Hardware Design and Analysis of the ACE and WAGE Ciphers

Mark D. Aagaard, Marat Sattarov, and Nuša Zidarić
Department of Electrical and Computer Engineering
University of Waterloo, Ontario, Canada
{maagaard,msattaro,nzidaric}@uwaterloo.ca

Abstract

This paper presents the hardware design and analysis of ACE and WAGE, two candidate ciphers for the NIST Lightweight Cryptography standardization. Both ciphers use sLiSCP’s unified sponge duplex mode. ACE has an internal state of 320 bits, uses three 64 bit Simeck boxes, and implements both authenticated encryption and hashing. WAGE is based on the Welch-Gong stream cipher and provides authenticated encryption. WAGE has 259 bits of state, two 7 bit Welch-Gong permutations, and four lightweight 7 bit S-boxes. ACE and WAGE have the same external interface and follow the same I/O protocol to transition between phases. The paper illustrates how a hardware perspective influenced key aspects of the ACE and WAGE algorithms. The paper reports area, power, and energy results for both serial and parallel (unrolled) implementations using four different ASIC libraries: two 65 nm libraries, a 90 nm library, and a 130 nm library. ACE implementations range from a throughput of 0.5 bits-per-clock cycle (bpc) and an area of 4210 GE (averaged across the four ASIC libraries) up to 4 bpc and 7260 GE. WAGE results range from 0.57 bpc with 2920 GE to 4.57 bpc with 11080 GE.

1 Introduction

In 2013, NIST started the Lightweight Cryptography (LW) project [1], with the end goal of creating a portfolio of lightweight algorithms for authenticated encryption with associated data (AEAD), and optionally hashing, in constrained environments [2]. For hardware-oriented lightweight algorithms, hardware implementation results are an important criteria for assessment and comparison. In the first round of the LWC evaluation, more than half of the candidates [3] reported hardware implementation results or their estimates, ranging from complete implementation and analysis to partial implementation results and theoretical estimates based on gate count. Various amounts of analysis, such as area reported only for a cryptographic primitive used or thorough area breakdown of all components, different design decisions, such as serial and unrolled implementations, and different ASIC and/or FPGA implementation technologies can be found. Furthermore, some authors report the results without interface, some with the interface, and in some cases, e.g. [6], CAESAR Hardware Applications Programming Interface (API) for Authenticated Ciphers [7] was used.

This paper explores different hardware design options for two of the LWC candidates, ACE [4] and WAGE [5]. The original and parallel implementations were synthesized using four different ASIC libraries, including 65nm, 90 nm and 130 nm technologies. ACE implementations range from a throughput of 0.5 bits-per-clock cycle (bpc) and an area of 4210 GE (averaged across the four ASIC libraries) up to 4 bpc and 7260 GE. WAGE results range from 0.57 bpc with 2920 GE and 4.57 bpc with 11080 GE.

The paper is organized as follows: Section 2 briefly introduces ACE and WAGE, Section 3 lists design principles and presents the interface with the environment and describes the implementations of both ciphers. Section 4 describes the parallel implementations of ACE and WAGE. Implementation technologies and results are summarized in Section 5.

2 Specifications of ACE and WAGE

Both ACE and WAGE permutations operate in a unified duplex sponge mode [9]. The 320 bit ACE permutation offers both AEAD and hashing functionalities, and the 259 bit WAGE permutation supports AEAD functionality. Because of the similarities between ACE and WAGE, this section begins with a short description of ACE and WAGE permutations, followed by a discussion on the unified duplex sponge mode for both schemes, highlighting some differences.

This work was supported in part by the Canadian National Science and Engineering Research Council (NSERC); the Canadian Microelectronics Corp (CMC); and Grant 60NANB16D289 from the U.S. Department of Commerce, National Institute of Standards and Technology (NIST).
2.1 ACE Permutation

ACE has a 320 bit internal state $S$, divided into five 64 bit registers, denoted A, B, C, D, and E. The 320 bit ACE permutation uses the unkeyed reduced-round Simeck block cipher [10] with a block size of 64 and 8 rounds, denoted S-box SB-64, as the nonlinear operation. SB-64 is a lightweight permutation, consisting of left cyclic shifts, AND and XOR gates. Each round is parameterized by a single bit of round constant $rc_i = (q_0, q_1, \ldots, q_7)$, $q_j \in \{0, 1\}$ and $0 \leq j \leq 7$. The algorithmic description of SB-64 is shown at the end of Algorithm 1, with the 64 bit input and output split into half, i.e. $(x_1|x_0)$ and $(x_9|x_8)$ respectively.

To construct the ACE-step($S^i$), $0 \leq i \leq 15$, SB-64 is applied to three registers A, C and E, with each using its own round constant $rc_i^A, rc_i^C, rc_i^E$. The 8 bit constants $rc_i^j$, are generated by an LFSR with the feedback polynomial $f(x) = x^7 + x^3 + x^2 + x + 1$ run in a 3-way parallel configuration to produce one bit of each $rc_i^j$ per clock cycle. At each step, the outputs of SB-64 are added to registers B, D and E, which are further parameterized by step constants $(sc_0, sc_1, sc_2)$. The computation of step constants does not need any extra circuitry, but rather uses the same LFSR as the round constants: the three feedback values together with all 7 state bits yield 10 consecutive sequence elements, which are then split into three 8 bit step constants. The step constants are used once every 8th clock cycle. The step function is then concluded by a permutation of all five registers. For the properties of SB-64, the choice of the final permutation, and the number of rounds and steps refer to [4].

2.2 WAGE Permutation

WAGE is a hardware oriented AE scheme, built on top of the initialization phase of the well-studied, LFSR based, Welch-Gong (WG) stream cipher [11, 12]. The WAGE permutation is iterative and has a round function derived from the LFSR, the decimated Welch-Gong permutation WGP, and the small S-boxes SB. Details, such as differential uniformity and nonlinearity of the WGP and SB and selection of the LFSR polynomial can be found in [5]. The parameter selection for WAGE was aimed at balancing the security and hardware implementation area, using hardware implementation results for many design decisions, e.g., field size, representation of field elements, LFSR polynomial, etc.

Both LFSR and WGP are defined over $\mathbb{F}_{2^7}$ and the S-box is a 7 bit permutation. $\mathbb{F}_{2^7}$ is defined with the primitive polynomial $f(x) = x^7 + x^3 + x^2 + x + 1$, and the field elements are represented using the polynomial basis $PB = \{1, \omega, \ldots, \omega^6\}$, where $\omega$ is the root of $f(x)$ (Table 1). The LFSR is defined by the feedback polynomial $\ell(x)$ (Table 1), which is primitive over $\mathbb{F}_{2^7}$. The 37 stages of the LFSR also constitute the internal state of WAGE, denoted $S^i = (S^i_{36}, S^i_{35}, \ldots, S^i_1, S^i_0)$; the subscript $i$ is used to mark the $i$-th iteration of the permutation. For the element $x \in \mathbb{F}_{2^7}$, the decimated WG permutation with decimation $d = 13$ is defined in Table 1. The 7 bit SB uses a nonlinear transformation $Q$ and a permutation $P$, which together yield one-round $R = P \circ Q$. The SB itself iterates the function $R$ 5 times, applies $Q$ once, and then complements the 0th and 2nd bit (Table 1).

| $\alpha \in \mathbb{F}_{2^7}$ | $\alpha = \sum_{i=0}^{6} a_i \omega^i, a_i \in \mathbb{F}_2$ | vector representation: | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |
|---|---|---|---|
| LFSR | $\ell(y) = y^7 + y^6 + y^3 + y^2 + y + 1$ | $\alpha$ | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |
| WGP | $\text{GF}^{(x)} = x^d + (x^d + 1)^{33} + (x^d + 1)^{39} + (x^d + 1)^{41} + (x^d + 1)^{104}$ | $\alpha$ | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |
| SB Q | $Q(x_0, x_1, x_2, x_3, x_4, x_5, x_6) \rightarrow (x_0 \oplus (x_2 \land x_3), x_1, x_2, x_3 \oplus (x_5 \land x_6), x_4, x_5 \oplus (x_2 \land x_4), x_6)$ | $\alpha$ | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |
| SB P | $P(x_0, x_1, x_2, x_3, x_4, x_5, x_6) \rightarrow (x_6, x_3, x_0, x_1, x_2, x_3, x_1)$ | $\alpha$ | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |
| SB R | $R(x_0, x_1, x_2, x_3, x_4, x_5, x_6) \rightarrow (x_6, x_3 \oplus (x_5 \land x_6), x_0 \oplus (x_2 \land x_3), x_4, x_2, x_5 \oplus (x_2 \land x_4), x_1)$ | $\alpha$ | $|\alpha|_{PB} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6)$ |

Table 1: Specification parameters of WAGE
Algorithm 1 ACE permutation

1: Input: $S^0 = A^0||B^0||C^0||D^0||E^0$
2: Output: $S^{16} = A^{16}||B^{16}||C^{16}||D^{16}||E^{16}$
3: for $i = 0$ to $15$ do:
4: \quad $S^{i+1} \leftarrow$ ACE-step($S^i$)
5: return $S^{16}$

6: Function ACE-step($S^i$):
7: \quad $A^i \leftarrow$ SB-64($A_1^i||A_0^i, r_{c_0}^i$)
8: \quad $C^i \leftarrow$ SB-64($C_1^i||C_0^i, r_{c_1}^i$)
9: \quad $E^i \leftarrow$ SB-64($E_1^i||E_0^i, r_{c_2}^i$)
10: \quad $B^i \leftarrow B^i \oplus C^i \oplus (1^{56}||s_{c_0}^i)$
11: \quad $D^i \leftarrow D^i \oplus E^i \oplus (1^{56}||s_{c_1}^i)$
12: \quad $E^i \leftarrow E^i \oplus A^i \oplus (1^{56}||s_{c_2}^i)$
13: \quad $A^{i+1} \leftarrow D^i$
14: \quad $B^{i+1} \leftarrow C^i$
15: \quad $C^{i+1} \leftarrow A^i$
16: \quad $D^{i+1} \leftarrow E^i$
17: \quad $E^{i+1} \leftarrow B^i$
18: return $(A^{i+1}||B^{i+1}||C^{i+1}||D^{i+1}||E^{i+1})$

19: Function SB-64($x_1||x_0, r_c$):
20: \quad $rc = (q_7, q_6, \ldots, q_0)$
21: for $j = 2$ to $9$ do
22: \quad $x_j \leftarrow (L^3(x_{j-1}) \oplus x_{j-1}) \oplus L^1(x_{j-1}) \oplus x_{j-2} \oplus (1^3||q_{j-2})$
23: return $(x_9||x_8)$
( L is left-rotation )

Algorithm 2 WAGE permutation

1: Input: $S^0 = (S_{36}^0, S_{35}^0, \ldots, S_0^0)$
2: Output: $S^{111} = (S_{36}^{111}, S_{35}^{111}, \ldots, S_1^{111}, S_0^{111})$
3: for $i = 0$ to $110$ do:
4: \quad $S^{i+1} \leftarrow$ WAGE-StateUpdate($S^i, r_{c_0}^i, r_{c_1}^i$)
5: return $S^{111}$

6: Function WAGE-StateUpdate($S^i$):
7: \quad $fb = S_{31}^i \oplus S_{30}^i \oplus S_{29}^i \oplus S_{28}^i \oplus S_{27}^i \oplus S_{26}^i \oplus S_{25}^i \oplus S_{24}^i \oplus S_{23}^i \oplus S_{22}^i \oplus S_{21}^i \oplus S_{20}^i \oplus S_{19}^i \oplus S_{18}^i \oplus S_{17}^i \oplus S_{16}^i \oplus S_{15}^i \oplus S_{14}^i \oplus S_{13}^i \oplus S_{12}^i \oplus S_{11}^i \oplus S_{10}^i \oplus S_{9}^i \oplus S_{8}^i \oplus S_{7}^i \oplus S_{6}^i \oplus S_{5}^i \oplus S_{4}^i \oplus S_{3}^i \oplus S_{2}^i \oplus S_{1}^i \oplus S_{0}^i$
8: \quad $S_{4}^{i+1} \leftarrow S_{5}^i \oplus SB(S_{8}^i)$
9: \quad $S_{10}^{i+1} \leftarrow S_{11}^i \oplus SB(S_{15}^i)$
10: \quad $S_{18}^{i+1} \leftarrow S_{19}^i \oplus WGP(S_{18}^i) \oplus r_{c_0}^i$
11: \quad $S_{23}^{i+1} \leftarrow S_{24}^i \oplus SB(S_{27}^i)$
12: \quad $S_{29}^{i+1} \leftarrow S_{30}^i \oplus SB(S_{34}^i)$
13: \quad $S_{36}^{i+1} \leftarrow fb \oplus WGP(S_{36}^i) \oplus r_{c_1}^i$
14: \quad $S_{j}^{i+1} \leftarrow S_{j+1}^i$ where $j \in \{0, \ldots, 36\} \setminus \{4, 10, 18, 23, 29, 36\}$
15: return $S^{i+1}$

As mentioned before, the WAGE permutation is iterative, and repeats its round function WAGE-StateUpdate($S^i$) 111 times, as shown in Algorithm 2. In each round, 6 stages of the LFSR are updated nonlinearly, while all the remaining stages are just shifted. A pair of 7 bit round constants ($r_{c_0}^i, r_{c_1}^i$) is XORed with the pair of stages (18, 36). Round constants are produced by an LFSR of length 7 with feedback polynomial $x^7 + x + 1$, implemented in a 2-way parallel configuration, see [5] for details.

2.3 The Unified Duplex Sponge Mode

ACE-Æ-128 and WAGE-Æ-128 use the unified duplex sponge mode from sLiSCP [9] (Figure 1). The phases for encryption and decryption are: initialization, processing of associated data, encryption (Figure 1(a)) or decryption (Figure 1(b)), and finalization. Figure 1 also shows the domain separators for each phase. The internal state is divided into a capacity part $S_c$ (256 bits for ACE-Æ-128 and 195 bits for WAGE) and a 64 bit rate $S_r$, which for:

- ACE-Æ-128 consists of bytes $A[7], A[6], A[5], A[4], C[7], C[6], C[5], C[4]$
- WAGE-Æ-128 consists of 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 $S_8$

The input data (associated data $AD$, message $M$ or ciphertext $C$) is absorbed (or replaced) into the rate part of the internal state. If the input data length is not a multiple of 64, padding with $(10^*\ast)$ is needed. In Figure 1, $d$ denotes the number of 64bit blocks of $AD$ and $m$ the number of 64bit blocks of $M$ and $C$ after padding. Refer to [4, 5] for further padding rules. No padding is needed during initialization and
finalization because both schemes use a 128 bit key. With the exception of tag extraction, both schemes generate an output only during the encryption and decryption phases: the 64 bit output block is obtained by the XOR of the current input and rate.

Figure 1 also shows functions load-AE(N, K) and tagextract(S), which are straightforward for ACE. The ACE load-AE(N, K) performs the loading of the 128 bit key K and nonce N, where the key is loaded into registers A and C, the nonce into B and E, and the register D is loaded with zeros. The ACE tagextract(S) extracts the 128 bit tag from registers A and C. Special care was taken in the specification of load-AE(N, K) and tagextract(S) of WAGE to take advantage of the shifting nature of the LFSR, which will be discussed in more detail in Section 3.3.

The ACE HASH functionality is shown in Figure 2, with only two phases, namely absorbing and squeezing. The only input is now the message M. Since the hash has a fixed length of 256 bits, the length of the squeezing phase is fixed. ACE-H-256 is unkeyed, and the state is loaded with a fixed initialization vector IV. More specifically, the function load-H(IV) loads the state bytes B[7], B[6] and B[5] with bytes 0x80, 0x40, and 0x40 respectively, and sets all other state bits to zero.

3 Hardware Implementations

3.1 Hardware Design Principles and Interface with the Environment

The design principles and assumptions followed by the hardware implementations:

1. Multi-functionality module. The system should include all supported operations in a single module (Figure 3), because lightweight applications cannot afford the extra area for separate modules.
2. **Single input/output ports.** In small devices, ports can be expensive. To ensure that ACE and WAGE are not biased in favour of the system, at the expense of the environment, the ciphers have one input and one output port (Table 2). That being said, the authors agree with the proposed lightweight cryptography hardware API’s [8] use of separate public and private data ports and will update implementations accordingly.

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 memory and write it to the system’s input port. The receiving entity must capture the data in a single clock cycle (Figure 4). In reality, the environment can stall as well. In the future, ACE and WAGE implementations will be updated to match the proposed lightweight cryptographic hardware API’s use of a valid/ready protocol for both input and output ports.

4. **Use a “pure register-transfer-level” implementation style.** In particular, use only registers, not latches; multiplexers, not tri-state buffers; synchronous, not asynchronous reset; no scan-cell flip-flops; clock-gating is used for power and area optimization.

Since both ACE and WAGE use a unified sponge duplex mode, they share a common interface with the environment (Table 2). The environment separates the associated data and the message/ciphertext, and performs padding if necessary. The domain separators shown in Figure 1 are provided by the environment and serve as an indication of the phase change for AEAD functionality. For $ACE-H-256$, the phase change is indicated by the change of the i_mode(0) signal, as shown in Table 3. The hardware is unaware of the lengths of individual phases, hence no internal counters for the number of processed blocks are needed.

| Input signal | Meaning                      |
|--------------|------------------------------|
| reset        | resets the state machine     |
| i_mode       | mode of operation            |
| i_dom_sep    | domain separator             |
| i_padding    | the last block is padded     |
| i_data       | input data                   |
| i_valid      | valid data on i_data         |

| Output signal | Meaning                      |
|---------------|------------------------------|
| o_ready       | hardware is ready            |
| o_data        | output data                  |
| o_valid       | valid data on o_data         |

| i_mode | Mode | Operation or phase |
|--------|------|--------------------|
| (1)    | (0)  |                    |
| 0 0    | ACE-E| Encryption         |
| 0 1    | ACE-D| Decryption         |
| 1 0    | ACE-H-256| Absorb   |
| 1 1    | ACE-H-256| Squeeze   |
| - 0    | WAGE-E| Encryption         |
| - 1    | WAGE-D| Decryption         |

| reset | top_level_module | i_valid |
|-------|------------------|---------|
| FSM   |                  |         |
| i_mode| i_data           | o_ready |
|       | i_dom_sep        |         |
|       | i_padding        |         |
|       | control          |         |
|       | datapath         |         |
|       |                  | o_valid |

Figure 3: Top-level module and interface

Figure 4: Timing diagram for ACE during encryption

The top-level module, shown in Figure 3, is also very similar for both ACE and WAGE. It depicts the interface signals from Table 2, with only slight differences in bitwidths. Figure 4 shows the timing diagram for ACE during the encryption phase of message blocks $M_5$ and $M_6$, which clearly shows the valid-bit protocol. The first five lines show the top-level interface signals and line six shows the value of the permutation counter $pcount$, which is a part of the ACE finite state machine (FSM) and keeps track of the 128 clock
cycles needed for one ACE permutation. After completing the previous permutation, the top-level module asserts \texttt{o\_ready} to signal to the environment that an ACE permutation just finished and new data can be accepted. The environment replies with a new message block \(M_5\) accompanied by an \texttt{i\_valid} signal. The hardware immediately encrypts, returns \(C_5\) and asserts \texttt{o\_valid}. This clock cycle is also the first round of a new ACE permutation and the \texttt{o\_ready} is deasserted, indicating that the hardware is busy. Figure 4 shows the ACE hardware remaining busy (\texttt{o\_ready} = 0) for the duration of one ACE permutation. When \texttt{pcount} wraps around from 127 to 0, the hardware is again idle and ready to receive new input, in this case \(M_6\). A few more details about the use of \texttt{pcount} will follow in Subsection 3.2. The interaction between the top-level module and the environment during the encryption phase of WAGE is very similar, with 111 clock cycles for the completion of one permutation. More significant differences for the interaction with the environment arise during loading, tag extract and of course ACE-\(\mathcal{H}\)-256.

### 3.2 ACE Datapath

Figure 5(a) shows the ACE datapath. The top and bottom of the figure depict the five 64 bit registers A, B, C, D and E, followed by the hardware components required for normal operation during permutation, absorbing, and replacing, which imposes input multiplexers controlled by the mode and the counter \texttt{pcount}. Similarly, the output multiplexers are needed to accommodate encryption/decryption and tag generation for ACE-AE-128 and squeezing for ACE-\(\mathcal{H}\)-256. Furthermore, the output is forced to 0 during normal operation. The registers A, C and E are split in half to accommodate inputs and outputs. The rest of Figure 5(a) shows one step of the ACE permutation (Algorithm 1). The rounds and steps always use the same hardware, but in different clock cycles, which forces the use of multiplexers inside the ACE permutation. The last row of multiplexers accommodates loading.

![Figure 5: The ACE module datapath and parallelization](image)

### 3.3 WAGE Datapath

Because of the shifting nature of the LFSR, which in turn affects loading, absorbing and squeezing, the WAGE datapath is slightly more complicated than the ACE datapath and hence is explained in two levels:
1. **wage_lfsr** treated as a black box in Figure 6 with \( p = 1 \) (no parallelization)
   - **wage_lfsr**: The LFSR has 37 stages with 7 bits per stage, a feedback with 10 taps and a module for multiplication with \( \omega \) (Table 1). The internal state of **wage_lfsr** is also the internal state \( S \) of **WAGE**.
   - **WGP** module implementing WGP: 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 using components such as multiplication and exponentiation to powers of two \([13,14]\). However, the WGP is not stored in hardware as a memory array, but rather as a net of \( \omega \) gates, derived and optimized by the synthesis tools.
   - **SB** module: The SB is implemented in unrolled fashion, i.e. as purely combinational logic, composed of 5 copies of \( R \), followed by a \( Q \) and the final two \( \text{NOT} \) gates (Table 1).
   - **lfsr_c**: The **lfsr_c** for generating the round constants was implemented in a 2-way parallel fashion. It has only 7 1 bit stages and two \( \text{XOR} \) gates for the two feedback computations.

2. Extra hardware for the **wage_lfsr** in sponge mode. Figure 7 shows details for stages \( S_0, \ldots, S_{10} \). The grey line represents the path for normal operation during the **WAGE** permutation. The additional hardware for the entire **wage_lfsr** is listed below, with examples in brackets referring to Figure 7.
   - The 64 bit **i_data** is padded with zeros to 70 bits, then fragmented into 7 bit **wage_lfsr** inputs \( D_k, k = 0, \ldots, 9 \), corresponding to the rate stages \( S_r \). For each data input \( D_k \) there is a corresponding 7 bit data output \( O_k \) (\( D_1, O_1 \) and \( D_0, O_0 \) in Figure 7).
   - 10 XOR gates must be added to the \( S_r \) stages to accommodate absorbing, encryption and decryption (\( \text{XORs} \) at stages \( S_9, S_8 \)).
   - 10 multiplexers to switch between absorbing and normal operation (\( \text{Amux1, Amux0} \) at \( S_9, S_8 \)).
   - An XOR and a multiplexer are needed to add the domain separator **i_dom_sep** (\( \text{Amux} \) at \( S_0 \)).
   - To replace the contents of the \( S_r \) stages, 10 multiplexers are added (\( \text{Rmux1} \) at stage \( S_0 \)).
As mentioned in Section 2, special care was given to the design of loading and tag-extract. The existing

extracted in a similar fashion as loading, but from the data output

new key. The state of stages

shows the key shifting through the LFSR stages in 9 clock cycles. The stages are shown in the second row

loading process is illustrated in Table 4, where \( \hat{D}_4 \) is loaded through input \( D_0 \), however, instead of storing \( D_0 \oplus S_8 \), the \( D_0 \) data is fed directly into \( S_8 \), i.e. the \( R\text{Lmux}0 \) disconnects the \( A\text{Mux}0 \) output. The remaining stages in this region are loaded by shifting, which requires the \( S\text{Bmux} \) at \( S_4 \). Note that there is no need to disconnect the two WGP, because they are automatically disabled by loading through \( D_9 \) and \( D_4 \), located at stages \( S_{36} \) and \( S_{18} \) respectively. The loading process is illustrated in Table 4, where \( K_t \) is the \( t \)th 7 bit block of the 128 bit key \( K \). Table 4 shows the key shifting through the LFSR stages in 9 clock cycles. The stages are shown in the second row of Table 4, and the values "-" in the table denote the old, unknown values that are overwitten by the new key. The state of stages \( S_8, \ldots S_0 \) after after loading is finished is shown in the last row. The tag is extracted in a similar fashion as loading, but from the data output \( O_k \) at the end of a particular loading region, e.g., the region \( S_0, \ldots, S_{16} \), loaded through \( D_3 \), is extracted through \( O_1 \). The longest tag extraction region is of length 9, which is the same as the longest loading region.

3.4 Hardware-Oriented Design Decisions

The design process for ACE and WAGE tightly integrated cryptanalysis and hardware optimizations. A few key hardware-oriented decisions are highlighted here; more can be found in the design rationale chapters of [4,5].

Functionally, it is equivalent for the boundary between phases to occur either before or after the permutation. For ACE and WAGE, the boundary was placed after the permutation updates the state register. This means that the two-bit domain separator is sufficient to determine the value of many of the multiplexer select lines and other control signals. All phases that have a domain separator of "00" have the same multiplexer select values. The same also holds true for "01". Unfortunately, this cannot be achieved for "10", because encryption and decryption require different control signal values, but the same domain separator. Using the domain separator to signal the transition between phases for encryption and decryption also simplifies the control circuit. For hashing, the change in phase is indicated by the \( \text{i_mode} \) signal.

In applications where the delay through combinational circuitry is not a concern, such as with lightweight cryptography, where clock speed is limited by power consumption, not by the delay through combinational circuitry, it is beneficial to lump as much combinational circuitry as possible together into a single clock cycle. This provides more optimization opportunities for the synthesis tools than if the circuit was separated by registers. For this reason, the ACE datapath was designed so that the input and output multiplexers, one round of the permutation, and state loading multiplexers together form a purely combinational circuit, followed by the state register.

4 Parallel Implementations

4.1 Parallelization in General

Both ciphers can be parallelized (unrolled) to execute multiple rounds per clock cycle, at the cost of increased area. In the top-level schematic in Figure 3, the dashed stacked boxes indicate parallelization. The FSM is parameterized with parameter \( p \) and used for un-parallelized (\( p=1 \)) and parallelized (\( p>1 \)) implementations. Other components are replicated to show \( p \) copies, with \( p=3 \) in Figure 3. Such a
representation is symbolic; parallelization is applied only to the permutation, not the entire datapath. The interface with the environment remains the same.

4.2 ACE

The \( p=1 \) un-parallelized ACE permutation performs a single round per clock cycle, which implies 8 clock cycles per step. Parallel, *i.e.* unrolled, versions perform \( p \) rounds per clock cycle, and were implemented for divisors of 8, *i.e.* \( p = 2, 4, 8 \). The ACE permutation could be parallelized further, *e.g.* two or more steps in a single clock cycle. Figure 5(b) shows the example \( p=4 \) for registers A and B, with \( p=4 \) copies of SB-64 connected in series. Each SB-64 has its own round constant \( \omega_k^p \), \( k = 0, \ldots, p - 1 \). The round vs. step multiplexers are still needed, and can be removed only for values of \( p \), that are multiples of 8. Also note the step constant indicated as \( \omega_k^{p-1} \). For \( p=4 \) a step is concluded in 2 clock cycles. However, this requires a modification to the lfsr \( c \), which must now generate \( p \cdot 3 \) round constant bits \( \omega_{j}^k \), \( j = 0,1,2 \), \( k = 0, \ldots, p - 1 \) per clock cycle. The last cycle within a step requires 7 additional bits, which together with \( \omega_{j}^{p-1} \) yield 10 bits for the step constant generation \( \omega_k^{p-1} \). In the case \( p=4 \) the lfsr \( c \) must generate 12 constant bits in the first cycle and 19 constant bits in the second clock cycle of the step, which are then used for \( \omega_{j}^2 \) and \( \omega_{j}^3 \). For the extra constant bits, the lfsr \( c \) feedback was replicated, *i.e.* \(( p-1) \cdot 3 \) feedbacks in addition to the original 3.

4.3 WAGE

WAGE performs one clock cycle for the interaction with the environment, *i.e.* absorbing or replacing the input data into the state, followed by 111 clock cycles of the WAGE permutation. Because 111 is divisible only by 3 and 37, the opportunities to parallelize WAGE appear rather limited. However, by treating the absorption or replacement of the input data into the internal state as an additional clock cycle in the permutation, we increase the the length of the permutation to 112 clock cycles. Because 112 has many divisors, this allows parallelism of \( p = 2, 3, 4, 6, 8 \). The cost is a less than 1% decrease in performance for the additional clock cycle and some additional multiplexers, because the clock cycle that loads data has different behaviour than the normal clock cycles.

Figure 8 shows the 3-way parallel wage_lfsr including all nonlinear components and their copies. Multiplexers are not replicated, and hence, are not shown. For the components \( f, r, \text{WGP and SB} \) in Figure 8, the superscript \( k \) indicates the original \( k = 0 \) and the two copies \( (k = 1, 2) \). Computation of the three feedbacks \( f^k \) is not shown but is conducted as \( f^k = S_{31+k} \oplus S_{30+k} \oplus S_{26+k} \oplus S_{24+k} \oplus S_{19+k} \oplus S_{13+k} \oplus S_{12+k} \oplus S_{8+k} \oplus S_{6+k} \oplus (\omega \otimes S_{0+k}) \). Similar to ACE, the generation of WAGE round constants \( \omega_k^1, \omega_k^0 \).
must be parallelized as well. For readability, the two WGP were labelled WGP^k, WGP_0^k, with WGP^0_0, WGP^0_1, WGP^0_2, WGP^0_3 being the original WGP's positioned at S_36, S_18, just like r_1, r_0. Similarly, the SBs were also labelled SB^k_j, j = 3, 2, 1, 0, in the decreasing order, i.e. SB^0_3 is the original SB with input S_34.

5 Implementation Technologies and ASIC Implementation Results

Logic synthesis was performed with Synopsys Design Compiler version P-2019.03 using the compile.ultra command and clock gating. Physical synthesis (place and route) and power analysis were done with Cadence Encounter v14.13 using a density of 95%. Simulations were done in Mentor Graphics ModelSim SE v10.5c. The ASIC cell libraries used were ST Microelectronics 65 nm CORE65LPLVT 1.25V, TSMC 65 nm tpfn65gpgv2od3 200c and tcbn65gplus 200a at 1.0V, ST Microelectronics 90 nm CORE90GPLVT and CORX90GPLVT at 1.0V, and IBM 130nm CMRF8SF LPVT with SAGE-X v2.0 standard cells at 1.2V.

Some past works have used scan-cell flip-flops to reduce area, because these cells include a 2:1 multiplexer in the flip-flop which incurs less area than using a separate multiplexer. Scan-cell flip-flops were not used because their use as part of the design would prevent their insertion for fault-detection and hence, prevent the circuit from being tested for manufacturing faults. Furthermore, chip enable signals were removed from all datapath registers, which are controlled by clock gating instead. This allows a further reduction of the implementation area.

![Diagram](image.png)

Figure 9: Area^2 vs Throughput

Throughput is measured in bits per clock cycle (bpc), and plotted on a log scale axis. The area axis is scaled as log(Area^2).

Throughput is increased by increasing the degree of parallelization (unrolling), which reduces the number of clock cycles per permutation round. For p=1, the area of WAGE (W-1) is less than that of ACE (A-1), because WAGE has 259 registers, compared to 320 for ACE. As parallelization is increased, WAGE’s area grows faster than ACE’s, because of the larger size of WAGE’s permutation. Going from p=1 to p=8 results
Table 5: Post-PAR implementation results

| Label | Tput [A/W-p] | ST Micro 65 nm | TSMC 65 nm | ST Micro 90 nm | IBM 130 nm |
|-------|-------------|---------------|------------|---------------|------------|
|       |             | A [GE] | f [MHz] | E [nJ] | A [GE] | f [MHz] | E [nJ] | A [GE] | f [MHz] | E [nJ] |
| ACE   |             |         |         |        |         |         |        |         |         |        |
| A-1   | 0.5         | 4250   | 720    | 27.9   | 4600   | 705    | 20.1   | 3660   | 657    | 62.2   |
| A-2   | 1           | 4780   | 618    | 18.4   | 5290   | 645    | 12.4   | 4300   | 628    | 35.8   |
| A-4   | 2           | 5760   | 394    | 15.1   | 6260   | 588    | 8.51   | 4900   | 484    | 25.4   |
| A-8   | 4           | 7240   | 246    | 11.4   | 8090   | 493    | 6.40   | 6170   | 336    | 19.4   |
| WAGE  |             |         |         |        |         |         |        |         |         |        |
| W-1   | 0.57        | 2900   | 907    | 20.0   | 3290   | 1120   | 13.0   | 2540   | 940    | 39.2   |
| W-2   | 1.14        | 4960   | 590    | 19.1   | 5310   | 693    | 10.6   | 4280   | 493    | 34.4   |
| W-3   | 1.68        | 5480   | 397    | 20.4   | 5930   | 527    | 10.7   | 4770   | 414    | 31.2   |
| W-4   | 2.29        | 6780   | 307    | 24.0   | 7460   | 387    | 12.1   | 5790   | 277    | 32.9   |
| W-8   | 4.57        | 12150  | 192    | 38.5   | 11870  | 204    | 19.9   | 9330   | 137    | 49.9   |

Note: Energy results done with timing simulation at 10 MHz.

in 1.72× area increase for ACE and 3.80× for WAGE on average. Optimality for WAGE reaches a maximum at $p=3$. For ACE, optimality continues to increase beyond $p=8$.

As can be seen by the relative constant size of the shaded rectangles enclosing the data points, the relative area increase with parallelization is relatively independent of implementation technology.

Table 5 represents the same data points as Figure 9 with the addition of maximum frequency (f, MHz) and energy per bit (E, nJ). Energy is measured as the average value while performing all cryptographic operations over 8192 bits of data at 10 MHz. As the ACE throughput increases, energy per bit decreases consistently, despite higher circuit area and, therefore, power consumption. However, this is not the case with WAGE. This phenomena can be explained by the higher relative area increase for WAGE which comes from the higher complexity of WGP with respect to SB-64. Connecting more WGPs in a combinational chain results in an exponential increase of the number of glitches, which drastically increases power consumption.

Table 6 summarizes the area on ST Micro 65 nm of the LWC submissions [3] that included synthesizable VHDL or Verilog code. Table 6 reports the area results obtained using the ST Micro 65 nm process and tool flow from this paper and the results reported in the submission. The various ciphers use different protocols and interfaces, sometimes provide different functionality (e.g., with or without hashing), and use different key sizes. As such, this analysis is very imprecise, but gives a rough comparison to ACE and WAGE results. As the LWC competition progresses and the hardware API matures, more precise comparisons will become possible. This preliminary analysis indicates that ACE and WAGE are among the smaller cipher candidates.

Table 6: Area of LWC candidates on ST Micro 65 nm (post-PAR)

| Cipher   | Module                                | This work | Reported in submission documents [3] |
|----------|---------------------------------------|-----------|-------------------------------------|
|          |                                       | Area (kGE) | Area (kGE) ASIC technology used     |
| Drygascon| drygascon128_lround_cycle              | 29.6      | N/A                                 |
| Gage     | gage1h256c224r008AllParallel           | 10.4      | N/A                                 |
| Lilliput-AE| lilliputael28v1_encryptdecrypt       | 9.9       | 4.2 theoretical estimate for 5 lanes |
| Remus    | remus_top                             | 7.4       | 3.6 TSMC 65nm                       |
| Subterranean| crypto_aead simple_axid_lite         | 6.5       | 5.7 FreePDK 45nm                    |
| Triadx   | triadxl                              | 1.5       |                                     |
| Thash    | thash1                                | 1.5       |                                     |


6 Conclusion

The goal of the ACE and WAGE design process was to build on the well studied Simeck S-Box and Welch-Gong permutation. The overall algorithms were designed to lend themselves to efficient implementations in hardware and to scale well with increased parallelism. ACE has a larger internal state: 320 bits, vs 259 for WAGE, but the ACE permutation is smaller than that of WAGE. This means the non-parallel version of WAGE is smaller than that of ACE, but as parallelism increases, WAGE eventually becomes larger than ACE. At 1 and 2 bits-per-cycle, the designs are relatively similar in area. A number of the NIST LWC candidate ciphers provided synthesizable source code. A preliminary comparison with these ciphers on ST Micro 65 nm indicates that ACE and WAGE are likely to be among the smaller candidates.

Acknowledgements

This work benefited from the collaborative environment of the Communications Security (ComSec) Lab at the University of Waterloo, and in particular discussions with Kalikinkar Mandal, Raghvendra Rohit, and Guang Gong.

References

[1] NIST Lightweight Cryptography https://csrc.nist.gov/Projects/Lightweight-Cryptography
[2] Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf
[3] NIST Lightweight Cryptography round 1 candidates https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
[4] M.D. Aagaard, R. AlTawy, G. Gong, K. Mandal, R. Rohit, “ACE: An Authenticated Encryption and Hash Algorithm — Submission to the NIST LWC Competition”, March 2019, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ace-spec.pdf
[5] M.D. Aagaard, R. AlTawy, G. Gong, K. Mandal, R. Rohit, “WAGE: An Authenticated Cipher — Submission to the NIST LWC Competition”, March 2019, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/wage-spec.pdf
[6] B. Rezvani, W. Diehl, “Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look”, Cryptology ePrint Archive, Report 2019/824, 2019.
[7] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.P. Kaps, K. Gaj, “CAESAR Hardware API.” Cryptology ePrint Archive, Report 2015/669, 2016.
[8] J.P. Kaps, W. Diehl, M. Tempelmeier, E. Homsirikamol, K. Gaj, “Hardware API for Lightweight Cryptography”, 2019
[9] R. AlTawy, R. Rohit, M. He, K. Mandal, G. Yang and G. Gong. sLiSCP: Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives. In SAC (2017), C. Adams and J. Camenisch, Eds., Springer, pp 129-150.
[10] G. Yang, B. Zhu, V. Suder, M.D. Aagaard, and G. Gong. The Simeck family of lightweight block ciphers. In CHES (2015), T. Güneysu and H. Handschuh, Eds., Springer, pp. 307-329.
[11] Y. Nawaz and G. Gong. The WG stream cipher. ECRYPT Stream Cipher Project Report 2005 33 (2005).
[12] Y. Nawaz, and G. Gong. WG: A family of stream ciphers with designed randomness properties. Inf. Sci. 178, 7 (Apr. 2008), 1903-1916.
[13] M.D. Aagaard, G. Gong, and R.K. Mota. Hardware implementations of the WG-5 cipher for passive RFID tags. In Hardware-Oriented Security and Trust (HOST), 2013, IEEE, pp. 29-34.
[14] Y. Luo, Q. Chai, G. Gong, and X. Lai. A lightweight stream cipher WG-7 for RFID encryption and authentication. In 2010 IEEE Global Telecommunications Conference GLOBECOM 2010 (Dec 2010), pp. 1-6.