Reconfigurable Instruction Set Extension for Enabling ECC on an 8-bit Processor

Sandeep Kumar and Christof Paar

Chair for Communication Security
Ruhr-Universität Bochum, 44780 Bochum, Germany
{kumar,cpaar}@crypto.rub.de

Abstract. Pervasive networks with low-cost embedded 8-bit processors are set to change our day-to-day life. Public-key cryptography provides crucial functionality to assure security which is often an important requirement in pervasive applications. However, it has been the hardest to implement on constraint platforms due to its very high computational requirements. This contribution describes a proof-of-concept implementation for an extremely low-cost instruction set extension using reconfigurable logic, which enables an 8-bit micro-controller to provide full size elliptic curve cryptography (ECC) capabilities. Introducing full size public-key security mechanisms on such small embedded devices will allow new pervasive applications. We show that a standard compliant 163-bit point multiplication can be computed in 0.113 sec on an 8-bit AVR micro-controller running at 4 Mhz with minimal extra hardware, a typical representative for a low-cost pervasive processor. Our design not only accelerates the computation by a factor of more than 30 compared to a software-only solution, it also reduces the code-size, data-RAM and power requirements.

1 Introduction

Ubiquitous computing with low cost pervasive devices has started to become reality with RFID applications and smart textiles. These computing devices form large-scale collaborating networks by exchanging information. Privacy and security of this information is important for the overall reliability of these networks and ultimately to the trustworthiness of pervasive applications. In fact, security is often viewed as a crucial feature, a lack of which can be an obstacle to the wide-spread introduction of pervasive applications.

High-volume, low-cost and very small power budgets of pervasive devices implies they have limited computing power, often not exceeding an 8-bit processor clocked at a few MHz. Under these constraints, secure public-key cryptography for authentication are nearly infeasible in software and are therefore usually not available in these systems. On the other hand, public-key cryptography offers major advantages when designing security solutions in pervasive networks. An alternative is to use a cryptographic co-processor such as used for high security applications like smart-cards, but its downside are considerable costs (in terms of power and chip area) which makes it unattractive for many cost sensitive pervasive applications. In addition, a fixed hardware solution may not offer the cryptographic flexibility (i.e., change of parameters or key length) that can be required in real-world applications. An instruction set extension (ISE) is a more viable option because of the smaller amount of additional hardware required and because of its flexibility. The efficiency of an ISE is not just measured by the speed-up it achieves, but also in the decrease in code-size, data-RAM and power consumption.

We use reconfigurable hardware attached to an 8-bit processor to simulate the ISE and to obtain reliable cost/benefit estimates. However, we view the reconfigurability not only useful for prototyping purposes, but a small reconfigurable hardware extension is also an attractive platform for embedded devices as the extension can offer many speed and power benefits for computationally intensive applications as demonstrated in this paper. It should be noted that public-key operations are typically only needed at the initial or final stage of a communication session. Hence, it is perceivable that the ISE can be runtime reconfigured for other applications when public-key operations are not required.

The paper is organized as follows. Section 2 discusses previous work in this field. In Sect. 3 we describe ISEs and in Sect. 4 the mathematical background of ECC is discussed. Section 5 discusses our implementation and results.

2 Previous work

The use of group of points of an elliptic curve in cryptography was first suggested by Neal Kobilitz [14] and independently by Victor Miller [17]. There has been considerable work since then on efficient implementation of ECC in software, typically targeting high end processors [19, 21, 7, 9]. In the following is a discussion of ECC implementations on constrained environment.

ECC on 8-bit processors have been reported in [5] and [22], both implemented over Optimal Extension Fields (OEF's), originally introduced in [2]. It should be noted that OEFs are not standardized, and their security in conjunction of ECC is not clear. In [5], the ECC implementation is over the field $\mathbb{F}_{p^m}$ with $p = 2^{16} - 165$, $m = 10$, and irreducible polynomial $f(x) = x^{10} - 2$. A performance of 122 msec at 20 Mhz is reported for a 160-bit point multiplication. The sub-field multiplication is done using the math co-processor. [22] implements ECC over $GF((2^8 - 17)^{17})$ on an 8051 micro-controller without co-processor but instead uses the internal 8-by-8-bit integer multiplier. The authors achieve a speed of 1.95 sec for a 134-bit fixed point multiplication using 9 pre-computed points and 8.37 sec for a general point multiplication using binary method of exponentiation. In [15], the authors improve the general point multiplication, to set up an end-to-end wireless ECDH key exchange within 3 sec on a Chipcon CC1010, which is based on the 8051 architecture. An ECDSA implementation on a 16-bit microcomputer M16C, running at 10 Mhz, is described in [10]. The authors propose the use of a field $\mathbb{F}_p$ where prime characteristic $p = e2^c \pm 1$, e an integer within the machine size and c a multiple of machine word size.

The implementation uses a 31-entry table of precomputed points to generate an ECDSA signature in 150 msec and ECDSA verification takes 630 msec. A scalar multiplication of a random point takes 480 msec. The authors in [7] describe an efficient implementation of ECC over $\mathbb{F}_p$ on the 16-bit TI MSP430x33x family of low-cost micro-controllers running at 1 Mhz. A scalar point multiplication over $GF(2^{128} - 2^{97} - 1)$ is performed in 3.4 sec without any precomputation.

It should be stressed that all previous ECC implementations on 8-bit processors have been based on non-standardized ECC parameters in order to overcome the performance bottleneck. This has two disadvantages: first, such solutions are incompatible with standardized protocols; secondly, there is always the possibility that non-standardized parameters have security shortcomings, e.g., new attacks through special mathematical properties introduced in non-standardized underlying finite fields.

Another approach has been to add a crypto co-processor to these micro-controllers. A survey of commercially available co-processors can be found in [8]. However, a full-size ECC co-processor is may be prohibitively expensive for many pervasive applications. Hardware assistance in terms of Instruction Set Extensions (ISE) is more favorable as the cost of extra hardware is quite negligible compared to the whole processor. Previous attempts in this direction [6, 13] are only reported for ECC with not more than 133-bits.

3 Instruction Set Extension

The Instruction Set Architecture (ISA) of a microprocessor is the unique set of instructions that can be executed on the processor. General purpose ISA are often insufficient to satisfy the special computational needs in cryptographic applications. A more promising method is extending the ISA to build Application Specific Instruction-set Processors (ASIP.)

There are different ways of extending a processor. We consider an extension as shown in Fig. 1. Here the additional hardware is closely coupled with the arithmetic logic unit (ALU) of the processor, reducing the interface circuitry. The control circuit of the processor is extended to support this extra hardware. The extension can also directly access the data-RAM which is important if the computation is done over several data elements. For multi-cycle instructions, the software has to take special care not to call the custom hardware until the multi-cycle operation is completed.

An efficient ISE implementation requires a tightly coupled hardware and software co-design. In a first step, we used a software-only implementation of ECC to identify the functional elements and code-segments that would provide efficiency gains if implemented as an ISE. Then, a hardware model of the new processor determines the effects of the new extension on the parameters running time, code-size and data-RAM usage.

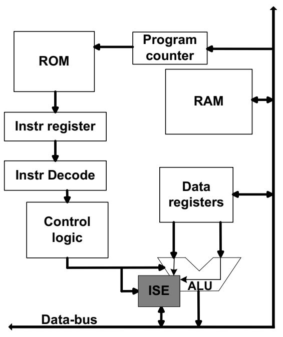

Fig. 1. Processor Core with Extension

4 Elliptic Curve Cryptography

Among public-key algorithms, there are three established families: RSA, DL and elliptic curve cryptography (ECC.) Among them, ECC is considered the most attractive one for embedded environments [4] due to its smaller operand lengths and *relatively* lower computational requirements. ECC is also standardized [18, 1, 11, 12] and has become a commercially accepted method. ECC is an universal public-key family, allowing mechanisms such as digital signature, key exchange and data encryption. The vast majority of modern protocols, including wireless protocols attractive for pervasive applications, require these public-key functions. A more detailed description of the main operations for an ECC realization can be found in, e.g., [3].

4.1 Elliptic Curves

An elliptic curve over $\mathbb{F}_{2^m}$is of the form $y^2 + xy = x^3 + ax^2 + b,$ where $a, b \in \mathbb{F}_{2^m}$ and $b \neq 0$. (x, y) satisfying this equation is called a *point* $\mathbf{P}$ on the curve. Together with $\mathcal{O}$, the identity element, they constitute an abelian set $E(\mathbb{F}_{2^m})$. For our implementation, we use a NIST-recommended 163-bit random curve [18].

The main EC cryptographic operation is the *scalar point multiplication*, $\mathbf{Q} = k \cdot \mathbf{P}$, where $\mathbf{P}, \mathbf{Q}$ are points on the elliptic curve and k is an m-bit integer.

4.2 Elliptic Curve Point Arithmetic

For the implementation of the scalar point multiplication, $k \cdot \mathbf{P}$, we need to implement the group operations point addition and point doubling. The arithmetic is described in detail in [19]. For our implementation, we use the projective coordinates where the projective point (X, Y, Z) with $Z \neq 0$ corresponds to the affine point (x, y) where $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$. This projective co-ordinate is chosen because we use the Montgomery point multiplication introduced in [16].

The efficiency of EC group operations depends largely on the efficiency of the underlying field arithmetic. We use the polynomial basis representation $(a_{m-1}...a_1a_0)$ with the reduction polynomial $F(x) = x^{163} + x^7 + x^6 + x^3 + 1$. In our implementation, an element from $\mathbb{F}_{2^{163}}$ is represented as an array of 21 eight bit words, with the five last most-significant bits being ignored.

Field Addition is the simplest of all operations, since it is a bit by bit addition in $\mathbb{F}_2$ which maps to word-level XOR operation in software.

Field Multiplication of two elements is a polynomial multiplication followed by reduction modulo $F(x)$. The polynomial multiplication can be implemented in software efficiently using the comb method.

Field Squaring is a simple expansion in $\mathbb{F}_{2^m}$in polynomial basis which is efficiently implemented as a table-lookup.

Field Reduction. Multiplication and squaring require reduction of the polynomial of degree not greater than $(2m - 1)$. Reduction is effectively done using a table-lookup for the 8-bit locations in a byte.

5 Implementation

Our development platform is the Atmel Inc.'s AT94K family of FPSLIC devices (Field Programmable System Level Integrated Circuits). This architecture integrates an AVR 8-bit micro-controller core (used widely in smart-cards and micromotes), FPGA resources, several peripherals and 36K bytes SRAM within a single chip. The platform is appealing for simulating an ISE. The implementations are done on the ATSTK94 FPSLIC demonstration board clocked at 4 Mhz.

We first implemented the ECC algorithm on the 8-bit AVR processor in assembly. The results of the software only implementation is given in Table 1. It is important to mention that the multiplication routine based on the comb method that we used is among the fastest known software algorithms for Galois field multiplication.

The analysis of the software-only implementation shows that $\mathbb{F}_{2^m}$multiplication is the most costly operation with respect to execution time and memory requirement. Moreover, in the Montgomery algorithm (Sect. 4.2), field multiplications are extremely frequent making it the bottleneck operation for ECC. A closer look at the multiplication block showed that the major part of the time was spent for load/store operations because of the small number of registers which cannot hold the large operands. Therefore an ISE for this functional block which also reduces the memory bottleneck can greatly speed-up ECC.

The popular approach to multimedia extensions has been to divide a large 32/64-bit data-bus into smaller 8-bit or 16-bit multimedia variables, and to run

Table 1. $\mathbb{F}_{2^{163}}$ ECC software-only performance on an 8-bit AVR μC (@4 Mhz)

Operation	**Time** (clocks)	**Code-size** (bytes)	**Data RAM** (bytes)
Addition	151	180	42
Multiplication	15044	384	147
Squaring	441	46	63
Reduction	1093	196	63
Point Multiplication ($k.P$)	4.14 sec	8208	358

those in parallel as an SIMD instruction. But for public-key cryptographic applications the reverse is true: The operands are much larger than the data-path, requiring many bit operations on long operands. For such applications, bit-parallel processing is required where multiple data-words are operated upon simultaneously. One important issue here is the provision of the ISE with the operands. We approached this situation by implementing a complete $\mathbb{F}_{2^{163}}$ multiplier with minimum possible area.

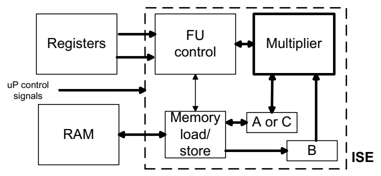

Fig. 2. ISE Interface and Structure

Figure 2 shows the general layout of functional unit (FU) of the ISE we are simulating. Four processor registers are initially loaded with the memory addresses of the two operands A and B. The ISE is then initiated by a control signal to the FU control (FUC) along with the first memory address byte. In our proof-of-concept implementation, this behavior is achieved by sending the byte over the data-lines from the processor to the FPGA, and confirming its reception through interrupt-lines from the FPGA to the processor. After the last memory address byte is received, the FUC initiates the memory load/store circuit within the ISE to load both the 21-byte operands directly from the SRAM in 21-cycles each. Then, the FUC runs the multiplier for 163-cycles to get the result C. During this period, the processor loads the memory address of C, sends it to the FPGA and goes into polling state for the final interrupt from the FPGA. After the result C is obtained, the ISE stores it back directly in the memory in another 21-cycles and then sends the final interrupt signalling the completion of the multiplication. This method of handshaking leads to extra control overheads which can be reduced by having a more tightly coupled ISE to the processor without requiring confirmation interrupts. During the idle polling state, the processor could also be used in other computational work which is independent of the multiplication result. Memory access conflicts during such computation between the processor and the ISE is avoided by using a dual ported SRAM.

5.1 Bit-Serial Multiplier

A bit-serial $\mathbb{F}_{2^m}$ hardware multiplier is the most simple solution and requires the least area. The core of the multiplier is as shown in Fig. 3. The reduction circuit is hardwired here. A modification for implementing a more general reduction polynomial or variable size multiplication is discussed in Sect. 5.3. A 163-by-163 multiplication is computed in 163 clocks, excluding data input and output. In our implementation, control and memory access overheads lead to a total execution time of 313 clocks. Since the multiplier is much faster than the squaring in software, we use the multiplier also for squaring by loading $A = B$. The results (Table 2) show a drastic speed-up using this multiplier. It should be noted that the control overhead can be considerably reduced when the hardware is more tightly coupled within the processor, e.g., in an ASIC implementation of our architecture.

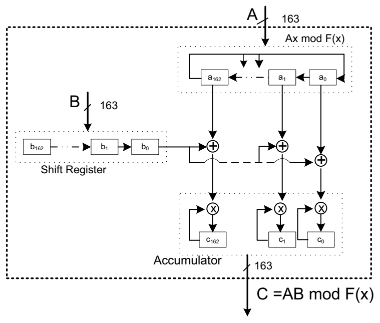

Fig. 3. Bit-Serial LSB Multiplier

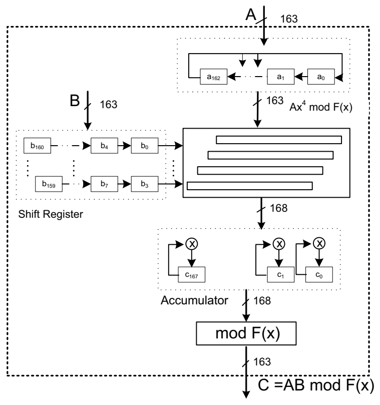

Fig. 4. Digit-4 Serial LSD Multiplier

5.2 Digit-Serial Multiplier

Another trade-off between area and speed is possible by using digit-serial multipliers [20]. Compared to the bit-serial multiplier where only one bit of operand B is used in each iteration, here multiple bits (equal to the digit-size) of B are multiplied to the operand A in each iteration (Fig. 4). We use a digit size of 4 as it gives a good speed-up without drastically increasing the area requirements. A 163-by-163 multiplication with reduction requires 42 clocks. In our implementation, the control overheads leads to a total of 193 clocks.

Table 2. ICE-based ECC point multiplication (@4 Mhz)

Multiplier Type	**CLBs**	**Time** (sec)	**Code-size** (bytes)	**Data RAM** (bytes)
Software-only		4.14	8208	358
163*163 multiplier	245	0.169	2048	273
163*163 digit-4	498	0.113	2048	273

5.3 A Flexible Multiplier

Flexibility of crypto algorithm parameters (especially operand lengths) can be very attractive because of the need to alter them when deemed insecure in the future, or for providing compatibility in different applications. Considering the high-volume of pervasive devices, replacing each hardware component seems improbable. We discuss here how the multiplier can be made more flexible to satisfy these needs.

Support of a generic reduction polynomial with a maximum degree of m of the form $F(x) = x^m + G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i$ requires storage of the reduction coefficients and additional circuitry as shown in Fig. 5 (a similar implementation for a digit-serial multiplier is straightforward). The reduction polynomial can be initialized once in the beginning of the point multiplication. Thus the total number of clocks required for multiplication remains the same.

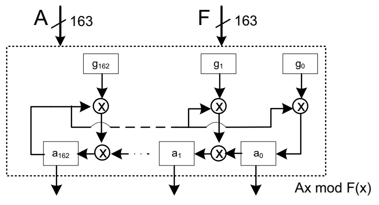

Fig. 5. Bit-serial reduction circuit

Different bit-length multipliers for different key-length ECC can also be supported using this structure. We show as an example, how the 163-bit multiplier could be also used to multiply two 113-bit operands A' and B' with 113-bit reduction polynomial G'.

The operands A, B and the reduction polynomial are initially loaded as

$$\begin{aligned} A &= (a'_{112}...a'_1a'_0 0...0) = A'x^{50} \\ B &= (0...0b'_{112}...b'_1b'_0) \ = B' \\ G &= (g'_{112}...g'_1g'_0 0...0) = G'x^{50} \end{aligned}$$

If $C' = A' \cdot B' \bmod F'(x)$ then

$$A \cdot B \bmod F(x) = A'x^{50} \cdot B' \bmod (F'(x)x^{50}) = C'x^{50}$$

Thus the result is stored in the most-significant bits of operand C after 113 clock cycles. The memory load/store circuit and the FU control unit takes care to load the operands appropriately and to fetch the result after the required number of clocks from the multiplier and store it back appropriately in memory.

6 Conclusions

Huge performance gains are possible in small 8-bit processors by introducing small amounts of extra hardware. The results show 1–2 orders of magnitude increase in speed-up for the ECC implementation. The hardware costs are in the range of 250–500 extra CLBs. There is also saving in the code size and data RAM usage for the algorithm. The performance gain due to the ISE can be used to reduce the total power consumption of the devices by running the whole device at a lower frequency, which can be a major benefit in wireless pervasive applications. The proof-of-concept implementation can also be used directly as a reconfigurable ISE. Since public-key exchange is done only in the initial phase of the communication, the FPGA can be run-time reconfigured for an ISE suitable for a different application (like signal processing) running later on the device. Thus two different sets of ISEs can be run on the same constrained device, accelerating both applications without increasing the total hardware cost.

References

1. ANSI X9.62-1999. The Elliptic Curve Digital Signature Algorithm. Technical report, ANSI, 1999.
2. D. V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume LNCS 1462, pages 472–485, Berlin, 1998. Springer-Verlag.
3. I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, London Mathematical Society Lecture Notes Series 265, 1999.
4. M. Brown, D. Cheung, D. Hankerson, J. L. Hernandez, M. Kirkup, and A. Menezes. PGP in Constrained Wireless Devices. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.
5. Jae Wook Chung, Sang Gyoo Sim, and Pil Joong Lee. Fast Implementation of Elliptic Curve Defined over $GF(p^m)$ on CalmRISC with MAC2424 Coprocessor. In Çetin K. Koç and Christof Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 57–70, Berlin, 2000. Springer-Verlag.
6. M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Blümel. A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2n). In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, pages 381–399. Springer-Verlag, 2002.
7. J. Guajardo, R. Bluemel, U. Krieger, and C. Paar. Efficient Implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers. In K. Kim, editor, *Fourth International Workshop on Practice and Theory in Public Key Cryptography - PKC 2001*, volume LNCS 1992, pages 365–382, Berlin, February 13-15 2001. Springer-Verlag.

8. Helena Handschuh and Pascal Paillier. Smart Card Crypto-Coprocessors for Public-Key Cryptography. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Proceedings of the The International Conference on Smart Card Research and Applications*, pages 372–379. Springer-Verlag, 2000.
9. D. Hankerson, J. López Hernandez, and A. Menezes. Software Implementation of Elliptic Curve Cryptography Over Binary Fields. In Ç. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS, Berlin, 2000. Springer-Verlag.
10. Toshio Hasegawa, Junko Nakajima, and Mitsuru Matsui. A Practical Implementation of Elliptic Curve Cryptosystems over $GF(p)$ on a 16-bit Microcomputer. In Hideki Imai and Yuliang Zheng, editors, *First International Workshop on Practice and Theory in Public Key Cryptography — PKC'98*, volume LNCS 1431, pages 182–194, Berlin, 1998. Springer-Verlag.
11. IEEE. *Standard Specifications for Public-Key Cryptography*, 2000.
12. ISO/IEC. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves*, 2002.
13. S. Janssens, J. Thomas, W. Borremans, P. Gijsels, I. Verhauwhede, F. Vercauteren, B. Preneel, and J. Vandewalle. Hardware/software co-design of an elliptic curve public-key cryptosystem, 2001.
14. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, vol. 48:203–209, 1987.
15. S. Kumar, M. Girimondo, A. Weimerskirch, C. Paar, A. Patel, and A. S.Wander. Embedded End-to-End Wireless Security with ECDH Key Exchange. In *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems — MWSCAS 2003*, December 2003.
16. J. López and R. Dahab. Fast multiplication on elliptic curves over GF(2 m) without precomputation. In Ç. Koç and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES '99*, volume LNCS 1717, pages 316–327, Berlin, August 1999. Springer-Verlag.
17. V. S. Miller. Use of elliptic curves in cryptography. *CRYPTO '85*, pages 417–426, 1986.
18. NIST. *Recommended Elliptic Curves for Federal Government Use*, May 1999.
19. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. In D. Coppersmith, editor, *Advances in Cryptology — CRYPTO '95*, volume LNCS 963, pages 43–56, Berlin, 1995. Springer-Verlag.
20. L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, June 1998.
21. E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In *Asiacrypt '96*, volume LNCS 1233, pages 65–76, Berlin, 1996. Springer-Verlag.
22. A. Woodbury, D. V. Bailey, and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *CARDIS 2000*, Bristol, UK, September 20–22 2000. Kluwer.