Available online at www.sciencedirect.com
SCIENCE DIRECT®
INTEGRATION, the VLSI journal 34 (2003) 1–64
INTEGRATION theVLSI journal
www.elsevier.com/locate/vlsi

Hardware architectures for public key cryptography ☆

Lejla Batina[a,b], Sıddıka Berna Örs[a,*], Bart Preneel[a], Joos Vandewalle[a]

[a] *K.U. Leuven ESAT/COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium*
[b] *SafeNet BV, Boxtelseweg 26a, 5261 NE Vught, The Netherlands*

Received 30 August 2002; accepted 30 August 2002

Abstract

This paper presents an overview of hardware implementations for the two commonly used types of public key cryptography, i.e. RSA and elliptic curve cryptography, both based on modular arithmetic. We first discuss the mathematical background and the algorithms to implement these cryptosystems. Next an overview is given of the different hardware architectures which have been proposed in the literature.

Keywords: Hardware architectures; Public key cryptography; RSA; Elliptic curve cryptography

1. Introduction

Growing demands for security are characterizing the vast majority of communication and computer systems. In order to secure a system, various cryptographic protocols are required. These protocols can be implemented in hardware and software. However, hardware appeared to be the ultimate choice considering security, as well as efficiency. Indeed, hardware implementations are faster in general, offering at the same time more intrinsic security. The "ideally designed" hardware should offer a scalable architecture to overcome the well-known drawback of limited flexibility.

In 1976, Diffie and Hellman introduced the idea of public key cryptography [1]. They used this concept to eliminate the need for prior agreement of a secret to exchange confidential information. Also, digital signatures were introduced which allow to uniquely bind a message to its sender.

☆ The work was partially supported by Concerted Research Action GOA-MEFISTO-666 of the Flemish Government.
*Corresponding author. Tel.: +32-1632-1885; fax: +32-1632-1969.
E-mail addresses: lejla.batina@esat.kuleuven.ac.be (L. Batina), siddika@esat.kuleuven.ac.be, siddika.bernaors@esat.kuleuven.ac.be (S. Berna Örs), bart.preneel@esat.kuleuven.ac.be (B. Preneel), joos.vandewalle@esat.kuleuven.ac.be (J. Vandewalle).

0167-9260/03/$ - see front matter
PII: S0167-9260(02)00053-6

Since then, numerous public-key cryptosystems have been proposed and all these systems based their security on the difficulty of some mathematical problem. The most prominent examples are RSA, named after its inventors Rivest, Shamir and Adleman [2] and Elliptic Curve Cryptosystems (ECC), which were proposed by Koblitz [3] and Miller [4]. When comparing these two most popular public-key cryptosystems, there are several aspects to be taken into account such as: security, key lengths, speed and implementation issues. For security, the hardness of the underlying mathematical problem is essential. It is important to point out that ECC offer equivalent security as RSA for much smaller key sizes. The reason is that all algorithms solving the mathematical problem on which ECC are based take fully exponential time. Other benefits include higher speed, lower power consumption and smaller certificates which is especially useful in constrained environments (smart cards, cellular phones, pagers, etc.). It also seems that it may be easier to secure ECC implementations against side channel attacks. The basic operation for ECC is point multiplication which relies on efficient finite field multiplication. Commonly used finite fields in ECC protocols are $GF(p)$ and $GF(2^n)$. The basic operation for RSA is multiplication in $GF(p)$. As a consequence, a substantial amount of research is focused on efficient and secure implementation of modular multiplication in hardware.

Schaumont and Verbauwhede presented the engineer's view on the security domain in the form of a security pyramid as shown in Fig. 1 [5]. The pyramid form represents the design space at multiple levels of abstraction. The most abstract representation of a cryptographic application is the security protocol architecture, which details what steps make up a secure communication. The next level represents the security algorithms. RSA and ECC are at this level. The operations used for RSA and ECC are derived from number theory and create the next level. Beyond the level of number theory we run into levels that deal with implementation issues. Finally, at the bottom level we express all aspects of a security algorithm in terms of targeted platform technology.

An introductory report about hardware implementations of different public key cryptography systems can be found in Abdelgurfi et al. [6] and in Beth and Gollmann [7]. A comparison of Public Key Cryptosystems (PKCs) for security and efficiency can be found in [8].

The remainder of this paper is organized as follows. Section 2 gives the requirements for implementations of public-key algorithms in numerous applications. In Section 3, we introduce

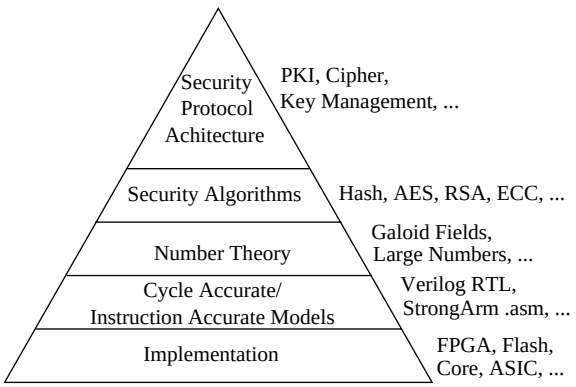

Fig. 1. Security pyramid.

the RSA cryptosystem, which is still the most popular public key cryptosystem. We present some commonly known algorithms for modular multiplication and exponentiation, which have advantages in hardware. We elaborate on the algorithm of Montgomery which is proven to be very efficient in avoiding the time consuming trial division and hence a common choice in the majority of architectures. We also discuss relevant implementations which are mainly based on systolic arrays. We conclude this section with the leading examples of RSA cryptosystems including the state of art in special applications, such as various Chinese Remainder Theorem (CRT) schemes. Section 4 introduces the basic details of the Discrete Logarithm based cryptosystems. We give the mathematical background and a number of examples in different types of basis. Section 5 describes the elliptic curve arithmetic and algorithms for the basic operations in a finite field. We list the most relevant ECC processors in both types of commonly used finite fields i.e., of characteristic 2 or some big prime number. Section 6 concludes the paper.

2. Requirements for PKC applications in hardware

As already mentioned, *scalability*, together with interoperability and security, represents one of the most important requirements of nowadays cryptographic applications. That property results in increased flexibility of hardware which was usually associated only with the FPGA solutions. Details can be found in the presentation by Paar [9]. According to Tenca and Koç [10], an arithmetic unit is called scalable if: "the unit can be reused or replicated in order to generate long-precision results independently of the data path precision for which the unit was originally designed". More precisely, the longest path should be "short" and independent of operands' length and designed in such a way that it fits in restricted hardware regions, as defined by Gutub et al. in [11,12]. This means that the arithmetic unit can handle arbitrary bit-lengths with the exception of memory limitations and the number of clock cycles per operation is dependent only on the actual size of the operands. Typical scalable architecture has performance graphs as presented in Fig. 2.

Considering bit-lengths required for RSA and ECC, ECC keys are much shorter than RSA keys. In short, for RSA 1024 bit-lengths are still in use but it is assumed to upgrade those to 2048 and even 4096 bits. On the other hand ECC keys are expected to stay in the range from 160 to 256 bits within the next 10–15 years. Lenstra and Verheul [14] showed that 1937-bit key size RSA may be considered to have an equivalent security as 190-bit key size ECC. In the same work, the authors estimated that 190-bit ECC keys are expected to be suitable for commercial security in the year 2021.

Implementation attacks exploit weaknesses in specific implementations of some cryptographic algorithm. Undesired result is often some leakage of side-channel information, which is usually correlated to the secret key. There are various types of information that might leak from an implementation. Usually we distinguish between *invasive* and *non-invasive* attack.

Microprobing techniques are example of invasive attacks. These techniques can be used to access the chip surface directly. Thus one can observe, manipulate and interfere with the integrated circuit as explained by Anderson and Kuhn in [15,16]. They usually require several hours of work in a specialized laboratory and during the process the packaging is destroyed.

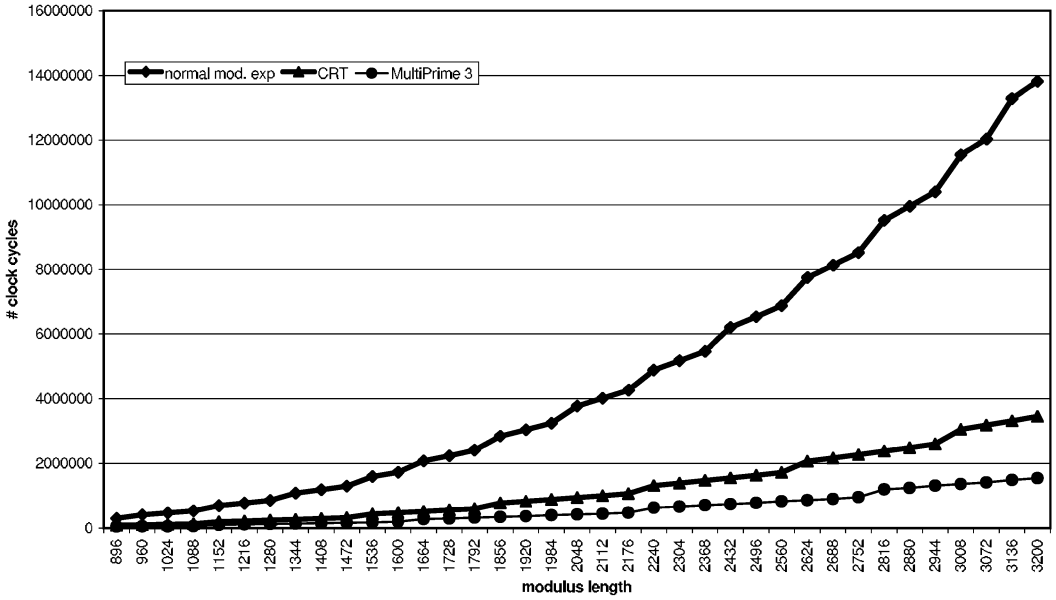

Fig. 2. Performance of modular exponentiation for three different types of RSA technology [13].

Smartcard processors are particularly vulnerable to non-invasive attacks, because the attacker has full control over the power and clock supply lines. Examples are: glitch attacks [16], attacks using electromagnetic radiation [17], timing attacks [18,19], power analysis attacks [20,21], etc.

A *timing attack* uses the amount of time to run a cryptographic algorithm as a basis for the attack. These attacks were first mentioned by Quisquater in 1989. In 1996 Kocher [19] showed how to attack both symmetric-key and public-key algorithms with this type of attack.

The most recent type of non-invasive attacks, the so-called *differential power analysis attacks*, are discovered by Kocher and his associates. The basic idea behind this type of attack is the observation that the power consumed at any particular time during a cryptographic operation is related to the instruction and (possibly sensitive) data being processed. There are two basic attacks: simple power analysis attack (SPA for short) and differential power analysis attack (DPA). While SPA allows for extracting sensitive information easily, requiring only a single (or a few) power-consumption graphs, DPA is more demanding. It consists of performing a statistical analysis of many executions of the same algorithm with different inputs. DPA exploits the correlation between power consumption and specific key-dependent bits which are used at some steps of the protocol. We can say that DPA is more powerful than SPA but also is a "high-cost" attack in the sense of feasibility. Both attacks, especially combined together or with other attacks are serious threats to smart card security. A special type of attack called *fault attack* influences either external (power supply, clock) or internal parameters (voltage or light on a single cell or transistor) to modify the state of the circuit. This attack was introduced by Boneh et al. in 1997 [22]. The idea behind this type of attack is the fact that from time to time the hardware performing the computation may introduce errors.

While all attacks on and defenses of cryptographic hardware get increasingly sophisticated, experience has learned that protecting hardware implementations of cryptographic algorithms is easier than protecting software only implementations.

3. RSA cryptosystem

3.1. The algorithm

The RSA cryptosystem was invented by Rivest et al. in 1977 [2]. It is a widely used system for providing privacy and ensuring authenticity of digital data, i.e., in all applications where security of digital data is a concern. RSA is still the most popular cryptosystem, especially for high-end devices that are typically used in e-commerce and virtual private network (VPN) servers. RSA is based on modular exponentiation.

The private key of a user consists of two large primes p and q and an exponent d. The public key consists of modulus $N = p \times q$ (at least 1024 bits) and an exponent e such that $e = d^{-1} \bmod lcm((p-1),(q-1))$. To encrypt a message M the user computes: $C = M^e \bmod N$ and decryption is done by: $M = C^d \bmod N$. This equality follows by Euler's theorem (see [23]). The RSA function is usually defined as $x \mapsto x^e \bmod N$, and private exponent d is referred to as a trapdoor enabling one to invert the function. Since its invention RSA was carefully scrutinized and no devastating attack has been found in the sense of breaking RSA by inverting its function. Some choices have to be avoided mainly considering required bit-lengths but it seems that proper implementation can still be trusted. Boneh [24] gives a nice overview of attacks on RSA. It is evident that modular exponentiation and also modular multiplication are the most important operations which have to be considered in detail.

3.2. Modular exponentiation

The dominant cost operation in the RSA cryptosystem is modular exponentiation, namely computing $M^e \bmod N$. The basic technique for exponentiation is repeated square and multiply (see [25]). In [26] this method is called left-to-right binary exponentiation (see [26]). A similar algorithm is also used for point multiplication in ECC. In this case the analogous scheme is called double-and-add (see [27]).

Algorithm 1. Modular exponentiation

INPUT: integers $0 \leqslant M < N$, $0 < e < N$, $e = (e_{t-1}, e_{t-2}, \ldots, e_0)_2$
$e_{t-1} = 1$ and N
OUTPUT: $M^e \bmod N$
1. $A \rightarrow M$
2. For i from $t-2$ to 0 do:
   2.1 $A \rightarrow AA \bmod N$
   2.2 If $e_i = 1$, then $A \rightarrow AM \bmod N$
3. Return (A)

Numerous methods for speeding-up exponentiation and point multiplication have been proposed in the literature; for a survey see [28]. Recently, side-channel security is also considered to be an important factor for the choice of a suitable exponentiation algorithm. These methods are divided into three large groups: generic methods i.e., methods which can be applied to any finite abelian group, exponent recoding techniques and special methods developed for particular operations (as in [29]). We will give a short description of these methods.

3.2.1. Generic methods for exponentiation

These methods can be widely applied in all types of groups hence also in ECC algorithms.

Mostly mentioned are various *windowing techniques* as a generalization of the basic algorithm in which more than one bit of the exponent is processed per iteration. The basic idea is as follows: the exponent is divided into digits (windows). Algorithm 2 can thus be considered as a special case where the window size is equal to 1. For the sliding window methods these windows do not necessarily have the same length. The exponent is divided into zero and non-zero windows. In the work of Koç [30] different variants of the sliding window method are proposed. These methods also include a precomputation phase in which some powers of M are precalculated. The number of pre-calculations varies according to the window size. Considering different parameters various trade-offs are possible.

A similar idea is used in the *comb techniques*, which precompute tables that depend on M [31]. These methods are very useful for applications with fixed base M such as signature schemes.

When the exponent is fixed, it is very useful to use *addition chains* [26]. Namely, subtractions have the same cost as additions, so it is possible to combine both operations for point multiplication on elliptic curves. The purpose of addition chains is to minimize the number of multiplications required for an exponentiation. A comprehensive and detailed discussion can be found in [25].

3.2.2. Exponent recoding techniques

Exponent recoding techniques replace the binary representation of an exponent with a representation which has fewer non-zero terms (see [32]). Many techniques for exponent recoding have been proposed in the literature. In this section we will discuss *signed-digit representation*.

Signed digit representation. Consider an integer representation of the form $k = \sum_{i=0}^{l} s_i 2^i$, where $s_i \in \{-1, 0, 1\}$. This is called the (binary) *signed digit* (SD) representation [26]. The representation is redundant. For example, the integer 3 can be represented as $(011)_2$ or $(10\bar{1})_2$, where $\bar{1} = -1$. It is said that an SD representation is *sparse* if it has no adjacent non-zero digits. A sparse SD representation is also called a *non-adjacent form* (NAF). Every integer k has a unique NAF which has the minimum weight of any signed digit representation of k.

3.2.3. Special methods for exponentiation

Many special techniques have been proposed which take advantage of some special property of an underlying field, or an elliptic curve, etc. As an example, Walter introduced the MIST algorithm [33,34], which is claimed to offer a good protection against power analysis-based attack without losing too much on efficiency. His method selects are random between different short windows.

3.3. Modular multiplication

Modular multiplication forms the basis of modular exponentiation which is the core operation of the RSA cryptosystem. It is also present in many other cryptographic algorithms including those based on ECC. The most popular algorithm for modular multiplication is Montgomery's method [35].

3.3.1. Montgomery's multiplication method (MMM)

In 1985 Peter Montgomery introduced a new method for modular multiplication [35]. The approach of Montgomery avoids the time consuming trial division that is the common bottleneck of other algorithms. His method is proven to be very efficient and is the basis of many implementations of modular multiplication in hardware as well as software.

The notation is as follows:

$$Mont(X, Y) = XYR^{-1} \bmod N. \tag{1}$$

For a word base $b = 2^{\alpha}$, R is usually chosen such that $R = 2^{r} = (2^{\alpha})^{l} > N$. There is a one-to-one correspondence between each element $x \in \mathbb{Z}_N$ and its representation $xR \bmod N$. This representation is usually referred to as Montgomery representation. The method requires conversion of x and y to an N-residue domain and conversion of the calculation result back to the integer domain. The procedure is as follows. To compute $Z = xyR \bmod N$, one first has to compute the Montgomery function of x and $R^2 \bmod N$ to get $Z' = xR \bmod N$. $Mont(Z', y)$ gives the desired result. When computing the Montgomery product $T = XYR^{-1} \bmod N$, the following procedure was proposed [26]:

Algorithm 2. Montgomery Modular Multiplication

INPUT: Integers $N(odd), x \in [0, N-1], y \in [0, N-1], R = 2^{r}$, and $N' = -N^{-1} \bmod 2^{r}$
OUTPUT: $xyR^{-1} \bmod N$
1. T $\leftarrow 0$. (Notation $T = (t_l t_{l-1} \ldots t_0)$)
2. For i from 0 to $(l-1)$ do:
   2.1 $m_i \leftarrow (t_0 + x_i y)N' \bmod 2^{\alpha}$
   2.2 $T \leftarrow (T + x_i y + m_i N)/2^{\alpha}$
3. If $T \geqslant N$, then $T \leftarrow T - N$
4. Return (T)

In the original notation of Montgomery after each multiplication a reduction was needed (step 3 in the algorithm above). The input had the restriction $X, Y < N$ and the output T was bounded by $T < 2N$. The result of this is that in the case $T > N$, N must be subtracted so that the output can be used as input of the next multiplication. (As modular exponentiation consists of repeated modular multiplications.) To avoid this subtraction a bound for R is presented by Walter in [36] such that for inputs $X, Y < 2N$ also the output is bounded: $T < 2N$.

The work of Walter offers many other useful results for Montgomery's techniques. In [37], which is further improved in [36], he showed that the Montgomery exponentiation method

requires no final subtraction, which is very important for fast implementation. Another benefit is that conditional statements, which may be subject to side-channel attacks such as timing attack, power analysis attack etc. may be omitted. Some other results considering constant time implementations which is presumed to be a first step towards secure hardware solutions are proposed in [38].

A bound on R has to be found such that with $X, Y < 2N$ the output of the Montgomery multiplication $T < 2N$. Batina and Muurling [13] found this bound as follows: write $R \geqslant kN$, then:

$$T = \frac{XY + mN}{R} = \frac{XY}{R} + \frac{m}{R}N < \frac{4}{k}N + N \tag{2}$$

where, $m = (XY \bmod R)N' \bmod R$.

Hence, $T < 2N$ for $k \geqslant 4$, implying: $4N \leqslant R$. To guarantee the existence of the modular inverse of R, R and N should be relatively prime. This excludes $4N = R$. The final round in the modular exponentiation is the conversion to the integer domain, i.e., calculating the Montgomery function of the last result and 1. The same arguments as above prove that this final step remains within the following bound: $Mont(T, 1) \leqslant N$.

Koç et al. discussed five different Montgomery multiplication algorithms in the [39]. For all five methods the space and time requirements have been analyzed in detail. For a general class of processors the most efficient method is identified. The algorithms differ on the basis of two factors. The first factor is whether multiplication and reduction are *separated* or *integrated*. In the separated approach, reduction is done after multiplication. In the integrated approach, the algorithm alternates between multiplication and reduction which can be either *coarse-grained* or *fine-grained*, depending on how often one switches between these two operation. The second factor is the type of scanning which can be either *operand scanning* or *product scanning*. As the most efficient method, the authors identified the coarsely integrated operand scanning.

An architecture based on Montgomery's algorithm is probably the best studied architecture in hardware. Differences appeared because of a different approach for avoiding long carry chains. Most common ways to do so are: systolic array and redundant representation.

3.3.2. Other algorithms

The basic algorithm for computing $A \cdot B \bmod N$ (starting from the MSB) repeatedly adds to a running total of R, the product B^* of the next digit of multiplicand A with the multiplier B. After that, a shift up is performed. At the end the result R is reduced by a multiple of the modulus N to yield the residue. This procedure can be sped up with interleaving modular subtraction by repeated shift and add. Then R stays roughly the same as the size of the modulus which saves the register space. Eldridge and Walter proposed other techniques for speeding up modular multiplication in [40]. In comparison with Montgomery's algorithm, this algorithm: reverses the order of handling the digits of multiplicand A, performs a shift up instead of down and does a subtraction rather than an addition. A detailed comparison between these two algorithms with respect to hardware is presented in the work of Walter [41].

3.4. Architectures for RSA

Soon after its invention, the first proposals for RSA hardware implementations appeared, such as [42] by Rivest. In the past two decades different architectures were proposed. The systolic array architecture, that was proposed already in 1965 by Atrubin [43], still appears to be the best solution for modular multiplication with very long integers. This architecture has been studied intensively, both from a theoretical and a practical viewpoint.

3.4.1. Systolic array

A systolic array is typically defined as a grid-like structure of special processing elements (PEs) that processes data much like an n-dimensional pipeline (see [44]). Each line indicates a communication path and each intersection represents a cell or a systolic element. One simple example is given in Fig. 3 which represents a one-dimensional systolic array.

The systolic array architecture, as an ideal candidate for computationally intensive operations, is often used for RSA implementations. Nevertheless, these types of architectures can be used in various applications of PKC. In 1992 Dixon and A. Lenstra [45] used a systolic array architecture to build hardware for the elliptic curve factoring method (ECM) which was proposed by H. Lenstra in [46]. The factoring algorithms are of continuing interest for the analysis of various public-key cryptosystem for obvious reasons. It is clear that factorization of the modulus results in recovery of the secret key, and hence in breaking RSA. The ECM has expected run time dependent on the size of the factors which makes this method very suitable for breaking various "multi-factor" schemes. The systolic array-based hardware presented in [45] is consisting of 128×128 array of processing elements (PE array). Each of the PE's can carry out approximately 2×10^5 additions/s on 32 bit integers and has 64 KB of memory. The implementation is massively parallel and based on Montgomery's multiplication.

Various systolic arrays for modular multiplication have been proposed in the past ten years, for example [40,47–50], but very few practical implementations have been reported to our knowledge (see also [51]). Some of the proposals are 1-D arrays such as [52–54], others are 2-D arrays (e.g., [49,55]).

Even was the first one to realize that a systolic array combined with Montgomery's multiplication facilitates cryptographic systems [56]. He used the Atrubin's multiplier [43] to which he added a systolic array for modular reduction. This modular reduction circuit was required for division by Montgomery's parameter R.

Other early proposals for hardware for Montgomery's multiplication method (MMM) include [40] by Eldridge and Walter and [52] by Kornerup.

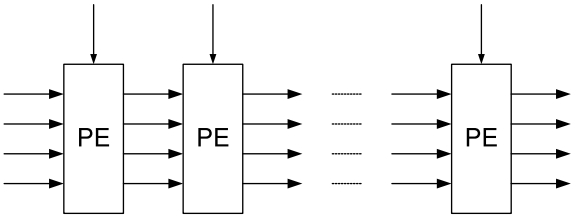

Fig. 3. Systolic array (one-dimensional) with processing elements (PEs).

Eldridge and Walter have observed that the algorithm of Montgomery in a systolic array offers an ideal solution for RSA cryptography. The reason is that in this case the modular correction and carry signals are similarly corrected. Before this work, it was believed that redundant number systems were required, because of the clash in the direction between the movement of the carries and that of the multiple of the modulus. This was resolved by Montgomery's reversing the multiplication order i.e., choosing digits from least to most significant and shifting down instead of up.

However, the work of Iwamura et al. [57] is the first one to our knowledge presenting the systolic array which can execute a modular exponentiation operation using Montgomery modular multiplication. They also proposed a novel algorithm for modular exponentiation without subtracting N for every MMM. In this way an output of each MMM can be directly fed back into the next MMM. The same authors were also first to introduce the idea of a scalable architecture [48]. They did not name this idea scalability, but they suggest the design's ability to cope with various precision in bits.

The first definition of scalability is given in [10]. The authors introduced a pipelined Montgomery multiplier, which has the ability to work on any given operand precision and is adjustable to any chip area. The first feature they call scalability and treat as unique in comparison to other designs. Apparently, the architecture described in that work is not purely systolic and has the flavor of a serial–parallel implementation.

Systolic arrays based on Montgomery's algorithm. The systolic array presented by Walter [49] has a throughput of one modular multiplication every clock cycle and a latency of $2n+2$ cycles for n-digits multiplicands. A full, rectangular, systolic array with one row for each addition step is described. The number of rows is as large as necessary to complete a modular multiplication. Further the author discusses possibilities of fewer rows and mentions the linear form as a single row which would perform two multiplications in parallel by feeding the output directly back in. In [40] this architecture is compared with previous techniques showing that this method is up to twice as fast and more suitable for hardware than other techniques (proposed by Brickell [58], Orup [59], etc.) Decrease of the depth of combinatorial logic implied the ability to use a faster clock speed.

In [47] Walter shows how to adapt the array [49] to modular exponentiation without having to buffer results or idle between successive multiplications. As further improvement PEs can be reduced from two multipliers to one which makes them work on every cycle instead of only on alternate cycles. This array was an alternative array to linear arrays of Kornerup [52] and Jeong and Burleson [55].

A linear, purely systolic array forming a digit-serial multiplier was introduced by Kornerup [52]. He was also inspired by the array of Atrubin [43] which is linear systolic array of the similar type, but with a much higher cell complexity. The linear array of Kornerup performs multiplication along slanted lines (see Fig. 4). In this way each cell forms and accumulates two terms from each column in each cycle. Therefore, only $\lceil n/2 \rceil$ cells are needed and the result of multiplication is produced in $2n$ cycles. As it can be observed from Fig. 4 each cell communicates only with the nearest neighbors, thus making this array suitable for large multipliers operating at a very high clock frequency. It is also shown how the multiplier can perform modular division and facilitates the modular inversion. This work offers some original ideas in order to fully utilize the array. Two halves of his PEs are running in parallel. His array has superior latency but requires

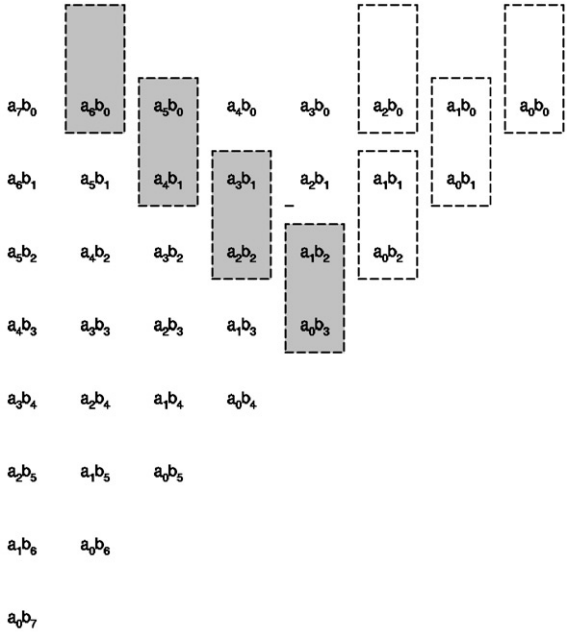

Fig. 4. The organization of systolic array of the architecture [52].

two copies of the array for efficient exponentiation. Hence *area* × *time* complexity is thus similar to that of Walter.

Iwamura et al. [48] proposed two types of systolic arrays that perform the Montgomery method without conditional subtraction of the modulus. The systolic arrays consist of a fixed number of PEs. These two new arrays were compared with their previous proposal [60] and the conclusion was that hardware based on Montgomery method is more efficient then systolic arrays based on non-Montgomery modular multiplication. Here, the hardware efficiency is defined as a ratio of processing speed and circuit scale (# PEs). The inputs A and B, T (result of $Mont(A,B) = ABR^{-1} \bmod N$) and modulus N are divided into digits, where digit-length of A, say d is bounded from below with the digit-length of remaining parameters, say v (hence $v \leqslant d$). However this requirement is not affecting the flexibility they claim. The processing speed can be further improved by increasing the number of chips which are supposed to be realized as some kind of a cascade connection. The authors have also considered the usual bottleneck for hardware implementations of Montgomery's algorithm, i.e., the fact that the number of output bits may exceed the number of input bits. They derived the bound $R > 2^{n+2}$ for $R = 2^r$ and concluded that $r = n + 2$, is the minimum possible value for which the examination of the size of the output each time the Montgomery method is executed, may be omitted. Here, n is the maximal number of bits for N, so $N < 2^n$. This bound can be further improved to the condition $R > 4N$, which is according to work presented in [36] by Walter, proved to be the best possible bound in practice.

The work of Tiountchik and Trichina [61,62] aims to design a pure systolic array such that the complete exponentiation process can be carried out by a single systolic unit. They used the ideas of Walter and Eldridge [49,40] to construct the array that does not need feeding back output as a

new input nor joining separate units for multiplication and squaring. The dependence graphs (DGs) were created for Montgomery multiplication and for squaring separately. The chain of DGs for these two stages forms a new DG for Montgomery based exponentiation. Hence, each PE has to be able to operate in two modes. To control the operation modes, one bit control signal is fed into the rightmost PE and then pumped through the array.

Considering modular exponentiation on FPGAs, more recent work include [63–65]. Elbirt and Paar evaluated some options in FPGA architecture suitable for PKC in [66]. First of all, redundant representations and systolic array implementations are both found to be suitable for high-speed addition algorithms. This results in a performance increase for modular multiplication and therefore for cryptographic algorithms.

In [64], Blum and Paar have derived a modular exponentiation architecture based on Montgomery's method on a single FPGA (for bit lengths of up to 1024 bits). The Montgomery parameter R is set as $R = 2^{n+3}$ instead of $R = 2^{n+2} > 4N$. In this way, the number of repetitions for Montgomery's algorithm is $n + 3$ for radix 2 implementations. Still, they have avoided the originally proposed final comparison and subtraction by performing a pipelined exponentiation algorithm. The PEs are consisted of $u = 4, 8, 16$ bits and only n/u PEs are used. Similar to the approach by Kornerup [52], squarings and multiplications are performed in parallel. A full modular exponentiation is computed in $2(n + 2)(n + 4)$ clock cycles with a latency of n/u clock cycles. Note that Blum and Paar have also reported an implementation with high-radix in [65]. In order to speed-up the design they describe a high radix version of Montgomery's algorithm which reduces the amount of cycles per modular multiplication. A full modular multiplication of $2(n + 2)(m + 10)$ clock cycles with an n-bit exponent and an m-digit modulus is achieved. For an optimal speed area trade-off a radix of 16 was chosen.

Su et al. [53] rewrote the algorithm of Montgomery in order to execute modular multiplication two times faster. Each iteration in this algorithm requires only one addition, compared with two of Montgomery. The number of iteration remains unchanged resulting in the doubled speed for modular multiplication. The proposed algorithm is implemented by a 2's complement multiplier and a modular shifter-adder, both of which are designed as linear cellular arrays with n cells for a word length of n. Each cell contains one full adder and some controlling logic. They also proposed a suitable modular exponentiation algorithm which calculates a modular exponentiation in $2n^2$ clock cycles.

Two new systolic architectures are proposed in [54] to improve the computation speed of modular multiplication. The comparison with some other prominent examples such as those of Walter and Kornerup indicates that this proposal offers the highest speed, lower hardware complexity and lower power consumption (for 1024-bit RSA). However, as the issue of scalability was not addressed the remaining question is how it would perform for diverse bit-lengths as required in nowadays applications.

Other systolic array, non-Montgomery. As Montgomery's algorithm showed to be the best option by far in hardware, there were not so many other proposals. Some of them are worth mentioning such as those of Iwamura et al., Freking and Parhi, etc.

Iwamura et al. considered in [60] two implementations of RSA cryptography in hardware using modular multiplication other than the one of Montgomery. They tried to create a flexible architecture, where they defined flexibility as "the degree of linearity between the processing speed and the circuit scale". The number of PEs on a chip was therefore optional to facilitate the choice

for a favorite scale. However, in their later work [48] they conclude that this array is not as efficient as the hardware based on the method of Montgomery.

Jeong and Burleson [55] described two algorithms for modular multiplication and their array structures. The algorithms are based on the iterative Horner's rule and the proposed arrays can be fully pipelined. However, these linear arrays require 50% more cycles per multiplication than the array of Walter [47]. Only one multiplication is also performed at once, comparing with two in [47] and although they use the same number of PEs as Walter, they are more complex. Thus their PEs only operate on every third cycle.

Freking and Parhi [67] proposed three new architectures and compared all of them. They obtained a linear performance-area trade-off. Signal broadcasting is eliminated due to bit-level based systolic array thus providing a possibility for very high clock rates. The authors have achieved a scalable array without taking advantage of the algorithm of Montgomery. Three new architectures they named as follows: cascade, cylindrical and higher-radix. The first two methods are based on binary modular multiplication. The cylindrical array is a 2-D array which deploys a feedback pipelining technique. An alternative approach to obtain scalability is based on a parallelization of a partitioned computation which is used in the array cascade approach. This array consists of a cascade of linear arrays, each assigned to process one partition. When comparing these two architectures, it is obvious that the cylindrical array exhibits some pipelining overhead which is overcome with the array cascade method. Therefore, the cylindrical array does not feature constant average computation time for some bit-length n. It is also dependent on the number of rows in the array. For the third array scalability is achieved through adjustment of the radix. However, as in [59], the radix of the implementation will not be matched to the algorithmic radix in order to prevent exponential cell growth. The algorithm chosen instead of Montgomery was invented by the authors in 1999 [68] and named ''Power-of-Two Radix IRA algorithm''. Namely, they considered the algorithm of Montgomery being not so well-suited for systolic high-radix computations.

3.4.2. *Non-systolic array*

Architectures based on Montgomery. The use of high radices in modular multiplication was for a long time considered not to be an efficient solution due to complex determination of quotient digits for the modular reduction (see [40]). The algorithm of Montgomery did not change that perception completely, but some signs of possible improvements were obvious. Namely, the high radix approach gives potential for substantial speed improvements of modular multiplication although often criticized for a large hardware depth (meaning a slow clock frequency). Hence, a vast majority of reports from this area was oriented on systolic arrays.

There are only few Montgomery based architectures which are not systolic arrays. One of the most mentioned is the one of Orup. He introduced rewritten MMM, suitable for high radices where the obtainable clock frequency was meant to be independent of the choice of radix. Orup [59] presented an alternative to systolic architecture where bit-level, carry-save arithmetic was utilized to compute high-radix modular multiplication. An example of this architecture was given for the radix 2^8 with 3 pipeline stages. The partial result adder was composed of 4-2 carry–save adders rather than a high-radix component. Scalability was targeted through algorithmic radix adjustment (similar to Freking and Parhi [67]), but the design is characterized by low clock rates due to global broadcast signals.

Marnane [69] presented an architecture for bit serial modular multiplication in FPGAs. It is also a scalable architecture due to the re-programmability of FPGAs.

We will mention one more recent work by Kim et al. [70]. They are also using the value of 2^{n+2} for R in MMM to achieve reuse of the result as an input for the next modular multiplication (while performing exponentiation). The entire system of their 1024-bit RSA processor is composed of the REDC part, which calculates $AB \cdot R^{-1}$ and the control part. The REDC part computes the Montgomery modular multiplication and the control part controls all inputs and outputs of the REDC. Since this architecture is not systolic, the addition could not be operated by a full adder (then only one clock is needed). Hence, this architecture requires 1024-bit registers to store the operands in computing the addition. This would increase the hardware resources so instead they used 32-bit carry propagation adders (CPA) with a shift register. More precisely they used two 32-bit registers, 2 multiplexors and a 32-bit CPA, which is smaller then the previous solution. This solution requires 32 additional clocks, but hardware requirements are minimized which reduces the chip size substantially. They achieved a timing of 43 ms (at most) for a 1024-bit RSA operation. The lack of scalability is probably the most important disadvantage of this proposal.

Other-non-Montgomery. In 1981, Norris and Simmons [71] proposed *delayed-carry adder* to produce a hardware modular multiplier which computes the product of two t-bit operands modulo t-bit modulus in $2t$ clock cycles. Brickell [58] improved this concept to finish a modular multiplication in only $t + 7$ clock cycles. A circuit for Brickell's multiplier is described in the work of Walter and Eldridge [72].

Orton et al. gave a nice collection of modular multiplication algorithms in [73]. They presented the implementation results of one of the algorithms which was most efficient according to their results. The circuit included three K-bit adders, K is the number of bits of modulus. It could finish one modular multiplication in $K + 2$ clock cycles and had the throughput of 40 kb/s at a clock speed of 200 kHz. By using the modular multiplication block a RSA chip was produced by using 3 μm CMOS technology and shown to correctly perform RSA encryption (or decryption) at a clock speed of 200 kHz, which corresponds to a rate of 4 kb/s. In the same work the authors presented a standard basis serial finite field $GF(2^m)$ multiplier implementation.

In 1987, Sedlak [74] proposed a RSA cryptography processor (CP). In this architecture exponentiation is implemented as a sequence of multiplications and multiplication is implemented as a sequence of additions. Also the modulo operation is implemented as a sequence of subtractions. The difference of this approach and the classical algorithm is in the look-ahead algorithms which decrease the maximum number of additions for multiplication and subtractions for modulo operation are used. Because a realistic algorithm cannot look ahead an unbounded number of bits, probabilistic functions are used to define the ciphering rate.

In 1988, Hoornaert et al. [75] described a hardware implementation of the RSA algorithm. They improved the classical double-and-add algorithm for modular multiplication. The result was a more complex but more efficient circuit. They designed a 120-bit chip and chips could be concatenated to arbitrary bit-lengths.

Vandemeulebroecke et al. [76] presented a single chip 1024-bit RSA processor in 1989. They used redundant signed digit (RSD) arithmetic in an RSA computation for the first time. In the RSD representation, a number X can be viewed as the difference between two positive binary

numbers X^* and X^{**} as shown follows:

$$X = \sum_{i=0}^{n} x_i 2^i = \sum_{i=0}^{n} (x_i^* - x_i^{**}) 2^i, \quad x_i^*, x_i^{**} \in \{1, 0\}. \tag{3}$$

As it can be seen from the above equation, each digit x_i belongs to the set $\{1, 0, \bar{1}\}$ where the upper bar indicates a negative value. Let $Y = Y^* - Y^{**}$ be a number in redundant form, Z a number in two's complement form and $S = S^* - S^{**}$ the result of the operation of $Y + Z$ or $Y - Z$, with $y_i^*, y_i^{**}, z_i, s_i^*, s_i^{**} \in \{0, 1\}$. The two's complement of Z can be found as follows:

$$\text{if} \quad Z = \sum_{i=0}^{n} z_i 2^i = (z_n z_{n-1} \ldots z_0)$$

$$\text{then} \quad -Z = \sum_{i=0}^{n} (1 - z_i) 2^i + 1 = (1 - z_n 1 - z_{n-1} \ldots 1 - z_0) + 1. \tag{4}$$

So a subtraction reduces to an addition with an inversion of all bits and adding of 1. The computation scheme can be implemented with one level of full adders. The addition is fully parallel, thus requiring no carry propagation.

The multiplication and modulo operations are implemented by using repeatedly addition and subtraction respectively. Considering a unity add time for the addition, both multiplication and modulo are ideally performed in n cycles. The main features of the RSA chip are given in Table 1.

Findlay and Johnson proposed recursive sums of residues technique for modulo reduction in [77]. If the number P, to be reduced by the modulus m, is expressed as follows:

$$P = [p_1, p_2, \ldots, p_{2n}], \quad p_i \in \{0, 1\} \quad \text{for } i = 1 \text{ to } 2n. \tag{5}$$

then the modulo reduction can be written as

$$P \bmod m = \left(\sum_{i=1}^{2n} p_i 2^{i-1} \right) \bmod m$$

$$\left(\sum_{i=1}^{2n} p_i (2^{i-1} \bmod m) \right) \bmod m. \tag{6}$$

The reduction is simply a conditional sum of powers of 2 reduced modulo m: *residues*, hence the name of sums-of-residues (SOR) reduction. Details of the design steps of the SOR architecture is

Table 1
Main features of the RSA chip

Technology	CMOS 2 μm with 2 metal layers
Transistor count	180,000
Chip size	80 mm^2 (9.3 × 8.7)
Power dissipation at 25 MHz	500 mW under 5 V
Baudrate (all 1024 bits words)	8 kbits/s with a 25 MHz clock
Key storage	Static, 2 pairs on the chip
I/O	Serial or 8 bits parallel, asynchronous
Control	On chip

given in [77]. A hardware modulo exponentiator system consists of a hardware bit serial multiplier, followed by a SOR reduction unit.

Morita proposed an algorithm based on higher radix than 2 in [78]. The proposed algorithm is carried out by using the following equation repeatedly.

$$R_{(k-1)} \leftarrow rR_{(k)} + b(k)A - c(k)N, \tag{7}$$

where "k" is the step number of repeated processing, "r" is the radix number, $R_{(k)}$ and $R_{(k-1)}$ are partial remainders, $b(k)A$ is a partial product and $c(k)N$ is a modular subtractor. The algorithm is implemented in hardware for radix 4. It is reported in [78] that the 512-bit modular multiplier had about 50 Kgates and a delay time of about 8 μs. The delay time for 512-bit modular exponentiation is reported as 6 ms.

Orup et al. proposed a hardware implementation of Eq. (7) [79] where they estimated $c(k)$ according to δ most significant bits of $R_{(k)}$. They gave the simulation results for timing of one multiplication as 85 ns. The estimated area is reported as 100 mm^2.

Takagi and Yajima proposed to use the RSD representation of multiplicand and multiplier to fasten the additions in Eq. (7) [80,81]. Takagi gave also the hardware implementation results of his technique. He reported that a serial–parallel modular multiplier based on the proposed algorithm had a regular cellular array structure with a bit slice feature. The depth of its combinational circuit part is a constant independent of n, bit length of the modulus. The gate count of the combinational circuit part is about $19n$. The total number of bits for the registers is about $7n$.

Walter proposed several techniques for speeding up modular exponentiation. The first work is from 1991 and he used modulus scaling to calculate Eq. (7) [82]. In this approach N is scaled by a factor f such that fN has its q most significant digits fixed. The algorithm uses fN instead of N. Estimation of $c(k)$ is easier because it no longer depends on any q most significant digits of N, as they are fixed. It is claimed in this work that this shortens the cycle time of each iteration and gives a speedup factor of 70%. The second work was in the same year and it was about an algorithm which calculates a residue R and an integer quotient Q satisfying $A \times B = M \times Q + R$ [83]. R is either the smallest non-negative residue of $A \times B \bmod M$ or differs by at most M from it. $A \times B \bmod M$ is calculated as $(AS \times B \bmod MS)/S$ for a fixed r-power $S = r^E$. Henceforth, Eq. (7) becomes

$$\begin{aligned} Q_k &= ApproxQuot(rRS, MS), \\ RS_k &= 2RS + AS(k)B - Q_k MS, \end{aligned} \tag{8}$$

where "k" is the step number of repeated processing, "r" is the radix number. In the paper different functions for *ApproxQuot* are investigated and a solution is given.

He also proposed to split the computation of $(A \times B) \bmod M$ into six distinct phases in $rnd(fract((A \times B) \times (1/M)) \times M$ where *fract* discards the integer part of a real number, and retains the non-negative fractional part, and *rnd* rounds a real to the nearest integer [84]. It is assumed that M^{-1} is already known to just over $3m$ places after the point where m is the number of digits in M, A and B. A and B and all intermediate results are represented in RSD form, but not M or $1/M$, which are in binary form. The multiplications are done by a Wallace tree [85] construction. This tree has a maximum depth of about $\log_{3/2} n$ 3-to-2 bit adders. However, this logarithmic time requires $O(n^2)$ area. He used the iterative application of a decomposition $e = me' + r$, where r is usually the least non-negative residue of $e \bmod m$, to reduce the number of

multiplications used in exponentiation [86]. At each repetition the divisor m is selected by reference to a pre-determined set of inexpensive pairs (m, r) and the powers A^m and A^r are computed. A^e satisfies the relationship

$$A^e = (A^m)^{e'} A^r. \tag{9}$$

When $e' = 0$ has been processed, the partial product register contains the required output. So, if $m_0, m_1, \ldots, m_n$ is the list of divisors which this generates, and $r_0, r_1, \ldots, r_n$ are the associated remainders then,

$$A^e = A^{r_0 + m_0(r_1 + m_1(\cdots + m_{n-1}(r_{n-1} + m_{n-1} r_n)\cdots))} \tag{10}$$

and evaluation is performed by processing the exponent expression from left to right. It is assumed that it is known how to calculate A^m and A^r in the most convenient way . So divisor/residue pairs (m, r) must be selected from a set for which this information is known. Then at each step, the cheapest such decomposition can be chosen from this fixed set of pairs. A sequence of pairs (m, r) used to direct an exponentiation in this way is called a *division chain*. In [86] Walter gave the algorithm to find the most efficient division chain, the examples and the test results.

In [87] Chiang and Chen proposed a new modular exponentiation and multiplication algorithms to reduce the number of iterations for both of those. They also proposed a hardware structure for their algorithms and reported the simulation results. Cho et al. proposed a radix-4 modular multiplication algorithm based on sign estimation technique (see [88,89]). They reduced the number of partial products by using the radix-4 Booth's algorithm [90]. In a carry save adder, a partial sum, S, and carry, C, sequence are generated in the intermediate stages and the carry propagation occurs only at the last stage. The sign of a number in one's or two's complement representations is indicated by the most significant bit. However, in the carry save representation, the most significant bit (the sign bit) is not readily available. In order to compute the exact sign of the partial sum $C + S$, C and S have to be summed in full precision. To solve this problem, the sign estimation algorithm estimates the sign of a number represented by a carry-sum pair produced by a carry save adder.

The algorithm proposed requires $n/2 + 3$ iterations for one modular multiplication. Two Radix-4 modular multiplications are executed simultaneously. The number of clock cycles required to complete the modular exponentiation is $n(n/2 + 3)$ clock cycles. The total time for one modular exponentiation is 13 ms at 40 MHz and the number of gates is 230,000.

3.4.3. *Residue number system*

Nowadays even faster arithmetic is demanded due to the constant improvements in factoring and the resulting requirements for even longer key sizes. In order to achieve that, the residue number system (RNS) is an alternative to the radix representation. RNS arithmetic is a very old idea which relies on the Chinese Remainder Theorem (CRT) and it provides a good means for very long integer arithmetic.

Let $\langle x \rangle_a$ denote an RNS representation of x, then

$$\langle x \rangle_a = (x[a_1], x[a_2], \ldots, x[a_n]), \tag{11}$$

where, $x[a_i] = x \bmod a_i$. The set $a = \{a_1, a_2, \ldots, a_n\}$ is called a base (of size n). It is required that $gcd(a_i, a_j) = 1$ for $i \neq j$. CRT implies that the integer x which satisfies $0 \leqslant x < \prod_{i=1}^{n} a_i$ is uniquely represented by $\langle x \rangle_a$.

A well known advantage of RNS is that to add, subtract and multiply such numbers we only need to compute the addition, subtraction and multiplication of their components, of size very much smaller than the original modulus. Also carry-free arithmetic makes parallelization possible which is a very desirable property in hardware. The final result is obtained by the CRT. The disadvantages of an RNS representation are that it is difficult to compare the size of elements and to perform division. To overcome this disadvantage, a combination with Montgomery multiplication was proposed [91].

According to majority of the previous work on this topic, most of the processing time for RNS Montgomery multiplication is devoted to base extension. A VLSI implementation for modular multiplication using RNS was proposed by Alia and Martinelli in [92]. Two different implementations were proposed in [93] for an RNS Montgomery multiplication. In the work of Posch and Posch [91] it was proposed that RNS should be used as the input and output representation of the algorithm, but they did not provide any algorithm for Radix-to-RNS (or vice versa) transformation. It is provided in the work of Kawamura et al. [94] and Nozaki et al. [95]. The main contribution of the work of Kawamura et al. is that they provided a new base extension algorithm. They proposed a Cox–Rower architecture for the RNS Montgomery multiplication. In this architecture, a base extension algorithm is executed in parallel by n sets of Rower units controlled by a Cox unit. Each Rower unit has a multiplier and accumulator with modular reduction by a_i or b_i where $\langle a \rangle$ and $\langle b \rangle$ are two RNS bases. The performance was only roughly estimated as the VLSI design work was still ongoing. Detailed performance including CRT figures is presented in [95]. An LSI prototype adopting the proposed Cox–Rower architecture achieves 1024-bit transaction in 4.2 and 2.4 ms without and with CRT, respectively, which was comparable with the best performances of commercial chips. This design can deal with key lengths up to 4 096 bits in CRT mode. Relevant proposals for RNS hardware implementation include the work of Bajard et al. [93]. The authors have proposed two different implementations using MMM.

3.4.4. CRT based implementations

Here, we explain about the CRT implementation of RSA which is an efficient way to reduce the work factor of the decryption process. Use of CRT for RSA was proposed in 1982 by Quisquater and Couvreur [96]. By means of the chinese remainder theorem (CRT), the speed for the RSA decryption scheme can be increased up to 4 times (see the book by Koblitz [23]). This possibility is very attractive for practical applications. However, it includes some pitfalls on security, so it has to be carefully implemented.

Let us consider again the RSA protocol for privacy. If user A, say Alice, wants to send a message to user B (Bob), she represents her message in any standardized way by a number M, $0 < M < N$. Number N is the modulus, i.e., $N = p \cdot q$. Next, Alice looks up the public exponent e_B of Bob. She will send the cipher text C computed from

$$C = M^{e_B} \bmod N. \tag{12}$$

Bob can recover M from C by raising it to the power d_B (private exponent of Bob) which he only knows (here, $e_B d_B = 1 (\mathrm{mod}\ \phi(N))$):

$$C^d \equiv M^{ed} \equiv M^{1+l\cdot\phi(N)} \equiv M * M^{\phi(N)l} \equiv M \bmod N. \tag{13}$$

where $\phi(N)$ is Euler's Totient Function for which $\phi(N) = (p-1)(q-1)$.

Then if Bob knows the factorization of N into p and q, he can do calculations of $\bmod p$ and $\bmod q$ instead of $\bmod N$. He computes $M_p \equiv C_1^d (\mathrm{mod}\, p)$ and $M_q \equiv C_2^d (\mathrm{mod}\, q)$, (where $C_1 \equiv C(\mathrm{mod}\, p)$ and $C_2 \equiv C(\mathrm{mod}\, q)$). All these calculations are done modulo integers p and q that are typically half of the length of N. The linear combination of M_p and M_q is the original message M. More precisely, Bob first precomputes integers a and b satisfying

$$\begin{aligned} a &\equiv 1 (\mathrm{mod}\, p), \\ a &\equiv 0 (\mathrm{mod}\, q) \end{aligned} \tag{14}$$

and

$$\begin{aligned} b &\equiv 0 (\mathrm{mod}\, p), \\ b &\equiv 1 (\mathrm{mod}\, q). \end{aligned} \tag{15}$$

By the Chinese Remainder Theorem, M is now given by: $M = aM_p + bM_q (\mathrm{mod}\, N)$. This method is known in the literature as Gauss algorithm [26]. Another efficient algorithm for reconstructing the message M is Garner's algorithm [26]. It reconstructs the M as follows.

$$M \equiv M_q + q \cdot (s \cdot (q^{-1} \bmod p) \bmod p), \tag{16}$$

where $s \equiv (M_p - M_q) \bmod p)$.

These computations can be performed in $O((\lg n)^2)$ bit operations. Altogether, this way of decryption can reduce the workload by a factor of 4. (Here, we assume cubic complexity of exponentiation.)

In April 2000 Compaq and RSA security Inc. announced a new patented technology *MultiPrime*™ as a generalization of standard RSA scheme. Instead of a modulus $N = pq$, as in traditional RSA system, N is a product of three or more (distinct) prime numbers. The idea was that increasing the number of factors and using CRT with parallel exponentiators increases performance. In general, the dependence of the performance of modular exponentiation and the length of modulus is not linear. It is approximately cubic or quadratic depending on the implementation. However, high level of security has to be preserved. The length of N is not the only relevant factor that provides it. Smaller factors make some methods of factoring more efficient, for example the Elliptic Curve factoring Method (ECM). Current record is 53 digits for smallest factor [97], which results in the use of not more than 4 prime factors of N for the modulus lengths of 2 048 bits and up to three factors for 1024 bit modulus. Fig. 2 presents improvements of performances for CRT and Multiprime compared with the common RSA scheme.

Another improvement in efficiency is expected for the situation where not all factors differ. Instead of $N = pqr$, we consider for example $N = p^2q$. This is a special case of the RSA scheme where $N = p^n q$, which was introduced in 1998 by Takagi [98]. This method appears to be more efficient than MultiPrime with 3 different primes only when the public exponent e is relatively small compared to the modulus length. Fig. 5 represents the performances for different e for architecture in [13] for all 3 CRT schemes ($N = pq$, pqr and p^2q).

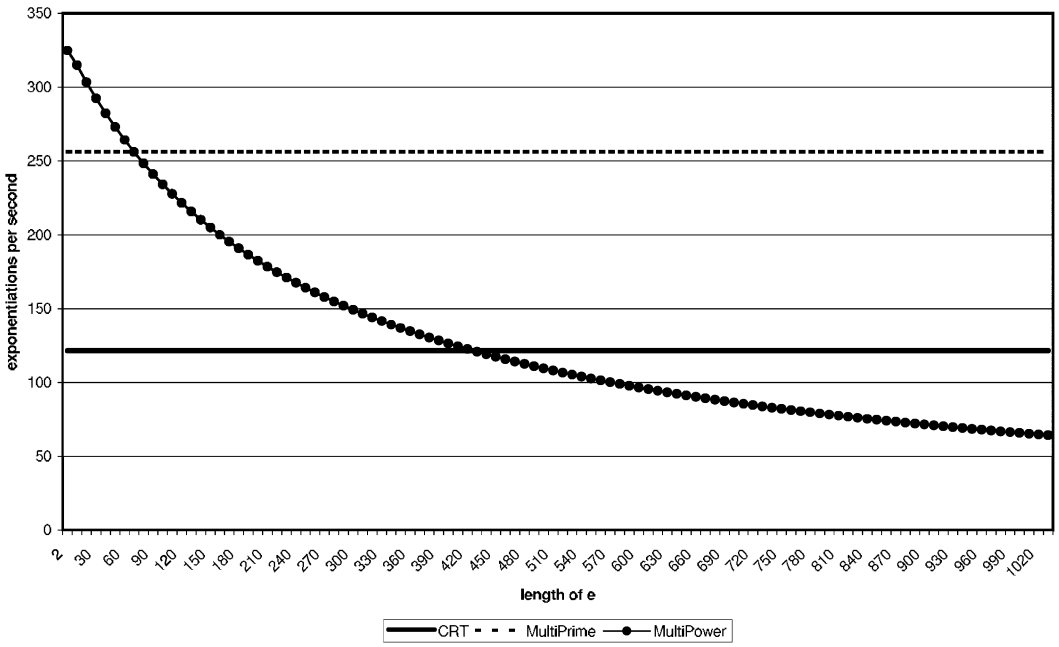

Fig. 5. Performances of modular exponentiation in relation with the size of the public exponent e for three different types of CRT technology [13].

One example of utilization of CRT is presented in Grosschädl [99]. In this paper the multiplier architecture of the RSA crypto chip is presented. The multiplier datapath is reconfigurable to execute either one 1024 or two 512 bit modular exponentiation in parallel. In CRT-mode, the decryption rate is increased by a factor of 3.5. The utilized multiplication algorithm is based on the special type of modular reduction, so-called Barret's modular reduction [26]. The binary exponentiation method (square-and-multiply) is used for modular exponentiation. The author presents this architecture as a highly scalable platform. Nevertheless, this prototype was optimized for 1024 bits of modulus length and a multiplier of word-size 16, and other values of parameters were not tested.

Boneh et al. showed in [22] that for an implementation of RSA based on the CRT, given one faulty version of the RSA signature, the adversary is able to factor the RSA modulus. The attack can be applied on the general RSA signature scheme (without CRT) as well, as introduced by Joye et al. in [100]. Some countermeasures, that would check the computation before releasing the signature, were proposed. One of the most recent is suggested by Aümuller et al. in [101]. They were also first to prove that this type of attack is also practical.

3.5. State of art for RSA hardware implementation

Soon after the discovery of the RSA encryption algorithm, there were chips available for performing it. The "first" RSA chip was designed by Rivest et al. in 1980 [102]. It was a single-chip nMOS design which occupied 42 mm^2. It contained a 512-bit register for storage of

intermediate results, carry-save adder logic, and up-down shifter logic. It contained approximately 40,000 transistors but failed to work reliably. One of the other early proposals dates back from 1981 by Lau and McPherson [103]. System hardware consisted of a DES processor, some RAM and a 16 hardware multiplier. Encrypting a message took 4.6 s and decrypting was up to 21 s (240 bits). However, a fast development of VLSI RSA circuits was forecasted. R. Rivest reviewed the issues involved in building a special-purpose chip for performing RSA encryption/decryption [102].

In the 1980s already several methods for implementing modular reduction were known. In Brickell [58] the quotient digits are approximated using only the high order bits of the divisor and the current remainder.

In 1985 Kochanski [104] proposed a RSA chip in which the modular exponentiation operation was divided between a CMOS array and a microprocessor. A full length modular multiplication took place in CMOS array by the commands from microcontroller. The complete modular exponentiation was performed by microcontroller. The worst-case time for a 511-bit modular exponentiation was 133 ms and the average was 100 ms.

In 1986, Rankine [105] introduced an implementation of a 512-bit modulus exponentiator for RSA applications. A wide range of applications including smartcards was claimed. It was an ASIC employing 4 kB of RAM and ALU functions were based on 64-bit operations. This device, called THOMAS, was able to execute a 512-bit exponentiation in less than a second. More precisely, a full 512-bit decryption was performed in 750 ms.

Various speed-up techniques in hardware and software have been analyzed and compared in Shand and Vuillemin [106]. These include: chinese remainders, MMM, addition chains, quotient pipelining etc. The hardware used was a PAM (programmable active memory) implementation of RSA which is based on PGA technology. The conclusions were that some small speed-ups, i.e., 1.25 and 1.5, are possible with precomputation of small powers for addition chains (in hardware) and the use of MMM, respectively. However, chinese remainders and quotient pipelining (facilitated with MMM) offer eventual speed-up of 4.

Various solutions for systolic arrays were proposed in the past ten years, for example [40,47–50], but no implementations have been reported to our knowledge (see also the work by Blum and Paar [64]).

The work of Tenca and Koç introduces the notion of scalable hardware. The authors have described a pipelined Montgomery multiplier, which has the ability to work on any given operand precision and is adjustable to the available area and/or performance. The architecture described in that work is semi-systolic and has a flavor of serial–parallel implementation. The authors propose an algorithm in which the operand Y (multiplicand) is scanned word-by-word and the operand X (multiplier) is scanned bit-by-bit. This organization allowed some parallelism which has maximal degree of $p_{\max} = \lceil (e+1)/2 \rceil$, where for operands with m bits of precision, $e = \lceil (m+1)/w \rceil$ words are required. If less than $p_{\max}$ processing units (PUs) are available, it causes the pipeline to stall. In that case some buffering is necessary and the total execution time increases. The certain number of pipeline stages is also a parameter considered in the design. The propagation delay of the PU influences the clock cycle, although it can be treated as independent of the wordsize w when w is relatively small. Yet, having only few stages yields very poor performance for the high precision range (512–1024 bits) due to pipeline stalls. In this case the fixed area becomes insufficient for large wordsizes. However, the flexibility of this design provides various

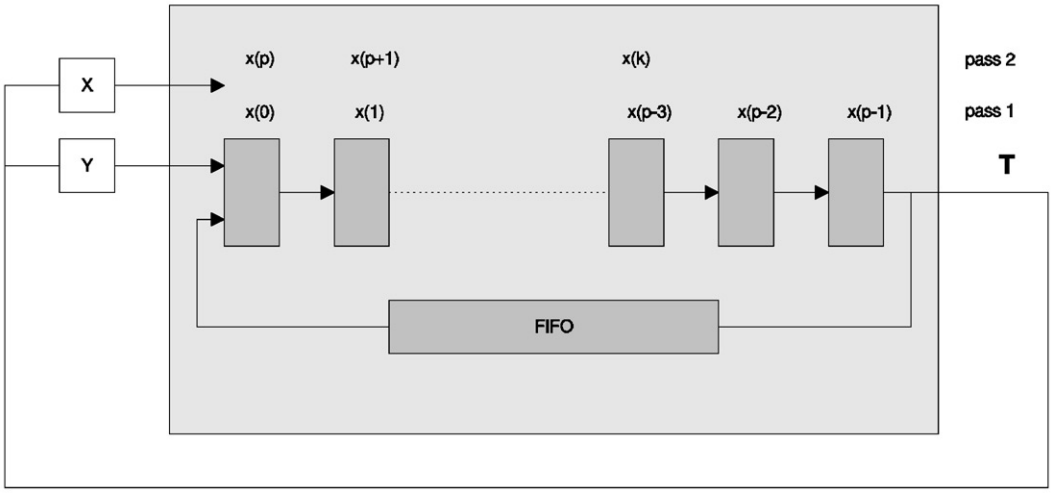

Fig. 6. The systolic array presented in the architecture [13].

design trade-offs. In this work the radix-2 algorithm is addressed and higher radix was dealt with in [107].

Savaş et al. [108] used the same design methodology to obtain a dual-field multiplier for both fields of use in cryptography i.e., GF(p) and GF(2^n), without compromising scalability. The basic observation to provide the unified Montgomery multiplier is that an adder module, equipped with the property of performing addition with or without carry, is available. This dual-field adder is basically a full adder which is capable of performing both types of addition. This so-called dual multiplier would have obvious benefits for many applications of public key cryptography.

The contribution of Batina and Muurling [13] is in combining a systolic array architecture (see Fig. 6), with a Montgomery based RSA implementation, achieving the same notion of scalability as introduced in [10]. Nevertheless, comparison of performances goes in favor of [13]. This is purely systolic array based architecture with high levels of flexibility and scalability which were previously treated as exclusive property of FPGAs based platforms or other non-systolic ASICs such as one in Tenca and Koç. The optimal bound for Montgomery's parameter R is achieved which, with some savings in hardware, omits completely all reduction steps that are presumed to be vulnerable to side-channel attacks. The ASIC product which is featuring this architecture is presented in Fig. 7.

The most prominent chip manufacturers that have PKC implementations are Atmel, Spyrus, Infineon, Siemens, Motorola, Philips and Certicom. The security chip from Atmel is called NIMBUS [109]. It has the capability of generating digital signatures with RSA for 1024-bits. If the RSA algorithm with CRT is used the time needed for one signature generation is 56 ms. This time is 225 ms if CRT is not used. Spyrus Inc. has a Universal Serial Bus (USB) crypto token which is called Rosetta USB [110]. It is FIPS 140-1 Certified and supports standards. Infineon's SLE 66CUX640P is a security controller to use in USB applications [111]. SLE 66CUX640P includes a public key coprocessor for modular arithmetic. It supports RSA (up to 1024 Bit) and ECC (up to 256 Bit) over $GF(p)$. The architecture of it is optimized for minimum power consumption.

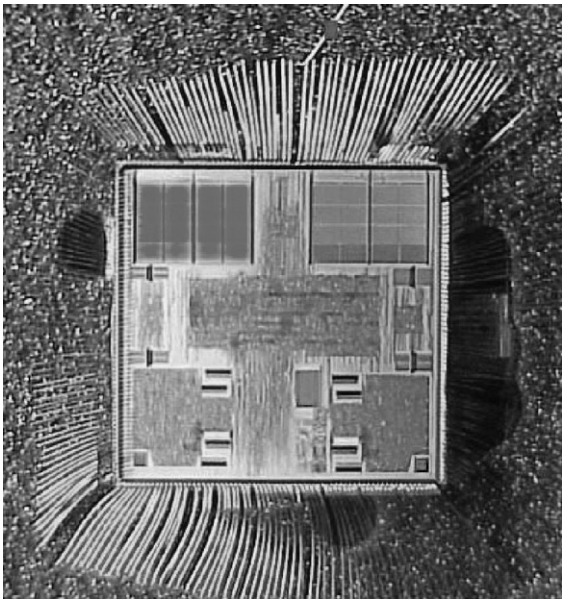

Fig. 7. Photo of the chip [13].

Table 2
The performance data of SLE 66CUX640P

Operation	Length of modulus (bit)	Execution time (@15 MHz) (ms)
[k]P on EC over $GF(p)$	160	83
[k]P on EC over $GF(p)$	256	234
$a^b \bmod N$	1024	220

Maximum clock frequency is 15 MHz. The total area of public key coprocessor is 1 mm^2 with 0.25 μm technology. The performance data of the coprocessor is given in Table 2.

Siemens has an encryption integrated circuit (IC) called PLUTO-IC [112]. The encryption rate of PLUTO-IC is reported as 2 Gbit/s. It is produced for ECCs based on curves over $GF(p)$, p of length 320 bit. ELCRODAT-6-2 is another product from Siemens which is an encryption device for the ISDN-telecommunication network. It is also designed for ECC based on curves over $GF(p)$, the bit-length of p is 256. Motorola has four different security processors called MPC180, MPC184, MPC185 and MPC190 [113]. They are designed to enhance system performance by executing computationally intense operations associated with the processing of IP Security Protocol (IPSEC), Internet Key Exchange (IKE), Secure Sockets Layer (SSL) and Wireless Transport Layer Security (WTLS) protocols used in Broadband Access, Customer Premise Equipment (CPE), Routers, WAP Gateways and other Access applications. They support ECC for WAP/WTLS support in wireless applications. They include Public Key Execution Units (PKEUs) which can RSA, ECC, Diffie–Hellman with programmable bit-lengths. Philips has SmartXA as a 16-bit smart card IC [114]. It includes cryptographic co-processors for RSA. A

1024 bit exponentiation takes 400 ms. Certicom provides the Luna CA3-ECC and Luna 2-ECC hardware security tokens [115]. These PCMCIA type tokens support ECDSA (163-256 bits), RSA (512-4096 bits), and Digital Signature Algorithm (DSA) (512-1024 bits).

There are two surveys published about previous RSA implementations one is by Brickell in 1989 [116] and the other is by Koç in 1995 [117].

4. Discrete logarithm based cryptosystems

This section presents the public key cryptosystems which are based on discrete logarithm problem (DLP). These cryptosystems include Diffie–Hellman key agreement [1] and its derivatives, ElGamal encryption and ElGamal signature scheme [118,119] and its variants.

Diffie–Hellman key agreement provided the first practical solution to the key distribution problem, allowing two parties, never having met in advance or shared keying material, to establish a shared secret by exchanging messages over an open channel [26]. The basic version of this protocol is as follows:

(1) *One time setup*: An appropriate prime p and a generator α of $\mathbb{Z}_p^*$ where $2 \leqslant \alpha \leqslant p-2$ are selected and published.

$$A \to B : \alpha^x \bmod p \quad (1).$$

(2) *Protocol messages*:

$$B \to A : \alpha^y \bmod p \quad (2).$$

(3) *Protocol actions*: Perform the following steps each time a shared key is required.
   (a) A chooses a random secret x, $1 \leqslant x \leqslant p-2$ and sends B message (1).
   (b) B chooses a random secret y, $1 \leqslant y \leqslant p-2$ and sends A message (2).
   (c) B receives α^x and computes the shared key as $K = (\alpha^x)^y \bmod p$.
   (d) A receives α^y and computes the shared key as $K = (\alpha^y)^x \bmod p$.

The basic ElGamal encryption scheme is described as follows [26]:

(1) *Encryption*: B should do the following:
   (a) Obtain A's authentic public key (p, α, α^a).
   (b) Represent the message as an integer m in the range $\{0, 1, \ldots, p-1\}$.
   (c) Select a random integer k, $1 \leqslant k \leqslant p-2$.
   (d) Compute $\gamma = \alpha^k \bmod p$ and $\delta = m \cdot (\alpha^a)^k \bmod p$.
   (e) Send the ciphertext $c = (\gamma, \delta)$ to A.
(2) *Decryption*: To recover plaintext m from c, A should do the following:
   (a) Use the private key a to compute $\gamma^{p-1-a} \bmod p$.
   (b) Recover m by computing $(\gamma^{-a} \cdot \delta \bmod p)$.

Let G be a finite cyclic group of order n. Let α be a generator of G and let $\beta \in G$. The *discrete logarithm of β to the base α*, is the unique integer x, $0 \leqslant x \leqslant n-1$, such that $\beta = \alpha^x$. By knowing β

to find x is called *discrete logarithm problem*. Two examples for G are finite fields F_p, p is prime, and F_{2^m}. Detailed information about finite fields can be found in [26,120].

As it is shown in the algorithms above, the main operations are modular multiplication and modular exponentiation. The architectures that are given in Sections 3.3 and 3.2 can be used for the finite field F_p. In the following sections we will present the different modular multiplication structures for the finite field F_{2^m}.

4.1. Introduction and mathematical background

There are three main different ways to represent the elements in the finite field F_{2^m}, polynomial, normal and dual bases. In the following we give brief descriptions of these representations.

4.1.1. Polynomial (or standard or canonical) basis

The polynomial basis (PB) is given by the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ where α is a root of the prime polynomial $P(x)$ of degree m used to construct $GF(q^m)$ from $GF(q)$.

4.1.2. Dual basis

Trace. $F = GF(q)$, $K = GF(q^n)$ and $\alpha \in K$. The trace of α relative to subfield F is

$$Tr_F^K(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{n-1}} \tag{17}$$

$\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ is a basis for $GF(2^m)$ over $GF(2)$, so the elements of the set are linearly independent over $GF(2)$. The corresponding dual basis (DB) is

$\{\beta_0, \beta_1, \ldots, \beta_{m-1}\} \subseteq GF(2^m)$ such that

$$Tr(\alpha_i \beta_j) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{18}$$

4.1.3. Normal basis

α is a root of the prime polynomial $P(x)$ of degree m used to construct $GF(q^m)$ from $GF(q)$. Then the set $\{\alpha, \alpha^2, \ldots, \alpha^{2^{m-1}}\}$ forms a normal basis (NB).

4.2. Architectures for modular multiplication in $GF(2^m)$

Mastrovito's thesis [121] serves as an extensive reference of hardware architectures for performing $GF(2^m)$ multiplication. In this section we will give an overview of bit-serial multiplication architectures to serve as a starting point for further sections.

4.2.1. Polynomial-basis multipliers

Serial. In this section an architecture for bit-serial computation of products of two elements of $GF(2^m)$ that are represented by PB will be described. This architecture is called as serial shift register (SSR) multiplier.

Let $A(x) = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$ and $B(x) = b_0 + b_1 x + \cdots + b_{m-1} x^{m-1}$ are two field elements that will be multiplied. $C(x) = c_0 + c_1 x + \cdots + c_{m-1} x^{m-1}$ is the result of the

multiplication. Then

$$C(x) = A(x)B(x) \bmod P(x)$$
$$= [b_0 A(x) + b_1 x A(x) + \cdots + b_{m-1} x^{m-1} A(x)] \bmod P(x). \quad (19)$$

Each term $b_i x^i A(x)$ in Eq. (19) can be computed recursively in the following way:

$$b_i x^i A(x) \bmod P(x)$$
$$= (\cdots((b_i x A(x) \bmod P(x))x \bmod P(x))\cdots)x \bmod P(x). \quad (20)$$

So $C(x)$ can be written as follows:

$$C(x) = (((b_{m-1} x A(x) + b_{m-2} A(x))x + b_{m-2} A(x))x + \cdots)x$$
$$+ b_0 A(x) \bmod P(x). \quad (21)$$

The block diagram of SSR multiplier is shown in Fig. 8. The product $C(x)$ is found in $C = [c_{m-1}, c_{m-2}, \ldots, c_0]$ register after m clock cycles. A schematic view of SSR multiplier cell is shown in Fig. 9

Properties of SSR Multiplier. The length of the critical path is 4 (one register, one AND-gate and two XOR-gates), independently of the choice of $P(x)$. This implies that the SSR allows the same clock frequency for any m. The complexity is linear in m, exactly $6m$ ($2m$ registers and $4m$ gates).

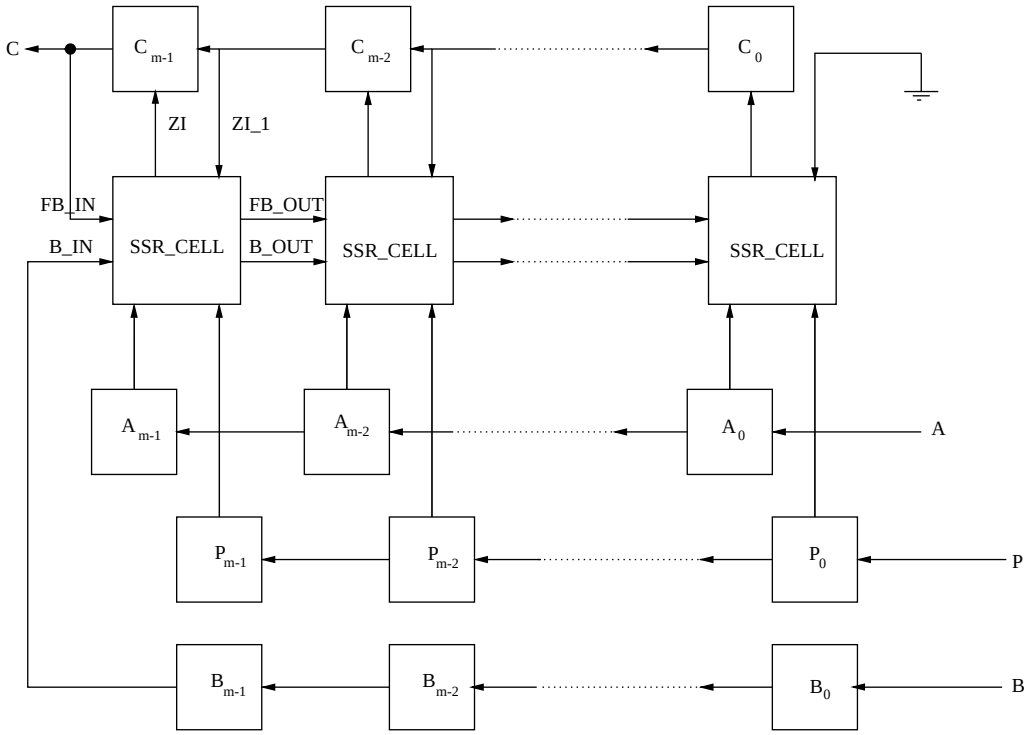

Fig. 8. SSR multiplier block diagram.

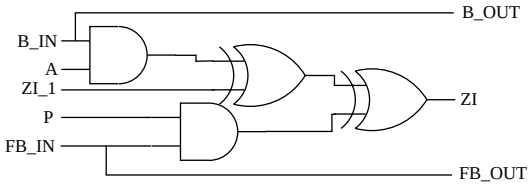

Fig. 9. Schematic view of SSR multiplier cell.

Parallel. Mastrovito proposed in his thesis [121] in 1991 a parallel multiplier which is named after him. The architecture can be explained as follows.

The product $C(x) = A(x)B(x) \bmod P(x)$ can be written in matrix form as $C = ZB$, where Z is a binary m by m matrix. The entry $z_{i,j}$ of Z is denoted by $f_{i,j}(A)$ or simply $f_{i,j}$. The elements of $C(x)$ can be written as following:

$$c_i = b_0 f_{i,0}(A) + b_1 f_{i,1}(A) + \cdots + b_{m-1} f_{i,m-1}(A). \tag{22}$$

In matrix notation the product can be written as follows:

$$C = \begin{pmatrix} f_{0,0} & f_{0,1} & \cdots & f_{0,m-1} \\ f_{1,0} & f_{1,1} & \cdots & f_{1,m-1} \\ \vdots & \vdots & & \vdots \\ f_{m-1,0} & f_{m-1,1} & \cdots & f_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix} = ZB. \tag{23}$$

It is desirable to have explicit formulas for functions $f_{i,j}$. The following symbol is introduced for simplicity

$$s_k = \sum_{u+v=k} b_u a_v \;\; u, v \in \{0, 1, \ldots, m-1\}. \tag{24}$$

Further, an $m-1$ by m binary matrix Q which is called *reduction matrix* is defined as

$$\begin{pmatrix} x^m \\ x^{m+1} \\ \vdots \\ x^{2m-1} \end{pmatrix} = Q \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{pmatrix} = \begin{pmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,m-1} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,m-1} \\ \vdots & \vdots & & \vdots \\ q_{m-1,0} & q_{1,m-1} & \cdots & q_{m-1,m-1} \end{pmatrix} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{pmatrix}. \tag{25}$$

The reduction matrix is obtained from $P(x)$ by a simple shift & add-on-overflow procedure. Because,

$$\begin{aligned} x^m \bmod P(x) &= p_{m-1} x^{m-1} + \cdots + p_1 x + p_0 \\ &= q_{0,0} + q_{0,1} x + \cdots + q_{0,m-1} x^{m-1}, \end{aligned} \tag{26}$$

$q_{0,i} = p_i$ for $i = 0, 1, \ldots, m-1$.

$$\begin{aligned} & x^{m+1} \bmod P(x) \\ & \quad = q_{0,0}x + q_{0,1}x^2 + \cdots + q_{0,m-1}x^m \\ & \quad = q_{0,0}x + q_{0,1}x^2 + \cdots + q_{0,m-2}x^{m-1} \\ & \qquad + q_{0,m-1}(q_{0,0} + q_{0,1}x + \cdots + q_{0,m-1}x^{m-1}) \\ & \quad = q_{0,m-1}q_{0,0} + (q_{0,m-1}q_{0,1} + q_{0,0})x \\ & \qquad + (q_{0,m-1}q_{0,2} + q_{0,1})q_{0,1}x^2 + \cdots + (q_{0,m-1}q_{0,m-1} + q_{0,m-2})x^{m-1}, \end{aligned} \tag{27}$$

$$\begin{aligned} & q_{i,0} = q_{i-1,m-1}q_{i-1,0}, \\ & q_{i,1} = q_{i-1,m-1}q_{i-1,1} + q_{i-1,0} \\ & \vdots \\ & q_{i,m-1} = q_{i-1,m-1}q_{i-1,m-1} + q_{i-1,m-2}, \end{aligned} \tag{28}$$

$$\begin{aligned} C(x) &= \left(\sum_{i=0}^{m-1} a_i x^i\right)\left(\sum_{j=0}^{m-1} b_j x^j\right) \bmod P(x) \\ &= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j x^{i+j} \bmod P(x) \\ &= \sum_{k=0}^{2m-2} s_k x^k \bmod P(x) \\ &= \sum_{k=0}^{m-1} s_k x^k + \sum_{k=m}^{2m-2} s_k x^k \bmod P(x) \\ &= \sum_{k=0}^{m-1} s_k x^k + \sum_{k=m}^{2m-2} s_k \sum_{i=0}^{m-1} q_{k-m,i} x^i \\ &= \sum_{k=0}^{m-1} s_k x^k + \sum_{k=0}^{m-2} s_{k+m} \sum_{i=0}^{m-1} q_{k,i} x^i \\ &= \sum_{k=0}^{m-1} s_k x^k + \sum_{k=0}^{m-2} s_{k+m} q_{k,0} + \sum_{k=0}^{m-2} s_{k+m} q_{k,1} x \\ &\quad + \sum_{k=0}^{m-2} s_{k+m} q_{k,2} x^2 + \cdots + \sum_{k=0}^{m-2} s_{k+m} q_{k,m-1} x^{m-1}, \end{aligned} \tag{29}$$

$$c_i = f_{i,0}b_0 + f_{i,1}b_1 + \cdots + f_{i,m-1}b_{m-1}, \tag{30}$$

$$
\begin{aligned}
f_{i,0} &= a_i,\\
f_{i,1} &= a_{i-1} + q_{0,i}a_{m-1},\\
f_{i,2} &= a_{i-2} + q_{0,i}a_{m-2} + q_{1,i}a_{m-1},\\
f_{i,3} &= a_{i-3} + q_{0,i}a_{m-3} + q_{1,i}a_{m-2} + q_{2,i}a_{m-1},
\end{aligned} \tag{31}
$$

$$
f_{i,j} = \sigma(i-j)a_{i-j} + \sum_{t=0}^{j-1} q_{j-1-t,i}a_{m-1-t}, \tag{32}
$$

where $\sigma(k)$ is a step function defined by

$$
\sigma(k) = \begin{cases} 1 & k \geqslant 0,\\ 0 & k < 0. \end{cases} \tag{33}
$$

The multiplier is divided into two subsystems. The first subsystem computes the functions $f_{i,j}$ and is called f-network. The second subsystem is called the IP network and consists of m identical cells.

The Properties of Parallel PB Multiplier. The total complexity (in gates) is denoted by C and the length of the critical path through the whole multiplier is denoted by L.

$2 + \lceil \log_2 m \rceil \leqslant L \leqslant 1 + 2\lceil \log_2 m \rceil$.

$1 + m(2m-1) \leqslant C \leqslant (m-1)(\omega_P - 2) + m(2m-1) \leqslant 3m(m-1) + 1$, where ω_P is the Hamming weight of $P(x)$.

The C and L depend on the selection of field generator.

4.2.2. Serial dual-basis multiplier

Let $A, B, C \in GF(2^m)$ and $C = A * B$ be product. We assume that A is DB representation and B is in PB representation, i.e, $A = \sum a_i\beta_i$ and $B = \sum b_i\alpha_i$.

From duality relation and the properties of the trace function c_0 can be obtained as following (see Mastrovito [121, Section 3.2]):

$$
\begin{aligned}
c_0 &= Tr(AB) = Tr(b_0A) + Tr(b_1\alpha A) + \cdots + Tr(b_{m-1}\alpha^{m-1}A)\\
&= a_0b_0 + a_1b_1 + \cdots + a_{m-1}b_{m-1}\\
&= A \cdot B,
\end{aligned} \tag{34}
$$

where "$\cdot$" denotes the inner product.

To obtain the second coefficient c_1 A is replaced by αA in Eq. (34) and compute the inner product $(\alpha A) \cdot B$, the third coefficient c_2 is obtained by computing $(\alpha^2 A) \cdot B$ and so on for the remaining coefficients.

The block diagram of DB multiplier is shown in Fig. 10.

Properties of DB Multiplier. The complexity of DB multiplier is $6m - 2$ ($2m$ registers and $4m - 2$ gates). The critical path has length $2 + \lceil \log_2 m \rceil$ (one register+the inner-product logic).

4.2.3. Serial normal-basis (Massey–Omura) multiplier

NB multiplier was first proposed by Massey and Omura [122]. So it was named after them as Massey–Omura (MO) multiplier. The key property of the NB representation is that squaring is a

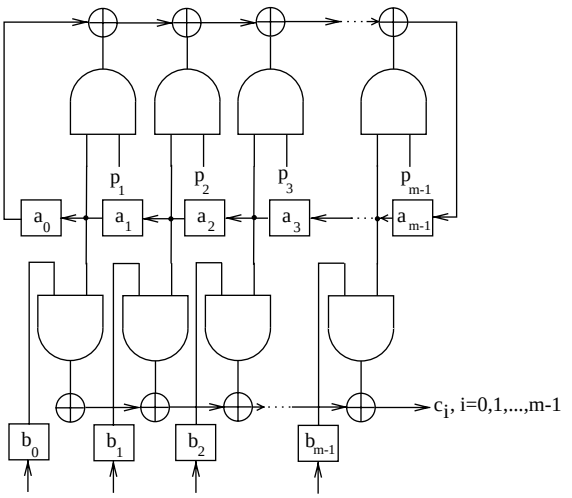

Fig. 10. DB multiplier block diagram.

very simple operation. Let $A = a_0\alpha + a_1\alpha^2 + \cdots + a_{m-1}\alpha^{2^{m-1}}$ be in $GF(2^m)$. Then

$$\begin{aligned} A^2 &= a_0\alpha^2 + a_1\alpha^4 + \cdots + a_{m-1}\alpha^{2^m} \\ &= a_{m-1}\alpha + a_0\alpha^2 + a_1\alpha^4 + \cdots + a_{m-2}\alpha^{2^{m-1}}. \end{aligned} \tag{35}$$

In Eq. (35) the linearity of the squaring operation and the fact that $\alpha^{2^m} = \alpha$ is used. Squaring in NB is a simple cyclic shift of the element's bits.

Let $C = AB = c_0\alpha + c_1\alpha^2 + \cdots + c_{m-1}\alpha^{2^{m-1}}$ be the product. Then the last coefficient c_{m-1} of C is a function of coefficients of A and B, i.e.,

$$c_{m-1} = f(a_0, a_1, \ldots, a_{m-1}; b_0, b_1, \ldots, b_{m-1}). \tag{36}$$

By Eq. (35) C^2 can be written as follows:

$$\begin{aligned} C^2 &= A^2B^2 \\ &= (a_{m-1}, a_0, \ldots, a_{m-2}) \cdot (b_{m-1}, b_0, \ldots, b_{m-2}) \\ &= (c_{m-1}, c_0, \ldots, c_{m-2}), \end{aligned} \tag{37}$$

which means that the last coefficient c_{m-2} of C^2 can be obtained by applying the same function f to the components of A^2 and B^2. By squaring C repeatedly, all the coefficients of C can be found as

$$\begin{aligned} c_{m-1} &= f(a_0, a_1, \ldots, a_{m-1}; b_0, b_1, \ldots, b_{m-1}), \\ c_{m-2} &= f(a_{m-1}, a_0, \ldots, a_{m-2}; b_{m-1}, b_0, \ldots, b_{m-2}) \\ &\vdots \\ c_0 &= f(a_1, a_2, \ldots, a_0; b_1, b_2, \ldots, b_0). \end{aligned} \tag{38}$$

The complexity of this multiplier depends on the function f which in turn, depends on the choice of $P(x)$. The schematic view of MO multiplier over $GF(2^4)$ with $P(x) = x^4 + x^3 + 1$ is shown in Fig. 11.

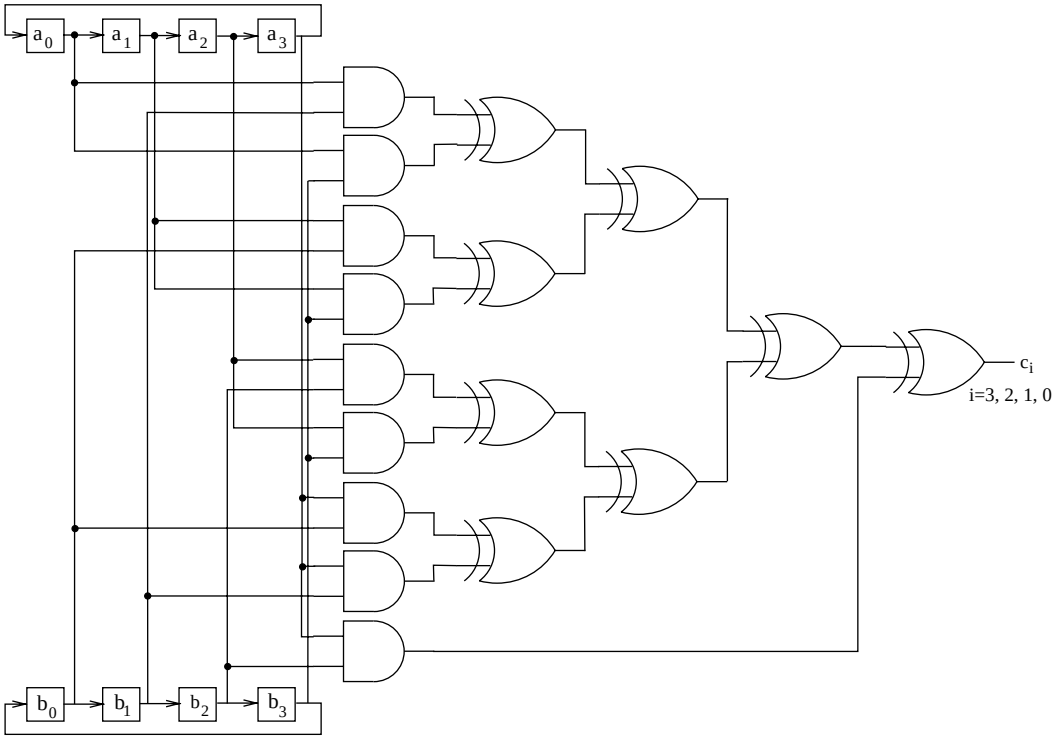

Fig. 11. The schematic view of MO multiplier over $GF(2^4)$ with $P(x) = x^4 + x^3 + 1$.

The Properties of Serial NB (Massey–Omura) Multiplier. As a measure of the complexity of the Massey–Omura multiplier, the number of terms $a_i b_j$ in f is used. This number is denoted by N_m. The complexity of MO multiplier is: $(N_m - m)$ XOR gates $+ (2m - 1)$ AND gates $+ 2m$ registers. The length of the critical path is $2 + \lceil \log_2 N_m \rceil$ (one register and the logic function f).

Mullin et al. showed that $N_m \geqslant 2m - 1$ for any NB in [123] and the concept of *optimal normal basis* (ONB) is introduced. An NB is said to be optimum if $N_m = 2m - 1$. There are two constructions given in [123] to obtain an ONB in $GF(2^m)$. These constructions are called types I and II.

4.2.4. Other bases representation

Parker and Benaissa presented multiplier architectures for *redundant bases representation* (RBR) in [124]. RBR was proposed by Parhami in [125]. Wu et al. presented multiplication, conversion and squaring architectures for finite field computation using RBR in [126]. Drolet proposed a new representation for finite field elements called *polynomial ring representation* (PRR) in [127]. He also gave parallel and serial multiplier, exponentiation, inversion and division architectures with PRR in [127]. Silverman proposed a new representation for finite field elements in 1999 [128]. He called this method *Ghost Bit Basis* (GBB). There is no multiplier hardware structure for this representation so far.

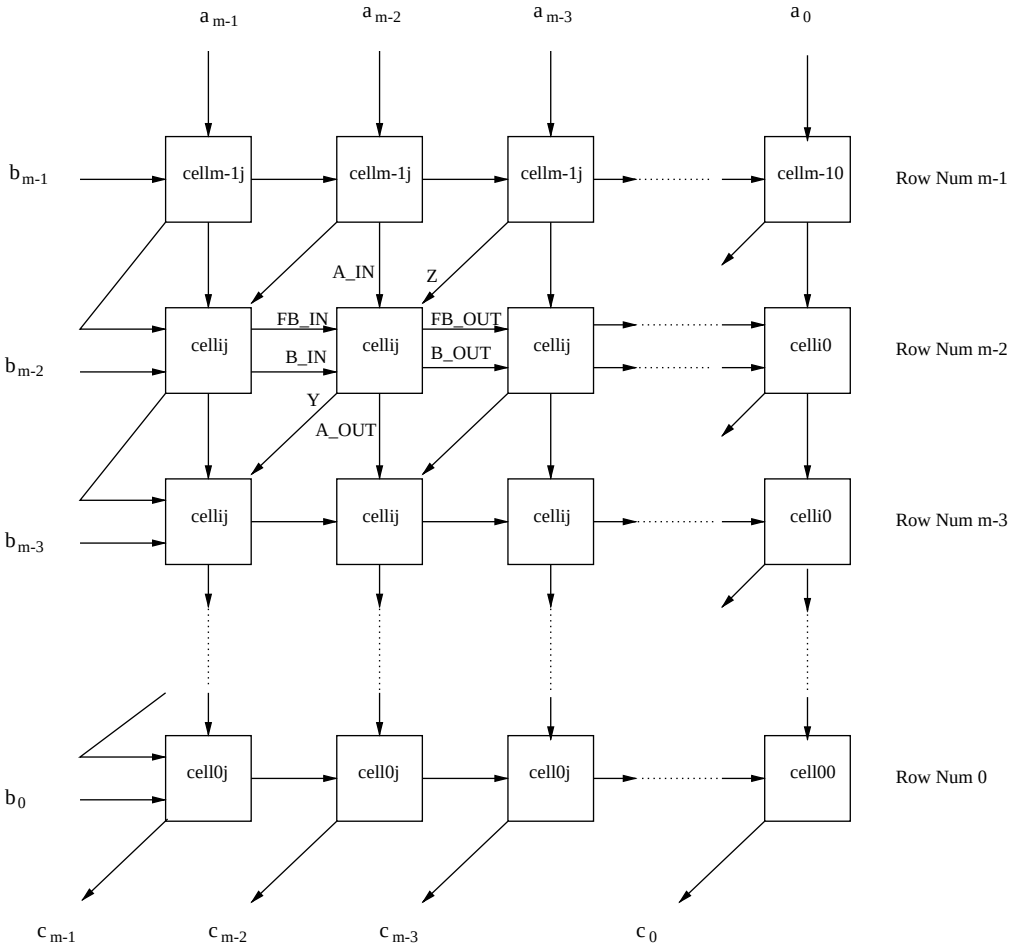

Fig. 12. Block diagram of cellular-array multiplier.

4.3. State of art for $GF(2^m)$ multiplier hardware implementation

4.3.1. Standard bases

In 1971 Laws and Rushforth proposed a cellular-array multiplier [129]. This array exhibits a high degree of regularity and can be viewed as a device for performing computations in space rather than in time. The block diagram of cellular-array multiplier is shown in Fig. 12. The schematic view of a general individual cell is shown in Fig. 13. The other work about this structure is from Jain et al. [130]. They also fabricated a multiplier over $GF(2^4)$ chip using CMOS 1.2 μm technology. The chip has an active area of 0.434 mm^2 and requires 1076 transistors and is programmable for different irreducible polynomials.

In 1984 Yeh et al. [131] proposed similar architecture as [129]. They wanted to implement not only $AB \bmod P$ but also $AB + C \bmod P$. This is straightforward, because addition is simply bitwise XOR of elements. So they add one more XOR gate to the cells of SSR and cellular-array multipliers to get their architecture.

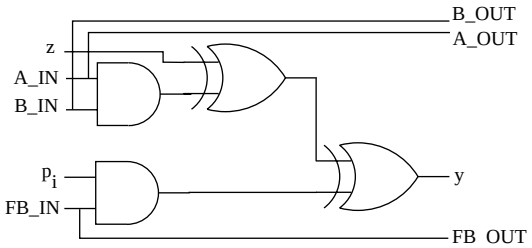

Fig. 13. The schematic view of cellular-array multiplier cell.

Hasan and Bhargava proposed a bit-serial systolic architecture for multiplication in [132]. Hasan also proposed to use look-up table approach in [133]. Hasan et al. proposed two structures of parallel multipliers based on an irreducible all one polynomial (AOP) of degree m and equally spaced polynomial (ESP) of degree $m(m+1)^i$ [134]. A polynomial $f(z)=\sum_{i=0}^{m} f_i z^i$ over $GF(2)$ is called AOP of degree m if $f_i=1$ for $i=0,1,\ldots,m$. A polynomial $g(z)=\sum_{i=0}^{sm} g_i z^i = z^{sm}+z^{s(m-1)+\cdots+z^s+1}=f(z^s)$ over $GF(2)$, where $f(z)$ is an AOP of degree m of degree m over $GF(2)$ is called an s-ESP of degree sm. The details of the proposed algorithms will not be given in this work. The total delay of parallel AOP multiplier is $D_A+(m+\lceil \log_2(m-1)\rceil)D_X$ where D_X and D_A denote the delay for an XOR gate and an AND gate, respectively. The total number of XOR and AND gates are m^2+m-2 and m^2. Another work about multipliers for fields defined by AOP and ESP was reported by Lee et al. in [135]. They used systolic architecture for bit-parallel multipliers.

The first work about composite field, $GF((2^n)^m)$ is from Paar [136,137]. Also Paar and Rosner gave a comparison of multiplication in composite fields, $GF((2^n)^m)$ and prime fields, $GF(2^m)$, m is prime in [138].

Wei et al. and Wang used systolic architectures for parallel multiplication in [139,140], respectively. Wu proposed explicit formulas to calculate the complexity of parallel multiplication in [141].

Orlando and Paar gave the results of their implementation of serial multiplier on several FPGAs in [142]. The prototype implementations were done using Xilinx XC4000X FPGAs of speed grade -09. The implementation results are summarized in Tables 2 and 3 in [142]. They also gave timing estimations of elliptic curve point multiplication using projective coordinates in their work.

Song and Parhi, Sunar and Koç, Halbutoğulları and Koç reported their results on the Mastrovito multiplier in [143], [144] and [145], respectively. A systematic design way for Mastrovito multiplier was proposed by Zhang and Parhi in [146].

The multiplier architecture from Hasan and Wassal [147] which was designed by using triangular and polynomial basis together was fabricated. The design was optimized in CMOS 0.5 μm technology for a 3.3 V supply voltage. The prototype can support operations over finite fields up to $GF(2^{64})$. The silicon area used was approximately 3445 mm × 3827 mm for the whole chip packaged in a 68 pin PGA package. The prototype chip was tested at a clock frequency of 50 MHz. This frequency limitation is reported as due to the packaging technology used. It is also reported in the paper that the implemented chip core could run at a frequency of more than

80 MHz, while an implementation with $m = 256$ was estimated to run at a frequency of more than 75 MHz.

Montgomery modular multiplication method (see Section 3.3.1) can be used for multiplication over $GF(2^m)$ also. The papers by Wu [148,149] are good references to learn about this technique.

Gao and Parhi implemented parallel multiplier by using two different approaches, irregular and regular semi-systolic [150]. They used 0.35 μm CMOS technology to implement both structures for $GF(2^8)$. They gave the area and latency data in Table 2 which can be used for comparison of them. Another parallel structure by Koç and Sunar can be found in [151].

Yu gives comparisons for power consumption of semi-systolic array, Mastrovito and composite field in [152].

4.3.2. Normal bases

Wang et al. proposed a parallel architecture for Massey–Omura multiplier in [153]. They fabricated a chip for $GF(2^4)$ by 4 μm NMOS technology. The chip has 8 pins and takes area of 1248 μm × 996 μm. Other parallel structures were proposed by Koç and Sunar, Reyhani-Masoleh and Hasan in [151] and [154], respectively.

Because multiplication function, f, given in Section 4.2.3 depends on the irreducible polynomial, every time the polynomial is changed the multiplier circuit has to be changed. An algorithmic way to find f depending on any irreducible polynomial was proposed by Wang in [155,156].

In 1988 Onyszchuk et al. proposed a different approach from Massey–Omura multiplier and this invention can be found in [157]. Agnew et al. improved the previous work and reported the results in [158]. In 1992 the same authors reported the VLSI implementation of normal bases multiplier in [159]. The chip was fabricated using 1.5 μm HCMOS gate array with a clock speed of 40 MHz and required less than 12,000 gates. Gao and Sobelman gave some VLSI design results in [162].

Lu gave maximum, minimum and average values of the measure of the complexity of the Massey–Omura multiplier, N_m, for different m in [160]. The definition of N_m is given in Section 4.2.3. He proposed a way to find the optimal multiplication function f.

An invention to realize composite field multiplication was made by Mullin and presented in [161]. Because he used composite fields the multiplication is done with more than one bit of the operands in one clock cycle.

In [163] it is shown that composite fields using modified optimal normal bases can be classified into three cases as follows:

- *Type I composite field*: A subfield $GF(2^n)$-Type II optimal normal basis and an extension field $GF(2^{nm})$-Type I optimal normal basis.
- *Type II composite field*: A subfield $GF(2^n)$-Type I optimal normal basis and an extension field $GF(2^{nm})$-Type II optimal normal basis.
- *Type III composite field*: A subfield $GF(2^n)$-Type II optimal normal basis and an extension field $GF(2^{nm})$-Type II optimal normal basis.

In [163] multiplier structures for first two are given.

Sunar and Koç gave algorithms for a type II ONM multiplier in [164]. Sutikno and Surya proposed architectures for ONB multiplier in extension and prime field [165].

Reyhani-Masoleh and Hasan proposed architectures for serial NB multiplier in [166]. The same authors gave comparisons of algorithms for NB, type I ONB and composite field multiplication in [167].

4.3.3. *Dual bases*

Bit-serial systolic multiplier structures were proposed by Diab and Poli, in [168], respectively. Fenn et al. proposed bit-serial and parallel multipliers and gave their delay and area comparisons in [169]. Wu et al. proposed parallel multipliers in [170,171]. Lee and Lim defined a new dual basis called *circular dual basis* (CDB) in [172]. They gave the algorithms for multiplication, squaring, conversion and inversion in [172]. Gollmann reported some results about ESP and dual basis in [173].

4.3.4. *Works for comparison of different bases*

Hsu et al. reported their VLSI implementation results of 8-bit finite field multipliers using dual, normal and standard bases in [174]. Their concluding remarks about them were following:

- The DB multiplier occupies the smallest amount of chip area.
- As the order of the field goes higher, the DB multiplier will increase its advantage over the others.
- The NB multiplier is very effective in performing operations such as inversion, squaring, exponentiation.
- The area of the NB multiplier grows dramatically as the order of the field goes up.
- The SB (PB) does not require basis conversion.
- Due to PB's regularity and simplicity, the design and expansion to higher order finite fields are easier.

Paar and Lange presented VLSI implementation results of finite field multipliers using dual, normal and standard bases for different orders in [175].

Ahlquist et al. presented FPGA implementation results of different finite field multipliers in [176]. The performance of each finite field multiplier were characterized on the Xilinx XC4062 FPGA. Table 2 in [176] shows resource usage and clock speed data for all designs. From the results they concluded that finite field multipliers optimized specifically for VLSI are not necessarily optimized for FPGAs. They claimed that the following three significant flaws are effective on the previous conclusion:

- Multi-clock cycle operation.
- Long unregistered datapaths.
- Underutilized logic elements.

They modified their designs to avoid these flaws [176].

5. Elliptic curve cryptosystems

In this section, we introduce some basic notation and necessary mathematical background about elliptic curves and elliptic curves based cryptosystem. For a detailed exposure we refer to the cited literature [4,3,177,178,27].

5.1. Introduction and mathematical background

In order to introduce a public key cryptosystem based on elliptic curves, we will first describe the elliptic curve group and define the Elliptic Curve Discrete Logarithm Problem i.e., ECDLP, which is similar to DLP in the case of arbitrary cyclic group.

Consider the set of all points on an elliptic curve E, together with one special point, the so-called point at infinity (usually denoted as $\mathcal{O}$). These points form an additive abelian group with point addition as a group operation. (For details on elliptic curve "group operation" see for example [177,179,27].)

Now we introduce the elliptic curve discrete logarithm problem (ECDLP) whose difficulty is guarding the ECC protocols. Let $E/\mathbb{F}_q$ be an elliptic curve and let $P \in E(\mathbb{F}_q)$ be a point of order m. (Point is of order m if $mP = \mathcal{O}$ and m is the smallest integer satisfying this equation.). Let $Q \in \langle P \rangle$, so that $Q = aP$ for some integer $a, 0 \leqslant a < m$. The *elliptic curve discrete logarithm problem* (*ECDLP*) is defined to be the problem of finding the number a for a given P and Q. This problem is believed to be hard i.e., it is still unknown if there exists an algorithm to solve it in less than fully exponential time.

The basic operation for ECC algorithms is point or scalar multiplication which efficiency of which is mostly determined by the implementation of finite field arithmetic. Both cases for finite fields of interest are considered.

5.2. Finite field $GF(p)$

An elliptic curve, defined over field $GF(p)$, is the set of solutions (x, y) to an equation of the form

$$y^2 = x^3 + ax + b \pmod{p}, \tag{39}$$

where $a, b \in GF(p)$ with $4a^3 + 27b^2 \neq 0 \pmod{p}$.

If (x, y) satisfies the above equation then point $P = (x, y)$ is a point on elliptic curve.

5.2.1. Point addition

When E is a curve defined as above, the inverse of the point $P = (x_1, y_1)$ is $-P = (x_1, -y_1)$. The sum $P + Q$ of the points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ (assume that $P, Q \neq \mathcal{O}$, and $P \neq \pm Q$) is the point $R = (x_3, y_3)$ where

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1}, \\ x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= (x_1 - x_3)\lambda - y_1. \end{aligned} \tag{40}$$

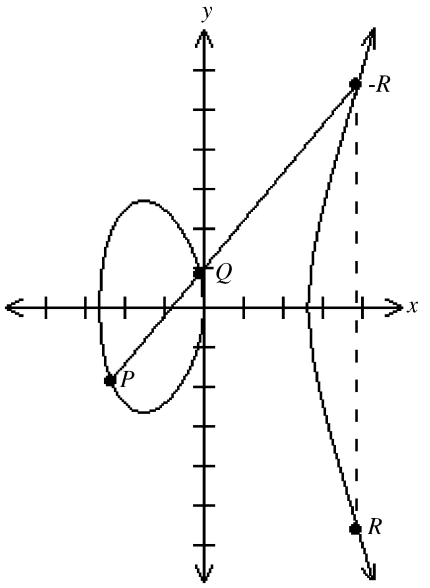

Fig. 14. Geometrical representation of point addition in the affine plane over $\mathbb{R}$. $P+Q+(-R)=0$ hence $R=P+Q$.

For $P=Q$, we get the following, "doubling" formulae:

$$\lambda = \frac{3x_1^2 + a}{2y_1},$$
$$x_3 = \lambda^2 - 2x_1,$$
$$y_3 = (x_1 - x_3)\lambda - y_1. \tag{41}$$

The point at infinity $\mathcal{O}$ plays a role analogous to that of the number 0 in ordinary addition. Thus, $P+\mathcal{O}=P$ and $P+(-P)=\mathcal{O}$ for all points P. In Fig. 14 the example of point addition for an elliptic curve over field $\mathbb{R}$ is given. A convenient way of thinking of this addition is that the three points which are the intersections of a line with the curve have sum equal to 0.

There are many types of coordinates in which an elliptic curve may be represented. In the equation above affine coordinates are used, but so-called projective coordinates have some implementation advantages. The main conclusion is that point addition can be done in projective coordinates using only field multiplications, with no inversions required. Thus, inversions are deferred, and only one needs to be performed at the end of a point multiplication operation. A projective point (X, Y, Z) on the curve satisfies the homogeneous Weierstrass equation:

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{42}$$

and, when $Z \neq 0$, it corresponds to the affine point $(X/Z, Y/Z)$. It appears that other projective representations result in more efficient implementations of the group operation. In particular, a weighted projective representation (also referred to as Jacobian representation) is preferred in the sense of faster arithmetic on elliptic curves [27,180]. In this representation a triplet (X, Y, Z)

corresponds to the affine coordinates $(X/Z^2,\ Y/Z^3)$ for $Z \neq 0$. In this case we have a weighted projective curve equation of the form

$$E: Y^2 = X^3 + aXZ^4 + bZ^6. \tag{43}$$

Weighted projective coordinates provide very fast arithmetic. Conversion from projective to affine coordinates costs 1 inversion and 4 multiplications, while vice versa is trivial. If one implements addition and doubling in a way specified in [180], the total costs for general addition is $1I + 3M$ in affine coordinates and $16M$ in projective coordinates ($11M$ if $Z_1 = 1$ i.e., one point is given in affine coordinates, and the other one in projective coordinates). Here, I and M denote the modular inversion and multiplication operations, respectively. In the case of doubling (with $a = p - 3$), this relation is $1I + 4M$ in affine coordinates against $8M$ in projective coordinates. Thus, the choice of coordinates is determined by the ratio $I : M$. Therefore, multiplication in finite field is the most important operation to focus on when working with projective coordinates. On the other hand, the extra inverter is required for affine coordinates' representation because one inversion has to be performed for every field multiplication.

5.2.2. Point multiplication

One can visualize this operation in a hierarchy structure as follows. At the top is point multiplication. It is realized by means of repeated point additions and doublings. At the next (lower) level are these operations which are closely related to the coordinates used to represent the points. At the bottom level are finite field operations such as addition, subtraction, multiplication and inversion required to perform the group operation (see formulas (40)–(41)). Some works especially done for elliptic curve point multiplication algorithm can be found in [181–183].

Point multiplication in elliptic curves is a special case of the general problem of exponentiation in abelian groups. As such, it benefits from all the techniques available for the *shortest addition chain* problem.

Certain properties of elliptic curve can be taken into account to obtain faster algorithms. These properties are as follows:

(1) Elliptic curve subtraction has the same cost as addition, so the search space for fast algorithms can be expanded to include *shortest addition–subtraction chains* and *signed representations* (see [27]).
(2) The relative complexities of general point addition and doubling have to be considered.
(3) For certain families of elliptic curves, specific shortcuts are available that can significantly reduce the computational cost.

Specific work on elliptic curve point multiplication algorithms can be found in [184] by Morain and Olivos. Solinas explained how to use NAF for an elliptic curve point multiplication in [185].

5.2.3. Hardware implementations

Information about the hardware implementation choices for elliptic curve cryptosystems can be found in [186,187].

Multiplication. As already explained, the performance of an elliptic curve cryptosystem is primarily determined by the efficient realization of the arithmetic operations (addition,

multiplication and inversion) in the underlying finite field. If projective coordinates are used the inversion operation becomes irrelevant. Therefore, coprocessors for elliptic curve cryptography are priorly designed to accelerate the field multiplication. Considering multiplication in the prime field i.e., $GF(p)$, the whole work which is done for the RSA implementation is relevant. The only difference is that shorter bit-lengths are used i.e., 160–300 bits. Scalability is again a point of concern and even more interoperability between different implementations. Therefore, the idea about unified multiplier that operates in both types of finite field $GF(p)$ and $GF(2^n)$ was naturally implied.

This idea was introduced by Savaş et al. in [108]. The authors have discussed a scalable and unified architecture for a Montgomery multiplication module. They deployed an array of word size processing units organized in a pipeline. The unified architecture requires only slightly more area than that of the multiplier architecture for the field $GF(p)$.

The same idea is the basis of work in Grosschädl [188]. The bit-serial multiplier which is introduced is performing multiplications in both types of fields. The author also modified the classical MSB-first version for iterative modular multiplication. Namely, the modular reduction in this version is also performed during multiplication. All concepts are introduced in detail, but the actual VLSI implementation is planned as a future work.

Inversion. Modular inversion is also a vital operation for PKC protocols. It is used for example, to calculate a private RSA key for decryption, for Elliptic Curve Digital Signature Algorithm (ECDSA for short) [180], etc. It is known to be the slowest operation that influenced expansion of various implementations using projective coordinates. Yet, some inversions have to be performed even then and the fastest and the most secure way to do so is a dedicated inverter in hardware.

Several inversion hardware architectures were introduced but mostly in $GF(2^n)$. Modular inversion is often performed by the Extended Euclidean Algorithm (EEA) [23]. Kaliski [189] proposed a method of Montgomery Inverse which is also derived from the EEA. Some other modular inverse studies based on this technique followed, see the work by Savaş and Koç [190]. The algorithm proposed there, was implemented in hardware in [12]. In this work two VLSI hardware implementations are presented. Both are based on the same inversion algorithm with the difference of one being fixed but fully parallel and the other being scalable. Both designs have been compared based on their speed and area. The area of the scalable design is on average 42% smaller than the fixed one. This (scalable) design is proved to be superior in almost all requirements presenting a very attractive solution for ECC.

5.2.4. State of art for ECC implementations over $GF(p)$

To the best of our knowledge, the only documented ECC processor over fields $GF(p)$ was proposed in Orlando and Paar [191]. This so-called Elliptic Curve Processor (ECP) is scalable in terms of area and speed and especially suited for FPGAs. The ECP is also best suited for projective coordinates and it uses a new type of high-radix precomputation-based Montgomery multiplier (see Section 3.3.1). It consists of three main components: the main controller (MC), the arithmetic unit controller (AUC) and the arithmetic unit (AU). The MC is controlling the point multiplication. The AUC is responsible for point operations and it controls the AU. The AU is in charge for $GF(p)$ operations. It consists of a register file, an adder and a multiplier as the most critical component. This multiplier performs a generalized version of the Montgomery

multiplication algorithm with quotient pipelining introduced in Orup [59]. This version supports positive and negative operands and features Booth recoding and precomputation. Positive and negative numbers appear often in ECC algorithms and both are equally treated as subtraction has the same cost as addition. The proposed ECP architecture was verified on an example of the field $GF(2^{192} - 2^{64} - 1)$ which is one of the field recommended by various standards. The scalability of the multiplier to larger fields was also verified in the field whose size is 521 bits. The authors have estimated eventual timing of 3 ms for computing one point multiplication.

5.3. Finite field $GF(2^n)$

The performance of an elliptic curve cryptosystem and of other public key cryptosystems in general, is mostly determined by the efficient implementation of finite field arithmetic. Special properties of binary finite fields $GF(2^n)$ such as carry-free arithmetic, linearity of squaring, etc. make them very attractive for hardware implementations.

Composite extension fields, $GF((2^n)^m)$ are not recommended for security reasons [192–194]. Namely, the method of Weil descent for solving the ECDLP, as introduced by Frey [195], can be applied to those fields. This method does not apply for curves over $GF(2^p)$ where p is prime, which are mainly defined in the standards.

Affine coordinates. We consider only non-supersingular curves, defined by equations of the form

$$E : y^2 + xy = x^3 + ax^2 + b \tag{44}$$

with $a, b \in GF(2^n), b \neq 0$.

Projective coordinates. We will now introduce weighted projective coordinates, where a projective point (X, Y, Z), $Z \neq 0$, maps the affine point $(X/Z^2, Y/Z^3)$. This corresponds to the use of a weighted projective curve equation of the form

$$E : Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6. \tag{45}$$

5.3.1. Point addition and doubling

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on E given in affine coordinates. Assume $P_1, P_2 \neq \mathcal{O}$ and $P_1 \neq -P_2$. The sum $P_3 = (x_3, y_3) = P_1 + P_2$ is computed as follows:

If $P_1 \neq P_2$,

$$\begin{aligned} \lambda &= \frac{y_2 + y_1}{x_2 + x_1}, \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1. \end{aligned} \tag{46}$$

If $P_1 = P_2$,

$$\begin{aligned} \lambda &= \frac{y_1}{x_1} + x_1, \\ x_3 &= \lambda^2 + \lambda + a, \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1. \end{aligned} \tag{47}$$

For a field of characteristic 2, the group inverse of a point $P = (x, y)$ is $-P = (x, x + y)$. In either case, the computation requires on field inversion (I), two field multiplications (M) and one squaring (S), or $1I + 2M + 1S$.

Conversion from projective to affine coordinates costs, in this case, $1I + 3M + 1S$. The computation sequences for point addition in this representation are as follows: If $P_1 \neq P_2$,

$$\begin{aligned} T_1 &= X_1 Z_2^2, \\ T_2 &= X_2 Z_1^2, \\ T_3 &= T_1 + T_2, \\ T_4 &= Y_1 Z_2^3, \\ T_5 &= Y_2 Z_1^3, \\ T_6 &= T_4 + T_5, \\ T_7 &= Z_1 T_3, \\ T_8 &= T_6 X_2 + T_7 Y_2, \\ Z_3 &= T_7 Z_2, \\ T_9 &= T_6 + Z_3, \\ X_3 &= a Z_3^2 + T_6 T_9 + T_3^3, \\ Y_3 &= T_9 X_3 + T_8 T_7^2. \end{aligned} \tag{48}$$

If $P_1 = P_2$,

$$\begin{aligned} Z_3 &= X_1 Z_1^2, \\ X_3 &= (X_1 + d Z_1^2)^4, \\ T &= Z_3 + X_1^2 + Y_1 Z_1, \\ Y_3 &= X_1^4 Z_3 + T X_3. \end{aligned} \tag{49}$$

The different costs for point addition and doubling in characteristic 2 are given in Table 3.

5.3.2. Hardware implementations

Squaring. Hasan et al. proposed a squaring structure which requires $m - 1$ XOR gates and has a time delay of one XOR gate in [134]. Jain et al. proposed semi-systolic architectures for parallel squaring in polynomial basis [130]. Information about the complexity of squaring can be found in

Table 3
Costs of point addition and doubling in characteristic 2

Operation	Coordinates		
	Affine	Mixed	Projective
General addition ($a_2 \neq 0$)	$1I + 2M + 1S$	$11M + 4S$	$15M + 5S$
General addition ($a_2 = 0$)	$1I + 2M + 1S$	$10M + 3S$	$14M + 4S$
Doubling	$1I + 2M + 1S$	n/a	$5M + 5S$

the works by Wu [141,196]. Orlando and Paar proposed a squaring architecture which can be used also for multiplications [197]. Wu used Montgomery modular multiplication method (see Section 3.3.1) for squaring in polynomial basis in [148].

Inversion. For any $\alpha \in GF(2^m)$, $\alpha^{-1} = \alpha^{2^m-2}$. Let $2^m - 2$ be decomposed as $2 + 2^2 + 2^3 + \cdots + 2^{m-1}$; then α^{-1} can be expressed as

$$\alpha^{-1} = (\alpha^2)(\alpha^{2^2})\cdots(\alpha^{2^{m-1}}). \tag{50}$$

Thus, the computation of inverse of α requires $m-1$ squaring operations and $m-2$ multiplications [153,134].

Wang et al. proposed a chip to realize Eq. (50) by using repeated square and multiply operations in normal bases [153]. Hasan et al. proposed a work for the same purpose but by use of polynomial base representation [134]. Feng, Fenn et al., Calvo and Torres and Takagi et al. proposed different representations for $2 + 2^2 + 2^3 + \cdots + 2^{m-1}$ in [198], [199], [200] and [201], respectively. Wei used the systolic architecture for modular multiplication and used this architecture for repeated multiplications to realize Eq. (50) in [139]. Wang and Guo also proposed a systolic architecture and claimed that their architecture has half the area of Wei's in [202,203].

Another decomposition for $2^m - 2$ is as follows:

$$2^m - 2 = (2^{m/2} + 1)(2^{m/2} - 2) + 2^{m/2}. \tag{51}$$

therefore α^{-1} can be computed by [204]

$$\begin{aligned}
\alpha^{-1} &= (\alpha^{2^{m/2}} \cdot \alpha)^{(2^{m/2}-2)} \cdot \alpha^{2^{m/2}} \\
&= y_1^{(2^{m/2}-2)} \cdot \alpha^{2^{m/2}} \\
&= y_1^{-1} \cdot \alpha^{2^{m/2}} \\
&= (y_1^{2^{m/4}} \cdot y_1)^{(2^{m/4}-2)} \cdot y_1^{2^{m/4}} \cdot \alpha^{2^{m/2}} \\
&= y_2^{-1} \cdot y_1^{2^{m/4}} \cdot \alpha^{2^{m/2}} \\
&\ \ \vdots
\end{aligned} \tag{52}$$

Applying similar procedure iteratively a recursive algorithm for computing multiplicative inverses in $GF(2^m)$ is given in [204] by Itoh and Tsujii. They used normal bases representation to be able to find the square easily. Asano et al. generalized the recursive algorithm proposed in [204] for multiplicative inverse computation for composite field $GF((2^m)^n)$ in [205].

Brunner et al. proposed a multiplicative structure which uses Euclid's algorithm to find the greatest common divisor (GCD) of two polynomials in [206]. Yan and Sarwate proposed systolic architecture for implementing Euclid's algorithm in [207].

Paar and Rosner gave the result of their FPGA implementation of inversion in composite fields in [138]. Information about the complexity of inverse computation can be found in the work by Wu [141]. Gao and Sobelman gave some VLSI design results by using normal bases in [162]. Hasan and Wassal [147] implemented inversion circuit which they proposed in the same work as a part of their $GF(2^m)$ arithmetic processor. They used normal bases for representation and extended Euclidean algorithm for GCD computation.

5.3.3. State of art for ECC implementations over $GF(2^m)$

The previous results about hardware implementations of elliptic curve point multiplication over $GF(2^m)$ will be given in the following sections.

In 1989 Agnew et al. reported the first result for performing the elliptic curve operations on hardware [208]. To achieve this they used their earlier normal basis multiplier as arithmetic unit and a Motorola M68008 as control unit. The system could calculate about 9 elliptic curve points per second for k with Hamming weight of 30. The throughput of the system is about 5 kb/s. Agnew et al. also used Motorola M68030 as control unit and implemented a $GF(2^{155})$ processor in [159]. If point multiplication by an integer with Hamming weight 30 is considered, this will require about 154 point doublings and 29 additions. The device will be able to perform at least 145 integer multiplication per second, approximately 50 kb/s. In elliptic curve systems, the same base point P can be used repeatedly. Then all of the squares can be precomputed, which will increase the throughput by a factor of 4 to approximately 200 kb/s. The storage requirements for the point squarings is less than 6 Kbytes.

In the work of Agnew et al., how Diffie–Hellman and ElGamal protocols can be efficiently implemented using the group of an elliptic curve over a finite field is described. VLSI implementation of an arithmetic processor in $GF(2^{155})$ is discussed [209]. In designing the arithmetic coprocessor, the following criteria were deemed essential:

- a compact architecture consisting of only the essential registers and the interconnections required to realize the optimal normal basis multiplier,
- the coprocessor must have an extremely fast and wide I/O system,
- a simple and efficient instruction set.

The instruction set and cycle counts are shown in Table 4.

The processor is called GF155MIC. The finished device required the equivalent of about 11,000 gates and ran at the design target speed of 40 MHz (clock).

Sutikno et al. also proposed a VLSI design and implementation of arithmetic processor over $GF(2^{155})$ in [210]. In this work the field inversion is performed by the method of Agnew et al. [211]. It needs 23 field multiplications for $m = 155$. This method is used for accomplishing area optimization. The arithmetic processor has 32-bit wide internal bus. It has 15 instructions and 5-bit length for its operation codes (3 bit LSB used for selecting instruction and 2 MSB for selecting the active register.)

The instruction set and cycle counts are shown in Tables 4 and 5.

The architecture of the arithmetic processor is shown in Fig. 15. The arithmetic processor includes the subsystems such as

(1) Arithmetic module.
(2) Main controller.
(3) Register selector.
(4) Internal 32-bit bidirectional bus.

The arithmetic processor is implemented on FPGA based using Xilinx XC4020XL. The finished device requires the equivalent of about 17,000 gates and runs at the design target speed of 10–15 Mhz.

Table 4
Instruction set

Operation	Size	Clock cycles
MULT	155 bit blocks	156
INVERSE	24 multiplications	Approx. 3 800
I/O	5-32 bit transfers per register	10
WRITE(A, B, or C)	Write to register	2
READ(A, B, or C)	Read from register	2
NOP		
ROTATE(A, B, or C)	155 bit parallel operation	2
COPY	155 bit parallel operation	2
($A \leftarrow B$)		
($A \leftarrow C$)		
($B \leftarrow A$)		
($B \leftarrow C$)		
SWAP($A \leftrightarrow B$)	155 bit parallel operation	2
CLEAR(A, B, or C)	155 bit parallel operation	2
SET(A, B, or C)	155 bit parallel operation	2
ADD($A \oplus B$)	155 bit parallel operation	2
ACCUMULATE	155 bit parallel operation	2

Table 5
Instruction set

Operation	Size	Clock cycles
MULT	155 bit blocks	157
INVERSE	23 multiplications	Approx. 3 887
ADD($A \oplus B$)	155 bit parallel operation	1
Input(A or B)	Store data to A or B by 5-32 bit transfers	11
Output(A, B or C)	Load data from A, B, or C	11
	by 5-32 bit transfers	11
ROTATE(A, B)	155 bit parallel operation	1
COPY	155 bit parallel operation	5
($A \leftarrow C$)		
($B \leftarrow C$)		
SET(A, B, or C)	155 bit parallel operation	1

Sutikno et al. also presented the use of non-supersingular elliptic curve group over $GF(2^{155})$ and a VLSI implementation of an ElGamal ECC processor in [212]. The architecture of the ElGamal ECC processor is shown in Fig. 16. There are eleven 155-bit wide registers to store data and intermediate results from arithmetic process. 5 registers to store x_1, y_1, z_1, x_2 and y_2; 4 registers to store intermediate results A, B, C and D; 1 register to store a_6 and 1 cyclic register to store k. In the control module, there are five control blocks: main control, repeat square and multiply control, adding point control, doubling point control and inversion control. The throughput rate

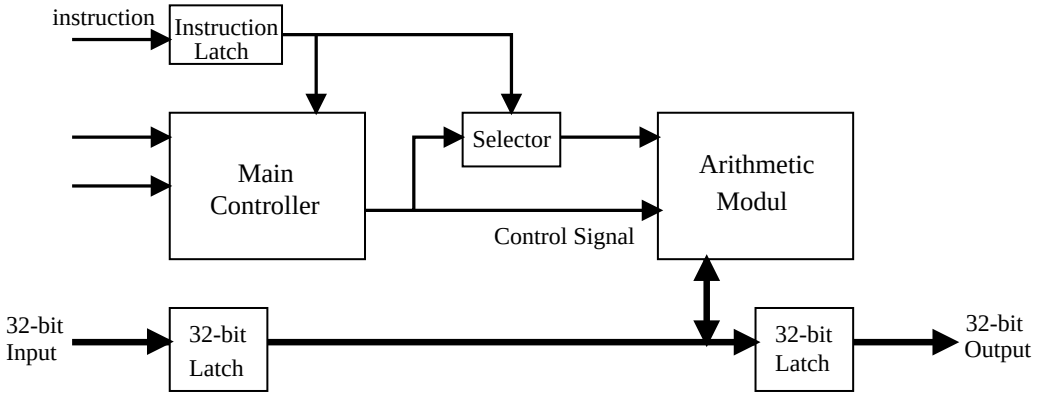

Fig. 15. Architecture of arithmetic processor.

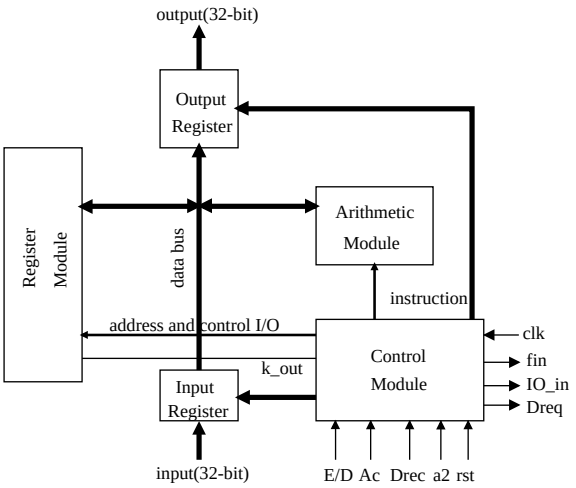

Fig. 16. Architecture of the ElGamal ECC processor.

for encryption is estimated as 6.5×10^{-4} bit/clock cycle and for decryption 13.1×10^{-4} bit/clock cycle. The area on FPGA is estimated as 40–50 thousand gates.

Gao et al. proposed an elliptic curve cryptosystem coprocessor with variable key size, which utilizes the internal SRAM/registers in an FPGA in [213,214]. The architecture of the arithmetic processor is shown in Fig. 17. The controller is the kernel of the scalar multiplier and has the format of an FSM with table look-up to implement logic functions. All operations can be categorized as one of the following atomic operations: unconditional jump, conditional jump, operand load, operand store, finite field addition, finite field squaring, finite field multiplication and finite field inversion. The scalar is decomposed as a NAF and the scalar multiplication is done with a series of addition/subtractions of elliptic curve points. The mapping/layout results are shown in Table 6.

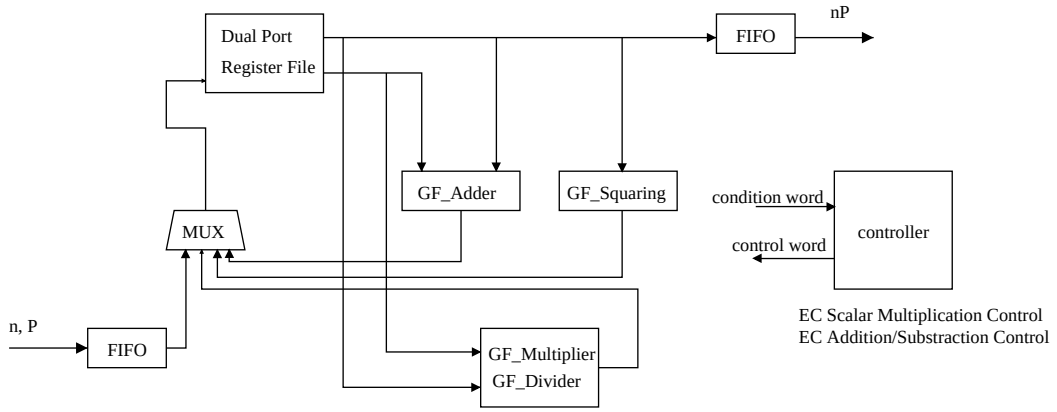

Fig. 17. Architecture of arithmetic processor.

Table 6
FPGA chip area utilization and throughput

Value of m	XC4000XL device	CLB usage	Clock cycles	Throughput (scalar mul/s)
5	4010XL	$272/400 = 68\%$	126	179,856
11	4013XL	$478/576 = 83\%$	825	19,230
29	4028XL	$962/1024 = 93\%$	7158	1653
53	4044XL	$1626/1936 = 84\%$	26,753	417

The expected mapping results with an estimation formula are listed below:

(1) CLB structure of Xilinx XC4000XL-series:
- 2 FFs per CLB,
- 2 function generators (FGs) per CLB (4 input, single output logic unit),
- 2 single port 16×1 RAMs per CLB (using two logic units),
- 1 dual-port 16×1 RAMs per CLB (using two logic units).

(2) Cost of implementation of basic components
- two m-bit registers $2m$ FFs,
- three m-bit shift registers take $3m$ FFs and FGs,
- four m-bit 2:1 MUXs take $4m$ FGs,
- one m-bit GF Adder takes m FGs,
- one m-bit dual-port 9 word RAM takes $2m$ FGs,
- one m-bit 6 word FIFO takes $6(m+1)$ FFs and FGs,
- one m-bit 2 word FIFO takes $2(m+1)$ FFs and FGs,
- one m-bit GF multiplier takes $5m$ FFs and $3m$ FGs,
- one m-bit GF inverter takes $3(m+\log_2 m)$ FFs and FGs (excluding GF multiplier).

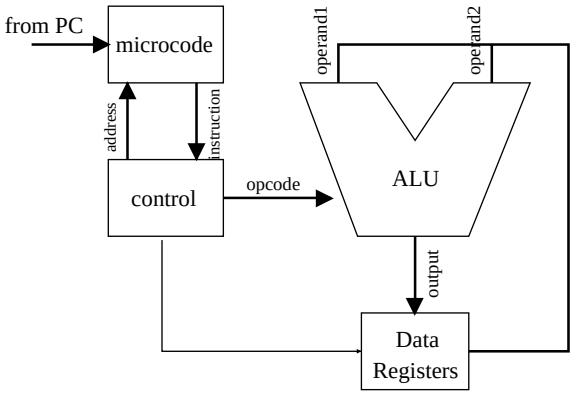

Fig. 18. ECC processor architecture.

Table 7
Instruction set

Operation	Clock cycles
NOP	1
XOR	1
Rotate left, ROTL	1
Shift right, SHFR	1
Field multiplication, MUL	$n+1$
Transfer register value, TFR	1
Jump instructions, JKZ, JCZ, JMP	

From these building blocks, the cost of one EC scalar multiplier with key size m is derived as:

- Total FFs $= 21m + 3\log_2 m + 48$.
- Total FGs $= 24m + 3\log_2 m + 308$.
- Minimal number of CLBs $= 12m + (3\log_2 m)/2 + 154$.
- Maximal number of CLBs $= 45m + 6\log_2 m + 356$.

Hauck et al. proposed a elliptic curve cryptosystem chip using asynchronous wave pipelines (AWP) in [215]. They give the simulation results of their design and reported that circuit run at a rate of 1.5 GHz in 0.35 μm CMOS technology.

Leung et al. described a Xilinx Virtex based FPGA implementation of an elliptic curve processor in [216]. A block diagram of the elliptic curve processor is shown in Fig. 18. It consists of an arithmetic logic unit (ALU), register file, a microcode sequencer and microcode storage. In ALU a Massey-Omura Multiplier is used. A $16 \times n$-bit dual-port synchronous register file is constructed from the 16×1-bit distributed RAM feature of the Xilinx Virtex series. The instruction set of the processor is shown in Table 7. Apart from instructions which directly control the ALU, there are three types of jump instructions: JMP-jump unconditionally, JKZ-jump if the least significant bit of K counter is zero and JCZ-jump if the C register is zero. Another FPGA implementation for 270-bit operations similar to the previous was proposed by Ernst et al. in

Table 8
Resource utilization and maximum clock rate for different n on Xilinx XCV300-4. The Xilinx XCV300 contains 3 072 slices (1536 CLBs)

n	No. of slices	Max. freq (MHz)
113	1290	45
155	1567	36
281	2622	33

Table 9
Execution time for elliptic curve point multiplication

n	No. of cycles	Time (ms)
113	166,738	3.7
155	266,443	6.8
281	474,504	14.4

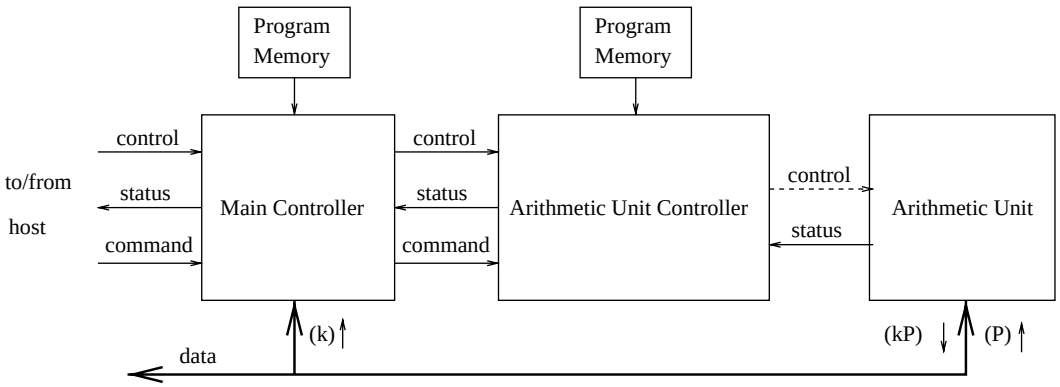

Fig. 19. Elliptic curve processor architecture.

[217]. They used XC4085XLA FPGA for implementation and reported that 180,000 system gates were used. The clock speed was 34 MHz and resulting performance is 146 point multiplications per second.

Table 8 shows the resource utilization and maximum clock rate reported by the Xilinx tools for designs with different n.

The total number of cycles required for an elliptic curve multiplication for various n is given in Table 9. The time required for an elliptic curve multiplication at the maximum frequency is shown in Table 9.

Orlando and Paar proposed in 2000 a scalable elliptic curve processor architecture which operates over finite fields $GF(2^m)$ in [218]. The elliptic curve processor (ECP), shown in Fig. 19, consists of three main components. These components are the main controller (MC), the arithmetic unit controller (AUC) and the arithmetic unit (AU). The MC is the ECP's main controller. It orchestrates the computation of kP and interacts with the host system. The AUC

controls the AU. It orchestrates the computation of point additions, point doublings, and coordinate conversions. The AU performs the $GF(2^m)$ field additions, squares, multiplications and inversions under AUC control. The MC executes the double-and-add and the Montgomery scalar multiplication (see Section 3.3.1) functions, the AUC performs all the other subroutines and the AU is the hardware that computes the finite field operations.

The following is a typical sequence of steps for the computation of kP in the ECP using the double-and-add algorithm and projective coordinates. First, the host loads k into the MC, loads the coordinates of P into the AU and commands the MC to start processing. Then, the MC does its initialization, which includes finding the most significant non-zero coefficient of k. The MC then commands the AUC to perform its initialization, which includes the conversion of P from affine to projective coordinates. During the computation of kP, the MC scans one bit of k at time starting with the second most significant coefficient and ending with the least significant one. In each of these iterations, the MC commands the AU/AUC to do a point double. If the scanned bit is a 1, it also commands the AU/AUC to do a point addition. For each of these point operations, the AUC generates the control sequence that guides the AU through the computation of the required field operations. After the least significant bit of k is processed, the MC commands the AU/AUC to convert the result back to affine coordinates. When the AU/AUC finishes this operation, the MC signals to the host the completion of the kP operation. Finally, the host reads the coordinates of kP from the AU.

Three ECP prototypes were built. These prototypes support elliptic curves over the field $GF(2^{167})$, with this field being defined by the field polynomial $F(x) = x^{167} + x^6 + 1$.

Each prototype used a 16-bit MC processor with 256 words of program memory, a 24-bit AUC processor with 512 words of program memory and 128 registers, each of which is 167 bits wide. They also provided 32-bit I/O interface to the host system. The prototypes used LSD multipliers with digit sizes equal to 4, 8 and 16. The prototypes were implemented using the Xilinx XCV400E8BG432 (Virtex E) FPGA. The ECP prototypes were tested with two programs. One of the programs implemented the projective coordinates version of the Montgomery scalar multiplication algorithm and the other the projective coordinates version of the traditional double-and-add algorithm, none of which relies on precomputation. The number of clock cycles required to compute kP for each of the programs is summarized in Table 10. D is the digit size of the multiplier being used.

For both elliptic curve algorithms, the MC program used 56% of the MC's program memory. The AUC program used 90–98% of the AUC's program memory depending on the algorithm and

Table 10
Number of clock cycles required to compute kP over $GF(2^{167})$

Operation	Double-and-add No. clock cycles	Montgomery No. clock cycles
Point double	$5\lceil 167/D \rceil + 25$	$6\lceil 167/D \rceil + 17$
Point add	$11\lceil 167/D \rceil + 31$	
Coor. Conv., etc.,	$13\lceil 167/D \rceil + 575$	$20\lceil 167/D \rceil + 764$
kP	$(10.5\lceil 167/D \rceil + 47.5)*166$ $+13\lceil 167/D \rceil + 575$	$(6\lceil 167/D \rceil + 24)*166$ $+20\lceil 167/D \rceil + 764$

Table 11
Approximation of number of clock cycles required to compute kP over $GF(2^m)$

Operation	Double-and-add No. Clockcycles	Montgomery No. Clockcycles
Point double	$5\lceil m/D \rceil$	$6\lceil m/D \rceil$
Point add	$11\lceil m/D \rceil$	
Coor. conv., etc.,	$(3 + (\lfloor \log_2(m-1) \rfloor + W(m-1) - 1))\lceil m/D \rceil$	$(10 + (\lfloor \log_2(m-1) \rfloor + W(m-1) - 1))\lceil m/D \rceil$
kP	$(10.5(m-1) + 3 + (\lfloor \log_2(m-1) \rfloor + W(m-1) - 1))\lceil m/D \rceil$	$(6(m-1) + 10 + (\lfloor \log_2(m-1) \rfloor + W(m-1) - 1))\lceil m/D \rceil$

Table 12
Logic complexity of ECP prototypes

Digit size	No. LUTs	No. FFs	No. Block RAMs
4	1627	1745	10
8	2136	1753	10
16	3002	1769	10

the digit size. Table 11 approximates the number of cycles required for the computation of point multiplication for arbitrary $GF(2^m)$ fields. The approximations are based exclusively on the number of multiplications and the number of clock cycles required to compute them with an LSD multiplier with digit size D. The inversion is assumed to require $\lfloor \log_2(m-1) \rfloor + W(m-1) - 1$ multiplications, where $W(m-1)$ represents the number of non-zero coefficients in the binary representation of $m-1$.

The logic complexity of the ECP prototypes is summarized in Table 12 in terms of the main components of modern FPGAs. These components are lookup tables (LUT) which are used as programmable gates, flip-flops (FF) and BlockRAM which are configurable 4 kbit RAMs. The normalized complexity of the ECP prototypes is approximately $228 + 6.6m + (\lceil 2D/3 \rceil - 1)m$ LUTs, $224 + 9.2m$ FFs and $4 + \lceil m/32 \rceil$ 4 kbit BlockRAMs for $m \gg D$, 4-input LUTs, 32-bit BlockRAMs and D a multiple of 4.

The prototype implementations used between 15% and 28% of the LUTs (depending on the digit size), 16% of the FFs and 25% of the BlockRAMs available in the XCV400E8BG432 FPGA. Together, the AUC and the MC processors, ignoring the complexity of the register that holds the k operand, used less than 13% of the logic resources and 40% of the memory elements. In turn, the AU used 76–87% of the LUTs, 59% of the flip-flops and 60% of the memory elements. The remaining resources were used by system I/O logic.

Goodman and Chandrakasan proposed a domain-specific reconfigurable cryptographic processor (DSRCP) in [219]. The instruction set definition of the DSRCP was dictated by the IEEE 1363 Public Key Cryptography Standard document [180]. A list of the arithmetic functions required to implement the various primitives defined in the standard was tabulated in a functional matrix, which was then used to define the instruction set architecture (ISA) of the processor. The ISA contains 24 instructions broken up into six types of operations: conventional arithmetic, modular integer arithmetic, GF arithmetic, elliptic curve field arithmetic over GF, register

Table 13
DSRCP instruction mapping within the control hierarchy

Part	Instructions
I	ADD, SUB, COMP, GF_INV, GF_INVMULT, GF_MULT, GF_ADD, MONTRED, MONTMULT, MONTRED_A, SET_LENGTH
II	EC_DOUBLE, EC_ADD, MOD_ADD, MOD_SUB, GF_EXP
III	EC_MULT, MOD_MULT, MOD, MOD_INV, MOD_EXP

manipulation and processor configuration. The processor consists of four main architectural blocks: the global controller and microcode ROMs, the I/O interface, the shutdown controller and the reconfigurable datapath. Operands used within the processor can vary in size from 8 to 1024 bits requiring the use of a flexible I/O interface that allows the user to transfer data to/from the processor in a very efficient manner. The primary component of the DSRCP is the reconfigurable datapath. The datapath consists of four major functional blocks: an eight-word register file, a fast adder unit, a comparator unit and the main reconfigurable logic unit.

The DSRCP performs a variety of algorithms ranging from modular integer arithmetic to elliptic curve arithmetic over GF. All operations are universal in that they can be performed using any valid n-bit modulus ($8 \leqslant n \leqslant 1024$), $GF(2^n)$ field polynomial and non-supersingular elliptic curve over $GF(2^n)$. The various complex modular arithmetic operations (multiplication, reduction, inversion and exponentiation) are implemented using microcode, while simple operations (addition and subtraction) are implemented directly in hardware using the wide adder and comparator units. Multiplication is performed using Montgomery multiplication [35].

The instruction set partitioning of the three-level control hierarchy is shown in Table 13. The first part of control corresponds to those instructions that are implemented directly in hardware. The second part of control represents the first level of microcoded instructions that are composed of sequences of first-part instructions. Similarly, the third part of control represents instructions that consist of sequences of both first- and second-part instructions. Each microcode controller consists of a small ROM core, an input selector which gates the appropriate values onto the corresponding operand signals and a control FSM that also serves as the ROM address generator.

The processor is fabricated in a 0.25 μm CMOS technology with five levels of metallization. The core contains 880,000 devices and measures 2.9×2.9 mm^2. The datapath consists of 1024 processing bit-slices, each of which measures 30×150 μm^2. At 50 MHz, the processor operates at a supply voltage of 2 V and consumes at most 75 mW of power. In ultra-low-power mode (3 MHz at $V_{DD} = 0.7V$), the processor consumes at most 525 μW.

Janssens et al. proposed a high-speed hardware/software co-design for computing elliptic curve point multiplications and implemented on an Atmel Field Programmable System Level Integration Circuit (FPSLIC) [220]. The design hierarchy is shown in Fig. 20. The design consists of a Data-path, which performs the finite field arithmetic and two finite state machines that each controls a particular part of the functionality at a particular level of hierarchy. At the highest level the Software Controller is the master. It gives instructions to the Hardware Controller, which

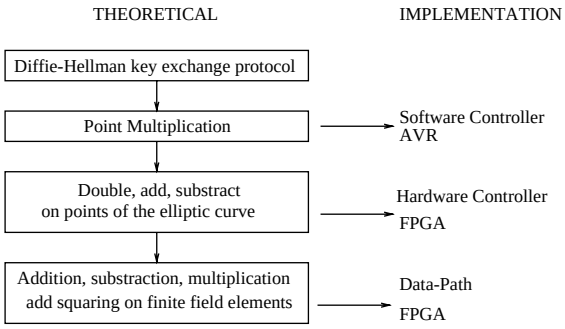

Fig. 20. The design hierarchy.

Table 14
FPSLIC chip area utilization and throughput

Value of n	CLB usage	Clock cycles	Point mult/s 10 MHz	Point mult/s 200 MHz
8	668	768	13,020	260,416
16	852	3 072	3 255	65,104
72	2 189	62,208	160	3 215
192	4 097	442,368	22	452

translates these instructions in a sequence of direct control signals for the operators in the Data-path.

The design flow which is proposed by the Atmel System Designer software was followed. On average, $125n$ instructions for the AVR are needed to read in data, perform the point multiplication and write out the results. Since the embedded AVR core achieves throughput approaching 1 MIPS per MHz, this corresponds with $125n$ clock cycles. The critical path after synthesis is 33 *ns*. The corresponding maximum clock frequency is nearly 30 MHz. After place-and-route it turned out that 23 Combinatorial Logic Blocks (CLBs) are needed per bit slice and 496 CLBs for the hardware controller. Since there are only 2 304 CLBs available on the FPGA, a design for a key length of 72 bits can be implemented.

The results above are used to estimate the total time it takes to perform a complete point multiplication. Since the Software Controller is always waiting on an interrupt from the Hardware Controller before sending a new instruction, the Hardware Controller receives this new instruction almost directly after it has finished performing the former instruction. Therefore we can take only the number of clock cycles needed by the hardware part, which is in the average case $12n^2$. Table 14 presents the number of clock cycles, the corresponding throughput and CLB Usage for different key lengths.

Schaumont and Verbauwhede introduced an Elliptic Curve Processor over $GF(2^n)$ in [5,221]. The architecture has a layered structure with the layers corresponding to the operations described in the security pyramid shown in Fig. 1. The authors propose a language and simulation environment that allows to explore the design of security domain specific processors at a high abstraction level.

Bednara et al. gave a comparison of several ways for hardware implementation of elliptic curve cryptosystems in [222]. They especially focused on FPGA based implementations. Kim and Lee presented in [223] an elliptic curve processor which consists of control unit, arithmetic unit and register unit.

6. Conclusions

We have presented an overview of the wide variety of architectures which have been designed to implement Public Key Cryptography. Creating a working implementation was a significant challenge in the 1980s; the number of hardware implementations that made it to prototype or production phase was very limited. In the 1990s, we have seen significant progress due to a combination of better algorithms and advances in VLSI technology. In addition, Elliptic Curve Cryptography may allow more compact implementations. Cryptographic hardware accelerator modules are now a commodity for Virtual Private Networks (VPNs) and e-commerce transactions; they can even be found in smartcard co-processors. In the area of smartcards, we have seen an increasing number of compact yet performative co-processors for Public Key Cryptography.

To summarize reviewing various hardware architectures for PKC one conclusion is imposing itself. Hardware as seen in 1980s is still relevant in the sense of its fundamentals such as systolic array and RNS realizations. On the other hand, for ECC hardware is still a long way to go. Also, both algorithms are constantly being pushed by current applications to be even faster (which usually implies more secure), in order to fulfill industry and government demands.

There is no doubt that in the coming years even more performant hardware implementations will be developed for high-end applications. We believe that the biggest challenge ahead may be the development of very compact and inexpensive low-power implementations that allow protection for personal devices and devices used in the context of ambient intelligence. A second challenge is to develop efficient implementations that offer adequate security against the ever more sophisticated side-channel attacks.

References

[1] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (1976) 644–654.
[2] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.
[3] N. Koblitz, Elliptic curve cryptosystem, Math. Comp. 48 (1987) 203–209.
[4] V. Miller, Uses of elliptic curves in cryptography, in: H.C. Williams (Ed.), Advances in Cryptology: Proceedings of CRYPTO'85, Lecture Notes in Computer Science, Vol. 218, Springer, Berlin, 1985, pp. 417–426.
[5] P. Schaumont, I. Verbauwhede, A reconfiguration hierarchy for Elliptic Curve Cryptography, in: Proceedings of the 35th Asilomar Conference on Signals, Systems, and Computers, 2001.
[6] M. Abdelguerfi, B.S. Kaliski Jr., W. Patterson, Public key security systems, IEEE Micro. 16 (3) (1996) 10–13.
[7] T. Beth, D. Gollmann, Algorithm engineering for public key algorithm, IEEE J. Selected Areas in Commun. 7 (4) (1989) 458–465.
[8] Certicom Corp., Current public-key cryptographic systems, Whitepaper, http://www.certicom.com (2000).

[9] C. Paar, Reconfigurable hardware in modern cryptography, Presentation on the 4th Workshop on Elliptic Curve Cryptography (ECC 2000), http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc00/slides.html (October 2–4 2000).
[10] A.F. Tenca, Ç.K. Koç, A scalable architecture for Montgomery multiplication, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, 1999, pp. 94–108.
[11] A. Gutub, A scalable hardware for montgomery modular inverse computation, Tech. Report, Information Security Laboratory, Electrical and Computer Engineering Department, Oregon State University, Corvallis, Oregon 97331 (June 2001).
[12] A. Gutub, A.F. Tenca, Ç.K. Koç, Scalable VLSI architecture for $GF(p)$ Montgomery modular inverse computation, in: Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2002, pp. 53–58.
[13] L. Batina, G. Muurling, Montgomery in practice: How to do it more efficiently in hardware, in: B. Preneel (Ed.), Proceedings of RSA 2002 Cryptographers' Track, Lecture Notes in Computer Science, Vol. 2271, Springer, San Jose, USA, 2002, pp. 40–52.
[14] A. Lenstra, E. Verheul, Selecting cryptographic key sizes, in: H. Imai, Y. Zheng (Eds.), Proceedings of Third International Workshop on Practice and Theory in Public Key Cryptography (PKC 2000), Lecture Notes in Computer Science, Vol. 1751, Springer, Berlin, 2000, pp. 446–465.
[15] R. Anderson, M. Kuhn, Tamper resistance—a cautionary note, in: Proceedings of the 2nd USENIX Workshop on Electronic Commerce, 1996, pp. 1–11.
[16] R. Anderson, M. Kuhn, Low cost attacks on tamper resistant devices, in: B. C, et al., (Eds.), Proceedings of 1997 Security Protocols Workshop, Lecture Notes in Computer Science, Vol. 1361, Springer, Berlin, 1997, pp. 125–136.
[17] K. Gandolfi, C. Mourtel, F. Olivier, Electromagnetic analysis: concrete results, in: Ç.K. Koç, D. Naccache, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001), Lecture Notes in Computer Science, Vol. 2162, Springer, Berlin, 2001, pp. 255–265.
[18] G. Hachez, F. Koeune, J.-J. Quisquater, Timing attack: what can be achieved by a powerful adversary?, in: A. Barbé, E.C. van der Meulen, P. Vanroose (Eds.), Proceedings of the 20th Symposium on Information Theory in the Benelux, 1999, pp. 63–70.
[19] P. Kocher, Timing attacks on implementations of Diffie–Hellman, RSA, DSS and other systems, in: N. Koblitz (Ed.), Advances in Cryptology: Proceedings of CRYPTO'96, Lecture Notes in Computer Science, Vol. 1109, Springer, Berlin, 1996, pp. 104–113.
[20] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: M. Wiener (Ed.), Advances in Cryptology: Proceedings of CRYPTO'99, Lecture Notes in Computer Science, Vol. 1666, Springer, Berlin, 1999, pp. 388–397.
[21] T.S. Messerges, E.A. Dabbish, R.H. Sloan, Examining smart-card security under the threat of power analysis attacks, IEEE Trans. Comput. 51 (5) (2002) 541–552.
[22] D. Boneh, R.A. DeMillo, R.J. Lipton, On the importance of checking cryptographic protocols for faults (extended abstract), in: W. Fumy (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'97, Lecture Notes in Computer Science, Vol. 1233, Springer, Berlin, 1997, pp. 37–51.
[23] N. Koblitz, A Course in Number Theory and Cryptography, 2nd Edition, Graduate Text in Mathematics, Vol. 114, Springer, Berlin, Germany, 1994.
[24] D. Boneh, Twenty years of attacks on the RSA cryptosystem, Notices Am. Math. Soc. 46 (2) (1999) 203–213.
[25] D.E. Knuth, The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, MA, 1997.
[26] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, 1997.
[27] I. Blake, G. Seroussi, N.P. Smart, Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series, Cambridge University Press, Cambridge, 1999.
[28] D.M. Gordon, A survey of fast exponentiation methods, J. Algorithms 27 (1998) 129–146.
[29] R. Gallant, R. Lambert, S. Vanstone, Faster point multiplication on elliptic curves with efficient endomorphisms, in: J. Kilian (Ed.), Advances in Cryptology: Proceedings of CRYPTO'01, Lecture Notes in Computer Science, Vol. 2139, Springer, Berlin, 2001, pp. 190–200.

[30] Ç.K. Koç, Analysis of sliding window techniques for exponentiation, Comput. Math. Appl. 10 (30) (1995) 17–24.
[31] C.H. Lim, P.J. Lee, More flexible exponentiations with precomputation, in: Y. Desmedt (Ed.), Advances in Cryptology: Proceedings of CRYPTO'94, Lecture Notes in Computer Science, Vol. 839, Springer, Berlin, 1994, pp. 95–107.
[32] D. Gollmann, Y. Han, C. Mitchell, Redundant integer representation and fast exponentiation, Designs, Codes Cryptogr. 7 (1998) 135–151.
[33] C.D. Walter, MIST: an efficient, randomized exponentiation algorithm for resisting power analysis, in: B. Preneel (Ed.), Proceedings of RSA 2002 Cryptographers' Track, Lecture Notes in Computer Science, Vol. 2271, Springer, San Jose, USA, 2002, pp. 53–66.
[34] C.D. Walter, Some security aspects of the MIST randomized exponentiation algorithm, in: B.S. Kaliski Jr., Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, Springer, Berlin, 2002.
[35] P. Montgomery, Modular multiplication without trial division, Math. Comput. 44 (1985) 519–521.
[36] C.D. Walter, Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli, in: B. Preneel (Ed.), Proceedings of Topics in Cryptology- CT-RSA 2002, Lecture Notes in Computer Science, Vol. 2271, Springer, Berlin, 2002, pp. 30–39.
[37] C.D. Walter, Montgomery exponentiation needs no final subtraction, Electron. Lett. 35 (21) (1999) 1831–1832.
[38] G. Hachez, J.-J. Quisquater, Montgomery exponentiation with no final subtractions: Improved results, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), Lecture Notes in Computer Science, Vol. 1965, 2000, pp. 293–301.
[39] Ç.K. Koç, T. Acar, B.S. Kaliski Jr., Analyzing and comparing Montgomery multiplication algorithms, IEEE Micro. (1996) 26–33.
[40] S.E. Eldridge, C.D. Walter, Hardware implementation of Montgomery's modular multiplication algorithm. IEEE Trans. Comput. 42 (93) 693–699.
[41] C.D. Walter, Montgomery's multiplication technique: How to make it smaller and faster, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, 1999, pp. 80–93.
[42] R.L. Rivest, A short report on the RSA chip, in: D. Chaum, R.L. Rivest, A.T. Sherman (Eds.), Advances in Cryptology: Proceedings of CRYPTO'82, Lecture Notes in Computer Science, Vol. 740, Springer, Berlin, 1982, p. 327.
[43] A.J. Atrubin, A one-dimensional real-time iterative multiplier, IEEE Trans. Electron. Comput. 14 (1965) 394–399.
[44] K.T. Johnson, A.R. Hurson, B. Shirazi, General-purpose systolic arrays, IEEE Comput. 26 (11) (1993) 20–31.
[45] B. Dixon, A.K. Lenstra, Massively parallel elliptic curve factoring, in: R.A. Rueppel (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'92, Lecture Notes in Computer Science, Vol. 658, Springer, Berlin, 1992, pp. 183–193.
[46] H.W. Lenstra Jr., Factoring integers with elliptic curves, Ann. Math. 126 (1987) 649–673.
[47] C.D. Walter, An improved linear systolic array for fast modular exponentiation, IEEE Comput. Digital Tech. 147 (5) (2000) 323–328.
[48] K. Iwamura, T. Matsumoto, H. Imai, Montgomery modular multiplication method and systolic arrays suitable for modular exponentiation, Electron. Commun. Japan 77 (3) (1994) 40–50.
[49] C.D. Walter, Systolic modular multiplication, IEEE Trans. Comput. 42 (1993) 376–378.
[50] E. Trichina, A. Tiountchik, Scalable algorithm for Montgomery multiplication and its implementation on the coarse-grain reconfigurable chip, in: D. Naccache (Ed.), Proceedings of Topics in Cryptology-CT-RSA 2001, Lecture Notes in Computer Science, Vol. 2020, Springer, Berlin, 2001, pp. 235–249.
[51] T. Blum, Modular exponentiation on reconfigurable hardware, Master's Thesis, Worcester Polytechnic Institute, April 1999.
[52] P. Kornerup, A systolic, linear-array multiplier for a class of right-shift algorithms, IEEE Trans. Comput. 43 (8) (1994) 892–898.
[53] C.-Y. Su, S.-A. Hwang, P.-S. Chen, C.-W. Wu, An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem, IEEE Trans. VLSI systems 7 (2) (1999) 280–284.

[54] W.-C. Tsai, C. B. Shung, S.-J. Wang, Two systolic architectures for modular multiplication, IEEE Trans. VLSI systems 8 (1) (2000) 103–107.
[55] Y.-J. Jeong, W.P. Burleson, VLSI array algorithms and architectures for RSA modular multiplication, IEEE Trans. VLSI systems 5 (2) (1997) 211–217.
[56] S. Even, Systolic modular multiplication, in: A.J. Menezes, S.A. Vanstone (Eds.), Advances in Cryptology: Proceedings of CRYPTO'90, Lecture Notes in Computer Science, Vol. 537, Springer, Berlin, 1990, pp. 619–624.
[57] K. Iwamura, T. Matsumoto, H. Imai, Systolic-arrays for modular exponentiation using Montgomery method, in: R.A. Rueppel (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'92, Lecture Notes in Computer Science, Vol. 658, Springer, Berlin, 1992, pp. 477–481.
[58] E.F. Brickell, A fast modular multiplication algorithm with application to two key cryptography, in: D. Chaum, R.L. Rivest, A.T. Sherman (Eds.), Advances in Cryptology: Proceedings of CRYPTO'82, Vol. 740, Plenum Press, New York and London, Springer, Berlin, 1982, pp. 51–60.
[59] H. Orup, Simplifying quotient determination in high-radix modular multiplication, in: Proceedings of the 12th Symposium on Computer Arithmetic, IEEE, 1995, pp. 193–199.
[60] K. Iwamura, T. Matsumoto, H. Imai, High-speed implementation methods for RSA scheme, in: R.A. Rueppel (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'92, Lecture Notes in Computer Science, Vol. 658, Springer, Berlin, 1992, pp. 221–238.
[61] A.A. Tiountchik, Systolic modular exponentiation via Montgomery algorithm, Electron. Lett. 34 (9) (1998) 874–875.
[62] A.A. Tiountchik, E. Trichina, RSA acceleration with field programmable gate arrays, in: Proceedings of ACISP 1999, 1999, pp. 164–176.
[63] J. Poldre, K. Tammemae, M. Mandre, Modular exponent realization on FPGAs, in: Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm (FPL'98), Tallinn, Estonia, 1998, pp. 336–347.
[64] T. Blum, C. Paar, Montgomery modular exponentiation on reconfigurable hardware, in: Proceedings of the 14th IEEE Symposium on Computer Arithmetic, 1999, pp. 70–77.
[65] T. Blum, C. Paar, High-radix Montgomery modular exponentiation on reconfigurable hardware, IEEE Trans. Comput. 50 (7) (2001) 759–764.
[66] A.J. Elbirt, C. Paar, Towards an FPGA architecture optimized for public-key algorithms, in: J. Schewel et al. (Eds.), Proceedings of the SPIE's Symposium on Voice, and Communications, Boston, MA, USA, 1999, pp. 33–42.
[67] W.L. Freking, K.K. Parhi, Performance-scalable array architectures for modular multiplication, in: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors, IEEE, 2000, pp. 149–160.
[68] W.L. Freking, K.K. Parhi, A unified method for iterative computation of modular multiplication and reduction operations, in: Proceedings of the IEEE International Conference on Computer Design, IEEE, 1999, pp. 80–87.
[69] W.P. Marnane, Optimised bit serial modular multiplier for implementation on field programmable gate arrays, Electron. Lett. 34 (8) (1998) 738–739.
[70] Y.S. Kim, W.S. Kang, J.R. Choi, Implementation of 1024-bit modular processor for RSA cryptosystem, in: Proceedings of Asia-Pasific Conference on ASIC (AP-ASIC), Cheju Island, Korea, 2000.
[71] M.J. Norris, G.J. Simmons, Algorithms for high-speed modular arithmetic, Congr. Numer. 31 (1981) 153–163.
[72] C.D. Walter, S.E. Eldridge, A verification of Brickell's fast modular multiplication algorithm, Int. J. Comput. Math. 33 (1990) 153–169.
[73] G. Orton, M. Roy, L. Peppard, S. Tavares, VLSI implementation of public-key encryption algorithms, in: A.M. Odlyzko (Ed.), Advances in Cryptology: Proceedings of CRYPTO'86, Lecture Notes in Computer Science, Vol. 263, Springer, Berlin, 1986, pp. 277–301.
[74] H. Sedlak, The RSA cryptography processor, in: D. Chaum, W.L. Price (Eds.), Advances in Cryptology: Proceedings of EUROCRYPT'87, Lecture Notes in Computer Science, Vol. 304, Springer, Berlin, 1987, pp. 95–105.

[75] F. Hoornaert, M. Decroos, J. Vandewalle, R. Govaerts, Fast RSA-hardware: dream or reality, in: C.G. Günther (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'88, Lecture Notes in Computer Science, Vol. 330, Springer, Berlin, 1988, pp. 257–264.
[76] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, P.G.A. Jespers, A single chip 1024 bits RSA processor, in: J.-J. Quisquater, J. Vandewalle (Eds.), Advances in Cryptology: Proceedings of EUROCRYPT'89, Lecture Notes in Computer Science, Vol. 434, Springer, Berlin, 1989, pp. 219–236.
[77] P.A. Findlay, B.A. Johnson, Modular exponentiation using recursive sums of residues, in: G. Brassard (Ed.), Advances in Cryptology: Proceedings of CRYPTO'89, Lecture Notes in Computer Science, Vol. 435, Springer, Berlin, 1989, pp. 371–386.
[78] H. Morita, A fast modular-multiplication algorithm based on a higher radix, in: G. Brassard (Ed.), Advances in Cryptology: Proceedings of CRYPTO'89, Lecture Notes in Computer Science, Vol. 435, Springer, Berlin, 1989, pp. 387–399.
[79] H. Orup, E. Svendsen, E. Andreasen, VICTOR: an efficient RSA hardware implementation, in: I.B. Damgård (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'90, Lecture Notes in Computer Science, Vol. 473, Springer, Berlin, 1990, pp. 246–252.
[80] N. Takagi, A radix-4 modular multiplication hardware algorithm for modular exponentiation, IEEE Trans. Comput. 41 (8) (1992) 949–956.
[81] N. Takagi, S. Yajima, Modular multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem, IEEE Trans. Comput. 41 (7) (1992) 887–891.
[82] C.D. Walter, Fast modular multiplication by operand scaling, in: J. Feigenbaum (Ed.), Advances in Cryptology: Proceedings of CRYPTO'91, Lecture Notes in Computer Science, Vol. 576, Springer, Berlin, 1991, pp. 313–323.
[83] C.D. Walter, Fast modular multiplication using 2-power radix, Int. J. Comput. Math. 3 (1991) 21–28.
[84] C.D. Walter, Logarithmic speed modular multiplication, IEE Electron. Lett. 30 (17) (1994) 1397–1398.
[85] C.S. Wallace, A suggestion for a fast multiplier, IEEE Trans. Electron. Comput. EC-13 (1964) 14–17.
[86] C.D. Walter, Exponentiation using division chains, IEEE Trans. Comput. 47 (7) (1997) 757–765.
[87] J.-S. Chiang, J.-K. Chen, An efficient VLSI architecture for RSA public-key cryptosystem, in: Proceedings of the IEEE International Circuits and Systems (ISCAS'99), Vol. 1, IEEE, Silver Spring, MD, 1999, pp. 496–499.
[88] Ç.K. Koç, C.Y. Hung, Carry save adders for computing the product AB modulo N, Electron. Lett. 26 (13) (1990) 899–900.
[89] K.-S. Cho, J.-H. Ryu, J.-D. Cho, High-speed modular multiplication algorithm for RSA cryptosystem, in: Proceedings of IECON, 2001, pp. 479–483.
[90] A.D. Booth, A signed binary multiplication technique, Quart. J. Mech. Appl. 4 (2).
[91] K.C. Posch, R. Posch, Modulo reduction in residue number systems, IEEE Trans. Parallel and Distributed Systems 6 (5) (1998) 449–454.
[92] G. Alia, E. Martinelli, A VLSI modulo m multiplier, IEEE Trans. Comput. 40 (7) (1991) 873–878.
[93] J.C. Bajard, L.S. Didier, P. Kornerup, An RNS montgomery's modular multiplication, IEEE Trans. Comput. 19 (2) (1998) 167–178.
[94] S. Kawamura, M. Koike, F. Sano, A. Shimbo, Cox-Rower architecture for fast parallel montgomery multiplication, in: B. Preneel (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'00, Lecture Notes in Computer Science, Vol. 1807, Springer, Berlin, 2000, pp. 523–538.
[95] H. Nozaki, M. Motoyama, A. Shimbo, S. Kawamura, Implementation of RSA algorithm based on RNS montgomery multiplication, in: Ç.K. Koç, D. Naccache, C. Paar (Eds.), Proceedings of Workshop on Cryptograpic Hardware and Embedded Systems (CHES 2001), Lecture Notes in Computer Science, Vol. 2162, Springer, Berlin, Paris, France, 2001, pp. 372–385.
[96] J.-J. Quisquater, C. Couvreur, Fast decipherment algorithm for RSA public-key cryptosystem, Electron. Lett. 18 (1982) 905–907.
[97] ECMNET, The ECMNET project, http://www.loria.fr/zimmerma/records/ecmnet.html (May 2002).
[98] T. Takagi, Fast RSA-type cryptosystem modulo $p^k q$, in: H. Krawczyk (Ed.), Advances in Cryptology: Proceedings of CRYPTO'98, Lecture Notes in Computer Science, Vol. 1462, Springer, Berlin, 1998, pp. 318–326.

[99] J. Grosschädl, The Chinese remainder theorem and its application in a high-speed RSA crypto chip, in: Proceedings of the 16th Annual Computer Security Application Conference, IEEE Computer Society Press, New Orleans, Louisiana, USA, 2000, pp. 384–393.
[100] M. Joye, A.K. Lenstra, J.-J. Quisquater, Chinese remaindering based cryptosystem in the presence of faults, J. Cryptol. 4 (12) (1999) 241–245.
[101] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, J.-P. Seifert, Fault attacks on RSA with CRT: concrete results and practical countermeasures, in: B.S. Kaliski Jr., Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, Springer, Berlin, 2002.
[102] R.L. Rivest, RSA chips: Past/present/future, in: T. Beth, N. Cot, I. Ingemarsson (Eds.), Advances in Cryptology: Proceedings of EURO-CRYPT'84, Lecture Notes in Computer Science, Vol. 209, Springer, Berlin, 1984, pp. 159–168.
[103] Y.A. Lau, T.R. McPherson, Implementation of a hybrid RSA/DES management system, in: A. Gersho (Ed.), Advances in Cryptology: Proceedings of CRYPTO'81, IEEE, 1981, p. 83.
[104] M. Kochanski, Developing an RSA chip, in: H.C. Williams (Ed.), Advances in Cryptology: Proceedings of CRYPTO'85, Lecture Notes in Computer Science, Vol. 218, Springer, Berlin, 1985, pp. 350–357.
[105] G. Rankine, Thomas—a complete single chip RSA device, in: A.M. Odlyzko (Ed.), Advances in Cryptology: Proceedings of CRYPTO'86, Lecture Notes in Computer Science, Vol. 263, Springer, Berlin, 1986, pp. 481–487.
[106] M. Shand, J. Vuillemin, Fast implementations of RSA cryptography, in: Proceedings of the 11th IEEE Symposium on Computer Arithmetic, Windsor, Ont., Canada, 1993.
[107] A.F. Tenca, G. Todorov, Ç.K. Koç, High-radix design of a scalable modular multiplier, in: Ç.K. Koç, D. Naccache, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001), Lecture Notes in Computer Science, Vol. 2162, Springer, Berlin, 2001, pp. 189–205.
[108] E. Savaş, A.F. Tenca, Ç.K. Koç, A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$, in: C. Paar, Ç.K. Koç (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), Lecture Notes in Computer Science, Vol. 1965, Springer, Berlin, 2000, pp. 281–296.
[109] Atmel Corp., NIMBUS security chip, Presented at the 7th CACR Information Security Workshop on Smart Cards: Technology, Applications and Security, http://www.cacr.math.uwaterloo.ca/conferences.html (April 25 2001).
[110] Spyrus Inc., USB token, the reader-less smart card, Presented at the 7th CACR Information Security Workshop on Smart Cards: Technology, Applications and Security, http://www.cacr.math.uwaterloo.ca/conferences.html (April 25 2001).
[111] Infineon Technologies AG, SLE 66CUX640P: USB Security IC, St. Martin Strasse 53, D-81541 Munchen, Germany, http://www.infineon.com (August 2001).
[112] E. Hess, Aspects of public key cryptosystems in practice, Presentation on the 4th Workshop on Elliptic Curve Cryptography (ECC 2000), http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc00/slides.html (October 2–4 2000).
[113] Motorola Inc., MPC18x, MPC190: Security processors, http://e-www.motorola.com/index.html.
[114] Koninklijke Philips Electronics N.V., The 2nd generation SmartXA, http://www.semiconductors.philips.com/markets/identification/products.
[115] Certicom Corp., CHRYSALIS-ITS, http://www.certicom.com/products/chrysalis/chrysalis_feat.html.
[116] E.F. Brickell, A survey of hardware implementations of RSA, in: G. Brassard (Ed.), Advances in Cryptology: Proceedings of CRYPTO'89, Lecture Notes in Computer Science, Vol. 435, Springer, Berlin, 1989, pp. 368–370.
[117] Ç.K. Koç, RSA hardware implementation, Tech. Report, RSA Laboratories, RSA Data Security, Inc., Redwood City, CA (August 1995).
[118] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in: G.R. Blakley, D. Chaum (Eds.), Advances in Cryptology: Proceedings of CRYPTO'84, Lecture Notes in Computer Science, Vol. 196, Springer, Berlin, 1985, pp. 10–18.
[119] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inform. Theory 31 (1985) 469–472.
[120] A. Menezes, Y.-H. Wu, The discrete logarithm problem in GL(n, q), Ars Combin. 47 (1998) 23–32.

[121] E.D. Mastrovito, VLSI architectures for computations in Galois fields, Ph.D. thesis, Department of Electrical Engineering, Linkøping University, S-581 83 Linkøping, Sweden, 1991.
[122] J.K. Omura, J.L. Massey, Computation method and apparatus for finite field arithmetic, United States Patent, Patent Number: 4.587.627, May 6, 1986.
[123] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, Optimal normal bases in $GF(p^n)$, Discrete Appl. Math. 22 (1989) 149–161.
[124] M.G. Parker, M. Benaissa, Modular arithmetic using low order redundant bases, IEEE Trans. Comput. 46 (5) (1997) 611–616.
[125] B. Parhami, Generalized signed-digit number systems: A unified framework for redundant number representations, IEEE Trans. Comput. 39 (1) (1990) 89–98.
[126] H. Wu, M.A. Hasan, I.F. Blake, Highly regular architectures for finite field computation using redundant basis, in: Ç. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, Worcester, MA, USA, 1999, pp. 280–291.
[127] G. Drolet, A new representation of elements of finite fields $GF(2^m)$ yielding small complexity arithmetic circuits, IEEE Trans. Comput. 47 (9) (1998) 938–946.
[128] J.H. Silverman, Fast multiplication in finite field $GF(2^N)$, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, Worcester, MA, USA, 1999, pp. 123–134.
[129] B.A. Laws, C.K. Rushforth, A cellular-array multiplier for $GF(2^m)$, IEEE Trans. Comput. C-20 (1971) 1573–1578.
[130] S.K. Jain, L. Song, K.K. Parhi, Efficient semisystolic architectures for finite-field arithmetic, IEEE Trans. VLSI systems 6 (1) (1998) 101–113.
[131] C.-S. Yeh, I.S. Reed, T.K. Truong, Systolic multipliers for finite fields $GF(2^m)$, IEEE Trans. Comput. C-33 (4) (1984) 357–360.
[132] M.A. Hasan, V.K. Bhargava, Bit-serial systolic divider and multiplier for finite fields $GF(2^m)$, IEEE Trans. Comput. 41 (8) (1992) 972–980.
[133] M.A. Hasan, Look-up table-based large finite field multiplication in memory constrained cryptosystems, IEEE Trans. Comput. 49 (7) (2000) 749–758.
[134] M.A. Hasan, M. Wang, V.K. Bhargava, Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$, IEEE Trans. Comput. 41 (8) (1992) 962–971.
[135] C.-Y. Lee, E.-H. Lu, J.-Y. Lee, Bit-parallel systolic multipliers for $GF(2^m)$ fields defined by all-one and equally spaced polynomials, IEEE Trans. Comput. 50 (5) (2001) 385–393.
[136] C. Paar, A new architecture for a parallel finite field multiplier with low complexity based on composite fields, IEEE Trans. Comput. 45 (7) (1996) 856–860.
[137] C. Paar, Fast arithmetic architectures for public-key algorithms over galois fields $GF((2^n)^m)$, in: W. Fumy (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'97, Lecture Notes in Computer Science, Vol. 1233, Springer, Berlin, 1997, pp. 363–378.
[138] C. Paar, M. Rosner, Comparison of arithmetic architectures for Reed-Solomon decoders, in: K.L. Pocek, J. Arnold (Eds.), Proceedings of FCCM'97, Napa Valley, CA, 1997, pp. 219–225.
[139] S.-W. Wei, VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF(2^m)$, IEEE Transactions on Circuits and Systems II: Analog Digital Signal Process. 44 (10) (1997) 847–855.
[140] W.-C. Tsai, C. B. Shung, S.-J. Wang, A systolic architecture for elliptic curve cryptosystems, in: Proceedings of the ICSP2000, 2000, pp. 591–597.
[141] H. Wu, Low complexity bit-parallel finite field arithmetic using polynomial basis, in: Çetin Koç, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, Worcester, MA, USA, 1999, pp. 280–291.
[142] G. Orlando, C. Paar, A super-serial galois fields multiplier for FPGAs and its application to public-key algorithms, in: Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99), 1999, pp. 232–239.
[143] L. Song, K.K. Parhi, Low-complexity modified Mastrovito multipliers over finite fields $GF(2^M)$, in: Proceedings of the International Symposium on Circuits and Systems (ISCAS'99), IEEE, 1999, pp. 508–512.

[144] B. Sunar, Ç.K. Koç, Mastrovito multiplier for all trinomials, IEEE Trans. Comput. 48 (5) (1999) 522–527.
[145] A. Halbutoğulları, Ç.K. Koç, Mastrovito multiplier for general irreducible polynomials, IEEE Trans. Comput. 49 (5) (2000) 503–518.
[146] T. Zhang, K.K. Parhi, Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials, IEEE Trans. Comput. 50 (7) (2001) 734–748.
[147] M.A. Hasan, A. G. Wassal, VLSI algorithms, architectures, and implementation of a versatile $GF(2^m)$ processor, IEEE Trans. Comput. 49 (10) (2000) 1064–1072.
[148] H. Wu, Montgomery multiplier and squarer in $GF(2^m)$, Tech. Report CORR 2000-28, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2000.
[149] H. Wu, Montgomery multiplier and squarer for a class of finite fields, IEEE Trans. Comput. 51 (5) (2002) 521–529.
[150] L. Gao, K.K. Parhi, Custom VLSI design of efficient low latency and low power finite field multiplier for Reed-Solomon codec, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2001), Vol. 4, 2001, pp. 574–577.
[151] Ç.K. Koç, B. Sunar, Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields, IEEE Trans. Comput. 47 (1998) 353–356.
[152] Z. Yu, Low power finite field multiplication and division in re-configurable Reed-Solomon codes, in: Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002), 2002, pp. 785–788.
[153] C.A. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, I.S. Reed, VLSI architectures for computing multiplications and inverses in $GF(2^m)$, IEEE Trans. Comput. 34 (8) (1985) 709–717.
[154] A. Reyhani-Masoleh, M.A. Hasan, A new construction of Massey-Omura parallel multiplier over $GF(2^m)$, IEEE Trans. Comput. 51 (5) (2002) 511–520.
[155] C.C. Wang, Algorithm to design finite-field normal-basis multipliers, Tech. Report NPO-17109, NASA's Jet Propulsion Laboratory, Pasadena, CA, 1986.
[156] C.C. Wang, An algorithm to design finite field multipliers using a self-dual normal bases, IEEE Trans. Comput. 38 (10) (1989) 1457–1460.
[157] I.M. Onyszchuk, R.C. Mullin, S.A. Vanstone, Computational method and apparatus for finite field multiplication, United States Patent, Patent Number: 4.745.568, May 17, 1988.
[158] G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, An implementation for a fast public key cryptosystem, J. Cryptol. 3 (2) (1991) 63–79.
[159] G.B. Agnew, R.C. Mullin, S.A. Vanstone, On the development of a fast Elliptic Curve Cryptosystem, in: R.A. Rueppel (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'92, Lecture Notes in Computer Science, Springer, Berlin, Hungary, 1992, pp. 482–487.
[160] C.-C. Lu, A search of minimal key functions for normal basis multipliers, IEEE Trans. Comput. 46 (5) (1997) 588–592.
[161] R.C. Mullin, Multiple bit multiplier, United States Patent, Patent Number: 5,787,028, July 28, 1998.
[162] L. Gao, G.E. Sobelman, Improved VLSI designs for multiplication and inversion in $GF(2^m)$, in: Proceedings of the 13th Annual International ASIC/SOC Conference, Washington DC, 2000, pp. 97–101.
[163] S. Oh, C.H. Kim, J. Lim, D.H. Cheon, Efficient normal basis multipliers in composite fields, IEEE Trans. Comput. 49 (10) (2000) 1133–1138.
[164] B. Sunar, Ç.K. Koç, An efficient optimal normal basis type ii multiplier, IEEE Trans. Comput. 50 (1) (2001) 83–87.
[165] S. Sutikno, A. Surya, An architecture of $F_{2^{2N}}$ multiplier for elliptic curves cryptosystem, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000), Vol. 1, Geneva, Switzerland, 2000, pp. 279–282.
[166] A. Reyhani-Masoleh, M. A. Hasan, Efficient digit-serial normal basis multipliers over $GF(2^m)$, in: Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002), 2002, pp. 781–784.
[167] A. Reyhani-Masoleh, M. A. Hasan, Efficient multiplication beyond optimal normal bases, Tech. Report CORR 2002-12, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, 2002.
[168] M. Diab, A. Poli, New bit-serial systolic multiplier for $GF(2^m)$ using irreducible trinomials, Electron. Lett. 27 (13) (1991) 1183–1184.

[169] S.T.J. Fenn, M. Benaissa, D. Taylor, $GF(2^m)$ multiplication and division over the dual basis, IEEE Trans. Comput. 45 (3) (1996) 319–327.
[170] H. Wu, M.A. Hasan, Low complexity bit-parallel multipliers for a class of finite fields, IEEE Trans. Comput. 47 (8) (1998) 883–887.
[171] H. Wu, M.A. Hasan, I.F. Blake, New low-complexity bit-parallel finite field multipliers using weakly dual nases, IEEE Trans. Comput. 47 (11) (1998) 1223–1234.
[172] C.-H. Lee, J.-I. Lim, A new aspect of dual basis for efficient field arithmetic, in: H. Imai, Y. Zheng (Eds.), Proceedings of Second International Workshop on Practice and Theory in Public Key Cryptography (PKC 1999), Lecture Notes in Computer Science, Vol. 1560, Springer, Berlin, 1999, pp. 12–28.
[173] D. Gollman, Equally spaced polynomials, dual bases, and multiplication in F_{2^n}, IEEE Trans. Comput. 51 (5) (2002) 588–591.
[174] I.S. Hsu, T.K. Truong, L.J. Deutsch, I.S. Reed, A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases, IEEE Trans. Comput. 37 (6) (1988) 735–739.
[175] C. Paar, N. Lange, A comparative VLSI synthesis of finite field multipliers, in: Proceedings of the Third International Symposium on Communication Theory & Applications, Lake District, UK, 1995.
[176] G.C. Ahlquist, B.E. Nelson, M. Rice, Optimal finite field multipliers for FPGAs, in: P. Lysaght, J. Irvine, R.W. Hartenstein (Eds.), Proceedings of the Ninth International Workshop on Field-Programmable Logic and Applications (FPL), Lecture Notes in Computer Science, Vol. 1673, Springer, Berlin, 1999, pp. 51–60.
[177] A.J. Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, Dordrecht, 1993.
[178] E. De Win, B. Preneel, Elliptic curve public-key cryptosystems—an introduction, in: B. Preneel, V. Rijmen (Eds.), State of the Art in Applied Cryptography, Lecture Notes in Computer Science, Vol. 1528, Springer, Berlin, 1997, pp. 132–142.
[179] N. Koblitz, A. Menezes, S. Vanstone, The state of elliptic curve cryptography, Des. Codes Cryptogr. 19 (2000) 173–193.
[180] IEEE P1363, Standard specifications for public key cryptography, 1999.
[181] A. Miyaji, T. Ono, H. Cohen, Efficient elliptic curve exponentiation, in: Y. Han, T. Okamoto, S. Qing (Eds.), Proceedings of ICICS'97, Lecture Notes in Computer Science, Vol. 1334, Springer, Berlin, 1997, pp. 282–290.
[182] J. Guajardo, C. Paar, Efficient algorithms for elliptic curve cryptosystems, in: B.S. Kaliski Jr. (Ed.), Advances in Cryptology: Proceedings of CRYPTO'97, Lecture Notes in Computer Science, Vol. 1294, Springer, Berlin, 1997, pp. 342–356.
[183] H. Cohen, A. Miyaji, T. Ono, Efficient elliptic curve exponentiation using mixed coordinates, in: K. Ohta, D. Pei (Eds.), Proceedings of ASIACRYPT 1998, Lecture Notes in Computer Science, Vol. 1514, Springer, Berlin, 1998, pp. 51–65.
[184] F. Morain, J. Olivos, Speeding up the computations on an elliptic curve using addition-subtraction chains, Inform. Théor. Appl. 24 (1990) 531–544.
[185] J. Solinas, An improved algorithm for arithmetic on a family of elliptic curves, in: B. Kaliski Jr. (Ed.), Advances in Cryptology: Proceedings of CRYPTO'97, Lecture Notes in Computer Science, Vol. 1294, Springer, Berlin, 1997, pp. 357–371.
[186] C. Paar, Implementation options for finite field arithmetic for elliptic curve cryptosystems, Presented at the Third workshop on Elliptic Curve Cryptography (ECC 1999), http://www.cacr.math.uwaterloo.ca/conferences/1999/ecc99/slides.html (November 1–3, 1999).
[187] Certicom Corp., The elliptic curve cryptosystem for smart cards, The seventh in a series of ECC white papers, http://www.certicom.com.
[188] J. Grosschädl, A bit-serial unified multiplier architecture for finite fields $GF(p)$ and $GF(2^n)$, in: Ç.K. Koç, D. Naccache, C. Paar (Eds.), Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001), Lecture Notes in Computer Science, Vol. 2162, 2001, pp. 206–223.
[189] B. Kaliski Jr., The Montgomery inverse and its applications, IEEE Trans. Comput. 44 (8) (1995) 1064–1065.
[190] E. Savaş, Ç.K. Koç, The Montgomery modular inverse revisited, IEEE Trans. Comput. 49 (7) (2000) 763–766.

[191] G. Orlando, C. Paar, A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware, in: Ç.K. Koç, D. Naccache, C. Paar (Eds.), Proceedings of Workshop on Cryptograpic Hardware and Embedded Systems (CHES 2001), Lecture Notes in Computer Science, Vol. 2162, Springer, Berlin, Paris, France, 2001, pp. 356–371.
[192] P. Gaudry, F. Hess, N.P. Smart, Constructive and destructive facets of Weil descent on elliptic curves, J. Cryptol. 15 (2002) 19–46.
[193] S.D. Galbraith, F. Hess, N.P. Smart, Extending the GHS weil descent attack, in: L. Knudsen (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'02, Lecture Notes in Computer Science, Vol. 2332, Springer, Berlin, 2002, pp. 29–44.
[194] N.P. Smart, How secure are elliptic curves over composite extension fields?, in: B. Pfitzmann (Ed.), Advances in Cryptology: Proceedings of EUROCRYPT'01, Lecture Notes in Computer Science, Vol. 2045, 2001, pp. 30–39.
[195] G. Frey, How to disguise an elliptic curve, Talk at Waterloo workshop on the ECDLP, http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html (1998).
[196] H. Wu, On complexity of squaring using polynomial basis in $GF(2^m)$, Tech. Report CORR 2000-28, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2000.
[197] G. Orlando, C. Paar, Squaring architecture for $GF(2^m)$ and its applications in cryptographic systems, Electron. Lett. 36 (13) (2000) 1116–1117.
[198] G.-L. Feng, A VLSI architecture for fast inversion in $GF(2^m)$, IEEE Trans. Comput. 38 (10) (1989) 1383–1386.
[199] S.T.J. Fenn, M. Benaissa, D. Taylor, Fast normal basis inversion in $GF(2^m)$, Electron. Lett. 32 (17) (1996) 1566–1567.
[200] I.J. Calvo, M. Torres, Complexity of the inversion in $GF(2^m)$, Electron. Lett. 33 (3) (1997) 194–195.
[201] N. Takagi, J. Yoshiki, K. Takagi, A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis, IEEE Trans. Comput. 50 (5) (2001) 394–398.
[202] J.-H. Guo, C.-L. Wang, Systolic array implementation of Euclid's algorithm for inversion and division in $GF(2^m)$, IEEE Trans. Comput. 47 (10) (1998) 1161–1167.
[203] C.-L. Wang, J.-H. Guo, New systolic arrays for $C + AB^2$, inversion, and division in $GF(2^m)$, IEEE Trans. Comput. 49 (10) (2000) 1120–1125.
[204] T. Itoh, S. Tsujii, Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$, Electron. Lett. 24 (6) (1988) 334–335.
[205] Y. Asano, T. Itoh, S. Tsujii, Generalized fast algorithm for computing multiplicative inverses in $GF(2^m)$, Electron. Lett. 25 (10) (1989) 664–665.
[206] H. Brunner, A. Curiger, M. Hofstetter, On computing multiplicative inverses in $GF(2^m)$, IEEE Trans. Comput. 42 (8) (1993) 1010–1014.
[207] Z. Yan, D.V. Sarwate, Systolic architectures for finite field inversion and division, in: Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002), 2002, pp. 789–792.
[208] G.B. Agnew, R.C. Mullin, S.A. Vanstone, A fast elliptic curve cryptosystem, in: J.-J. Quisquater, J. Vandewalle (Eds.), Advances in Cryptology: Proceedings of EUROCRYPT'89, Lecture Notes in Computer Science, Vol. 434, Springer, Berlin, 1989, pp. 706–708.
[209] G.B. Agnew, R.C. Mullin, S.A. Vanstone, An implementation of elliptic curve cryptosystem over $F_{2^{155}}$, IEEE J. Selected Areas Commun. 11 (5) (1993) 804–813.
[210] S. Sutikno, R. Effendi, A. Surya, Design and implementation of arithmetic processor $GF(2^{155})$ for elliptic curve cryptosystems, in: Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'98), 1998, pp. 647–650.
[211] G.B. Agnew, T. Beth, R.C. Mullin, S.A. Vanstone, Arithmetic operations in $GF(2^m)$, J. Cryptol. 6 (1993) 3–13.
[212] S. Sutikno, A. Surya, R. Effendi, An implementation of elgamal elliptic curves cryptosystem, in: Proceedings of the 1998 IEEE Asia Pacific Conference on Circuits and Systems, Chiangmai, Thailand, 1998, pp. 483–486.
[213] L. Gao, S. Shrivastava, H. Lee, G.E. Sobelman, A compact fast variable key size elliptic curve cryptosystem coprocessor, in: Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 1999, pp. 304–305.
[214] L. Gao, S. Shrivastava, G.E. Sobelman, Elliptic curve scalar multiplier design using FPGAs, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of the First International Workshop on Cryptographic Hardware and Embedded

Systems (CHES 1999), Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin, Worcester, MA, USA, 1999, pp. 257–305.
[215] O. Hauck, A. Katoch, S.A. Huss, VLSI system design using asynchronous wave pipelines: a 0.35 μm CMOS 1.5 GHz elliptic curve public key cryptosystem chip, in: Proceedings of Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000), IEEE, 2000, pp. 188–196.
[216] K.H. Leung, K.W. Ma, W.K. Wong, P.H.W. Leong, FPGA implementation of a microcoded elliptic curve cryptographic processor, in: Proceedings of Field-Programmable Custom Computing Machines (FCCM'00), 2000, pp. 68–76.
[217] M. Ernst, S. Klupsch, O. Hauck, S.A. Huss, Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems, in: Proceedings of 12th IEEE Workshop on Rapid System Prototyping, Monterey, CA, 2001, pp. 24–31.
[218] G. Orlando, C. Paar, A high-performance reconfigurable elliptic curve processor for $GF(2^m)$, in: Ç.K. Koç, C. Paar (Eds.), Proceedings of Workshop on Cryptograpic Hardware and Embedded Systems (CHES 2000), Lecture Notes in Computer Science, Vol. 1965, Springer, Berlin, 2000, pp. 41–56.
[219] J. Goodman, A.P. Chandrakasan, An energy-efficient reconfigurable public-key cryptography processor, IEEE J. Solid-State Circuits 36 (11) (2001) 1808–1820.
[220] S. Janssens, J. Thomas, W. Borremans, P. Gijsels, I. Verbauwhede, F. Vercauteren, B. Preneel, J. Vandewalle, Hardware/software co-design of an elliptic curve public-key cryptosystem, in: Proceedings IEEE Workshop on of Signal Processing Systems, 2001, pp. 209–216.
[221] P. Schaumont, I. Verbauwhede, K. Keutzer, M. Sarrafzadeh, A quick safari through the reconfiguration jungle, in: Proceedings of 38th Design Automation Conference (DAC'01), Las Vegas, NV, USA, 2001, pp. 172–177.
[222] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, J. Shokrollahi, Tradeoff analysis of FPGA based Elliptic Curve Cryptosystems, in: Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002), 2002, pp. 797–800.
[223] J.-H. Kim, D.-H. Lee, A compact finite field processor over $GF(2^m)$ for Elliptic Curve Cryptography, in: Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002), 2002, pp. 340–343.

Lejla Batina received the M.Sc. degree in Mathematics from the University of Zagreb, Croatia in 1995 and the MTD degree (Master of Technological Design) from the Technical University of Eindhoven, The Netherlands in 2001 (Mathematics for Industry—Postmasters study). She has held research positions in the Department of Mathematics at the Eindhoven University and the Institute for Experimental Mathematics in Essen, Germany. She is currently working as cryptographer at SafeNet, Vught in The Netherlands. She is also a Ph.D. student in the COSIC group of the Department of Electrical Engineering (ESAT) of the K.U. Leuven, Belgium. Her research interests include elliptic curve cryptography (hardware implementations) and side-channel security.

Sıddıka Berna Örs was born in Ankara, Turkey, in 1973. She received the B.Sc. degree in Electronics and Communications Engineering from Istanbul Technical University, Electrical and Electronic Engineering Faculty, Turkey in 1995 and the M.Sc degree in Electronics and Communications Engineering from Istanbul Technical University, Institute of Science and Technology, Turkey in 1998. She worked as a research and teaching assistant at Istanbul Technical University, Electrical and Electronics Faculty between 1995–1999. She is currently a Ph.D. student and researcher in the COSIC group of the Department of Electrical Engineering (ESAT) of K.U. Leuven, Belgium. Her research interests include hardware implementations for elliptic curve cryptography.

Bart Preneel received the Electrical Engineering degree and the Doctorate in Applied Sciences in 1987 and 1993, respectively, both from the Katholieke Universiteit Leuven (Belgium). He is currently a professor at the Katholieke Universiteit Leuven and a visiting professor (Professor II) at the University of Bergen in Norway and the University of Ghent in Belgium. During the academic year 1993–1994, he was a research fellow of the EECS Department of the University of California at Berkeley. His main research interests are cryptography, computer security and network security. Bart Preneel has authored and co-authored more than 70 scientific publications, and is inventor of one patent. He is a member of the Board of Directions of the IACR (International Association for Cryptologic Research) and of the Editorial Board of the Journal of Cryptology and of Cryptologia. He was program chair of the 1994 workshop on Fast Software Encryption, of the IFIP conference CMS'99 (Communications and Multimedia Security), and of Eurocrypt 2000. He is currently project manager of NESSIE (New European Schemes for Signature, Integrity and Encryption).

Joos Vandewalle obtained the degree of M. Sc. in Electrical Engineering, a doctoral degree and a special doctoral degree respectively in 1971, 1976 and 1984 at the K.U. Leuven. He was a postdoctoral researcher during 1976–1978 and visiting assistant Professor during 1978–1979 at the Electrical Engineering and Computer Science Department of the University of California, Berkeley. Since 1979 he is appointed at the Electrical Engineering Department of the K.U. Leuven, where he is Full Professor since 1986. He is a Fellow of IEEE and is currently Vice-President for Region 8 of the IEEE Society on Circuits and Systems. His main research interests are in system theory and its applications in circuit theory, signal processing, cryptography and neural networks. He is currently the head of the Department of Electrical Engineering ESAT, and the research group ESAT/SISTA-COSIC. He is one of the three coordinators of the Interdisciplinary Center for Neural Networks (ICNN), which is regrouping some 50 researchers at the K.U. Leuven. In 1992 he held the Francqui chair on neural networks at the Université de Liège.