RTL Implementations and FPGA Benchmarking of Three Authenticated Ciphers Competing in CAESAR Round Two

William Diehl and Kris Gaj
Department of Electrical and Computer Engineering, George Mason University
Fairfax, U.S.A.
e-mail: {wdiehl, kgaj}@gmu.edu

***Abstract*— Authenticated ciphers are cryptographic transformations which combine the functionality of confidentiality, integrity, and authentication. This research uses register transfer-level (RTL) design to describe selected authenticated ciphers using a hardware description language (HDL), verifies their proper operation through functional simulation, and implements them on target FPGAs. The authenticated ciphers chosen for this research are the CAESAR Round Two variants of SCREAM, POET, and Minalpher. Ciphers are discussed from an engineering standpoint, and are compared and contrasted in terms of design features. To ensure conformity and standardization in evaluation, all three candidates are implemented with an identical version of the CAESAR Hardware API for authenticated ciphers. Functionally correct implementations of all three ciphers are realized, and results are compared against each other and previous results in terms of throughput, area, and throughput-to-area (T/A) ratio. SCREAM is found to have the highest T/A ratio of these three ciphers in the Virtex-6 FPGA, while Minalpher has the highest T/A ratio in the Virtex-7 FPGA.**

Keywords- Authentication, cipher, cryptography, encryption, field programmable gate array

I. INTRODUCTION

AUTHENTICATED ciphers combine the functions of confidentiality, integrity, and authentication. Input to authenticated ciphers consists of a plaintext message, associated data AD (which may include, for example, a header or trailer of a packet used in communication protocols), a public message number N_{PUB}, and an optional secret message number N_{SEC}. The resulting ciphertext C, and optional encrypted N_{SEC}, are computed as a function of N_{PUB}, N_{SEC}, AD, message, and key. This transformation ensures the confidentiality of the transaction. At the conclusion of plaintext encryption a tag is produced which is a keyed-hash function computed from all blocks of the AD and plaintext, as well as N_{PUB}, N_{SEC}, and key. The tag is appended to the end of the ciphertext to assure and verify the integrity and authenticity of the transaction. Decryption of the ciphertext and optional encrypted N_{SEC} is conducted in a similar fashion. Identical parameters, including AD, key, and message numbers, are required for validation. *Tag'* is then computed as above, and checked against the concatenated *Tag*. If *Tag* = *Tag'* then authentication and integrity of the transaction are assured; otherwise the decrypted ciphertext is not released. If authenticity and integrity are verified, the outputs of the transaction are the AD, plaintext, and optional decrypted N_{SEC}. A notional authenticated cipher is depicted in Figure 1.

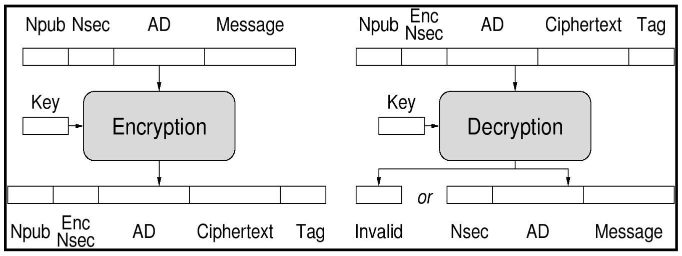


Fig. 1. Notional authenticated encryption.

In July 2015, CAESAR, the Competition for Authenticated Encryption: Security, Applicability, and Robustness, announced the candidates selected for advancement to Round Two, including SCREAM, POET, and Minalpher [1]. Specification updates (so called "tweaks") were permitted before the beginning of Round Two, and a hardware implementation is additionally required close to the conclusion of this round. This research implements SCREAM, POET, and Minalpher using Round Two specifications and compares them in terms of throughput, area, and throughput-to-area (T/A) ratio on two hardware platforms – the Virtex-6 and Virtex-7 FPGAs. Since SCREAM, POET and Minalpher all leverage Liskov, Rivest and Wagner's Tweakable Block Cipher (TBC) as a common building block, there is a strong basis for comparison among the implementations of these three ciphers [2].

II. PREVIOUS WORK

A. *Hardware evaluation in earlier competitions*

Hardware efficiency has been an evaluation criteria in several cryptographic competitions in the last two decades, including Advanced Encryption Standard (AES), eSTREAM, and the new secure hash function (SHA-3).

Hardware implementations are often examined upon down-selection to later rounds, as the amount of work to evaluate all early round candidates for hardware efficiency can be daunting. Yet hardware evaluations are valuable, as algorithms that perform well in software sometimes perform poorly, or require excessive area or complexity, in hardware.

Many CAESAR Round One submissions included results from a hardware implementation, either in FPGA or ASIC. Typically, the submitters evaluate their own candidate against the existing or de-facto standards, such as Triple DES, SHA-2, or AES-GCM, in the AES, SHA-3, and CAESAR contests, respectively. For example, the SCREAM and Minalpher Round One submissions contain evaluations of this sort [3, 4]. Several single-candidate evaluations of hardware implementations by CAESAR submitters were presented at Directions in Authenticated Ciphers (DIAC) workshops in 2014 and 2015. Additionally, there have been several third-party implementations of single CAESAR authenticated ciphers in hardware, such as POET, as described in [5].

Research in which third parties design and implement a significant set of ciphers from published specifications and compare them using a common hardware interface tends to emerge during the latter rounds of cryptographic competitions, as the number of candidates dwindles. For example, several large-scale hardware comparisons of SHA-3 candidates were presented at the Second SHA-3 Candidate Conference in 2010. Examples of two comparisons of all 14 candidates are [6] and [7]. More recently, a comparison of four CAESAR Round One candidates (ICEPOLE, Prøst, Tiaoxin-346, and Silver) in FPGAs was conducted in [8]. Large-scale (i.e., 10 or more ciphers) comparisons of CAESAR Round One candidates were also conducted as part of the foundational research of [9] and [10].

B. *Previous Implementations of Round Two Tweaks of CAESAR Candidates*

FPGA implementations of SCREAM Version 3, POET Version 2.01, and Minalpher Version 1.1 are available at [11], [12], and [13], respectively. In general, the main contribution of this research beyond what is accomplished in previous implementations is that all three ciphers are embedded in a universal hardware interface – the CAESAR Hardware Applications Programming Interface (API) for authenticated ciphers [14, 15]. This interface provides flexible and comprehensive input-output (I/O) functionality, services common to multiple authenticated ciphers, and a fair and common basis for evaluation of the ciphers against one another in terms of area and performance. The implementations in [11, 12, 13] will be further compared and contrasted to our implementations in subsequent sections, after necessary background has been made available to the reader.

III. DISCUSSION OF AUTHENTICATED CIPHERS

A. *SCREAM*

SCREAM (Side-Channel Resistant Authenticated Encryption with Masking) is an authenticated cipher built around Tweakable Authenticated Encryption (TAE) [16]. A unique tweak key, which is a function of the secret key, a nonce, and a counter (such as block number) is produced for each block encryption. Incomplete blocks of associated data are padded with a '1' in the most significant vacant bit, followed by '0' in the remaining bits (this padding scheme will be referred to as "10*" for the remainder of this article). The simplified block diagram for SCREAM is shown in Figure 2.

Each step begins with the computation of a tweak key TK_0. The 128-bit status word then goes through a substitution (S-Box) and a permutation (L-Box). In between, a round constant $RC(\rho,\sigma)$ is added to the status word. The ρ parameter is the round number, where $\rho \in \{0,1\}$, and the σ parameter is the step number, where $\sigma \in \{0,1,\ldots,N_S - 1\}$. After two rounds, the next tweak key TK_σ is added to the state variable prior to commencing the next step. The tweaks may either be pre-computed or computed "on-the-fly." We follow the reference implementation by computing tweaks "on-the-fly." Tweaks computed on the fly are simpler to implement, but incur a slight performance penalty.

There are two substantive updates to SCREAM in Round Two. The first is an update to the S-Box. The S-Box retains the "bitslice" construction used in Versions 1 and 2, but uses a slightly expanded set of new equations, in order to improve differential properties and algebraic degree [16].

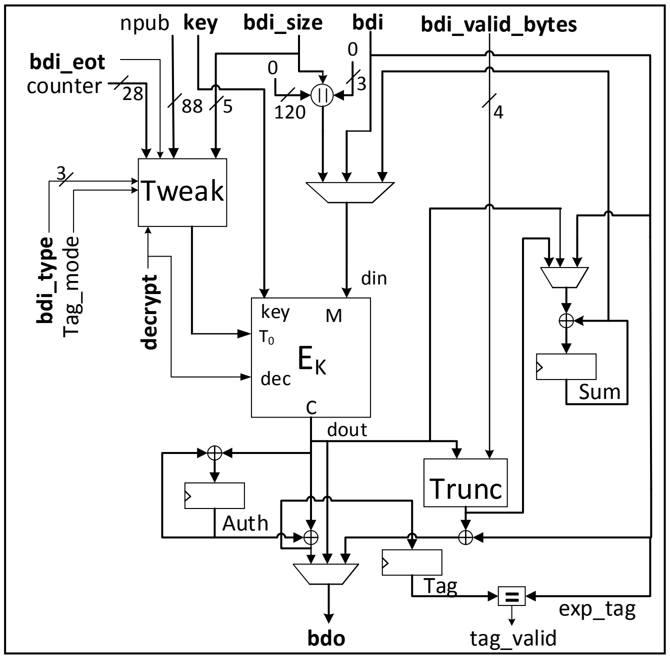


Fig. 2. SCREAM block diagram. All boldface type signals are described in the CAESAR Hardware API [15]. Bus width of thick wires is 128 bits unless indicated. Bus width of thin wires is one bit.

The Version 3 S-Boxes are "nearly involute;" therefore it is possible to reuse S-Boxes for encryption and decryption. This results in reduced area in terms of LUTs. The authors of [11] chose this method in their implementation. However, this comes at the cost of several 128-bit multiplexers, which increase the critical path. We found that the use of separate S-Boxes and Inverse S-Boxes provided a better T/A ratio, and chose this approach.

The second update is to the round constant. The round constant, previously XORed with the most significant byte of the status variable as $RC(\rho,\sigma) = 27(2\sigma+\rho) \bmod 2^8$ is now XORed with the most significant half-word (16 bits) of the status variable, as $RC(\rho,\sigma) = 2199(2\sigma+\rho) \bmod 2^{16}$.

In Versions 1 and 2, the multiplication by 27 was efficiently computed using shifts and additions, incurring a path delay of four shifts and three additions. In Version 3, multiplication by the constant 2199 incurs a path delay of six shifts and five additions. In this unrolled architecture, this adds eight levels of gate delay to the critical path. Therefore, the authors decided to instantiate precomputed look-up tables (LUT) for all possible values of $2199(2\sigma+\rho)$ for all values of σ up to $N_S = 12$. Tabulated round constants are additionally shared between encryption and decryption chains to save area.

B. POET

POET (Pipelineable On-line Encryption with Authentication Tag) is an authenticated cipher designed to be independent of a specific block cipher and hash function, i.e., the user is free to choose any block cipher or hash function which meets criteria identified in [17]. The cipher consists of a middle layer which uses the Electronic Code Book (ECB) block cipher mode sandwiched between upper and lower layers, which generate secret values that are added to the ECB layer. The upper ("top") and lower ("bottom") layer should contain ϵ-AXU (i.e., "Almost (XOR) Universal) hash functions keyed by pairwise independent subkeys. Figure 3 shows the POET simplified block diagram.

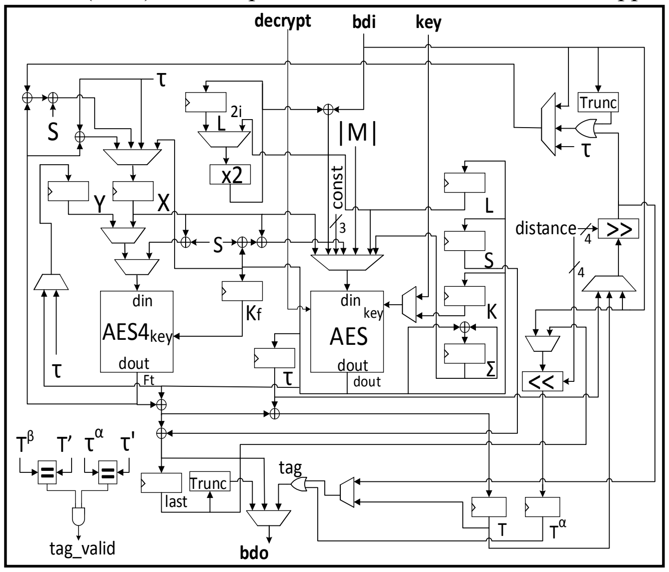


Fig. 3. POET block diagram. >> and << are 16-byte variable right shift and left shift, respectively. "x2" is the Galois Field (GF2) multiplier. Bus width of thick wires is 128 bits unless indicated. Bus width of thin wires is one bit.

The POET authors introduced significant changes in their Round Two tweaks, beginning with POET Version 2.0. The most significant changes include reducing the number of required subkeys from five to three and elimination of field multiplications by factors of 3 or 5 for the final header block. The authors also provide optional support for intermediate tags, which is a potentially useful feature for high-speed networks, but is not required by the CAESAR competition. Specifically, this research implements POET using the authors' recommended configuration of AES-128 for main encryption, AES-4 for the ϵ-AXU hash function, and no intermediate tags, i.e. $l_s = 0$, $l_t = 0$. The source code for the AES-128 implementation used in this research is available at [18].

Prior to commencing operation, POET produces three subkeys computed from key constants using AES-main. The K subkey is used for all AES-main operations. Blocks of AD are first masked by multiples of subkey L. All multiplications are Galois Field multiplications and are defined on GF (2^{128}) using the Galois Counter Mode (GCM) polynomial: $x^{128} + x^7 + x^2 + x + 1$. The AD block is then encrypted using AES-main and summed to a running hash value. The final block of associated data is padded using 10* padding. The hash is then encrypted to form the authentication value τ.

In their Round Two specification update, the POET authors eliminated the separate subkeys for AES-Top and AES-Bottom (Lf_{TOP} and Lf_{BOT}, respectively), and introduced a common subkey Kf. This opened the possibility of using a total of two AES cores, one 10-round AES main (consisting of encryption and decryption) and one "AES-4" core, consisting of encryption only. Since AES cores consume significant resources in the design, elimination of one core can produce significant area savings. For example, a single AES-4 core requires 1439 LUTs on the Virtex-6. However, there is increased cost in switching multiplexers in the datapath, and additional states and control signals required in the controller, since multiple functions must now time-share the AES hash core.

AES-top and AES-bottom registers X and Y are initialized with τ and $\tau \oplus 1$, respectively. AES-4 hash is keyed with a single subkey, Kf. Subsequently, the output of AES-top is added to blocks of plaintext M_i as the input to AES-main. The output of AES-main is combined with the output of AES-bottom to produce the resulting ciphertext block C_i. The input to AES-main and the output from AES-main are used as the subsequent inputs for AES-top and AES-bottom, respectively.

In the computation of the final block, the least significant $128 - |M_N|$ bits of the plaintext are filled with the most significant $128 - |M_N|$ bits of τ. This implementation accomplishes this merge by using variable shifters. Note that this requires run-time determination of the number of bits of τ to be included in M_N, which is why the shifts must be variable. There are a total of five signals which require variable shifters (three right-shifts and two left-shifts). This implementation multiplexes all inputs into either the right or left shifter, as appropriate. This provides significant area savings, as these 16-byte variable shifters consume about 200 LUTs each.

The plaintext-to-ciphertext computation of the final block is masked at both ends using the value of S, or $E_K(|M|)$. Note that the output is always a full 128-bit block of ciphertext. This final block must be apportioned between ciphertext C_N and tag T. The amount of the final ciphertext block that will be used as "tag" is equal to the amount of bits that were borrowed from τ to fill the final block of plaintext. Therefore, the final ciphertext block consists of $C_N||T^{\alpha}$.

The tag is computed using the same sequence as the final ciphertext block, except that full τ value is used as the input to the chain. The output consists of a full 128-bit word. However, the $|M_N|$ bits of the tag computation, denoted T^{β}, are extracted to form the least significant $|M_N|$ bits of the tag. Therefore, the formal "tag" consists of $T^{\alpha}||T^{\beta}$. The remainder of the tag computation is discarded.

In contrast to our implementation, the implementation at [12] uses three AES cores, all of which are 10-round AES-128. The implementation in [12] has an AES S-Box using combinational logic to perform inversions based on composite field arithmetic in subfields of GF(2^8). While such an implementation is desirable for ASIC, it is less efficient than an AES using LUTs in FPGA. For example, an AES using composite field inversion is 20% larger in terms of LUTs and has a critical path 80% longer than the equivalent AES version using LUTs, according to our measurements.

The most significant difference between [12] and our implementation is that the implementation in [12] provides the value τ to an output, but does not attempt to perform tag generation or verification. As seen above, the main complexity in POET is tag generation and verification, which requires variable shifts and truncations for all but the simplest cases. Given this large delta in functionality, a direct comparison between our results and [12] is difficult.

C. Minalpher

Minalpher also leverages Tweakable Block Ciphers (TBC). Specifically, Minalpher defines a cryptographic core called the Tweakable Even-Mansour (TEM). The core of the TEM is the Minalpher-P computation [19]. The Minalpher-P consists of four transformations. They include "S" *SubNibbles*, which is a four-bit S-Box (designed for embedded applications); "T" which is *ShuffleRows* and *SwapMatrices*; "M" which is *MixColumns* and *XorMatrix*; and "E" which is a Round Constant. The S, E, and M primitives are involute, whereas T has an inverse specified by T^{-1}. Therefore, the only difference between "forward" (encryption) and "backward" (decryption) operations are a reversal of the orders of the E and M calculations. A TEM computation consists of 17.5 Minalpher-P rounds; the "0.5" round is a final additional T and S calculation.

The tweak is produced for each call to TEM. Using a seed consisting of the secret key K, a flag (different for AD or message computation) and a 13-byte nonce, the tweak is calculated as a function of $y^i(y+1)^j L$, where L is an initial tweak value, y is a root of a composite GF polynomial, and i and j are variable exponents which increase by either 0, 1, or 2 with each new block. Multiplications in Minalpher occur on GF(2^{256}), which is a composite field formed by GF(2^8) using polynomial $x^8 + x^7 + x^5 + x + 1$ and GF(2^{32}) using the polynomial $y^{32} + y^3 + y^2 + x$. In this RTL implementation, the initial L is treated as a subkey and is computed by a Minalpher-P computation based on key, flag, and N_{PUB}. Subsequently, L is updated in every Minalpher-P round using the "*TweakGen*" function, which also computes the correct i and j exponents for the proper type of block (i.e., associated data, plaintext, full block, etc.) and provides them to TEM. In this methodology, only δi and δj are recomputed in each round, and therefore all *TweakGen* computations are combinational.

Minalpher uses 10* padding for both associated data and for plaintext processing. However, a Minalpher plaintext block must always have 10* padding, even if it is a full block. Therefore, if the last plaintext block is a full block, and there are N plaintext blocks, there will be $N+1$ ciphertext blocks. The Minalpher simplified block diagram is shown in Figure 4.

The associated data (AD) subkey is first computed using TEM. Next, each block of AD is processed through a TEM

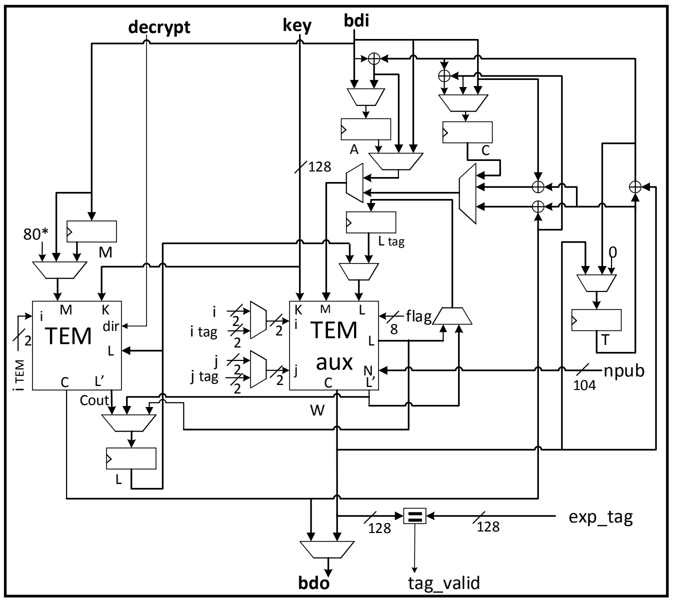


Fig. 4. Minalpher block diagram. All signals in boldface type are described in the CAESAR Hardware API [15]. Bus width of thick wires is 256 bits, except where indicated. Bus width of thin wires is one bit.

core. The result is added to a running hash which is retained at the conclusion of AD processing. Prior to the start of message processing, a new subkey (valid for both plaintext and tag processing) is computed by TEM. Plaintext tag processing in Minalpher requires two TEM cores for parallel operation. One core, "TEM," consists of forward and backward Minalpher-P functions. The second TEM, "TEM aux," is smaller in that it performs only forward transformations. During authenticated encryption, the first TEM core processes a plaintext block M_i into a corresponding ciphertext block C_i. The result C_i is then sent through a second TEM core, and the result is added to the running hash computed during AD processing. While the second TEM is computing the tag hash, the first TEM is freed to compute the next ciphertext block, C_{i+1}. After the computation of the final plaintext block M_N, the running hash is then added to the final plaintext block and is used as the input to the final TEM. The most significant 128 bits of the 256-bit TEM output are used as the tag. The authenticated decryption flow is slightly simpler, in that each block of ciphertext C_i can be simultaneously entered into the two TEM cores. Therefore, the tag is produced simultaneously upon the decryption of the final block of ciphertext. Note that this parallel operation still requires two distinct TEM cores, since the tweak parameters i and j for ciphertext decryption and tag computation are not identical.

In Minalpher, all ciphertext blocks are full blocks (i.e., 32 bytes long). Therefore, the decryption processor does not know the length of the final plaintext block M_N until after decryption is complete. One way to ensure the proper length of M_N is to use a parallel "truncator," which checks all of 32 possible locations of the original 10* padding, and eliminates it from the output. This method executes in a single clock cycle, but is area-intense, and could increase the critical path if not pipelined. A second method is to cycle through all possible combinations of M_N until the 10* padding is located, one byte at a time. This method has a low-area requirement and minimal effect on the critical path, but requires up to 32 clock cycles to process M_N. We have chosen the latter method for this implementation.

In contrast to our design, the Minalpher implementation described in [13] uses both straightforward and four-state pipelined implementations of the Minalpher-P transformation. While both the straightforward and pipelined implementations achieve very high throughput (i.e., 2 – 3 times that of AES-GCM), it is not clear to what extent the full authenticated encryption is implemented, as there is no discussion of top-level block diagram, interface, or test cases.

D. *Hardware API for Authenticated Ciphers*

Authenticated cipher implementations in this research use the CAESAR Hardware API for authenticated ciphers [14, 15]. Based on the George Mason University (GMU) Hardware Applications Programming Interface (API) for Authenticated Ciphers, the CAESAR Hardware API was adopted by the CAESAR committee as the interface standard for hardware implementations of CAESAR authenticated cipher candidates in May 2016 [20]. This input-output (I/O) interface provides a common, flexible, and practical I/O standard and is especially applicable for the fair evaluation of large numbers of hardware design candidates, as the employment of custom interface solutions causes wide variations in throughput and area calculations, and makes it more difficult for evaluators to judge efficiencies based on a standard baseline.

The authenticated cipher interface provides input and output processors, which receive data, parse data into applicable segments to be used by ciphers, and format output. Serial-input/parallel-output (SIPO) and parallel-input/serial-output (PISO) allow for fewer I/O connections (e.g. 32 bits) for public and secret data interfaces, which expands the available FPGA platforms on which designs can be implemented. For example, all three designs in this research use a 32-bit bus width for public data interface (PDI), secret data interface (SDI), and data output (DO). The total I/O pin requirement during our implementations is 107 – well within the pin limits of low-end FPGAs.

The interface also provides services common to many cryptographic operations that would otherwise be costly to implement. One example is 10* padding. All three ciphers in this research use 10* padding for associated data, and for plaintext in the case of Minalpher. In the cases of POET (associated data) and Minalpher (plaintext), final blocks must terminate with 10*, which sometimes results in the creation of an extra block. The padding and ciphertext expansion modes of the CAESAR Hardware Interface efficiently create extra blocks when necessary which reduces both datapath and controller complexity. A second example is truncation of plaintext or ciphertext output during final block processing. Both SCREAM and POET

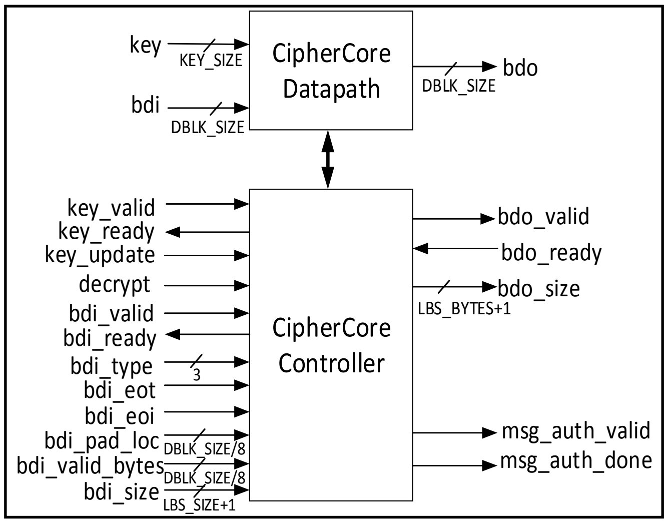


Fig. 5. CAESAR Hardware API for authenticated ciphers – Interface between CipherCore and I/O Processors. Further details in [15].

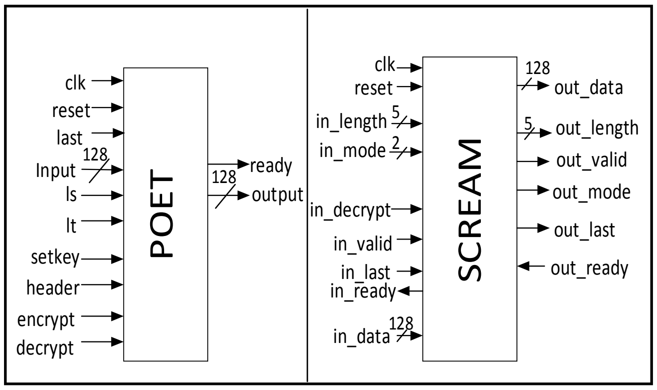
require varying levels of truncation. Truncation can be costly to realize in hardware when the truncation length is not known at synthesis time. However, it is efficiently computed using the *bdi_valid_bytes* input from the interface at the cost of only one 128-bit XOR.

The CAESAR Hardware API for authenticated ciphers is "universal," in that it allows for specific functionality required in all CAESAR Round Two candidates. Designers of high-speed implementations should insert their cipher core into "CipherCore Datapath" and "CipherCore Controller" in order to use the corresponding interface. Note that this requires a robust controller design, which can implement all the features required in authenticated encryption and decryption, and interpret and interact with interface control signals, including those driving external I/O. The interface to the authenticated cipher core (CipherCore) is shown in Figure 5. Specifications for signals and additional features of this hardware API are available in [15].

In contrast to our implementations, other previous CAESAR implementations use their own custom-designed interfaces, with no standardization and with varying degrees of complexity. The top-level POET interface used in [12], and SCREAM interface in [11], are shown in Figs. 6a and 6b, respectively. The top-level interface used in the Minalpher implementation at [13] is not known, as neither interface nor source codes are available.

For example, the interface for POET used in [12] provides significantly less functionality than our implementation. It provides only one output port, meaning that the details of tag generation and verification are left to a higher level of protocol. Likewise, it does not provide the core information on the word length of the final block. This is a critical parameter in POET, which determines the required variable shifts in tag generation and verification.

The SCREAM interface described in [11] is more complex, uses I/O protocol designed to mirror AXI4-stream protocol, includes a 5-bit "length" field, which enables an embedded controller to process partial blocks, and is arguably adequate for the full range of SCREAM functionality. However, the interface does not include a



Figs. 6a (POET) and 6b (SCREAM) top-level interfaces, based on analysis of source codes in [11] and [12].

mechanism for tag verification in authenticated decryption.

Our point is not to show the adequacy or inadequacy of individual cipher interfaces, but rather to show that fair comparison of authenticated ciphers with interfaces that allow varying degrees of functionality is not practical. Our implementation of all candidates in a standardized, robust, and flexible interface levels the playing field and is essential for wide-ranging hardware evaluations in competitions such as CAESAR.

IV. Design Methodology

The implementations of the three RTL candidates in this research were developed using the published specifications submitted for CAESAR Round Two, and include any applicable Round Two tweaks. Functional verification of correct results was accomplished for all three ciphers using Xilinx iSim. Test vectors were generated using a combination of manually generated vectors from the Reference C implementations (modified as applicable for Round Two specifications), and the Authenticated Encryption Test Vector generation script *aeadtvgen* [15, 21]. The implementations were coded using VHDL; synthesis and implementation were performed using Xilinx WebPack 14.7 Integrated Synthesis Environment (ISE). All results are optimized using the ATHENa optimization tool, using the ATHENa "GMU_Optimization_1" algorithm with a balanced approach designed to optimize throughput/area ratio under the assumption of no use of BRAMs [22]. Implementation results are specified for the Xilinx Virtex-6 (xc6vlx240tff1156–3) and Virtex-7 (xc7vx485tffg1761–3).

V. Results

A. *Direct comparison of the three ciphers*

The results for the three RTL implementations are described below. These RTL implementations meet all the minimum compliance criteria for hardware implementations of CAESAR candidates [14]. Namely, they are required to check for "boundary conditions," such as null data (i.e., empty associated data or message), support multiple key activations, continuously process messages (i.e., complete one message and immediately proceed to the next), process full and partial final blocks, support message lengths up to $2^{32} - 1$ bytes, and to support conditions which "stall" the processor, such as input or output "not ready" conditions [14]. Adherence to minimum compliance criteria means that Algorithmic State Machine (ASM) controllers can be significantly larger and more complex than controllers supporting simple encryption or decryption.

TABLE I
FORMULAS FOR EXECUTION TIME AND THROUGHPUT FOR LONG MESSAGES

	SCREAM	POET	Minalpher
Key Setup	0	50	38
Encryption	11a + 11n + 11	59 + 11a + 10n	19a + 19n + 19
Decryption	11a + 11c + 11	59 + 11a + 10c	19a + 19c + $\|M_N\|$
Throughput	(128 / 11) * f_{CLK}	(128 / 10) * f_{CLK}	(256 / 19) * f_{CLK}

Explanation of Table 1 calculations: *a* is number of AD blocks; *n* is number of plaintext blocks, and *c* is the number ciphertext blocks. Key setup time includes AES round key initialization and subkey generation. The constant in encryption and decryption fields includes tag generation and other factors which are non-recurring. $|M_N|$ refers to the number of bytes of the final decrypted plaintext block. Throughput is calculated for "long messages," for which the dominant component is assumed to be number of plaintext or ciphertext blocks.

The three implementations all use the same version of the CAESAR Hardware API, which can increase the area by up to 25%, but usually does not increase the clock period [10]. All implementations use the AEAD_Wrapper configuration, which includes all cipher core (i.e., datapath and controller) functions, pre (input), and post (output) processors, and ensures the ability to compute a minimum clock period even when I/O pin requirements exceed the target device maximum

The formulas used to calculate throughput for long messages are shown in Table 1.

Implementation statistics in the Virtex-6 and Virtex-7 FPGAs are shown together in Table 2. These results are compared using Throughput / Area (T/A) ratio as the primary metric. Throughput is defined as 10^6 bits/second (Mbps), and area is defined in terms of LUTs (although area can also be defined in terms of slices). By this metric, SCREAM has the highest T/A ratio on the Virtex-6 at 0.441, followed by Minalpher at 0.433, and POET at 0.336. However, Minalpher has the highest T/A on the Virtex-7 at 0.524, followed by SCREAM at 0.459, and POET at 0.366.

Although there is a "split decision" in terms of the highest T/A ratio among the two FPGAs, SCREAM is consistently the smallest in terms of area, with 37% and 49% of the LUTs of POET and Minalpher, respectively. Conversely, POET and Minalpher both have more than twice the throughput of SCREAM and are nearly on par with each other. However, Minalpher dominates over POET in terms of T/A since it has only 76% the amount of LUTs of POET. The results are recorded in the "Database of FPGA Results for Authenticated Ciphers" at the link in [23].

TABLE II
RESULTS OF IMPLEMENTATION IN VIRTEX-6 (V-6) AND VIRTEX-7 (V-7)

	SCREAM		POET		Minalpher	
FPGA Platform	V-6	V-7	V-6	V-7	V-6	V-7
Registers	1545	1545	3822	3823	3966	3966
LUTs	2898	2906	7805	7791	5955	5954
Slices	1057	834	2509	2109	1668	1608
T_{CLK} (ns)	9.09	8.72	4.88	4.49	5.23	4.32
f_{CLK} (MHz)	110.0	114.7	205.0	222.7	191.2	231.6
Throughput(Mbps)	1280	1334	2625	2851	2576	3120
Throughput/Area	0.441	0.459	0.336	0.366	0.433	0.524

B. Comparison with other Round Two FPGA implementations

Direct comparison with previous results is generally difficult, since other implementations either have less functionality, have significantly different interface standards and assumptions, use different architectural strategies, or lack published source codes.

In the case of SCREAM, we were able to synthesize the source codes available at [11] on Virtex-6 hardware and perform a rough comparison. This implementation used 2672 LUTs, with a minimum clock period of 9.9 ns (equivalent to 101.1 MHz clock frequency). This is nearly analogous to our own implementation of SCREAM's cryptographic core using shared S-Boxes, which required 2304 LUTs, but less than our full authenticated cipher, which requires about 2900 LUTs. The constructions are remarkably similar; the differences are explained by our addition of the CAESAR Hardware API for authenticated ciphers, the registers required to store intermediate results, and the more complex controller in our implementation.

We were also able to synthesize the POET implementation at [12]. This implementation requires 6254 LUTs in the Virtex-6, and has a minimum clock period of 7.83 ns (equivalent to 127.6 MHz clock frequency). However, 12 cycles are required between computation of blocks, vice the 10 cycles in our implementation, which leads to a throughput of 1361 Mbps. The area of 6254 LUTs is less than our implementation, which requires 7805 LUTs. However, our implementation is significantly more functional, in that it includes the CAESAR Hardware API for authenticated ciphers, and performs full tag generation and verification, which require several hardware-intense variable shifts.

The Minalpher implementation at [13] includes comparisons of dual architectures (straightforward versus four-state pipelined) against AES-GCM on multiple FPGAs. Its stated frequencies for the Virtex-6 of 479 MHz (straightforward) and 599 MHz (pipelined) outperform our design. However, it is not clear that full authenticated encryption and decryption are performed on test vectors conforming to the CAESAR minimal compliance standards, as only the implementation of the Minalpher-P transformation is discussed. Additionally, the stated Virtex-6 area usage of 2925 LUTs (straightforward) and 3033 LUTs (pipelined) are about 50% of the area of our implementations, and thus are likely to lack functionality found in our design. Finally, there is no reference to the top-level interface, and source codes are not available for examination.

VI. CONCLUSION

These three CAESAR Round Two candidates, SCREAM, POET, and Minalpher, were successfully implemented using RTL design in VHDL on the Virtex-6 and Virtex-7 FPGAs. SCREAM was found to have the highest T/A ratio on the Virtex-6 FPGA, followed by Minalpher, and then POET. However, Minalpher had the highest T/A ratio on the Virtex-7 FPGA, followed by SCREAM, and then POET.

All three ciphers were evaluated using an identical version of the CAESAR Hardware API for authenticated ciphers. The use of a standardized interface is critical to ensuring the fair and efficient evaluation of large numbers of hardware implementations of authenticated cipher candidates.

In all three ciphers, the influence of control signals on the critical path has been nearly eliminated by computing control inputs and registering them before they are used. In all cases, this increases the number of required registers, the number of control signals, the number of control states, and the complexity and size of the controller. Controllers for authenticated ciphers are inherently very complex – much more so than the controllers for symmetric block ciphers or secure hash functions.

VII. Areas for Further Research

In all three of these implementations, design choices were made involving trade-offs between logic and routing delay, critical path, the number of control signals and controller states, the number of registers, and pipelining techniques, and area. It is possible that improved performance with reduced area could be achieved by a more optimal combination of the above factors. Such techniques could be explored by the authors of the ciphers or by third parties to produce more efficient implementations. For example, analysis of results at [11] suggests that a SCREAM architecture which computes one round per clock cycle might have a higher T/A ratio than our architecture which computes two rounds per clock cycle. The authors intend to investigate alternative architectures and post results at [23].

References

[1] "CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness." Internet: http://competitions.cr.yp.to/caesar.html, Jun. 16, 2014 [Jun. 23, 2016].

[2] M. Liskov, R. Rivest, and D. Wagner, "Tweakable Block Ciphers," *Journal of Cryptology*, Vol. 24, No. 3, Jul. 2011, pp. 588-613.

[3] V. Grosso, G. Leurent, F. Standaert, K. Varici, F. Durvaux, L. Gaspar, S. Kerckhof, "SCREAM & iSCREAM, Side-Channel Resistant Authenticated Encryption with Masking," presented at Directions in Authenticated Ciphers (DIAC 2014), 23 – 24 Aug. 2014, Santa Barbara, CA.

[4] Y. Sasaki, Y. Todo, K. Aoki, Y. Naito, T. Sugawara, Y. Murakami, M. Matsui, S. Hirose, "Minalpher" presented at Directions in Authenticated Ciphers (DIAC 2014), 23 – 24 Aug. 2014, Santa Barbara, CA.

[5] A. Moradi, "A hardware implementation of POET," University of Bochum, Germany, Feb. 2014.

[6] E. Homsirikamol, M. Rogawski, K. Gaj, "Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs," Dec. 21, 2010, Internet: https://eprint.iacr.org/2010/445.pdf [Jun. 23, 2016].

[7] M. Knezevic, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, U. Kocabas, F. Junfeng, T. Katashita, T. Sugawara, L. Sakiyama, I. Verbauwhede, K. Ohta,N. Homma, T. Aoki, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on (Vol. 20, Issue 5), pp. 827-840, Apr. 29, 2011.

[8] C. Arnould, "Towards Developing ASIC and FPGA Architectures of High-Throughput CAESAR Candidates," MS Thesis, Swiss Federal Institute of Technology in Zurich (ETHZ), Mar. 2015.

[9] E. Homsirikamol, W. Diehl, F. Farahmand, A. Ferozpuri, K. Gaj, "C vs. VHDL: Benchmarking CAESAR Candidates Using High-Level Synthesis and Register-Transfer Level Methodologies", presented at Directions in Authenticated Ciphers (DIAC) 2015, Singapore, Sep. 27, 2015, Internet: http://www1.spms.ntu.edu.sg/~diac2015/slides/diac2015_22_hls.pdf [Jun. 23, 2016].

[10] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M.U. Sharif, and K. Gaj, "A Universal Hardware API for Authenticated Ciphers," 2015 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2015, Mayan Riviera, Mexico, Dec. 7-9, 2015.

[11] S. Kerckhof, L. Gaspar, SCREAM Version 3, Université Catholique de Louvain, Dec. 13, 2015, available at http://www.uclouvain.be/crypto/static/SCREAM/2015_12_13_Scream_TAE_HW_reference.zip [Jun. 23, 2016].

[12] A. Moradi, "A Hardware Implementation of POET 2," Ruhr-Universität Bochum, Germany, https://www.uni-weimar.de/de/medien/professuren/mediensicherheit/research/poet/ [Jun. 23, 2016].

[13] M. Kosug, M. Yasuda, A. Satoh, "FPGA implementation of authenticated encryption algorithm Minalpher," 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE), pp. 572- 576, 20-30 Oct. 2015, Osaka, Japan.

[14] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.P. Kaps, K. Gaj, "CAESAR Hardware API," Cryptology ePrint Archive, Report 2016/626, Internet: http://eprint.iacr.org/2016/626.pdf [Jun. 22, 2016].

[15] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, K. Gaj "Implementer's Guide to the CAESAR Hardware API v1.1" available at https://cryptography.gmu.edu/athena/index.php?id=download

[16] V. Grosso, G. Leurent, F. Standaert, K. Varici, A. Journault, F. Durvaux, L. Gaspar, S. Kerckhof, "SCREAM, Side-Channel Resistant Authenticated Encryption with Masking," Version 3 (Second Round Specifications), Internet: http://competitions.cr.yp.to/round2/screamv3.pdf Aug. 2015 [Jun. 23, 2016].

[17] F. Abed, S. Fluhrer, J. Foley, C. Forler, E. List, S. Lucks, D. McGrew, J. Wenzel, "The POET Family of On-Line Authenticated Encryption Schemes," Version 2.01, Internet: https://www.uni-weimar.de/de/medien/professuren/mediensicherheit/research/poet/ Sep. 15, 2015 [Jun. 22, 2016].

[18] Cryptographic Engineering Research Group (CERG), "GMU Source Code of ASCON, AES, AES-HLS, and Keccak Permutation F", Jun. 20, 2016, Internet https://cryptography.gmu.edu/athena/index.php?id=download [Jun. 22, 2016].

[19] Y. Sasaki, Y. Todo, K. Aoki, Y. Naito, T. Sugawara, Y. Murakami, M. Matsui, S. Hirose, "Minalpher v1.1," Internet: http://competitions.cr.yp.to/caesar-submissions.html, Aug. 29, 2015 [Jun. 23, 2016].

[20] E. Homsirikamol, W. Diehl, F. Farahmand, A. Ferozpuri, M. U. Sharif, K. Gaj, "GMU Hardware API for Authenticated Ciphers," Cryptology ePrint Archive, Report 2015/669, Dec. 6, 2015, Internet: https://eprint.iacr.org/2015/669.pdf [Jun. 22, 2016].

[21] "SUPERCOP, European Network of Excellence in Cryptology II," Internet: (http://bench.cr.yp.to/supercop.html), Sep. 11, 2014 [Jun. 23, 2016].

[22] Cryptographic Engineering Research Group (CERG) at GMU, "Automated Tool for Hardware Evaluation (ATHENa)", Internet: https://cryptography.gmu.edu/athena/ [Jun. 29, 2016].

[23] "Database of FPGA Results for Authenticated Ciphers," George Mason University, Fairfax, U.S.A., available at https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/table_view