# Accelerating Cryptographic Protocols: A Review of Theory and Technologies

Antti Hakkala[1,2] and Seppo Virtanen[2]

Turku Centre for Computer Science TUCS[1]
Department of Information Technology[2]
University of Turku
Turku, Finland
Email: {antti.hakkala,seppo.virtanen}@utu.fi

***Abstract*—Modern cryptography applications require significant processing power and resources on computers. To make implementations of these algorithms comply with the rising requirements of speed and throughput of modern applications, the use of instruction set extensions and external cryptography processors has become more and more commonplace. While cryptography algorithms can and do differ significantly in their design and functionality, there are certain algebraic operations which are used throughout all algorithm types. In this paper, we review the fundamental operations that can be found behind several cryptography algorithms currently in use. We examine the need for enhancing the performance of cryptographic protocols and available methods for accelerating the computation of such algorithms. We discuss current methods for accelerating their performance and examine the use of instruction set extensions for cryptography algorithms, particularly the cases where an instruction set can be used for multiple purposes. We conclude that future applications require making these instruction sets as general as possible to support a wide range of algorithms.**

***Keywords-Cryptography; Instruction Set Extensions; Embedded Security; ASIP.***

## I. Introduction

Cryptography is the science—and some say, art [1]—of hiding and securing information. Cryptography has an interesting and far-reaching history dating back thousands of years, but what can be considered modern cryptography began in the 20th century. Driving forces to this development were the advent of computers and the pressure from the second World War, where cryptography and cryptanalysis played a vital role in its' outcome [2][3]. Currently, the whole global communication network is heavily dependent on cryptography and its applications.

As a result of this progress, applications handling personal data, banking information, spatial data and private communications have spread to mobile and embedded devices. This data must be secured at all points in the communication chain. Cryptography is computationally intensive, and thus modern embedded devices that must be able to process encrypted information require different cryptography acceleration methods to function properly in this environment. There are many different possible approaches to solving this problem, and in this paper we consider the problem with focus on Instruction Set Extensions (ISE) for a general-purpose CPU (GPCPU). This approach allows the device to execute certain pre-defined operations more effectively than a software approach. A dedicated hardware solution on silicon would be faster, but these solutions sacrifice flexibility and programmability for speed. Properly designed ISEs can provide concrete improvements to performance with additional programmability added to the system, but before this is possible, the central building blocks from number theory that are required for cryptography algorithms must be identified. If these blocks can be mapped to other, seemingly unrelated functions, the instruction sets can be designed to accommodate even broader functions than cryptography.

In this paper we review and examine the building blocks of cryptography algorithms at different security model abstraction levels. We discuss current methods for accelerating their performance and examine the use of instruction set extensions for cryptography algorithms. We start the discussion by defining the layered security model we use for categorizing security elements in this paper in Section II. In Section III, we proceed to give an introduction to the foundations of ISEs. Then, in Section IV, we proceed to review the predominant types and functions of modern cryptographic algorithms. In Section V, we review and discuss methods for accelerating the processing of cryptographic protocols by examining the mathematical foundations behind cryptographic algorithms. In Section VI, we review proposed solutions for cryptographic instruction set extensions, including co-processor and application-specific instruction-set processor (ASIP) approaches. In Section VII, we suggest some of the interesting future research directions regarding embedded security, cryptographical primitives and ISE design for embedded systems. Our concluding remarks are discussed in Section VIII.

## II. The Pyramid Model of Security

The structure of any information security design can be generalized into the form of a security pyramid [4], divided into five different levels ordered by the level of abstraction, from highest to lowest. At the top abstraction level is the security protocol architecture, which describes the protocols used for secure communications. In the next layer are the different algorithms used to achieve the security specified by

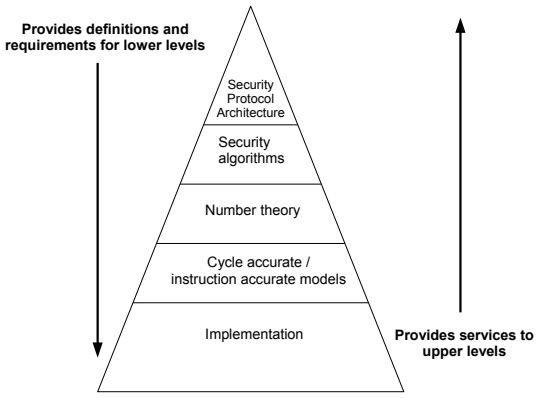


Fig. 1. Security Pyramid [4]

the protocols. These algorithms include the basic signature, encryption and hash algorithms required for cryptography. The third level contains the basic building blocks of the algorithms themselves. These include the number theoretic foundations of the algorithms, and different number representations for speeding up cryptography algorithm implementations. The fourth level contains the platform-independent representations of the algorithms, and the fifth and final level is the actual physical silicon implementation of the algorithms in the protocol architecture.

The security pyramid model can also be seen as a service architecture. The upper levels define requirements (what needs to be done) and general architecture guidelines (how this can be done), and the lower levels provide services upwards by implementing the requirements of the upper level. For example, when the security protocol is chosen, it defines what security algorithms will be required for implementation, but it does not explicitly specify *how* they should be implemented, as long as they meet the requirements. Similarily, the selection of security algorithms leave the designer free to explore actual algorithm implementation options based on the requirements from the upper level (optimization for speed, energy efficiency, side-channel resistance). The instruction accurate level provides the actual number theoretic functionalities of the third level in the form of instruction sets. Finally this continues down to the implementation level, where the requirements and specifications from the upper levels define the actual physical implementation and layout of the device.

The interactions between levels are straightforward when the security pyramid is considered to be separate from the rest of the design process of the whole device the security features are a part of. When the whole process is considered, this model is not accurate enough to represent the problems and relations in security design for devices.

A similar security pyramid for embedded systems has been presented in [5]. It also contains five levels of abstraction, but with little differences. The top level, similarly to the previous model, is the protocol level. The second level is the algorithm level. The third level is the architecture level, which consists of secure design architectures and partitioning to prevent attacks against the systems from software. The fourth level is the microarchitecture level, which contains secure hardware design guidelined by the architecture level decisions. The lowest level is the circuit level, which contains secure physical layer implementations for the chosen microarchitecture. In this model, number theory has been absorbed by the algorithm level to make way to secure design and partitioning, an important issue with secure embedded systems.

What is common for both models is that, as is pointed out in [5], the whole problem of secure system design is not necessarily concentrated on a single level of abstraction. Some problems can be addressed on a single level, but others must be addressed on several levels or the security of the system breaks down. Because adversaries usually have full physical access to an embedded device they are trying to compromise, the security must be sound on all levels on the security pyramid, or the whole system is compromised. This is challenging from a design standpoint, and more research needs to be directed to ensuring built-in system security in all abstraction layers and all phases of the design process.

## III. Introduction to Instruction Set Extensions

Cryptographic algorithms utilize certain algebraic operations, which are often not directly implemented in or supported by the *Instruction Set Architecture* (ISA) of modern digital systems. Cryptography applications require a large amount of computation resources, so identifying and streamlining the calculation of these algebraic operations is essential to good performance on digital computers. This can be implemented with software, but this approach is slow and requires a lot of power. Conversely, pure hardware solutions are very fast and efficient, but lack the flexibility of software. Thus hybrid methods that use both techniques are required. There are several methods available to achieve performance gains, and they are dependent on the overall architecture and purpose of the target system [6]. General purpose embedded processors can be paired up with co-processors, FPGA solutions or hardware accelerators, or specifically designed ASICs or ASIPs can handle the extra computational load. This review is centered on specific processor ISEs used to handle these operations. ISEs expand the available native instructions for a processor, and when an operation is supported by a specific ISE, its' processing is substantially faster than an implementation purely based on software.

ISEs can be mapped to the security pyramid, where they occupy the fourth level, as is seen on Figure I. As ISEs are dependent on the building blocks provided by their respective upper layers, the fundamentals of these operations must be understood so that the best possible choices can be made for the basis of ISE design. These are naturally dependent on other architectural choices, so this review will take into account the most basic and general methods.

When considering ISEs and their implementation, it must be realized that the pressure for effective implementation comes from the upper levels of the security pyramid. At the implementation level, there is not much we can do after

the specifications have been decided and the chip design is finished. Similarly, there is nothing a designer can do about cryptography protocol or specific algorithm design, the designer must work between these layers to provide good service to both upper and lower layers in the security pyramid.

An instruction for a single round of a block cipher can be implemented, and it provides very fast performance for that single block cipher. What happens if the cipher suite needs to be changed by altered application demands? Even though a new device can be brought in to handle the changed requirements, it is not always practical to change the hardware in applications for which the cipher suites can change regarding to the service they provide. If the device is required to handle several different cryptography protocols for different purposes, it is desirable to have as broad a functionality as possible in the single chip without losing performance significantly. Also, the designer must pay attention to related functions that overlap with the security features [4]. This means that the instructions provided for cryptography applications can be used for other purposes as well, if the underlying functions have sufficient similarities, such as using the same permutations or field operations.

## IV. Modern Cryptography Algorithms

Contemporary communication systems are heavily dependent on secure communication of information. Algorithms for secure communications have been developed throughout history, but many of them have been broken by cryptanalysis. Here we focus on the three main groups of cryptography algorithms. When considering an algorithm for cryptography purposes, a designer has a wide range of choices available, as the number of available and secure ciphers is very large and growing constantly. Cryptographic algorithms can be broadly divided into *symmetric* or *private-key*, *asymmetric* or *public-key* ciphers and *cryptographic hash functions*.

The two basic techniques for obfuscating the relationship between plaintext and ciphertext are *confusion* and *diffusion* [7]. Confusion aims to blur the relationship between the plaintext, key and ciphertext, while diffusion eliminates the statistical redundancy inherent in the plaintext and distributes it over the ciphertext. Good ciphers should employ both techniques, and several different operations for both public and private key systems exist for achieving confusion and diffusion. Symmetric cryptography algorithms use the same key for encryption and decryption. Before two communicating parties can begin to exchange encrypted messages, they have to agree on the algorithm to be used and a common key both will use for encryption and decryption. After a key has been agreed upon, both parties use the same key for both encryption and decryption.

In 1976, Whitfield Diffie and Martin Hellman introduced the concept of public-key cryptography [8]. A public key algorithm has a pair of keys, one for encryption (public key), the other for decryption (private key). The public key for encrypting messages to the recipient is published and the private key for decrypting received messages is kept secret. The security of public-key cryptography lies on a difficult mathematical problem that is easy to solve with certain specific information, but next to impossible otherwise. The intractability of *integer factorialization* and *discrete logarithms* in certain groups are the most used problems as the basis of public-key cryptography algorithms. Some problems thought at first to be intractable have been shown to be solvable, thus breaking the cryptosystem built on it.

Hash functions are essential to many cryptographic protocols, because they provide condensed versions of their input in a manner which cannot be reversed. This has several applications in message authentication schemes and integrity checks, among other important functions. An ideal hash function is a one-way function that generates a fixed length *digest* $h(m)$ of an arbitrary length input value $m$. The one-wayness of a function means that evaluating the function is easy, but evaluating its' inverse is not feasible.

## V. Methods and Algorithms for Accelerating Cryptographic Protocols

To effectively realize fast implementations for cryptography algorithms, it is necessary to examine the number theoretical foundations of these algorithms. The following approaches are mainly algorithmical or mathematical approaches to accelerating the calculation of central operations in cryptography. In addition to the fundamental basis of increasing speed of cryptographic calculations, the understanding of these operations also help to give insights into cipher design and requirements.

The techniques examined here address modular arithmetic, point scalar multiplication and operations for symmetric ciphers. They are based on the third level of the security pyramid, and form the basis of instruction set extensions found on the fourth level. As such, they essentially define how fast we are able to perform the required operations for cryptography algorithms in the first place.

### A. Exponentiation and modular arithmetic

Exponentiation is a central arithmetic operation in most algorithms, and consequently the speed of exponentiation significantly affects their performance. Modular arithmetic operates in a residual number system, where all operands are reduced *modulo* an integer. Cryptography combines these two central operations in many ciphers, and because of this, they are critical to the performance of cryptography applications.

Trivial multiplication algorithms have a complexity of $O(n^2)$ [9]. Algorithms such as Karatsuba-Ofman multiplication [10] have a complexity of $O(n^{lg\ 3})$. Efficient exponentiation has many solutions, and a good review of fast methods and their mathematical fundamentals can be found in [11], where addition chains, the binary method, $m$-ary method, different window methods and redundant number systems are considered. It was found in the review that the choice of the multiplication algorithm depends on the situation. Sometimes, as is with Diffie-Hellman key exchange, it is more prudent to use precomputation methods, while in other situations the choice of the group will make all the difference, as is with $\mathbb{F}_{2^n}$

in Elliptic Curve Cryptography (ECC), where it is possible to use different coordinate systems for a substantial advantage. Other number representations for exponentiation have been proposed, such as the Fibonacci $m$-ary representation [12], which is a sparse number representation based on representing numbers as sums of Fibonacci numbers.

The Chinese Remainder Theorem (CRT) is often used to enhance RSA performance, as it can be used to effectively halve the required modulus length by calculating exponentiations $\mod p$ and $\mod q$ instead of $\mod (pq)$. A description of CRT can be found in, for example, [1]. All methods where the modulus is a composite and require modular exponentiation can benefit from using CRT, with some reservations. For example RSA decryption greatly benefits from this, but encryption does not, as the factors of the composite modulus $n = pq$ are required for the CRT, and the security of RSA depends on only the receiver knowing the factors for $n$. The effect of using CRT on RSA results in theoretical decryption speed increase of 4x, and for the implementation in [13], actual speedup is 3.8x.

The use of modular arithmetic requires modular reduction and inversion operations. These are computationally expensive, and methods to improve their performance are well-documented. Montgomery presented a method [14] for accelerating modular exponentiation by transforming the calculations into a new representation where divisions and reductions by $n$ can be performed as divisions and reductions by some arbitrary binary number $2^i$. Given that reductions and divisions by powers of 2 are trivial in digital computers, this greatly increases the speed of modular exponentiation. Although the benefit is lost for minor operations, as the cost of transforming the operands to the Montgomery representation is not free, repeated operations benefit from this approach greatly. Discussion and algorithms for Montgomery reduction and multiplication can be found in, for example, [15]. The Montgomery method is widely implemented in digital systems of all kind where efficient modular reduction is required, and improvements to the original algorithm are widely researched. An algorithm presented in [16] runs faster than Montgomery's, and differs from the Montgomery method by integrating the multiplication and reduction steps. In [17], a modified version of Montgomery multiplication is presented, with a reported reduction in multiplication steps of 26.68% to 38.9% compared to previous improvements.

An alternative method for fast modular reduction [18] was presented by Barrett. It is based on precomputing a single parameter for later use. This precomputation cost can be considered negligible if the modulus does not change during calcualtions. So far, the Montgomery method is more prominent in existing implementations. A modified Barrett implementation was presented in [19]. It is based on modifying the precomputation phase and by implementing a modified Karatsuba-Ofman multiplication algorithm to provide substantial improvement to performance compared to the Montgomery method. When compared on three different platforms, the performance of the modified Barrett algorithm was consistently better than Montgomery's, and increased with larger moduli.

Calculating multiplicative inverses are an important part in any modular arithmetic based cryptosystem. This can be done via Fermat's Little Theorem, which states that $a^{p-1} \equiv 1 \mod p$ and thus the multiplicative inverse of element $a$ is $a^{p-2} = a^{-1} \mod p$. An another way to calculate inverses is the Extended Eucleidian Algorithm, of which a description can be found in, for example, [20].

### B. Elliptic Curve Cryptography

Elliptic Curve cryptosystems are based on elliptic curves over finite fields [21], [22]. A basic introduction to ECC can be found in [23], and a more comprehensive treatment in [20]. While cryptosystems operating in $\mathbb{Z}_p$ use modular exponentiation, the central operation for ECC is point scalar multiplication, where a point $P$ on the elliptic curve is added to itself $k$ times. The reversal of this calculation is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP), and the intractability of this problem in the chosen underlying field is the basis of ECC. Attacks on ECC exist—like RSA, the ECDLP has not been *proven* to be intractable—but they can be alleviated with proper choice of system parameters [20].

Modular arithmetic is present in ECC systems, as the operands are reduced modulo a prime for prime fields $\mathbb{F}_q$, and modulo an irreducible polynomial in extension fields $\mathbb{F}_{2^n}$. Similar $m$-ary techniques, windowing techniques and NAF representations discussed in Section V-A can be used for accelerating such calculations [23]. The coordinate system used to represent the points on an elliptic curve can also be modified to provide faster calculations by reducing or removing the need for inversions, which are costly to compute. Itoh-Tsujii inversion [24], derived from Fermat's Little Theorem, can be used to calculate inversions in both prime and extension fields. The use of repeated squarings to compute inversions in binary fields is discussed in [25]. Also, the choice of both the curve and the underlying field can have a significant effect in speeding up calculations. See [20] for discussion on accelerating ECC systems.

## VI. Instruction Set Extensions for Cryptography

Given the demonstrated complexity of operations in cryptography algorithms and the fact that computers are used by necessity to perform these operations, the optimization of performance for computer systems is essential for any cryptography application. Wu et al. [26] state that for a fast and efficient platform for cryptography applications, the platform must be scalable and it must perform the essential operations required in cryptography efficiently.

ASIPs have been intensively studied already for more than a decade. The aim in ASIP design is to find sequences of general purpose operations in the target application, and group these sequences into a hardware implementation [27], [28], [29], [30]. Often the target application is profiled using tool-based automation to find such code sequences. The resulting recurring command sequences that are chosen for hardware implementation are often quite short, usually less than 10

and often only 2-3 general purpose processor commands of length. In ASIP methodologies often a general purpose processing core is enhanced with hardware execution units for the detected command sequences with the aim that the special hardware units speed up overall application processing speed considerably. For example, in an early study presented in [27] it was determined that this kind of an approach provides a performance increase of no more than 30 % when compared to a general purpose processor designed with equal area and power constraints.

ASIP design methodologies have established their position as an attractive alternative to embedded processor design, especially in the digital signal processing application domain. The use of ASIPs for cryptographic applications has also been studied quite intensively in recent years. For example, in [31], an ASIP for speeding up AES, DES, MD5 and SHA processing is designed. The design process attempts to identify all possible levels of parallelism in the target application both at the instruction level (where detected sequences are 5-10 instructions of length) and at the task level, thus aiming for a better result than in the traditional instruction level profiling. The results showed up to 2.7x cycle time improvement for the target algorithms. In [32], an existing ASIP design platform is used to design an ASIP for AES processing in sensor networks. The optimization includes the design of two new hardware units to speed up the AES MixColumns and SubBytes transformations. In comparison to an unoptimized reference model, the resulting optimized ASIP and its code show cycle count reductions of 33 % and 45 % for encryption and decryption, respectively. In [33], the design space is explored to design an ASIP for cryptographic pairings based on an existing RISC core. Based on the exploration, the key extensions to the core are a multi-cycle scalable Montgomery multiplier and an enhanced memory architecture. The resulting ASIP instances range from a fast ASIP instance suitable to embedded systems to a smaller and slower instance suitable to for example smart card applications.

Instruction set design can also affect the choice of underlying algorithms and methods dramatically. In [34], an instruction set is customized to provide superior multiplication performance with an algorithm clearly inferior to other available options when considered without the instruction set.

The question whether to accelerate public or private key algorithms, or both, is heavily dependent on the target application. For example the effects on cryptography acceleration on SSL is analyzed in [35], where it was found that given very short SSL sessions, public-key traffic for key exchange and setup dominates the total amount of encrypted traffic. In SSL, public-key encryption is used for key exchange and authentication, and symmetric encryption is used for payload encryption. As the amount of traffic increases, symmetric encryption becomes increasingly dominant in the overall traffic. If a server has to handle multiple distinct encrypted connections for a prolonged amount of time, public-key encryption traffic becomes increasingly important. For a client which has to handle only one connection, symmetric encryption becomes more and more meaningful as the amount of traffic grows.

### A. Synergy benefits from instruction sets

In [36], a method for accelerating AES with an instruction set designed for ECC is presented. It exploits the fact that AES uses finite field arithmetic in the field GF($2^8$), and thus ECC accelerators for fields of the form GF($2^m$) can be used also for accelerating AES. The instruction set contained an instruction for binary polynomial multiplication, and this was used to enhance performance. The authors report a speed increase of up to 25% for AES when utilizing instructions for ECC in a SPARC V8 LEON-2 processor. Although the speed increase is not on par with dedicated AES implementations on FPGA:s, for example, it shows how it is possible to gain performance from exploiting the common building blocks of cryptographic algorithms.

The effects of the Intel AES instruction set [37][38] for performance on SHA-3 hash function candidates is analyzed in [39]. A significant majority of the SHA-3 candidates are directly based on or utilize the AES round operation, so direct performance improvements on the operation of such functions is intuitive. The analysis is based on simulations on the information available of the Intel AES ISE. As the instructions perform one complete round of AES, algorithms which do not directly comply to the round function do not gain performance benefits from these instructions, even though they have similar building blocks. This highlights the disadvantages of very specialized instruction sets; it is always a tradeoff between flexibility and performance, as the instructions perform a complete round, any deviation from the standard round causes the instructions to be unusable, even though the underlying operations are the same.

It would be reasonable to assume that any methods for accelerating AES performance would also have similar impact on the performance of the SHA-3 candidates that are based on AES. Especially instructions for ECC are interesting in this regard, as it has been previously noted that they can be used to accelerate AES, and thus would be by extension capable of doing the same to some of the SHA-3 candidates. To the best of our knowledge, no research in this direction has yet been published, though.

## VII. Open Issues and Future Work

Because the capability of devices to execute a wide variety of cryptographic protocols effectively is directly tied to the security and privacy aspects of future communication systems, the creative use of instruction sets to enhance the performance of cryptography operations is an interesting research topic. This is especially true for devices which must provide a wide variety of functions using limited processing capabilities.

When considering the larger scale of secure system design, overcoming the information security challenges of embedded communication systems of the future requires continuous fundamental and cross-disciplinary research. As an enabling factor to approaching these challenges we see the need for a paradigm shift: instead of treating security as an incremental

feature of embedded communication systems, research efforts should focus on pre-empting the emergence of information security threats and vulnerabilities in future embedded communication systems. This requires incorporating information security as an integral part into the design process of such systems and their subsystems. Such a paradigm shift raises challenging novel research problems in both hardware and software development as well as related processes. Focusing research efforts on security-enabled hardware/software codesign solutions and systematic approaches for designing secure embedded communication systems would be an important first step in this direction.

In our view, the initial step towards more robust and secure system design could be integration of the design-time "building security in" philosophy [40] from software engineering to the run-time ability of adapting to new application-domain specific applications, as conceptualized in Software-Defined Radio (SDR) [41] and Open Wireless Architecture (OWA) [42], in the overall embedded communication system design process [43]. Bringing these research directions together to design secure embedded communication systems would be an extremely interesting approach. Some aspects of the research required to accomplish this are outlined in [44]. A thorough overview of design challenges for future embedded communication systems including their security is given in [45].

Security interacts directly with the price, power consumption, testability, performance and reliability factors of the design process [23]. This poses steep requirements to system design processes and methodologies. To achieve these goals, the requirements for such design processes should be explicitly defined and examined. Building security into embedded systems has effects in all vital areas of system design, and this is an interesting direction for future research.

## VIII. Conclusion

Effective methods for accelerating different cryptosystems have been presented in literature. It remains as the designers dilemma to select the best of these methods which are applicable to the current problem and use them effectively to accelerate the required operations used in the design. The design process of an embedded system should incorporate security features to the core of the process, thus moving towards design methods where security is already built in. Security models such as the security pyramid are efficient in pointing out that security is not a feature that can be added in as an afterthought, but is a complex problem that involves all layers of abstraction.

Implementing efficient methods for essential algebraic and other operations can make a large difference in cryptography application performance. The choice of these operations depend on the nature of the cryptography application, but as some operations are central to many different algorithms, synergy benefits from implementing a broad base of operations are worth consideration for implementations where flexibility is a design parameter.

Instruction set extensions for general purpose processors have been demonstrated to be very efficient in improving cryptography application performance by providing instructions for particular operations that are ill-suited for execution in traditional CPUs. Performance increases have been studied and have been observed to be even an order of magnitude faster than implementations in pure software.

Instruction sets for particular algorithms are naturally faster in performance than generalized approaches, but fixed word lengths and logic cause loss of flexibility in choosing different algorithms for implementation. Instruction sets that target some general problematic operation that is shared between different ciphers are naturally more flexible, while still providing significant performance increase compared to pure software solutions. If the goal is to create a flexible solution for cryptography acceleration, these generalized instruction sets are a very tempting choice over specific instruction sets.

## References


[1] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source code in C, 2nd edition*. John Wiley & Sons, 1996.

[2] D. Kahn. *The Codebreakers*. Macmillan Publishing Company, New York, 1967.

[3] S. Singh. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography, 1st edition*. Doubleday, New York, NY, USA, 1999.

[4] P. Schaumont and I. Verbauwhede. A reconfiguration hierarchy for elliptic curve cryptography. In *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, 2001.*, volume 1, pages 449–453. IEEE, 2002.

[5] D.D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede. Securing embedded systems. *Security and Privacy, IEEE*, 4(2):40 –49, March-April 2006.

[6] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Design Automation Conference, San Diego, CA, USA, June 07 - 11*. ACM, New York, 2004.

[7] C.E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28:656–715, Oct. 1949.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[9] D.E. Knuth. *The art of computer programming. Volume 2. Seminumerical algorithms, Third Edition*. Addison-Wesley Reading, MA, 1998.

[10] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet Physics Doklady*, volume 7, page 595, 1963.

[11] D.M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129 – 146, 1998.

[12] S.T. Klein. Should one always use repeated squaring for modular exponentiation? *Information Processing Letters*, 106(6):232–237, 2008.

[13] K. Hansen, T. Larsen, and K. Olsen. On the Efficiency of Fast RSA Variants in Modern Mobile Phones. *International Journal of Computer Science and Information Security*, 6(3):136–140, 2009.

[14] P.L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[15] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC, 1997.

[16] S.M. Hong, S.Y. Oh, and H. Yoon. New modular multiplication algorithms for fast modular exponentiation. In *Advances in Cryptology—EUROCRYPT96*, pages 166–177. Springer, 1996.

[17] C.-L. Wu. An efficient common-multiplicand-multiplication method to the montgomery algorithm for speeding up exponentiation. *Information Sciences*, 179(4):410 – 421, 2009.

[18] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology—CRYPTO86*, pages 311–323. Springer, 1987.

[19] W. Hasenplaugh, G. Gaubatz, and V. Gopal. Fast modular reduction. *18th IEEE Symposium on Computer Arithmetic (ARITH'07)*, 2007.

[20] D.R. Hankerson, S.A. Vanstone, and A.J. Menezes. *Guide to elliptic curve cryptography*. Springer-Verlag New York Inc., 2004.

[21] V. Miller. Use of elliptic curves in cryptography. *Proceedings of the Advances in Cryptology—CRYPTO'85*, 1986.

[22] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[23] C.H. Gebotys. *Security in Embedded Devices*. Springer Verlag, 2010.

[24] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF($2^m$) using normal bases. *Information and Computation*, 78(3):171–177, 1988.

[25] K.U. Järvinen. On repeated squarings in binary fields. In *Proceedings of the 16th International Workshop on Selected Areas in Cryptography, SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 331–349. Springer-Verlag, 2009.

[26] L. Wu, C. Weaver, and T. Austin. CryptoManiac: a fast flexible architecture for secure communication. *ACM SIGARCH Computer Architecture News*, 29(2):110–119, 2001.

[27] M. Arnold and H. Corporaal. Designing domain specific processors. In *Proceedings of the 9th International Symposium on Hardware/Software Codesign (CODES'01)*, pages 61–66, 2001.

[28] A. Both, B. Biermann, R. Lerch, Y. Manoli, and K. Sievert. Hardware-software-codesign of application specific microcontrollers with the ASM environment. In *Proceedings of the Conference on European Design Automation*, pages 72–76, Grenoble, France, September 1994.

[29] T.V.K. Gupta, P. Sharma, M. Balakrishnan, and S. Malik. Processor evaluation in an embedded systems design environment. In *Proceedings of the 13th International Conferenence on VLSI Design*, pages 98–103, Calcutta, India, January 2000.

[30] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man. Instruction set definition and instruction selection for ASIP. In *Proceedings of the Seventh International Symposium on High-Level Synthesis*, pages 11–16, Niagara-on-the-lake, Canada, May 1994.

[31] D. Montgomery and A. Akoglu. Methodology and Toolset for ASIP Design and Development Targeting Cryptography-Based Applications. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 365–370, Montreal, Canada, July 2007.

[32] N. Suarez, G.M. Callico, R. Sarmiento, O. Santana, and A.A. Abbo. Processor customization for software implementation of the AES algorithm for wireless sensor networks. In *Proceedings of the 19th International Workshop on Power and Timing Modeling, Optimization and Simulation, LNCS 5953*, pages 326–335, Delft, The Netherlands, September 2009.

[33] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar. Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In *Proceedings of 11th International Workshop on Cryptographic Hardware and Embedded Systems, LNCS 5747*, pages 254–271, 2009.

[34] J. Großschädl, P. Ienne, L. Pozzi, S. Tillich, and A.K. Verma. Combining algorithm exploration with instruction set design: a case study in elliptic curve cryptography. In *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, volume 1, pages 1–6. IEEE, 2006.

[35] J. Burke, J. McDonald, and T. Austin. Architectural support for fast symmetric-key cryptography. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 178–189. ACM, 2000.

[36] S. Tillich and J. Großschädl. Accelerating AES using instruction set extensions for elliptic curve cryptography. *Computational Science and Its Applications–ICCSA 2005*, pages 665–675, 2005.

[37] S. Gueron. Intel® Advanced Encryption Standard (AES) Instructions Set. White paper, Intel Corporation, http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/, January 2010.

[38] S. Gueron. Intel's New AES Instructions for Enhanced Performance and Security. In *Fast Software Encryption: 16th International Workshop, FSE 2009 Leuven, Belgium, February 22-25, 2009*.

[39] R. Benadjila, O. Billet, S. Gueron, and M.J. Robshaw. The Intel AES Instructions Set and the SHA-3 Candidates. In *ASIACRYPT '09: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security*, pages 162–178. Springer-Verlag, 2009.

[40] G. McGraw. *Software Security: Building Security In*. Addison-Wesley, 2006.

[41] W.H.W. Tuttlebee. Software-defined radio: facets of a developing technology. *Personal Communications, IEEE*, 6(2):38–44, Apr. 1999.

[42] W.W. Lu. Open wireless architecture and enhanced performance. *Communications Magazine, IEEE*, 41(6):106 – 107, June 2003.

[43] D.D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Verlag, 2009.

[44] J. Björkqvist and S. Virtanen. Convergence of hardware and software in platforms for radio technologies. *Communications Magazine, IEEE*, 44(11):52 –57, Nov. 2006.

[45] J. Isoaho, S. Virtanen, and J. Plosila. Current challenges in embedded communication systems. *International Journal of Embedded and Real-Time Communication Systems*, 1(1):1–21, 2010.