgery with the knowledge of the internal state is not practical.

- Domain separators are used for each processed data block and they are changed with each new phase, rather than with last data block in the previous phase. This leads to a more efficient hardware implementation. This method was shown to be secure in [14].

4.2 WAGE State Size

Our main aim was to choose b (state size) that provides 128-bit AE security. For a b-bit permutation with $b = r + c$ (r-bit rate and c-bit capacity), operating in sponge duplex mode, the best known bound is $\min\{2^{b/2}, 2^c, 2^k\}$ [14]. This implies that for $k = 128$, $b \geq 256$. In section 4.4.1 we choose the operating finite field as $\mathbb{F}_{2^7}$ and accordingly $b = 259$. The values of $r = 64$ and $c = 195$ are chosen to have an efficient and low-cost hardware implementation. Our choice of (b, r, c) satisfies the NIST-LWC requirements [19] and 2^{64} bits of data can be processed per key.

4.3 Choice of Linear Layer

The linear layer of WAGE is composed of 1) $\mathcal{L}_1$: a feedback polynomial of degree 37, which is primitive over $\mathbb{F}_{2^7}$ and 2) $\mathcal{L}_2$: input and output tap positions of WGP and SB Sboxes. There exist many choices for $\mathcal{L}_1$ and $\mathcal{L}_2$, which results in a tradeoff between security and efficient implementations. Thus, we restrict our search to ones which are lightweight and offer good security bounds. Note that we can not have only $\mathcal{L}_1$ or only $\mathcal{L}_2$ as the linear layer, because that would result in slower diffusion. The required criteria for $\mathcal{L}_1$ and $\mathcal{L}_2$ are:

1. To have a lighter $\mathcal{L}_1$ we look for a feedback polynomial of the form

$$\ell(y) = y^{37} + \sum_{j=1}^{36} c_j y^j + \omega, \qquad c_j \in \mathbb{F}_2,$$

   where ω is the root of the chosen field polynomial $f(x)$, which is also a primitive element of $\mathbb{F}_{2^7}$. Including ω, we chose feedback polynomials with 10 nonzero tap positions ($c_j = 1$) that are symmetric and need only 70 XOR gates to implement in hardware. In order to allow hardware optimizations in the future, e.g.parallelization, we prefer polynomials that minimize the position j of the biggest non-zero c_j. This pushes the taps as far to the right as possible, therefore we fixed the highest coefficients to zero.

2. A combination of $\mathcal{L}_1$ and $\mathcal{L}_2$ for which computing the minimum number of active Sboxes is feasible and enable us to provide bounds for differential/linear distinguishers.

We found 23 symmetric polynomials with 10 non-zero taps (Table 5.1 in Section 5)[12]. The first column shows the candidate polynomials listed with their nonzero coefficients c_j. We chose the one that provides the maximum resistance against cryptanalytic attacks, such as differential and linear attacks. More precisely, we have:

$$\mathcal{L}_1 : y^{37} + y^{31} + y^{30} + y^{26} + y^{24} + y^{19} + y^{13} + y^{12} + y^{8} + y^{6} + \omega,$$
$$\mathcal{L}_2 : \{(36,36),(34,30),(27,24),(18,19),(15,11),(8,5)\},$$

where $(a,b) \in \mathcal{L}_2$ denotes the (input, output) position of an Sbox (Figure 2.1).

4.4 Nonlinear Layer of WAGE

We now justify the choices for the components in the nonlinear layer of the WAGE permutation. The nonlinear layer consists of two WGPs and four sboxes SB, specified in Section 2.3.3. The number of WGPs and sboxes was chosen to achieve faster confusion and diffusion.

4.4.1 The Welch-Gong permutation (WGP)

The natural choice of the finite field for low-cost hardware, while maintaining ease of software implementations, is $\mathbb{F}_{2^8}$. However, the pre-PAR hardware area for the WGP over $\mathbb{F}_{2^8}$, averaged over all irreducible polynomials, is 546 GE, which is bigger than two $\mathbb{F}_{2^7}$ WGP hardware modules, hence we choose the finite field $\mathbb{F}_{2^7}$ for WAGE.

The polynomial basis $\mathsf{PB}_i = \{1, \omega_i, \ldots, \omega_i^6\}$ was chosen for the representation of the field elements, where ω_i is a root of the defining polynomial $f_i(x)$, i.e. $f_i(\omega_i) = 0$. The polynomial $f_i(x)$ was chosen to minimize the hardware implementation area of WGP with a decimation exponent 13 and of multiplication with the constant term of the LFSR feedback polynomial. As we use the polynomial basis, the smallest area constant term is ω_i. To estimate the area of the constant term multiplier, we used the Hamming weight of the matrix for multiplication by ω_i w.r.t. to the basis PB_i. The pre-PAR results for CMOS 65 nm implementations of the WGP modules and the constant terms are listed in Table 4.1: they show 18 primitive polynomials of degree 7, denoted $f_i(x)$. Each of the f_i has a different root ω_i, which in turn gives a different PB_i. Thus, the implementation results change with the field defining polynomial. The smallest area for WGP and constant term multiplier was found for the defining polynomial $x^7 + x^3 + x^2 + x + 1$.

4.4.2 The 7-bit sbox (SB)

The search for lightweight 7-bit Sboxes explored variants with different nonlinearity, differential uniformity and number of rounds, balancing with small hardware cost; the Sboxes explored were in the range of 55 − 65 GE for their pre-PAR implementation area. While constructing the 7-bit Sboxes, we chose the nonlinear transformations Q that have efficient hardware implementation and varied all 5040 (= 7!) bit permutations (P). The chosen Sbox SB, described in Section 2.3.3, has differential uniformity 8 and nonlinearity 44, and can be implemented with just 58 GE.

Table 4.1: Area implementation results for the defining polynomials $f_i(x)$ for $\mathbb{F}_{2^7}$

Defining polynomial $f_i(x)$	constant term area [GE]	WGP area [GE]	sum† [GE]
x^7+x+1	2	258	260
x^7+x^3+1	16	247	263
$x^7+x^3+x^2+x+1$	10	245	255
x^7+x^4+1	23	243	266
$x^7+x^4+x^3+x^2+1$	22	255	277
$x^7+x^5+x^2+x+1$	24	258	282
$x^7+x^5+x^3+x+1$	6	261	267
$x^7+x^5+x^4+x^3+1$	16	264	280
$x^7+x^5+x^4+x^3+x^2+x+1$	19	251	270
x^7+x^6+1	14	270	284
$x^7+x^6+x^3+x+1$	28	248	276
$x^7+x^6+x^4+x+1$	29	261	290
$x^7+x^6+x^4+x^2+1$	27	265	292
$x^7+x^6+x^5+x^2+1$	16	257	273
$x^7+x^6+x^5+x^3+x^2+x+1$	26	257	283
$x^7+x^6+x^5+x^4+1$	31	259	290
$x^7+x^6+x^5+x^4+x^2+x+1$	20	254	274
$x^7+x^6+x^5+x^4+x^3+x^2+1$	14	255	269

† sum of the constant term impl. area and the WGP impl. area

4.5 Number of Rounds

Our rationale for the number of rounds (say n_r) is to choose a value for which the behavior of WAGE is close to a random permutation. We now justify our choice of $n_r = 111$ as follows.

1. WAGE adopts a shift register based structure with 37 7-bit words, and hence $n_r \geq 37$, otherwise the words will not be mixed among themselves properly, which leads to meet/miss-in-the-middle attacks.

2. For $n_r = 74$, the MEDCP of WAGE equals $2^{-4\times 59} = 2^{-236} > 2^{-259}$. Thus, to push the MEDCP value below 2^{-259}, $n_r \geq 74$. However, it is infeasible to compute the value of MEDCP for $n_r \geq 74$. Thus, we expect that for $n_r = 111$, MEDCP $<< 2^{-259}$ (see Section 5.1.1).

4.6 Round Constants

The round constants are added to mitigate the self-symmetry based distinguishers as mentioned in Section 2.3.5. We use a single 7-stage LFSR to generate a pair of constants at each round. Our choice of the utilized LFSR polynomial ensures that each pair of such constants does not repeat, due to the periodicity of the 8-tuple sequence constructed from the decimated m-sequence of period 127. Below we provide the details of how to generate the round constants.

4.6.1 Generation of round constants

We use an LFSR of length 7 with feedback polynomial $x^7 + x + 1$ to generate the round constants of WAGE. To construct these constants, the same LFSR is run in a 2-way parallel configuration, as illustrated in Figure 4.1. Let $\underline{a}$ denote the sequence generated by the initial state $(a_0, a_1, \ldots, a_6)$ of the LFSR without parallelization. The parallel version of this LFSR outputs two sequences, both of them using decimation exponent 2. More precisely,

- rc_0^i corresponds to the sequence $\underline{a}$ with decimation 2
- rc_1^i corresponds to the sequence $\underline{a}$ shifted by 1, then decimated by 2

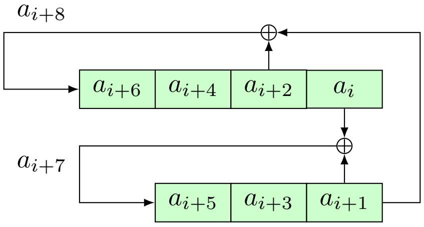


Figure 4.1: The LFSR for generating WAGE round constants.

The computation of round constants does not need any extra circuitry, but rather uses a feedback value a_{i+7} together with all 7 state bits, annotated in Figure 4.1. In Figure 4.2 we show how the 8 consecutive sequence elements are used to generate round constants. The round constants are given by:

$$rc_0^i = a_{i+6} \| a_{i+5} \| a_{i+4} \| a_{i+3} \| a_{i+2} \| a_{i+1} \| a_i$$
$$rc_1^i = a_{i+7} \| a_{i+6} \| a_{i+5} \| a_{i+4} \| a_{i+3} \| a_{i+2} \| a_{i+1}$$

$$\overbrace{a_{i+7}, \rlap{$\underbrace{\phantom{a_{i+6}, a_{i+5}, a_{i+4}, a_{i+3}, a_{i+2}, a_{i+1}, a_i}}_{rc_0^i}$}a_{i+6}, a_{i+5}, a_{i+4}, a_{i+3}, a_{i+2}, a_{i+1}}^{rc_1^i}, a_i$$

Figure 4.2: Two 7-bit step constants, generated from 8 consecutive sequence elements

We provide an example of the hex conversion of constants from LFSR sequence in Appendix A.3. The first five round constant pairs are shown in Table A.1.

4.7 Loading and Tag Extraction

The 128-bit key K and 128-bit nonce N are divided into 7-bit tuples In software we work with bytes, and since WAGE is using 7-bit tuples, we have "left-over" bits k_{63} and n_{63}; instead of shifting all remaining key and nonce bits by 1, the bits n_{63} and k_{63} are put into the last key block $\widehat{K}^*_{18}$, which makes the initialization and key absorption efficient for the software implementation.

Recall the data inputs $D_k, k = 0, \ldots 9$, in the shift register as shown in Figure 2.1. In order to minimize the hardware overhead, we reuse the data inputs D_k for loading. However, instead of XORing the D_k with previous stage content, the D_k data is fed directly into the corresponding stage. We have 10 D_k inputs, but must load the entire state, i.e. 37 stages. The stages without D_k inputs will be loaded by shifting. We divide the stages without D_k inputs into loading regions, e.g.the loading region $S_8, \ldots, S_0$ can be loaded through the data input D_0 and has length 9, hence will require 9 shifts for loading. The loading region $S_8, \ldots, S_0$ is the last part of the shift register in Figure 2.3, and has a nonlinear input from the SB, which must be disconnected during the loading. The remaining 3 SB must also be grounded. By inspecting the shift register, we find two other loading regions of length 9, namely region $S_{27}, \ldots, S_{19}$ (loaded through D_5) and region $S_{36}, \ldots, S_{28}$ (loaded through D_9). We decided to split the remaining 10 consecutive stages into two regions, one of length 8 and another of length 2. The region of length 8 are the stages $S_{16}, \ldots, S_9$, loaded through D_3, and the region of length 2 the stages S_{18}, S_{17}, loaded through D_4. Note that there is no need to disconnect the two WGP because they are automatically disabled by loading through D_9 and D_4.

These five loading regions, annotated with D_k used for loading, are listed below in a way that reflects their respective lengths. The K_i and N_j tuples on the right show the contents of the stages S_j after the loading is complete.

$$\begin{array}{rcl}
S_{36}, S_{35}, S_{34}, S_{33}, S_{32}, S_{31}, S_{30}, S_{29}, S_{28} & \leftarrow^{D_9} & \widehat{N}_{16}, \widehat{N}_{14}, \widehat{N}_{12}, \widehat{N}_{10}, \widehat{N}_8, \widehat{N}_6, \widehat{N}_4, \widehat{N}_2, \widehat{N}_0 \\
S_{27}, S_{26}, S_{25}, S_{24}, S_{23}, S_{22}, S_{21}, S_{20}, S_{19} & \leftarrow^{D_5} & \widehat{K}_{17}, \widehat{K}_{15}, \widehat{K}_{13}, \widehat{K}_{11}, \widehat{K}_9, \widehat{K}_7, \widehat{K}_5, \widehat{K}_3, \widehat{K}_1 \\
S_{18}, S_{17} & \leftarrow^{D_4} & \widehat{K}^*_{18}, N_{15} \\
S_{16}, S_{15}, S_{14}, S_{13}, S_{12}, S_{11}, S_{10}, S_9 & \leftarrow^{D_3} & \widehat{N}_{17}, \widehat{N}_{13}, \widehat{N}_{11}, \widehat{N}_9, \ \widehat{N}_7, \widehat{N}_5, \widehat{N}_3, \widehat{N}_1 \\
S_8, \ S_7, \ S_6, \ S_5, \ S_4, \ S_3, \ S_2, \ S_1, \ S_0 & \leftarrow^{D_0} & \widehat{K}_{16}, \widehat{K}_{14}, \widehat{K}_{12}, \widehat{K}_{10}, \widehat{K}_8, \widehat{K}_6, \widehat{K}_4, \widehat{K}_2, \widehat{K}_0
\end{array}$$

The actual loading process for regions $S_{18}, \ldots, S_9$ and $S_8, \ldots, S_0$ is shown in Table 4.2. The table shows the shifting of data through the register stages in 9 shifts. The stages are shown in the second row of Table 4.2, and the values "-" in the table denote the old, unknown values, which will be overwritten by the specified K_i and N_j blocks by the time the loading is finished. The state of stages $S_{18}, \ldots S_0$ after shifting 9 times, i.e. after the loading is finished, is visible from the last row.

The tag is extracted in a similar fashion, from the positions that were loaded with nonce tuples. For example, the state region $S_{16}, \ldots, S_9$, which was loaded through D_3, will be extracted through the output that belongs to the D_1 input. Similarly, the state region S_{18}, S_{17} will be extracted through the output belonging to the D_3 input and the region $S_{36}, \ldots, S_{28}$ through the output belonging to the D_6 input.

Table 4.2: Loading into the shift register through data inputs D_4, D_3 and D_0

shift	D_4	D_3	D_0
count	S_{18},S_{17}	$S_{16}, S_{15}, S_{14}, S_{13}$,$S_{12}$,$S_{11}$,$S_{10}$,$S_9$	S_8, S_7, S_6, S_5, S_4, S_3, S_2, S_1, S_0
1	- -	- - - - - - - -	$\widehat{K}_0$ - - - - - - - -
2	- -	$\widehat{N}_1$ - - - - - - -	$\widehat{K}_2$, $\widehat{K}_0$ - - - - - - -
3	- -	$\widehat{N}_3$, $\widehat{N}_1$ - - - - - -	$\widehat{K}_4$, $\widehat{K}_2$, $\widehat{K}_0$ - - - - - -
4	- -	$\widehat{N}_5$, $\widehat{N}_3$, $\widehat{N}_1$ - - - - -	$\widehat{K}_6$, $\widehat{K}_4$, $\widehat{K}_2$, $\widehat{K}_0$ - - - - -
5	- -	$\widehat{N}_7$, $\widehat{N}_5$, $\widehat{N}_3$, $\widehat{N}_1$ - - - -	$\widehat{K}_8$, $\widehat{K}_6$, $\widehat{K}_4$, $\widehat{K}_2$, $\widehat{K}_0$ - - - -
6	- -	$\widehat{N}_9$, $\widehat{N}_7$, $\widehat{N}_5$, $\widehat{N}_3$,$\widehat{N}_1$ - - -	$\widehat{K}_{10}$,$\widehat{K}_8$, $\widehat{K}_6$, $\widehat{K}_4$, $\widehat{K}_2$,$\widehat{K}_0$ - - -
7	- -	$\widehat{N}_{11}$,$\widehat{N}_9$, $\widehat{N}_7$, $\widehat{N}_5$,$\widehat{N}_3$,$\widehat{N}_1$ - -	$\widehat{K}_{12}$,$\widehat{K}_{10}$,$\widehat{K}_8$, $\widehat{K}_6$, $\widehat{K}_4$,$\widehat{K}_2$,$\widehat{K}_0$ - -
8	$\widehat{N}_{15}$ -	$\widehat{N}_{13}$,$\widehat{N}_{11}$,$\widehat{N}_9$, $\widehat{N}_7$,$\widehat{N}_5$,$\widehat{N}_3$,$\widehat{N}_1$ -	$\widehat{K}_{14}$,$\widehat{K}_{12}$,$\widehat{K}_{10}$,$\widehat{K}_8$, $\widehat{K}_6$,$\widehat{K}_4$,$\widehat{K}_2$,$\widehat{K}_0$ -
9	$\widehat{K}_{18}\widehat{N}_{15}$	$\widehat{N}_{17}$,$\widehat{N}_{13}$,$\widehat{N}_{11}$,$\widehat{N}_9$,$\widehat{N}_7$,$\widehat{N}_5$,$\widehat{N}_3$,$\widehat{N}_1$	$\widehat{K}_{16}$,$\widehat{K}_{14}$,$\widehat{K}_{12}$,$\widehat{K}_{10}$,$\widehat{K}_8$,$\widehat{K}_6$,$\widehat{K}_4$,$\widehat{K}_2$,$\widehat{K}_0$

The longest tag extraction region is also of length 9.

4.8 Choice of Rate Positions

The internal state constitutes of a rate part and a capacity part in which the adversary has freedom to inject messages into the state through the rate part. The rate positions in the state, as given in Section 2.4.1, are chosen by considering the security and efficient hardware implementation. From a security point of view, the chosen rate positions allow the input bits to be processed by the six Sboxes and diffused by the feedback polynomial as soon as possible after absorbing the message into the state, thus a faster confusion and diffusion is achieved. Moreover, our choice ensures any injected differences to activate Sboxes in the first two rounds which also enhances resistance to differential and linear cryptanalysis.

Exploiting the shifting property, the length of the process of updating the rate positions is minimized. The current choice of rate positions also allows an efficient loading and tag extraction within 9 consecutive clock cycles.

4.9 Relationship to WG ciphers

The WG cipher is a family of word-oriented stream ciphers based on an LFSR, a WG transformation and a WG permutation module over an extension field. The first family member, WG-29 [20], proceeded to Phase 2 of the eSTREAM competition [8]. Later, the lightweight variants WG-5 [2], WG-7 [17] and WG-8 [10] were proposed for constrained environments, e.g. RFID, and WG-16 [24, 11, 9] was proposed for 4G LTE.

We adopt the initialization phase of the WG cipher where we chose a decimated WG permutation with good cryptographic properties and tweak it to construct the round function of WAGE. Our proposed tweak brings faster confusion and diffusion in the state update. We choose the decimated WG permutation with decimation $d = 13$ for which its differential uniformity is 6 and nonlinearity 42 [18].

We make the tweak hardware efficient so that by disconnecting the second WGP module and all four SB modules, and keeping the domain separator 0, the round function of WAGE becomes identical to the WG initialization phase.

4.10 Statement

The authors declare that there are no hidden weaknesses in WAGE-$\mathcal{AE}$-128.

Chapter 5

Security Analysis

5.1 Security of WAGE Permutation

In this section, we analyze the security of the WAGE permutation against generic distinguishers. Formally, we show that WAGE with 111 rounds is indistinguishable from a random permutation. In the following, we denote the nonzero coefficients $c_i \in \{0, 1\}$ of a degree 37 primitive polynomial $l(y) = y^{37} + \sum_{i=1}^{36} c_i y^i + \omega \in \mathbb{F}_{2^7}$ by the vector $\vec{c}$.

5.1.1 Differential distinguishers

In WAGE, we use two distinct 7-bit Sboxes namely, WGP and SB as the nonlinear components. The differential probabilities of the Sboxes are $2^{-4.42}$ and 2^{-4}, respectively. To evaluate the maximum expected differential characteristic probability (MEDCP), we bound the minimum number of active Sboxes using a Mixed Integer Linear Programming (MILP) model that takes as input $\vec{c}$, the position of Sboxes and the number of rounds r. It then returns the minimum number of active Sboxes denoted by $n_r(\vec{c})$. In Table 5.1, we list the values of $n_r(\vec{c})$ for varying $\vec{c}$ and $r \in \{37, 44, 51, 58, 74\}$.

The MEDCP is then given by:

$$\text{MEDCP} = \max(2^{-4.42}, 2^{-4})^{n_r(\vec{c})} = 2^{-4 \times n_r(\vec{c})}.$$

Note that for $r = 74$ and $\vec{c} = (31, 30, 26, 24, 19, 13, 12, 8, 6)$, we have MEDCP $= 2^{-4\times 59} = 2^{-236} > 2^{-259}$. Since, the MILP solver [1] is unable to finish for $r > 74$, we expect that for our choice of $\vec{c}$, $n_{111}(\vec{c}) \geq 65$. This is because for each additional 7 rounds, the number of active Sboxes increases by at least 6 (see row 10 in Table 5.1) which implies MEDCP $\leq 2^{-260} < 2^{-259}$.

5.1.2 Diffusion behavior

To achieve full bit diffusion, i.e., each output bit of the permutation depends on all the input bits, we need at least 21 rounds. This is because the 7 bits of S_{36} is shifted to S_0 in 21 clock cycles. However, as the feedback function consists of 10 taps and all six Sboxes (2 WGP and 4 SB) individually have the full bit diffusion property,

Table 5.1: Minimum number of active Sboxes $n_r(\vec{c})$ for varying primitive polynomials

Primitive poly. coefficients	Rounds r				
$\vec{c}$	37	44	51	58	74
24, 23, 22, 21, 19, 6, 5, 4, 3	18	26	30	35	51
29, 27, 24, 23, 19, 11, 9, 6, 5	23	31	36	41	54
29, 28, 23, 22, 19, 11, 10, 5, 4	21	28	34	40	54
29, 28, 24, 20, 19, 11, 10, 6, 2	21	27	34	40	54
30, 28, 27, 21, 19, 12, 10, 9, 3	22	30	34	39	54
30, 29, 28, 26, 19, 12, 11, 10, 8	20	30	37	44	57
31, 25, 23, 21, 19, 13, 7, 5, 3	20	29	33	38	54
31, 26, 23, 20, 19, 13, 8, 5, 2	20	26	34	39	54
31, 28, 23, 21, 19, 13, 10, 5, 3	19	27	33	39	53
31, 30, 26, 24, 19, 13, 12, 8, 6	24	30	38	44	59
32, 25, 24, 21, 19, 14, 7, 6, 3	19	28	34	39	54
32, 29, 25, 22, 19, 14, 11, 7, 4	19	28	36	41	57
32, 29, 27, 22, 19, 14, 11, 9, 4	23	31	37	41	57
32, 29, 27, 24, 19, 14, 11, 9, 6	23	31	37	39	55
32, 30, 28, 24, 19, 14, 12, 10, 6	23	29	38	44	58
32, 31, 21, 20, 19, 14, 13, 3, 2	21	26	30	36	47
33, 27, 26, 20, 19, 15, 9, 8, 2	21	30	35	39	55
33, 29, 28, 21, 19, 15, 11, 10, 3	22	27	35	39	53
33, 30, 29, 26, 19, 15, 12, 11, 8	21	31	38	44	57
33, 31, 23, 22, 19, 15, 13, 5, 4	23	31	36	41	55
33, 31, 28, 23, 19, 15, 13, 10, 5	23	30	36	41	-
33, 31, 29, 22, 19, 15, 13, 11, 4	22	32	37	44	-
33, 31, 30, 25, 19, 15, 13, 12, 7	23	34	39	44	-

WAGE achieves the full bit diffusion in at most 37 rounds. Accordingly, we claim that meet/miss-in-the middle distinguishers may not cover more than 74 rounds as 74 rounds guarantee full bit diffusion in both the forward and backward directions.

5.1.3 Algebraic degree

The WGP and SB sboxes have an algebraic degree of 6. Note that if we only have WGP sbox at position S_{36} along with feedback polynomial and exclude all other sboxes and intermediate XORs, then we get the original WG stream cipher [20]. Such a stream cipher is resistant to attacks exploiting the algebraic degree if nonlinear feedback is used in the key generation phase [23, 22].

Given that WAGE has 6 Sboxes and we use nonlinear feedback for all of them, we expect that 111-round WAGE is secure against integral attacks.

5.1.4 Self-symmetry based distinguishers

WAGE employs two 7-bit round constants, rc_0 and rc_1, which are XORed to S_{36} and S_{18}, respectively. The round constant tuple is distinct for each round, i.e., $(rc_0^i, rc_1^i) \neq (rc_0^j, rc_1^j)$ for $0 \leq i, j \leq 110$ and $i \neq j$. This property ensures that all the rounds of WAGE are distinct and thwart attacks which exploit the symmetric properties of the round function [7, 16].

5.2 Security of WAGE-$\mathcal{AE}$-128

The security proofs of modes based on the sponge construction rely on the indistinguishability of the underlying permutation from a random one [4, 6, 5, 14]. In previous sections, we have shown that the behavior of the WAGE permutation for 111 rounds is close to a random permutation. Thus, the security bounds of the sponge duplex mode are applicable to WAGE-$\mathcal{AE}$-128. Moreover, we assume a nonce-respecting adversary, i.e, for a fixed K, nonce N is never repeated during encryption queries. Then, considering a data limit of 2^d, the k-bit security is achieved if $c \geq k + d + 1$ and $d \ll c/2$ [5]. The parameter set of WAGE (see Table 2.2) with actual effective capacity 193 (2 bits are lost for domain separation) satisfies this condition, and hence WAGE-$\mathcal{AE}$-128 provides 128-bit security for confidentiality, integrity and authenticity.

Chapter 6

Hardware Design And Analysis

In this chapter, we describe the hardware implementation of the WAGE permutation and the WAGE_module, which supports both authenticated encryption and verified decryption functionalities.

6.1 Hardware Design Principles

In this section, we describe the design principles and assumptions that we follow while implementing WAGE and WAGE_module.

1. **Multi-functionality module.** The system should support both the operations, namely authenticated encryption and verified decryption, in a single module, because lightweight applications generally cannot afford the extra area for separate modules. As a result, the area for the system will be greater compared to a single-function module.

2. **Single input/output ports.** In small devices, ports can be expensive, and optimizing the number of ports may require additional multiplexers and control circuitry. To ensure that we are not biasing our design in favour of the system and at the expense of the environment, the key, nonce, associated data, and message all use a single data-input port. Similarly, the output ciphertext, tag, and hash all use a single output port.

3. **Valid-bit protocol and stalling capability.** The environment may take an arbitrarily long time to produce any piece of data. For example, a small microprocessor could require multiple clock cycles to read data from the memory and write it to the system's input port. We use a single-phase valid bit protocol, where each input or output data signal is paired with a valid bit to denote when the data is valid. The receiving entity must capture the data in a single clock cycle, which is a simple and widely applicable protocol. The system shall wait in an idle state, while signalling the environment that it is ready to receive.

4. **Use a "pure register-transfer-level" implementation style.** In particular, use only registers, not latches; multiplexers, not tri-state buffers; and synchronous, not asynchronous reset.

6.2 Interface and Top-level Module

In Figure 6.1, we depict the block diagram of the top-level WAGE_module. The description of each interface signal is given in Table 6.1.

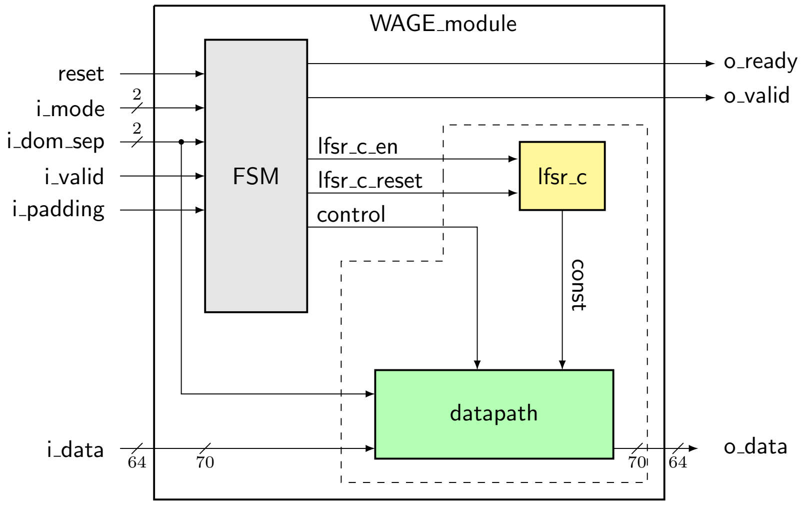


Figure 6.1: Top-level module and interface

Table 6.1: Interface signals

Input signal	Meaning	Output signal	Meaning
reset	resets the state machine	o_ready	hardware is ready
i_mode	mode of operation	o_data	output data
i_dom_sep	domain separator	o_valid	valid data on o_data
i_padding	the last block is padded		
i_data	input data		
i_valid	valid data on i_data		

WAGE-$\mathcal{AE}$-128 performs two operations the authenticated encryption (WAGE-$\mathcal{E}$) and verified decryption (WAGE-$\mathcal{D}$). We use the i_mode input signal to distinguish between the two operations.

The environment separates the associated data and the message/ciphertext, and performs their padding if necessary, as specified in Section 2.4.2. The control input i_pad is used to indicate that the last i_data is padded. The domain separators, provided by the environment, serve as an indication of the phase change, e.g., for transition between processing associated data, encryption/decryption, and finalization. The WAGE_module is unaware of the lengths ℓ_{AD}, ℓ_M and ℓ_C, hence no internal counters for the number of processed blocks are needed.

6.3 The WAGE Datapath

Figure 6.2 shows the schematic for the WAGE datapath. The right side of the figure depicts the two symmetrical nonlinear portions of WAGE, namely the SBs and WGPs. The two round constants produced by lfsr_c are being XORed to the outputs of the

WGPs. Note that the lfsr_c used for the generation of the round constants is not shown in Figure 6.2.

In the following sections, we first briefly illustrate the implementation details of WAGE datapath and WAGE_module. We also provide an estimate and the actual numbers for their hardware areas.

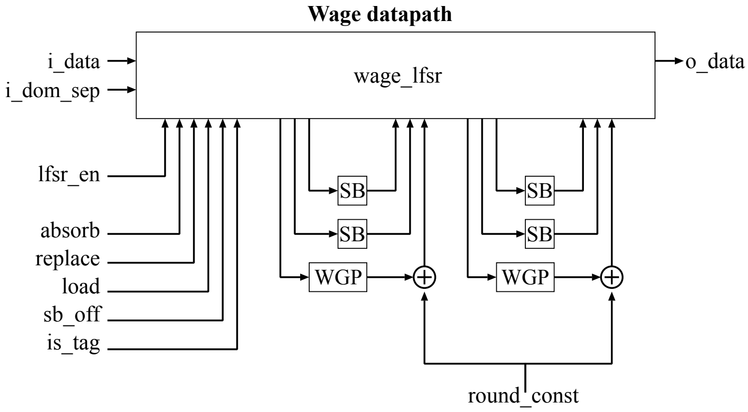


Figure 6.2: WAGE datapath

6.3.1 Components of **WAGE** datapath

Below we list the building blocks of WAGE datapath with a short descriptions of their implementations.

- wage_lfsr. The LFSR has 37 stages with 7 bits per stage, and a feedback with 10 taps and a module for multiplication with the constant ω.
- WGP module. For smaller fields like $\mathbb{F}_{2^7}$, the WGP area, when implemented as a constant array in VHDL/Verilog, i.e., as a look-up table, is smaller than when implemented with basic arithmetic blocks (implementing multiplications and exponentiations to the powers of two) [2, 17, 10]. However, the WGP is not stored in hardware as a memory array, but rather as a net of AND, OR and NOT gates, derived and optimized by the synthesis tools.
- SB module. The SB is implemented in unrolled fashion, i.e., as a purely combinational logic, composed of 5 copies of R, followed by a Q and the final two NOT gates (Section 2.3.3).

In Table 6.2, we provide an estimate of the hardware area needed for implementation of the WAGE permutation datapath. For the CMOS 65 nm we use an estimate of 3.75 GE for a 1-bit register and 2.00 GE for a 2-input XOR gate. The row “other XORs” contains the XOR gates needed to add the values from the modules SB and WGP, and the lfsr_c constants to the wage_lfsr stages. We also report the actual implementation results for the WAGE permutation area.

Table 6.2: WAGE permutation hardware area estimate

Component	**Estimate per unit** [GE]	**Count**	**Estimate per component** [GE]
wage_lfsr registers	3.75	37×7	971
wage_lfsr feedback XORs	2.00	10×7	140
wage_lfsr feedback ω	10†	1	10
SB	58†	4	232
WGP	245†	2	490
lfsr_c	45†	1	45
Other XORs	2.00	8×7	112
WAGE permutation - Total estimated area			2000

pre-PAR CMOS 65 nm implementation area results	
WAGE permutation	2051†

† pre-PAR implementation results

6.3.2 The WAGE_module and the control

During the WAGE permutation, most stages will shift just as in a regular LFSR. The only exception are the stages where the nonlinear inputs, i.e., the outputs from both WGPs and all four SB, are XORed into the state. For the encryption, the WAGE permutation datapath is modified to accommodate loading and absorbing (during initialization, processing of associated data, finalization and encryption). Firstly, XOR gates and accompanying multiplexers are added to the S_r stages of wage_lfsr for absorbing. Another XOR and a multiplexer is needed for the the domain separator. To support decryption as well, another layer of multiplexers is added to wage_lfsr. Finally, more multiplexers are used to turn off the nonlinear components during the loading and tag extraction. All added signals, gates and the multiplexers, except the ones needed for the domain separator, are 7-bits wide. The control signals shown in Figure 6.2 are self-explanatory and details are omitted.

6.4 Hardware Implementation Results

In this section, we provide the ASIC CMOS and FPGA implementation results of WAGE permutation and WAGE_module. We first give the details of the used synthesis and simulation tools and then present the performance results.

	Synthesis and simulation tools and libraries for the ASIC implementation
Logic synthesis	Synopsys Design Compiler vN-2017.09
Physical synthesis	Cadence Encounter 2014.13-s036_1
Simulation	Mentor Graphics QuestaSim 10.5c
ASIC cell library	65 *nm* STMicroelectronics CORE65LPLVT, 1.25V, 40C

	Synthesis tools for the FPGA implementation
Logic synthesis	Mentor Graphics Precision 64-bit 2016.1.1.28 (for Intel/Altera), Xilinx ISE Project Navigator 14.4 P.49d (for Xilinx)
Physical synthesis	Altera Quartus Prime 15.1.0 SJ Standard Edition (for Intel/Altera), Xilinx ISE Project Navigator 14.4 P.49d (for Xilinx)

6.4.1 Performance results

In Tables 6.3 and 6.4, we present the performance results of the WAGE permutation and the WAGE_module.

Table 6.3: pre-PAR ASIC CMOS 65 nm implementation results

	Frequency	**Area**	**Throughput**
Module	[MHz]	[GE]	[Mbps]
WAGE permutation	1429	2051	–
WAGE_module	1053	2994	607

Table 6.4: post-PAR FPGA implementation results

	Extract†	**Frequency**	**# of**	**# of**	**# of**
Module	**attribute**	[MHz]	**Slices**	**FFs**	**LUTs**
Xilinx Spartan 3 (xc3s200-5ft256)					
WAGE permutation	yes	145	139	161	168
	no	160	282	237	313
WAGE_module	yes	96	326	212	531
	no	92	455	284	699
Xilinx Spartan 6 (xc6slx9-3ftg256)					
WAGE permutation	yes	214	42	161	134
	no	218	89	237	211
WAGE_module	yes	129	144	232	367
	no	134	149	281	431

	Frequency	**# of**	**# of**	**# of**
Module	[MHz]	**LC**	**FFs**	**LUTs**
Intel / Altera Stratix IV (EP4SGX70HF35M3)				
WAGE permutation	92	195	195	129
WAGE_module	73	372	372	259

† WAGE_module includes a shift register wage_lfsr and two constant array modules (WGPs) . We set the attributes SHREG_EXTRACT, ROM_EXTRACT and RAM_EXTRACT to (dis)allow optimizations to shift-register configuration LUTs and Block RAMs, hence two sets of implementation results. When memory is inferred, 1 RAMB16 is used for Spartan 3, and 1 RAMB8BWER for Spartan 6.

Chapter 7

Software Efficiency Analysis

The WAGE permutation is designed to be efficient on heterogeneous resource constrained devices, which imposes the primitive to be efficient in hardware as well as in software. We assess the efficiency of the WAGE permutation and its modes on three different microcontroller platforms.

7.1 Software: Microcontroller

We implemented the WAGE permutation and WAGE-$\mathcal{AE}$-128 on three distinct microcontroller platforms. For WAGE-$\mathcal{AE}$-128, we implement only encryption, because decryption is the same as encryption, except updating the rate with ciphertext. Our codes were written in assembly language to achieve optimal performance. We choose: 1) the Atmel ATmega128, an 8-bit mocrocontroller with 128 Kbytes of programmable flash memory, 4.448 Kbytes of RAM, and 32 general purpose registers of 8 bits, 2) MSP430F2370, a 16-bit mocrocontroller from Texas Instruments with 2.3 Kbytes of programmable flash memory, 128 Bytes of RAM, and 12 general purpose registers of 16 bits, and 3) ARM Cortex M3 LM3S9D96, a 32-bit microcontroller with 524.3 Kbytes of programmable flash memory, 131 Kbytes of RAM, and 13 general purpose registers of 32 bits. We focus on four key performance measures, namely throughput, code size (Kbytes), energy (nJ), and RAM (Kbytes) consumption.

For WAGE-$\mathcal{E}$, the scheme is instantiated with a random 128-bit key and a 128-bit nonce. Note that the throughput of the WAGE-$\mathcal{AE}$, which is processing words, is smaller than that of the WAGE permutation. For producing a ciphertext and a tag by WAGE-$\mathcal{E}$, $(5 + \ell)$ executions of the permutation are required where ℓ is the total number of the 64-bit data blocks including the associated data, plaintext message and padding if needed. We chose two combinations of the numbers of the AD block (ℓ_{AD}) and the message block (ℓ_M), which are: 1) $(\ell_{AD}, \ell_M) = (0, 16)$, meaning no AD and 1024-bit plaintext message; and 2) $(\ell_{AD}, \ell_M) = (2, 16)$, meaning 128-bit AD and 1024-bit plaintext message. Table 7.1 presents the performance of the WAGE permutation and its modes for these two choices of AD and message lengths.

Table 7.1: Performance of WAGE on microcontrollers

Cryptographic	**Platform**		**Clock freq.**	**Memory usage** [Bytes]		**Setup**	**Throughput**	**Energy/bit**
primitive	Device	Bit	[MHz]	SRAM	Flash	[Cycles]	[Kbps]	[nJ]
WAGE Permutation	ATmega128	8	16	802	4132	19011	217.98	568
WAGE Permutation	MSP430F2370	16	16	4	5031	23524	176.16	135
WAGE Permutation	LM3S9D96	32	16	3076	5902	14450	286.78	1162
WAGE-$\mathcal{E}$ ($l_{AD} = 0, l_M = 16$)	ATmega128	8	16	808	4416	362888	45.15	2741
WAGE-$\mathcal{E}$ ($l_{AD} = 0, l_M = 16$)	MSP430F2370	16	16	46	5289	433105	37.83	628
WAGE-$\mathcal{E}$ ($l_{AD} = 0, l_M = 16$)	LM3S9D96	32	16	3084	6230	278848	58.76	5673
WAGE-$\mathcal{E}$ ($l_{AD} = 2, l_M = 16$)	ATmega128	8	16	808	4502	397260	41.24	3001
WAGE-$\mathcal{E}$ ($l_{AD} = 2, l_M = 16$)	MSP430F2370	16	16	46	5339	474067	34.56	687
WAGE-$\mathcal{E}$ ($l_{AD} = 2, l_M = 16$)	LM3S9D96	32	16	3084	6354	305284	53.67	6210

Acknowledgment

The submitters would like to thank Marat Sattarov for his help in the part of hardware implementation and Yunjie Yi for the microcontroller implementation.

Bibliography

[1] Gurobi: MILP optimizer. `http://www.gurobi.com/`.

[2] Aagaard, M. D., Gong, G., and Mota, R. K. Hardware implementations of the WG-5 cipher for passive rfid tags. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on* (2013), IEEE, pp. 29–34.

[3] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. sLiSCP: Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives. In *SAC* (2017), C. Adams and J. Camenisch, Eds., Springer, pp. 129–150.

[4] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge functions. In *ECRYPT hash workshop* (2007), vol. 2007.

[5] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. On the security of the keyed sponge construction. In *Symmetric Key Encryption Workshop* (2011).

[6] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *SAC* (2012), A. Miri and S. Vaudenay, Eds., Springer, pp. 320–337.

[7] Biryukov, A., and Wagner, D. Slide attacks. In *FSE* (1999), L. Knudsen, Ed., Springer, pp. 245–259.

[8] eSTREAM: the ecrypt stream cipher project. `http://www.ecrypt.eu.org/stream/`.

[9] El-Razouk, H., Reyhani-Masoleh, A., and Gong, G. New hardware implementationsof WG($\mathbf{29}, \mathbf{11}$) and WG-**16** streamciphers using polynomial basis. *IEEE Transactions on Computers 64*, 7 (July 2015), 2020–2035.

[10] Fan, X., Mandal, K., and Gong, G. WG-8: A lightweight stream cipher for resource-constrained smart devices. In *Quality, Reliability, Security and Robustness in Heterogeneous Networks* (Berlin, Heidelberg, 2013), K. Singh and A. K. Awasthi, Eds., Springer Berlin Heidelberg, pp. 617–632.

[11] Fan, X., Zidaric, N., Aagaard, M., and Gong, G. Efficient hardware implementation of the stream cipher WG-16 with composite field arithmetic.

In *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices* (New York, NY, USA, 2013), TrustED '13, ACM, pp. 21–34.

[12] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.10.0*, 2018.

[13] Gong, G., and Youssef, A. M. Cryptographic properties of the welch-gong transformation sequence generators. *IEEE Transactions on Information Theory 48*, 11 (Nov 2002), 2837–2846.

[14] Jovanovic, P., Luykx, A., and Mennink, B. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *ASIACRYPT* (2014), P. Sarkar and T. Iwata, Eds., Springe, pp. 85–104.

[15] Kölbl, S., Leander, G., and Tiessen, T. Observations on the Simon block cipher family. In *CRYPTO* (2015), R. Gennaro and M. Robshaw, Eds., Springer, pp. 161–185.

[16] Leander, G., Abdelraheem, M. A., AlKhzaimi, H., and Zenner, E. A cryptanalysis of printcipher: The invariant subspace attack. In *CRYPTO* (2011), P. Rogaway, Ed., Springer, pp. 206–221.

[17] Luo, Y., Chai, Q., Gong, G., and Lai, X. A lightweight stream cipher WG-7 for rfid encryption and authentication. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010* (Dec 2010), pp. 1–6.

[18] Mandal, K., Gong, G., Fan, X., and Aagaard, M. Optimal parameters for the WG stream cipher family. *Cryptography Commun. 6*, 2 (June 2014), 117–135.

[19] McKay, K., Bassham, L., Sönmez Turan, M., and Mouha, N. Report on lightweight cryptography (NISTIR8114), 2017.

[20] Nawaz, Y., and Gong, G. The WG stream cipher. *ECRYPT Stream Cipher Project Report 2005 33* (2005).

[21] Nawaz, Y., and Gong, G. WG: A family of stream ciphers with designed randomness properties. *Inf. Sci. 178*, 7 (Apr. 2008), 1903–1916.

[22] Rohit, R., AlTawy, R., and Gong, G. Milp-based cube attack on the reduced-round wg-5 lightweight stream cipher. In *Cryptography and Coding* (Cham, 2017), M. O'Neill, Ed., Springer International Publishing, pp. 333–351.

[23] Rønjom, S. Improving algebraic attacks on stream ciphers based on linear feedback shift register over f_{2^k}. *Des. Codes Cryptography 82*, 1-2 (2017), 27–41.

[24] Zidaric, N., Aagaard, M., and Gong, G. Hardware optimizations and analysis for the WG-16 cipher with tower field arithmetic. *IEEE Transactions on Computers 68*, 1 (Jan 2019), 67–82.

Appendix A

Test Vectors

A.1 WAGE Permutation

Input:00000000000000000000000000000000000000000000000000000000000000000
Output:0FA82908FEA670F1B8609F00420FC3376A52DCA922061FED7C568F785C22B4A4C

A.2 WAGE-$\mathcal{AE}$-128

Key	:	00111122335588DD 00111122335588DD
Nonce	:	111122335588DD00 111122335588DD00
Associated data	:	1122335588DD0011 1122335588DD00
Plaintext	:	335588DD00111122 335588DD001111
Ciphertext	:	4B7CD23D07D75575 5EA2ADEC4FEFF3
Tag	:	D03CF7894D6D3697 C2B1758D41E78344

A.3 Round Constants Conversion

The round constants are translated to HEX values as shown in Table 2.2.

Table A.1: Generation of the first five round constant pairs (rc_1^i, rc_0^1)

clk.	(current)										
cycle	LFSR state	(current) subsequence bits								HEX	
		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0		
0	1 1 1 1		1	1	1	1	1	1	1	7 F	← rc_0^0
	1 1 1	0	1	1	1	1	1	1		3 F	← rc_1^0
		a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2		
1	0 1 1 1		0	0	1	1	1	1	1	1 F	← rc_0^1
	0 1 1	0	0	0	1	1	1	1		0 F	← rc_1^1
		a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4		
2	0 0 1 1		0	0	0	0	1	1	1	0 7	← rc_0^2
	0 0 1	0	0	0	0	0	1	1		0 3	← rc_1^2
		a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6		
3	0 0 0 1		0	0	0	0	0	0	1	0 1	← rc_0^3
	0 0 0	1	0	0	0	0	0	0		4 0	← rc_1^3
		a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8		
4	0 0 0		0	1	0	0	0	0	0	2 0	← rc_0^4
	1 0 0	0	0	1	0	0	0	0		1 0	← rc_1^4