Cryptography in Embedded Systems: An Overview

Thomas Wollinger, Jorge Guajardo, and Christof Paar

Department of Electrical Engineering and Information Sciences
Communication Security Group (COSY)
Ruhr-Universität Bochum, Germany
Universitaetsstrasse 150
44780 Bochum, Germany
{wollinger,guajardo,cpaar}@crypto.ruhr-uni-bochum.de

Proceedings of the Embedded World 2003 Exhibition and Conference, pp. 735-744, Design & Elektronik, Nuernberg, Germany, February 18-20, 2003.

Abstract. It is widely recognized that data security will play a central role in the design of future IT systems. Many of those IT applications will be realized as embedded systems which rely heavily on security mechanisms. Examples include security for wireless phones, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products and digital cinemas. Note that a large share of those embedded applications will be wireless, which makes the communication channel especially vulnerable.
All modern security protocols use symmetric-key and public-key algorithms. This contribution surveys several important cryptographic concepts and their relevance to embedded system applications. We give an overview of the previous work in the area of embedded systems and cryptography.

1 Introduction

It is widely recognized that data security will play a central role in the design of future IT systems. Until a few years ago, the PC had been the major driver of the digital economy. Recently, however, there has been a shift towards IT applications realized as embedded systems. Many of those applications rely heavily on security mechanisms, including security for wireless phones, faxes, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products and digital cinemas. Note that a large share of those embedded applications will be wireless, which makes the communication channel especially vulnerable and the need for security even more obvious.

This merging of communications and computation functionality requires data processing in real time, and embedded systems have shown to be good solutions for many applications. Examples of such applications are cellular phones, faxes, pagers, and Internet solutions such as modems, multi-service network solutions that allow the implementation of IP telephony, Digital Subscriber Line (DSL) technologies, and some electronic commerce devices, to name just a few. Since many of these applications will need security functionality in the future, this paper discusses cryptographic algorithms and their implementation on embedded systems.

In addition to embedded devices, the explosive growth of digital communications also brings additional security challenges. Millions of electronic transactions are completed each day, and the rapid growth of eCommerce has made security a vital issue for many consumers. In the future, valuable business opportunities will be realized over the Internet and megabytes of sensitive data will be transferred and moved over insecure communication channels around the world. Thus, it is imperative for the success of modern businesses that all these transactions be realized in a secure manner. Specifically, unauthorized access to information must be prevented, privacy must be protected, and the authenticity of electronic documents must be established. Cryptography, or the art and science of keeping messages secure [26], allows us to solve these problems. We believe that cryptographic engines realized on embedded systems are a promising option for protecting eCommerce systems.

The implementation of cryptographic systems presents several requirements and challenges. First, the performance of the algorithms is often crucial. One needs encryption algorithms to run at the transmission rates of the communication links. Slow running cryptographic algorithms translate into consumer dissatisfaction and inconvenience. On the other hand, fast running encryption might mean high product costs since traditionally, higher speeds were achieved through custom hardware devices.

In addition to performance requirements, guaranteeing security is a formidable challenge. An encryption algorithm running on a general-purpose computer has only limited physical security, as the secure storage of keys in memory is difficult on most operating systems. On the other hand, hardware encryption devices can be securely encapsulated to prevent attackers from tampering with the system. Thus, custom hardware is the platform of choice for security protocol designers. Hardware solutions, however, come with the well-known drawback of reduced flexibility and potentially high costs. These drawbacks are especially prominent in security applications which are designed using new security protocol paradigms.

Many of the new security protocols decouple the choice of cryptographic algorithm from the design of the protocol. Users of the protocol negotiate on the choice of algorithm to use for a particular secure session. The new devices to support these applications, then, must not only support a single cryptographic algorithm and protocol, but also must be "algorithm agile," that is, able to select from a variety of algorithms. For example, IPSec (the security standard for the Internet) allows to choose out of a list of different symmetric as well asymmetric ciphers. Some of the symmetric-key algorithms are: DES, 3DES, Blowfish, CAST, IDEA, RC4, RC6, and so on. Thus, software-based systems would seem to be a better fit because of their flexibility. However, the security engineer is faced with a difficult choice. Should he/she choose in favor of performance and security, and pay the price of inflexibility and higher costs? Or should he/she favor flexibility instead? Fortunately, many embedded processors combine the flexibility of software on general-purpose computers with the near-hardware speed and better physical security than general-purpose computers.

Embedded processors are already an integral part of many communications devices and their importance will continue to increase. If we combine this with their flexibility to be programmed and their ability to perform arithmetic operations at moderate speeds, it is easy to see that they are a very promising platform to implement cryptographic algorithms.

This paper focuses on the basics of cryptography and the implementation of cryptographic applications on embedded systems. In Section 2, we introduce the general theory and concepts of symmetric-key and public-key cryptography as well as the operations which are most commonly performed. We will show that public-key operations are very computationally intensive and therefore require platforms which have strong arithmetic capabilities. In Section 3, a survey of previous cryptographic implementations on embedded systems is presented, as well as some of the characteristics of the proposed algorithms. We give an overview of implementations of symmetric-key and public-key algorithms. Finally, we end this contribution with some conclusions.

2 Cryptography: Public-key and Symmetric-key Algorithms

2.1 What We Can Do with Cryptography

Cryptography involves the study of mathematical techniques that allow the practitioner to achieve or provide the following objectives or services [22, 27]:

- *Confidentiality* is a service used to keep the content of information accessible to only those authorized to have it. This service includes both protection of all user data transmitted between two points over a period of time as well as protection of traffic flow from analysis.
- *Integrity* is a service that requires that computer system assets and transmitted information be capable of modification only by authorized users. Modification includes writing, changing, changing the status, deleting, creating, and the delaying or replaying of transmitted messages. It is important to point out that integrity relates to active attacks and therefore, it is concerned with detection rather than prevention. Moreover, integrity can be provided with or without recovery, the first option being the more attractive alternative.
- *Authentication* is a service that is concerned with assuring that the origin of a message is correctly identified. That is, information delivered over a channel should be authenticated as to the origin, date of origin, data content, time sent, etc. For these reasons this service is subdivided into two major classes: entity authentication and data origin authentication. Notice that the second class of authentication implicitly provides data integrity.
- *Non-repudiation* is a service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions.

These security services are provided by using cryptographic algorithms. There are two major classes of algorithms in cryptography: Private-key or Symmetric-key algorithms and Public-key algorithms. The next two sections will describe them in detail.

2.2 Symmetric-key Algorithms

Private-key or Symmetric-key algorithms are algorithms where the encryption and decryption key is the same, or where the decryption key can easily be calculated from the encryption key and vice versa. The main function of these algorithms, which are also called secret-key algorithms, is encryption of data, often at high speeds. Private-key algorithms require the sender and the receiver to agree on the key prior to the communication taking place. The security of private-key algorithms rests in the key; divulging the key means that anyone can encrypt and decrypt messages. Therefore, as long as the communication needs to remain secret, the key must remain secret.

There are two types of symmetric-key algorithms which are commonly distinguished: block ciphers and stream ciphers [26]. Block ciphers are encryption schemes in which the message is broken into strings (called blocks) of fixed length and encrypted one block at a time. Examples include the Data Encryption Standard (DES) [11], the International Encryption Standard (IDEA) [19, 21], and the Advanced Encryption Standard (AES) [29]. Note that, due to its short block size and key length, DES expired as a US standard in 1998, and that the National Institute of Standards (NIST) selected Rijndael algorithm as the AES in October 2000. AES has a minimum block size of 128 bits and the ability to support keys of 128, 192 and 256 bits in length.

Stream ciphers operate on a single bit of plaintext at a time. In some sense, they are block ciphers having block length equal to one. They are useful because the encryption transformation can change for each symbol of the message being encrypted. In particular, they are useful in situations where transmission errors are highly probable because they do not have error propagation. In addition, they can be used when the data must be processed one symbol at a time because of lack of equipment memory or limited buffering.

It is important to point out that the trend in modern symmetric-key cipher design has been to optimize the algorithms for efficient software implementation in modern processors. This is evident if one looks at the performance of the AES on different platforms. The internal AES operations can be broken down into 8-bit operations, which is important because many cryptographic applications run on smart cards. Furthermore, one can combine certain steps to get a suitable performance in the case of 32-bit platforms.

As a final remark, notice that one of the major issues with symmetric-key systems is the need to find an efficient method to agree on and exchange the secret keys securely [22]. This is known as the key distribution problem. In 1977, Diffie and Hellman [9] proposed a new concept that would revolutionize cryptography as it was known at the time. This new concept was called public-key cryptography.

2.3 Public-key Algorithms

Public-key (PK) cryptography is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. Anyone that wants to send a message to party A can encrypt that message using A's *public key* but only A can decrypt the message using her *private key*. In implementing a public-key cryptosystem, it is understood that A's private key should be kept secret at all times. Furthermore, even though A's public key is publicly available to everyone, including A's adversaries, it is impossible for anyone, except A, to derive the private key (or at least to do so in any reasonable amount of time).

In general, one can divide practical public-key algorithms into three families:

- Algorithms based on the *integer factorization problem:* given a positive integer n, find its prime factorization. RSA [25], the most widely used public-key encryption algorithm, is based on the difficulty of solving this problem.
- Algorithms based on the *discrete logarithm problem:* given α and β find x such that $\beta = \alpha^x \bmod p$. The Diffie-Hellman key exchange protocol is based on this problem as well as many other protocols, including the Digital Signature Algorithm (DSA).
- Algorithms based on *Elliptic Curves.* Elliptic curve cryptosystems are the most recent family of practical public-key algorithms, but are rapidly gaining acceptance. Due to their reduced processing needs, elliptic curves are especially attractive for embedded applications.

Despite the differences between these mathematical problems, all three algorithm families have something in common: they all perform complex operations on very large numbers, typically 1024–2048 bits in length for the RSA and discrete logarithm systems or 160–256 bits in length for the elliptic curve systems. Since elliptic curves are somewhat less computationally intensive than the other two algorithm families, they seem especially attractive for embedded applications.

The most common operation performed in public-key schemes is modular exponentiation, i.e., the operation $x^e \bmod n$. Performing such an exponentiation with, e.g., 1024-bit long operands is extremely computationally intensive. Interestingly enough, modular exponentiation with long numbers requires arithmetic which is very similar to that performed in signal processing applications [18], namely integer multiplication.

Public-key cryptosystems solve in a very elegant way the key distribution problem of symmetric-key schemes. However, PK systems have a major disadvantage when compared to private-key schemes. As stated above, public-key algorithms are very arithmetic intensive and — if not properly implemented or if the underlying processor has a poor integer arithmetic performance — this can lead to a poor system performance. Even when properly implemented, all PK schemes proposed to date are several orders of magnitude slower than the best known private-key schemes. Hence, in practice, cryptographic systems are a mixture of symmetric-key and public-key cryptosystems. Usually, a public-key algorithm is chosen for key establishment and authentication through digital signatures, and then a symmetric-key algorithm is chosen to encrypt the communications and the data transfer, achieving in this way high throughput rates.

3 Embedded Systems and Cryptography

The field of efficient algorithms for the implementation of cryptographic schemes is a very active one (for an overview on current techniques see [22, Chapter 14]). However, essentially all cryptographic research is being conducted independent of hardware platforms, and little research focuses on algorithm optimization for specific processors.

In the following, we will review previous implementations of symmetric-key and public-key algorithms on embedded systems. We will also summarize two of the fastest software implementations of PK schemes on general purpose computers. This will give the reader an idea as to the kind of speeds that are to be expected on general purpose machines, and which speeds can be expected in embedded system applications.

3.1 Symmetric-key Algorithms on DSPs

In [31], the authors investigated how well high-end DSPs are suited for the implementation of the final five AES candidate algorithms. In particular, the implementations are on a 200 MHz TMS320C6201 which performs up to 1600 million instructions per second (MIPS) and provides thirty two 32-bit registers and eight independent functional units. In what follows, we briefly describe the way [31] chose to code the Rijndael algorithm because there are several implementation options.

In [7], the authors of Rijndael proposed a way of combining the different steps of the round transformation into a single set of table lookups. Thus, the implementation in [31] uses 4 tables with 256 4-byte word entries. In addition to the optimizations described above, a second version of code in which data blocks can be processed in parallel was implemented. With parallel processing, the encryption and the decryption functions can operate on more than one block at a time using the same key. This allows better utilization of the DSP's functional units which leads to better performance. With parallel processing, however, the speedups may only be exploited in modes of operations which do not require feedback of the encrypted data, such as Electronic Code-Book (ECB) or Counter Mode. When operating in feedback modes such as Ciphertext Feedback mode, the ciphertext of one block must be available before the next block can be encrypted. The authors in [31] noticed that the Rijndael code can be optimized by the tools very efficiently. Thus, no performance advantage is obtained by parallel processing, which results in the same speed for single-block and multi-block modes. Table 1 summarizes the performance of Rijndael on the TMS320C6201.

Table 1. Performance results for the Rijndael algorithm on the TMS320C6201 [31]

	DSP multi-block mode @ 200MHz		DSP single-block mode @ 200MHz		Pentium-Pro @ 200MHz	DSP multi-block mode/Pentium
	cycles	Mbit/sec	cycles	Mbit/sec	Mbit/sec	
Rijndeal encryption	228	112.3	228	112.3	70.5 [12]	1.6
decryption	269	95.2	269	95.2	70.5 [12]	1.4

All the timings are obtained from a C implementation using the compiler version 4.0 alpha

3.2 Public-key Algorithms on Embedded Systems

In [3], the Barret modular reduction method is introduced. The author implemented RSA on the TI TMS32010 DSP. A 512-bit RSA exponentiation took on the average 2.6 seconds running at the DSP's maximum speed of 20 MHz. Reference [10] describes the implementation of a cryptographic library designed for the Motorola DSP56000 which was clocked at 20 MHz. The authors focused on the integration of modular reduction and multi-precision multiplication according to Montgomery's method [18, 23]. This RSA implementation achieved a data rate of 11.6 Kbits/s for a 512-bit exponentiation using the Chinese Remainder Theorem (CRT) and 4.6 Kbits/s without using it.

The authors in [15] described an ECDSA implementation over $GF(p)$ on the M16C, a 16-bit 10 MHz microcomputer. Reference [15] proposes the use of a field of prime characteristic $p = e2^c \pm 1$, where e is an integer within the machine word size and c is a multiple of the machine word size. This choice of field allows to implement multiplication in $GF(p)$ in a small amount of memory. Notice that [15] uses a randomly generated curve with the a coefficient of the elliptic curve equal to $p - 3$. This reduces the number of operations needed for an EC point doubling. They also modify the point addition algorithm in [24] to reduce the number of temporary variables from 4 to 2. This contribution uses a 31-entry table of precomputed points to generate an ECDSA signature in 150 msec. On the other hand, scalar multiplication of a random point takes 480 msec and ECDSA verification 630 msec. The whole implementation occupied 4 Kbyte of code/data space.

In [16], two new methods for implementing public-key cryptography algorithms on the 200 MHz TI TMS320C6201 DSP are proposed. The first method is a modified implementation of the Montgomery variant known as the Finely Integrated Operand Scanning (FIOS) algorithm [18] suitable for pipelining. The second approach suggests a method for reducing the number of multiplications and additions used to compute $2^m P$, for P a point on the elliptic curve and m some integer. The final code implemented RSA and DSA combined with the k-ary method for exponentiation, and ECDSA combined with the improved method for multiple point doublings, sliding window exponentiation, and signed binary exponent recoding. The total instruction code was 41.1 Kbytes. They achieved 11.7 msec for a 1024-bit RSA signature using the CRT (1.2 msec for verification assuming a 17-bit exponent) and 1.67 msec for a 192-bit ECDSA signature over $GF(p)$ (6.28 msec for verification and 4.64 msec for general point multiplication).

Recently, two papers have introduced fast implementations on 8-bit processors over Optimal Extension Fields (OEFs), originally introduced in [1]. Reference [5] reports on an ECC implementation over the field $GF(p^m)$ with $p = 2^{16} - 165$, $m = 10$, and $f(x) = x^{10} - 2$ is the irreducible polynomial. The authors use the column major multiplication method for field multiplication and squaring, for the specific case in which $f(x)$ is a binomial. They achieve better performance than when using Karatsuba multiplication because in this processor additions and multiplications take the same number of cycles. Modular reduction is done through repeated use of the division step instruction. For inversion, they use the variant of the Itoh and Tsujii Algorithm [17] proposed in [2]. For EC arithmetic they combine the mixed coordinate system methods of [6] and [20]. These combined methods allow them to achieve 122 msec for a 160-bit point multiplication on the CalmRISC with MAC2424 math coprocessor running at 20 MHz. The second paper [32] describes a smart card implementation over the field $GF((2^8 - 17)^{17})$ without the use of a coprocessor.

Reference [32] focuses on the implementation of ECC on the 8051 family of microcontrollers, popular in smart cards. The authors compare three types of fields: binary fields $GF(2^k)$, composite fields $GF((2^n)^m)$, and OEFs. Based on multiplication timings, the authors conclude that OEFs are particularly well suited for this architecture. A key idea of this contribution is to allow each of the 16 most significant coefficients resulting from a polynomial multiplication to accumulate over three 8-bit words instead of reducing modulo $p = 2^8 - 17$ after each 8-bit by 8-bit multiplication. Fast field multiplication allows the implementation to have relatively fast inversion operations following the method proposed in [2]. This, in turn, allows for the use of affine coordinates for point representation. Finally, the authors combine the methods above with a table of 9 precomputed points to achieve 1.95 sec for a 134-bit fixed point multiplication and 8.37 sec for a general point multiplication using the binary method of exponentiation.

We end this section by summarizing the contributions in [13] and [30]. In [13], an ECC implementation over prime fields on the 16-bit TI MSP430x33x family of low-cost microcontrollers is described. The authors in [13] show that it is possible to implement EC cryptosystems in highly constrained embedded systems and still obtain acceptable performance at low cost. They modified the EC point addition and doubling formulae to reduce the number of intermediate variables while at the same time allowing for flexibility. In addition, [13] use Generalized-Mersenne primes to implement the arithmetic in the underlying field, taking advantage of the special form of the moduli to minimize the number of precomputations needed to implement the underlying arithmetic. These ideas are combined to achieve an EC scalar point multiplication in 3.4 seconds without any stored/precomputed values and the processor clocked at 1 MHz. The authors in [30] implemented EC over binary fields on a Motorola Dragonball CPU which is used on the popular Palm Personal Digital Assistants (PDAs). The Dragonball offers 16-bit and 32-bit operations and runs at 16 MHz. Using Koblitz curves over $GF(2^{163})$, [30] shows that it is possible to perform an ECDSA signature generation operation in less than 0.9 sec. while a verification operation requires less than 2.4 sec. The authors point out that Koblitz curves over fields $GF(2^{163})$ provide about the same level of security as RSA with a 1024-bit length, while at the same time providing acceptable performance which is not possible to achieve by using RSA-based systems since the integer multiplier in the Dragonball processor is very slow.

3.3 Fast Software Implementations

The purpose of this subsection is to give timings of software implementations of public-key algorithms, in order to get a better appreciation for the timings of similar implementations on embedded processors.

The primary focus of [4] and [14] is to describe fast software implementations of elliptic curve cryptosystems over binary and prime fields on a Pentium II 400 MHz-based PC. The implementations used NIST recommended curves according to FIPS 186-2 [28]. The best timings from [4] and [14] are presented in Table 2.

Table 2. Timings for elliptic curve operations [4, 14]. All times in ms.

Type of Operation	ECDSA $GF(2^{163})$	ECDSA $GF(p), p = P-192$ from FIPS 186-2
signature	1.68	0.68
verification	4.97	2.59
general point multiplication	1.68	0.68

Reference [8] describes a fast software implementation of the elliptic curve version of DSA (known as ECDSA). In addition, the paper provides speed comparisons between different public-key schemes. These types of comparisons are scarce in the literature. The authors implemented RSA, DSA, and ECDSA, for both $GF(2^n)$ and $GF(p)$, on a Pentium-Pro 200 MHz-based PC running Windows NT 4.0 and using MSVC 4.2 and maximal optimization. RSA and DSA used a 1024-bit long modulus whereas elliptic curve operations used a 191-bit long modulus instead. Notice that elliptic curve cryptosystems with 191-bit arithmetic is slightly stronger than RSA with 1024-bit operations. Table 3 presents the timings for the signature operation using the four algorithms discussed.

Table 3. Comparison of ECDSA, DSA, and RSA signature operations [8]. All times in ms.

Type of Operation	ECDSA $GF(2^n)$	ECDSA $GF(p)$	RSA	DSA
signature	11.3	6.3	43.3	23.6
verification	60	26	0.65	28.3
general point multiplication	50	21.1	–	–

4 Conclusions

We have introduced the basic concepts, characteristics, and goals of various cryptographic algorithms. We have shown how embedded systems are essential parts of most communications systems and how this makes them especially attractive as a potential platform to implement cryptographic algorithms. Furthermore, although a challenging task, previous implementations of arithmetic intensive cryptographic algorithms seem to indicate that they can achieve acceptable performance on embedded processors and constrained platforms. Thus, it is our view that designing and implementing efficient cryptographic algorithms on embedded systems will continue to be an active research area.

References

1. 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, Germany, 1998. Springer-Verlag.
2. D. V. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.
3. P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume LNCS 263, pages 311–323, Berlin, Germany, August 1986. Springer-Verlag.
4. M. Brown, D. Hankerson, J. López, and A. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In D. Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, volume LNCS 2020, pages 250–265, Berlin, April 2001. Springer-Verlag.
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. Henry Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, volume LNCS 1514, pages 51–65, Berlin, 1998. Springer-Verlag.
7. J. Daemen and V. Rijmen. AES Proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference*, Ventura, California, USA, 1998.
8. E. DeWin, S. Mister, B. Preneel, and M. Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. In J. P. Buhler, editor, *Algorithmic Number Theory: Third International Symposium (ANTS 3)*, volume LNCS 1423, pages 252–266. Springer-Verlag, June 21–25 1998.
9. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.
10. S. R. Dussé and B. S. Kaliski. A Cryptographic Library for the Motorola DSP56000. In I. B. Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume LNCS 473, pages 230–244, Berlin, Germany, May 1990. Springer-Verlag.
11. Federal Information Processing Standards, National Bureau of Standards, U.S. Department of Commerce. *NIST FIPS PUB 46, Data Encryption Standard*, January 15, 1977.
12. B. Gladman. AES Algorithm Efficiency. World Wide Web, 2002. Available at `http://fp.gladman.plus.com/cryptography_technology/aes/index.htm`.
13. 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.
14. 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.

15. 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.
16. K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara. Fast Implemenation of Public-Key Cryptography on a DSP TMS320C6201. In Çetin K. Koç and Christof Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES'99*, volume LNCS 1717, pages 61–72, Berlin, Germany, August 1999. Springer-Verlag.
17. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.
18. Ç. K. Koç, T. Acar, and B. Kaliski, Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, pages 26–33, June 1996.
19. X. Lai and J. L. Massey. Markov Ciphers and Differential Cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume LNCS 547, pages 17–38, Berlin, Germany, 1991. Springer-Verlag.
20. Chae Hoon Lim and Hyo Sun Hwang. Fast Implementation of Elliptic Curve Arithmetic in $GF(p^n)$. In Hideki Imai and Yuliang Zheng, editors, *Third International Workshop on Practice and Theory in Public Key Cryptography — PKC 2000*, volume LNCS 1751, pages 405–421, Berlin, 2000. Springer-Verlag.
21. J. L. Massey and X. Lai. Device for converting a digital block and the use thereof. European Patent, Patent Number 482154, April 29, 1992.
22. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, Florida, USA, 1997.
23. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
24. *IEEE P1363 Standard Specifications for Public Key Cryptography*, November 1999. Last Preliminary Draft.
25. R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
26. B. Schneier. *Applied Cryptography.* John Wiley & Sons Inc., New York, New York, USA, 2nd edition, 1996.
27. W. Stallings. *Cryptography and Network Security.* Prentice Hall, Upper Saddle River, New Jersey, USA, second edition, 1999.
28. U.S. Department of Commerce/National Institute of Standard and Technology. *FIPS 186-2, Digital Signature Standard (DSS)*, February 2000. Available at `http://csrc.nist.gov/encryption`.
29. U.S. Department of Commerce/National Institute of Standard and Technology. *FIPS PUB 197, Specification for the Advanced Encryption Standard (AES)*, November 2001. Available at `http://csrc.nist.gov/encryption/aes`.
30. A. Weimerskirch, C. Paar, and S. Chang Shantz. Elliptic Curve Cryptography on a Palm OS Device. In V. Varadharajan and Y. Mu, editors, *The 6th Australasian Conference on Information Security and Privacy — ACISP 2001*, volume LNCS 2119, pages 502–513, Berlin, 2001. Springer-Verlag.
31. T. Wollinger, M. Wang, J. Guajardo, and C. Paar. How well are high-end DSPs suited for the AES algorithms? In *The Third Advanced Encryption Standard Candidate Conference*, pages 94–105, New York, New York, USA, April 13–14 2000. National Institute of Standards and Technology.
32. A. Woodbury, D. V. Bailey, and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *IFIP CARDIS 2000, Fourth Smart Card Research and Advanced Application Conference*, Bristol, UK, September 20–22 2000. Kluwer.