

Less Power Security Techniques for Sensor Networks

Junaid Majeed*

* Lecturer Hamdard University junaid.majeed@hamdard.edu.pk

Abstract

Wireless sensor networks have a wide spectrum of civil and military applications that call for security, e.g., target surveillance in hostile environments. Wireless sensor networks consist of many inexpensive wireless nodes, each having sensing capability with some computational and communication power. Due to limited computation, power, and storage resources available on sensor nodes asymmetric cryptographic algorithms cannot provide security on wireless sensor networks. Therefore, we have proposed a security protocol for wireless sensor networks, which does not consume much energy compared to conventional security protocols. In addition, when communication is wireless, it is impossible to use the location of a user's network connection as an element in validating their identity. To exploit the potential of the technology, people must be able to roam freely with their portable communications products and, from the standpoint of the network infrastructure at least, to pop up unexpectedly in new places. While this characteristic provides the user with new freedom, it confronts the service provider and the system administrator with unprecedented security challenges.

In this paper, we describe generic and scalable architecture of finite field coprocessors, which is implemented within the latest family of Field Programmable System Level Integrated Circuits FPSLIC from Atmel, Inc. The HW architectures are adapted from Karatsuba's divide and conquer algorithm and allow for a reasonable speedup of the top-level elliptic curve algorithms. The VHDL hardware models are automatically generated based on an eligible operand size, which permits the optimal utilization of a particular FPSLIC device.

1 Introduction

Wireless sensor networks combine the characteristic of ad hoc mobile wireless networks (ad hoc networks in short) at the system level, with the characteristics of sensors at the component level. Ad hoc networks are: (1) ad hoc: the network set-up is possibly short-lived; (2) mobile: the nodes are not attached to any fixed communications infrastructure as well as fixed energy supply; (3) wireless: the nodes communicate wirelessly. These three properties imply a series of constraints, that together with the constraints imposed by sensors (among which energy being the predominant), form the starting point of our security research. Energy efficiency is the utmost concern for wireless sensor networks (WSNs). Every algorithm to be implemented on these wireless sensors has to be lean in terms of energy and computational resource requirements. Cryptographic algorithms are an indispensable part of the security architecture of WSNs, and they tend to have high invocation rates, hence their energy efficiency makes a large impact on the system as a whole. Unfortunately, there is currently no off-the shelf cryptographic library that readily satisfies the specific needs of WSNs. For example, OpenSSL (www.openssl.org) is optimized for mainstream computing platforms but is narrow in implementation scope. Other libraries such as Crypto++ (www.eskimo.com/~weidai/cryptlib.html) and Botan (botan.randombit.net) use C++ instead of the less resource-demanding C language.

In the near future, the wireless sensor networks are anticipated to consist of thousands of inexpensive nodes, each having sensing

capability with limited computational and communication power [1]. Sensor nodes with inbuilt chips and software for processing encryptions are provided in [3], but the limitation on processing power still exists. Such sensor networks are expected to be widely deployed in a vast variety of environments for commercial, civil, and military applications including surveillance, vehicle tracking, climate and habitat monitoring, intelligence, medical, and acoustic data gathering. Although, many of these application areas require a certain level of information security, unfortunately little research on security in sensor networks has been reported so far. Since wireless sensor nodes have strict power constraints, providing security to wireless sensor networks is harder than conventional network security applications. For example, asymmetric cryptographic algorithms are not applicable in wireless sensor networks due to the limited computation, power, and storage resources available on sensor nodes [2].

Many compelling applications like distributed information gathering and distributed micro sensing in radiology, military, and manufacturing drive the research in sensor networks. Typically, sensor networks comprise of a large set of distributed low power sensors scattered over the area to be monitored. The sensors have the ability to gather data, and process and forward it to a central node for further processing. A major challenge for the sensor networks is the limitations on the sensor hardware. Contemporary wireless sensors have limited battery, computation, and memory capacity. Such resource constrained environment has motivated extensive research that addresses energy-aware hardware and software design issues [4, 5]. Much effort has been on the energy-efficient communication protocols [5, 6, 7]. The comparative progress in making these networks secure has been insignificant. This is despite the fact that in certain applications of sensor networks, like military applications, security becomes important. The energy-constrained nature of the sensor networks makes the problem of incorporating security very challenging. Many well-known security mechanisms introduce significant computational/memory-wise overhead. The design of the security protocols for sensor networks should be geared towards conservation of the sensor resources. The level of security versus the consumption of energy, computation, and memory resources constitute a major design trade-off.

The trend to ubiquitous computing goes further from mobile communication and personal computing level to low-power autonomous devices. Here are a few examples of this trend. Piconet [8] is a general-purpose, low-power ad hoc radio network. It can connect a full range of portable and embedded sensing and computing objects. RFIDs are being used to replace bar codes on goods and to track inventory [11]. "Smart Dust" motes [9] are tiny autonomous nodes containing sensors, transceivers and a power source, and they have limited computing power. Common to all these devices is that they communicate wirelessly and their energy source is extremely limited. Batteries for these devices are tiny and can supply 10 μ W for only one day [9]. Moreover, some of these technologies collect energy from environmental sources, such as light, heat, noise, or vibration. Devices that harvest power from environmental sources are commonly referred to as *power scavengers*, and autonomous nodes that use scavengers are called *self-powered*. Newer scavengers are based on microelectromechanical systems (MEMS). They can be integrated on the chip and therefore reduce the cost and size. The scavenger shown in [10] produces around 8 μ W of energy relying solely on ambient vibration.

Since power consumption is the vital issue in many applications of these technologies, highly complex cryptographic schemes such as public key cryptography are seldom feasible. On low-end computing platforms where processing speed and communication bandwidth are critical, digital signatures may not be the best available choice. A good example of security architecture on existing mobile digital cellular networks is the GSM (Global System for Mobile) which was the first digital cellular system to provide security services, e.g., user authentication, message confidentiality, and key distribution [15]. However, some of the security algorithms of the GSM architecture have recently been cryptanalyzed [16, 12]. Here, we introduce a new secure technique for wireless mobile communication systems.

The protocol is based on elliptic curve cryptography. ECC allows using *one device and one key-pair per person* for the entire application. A very fine granular control is possible and in contrast to present systems, which are mostly based on symmetric ciphers, there is no problem regarding the key handling. Depending on the application, the performance of genuine SW implementations of ECC is not sufficient. In this paper generic and scalable architecture of *Finite Field* coprocessors for the acceleration of ECC is presented. This is a finite field coprocessor (FFCP), implementing field multiplication, addition and squaring completely within HW. The multi-segment Karatsuba multiplication together with a cleverly selected sequence of intermediate result computations permits high-speed ECC even on devices offering only approx. 40K system gates of HW resources. Recently, Atmel, Inc. introduced AT94K family of FPSLIC devices (Field Programmable System Level Integrated Circuits). This architecture integrates FPGA resources, an AVR microcontroller core, several peripherals and SRAM within a single chip. Based on HW/SW co-design methodologies, this architecture is perfectly suited for System on Chip (SoC) implementations of ECC.

2 Background

2.1 Elliptic Curve Cryptography

The protocol described in this paper depends on the security of the so-called elliptic curve primitives, e.g., key generation, signature generation, and signature verification. These operations utilize the arithmetic of points which are elements of the set of solutions of an elliptic curve equation defined over a finite field. The security of the protocol depends on the intractability of the elliptic curve analogue of the discrete logarithm problem which is a well known and extensively studied computationally hard problem.

We first define the nomenclature and then provide a general overview of these primitives. The mathematical background of the elliptic curve cryptography can be found in [13, 14].

2.1.1 Basic Definitions and Notation

- **Scalar:** An element belonging to either one of the fields $GF(p)$ or $GF(2^k)$ is called a scalar. Scalars are named with lowercase letters: r, s, t , etc.
- **Scalar Addition:** Two or more scalars can be added to obtain another scalar. In the $GF(p)$ case, this is the ordinary integer addition modulo p . When $GF(2^k)$ is used, this is equivalent to polynomial addition modulo an irreducible polynomial of degree k , generating the field $GF(2^k)$. We will denote the scalar addition of r and s giving the result e by $e = r + s$.
- **Scalar Multiplication:** Two or more scalars can be multiplied to obtain another scalar. In the $GF(p)$ case, this is the ordinary integer multiplication modulo p . When $GF(2^k)$ is used, this is equivalent to polynomial multiplication modulo an irreducible polynomial of degree k , generating the field $GF(2^k)$. We will denote the scalar multiplication of r and s giving the result e by $e = r.s$.
- **Scalar Inversion:** The multiplicative inverse of an element a in $GF(p)$ or $GF(2^k)$ is denoted as a^{-1} which is the number with the

property $a \cdot a^{-1} = 1$. It is often computed using the Fermat's method or the extended Euclidean algorithm.

- **Point:** An ordered pair of scalars satisfying the elliptic curve equation is called a point. Capital letters are used to denote these elements: P, Q , etc. We will also denote a point P using its coordinates $P = (x, y)$, where x and y belong to the field. Furthermore, the x and y coordinates of P will be denoted by $P.x$ or $P.y$, respectively.
- **Point Addition:** There is a method to obtain a third point R on the curve given two points P and Q , using a set of rules. This is called an elliptic curve point addition. We will use the symbol '+' to denote the elliptic curve addition $R = P + Q$. This should not be confused with scalar addition.
- **Elliptic Curve Group:** The set of the solutions of the elliptic curve equation together with a special point called point-at-infinity form an additive group if the point addition operation defined above is taken as the group operation.
- **Point Multiplication:** The multiplication of an elliptic curve point P by an integer e will be denoted by $e \times P$. It is equivalent to adding P to itself e times, which yields another point on the curve.

In addition to the above elliptic curve cryptographic primitives, we need a secret key cryptographic algorithm and a one-way hash (message digest) function which are defined below:

- **Secret Key Algorithm:** A secret-key encryption algorithm is used to encrypt (hide) the data in the protocol. A conventional stream cipher (RC4 or SEAL) or a block cipher (DES, 3DES, IDEA) in the cipher-block-chaining mode can be used. The encryption and decryption operations using the key K acting on the plaintext M and the cipher text C are denoted as $C := E(K, M)$ and $M := D(K, C)$, respectively.
- **Message Digest Function:** A message digest function compresses a long message into a short value which is usually 128 or 160 bits long. Two widely used and standardized hash functions are MD5 and SHA. We will denote the message digest of a message M by $H(M)$. The signature functions take $H(M)$ as an input for efficiency reasons. The hash of the concatenation of two messages $M1$ and $M2$ is denoted as $H(M1, M2)$.

2.2 Elliptic Curve Digital Signature Algorithm

First, an elliptic curve E defined over $GF(p)$ or $GF(2^k)$ with large group of order n and a point P of large order is selected and made public to all users. Then, the following key generation primitive is used by each party to generate the individual public and private key pairs. Furthermore, for each transaction the signature and verification primitives are used. We briefly outline the Elliptic Curve Digital Signature Algorithm (ECDSA) below, details of which can be found in [26].

ECDSA Key Generation The user A follows these steps:

1. Select a random integer $d \in [2, n - 2]$.
2. Compute $Q = d \times P$.
3. The public and private key of the user A are (E, P, n, Q) and d , respectively.

ECDSA Signature Generation The user A signs the message m using the following steps.

1. Select a random integer $k \in [2, n - 2]$.
2. Compute $k \times P = (x_1, y_1)$ and $r = x_1 \bmod n$.
If $x_1 \in GF(2^k)$, then it is assumed that x_1 is represented as a binary number.
If $r = 0$ then go to Step 1.
3. Compute $k^{-1} \bmod n$.

4. Compute $s = k^{-1}(H(m) + d \cdot r) \bmod n$.
Here H is the secure hash algorithm SHA.
- If $s = 0$ go to Step 1.
5. The signature for the message m is the pair of integers (r, s).

ECDSA Signature Verification The user B verifies A's signature (r, s) on the message m by applying the following steps:

1. Compute $c = s^{-1} \bmod n$ and $H(m)$.
2. Compute $u_1 = H(m) \cdot c \bmod n$ and $u_2 = r \cdot c \bmod n$.
3. Compute $u_1 \times P + u_2 \times Q = (x_0, y_0)$ and $v = x_0 \bmod n$.
4. Accept the signature if $v = r$.

2.3 Finite Field Arithmetic

An elliptic curve over $GF(2^k)$ is defined as the cubic equation

$$E: y^2 + xy = x^3 + ax^2 + b \quad (1)$$

with $a, b, x, y \in GF(2^k)$ and $b \neq 0$. The set of solutions $\{(x, y) \mid y^2 + xy = x^3 + ax^2 + b\}$ is called the points of the elliptic curve E. By defining an appropriate addition operation and an extra point O, called the point at infinity, these points become an additive, abelian group with O the neutral element. The EC point multiplication is computed by repeated point additions such as

$$\underbrace{P + P + \dots + P}_{k \text{ times}} = k \cdot P = R$$

As previously mentioned, the EC arithmetic is based on a Finite Field of characteristic 2 and extension degree $k:GF(2^k)$, which can be viewed as a vector space of dimension k over the field $GF(2)$. There are several bases known for $GF(2^k)$. The most common bases, which are also permitted by the leading standards concerning ECC (IEEE 1363 [21] and ANSI X9.62 [22]) are *polynomial bases* and *normal bases*. The representation treated in this paper is a polynomial basis, where field elements are represented by binary polynomials modulo an irreducible binary polynomial (called reduction polynomial) of degree k . Given an irreducible polynomial $P(x) = x^k + \sum_{i=0}^{k-1} p_i x^i$, with $p_i \in GF(2)$; an element $A \in GF(2^k)$ is represented by a bit string $(a_{n-1}, \dots, a_2, a_1, a_0)$, so that

$$A(x) = \sum_{i=0}^{k-1} a_i x^i = a_{k-1} x^{k-1} + \dots + a_2 x^2 + a_1 x + a_0$$

is a polynomial in k of degree less than k with coefficients $a_i \in GF(2)$. By exploiting a field of characteristic 2, the addition is reduced to just XOR-ing the corresponding bits. The sum of two elements $A, B \in GF(2^k)$ is given by

$$C(x) = A(x) \oplus B(x) = \sum_{i=0}^{k-1} (a_i \oplus b_i) x^i \quad (2)$$

and therefore takes a total of k binary XOR operations. The multiplication of two elements $A, B \in GF(2^k)$ is equivalent to the product of the corresponding polynomials:

$$C(x) = A(x)B(x) = \sum_{i=0}^{2k-2} c_i x^i \text{ denoting } c_k = \sum_{i=0}^k a_i b_{k-i} \text{ for } 0 \leq k \leq 2n-2 \quad (3)$$

With $a_i = 0$ and $b_i = 0$ for $i \geq n$. At the bit level the multiplication in $GF(2)$ is performed with Boolean AND operation.

Squaring is a special case of multiplication. For $A \in GF(2^k)$ the square is given by:

$$A^2(x) = \sum_{i=0}^{k-1} a_i x^{2i} \quad (4)$$

In the case of multiplication and squaring a polynomial reduction step has to be performed.

Karatsuba Multiplication. In 1963 A. Karatsuba and Y. Ofman discovered that multiplication of two k bit numbers can be done with a bit complexity of less than $O(k^2)$ using an algorithm now known as Karatsuba multiplication [23]. For multiplication in $GF(2^k)$ the Karatsuba multiplication scheme can be applied as well. Therefore, a polynomial $A \in GF(2^k)$ is subdivided into two segments and expressed as $A = A_1 x^{k/2} \oplus A_0$

For polynomials $A, B \in GF(2^k)$ the k -bit multiplication $C = A \cdot B$ is subdivided into $k/2$ -bit multiplications as follows:

$$\begin{aligned} C &= A \cdot B \\ &= (A_1 x^{k/2} \oplus A_0) \cdot (B_1 x^{k/2} \oplus B_0) \\ &= A_1 \cdot B_1 x^k \oplus (A_1 \cdot B_0 \oplus A_0 \cdot B_1) x^{k/2} \oplus A_0 \cdot B_0 \end{aligned}$$

By defining some additional polynomials

$$\begin{aligned} T_1 &= A_1 \cdot B_1 \\ T_2 &= (A_1 \oplus A_0) \cdot (B_1 \oplus B_0) = A_1 B_0 \oplus A_0 B_1 \oplus A_1 B_1 \oplus A_0 B_0 \\ T_3 &= A_0 \cdot B_0 \end{aligned}$$

one gets

$$A \cdot B = T_1 x^k \oplus (T_2 \oplus T_1 \oplus T_3) x^{k/2} \oplus T_3 \quad (5)$$

This results in a total of three $k/2$ -bit multiplications and some extra additions (XOR operations) to perform one k -bit multiplication.

Multi-Segment Karatsuba Multiplication. The fundamental Karatsuba multiplication for polynomials in $GF(2^k)$ is based on the idea of *divide and conquer*, since the operands are divided into two segments. One may attempt to generalize this idea by subdividing the operands into more than two segments. [24] reports on such an implementation with a fixed number of three segments denoted as *Karatsuba-variant multiplication*. The Multi-Segment Karatsuba (MSK) multiplication scheme, which is detailed subsequently, is more general because an arbitrary number of segments is supported. Disregarding some slight arithmetic variations, the Karatsuba-variant multiplication is a special case of the MSK approach.

Two polynomials in $GF(2^k)$ are multiplied by a n -segment Karatsuba multiplication (MSK_n) in the following way: It is assumed that $k \bmod n = 0$; if not, the polynomials are padded with the necessary number of zero coefficients. A polynomial $A \in GF(2^k)$ is divided into n segments

such that $A = \bigoplus_{i=0}^{n-1} A_i \cdot \hat{x}^i$, with $\hat{x} = x^{k/n}$. With Eqn. 6 holds

$C = A \cdot B = MSK_n(A, B)$ for any polynomials $A, B \in GF(2^k)$:

$$MSK_n(A, B) = \left(\bigoplus_{i=1}^n S_{i,0}(A, B) \hat{x}^{i-1} \right) \oplus \left(\bigoplus_{i=1}^{n-1} S_{n-i,i}(A, B) \hat{x}^{i-1+n} \right) \quad (6)$$

with

$$S_{m,l}(A, B) = \left(\bigoplus_{i=1}^{m-1} S_{i,l}(A, B) \right) \oplus \left(\bigoplus_{i=1}^{m-1} S_{i,l+m-i}(A, B) \right) + M_{m,l}(A, B) \quad (7)$$

$$S_{1,l}(A, B) = M_{1,l}(A, B) \text{ and } M_{m,l}(A, B) = \left(\bigoplus_{i=1}^{l+m-1} A_i \right) \cdot \left(\bigoplus_{i=1}^{l+m-1} B_i \right)$$

According to Eqn. 6 the entire product $C = A \cdot B = MSK_n(A, B)$ is composed of the partial sums $S_{m,l}(A, B)$. Each partial sum consists of partial products $M_{m,l}(A, B)$ according to Eqn. 7. The total number $N(n)$ of required k/n -bit multiplications in order to perform one k -bit multiplication using the MSK_n scheme is given by

$$N(n) = \sum_{i=1}^n i = \frac{(n+1) \cdot n}{2} \quad (8)$$

Polynomial Reduction. For $A, B \in GF(2^k)$, the maximum degree of the resulting polynomial $C(x) = A(x) \cdot B(x)$ is $2k-2$. Therefore, in order

In the data-path, these patterns are computed by some additional combinational logic.

Second, the resulting patterns are ordered by descending i of their factor \hat{x}^i . In this way, the product can be accumulated easily in a shift register.

As the third optimization criterion the remaining degree of freedom is taken advantage of in the following way: The patterns are once more partially reordered, such that when iterating over them from top to bottom, one of the two following conditions holds: Either the current pattern is constructed out of a single segment product (e.g. $A_4 \otimes B_4$), or the set of indices of the patterns segments differs only by one index from its predecessor (as in the partial products $(A_0 \oplus A_1) \cdot (B_0 \oplus B_1)$ and $(A_0 \oplus A_1 \oplus A_2) \otimes (B_0 \oplus B_1 \oplus B_2)$). In Fig. 3.1b this criterion is met for all but one iteration step (namely it is not met for the step from “23” to “1234”). Thus, based on the data-path in Fig. 3.2 the computation of the partial product “1234” takes a total of two clock cycles, which is one more compared to all other iteration steps. The number of additional clock cycles due to the fact that this third criterion can not be met increases slowly with the number of segments n .

The complete data path is depicted in Fig. 3.2. In part a) the two operand registers of width $l=k \cdot m$ are shown as well as their partitioning into five segments. Both are implemented as shift-registers in order to allow data exchange with the external controller.

The multiplexers in part b) select one from the five segments of the operands. They can both be controlled by the same set of signals, since they are always operating synchronously. Besides the combinational addition and squaring blocks, part c) illustrates the two accumulator registers. Both can either be loaded with a new segment, or they can accumulate intermediate segment sums. Section d) of Fig. 3.2. Part e) covers the pattern generation stage, which is mainly composed of multiplexers. Finally, in part f) the multiplication accumulator register is shown. It can either hold its value or the current pattern can be added to it in each cycle. Each time the intermediate result is shifted left by m bit, an interleaved reduction step according to Eqn. 10 is performed. This way, the accumulator needs only to be n bits wide, where n is the degree of the reduction polynomial. Furthermore, the necessary number of logic elements for the reduction step is minimized and no additional clock cycle is needed.

4 Atmel FPSLIC Hardware Platform

For the implementation of the previously detailed FF coprocessors the AT94K FPSLIC hardware platform from Atmel, Inc. is used within this work [25]. This product family integrates FPGA resources, an AVR 8-bit RISC microcontroller core, several peripherals and up to 36K Bytes SRAM within a single chip. The AVR microcontroller core is a common embedded processor, e.g., on SmartCards and is also available as a stand-alone device. The AVR is capable of 129 instructions, most of which can be performed within a single clock cycle. This results in a 20+ MIPS throughput at 25 MHz clock rate.

The FPGA resources within the FPSLIC devices are based on Atmel’s AT40K FPGA architecture. A special feature of this architecture is FreeRam4 cells which are located at the corners of each 4x4 cell sector. Using these cells results in minimal impact on bus resources and by that in fast and compact FPGA designs. The FPGA part is connected to the AVR over an 8-bit data bus. The amount of available FPGA resources ranges from about 5K system gates within the so-called μ FPSLIC to about 40K system gates within the AT94K40.

Both, the AVR microcontroller core and the FPGA part are connected to the embedded memory separately. Up to 36K Bytes SRAM are organized as 20K Bytes program memory, 4K Bytes data memory and 12K Bytes that can dynamically be allocated as data or program memory.

Atmel provides a complete design environment for the FPSLIC including tools for software development (C Compiler), tools for hardware development (VHDL synthesis tools) and a HW/SW co-

verification tool, which supports the concurrent development of hardware and software.

5 Implementation

5.1 Hardware Implementation

Operation	Bit Width	FFCP Clock Cycles	
		Best case	Worst case
FF-Mult	160	36	152
FF-Add	160	19	136
FF-Square	160	1	91
EC-Double	160		493
EC-Add	160		615
k.P	160		130.200

Table 5.1 FFCP performance values

The subsequently detailed FPGA designs have been implemented by using the design tools which are packaged with the utilized demonstration board. For hardware synthesis this is *Leonardo v2000.1b* from Mentor, Inc. The FPGA mapping is done with *Figaro IDS v7.5* from Atmel, Inc. Also from Atmel, Inc. there is the top-level design environment called *System Designer v2.1*, which is required to build up the entire design based on the AVR and the FPGA part.

For the implementation presented here, the particular design parameters are fixed to 160-bit operand width and 5-segment Karatsuba multiplication (MSK₅). This results in a FPGA utilization of 96% for the entire FFCP design.

Due to the fact that the result of each operation is fed back into one of the operand registers, the cycle count of a particular operation (I/O overhead plus actual computation) differs regarding to data dependencies. The corresponding best- and worst-case value for each FF operation is denoted in Tab. 5.1.

Tab. 5.1 unveils that the major part of cycles is necessary to transfer 160-bit operands over the fixed 8-bit interface between AVR and FPGA. These transfers can be avoided almost completely with an additional register file on the FFCP and an extended version of the finite state machine, which interprets commands given by the software running on the AVR. Assuming a 2-byte command format (4 bit opcode, 12 bit to specify the destination and the source registers) results in cycle counts according to the right column of Tab. 5.1.

5.2 Performance Comparison

Target Platform	Bit Width	k.P
FPGA(XCV300, 45MHz)[18]	113	3.7ms
FPGA(XC4085XLA, 37MHz)[17]	155	1.3ms
FPSLIC with FFCP(AT94K40,12MHz)	160	10.9ms

Table 5.2 Performance Comparison

There are several FPGA based hardware implementations of EC point multiplication documented in the literature [17,18,19, 20]. The performance values of these state-of-the-art implementations are given in Tab. 5.2. Additionally, Tab. 5.2 comprises the particular figures of the previously described FPSLIC based implementations.

A performance comparison of hardware implementations against each other is in general not straight forward. This is mostly because of different key sizes and due to the fact that different FPGA technologies are used for their implementation.

A basically scalable HW architecture is common to all implementations referenced in Tab. 5.2. In contrast to our SoC approach, the implementations in [17, 18, 19 and 20] are mainly focusing on high-security, server based applications. Their functionality is entirely implemented within a relatively large FPGA and no arrangements against side-channel attacks are documented.

In [17] and [18] the underlying field representation is an optimal normal basis. Both implementations are based on FPGAs from Xilinx,

Inc. Furthermore, VHDL module generators are used in both cases to derive the particular HW descriptions. The approach in [18] allows for a parameterization of the key size only. Parallelization, which is essential in order to achieve maximum performance from a specific FPGA, is additionally supported by the design in [17]. For the implementation in [18] a XCV300 FPGA with a complexity of about 320K system gates is used. The design in [17] is based on a XC4085XLA device with approx. 180K system gates. The implementations in [19] and [20] are both designed for polynomial bases and the field multiplications are in principle composed of partial multiplications.

The design in [19] is based on an Altera Flex10k family device with a complexity of about 310K system gates. The best performing implementation, representing the current benchmark with respect to k.P performance, is described in [20]. It is highly optimized, exploiting both pipelining and concurrency. The field multiplication is performed with a digit-serial multiplier. A Xilinx XCV400E FPGA with a complexity of about 570K system gates, running at 76.7 MHz is used for the implementation. Compared to our design this signifies a factor of more than 10 in space and a factor of about 6 in speed.

6 Conclusion

In this paper, we propose a communication security scheme for sensor networks. The straightforward approach to the secure communication in sensor networks could be the application of a single security mechanism for all data in the network. However, if the mechanism is chosen according to the most sensitive data in the network, security related resource consumption might be unacceptable. On the other hand, a less consuming mechanism could allow for serious security threats. Therefore, the solution lies in the identification of appropriate security requirements for various types of data and the application of suitable security mechanisms.

We have described a secure technique for wireless communication based on elliptic curve cryptographic techniques and implementation of Elliptic curve over Atmel FPSLIC Hardware platform. With a 160-bit modulus, an elliptic curve system seems to offer the same level of cryptographic security as DSA or RSA with 1024-bit moduli. The smaller key sizes result in smaller system parameters, smaller public-key certificates, bandwidth savings, faster implementations, lower power requirements, and smaller hardware processors [14].

The RSA-based protocols have significant problems in terms of the bandwidth and storage requirements. Currently, the RSA algorithm requires that the key length be at least 1024 bits for long term security, however, it seems that 160 bits are sufficient for elliptic curve cryptographic functions. Thus, the use of the ECC in wireless communication system is highly recommended. The proposed technique is a step in this direction.

7 Bibliography

[1] A. Sinha and A. Chandrakasan, "Dynamic power management in wireless sensor networks" IEEE Design and Test of Computers, pp. 62-74, March-April 2001.

[2] H. Çam, S. Ozdemir, D. Muthuavinashiappan, and P. Nair, "Energy-efficient security protocol for wireless sensor networks", IEEE VTC Fall 2003 Conference, October 4-9, Orlando, 2003

[3] P. Yanbin, W. Xiangyu and W. Youcha, "The sensor network based on LONWORKS Technology", SICE 1999, Pr0001-3/99/0000-0897.

[4] J. Rabaey, et al., "PicoRadios for Wireless Sensor Networks: The Next Challenge in Ultra-Low-Power Design," in Proceedings of the International Solid-State Circuits Conference, (San Francisco, CA), February 2002.

[5] M. Younis, M. Youssef, and K. Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks," in Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and

Simulation of Computer and Telecommunication Systems (MASCOTS2002), (Forth Worth, TX), October 2002.

[6] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy Aware Wireless Microsensor Networks," IEEE Signal Processing Magazine, March 2002.

[7] E. Shih, B. Calhoun, S.-H. Cho, and A. Chandrakasan, "Energy-Efficient Link Layer for Wireless Microsensor Networks," in Proceedings of the Workshop on VLSI 2001 (WVLSI '01), (Orlando, Florida), April 2001.

[8] Frazer Bennett, David Clarke, Joseph B. Evans, Andy Hopper, Alan Jones, and David Leask. Piconet: Embedded mobile networking. *IEEE Personal Communications*, 4(5):8-15, Oct 1997.

[9] J. M. Kahn, Katz R. H., and K. S. J. Pister. Next century challenges: mobile networking for "smart dust". In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 271-278. ACM, 1999.

[10] Scott Meininger, Jose Oscar Mur-Miranda, Rajeevan Amirtharajah, Anantha P. Chandrakasan, and Jeffrey H. Lang. Vibration-to-electric energy conversion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):64-76, Feb 2001.

[11] Sanjay Sarma, David L. Brock, and Kevin Ashton. The networked physical world - proposals for engineering the next generation of computing, commerce & automatic identification. White paper, MIT: AutoID Center, Oct 2000.

[12] J. Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT 97*, Lecture Notes in Computer Science, No. 1233, pages 239-255. Springer-Verlag, Berlin, Germany, 1997.

[13] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, Berlin, Germany, Second edition, 1994.

[14] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA, 1993.

[15] S. M. Redl and M. K. Weber. *An Introduction to GSM*. Artech House, Norwood, MA, 1995.

[16] B. Schneier. *Applied Cryptography*. John Wiley & Sons, New York, NY, Second edition, 1996.

[17] M. Ernst, S. Klupsch, O. Hauck and S. A. Huss, "Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems," *Proc. 12th IEEE Workshop on Rapid System Prototyping (RSP01)*, Monterey, CA, 2001.

[18] K.H. Leung, K.W. Ma, W.K. Wong and P.H.W. Leong, "FPGA Implementation of a Micro coded Elliptic Curve Cryptographic Processor," *Proc. IEEE FCCM 2000*, pp. 68-76, Napa Valley, 2000.

[19] S. Okada, N. Torii, K. Itoh and M. Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an FPGA," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS 1965, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 25-40, 2000.

[20] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS 1965, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 41-56, 2000.

[21] IEEE 1363, "Standard Specifications For Public Key Cryptography," <http://grouper.ieee.org/groups/1363/2000>.

[22] ANSI X9.62, "Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)," (available from the ANSI X9 catalog), 1999.

[23] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Sov. Phys.-Dokl (Engl. transl.)*, vol. 7, no. 7, pp. 595-596, 1963.

[24] D. V. Bailey and C. Paar, "Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography," *Journal of Cryptology*, vol. 14, no. 3, pp. 153-176, 2001.

[25] Atmel, Inc. "Configurable Logic Data Book," 2001.

[26] IEEE P1363. Standard specifications for public-key cryptography. Draft Version 7, September 1998.