100 Gbit/s Authenticated Encryption Based on Quantum Key Distribution

Michael Muehlberghuber, Christoph Keller,
and Norbert Felber
ETH Zurich
Integrated Systems Laboratory (IIS)
Gloriastrasse 35, 8092 Zurich, Switzerland
Email: {mbgh, chrikell, felber}@iis.ee.ethz.ch

Christian Pendl
Graz University of Technology
Institute for Applied Information Processing
and Communications (IAIK)
Inffeldgasse 16a, 8010 Graz, Austria
Email: christian.pendl@student.tugraz.at

***Abstract*—We propose a block-cipher-based hardware architecture for authenticated encryption (AE) applications supporting the Ethernet standard IEEE 802.3ba. Our main design goal was to achieve high throughput on FPGA platforms. Compared to previous works aiming at data rates beyond 100 Gbit/s, our design makes use of an alternative block cipher and an alternative mode of operation, namely Serpent and the offset codebook mode of operation, respectively. Using four cipher cores for the encryption part of the AE architecture, we achieve a throughput of 133 Gbit/s on an Altera Stratix IV FPGA. The design requires 30 kALMs and runs at a maximum clock frequency of 260 MHz. This represents, to the best of our knowledge, the fastest full implementation of an AE scheme on FPGAs to date.**

***Index Terms*—Authenticated encryption, High-throughput architecture, FPGA, Pipelining, Serpent, OCB, AES, GCM.**

I. Introduction

Two of the most important cryptographic goals are confidentiality and authentication. Confidentiality assures that any eavesdropping adversary is unable to decipher the message, even if she has access to the transmission medium. Authentication refers to the cryptographic service that ensures that the receiver of a message can be sure of its origin, i.e., that an attacker has not impersonated the sender. Authenticated encryption (AE) combines these two services and allows a *secure* and *authentic* communication between two parties.

In order to provide high-throughput implementations for AE based on block ciphers, so-called *combined* modes of operation have been designed throughout the last decade. They allow a higher throughput by interleaving the authentication part and the encryption part instead of calculating them consecutively (as traditional AE methods do). The two most widely accepted AE modes of operation are CCM [1] and GCM [2]. Their acceptance is due to the fact that they have been recommended by the National Institute of Standards and Technology (NIST) (cf. [1] and [3]). Since then, they have been applied to technologies and protocols such as WiFi 802.11 [4] and IPsec [5]. Although the specifications of these modes do not determine the underlying block cipher, most applications make use of the Advanced Encryption Standard (AES) [6] since it is another algorithm standardized by the NIST.

In the present work, we describe a block cipher-based hardware architecture for AE, targeting high throughput on FPGA platforms. Our architecture has been developed as to fulfill the requirements of the Ethernet standard IEEE 802.3ba [7], which allows for transmission speeds of up to 100 Gbit/s. Our work has been designed as part of a system that employs quantum key distribution (QKD) for synchronizing multiple private key exchanges within one second, and provides authenticated encryption service using conventional cryptographic primitives. Fig. 1 illustrates the overall system setup. Throughout this paper, we focus on the *Authenticated Encryption* part of Fig. 1. So far, our system employed a common GCM-AES-based cryptographic primitive in order to achieve the required throughput.

In this work, we examine alternatives for both the block cipher as well as the mode of operation used and compare the performance of these alternatives to the established cryptographic primitives. Besides exploring more efficient hardware implementations, this work is also motivated by providing an alternative AE scheme, in case successful attacks are developed against the existing primitives. We evaluate the Serpent block cipher [8] and the offset codebook (OCB) mode of operation [9] and we provide results of hardware implementations for different mode of operation/block cipher combinations, namely:

- OCB-Serpent
- OCB-AES
- GCM-Serpent
- GCM-AES

Our fastest AE implementation is based on an OCB-Serpent architecture and requires 30 kALMs (Adaptive Logic Modules) on an Altera Stratix IV FPGA. It is based on four cipher cores

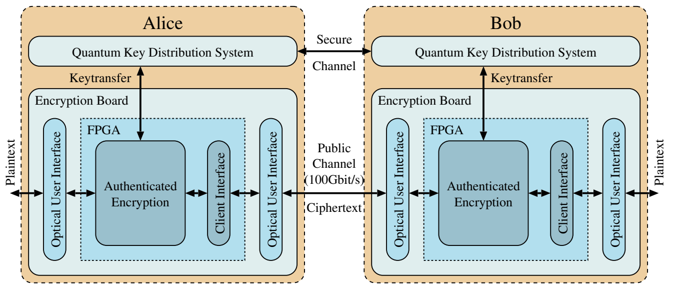

Fig. 1. High-speed authenticated encryption system setup.

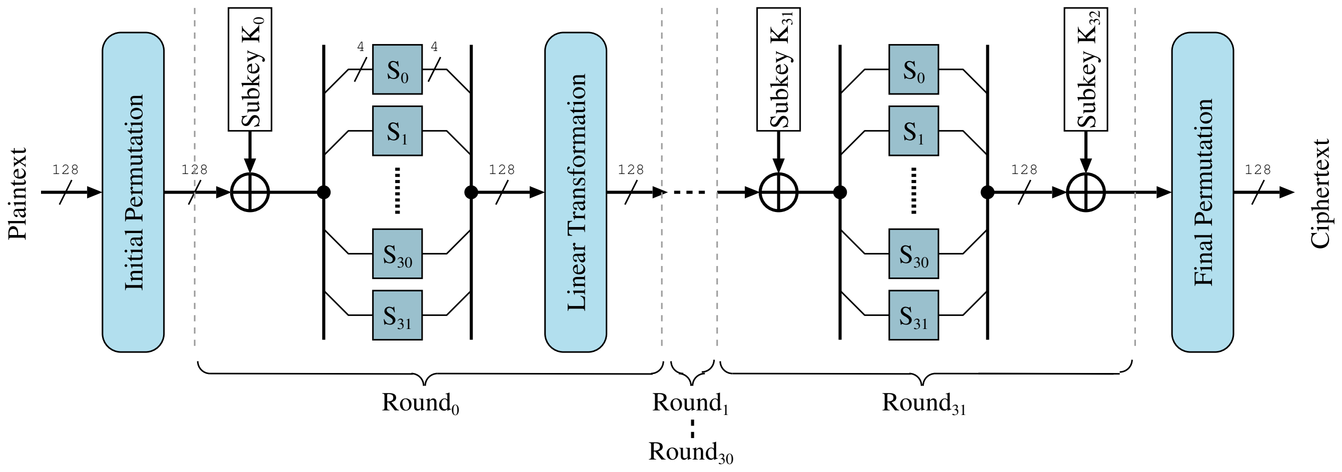

Fig. 2. Serpent block cipher.

for the encryption part and reaches a throughput of 133 Gbit/s, running at 260 MHz. Furthermore, our final design has actually been tested using a custom-designed printed circuit board (PCB).

The remainder of this paper is structured as follows. In the next section, we present an overview of related work on hardware architectures targeting high throughput AE designs. Throughout Section III, a brief description of Serpent and OCB is given. The actual hardware architecture of our design is presented in Section IV. In Section V, we summarize our results, followed by a short discussion, and finally we conclude our work in Section VI.

II. Related Work

As it was standardized by the NIST, GCM-AES has received significant attention from the research community, and several implementations targeting FPGAs can be found in the literature.

In 2009, Zhou et al. [10] presented a single-core GCM-AES design, which targets a Xilinx Virtex-5 FPGA and achieved a throughput of 41.5 Gbit/s based on the 128-bit version of AES. Henzen and Fichtner [11] showed that it is possible to break the 100 Gbit/s barrier on a Virtex-5. They made use of four fully unrolled AES cores for the encryption part and used four Karatsuba-Ofman (KO) multipliers in order to realize the authentication part. Their design reaches a throughput of 119.3 Gbit/s.

When using GCM, the most complex operation during the computation of a message digest is actually the multiplication in the binary finite-field $GF(2^{128})$, which is part of the universal hashing function called GHASH. Therefore, most of the effort in improving GCM implementations has been spent on speeding up this calculation. Wang et al. [12] presented a GHASH architecture based on four GHASH cores that achieved a throughput of 123.1 Gbit/s on a Virtex-5. In [13], Crenne et al. reached 238.1 Gbit/s by using 8 parallel finite-field multipliers, also targeting a Xilinx Virtex-5 FPGA.

To the best of our knowledge, no hardware architecture based on a block cipher other than AES and trageting a high-throughput AE implementation has been presented so far. Moreover, no AES design has been published to date, which makes use of an operation mode different than GCM in order to achieve throughputs up to 100 Gbit/s.

III. 100 Gbit/s AE Alternatives

To reach throughputs exceeding 100 Gbit/s on commercial FPGA devices, it is necessary to use multiple parallel instances of cryptographic primitives. Although AES running in GCM mode is currently the most widespread option for high-throughput hardware architectures, this is not a requirement. A different block cipher and a different mode of operation can be used for an AE system as well.

A. Serpent Block Cipher

Serpent was the runner-up of the AES block cipher competition. Although it has not been chosen by the NIST during the competition, it was considered to be a close alternative and is still known to be secure from a cryptography point of view as the considerably large number of rounds contributes to its security [14].

In the following we will briefly discuss the basic structure of Serpent using the *conventional* implementation described in the proposal [15]. For detailed information on the cipher, please refer to the official proposal document.

The main components of Serpent are the key schedule and the cipher itself. The key schedule takes a 256-bit cipher key[1] and expands it to 33 128-bit subkeys denoted by K_i. The subkeys are required during the round transformations of the cipher. Fig. 2 illustrates the Serpent cipher which consists of an initial permutation (IP), 32 rounds, and a final permutation (FP). The first 31 rounds of the cipher include a *key-mixing stage*, a *substitution stage*, and an *avalanche stage* (i.e., a stage where a linear transformation takes place). In the last round of

[1]Keys shorter than 256 bits are padded accordingly.

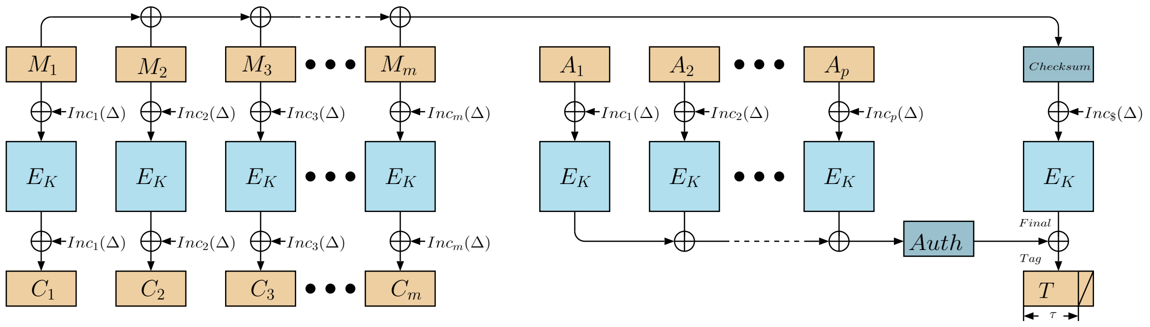

Fig. 3. Overview of the encryption and the authentication part of OCB.

the cipher, the linear transformation is omitted and replaced by another key-mixing operation.

B. Offset CodeBook Mode

The offset codebook (OCB) block cipher mode of operation is a combined AE scheme and has first been published by Rogaway et al. [9] in 2001. It is strongly related to the Integrity Aware Parallelizable Mode (IAPM) by C. Jutla [16] and three different versions have been made public since 2001. Throughout the remainder of this paper we solely refer to the third version of it, i.e., OCB3 [17].

To start the authenticated encryption scheme according to OCB, a plaintext message, denoted by M, gets split into m different blocks, each of length n and an optional block M_* of length smaller than n as follows[2]:

$$M = \begin{cases} M_1, \ldots, M_m, & \text{if } |M| = k \cdot n \text{ and } k \in \mathbb{N}, \\ M_1, \ldots, M_m, M_*, & \text{else.} \end{cases}$$

Algorithm 1 and Fig. 3 describe the authenticated encryption according to OCB using pseudo-code and a block diagram, respectively. For simplicity, only the cases for full message blocks, i.e., $M_* = \varnothing$ are shown. It starts with a setup and initialization step (cf. line 4 and 5). Thereafter, each message block can be processed independently of each other (line 6 to 16). Finally, the authentication tag T is determined throughout line 17 to 21. The characters $\|$, $\oplus$, and $\wedge$ denote the concatenation, bitwise exclusive or, and bitwise and operation, respectively. The term *ntz(i)* describes the number of trailing zeroes of i in binary representation. 0^n and 1^n represent bit strings of length n containing only zeros and ones, respectively. Furthermore, $trunc(X, y)$ truncates the bit string X to its y least significant bits. We use $\varnothing$ to represent both an empty binary string of length n (cf. line 3), and an *empty set* as within line 11. Appendix A provides listings for the *Setup*, *Init*, and $Hash_K$ procedures used throughout Algorithm 1.

When using a counter for the nonce N, the calculation of the initial offset Δ requires a block cipher call only every 64*th* initialization. This is due to the fact that the least significant six bits of N are set to zero before passing it to the block cipher (cf. line 2 of Algorithm 3). This fact together with the parallelizable processing of the message blocks, makes OCB suitable for high-throughput applications.

Algorithm 1 OCB authenticated encryption.

```
Input: Message $M$, Message block length $n$, Cipher key $K$,
    Nonce $N$, Associated data $A$, Tag length $\tau$
Output: Ciphertext $C$, Authentication tag $T$
 1: if $|N| \geq n$ then return INVALID
 2: $\{M_1, \ldots, M_m, M_*\} \leftarrow M$, with $|M_i| = n$ and $|M_*| < n$
 3: $Checksum \leftarrow \varnothing; C \leftarrow \varnothing$
 4: $L_*, L_\$, L[0] \ldots L[\lfloor log_2(m) \rfloor] \leftarrow Setup(K, m)$
 5: $\Delta \leftarrow Init(N, n, K)$
 6: for $i = 1$ to $m$ do
 7:     $\Delta \leftarrow \Delta \oplus L[ntz(i)]$                    ▷ $Inc_i(\Delta)$
 8:     $C \leftarrow C \| E_K(M_i \oplus \Delta) \oplus \Delta$
 9:     $Checksum \leftarrow Checksum \oplus M_i$
10: end for
11: if $M_* \neq \varnothing$ then
12:     $\Delta \leftarrow \Delta \oplus L_*$
13:     $Pad \leftarrow E_K(\Delta)$
14:     $C \leftarrow C \| M_* \oplus (Pad \wedge (1^{|M_*|}))$
15:     $Checksum \leftarrow Checksum \oplus M_* 10^*$, with
            $M_* 10^* = M_* \| 1 \| 0 \ldots 0$, such that $|M_* 10^*| = n$
16: end if
17: $\Delta \leftarrow \Delta \oplus L_\$$                          ▷ $Inc_\$(\Delta)$
18: $Final \leftarrow E_K(Checksum \oplus \Delta)$
19: $Auth \leftarrow Hash_K(A)$
20: $Tag \leftarrow Final \oplus Auth$
21: $T \leftarrow trunc(Tag, \tau)$
        return $C \| T$
```

IV. OCB-Serpent Hardware Architecture

For the OCB-Serpent architecture, supporting the IEEE 802.3ba Ethernet standard, we assume the following prerequisites:

- The size of the message block counter i is restricted to 7 bits, since 2^7 message blocks are capable of a full Ethernet frame.
- As our target application ensures solely full message blocks, we do not handle short final message blocks

[2]We refer to the length of x in bits using the following notation: $|x|$

separately.

As mentioned before, to achieve extremely high throughputs, a multi-core approach has proven to be the only viable option when implementing AE on commercial FPGA platforms. Similar to GCM, OCB also allows two successive message blocks to be processed independently of each other. We have taken advantage of this fact and decided to use four parallel cipher cores in order to achieve the desired throughput. Fig. 4 illustrates the OCB architecture for authenticated encryption based on four Serpent cores.

A. Pipelined Four-Core Serpent Architecture

Each of the four Serpent cores handles a single 128-bit message block. Therefore, the overall design can process a 512-bit message at a time. As can be seen from Fig. 4, we fully unrolled the 32 rounds of Serpent. Furthermore, we inserted pipeline stages after each round in order to increase the maximum frequency of the cipher cores. Although the pipelined architecture allows us to clock the Serpent cores at a higher frequency, one problem inherent to all pipeline architectures has to be taken into consideration: When the normal flow of operations has to be suspended, the entire pipeline must be stopped in order to allow the rest of the operation to resume. Such an occurrence is known as a *pipeline stall* and can, e.g., occur during a key change.

Due to the unrolling of the Serpent rounds, we have to store 1024 of the 4-bit S-boxes for each core, which requires a considerable amount of memory. The subkeys for the key-mixing stage of all four Serpent cores are provided from a single key schedule, i.e., the cores always operate on the same cipher key.

B. OCB - Authenticated Encryption

OCB can, in general, be subdivided into three stages: *Initialization*, *encryption/authentication*, and *finalization*. During the initialization phase, two potential pipeline stalls may occur if not handled properly. First, after each key change a cipher call $E_K(\varnothing)$ is required in order to compute the table values $L[..]$ (cf. Algorithm 2). Second, each new message needs a fresh nonce N, and thus a new offset value Δ. Since the calculation of the initial offset also requires a cipher call, this may result in another pipeline stall. In order to reduce the number of pipeline stalls to a minimum, we precompute the initial offset values.

The limitation of message lengths to a maximum of 2^7 blocks facilitates the precomputation of the $L[..]$-values, as it limits the maximum number of trailing zeroes of the block counter i to six. Thus, only $L_\$$, L_*, and $L_0 \ldots L_6$ have to be precomputed and stored in registers. In fact, when the result of $E_K(\varnothing)$ is available, all nine table values can be computed in a single clock cycle[3].

When processing a block in authenticated-encryption mode, the message gets XOR-ed with the current offset Δ_i, encrypted, and finally XOR-ed with Δ_i again. As pipeline stages

[3]The calculation only depends on operations cheap to implement in hardware, i.e., fixed shift and conditional exclusive or operations.

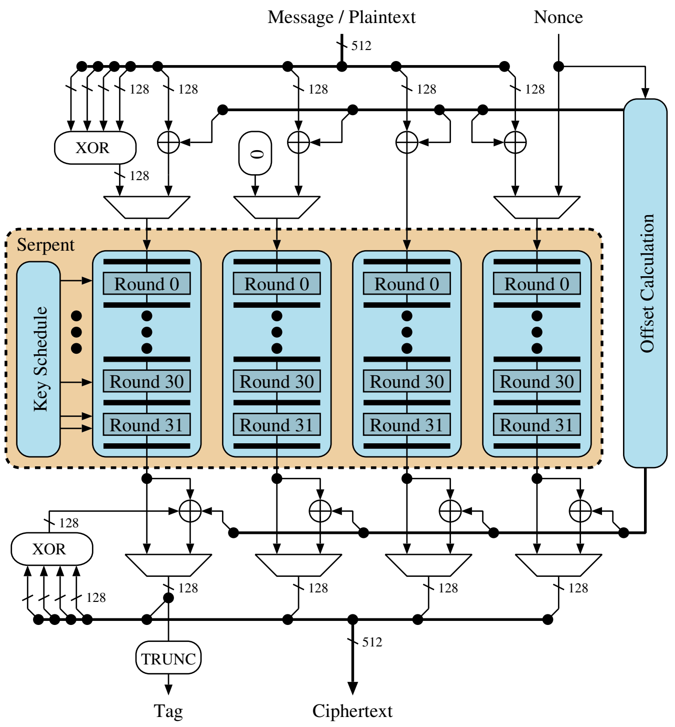

Fig. 4. OCB-Serpent architecture.

were introduced into the cipher cores, the Δ-values either have to be stored or recalculated. We decided to recompute the offsets as it makes the implementation less dependent on the underlying block cipher and the number of pipeline stages used. Since the multi-core design processes four message blocks in parallel, the offset-calculation needs to be capable of providing four offset values at a time. In fact, the calculation of the four offset values is relatively cheap as the initial offset only has to be XOR-ed with the precomputed table values.

As described in Algorithm 3, a nonce-dependent call to the encryption of the block cipher is required. The result of this operation, further-on called *Ktop*, then has to be shifted by a 6-bit nonce-dependent value *Bottom*. First, in order to be able to perform this shift-operation, *Bottom* has to be buffered until the result of $E_K(Top)$ is available. Second, the 6-bit variable shift is done using a 192-bit by 6-bit barrel shifter. Although using a counter for the nonce N could avoid the resource-expensive barrel shifter, we decided to keep it in order to stay independent of the actual structure of the chosen nonce.

C. Decryption

Authenticated decryption according to OCB is very similar to the encryption process. Exchanging the encryption operation E_K of the underlying cipher by the decryption operation D_K in line 8 and 13 of Algorithm 1 and in line 6 and 10 of Algorithm 4, turns OCB into decryption mode. However, the encryption operations during the initialization phase, described in Algorithm 2 and Algorithm 3, remain. Thus, the multi-core decryption unit contains four block-cipher decryption

TABLE I
ENCRYPTION-ONLY AND AUTHENTICATED ENCRYPTION RESULTS BASED ON AN ALTERA STRATIX IV (EP4S100G5F45) PLATFORM.

Block	**Mode of**	**Number of**	**Area**		f_{max}	**Throughput**	
Cipher	**Operation**	**Cipher Cores**	[ALMs]	[M9K Blocks]	[MHz]	**Absolute** [Gbit/s]	**Relative** [%]
Serpent	cipher-only	4	25,825	0	267	136	130
AES	cipher-only	4	6,916	314	252	124	118
Serpent	OCB	4	29,506	0	260	133	127
AES	OCB	4	10,060	314	220	112	107
Serpent	GCM	4	56,474	0	203	104	99
AES	GCM	4	24,313	314	206	105	100

cores, one encryption core required during initialization and finalization, and a common key schedule.

A minor drawback of authenticated decryption according to OCB is the fact that a delay, dependent on the number of pipeline stages p between the calculation of the plaintext and the calculation of the authentication tag exists. This delay is caused by the calculation of the authentication tag which requires the *Checksum* of the plaintext. Therefore, in order to verify the authentication tag of a message, p plaintext blocks have to be buffered, resulting in additional memory requirements.

V. RESULTS

We coded our architectures in VHDL. For the synthesis and place&route design steps, we used Altera Quartus II version 11.0. Functional correctness was verified with Modelsim 6.6e simulator. Synthesis was conducted using a speed-optimzed setting. Except of M9K block memories, no Altera-specific logic blocks had been used. No constraints regarding placement had been set.

In order to have some reference implementations regarding AES and GCM on our target platform (Altera Stratix IV), we also synthesized a four-core AES cipher architecture as well as GCM based on both Serpent and AES. The AES architectures were accomplished with an underlying four-core AES similar to the one proposed by Henzen et al. [11]. We used the 128-bit version of AES and fully unrolled the 10 rounds. Furthermore, pipelining registers have been inserted after each round similar to the Serpent cipher core design.

The first two rows of Table I contain the synthesis results of the multi-core encryption architectures (i.e., without running any mode of operation). The subsequent rows present the results for the different combinations of block ciphers and modes of operation. Regarding the block ciphers, the fully unrolled four-core AES design requires less area as it only has to provide 160 8-bit S-boxes for each cipher core, compared to the 1024 4-bit S-boxes required by the Serpent cores. Since the high-throughput universal hash function GHASH of the GCM mode occupies a lot of area, OCB designs result in a smaller footprint than their GCM counterparts. Nevertheless, GCM is still the preferred mode of operation in literature when designing high-speed authenticated encryption hardware architectures based on block ciphers. This might be due to the facts that there are some US patents on OCB and that in contrast to GCM it has not been recommended by the NIST. Table II summarizes the properties of the two AE block cipher modes of operation.

TABLE II
BLOCK CIPHER - MODES OF OPERATION COMPARISON.

Property	**OCB**	**GCM**
Patented	Yes (US)	No
Parallelizable	Yes (Encr. + Auth.)	Yes (Encr. + Auth.)
Decryption required	Yes	No
Cipher calls (Initialization)	1	1
Cipher calls (Encryption)	$\lceil \lvert M \rvert / n \rceil + 1.016$	$\lceil \lvert M \rvert / n \rceil + 1$

Regarding the throughput, all architectures met the constraint of 100 GBit/s. However, the OCB versions are considerably faster. This is mainly because of the simpler architecture of OCB compared to GCM, which requires the resource-expensive GHASH function.

Our designs have finally been tested on a self-designed printed circuit board (PCB), which has solely been developed for high speed authenticated encryption architectures running on an Altera Stratix IV FPGA.

VI. CONCLUSION

In the present work, we described a hardware architecture for high-speed authenticated encryption (AE) on FPGAs, based on *alternative* cryptography primitives. Our design operates in the offset codebook (OCB3) mode of operation and contains four parallel Serpent block cipher cores for the encryption part in order to achieve the desired data rates according to IEEE 802.3ba. The OCB3-Serpent architecture reaches a throughput of 133 Gbit/s and thus, outperforms all GCM-AES implementations on FPGAs available to date. Although OCB3 has not (yet) been approved by the NIST, its structure (as well as that of Serpent) is definitely suitable for high throughput implementations as shown during this work.

ACKNOWLEDGEMENT

This work is part of the QCRYPT project, which is evaluated by the Swiss National Science Foundation and financed by the Swiss Confederation with funding via Nano-Tera.ch.

The authors would like to thank the entire team from the Microelectronics Design Center at the ETH Zurich for their help during the development of the printed circuit board.

References

[1] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," RFC 3610 (Informational), Sep. 2003.

[2] D. A. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," *NIST Modes Operation Symmetric Key Block Ciphers*, 2005.

[3] M. Dworkin, "Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST, Tech. Rep., 2007.

[4] IEEE Std 802.11-2007, "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," LAN/MAN Standards Committee, New York, NY, USA, pp. C1–1184, Jun. 2007.

[5] J. Viega and D. Mcgrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," RFC 4106 (Proposed Standard), Internet Engineering Task Force (IETF), Jun. 2005.

[6] NIST, *Advanced Encryption Standard (AES) (FIPS PUB 197)*, National Institute of Standards and Technology, Nov. 2001.

[7] "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment 4: Media Access Control Parameters, Physical Layers and Management Parameters for 40 Gb/s and 100 Gb/s Operation," *IEEE Std 802.3ba-2010 (Amendment to IEEE Standard 802.3-2008)*, pp. 1–457, 22 2010.

[8] E. Biham, R. Anderson, and L. Knudsen, "Serpent: A New Block Cipher Proposal," in *Fast Software Encryption*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed. Springer Berlin / Heidelberg, 1998, vol. 1372, pp. 222–238.

[9] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," ACM Conference on Computer and Communications Security, 2001, pp. 196–205.

[10] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, J. Becker, R. Woods, P. Athanas, and F. Morgan, Eds. Springer Berlin / Heidelberg, 2009, vol. 5453, pp. 193–203.

[11] L. Henzen and W. Fichtner, "FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications," ESSCIRC, 2010 Proceedings of the, Sep. 2010, pp. 202 –205.

[12] J. Wang, G. Shou, Y. Hu, and Z. Guo, "High-speed architectures for GHASH based on efficient bit-parallel multipliers," Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on, 2010, pp. 582 –586.

[13] J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J.-P. Diguet, "Efficient key-dependent message authentication in reconfigurable hardware," Field-Programmable Technology (FPT), 2011 International Conference on, Dec. 2011, pp. 1–6.

[14] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)," National Institute of Standards and Technology (NIST), Tech. Rep., 2000.

[15] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," Proceedings of the First AES Candidate Conference. Ventura, CA, USA: National Institute of Standard and Technology, Jun. 1998.

[16] C. Jutla, "Encryption Modes with Almost Free Message Integrity," in *Advances in Cryptology EUROCRYPT 2001*, ser. Lecture Notes in Computer Science, B. Pfitzmann, Ed. Springer Berlin / Heidelberg, 2001, vol. 2045, pp. 529–544.

[17] T. Krovetz and P. Rogaway, "The Software Performance of Authenticated-Encryption Modes," in *Fast Software Encryption*, ser. Lecture Notes in Computer Science, A. Joux, Ed. Springer Berlin / Heidelberg, 2011, vol. 6733, pp. 306–327.

Appendix

Algorithm 2 lists the calculation of the table values $L[..]$ required during the OCB mode of operation. The *double*-procedure is defined according to:

$$double(X) = (X \lll 1) \oplus (msb(X) \cdot 0x87),$$

where $msb(X)$ represents the most significant bit of X using binary representation.

Algorithm 2 Table value calculations.

```
Input: Cipher key $K$, Number of message blocks $m$
Output: $Setup(K, m)$
1: $L_* \leftarrow E_K(\varnothing)$
2: $L_\$ \leftarrow double(L_*)$
3: $L[0] \leftarrow double(L_\$)$
4: for $i = 0$ to $\lfloor log_2(m) \rfloor$ do
5:     $L[i] \leftarrow double(L[i-1])$
6: end forreturn $L_*, L_\$, L[0] \ldots L[\lfloor log_2(m) \rfloor]$
```

The initial offset Δ is determined according to Algorithm 3. $X \lll i$ denotes a left shift operation of X by i bits.

Algorithm 3 Initial offset (Δ) calculation.

```
Input: Nonce $N$, Message block length $n$, Cipher key $K$
Output: $Init(N, n, K)$
1: $Nonce \leftarrow 1 \,\|\, N$
2: $Top \leftarrow (1^{122} \,\|\, 0^6) \wedge Nonce$
3: $Bottom \leftarrow Nonce \wedge 1^6$
4: $Ktop \leftarrow E_K(Top)$
5: $Stretch \leftarrow Ktop \,\|\, (Ktop \oplus (Ktop \lll 8))$
6: $\Delta \leftarrow (Stretch \lll Bottom) \wedge 1^n$
       return $\Delta$
```

Algorithm 4 describes the calculation of $Hash_K(A)$. Since the $Setup$ procedure already gets called during the actual encryption process of OCB (cf. Algorithm 1, line 4), line 3 in Algorithm 4 can be omitted as long as the table values $L[..]$ are globally available.

Algorithm 4 Authentication hash ($Hash_K(A)$) calculation.

```
Input: Associated data $A$, Associated data block length $q$,
    Cipher key $K$
Output: $Hash_K(A)$
1: $\{A_1, \ldots, A_p, A_*\} \leftarrow A$, with $|A_i| = q$ and $|A_*| < q$
2: $Sum \leftarrow \varnothing$; $\Delta \leftarrow \varnothing$
3: $L_*, L[0] \ldots L[\lfloor log_2(p) \rfloor] \leftarrow Setup(K, p)$
4: for $i = 1$ to $p$ do
5:     $\Delta \leftarrow \Delta \oplus L[ntz(i)]$
6:     $Sum \leftarrow Sum \oplus E_K(A_i \oplus \Delta)$
7: end for
8: if $A_* \neq \varnothing$ then
9:     $\Delta \leftarrow \Delta \oplus L_*$
10:    $Sum \leftarrow Sum \oplus E_K(A_*10^* \oplus \Delta)$, with
           $A_*10^* = A_* \| 1 \| 0 \ldots 0$, such that $|A_*10^*| = q$
11: end if
       return $Sum$
```