Information Processing Letters 110 (2010) 549–553

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm

Shay Gueron [a,b], Michael Kounavis [c,*]

[a] Department of Mathematics, University of Haifa, Israel
[b] Intel Architecture Group, Intel Corporation, Israel Development Center, Haifa, Israel
[c] Intel Architecture Group, Intel Corporation, Hillsboro, OR, USA

ARTICLE INFO

Article history:
Received 15 February 2010
Received in revised form 19 April 2010
Accepted 19 April 2010
Available online 21 April 2010
Communicated by D. Pointcheval

Keywords:
Cryptography
Galois fields
Message authentication
Authenticated encryption
AES-GCM

ABSTRACT

This paper describes a new method for efficient implementation of the Galois Counter Mode on general purpose processors. Our approach is based on three concepts: a) having a 64-bit carry-less multiplication instruction in the processor; b) a method for using this instruction to efficiently multiply binary polynomials of degree 127; c) a method for efficient reduction of a binary polynomial of degree 254, modulo the polynomial $x^{128} + x^7 + x^2 + x + 1$ (which defines the finite field of the Galois Counter Mode). The two latter concepts can be used for writing an efficient and lookup-table free software implementation of the Galois Counter Mode, for processors that have a carry-less multiplication instruction. Our approach uses only a generic carry-less multiplication instruction, without any field-specific reduction logic, making the instruction applicable to multiple use cases, and therefore an appealing addition to the instruction set of a general purpose processor. This research played a significant role in the process that eventually led to adding a carry-less multiplication instruction (called PCLMULQDQ) to the Intel Architecture. PCLMULQDQ and six AES instructions are introduced in the new 2010 Intel Core processor family, based on the 32 nm Intel microarchitecture codename "Westmere". On the new Westmere processors, the software that implements the methods described here, computes AES-GCM more than six times faster than the current, lookup table-based, state-of-the-art implementation. This new capability adds motivation to using AES-GCM for high performance secure networking.

1. Introduction

The Galois Counter Mode (GCM) is a recommended mode of operation for block ciphers, used for confidentiality and authentication. In particular, AES in the GCM mode (AES-GCM) is becoming a prominent algorithm used for packet processing in fast networking.

AES-GCM is defined in FIPS 800-38D [1], and was endorsed by the US government in 2006. It is also defined in the IEEE 802.1ae standard [2] and recommended for networking in rates higher than 10 Gbps. AES-GCM is also used in IPsec RFC 4106 [3], in the storage standard P1619.1 [4], and in Security Protocols over Fiber Channel (ISO-T11 Committee [5]). The SSL/TSL group also considers this mode as part of the TLS 1.2 protocol [6].

GCM is defined by means of the operations in the binary finite field $GF(2^{128})$ with reduction polynomial being equal to $g = g(x) = x^{128} + x^7 + x^2 + x + 1$ (called a 'pentanomial' because it has only five nonzero coefficients).

A general, lookup table-based method for performing multiplications in finite binary fields is described in [7], and improved in [8]. This method is the basis for the software implementation of AES-GCM in [9], which is considered the state-of-the-art for this algorithm.

We present here a new method that allows for a more efficient implementation of GCM. It is based on three con-

* Corresponding author.
E-mail address: michael.e.kounavis@intel.com (M. Kounavis).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2010.04.011

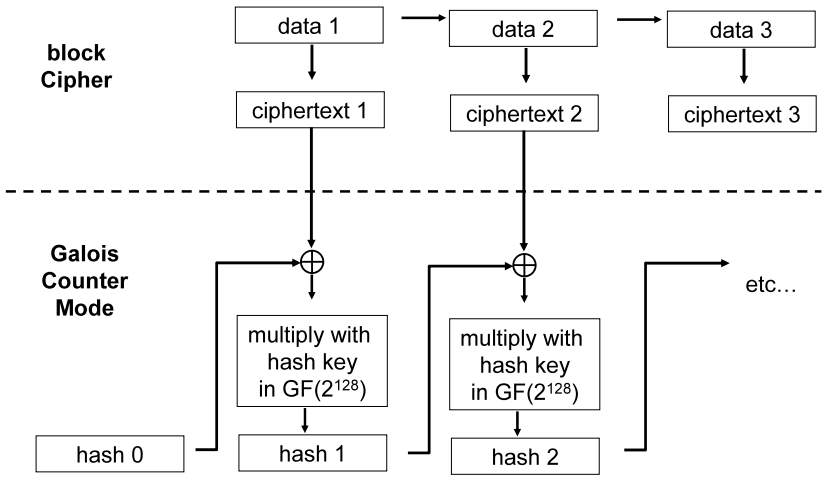

Fig. 1. The Galois Counter Mode.

cepts: a) adding an instruction that computes the carry-less product of two 64-bit inputs; b) developing a "carry-less Karatsuba multiplication" algorithm that uses this instruction for multiplying two binary polynomials of degree 127; c) developing a new method for reducing a binary polynomial of degree 254 modulo the specific pentanomial g, taking advantage of its special "sparse" structure. This is done in place of general methods for binary field multiplications.

The paper is structured as follows: in Section 2 we describe the carry-less multiplication operation. In Section 3 we describe the current software implementation of GCM. In Section 4 we describe our algorithms. Section 5 discusses the bit-reflection peculiarity of the GCM standard. Section 6 summarizes the paper with some concluding remarks.

2. Carry-less multiplication

Carry-less multiplication is defined as follows. Let A, B, be two n bits operands, denoted by the following array of bits

$$A = [a_{n-1}a_{n-2}\dots a_0] \tag{1}$$

and

$$B = [b_{n-1}b_{n-2}\dots b_0] \tag{2}$$

Let $C = A * B$ be the result of the carry-less multiplication of A and B, and denote it by the following array of bits

$$C = [c_{2n-1}c_{2n-2}\dots c_0] \tag{3}$$

Then, the bits of C are defined by

$$c_i = \bigoplus_{j=0}^{i} a_j b_{i-j} \tag{4}$$

for $0 \leqslant i \leqslant n-1$, and

$$c_i = \bigoplus_{j=i-n+1}^{n-1} a_j b_{i-j} \tag{5}$$

for $n-1 \leqslant i \leqslant 2n-2$ (note that $c_{2n-1} = 0$).

Hereafter, we identify each n-bit array $[u_{n-1}u_{n-2}\dots u_0]$ with the binary polynomial $u_{n-1}x^{n-1} + u_{n-2}x^{n-2} + \cdots + u_1x + u_0$, and view such arrays as elements in a binary finite field $GF(2^n)$, in polynomial representation, with some n-degree irreducible (binary) polynomial.

In this field, addition of two elements (binary polynomials of degree $n-1$) corresponds to the XOR value of the corresponding n-bit arrays. Polynomial multiplication of two binary polynomials of degree $n-1$ returns a binary polynomial of degree $2n-2$. This polynomial multiplication corresponds to the carry-less product of the corresponding n-bit arrays. The latter product is a $(2n-1)$-bit array but can be viewed as $2n$-bit array which most significant bit is zero. The finite field multiplication of the two elements, is the remainder of the $(2n-2)$-degree polynomial, modulo the reduction polynomial of the field. This polynomial corresponds to an n-bit array. In the context of GCM, $n = 128$.

It is interesting to see that carry-less multiplication is relatively cheap to implement in hardware, compared to integer multiplication where carry bits need to be generated and propagated. For example, a 64-bit carry-less multiplier can be implemented with only 3969 XOR and 4096 AND gates.

3. GCM in software

The Galois Counter Mode is illustrated in Fig. 1. It is defined via operations in the finite field $GF(2^{128})$ with the irreducible polynomial $g = g(x) = x^{128} + x^7 + x^2 + x + 1$. The GCM procedure produces a 128-bit message digest, called "Galois Hash", from the encrypted data. This digest ("authentication tag") is used for message authentication. The tag is computed iteratively: a current tag value is added to a newly computed 128-bit ciphertext block, and the result is multiplied with some fixed element of the field, called

hash key (which we view as a 128-bit string). All operations are in the finite field that defines GCM. Fig. 1 omits some initialization and finalization operations of GCM for the sake of simplicity.

Currently, the most efficient implementations of GCM use a table lookup algorithm [7–9] that consists of two phases:

Preprocessing phase. 16 different tables are created. Each table has 256 entries of 128 bits, where entry j of table T_i stores the value $(j * \text{hash key} * 2^{8i}) \bmod g$.

Run time phase. The algorithm takes a newly produced ciphertext block and XOR's it with the current tag value. The result is multiplied in $GF(2^{128})$, by the hash key as follows: the desired product value is segmented into 16 slices of 8 bits, and 16 table lookups are performed, using the slices for indexing the tables. The results from the table lookups are XOR-ed with each other.

For every 128-bit block, this algorithm performs 16 table lookups and 16 128-bit XOR operations. It is not very efficient, mainly due to the cost of table lookups, which total size is 64 K bytes. Since typical processors have only 32–64 K bytes in their first level cache, these tables may partially reside in the second level cache, depending on the processor workload. This decreases the performance, even if memory accesses are pipelined.

4. An efficient algorithm for implementing GCM

Our method assumes that a 64-bit carry-less multiplication instruction is available. We refer to this instruction as 'PCLMULQDQ' in this paper. We develop two algorithms: one that uses PCLMULQDQ to compute efficiently a 256-bit carry-less product of two 128-bit operands (the top bit is always 0), and the other one for efficiently reducing the result, modulo g. This is described in the following subsections.

4.1. A carry-less Karatsuba algorithm for computing 128-bit carry-less products using a 64-bit PCLMULQDQ instruction as a building block

We denote the 128-bit input operands by $[A_1 : A_0]$ and $[B_1 : B_0]$, where A_0, A_1, B_0 and B_1 are each 64 bits long, the symbol '$*$' means carry-less multiplication, and the symbol ":" denotes concatenation. The proposed algorithm is the following (see Algorithm 1).

Algorithm 1 (Carry-less Karatsuba)

Input: $[A_1 : A_0], [B_1 : B_0]$.

Step 1: Use the PCLMULQDQ instruction 3 times, as follows:

$$[C_1 : C_0] = A_1 * B_1, \qquad [D_1 : D_0] = A_0 * B_0,$$
$$[E_1 : E_0] = (A_0 \oplus A_1) * (B_0 \oplus B_1).$$

Step 2: Construct the 256-bit product $[A_1 : A_0] * [B_1 : B_0]$ as follows:

$$[A_1 : A_0] * [B_1 : B_0] = [C_1 : C_0 \oplus C_1 \oplus D_1 \oplus E_1 : D_1 \oplus C_0 \oplus D_0 \oplus E_0 : D_0] \quad (6)$$

(note that the most significant bit of the result is 0)

Output: $[C_1 : C_0 \oplus C_1 \oplus D_1 \oplus E_1 : D_1 \oplus C_0 \oplus D_0 \oplus E_0 : D_0]$

We call this algorithm a "carry-less Karatsuba multiplication" (see [10,18] for Karatsuba multiplication).

4.2. Reducing a polynomial of degree 254 modulo $g = x^{128} + x^7 + x^2 + x + 1$

Consider the polynomial as a 256-bit string with a zero most significant bit. We split this string to two 128-bit halves. The least significant half is just XOR-ed with the final remainder, since the degree of g is 128. For the most significant part, we develop an algorithm that realizes division via two (carry-less) multiplications. This algorithm can be viewed as an extension of the Barrett modular reduction algorithm [11] to modulo-2 arithmetic, or as an extension of the Feldmeier CRC generation algorithm [12] to dividends and divisors of arbitrary size.

Since we handle the least significant half of the input separately, we need to investigate here only the efficient generation of a remainder $p(x)$ defined as follows

$$p(x) = c(x) \cdot x^t \bmod g(x) \quad (7)$$

Here,

- $c(x)$ is a binary polynomial of degree $s-1$, representing the most significant bits of the input (in our case, $s = 128$).
- t is the degree of the polynomial g (in our case, $t = 128$).
- $g(x)$ is a polynomial of degree s (in our case, $g(x) = x^{128} + x^7 + x^2 + x + 1$).

We denote the coefficient of $p(x)$, $c(x)$, and $g(x)$ as follows:

$$c(x) = c_{s-1}x^{s-1} + c_{s-2}x^{s-2} + \cdots + c_1 x + c_0$$
$$p(x) = p_{t-1}x^{t-1} + p_{t-2}x^{t-2} + \cdots + p_1 x + p_0 \quad \text{and}$$
$$g(x) = g_t x^t + g_{t-1}x^{t-1} + \cdots + g_1 x + g_0 \quad (8)$$

Hereafter, we use the notation $L^u(v)$ to denote the coefficients of the u least significant terms of the polynomial v and use $M^u(v)$ to denote the coefficients of its u most significant terms. The polynomial $p(x)$ can be expressed as:

$$p(x) = c(x) \cdot x^t \bmod g(x) = g(x) \cdot q(x) \bmod x^t \quad (9)$$

where $q(x)$ is a polynomial of degree $s-1$ which is the quotient from the division of $c(x) \cdot x^t$ by g. The intuition behind Eq. (9) is that the t least significant terms of the dividend $c(x) \cdot x^t$ equal to zero. Further, the dividend $c(x) \cdot x^t$ can be expressed as

$$c(x) \cdot x^t = g(x) \cdot q(x) + p(x) \quad (10)$$

where '+' denotes polynomial addition (corresponding to an XOR operation). From Eq. (10), one can expect that the t least significant terms of the polynomial $g \cdot q$ are equal to the terms of the polynomial p. Only if these terms are equal, the t least significant terms of $g \cdot q \oplus p$ would be zero in. Hence,

$$p(x) = g(x) \cdot q(x) \bmod x^t = L^t\big(g(x) \cdot q(x)\big) \quad (11)$$

Now we define

$$g(x) = g_t x^t + g^*(x) \tag{12}$$

where the polynomial g^* represents the t least significant terms of the polynomial g. Obviously,

$$\begin{aligned} p(x) &= L^t\big(g(x)\cdot q(x)\big) \\ &= L^t\big(q(x)\cdot g^*(x) + q(x)\cdot g_t x^t\big) \end{aligned} \tag{13}$$

However, the t least significant terms of the polynomial $q \cdot g_t \cdot x^t$ are zero, and therefore,

$$p(x) = L^t\big(q(x)\cdot g^*(x)\big) \tag{14}$$

From Eq. (14) it follows that in order to compute the remainder p we need to know the value of the quotient q. This quotient can be calculated in a similar manner as in the Barrett reduction algorithm:

$$(9) \quad \Leftrightarrow \quad c(x)\cdot x^{t+s} = g(x)\cdot q(x)\cdot x^s + p(x)\cdot x^s \tag{15}$$

Let

$$x^{t+s} = g(x)\cdot q^+(x) + p^+(x) \tag{16}$$

where q^+ is an s-degree polynomial, equals to the quotient from the division of x^{t+s} by g, and p^+ is the remainder from this division. The degree of the polynomial p^+ is $t-1$. From Eqs. (15) and (16) we get

$$\left.\begin{matrix}(15)\\(16)\end{matrix}\right\} \quad \Leftrightarrow \quad \begin{aligned} &c(x)\cdot g(x)\cdot q^+(x) + c(x)\cdot p^+(x) \\ &\quad = g(x)\cdot q(x)\cdot x^s + p(x)\cdot x^s \end{aligned} \tag{17}$$

and

$$\begin{aligned} (17) \quad \Rightarrow \quad & M^s\big(c(x)\cdot g(x)\cdot q^+(x) + c(x)\cdot p^+(x)\big) \\ &= M^s\big(g(x)\cdot q(x)\cdot x^s + p(x)\cdot x^s\big) \end{aligned} \tag{18}$$

One can see that the polynomials $c\cdot g\cdot q^+$ and $g\cdot q\cdot x^s$ are of degree $t+2\cdot s-1$, the polynomial $c\cdot p^+$ is of degree $t+s-2$, and the polynomial $p\cdot x^s$ is of degree $t+s-1$. Therefore, the s most significant terms of the polynomials on the left- and the right-hand side of Eq. (18) are not affected by the polynomials $c\cdot p^+$ and $p\cdot x^s$. Consequently,

$$\begin{aligned} (18) \quad \Leftrightarrow \quad & M^s\big(c(x)\cdot g(x)\cdot q^+(x)\big) \\ &= M^s\big(g(x)\cdot q(x)\cdot x^s\big) \end{aligned} \tag{19}$$

Next, we observe that the s most significant terms of the polynomial $c\cdot g\cdot q^+$ equal to the s most significant terms of the polynomial $g\cdot M^s(c\cdot q^+)\cdot x^s$. We observe that the polynomial $M^s(c\cdot q^+)\cdot x^s$ results from $c\cdot q^+$, by zeroing its s least significant terms. This can be explained as follows: the s most significant terms of $c\cdot g\cdot q^+$ are calculated by adding the s most significant terms of $c\cdot q^+$ with each other in as many offset positions as defined by the terms of the polynomial g. Thus, the s most significant terms of $c\cdot g\cdot q^+$ do not depend on the s least significant terms of $c\cdot q^+$, and consequently,

$$\begin{aligned} (19) \quad \Leftrightarrow \quad & M^s\big(g(x)\cdot M^s\big(c(x)\cdot q^+(x)\big)\cdot x^s\big) \\ &= M^s\big(g(x)\cdot q(x)\cdot x^s\big) \end{aligned} \tag{20}$$

Eq. (20) is satisfied for q given by:

$$q = M^s\big(c(x)\cdot q^+(x)\big) \tag{21}$$

Since there is a unique quotient q satisfying Eq. (10), there is a unique quotient q satisfying Eq. (20). Therefore, q must equal to $M^s(c(x)\cdot q^+(x))$.

It follows that the polynomial p can be found by

$$p(x) = L^t\big(g^*(x)\cdot M^s\big(c(x)\cdot q^+(x)\big)\big) \tag{22}$$

Eq. (22) can be used to design the following algorithm for computing p:

Preprocessing. Compute the polynomials g^* and q^+ from the given irreducible polynomial g. The polynomial g^* is of degree $t-1$, consisting of the t least significant terms of g, whereas the polynomial q^+ is of degree s, equals to the quotient of the division of x^{t+s} by the polynomial g.

Calculation of the remainder polynomial.

Step 1: Multiply the input c by q^+. The result is a polynomial of degree $2s-1$.
Step 2: Multiply the s most significant terms of the polynomial obtained in Step 1, by $g*$. The result is a polynomial of degree $t+s-2$.
Step 3: Returns the t least significant terms of the polynomial obtained in Step 2. This is the desired remainder.

This algorithm can be further sped up, if the special form of g (pentanomial) is taken into consideration. The quotient from the division of x^{256} with g is g itself. The polynomial $g = g(x) = x^{128} + x^7 + x^2 + x + 1$ is essentially the bit sequence $[1 : \langle 120\ \text{zeros}\rangle : 10000111]$. Multiplying this carry-less with a 128-bit value and keeping the 128 most significant bits can be obtained by: (i) shifting the 64 most significant bits of the input by 63-, 62- and 57-bit positions to the right. (ii) XOR-ing these shifted copies with the 64 least significant bits of the input. Next, we carry-less multiply this 128-bit result with g, and keep the 128 least significant bits. This can be done by: (i) shifting the 128-bit input by 1, 2 and 7 positions to the left. (ii) XOR-ing the results. Applying the above, we get the following algorithm (see Algorithm 2).

5. Handling the GCM bit-reflection peculiarity

The GCM standard specifies that the bits of the state are reflected. Here, for a 128-bit quantity Q, we define its bit reflection, BitReflect(Q), as the 128-bit quantity R whose i-th bit equals to the $(127-i)$-th bit of Q, $0 \leqslant i \leqslant 127$.

This bit reflection peculiarity applies to the two operands which are multiplied in $GF(2^{128})$. It also applies to the order of bits of the reduction polynomial (when it is represented as a 128-bit string), which become $[11100001 : \langle 120\ \text{zeros}\rangle : 1]$ (corresponding to $x^{128} + x^{127} + x^{126} + x^{120} + 1$), as opposed to $[1 : \langle 120\ \text{zeros}\rangle : 10000111]$ (corresponding to $x^{128} + x^7 + x^2 + x + 1$ as above).

We briefly mention here two approaches for handling the bit reflection peculiarity. One approach is to bit-reflect the inputs. In our case, the hash key fixed for the whole

Algorithm 2 (Fast reduction modulo g)

Input: 256-bit string $[X_3 : X_2 : X_1 : X_0]$ where X_3, X_2, X_1, X_0 are 64-bit long each.

Step 1: Shift X_3 by 63-, 62- and 57-bit positions to the right, to compute:

$$A = X_3 \gg 63$$
$$B = X_3 \gg 62$$
$$C = X_3 \gg 57 \quad (23)$$

Step 2: XOR A, B, and C with X_2. Compute D as follows:

$$D = X_2 \oplus A \oplus B \oplus C \quad (24)$$

Step 3: Shift $[X_3 : D]$ by 1-, 2- and 7-bit positions to the left. Compute numbers:

$$[E_1 : E_0] = [X_3 : D] \ll 1$$
$$[F_1 : F_0] = [X_3 : D] \ll 2$$
$$[G_1 : G_0] = [X_3 : D] \ll 7 \quad (25)$$

Step 4: XOR $[E_1 : E_0], [F_1 : F_0]$, and $[G_1 : G_0]$ with each other and $[X_3 : D]$. Compute a number $[H_1 : H_0]$ as follows:

$$[H_1 : H_0] = [X_3 \oplus E_1 \oplus F_1 \oplus G_1 : D \oplus E_0 \oplus F_0 \oplus G_0] \quad (26)$$

Output: $[X_1 \oplus H_1 : X_0 \oplus H_0]$ (the reduction result)

process and can be pre-reflected during its generation. The other inputs are the ciphertext blocks, and they need to be reflected as they are produced. An alternative approach avoids bit reflecting the inputs by using the following identity:

$$\text{Bit-reflect}(A) * \text{Bit-reflect}(B) = \text{Bit-reflect}(A * B) \gg 1 \quad (27)$$

We point out that bit reflecting the inputs can be the faster choice if the GCM tags are computed only for a single block each time. Fast reflection can be implemented by using an available byte-wise permutation instruction (called PSHUFB). However, if k tags, from k ciphertext blocks, are computed together, and the reduction is deferred to the end of the process, the use of (27) yields a faster implementation. Full details are provided in [13].

6. Concluding remarks

We presented here a new approach for implementing the GCM mode, using a carry-less multiplication instruction (called PCLMULQDQ) and two new algorithms for a "carry-less Karatsuba multiplication" and for reducing a binary polynomial of degree 254, modulo the $x^{128} + x^7 + x^2 + x + 1$.

A carry-less multiplication instruction can be used in various applications other than the GCM mode. Two examples are elliptic curve cryptography over binary fields (FIPS 186-2 [14,15]), and CRC computations using any polynomial. It is therefore a suitable addition to a general purpose processor.

We started our research in order to motivate adding a carry-less multiplication instruction to Intel processors, and by now, the study has already materialized. The new 2010 Intel Core processor family, based on the 32 nm Intel microarchitecture codename Westmere, introduce the carry-less multiplication instruction PCLMULQDQ together with new AES instructions. PCLMULQDQ performs carry-less multiplication in the SIMD domain. When the algorithms described here, are used on a Westmere processor, the performance of AES-GCM is 3.54 cycles/byte (the encryption and the authentication). This is more than six times faster than the state-of-the-art implementation [9], which when measured on the same platform, performs at 22 cycles/byte. The new results make an IA processor capable of supporting 10 Gbps AES-GCM processing by utilizing no more than two cores.

Complete details, different variants of the algorithms described here, and complete code examples for AES-GCM are provided in [13]. The AES instructions are introduced in [19]. Details on the AES instructions and their usage, in particular for AES in CTR mode, can be found in [16,17].

References

[1] M. Dworkin, Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) for confidentiality and authentication, Federal Information Processing Standard Publication FIPS 800-38D, April 20, 2006, available at: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[2] IEEE 802.1AE – Media Access Control (MAC) security, IEEE 802.1 MAC Security Task Group Document, available at: http://www.ieee802.org/1/pages/802.1ae.html.

[3] J. Viega, D. McGrew, The use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP), IETF RFC 4106, available at: http://www.rfc-archive.org/.

[4] IEEE project 1619.1 home, available at: http://siswg.net/.

[5] The fibre channel security protocols project, ISO-T11 Committee Archive, available at: http://www.t11.org/.

[6] T. Dierks, E. Rescorla, The TLS protocol version 1.2, RFC 5246, August 2008, http://www.ietf.org/rfc/rfc5246.txt.

[7] C.H. Lim, P.J. Lee, More flexible exponentiation with precomputation, in: Advances in Cryptography (CRYPTO'94), 1997, pp. 95–107.

[8] J. Lopez, R. Dahab, High speed software multiplication in F_2^m, in: Lecture Notes in Computer Science, vol. 1977, 2000, pp. 203–212.

[9] Brian Gladman, AES and combined encryption/authentication modes, Public Domain Source Code, available at: http://fp.gladman.plus.com/AES/index.htm.

[10] A. Karatsuba, Y. Ofman, Multiplication of multidigit numbers on automata, Sov. Phys. Dokl. 7 (1963) 595–596.

[11] P. Barrett, Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor, Master's Thesis, University of Oxford, UK, 1986.

[12] D. Feldmeier, Fast software implementation of error correcting codes, IEEE Transactions on Networking (1995).

[13] S. Gueron, M.E. Kounavis, Intel carry-less multiplication instruction and its usage for computing the GCM mode (Rev. 2), Intel Software Network, http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode/, 2010.

[14] Digital signature standard, Federal Information Processing Standard Publication FIPS 186-2, January 27, 2000, available at: http://csrc.nist.gov/publications/fips.

[15] C. Koc, T. Acar, Montgomery multiplication in $GF(2^k)$, Des. Codes Cryptogr. 14 (1) (April 1998) 57–69.

[16] S. Gueron, Intel Advanced Encryption Standard (AES) instructions set (Rev. 3), Intel Software Network, http://software.intel.com/en-us/articles/advanced-encryption-standard-aes-instructions-set/, 2010.

[17] S. Gueron, Intel's, new AES instructions for enhanced performance and security, Fast software encryption, in: 16th International Workshop (FSE 2009), in: Lecture Notes in Computer Science, vol. 5665, 2009, pp. 51–66.

[18] M.E. Kounavis, New method for fast integer multiplication and its application to cryptography, in: International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2007), San Diego, California (USA), July 16–18, 2007.

[19] S. Chennupaty, S. Gueron, V. Gopal, W.K. Feghali, M.E. Kounavis, M. Raghunandan, M.G. Dixon, Flexible architecture and instruction for advanced encryption standard (AES), U.S. Patent Application 20080240426, filed in 2007.