Secure and Efficient Elliptic Curve Cryptography for Embedded Environments

by

Erich Wenger

A PhD Thesis
Presented to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Prof. Dr. Karl Christian Posch (Graz University of Technology, Austria)
Prof. Dr. Lejla Batina (Radboud University Nijmegen, The Netherlands)

October 2013

Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

Abstract

Radio-frequency identification (RFID), wireless sensor networks (WSN), and embedded systems are three of the most pressing research topics at the present time. Those technologies pose huge potentials for society and interesting requirements for designers. Such embedded devices must be inexpensive, small, power-efficient, energy-efficient, and/or sufficiently fast, while providing sufficient functionality for high-level applications. Especially cryptography is becoming one of the key enablers for many applications. However, in particular public-key cryptography is very demanding and requires application-specific optimizations for practical feasibility.

In this work, we focus on the practical realization and evaluation of elliptic curve cryptography (ECC) for embedded systems. We consider the resource restrictions which are imposed by the applications in the same way as practical physical attacks. Already during the conceptual design of our ECC implementations, power analysis and fault analysis attacks are considered. Practical evaluations are performed to validate our security concepts.

This thesis contains two parts. In the first part, elliptic curve cryptography is discussed from a high-level perspective and a side-channel secured scalar multiplication method is derived. In the second part, several of our papers are given without modification from their original publications. Those papers have been (co)authored by the author and anonymously reviewed, accepted, and presented at international conferences.

Acknowledgements

I would like to thank all the people who supported and challenged me during the last three years. Especially, I want to thank my family and my fiancee Violetta Krasnici. Without her mental support, this thesis would not have been possible. With her on my side, no mountain is too high to be climbed.

Further, I want to thank my supervisors Jörn-Marc Schmidt, Karl Christian Posch, and Manfred Aigner, who gave me the freedom to pursue any research, I was interested in, and never were lost for words when I needed a helping hand. My freedom can be traced back to the fact that the duties, I was assigned to by Jörn-Marc Schmidt, matched my interests, which is a sign for exceptional leadership. I will never forget Karl Christian Posch's exceptional guidance during the course IT-Security[1] and his challenging opinions, which broadened my horizon. I also want to thank Manfred Aigner for trusting in my skills when he hired me and for giving me spiritual guidance during the tasting of local hop juice.

No thesis would be possible without the right social environment, in particular my colleagues from the SEnSE group, Alexander Szekely, Florian Mendel, Hannes Groß, Marcel Medwed, Maria Eichelseder, Mario Kirschbaum, Mario Lamberger, Martin Feldhofer, Martin Schläffer, Michael Hutter, Raphael Spreitzer, Sandra Dominikus, Thomas Korak, Thomas Plos, and Tomislav Nad, with some of whom I also coauthored several papers. Additionally, I also want to thank my other coauthors Christian Pendl, Johannes Feichtner, Johann Großschädl, Mario Werner, Markus Pelnar, Norbert Felber, Thomas Baier, Thomas Unterluggauer, and Zhe Liu. Without them, this thesis would not have been possible.

Finally, I want to thank my assessor Lejla Batina for many valuable comments and taking the time to travel to Graz.

[1] IT-Security is a university course, I had the pleasure to lecture.

Contents

Part I

Elliptic Curve Cryptography from a High-Level Perspective

Chapter 1

Introduction

It is of upmost importance to work on the toughest challenge within one's scope of research. As our research deals with the implementation of cryptography for extremely constrained devices, the challenge clearly is to implement asymmetric, especially elliptic curve cryptography, as efficiently as possible. Within the big picture, elliptic curve cryptography is just a puzzle piece. However, it is one of the largest and most flexible puzzle pieces. With its huge design space, there is a magnitude of design options to trim the design to a specific set of needs. In this chapter, we revisit the main contributions of some of our papers and elaborate on the significance of our research.

1.1 Main Contributions

Our papers deal with software design, hardware design, and hardware/software co-design of ECC implementations. In regard to software implementations, it is important to highlight [102, 103, 104, 105, 110, 111, 61]. Within those papers, we did assembly-optimized software implementations for some of the most used embedded microprocessors. In particular, our focus was on the megaAVR series [7] by Atmel, the MSP430 series [95] by Texas Instruments and the Cortex-M0+ family [6] by ARM. The goals of our implementations were always rather similar: side-channel hardened implementations that require a minimum amount of memory and are as fast as possible (security over size over speed). Therefore, we hardly broke speed records, but we published some of the smallest implementations. Implementations that are practically more relevant than many of the related implementations. In [61], we layed the foundation for the fastest multi-precision integer multiplication formulas for an ATmega microprocessor. In [105], we even evaluated the latest non-standardized

Edwards [30], Montgomery [82], and GLV [38] curves and compared them with standardized elliptic curves.

We also worked on several papers on hardware/software co-design. In [102], we equipped our clone (JAAVR) of the popular ATmega128 with different, highly dedicated instruction-set extensions. Using these special instructions, we were able to improve the ECC performance up to 39-fold and improved the suitability of microprocessor-based designs for RFID applications. In [103], we investigated an openMSP430-based design regarding its potentials for wireless sensor networks. By equipping the openMSP430 with a dedicated ECC co-processor, its energy requirements dropped by up to a factor of 36. However, more interesting for microchip manufacturers is the drop-in concept in which you drop a special piece of hardware right between the CPU and the memory. Thus, neither the CPU nor the memory have to be modified. And the most important fact about the drop-in concept is that it only requires 4 kGE of chip area, while it nearly delivers stand-alone-ECC-processor-like performance.

Our smallest openMSP430-based design is 14 kGE in size. If you decide to go for a full-custom microprocessor, just designed for ECC, Neptun [101] can significantly reduce your area footprint. In [107], we were able to implement elliptic curve cryptography, while only needing 9 kGE of hardware. This is one of the smallest ECC implementations ever published. Further, in [108], we utilized Neptun to make a fair comparison between prime-field and binary-field based elliptic curve cryptography. It turns out that when side-channel attacks are considered, and Montgomery ladders are implemented, binary fields outperform prime fields by factors of 3.3 in runtime and 3.9 in energy. At the same time, the binary field based ECC design is around 20 % smaller.

Additionally, we did not only implement ECC, but also attacked a design regarding its side-channel vulnerabilities. In [109], we analyzed a protected ECC implementation, suitable for RFID tags, and were able to recover the secret key using various attack approaches. Based on this paper, future ECC designs must consider certain power-analysis attacks that have not been published before.

To emphasize the significance of our research, it is important to ask the question, why asymmetric cryptography, in particular elliptic curve cryptography is necessary in the first place. In order to answer this question, we start at the most fundamental problem, the key distribution problem.

1.2 Key Distribution Problem

The key distribution problem describes a scenario in which n users want to interact with each other using a symmetric cryptographic primitive. Unfortunately, for that to work, each user has to store $n-1$ keys, which brings the system to a total of $\frac{n^2-n}{2}$ key pairs. In practical terms, e.g., if a million users are within a network,

every user has to store 999,999 secret keys. But this number is small compared to 499,999,500,000, which is the total number of keys needed. However, storing 16 megabytes of keys is no problem for current desktop computers and mobile phones. Nevertheless, it is a problem for embedded devices, such as wireless sensor nodes or RFID tags which only possess a few kilobits of memory. There are three approaches to deal with this dilemma. One is to completely avoid cryptography, another is to use key distribution centers, and a third approach is to use asymmetric cryptography.

So, the most important question is whether we need cryptography in the first place? And the answer clearly is: Yes, we do. Achieving common practical goals, such as confidentiality, integrity, or privacy is infeasible without cryptography. Without cryptography, it would not be possible to do secure online banking, to do secure online shopping, or to confidentially exchange electronic mail. However, cryptography poses certain requirements to the user, the applications, and the hardware. About ten years ago it was believed that cryptography is not suitable for certain embedded applications. However, in 2004, Feldhofer *et al.* [35] wrote a landmark paper in which they proofed that cryptography is indeed suitable for RFID tags.

The conclusion is that cryptography is an unavoidable necessity, but how can we solve the problem of distributing keys? A solution which only uses symmetric cryptography is based on key distribution centers. For this solution every user only stores a single secret key, which she only shares with the key distribution center. The key distribution center stores keys for every user in the network. So, if two users want to communicate with each other, they ask the key distribution center for help[1]. Protocols based on this principle work well. However, these protocols are subject to several shortcomings: the key distribution center needs to be contacted for every session; the key distribution center poses a single point of failure; secure channels are needed during the initialization. Nevertheless, the biggest problem is that companies do not want to share key distribution centers, because they do not trust each other. So when, e.g., an RFID tag changes owners, changing the key distribution center is a troublesome task. In order to avoid such complex protocols, asymmetric cryptography could be used instead.

Asymmetric cryptography is at its foundation more complex than symmetric cryptography, as it splits keys into key pairs. A key pair consists of a public key and a private key. While the private key is only known to the user or the device, the public key can be known to everybody[2]. With this concept, it is possible to utilize protocols that are simply not feasible using symmetric cryptography. In this sense, the most important protocol is the Diffie-Hellman key-exchange algorithm [27]. With this fairly simple scheme it is possible that two users

[1]The key distribution center generates a random number, a random key, separately encrypts this random number with the keys of the two users, and sends the encrypted key to the two users. The users decrypt the key and use it for the following communication.

[2]Except in privacy preserving protocols, in which designers want to circumvent the fact that public keys can be used as unique identifiers.

generate a shared secret key via an insecure channel and therefore implicitly solve the key distribution problem. However, and this is a capital HOWEVER, this is not the full picture. Practically speaking, one needs to be aware of the huge performance difference between public-key and symmetric-key cryptography and the need of public-key infrastructures to avoid man-in-the-middle attacks.

There are big differences between the public-key schemes currently used. The most common and standardized public-key schemes are either based on RSA [90], ElGamal [31], or elliptic curve cryptography (ECC) [70, 80]. The big advantage of the RSA and ElGamal schemes in comparison to ECC is that they are older, therefore have been used and investigated for longer, and that they are simpler to implement. For RSA, all you have to implement are a multiplication and an exponentiation operation. For ECC, it is necessary to implement a finite-field multiplication, an addition, a subtraction, an inversion, as well as point arithmetics. However, the main drawback of RSA and ElGamal in comparison to ECC is the requirement of larger keys. According to NIST [12] and ECRYPT II [9], nicely visualized at `www.keylength.com` [26], an 160-bit elliptic curve has approximately the same security level as 1024-bit RSA, namely 80 bits. At the 128-bit security level, which is considered to be secure for at least the next 20 years[3], 256-bit ECC is equally difficult to attack as 3072-bit to 4096-bit RSA. This difference in required parameter sizes has serious implications for ECC and RSA implementations. For instance, in Section 1.5, we conclude that ECC is 14–90 times more efficient than RSA.

So, elliptic curve cryptography is the de-facto standard public-key algorithm for embedded systems. However, it still is sometimes (cf. [104]) more than 10,000 times slower than most of the comparable symmetric primitives. In a system that comprises of symmetric and asymmetric primitives, you should spend your precious development time on the most expensive component of the system (Amdahl's Law). And in our case, the most expensive component is elliptic curve cryptography. Therefore, any research which improves the runtime, the area footprint, the power consumption, or the energy consumption of an ECC implementation automatically improves the efficiency of the overall system.

In the remainder of this chapter, we review elliptic curve cryptography within the bigger picture. To ease readability, we first make an assumption regarding the background knowledge of the potential reader in Section 1.3. However, in Section 1.4 a review of elliptic curve cryptography from an top-level view is performed. Therefore, it is also necessary to understand the practical differences between RSA, ECC, and AES, which are reviewed in Section 1.5. Based on that comparison, we derive a premise in Section 1.6, on which we based our research in the last few years. Section 1.7 outlines the rest of the thesis.

[3]The general assumption is that no new algorithm is found that solves the integer factorization problem or the elliptic curve discrete logarithm problem more efficiently.

1.3 Assumptions on Prior Knowledge

After this rather basic introduction on the importance of elliptic curve cryptography, one usually introduces the building principles of elliptic curve cryptography. However, nowadays an introduction to elliptic curve cryptography is already taught in university courses, i.e. IT-Security. Therefore it is safe to assume that the potential reader of this text already has a basic knowledge of cryptography and is familiar with the building principles of elliptic curve cryptography. Additionally, one of the most excellent and most thorough introductions on elliptic curve cryptography has already been written by Hankerson, Menezes, and Vanstone [48]. In their "Guide to Elliptic Curve Cryptography" they give a thorough overview on finite-field arithmetic, elliptic-curve arithmetic, cryptographic protocols, and implementation issues. For more advanced literature we recommend Cohen *et al.*'s [22] "Handbook of Elliptic and Hyperelliptic Curve Cryptography".

1.4 Elliptic Curve Cryptography within the Bigger Picture

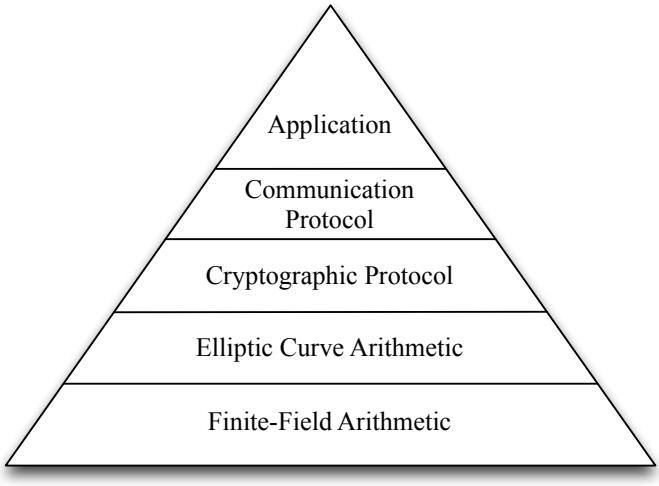

Figure 1.1: Hierarchic visualization of the dependencies between the application and an elliptic curve implementation.

It is important to review elliptic curve cryptography within the bigger picture. This is done in two different ways. In Chapter 2, elliptic curve cryptography is reviewed from the standpoint of embedded applications, whereas in the following, a hierarchical approach is taken. Common practice is to depict the bigger picture

in a pyramid (e.g., Hankerson *et al.* [48, page 226]). In Figure 1.1, an extended version of such a hierarchical chart is shown. It includes the application, the communication protocol, the cryptographic protocol, the elliptic curve point arithmetic, the finite-field arithmetic, as well as the integer and polynomial arithmetic. In the following, the six layers are elaborated in more detail:

Application. Usually, the objective is not a stand-alone cryptographic primitive. The goal always is to utilize a cryptographic primitive within a protocol for a specific application. Therefore, a designer who is responsible for a cryptographic library always has to be aware of the requirements which come from the designated application for which cryptography is used. Therefore, in Chapter 2 we perform a detailed review of the applications which are within the scope of this thesis.

Communication Protocol. The design and specification of a protocol for a certain application with certain use cases is one of the most fundamental tasks a system designer must get right. The application in connection with the communication protocol defines the requirements of the actual software or hardware design. Therefore, the definition of the high-level protocol has the biggest influence on the resulting implementation. Especially, the fact whether public-key-based or secret-key-based protocols are needed has a huge impact. In Section 1.5, we raise the awareness of the practical differences between symmetric and asymmetric primitives. The resulting message can be summed up in: "If it is possible to avoid public-key cryptography, then avoid it."

Cryptographic Protocol. When one defines a communication protocol, one uses cryptographic primitives and cryptographic protocols. The most important protocols within the scope of this thesis are the Elliptic Curve Digital Signature Algorithm (ECDSA) and the elliptic curve Diffie-Hellman (ECDH) key-exchange algorithm. ECDSA is standardized within ANSI X9.62 [5], FIPS 186-3 [85], IEEE 1363-2000 [62], and ISO/IEC 15946-2 [65] and therefore is the algorithm when user or entity authentication is needed. The elliptic curve Diffie-Hellman key exchange can be used to collaboratively derive a symmetric key over an insecure channel. Section 3.2 is dedicated to those protocols.

Elliptic Curve Arithmetic. The foundation of ECDSA and ECDH is the scalar multiplication, $Q \leftarrow kP$, in which the scalar k is multiplied with the point P to derive a point Q. This rather simple-looking operation is the most time-consuming part of all ECC-based protocols and therefore must be implemented with special care. Additionally, the applications that are within the scope of this PhD thesis (cf. Chapter 2) are vulnerable to practical power-analysis and practical fault attacks. Therefore, the scalar multiplication methodology must be implemented in a way such that the ECDSA and ECDH algorithm are practically secure and sufficiently fast,

small, power-efficient, and energy-efficient for the designated application. Chapter 3 elaborates different design options for Elliptic Curve scalar multiplications.

Finite-Field Arithmetic. Elliptic-curve point arithmetic is based on finite-field arithmetic, which is based on integer and polynomial arithmetic. The same requirements that apply to the scalar multiplication method also apply to the finite-field arithmetic: Provide side-channel security and be sufficiently fast. Usually more than 90 percent of the runtime of a scalar multiplication is due to the runtime of the finite-field multiplication. Hence, getting the finite-field multiplication "right" is of great importance. Naturally, the "right" finite-field multiplication algorithm depends on the design goals which are given by the application.

To summarize, the design goals given by the applications affect the design strategies used within the underlying layers which handle elliptic curve and finite-field arithmetic. On the other hand, the application is crucially dependent on the underlying layers. Therefore, the conclusion must be that elliptic curve cryptography is not an off-the-shelf commodity. Elliptic curve cryptography needs to be hand-crafted for the unique requirements posed by the application. In order to illustrate the huge difference in performance between asymmetric and symmetric ciphers the following section compares RSA, ECC, and AES.

1.5 RSA versus ECC versus AES

While RSA and ECC are two of the most important public-key algorithms, AES is the most important symmetric-key block cipher. The Advanced Encryption Standard (AES) [84] was established by the National Institute of Standards and Technology after a five-year competition which was won by Rijndael [25]. AES works on blocks of 128-bit of data, provides security levels of 128–256-bit, and reached such an importance that, e.g., Intel even introduced instructions [63] dedicated to improve the performance of AES.

In the following we review the software characteristics of RSA, ECC, and AES, and the hardware characteristics of ECC and AES.

Software Characteristics

As the focus of this work are embedded, light-weight applications, we exemplarily use the Atmel megaAVR architecture [7] as reference platform. Those small 8-bit RISC microprocessors are already used in a wide range of embedded applications and therefore are a good representative to compare the performance of RSA, ECC, and AES.

In 2004, Gura *et al.* [46] gave assembly-optimized implementations for RSA and ECC. Their results show that a private-key 1024-bit RSA exponentiation (using the Chinese remainder theorem for speedup) takes about 13.6 times longer than an equivalently secure scalar multiplication over an 160-bit elliptic curve. At the same time, the ECC implementation only needs a third of data memory. If a larger security level of, e.g. 128-bit, is required, the difference between RSA and ECC is even more significant. As both the runtimes for RSA and ECC scale cubicly with their respective parameter sizes, an approximation gives that at the 128-bit security level RSA is about $\frac{(3072/1024)^3}{(256/160)^3} \times 13.6 = 89.6$ times slower than ECC.

As ECC easily outperforms RSA, only the public-key algorithm ECC is compared with the symmetric-key cipher AES. This comparison is done at the 128-bit security level and is based on our own results [104]. Our latest scalar multiplication implementation over the standardized `secp256r1` [18] elliptic curve for an ATmega processor needs 34.9 million cycles. Our assembly-optimized implementation of AES-128 [104] needs 3,300 cycles for encryption. This means that a single public-key operation takes 10,575 times longer than a comparable symmetric-key operation.

However, one may argue that Rijndael [25] was specifically designed to give good performance on 8-bit processors. Therefore, the following comparison in hardware should complete the picture.

Hardware Characteristics

Again, we are not interested in fast implementations, but in small implementations. Feldhofer *et al.* [35], Hämäläinen *et al.* [47], and Moradi *et al.* [83] presented three of the smallest AES hardware implementations. Currently, the smallest AES hardware implementation by Hämäläinen *et al.* [47] requires only 3,100 GE[4] in size. If the application allows to use AES in encryption mode only, Moradi *et al.* [83] reduced the minimum AES size to 2,400 GE. As the smallest ECC implementations are all at the 80-bit security level, it is important to have also a reference implementation at the 80-bit security level. One of the most popular ciphers is PRESENT-80, by Bogdanov *et al.* [16]. This block cipher offers a security level of 80-bit while requiring only 32 cycles and 1,507 GE for encryption.

Two of the smallest ECC implementations in hardware are done by Wenger *et al.* [106] and Lee *et al.* [75]. Wenger *et al.* takes advantage of area-efficient RAM macros to achieve a size of 8,958 GE at the 80-bit security level. Without RAM macros, Lee *et al.*'s area-optimized implementation needs 11,904 GE. Thus, in terms of area their implementations are 2.9–7.9 times larger than the referenced AES and PRESENT implementations. However, in terms of speed the differences

[4]GE stands for Gate Equivalent, which is a technology-independent measurement for area. Note that a gate equivalent highly depends on the size of NAND gates and therefore designers should ideally always make comparisons using the same technology.

Table 1.1: Light-weight AES, PRESENT, and ECC implementations.

Implementation	**Cipher**	**Runtime**[a] [Cycles]	**Chip Area** [GE]
Feldhofer *et al.* [35]	AES-128	1,016	3,595
Hämäläinen *et al.* [47]	AES-128	160	3,100
Moradi *et al.* [83]	AES-128[b]	226	2,400
Bogdanov *et al.* [16]	PRESENT-80	32	1,507
Wenger *et al.* [106]	`sect163r2`	285,000	8,958
Lee *et al.* [75]	`sect163r2`	296,000	11,904

[a]Encryption runtimes of AES and PRESENT. Scalar multiplication runtimes of ECC.
[b]Encryption only.

are much larger. Comparing Bogdanov *et al.*'s PRESENT-80 implementation with Wenger *et al.*'s ECC implementation shows an 8,900 fold difference in performance. If we scale Wenger *et al.*'s multi-precision ECC implementation of a scalar multiplication to the 128-bit security level, we would get an excruciating runtime of $(\frac{256}{163})^3 \times 285{,}000 \approx 1{,}104{,}083$ cycles. Comparing this number with Hämäläinen *et al.*'s runtime gives a factor of 6,900 in terms of speed. Table 1.1 gives an overview of the presented results.

One important detail that was omitted in this comparison is that AES usually requires costly countermeasures against differential power analysis (DPA) attacks. Moradi *et al.*'s [83] threshold implementation which incorporates DPA countermeasures increases the size of their AES implementation from 2,400 GE to 11,031 GE. As ECC is usually used in settings in which ECC is not vulnerable to differential power analysis attacks, ECC does not need such countermeasures. Although this fact brightens the dimmed performance numbers of ECC, it should not distract from the huge difference in runtime.

Conclusion

No matter which platform is taken as reference, AES is around four magnitudes faster than ECC. And this huge disparity between AES and ECC brings us to the following premise.

1.6 The Premise for Using ECC

Based on those performance values, the premise is to avoid elliptic curve cryptography whenever possible. Symmetric ciphers are faster, smaller, more energy-efficient, and more power-efficient. However, for the rest of this thesis, we assume that both the application and protocol designers came to the conclusion

that public-key cryptography is required. As already discussed above, without public-key cryptography, many protocols are infeasible.

1.7 Thesis Outline

This thesis is split into two parts. In the first part, we review elliptic curve cryptography from a high-level perspective. In the second part, a list of publications as well as a digest of some of the most representative papers is given.

Chapter 2 highlights the requirements imposed by common embedded applications. In detail, the demands of RFID and wireless sensor nodes on practical ECC implementations are discussed. Those prerequisites were considered during the work on of the papers given in Part II.

Chapter 3 reviews the protocol layer and the ECC point arithmetic layer. Especially the danger of practical implementation attacks on commonly used protocols and the underlying scalar multiplication method are discussed. Attacks that cannot be handled on the protocol layer must be handled within the scalar multiplication algorithm. We give an algorithm which incorporates many countermeasures against practical attacks and analyze its vulnerability against the latest implementation attacks.

An excerpt of some of the most interesting future research topics are discussed in Chapter 4.

In the second part of this thesis a list of publications and some of our most representative papers are given.

Chapter 2

Requirements of Applications

Currently, the most active fields of research which require elliptic curve cryptography come with very scarce resources. Therefore one needs to utilize the full spectrum of design options in order to meet the specification. As the specification differs significantly from application to application, this chapter elaborates on the different requirements of embedded systems in general, and in particular wireless sensor networks and RFID systems.

2.1 Embedded Systems

According to Steve Heath [50], an embedded system is a "microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end-user". This definition of embedded systems is rather broad. More concrete is the classification by Koopman [73] who files embedded systems within four categories. While "signal processing", "mission critical", and "distributed" embedded systems are out of the scope of this thesis, "small" embedded systems most certainly are. A small system must be small in terms of physical dimensions and only utilize a minimum amount of resources (e.g. memory) in order to minimize the cost of the embedded system.

In order to better classify the requirements of embedded systems, the runtime, area, power, and energy characteristics are most commonly used to describe software and hardware implementations:

Runtime. A microprocessor always has to fulfill some functionality within a certain time frame. If some cryptographic operation has to be computed as part of that functionality, the time requirement directly passes itself on to the cryptographic implementation. In hardware design, the amount of necessary computation can be measured as cycle count t_c or actual

execution time t. The execution time can be computed as $t = \frac{t_c}{f}$, where f denotes the operating frequency.

Area. If a microchip is produced in large quantities, the manufacturing cost of a microchip is proportional to its size. Therefore, it is important to minimize the chip area in order to minimize the cost of a hardware design. If the microprocessor is fixed, it is only possible to minimize the area by removing functionality (e.g., embedded peripherals) or by reducing the memory footprint. Therefore, the requirement for software implementations is to minimize the amount of memory in order to keep the price as low as possible.

Power. Electric power P is defined as the rate at which energy is transferred. In complementary metal-oxide-semiconductor (CMOS) circuits, the power consumption can be expressed as $P = \alpha C V^2 f$, where α denotes the activity of a circuit, C the capacity of the switched circuit, V the power-supply voltage, and f the frequency of the clock source. The job of a system designer is to have a combination of α, C, V, and f that does not overload the power source. This is particularly important for RFID tags that are supplied wirelessly.

Energy is the product of power and runtime. Therefore, both the power and runtime must be optimized to achieve an energy-saving architecture. For hardware designers it is important to realize that $E = P \cdot t = \alpha C V^2 f \cdot \frac{t_c}{f} = \alpha C V^2 t_c$. Therefore, the energy consumption is independent of the operating frequency f of the device.

After this rather general introduction to embedded systems, the following two sections investigate the more specific requirements of wireless sensor networks and RFID tags.

2.2 Wireless Sensor Networks

A wireless sensor network consists of sensor nodes and a gateway so that the sensor nodes are accessible from the outside. A sensor node is equipped with a sensor, a communication interface, a light-weight and energy-saving microprocessor, as well as a battery. Using sensor nodes, a wide range of monitoring [39] and target-tracking [94] applications become feasible. In 2008, Yick *et al.* [114] performed an extensive study on wireless sensor networks, in particular the platform, the communication interface, and the network services.

As in some cases standard AAA batteries should last for a life-time of several years, it is important to keep the energy consumption as low as possible. This major requirement reflects in the used sensor, the used communication interface, and the embedded microprocessor.

Depending on the designated application, a wide range of sensors can be used. Possible sensors range from simple temperature and humidity probes [98, 99] to more expensive microphones and cameras [3]. Depending on the application, the sensors are only active when necessary to minimize energy consumption.

There are several standards that define communication interfaces for wireless sensor nodes. IEEE 802.15.4 [57] provides a low-cost solution especially suitable for wireless personal area networks. With its low complexity and low data rates it provides a platform for maximizing the battery life. On top of IEEE 802.15.4, Zigbee [115] automates the forming of mesh networks, in which hundreds of sensor nodes can join. Other protocols include, but are not limited to, WirelessHART [49], ISA100.11a [64], 6LoWPAN [96].

Apart from the sensor and the wireless interface, the biggest consumer of power is the embedded microprocessor. Current platforms [97] such as the IRIS, MICAz, MICA2, Imote2, and TelosB by Crossbow [24], or BEAN, COOKIES, EPIC mode, PowWow, Shimmer, TelosB, T-Mote Sky, and XM1000, utilize light-weight microprocessors such as the 8-bit Atmel megaAVR [7] or the 16-bit Texas Instruments MSP430 [95] processors. Those long-tested microprocessors made a name for themselves as being energy-efficient and used in a wide range of products. Lately, ARM has tried to place a foot in the door with one of the most inexpensive and most energy-efficient 32-bit microprocessor. The ARM Cortex-M0+ [6] is a direct competitor of the ATmega and MSP430 microprocessors. Several vendors already started to supply the market with their own Cortex-M0+ based sensor nodes [8].

In Wenger *et al.* [110], we did a comparison of the megaAVR, the MSP430, and the ARM Cortex-M0+ microprocessors and thoroughly opposed their respective ECC performance. It turns out that if only the performance of ECC matters, the Cortex-M0+ requires the smallest amount of energy for an elliptic curve scalar multiplication. However, if it is necessary to further reduce the energy requirement for an elliptic curve scalar multiplication, then it is necessary to perform optimizations on the architectural level. In our ACNS paper [103] from 2013, we introduced new architectural options in order to further decrease the energy consumption of sensor nodes.

We conclude that the energy consumption of a sensor node is of utmost importance. Price per sensor node and wall-clock performance are only of secondary interest. This is where RFID tags differ.

2.3 Radio Frequency Identification

Unlike wireless sensor nodes, passive RFID tags do not come with a battery. Passive RFID tags are wirelessly supplied with power. An antenna (e.g., coil) is

used to harvest power from an electromagnetic field. To minimize the cost, the same antenna is also used for communication purposes.

In general, in an RFID scenario it is important to distinguish between reader and tag devices. A reader, which is somehow connected to the overall network, emits an electromagnetic field. Thereby the general assumption is that the reader device is connected to a permanent or non-critical power source. In terms of computing power, any RFID reader easily outperforms an RFID tag. The most simple tags are only capable of providing an unique serial number, which can be used for identification purposes. In order to perform authentication, cryptography is a necessity. In particular RFID tags providing Triple DES and AES [86] are already used in order to do symmetric authentication. The drawback of those tags is the elevated power consumption. The power consumption is inversely proportional to the maximum reading distance. So the elevated power consumption of cryptographic tags has an effect on the maximum reading distance.

In order to get a better understanding of the additional requirements for RFID tags, it is important to review practical application scenarios. Finkenzeller [36] and Shepard [93] list applications like access control (public transport, ski tickets, electronic immobilization), personal identification (ski tickets, electronic passport), supply-chain management (clothing, commodity), and contact-less payment. The most common denominator of those applications is the requirement that RFID tags must be inexpensive. If a carton of milk is equipped with an RFID tag, the RFID tag must not have an influence on the price of the carton of milk. The price of an RFID tag is defined by its housing, its antenna and its ASIC, consisting of an analog and a digital integrated circuit. As the cost of an ASIC is mostly defined by the number of chips that fit on a single waver, the size of the chip must be minimal. So for a digital-circuit designer it is important to keep the circuit as small as possible[1].

So what is the role of elliptic curve cryptography within the context of RFID? The main problem of symmetric cryptography is the key distribution problem. A symmetric solution would be to host key-distribution centers with lists of symmetric keys. However, as soon as there are serval companies involved which do not want to share their secret keys, centralized key-distribution centers reach their limit. Such a problem can be easily circumvented using ECC.

Another scenario in which public-key cryptography is unavoidable comes from the sector of privacy preserving protocols, which are of great concern in the context of RFID. Garfinkel *et al.* [40] and Juels [68] give overviews on potential threats and elaborate possible solutions. Katherine Albrecht wrote a book [4] about how "Spychips" are going to be used to track "your every move". In a holistic investigation of possible privacy scenarios, Serge Vaudenay [100] comes to

[1] It is also necessary to avoid any expensive manufacturing steps and to not require too many metal layers. Such attributes also have a significant influence on the price of a microchip.

the conclusion that "[...] narrow-strong privacy[2][...] essentially needs public-key cryptography techniques."

The big research question is not whether ECC can be successfully implemented for RFID. Hutter *et al.* [58] proved it already by actually building a working RFID tag capable of symmetric-key and public-key authentication. The question is whether it is possible to implement ECC in such power-efficient and inexpensive manner such that an ECC-enabled RFID tag becomes feasible for large-scale applications.

In this context, our results from Wenger *et al.* [104] are highly encouraging. Although we did not build an ASIC, we synthesized our design as ASIC and successfully evaluated it on the IAIK RFID demo tag [87].

2.4 Conclusion

This chapter revisited the different needs of embedded systems in general and in particular wireless sensor nodes and RFID tags. The main criteria for cryptography on microprocessor-based embedded systems are *high speed* and *low memory* footprint. However, when those light-weight microprocessors are used within wireless sensor networks, it is most important to reduce the *energy* consumption. The less energy elliptic curve cryptography actually needs, the longer is the expected life time of a sensor node. On the other hand, an RFID tag is limited by the amount of energy per time (*power*) it consumes. Additionally, in order to be applicable for real-world systems, cryptographic protocols have to be handled within several hundreds of milliseconds. Therefore, there is a certain *runtime* requirement for RFID tags as well.

One of the most important criteria has not yet been discussed so far. Practical implementation security is crucial for all of the discussed scenarios. If there is some practical way to circumvent the cryptographic primitives, there is no use of implementing cryptography in the first place. Such practical implementation attacks are discussed in the following chapter.

[2]For a detailed definition of narrow-strong privacy, we refer to Vaudenay [100].

Chapter 3

Implementing Elliptic Curve Cryptography

There are few practical, light-weight, and standardized public-key algorithms which offer a similar huge design space as ECC does. With its rich mathematical structure, it is possible to optimize ECC on three algorithmic levels: the point arithmetic, the finite-field arithmetic, and the integer or polynomial arithmetic. On each level it is possible to crucially influence the practical outcome. It is therefore important to choose the appropriate algorithms for a given design goal. The most common design goals have already been discussed in the previous chapter. The target specification could demand to optimize for speed, area, power, or energy. However, it is also important to keep track of additional security requirements. Especially in a setting which contains wireless sensor nodes or RFID tags, it is important to be safe from practical physical attacks, such as passive and active implementation attacks.

In the following, we elaborate on different design options and discuss their respective implications in terms of speed, area, power, energy, and security. Hereby we look retrospectively at the lessons learned during the writing of our paper; some of which are explicitly given in Part II. As most of our work deals with embedded, area-constrained microprocessor and ASIC hardware[1] implementations of elliptic curve cryptography, we explicitly avoid to discuss strategies for high-speed and high-throughput applications.

[1] Doing area-optimized designs for ASICs largely differs from area-optimized designs for FPGAs. One only has to consider a design with a lot of combinatorial. This logic needs a lot of look-up tables which are part of slices. However, every slice also includes registers, which basically come for free as they hardly can be re-used once the combinatoric logic use the majority of the slices.

3.1 Notations

The following vocabulary is largely taken form the Guide to Elliptic Curve Cryptography by Hankerson *et al.* [48] as we consider their book as the standard book for implementing elliptic curve cryptography.

$K = \mathbb{F}_q$	General definition of a finite field. Most practical are prime fields $\mathbb{F}_p$ and binary extension fields $\mathbb{F}_{2^m}$.
p	The order of the prime field $\mathbb{F}_p$.
$f(z)$	The reduction polynomial (of degree m).
$E(K)$	Defines an elliptic curve over a finite field K. Commonly used short-forms for standardized [5, 62, 65, 85] Weierstrass equations are $E(\mathbb{F}_q) : y^2 = x^3 + ax + b$ for elliptic curves over prime fields and $E(\mathbb{F}_{2^m}) : y^2 + xy = x^3 + ax^2 + b$ for elliptic curves over binary fields. Non-Weierstrass-compatible elliptic curve equations are not considered throughout this chapter.
a, b	Coefficients of a given elliptic curve over either a prime field or binary extension field.
x, y	Coordinates of a point $P = (x, y)$ which satisfy the elliptic curve equation.
n	The prime order of a base point P.
h	The cofactor. $n \times h$ is equal to the number of points on $E(K)$, denoted as $\#E(K)$.

3.2 Cryptographic Protocols to be Considered

For the sake of completeness we give two schemes which are most commonly used within embedded environments. First, the Elliptic Curve Discrete Signature Algorithm (ECDSA) is a standardized signature scheme which can be used to reach common cryptographic goals like authentication and non-repudiation. Second, the Elliptic Curve Diffie-Hellman key exchange (ECDHKE) algorithm, based on Diffie and Hellman [27], is used to commonly derive a secret key over an insecure channel. Note that there also exist several ECC-based protocols especially designed for RFID tags that are non-standardized.

3.2.1 Elliptic Curve Discrete Signature Algorithm

The ECDSA algorithm is based on the Discrete Signature Algorithm (DSA) and widely standardized within ANSI X9.62 [5], FIPS 186-3 [85], IEEE 1363-2000 [62], and ISO/IEC 15946-2 [65], just to name some of the most important standards. Algorithms 1 and 2 show how to generate a signature using the private key d and how to verify a given signature (r, s) using the public key $Q = dP$.

Algorithm 1 ECDSA signature generation

Input: Domain parameters $D = (q, a, b, P, n, h)$, private key d, message m.
Output: Signature (r, s).

1: Select $k \in_R [1, n-1]$.
2: Compute $kP = (x_1, y_1)$ and convert x_1 to an integer $\overline{x}_1$.
3: Compute $r = \overline{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6: Return (r, s).

Algorithm 2 ECDSA signature verification

Input: Domain parameters $D = (q, a, b, P, n, h)$, public key Q, message m, signature (r, s).
Output: Acceptance or rejection of the signature.

1: Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails then return("Reject the signature").
2: Compute $e = H(m)$.
3: Compute $w = s^{-1} \bmod n$.
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5: Compute $X = u_1 P + u_2 Q$.
6: If $X = \infty$ then return("Reject the signature");
7: Convert the x-coordinate x_1 of X to an integer $\overline{x}_1$; compute $v = \overline{x}_1 \bmod n$.
8: If $v = r$ then return("Accept the signature");
Else return("Reject the signature").

The most straight-forward use case for ECDSA is to sign a document. With such a commitment, it is possible to authenticate a document, a person, or an entity and the signature can be verified by anybody who posseses the public key (non-repudiation). In embedded applications, ECDSA is usually used for challenge-response protocols. In such protocols one party wants to proof its identity to another party by signing a given random challenge. The signature is then checked by the person who generated the challenge. Thereby, secure, one-way authentication is performed.

Regarding side-channel security, Hutter *et al.* [60] practically demonstrated how one can attack the multiplication dr in line 5 of Algorithm 1 using a differential power analysis (DPA) attack. A straightforward countermeasure is to simply rearrange $s = k^{-1}(e + dr)$ to $s = k^{-1}e + (k^{-1}d)r$. Therefore, the private key d is always randomized with the random ephemeral k^{-1} and a DPA attack is not possible any more. DPA attacks on the actual scalar multiplication kP are not possible as random, ephemeral scalars are used.

Fault attacks on ECDSA have been shown by Schmidt and Medwed [92], Barenghi *et al.* [10, 11], and Giraud and Knudsen [41]. Therefore, countermeasures

Algorithm 3 Elliptic Curve Diffie-Hellman Key Exchange for party A.

Input: Domain parameters $D = (q, a, b, P, n, h)$, secret scalar k_A.
Output: Symmetric key K.
1: $Q_A \leftarrow k_A P$.
2: Send Q_A to party B and receive $Q_B = k_B P$ from party B.
3: $Q_{AB} \leftarrow k_A Q_B$.
4: $K \leftarrow KDF(Q_{AB})$.

against faults on the data-flow and the control-flow are necessary. Note that fault attacks not only threaten the ECDSA signature generation, which involves the private key, but also the ECDSA signature verification. It is possible to attack the conditional branches which are used to check the validity of r, s, and the equality of $v = r$.

Attacks on the actual scalar multiplication are not specific to the signature generation algorithm and are discussed after a short introduction of the Elliptic Curve Diffie-Hellman Key Exchange algorithm.

3.2.2 Elliptic Curve Diffie-Hellman Key Exchange

Algorithm 3 presents the pseudo-code for a Diffie-Hellman key exchange for one of the two participating parties. In order to derive the algorithm for the other party, some relabeling is necessary; replacing A with B and vice versa. A key derivation function KDF is used to derive the shared secret key.

During an ECDHKE, parties A and B choose random scalars and perform scalar multiplications with a random predefined base point P. Then, both parties exchange their respective ephemeral points ($Q_A = k_A P$, $Q_B = k_B P$) and perform scalar multiplications with the points from the other party ($Q_{AB} = k_A Q_B = k_B Q_A = k_A k_B P$). Using a key derivation function (KDF) it is then possible to use $K = KDF(Q_{AB})$ as symmetric key for further communications.

From an implementation point of view it is important to observe that the initial scalar multiplication is performed with a fixed point, while the latter is performed with a variable point. The scalar multiplication formula with the fixed base point provides huge optimization potential. So one could either use two performance-optimized scalar multiplication algorithms or a single, more general scalar multiplication algorithm which would potentially save chip area. In the following, we always assume to use a single scalar multiplication method, because the job of securing a scalar multiplication algorithm is twice as hard when two different algorithms must be secured.

In terms of implementation attacks, differential power analysis attacks are only possible when static keys k_A and k_B are used. In general, it is assumed that k_A and k_B are random ephemeral keys and therefore a DPA attack does not apply.

The same holds true for fault attacks. Fault attacks are not viable when random keys are used. Simple power analysis and horizontal collision correlation power analysis attacks are possible and are discussed in Section 3.4.1. It is up to the used scalar multiplication algorithm to deal with such attacks.

Note that the biggest flaw of the DHKE algorithm is a viable (wo)man-in-the-middle attack. Therefore, it is mandatory to use (standardized) protocols such as the station-to-station (STS) protocol by Diffie, van Oorschot and Wiener [28] or the three-pass key agreement protocol by Menezes, Qu and Vanstone, studied in Law *et al.* [74].

The station-to-station protocols extends the DHKE with symmetric message authentication codes (MAC) and asymmetric signatures. Basically, the STS protocol is a Diffie-Hellman key exchange which uses ECDSA to proof the authenticity of the keys. In all occurrences of security critical scalar multiplications within the STS protocol, ephemeral keys are used as scalars. Therefore, DPA attacks are not feasible for the ECDSA, the ECDHKE, or the STS protocol.

Apart from those attacks, it is possible to perform, e.g., SPA attacks on the scalar multiplication algorithm. Thus, it is mandatory to utilize a scalar multiplication algorithm that is aware of practical physical attacks and applies several countermeasures. In the following, we derive a secure scalar multiplication algorithm from related work.

3.3 Methods for Scalar Multiplication

Within the scope of ECC-based cryptographic protocols, side-channel threats that cannot be solved on the protocol layer must be handled within the point arithmetic layer. The core algorithm within this layer is the scalar/point multiplication algorithm. Using the right methodology crucially influences the performance and the physical vulnerability of an actual implementation.

In the following, we review some insecure scalar multiplication algorithms and build up to an algorithm which we consider to be secure against many types of physical attacks. The scalar multiplication algorithm is designed to be used within ECDHKE and ECDSA, two algorithms that only perform scalar multiplications with random scalars. Therefore, the scalar multiplication algorithm must not be secured against differential power analysis attacks. However, simple power analysis and fault analysis attacks are an omnipresent danger.

Algorithm 4 Left-to-right double-and-add scalar multiplication.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Output: kP
1: $Q \leftarrow P$
2: **for** i from $t-2$ downto 0 **do**
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **if** $k_i = 1$ **then**
5: $\quad\quad Q \leftarrow Q + P$
6: **end for**
7: Return Q

Algorithm 5 Left-to-right double-and-add-always scalar multiplication.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Output: kP
1: $Q \leftarrow P$
2: **for** i from $t-2$ downto 0 **do**
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **if** $k_i = 1$ **then**
5: $\quad\quad Q \leftarrow Q + P$
6: $\quad$ **else**
7: $\quad\quad T \leftarrow Q + P$
8: **end for**
9: Return Q

3.3.1 Methods for Insecure Scalar Multiplication

To understand how to securely perform scalar multiplications one needs to understand how standard elliptic curve implementations can be attacked. Algorithm 4 presents a standard left-to-right double-and-add scalar multiplication algorithm. The scalar k is $t = \lceil log_2(k) \rceil$ bits long and processed bit by bit. k is represented as $k = \sum_{i=0}^{t-1} 2^i k_i = 2^{t-1}k_{t-1} + \cdots + 2^2 k_2 + 2^1 k_1 + k_0$. The most serious problem of Algorithm 4 is its power profile. It is rather easy to detect when point doublings and additions are performed in dependency of the key bits k_i.

The standard solution to this problem is depicted in Algorithm 5. The double-and-add-always algorithm also performs a point addition even when the key bit is zero. Therefore, a constant timing and a constant power profile can be reached. However, this algorithm is vulnerable to fault attacks. During the computation of Algorithm 5, it is possible to induce a fault during the computation of a point addition. If the result Q is corrupted, k_i is equal to one, if Q is not corrupted, the dummy operation was attacked and k_i is equal to zero.

More craftily scalar multiplication methods are based on non-adjacent forms,

Algorithm 6 Scalar multiplication based on the atomicity principle.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Output: kP
1: $Q[0] \leftarrow P$
2: $Q[1] \leftarrow P$
3: $i \leftarrow t-2$, $d \leftarrow 0$
4: **while** $i \geq 0$ **do**
5: $\quad Q[0] \leftarrow Q[0] + Q[d]$
6: $\quad d \leftarrow d \oplus k_i$, $i \leftarrow i - (d \oplus 1)$
7: **end while**
8: Return $Q[0]$

window methods, and fixed-base comb methods, as thoroughly discussed in Hankerson *et al.* [48]. The underlying problem of those methods is the representation of the identity. There is no explicit representation of the identity $\infty = 0P$ in standard-conform Weierstrass curves. The identity must be specially handled and breaks the algorithm flow. Therefore, any algorithm that performs point additions in the form of $Q \leftarrow Q + P$ must fail.

As additional alternative to a Montgomery ladder, it is important to consider implementations based on the atomicity principle [19, 42, 77]. These specially crafted formulas use a single primitive to calculate both point additions and point doublings. The motive is that an attacker cannot distinguish the point addition from the doubling in a power profile. Algorithm 6 depicts the underlying algorithm. It is important to notice that its runtime directly depends on the Hamming weight (the number of ones) of k. This is quite troublesome if one ultimately wants to protect the control/program flow, as discussed later.

However, the rising research on horizontal collision correlation attacks has practically defeated several formulas that are based on the atomicity principle. Bauer *et al.* [13] showed how to distinguish operands which are re-used within finite-field multiplications. If a key-dependent correlation can be performed, it is possible to recover the key. Therefore, it is mandatory to use a different scalar multiplication methodology.

3.3.2 Montgomery Ladder

The Montgomery ladder is named after Peter L. Montgomery who wrote several of the most fundamental papers that are still used to improve the performance of ECC (e.g., [81, 82]). In [82], he proposed special ECC formulas for differential additions. In a differential point addition, the two points to be added, i.e., $Q[0]$ and $Q[1]$, have the special property that their difference $Q[1] - Q[0] = P$ is constant and known a priori. Using this knowledge, it is possible to derive fast point addition algorithms which can be used within a Montgomery ladder.

Algorithm 7 Elliptic curve Montgomery ladder.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Output: $R = kP$
1: $Q[0] \leftarrow P$
2: $Q[1] \leftarrow 2 \cdot P$
3: **for** $i = t - 2$ **downto** 0 **do**
4: $\quad Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$
5: $\quad Q[k_i \oplus 1] \leftarrow 2 \cdot Q[k_i \oplus 1]$
6: **end for**
7: Return $Q[0]$

However, the most important reason to use a Montgomery ladder nowadays is its resistance against many simple side-channel attacks.

As depicted in Algorithm 7, two points with the constant difference $Q[1] - Q[0] = P$ are used within the Montgomery ladder. The core of the Montgomery ladder are the point addition ($Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$) and the point doubling ($Q[k_i \oplus 1] \leftarrow 2 \cdot Q[k_i \oplus 1]$) operations. Those point operations are performed in a key-independent fashion and do not utilize any dummy operations. At this point we must urge a designer to actually implement $Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$, but not $Q[k_i] \leftarrow Q[0] + Q[1]$, as opposed to, e.g., Fan *et al.* [33]. Our experience in Wenger *et al.* [109] showed that using $Q[k_i] \leftarrow Q[0] + Q[1]$ as point addition eases horizontal collision correlation attacks on Montgomery ladders.

From an implementation point of view, many formulas to efficiently compute a Montgomery ladder were proposed. The original formulas by Peter Montgomery [82] are not suitable for standardized Weierstrass curves. For the standardized curves it is important to distinguish elliptic curves over binary (extension) fields and elliptic curves over prime fields. The fastest formulas for elliptic curves over binary fields are from López and Dahab [78]. For elliptic curves over prime fields, formulas have been introduced by Izu *et al.* [66], Hutter *et al.* [59], and Goundar *et al.* [44, 45].

Although a Montgomery ladder provides a good foundation against many implementation attacks it is not sufficient to use Algorithm 7 just as is. According to Fan *et al.* [33], a plain Montgomery ladder is not safe from comparative side-channel attacks [113], zero point attacks [1], and twisted curve attacks [37]. In the following section, a strengthened version of Algorithm 7 is introduced and thoroughly analyzed regarding side-channel attacks.

Algorithm 8 Hardened scalar multiplication algorithm for a Weierstrass curve over a prime field.

Input: Domain parameters $D = (p, a, b, P, n, h)$, $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_p) = (x, y)$.
Output: $R = kP$

1: **if** $y^2 \neq x^3 + ax + b$ **then** Perform Error Handling
2: $(X, Y, Z) \leftarrow (x \cdot \lambda, y \cdot \lambda, \lambda)$ ▷ Randomize Projective Coordinates
3: **if** $Y^2Z \neq X^3 + aXZ^2 + bZ^3$ **then** Perform Error Handling
4: $Q[0] \leftarrow (X, Z), Q[1] \leftarrow 2 \cdot (X, Y, Z)$ ▷ Initial Point Doubling
5: **for** $i = t - 2$ **downto** 0 **do** ▷ Montgomery Ladder
6: $\quad Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$
7: $\quad Q[k_i \oplus 1] \leftarrow 2 \cdot Q[k_i \oplus 1]$
8: **end for**
9: $(X, Y, Z) \leftarrow$ `y-recovery`$(Q[0], Q[1])$
10: **if** $Y^2Z \neq X^3 + aXZ^2 + bZ^3$ **then** Perform Error Handling
11: $(x, y) \leftarrow (XZ^{-1}, YZ^{-1})$
12: **if** $y^2 \neq x^3 + ax + b$ **then** Perform Error Handling
13: Return $R = (x, y)$

3.4 Side-Channel Hardened Point Arithmetic

The algorithm, we came up with in Wenger *et al.* [110], is depicted in Algorithm 8. Our initial assumption is that the most significant bit (MSB) of the scalar k in $Q = kP$ is always set to one. As we are also in control of the random number generation, this is a reasonable assumption. Further, if the length of the scalar is constant and constant-runtime finite-field operations are used, the Montgomery ladder finishes in constant time. Therefore, lattice-based timing attacks, e.g., by Brumley and Tuveri [17], on the Montgomery ladder are infeasible.

Algorithm 8 combines three countermeasures against side-channel attacks. First, a Montgomery ladder is used. The underlying formulas use a projective (X, Z) representation and therefore the recovery of the y-coordinate has to be performed. Second, randomized projective coordinates (cf. Coron [23]) make sure that the Montgomery ladder is processing randomized data. The processing of randomized data makes any power analysis or fault attack much more complex. As we assumed to have a random-number generator on the protocol layer, we can also assume to use the same random number generator on the point-arithmetic layer. Also, as projective coordinates are used anyways, this countermeasure is basically for free.

The third countermeasure consists of several point verification checks. They make sure that the given points satisfy the elliptic curve equation. Such a point verification check is relatively inexpensive to compute in comparison to the Montgomery ladder. Therefore, doing the point verifications multiple times

is not costly. The initial point verification makes sure that the input point P is indeed valid. After the point randomization, another point verification is performed. Attacking both the randomization step and the second point verification would already require multiple (different) fault attacks. After the computation of the Montgomery ladder and the recovery of the y-coordinate another two point verifications are performed. Like before, the processed point is both checked within its projective-coordinate representation and within the affine representation.

As the recovery of the y-coordinate can always be performed within the projective coordinates, it does not require the use of a costly inversion. Although ECDSA does not make use of the y-coordinate, the y-coordinate is needed for the final two point verifications.

Algorithm 8 shows point verification checks for a Weierstrass curve over a prime field. For a Weierstrass curve over a binary field, the implementation must check whether the point P fulfills the equation $y^2 + xy = x^3 + ax^2 + b$ instead of $y^2 = x^3 + ax + b$. Also the formulas used for the initial point doubling, the Montgomery ladder, and the `y-recovery` must be adapted according to the used underlying field. If Algorithm 8 is used for a non-standardized (non-Weierstrass) curve, the formulas used have to be modified as well.

3.4.1 Vulnerability Analysis

In the following, we discuss Algorithm 8 in relation to the latest related work on practical physical attacks and its nice security properties. The following analysis is somehow based on Fan *et al.* [33, 34], who wrote several overview papers on secure elliptic curve implementations and Karaklajić *et al.* [69], who did a thorough investigation of fault attacks. The physical attacks are categorized based on their feasibility to attack Algorithm 8.

Physical attacks that should not affect the security of Algorithm 8:

Timing Analysis. When the finite-field operations are implemented in a constant-time fashion a timing attack similar to the one introduced by Kocher [71] is not possible. Brumley and Tuveri [17] showed a timing attack that measures the number of most significant zero bits of the Montgomery ladder. In combination with the LLL algorithm [76], they recovered the scalars used for the scalar multiplications. As we are in control of the random number generator and manually set the most significant bit of the scalar, the attack of Brumley and Tuveri does not affect Algorithm 8.

Simple Power Analysis. According to Kocher [72], it should not be possible to directly interpret power-consumption measurements. While it is rather easy to detect conditionally performed point additions on a power trace, double-and-add operations within the Montgomery ladder are highly regular.

However, there is a key dependence ($Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$) within the Montgomery ladder. Therefore, it is a matter of practical evaluation to make sure that no critical information leaks. Also noise generators help in the suppression of this key-dependent information. Note that localized electromagnetic-emanation analysis is a type of simple power analysis attacks, but much more powerful. It is separately discussed below.

Differential Power Analysis (DPA). First introduced by Kocher *et al.* [72], the DPA provides statistical means to deduce the processed key from given power profiles. DPA attacks are very powerful and complex masking and hiding schemes must be used to guard the cryptographic implementation. For DPA attacks to work, several power measurements with known plaintexts and a constant key are necessary. As we initially assumed that Algorithm 8 is used in settings with random, ephemeral keys, a DPA attack is not applicable. The decryption operation of the Elliptic Curve Integrated Encryption Scheme (ECIES) performs a scalar multiplication involving a private key. For such scenarios additional countermeasures such as the random scalar split by Ciet and Joye [20] are necessary.

Template Attack. According to Medwed and Oswald [79], template attacks can be used to attack ECDSA and the only way to be safe from template-based SPA attacks is to be secure against differential power analysis attacks. In 2008, Herbst and Medwed [51] also showed how to use templates to attack the randomization step of Algorithm 8. According to Fan *et al.* [33], only the randomization of the coordinates provides protection against template attacks. Also, additional countermeasures must be in place to inhibit the building of templates in the first place (e.g., during the ECDSA verification).

Refined (Zero-Value) Power Analysis. In 2003, Akishita *et al.* [2] introduced the zero-value point analysis and Goubin [43] the refined power analysis. Both attacks are based on the same working principle. The attacker chooses the input point in such a way, that she can detect the occurrence of a "special" elliptic-curve point in the power trace. Thereby she can recover a bit of the key. As this attack is only practical in the case of constant scalars, which contradicts our initial assumption, we do not need to handle those attacks.

Comparative Side-Channel Analysis. Like the zero-value power analysis, the comparative power analysis chooses specially prepared input points to generate repeating patterns in the power profile. While zero-value power analysis aims to detect coordinates equal to zero, the comparative side-channel attack, first introduced by by Homma *et al.* [55, 56], tests for repeating patterns in multiple power profiles. Similar to above, they use random, ephemeral scalars counters comparative side-channel analysis.

C & M Safe-Error Analysis. Algorithm 5 is a great example for not being safe from safe-error analysis (cf. Joye and Yen [67, 112]). If a fault is

introduced right before or during the point addition, the fault will or will not propagate to the output, depending on the key bit k_i. Thereby the attacker can reveal one key bit per induced fault. As Algorithm 8 uses a dummy-operation-free Montgomery ladder and ephemeral keys, it is not threatened by safe-error attacks.

Weak Curve-Based Analysis. The core idea of those attacks is to change the used elliptic curve from E to E'. E' is a cryptographically weak curve with subgroups that are vulnerable to the Pohlig-Hellman attack[2] [88]. Performing point verification before and after scalar multiplication counters invalid-point attacks. For invalid-curve attacks, the stored curve parameters must be checked (see below). Biehl *et al.* [14] and Ciet and Joye [21] discuss details on invalid-point and invalid-curve attacks. Even more critical are the twist-curve-based attacks by Fouque *et al.* [37] which even affect Montgomery-ladder implementations. The problem is that the twists of the standardized elliptic curves `secp160-224r1` are below 60 bits. However, according to Ebeid and Lambert [29] recovering the y-coordinate and verifying the recovered point, counters the twist-curve-based attack by Fouque *et al.*.

Sign-Change Fault Analysis. Ebeid and Lambert also analyze the effect of the sign-change fault analysis by Blömer *et al.* [15] on the protected Montgomery ladder and come to the conclusion that the sign-change attack is not feasible on a protected Montgomery ladder.

Though the attacks mentioned above are handled by Algorithm 8, Algorithm 8 cannot solve all types of attacks. Below, we give some physical attacks that we believe to threaten an ECC implementation, even though it utilizes Algorithm 8:

Program-Flow Fault Attacks. Fault attacks which target the program flow, as done by Schmidt *et al.* [91, 92], can skip operations, function calls, or point-validation checks. To circumvent such types of attacks, we recommend to use additional hardware countermeasures which are subject to future work (see Chapter 4).

Invalid-Curve Analysis. Ciet and Joye [21] raise the question what happens when the stored elliptic-curve parameters get modified. Especially in combination with other fault attacks, an invalid curve is hard to detect using the presented countermeasures. As already mentioned above, we recommend the usage of hardware countermeasures which are subject to future work.

Electromagnetic-Emanation Analysis. In [52, 53, 54], Heyszl *et al.* perform localized electromagnetic-emanation analysis attacks. If an attacker is able

[2]Factorize the order n' of E', solve the ECDLP in all subgroups and derive the actually used k.

Algorithm 9 Elliptic curve Montgomery ladder with memory randomization.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Output: $R = kP$

```
1: T[0] ← P
2: T[1] ← 2 · P
3: r ← rand()
4: Q[r] ← T[0]
5: Q[r ⊕ 1] ← T[1]
6: for i = t − 2 downto 0 do
7:     T[0] ← Q[k_i ⊕ r]
8:     T[1] ← Q[k_i ⊕ r ⊕ 1]
9:     T[0] ← T[0] + T[1]
10:    T[1] ← 2 · T[1]
11:    r ← rand()
12:    Q[k_i ⊕ r] ← T[0]
13:    Q[k_i ⊕ r ⊕ 1] ← T[1]
14: end for
15: Return Q[r]
```

to detect which memory location is accessed at which cycle, she can derive the key from that information. A countermeasure must not randomize the data in the memory but the location where the data is actually stored. Algorithm 9 outlines how such a countermeasure might work. While the point-doubling and the point-addition steps are performed at fixed memory locations, the points $Q[0]$ and $Q[1]$ are stored at random locations within the memory. This countermeasure comes at the cost of four copy operations. However, another intrinsic countermeasure is related to improvement of CMOS (Moore's law). The smaller the CMOS manufacturing technologies become and the closer registers are physically located, the less exact localized electromagnetic-emanation attacks are.

Horizontal Collision Correlation Analysis have recently become a more active research topic. The idea is that there are two power-consumption profiles which do not leak any information by themselves, but when correlated, secret information leaks. E.g., in Wenger *et al.* [109], we investigated the leakage of a Montgomery ladder and we were able to recover the key. A countermeasure against horizontal collision correlation attacks might be to re-randomize the projective points after each double-and-add step, i.e., $(X, Y) \leftarrow (\lambda X, \lambda Y)$. It might also be sufficient to use a λ that is only 8 bits, 16 bits, or 32 bits long (depending on the word size of the microprocessor). It is subject to future work to investigate the vulnerability of formulas used within Montgomery ladders.

3.5 Conclusion

The challenge is not only to be safe from a single attack scenario, because every arbitrary combination of attacks is possible. E.g., at CHES 2011, Fan *et al.* [32] presented a technique in which they combined a fault attack with a power-analysis attack. Intentionally, Algorithm 8 is safe from their combined attack, but it is unclear with which attacks or combination of attacks researchers will come up with.

Chapter 4

Future Work

We implemented elliptic curve cryptography for many different platforms and in many different ways. We also evaluated some implementations regarding their side-channel security. However, because of the complexity of implementing elliptic curve cryptography there is a lot more research to be done.

Here is just a short digest of ideas for future research papers:

- It is just a matter of time until a horizontal collision correlation attack on a Montgomery Ladder is successfully performed in practice. We are certain that the threat of correlation analysis attacks on elliptic curve cryptography must be handled with the same seriousness as the differential power analysis attacks on symmetric ciphers. Future research must include practical evaluations of correlation analysis attacks on ECC as well as research on light-weight countermeasures.
- During the implementation of an assembly optimized ECC for the ATmega we made an implementation error within the assembly code. This is nothing exceptional and has to be expected in practice. However, the problem was the detection of the error. E.g., for NIST P-192, the error only occurred with a probability of approximately $\frac{2^{64}}{2^{192}} = 2^{-128}$ when testing with random data. Therefore, the error never occurred in practical randomly generated test cases. However, such an implementation error can be misused for fault analysis attacks. The conclusion from this anecdote is that future research must include a formal verification of the cryptographic implementation. Especially when carry-bits are involved.
- While elliptic curve cryptography is already used within a wide range of applications, paring-based cryptography is still in its childhood days. Using pairing-based cryptography, a larger range of protocols and applications become practically feasible. However, pairing-based cryptography from

an implementors standpoint is much more complex and therefore requires much more optimization.

- Apart from power analysis attacks on elliptic curve cryptography also fault attacks are a serious threat. Fault attacks on the data can easily be detected by doing point verifications. However, fault attacks on the control/program flow are a much more serious threat. They can happen anytime, at any position in the chip. And they do not only threaten cryptographic primitives and protocols, but also conditional branches. For instance, to check the validity of a password, a conditional branch is necessary. Using a fault attack on the control-flow, it is possible to circumvent the validity check, no matter how protected the cryptographic implementation is.

These are just a fraction of the more concrete ideas we have planned for the future.

Bibliography

[1] T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In C. Boyd and W. Mao, editors, *ISC*, volume 2851 of *Lecture Notes in Computer Science*, pages 218–233. Springer, 2003.

[2] T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In C. Boyd and W. Mao, editors, *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*, volume 2851 of *Lecture Notes in Computer Science*, pages 218–233. Springer, 2003.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4):921–960, 2007.

[4] K. Albrecht. *Spychips: How major corporations and government plan to track your every move with RFID.* Thomas Nelson Inc, 2005.

[5] American National Standards Institute (ANSI). AMERICAN NATIONAL STANDARD X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.

[6] ARM. Cortex-M0+ Processor, August 2013. Available online at `http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php`.

[7] Atmel Corporation. megaAVR Microcontroller. Available online at `http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx`, August 2013.

[8] Avnet, Inc. Avnet Wi-Go Module, August 2013. Available online at `http://www.em.avnet.com/en-us/design/drc/Pages/Avnet-Wi-Go-Module.aspx`.

[9] S. Babbage, D. Catalano, C. Cid, B. de Weger, O. Dunkelman, C. Gehrmann, L. Granboulan, T. Güneysu, J. Hermans, T. Lange, A. Lenstra, C. Mitchell, M. Näslund, P. Nguyen, C. Paar, K. Paterson, J. Pelzl, T. Pornin, B. Preneel, C. Rechberger, V. Rijmen, M. Robshaw,

A. Rupp, M. Schläffer, S. Vaudenay, F. Vercauteren, and M. Ward. ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012). Available online at `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`, September 2012.

[10] A. Barenghi, G. Bertoni, A. Palomba, and R. Susella. A novel fault attack against ECDSA. In *HOST*, pages 161–166. IEEE Computer Society, 2011.

[11] A. Barenghi, G. M. Bertoni, L. Breveglieri, G. Pelosi, and A. Palomba. Fault attack to the elliptic curve digital signature algorithm with multiple bit faults. In M. A. Orgun, A. Elçi, O. B. Makarevich, S. A. Huss, J. Pieprzyk, L. K. Babenko, A. G. Chefranov, and R. Shankaran, editors, *SIN*, pages 63–72. ACM, 2011.

[12] E. Barker, W. Barker, W. Burr, W. Polk, , and M. Smid. NIST Special Publication 800-57: Recommendation for Key Management – Part 1: General (Revision 3). Available online at `http://csrc.nist.gov/groups/ST/toolkit/key_management.html`, July 2012.

[13] A. Bauer, E. Jaulmes, E. Prouff, and J. Wild. Horizontal Collision Correlation Attack on Elliptic Curves. In *Selected Areas in Cryptography*, Lecture Notes in Computer Science. Springer, 2013. In press.

[14] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000.

[15] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *FDTC*, pages 36–52, 2006.

[16] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, September 2007. ISBN 978-3-540-74734-5.

[17] B. B. Brumley and N. Tuveri. Remote Timing Attacks Are Still Practical. In V. Atluri and C. Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 355–371. Springer, 2011.

[18] Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. Available online at `http://www.secg.org/`, January 2010.

[19] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004. ISSN 0018-9340.

[20] M. Ciet and M. Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In S. Qing, D. Gollmann, and J. Zhou, editors, *ICICS*, volume 2836 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2003.

[21] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Des. Codes Cryptography*, 36(1):33–43, 2005. Available online at `http://eprint.iacr.org/2003/028.pdf`.

[22] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006.

[23] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[24] Crossbow Technology. Crossbow Wireless Modules Portfolio. `http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=156`, August 2013.

[25] J. Daemen and V. Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.

[26] Damien Giry. Keylength - Cryptographic Key Length Recommendations. Available online at `http://www.keylength.com/`, August 2013.

[27] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[28] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[29] N. Ebeid and R. Lambert. Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, Lausanne, Switzerland, 2009, Proceedings*, pages 46–50, September 2009.

[30] H. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.

[31] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[32] J. Fan, B. Gierlichs, and F. Vercauteren. To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2011.

[33] J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on known Side-Channel Attacks and Countermeasures. In *Hardware-Oriented Security and Trust - HOST 2010, In 3rd IEEE International Symposium, California, USA, June 13-14, 2010, Proceedings.*, pages 76–87. IEEE, 2010.

[34] J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications*, LNCS. Springer, 2012.

[35] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithm. In *CHES*, pages 357–370, 2004.

[36] K. Finkenzeller. *RFID-Handbook*. Carl Hanser Verlag, 2nd edition, April 2003. ISBN 0-470-84402-7.

[37] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack on Elliptic Curve Montgomery Ladder Implementation. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography, Fifth International Workshop, FDTC 2008, Washington DC, USA, August 10, 2008, Proceedings*, pages 92–98. IEEE Computer Society, August 2008.

[38] R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.

[39] T. Gao, D. Greenspan, M. Welsh, R. Juang, and A. Alm. Vital Signs Monitoring and Patient Tracking Over a Wireless Network. In *27th Annual International Conference of the Engineering in Medicine and Biology Society*, pages 102–105. IEEE, 2006.

[40] S. Garfinkel, A. Juels, and R. Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security and Privacy Magazine*, 3(3):34–43, May-June 2005.

[41] C. Giraud and E. W. Knudsen. Fault Attacks on Signature Schemes. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, volume 3108 of *Lecture Notes in Computer Science*, pages 478–491. Springer, July 2004.

[42] C. Giraud and V. Verneuil. Atomicity Improvement for Elliptic Curve Scalar Multiplication. In D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application - CARDIS 2010, 9th International Conference, Passau, Germany, April 14-16, 2010. Proceedings 2010*, volume 6035 of *Lecture Notes in Computer Science*, pages 80–101. Springer, 2010.

[43] L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2003.

[44] R. Goundar, M. Joye, and A. Miyaji. Co-Z Addition Formulae and Binary Ladders on Elliptic Curves. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010, 12th International Workshop Santa Barbara, California, USA, 2010 Proceedings*, volume 6225 of *Lecture in Computer Science*, pages 65–79. Springer, 2010.

[45] R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli. Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic. *J. Cryptographic Engineering*, 1(2):161–176, 2011.

[46] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer, 2004.

[47] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2006), Dubrovnik, Croatia, 30. August-1 September, 2006. Proceedings*, pages 577–583. IEEE Computer Society, September 2006.

[48] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.

[49] HART Communication Foundation. WirelessHART Technology Website. `http://www.hartcomm.org/protocol/wihart/wireless_technology.html`.

[50] S. Heath. *Embedded systems design*. Newnes, 2002.

[51] C. Herbst and M. Medwed. Using Templates to Attack Masked Montgomery Ladder Implementations of Modular Exponentiation. In K.-I. Chung, M. Yung, and K. Sohn, editors, *9th International Workshop on Information Security Applications (WISA 2008), Jeju Island, Korea, September 23-25, 2008, Proceedings*, volume 5379 of *Lecture Notes in Computer Science*, pages 1–13. Springer, Februar 2008.

[52] J. Heyszl, A. Ibing, S. Mangard, F. De Santis, and G. Sigl. Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. Technical report, Cryptology ePrint Archive, Report 2013/438, 2013. `http://eprint.iacr.org`.

[53] J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized Electromagnetic Analysis of Cryptographic Implementations. In O. Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 231–244. Springer, 2012.

[54] J. Heyszl, D. Merli, B. Heinz, F. D. Santis, and G. Sigl. Strengths and Limitations of High-Resolution Electromagnetic Field Measurements for Side-Channel Analysis. In S. Mangard, editor, *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2012.

[55] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Collision-Based Power Analysis of Modular Exponentiation Using Chosen-Message Pairs. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2008.

[56] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE Transactions on Computers*, 59(6):795–807, 2010.

[57] I. Howitt and J. A. Gutierrez. IEEE 802.15.4 Low Rate-Wireless Personal Area Network Coexistence Issues. In *Wireless Communications and Networking*, volume 3, pages 1481–1486. IEEE, 2003.

[58] M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In *RFIDSec*, pages 189–202, 2010.

[59] M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 Fourth International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2011.

[60] M. Hutter, M. Medwed, D. Hein, and J. Wolkerstorfer. Attacking ECDSA-Enabled RFID Devices. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security – ACNS 2009, 7th International Conference, Paris-Rocquencourt, France, June 2-5, 2009, Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 519–534. Springer, May 2009.

[61] M. Hutter and E. Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In B. P. und Tsuyoshi Takagi, editor, *Cryptographic Hardware and Embedded Systems – CHES 2011, 13th International Workshop, Nara, Japan, September 28 - October 1, 2011, Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 459–474. Springer, 2011.

[62] IEEE. IEEE Standard 1363-2000: IEEE Standard Specifications for Public-Key Cryptography. Available online at `http://ieeexplore.ieee.org/servlet/opac?punumber=7168`, 2000.

[63] Intel Corporation. Advanced Encryption Standard (AES) Instructions Set. `http://softwarecommunity.intel.com/articles/eng/3788.htm`.

[64] ISA100 Wireless Compliance Institute. ISA100, Wireless Systems for Automation. `http://www.isa.org/isa100`.

[65] ISO. ISO/IEC 15946-2:2002: Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 2: Digital signatures, 2002.

[66] T. Izu, B. Möller, and T. Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In A. Menezes and P. Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 296–313. Springer, 2002.

[67] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2003.

[68] A. Juels. RFID Security and Privacy: A Research Survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, February 2006.

[69] D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede. Hardware Designer's Guide to Fault Attacks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, February 2013.

[70] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[71] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, pages 104–113, 1996.

[72] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[73] P. Koopman. Embedded System Design Issues (The Rest of the Story). In *International Conference on Computer Design: VLSI in Computers and Processors*, pages 310–317. IEEE, 1996.

[74] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Designs, Codes and Cryptography*, 28(2):119–134, March 2003.

[75] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.

[76] A. K. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[77] P. Longa. Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields. Master's thesis, School of Information Technology and Engineering, University of Ottawa, Canada, 2007.

[78] J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^{m}) without Precomputation. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1999.

[79] M. Medwed and E. Oswald. Template Attacks on ECDSA. In K.-I. Chung, M. Yung, and K. Sohn, editors, *9th International Workshop on Information Security Applications (WISA 2008), Jeju Island, Korea, September 23-25, 2008, Pre-Proceedings*, pages 14–27, 2008.

[80] V. S. Miller. Use of Elliptic Curves in Cryptography. In H. C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1986.

[81] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44:519–521, 1985.

[82] P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, January 1987. ISSN 0025-5718.

[83] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K. G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011, 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[84] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.

[85] National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.

[86] NXP Semiconductors Austria GmbH. Website mifare.net - contactless smartcard technology. `http://www.mifare.net`.

[87] T. Plos, M. J. Aigner, T. Baier, M. Feldhofer, M. Hutter, T. Korak, and E. Wenger. Semi-Passive RFID Development Platform for Implementing and Attacking Security Tags. *International Journal of RFID Security and Cryptography*, 1:16–24, 2012.

[88] S. Pohlig and M. Hellman. An Improved Algorithm for Computing Logarithms over GP(p) and Its Cryptographic Significance. *Information Theory, IEEE Transactions on*, 24(1):106–110, 1978.

[89] E. Prouff, editor. *Smart Card Research and Advanced Applications - 10th IFIP WG 8.8/11.2 International Conference, CARDIS 2011, Leuven, Belgium, September 14-16, 2011, Revised Selected Papers*, volume 7079 of *Lecture Notes in Computer Science*. Springer, 2011.

[90] 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. ISSN 0001-0782.

[91] J.-M. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography, Fifth International Workshop, FDTC 2008, Washington DC, USA, August 10, 2008, Proceedings*, pages 53–58. IEEE Computer Society, August 2008.

[92] J.-M. Schmidt and M. Medwed. A Fault Attack on ECDSA. In D. Naccache and E. Oswald, editors, *Fault Diagnosis and Tolerance in Cryptography, Sixth International Workshop, FDTC 2009, Lausanne, Switzerland, September 6, 2009, Procceedings*, pages 93–99. IEEE-CS Press, September 2009.

[93] S. Shepard. *RFID Radio Frequency Identification*. McGraw-Hill Companies, 2005.

[94] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12. ACM, 2004.

[95] Texas Instruments. Overview for MSP430 Ultra-Low Power 16-bit MCUs. Available online at `http://www.ti.com/msp430`, August 2013.

[96] The Wireless Embedded Internet. 6lowpan Website. `http://6lowpan.net/`.

[97] TIK WSN Research Group at ETH Zurich. The Sensor Network Museum. `http://www.snm.ethz.ch/snmwiki/Main/HomePage`, August 2013.

[98] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, et al. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63. ACM, 2005.

[99] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 154–165. ACM, 2005.

[100] S. Vaudenay. On Privacy Models for RFID. In *Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2007, 13th International Conference, Kuching, Malaysia, December 2-6, 2007. Proceedings*, volume 4833, pages 68–87, 2007.

[101] E. Wenger. Neptun - ECC Processor for RFID Tags and Smart Cards. Master thesis, Swiss Federal Institute of Technology Zürich, Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria, May 2010.

[102] E. Wenger. A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography. In G. Avoine and O. Kara, editors, *Lightweight Cryptography for Security and Privacy*, Lecture Notes in Computer Science, pages 1–15. Springer, 2013.

[103] E. Wenger. Hardware Architectures for MSP430-Based Wireless Sensor Nodes Performing Elliptic Curve Cryptography. In M. J. Jacobson, M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *ACNS*, volume 7954 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2013.

[104] E. Wenger, T. Baier, and J. Feichtner. JAAVR: Introducing the Next Generation of Security-Enabled RFID Tags. In *DSD*, pages 640–647. IEEE, 2012.

[105] E. Wenger and J. Großschädl. An 8-bit AVR-Based Elliptic Curve Cryptographic RISC Processor for the Internet of Things. In *MICRO Workshops*, pages 39–46. IEEE Computer Society, 2012.

[106] E. Wenger and M. Hutter. A Hardware Processor Supporting Elliptic Curve Cryptography for Less than 9 kGEs. In Prouff [89], pages 182–198.

[107] E. Wenger and M. Hutter. A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9kGEs. In E. Prouff, editor, *Smart Card Research and Advanced Application Conference - CARDIS 2011, 10th Conference, September 15-16, 2011, Leuven, Belgium, Proceedings*, volume 7079 of *Lecture Notes in Computer Science*, pages 182–198. Springer Berlin Heidelberg, 2011.

[108] E. Wenger and M. Hutter. Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations. In P. Laud, editor, *16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26-28, 2011, Revised Selected Papers*, volume 7161 of *Lecture Notes in Computer Science*, pages 256–271. Springer Berlin Heidelberg, 2011.

[109] E. Wenger, T. Korak, and M. Kirschbaum. Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware. In *Workshop on RFID Security 2013 (RFIDSec 2013), July 9-11, Graz, Austria*. Springer, 2013. in press.

[110] E. Wenger, T. Unterluggauer, and M. Werner. 8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors. In G. Paul and S. Vaudeney, editors, *INDOCRYPT*, Lecture Notes in Computer Science. Springer, 2013. in press.

[111] E. Wenger and M. Werner. Evaluating 16-Bit Processors for Elliptic Curve Cryptography. In Prouff [89], pages 166–181.

[112] S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. In *IEEE Transactions on Computers*, volume 49 of *IEEE Transactions on Computers*, pages 967–970. IEEE Computer Society, 2000.

[113] S.-M. Yen, L.-C. Ko, S.-J. Moon, and J. Ha. Relative Doubling Attack Against Montgomery Ladder. In D. Won and S. Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2005.

[114] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[115] ZigBee Alliance. The ZigBee Alliance Website. `http://www.zigbee.org/`.

Part II

Publications

List of Publications

Lecture Notes in Computer Science

1. Erich Wenger, Thomas Unterluggauer, and Mario Werner. 8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors. In Goutam Paul and Serge Vaudeney, editors, *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, Springer, 2013. Note: in press.

   - See page 51.

2. Erich Wenger. A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography. In Gildas Avoine and Orhun Kara, editors, *Second International Workshop on Lightweight Cryptography for Security and Privacy - LightSec 2013*, volume 8162 of *Lecture Notes in Computer Science*, pages 1–15, Springer, 2013.

   - See page 91.

3. Erich Wenger, Thomas Korak, and Mario Kirschbaum. Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware. Michael Hutter and Jörn-Marc Schmidt, editors, *RFIDSec*, volume 8262 of *Lecture Notes in Computer Science*, Springer, 2013. Note: in press.

   - See page 165.

4. Erich Wenger. Hardware Architectures for MSP430-based Wireless Sensor Nodes Performing Elliptic Curve Cryptography. In Michael Jacobson, Michael Locasto, Payman Mohassel and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 290–306, Springer, 2013.

   - See page 71.

5. Erich Wenger and Michael Hutter. Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations. In Peeter Laud, editor, *Information Security Technology for Applications*, volume 7161 of *Lecture Notes in Computer Science*, pages 256–271, Springer, 2012.

- See page 147.

6. Erich Wenger and Mario Werner. Evaluating 16-bit Processors for Elliptic Curve Cryptography. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 166–181, Springer, 2011.

7. Erich Wenger and Michael Hutter. A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9 kGEs. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 182–198, Springer, 2011.

   - See page 127.

8. Michael Hutter and Erich Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems, CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 459–474, Springer, 2011.

   - See page 109.

9. Erich Wenger, Martin Feldhofer, and Norbert Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In Yongwha Chung and Moti Yung, editors, *Information Security Applications – WISA*, volume 6513 of *Lecture Notes in Computer Science*, pages 92–106, Springer, 2011.

IEEE Proceedings

1. Erich Wenger, Thomas Baier, and Johannes Feichtner. JAAVR: Introducing the Next Generation of Security-enabled RFID Tags. In *Euromicro Conference on Digital System Design (DSD)*, pages 640–647, IEEE Computer Society, 2012.

2. Erich Wenger and Johann Großschädl. An 8-bit AVR-Based Elliptic Curve Cryptographic RISC Processor for the Internet of Things. In *International Symposium on Microarchitecture Workshops (MICROW)*, pages 39–46, IEEE Computer Society, 2012.

3. Martin Feldhofer, Manfred Josef Aigner, Michael Hutter, Thomas Plos, Erich Wenger, and Thomas Baier. Semi-Passive RFID Development Platform for Implementing and Attacking Security Tags. In *International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–6, 2010.

Under Review

1. Johann Großschädl, Zhe Liu, and Erich Wenger. MoTE-ECC: Energy-Scalable Elliptic Curve Cryptography for Wireless Sensor Networks. Under review at an international conference.

2. Michael Hutter and Erich Wenger. Fast Multi-precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. Under review at an international journal.

Book Chapter

1. Michael Hutter, Erich Wenger, Markus Pelnar, and Christian Pendl. Elliptic Curve Cryptography on WISPs. In Pedro Peris Lopez, Julio C. Hernandez-Castro, and Tieyan Li, editors, *Security and Trends in Wireless Identification and Sensing Platform Tags: Advancements in RFID*, IGI Global, pages 120–143, August, 2012.

Workshops

1. Erich Wenger. Comparing the Elliptic Curve Digital Signature Algorithm over GF(p) and GF(2m) on an Area-Optimized Custom Microprocessor. In Javier Lopez and Gene Tsudik, editors, *Applied Cryptography and Network Security – Industrial Track*, pages 103–118, 2011.

2. Erich Wenger, Martin Feldhofer, and Norbert Felber. A 16-Bit Microprocessor Chip for Cryptographic Operations on Low-Resource Devices. In *Austrochip – Workshop on Microelectronics*, pages 55–60, 2010. ISBN 978-3-200-01945-4.

Journal Paper

1. Thomas Plos, Manfred Aigner, Thomas Baier, Martin Feldhofer, Michael Hutter, Thomas Korak, and Erich Wenger. Semi-Passive RFID Development Platform for Implementing and Attacking Security Tags. In *International Journal of RFID Security and Cryptography (IJRFIDSC)*, Volume 1, pages 16–24, 2012.

Patent

1. Michael Hutter, Erich Wenger. Multiplication of large operands. WO 2013044276 A1, Filing September 27, 2011.

Chapter 5

8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors

Publication Data

Erich Wenger, Thomas Unterluggauer, and Mario Werner. 8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors. In Goutam Paul and Serge Vaudeney, editors, *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, Springer, 2013. Note: in press.

Contributions

Idea. ATmega clone. Software implementations (except for modified MSP430). Hardware evaluations. Algorithms, Tables, and Figures. 80 % of Text.

8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors

Erich Wenger, Thomas Unterluggauer, and Mario Werner

Graz University of Technology
Institute for Applied Information Processing and Communications
Erich.Wenger@iaik.tugraz.at,
{T.Unterluggauer,M.Werner}@student.tugraz.at

Abstract. The decision regarding the best suitable microprocessor for a given task is one of the most challenging assignments a hardware designer has to face. In this paper, we make a comparison of cycle-accurate VHDL clones of the 8-bit Atmel ATmega, the 16-bit Texas Instruments MSP430, and the 32-bit ARM Cortex-M0+. We investigate their runtime, chip area, power, and energy characteristics regarding Elliptic Curve Cryptography (ECC), one of the practically most resource-critical public-key cryptography systems. If ECC is not implemented with greatest care, its implementation can lead to excruciating runtimes or enable practical side-channel attacks. Considering those important requirements, we present a constant runtime, side-channel protected, and resource saving scalar multiplication algorithm. To tap the full potential of all three microprocessors, we perform assembly optimizations and add carefully crafted instruction-set extensions. To the best of our knowledge, this is the first thorough software and hardware comparison of these three embedded microprocessors.

Keywords: ATmega, MSP430, Cortex-M0+, Elliptic Curve Cryptography, Instruction-Set Extension, Software and Hardware Evaluation.

1 Introduction

Motivation. It is a well-known fact that embedded microprocessors play a significant role within a huge number of consumer, industrial, commercial and military applications. Microprocessors are being produced and deployed in huge numbers and are the beating heart of, e.g., smart cards, wireless sensor networks, or in future even RFID tags. Those applications require solutions that are highly optimized in order to be cheap, energy-efficient, and/or power-efficient, while being versatile and delivering the necessary performance.

To meet all these requirements, the high demands of security and cryptography have too often been disregarded. Especially public-key cryptography needs to be implemented with great care in order to achieve small, performant, and energy-saving solutions. Since RSA and ElGamal based crypto systems simply require too much memory, Elliptic Curve Cryptography (ECC) seems to be the

G. Paul and S. Vaudeney (Eds.): INDOCRYPT 2013, in press, 2013.

best choice. However, ECC is a highly demanding challenge within most applications, and therefore the decision regarding the most suitable microprocessor usually is the most discussed topic within a hardware manufacturer. To evaluate the performance of ECC in software and in hardware, we built VHDL clones of three of the most popular microprocessors.

Related Work. The research community recognized the challenge of efficiently implementing ECC. In this context, we want to cite [19, 20, 22, 33, 42, 45], just to name a few. Those papers presented and used a lot of different approaches to improve the performance of ECC on embedded microprocessors. Unfortunately, in many papers, the authors sacrifice practical crucial properties, e.g., the memory footprint or practical side-channel security threats for the sake of fastest runtimes. Other characteristics like power and energy consumptions are also often neglected. Szczechowiak et al [42] is a welcome exception as they presented measured power values for the ATmega and the MSP430 microprocessor. However, how comparable are those values as both processors were manufactured in different ASIC technologies by different vendors? A fair comparison of the investigated microprocessors must utilize a common design flow, common technologies, and practically secured software implementations.

Our Contribution. In this paper, we perform a systematic and comprehensive approach to evaluate ECC on cycle-accurate VHDL clones[1] of three of the most popular microprocessors: the 8-bit Atmel AVR ATmega, the 16-bit Texas Instruments MSP430, and the 32-bit ARM Cortex-M0+. Our contribution is composed of the following points:

- We derive a point multiplication methodology from previous work which is light-weight and secure against (most) side-channel attacks. The resulting algorithm can be applied to any future embedded designs.
- All our software implementations for the three processors are secure against side-channel attacks and highly optimized using state-of-the-art multi-precision integer multiplication techniques. Runtime, chip area, power, and energy results are given for four different standardized elliptic curves (`secp160-192-224-256r1`).
- We built three cycle-accurate clones of three of the most popular microprocessors and evaluate them in an 130 nm ASIC manufacturing process. The hardware models are based on publicly available design documents and software simulators. It is quite unlikely that Atmel, Texas Instruments, or ARM would have given us their cores for such a comparison. Their chips are produced in different technologies, so any comparison of actual chips is impracticable for our purposes. Regarding code quality, we want to emphasize that Atmel, Texas Instruments, and ARM use similar libraries and tools as we do. Therefore our designs are probably very close to the real deal.
- We are the first to integrate instruction-set extensions (ISE) in actual clones of those microprocessors. The only common denominator of those three processors is the 16-bit instruction-set. In every other key aspect, they differ (e.g.

[1] Closed source for now, done by the authors.

8, 16, 32 bit datapaths, Harvard/Von Neumann architecture, ...). Therefore the multiply-accumulate ISEs have to be carefully crafted for each CPU core.
- We are the first to optimize ECC on the Cortex-M0+.
- Our results represent state-of-the-art of side-channel protected, fast, lightweight, and standardized asymmetric cryptography for embedded processors.

The paper is structured as follows: Section 2 presents and analyzes the side-channel protected elliptic-curve point multiplication algorithm. Within Section 3 the three processors are reviewed and compared on an architectural level. Sections 4 and 5 summarize the assembly and instruction-set optimizations. A rigorous analysis of all implementation results is performed in Section 6. Section 7 concludes the paper.

2 Elliptic Curve Cryptography

When implementing elliptic curve cryptography, a designer has a multitude of options. In the following we present an algorithm for the point multiplication which was chosen based on four characteristics:

- It is easy to **tamper** with embedded microprocessors. Timing attacks, power-analysis attacks and fault-analysis attacks are a real and omnipresent danger. For that matter we will not claim to be secure against all kinds of attacks, but by choosing a methodology that is aware of many kind of attacks, we emphasize the practical significance of the results presented later.
- ECC is a **feature**. Unlike, e.g., the work in [42] (Comb method with window size $w = 4$), we think that only a small fraction of the available program and data memory resource should be used for ECC so that the actual application is not hindered in its operation. Therefore we choose a point-multiplication formula which does not allocate the whole memory for pre-computed or temporary points. Reduced memory requirements further allows hardware designers to save money by equipping the chip with smaller memories.
- Standards were made to be used and simplify the **interoperability** of products. Thus, by choosing a NIST [35] or SECG [7] standard, any company can be sure that their product is compatible with products from other vendors.
- Achieving a **high speed** is an ubiquitous goal of nearly every designer. By getting the most out of an available hardware, one reduces latency times (important in real-time protocols) and saves energy (important for battery-powered devices and from an economic point of view). As we do not sacrifice our security requirements for speed, we concentrate on improving the finite-field operations by doing assembly and instruction-set optimizations.

In Algorithm 1, we present the point multiplication formula used for all our practical evaluations. A detailed analysis of Algorithm 1 is given in Appendix A. Our goal was to design an algorithm which can be used for Diffie-Hellman key exchanges (DHKE) and elliptic-curve based signatures (ECDSA [35]), which are the major features embedded applications actually require. The algorithm

Algorithm 1 Elliptic curve point-multiplication algorithm used for evaluation.

Input: Domain parameters, secret scalar k with MSB = 1, point $P = (x, y)$.
Output: $R = k \times P$

1: **if** $y^2 \neq x^3 + ax + b$ **then** Perform Error Handling
2: $(X, Y, Z) \leftarrow (x \cdot \lambda, y \cdot \lambda, \lambda)$ ▷ Randomize Projective Coordinates
3: **if** $Y^2Z \neq X^3 + aXZ^2 + bZ^3$ **then** Perform Error Handling
4: $Q[0] \leftarrow (X, Z), Q[1] \leftarrow 2 \cdot (X, Y, Z)$ ▷ Initial Point Doubling
5: **for** $i = |k| - 2$ **downto** 0 **do** ▷ Montgomery Ladder
6: $\quad Q[k_i] \leftarrow Q[k_i] + Q[k_i \oplus 1]$
7: $\quad Q[k_i \oplus 1] \leftarrow 2 \cdot Q[k_i \oplus 1]$
8: **end for**
9: $(X, Y, Z) \leftarrow$ `y-recovery`$(Q[0], Q[1])$
10: **if** $Y^2Z \neq X^3 + aXZ^2 + bZ^3$ **then** Perform Error Handling
11: $R = (x, y) \leftarrow (XZ^{-1}, YZ^{-1})$
12: **if** $y^2 \neq x^3 + ax + b$ **then** Perform Error Handling

is using a Montgomery ladder [34] with Randomized Projective Coordinates (RPC) [10] and multiple point validation (PV) checks. After an initial PV check the coordinates are randomized. In step 3, the point is again checked within the projective coordinates. A fault attack on the randomized projective coordinates is much more complex. Then, an initial point doubling within the RPC is performed. The double of the original point is needed for the following Montgomery ladder. Here we use the common-z interleaved point addition and doubling formulas by Hutter, Joye, and Sierra [25]. As this is the most costly part of the algorithm, no PV checks are performed within it. For the following recovery of the y-coordinates, both $Q[0]$ and $Q[1]$ are used. Another two PV checks are performed before and after the inversion of the Z-coordinate. One may argue that several of the PV checks are redundant, but because they hardly have any impact on the speed, we perform them anyways.

Runtimes of all finite-field operations are constant and data-independent. Therefore, a finite-field inversion was implemented based on Fermat's little theorem (inversion by exponentiation). Particularly, the algorithm is based on the exponentiation trick by Itoh and Tsujii [27]. Although this trick is usually applied to elliptic curves over binary fields, we utilize it to optimize the inversion for the standardized Mersenne-like primes, nearly halving the number of multiplications needed for an exponentiation with a fixed exponent.

Summarizing, it is important to utilize the available resources. Therefore a detailed knowledge of the used microprocessors is necessary to achieve competitive results. Section 3 discusses the characteristics of the investigated embedded microprocessors.

3 Microprocessor Architectures

This paper focuses on three of the most popular embedded microprocessors: the 8-bit Atmel ATmega AVR, the 16-bit Texas Instruments MSP430 and the 32-

bit ARM Cortex-M0+ microprocessors. All of them were designed for embedded applications, in which price and power consumption matter more than the maximum clock speed or the amount of available data or program cache. In fact, those RISC processors do not have any cache. In this section, we introduce the three processor architectures and discuss their capabilities relevant for ECC.

Atmel ATmega AVR series. In 1996, two students from the Norwegian Institute of Technology developed the first AVR processor. Today, designers can choose from a vast range of descendants. Especially the ATmega series [2] has been and is used in a magnitude of commercial products.

The ATmega is a 8-bit RISC processor with separated program, data, and I/O memory buses (Harvard architecture). It comes with 32 general-purpose registers (GPR) and 91 (133 including simulated) instructions. To perform integer arithmetic, operands need to be loaded (2 cycles) to the GPRs, processed within the GPRs, and stored back (2 cycles) to the data memory. A for multi-precision integer arithmetic [9] interesting 8-bit multiply-accumulate operation (`LD`, `LD`, `MUL`, `ADD`, `ADC`, `ADC`) takes 9 cycles.

Texas Instruments MSP430 series 1. One of the most successful, direct competitor of the ATmega is the MSP430 processor series [43] by Texas Instruments. With its six low-power modes it is most interesting for low power, and low-energy applications. This is why it is used for many wireless sensor nodes such as the EPIC Mote, TelosB, T-Mote Sky, and XM1000 platforms.

The original series-1 MSP430 is a 16-bit RISC processor with a single combined data and program bus. Merely 12 of its 16 16-bit registers are actually usable as general-purpose registers. The MSP430 series comes with a fully orthogonal instruction set of only 27 instructions with 7 addressing modes. Unfortunately, there is no dedicated multiplication instruction, but a memory mapped $16 \times 16 \rightarrow 32$-bit multiplier, with multiply-accumulate feature, is available. Despite the high costs introduced by transfering the operands from data memory to the multiplier, a 16-bit multiply-accumulate operation (`MOV`, `MOV`, `NOP`, `ADD`) can be performed in 13 cycles.

ARM Cortex-M0+ series. In the recent years, ARM made a name for itself with supplying smart phones and tablets with powerful energy-saving processors, namely the Cortex-A series. For embedded applications however, Cortex-M series processors are more suitable. The Cortex-M0+ embedded microprocessor [1] is the smallest, most energy-efficient ARM ever built and supports a subset of the Thumb-2 instruction-set. This 32-bit RISC processor is designed as direct competitor for the ATmega and MSP430 processors. Launched in 2012, major companies (e.g., Freescale [15], Fujitsu [16], or NXP [36]) just started to introduce the Cortex-M0+ to their lineups.

Similar to the MSP430, the Cortex-M0+ comes with a Von Neumann architecture. Its 32-bit address and data buses enable the addressing of up to 4 GByte of data, preparing it perfectly for future memory requirements. Exactly 13 of its 16 32-bit registers can be used as general-purpose registers, but most of its 56 instructions can only access the lower 8 registers `R0`–`R7`. Registers `R8`–`R15` are accessible through a `MOV` instruction only. Optionally, the Cortex-M0+ comes with

Table 1. Summary of the embedded microprocessors.

Characteristic	**ATmega**	**MSP430** (1 Series)	**Cortex-M0+**
Data-Width	8 bits	16 bits	32 bits
Instruction-Word Size	16 bits	16 bits	16 bits
Architecture	Harvard	Von Neumann	Von Neumann
General-Purpose Registers	32	12/16	8/13/16
Number of Instructions	81	27	56
Max. Data Memory	16 kByte	10 KByte	4 GByte
Max. Program Memory	256 kByte	60 KByte	4 GByte
Multiply Accumulate[a]	9 cycles	13 cycles	29 cycles
Used Clone	JAAVR [47]	IDLE430 [50]	Xetroc-M0+ [44]
Core area[b]	6,140 GE	4,913 GE	15,262 GE
Registers	2,002 GE	1,732 GE	4,176 GE
Multiplier	372 GE	1,751 GE[c]	2,766 GE

[a] Load two operands, multiply and accumulate them.
[b] Including memory arbiter and necessary special function registers.
[c] Memory mapped and therefore not part of the core area.

a bit-serial or a single-cycle $32 \times 32 \rightarrow 32$-bit multiplication instruction. Note that for ECC, also the upper half of the product is necessary for multi-precision integer arithmetic. In the following, we use this multiplier as $16 \times 16 \rightarrow 32$-bit multiplier. Section 4 illustrates the implementation of an efficient 29-cycle 32-bit multiply-accumulate operation.

Summary of the Embedded Microprocessors. The common denominators of the three embedded microprocessors are that they are RISC processors, support single-cycle register-to-register operations, and have 16-bit instruction sets (with some 32-bit exceptions). The major differences are summarized in Table 1. The three microprocessors utilize different architectures, have different amounts of available registers, clearly distinct instruction sets and support different amounts of data and program memory. Those differences become apparent when the actual hardware footprint is evaluated. Most remarkably, the 16-bit MSP430 (4,913 GE) requires less chip area than the 8-bit ATmega (6,140 GE). This is the price the ATmega has to pay for its three memory buses and the vast instruction set. To efficiently perform integer multiplications, the MSP430 additionally needs a memory mapped multiplier which is 1,751 GE in size. Compared to the ATmega and the MSP430, the 32-bit Cortex-M0+ is much larger. It requires 15,262 GE in a configuration with a single-cycle 32-bit multiplier. The optional 32-cycle bit-serial multiplier saves 1,363 GE which brings the Cortex-M0+ to a minimum size of 13,899 GE in the used 130 nm UMC process.

Related Work. ARM specifies that their Cortex-M0+ is only 12 kGE large in a 90 nm process. When synthesizing our clone in a 90 nm UMC process it requires 12,436 GE. So in terms of area, our Cortex-M0+ is (as aimed for) very close to the original. As neither Atmel nor Texas Instruments released characteristics of their processors, we can only compare our clones to the versions

uploaded to `opencores.org`. Compared to the openMSP430 [37], our MSP430 is $5,958 - 4,913 = 1,045$ GE smaller. Compared to other (insufficiently tested) ATmega or AVR clones, our ATmega clone is similarly small.

4 Assembly Optimizations for ECC

As we already fixed the point arithmetic, we focus our effort on the finite-field operations. Most crucial is the finite-field multiplication as it contributes to 90 % of the runtime of a point multiplication. Hence, optimizing the runtime of the finite-field multiplication automatically improves the runtime of the point-multiplication algorithm. Additionally, it leads to a speedup of the finite-field squaring, exponentiation and inversion operation. To optimize the finite-field multiplication, one can either perform assembly optimizations or extend the instruction-set (see Section 5).

While the currently fastest multi-precision multiplication approaches for the ATmega and the MSP430 are based on related work, it was necessary to come up with a new technique for the Cortex-M0+ as it has not yet been investigated.

ATmega. In 2004, Gura *et al.* [22] presented an efficient multi-precision multiplication method and applied it to the ATmega. Hutter and Wenger [26] presented the "Operand Caching" method in 2011. It further reduced the number of memory load operations at the cost of some memory store operations. As it fully utilizes the available general purpose registers as caches and currently is one of the fastest ways to perform multi-precision multiplications on an ATmega, we used their technique.

MSP430. The MSP430 only has 12 useable GPRs, of which three registers have to be used as pointers and further three registers are necessary for the accumulation of intermediate results. With the remaining six registers, we applied the product-scanning technique by Comba [9]. This technique fully utilizes the multiply-accumulate functionality of the memory-mapped multiplier. It was first described by Gouvêa and López [19] and is fully tailored to the MSP430.

Cortex-M0+. ECC performance of the Cortex-M0+ has never been examined before. Most notable are the works of Aydos *et al.* [3], who optimized ECC for an ARM7TDMI processor, and Bernstein and Schwabe [4], who optimized the NaCl cryptographic library for a Cortex-A8. However, none of their work had to deal with the limitations of a Cortex-M0+: Its multiplier only computes 32-bit products, most instructions are restricted to registers `R0`–`R7`, and, for most instructions, the destination register must equal one of the source registers, i.e. in each multiplication one of the operands is overwritten by the product.

We evaluated several multiplication techniques and finally settled for a product-scanning multi-precision multiplication method. Its centerpiece is shown in Algorithm 2: the two operand references are moved from registers `R8` and `R9` to `R1` and `R2`. Consequently, we can load their values from the memory. Then, five registers are available to perform four $16 \times 16 \rightarrow 32$-bit multiplications (steps 9–13), whereby the 16-bit masking steps are performed only once (steps 5–7). Steps 14–27 accumulate the four 32-bit products into the registers `R3`–`R5`.

Algorithm 2 Multiply-Accumulate Operation on Cortex-M0+.

Input: R8 and R9 are pointers.
Output: R3, R4, R5 contain the sum.

```
 1: MOV  R1, R8
 2: LDR  R1, [R1, #offset1]
 3: MOV  R2, R9
 4: LDR  R2, [R2, #offset2]
 5: UXTH R6, R1
 6: UXTH R7, R2
 7: LSR  R1, R1, #16
 8: LSR  R2, R2, #16
 9: MOV  R0, R6
10: MUL  R0, R0, R7      ▷ Low–Low
11: MUL  R6, R6, R2      ▷ Low–High
12: MUL  R2, R2, R1      ▷ High–High
13: MUL  R1, R1, R7      ▷ High–Low
14: MOV  R7, #0
15: ADD  R3, R3, R0      ▷ Low–Low
16: ADC  R4, R4, R2      ▷ High–High
17: ADC  R5, R5, R7
18: LSL  R0, R6, #16     ▷ Low–High
19: LSR  R2, R6, #16
20: ADD  R3, R3, R0
21: ADC  R4, R4, R2
22: ADC  R5, R5, R7
23: LSL  R0, R1, #16     ▷ High–Low
24: LSR  R2, R1, #16
25: ADD  R3, R3, R0
26: ADC  R4, R4, R2
27: ADC  R5, R5, R7
```

The stack is used to temporarily store the product of the multi-precision integer multiplication. Hence, we benefit from addressing relative to the stack pointer and avoid moving the address to one of the registers R0–R7. In a second step, a reduction is performed by taking advantage of the Mersenne-like primes.

Assembler Optimized Results. We did all assembly optimizations for four standardized elliptic curves (secp160-192-224-256r1). In order to keep the general view, Table 2 depicts the memory footprints, the runtimes for finite-field operations and the point multiplication over the secp160r1 elliptic curve.

In terms of **ROM** size the processors behave converse their word sizes. The implementation for the ATmega takes up 7,762 bytes in ROM, which is twice as much as for the Cortex-M0+. This is mainly a result of the unrolled integer multiplication. The MSP430 behaves well as it requires only 13% more ROM than the Cortex-M0+. With respect to **RAM**, the ATmega (402 byte) and the Cortex-M0+ (404 byte) perform similarly, consuming about 40% more RAM than the MSP430 (290 byte). The increased RAM footprint of the ATmega is due to the elliptic curve constants that need to be loaded to the RAM at startup. The root of the increased memory usage of the Cortex-M0+ lies within its calling hierarchy. Each PUSH operation stores four bytes of data within the RAM. These facts combined make the MSP430 an economic platform in terms of memory footprint and chip area.

Impressingly, the **finite-field addition** on the Cortex-M0+ is 2.6 times faster than an addition on the MSP430. The main reason for that is the load-multiple LDM instruction of the Cortex-M0+, which allows loading multiple words into several registers requiring only $\#words + 1$ cycles. The load LDR instruction takes 2 cycles per word and therefore $2 \times \#words$ cycles would be needed.

Table 2. Benchmark data of assembly optimized implementations for `secp160r1`.

Processor	ROM [Bytes]	RAM [Bytes]	Addition [Cycles]	Mult. [Cycles]	Inversion [Cycles]	Point Mult. [Cycles]	Core Area [kGE]
Atmega	8,358	402	291	3,024	519,217	9,230,048	6,140
MSP430	4,788	290	163	1,905	327,366	5,779,957	7,003
CortexM0+	4,256	404	62	942	162,500	2,809,619	15,262
Relative Performance							
Atmega	1.96	1.46	4.69	3.21	3.20	3.29	1.00
MSP430	1.13	1.00	2.63	2.02	2.01	2.06	1.14
CortexM0+	1.00	1.39	1.00	1.00	1.00	1.00	2.49

As the runtime of **integer multiplication** scales quadratically, one expects the 32-bit Cortex-M0+ to be four times faster than the 16-bit MSP430 and the 16-bit MSP430 to be four times faster than the 8-bit ATmega. But they are not. The Cortex-M0+ is only twice as fast as the MSP430. The reason for that is the tremendous overhead needed to perform a $32 \times 32 \rightarrow 64$-bit multiplication using the $32 \times 32 \rightarrow 32$-bit integer multiplier (see the highly optimized Algorithm 2). But also the MSP430 is only 1.6 times faster than the ATmega. This is because the MSP430 has a memory mapped multiplier and the ATmega has an integrated multiplier.

By combining finite-field additions, multiplications, and inversions, the runtimes for `secp160r1` **point multiplications** were obtained. They are 9.2 million cycles for the ATmega, 5.8 million cycles for the MSP430, and 2.8 million cycles for the Cortex-M0+. As the finite-field multiplication contributes to the majority of this runtime, the ratios for the point multiplications are nearly identical to the ratios of the finite-field multiplication. Equipping the Cortex-M0+ with a bit-serial multiplier quadruples its runtime: With 11.9 million cycles the bit-serial multiplier simply defeates the purpose of having a 32-bit processor. Hence, we do not recommend implementing prime-field based ECC on a Cortex-M0+ without single-cycle multiplier. Consequently, instruction-set extensions were equipped to improve runtimes and to monitor how the performance ratios change.

5 Instruction-Set Modifications

We carefully crafted the VHDL clones of the ATmega, the MSP430, and the Cortex-M0+ to be cycle-compatible with its originals. During that process, we also observed some minor shortcomings regarding their respective potentials. This section is all about maximizing the performance by improving the runtime of existing instructions and adding multiply-and-accumulate [21] instructions. This `MULACC` instruction is optimal for multi-precision multiplication. It is used to multiply two n-bit registers and add the 2n-bit product to three accumulation registers. As the three processors differ significantly, `MULACC` had to be integrated differently for each processor.

ATmega. Our modifications of the ATmega are based on Wenger [46]. In this paper, we showed among other things how to improve load, store, and multiply

Table 3. Benchmarks of processors with instruction-set modifications for `secp160r1`.

Processor	ROM [Bytes]	RAM [Bytes]	Addition [Cycles]	Mult. [Cycles]	Inversion [Cycles]	Point Mult. [Cycles]	Core Area [kGE]
Atmega	5,828	402	176	984	170,053	3,268,486	7,039
MSP430	3,898	286	150	718	123,939	2,445,508	7,197
CortexM0+	3,088	408	62	369	64,859	1,231,946	18,700
Relative Performance of Modified Processors							
Atmega	1.89	1.41	2.84	2.67	2.62	2.65	1.00
MSP430	1.26	1.00	2.42	1.95	1.91	1.99	1.02
CortexM0+	1.00	1.43	1.00	1.00	1.00	1.00	2.66
Modified versus Assembly-optimized Implementations (Table 2)							
Atmega	−30.3%	±0.0%	−39.5%	−67.5%	−67.2%	−64.6%	14.6%
MSP430	−18.6%	−1.4%	−8.0%	−62.3%	−62.1%	−57.7%	2.8%
CortexM0+	−27.4%	1.0%	±0.0%	−60.8%	−60.1%	−56.2%	22.5%

instructions and execute them within a single cycle instead of two. Additionally, we added a single-cycle multiply-and-accumulate instruction, which was combined with the Operand-Caching multiplication method. In a special operating mode, activated by writing a memory-mapped configuration register, the existing `MUL` instruction is reinterpreted as `MULACC` instruction. Therefore, the software toolchain does not need to be modified. In fact, the trick of having a special mode for the instruction-set extension has also been applied for the MSP430 and Cortex-M0+.

MSP430. The advantage of the MSP430 is, that operands do not have to be explicitly loaded to core registers before their usage. The drawback is that the multiplier is only accessible via the memory. To get rid of this bottleneck, we removed the memory-mapped multiplier (saved 1,751 GE) and added a dedicated `MULACC` instruction within its core. Unfortunately, the 7 existing addressing modes were insufficient for our purposes. By perfectly utilizing the pre-existing auto-increment and a new auto-decrement addressing mode it is possible to load two operands, multiply-and-accumulate the data, and update the addressing registers. To omit manual pointer updates completely, we combined the new `MULACC` instruction with Wenger and Werner's [49] zig-zag product-scanning multiplication method. Other modifications with less impact on the ECC runtime improved move, jump, push, and call instructions by one to two cycles. This modifications do not only minimize the overhead of the C-calling convention, but also potentially improve the runtimes of any other algorithm run on the modified MSP430.

Cortex-M0+. As it is only possible to compute a $16 \times 16 \rightarrow 32$-bit product with the internal multiplier of the Cortex-M0+ and 29 cycles are necessary to perform a 32-bit (c.f. Algorithm 2) multiply-and-accumulate operation, it is specially important to equip the Cortex-M0+ with a `MULACC` extension. This extension reduced the for the product-scanning important sequence of `LDR`, `LDR`, `MULACC` instructions to mere 5 cycles. To save area, the pre-existing $32 \times 32 \rightarrow 32$-bit multiplier is reused and only extended to compute a 64-bit product. Therefore, the `MULACC` extension did only cost 3.4 kGE.

Results using Instruction-Set Modifications. In general, the core idea of all modifications was to improve the performance without adding unnecessary hardware. So the modifications of the ATmega (+14.8 %) and the Cortex-M0+ (+22.5 %) only marginally increased the size of the CPU cores. The effective size of the MSP430 only increased by 2.8 %. The slow, memory-mapped multiplier was approximately as large as the new dedicated datapath to multiply-and-accumulate within a single cycle. While the size of the CPU cores increased, the size of the program memory decreased by 19-30 %. The rather large unrolled integer multi-precision multiplication functions shrunk significantly and therefore the total chip areas actually decreased. However, the modifications only have very little impact on data memory utilization.

As intended, the modifications achieve a massive speedup of multiplications in the prime field (cf. Table 3). Throughout, the corresponding runtimes dropped by 60%, with the highest speedup achieved on the ATmega (-67%). As inversions are based on exponentiation to counteract side-channel attacks, the same impressing speedup is found there. Accordingly, point multiplication runtimes slumped by 65% on the ATmega and plunged by 57% on the others. Concerning addition, there are no performance gains for the Cortex-M0+ and the MSP430. Contrary to that, addition is performed 40% faster on the ATmega due to the improved timings of the load and store operations. Relating runtimes of the three modified processors, the Cortex-M0+ again achieves the best performance, being between 2-3 times faster than its competitors. However, its advantage diminishes slightly compared to the unmodified ATmega.

6 Discussion of Hardware Implementations

All our implementations for the three microprocessors, with and without instruction-set modifications, and over four elliptic curves were tested for correctness using externally generated test vectors, synthesized in a Faraday UMC 130 nm low-leakage ASIC library, placed-and-routed, and power-simulated (using Cadence RTL Compiler, Cadence Encounter). The huge number of results are accumulated within Table 4 and discussed in the following.

Memories. We used area-efficient single-port RAM macros as data memories and single-port Via-1 ROM macros as program memories. As their necessary sizes depend on the ECC implementation, they were chosen appropriately for each implementation. Experiments showed that synthesizing the program memories as standard logic cells resulted in smaller program memories after synthesis, but there were two problems: Firstly, it was virtually impossible to place and route the program memories without significantly decreasing the cell density, which actually increased the effective size of the program memory. Secondly, the ROM macros have a significantly lower power consumption compared to the synthesized program memories.

Runtime. The runtime is measured at 10 MHz and visualized in Figure 1. The time for a single, side-channel secured point multiplication varies between 123–923 ms. As expected, the 32-bit processor is faster than the 16-bit processor,

Table 4. Summary of all experiments.

Processor	Version	Program Memory [Bytes]	Data[a] Memory [Bytes]	Chip Area ROM [GE]	RAM [GE]	Core [GE]	Total [GE]	Power @10 MHz [µW]	Energy [µJ]	Runtime @10 MHz [ms]
`secp160r1`										
ATmega	ASM	8,358	402	11,807	3,754	6,140	21,701	545	503	923
MSP430	ASM	4,788	290	7,796	3,250	7,003	18,048	583	337	578
CortexM0+	ASM	4,256	404	8,270	4,308	15,262	27,840	718	202	281
ATmega	ISE	5,828	402	8,202	3,754	7,039	18,995	666	218	327
MSP430	ISE	3,898	286	6,363	3,225	7,197	16,786	794	194	245
CortexM0+	ISE	3,088	408	6,416	4,334	18,700	29,450	1,306	161	123
`secp192r1`, **NIST P-192**										
ATmega	ASM	10,238	462	11,807	4,107	6,140	22,054	556	839	1,509
MSP430	ASM	5,408	330	8,202	3,475	7,003	18,679	581	533	918
CortexM0+	ASM	4,860	448	8,270	4,560	15,262	28,092	716	329	459
ATmega	ISE	6,564	462	10,040	4,107	7,039	21,186	670	336	502
MSP430	ISE	4,142	330	7,796	3,475	7,197	18,468	801	283	353
CortexM0+	ISE	3,164	444	6,416	4,535	18,700	29,652	1,318	241	183
`secp224r1`, **NIST P-224**										
ATmega	ASM	12,570	526	15,484	4,485	6,140	26,109	571	1,326	2,321
MSP430	ASM	6,294	374	10,040	3,750	7,003	20,792	584	819	1,403
CortexM0+	ASM	5,672	496	8,270	4,838	15,262	28,369	716	496	693
ATmega	ISE	7,600	526	10,040	4,485	7,039	21,564	664	500	754
MSP430	ISE	4,588	370	7,796	3,725	7,197	18,718	805	419	521
CortexM0+	ISE	3,352	492	6,416	4,812	18,700	29,929	1,330	334	251
`secp256r1`, **NIST P-256**										
ATmega	ASM	16,112	590	17,029	4,838	6,140	28,006	548	1,914	3,493
MSP430	ASM	8,378	418	11,878	4,000	7,003	22,881	580	1,286	2,217
CortexM0+	ASM	7,168	540	10,123	5,089	15,262	30,475	719	771	1,073
ATmega	ISE	9,596	590	11,807	4,838	7,039	23,684	655	779	1,190
MSP430	ISE	6,168	416	10,040	3,975	7,197	21,212	791	717	907
CortexM0+	ISE	4,124	536	8,270	5,064	18,700	32,034	1,339	546	408

[a] Including Stack.

which in turn is faster than the 8-bit processor. Quite remarkably though is that the modified ATmega is nearly as fast as the native Cortex-M0+.

Area. The area visualized in Figure 1 accumulates the respective areas of the CPU, the ROM, the RAM, and core building blocks, such as an arbiter. Quite remarkably, the native and the modified MSP430 represent the smallest implementation, requiring around 16.8–18.0 kGE. Compared to that, the modified Cortex-M0+ (29 kGE) is 75 % larger.

Area-runtime-product. In the prestigious category of area-runtime-product, the modified implementations clearly outperform its native counterparts (see the dashed lines within Figure 1). The modified Cortex-M0+ system performs best, and the native ATmega system performs worst. However, the modified ATmega system provides a better performance for the used chip area than the native Cortex-M0+. Consequently, if some commercial company evaluates whether to switch to a more powerful Cortex-M0+, we can clearly recommend to replace the native MSP430 or ATmega with its modified counterpart,

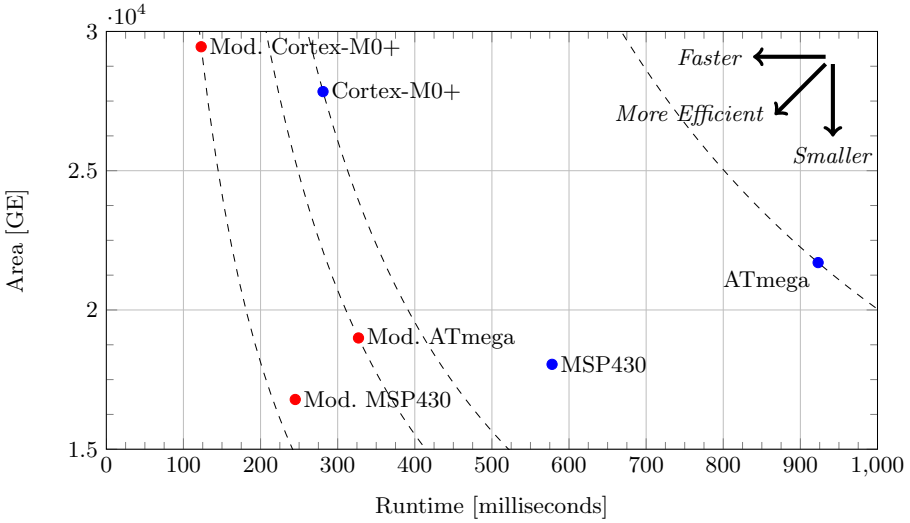

Fig. 1. Area-runtime-characteristics for `secp160r1` at 10 MHz.

presented within this paper. As nice side-effect, the software code-base does not have to be updated for a different processor.

Power. According to Figure 2, all designs require between 545–1,305 µW. The 8-bit ATmega requires the least amount of power, slightly less (6.5 %) than the MSP430. However, when their modified counterparts are compared, the modified ATmega needs 16 % less power than the modified MSP430. The Cortex-M0+ and the modified Cortex-M0+ need the most power.

Power-runtime-product: Energy. However, the same two processors shine within the energy-efficiency race. As represented by the dashed lines in Figure 2, the Cortex-M0+ based designs only need 161–202 µJ, while the other designs need 194–503 µJ. That is up to 60 % less. The MSP430 is 11–33 % more energy efficient than the ATmega.

Relating the Different Elliptic Curves. As initially stated and depicted in Table 4, we did not do our evaluation only with `secp160r1`, but also with `secp192r1`, `secp224r1`, and `secp256r1`. Most importantly, the results observed at the 80-bit security level are reproducible for the larger elliptic curves. On average, changing from one elliptic curve to the next larger one, costs 6 % of additional chip area, 53 % of additional runtime and 54 % of additional energy. The power consumption is not effected by the chosen elliptic curve.

Related Work. In terms of software implementations (c.f. Table 5), our implementations distinct themselves from related work with their low memory footprints and the side-channel countermeasures. In fact none of the implementations done by [20, 22, 33, 42, 45] have side-channel countermeasures built in. Therefore it is expected that, e.g., [20, 22, 42], achieve faster runtimes than we

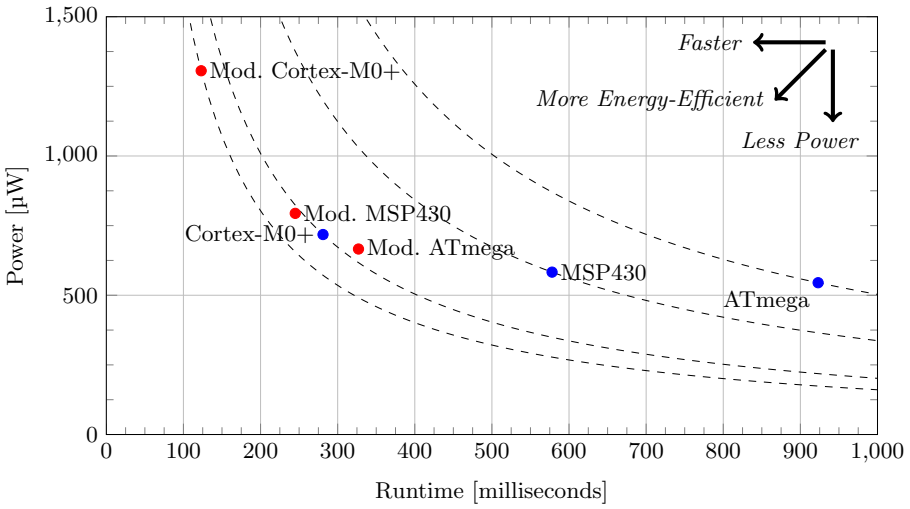

Fig. 2. Power-runtime-characteristics for `secp160r1` at 10 MHz.

do. However, e.g., [20, 42] need up to 7–10 times more program and data memory than we do.

For the sake of completeness, we also compare our modified processors with related hardware implementations (c.f. Table 6). Unfortunately, those dedicated hardware designs are faster than our flexible microprocessor based designs. However, in terms of chip area, our smallest modified MSP430 implementation is smaller than the work of [17, 24, 29, 38, 39]. Only the custom microprocessor design by Wenger *et al.* [48] is smaller. However, their microprocessor does not come with the vast (open-source) compiler toolchains, the ATmega, the MSP430, or the Cortex-M0+ provide.

7 Conclusion

In this work, three of the most popular micro-processors were evaluated regarding their runtime, chip area, power, and energy consumption on standard-compatible side-channel protected elliptic curve cryptography. By comparing them using a single design flow, the same application, and with a common technology, we achieve a fair comparison between the different architectures. Our work might help any system architect on their decision regarding best suitable processor, best suitable security level, and whether or not to implement hardware extensions. Our results show that the Cortex-M0+ is the fastest and most energy-efficient processor (e.g., ideal for Wireless Sensor Nodes), the MSP430 enables the smallest and least power consuming hardware design (e.g., ideal for RFID tags), and the ATmega gains the most performance when instruction-set modifications are applied (e.g., ideal for long-lived products that must be

Table 5. Comparison with related software implementations (80 bit security level).

Curve	ROM [Bytes]	RAM [Bytes]	Runtime [kCycles]
ATmega			
Custom [42]	46,100	1,800	9,376
`secp160r1` [33]	20,768	1,774	15,060
`secp160r1` [22]	3,682	282	6,480
Our `secp160r1`	8,358	402	9,230
MSP430			
Custom [42]	31,300	2,900	5,898
`secp160r1` [20]	23,300	2,800	2,528
`secp160r1` [33]	16,218	1,866	11,821
`secp160r1` [45]	12,500	1,300	28,080
Our `secp160r1`	4,788	290	5,780

Table 6. Comparison with related hardware implementations.

Implementation	Area [GE]	Runtime [kCycles]
96-bit security level		
Satoh *et al.* [39]	29,655	4,165
Hutter *et al.* [24]	19,115	859
Wenger *et al.* [48]	11,686	1,377
80-bit security level		
Öztürk *et al.* [38]	30,333	545
Fürbass *et al.* [17]	23,656	500
Kern *et al.* [29]	18,247	512
Our Mod. ATmega	18,995	3,268
Our Mod. MSP430	16,786	2,446
Our Mod. Cortex-M0+	29,450	1,232

equipped with ECC). Any designer now has to define their own metric and weigh the characteristics with each other. To the best of our knowledge, such an comprehensive evaluation has not been done before.

Acknowledgments

This work has been supported in part by the Austrian Government through the research program FIT-IT under the project number 835917 (project NewP@ss) and by the European Commission through the ICT Programme under contract ICT-SEC-2009-5-258754 TAMPRES.

References

1. ARM. Cortex-M0+ Processor, 2013. Available online at `http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php`.
2. Atmel Corporation. megaAVR Microcontroller, 2013. Available online at `http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx`.
3. M. Aydos, T. Yanik, and Ç . K. Koç. A High-Speed ECC-based Wireless Authentication Protocol on an ARM Microprocessor. In *ACSAC*. IEEE, 2000.
4. D. Bernstein and P. Schwabe. Neon crypto. *CHES*, 2012.
5. I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *CRYPTO*, LNCS. Springer, 2000.
6. B. Brumley and N. Tuveri. Remote timing attacks are still practical. *ESORICS*, 2011.
7. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, 2000.

8. M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography*, 2005.
9. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 1990.
10. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES*, LNCS. Springer, 1999.
11. N. Ebeid and R. Lambert. Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In *FDTC*, pages 46–50. IEEE Computer Society, 2009.
12. J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on known Side-Channel Attacks and Countermeasures. In *HOST*. IEEE, 2010.
13. J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications*, LNCS. Springer, 2012.
14. P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack onElliptic Curve Montgomery Ladder Implementation. In *FDTC*. IEEE Computer Society, 2008.
15. Freescale Semiconductor. Kinetis L Series MCUs, 2013. Available online at `http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=KINETIS_L_SERIES`.
16. Fujitsu Semiconductors. Fujitsu Semiconductor Widely Expands Lineup of 32-bit General Purpose Microcontrollers with the Release of Products Adopting 2 New ARM Cores, November 2012. Press Release.
17. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *IEEE International Symposium on Circuits and Systems*, 2007.
18. L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *PKC*, LNCS, 2003.
19. C. P. L. Gouvêa and J. López. Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller. In *INDOCRYPT*, 2009.
20. C. P. L. Gouvêa, L. Oliveira, and J. López. Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller. *Journal of Cryptographic Engineering*, 2012.
21. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES*, 2004.
22. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In *CHES*, LNCS. Springer, 2004.
23. J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl. Localized Electromagnetic Analysis of Cryptographic Implementations. *CT-RSA*, 2012.
24. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In *RFIDSec*, 2010.
25. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In *AFRICACRYPT*, 2011.
26. M. Hutter and E. Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In *CHES*, LNCS. Springer, 2011.
27. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, 1988.
28. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *CHES*, LNCS, 2003.
29. T. Kern and M. Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *ICECS*, pages 1236–1239. IEEE, 2010.

30. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, LNCS. Springer, 1996.
31. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, LNCS. Springer, 1996.
32. A. K. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 1982.
33. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks*, 2008.
34. P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 1987.
35. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009.
36. NXP Semiconductors. NXP Licenses ARM Cortex-M0+ Processor, March 2012. Press Release.
37. Olivier Girard. openMSP430, 2013. Available online at `http://opencores.org/project,openmsp430`.
38. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In *CHES*, LNCS, pages 92–106, 2004.
39. A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Transactions on Computers*, 2003.
40. J.-M. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In *FDTC*. IEEE Computer Society, 2008.
41. J.-M. Schmidt and M. Medwed. A Fault Attack on ECDSA. In *FDTC*. IEEE, 2009.
42. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks 5th European Conference*, LNCS. Springer, 2008.
43. Texas Instruments. MSP430 Ultra-Low Power 16-Bit Microcontrollers, 2013. Available online at `http://www.ti.com/msp430`.
44. T. Unterluggauer. Xetroc-M0+. An implementation of ARMs Cortex-M0+. Master project, Graz University of Technology, 2013.
45. H. Wang, B. Sheng, and Q. Li. Elliptic Curve Cryptography-based Access Control in Sensor Networks. *International Journal of Security and Networks*, 2006.
46. E. Wenger. A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography. In *LightSec*, LNCS, pages 1–15, 2013.
47. E. Wenger, T. Baier, and J. Feichtner. JAAVR: Introducing the Next Generation of Security-Enabled RFID Tags. In *DSD*, pages 640–647. IEEE, 2012.
48. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In *WISA*, 2010.
49. E. Wenger and M. Werner. Evaluating 16-bit Processors for Elliptic Curve Cryptography. In *Smart Card Research and Advanced Applications*. Springer, 2011.
50. M. Werner. IDLE430 - an ImproveD msp LikE processor. Master project, Graz University of Technology, 2013.
51. S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. In *IEEE Transactions on Computers*. IEEE, 2000.

A Analysis of Point Multiplication Formula

The following analysis discusses how Algorithm 1 holds up against the most common side-channel attacks. It is based on the overview papers of Fan *et al.* [12, 13]. Attacks that are considered not to affect the security of the given algorithm:

Timing analysis [30] is not possible, because the used Montgomery Ladder [25] has a key-independent runtime, and all finite-field operations have a constant runtime as well. To avoid leading-zero-bits timing attacks [6] based on the LLL algorithm [32], we set the most-significant bit of the secret ephemeral scalar to one.

Simple power analysis [31] is hindered by using a Montgomery Ladder and Randomized Projective Coordinates.

Differential power analysis [31], Refined power analysis [18] are not possible as random ephemeral scalars are used for DHKE and ECDSA.

M & C safe-error analysis [28, 51] are not possible because a Montgomery ladder in conjunction with random ephemeral scalars is used.

Invalid point analysis [5], Twist-curve based analysis [14] is not possible because in lines 1, 3, 10, and 12 point-validity checks are performed.

Subgroup analysis [5] is not possible because an y-recovery with subsequent point verification is performed. Ebeid and Lambert [11] provide a thorough analysis of the y-recovery as countermeasure.

Attacks that may affect the security of the given algorithm:

Program-flow fault analysis [40, 41]. A fault attack applied on the program flow can hardly be detected by the program flow itself. To circumvent such paths of attacks, hardware countermeasures are recommended.

Invalid-curve analysis [8] has to be additionally handled by checking the validity of the stored curve parameters. This is not done within Algorithm 1, but the point-validity checks certainly handicap any invalid-curve attack.

Electromagnetic-emanation analysis [23] is possible if the attacker can detect which memory locations are accessed at certain points in time. A countermeasure would be to randomize the memory access patterns.

Chapter 6

Hardware Architectures for MSP430-based Wireless Sensor Nodes Performing Elliptic Curve Cryptography

Publication Data

Erich Wenger. Hardware Architectures for MSP430-based Wireless Sensor Nodes Performing Elliptic Curve Cryptography. In Michael Jacobson, Michael Locasto, Payman Mohassel and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2013.

Contributions

Main author.

Hardware Architectures for MSP430-based Wireless Sensor Nodes Performing Elliptic Curve Cryptography

Erich Wenger

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
erich.wenger@iaik.tugraz.at

Abstract. Maximizing the battery lifetime of wireless sensor nodes and equipping them with elliptic curve cryptography is a challenge that requires new energy-saving architectures. In this paper, we present an architecture that drops a hardware accelerator between CPU and RAM. Thus neither the CPU nor the data memory need to be modified. In a detailed comparison with a software-only and a dedicated hardware architecture, we show that the drop-in concept is smaller than the dedicated hardware module, while achieving similarly fast runtimes. Most interesting for micro-chip manufacturers is that only 4 kGE of chip area need to be committed for the dedicated drop-in accelerator.

Keywords: MSP430, ASIC, Hardware, Software, Elliptic Curve Cryptography, Wireless Sensor Nodes.

1 Introduction

Privacy, authenticity, and confidentiality pose three of the most challenging current demands on wireless sensor networks. To solve those requirements the use of cryptography is essential. Unfortunately, it is hardly possible to solve this challenge using only symmetric cyphers. The most promising solutions are based on asymmetric cryptography, in particular Elliptic Curve Cryptography (ECC).

Efficiently implementing ECC is a complex task, especially when a designer also needs to be aware of the capabilities of the entities of a sensor network: A sensor node usually comes with a microprocessor, a sensor (e.g., for humidity), a wireless communication interface (e.g., IEEE 802.15.4 [16], ZigBee [31]), and a battery, which should keep the sensor-node alive for a lifetime (some years) within a hostile environment. This means that a solution to the initial requirements should be light-weight and efficient. For maximizing the battery live and keeping the price of a sensor node at a minimum, ECC has to be implemented with care. To realize the scope of the difficulty, be aware that within the time

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 290–306, 2013.
The original publication is available at http://link.springer.com/chapter/10.1007/978-3-642-38980-1_18

required for a single elliptic-curve point multiplication, several hundreds of symmetric encryptions and decryptions can be preformed. Thus ECC has a major impact on both communication latency and energy consumption.

A lot of research has been focused on efficiently and securely implementing ECC. The research is performed based on three different approaches: one is based on efficiently implementing ECC in software, one is based on adding dedicated hardware, and one is a combination of the two preceding approaches. Several papers discuss the use of assembly optimizations [12], instruction-set extensions [6, 10], and dedicated ECC hardware designs [19, 20]. The drawback of those techniques are the relatively low performance, the requirement to change the microprocessor, and the potential waste of precious chip area, respectively. As CPU vendors usually do not give away the source code of microprocessors, but obfuscated code instead, adding new instructions is a troublesome task. Dedicated hardware modules provide locally optimized solutions, but ignore the existence of already available hardware modules. Our paper fills this gap.

Our contribution. In this paper, we perform a fair comparison (common algorithms, technologies, tools) of three different hardware architectures, all capable of performing ECC. Using an openMSP430 at the core, we present (i) an area and speed-optimized software solution, (ii) a dedicated hardware module, and most importantly (iii) a novel ECC 'drop-in' architecture. For the drop-in architecture, a lightweight ECC accelerator is placed right between the CPU and its data memory. It requires less chip area than a dedicated hardware module, while being similarly fast. Compared to the optimized software solution, the energy consumption is reduced by a factor of 28, which certainly will make a major impact on the lifetime of a wireless sensor node. The drop-in concept is also most interesting for micro-chip manufacturers as only 4 kGE of dedicated chip area need to be committed for the drop-in accelerator.

The paper is structured as follows. Section 2 gives a short introduction on how to securely implement ECC and Section 3 discusses different architectures for ECC. The most promising architectures are then implemented within Sections 4–6 and compared within Section 7. Conclusions are drawn within Section 8.

2 A Short Introduction to ECC

Elliptic curves, used for cryptography, are built on top of finite fields. As finite field, one can either choose a prime field or a binary extension field. Prime fields are fast in software as they are based on integers and integer multipliers are available in nearly all (embedded) microprocessors. Binary-extension fields on the other hand are built on polynomials, which when implemented in hardware do not have the drawback of carry propagation. However, in software a multiplication of two polynomials has to be realized using branches, which are vulnerable to side-channel attacks.

The for us most interesting standardized elliptic curves [1, 2, 23] are all based on the Weierstrass equation: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. Depending on whether prime or binary-extension fields are used, this equation is simplified

Table 1. ECC formulas used within this paper.

Formula	Field	Registers	Finite-field operations per key bit		
			Add/Subtract	Square	Multiply
Hutter *et al.* [15]	$\mathbb{F}_p$	$7+3=10$	17	4	12
López and Dahab [22]	$\mathbb{F}_{2^m}$	$5+3=8$	3	5	6

to $y^2 = x^3 + ax + b$ or $y^2 + xy = x^3 + ax^2 + b$, respectively. Also the formulas used to perform point additions and doublings depend on the used finite field. For further information, the reader is referred to standard literature on elliptic curves [3, 13].

For the following comparison, it is important that all implementations are based on a common methodology. For the constant-runtime software implementations, the integer and polynomial arithmetic has been separated from the reduction operation. The reduction is performed using only simple shift and addition operations. Thereby advantage was taken of the used prime and irreducible polynomial. To perform an inversion in constant time, an exponentiation, based on Fermat's little theorem ($a^{q-2} \equiv a^{-1} \bmod (q)$) is used. For binary-extension fields an optimized inversion algorithm based on Itoh and Tsujii [17] is used.

More important than the used finite field is that ECC implementations are vulnerable to side-channel attacks [7]. Attackers can use runtime information, power consumption profiles, or induce faults to recover the secret key. This is a significant problem for the easily accessible wireless sensor nodes that usually are deployed within unsafe environments. Thus, a methodology must be utilized that minimizes the potential threats.

In this paper we take advantage of differential addition formulas optimized for Montgomery ladders. Table 1 gives a short summary of the used formulas. By using a Montgomery ladder, the underlying finite-field operations are independently performed from the used private scalar. Thus a key-independent constant runtime is achievable under the assumption that all finite-field operations are performed in constant time (which they are). The formulas are also lightweight. Only 7/5 registers are required during the point double-and-add operations. For the recovery of the y-coordinate another two registers are needed which store the original base point. Another register that stores the private scalar is also included in all comparisons within this paper.

To further increase the resistance against power-analysis attacks one would use Randomized Projective Coordinates [5] and to resist fault attacks, perform point verifications before and after each point multiplication. In practice the resistance against those attacks is verified by performing real-world evaluations. As those evaluations would go beyond the scope of this paper, they have not (yet) been done. However, the algorithms and methodologies used for our implementations are applicable to build real-world secure hardware.

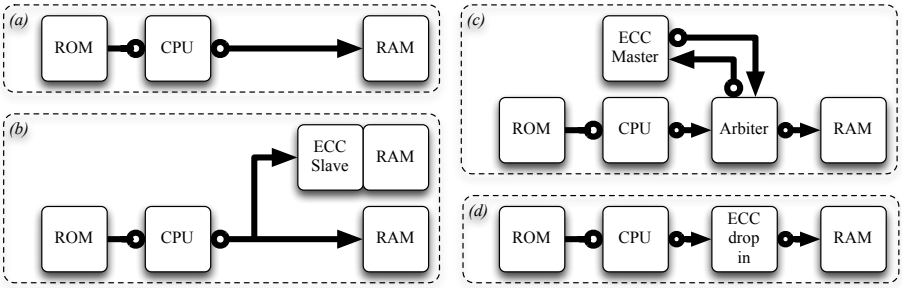

Fig. 1. Microprocessor-based architectures.

3 Architectures

The decision regarding the best architecture is most important for a final design as it greatly influences area, runtime, power, and energy characteristics. Only by considering all requirements and the system as a whole, a *global optimum* can be found. By optimizing a single (e.g., ECC) component it is probable to reach a *local optimum* only. Figure 1 shows four different architectures which are based on microprocessors, as microprocessors are the central component in all currently available sensor nodes. The ECC-independent components, such as the wireless interface and the actual sensor, are considered to be constants and therefore independent from the used architecture.

(a) The most straight-forward solution is to perform and optimize ECC in software[1]. The hardware designer only has to make sure that the data memory is sufficiently large and the assembly-optimized ECC code is placed within the program memory. The microprocessor (CPU) is then used to execute the code. In Figure 1, the program memory is simplified as ROM and the data memory as RAM. As ECC is very resource demanding and a software-only solution is in most cases insufficiently slow, one could add a memory-mapped ECC co-processor.

(b) Co-processors have already been extensively studied and optimized in related work [19, 20]. However, comparing area and power results of designs that use different technologies and tools is inaccurate. Therefore, Section 5 presents an ECC co-processor on-par with related work.

The drawback of so-called ECC slaves is that they waste chip area by having their own memory. A solution in which the global RAM is reused is preferable. Even when area-efficient RAM macros are used, practical evaluations show that one RAM macro with more entries is smaller than two RAM macros with fewer entries (c.f. 128×8-bit: 2,073 GE vs. 256×8-bit: 2,897 GE). An ECC accelerator without RAM, which only performs finite-field operations would be a solution. Unfortunately, for this solution the CPU has to manually move operands from the RAM to the ECC slave and vice versa, thus wasting potential performance.

[1] Section 4 discusses this solution.

Table 2. HW synthesis of openMSP430 [24].

Functional Blocks	**Chip-Area** [GE]
openMSP430	7,801
Execution unit	5,536
Register file	2,709
ALU	693
Multiplier	1,826
openMSP430 w/o Multiplier	5,958

Algorithm 1 Accessing the 16-bit memory-mapped multiplier.

```
1: MOV R4, &MPY
2: MOV R5, &OP2
3: NOP
4: MOV @RESLO, R6
5: MOV @RESHI, R7
```

(c) An ECC circuit which, like the CPU, is capable of accessing the global data memory by itself solves that problem: an ECC bus master. This assumes that the used microprocessor must support a multi-master scenario, which embedded light-weight microprocessors usually do not. Also, the required arbiter can have a significant impact on the total chip area.

(d) A more sophisticated concept is to unite the ECC master with the arbiter. This within the context of ECC novel concept "drops" an ECC accelerator right between the CPU and the data memory. From the viewpoint of the CPU it behaves as simple ECC slave and does not hinder any access to the data memory. From the viewpoint of the drop-in module, direct access to the data memory is possible. Advantageous is also that neither the CPU (compared to instruction-set extensions) nor the data-memory need to be modified. Section 6 discusses this solution in more detail.

Tools. For this paper we use the 130 nm low-leakage ASIC technology by UMC with the Faraday design libraries in combination with area-efficient single-port register-based RAM macros. For hardware synthesis Cadence RTL Compiler v08.10, for place-and-route and power simulation Cadence First Encounter v08.10, and for simulation Cadence NCSim v08.20 are used. In this technology, one gate equivalent is equal to 5.12 µm^2. All evaluations are performed at 1 MHz and can easily be synthesized to exceed an operating frequency of 50–100 MHz.

4 ECC on openMSP430

At the core of all previously discussed hardware designs is a microprocessor. The selection of an appropriate microprocessor crucially influences the final runtime, chip area, power, and energy results. The MSP430 [27] developed by Texas Instruments, is considered to be a role model when it comes to low-cost and low-power applications. It is currently already used for the sensor-node platforms BEAN, COOKIES, EPIC mode, PowWow, Shimmer, TelosB, T-Mote Sky, and XM1000, just to name a few. The MSP430 is a 16-bit RISC processor with a Von Neumann architecture. This is important for saving data memory, as constants do not have to be loaded to the expensive RAM before they are used. The

Algorithm 2 $ACC \leftarrow ACC + (A[0] \times B[2]) + (A[1] \times B[1]) + (A[2] \times B[0])$.

```
 1: ADD   #4   ,  OPB          9: MOV  @OPA   ,  &MAC
 2: MOV   @OPA+, &MPY         10: MOV  @OPB   ,  &OP2
 3: MOV   @OPB , &OP2         11: SUB  #4     ,   OPA
 4: DECD   OPB                12: ADD  @RESLO ,   ACC0
 5: MOV   @OPA+, &MAC         13: ADDC @RESHI ,   ACC1
 6: MOV   @OPB , &OP2         14: ADDC @SUMEXT,   ACC2
 7: DECD   OPB                15: MOV   ACC0  , 4(DEST)
 8: ADD @SUMEXT,  ACC2        16: CLR   ACC0
```

MSP430 comes with 16 16-bit registers, where R0 is the program counter, R1 is the stack pointer, R2 is the status register, and R3 is the constant-generator register. So only 12 registers (R4–R15) are useable as general-purpose registers. The MSP430 comes with only 27 instructions, from which none is a multiplication instruction. To perform a 16-bit integer multiplication, the MSP430 optionally has a memory-mapped multiplier. This will be discussed in detail later.

4.1 openMSP430

As our desired goal is a microprocessor-based hardware design, we need a hardware model of the MSP430. Olivier Girard programmed a synthesizable Verilog clone of the MSP430, called openMSP430 [24]. This clone fully supports the instruction set of the original MSP430 (with nearly identical timings), interrupts, and power-saving modes. It optionally comes with a 16×16-bit hardware multiplier, watchdog, timer, and GPIOs. A first evaluation of this core is depicted in Table 2. An openMSP430 without data or program memory (which will be chosen appropriately) requires 7,801 GE. Most of this chip area is spent on the execution unit (71 %), and the hardware multiplier (23 %). Without the multiplier, which is not necessary for binary-field based ECC, the openMSP430 only requires 5,958 GE.

4.2 Integer Arithmetic

In order to perform a 16-bit integer multiplication, four memory accesses are necessary. Algorithm 1 shows the assembly code necessary to multiply R4 with R5 and to store the product in R6 and R7. The code shown in Algorithm 1 needs $4 + 4 + 1 + 2 + 2 = 13$ cycles to complete.

As multiple words are needed to represent integers within the used finite field, the multi-precision product-scanning multiplication technique of Comba [4] is used. Algorithm 2 sketches the used methodology. Three registers are used to hold pointers to the operands (OPA and OPB) and the result (DEST), three registers for the accumulator (ACC0-2) and three registers to hold addresses of the memory-mapped multiplier (RESLO, RESHI, and SUMEXT). In order to avoid loading the product after each multiplication, multiply-accumulate operations

Algorithm 3 64 × 1-bit polynomial multiplication.

```
1: RLA B0          8: XOR #0, C0
2: JNC +10         9: XOR #0, C1
3: XOR A0, C0     10: XOR #0, C2
4: XOR A1, C1     11: XOR #0, C3
5: XOR A2, C2     12: NOP
6: XOR A3, C3     13: NOP
7: JMP +12
```

Table 3. Comparison with related work.

Curve	Type	ROM [Bytes]	RAM [Bytes]	Runtime [kCycles]
Gouvêa *et al.* [9] and Szczechowiak *et al.* [26]				
`secp160r1` [9]	$\mathbb{F}_p$	23,300	2,800	2,528
`sect163k1` [9]	$\mathbb{F}_{2^m}$	27,800	3,600	2,032
Custom [26]	$\mathbb{F}_p$	31,300	2,900	5,898
`sect163k1` [26]	$\mathbb{F}_{2^m}$	32,100	2,800	8,519
ours on MSP430				
`secp160r1`	$\mathbb{F}_p$	4,230	282	5,721
`sect163r2`	$\mathbb{F}_{2^m}$	4,126	294	7,447

are performed directly within the memory-mapped multiplier. The overflowing bit stored within the `SUMEXT` register needs to be loaded within line 8. After line 14, the accumulated product resides within the registers `ACC0-2`. This technique has already been presented by Gouvêa and López [8].

4.3 Polynomial Arithmetic

As the MSP430 lacks a carry-less multiplier, a polynomial multiplication has been implemented using branch operations. Algorithm 3 shows a 64×1-bit polynomial multiplication which was used to build a 64×32-bit multiplication. The 64×32-bit multiplication can be performed without the use of a single, costly memory load or store operation. Using the methodology of Karatsuba and Ofman a three-way split of a single 192-bit multiplication to 6 64-bit multiplications has been performed. On the MSP430 a 64×32-bit polynomial multiplication takes 383 cycles and a 192-bit polynomial multiplication takes 6,089 cycles. For constant runtime, lines 8-13 in Algorithm 3 perform dummy operations. Without the dummy operations, a speedup of 23 % is possible on average. For comparison, a 192-bit integer multiplication takes 2,254 cycles and therefore is 2.7 times faster. Gouvêa *et al.* [9] report an assembly optimized implementation for `sect163k1` which only needs 3,907 cycles, but their implementation is not safe from timing attacks.

4.4 Software Results

Four standardized elliptic curves providing security-levels of 80–96 bits have been implemented. `secp192r1` and `sect163r2` are chosen because they are the smallest elliptic curves within the NIST standard [2, 23], still providing a sufficient level of security. `secp160r1` has been chosen because it is popularly used within related work. As `c2tnb191v1` [1] provides a similar security level as `secp192r1` (95 vs 96 bits) it can be used for comparison.

Note that Table 3 shows the runtimes of our software implementation, simulated on a cycle-accurate model of the MSP430, while Table 4 shows the slightly

Table 4. Synthesized software implementations of ECC on the openMSP430.

Curve	Type	Security	ROM	RAM	ROM	RAM	Area	Runtime	Power	Energy
		[Bits]	[Bytes]	[Bytes]	[GE]	[GE]	[GE]	[kCycles]	[µW]	[µJ]
`secp160r1`	$\mathbb{F}_p$	80	4,230	282	5,907	3,175	16,638	5,445	55.9	304.3
`secp192r1`	$\mathbb{F}_p$	96	4,846	322	6,173	3,400	17,128	8,650	53.9	466.7
`sect163r2`	$\mathbb{F}_{2^m}$	81	4,126	294	5,737	3,275	14,167	7,217	49.1	354.3
`c2tnb191v1`	$\mathbb{F}_{2^m}$	95	3,994	310	5,735	3,375	14,014	8,376	55.4	463.8

better runtimes for an openMSP430. In Appendix A a detailed comparison of all software implementations is depicted.

In literature many speed-optimized ECC implementations for the MSP430 have been reported [9, 21, 26, 28] (cf. Table 3). Because of the extentsively performed assembler optimization, our software implementation outperforms the related work of Szczechowiak *et al.* [26] that also requires larger memories. The fastest (ECDSA) implementation was done by Gouvêa *et al.* [9] in 2012. Compared to our implementations, they report twofold faster runtimes at the expense of 7 times larger program and 12 times larger data memories. As we synthesize the program memory and choose appropriately large RAM macros, their implementation would result in a significantly larger hardware design, compared to ours.

Table 4 shows the measured chip area, runtime, power, and energy results for the four implemented elliptic curves. The biggest impact of up to 60 % on the total chip area is due to the size of the program memory and data memory. For the elliptic curves over $\mathbb{F}_{2^m}$ the integer multiplier has been removed. The binary-field-based ECC implementations are about 16 % smaller and similarly fast, compared to the prime-field-based ECC implementations. For `sect163r2` the used 176-bit polynomial multiplier which is based on the 192-bit multiplication algorithm discussed before, renders the runtime results inferior compared to `secp160r1`.

The elliptic curve requiring the least amount of energy is `secp160r1` (303.3 µJ). However, the biggest potential for hardware optimizations (cf. [29]) lies within binary-field based elliptic curves (354.3 µJ). Therefore `sect163r2` alias NIST B-163 has been selected for the following hardware implementations.

5 Stand-alone ECC Hardware

The dedicated hardware design used for this paper (cf. Figure 2) is strongly related to the works of Kumar and Paar [19] and Lee *et al.* [20], but uses a different memory architecture. As register-based memory is most expensive, it is replaced by latches, which are 27 % smaller in the used 130 nm technology. As latches are not synchronous, the depicted circuit only works because a common `Work` register is placed before the latches. At the positive clock level, activated via the clock gate (CG), the latch inherits the contents stored within the `Work` register. A single multiplexer is used to select the content of a latch which is

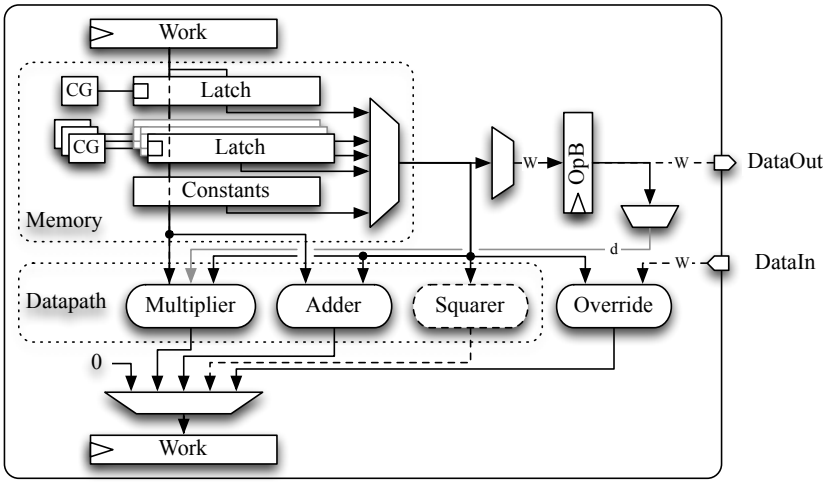

Fig. 2. Dedicated ECC hardware.

then used as operand `OpA` for the datapath. The datapath consists of an MSB-first digit-serial multiplier, an adder, and optionally a squaring unit. For the multiplication, an operand is split into W-bit sized parts which are stored in `OpB`. d of the W bits are then concurrently handled within the multiplication circuit. Dependent on the desired speed grade, it is possible to increase the size of d, or to use a dedicated squaring circuit. For interfacing the module with an external W-bit wide bus, the existing multiplexers are reused. For memory storing operations, W of the N-bit wide bus are overridden by the externally driven bus signal.

5.1 Stand-alone ECC Hardware Results

A complete ECC coprocessor including datapath, controlpath, memory, private scalar, modifiable base point, and resulting point with recovered y-coordinate needs at least 11,778 GE and up to 341,835 cycles. Table 5 summarizes our results for different d parameters. Adding a dedicated 1-cycle squaring unit only costs 884 GE (7.5 %) of additional hardware, but improves the runtime by a factor of approximately two. The most energy-efficient circuit is using $d = 2$. The circuit with the best scaled area-runtime product (SARP) is using $d = 4$.

Compared to related work [14, 19, 20, 25, 30], our designs are smaller or faster and therefore provide a better area-time product. In terms of power and energy, which are highly dependent on the used technology, our results are similar to related work.

The chip area shown in Table 5 does not include the area needed by the MSP430 (5,958 GE), its data memory (8×16-bit RAM – 1,443 GE), and its program memory (354 bytes – 801 GE). So all our dedicated ECC hardware designs

Table 5. Synthesis results of the dedicated ECC hardware design without MSP430.

Design	Technology [nm]	Area [GE]	Runtime [kCycles]	Power [µW]	Energy [µJ]	SARP
$d = 1$ w/o squ.	130	11,778	341,835	63.3	21.6	5.2
$d = 1$ w/ squ.	130	12,662	174,025	71.5	12.4	2.8
$d = 2$ w/ squ.	130	13,307	93,997	78.4	7.4	1.6
$d = 4$ w/ squ.	130	14,552	53,489	140.1	7.5	1.0
Kumar and Paar [19] $d = 1$	350	15,094	376,864	788.0	297.0	7.3
Hein *et al.* [14]	180	11,904	296,299	101.9	30.2	4.5
Lee *et al.* [20] $d = 1$	130	12,506	302,457	32.4	9.8	4.9
Lee *et al.* [20] $d = 5$	130	20,316	83,375	48.9	4.1	2.2

need additional 8,202 GE of hardware in order to provide the full functionality of an MSP430.

The major drawback of the ECC hardware module is the inefficient data memory. Unfortunately, there are no efficient RAM macros with a 163-bit interface. Even though latches are used, the memory requires 6,924 GE, or 59 % of the total hardware area. A comparable register-based RAM macro with $8 \times 163 = 1,467$ bits requires only 2,600 GE. That is 62 % less. For the drop-in concept discussed in the next section, such an area-efficient RAM macro is used.

6 Drop-in Concept

The drop-in concept has some similarities with instruction-set extensions. The drawback of ISE is that the HW designer needs to be able to modify both the controlpath and the datapath of the used processor, as well as the corresponding software toolchain. A different solution, based on a memory mapped carry-less multiply-accumulate unit has similarly large access times as the already existing integer multiply-accumulate unit of the MSP430. Therefore, it would only make a minor impact on the ECC runtime.

The drop-in concept provides full advantage even when the hardware designer is not able to modify the used microprocessor. Performance similar to dedicated ECC hardware is achievable and the verification and validation process regarding the used microprocessor does not have to be redone. The drop-in concept is also flexible: A hardware designer can shift control logic between the program memory and the dedicated hardware module. In this paper the drop-in module is designed to efficiently perform finite-field arithmetic (addition, squaring, and multiplication) only. The finite-field inverse as well as the point-multiplication algorithm are implemented in software.

As interface, the drop-in module provides three address, a command, and a status register. Before each operation, the address registers are written with two source and a destination memory address, and the operation is started by writing the command register. The status register is then polled to check whether the operation has been finished. Actually, experiments showed that waiting at the

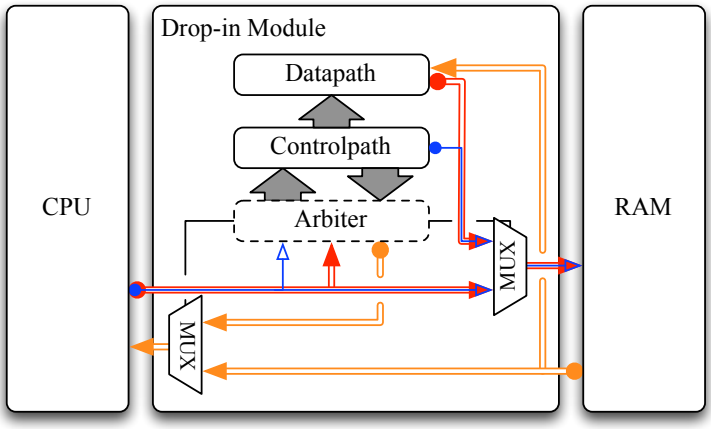

Fig. 3. Drop-in module for Elliptic Curve Cryptography.

beginning of the finite-field operations for the previous operations to finish is more performant. In this way the CPU and the drop-in module can partly work in parallel.

6.1 Drop-in Architecture

Figure 3 shows the architecture of the drop-in module. The data-bus is depicted in red and orange, the address bus in blue. The drop-in module consists of a lightweight arbiter, controlpath, and datapath. If both the CPU and the drop-in module want to access the data memory, the currently pending operation within the drop-in module is put on hold and the CPU is given access to the data memory. Therefore the drop-in module needs to be specially prepared for the case in which it is put on hold. For our ECC design, only 7 1-bit registers are necessary to provide this functionality. As a side note, the openMSP430 does not support to have delayed memory access.

The datapath within the drop-in module is very similar to the datapath of the dedicated ECC hardware module. Figure 4 shows that only two N-bit registers and a W-bit register are necessary for an MSB-first digit-serial multiplier. In each cycle, the N bits of `OpA` are multiplied with d bits of `OpB`, which are added to a d-bit shifted intermediate product, stored within the N-bit `Work` register. The $(N+d)$-bit sum is then reduced and used to update the `Work` register. At the beginning of the algorithm, `Work` is initialized with zero and `OpA` is initialized with the value stored within the data memory. The W-bit chunks of `OpB` are loaded on-demand, as it is shown within Figure 5 (d). When the multiplication is finished, the result within `Work` is stored back to the data memory.

Optionally, a dedicated squaring unit can be used. In our implementation (Figure 5 (a)), `OpA` is loaded from the data memory, the squaring is performed within a single cycle, and the result is stored back to the data memory. The

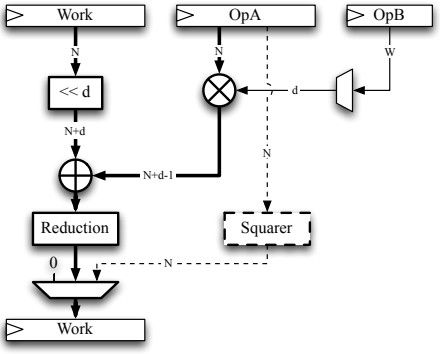

Fig. 4. Datapath of the ECC drop-in.

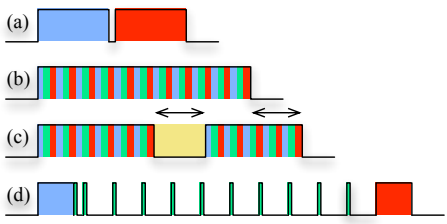

Fig. 5. Bus access during squaring (a), addition (b), addition with stall (c), and multiplication (d) operations. Memory operations regarding `OpA`, `OpB`, and `Dest` are colored in blue, green, and red, respectively.

datapath of the addition is not shown in Figure 4 as it only is a simple XOR-gate. For the finite-field addition (Figure 5 (b)) three times $\lceil N/W \rceil$ memory operations are necessary.

If at any moment, the CPU needs to do some (real-time) interrupt handling and needs access to the data memory, the operation in progress within the drop-in module is simply halted and continued when the data memory bus is free to use (Figure 5 (c)).

6.2 Drop-in Concept Hardware Results

Similar to before, the drop-in module was evaluated for different configurations (cf. Table 6). Independent of the size of the digit-serial multiplier and the availability of a squaring unit, the size of the CPU (5,715 GE), the program memory (1,426 bytes – 2,635 GE), and the data memory (222 bytes – 2,875 GE) are constant. The drop-in module only needs between 4,114 GE and 6,760 GE in chip area.

This is the most interesting number for microchip manufacturers. As not every customer actually needs ECC, they want to leave out unnecessary components, as they produce unnecessary costs. On the other hand, customers that require performant ECC can take advantage of the drop-in ECC module. Compared to a dedicated ECC hardware module, which requires 12–15 kGE, the drop-in module requires only a fraction of it: 35 %.

6.3 Related Work

In 2009, Guo and Schaumont [11] identified the data bus as potential bottleneck for ECC designs. Cause of that, they add the necessary data memory to the dedicated ECC accelerator to keep the number of necessary bus accesses at a

Table 6. Synthesis results of all ECC hardware architectures at 1 MHz for `sect163r2`.

Design	Module [GE]	Chiparea [GE]	Runtime [Cycles]	Power [µW]	Energy [µJ]
Architecture *(a)* – Software-only implementation					
openMSP430 w/o mult.	-	14,167	7,216,905	49.1	354.3
Architecture *(b)* – Dedicated ECC Hardware Accelerator					
$d = 1$ w/o squ.	11,778	19,980	342,724	93.8	32.1
$d = 1$ w/ squ.	12,662	20,864	174,910	112.9	19.7
$d = 2$ w/ squ.	13,307	21,509	94,882	152.4	14.5
$d = 4$ w/ squ.	14,552	22,754	54,376	181.7	9.9
Architecture *(d)* – Drop-in Module Based					
$d = 1$ w/o squ.	4,114	15,282	467,370	66.1	30.9
$d = 1$ w/ squ.	4,895	16,121	303,202	77.6	23.5
$d = 2$ w/ squ.	5,512	16,738	224,222	73.6	16.5
$d = 4$ w/ squ.	6,760	17,986	182,130	70.0	12.8

minimum. Thus their ECC accelerator becomes more like a dedicated hardware module. As it is an FPGA design, a comparison with our work is impracticable.

Most comparable to our drop-in concept is the work of Koschuch *et al.* [18]. They implemented a memory-less ECC accelerator and used a DMA controller for efficiently accessing the data memory. Their architecture is best comparable with the previously discussed architecture *(c)*. Their DMA controller is 1,029 GE large, their ECC accelerator is 11,618 GE large, and their total design for $\mathbb{F}_{2^{191}}$ requires 29,491 GE. For a scalar multiplication, they require 1,416 kCycles. Thus their design is slower and larger than our drop-in designs.

7 Comparison of Implemented Architectures

In the previous sections, architectures *(a)* - a plain software implementation, *(b)* - a dedicated ECC hardware module, and *(d)* - a drop-in module - have been presented and discussed in connection with the appropriate related work. Thereby all implementations are on-par with related work or outperform related work. Most important however is the comparison of the three implemented architectures *(a,b,d)* with each other.

Table 6 shows the area, runtime, power, and energy values of all architectures. The column 'Module' gives the area for the dedicated ECC hardware blocks, while 'Chiparea' accumulates the program memory, the data memory, the microprocessor, and the special hardware module. The runtimes of architecture *(b)* now include the calling overhead needed to trigger and poll the dedicated hardware module. In comparison to Table 5, the area and power values now also include the RAM, ROM, and CPU.

The smallest of all implementations is the plain software implementation *(a)* needing only 14,167 GE. Both the drop-in solution *(d)* (15,282 GE) and the dedicated hardware solution *(b)* (19,980 GE) are larger. However, those solutions

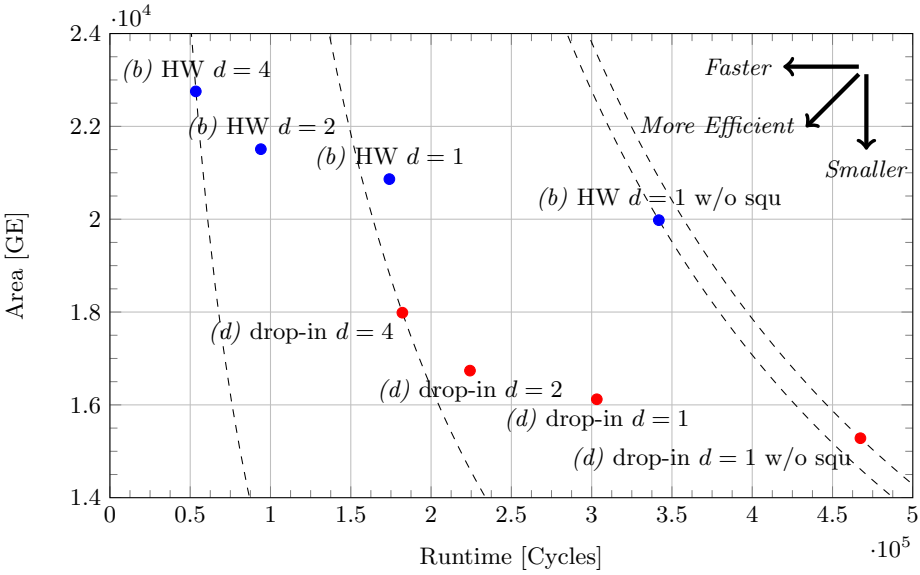

Fig. 6. Area-runtime-characteristics of the various ECC architectures.

are up to 132 times faster and up to 36 times more energy efficient. Thus architecture *(a)* can be considered as fall-back solution, but is practically to slow for most relevant applications. The runtime is nearly one second at a common sensors-node frequency of 8 Mhz.

Thus the question is whether architecture *(b)* or *(d)* is better. The drop-in concept *(d)* is 22 % smaller and requires 50 % less power. On the other hand, architecture *(b)* is faster. The comparison is visualized in Figure 6, which prints the chiparea values versus the runtimes. The dashed lines indicate constant area-runtime products. After investigating the results in detail, our conclusion is that both architectures *(b)* and *(d)* have the very right of existence. However, if the application requires that a point multiplication is finished within, e.g., 30 ms (@ 8 MHz), architecture *(d)* based on the drop-in concept with $d = 2$ is the smallest and therefore best solution.

8 Conclusion

This work proofs that the drop-in concept is a viable alternative to previously existing plain software and dedicated hardware solutions. Both the presented software-only and the presented dedicated hardware solution enable a fair comparison using a common side-channel aware methodology and identical tools. The software implementation is (supposed to be) side-channel secure and needs 7–12 times less memory compared to latest related work. The hardware implementation is more area-efficient compared to related work, because a specially

designed data memory is used. However, a plain hardware implementation is not aware of the versatile MSP430, which is usually available in wireless sensor nodes. Hereby the drop-in concept provides a novel solution which actually is smaller than the hardware module based architecture, while being similarly fast, and requiring 36 times less energy than the dedicated software solution. This makes the newly presented drop-in concept a great solution for microchip and sensor-node manufacturers.

Acknowledgments

The research described in this paper has been supported, in part, by the European Commission through the ICT Program under contract ICT-SEC-2009-5-258754 TAMPRES.

References

1. American National Standards Institute (ANSI). AMERICAN NATIONAL STANDARD X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.
2. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, September 2000.
3. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, Boca Raton, FL, 2006.
4. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, pages 526–538, 1990.
5. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
6. H. Eberle, A. Wander, N. Gura, S. Chang-Shantz, and V. Gupta. Architectural Extensions for Elliptic Curve Cryptography over GF(2^m) on 8-bit Microprocessors. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 343–349. IEEE Computer Society, 2005.
7. J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications*, LNCS, pages 265–282. Springer, 2012.
8. C. P. L. Gouvêa and J. López. Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller. In *INDOCRYPT*, LNCS, pages 248–262, 2009.
9. C. P. L. Gouvêa, L. Oliveira, and J. López. Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller. *Journal of Cryptographic Engineering*, 2:19–29, 2012.
10. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES*, pages 133–147, 2004.
11. X. Guo and P. Schaumont. Optimizing the HW/SW boundary of an ECC SoC design using control hierarchy and distributed storage. In *DATE*, pages 454–459, 2009.

12. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In *CHES*, pages 119–132, 2004.
13. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
14. D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *SAC*, LNCS, pages 401–413, 2008.
15. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In *AFRICACRYPT*, pages 170–187, 2011.
16. IEEE. IEEE Standard 802.15.4-2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), May 2003.
17. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, pages 334–335, 1988.
18. M. Koschuch, J. Großschädl, D. Page, P. Grabher, M. Hudler, and M. Krüger. Hardware/Software Co-Design of Public-Key Cryptography for SSL Protocol Execution in Embedded Systems. In *Workshop on Embedded Systems Security*, pages 63–79, 2009.
19. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security – RFIDSec 2006*, 2006.
20. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
21. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks*, pages 245–256, 2008.
22. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 316–327. Springer, 1999.
23. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009.
24. Olivier Girard. openMSP430, 2013. Available online at `http://opencores.org/project,openmsp430`.
25. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In M. Joye and J.-J. Quisquater, editors, *CHES 2004, Proceedings*, volume 3156 of *LNCS*, pages 92–106. Springer, August 2004.
26. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks 5th European Conference*, LNCS, pages 305–320, 2008.
27. Texas Instruments. MSP430C11x1 - Mixed Signal Microcontroller. Available online at `http://focus.ti.com`, 2008.
28. H. Wang, B. Sheng, and Q. Li. Elliptic Curve Cryptography-based Access Control in Sensor Networks. *International Journal of Security and Networks*, pages 127–137, 2006.
29. E. Wenger and M. Hutter. Exploring the design space of prime field vs. binary field ecc-hardware implementations. In *Information Security Technology for Applications*, LNCS, pages 256–271. Springer, 2012.
30. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto*, pages 78–91, 2005.
31. ZigBee Alliance. The ZigBee Alliance Website. `http://www.zigbee.org/`.

A Implementation Runtimes

Table 7 lists the constant key-independent runtimes of all implementations done for this paper. Architectures *(b)* and *(d)* implemented the elliptic curve `sect163r2`.

We distinguish between runtimes for the original MSP430 and the openMSP430. The runtimes of the openMSP430 are better, because several instructions of the openMSP430 perform the same operation in less cycles than the original MSP430. In average, the openMSP430 is 5 % faster for the prime field based elliptic curves (`secp160r1`, `secp192r1`) and 3 % faster for the binary field based elliptic curves (`sect163r2`, `c2tnb191v1`).

Table 7. Runtimes for finite-field addition/subtraction (ADD), squaring (SQU), multiplication (MUL), inversion (INV), and point-multiplication (P-MUL) operations.

Implementation	ADD [Cycles]	SQU [Cycles]	MUL [Cycles]	INV [Cycles]	P-MUL [Cycles]
(a) MSP430 `secp160r1`	163	1,905	1,905	327,366	5,721,420
(a) MSP430 `secp192r1`	191	2,559	2,559	526568	9,100,128
(a) MSP430 `sect163r2`	109	852	6,604	199,815	7,446,677
(a) MSP430 `c2tnb191v1`	118	778	6,566	229,297	8,610,906
(a) openMSP430 `secp160r1`	161	1,808	1,808	310,812	5,445,010
(a) openMSP430 `secp192r1`	189	2,426	2,426	499,331	8,650,455
(a) openMSP430 `sect163r2`	107	781	6,446	186,653	7,216,905
(a) openMSP430 `c2tnb191v1`	116	725	6,420	217,209	8,376,138
(b) HW $d = 1$ w/o squ	2	174	174	29,754	341,835
(b) HW $d = 1$	2	1	174	1,728	174,025
(b) HW $d = 2$	2	1	93	999	93,997
(b) HW $d = 4$	2	1	52	630	53,489
(d) drop-in $d = 1$ w/o squ	40	208	208	36,419	467,370
(d) drop-in $d = 1$	40	38	208	9,963	303,202
(d) drop-in $d = 2$	40	38	128	9,227	224,222
(d) drop-in $d = 4$	40	38	80	8,843	182,130

Chapter 7

A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography

Publication Data

Erich Wenger. A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography. In Gildas Avoine and Orhun Kara, editors, *Second International Workshop on Lightweight Cryptography for Security and Privacy - LightSec 2013*, volume 8162 of *Lecture Notes in Computer Science*, pages 1–15, Springer, 2013.

Contributions

Main author.

A Lightweight ATmega-based Application-Specific Instruction-Set Processor for Elliptic Curve Cryptography

Erich Wenger

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
erich.wenger@iaik.tugraz.at

Abstract. It is inevitable that future Radio-Frequency Identification (RFID) technology must support complex protocols and public-key cryptography. In this paper, we present an Application-Specific Instruction-Set Processor (ASIP) based on a clone of the ATmega128 microprocessor. A leakage-resilient, constant-runtime, and assembly-optimized software implementation of an elliptic curve point multiplication, which outperforms related work, requires 9,230–34,928 kCycles or 681–2,576 ms for standard conform elliptic curves (`secp160r1`, `secp192r1`, `secp224r1`, and `secp256r1`). Because this is too slow for most applications, the microprocessor has been equipped with a multiply-accumulate and a bit-serial instruction-set extension. Therefore, the runtime has been reduced to practically usable 96–248 ms, while keeping the power below 1.1 mW, and the area consumption between 19–27 kGE.

Keywords: ATmega, Elliptic Curve Cryptography, Instruction Set Extension, Application Specific Instruction-set Processor, Constant Runtime.

1 Introduction

The future Internet of things will consist of embedded smart cards, wireless sensor networks, and Radio-Frequency Identification (RFID) tags. Those devices must be capable to communicate with other entities over an air interface and must provide *privacy* and *security* capabilities. At Asiacrypt 2007, Serge Vaudenay [20] showed that "...an RFID scheme that achieves narrow-strong privacy ... essentially needs public-key cryptography techniques."

Among the three most popular public-key cryptographic systems (RSA, ElGamal, and ECC), Elliptic Curve Cryptography (ECC) is the least resource demanding and most suitable for embedded systems. In the past ECC has been well studied and standardized by SECG [2] and NIST [17]. One could also investigate non-standardized curves (e.g., by Gallant, Lambert, and Vanstone [7] or Bernstein *et al.* [1]), but for open-loop systems one should stick to the given standards. In this work we focus on the four prime-field based Weierstrass curves

G. Avoine and O. Kara (Eds.): LightSec 2013, LNCS 8162, pp. 1–15, 2013.
The original publication is available at http://link.springer.com/chapter/10.1007/978-3-642-40392-7_1

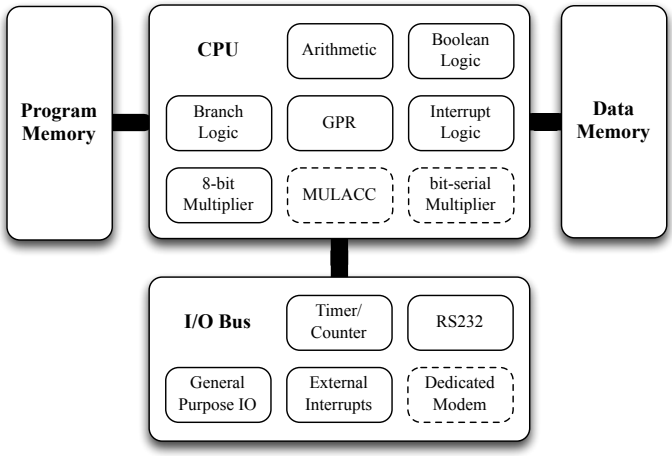

Fig. 1. Schematic diagram of the used processing architecture.

`secp160-256r1` as those are already used for mainstream applications such as TLS, IPSec, and SSH.

While public-key systems are very resource demanding, RFID tags must consume little power, be cheap (have a small chip area) and support real-time applications (respond within a given time). The traditional approach, which will pretty soon exceed its realms of possibility, is to equip the state machine of an RFID tag with a dedicated hardware block doing public-key cryptography. A more sophisticated solution is to base the design on a microprocessor, in particular on an Application-Specific Instruction-Set Processor (ASIP). An ASIP unites the advantages of programmable microprocessors (flexibility, extendability) with the advantages of dedicated hardware blocks (high-performance, low-power). Therefore, dedicated hardware units can be avoided and the overall hardware footprint decreases. In this paper, we transform a commercially available microprocessor into an ASIP targeting RFID and public-key cryptography.

For this paper, we base our design on the popular 8-bit Atmel ATmega128 AVR processor. This processor comes with an extensive instruction set and a dedicated hardware multiplier (important for prime-field arithmetic). The ATmega128 is used for a magnitude of applications and is supported by many toolchains (e.g., `avr-gcc`, IAR, Crossworks). In Wenger *et al.* [21], we presented 'Just Another AVR' (JAAVR, see Figure 1), a feature-complete clone of the popular ATmega128 which is written in VHDL and only requires 6,140 GE, making it perfectly suitable for area-sensitive embedded designs. In [21], we equipped an earlier version of JAAVR with a dedicated RFID modem, and evaluated ECC, AES, and Grøstl. As expected, those results show that ECC is the dominating component.

Our Contribution. In this paper we present an JAAVR-based ASIP optimized for ECC. First, we present new assembly optimized ATmega-compatible runtime results in which we outperform related software implementations (including our own in [21]). Second, we optimize the cycles-per-instruction (CPI) of all load, store, and multiply instructions of JAAVR in order to achieve speedups of 25–27 %. Third, we are the first to actually build an ATmega128-compatible processor with multiply-accumulate instruction-set extensions as ASIC. Previous work was either simulated or only performed on FPGAs. Fourth, we are also the first to build a tightly-coupled bit-serial multiplier as instruction-set extension of JAAVR for prime-field based ECC. Utilizing all those techniques, we present a 19 kGE small design suitable for RFID and other real-time applications.

This paper is structured as follows: Section 2 elaborates some basic design decisions. Section 3 discusses efficient software implementation techniques for ECC. Sections 4-6 deal with the improvement of the CPI of JAAVR, the utilization of a multiply-accumulate instruction, and the integration of a bit-serial multiplier, respectively. Section 7 discusses the results in connection with related work. Section 8 concludes the work. The most important results are gathered in Table 2. They are discussed throughout the paper.

2 Basic Reasoning

For RFID applications, the runtime of an algorithm is important in two respects. First, it must be sufficiently fast to support real-time applications. Second, by having a fast implementation, one can reduce the clock frequency and therefore reduce the power consumption. For a passively powered RFID tag, the power consumption is of upmost importance. For a typical ISO-14443-compatible [13] tag, we assume the following requirements. The clock should be an integer fraction of 13.56 MHz, the maximum power consumption below 2 mW, and the maximum runtime for an ECC point multiplication is 100-500 ms. It should be noted that all our hardware designs easily exceed this minimum clock frequency of 13.56 MHz.

Tools. For all of our implementations, we performed hardware synthesis (Cadence RTL Compiler v08.10), power simulations (Cadence First Encounter v08.10), and cycle-accurate post-synthesis and post-layout hardware simulations (using NCSim v08.20). As process technology the UMC 130 nm low-leakage CMOS technology with Faraday design libraries in combination with area-efficient single-port register-based RAM macros and Via-1 ROM macros is used. Previous experiments showed that synthesizing the program memory as standard logic cells results in smaller (post synthesis) but less routable designs (post place-and-route). A decreased cell density increases the size of the synthesized program memory to a point where the available Via-1 ROM macro is effectively smaller.

Practical Security. When implementing cryptography, the designer must consider practical attack scenarios such as timing, side-channel and fault attacks. Regarding timing attacks, all assembly-optimized implementations perform the point multiplication in constant runtime. Further, all implementations provide

Table 1. `secp160r1` point multiplication results using different multi-precision integer multiplication methods: operand-scanning (OS), product-scanning (PS), hybrid, and operand-caching (OC).

Impl.	Point-Multiplication [kCycles]	Integer-Multiplication [Cycles]	Program-Memory Size [Bytes]	Integer Mul. [%]	Chip Area [GE]
OS in C	37,168	9,807	4,188	3	17,738
OS	17,607	5,505	12,110	62	23,540
PS looped	17,226	5,367	4,636	4	17,638
PS	13,546	4,035	9,860	54	21,701[a]
Hybrid	10,609	2,972	9,050	49	21,701[a]
OC	9,230	2,473	8,218	46	21,701[a]

[a] Identical, because only certain discrete ROM macros are available.

a basic resistance against power-analysis attacks. The Montgomery ladder formula by Hutter *et al.* [11] performs key-independent double-and-add operations. With its requirement of 16 field multiplications and 17 field additions per key bit, it is reasonably fast. The finite-field multiplication is used for multiplications as well as for squarings. At the end of the Montgomery ladder, a y-coordinate-recovery and a constant-runtime inversion based on exponentiation (Fermat's little theorem) are performed. For side channel and fault security we also perform projective point randomization [4] before (against side-channel attacks) and point verification before and after (against fault attacks) the point multiplication. Because we did not perform practical power analysis attacks or fault simulations, we do not claim to be side channel or fault secure, but we use algorithms that improve resistance against those attacks. Thus all our results are practically relevant.

3 The Baseline: Efficient Software Implementation

By choosing an 8-bit processor we start with a rather small but "arithmetically speaking" slow processor. Our first not constant-time, plain-C implementation showed excruciatingly-slow runtimes of 37–131 million cycles, 2.7–9.6 seconds (@ 13.56 MHz). Thus optimizing the existing code in assembly is mandatory. For all following comparisons we consider our C implementations as baseline. In hardware it requires 16.9–19.5 kGE and 561–656 μW.

The first (and most laborious) optimization we have performed is the replacement of all field operations with constant-runtime assembly functions. This not only improves the runtime but also makes all timing attacks infeasible. The field addition and subtraction operations have been unrolled and perform the reduction without branches. For the field multiplications, we have taken advantage of the standardized Mersenne-like primes to get branch-free code using only addition and shift operations.

The most time-consuming algorithm is the multi-precision integer multiplication. Hutter and Wenger [12] did a thorough comparison between the Schoolbook's operand-scanning (OS), Comba's [3] product-scanning (PS), Gura *et al.*'s [9] hybrid and their own operand-caching (OC) multiplication methods. We implemented unrolled and looped versions of those algorithms in assembly. Table 1 shows that by doing so the runtimes of integer and point multiplications for `secp160r1` were improved by factors of 3.97 and 4.03, respectively. Our fastest implementation, based on the operand-caching method, achieved a runtime of 9,230 kCycles for a point multiplication. For comparison: Gura *et al.* [9], Szczechowiak *et al.* [19], Wenger *et al.* [21], and Liu *et al.* [16] achieved runtimes of 6,480 kCycles, 9,376 kCycles, 13,027 kCycles and 16,939 kCycles, respectively. However, most of those implementations do not consider side-channel attacks. For instance, Gura *et al.* used a Jacobian-based NAF point-multiplication formula. For reference, we applied the same technique as Gura *et al.* and improved their fastest implementation by 50 kCycles to 6,430 kCycles.

Apart from the expected runtime differences (OS > PS > Hybrid > OC), unrolling the integer multiplication has a huge impact on the size of the program code. Up to 62 % of the entries in the program memory are due to the unrolled integer multiplication. Compared to the C implementation, the chip size of the program memory increased by up to 76 %. Despite of that, assembly optimization and 'unrolling' improved the area-time-product by a factor of up to 3.3, thus establishing themselves as one of the most important optimization techniques.

The focus of this section was to perform software optimizations both applicable to the ATmega128 and JAAVR. In the next sections, we present hardware optimization techniques that improve both the execution time as well as the total hardware footprint.

4 Improving the CPU

Already during the design of JAAVR, we realized several avenues for optimization potentials. It was necessary to artificially introduce `NOP` operations in order to achieve identical cycles-per-instruction (CPI) counts compared to the original ATmega128. The most significant difference is that the ATmega128 uses a two-stage pipeline and JAAVR does not. So all we needed to improve the performance of *store* and *multiply* operations was to deactivate the `NOP` operations.

Unlike our previous paper [21], we also optimized memory *load* operations. For the cycle-accurate (CA) design, two cycles are needed to load data from the synchronous data memory. During the first cycle the address is applied to memory and during the second cycle the obtained data word is stored to a general purpose registers (GPR).

In order to reduce the latency of all *load* operations, we decided to introduce a pipelining structure. While the first cycle of the operation stays identical, the second cycle is performed as part of the subsequent operation. As Figure 2 shows, multiplexers before and after each general purpose register were added. MUX2 is used to update the next value stored within the GPR. MUX1 overrides the

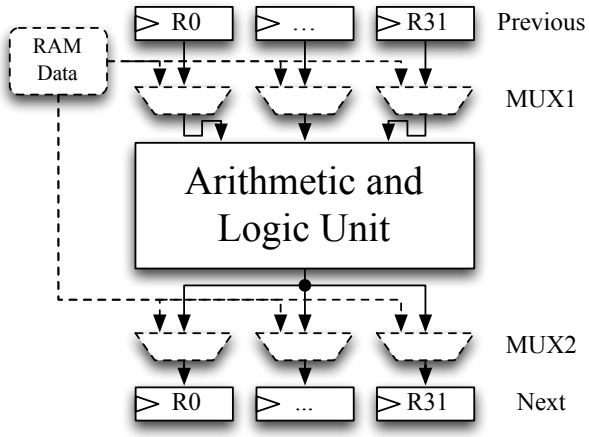

Fig. 2. The multiplexers were added to reduce the necessary cycles per load instruction.

current contents within the GPR. Thus the ALU is working on an updated set of GPR. The impact of the multiplexers on the critical path is hardly noticeable.

By switching JAAVR from the CA to the FAST mode, the following instructions improve: `MUL*`, `ST`, `STD`, `PUSH`, `LD`, `LDD`, `POP`, `IJMP`, `RJMP`, `CBI`, `SBI` ($2 \rightarrow 1$), `RCALL`, `ICALL`, `LPM`, `ELPM` ($3 \rightarrow 2$), `CALL`, `RET`, and `RETI` ($4 \rightarrow 3$). This increased the size of JAAVR form 6,140 GE to 6,791 GE (by 10 %), while the runtime of the fastest point multiplication improved by 26 %. Thus, the area-runtime product improved by a factor of 1.31.

After enabling those optimizations, JAAVR is still instruction-set compatible with the original ATmega128. So any (cryptographic) algorithm would benefit from the improved instruction-timing. The next two sections are dedicated on optimizing JAAVR for ECC using instruction-set extensions, transforming our design into an ASIP.

5 The Power of the MULACC Command

When investigating the instructions used for the unrolled product-scanning multi-precision multiplication, one can observe that there are four instructions, always used in consecutive order: `MUL`, `ADD`, `ADC`, and `ADC`. The idea behind the multiply-accumulate instruction-set extension is to combine those instructions into a single `MULACC` command, as it has been done in related work.

Already in 2004, Großschädl and Savaş [8] used five custom instructions to accelerate prime fields and binary extension fields on a MIPS32 core and gained a speedup of about six for binary extension fields. In 2005, Eberle *et al.* [5] presented multiply-accumulate instruction-set extensions for binary-extension fields on an ATmega128. They improved `sect223r1` by a factor of 13.6, but did not use ISE for prime fields as we do it in this paper.

In fact, we used the `MULACC` instruction to improve the fastest multi-precision multiplication formula: the operand-caching method. We are the first to combine the operand caching method with an instruction-set extension. Like the product-scanning method, this method uses the same sequence of instructions as mentioned above. So, by combining the operand-caching multiplication, which reduces the number of load and store operations, and the multiply-accumulate instruction, which reduces the number of additions, we achieved a new speed record: 631 cycles for a 160-bit integer multiplication.

There are two main challenges concerning the introduction of new instructions: First, most of the 2^{16} possibilities of the 16-bit instruction words are already assigned to existing instructions. Thus, the introduction of a new instruction would mean to modify existing instructions and being no longer compatible with the original ATmega128. Second, adapting the source code of `avr-gcc`, `avr-as`, and `avr-ld` to add new instructions does not seem to be straightforward.

Our solution is to introduce a new, within the I/O memory mapped, register that can switch the processor to a special operating mode. In this special operating mode, certain existing instructions are reinterpreted. For this solution, none of the `avr-gcc` tools had to be modified.

In order to improve the performance of the operand-caching multiplication, we introduced two instructions: `MULACC` and `ST_SHIFTACC`. `MULACC` multiplies two registers Rd and Rr and adds the result to the accumulator stored in R0-R2: $(R2, R1, R0) \leftarrow (R2, R1, R0) + Rd \times Rr$. This operation can be performed 2^8 times without the risk of an overflowing accumulator. This is more than the required $e = 10$ accumulations performed within the operand-caching multiplication algorithm (e is a parameter to adjust the operand caching method, see [12]). After e `MULACC` operations, `ST_SHIFTACC` is used to store the lowest byte of the accumulator (`ST R0,Z+`) and shifts the accumulator by 8 bits to the right ($R0 \leftarrow R1$, $R1 \leftarrow R2$, $R2 \leftarrow 0$). Because of those optimizations, we freed up two registers that were used as temporary storage of the product. In order not to waste them, we increased e from 10 to 11, which further decreased the number of necessary load and store instructions.

By using those instruction-set extensions, a 160×160-bit multiplication can be performed three times faster. It takes 631 cycles compared to 1,896 cycles. A detailed decomposition of the used instructions can be found in Appendix A. Further, the ISE had hardly any impact on the size of JAAVR. Only 257 GE or 3.8 % of additional logic had to be added. At the same time, the size of the program memory decreased: from 11,807 GE to 8,202 GE (-31 %). Adding all those improvements together, the area-time product improved by a factor of 2.4.

A point multiplication in `secp160r1` takes 3,268 kCycles. A profiling analysis showed that 83.2 % of the total runtime are spent on the field multiplications. The optimized reduction algorithm for the `secp160r1` prime $2^{160} - 2^{31} - 1$ utilizes 26.1 % of the total runtime or a third of the field multiplication. Thus, any further optimizations must not only consider the integer multiplication, but the field multiplication as a whole. This is done in the following section.

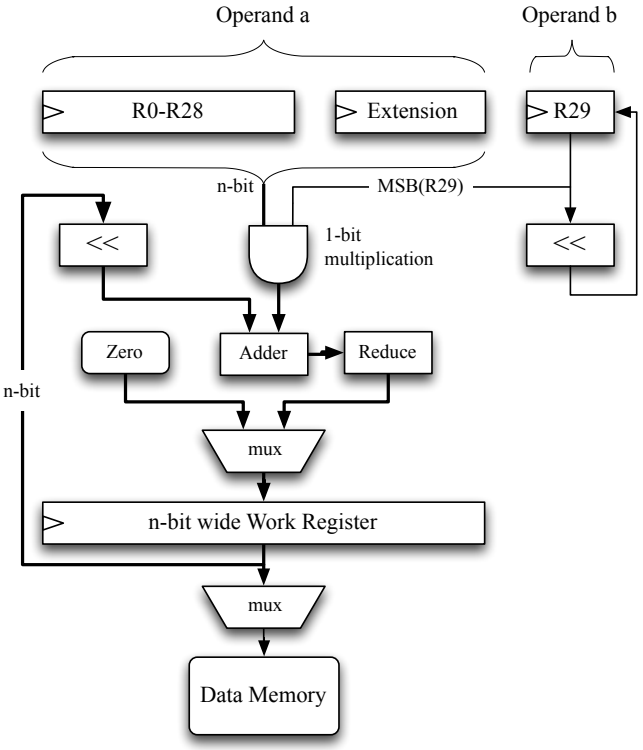

Fig. 3. The bit-serial multiplier is merged with the CPU.

6 Using a Dedicated digit-serial Multiplier

When investigating related work on ECC, one can either find ECC designs based on an word-level multiplier (cf. [10, 22]), as we used in the previous sections, or designs based on a digit-serial multiplier (cf. [6]). A digit-serial multiplier simultaneously operates on all digits of the multiplicand a, but only a single digit b_i of the multiplicand b.

In this section we want to introduce the concept of a 'tightly-coupled' bit-serial multiplier, which merges a bit-serial multiplier with the CPU. By reusing existing registers, we were able to keep the impact on the total chip area to a minimum and avoid unnecessary data transfers.

A block diagram of our bit-serial multiplier is depicted in Figure 3. Algorithm 1 shows the pseudo-code to control it. An MSB-first multiplier is used which accesses all b_i starting with the most significant bit. The Z-register (`R30`, `R31`) is used to address the memory, and `R29` is used to store the byte containing b_i. During each cycle, `R29` is shifted to the left using the `LSL` (Logic Shift Left) instruction of JAAVR. At the same time the `Work` register is updated in the

Algorithm 1 Pseudocode for the bit-serial multiplication.

```
1: PUSH all call-saved registers.
2: LD operand a to R0-R28 and Extension.
3: Switch to ISE mode.                          (memory mapped config register)
4: Clear Work register.
5: for i from ⌈n/8⌉ − 1 to 0 do
6:     LD R29, -Z                               (load b_i, pre-decrement pointer register Z)
7:     8 times: LSL R29                         (triggers bit-serial multiplier)
8: end for
9: Store Work register.
10: Switch back to normal mode.
11: POP all call-saved registers.
```

following manner: `Work` $\leftarrow (a \times b_i + \texttt{Work} \ll |b_i|) \pmod p$. In each cycle the most significant bit of `R29` (b_i) is multiplied with a, the product is added to a shifted version of the n-bit[1] `Work` register, and the `Work` register is updated with the reduced sum. After the last computation cycle, the product $a \times b \pmod p$ is stored in the `Work` register. A modified store instruction (`ST`) is used to write the `Work` register to memory, one byte at a time.

To reuse existing registers, a is stored in register `R0` to `R28` and the `Extension` register. For `secp256r1`, three IO-memory-mapped 8-bit `Extension` registers are necessary. So for `secp160-224r1` it was only necessary to add the `Work` register and the combinatoric logic.

A field multiplication for `secp160r1` takes 271 cycles. $9 \times 20 = 180$ cycles are used by the digit-serial multiplication, $2 \times 20 = 40$ cycles are necessary for loading operand a and storing the result, and 51 cycles are necessary to comply with the C-calling convention (`PUSH`, `POP`, `CALL`, `RET`) and to switch between the normal ATmega128 compatible operation mode and the instruction-set-extension mode.

Compared to the original software implementations in C, a speedup between 30 (`secp160r1`) and 44 (`secp256r1`) was achieved. The bit-serial approach is also 2.3–3.7 times faster than the `MULACC` instruction-set extension. The size of the program memory decreased significantly by 25 %–41 %. However, the size of the CPU core increased by 61 %–107 %. 4,551 GE are required for the bit-serial multiplier for `secp160r1` and 7,792 GE have to be added for `secp256r1`. The question now is whether adding the bit-serial multiplier improves or worsens the area-runtime product. In fact, it improves by a factor of 2.1–3.1, which makes the tightly-coupled digit-serial multiplier (by far) the fastest, even though not the smallest hardware implementation presented in this paper.

7 Results

The most important results of our implementations are summarized in Table 2 and have already been discussed in the previous sections. It contains figures that

[1] n relates to the number of bits needed to represent any value in $\mathbb{F}_p$.

Table 2. Summary of all experiments. SARP stands for 'scaled area-runtime product'.

Impl.	Runtime	Program Memory	Data Memory	Area Requirement JAAVR	ROM	RAM	Total	Power @13.56 MHz	Energy	Runtime @13.56 MHz	SARP
	[kCycles]	[Bytes]	[Bytes]	[kGE]	[kGE]	[kGE]	[kGE]	[μW]	[μJ]	[ms]	
secp160r1											
CA in C	37,168	4,188	418	6,140	7,744	3,855	17,738	561	1,539	2,741	22.5
CA	9,230	8,218	402	6,140	11,807	3,754	21,701	662	450	681	6.8
FAST	6,764	8,218	402	6,791	11,807	3,754	22,352	824	411	499	5.2
MULACC	3,268	5,688	402	7,048	8,202	3,754	19,004	850	205	241	2.1
bit-serial	1,298	4,286	350	11,341	7,744	3,452	22,537	1,013	97	96	1.0
secp192r1, NIST P-192											
CA in C	55,365	3,916	483	6,140	6,505	4,233	16,877	640	2,615	4,083	21.6
CA	15,093	10,070	462	6,140	11,807	4,107	22,054	677	753	1,113	7.7
FAST	11,101	10,070	462	6,791	11,807	4,107	22,705	832	681	819	5.8
MULACC	5,022	6,396	462	7,048	10,040	4,107	21,195	864	320	370	2.5
bit-serial	1,813	4,490	406	12,302	7,744	3,779	23,825	1,084	145	134	1.0
secp224r1, NIST P-224											
CA in C	86,058	3,926	569	6,140	6,363	4,712	17,215	663	4,208	6,346	23.9
CA	23,213	12,374	526	6,140	15,484	4,485	26,109	689	1,179	1,712	9.8
FAST	17,114	12,374	526	6,791	15,484	4,485	26,760	843	1,063	1,262	7.4
MULACC	7,537	7,404	526	7,048	10,040	4,485	21,573	848	472	556	2.6
bit-serial	2,469	4,808	466	13,237	7,744	4,132	25,113	1,032	188	182	1.0
secp256r1, NIST P-256											
CA in C	130,695	5,604	645	6,140	8,202	5,165	19,506	656	6,320	9,638	27.8
CA	34,928	15,888	590	6,140	17,029	4,838	28,006	663	1,707	2,576	10.7
FAST	26,290	15,888	590	6,791	17,029	4,838	28,657	811	1,572	1,939	8.2
MULACC	11,900	9,372	590	7,048	11,807	4,838	23,693	836	733	878	3.1
bit-serial	3,367	5,532	522	14,583	8,202	4,460	27,244	1,031	256	248	1.0

are characteristic for software and hardware implementations. Every row labeled with cycle accuracy (CA) is applicable for an ATmega128 as well as JAAVR. Using a TCL-based simulation script, we measured the data-memory requirements (including stack) of all implementations. The bit-serial designs needs the least data memory, because the memory for a temporary $2n$-bit product was saved. The RAM and ROM macros are chosen according to the data and program memory requirements. Because those macros are only available in certain sizes, not every difference measured in Bytes is reflected in the actual area of the ROM macro (in gate equivalents).

7.1 Reached Goals

All targeted goals (< 2 mW, < 500 ms @ 13.56 MHz) have been met. An exception are the larger 224-bit and 256-bit elliptic curves which render the `MULACC` based approach as too slow. The runtimes of 100–200 ms show that the clock frequency can be decreased by factors of 2–4, which in turn would decrease the power consumptions by a factor of 2–4.

7.2 Related Work

For a fair comparison with related work, it is important to not only consider plain numbers (chip area, runtime, power), but also the provided features. We distinguish whether a design is microprocessor-based (MCU), comes with a C-compiler, considers side-channel attacks, or performs binary- or prime-field based

Table 3. Comparison with related work.

Reference	Runtime [kCycles]	Area [GE]	Characteristics
ISE - `secp160r1`			
Gura [9]	4,720	-	ATmega-based
OC + `MULACC`	3,268	19,004	ATmega-based
Dedicated Hardware - `secp160r1`			
bit-serial	1,298	22,537	ATmega-based
OC + `MULACC`	3,268	19,004	ATmega-based
Fürbass [6]	362	19,000	ECDSA-like
Dedicated Hardware - `secp192r1`			
Satoh [18]	4,165	29,655	ECC
bit-serial	1,813	23,825	ATmega-based
Fürbass [6]	502	23,600	ECDSA-like
OC + `MULACC`	5,022	21,195	ATmega-based
Hutter [10]	859	19,115	ECDSA, MCU
Wenger [22]	1,377	11,686	ECDSA, MCU

ECC. One must also consider the used manufacturing technology, but this would go beyond the scope of this paper.

A fair comparison with dedicated hardware designs is tough. While they are optimized to the limit, they lack the rich set of features our ASIP provides. The ASIP is easily extendable and provides a compiler toolchain. Also our microprocessor-based approach has not yet reached its limits (c.f. Appendix C), but applying those ideas would make our design incompatible with a standard ATmega128. Table 3 summarizes the comparison with related work.

Fürbass *et al.* [6], Hutter *et al.* [10], Satoh *et al.* [18], and Wenger *et al.* [22] worked on dedicated prime-field based elliptic curve hardware designs. They require 12–30 kGE of hardware and 362–4,165 kCycles of runtime. Although most of their solutions are faster, it is important to notice that our solutions provide sufficiently fast response times. Hutter *et al.* and Wenger *et al.* implemented the full ECDSA signature algorithm and Fürbass *et al.* implemented ECDSA without a hash algorithm. The designs by Hutter *et al.* and Wenger *et al.* is micro controller based, but does not provide a C-compiler. The designs by Fürbass *et al.* and Satoh *et al.* are not micro controller based, so it probably is easier to adapt our designs for real-world scenarios.

Most comparable to this paper are the works of Gura *et al.* [9], Kumar and Paar [15], and Koschuch *et al.* [14]. Gura *et al.* added simulated ISE to an AVR processor, but achieved slower runtimes results. Kumar and Paar used the ATSTK94 FPSLIC demonstration board to extend an AVR processor with a bit-serial multiplier extension for binary extension fields. They however have not applied their methodology to prime fields and do not provide results for an ASIC. Koschuch *et al.* synthesized an 8051-compatible microprocessor and equipped it with a hardware accelerator for binary extension fields. In total, they

needed 29 kGE and 1.2 MCycles. Even though we use prime fields, our results are smaller and approximately of similar speed.

8 Conclusion

After our thorough analysis of instruction-set extensions for ECC, the following conclusions can be drawn: First, if the area footprint is most important (e.g., for RFID) our `MULACC`-based ASIP is the best choice. The chip area is on par with related work and reasonable response times of less than 370 ms are achievable. Second, for applications such as wireless sensor networks or embedded smart cards, the tightly-coupled bit-serial ASIP approach is most suitable. It provides the best energy efficiency and the best area-time product. Third, the design space for ECC implementations is huge: the ratios between the best and the worst implementation across all tested elliptic curves in the categories of area-runtime product, runtime, and energy are 87:1, 101:1, and 65:1, respectively. Our results show that the figures vary by up to two orders of magnitude across the hardware/software design space, which gives a designer a multitude of options to fine-tune a design for a given set of requirements.

Acknowledgements

The author wants to thank Thomas Plos for the support during the creation of this paper. This work has been supported by the Austrian Science Fund (FWF) under grant number TRP 251-N23 (Realizing a Secure Internet of Things - ReSIT).

References

1. D. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards Curves. In *AFRICACRYPT*, volume 5023 of *LNCS*, pages 389–405, 2008.
2. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, 2000.
3. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, October 1990.
4. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
5. H. Eberle, A. Wander, N. Gura, S. Chang-Shantz, and V. Gupta. Architectural Extensions for Elliptic Curve Cryptography over GF(2^m) on 8-bit Microprocessors. In *International Conference on Application-specific Systems, Architectures and Processors*, pages 343–349. IEEE Computer Society, July 2005.
6. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of 2007 IEEE International Symposium on Circuits and Systems*. IEEE, IEEE, May 2007.

7. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In *CRYPTO*, LNCS, pages 190–200, 2001.
8. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES*, pages 133–147, 2004.
9. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In M. Joye and J.-J. Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 119–132. Springer, 2004.
10. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In *RFIDSec*, pages 189–202, 2010.
11. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In *AFRICACRYPT*, volume 6737 of *LNCS*, pages 170–187, 2011.
12. M. Hutter and E. Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In B. P. und Tsuyoshi Takagi, editor, *CHES*, volume 6917 of *LNCS*, pages 459–474. Springer, 2011.
13. International Organization for Standardization (ISO). ISO/IEC 14443-3: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part3: Initialization and Anticollision, 2001.
14. M. Koschuch, J. Lechner, A. Weitzer, J. Großschädl, A. Szekely, S. Tillich, and J. Wolkerstorfer. Hardware/Software Co-design of Elliptic Curve Cryptography on an 8051 Microcontroller. In *CHES*, 2006.
15. S. Kumar and C. Paar. Reconfigurable Instruction Set Extension for Enabling ECC on an 8-Bit Processor. In *Field Programmable Logic and Application*, volume 3203 of *LNCS*, pages 586–595, 2004.
16. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks*, pages 245–256, 2008.
17. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009.
18. A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Transactions on Computers*, 52:449–460, 2003.
19. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks*, LNCS, pages 305–320. Springer, 2008.
20. S. Vaudenay. On Privacy Models for RFID. In *ASIACRYPT*, LNCS, pages 68–87, 2007.
21. E. Wenger, T. Baier, and J. Feichtner. JAAVR: Introducing the Next Generation of Security-Enabled RFID Tags. In *DSD*, pages 640–647, 2012.
22. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In *WISA*, pages 92–106, 2010.

A Decomposition of Instructions

Table 4 shows the decomposition of used instructions for performing a 160×160-bit multiplication with and without instruction-set extension. Using the ISE, the necessary additions were nearly eliminated.

Table 4. Decomposition of the number of cycles per type of instruction necessary for a 160×160-bit multiplication.

Instruction	CA	FAST	ISE
`MUL`	800	400	0
`MULACC`	0	0	400
`LD`	160	80	76
`ST`	120	60	0
`ST_SHIFTACC`	0	0	58
`ADC`	820	820	18
`ADD`	420	420	18
`CLR`	63	63	4
`PUSH`	36	18	18
`POP`	36	18	18
Others	18	17	21
Total	**2,473**	**1,896**	**631**

B Runtimes of Finite-Field Operations

The runtimes of all finite-field operations for `secp160r1` are presented in Table 5. Especially the comparable slow finite-field multiplication greatly profited from the performed optimizations. Because the inversion is based on an exponentiation, the speedup of the inversion is a direct reflection of the speedup of the multiplication. Using the bit-serial multiplier, the ratio between additions and multiplications is only 1.5. This needs to be taken in concern when a method or formula for the point multiplication is chosen.

Table 5. Runtimes of finite-field operations for `secp160r1`.

Operation	CA	FAST	ISE	bit-serial
Addition	291	176	176	176
Subtraction	291	176	176	176
Multiplication	3,024	2,249	984	271
Inversion	519,217	386,368	170,053	48,130

C The Limits

In this paper we concentrated on delivering sufficiently fast and power-aware ASIPs for future RFID technology. We sticked to modifying the processing core and only optimized the finite-field operations in assembly language. However, if you want to make our design into an actual product, further optimizations need to be considered.

Constants consume space within the program and data memory. At startup they are loaded from the program memory and stored to the data memory. If one would add a memory-mapped table within the data memory bus, one could significantly reduce the size of the necessary RAM macro. The RAM macro could be shrunken by at least seven times 160-bit = 140 bytes.

Memory Management is currently performed by the compiler by reserving memory on the stack. If the whole source code would be written in assembly, unnecessary and redundant data memory entries could be avoided.

Processor Features that are not needed for ECC could be removed. E.g. the I/O bus is mapped within the data memory, or `MOVW` instructions are not needed for the finite-field operations. Removing those feature would decrease the size of JAAVR by 420 GE.

Processor Instructions that are not needed for ECC could be removed. For instance the `FMUL*` and `MULS*` multiplier instructions are not needed for an ECC point multiplication.

Program Memory is currently synthesized as Via-1 ROM macros. Using smaller ROM macros would significantly decrease the size of the program memory. Because the program memory is the largest part of the presented design, decreasing its size has a significant impact on the total chip area.

Chapter 8

Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors

Publication Data

Michael Hutter and Erich Wenger. Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems, CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 459–474, 2011.

Contributions

Generalization of the idea. Figures. All implementation results except of the 160-bit implementations. Proofreading of the text.

Fast Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors

Michael Hutter and Erich Wenger

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Michael.Hutter,Erich.Wenger}@iaik.tugraz.at

Abstract. Multi-precision multiplication is one of the most fundamental operations on microprocessors to allow public-key cryptography such as RSA and Elliptic Curve Cryptography (ECC). In this paper, we present a novel multiplication technique that increases the performance of multiplication by sophisticated caching of operands. Our method significantly reduces the number of needed *load* instructions which is usually one of the most expensive operation on modern processors. We evaluate our new technique on an 8-bit ATmega128 microcontroller and compare the result with existing solutions. Our implementation needs only 2,395 clock cycles for a 160-bit multiplication which outperforms related work by a factor of 10 % to 23 %. The number of required *load* instructions is reduced from 167 (needed for the best known hybrid multiplication) to only 80. Our implementation scales very well even for larger Integer sizes (required for RSA) and limited register sets. It further fully complies to existing multiply-accumulate instructions that are integrated in most of the available processors.

Keywords: Multi-Precision Arithmetic, Microprocessors, Elliptic Curve Cryptography, RSA, Embedded Devices.

1 Introduction

Multiplication is one of the most important arithmetic operation in public-key cryptography. It engross most of the resources and execution time of modern microprocessors (up to 80 % for Elliptic Curve Cryptography (ECC) and RSA implementations [6]). In order to increase the performance of multiplication, most effort has been put by researchers and developers to reduce the number of instructions or minimize the amount of memory-access operations.

Common multiplication methods are the schoolbook or Comba [4] technique which are widely used in practice. They require at least $2n^2$ *load* instructions to process all operands and to calculate the necessary partial products. In 2004, Gura et al. [6] presented a new method that combines the advantages of these methods (hybrid multiplication). They reduced the number of *load* instructions to only $2\lceil n^2/d \rceil$ where the parameter d depends on the number of available

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 459-474, 2011.
The original publication is available at http://link.springer.com/chapter/10.1007/978-3-642-23951-9_30

registers of the underlying architecture. They reported a performance gain of about 25 % compared to the classical Comba multiplication. Their 160-bit implementation needs 3,106 clock cycles on an 8-bit ATmega128 microcontroller. Since then, several authors applied this method [7, 12, 14, 15, 17] and proposed various enhancements to further improve the performance. Most of the related work reported between 2,593 and 2,881 clock cycles on the same platform.

In this paper, we present a novel multiplication technique that reduces the number of needed *load* instructions to only $2n^2/e$ where $e > d$. We propose a new way to process the operands which allows efficiently caching of required operands. In order to evaluate the performance, we use the ATmega128 microcontroller and compare the results with related work. For a 160-bit multiplication, 2,395 clock cycles are necessary which is an improvement by a factor of 10 % compared to the best reported implementation of Scott et al. [14] (which need 2,651 clock cycles) and by a factor of about 23 % compared to the work of Gura et al. [6]. We further compare our solution with different Integer sizes (160, 192, 256, 512, 1,024, and 2,048) and register sizes ($e = 2$, 4, 8, 10, and 20). It shows that our solution needs about 15 % less clock cycles for any chosen Integer size. Our solution also scales very well for different register sizes without significant loss of performance. Besides this, the method fully complies with common architectures that support multiply-accumulate instructions using a (Comba-like) triple-register accumulator.

The paper is organized as follows. In Section 2, we describe related work on that topic and give performance numbers for different multiplication techniques. Section 3 describes different multi-precision multiplication techniques used in practice. We describe the operand scanning, product scanning, and the hybrid method and compare them with our solution. In Section 4, we present the results of our evaluations. We describe the ATmega128 architecture and give details about the implementation. Summary and conclusions are given in Section 5.

2 Related Work

In this section, we describe related work on multi-precision multiplication over prime fields. Most of the work given in literature make use of the hybrid-multiplication technique [6] which provides best performance on most microprocessors. This technique was first presented at CHES 2004 where the authors reported a speed improvement of up to 25 % compared to the classical Comba-multiplication technique [4] on 8-bit platforms. Their implementation requires 3,106 clock cycles for a 160-bit multiplication on an ATmega128 [1]. Several authors adopted the idea and applied the method for different devices and environments, e.g. sensor nodes. Wang et al. [18] and Ugus et al. [16] made use of this technique and implemented it on the MICAz motes which feature an ATmega128 microcontroller. Results for the same platform have been also reported by Liu et al. [11] and Szczechowiak et al. [15] in 2008 who provide software libraries (TinyECC and NanoECC) for various sensor-mote platforms. One of the first who improved the implementation of Gura has been due to Uhsadel et

al. [17]. They have been able to reduce the number of needed clock cycles to only 2,881. Further improvements have been also reported by Scott et al. [14]. They introduced additional registers (so-called *carry catchers*) and could increase the performance to 2,651 clock cycles. Note that they fully unrolled the execution sequence to avoid additional clock cycles for loop instructions. Similar results have been also obtained by Kargl et al. [7] in 2008 which reported 2,593 clock cycles for an un-rolled 160-bit multiplication on the ATmega128.

In 2009, Lederer et al. [9] showed that the needed number of addition and move instructions can be reduced by simply rearranging the instructions during execution of the hybrid-multiplication method. Similar findings have been also reported recently by Liu et al. [12] who reported the fastest looped version of the hybrid multiplication needing 2,865 clock cycles in total.

3 Multi-Precision Multiplication Techniques

In the following subsections, we describe common multiplication techniques that are often used in practice. We describe the operand scanning, product scanning, and hybrid multiplication method[1]. The methods differ in several ways how to process the operands and how many *load* and *store* instructions are necessary to perform the calculation. Most of these methods lack in the fact that they load the same operands not only once but several times throughout the algorithm which results in additional and unnecessary clock cycles. We present a new multiplication technique that improves existing solutions by efficiently reducing the *load* instructions through sophisticated caching of operands.

Throughout the paper, we use the following notation. Let a and b be two m-bit large Integers that can be written as multiple-word array structures $A = (A[n-1], \ldots, A[2], A[1], A[0])$ and $B = (B[n-1], \ldots, B[2], B[1], B[0])$. Further let W be the word size of the processor (*e.g.* 8, 16, 32, or 64 bits) and $n = \lceil m/W \rceil$ the number of needed words to represent the Integers a or b. We denote the result of the multiplication by $c = ab$ and represent it in a double-size word array $C = (C[2n-1], \ldots, C[2], C[1], C[0])$.

3.1 Operand-Scanning Method

Among the most simplest way to perform large Integer multiplication is the operand-scanning method (or often referred as *schoolbook* or *row-wise* multiplication method). The multiplication can be implemented using two nested loop operations. The outer loop loads the operand $A[i]$ at index $i = 0 \ldots n-1$ and keeps the value constant inside the inner loop of the algorithm. Within the inner loop, the multiplicand $B[j]$ is loaded word by word and multiplied with the operand $A[i]$. The partial product is then added to the intermediate result of the same column which is usually buffered in a register or stored in data memory.

[1] Note that we do not consider multiplications methods such as Karatsuba-Ofman or FFT in this paper since they are considered to require more resources and memory accesses on common microcontrollers than the given methods [8].

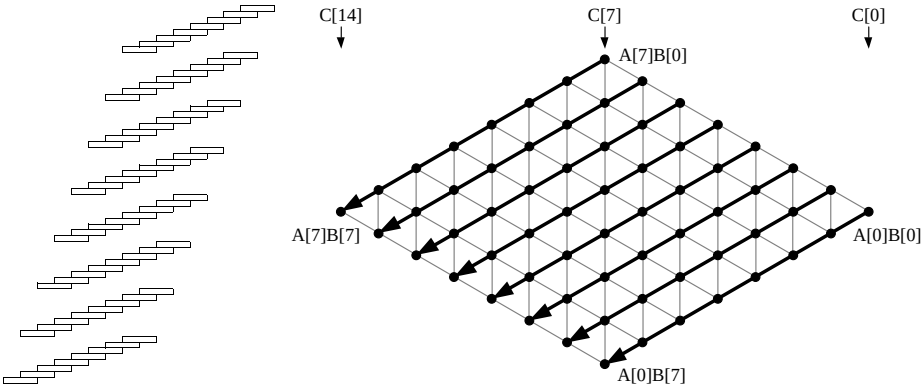

Fig. 1: Operand-scanning multiplication of 8-word large Integers a and b.

Figure 1 shows the structure of the algorithm on the left side. The individual row levels can be clearly discerned. On the right side of the figure, all n^2 partial products are displayed in form of a rhombus. Each point in the rhombus represents a multiplication $A[i] \times B[j]$. The most right-sided corner of the rhombus starts with the lowest indices $i, j = 0$ and the most left-sided corner ends with the highest indices $i, j = n - 1$. By following all multiplications from the right to the lower-mid corner of the rhombus, it can be observed that the operand $A[i]$ keeps constant for any index $i \in [0, n)$. The same holds true for the operand $B[j]$ and $j \in [0, n)$ by following all multiplications from right to the upper-mid corner of the rhombus. Note that this is also valid for the left-handed side of the rhombus.

For the operand-scanning method, it can be seen that the partial products are calculated from the upper-right side to the lower-left side of the rhombus (we marked the processing of the partial products with a black arrow). In each row, n multiplications have to be performed. Furthermore, $2n$ *load* operations and n *store* operations are required to load the multiplicand and the intermediate result $C[i + j]$ and to store the result $C[i + j] \leftarrow C[i + j] + A[i] \times B[j]$. Thus, $3n^2 + 2n$ memory operations are necessary for the entire multi-precision multiplication. Note that this number decreases to $n^2 + 3n$ for architectures that can maintain the intermediate result in available working registers.

3.2 Product-Scanning Method

Another way to perform a multi-precision multiplication is the product-scanning method (also referred as *Comba* [4] or *column-wise* multiplication method). There, each partial product is processed in a column-wise approach. This has several advantages. First, since all operands of each column are multiplied and added consecutively (within a multiply-accumulate approach), a final word of the result is obtained for each column. Thus, no intermediate results have to be stored or loaded throughout the algorithm. In addition, the handling of carry propagation

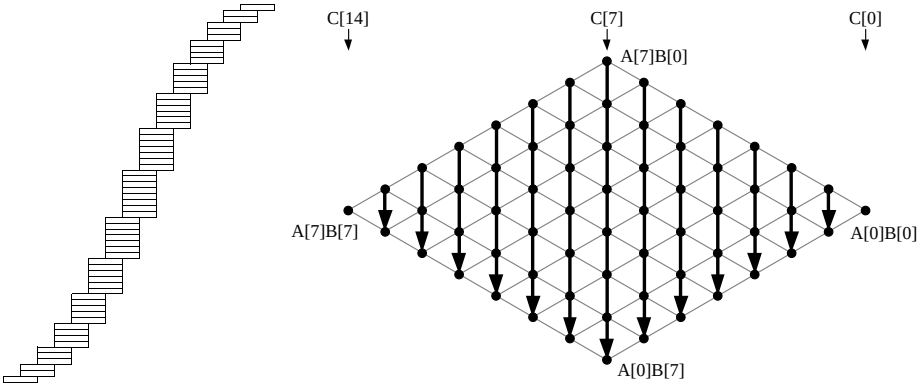

Fig. 2: Product-scanning multiplication of 8-word large Integers a and b.

is very easy because the carry can be simply added to the result of the next column using a simple register-copy operation. Second, only five working registers are needed to perform the multiplication: two registers for the operand and multiplicand and three registers for accumulation[2]. This makes the method very suitable for low-resource devices with limited registers.

Figure 2 shows the structure of the product-scanning method. By having a look at the rhombus, it shows that by processing the partial products in a column-wise instead of a row-wise approach, only one *store* operation is needed to store the final word of the result. For the entire multi-precision operation, $2n^2$ *load* operations are necessary to load the operands $A[i]$ and $B[j]$ and $2n$ *store* operations are needed to store the result. Therefore, $2n^2+2n$ memory operations are needed.

3.3 Hybrid Method

The hybrid multiplication method [6] combines the advantages of the operand-scanning and product-scanning method. It can be implemented using two nested loop structures where the outer loop follows a product-scanning approach and the inner loop performs a multiplication according to the operand-scanning method.

The main idea is to minimize the number of *load* instructions within the inner loop. For this, the accumulator has to be increased to a size of $2d + 1$ registers. The parameter d defines the number of rows within a processed block. Note that the hybrid multiplication is equals to the product-scanning method if parameter d is chosen as $d = 1$ and it is equal to the operand-scanning method if $d = n$.

Figure 3 shows the structure of the hybrid multiplication for $d = 4$. It shows that the partial products are processed in form of individual blocks (we marked

[2] We assume the allocation of three registers for the accumulator register whereas $2 + \lceil log_2(n)/W \rceil$ registers are actually needed to maintain the sum of partial products.

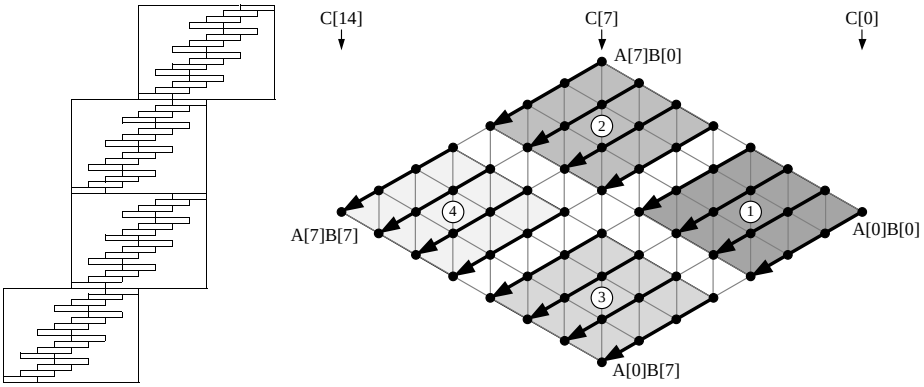

Fig. 3: Hybrid multiplication of 8-word large Integers a and b ($d = 4$).

the processing sequence of the blocks from 1 to 4). Within one block, all operands are processed row by row according to the operand-scanning approach. Note that these blocks use operands with a very limited range of indices. Thus, several *load* instructions can be saved in cases where enough working registers are available. However, the outer loop of the hybrid method processes the blocks in a column-wise approach. So between two consecutive blocks no operands can be shared and all operands have to be loaded from memory again. This becomes clear by having a look at the processing of Block 1-3. Block 2 and 3 do not share any operands that possess the same indices. Therefore, all operands that have already been loaded for Block 1 and that can be reused in Block 3 have to be loaded again after processing of Block 2 which requires additional and unnecessary *load* instructions. However, in total, the hybrid method needs $2\lceil n^2/d \rceil + 2n$ memory-access instructions which provides good performances on devices that feature a large register set.

3.4 Operand-Caching Method

We present a new method to perform multi-precision multiplication. The main idea is to reduce the number of memory accesses to a minimum by efficiently caching of operands. We show that by spending a certain amount of *store* operations, a significant amount of *load* instructions can be saved by reusing operands that have been already loaded in working registers.

The method basically follows the product-scanning approach but divides the calculation into several rows. In fact, the product-scanning method provides best performance if all needed operands can be maintained in working registers. In such a case, only $2n$ *load* instructions and $2n$ *store* instructions would be necessary. However, the product-scanning method becomes inefficient if not enough registers are available or if the Integer size is too large to cache a significant amount of operands. Hence, several *load* instructions are necessary to reload and overwrite the operands in registers.

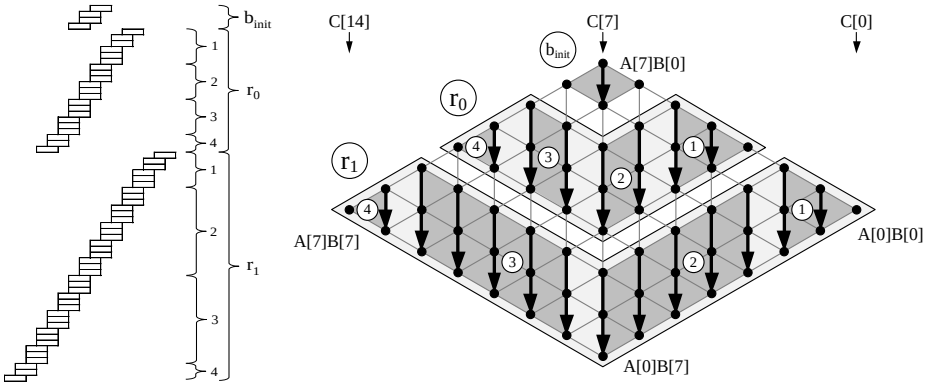

Fig. 4: Operand-caching multiplication of 8-word large Integers a and b ($e = 3$).

In the light of this fact, we propose to separate the product-scanning method into individual rows $r = \lfloor n/e \rfloor$. The size e of each row is chosen in a way that all needed words of one operand can be cached in the available working registers. Figure 4 shows the structure of the proposed method for parameter $e = 3$. That means, 3 registers are reserved to store 3 words of operand a and 3 registers are reserved to store 3 words of operand b. Thus, we assume $f = 2e + 3 = 9$ available registers including a triple-word accumulator. The calculation is now separated into $r = \lfloor 8/3 \rfloor = 2$ rows, *i.e.* r_0 and r_1, and consists of one remaining block which we further denote as initialization block b_{init}. This block calculates the partial products which are not processed by the rows.

All rows are further separated into four parts. Part 1 and 4 use the classical product-scanning approach. Part 2 and 3 perform an efficient multiply-accumulate operation of already cached operands.

The algorithm starts with the calculation of b_{init} and processes the individual rows afterwards (starting from the the smallest to the largest row, *i.e.* from the top to the bottom of the rhombus). Furthermore, all partial products are generated from right to left. In the following, we describe the algorithm in a more detail.

Initialization Block b_{init}. This block (located in the upper-mid of the rhombus) performs the multiplication according to the classical product-scanning method. The Integer size of the b_{init} multiplication is $(n - re)$, *i.e.* $8 - 6 = 2$ in our example, which is by definition smaller than e. Because of that, all operands can be loaded and maintained within the available registers resulting in only $4(n - re)$ memory-access operations. Note that the calculation of b_{init} is only required if there exist remaining partial products, *i.e.* n mod $e \neq 0$. If n mod $e = 0$, the calculation of b_{init} is skipped. Furthermore, consider the special case when $n < e$ where only b_{init} has to be performed skipping the processing of rows (trivial case).

Processing of Rows. In the following, we describe the processing of each row $p = r - 1 \ldots 0$. Each row consists of four parts.

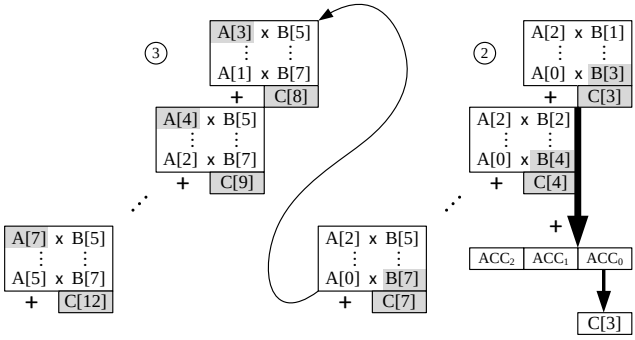

Fig. 5: Processing of Part 2 and 3 of the row r_1.

Part 1. This part starts with a product-scanning multiplication. All operands for that row are first loaded into registers, *i.e.* $A[i]$ with $i = pe \dots e(p+1) - 1$ and $B[j]$ with $j = 0 \dots e - 1$. The sum of all partial products $A[i] \times B[j]$ is then stored as intermediate result to the memory location $C[i]$ (same index range as $A[i]$). Therefore, $2e$ *load* instructions and e *store* instructions are needed.

Part 2. The second part, processes $n - e(p+1)$ columns using a multiply-accumulate approach. Since all operands of $A[i]$ were already loaded and used in Part 1, only one word $B[j]$ has to be loaded from one column to the next. The operands $A[i]$ are kept constant throughout the processing of Part 2. Next to the needed *load* instructions for $B[j]$, we have to load and update the intermediate result of Part 1 with the result obtained in Part 2. Thus, $2(n - e(p+1))$ *load* and $n - e(p+1)$ *store* instructions are required for that part.

Part 3. The third part performs the same operation as described in Part 2 except that the already loaded operands $B[j]$ are kept constant and that one word $A[i]$ is loaded for each column. Figure 5 shows the processing of Part 2 and 3 of row r_1 ($p = 0$). For each column, two *load* instructions are necessary (marked in grey). All other operands have been loaded and cached in previous parts. Operands which are not required for further processing are overwritten by new operands, *e.g.* $B[1] \dots B[4]$ in Part 2 of our example.

Part 4. The last part calculates the remaining partial products. In contrast to Part 1, no *load* instructions are required since all operands have been already loaded in Part 3. Hence, only e memory-access operations are needed to store the remaining words of the (intermediate) result c.

Table 1 summaries the memory-access complexity of the initialization block and the individual parts of a row p. By summing up all *load* instructions, we get

$$2(n - re) + \sum_{p=0}^{r-1}(4n - 4pe - 2e) = 2n + 4rn - 2er^2 - 2er \leq \frac{2n^2}{e}. \qquad (1)$$

Table 1: Memory-access complexity of b_{init} and each part of row $p = 0 \dots r-1$.

Component	Load Instr.	Store Instr.	Total
b_{init}	$2(n-re)$	$2(n-re)$	$4(n-re)$
Part 1	$2e$	e	$3e$
Part 2	$2(n-e(p+1))$	$n-e(p+1)$	$3(n-e(p+1))$
Part 3	$2(n-e(p+1))$	$n-e(p+1)$	$3(n-e(p+1))$
Part 4	0	e	e

The total number of *store* operations can be evaluated by

$$2(n-re)+\sum_{p=0}^{r-1}(2n-2pe)=2n+2rn-er^2-er\leq\frac{n^2}{e}+n. \tag{2}$$

Table 2 lists the complexity of different multi-precision multiplication techniques. It shows that the hybrid method needs $2\lceil\frac{n^2}{d}\rceil$ *load* instructions whereas the operand-caching technique needs about $\frac{2n^2}{e}$. Since the total number of available registers f equals to $2e+3$ for the operand-caching technique ($2e$ registers for the operand registers and three registers for the accumulator) and $3d+2$ for the hybrid method ($d+1$ registers for the operands and $2d+1$ registers for the accumulator), we obtain

$$2e+3=3d+2\implies e=\frac{3d-1}{2}\quad\text{and}\quad e>d. \tag{3}$$

If we compare the total number of memory-access instructions for the hybrid and the operand-caching method and express both runtimes using f, we get

$$2\left\lceil\frac{3n^2}{f-2}\right\rceil+2n>\frac{6n^2}{f-3}+n \tag{4}$$

Note that there are more parameters to consider. The number of additions of the operand-caching method is $3n^2$ and the number of additions of the hybrid method is $n^2(2+d/2)$ (upper bound). Also the pseudocode of Gura et al. [6] for the hybrid multiplication method is inefficient in the special case of $n \bmod d \neq 0$.

Table 2: Memory-access complexity of different multiplication techniques.

Method	Load Instructions	Store Instructions	Memory Instructions
Operand Scanning	$2n^2+n$	n^2+n	$3n^2+2n$
Product Scanning [4]	$2n^2$	$2n$	$2n^2+2n$
Hybrid [6]	$2\lceil n^2/d\rceil$	$2n$	$2\lceil n^2/d\rceil+2n$
Operand Caching	$\boldsymbol{2n^2/e}$	$\boldsymbol{n^2/e+n}$	$\boldsymbol{3n^2/e+n}$

Table 3: Unrolled instruction counts for a 160-bit multiplication on the ATmega128.

Method	**Instruction**						**Clock**
	LD	ST	MUL	ADD	MOVW	Others	**Cycles**
Operand Scanning	820	440	400	1,600	2	464	5,427
Product Scanning	800	40	400	1,200	2	159	3,957
Hybrid (d=4)	200	40	400	1,250	202	109	2,904
Operand Caching (e=10)	**80**	**60**	**400**	**1,240**	**2**	**68**	**2,395**

4 Results

We used the 8-bit ATmega128 microcontroller for evaluating the new multiplication technique. The ATmega128 is part of the megaAVR family from Atmel [1]. It has been widely used in embedded systems, automotive environments, and sensor-node applications. The ATmega128 is based on a RISC architecture and provides 133 instructions [2]. The maximum operating frequency is 16 MHz. The device features 128 kB of flash memory and 4 kB of internal SRAM. There exist 32 8-bit general-purpose registers (R0 to R31). Three 16-bit registers can be used for memory addressing, i.e. R26:R27, R28:R29, and R30:R31 which are denoted as X, Y, and Z. Note that the processor also allows pre-decrement and post-increment functionalities that can be used for efficient addressing of operands. The ATmega128 further provides an hardware multiplier that performs an 8×8-bit multiplication within two clock cycles. The 16-bit result is stored in the registers R0 (lower word) and R1 (higher word).

We used register R22 to store a zero value. Furthermore, we reserved R23, R24, and R25 as accumulator register. Thus, 20 registers, *i.e.* R2...R21, can be used to store and cache the words of the operands ($e = 10$ registers for each operand a and b). All implementations have been done by using a self-written code generator that allows the generation of (looped & unrolled) assembly code.

In order to demonstrate the performance of our method, we implemented all multiplication techniques described in Section 3. For comparison reasons, we decided to implement a 160×160-bit multiplication as it has been done by most of the related work. Note that for RSA and ECC, larger Integer sizes are recommended in practice [10, 13]. The Standards for Efficient Cryptography (SEC) already removed the recommended secp160r1 elliptic curve from their standard since SEC version 2 of 2010 [3].

Table 3 summarizes the instruction counts for the operand scanning, product scanning, hybrid, and operand-caching implementation. The operand-scanning and product-scanning methods have been implemented without using all the available registers (as it usually would be implemented). For hybrid multiplication, we applied $d = 4$ because it allows a better optimization regarding necessary addition operations compared to a multiplication with $d = 5$. The carry propagation problem has been solved by implementing a similar approach as proposed by Liu et al. [12]. Thus, 200 `MOVW` instructions have been necessary to handle the carry propagation accordingly. For a fair comparison, all methods have been

Table 4: Comparison of multiplication methods for different Integer sizes.

Size [bit]	Op. Scan.	Prod. Scan.	Hybrid Method	Operand Caching
160	5,427	3,957	2,904	2,395
192	7,759	5,613	4,144	3,469
256	13,671	9,789	7,284	6,123
512	53,959	38,013	28,644	24,317
1,024	214,407	149,757	113,604	96,933
2,048	854,791	594,429	452,484	387,195

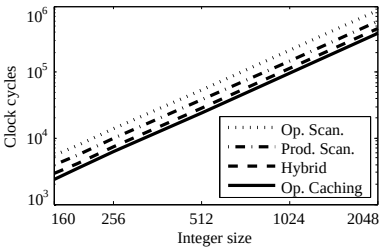

Fig. 6: Comparison chart.

optimized for speed and provide unrolled instruction sequences. Furthermore, we implemented all accumulators as ring buffers to reduce necessary `MOV` instructions. After each partial-product generation, the indices of the accumulator registers are shifted so that no `MOV` instructions are necessary to copy the carry.

Best results have been obtained for the operand-caching technique. By trading additional 20 *store* instructions, up to 120 *load* instructions could be saved when we compare the result with the best reference values (hybrid implementation). Note that *load*, *store*, and *multiply* instructions on the ATmega128 are more expensive than other instructions since they require two clock cycles instead of only one. For operand-caching multiplication, almost the same amount of *load* and *store* instructions are required. In total 2,395 clock cycles are needed to perform the multiplication. Compared to the hybrid implementation, a speed improvement of about 18 % could be achieved.

We also compare the performance of the implemented multi-precision methods for different Integer sizes. Table 4 shows the result for Integer sizes from 160 up to 2,048 bits[3]. The operand-caching technique provides the best performance for any Integer size. It is therefore well suited for large Integer sizes such as it is in the case of RSA. In average, a speed improvement of about 15 % could be achieved compared to the hybrid method. Figure 6 shows the appropriate performance chart in a double logarithmic scale.

[3] Note that due to a fully unrolled implementation such large Integer multiplications might be impractical due to the huge amount of code.

Table 5: Performance of operand-caching multiplication for different Integer sizes and available registers.

Size	e=2	e=4	e=8	e=10	e=20
160	3,915	2,965	2,513	2,395	2,205
192	5,611	4,255	3,577	3,469	3,207
256	9,915	7,531	6,339	6,123	5,671
512	39,291	29,915	25,227	24,317	22,451
1,024	156,411	119,227	100,635	96,933	89,529
2,048	624,123	476,027	401,979	387,195	357,581

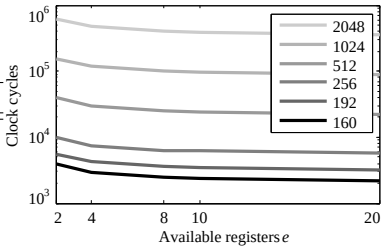

Fig. 7: Performance chart.

Table 6: Comparison with related work.

Method	Instruction						Clock Cycles
	LD	ST	MUL	ADD	MOVW	Others	
Hybrid							
Gura et al. [6] (d=5)	167	40	400	1,360	355	197	3,106
Uhsadel et al. [17] (d=5)	238	40	400	986	355	184	2,881
Scott et al. [14] (d=4)[a]	200	40	400	1,263	70	38	2,651
Liu et al. [12] (d=4)	200	40	400	1,194	212	179	2,865
Operand Caching							
with looping[a,c] (e=9)	**92**	**66**	**400**	**1,252**	**41**	**276**	**2,685**
unrolled[b,c] (e=10)	**80**	**60**	**400**	**1,240**	**2**	**68**	**2,395**

[a] b_{init}, Part 1, and Part 4 unrolled. Part 2 and Part 3 looped.
[b] Fully unrolled implementation without overhead of loop instructions.
[c] w/o `PUSH/POP/CALL/RET`.

Table 5 and Figure 7 show the performance for different Integer sizes in relation to parameter e. The parameter e is defined by the number of available registers to store words of one operand, *i.e.* $e = \frac{f-3}{2}$, where $f = 2e + 3$ denotes the number of available registers in total (including the triple-size register for the accumulator). It shows that for $e > 10$ no significant improvement in speed is obtained. The performance decrease for smaller e and higher Integer sizes. However, if we compare our solution (160-bit multiplication with smallest parameter $e = 2 \rightarrow f = 7$ registers) with the product-scanning method (needing $f = 5$ registers), we obtain 3,915 clock cycles for the operand-caching method and 3,957 clock cycles for the product scanning method. It therefore provides a good performance even for a smaller set of available registers. For the special case $e = 20$, where all 20 words of one 160-bit operand can be maintained in registers (ideal case for product scanning), it shows that the number of clock cycles reaches nearly the optimum of 2,160 clock cycles, i.e. $4n = 80$ memory-access instructions, $n^2 = 400$ multiplications, and $3n^2 = 1,200$ additions.

We compare our result with related work in Table 6. For a fair comparison, we also implemented a operand-caching version that does not unroll the algorithm but includes additional loop instructions. It shows that the operand-caching method provides best performance. Compared to Gura et al. [6] 23 % less clock cycles are needed for a 160-bit multiplication. A 10 % improvement could be achieved compared to the best solution reported in literature [14]. Note that most of the related work need between 167 to 238 *load* instructions which mostly explains the higher amount of needed clock cycles.

5 Conclusions

We presented a novel multiplication technique for embedded microprocessors. The multiplication method reduces the number of necessary *load* instructions

through sophisticated caching of operands. Our solution follows the product-scanning approach but divides the processing into several parts. This allows the scanning of sub-products where most of the operands are kept within the register-set throughout the algorithm.

In order to evaluate our solution, we implemented several multiplication techniques using different Integer sizes on the ATmega128 microcontroller. Using operand-caching multiplication, we require 2,395 clock cycles for a 160-bit multiplication. This result improves the best reported solution by a factor of 10 % [14]. Compared to the hybrid multiplication of Gura et al. [6], we achieved a speed up of 23 %. Our evaluation further showed that our solution scales very well for different Integer sizes used for ECC and RSA. We obtained an improvement of about 15 % for bit sizes between 256 and 2,048 bits compared to a reference implementation of the hybrid multiplication.

It is also worth to note that our multiplication method is perfectly suitable for processors that support multiply-accumulate (`MULACC`) instructions such as ARM or the dsPIC family of microcontrollers. It also fully complies to architectures which support instruction-set extensions for `MULACC` operations such as proposed by Großschädl and Savaş [5].

Acknowledgements. The work has been supported by the European Commission through the ICT program under contract ICT-2007-216646 (European Network of Excellence in Cryptology - ECRYPT II) and under contract ICT-SEC-2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES).

References

1. Atmel Corporation. 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash. Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`, August 2007.
2. Atmel Corporation. 8-bit AVR Instruction Set. Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf`, May 2008.
3. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. Available online at `http://www.secg.org/`, January 2010.
4. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, October 1990.
5. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004.*, volume 3156 of *LNCS*, pages 133–147. Springer, August 2004.
6. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In *CHES 2004, 6th International Workshop, Cambridge, USA, August 11-13, 2004.*, pages 119–132. Springer, 2004.
7. A. Kargl, S. Pyka, and H. Seuschek. Fast Arithmetic on ATmega128 for Elliptic Curve Cryptography. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2008/442, October 2008.

8. Ç. K. Koç. High Speed RSA Implementation. Technical report, RSA Laboratories, RSA Data Security, Inc. 100 Marine Parkway, Suite 500 Redwood City, 1994.
9. C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, and S. Tillich. Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks. In *3rd International Workshop in Information Security Theory and Practices – WISTP 2009, Brussels, Belgium, September 1-4, 2009.*, volume 5746 of *LNCS*, pages 112–127. Springer, 2009.
10. A. Lenstra and E. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
11. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks - IPSN 2008, April 22-24, 2008, St. Louis, Missouri, USA.*, pages 245–256, St. Louis, MO, April 2008.
12. Z. Liu, J. Großschädl, and I. Kizhvatov. Efficient and Side-Channel Resistant RSA Implementation for 8-bit AVR Microcontrollers. In *Workshop on the Security of the Internet of Things - SOCIOT 2010, 1st International Workshop, November 29, 2010, Tokyo, Japan.* IEEE Computer Society, 2010.
13. National Institute of Standards and Technology (NIST). SP800-57 Part 1: DRAFT Recommendation for Key Management: Part 1: General. Available online at `http://csrc.nist.gov/publications/drafts/800-57/Draft_SP800-57-Part1-Rev3_May2011.pdf`, May 2011.
14. M. Scott and P. Szczechowiak. Optimizing Multiprecision Multiplication for Public Key Cryptography. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2007/299, 2007.
15. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In R. Verdone, editor, *Wireless Sensor Networks 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008.*, volume 4913 of *LNCS*, pages 305–320. Springer, 2008.
16. O. Ugus, A. Hessler, and D. Westhoff. Performance of Additive Homomorphic EC-ElGamal Encryption for TinyPEDS. In *GI/ITG KuVS Fachgesprch Drahtlose Sensornetze, RWTH Aachen, 2007.* UbiSec, July 2007.
17. L. Uhsadel, A. Poschmann, and C. Paar. Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes. In *Security and Privacy in Ad-hoc and Sensor Networks 4th European Workshop, ESAS 2007, Cambridge, UK, July 2-3, 2007.*, 2007.
18. H. Wang and Q. Li. Efficient Implementation of Public Key Cryptosystems on Mote Sensors. In *Information and Communications Security 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006.*, volume 4307 of *LNCS*, pages 519–528. Springer, 2006.

A Algorithm for Operand-Caching Multiplication

The following pseudo code shows the algorithm for multi-precision multiplication using the operand-caching method. Variables that are located in data memory are denoted by M_x where x represents the name of the Integer a or b. The parameter e describes the number of locally usable registers $R_a[e-1,\ldots,0]$ and $R_b[e-1,\ldots,0]$. The triple-word accumulator is denoted by $ACC = (ACC_2, ACC_1, ACC_0)$.

Require: word size n, parameter e, $n \geq e$, Integers $a, b \in [0, n), c \in [0, 2n)$.
Ensure: $c = ab$.

```
r = ⌊n/e⌋.
R_A[e − 1, ..., 0] ← M_A[n − 1, ..., re].
R_B[e − 1, ..., 0] ← M_B[n − re − 1, ..., 0].
ACC ← 0.
for i = 0 to n − re − 1 do
  for j = 0 to i do
    ACC ← ACC + R_A[j] ∗ R_B[i − j].
  end for
  M_C[re + i] ← ACC_0.
  (ACC_1, ACC_0) ← (ACC_2, ACC_1).
  ACC_2 ← 0.
end for
for i = 0 to n − re − 2 do
  for j = i + 1 to n − re − 1 do
    ACC ← ACC + R_A[j] ∗ R_B[n − re − j + i].
  end for
  M_C[n + i] ← ACC_0.
  (ACC_1, ACC_0) ← (ACC_2, ACC_1).
  ACC_2 ← 0.
end for
M_C[2n − re − 1] ← ACC_0.
ACC_0 ← 0.                                          } b_init
for p = r − 1 to 0 do                               } Row Loop:
  R_A[e − 1, ..., 0] ← M_A[(p + 1)e − 1, ..., pe].
  R_B[e − 1, ..., 0] ← M_B[e − 1, ..., 0].
  for i = 0 to e − 1 do
    for j = 0 to i do
      ACC ← ACC + R_A[j] ∗ R_B[i − j].
    end for
    M_C[pe + i] ← ACC_0.
    (ACC_1, ACC_0) ← (ACC_2, ACC_1).
    ACC_2 ← 0.
  end for                                           } Part 1
  for i = 0 to n − (p + 1)e − 1 do
    R_B[e − 1, ..., 0] ← M_B[e + i], R_B[e − 2, ..., 1].
    for j = 0 to e − 1 do
      ACC ← ACC + R_A[j] ∗ R_B[e − 1 − j].
    end for
    ACC ← ACC + M_C[(p + 1)e + i].
    M_C[(p + 1)e + i] ← ACC_0.
    (ACC_1, ACC_0) ← (ACC_2, ACC_1).
    ACC_2 ← 0.
  end for                                           } Part 2
```

for $i = 0$ **to** $n - (p+1)e - 1$ **do**
  $R_A[e-1, \ldots, 0] \leftarrow M_A[(p+1)e + i], R_A[e-2, \ldots, 1]$.
  for $j = 0$ **to** $e - 1$ **do**
    $ACC \leftarrow ACC + R_A[j] * R_B[e-1-j]$.
  end for
  $ACC \leftarrow ACC + M_C[(n+i]$.
  $M_C[n+i] \leftarrow ACC_0$.
  $(ACC_1, ACC_0) \leftarrow (ACC_2, ACC_1)$.
  $ACC_2 \leftarrow 0$.
end for
} **Part 3**

for $i = 0$ **to** $e - 2$ **do**
  for $j = i + 1$ **to** $e - 1$ **do**
    $ACC \leftarrow ACC + R_A[j] * R_B[e - j + i]$.
  end for
  $M_C[2n - (p+1)e + i] \leftarrow ACC_0$.
  $(ACC_1, ACC_0) \leftarrow (ACC_2, ACC_1)$.
  $ACC_2 \leftarrow 0$.
end for
$M_C[2n - 1 - pe] \leftarrow ACC_0$.
$ACC_0 \leftarrow 0$.
} **Part 4**

end for
Return c.

B Example: 160-Bit Operand-Caching Multiplication

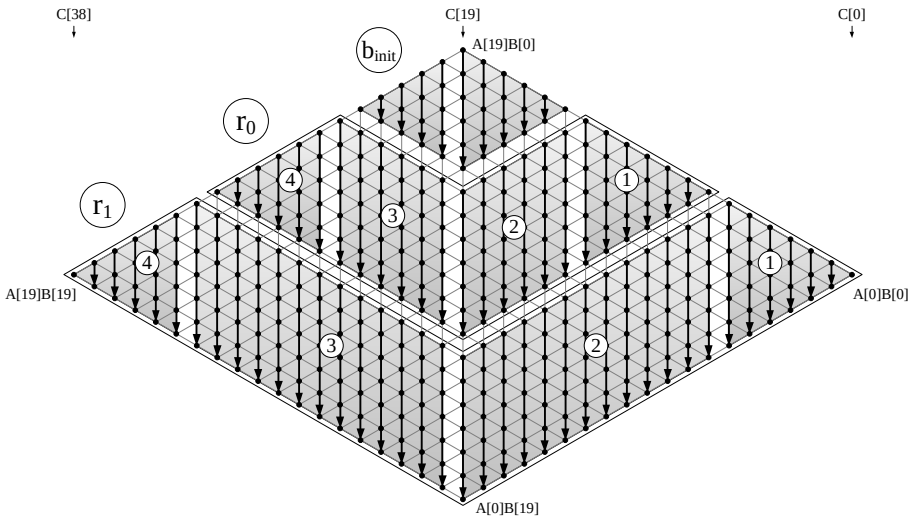

Fig. 8: Operand-caching multiplication for $n = 20$ and $e = 7$.

Chapter 9

A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9 kGEs

Publication Data

Erich Wenger and Michael Hutter. A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9 kGEs. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 182–198. Springer, 2011.

Contributions

Main author. Idea. Implementations. Figures. Tables. Algorithms. 50 % of Text.

A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9 kGEs

Erich Wenger and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Erich.Wenger,Michael.Hutter}@iaik.tugraz.at

Abstract. Elliptic Curve Cryptography (ECC) based processors have gained large attention in the context of embedded-system design due to their ability of efficient implementation. In this paper, we present a low-resource processor that supports ECC operations for less than 9 kGEs. We base our design on an optimized 16-bit microcontroller that provides high flexibility and scalability for various applications. The design allows the use of an optimized RAM-macro block and reduces the complexity by sharing various resources of the controller and the datapath. Our results improves the state of the art in low-resource $\mathbb{F}_{2^{163}}$ ECC implementations (14 % less area needed compared to the best solution reported). The total size of the processor is 8,958 GEs for a 0.13 μm CMOS technology and needs 285 kcycles for a point multiplication. It shows that the proposed solution is well suitable for low-power designs by providing a power consumption of only 3.2 μW at 100 kHz.

Keywords: Low-Resource Hardware Implementation, Elliptic Curve Cryptography, Binary Extension Field, Embedded Systems.

1 Introduction

With the rapid development of more powerful and energy-saving devices, we unwittingly move towards the vision of the Internet of things. The required security services within this vision can be particularly achieved using Elliptic Curve Cryptography (ECC). This paper focuses on a low-resource hardware processor that provides ECC capabilities while meeting the low-area and low-power requirements of embedded systems.

There exist many proposals for low-resource ECC processors. Most of the processors operate on binary-field elliptic curves and use full-precision arithmetic to increase the performance of point multiplication [4, 13, 25, 35]. One of the most efficient solutions in terms of low-resource requirements has been reported by Lee et al. [26].

They presented a processor supporting a small elliptic curve over $\mathbb{F}_{2^{163}}$ which makes use of a tiny 8-bit microcontroller to handle higher-level protocol implementations. The ECC operation of $k \cdot P$ is performed by a separated Modular Arithmetic Logic Unit (MALU). The processor needs 12,506 GEs and 276 kcycles

E. Prouff (Ed.): CARDIS 2011, LNCS 7079, pp. 182–198, 2011.
The original publication is available at http://link.springer.com/chapter/10.1007/978-3-642-27257-8_12

to perform a point multiplication. However, the area estimations do not including program ROM and RAM to store intermediate results and the necessary secret scalar k. Similar datapath architectures have been reported by Batina et al. [2] and Sakiyama et al. [32]. Hein et al. [17] reported a very efficient co-processor (without microcontroller) for the same elliptic curve supporting multi-precision arithmetics. They applied a finite-state machine based control-engine needing 11,904 GEs including a standard-cell based RAM memory.

In this paper, we present a low-resource hardware processor that is based on a 16-bit multi-precision architecture and an area-optimized custom microcontroller. This combination allows several optimizations. First, it allows the use of an efficient RAM-macro block that reduces the area requirements for short-term memory significantly. Second, since both the microcontroller and the datapath use a 16-bit architecture, all resources are shared to minimize the area footprint of the processor. As an outcome, we present a complete solution including memory for short-term (RAM) as well as long-term storage (program ROM), controller, and datapath using a polynomial multiply-accumulate (MAC) unit. In addition, we present results of higher-level protocol implementations of the Elliptic Curve Digital Signature Algorithm (ECDSA) [30] and give results for digital signature generation as well as verification. For a point multiplication, our NIST B-163 based processor needs only 8,958 GEs in total and performs a point multiplication within 285 kcycles. We demonstrate that the proposed solution is also well suitable for low-resource embedded systems by providing a power consumption of only 3.2 μW at 100 kHz.

The rest of the article is structured as follows. In Section 2, a brief introduction into elliptic curve cryptography is given. In Section 3, we face the challenge of low-resource ECC hardware implementations and explore various design possibilities. We evaluate appropriate word sizes of a processor and analyze different memory types. Section 4 presents details about the hardware architecture of our processor. Details about the implementation are given in Section 5. In Section 6, the results are presented. Conclusions are drawn in Section 7.

2 Elliptic Curve Cryptography

Within Elliptic Curve Cryptography (ECC), not only a single number or polynomial is used, but a pair of those. Each pair (x, y) of such numbers that satisfy the general Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

is called a point on an elliptic curve. When a certain type of number is used, in our case binary polynomials within $GF(2^m)$, the Weierstrass equation can be reduced to

$$y^2 + xy = x^3 + ax^2 + b. \tag{2}$$

Among the most critical operation in terms of speed and security is the ECC point multiplication. The implementation of this multiplication has to be secure

against various implementation attacks such as side-channel and fault-analysis attacks. The Montgomery ladder [28, 21] provides very beneficial properties in this context. We therefore decided to use it for our design and applied the very fast group-operation formulas of López and Dahab [27]. The formulas are based on projective coordinates (which avoid expensive field inversions) that can be nicely combined with proposed countermeasures (see also the work of Junfeng Fan et al. [11]) such as randomized projective coordinates (RPC) [6] or point-validity checks [8].

We use the following notations throughout the paper (similar to [16]). Let $f(z) = z^m + r(z)$ denote an irreducible binary polynomial of degree m. The elements of $\mathbb{F}_{2^m}$ are binary polynomials of degree at most $m-1$. An addition of field elements is the usual addition of binary polynomials. Multiplication is performed modulo $f(z)$. A field element $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ is associated with the binary vector $a = (a_{m-1}, \ldots, a_2, a_1, a_0)$ of length m. Furthermore, let $N = \lceil m/W \rceil$ be the number of words with width W needed to store $a(z)$. $A = (A[N-1], \ldots, A[2], A[1], A[0])$, where the rightmost bit of $A[0]$ is a_0, and the leftmost $(WN - m)$ bits of $A[N-1]$ are unused (always set to zero).

For further readings on ECC we refer to several books [1, 3, 16, 23] that discuss the topic extensively.

3 Design-Space Exploration

In this section, we will explore different hardware-design options to obtain best results for a low-resource ECC processor. The design goals have been to meet all requirements of embedded systems which are low area (due to the production costs), low power (due to a possible contactless operation), appropriate speed (required for certain applications), security and flexibility. Due to the latter requirement, we decided to base our design on a customized microcontroller. This has the advantage of being modular in terms of protocol implementations and modifications of already implemented solutions.

By following the principles of hardware/software co-design, it showed that the dominant factors of ECC processors are the finite-field hardware multiplier and the type and size of the applied data memory. In the following, we discuss these factors and explore the design space to find the best solution for our objectives.

3.1 The Hardware Multiplier

One of the most area consuming parts within the ALU of an ECC-hardware design is the finite-field multiplier. The size, speed, and power consumption of such a multiplier largely depends on the word size of the processor and the underlying finite field. Figure 1 shows the hardware architecture of a 4-bit multiplier for binary-field (carry-less multiplier), prime-field (integer multiplier), and dual-field arithmetic. The basic structure of all three types of multiplier is the same. Only the adder structure needs to be adopted.

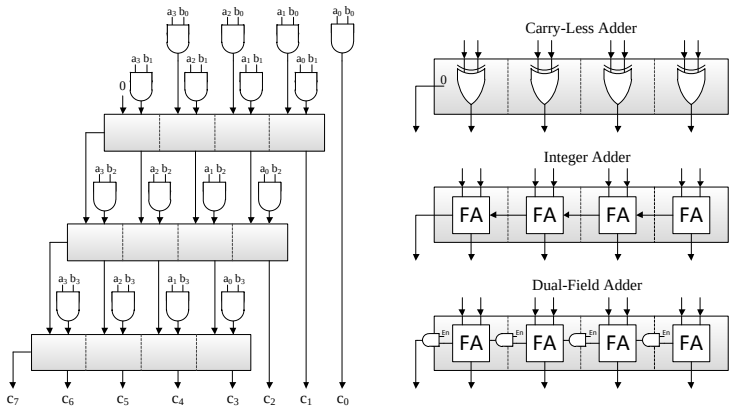

Fig. 1. General 4-bit multiplier structure to the left. Carry-less, integer, and dual-field adder (from top to bottom) on the right.

Table 1 shows the area evaluation of different hardware-multiplier types. We evaluated multipliers for prime-field, binary-field, and dual-field arithmetic for word sizes of 8, 16, 32, and 64 bits (on register-transfer level). For the evaluation we used the UMC-L130 CMOS technology where an AND gate needs 1.25 GEs, a XOR gate needs 2.75 GEs, and a full-adder cell needs 5.5 GEs.

Obviously the area requirement scales quadratically with the given word size and carry-less multipliers provide the lowest area footprint and lowest increase in area for all given word sizes. Runtime approximations for an ECC point multiplication showed that the word size of the carry-less multiplier must be at least 16 bits in order to achieve a sensible runtime.

Next to a carry-less multiplier, an integer multiplier is necessary to provide operations for higher-level protocols (*e.g.* ECDSA). Note that this multiplier is needed only very few times for most protocols (only four prime field multiplications are required for ECDSA signature generation, for instance). Thus, lower word sizes are acceptable since no significant reduction in speed is expected. We therefore decided to implement a 16-bit carry-less multiplier (to provide an appropriate speed for a point multiplication) and an 8-bit integer multiplier instead of a dual-field 16-bit multiplier (which needs 1,946 GEs). This would sum up to 1,226 GEs which is 720 GEs less than for a dual-field multiplier.

Table 1. Area evaluation of different hardware-multiplier types.

Finite Field	**Required adder cells per bit**	**8 bit [GE]**	**16 bit [GE]**	**32 bit [GE]**	**64 bit [GE]**
$GF(2^m)$	XOR	211	850	3,389	13,508
$GF(p)$	FA	376	1,616	6,688	26,336
Dual field	AND + FA	458	1,946	8,018	31,514

Table 2. Area evaluation of different 16 × 128-bit RAM architectures.

Type	Port	Storage [GEs]	Logic [GEs]	Total [GEs]
Std. cells (registers)	Single	10,281	3,645	13,926
Std. cells (latches)	Single	8,388	3,833	12,221
Macro S-RAM	Dual	-	-	6,737
Macro S-RAM[a]	Single	-	-	6,000
Macro register-file	**Single**	-	-	**2,955**

[a] Approximated based on UMC 180 nm technology.

3.2 The Memory Type and Architecture

One of the most area expensive chip components of ECC processors is the Random Access Memory (RAM). RAM is necessary to store intermediate values (*e.g.* point coordinates during point multiplication $k \cdot P$) and the secret scalar k. The size of the memory varies depending on the requirements of the ECC formulas (the formulas of López Dahab [27] need at least 5 registers of memory for full-precision architectures and 6 registers for multi-precision architectures due to the need of intermediate storage of in-place operations).

In Table 2, we compare different 16 × 128-bit RAM types concerning their area requirements. We compare standard-cell based implementations with dedicated RAM macro blocks synthesized in CMOS UMC-L130 technology. The standard-cell based RAM implementations (register and latch based) have been designed on RTL-level and synthesized using Cadence RTL compiler [5]. The RAM-macro blocks have been generated using the Standard Memory Compiler FSA0A Memaker 200901.1.1 by the Faraday Technology Corporation [12]. All except of one type of RAM provide a single read-port and a single write-port. There is one S-RAM macro that features a dual-port read/write interface.

It shows that the latch-based RAM is about 12 % smaller than the register-based RAM. This is because the size of a flip-flop is 5 GE and the size of a latch is 4 GE. This 25 % difference in area is debilitated because some additional registers and control logic is required so that the latch-based RAM works the same way as the register-based RAM. Adding a second read port to those RAMs would be relatively cheap in terms of chip area (it would require about 3,000 GEs in addition by introducing a second multiplexer at the output). Note that a dual-port memory would increase the performance of a multi-precision multiplication by a factor of about two.

From the two available single-port RAM macros, the register-file macro is about 50 % smaller than the S-RAM macro. The dual-port S-RAM macro, in contrast, is only 12 % larger than the single-port S-RAM macro, however, it is about 2.3 times larger than the register-file based RAM macro.

The register-file RAM macro provides best performance in our evaluation scenario. We performed several power simulations using Cadence Encounter and obtained similar results for the register-file RAM macro and the standard-cell

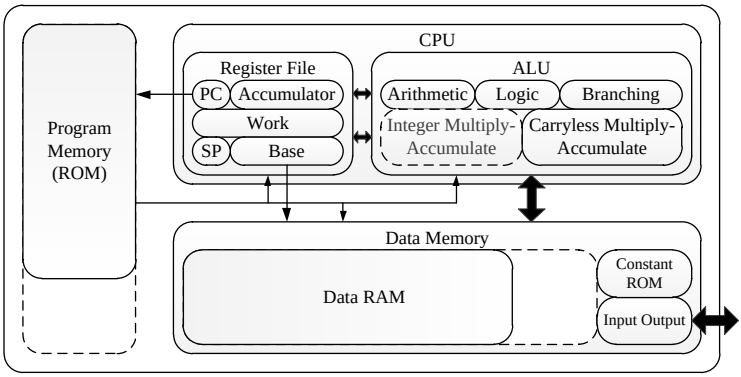

Fig. 2. High-level block diagram of the processor. Components for higher-level protocols are drawn with slashed lines (*i.e.* integer multiplier, program and data memory).

based RAM architectures. The main disadvantages of the register-file macro are the lack of a second read port (speed) and the limit of clock-synchronous read operations. The lack of a second read port can be compensated by using temporary working registers. The lack of an asynchronous read functionality can be balanced with a more difficult control logic.

4 Hardware Architecture

In this section, we introduce the hardware architecture of our processor. It is based on the microprocessor design called Neptun [34], which uses a Harvard architecture. This allows to fetch, decode, execute, and store data within the same clock cycle and allows low-area optimizations due to the choice of different memory types and sizes. Figure 2 shows the block diagram of the architecture. It is mainly composed of a Central Processing Unit (CPU) including register file and Arithmetic Logic Unit (ALU), and memories for program code, constants, and data.

4.1 Central Processing Unit (CPU)

The heart of the processor is the 16-bit CPU. It is composed of several internal registers and an ECC optimized ALU. The register file consists of a program counter (PC), a stack pointer (SP), three base registers, four working registers, and an accumulator register: The program counter is used as index for the program memory. The stack pointer (SP) is needed to store registers on the data memory. The stack is also used to store program-return addresses that are needed for function calls. In order to address certain base addresses within the data memory, three base registers are used. We integrated two source registers and one destination register. They are used together with a 4-bit offset to address data in the memory. The offset address is stored within a program word. We

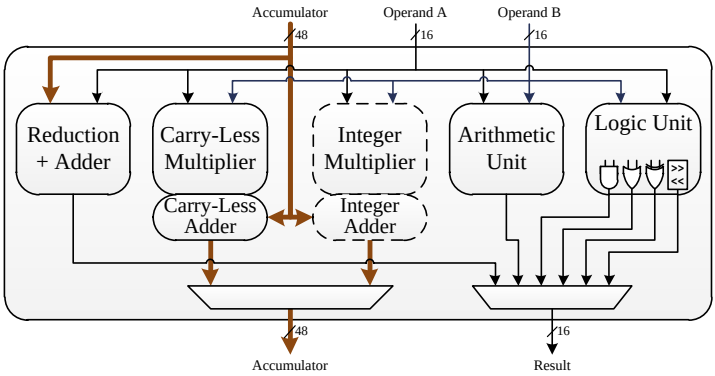

Fig. 3. High-level diagram of the arithmetic logic unit.

implemented four 16-bit working registers that can be used as general-purpose registers. The registers are needed for almost any ECC operation and are used to reduce the number of memory-read cycles within the finite-field multiplication. The accumulator register (ACC) is needed for the multiply-accumulate operation of the 163-bit multi-precision multiplication.

We integrated several optimizations to increase the performance of ECC operations. First, the ALU accesses data directly without loading it first into CPU registers (as it is in the case of conventional microcontrollers). In the first clock cycle, the data is addressed in the memory. In the second cycle, the data is processed by the ALU and the result is stored back in memory within the same clock cycle. This increases the performance of memory-access operations significantly. Second, loading and processing of data is done simultaneously by the processor. This avoids unnecessary idle cycles and improves the efficiency of multi-precision arithmetic operations. Those optimizations are described in more detail in [34].

Arithmetic Logic Unit (ALU). The arithmetic logic unit (ALU) mainly consists of a reduction-logic unit, a carry-less multiplier, an arithmetic unit (addition/subtraction), and a logic unit (supporting OR, AND, XOR, and shift operations). For higher-level protocols, an integer multiplier is needed in addition (drawn with dashed lines). Figure 3 shows a high-level diagram of the ALU. We also integrated an operand isolation technique for each submodule which reduces the power-consumption significantly.

4.2 Memory for Program, Data, and Constants

Our processor provides a long-term storage memory that mainly stores the program for ECC point multiplication. The memory provides 72 control signals and contains up to 1,800 entries depending on the implemented algorithms and higher-level protocols. Most of the control signals are used to control the dataflow

within the CPU. Best area results have been achieved by directly synthesizing the memory table as Read Only Memory (ROM) using standard cells. Experiments in which a 16-bit instruction set or a ROM macro have been introduced resulted in a larger area requirement.

For short-term data storage, we used a 16-bit RAM macro (register-file based) as discussed in Section 3. Note that in contrast to most processors reported in literature [4, 25, 26, 31], we include the number for the required storage of the secret scalar k. For an ECC point multiplication, 1,296 bits (81 entries) are necessary (we used a 16×84 macro in that case). For higher-level protocols, additional memory is needed (*e.g.* 1,536 bits for ECDSA signature generation (16×96 macro) and 2,384 bits for ECDSA signature verification (16×152 macro)).

ECC constants have been stored in a ROM. The ROM has been implemented as a look-up table and stores between 880 and 2,564 bits such as the x and y coordinate of the base point P, the ECC parameters a and b (see Equation (2)), and the irreducible polynomial $f(z)$.

The input/output of data has been realized via memory mapped I/O. Data can be written and read using a 16-bit parallel interface.

5 Implementation Details

In the following, we give details about the implemented carryless multiply-accumulate unit and the modular arithmetics in order to perform ECC operations.

5.1 Carry-Less Multiply-Accumulate Unit

The multi-precision multiplication over $\mathbb{F}_{2^{163}}$ has been realized following a multiply-accumulate (MAC) approach. There exist several publications that make use of MAC units to increase the performance of modular multiplication (see *e.g.* the work of [9, 14, 15, 17, 33]). We implemented the multiplication by a product-scanning form (often referred as Comba multiplication), where each partial product of $A[i] \cdot B[j]$ gets accumulated to a common sum (ACC_1, ACC_0), *i.e.* $(ACC_1, ACC_0) \leftarrow (ACC_1, ACC_0) + A[i] \cdot B[j]$.

Note that for the polynomial MAC unit the handling of carry propagation is not needed. Thus, the accumulator register needs a size of only $(2W - 1)$ bits.

We implemented several improvements to increase the performance. First, the entire multiplication algorithm has been unrolled so that no extra cycles are wasted for loop operations. Second, we reused the working registers as a memory cache to reduce the number of necessary load operations. With each working register used, the total number of read operations has been reduced by about $2N$. Third, we added a third word to the accumulator register (ACC_2, ACC_1, ACC_0) in order to allow efficient reduction of the accumulated sum. Thus, the MAC operation is performed on the words (ACC_2, ACC_1) instead of (ACC_1, ACC_0) and

Algorithm 1 Polynomial multiplication with interleaved reduction.

Require: Binary polynomials $a(z)$ and $b(z)$ of degree at most $m-1$.
Ensure: $c(z) = a(z) \cdot b(z) \bmod f(z)$.
1: $ACC \leftarrow 0$
2: **for** i **from** 0 **to** $N-1$ **do**
3: **for** each element of $\{(i,j)|i+j=k, 0 \leq i,j \leq N-1\}$ **do**
4: $(ACC_2, ACC_1) \leftarrow (ACC_2, ACC_1) + A[i] \cdot B[j]$.
5: **end for**
6: $C[k] \leftarrow ACC_1$.
7: $ACC \leftarrow ACC \gg W$.
8: **end for**
9: $ACC \leftarrow higher(ACC)$.
10: **for** k **from** t **to** $2N-2$ **do**
11: **for** each element of $\{(i,j)|i+j=k, 0 \leq i,j \leq t-1\}$ **do**
12: $(ACC_2, ACC_1) \leftarrow (ACC_2, ACC_1) + A[i] \cdot B[j]$.
13: **end for**
14: $C[k-N-1] \leftarrow C[k-N-1] + reduce(ACC)$.
15: $ACC \leftarrow ACC \gg W$.
16: **end for**
17: $C[N-1] \leftarrow lower(C[N-1]) + reduce(ACC)$.
18: $ACC \leftarrow ACC \gg W$.
19: $C[0] \leftarrow C[0] + reduce(ACC + higher(C[N-1])) \gg W$.
20: $C[N-1] \leftarrow lower(C[N-1]) \gg W$.
21: **Return**(c).

ACC_0 is used to store the previous intermediate result. A detailed description of the reduction method is given in the following subsection.

Algorithm 1 shows the algorithm of the implemented polynomial multiplication. The polynomials $a(z)$ and $b(z)$ get multiplied and the reduced result is stored in $c(z)$. In the lines 1 to 8, the lower N words of the result $c(z)$ are calculated. Note that in this phase the ACC_0 register is not used. In line 9, the lower $(m-W(N-1))$ bits of the accumulator need to be cleared. Those are the bits of the results that do not need to be reduced. The lines 10-16 calculate the higher N words of $c(z)$ and reduce them immediately. According to the recommended NIST irreducible polynomial B-163 $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, the reduction function (line 14) can be written as

$$reduce(ACC) = \Big(ACC \gg (W+3) + ACC \gg W + ACC \gg (W-3) + ACC \gg (W-4)\Big) \wedge (2^W - 1). \quad (3)$$

Finally, in lines 17-20 the rest of the accumulator and the higher bits of $C[N-1]$ get reduced.

Polynomial NIST B-163 Reduction Logic. We make use of the recommended NIST irreducible polynomial B-163 to perform a very efficient modular

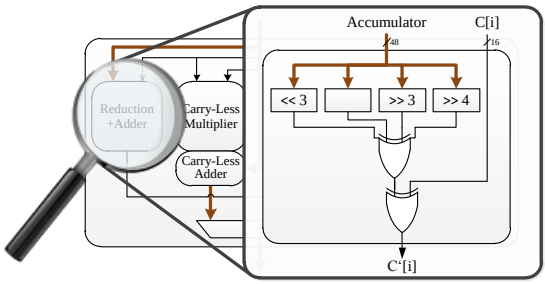

Fig. 4. Using the dedicated reduction logic, the content of the accumulator is reduced and stored in $C'[i]$ (hold in data memory) with index i.

reduction for modular multiplication and squaring. The reduction logic is shown in Figure 4. We hard-wired the output of the appropriate accumulator register according to Equation (3). The reduction logic takes the output of the 48-bit accumulator register, performs 4×16 XOR operations and the result is added with the intermediate result $C[i] = C[k - N - 1]$ (see line 14 in Algorithm 1). After the addition (XOR), the variable $C[i]$ is updated with $C'[i]$ in the data memory. Only one clock cycle is needed to reduce the intermediate result of the accumulator and sum of partial products, respectively. Figure 4 shows the dedicated reduction logic.

It should be noted that although the reduction logic has been specially optimized for NIST B-163, the CPU is capable of handling arbitrary irreducible polynomials. Thus requirements such as flexibility and extendability are ensured.

5.2 Modular Arithmetic

Modular Addition. The simplest operation is the modular addition. It is a simple XOR operation. Neither a carry flag nor a finite-field reduction need to be considered. Modular addition over $\mathbb{F}_{2^{163}}$ needs 35 clock cycles on our processor.

Modular Multiplication. Modular multiplication has been realized using the carryless multiply-accumulate unit described in Section 5.1. Our processor needs 222 clock cycles for a 163-bit multiplication.

Modular Squaring. Modular squaring can be performed very efficiently. The binary representation of the polynomial can be easily squared by inserting a 0 between each consecutive bit of the polynomial, e.g. $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ would results in $a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0$. This can be realized with only a few additional hardware components. The polynomial-reduction logic can be reused for squaring. One modular squaring needs 41 clock cycles on our processor and thus is 5.4 times faster than a modular multiplication.

Modular Inversion. Modular inversion is required to transform the projective coordinates back into affine. For this operation, we made use of Fermat's

Table 3. Size and power estimations of our processor for different CMOS technologies using Latch-based RAMs.

Technology	Area $[\mu m^2]$	NAND Gate $[\mu m^2]$	Total Area $[GE]$	Power $[\mu W$@1MHz$]$	Leakage $[\mu W]$
AMS c35b4	693,948	54.600	12,710	696.3	0.63
UMC f180GII	139,469	9.374	14,878	107.1	0.53
UMC f130SP	71,745	5.120	14,013	31.4	1.37
UMC f090SP	39,550	3.136	12,612	70.1	54.32

little theorem [20] that states that $a = a^{2^m} \bmod f(z) \, \forall a \in \mathbb{F}_{2^m}$. As a result, $a^{-1} \equiv a^{2^m-2} \bmod f(z)$. This exponentiation can be performed using 162 squaring and only 9 multiplications for the NIST B-163 binary field. As a result 11,031 cycles are needed for an inversion.

6 Results

We synthesized our processor using different CMOS technologies from various manufacturers. For synthesis, we used the Cadence RTL compiler [5] Version v08.10. Table 3 shows the total area and power-consumption estimation of the processor using latch-based RAMs[1] (described in Section 3.2). The power-consumption estimations were made using Cadence Encounter Version v08.10. All obtained area results are within a 20 % margin. In view of power consumption, best performance had been obtained for the UMC-L130 technology. For all following approximations we used register-based RAM macros.

In Table 4, the area and power requirements for individual chip components are listed. The memory needs most of the area which is 5,399 GEs. The CPU needs 3,556 GEs in total where only 849 GEs are used for the carry-less multiplier. The total size of the processor sums up to 8,958 GEs.

In Table 5, we compare our results with related work. There exist many publications of ECC processors over $\mathbb{F}_{2^{163}}$. Most of those processors use full-precision arithmetic to perform the point multiplication. For a fair comparison, we listed the results of the authors for different digit sizes (d=1...8). All implementations need between 10,392 GEs and 16,247 GEs of chip area and between 47 and 430 kcycles for the computation of $k \cdot P$. Our implementation needs 8,958 GEs of area which is 1,434 GEs less area than the best reported solution. This is an area improvement by about 14 %. The number of needed clock cycles can be compared with the full-precision solutions with d=1. The power and energy consumption is very low and fulfills most requirements of embedded-system designs.

6.1 Results for Higher-Level Protocol Implementations

As a higher-level protocol, we implemented the Elliptic Curve Digital Signature Algorithm (ECDSA) [30]. In addition to a point multiplication over the binary

[1] We did not have access to RAM macros for all those technologies.

Table 4. Size and power consumption of individual chip components.

Component	**Area** [GE]	**Area** [%]	**Power** [μW@1MHz]	**Power** [%]
Memory	5,399	60.27	11.57	35.77
Program memory	2,471	27.58	4.24	13.10
Data RAM	2,528	28.22	4.66	14.41
Constant ROM	256	2.56	1.62	5.01
CPU	3,556	39.70	18.93	58.54
ALU	1,837	20.51	11.05	34.16
Carry-less multiplier	849	9.48	2.30	7.12
Logic unit	348	3.88	2.15	6.65
Arithmetic unit	93	1.04	0.37	1.15
Register Set	875	9.77	1.48	4.58
Total Area	**8,958**	**100.00**	**32.34**	**100.00**

field $\mathbb{F}_{2^{163}}$, ECDSA needs a hash function and several prime-field arithmetic operations to generate and verify a digital signature. As a hash function, we implemented the 160-bit SHA-1 algorithm according to ISO/IEC FIPS-180-3 [29]. Replacing the SHA-1 algorithm with one of the current SHA-3 candidates [19] would be easily possible. For prime-field multiplications and inversion, we decided to implemented Montgomery-arithmetic operations. We implemented the Finely Integrated Product Scanning Form (FIPS) according to Koç et al. [24]. The algorithm is used only four times, so we optimized the code for low area (no

Table 5. Comparison with related work.

Related Work	**Area** [GE]	**Cycles** [kCycles]	**Power** [μW@1MHz]	**Energy** [μJ]	**CMOS Technology**
Kumar06 d=1 [25]	15,094	430	-	-	AMI C35
Batina06[a] d=4 [2]	14,816	95	27.00	2.57	130 nm
Batina06[a] d=3 [2]	14,258	125	27.00	3.38	130 nm
Batina06[a] d=2 [2]	13,681	182	27.00	4.91	130 nm
Batina06[a] d=1 [2]	13,104	354	27.00	9.56	130 nm
Bock08 d=8 [4]	16,247	47	148.76	6.99	INF SRF55V01P
Bock08 d=4 [4]	12,876	80	93.27	7.46	INF SRF55V01P
Bock08 d=1 [4]	10,392	280	54.31	15.21	INF SRF55V01P
Lee08 d=4 [26]	15,356	79	37.39	2.95	UMC L130
Lee08 d=3 [26]	14,729	101	38.32	3.87	UMC L130
Lee08 d=2 [26]	14,064	145	36.52	5.30	UMC L130
Lee08 d=1 [26]	12,506	276	32.42	8.95	UMC L130
Hein08 16-bit [17]	11,904	296	101.87	30.15	UMC L180
This work 16-bit	**8,958**	**286**	**32.34**	**9.25**	**UMC L130**

[a] For a fair comparison a RAM approximated with 4,890 GE was added. The power values lack the power consumption of this RAM.

Table 6. Area and power estimations of our processor supporting ECDSA.

Program	Area [GE]	Cycles [kCycles]	Lines of Code	Power [μW@1MHz]	Energy [μJ]
ECC Only	8,958	294	637	32.09	9.43
ECC Protected[a]	9,728	298	828	32.48	9.68
ECDSA Sign[a,b]	15,387	378	1771	41.11	15.54
ECDSA Verify[b]	16,005	605	1784	40.76	24.66

[a] The numbers include y-recovery, randomized projective coordinates (RPC) side-channel countermeasure [6], and ECC point-validity check [8].

[b] Includes the SHA-1 hash function [29], Random Number Generation (RNG) [30], and prime-field arithmetics.

loop unrolling etc.). Furthermore, we implemented the Montgomery-inversion algorithm according to Kalinski et al. [22].

For signature verification, we applied Shamir's trick [7, 10] to improve the performance of multiple-point multiplication. All described operations for ECDSA have been implemented as Assembler functions for our processor and have been stored in program memory. Table 6 shows the results after synthesizing the processor. For ECDSA signature generation, our processor needs 15,387 GEs which outperforms existing solutions in terms of area, power, and speed [13, 18, 34, 35]. Signature verification can be realized using a chip area of 16,005 GEs.

7 Conclusions

In this paper, we presented a low-resource implementation of an ECC hardware processor. The processor needs 8,958 GEs and performs a point multiplication within 285 kcycles. The power consumption is about 3.2 μW at 100 kHz. We met the low-resource constraints of embedded systems by applying a very modular microcontroller architecture that allows the execution of higher-level protocols like ECDSA. The elliptic-curve operations have been performed over the NIST $\mathbb{F}_{2^{163}}$ elliptic curve using multi-precision arithmetic. The outcome improves the state of the art in low area ECC hardware designs and provides even a smaller area footprint than most of the proposed SHA-3 candidates [19].

Acknowledgements

We would like to thank Mario Kirschbaum and Martin Feldhofer for several fruitful discussions.

This work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems under the project number 825743 (project PIT).

References

1. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, 2005.
2. L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks – ESAS 2006, Third European Workshop, Hamburg, Germany, September 20-21, 2006, Revised Selected Papers*, volume 4357, pages 6–17, Berlin Heidelberg, 2006. Springer-Verlag.
3. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Notes Series.* Cambridge University Press, Cambridge, UK, 1999.
4. H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008, July 2008.
5. Cadence Design Systems. The Cadence Design Systems Website. `http://www.cadence.com/`.
6. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
7. P. de Rooij. Efficient Exponentiation using Procomputation and Vector Addition Chains. In A. D. Santis, editor, *Advances in Cryptology EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399. Springer Berlin / Heidelberg, 1994.
8. N. Ebeid and R. Lambert. Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, Lausanne, Switzerland, 2009, Proceedings*, pages 46–50, September 2009.
9. H. Eberle, N. Gura, S. C. Shantz, V. Gupta, and L. Rarick. A Public-key Cryptographic Processor for RSA and ECC. In *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2004)*, pages 98–110. IEEE Computer Society, September 2004.
10. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.
11. J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on known Side-Channel Attacks and Countermeasures. In *Hardware-Oriented Security and Trust - HOST 2010, In 3rd IEEE International Symposium, California, USA, June 13-14, 2010, Proceedings.*, pages 76–87. IEEE, 2010.
12. Faraday Technology Corporation. Faraday FSA0A_C 0.18 μm ASIC Standard Cell Library, 2004. Details available online at `http://www.faraday-tech.com`.
13. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of 2007 IEEE International Symposium on Circuits and Systems.* IEEE, IEEE, May 2007.

14. J. Großschädl. Full-Custom VLSI Design of a Unified Multiplier for Elliptic Curve Cryptography on RFID Tags. In F. Bao, M. Yung, D. Lin, and J. Jing, editors, *Information Security and Cryptology - 5th Internationa Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*, volume 6151 of *Lecture Notes in Computer Science*, pages 366–382. Springer, 2011.
15. J. Großschädl and G.-A. Kamendje. Optimized RISC Architecture for Multiple-Precision Modular Arithmetic. In D. Hutte, G. Müller, W. Stephan, and M. Ullmann, editors, *Security in Pervasive Computing - SPC 2003*, volume 2802 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.
16. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
17. D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, Canada, August 14-15, 2008, Revised Selected Papers*, Lecture Notes in Computer Science (LNCS), September 2008.
18. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In S. B. O. Yalcin, editor, *Workshop on RFID Security – RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, volume 6370 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2010.
19. IAIK. Hash Function Zoo. `http://ehash.iaik.tugraz.at/index.php/HashFunctionZoo`.
20. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, 24(6):334–335, March 1988.
21. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2003.
22. B. Kaliski. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
23. N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994. ISBN 0-387-94293-9.
24. Ç. K. Koç, T. Acar, and B. S. K. Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
25. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.
26. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
27. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over $\mathrm{GF}(2^{\mathrm{m}})$ without Precomputation. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1999.
28. P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, January 1987. ISSN 0025-5718.

29. National Institute of Standards and Technology (NIST). FIPS-180-3: Secure Hash Standard, October 2008. Available online at `http://www.itl.nist.gov/fipspubs/`.
30. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.
31. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2004.
32. K. Sakiyama, L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede. Small-footprint ALU for public-key processors for pervasive security. In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.
33. S. Tillich and J. Großschädl. VLSI Implementation of a Functional Unit to Accelerate ECC and AES on 32-bit Processors. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 40–54. Springer, June 2007.
34. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In Y. Chung and M. Yung, editors, *WISA*, volume 6513, pages 92–106. Springer, 2010.
35. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto, July 13-15, 2005, Graz, Austria*, pages 78–91, 2005.

A Statistics for ECC multiplication

During the development of the ECC and ECDSA functions we used a statistics feature of our tool-chain to investigate the code-line and cycle consumption of each function. Table 7 shows the number of times each function is called, the size of each function in code lines and the total runtime of each function. Even though the multiplication algorithm is optimized down to 222 cycles it still covers 74 % of the total runtime.

Table 7. Functions used during ECC point multiplication with y-recovery and point-validity check.

Function	Calls	Code Lines	Cycles
B163.Multiplication	990	222	219,780
B163.Square	969	41	39,729
B163.Add	490	35	17,150
PointOperation.Multiplication	1	148	16,636
B163.FermatInverseHelp	7	31	2,041
Utilities.Copy	16	24	384
PointOperation.yRecovery	1	90	90
B163.FermatInverse	1	88	88
PointOperation.isValidPoint	1	44	44
Utilities.CMP	1	35	35
Utilities.Clear	2	13	26
TOTAL	**2,479**	**771**	**296,003**
TOTAL including test functions	**2,480**	**828**	**296,547**

Table 8 shows how often each and every type of instruction is used. The parallelized commands are a combination of other commands. They cover 71 % of the total runtime. Note that only 4.4 % of the total runtime is used for program-flow instructions such as RET, CALL, BRA, and JMP. This overhead would not exist if a dedicated state machine instead of a CPU with instruction set would be used.

Table 8. Instructions used during an ECC point multiplication with y-recovery and point-validity check.

Mnemonic	Description	CPI	Cycles	Used
PAR: BMULACC | LD		1	109,869	111
PAR: MOVNF | LD		1	65,707	83
LD	Load from memory	1	35,410	65
PAR: BREDUCE_ADD_ST | BRSACC		1	13,818	14
PAR: BMULACC | ST | BRSACC		1	11,859	12
PAR: BREDUCE_ADDBYTE_ST | BRSACC		1	9,690	10
CALL	Call a function	3	7,440	70
LDI	Load Immediate	1	6,900	105
AND	Logic AND	1	6,038	7
PAR: XOR | ST		1	5,390	11
MOVNF	Copy register to register without flag update	1	5,269	47
RET	Return from function	2	4,960	13
BMULACC	Binary multiply-accumulate	1	3,876	4
STR	Store a register to memory	1	2,725	32
XOR	Logic XOR	1	2,120	3
LDR	Load from memory and store to register	2	1,942	10
ADDI	Add with carry	1	573	11
BRA	Branch if flag is set/cleared	1	488	6
PUSH	Push a value to the stack	2	338	5
POP	Pop a value from the stack	2	336	4
LSI	Left shift by immediate	1	326	4
SUBI	Subtract with carry	1	324	6
ADD	Add	1	163	2
RS	Right shift	1	163	2
RSI	Right shift immediate	1	163	2
SUB	Subtract	1	163	2
ASRI	Arithmetic shift right	1	161	1
JMP	Jump to address	1	160	1
CMP	Compare	1	84	2
MOV	Copy register to register	1	82	1
CMPC	Compare with carry	1	10	10
TOTAL			**296,547**	**656**

Chapter 10

Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations

Publication Data

Erich Wenger and Michael Hutter. Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations. In Peeter Laud, editor, *Information Security Technology for Applications*, volume 7161 of *Lecture Notes in Computer Science*, pages 256–271, 2012.

Contributions

Main author. Idea. Implementations. Figures. Tables. Algorithms. 50 % of Text.

Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations

Erich Wenger and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Erich.Wenger,Michael.Hutter}@iaik.tugraz.at

Abstract. In this paper, we answer the question whether binary extension field or prime-field based processors doing multi-precision arithmetic are better in the terms of area, speed, power, and energy. This is done by implementing and optimizing two distinct custom-made 16-bit processor designs and comparing our solutions on different abstraction levels: finite-field arithmetic, elliptic-curve operations, and on protocol level by implementing the Elliptic Curve Digital Signature Algorithm (ECDSA). On the one hand, our $\mathbb{F}_{2^m}$ based processor outperforms the $\mathbb{F}_p$ based processor by 19.7 % in area, 69.6 % in runtime, 15.9 % in power, and 74.4 % in energy when performing a point multiplication. On the other hand, our $\mathbb{F}_p$ based processor (11.6 kGE, 41.4 μW, 1,313 kCycles, and 54.3 μJ) improves the state-of-the-art in $\mathbb{F}_{p_{192}}$ ECC hardware implementations regarding area, power, and energy results. After extending the designs for ECDSA (signature generation and verification), the area and power-consumption advantages of the $\mathbb{F}_{2^m}$ based processor vanish, but it still is 1.5-2.8 times better in terms of energy and runtime.

Keywords: Hardware Implementation, Elliptic Curve Cryptography, ECC, ECDSA, Binary-Extension Field, Prime Field.

1 Introduction

Elliptic Curve Cryptography (ECC) has been introduced in the 1980s and is used nowadays in a variety of different applications. Every application has its own design criteria and raises special requirements for hardware designs. While contactless powered devices have to meet low-power constraints, battery-powered devices need energy-aware implementations that consume as little energy as possible to increase the life-time of the battery.

The most fundamental decision concerning future hardware designs is whether to use a binary-extension field or a prime field as basis of the used elliptic curve. Most related work in dedicated hardware designs has been done in implementing ECC over binary fields using full-precision arithmetic. Only a few papers compared binary and prime fields in hardware. Wolkerstorfer [36] and Satoh [32] used full-precision dual-field hardware with bit-serial multipliers. We however are interested in multi-precision designs, where the big integers are split and

P. Laud (Ed.): NordSec 2011, LNCS 7161, pp. 256–271, 2012.
The original publication is available at http://link.springer.com/chapter/10.1007/978-3-642-29615-4_18

processed in small words. This design methodology has the advantage that the Central Processing Unit (CPU) can be reused to perform other work (*e.g.* protocol handling). In this paper we want to answer the following questions:

- What are the advantages and disadvantages of prime and binary-field processors in custom multi-precision hardware?
- How big are the differences when identical design methodologies and elliptic curves with similar security level are used?
- How does the performance of prime and binary-field processors scale in higher-level protocols?
- Does the speed advantage of carry-less operations makes up the additional need of prime-field arithmetics?

In this paper, we answer these questions by presenting two distinct custom 16-bit processors that leverage binary-field operations and prime-field operations and are based on [35] and [34]. Using a metric consisting of **area**, **speed**, **power**, and **energy**, we not only compare both designs in terms of finite-field operation and ECC point-multiplication performance, we also investigate a higher-level protocol. When performing an ECC-point-multiplication, the $\mathbb{F}_{2^m}$ based processor (9.3 kGE, 34.8 μW, 400 kCycles, and 13.9 μJ) is 3.3 times faster, 20 % smaller, uses 16 % less power, and needs 3.9 times less energy compared to the $\mathbb{F}_p$ based processor (11.6 kGE, 41.4 μW, 1,313 kCycles, and 54.3 μJ). Nevertheless our $\mathbb{F}_p$ based processor improves the state-of-the-art in area, power, and energy results for prime-field based ECC (doing point multiplication).

We further present two full hardware implementations of the Elliptic Curve Digital Signature Algorithm (ECDSA). It shows that the $\mathbb{F}_{2^m}$ based processor does not outperform the $\mathbb{F}_p$ based processor in every category of the metric. The $\mathbb{F}_{2^m}$ based processor is 4.4-5.5 % larger, needs up to 6.3 % more power, but still is 2.8 times faster and needs 2.8 times less energy when calculating a signature. The runtime and energy advantage drops down to a factor of 1.5 when the verification is done.

The paper is organized as follows. Section 2 discusses related work on ECC implementations. Section 3 gives an introduction to elliptic curve cryptography and introduces a metric. Whereas Section 4 gives a comparison, Section 5 thoroughly discusses all implementation results. Conclusions are given in Section 6.

2 Related Work on ECC-Hardware Implementations

There exist many hardware implementations of elliptic-curve cryptography. In the following, we consider only lightweight implementations that address embedded systems, wireless sensors, and contactless-powered applications. Most of the given implementations are based on either binary field, prime field, or dual-field arithmetic. A very tiny ECC processor over binary fields has been proposed by Y. K. Lee et al. [25] in 2008. They based their design on a compact architecture of a Modular Arithmetic Logic Unit (MALU) that has been first presented by the work of L. Batina et al. [4] in 2006. The processor performs (full-precision)

operations in $\mathbb{F}_{2^{163}}$ and calculates a scalar multiplication between about 80 000 and 300 000 clock cycles (depending on the digit size of the hardware multiplier). The final architecture needs about 12-20 kGEs of area. Similar results have been also reported by S. Kumar and C. Paar [24] who presented a generic binary-field processor over $\mathbb{F}_{2^{113-193}}$. The run-time and area requirements of the proposed processor is similar, needing between 170 000 and 560 000 clock cycles and 10-19 kGEs. D. Hein et al. [15] reported a low-resource co-processor for passive Radio Frequency Identification (RFID) applications. In contrast to the previous work, they applied multi-precision arithmetic over $\mathbb{F}_{2^{163}}$. Their ECC design needs about 300 000 clock cycles for one scalar multiplication and consumes about 11 kGEs of chip area. In view of power consumption, all described designs need between 8 and 30 μWs of power at 100 kHz and are thus well applicable to the targeted applications.

Prime-field based processors have been reported by, for example, E. Öztürk et al. [31] in 2004. They presented an ECC architecture over the prime field $\mathbb{F}_{2^{(167+1)}/3}$. Their design needs 545 440 clock cycles for one scalar multiplication and requires about 30 kGEs of area. Similar results have also been reported by F. Fürbass and J. Wolkerstorfer [11] in 2007. Their $\mathbb{F}_{p_{192}}$ processor needs 502 000 clock cycles and about 23 kGEs of area. Recently, M. Hutter et al. [16] presented an ECC processor over the same prime field needing about 750 000 clock cycles for a scalar multiplication and about 19 kGEs of area. E. Wenger et al. [34] reduced the area requirements even further to only about 12 kGEs but their design needs about 1.4 million clock cycles. The power consumption of most of the reported prime-field processors is about 20 to several hundred μWs of power at 100 kHz.

By the given related work, it seems that binary-field processors benefit from a more efficient computation for application-specific hardware implementations. However, it is impossible to make a fair comparison since the authors used different design techniques, synthesis tools, bit/word sizes, and EC parameters. This renders a comparison largely unfeasible. Nevertheless, there exist only a few publications that reported dual-field processors for ECC that give detailed comparison results. A. Satoh and K. Takano [32] presented a processor over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ supporting 160 to 256 bits. They show that the binary-field operations can be performed about six times faster than their prime-field opponents (1.21 ms vs. 0.19 ms for a 160-bit scalar multiplication). Furthermore, the area requirements for the prime-field controller is 1.47 times larger than the binary-field controller (6 606 GEs vs. 4 490 GEs for 8-bit word size and a 160-bit scalar). J. Wolkerstorfer [36] also presented a dual-field processor that supports 190 to 256 bits. One of his outcomes has been that binary-field operations can be performed about 1.58, 1.42, and 1.27 times faster than prime-field operations for 191/192, 233/224, and 283/256 bits respectively. However, he did not compare the hardware requirements of both types of supported fields and reported only the total area requirements of his processor which is between 24-31 kGEs.

In the following, we design both a binary and prime-field based ECC processor in order to compare them in a fair environment. In contrast to existing

work, we consider not only scalar multiplication but evaluate and compare the performance also for higher-level protocols such as ECDSA. First of all, we give a brief introduction into ECC and define a metric to compare different criteria which is done in the next section.

3 Implementations of Elliptic-Curve Cryptography

Elliptic curves have been introduced by Koblitz [22] and Miller [28] in the 1980s and they have been thoroughly analyzed by the community throughout the last decades. They are based on the Weierstrass equation which can be written as

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

with $a_{i=1,2,3,4,6}, x, y \in K$. K defines the finite field. A point $P = (x, y)$ is a valid point on the elliptic curve if it fulfills the Weierstrass equation, *i.e.* Equation (1). The basic operations performed on the elliptic curve are point addition and point doubling. Using those operations, a point multiplication (often referred as scalar multiplication) $Q = k \times P$ can be calculated. The Elliptic Curve Discrete Logarithm Problem (ECDLP) states that finding k is a mathematical hard problem if the points P and Q are given. For a more detailed introduction into elliptic curves and its properties we refer the reader to [3, 5, 13, 23].

Figure 1 shows the hierarchy of ECC implementations. All ECC operations are based on finite-field arithmetics. Higher-level protocols make use of the underlying ECC operations to provide various cryptographic services such as authentication, data integrity, non-repudiation, or confidentiality. Note that most of these protocols (such as ECDSA) require different operations over finite fields such as prime-field addition or multiplication.

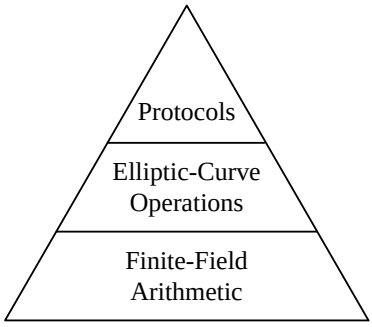

Fig. 1. Hierarchy of ECC implementations.

Among the most commonly used types of finite fields are prime fields $\mathbb{F}_p$ and binary-extension fields $\mathbb{F}_{2^m}$. These types have different characteristics so that the Weierstrass equation can be simplified and different formulas for point addition and point doubling can be derived. Due to the differences of those fields, the performance of both software and hardware implementations can vary significantly.

In this paper, we compare two ECC implementations that are based on $\mathbb{F}_{2^{191}}$ and on $\mathbb{F}_{p_{192}}$. Both implementations use multi-precision arithmetic that means that all finite-field elements are split into smaller bit vectors of size W. Note that the one-bit difference does not have an impact in a relative comparison of ECC implementations since we use the same metric for both implementations. As elliptic curves, we decided to use the recommended NIST prime-field

curve P-192 [30] and the ANSI X9.62 compliant binary-field curve B-191 [1], *i.e.* `c2tnb191v1`. This is because we would like to compare curves with nearly identical bit sizes (191 vs. 192 bits). The one-bit difference between those two fields can be considered as negligible.

Throughout the paper, we used the following notation. For prime fields with modulo p, $n = \lceil log_2(p) \rceil$ bits are required to represent a number. For binary fields with $f(z) = z^m + r(z)$ denoting an irreducible binary polynomial of degree m, a bit-vector with m entries can be used to represent any binary polynomial. Consequently the number of needed words to represent a $\mathbb{F}_p$ number is $N = \lceil n/W \rceil$ and number of needed words to represent a $\mathbb{F}_{2^m}$ polynomial is $M = \lceil m/W \rceil$.

3.1 Comparison Metric and Criteria

The efficiency of ECC-hardware implementations depends on different criteria. In order to make a fair comparison, we introduce the following metric consisting of four main attributes:

- The **area** requirement of a chip is important for any cost-sensitive application. This is because the area largely determines the chip costs at fabrication.
- Embedded systems require **low-power** and
- **low-energy** designs. This is an important issue especially in battery-powered environments.
- **Speed** of computation is important for many applications to be applicable in practice. The most neutral unit for measurement is the number of cycles it takes to perform a certain operation.

The maximum **frequency** that can be used to clock a design has a direct impact on the resulting execution time (speed) of any algorithm. But the previously mentioned applications heavily constrain the maximum frequency anyways, so we do not include the frequency measure into our metric.

Because the energy $W = Pt$ is defined as product of the electrical power P and time t, its properties are not handled explicitly within Section 4.

4 Comparing ECC-Hardware Designs over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$

In this section, we compare ECC hardware designs and the respective algorithms over $\mathbb{F}_{2^m}$ and over $\mathbb{F}_p$. We describe the differences of the finite-field operations, the respective elliptic-curve group operations, and compare the hardware designs of both types of fields regarding cryptographic protocols like ECDSA.

4.1 Finite-Field Arithmetics

Modular Addition and Subtraction. The most basic finite-field algorithms are addition and subtraction. Algorithm 1 and Algorithm 2 show modular-addition

Algorithm 1 Prime-field addition.

Require: Two integers $a, b \in [0, p-1]$ and modulus p.
Ensure: $c = (a + b) \pmod p$.
1: $(\varepsilon, C[0]) \leftarrow A[0] + B[0]$.
2: **for** i from 1 to $N - 1$ **do**
3: $\quad(\varepsilon, C[i]) \leftarrow A[i] + B[i] + \varepsilon$.
4: **end for**
5: **if** $\varepsilon = 1$ or $c \geq p$ **then**
6: $\quad(\varepsilon, C[0]) \leftarrow C[0] - P[0]$.
7: $\quad$**for** i from 1 to $N - 1$ **do**
8: $\quad\quad(\varepsilon, C[i]) \leftarrow C[i] - P[i] - \varepsilon$.
9: $\quad$**end for**
10: **end if**
11: Return(c).

Algorithm 2 Binary-field addition.

Require: Binary polynomials $a(z), b(z)$ with maximum degree m-1.
Ensure: $c(z) = a(z) + b(z)$.
1: **for** i from 0 to $M - 1$ **do**
2: $\quad C[i] \leftarrow A[i] \oplus B[i]$.
3: **end for**
4: Return(c).

algorithms over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. The major difference of those algorithms is the carry propagation ε. The polynomial addition is a simple XOR operation that does not incorporate a carry. A $\mathbb{F}_p$ addition, in contrast, needs up to three times more operations: the actual addition, a comparison of the result c with the prime p, and a modular reduction afterwards. By extending the range of the integers a, b, c form $[0, p-1]$ to $[0, 2^{N*W} - 1]$, the comparison operation $(c \geq p)$ can be avoided, which reduces the total number of arithmetic operations by about a third. Notice that for this partial reduction, all other operations handling a, b, c must be prepared for their extended range. A modular subtraction works similar as the modular addition.

Modular Multiplication. Modular multi-precision multiplications are usually realized by following an operand-scanning or product-scanning multiplication

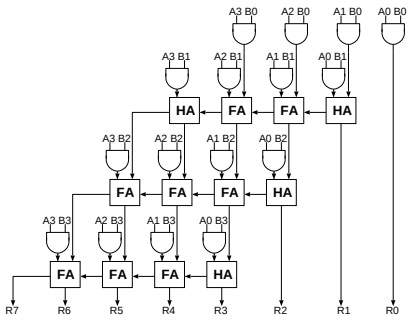

Fig. 2. 4-bit integer multiplier for $\mathbb{F}_p$.

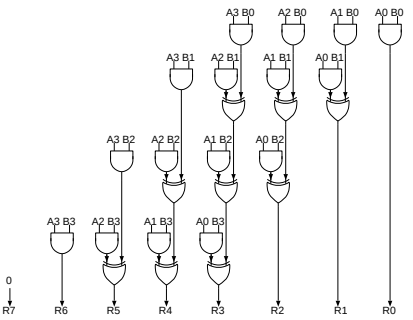

Fig. 3. 4-bit carry-less multiplier for $\mathbb{F}_{2^m}$.

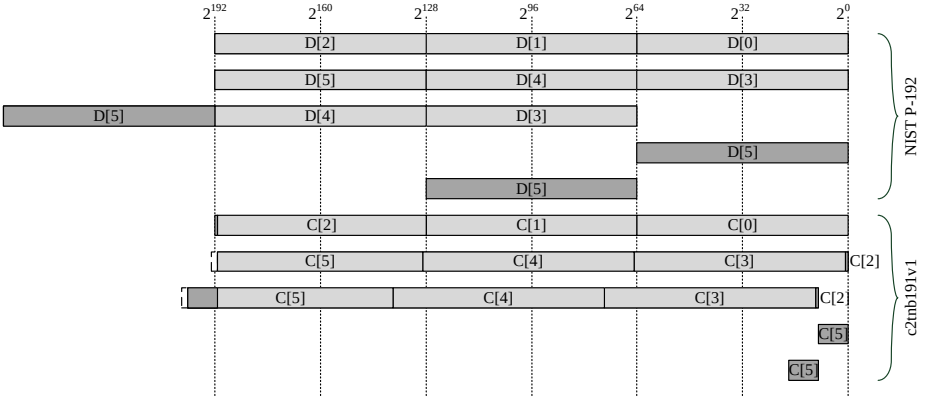

Fig. 4. Reduction using $p = 2^{192} - 2^{64} - 1$ on the top. Reduction using $f(z) = z^{191} + z^9 + 1$ on the bottom. In both cases, the product must be shifted and summed up.

approach. A multiply-accumulate unit (cf. [12, 15, 16]) can be used to increase the efficiency of the product-scanning method. Such a multiply-accumulate unit can be designed for $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. Figures 2 and 3 show the internal structure of 4-bit multipliers for integers and polynomials. The biggest advantage of the carry-less multiplier for $\mathbb{F}_{2^m}$ are the shorter critical path and the smaller area requirement (logical XOR cells are used instead of full-adder standard cells). Thus, the difference between a $\mathbb{F}_{2^m}$ and a $\mathbb{F}_p$ multiplication module can be up to 40 % in terms of area requirement. Also the power consumption for a multiplier designed out of XORs instead of full-adders is lower. However the execution times (in cycles) for an integer or binary-polynomial multiplication using the product-scanning method are equivalent.

A finite-field multiplication always needs a reduction. There exist many ways to realize modular reduction in hardware. One efficient way is to apply a (fast) reduction method using special primes, so called Mersenne-like primes, which are often used for recommended and standardized elliptic curves (e.g. the NIST recommended curves [30]). Figure 4 shows how intermediate multiplication results can be reduced using this fast reduction method for primes and polynomials over the curves NIST P-192: $p = 2^{192} - 2^{64} - 1$ and ANSI X9.62 `c2tnb191v1`: $f(z) = z^{191} + z^9 + 1$. The reduction can be performed with only shifts and additions. The for NIST P-192 necessary shift operations fit very well within the addressing scheme of 8-bit, 16-bit, or 32-bit architectures. The shift operations required by `c2tnb191v1` do not fulfill this property. However, in cases where the shift operations are smaller than W, an additional hardcoded reduction logic can be used. In terms of area, this reduction logic is very cheap (about the size of a $\mathbb{F}_{2^m}$ addition).

Modular Squaring. Modular squaring is equivalent to a modular multiplication with two identical operands. Thus, an explicit implementation is often not necessary, especially in implementations where low area is a stringent require-

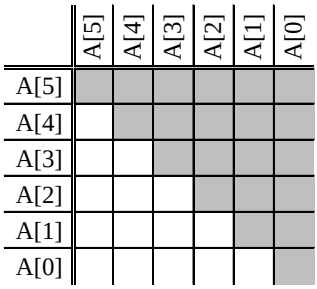

Fig. 5. Prime-field squaring operation. The necessary intermediate multiplications are shaded.

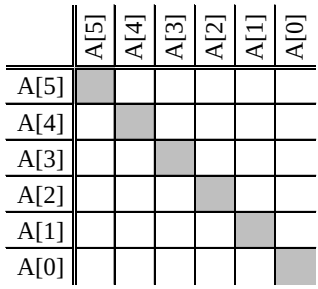

Fig. 6. Binary-field squaring operation. The necessary intermediate multiplications are shaded.

ment. However, if implemented it improves the performance since it is typically faster than modular multiplications [13].

During a prime-field squaring operation, the two intermediate products $A[i] \times A[j]$ and $A[j] \times A[i]$, $\forall i \neq j \in [0, N-1]$, are identical. Figure 5 shows the operands of a 6-word squaring operation where only the necessary operations (multiplications) are shaded. Thus, the squaring operation can be up to two times faster than a multiplication.

Squarings over binary fields, as opposed, have the nice property that $a_i \times a_j + a_j \times a_i = 0$, $\forall i \neq j \in [0, m-1]$. If $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$, then $a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0$. Thus, zero values are simply inserted between two consecutive bits a_i. Utilizing the binary multiplier from Figure 3, only M multiplications $A[i] \times A[i]$ are required to perform a binary-field squaring operation. As it can also be seen in Figure 6, the squaring operation is M times faster than a binary field multiplication. It can be performed with a similar runtime complexity as a modular addition.

In terms of runtime and lines-of-code, a $\mathbb{F}_{2^m}$ squaring can be up to $\frac{N^2}{2M}$ times faster than a $\mathbb{F}_p$ squaring.

Modular Inversion. What the inversion operations for prime and binary fields have in common is the very slow execution time. There are two common inversion methods. One is based on the extended Euclidean algorithm and one is based on Fermat's little theorem ($a = a^{2^m} \bmod f(z)\ \forall a \in \mathbb{F}_{2^m}$). For this paper the Montgomery inversion technique by Kalinski *et al.* [20] has been used for prime field inversion operations. Using Fermat's little theorem [18] for binary field inversions, with $a^{-1} \equiv a^{2^m-2} \bmod f(z)$, a field inversion can be performed by using $m-1$ squarings and several multiplications. In the case of `c2tnb191v1`, 190 squarings and 12 multiplications are necessary. Because of the fast squaring operations within binary fields, the runtime of this method exceeds any Euclidean-based algorithm. [14] gives a comparison of different algorithms for an inversion within the NIST B-163 field.

4.2 Elliptic-Curve Operations

The performance of EC-group operations over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ differ significantly. We used formulae that reflect the state of the art in efficient ECC implementations. For binary-field arithmetic, we applied the formulae proposed by J. López and R. Dahab [27]. Their formulae need six finite-field multiplications, five squarings, and three additions per key bit. For prime-field arithmetic, we applied the formulae of M. Hutter et al. [17] needing 12 multiplications, four squarings, and 16 additions (incl. subtractions). Both formulae have been applied within the Montgomery powering ladder scalar multiplication [19]. By comparing the formulae, it clearly shows that the binary formulae need 50 % less multiplications than the formulae over prime-field arithmetic. This is one of the most advantageous properties that encourages the use of $\mathbb{F}_{2^m}$ operations in ECC-hardware implementations. Note that both formulae use projective coordinates that means that no modular inversion is needed throughout the scalar multiplication[1].

4.3 Cryptographic Protocols

After the basic elliptic-curve operation of a scalar multiplication, we compare the performance of $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ processors in terms of higher-level protocols. In particular, we implemented ECDSA [30] on both types of (binary and prime-field based) processors. The main additional operations needed to support ECDSA is the SHA-1 [29] algorithm[2] to calculate the message digest of the message m and some prime-field operations, *i.e.* modular addition, multiplication, and inversion to calculate the digital signature $(r, s) = (k \times P, k^{-1}(\text{SHA-1}(m) + rd))$, where d represents the used private key.

For a more efficient ECDSA-verify algorithm, we additionally implemented a different methodology for calculating point multiplications. First, we applied Shamir's trick [8, 9] to improve the performance of multiple point multiplication. Second, we used different formulae to perform the verification using Jacobian-projective coordinates [13] for the prime-field processor and López-Dahab coordinates [13, 26] for the binary-field processor.

5 Comparison Results

For a fair comparison of binary field and prime-field ECC implementations, it is important to select a common controlling engine, common development tools, the same process technology, and elliptic curves of nearly the same bit size.

As a controller, we decided to use our own 16-bit microcontroller called Neptun [33–35] that is especially optimized for elliptic-curve cryptography. The processor comes with twelve special-purpose registers and uses a Harvard architecture with separated program and data memory. The usually area consuming

[1] Inversion is only needed after the calculation of $k \times P$ to convert the projective coordinates back to affine coordinates.

[2] Included in the design. 16-bit CPU is used to calculate the hash.

Table 1. Prime-field vs. binary-field operations of our ECC-hardware architecture.

	Cycles		Lines of Code		Operations/key-bit	
	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$ [17]	$\mathbb{F}_{2^{191}}$ [27]
Addition/Subtraction	64	38	64	38	16	3
Multiplication	329	265	329	265	12	6
Squaring	190	45	190	45	4	5
Inversion	46,560	14,611	397	117	-	-

data memory is made from a very area-efficient single-port RAM macro[3]. The program memory is a synthesized lookup table stored as Read-Only Memory (ROM). In fact, the area requirements of this lookup table is proportional to the number of lines-of-code (LOC) stored within the program memory. The central processing unit (CPU) is capable of the most basic arithmetic operations such as addition/subtraction, logic operations (AND, OR, XOR), and shift operations.

As target technology, we selected a 130nm low-leakage CMOS technology by UMC. This technology needs fewer power compared to larger 180 nm and 350 nm technologies and has a lower power leakage than smaller (*e.g.* 90 nm) technologies. The standard-cell library has been provided by Faraday Technology. The RAM-macro blocks have been generated using the Standard Memory Compiler FSA0A Memaker 200901.1.1 by the Faraday Technology Corporation [10]. For synthesis we used the Cadence RTL compiler [7] Version v08.10. For power-simulations we used Cadence First Encounter Version v08.10.

5.1 Finite-Field Arithmetic

The finite-field algorithms have been implemented as described in Section 4.1. All algorithms (except the algorithms for modular inverses) have been unrolled and optimized for our custom microcontroller instruction set (Assembler language). All results are summarized in Table 1.

It shows that our processor performs the binary-field addition about 40.6 % faster than the prime-field addition (the same holds for modular subtraction). Binary-field multiplication is 19.5 % faster than its prime-field counterpart because of the extra reduction logic provided to take advantage of the Mersenne-like irreducible polynomial. However, it shows that even when multi-precision arithmetic is used, the biggest advantage of binary-field operations is within the squaring operation. Its runtime is 4.2 times faster than the prime-field squaring operation. Finally, the two very distinct inversion techniques, discussed in Section 4.1, result in very different runtime and LOC results. The binary-field inversion implementation is 3.19 times faster and needs only 29.5 % LOC. It reuses the squaring and multiplication methods and subsequently only works for a single irreducible polynomial. The prime-field inversion, in contrast, works for any prime. The main reason for the higher code size are actually the additional

[3] All following area-related results would be different if latch or register-based RAM's are used.

Table 2. Comparison of prime field vs. binary-field ECC implementations.

Algorithm	**Q = k × P**		**ECDSA Sign**		**ECDSA Verify**	
	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{191}}$
Integer multiplier	required	–	required	required	required	required
Carry-less multiplier	–	required	–	required	–	required
Cycles	**1,312,616**	**399,635**	**1,393,523**	**494,983**	**1,417,422**	**892,124**
RAM entries	100	90	112	103	174	162
Program entries	1,207	699	1,662	1,689	1,519	1,875
Constants	61	60	100	129	100	141
Area requirements [GE]						
CPU	4,041	3,653	4,049	4,393	4,066	4,422
Program memory	4,494	2,683	7,203	7,432	7,589	8,031
Data memory	3,040	2,963	3,390	3,412	4,088	4,160
Total area	**11,579**	**9,301**	**14,644**	**15,293**	**15,747**	**16,618**
Power consumption @ 1 MHz [μW]						
CPU	20.40	19.99	18.36	18.48	17.93	20.38
Program memory	8.95	4.01	7.89	8.22	8.89	8.16
Data memory	10.59	8.92	11.91	10.86	11.80	12.52
Total power	**41.37**	**34.78**	**39.54**	**39.47**	**40.55**	**43.12**
Energy consumption [μJ]						
Energy	**54.30**	**13.90**	**55.10**	**19.53**	**57.48**	**38.47**

utility functions (addition, subtraction, multiplication with 2, division by 2) that had to be implemented.

5.2 Elliptic-Curve Operations

Table 2 compares the absolute values and Table 3 compares the relative differences of the implemented prime-field and binary-field ECC implementations. The relative differences shown in Table 3 have been calculated using the Formula $\frac{Param(\mathbb{F}_{2^m})}{Param(\mathbb{F}_p)} - 1$. In the following, we separately consider point multiplication as well as signature generation and verification of the higher-level protocol of ECDSA.

In view of point multiplication, it shows that the binary-field based implementation is 3.28 times faster than the prime-field based opponent. The area requirement is 19.7 % better and the power consumption is 15.9 % lower for the binary-field processor. This results in an energy consumption which is 3.91 times

Table 3. Relative difference between $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ based implementations.

Algorithm	**Q = k × P**	**ECDSA Sign**	**ECDSA Verify**
Runtime	−69.6 %	−64.5 %	−37.1 %
Area	−19.7 %	+4.4 %	+5.5 %
Power	−15.9 %	±0.0 %	+6.3 %
Energy	−74.4 %	−64.6 %	−33.1 %

Table 4. Comparison of different ECC implementation with related work.

	ECC Curve	Area [GE]	Cycles [kCycles]	Power[a] μW	Energy μJ	VLSI technology
Auer 2009 [2]	$\mathbb{F}_{p_{192}}$	24,750	1,031	613.65	632.67	AMS C35
Fürbass 2007 [11]	$\mathbb{F}_{p_{192}}$	23,656	500	1,692.11	846.06	AMS C35
Wolkerstorfer 2005 [36]	$\mathbb{F}_{p_{192}}$	23,818	678	500.00	340.00	350nm
This work 2011	$\mathbb{F}_{p_{192}}$	**11,579**	**1,313**	**41.37**	**54.30**	**UMC L130**
Lee 2008 d=4 [25]	$\mathbb{F}_{2^{163}}$	15,356	79	37.39	2.95	UMC L130
Lee 2008 d=1 [25]	$\mathbb{F}_{2^{163}}$	12,506	276	32.42	8.95	UMC L130
Batina[b] 2006 d=4 [4]	$\mathbb{F}_{2^{163}}$	14,816	95	27.00	2.57	130nm
Batina[b] 2006 d=1 [4]	$\mathbb{F}_{2^{163}}$	13,104	354	27.00	9.56	130nm
Bock 2008 d=8 [6]	$\mathbb{F}_{2^{163}}$	16,247	47	148.76	6.99	INF SRF55V01P
Bock 2008 d=1 [6]	$\mathbb{F}_{2^{163}}$	10,392	280	54.31	15.21	INF SRF55V01P
Hein 2008 [15]	$\mathbb{F}_{2^{163}}$	11,904	296	101.87	30.15	UMC L180
Kumar 2006 [24]	$\mathbb{F}_{2^{163}}$	15,094	430	-	-	AMI C35
Kumar 2006 [24]	$\mathbb{F}_{2^{193}}$	17,723	565	-	-	AMI C35
Wolkerstorfer 2005 [36]	$\mathbb{F}_{2^{191}}$	23,818	426	500.00	213.00	350nm
This work 2011	$\mathbb{F}_{2^{191}}$	**9,301**	**399**	**34.78**	**13.90**	**UMC L130**

[a] All reference values were scaled to 1 MHz.
[b] RAM approximated with 4,890 GE. Power-consumption values do not include RAM.

lower than the calculation over prime fields. Note that the area difference mostly comes from the size of the program memory, the used multiplier within the CPU and the size of the necessary RAM macro. Even note that in both designs, about 50 % of the total power is consumed within the CPU.

5.3 Cryptographic Protocols

For ECDSA, only 455 lines of code (38 %) have to be added to the prime-field ECC processor to support all operations to sign data. This and the small increase of necessary RAM entries increased the total area requirement by 26.5 %. The execution time is increased by only 6.2 %. The differences in power and energy consumption are hardly noticeable. The changes to the binary-field ECC processor are much more significant. The CPU had to be extended with a small 8-bit integer multiply-accumulate unit, making it capable of prime and binary-field operations, increasing the area requirements of the CPU by 20 %. Adding all those algorithms increased the size of the program memory by 177 % and the total area of the processor by 64 %. Also the power and energy consumption increased by 13.5 % and 40.5 %. However, the runtime of the binary-field based ECDSA processor is still 2.82 times faster than the runtime of the prime-field based ECDSA processor. Even though the area and power consumption are approximately identical, the binary-field ECDSA processor needs 2.82 times less energy than the prime-field ECDSA processor.

The ECDSA verification needs one additional point multiplication compared to the ECDSA-signature generation algorithm which needs only one. Cause of

Table 5. Comparison of our ECDSA implementations with related work.

	ECC Curve	Area [GE]	Cycles [kCycles]	Power[a] μW	Energy μJ	VLSI technology
Kern 2010 [21]	$\mathbb{F}_{p_{160}}$	18,247	512	860.00	440.32	AMS C35
Hutter 2010 [16]	$\mathbb{F}_{p_{192}}$	19,115	859	1,507.79	1,295.19	AMS C35
Wenger[b] 2010 [34]	$\mathbb{F}_{p_{192}}$	11,686	1,377	113.86	156.79	UMC L180
This work[b] 2011	$\mathbb{F}_{p_{192}}$	**14,644**	**1,394**	**39.54**	**55.10**	**UMC L130**
This work 2011	$\mathbb{F}_{2^{191}}$	**15,293**	**495**	**39.47**	**19.53**	**UMC L130**

[a] All reference values have been scaled to 1 MHz.
[b] Nearly identical designs were used. The differences in area and power come from the different technologies and synthesizers used.

Shamir's trick the runtime for the prime-field based algorithms differ by only 2 %. The area differs by 7.5 % and the power and energy results are almost identical. The ECDSA-signature verification algorithm over binary fields does not handle the two point multiplications as well. Whereas the area increased by only 8.7 %, the runtime increased by 80 %. This doubles the required energy needed for an ECDSA-signature verification compared to an ECDSA-signature generation.

5.4 Comparison with Related Work

Table 4 gives a comparison with related work. All power results have been scaled to 1 MHz. The first five rows give related work over prime fields. The remaining rows contain related work over binary fields. Our $\mathbb{F}_p$ processor is 51 % smaller than the best related design by Wolkerstorfer [36]. In terms of cycles this processor is above average. Only the energy requirement by Öztürk [31] design is lower, but their design is not based on NIST P-192. The area results of the math processor are 10.4 % smaller than the smallest related implementation. Our speed, power, and energy results are larger than many other designs, but it should be noted that those designs have an advantage cause of the smaller elliptic curve used.

Table 5 summarizes related work regarding low-resource ECDSA-hardware implementations. In terms of power and energy consumption, we outperform existing solutions. The area requirements are lower than the work of Kern [21] and Hutter [16] but are higher than the work of Wenger [34].

6 Conclusion

In this paper, we compared the performance of two distinct ECC-hardware implementations that are based on prime-field (NIST P-192) and binary-field (ANSI `c2tnb191v1`) arithmetic. The comparison of the finite-field algorithms showed us the clear runtime advantage of the squaring (4.2 times) and addition (1.7 times) operations within the binary-extension field. When doing point multiplications, the $\mathbb{F}_{2^m}$ based processor outperforms the $\mathbb{F}_p$ based processor by

19.7 % in area, 69.6 % in runtime, 15.9 % in power, and 74.4 % in energy. In addition to these outcomes, we analyzed the impact of higher-level protocols on the finite-field processors. The implementation of both digital-signature generation and verification using ECDSA had led us to interesting findings. It was shown that the area and power advantages for the $\mathbb{F}_{2^m}$ based processor vanish while it still is 1.5-2.8 times faster and consequently more energy efficient than the $\mathbb{F}_p$ based processor.

These results can be applied to any future design of an ASIC ECC processor that is integrated in an area, power, or energy constrained device.

Acknowledgements. The work has been supported by the European Commission through the ICT program under contract ICT-SEC-2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES) and by Austrian Science Fund (FWF) under grant number P22241-N23.

References

1. American National Standards Institute (ANSI). AMERICAN NATIONAL STANDARD X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.
2. A. Auer. Scaling Hardware for Electronic Signatures to a Minimum. Master thesis, University of Technology Graz, October 2008.
3. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, 2005.
4. L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks – ESAS 2006, Third European Workshop, Hamburg, Germany, September 20-21, 2006, Revised Selected Papers*, volume 4357, pages 6–17, Berlin Heidelberg, 2006. Springer-Verlag.
5. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Notes Series.* Cambridge University Press, Cambridge, UK, 1999.
6. H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008, July 2008.
7. Cadence Design Systems, Inc. The Cadence Design Systems Website. `http://www.cadence.com/`, 2011. San Jose, California, United States.
8. P. de Rooij. Efficient Exponentiation using Procomputation and Vector Addition Chains. In A. D. Santis, editor, *Advances in Cryptology EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399. Springer Berlin / Heidelberg, 1994.
9. T. Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

10. Faraday Technology Corporation. Faraday FSA0A_C 0.13 μm ASIC Standard Cell Library, 2004. Details available online at `http://www.faraday-tech.com`.
11. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of 2007 IEEE International Symposium on Circuits and Systems*. IEEE, IEEE, May 2007.
12. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 133–147. Springer, August 2004.
13. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
14. D. Hein. Elliptic Curve Cryptography ASIC for Radio Frequency Authentication. Master thesis, Technical University of Graz, April 2008.
15. D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, Canada, August 14-15, 2008, Revised Selected Papers*, Lecture Notes in Computer Science (LNCS), September 2008.
16. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In S. B. O. Yalcin, editor, *Workshop on RFID Security – RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, volume 6370 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2010.
17. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 Fourth International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2011.
18. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, 24(6):334–335, March 1988.
19. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2003.
20. B. Kaliski. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
21. T. Kern and M. Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *17th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2010), December 12-15th, 2010, Athens, Greece, Proceedings*, pages 1236–1239. IEEE, 2010.
22. N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
23. N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994. ISBN 0-387-94293-9.
24. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.

25. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
26. J. López and R. Dahab. Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). In *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 1998.
27. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^m) without Precomputation. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1999.
28. V. S. Miller. Use of Elliptic Curves in Cryptography. In H. C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1986.
29. National Institute of Standards and Technology (NIST). FIPS-180-3: Secure Hash Standard, October 2008. Available online at `http://www.itl.nist.gov/fipspubs/`.
30. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.
31. E. Öztürk, B. Sunar, and E. Savaş. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2004.
32. A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Transactions on Computers*, 52(4):449–460, 2003.
33. E. Wenger, M. Feldhofer, and N. Felber. A 16-Bit Microprocessor Chip for Cryptographic Operations on Low-Resource Devices. In *Proceedings of Austrochip 2010, October 6, 2010, Villach, Austria*, pages 55–60, October 2010. ISBN 978-3-200-01945-4.
34. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In Y. Chung and M. Yung, editors, *WISA*, volume 6513, pages 92–106. Springer, 2010.
35. E. Wenger and M. Hutter. A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9kGEs. In *Proceedings of the Tenth Smart Card Research and Advanced Application Conference, CARDIS 2011, September 15-16, 2011, Leuven, Belgium, Proceedings*, 2011.
36. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto, July 13-15, 2005, Graz, Austria*, pages 78–91, 2005.

Chapter 11

Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware

Publication Data

Erich Wenger, Thomas Korak, and Mario Kirschbaum. Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware. Michael Hutter and Jörn-Marc Schmidt, editors, *RFIDSec*, volume 8262 of *Lecture Notes in Computer Science*, Springer, 2013. Note: in press.

Contributions

Hardware implementations. VCD simulations. Idea to analyze the leakage of the digit-serial multiplier for SPA attacks. Text: major parts of abstract and sections 1, 3, 6, 8, 9.

Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware

Erich Wenger, Thomas Korak, and Mario Kirschbaum

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
{Erich.Wenger,Thomas.Korak,Mario.Kirschbaum}@iaik.tugraz.at

Abstract. Using RFID tags for security critical applications requires the integration of cryptographic primitives, e.g., Elliptic Curve Cryptography (ECC). It is specially important to consider that RFID tags are easily accessible to perform practical side-channel attacks due to their fields of applications. In this paper, we investigate a practical attack scenario on a randomized ECC hardware implementation suitable for RFID tags. This implementation uses a Montgomery Ladder, Randomized Projective Coordinates (RPC), and a digit-serial hardware multiplier. By using different analysis techniques, we are able to recover the secret scalar while using only a single power trace. One attack correlates two consecutive Montgomery ladder rounds, while another attack directly recovers intermediate operands processed within the digit-serial multiplier. All attacks are verified using a simulated ASIC model and an FPGA implementation.

Keywords: Implementation Attack, Correlation Power Analysis, Simple Power Analysis, Digit-Serial Multiplier, Elliptic Curve Cryptography.

1 Introduction

When it comes to RFID security research, many research groups all around the world investigate the viability of new protocols, new optimized algorithms, and new hardware implementation for RFID tags. Those designs have to cope with the restrictive area, power, and runtime challenges that are mandatory for practically usable RFID tags. Additionally, also power-analysis attacks have to be considered. Those attacks can be used to recover keys, even though the actual protocol or algorithm is mathematically secure.

The most promising public-key protocols are based on Elliptic Curve Cryptography (ECC) as ECC offers comparably small memory and practically useable runtime properties. RSA and ElGamal based public key schemes simply need too much memory or only provide inferior runtimes. Some of the most notable ECC implementations are [5, 12, 22, 23, 33, 39]. Unfortunately, there have been too few practical evaluations of those state-of-the-art hardware implementations. Especially the design of Lee et al [23] raised our interest. They use a digit-serial

M. Hutter and J-M. Schmidt (Eds.): RFIDSec 2013, LNCS 8262, in press, 2013.

multiplier with a López and Dahab-like [24] Montgomery Ladder. Further we assume that the design-under-attack performs in constant key-independent runtime, utilizes Randomized Projective Coordinates [9] (RPC), and use the private scalar only once (as it is done for ECDSA signatures and Diffie-Hellman key exchanges). Therefore we have very strong assumptions regarding the actual implementation under investigation and many of the related power analysis techniques [7, 10, 17, 20, 21, 26] simply cannot be mounted.

Our contribution. In this paper, we successfully perform simple and correlation-based power analysis attacks on a protected ECC hardware implementation using only a single power trace. The investigated hardware design utilizes a López and Dahab Montgomery ladder, RPC, ephemeral secret keys, a digit-serial binary field multiplier, and performs in constant runtime. We show the practicability of our attacks using simulated and FPGA-measured power traces. The correlation based attack shows how the hamming distance of consecutive key bits can be used to recover the secret scalar. Additionally, we thoroughly analyze the power consumption of bit-serial and digit-serial binary field multiplier. We are able to recover one of the processed operands and discuss how to utilize this information to perform more advanced attacks.

The paper is structured as follows: Section 2 discusses related work. Sections 3–5 elaborate the design under attack, some attack-related prerequisites, and the basic setup for conducting the attacks, respectively. In Section 6 we assure that there is no key-dependent leakage. Section 7 discusses the correlation of consecutive rounds and Section 8 discusses the recovery of intermediate values. Section 9 concludes the paper.

2 Related Work

There exist many papers that describe hardware implementations of ECC. Most of them make use of digit-serial multipliers over $GF(2^m)$, see for example [1, 4, 5, 11, 14, 23, 30]. The reason for that choice lies in several facts. First, binary-field multipliers significantly improve the performance of ECC since they avoid carry-propagation as opposed to prime-field based implementations. Second, since they are based on binary polynomials, they can be efficiently implemented in hardware which makes them especially attractive for embedded systems and low-area designs. Therefore, they are commonly used and applied in real-world applications such as contactless smart cards, RFIDs, or Java Cards [1, 6, 29].

In view of side-channel attacks, there exist many papers that discuss attacks and countermeasures on ECC, for example presented in [8, 9, 13, 18, 27, 31, 32, 36, 37]. Most of the work exploits the weakness of different scalar-multiplication algorithms such as double-and-add which allows to perform SPA attacks in order to distinguish between a single *double* or *add* operation.

Most notable is the work of Walter [36], who attacked RSA using only a single trace. In a sliding-window RSA multiplication, he correlates the preprocessing step with the per-bit multiplication. He notes that such an attack can even work using a single power trace. The attack on the RSA modular exponen-

tiation by Wittemann *et al.* [38] can be performed even in the presence of the message blinding and the multiply always countermeasures. They take advantage of the fact that multiplications followed by squarings share operands in a key-dependent manner. In order to extract the bits of the secret exponent one power measurement recorded during the exponentiation is sufficient. In 2010, Clavier *et al.* [8] introduced the terms vertical and horizontal power analysis. While a vertical power analysis attacks the same time sample on many curves, a horizontal power analysis correlates parts of a single power trace. They performed a horizontal correlation analysis on RSA and similar to Amiel *et al.* [3] distinguished multiplication from squaring operations. Also in 2010, Homma *et al.* [18] generated collisions between squaring operations by recording only two power traces.

While those attacks work on the group structure of RSA and ECC, there exist only a few papers that demonstrate the susceptibility of underlying finite-field arithmetics. In 2006, Akishita et al. [2] demonstrated an attack on ECC by measuring the difference of modular multiplication and squaring. Since they performed attacks targeting ECC-field operations instead of ECC-group operations, their attack is applicable to countermeasures such as unified-addition formulae or SPA-resistant algorithms like the Montgomery-powering ladder. Recently, Pan et al. [34] presented a correlation power-analysis (CPA) attack on a digit-serial multiplier. They targeted the output register of the multiplier and successfully extracted intermediate values from an FPGA implementation. However, since they applied a CPA attack using 1,000 traces, their attack cannot be applied on ECC implementations that use random scalars.

In the following, we present a power-analysis attack that extracts the secret scalar from an ECC implementation consisting of a Montgomery ladder and RPC using only one single power trace. The attack can therefore be even applied to reveal ephemeral keys such as used in the Elliptic Curve Digital Signature Algorithm (ECDSA) or in Elliptic Curve Diffie Hellman (ECDH) protocols.

3 Design Under Attack

In the following, the ECC implementation used for the conducted experiments is presented. The objective of the implementation is to provide resistance against side-channel attacks as well as being flexible in size and runtime. The SCA resistance is achieved by using a constant-runtime Montgomery ladder with randomized projective coordinates and the flexibility in size and runtime is achieved by using a digit-serial multiplier.

3.1 Chosen Top-Level Algorithms

Under the consideration and investigation of related work on timing, power-analysis, and fault attacks applicable on elliptic curve cryptography, we decided to use a left-to-right Montgomery ladder for EC point multiplications $Q \leftarrow k \times P$. The key-independent structure of the Montgomery ladder provides a good

foundation against many side-channel attacks. In order to be independent of the most significant bits of the scalar k, the order of the elliptic curve is added to k. Additionally by randomizing the projective coordinates of the base-point, most attacks become infeasible. So a combination of a constant-runtime Montgomery ladder with randomized projective coordinates provides strong resistance against most side-channel attacks[1].

3.2 Hardware Design

Similar to Lee *et al.* [23], our hardware is specially optimized for binary extension fields using the NIST [28] standardized elliptic curve B-163. The hardware design comes with a two-port 163-bit memory, a 163-bit adder, and an MSB-first digit-serial multiplier. In order to save chip area, no dedicated hardware squaring unit is used. For point multiplication we use the formulas by López and Dahab [24] (see Appendix A).

3.3 Digit-Serial Multiplier

Most critical for the size of the hardware design, the runtime of the design and the following attacks is the multiplier used in the design. In the following we give a brief introduction to digit-serial multipliers, which are used in our design.

The term "digit serial" indicates that only a limited number of digits d of operand OpB are processed in each clock cycle. In the special case of $d = 1$, the multiplication approach is also known as bit-serial multiplication approach. A useful property of the digit-serial multiplication approach is that it can be sped up by increasing the digit size d. Thus, the designer can easily change the design to meet the desired area and runtime constraints.

Digit-serial multipliers are used to calculate the product $C \leftarrow OpA \times OpB$, where N bits are required to represent OpA, OpB and C. In the remainder of this paper, we use the notation OpB_i to index a single digit with index i, with $0 \leq i < \lceil N/d \rceil$ and $\lceil N/d \rceil - 1$ indexing the most significant digit. Figure 1 shows two approaches for bit-serial multiplications ($d = 1$) using a fixed reduction polynomial (cf. [15]). Either the most significant bit (MSB) is processed first (shown on the left) or the least significant bit (LSB) is processed first (shown on the right). Since for the LSB-first multiplier, two registers (a_i and c_i) are modified in each cycle, we concentrate on the MSB-first multiplier. The single active (working) register C (respectively c_i) is shifted by d digits to the left and is implicitly reduced using a fixed reduction polynomial. Additionally, OpA (resp. a_i) is multiplied with OpB_i using a simple AND gate. The product of the multiplication is then added to the (by d bits) shifted and reduced work register. After $\lceil N/d \rceil$ cycles, the product ($C \leftarrow OpA \times OpB$) of the finite-field multiplication is stored in register C.

[1] Note that we are aware of fault attacks, but those type of attacks are not subject of this paper.

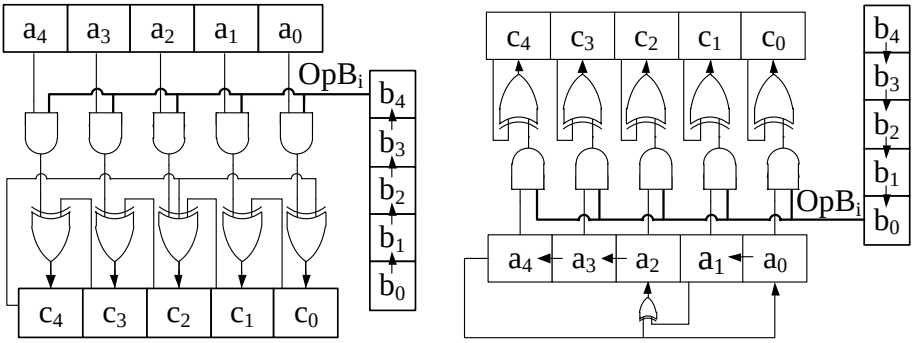

Fig. 1. Finite-field multiplier with fixed reduction polynomial $f(z) = z^5 + z^2 + 1$.

This multiplication approach can be applied to binary-extension fields as well as prime fields. For prime fields, the XOR-gates have to be replaced with full adders. Thus digit-serial multipliers can be used for RSA as well as Elliptic Curve Cryptography.

4 Attack Prerequisites

In order to perform the attacks presented in the following sections, the design has to fulfill some prerequisites which are discussed here.

4.1 Constant Runtime Scalar Multiplication

After recording one power trace while the device under attack performs the scalar multiplication a post-processing step is necessary. The trace T is split in $|k|$ traces after removing the initialization (t_{init}) and final y-recovery phases. Here we take advantage of the fact that all round transformations have an equal runtime. For simplification purposes we assume that $|k| = N$. Sub-traces $R_i = T(t_{init}+t_{round}(N-1-i) \dots t_{init}+t_{round}(N-i))$ represent a López-Dahab double-and-add round operation of the secret scalar k_i, with $i = [0, N-1]$. According to our notation k_{N-1} represents the MSB and k_0 the LSB of k. One method to identify the length of one round (t_{round}) is to perform a cross correlation on the trace T. The result of the cross correlation shows significant, equidistant peaks. The distance corresponds to t_{round}. In the following $R_i(o, o+w)$ denotes a part of the trace of round i with an offset o from the beginning and a length of w samples.

Figure 2 shows a comparison of the simulated and measured power consumption with $d = 1$. For the figures the traces $R_0 \dots R_{N-1}$ are plotted overlayed. Apparently both traces show identical characteristics. The left half of the traces (cycles 995-1150) shows a multiplication of pseudo-random OpA with pseudo-random OpB. In the right half (cycles 1150-1320) OpB has been kept unchanged during the N round transformations. In fact a multiplication with the constant

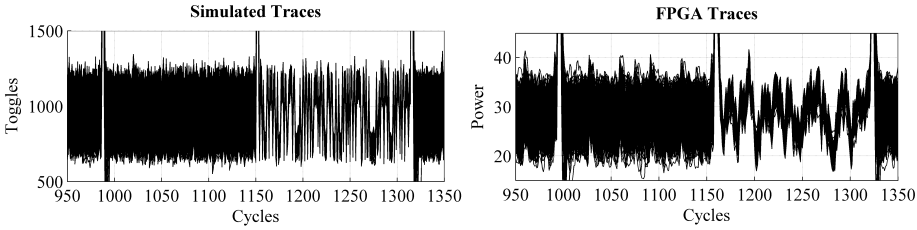

Fig. 2. Comparison of simulated traces (on the left) and traces recorded on the FPGA (on the right).

$c = b^{2^{m-1}} \mod f(z)$ which is used within the López-Dahab formula [24] is performed in this interval. This unchanged operand leads to a similar power consumption in this interval for all round transformations.

4.2 Leakage of the Digit-Serial Multiplier

With the field multiplication being the most time-consuming part of an EC point multiplication and the digit-serial multiplier being the most active part of the circuit, the side-channel vulnerability is highly dependent on the power-consumption of a digit-serial multiplier.

The main source of leakage within the digit-serial multiplier lies within the multiplication of the full word OpA with a single digit OpB_i. In a first model (which was wrong) we theorized that there will be a higher power consumption when $OpB_i = 1$ (with $d = 1$) and a lower power consumption with $OpB_i = 0$. Such a power model may be true for a logic style such as Transistor-Transistor-Logic (TTL), but for a CMOS process our power model had to be refined. In CMOS, the power consumption does not depend on the current state of a signal, but on the transition from the previous state to the next state. So the power consumption is related to the Hamming distance of two consecutive values of OpB_i. Let $w(\cdot)$ denote the Hamming weight and $hd(\cdot, \cdot)$ denote the Hamming distance. Experiments showed that the higher the Hamming distance $hd(OpB_i, OpB_{i+1})$, the higher is the power consumption of the circuit. Let $hd(OpB)$ be a vector with all $hd(OpB_i)$ and $hd(OpB_i)$ be a short form for $hd(OpB_i, OpB_{i+1})$.

Before we discuss the impact of the digit-serial multiplier on the side-channel leakage in Section 8 in detail, we must elaborate our basic side-channel analysis approach and related assumptions.

5 Setup for Conducting the Attacks

Two approaches to record the required power trace for the performed attacks are used. On the one hand the power traces are generated using a simulation of the implementation. On the other hand the design was implemented on an FPGA and the power consumption was measured using an oscilloscope. The achieved results were compared and prove the correctness of the attack assumptions.

5.1 Simulator Toolchain

A simulator toolchain was used in order to enable an attack on the ECC implementation in a noise-free environment. This toolchain consists of four elements: Cadence RTL Compiler v08.10 was used to synthesize the VHDL code to UMC L-130 logic gates; Cadence First Encounter v08.10 was used to place and route the design; NCSim v08.20 was used to simulate the routed design and generate a value-change-dump (VCD) file; and a VCD analyzer was used to generate simulated power traces from the VCD file by applying a toggle-counting technique. By accumulating all toggles within a clock cycle, the resulting trace contained one data value per clock cycle. As shown in Kirschbaum *et al.* [19], simulated power traces derived from toggle counts are well comparable to SPICE power simulations as well as to real power measurements of an integrated circuit.

5.2 FPGA Measurement Setup

We furthermore synthesized the design (with $d = 1$) on an FPGA. This step enables us to record the power consumption during the targeted point multiplication performed on a real-world device. As FPGA a Virtex-II Pro xc2vp7 was used which is part of the SASEBO [35] side-channel evaluation board. As the development environment, Xilinx 10.1.03 was used. For recording the power traces we have used a LeCroy WP725Zi oscilloscope with a sampling rate of 2.5 GS/s and operate the device under test at a clock rate of 25 MHz. Our first measurements showed that for the attacks an accurate clock frequency is beneficial in order to avoid the introduction of noise due to clock jitter. Using an off-the-shelf quartz increases the effort for the attack as additional postprocessing steps on the recorded trace are required. In particular, a fixed number of samples per clock period is required for the attack, that means the ratio between the sampling rate and the clock rate must be integer. If this is not the case an upsampling of the recorded trace is required in order to achieve this ratio. To circumvent this we used an Agilent 33250A signal generator as external clock source. Further the complete point multiplication must be finished before the used oscilloscope runs out of "input buffer". Our LeCroy WP725Zi oscilloscope is capable of storing 64 million samples per trace.

In order to decrease the calculation and memory effort we performed a downsampling step on the recorded trace. In this preprocessing step, all the sample points within one clock period of the device were summed up. Once again, we want to emphasize that the following attacks work using only *a single power trace* of the scalar multiplication.

6 Assuring Side-Channel Resistance

In order to make sure that our implementation has no key dependent leakage, we performed a basic difference-of-means analysis with a known scalar. By redesigning the hardware, all sources of leakage were eliminated. The difference-of-means trace in Figure 3 shows the leakage of a preliminary version of our hardware design under investigation.

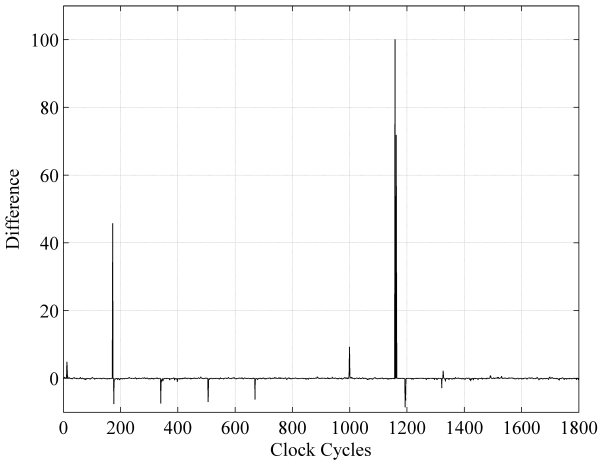

Fig. 3. Difference-of-means of a simulated power trace.

Theory. To launch a difference-of-means attack (cf. [25]), all key-dependent round traces are categorized according to k_i. Next, the average $avg(R_i|_{k_i,\forall i})$ and the difference of the means in the respectable categories $avg(R_i|_{k_i=0,\forall i}) - avg(R_i|_{k_i=1,\forall i})$ are calculated. By investigation of the resulting plot it is possible to find key-dependent operations in the trace.

Practical Results. A difference-of-means calculation was performed using the subtraces R_i extracted from a single measured power trace from an early FPGA implementation and is depicted in Figure 3. The operations at clock cycles with a high difference show some key-dependent behavior. The higher the difference is, the easier it is for an attacker to find the key-dependent parts. This results allow a designer to identify key-dependent operations and improve the design. In our special case we found that at clock cycle 1160, the access to the data memory was dependent on the key bit k_i. For all subsequent experiments this error was obviously fixed.

We redesigned our hardware implementation until the most significant peaks of the difference-of-means analysis vanished. Therefore we covered the basics and assured the significance of the following power analysis attacks.

7 Correlation of Consecutive Rounds

The assumption is that in consecutive rounds R_i and R_{i-1} similarities dependent on the processed bits k_i and k_{i-1} can be found. This holds for our implementation using the formulas of López and Dahab (cf. Appendix A) for point multiplication but is also applicable to other algorithms.

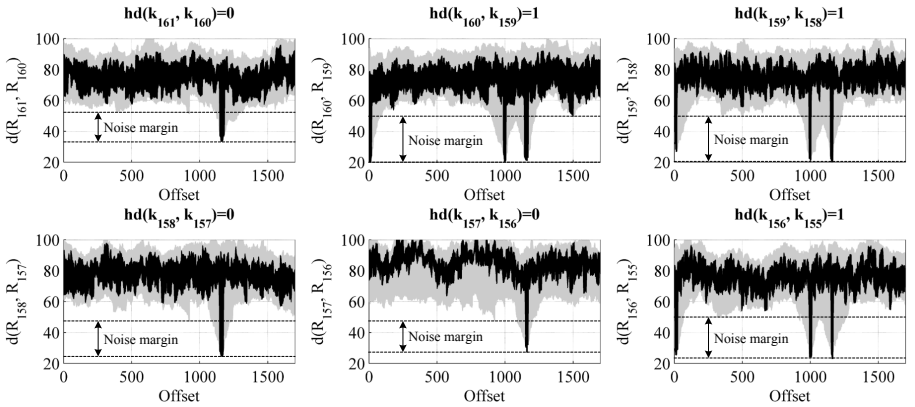

Fig. 4. Euclidean distance traces.

Theory. In this scenario we focus on finding similarities in the power traces R_i and R_{i-1} of two consecutive rounds. Once again it has to be mentioned that these power traces are subtraces of equal length of one power trace recorded during the point multiplication. The idea behind this approach is that the power profile of a digit-serial multiplication is mostly dependent on OpB. If the same operand OpB has been used in two consecutive rounds similarities can be found. Witteman *et al.* [38] have used a similar assumption in their attack on the RSA modular exponentiation. They take advantage of the fact that the square-and-multiply-always algorithm reuses operands in a key-dependent manner. As similarity measure we have tried the correlation coefficient as well as the Euclidean distance. Both approaches lead to comparable results with runtime advantages for the Euclidean distance. Equation 1 shows how to calculate the Euclidean distance of two consecutive rounds with offset o_i, o_{i-1} respectively and a window size w. In Equation 2, the formula to generate the distance matrix for the rounds i and $i-1$ with the given limits for j and l can be found.

$$d(R_i, R_{i-1})|_{o_i,o_{i-1}} = ||R_i(o_i, o_i + w), R_{i-1}(o_{i-1}, o_{i-1} + w)|| \tag{1}$$

$$Dist_l^j(R_i, R_{i-1}) = d(R_i, R_{i-1})|_{j,l} \tag{2}$$

$$|k| \geq i \geq 1; 0 \leq j \leq t_{round} - w; 0 \leq l \leq t_{round} - w;$$

Practical Results. Figure 4 shows the results of a windowed correlation of two consecutive rounds with a window size $w = 163$ using the traces recorded from the FPGA. The length of the subtraces R_i is 1796 samples. Small Euclidean distances are indicators for similar intermediate values. Those traces with small Euclidean distances have been highlighted. Two cases are distinguishable in Figure 4. In the first case (e.g., between rounds 160 and 159) three locations with a small Euclidean distance can be identified. In the second case (e.g., between

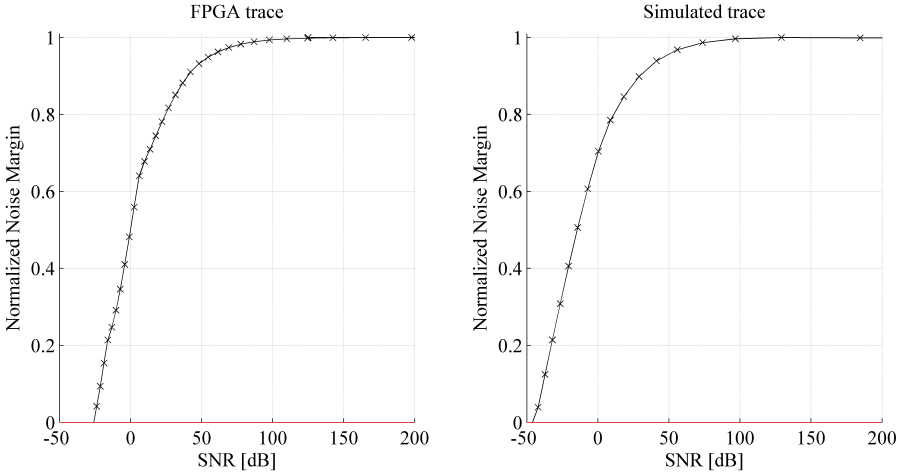

Fig. 5. Influence of noise on the noise margin.

rounds 161 and 160) only one location with a small Euclidean distance can be identified. By investigating the formulas of López and Dahab [24], we identified the peak around offset 1150 as $OpB = c = b^{2^{m-1}} \mod f(z)$. This single peak appears in all correlation figures at the same position. The other two peaks appear, when the Hamming distance $hd(k_i, k_{i-1}) = 1$. In other words: the peaks appear, when the key bit k_i is different from k_{i-1}. In such a case one operand OpB from round i is used again (unchanged) in round $i-1$. For a better understanding of the underlying problem, we attached the used formulas in Appendix A.

We also performed the same attack on the simulated traces and achieved comparable results. As the simulated traces do not contain measurement noise, the minimum Euclidean distances are even smaller. The presented attack worked for $d = 1$, $d = 2$, and $d = 4$ (used window sizes: $w = 163$, $w = 82$, and $w = 41$ respectively).

In order to visualize the influence of noise (e.g. introduced by the measurement environment or active noise countermeasures), simulated gaussian noise with different power levels has been added to the simulated as well as the measured trace. Figure 5 shows the normalized noise margin as a function of the signal to noise ratio SNR. The SNR has been calculated according to Equation 3. The evolution of the noise margin for simulation and FPGA experiment is similar, only the value of the SNR where the noise margin comes below zero is different. For the FPGA experiment the noise margin comes below zero at -24 dB and for the simulation at -43 dB. The difference can be explained by the fact that the trace recorded with the FPGA already contains some noise. The trace extracted using the simulation on the other hand can be seen as noise free.

$$\mathrm{SNR} = 20 \cdot \log \frac{U_{\mathrm{eff,Signal}}}{U_{\mathrm{eff,Noise}}} \tag{3}$$

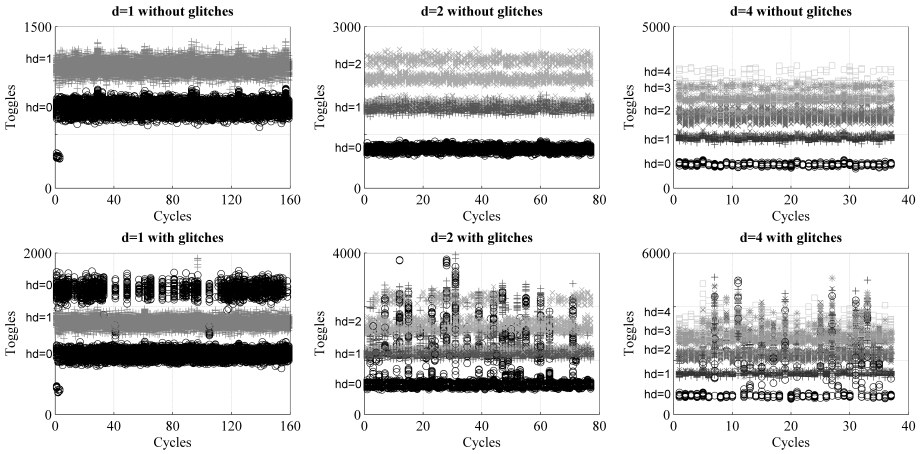

Fig. 6. Simulated power profiles for different d in the presence of glitches.

8 Revealing Intermediate Operands

In this section, we want to uniquely identify the intermediate finite-field polynomials used as operand OpB and discuss some more advanced attack scenarios. For a better understanding of the associated challenges, let us first investigate practical simulations.

Assumptions & Transferability. For this scenario we need to assume that a digit-serial multiplier is used. It is applicable to all designs based on digit-serial multipliers.

Practical Results. Figure 6 shows simulated toggle counts for $|k|$ overlayed N-bit multiplications. Using reference simulations, we marked the different Hamming distances $hd(OpB_i)$ with respective symbols. The following three characteristics are eye-catching: The different Hamming distances are distinguishable. The larger the digit size d is, the less distinguishable are the different levels of power consumption. And the more glitches occur on OpB_i, the less separable are those power levels. In our hardware design, we used an N-bit wide 5:1 multiplexer and a N:d multiplexer to select OpB_i. The resulting glitches are clearly visible in the lower row of Figure 6. The upper row is the result of a d bit register being added after the multiplexers to suppress those glitches[2]. Also if the multiplexers are replaced by shift registers, there are virtually no glitches.

Interestingly, in the case of $d = 1$, glitches only occurred in a $1 \rightarrow 1$ transition. Obviously, the AOI-gates used in our design suppress glitches in $0 \rightarrow 0$ transitions.

[2] For high-performance ECC implementations those registers are necessary in order to achieve the desired timings.

Table 1. Average number of solutions for OpB assuming $hd(OpB)$ is given.

Parameter	**N = 163**	**N = 256**
$d = 1$	$2^1 = 2^1$	$2^1 = 2^1$
$d = 2$	$2^2 2^{0.5\times 81} = 2^{42.5}$	$2^2 2^{0.5\times 127} = 2^{65.5}$
$d = 3$	$2^3 3^{0.75\times 54} = 2^{67.2}$	$2^3 3^{0.75\times 85} = 2^{104}$
$d = 4$	$2^4 4^{0.5\times 40} 6^{0.375\times 40} = 2^{82.8}$	$2^4 4^{0.5\times 63} 6^{0.375\times 63} = 2^{128.1}$

Identifying the Intermediates. At this point, the attacker wants to identify the intermediate values processed during the field multiplications by using the given power traces. The identification of the operand(s) OpB_i has to be done in two phases:

At first, the leaking Hamming distances in the power trace have to be classified. By overlaying all $|k|$ rounds, as it has been done in Figure 6, it is possible to distinguish clusters of different Hamming distances. This classification can for instance be performed using a k-means algorithm [16]. We performed this classification on our simulated power traces and were able to detect the correct Hamming distances with the following error rates: 0.33% for $d = 1$ without and with glitches, 4.13% for $d = 2$ without glitches, 9.31% for $d = 2$ with glitches, 4.96% for $d = 4$ without glitches and 9.04% for $d = 4$ with glitches.

During the second phase, we used the Hamming distances $hd(OpB)$ to iterate through all possible solutions. Several possible solutions for OpB result in the same $hd(OpB)$. Equation 4 gives an average number of possible solutions $N_{solutions}$ in dependence of N and d, when $hd(OpB)$ is given. $\#(hd = h)$ is the number of possible solutions for a fixed Hamming distance h and $p(hd = h)$ is the probability of the Hamming distance h. Since it is necessary to guess OpB_{MSB}, 2^d is a multiplicand factor within the equation. Table 1 gives exemplary numbers for $N = 163$, $N = 256$, and $d = 1...4$.

$$N_{solutions} = 2^d \cdot \left(\prod_{h=0}^{d} \#(hd = h)^{p(hd=h)} \right)^{\lceil \frac{N}{d} \rceil - 1} \tag{4}$$

The following short example should clarify the problematic: When $d = 2$, $hd(OpB_i, OpB_{i+1})$ is either 0, 1, or 2. Whereas $hd = 0$ and $hd = 2$ uniquely map OpB_i to a single OpB_{i+1}, $hd = 1$ does not. Let us assume $OpB_i = 00_b$. Then there are two solutions $hd(00_b, 01_b) = hd(00_b, 10_b) = 1$. With our power model[3], those two solutions cannot be distinguished in the power trace. In average there are $2^{42.5}$ possible solutions (cf. Table 1) for $d = 2$ and $N = 163$. Using brute-force, breaking $2^{42.5}$ is practicable.

Other exemplary cases such as $d > 2$ or $N = 256$ are theoretically possible but impractical. However Table 1 depicts an *average* case. In practice the number of possibilities $N_{solutions}$ for a certain OpB depends on $hw(OpB)$. Hence, it is clever to only attack OpB with a small search space. An approximation of the necessary runtime can be performed in constant time. Furthermore we are

[3] Note that this might be possible using more advanced power models.

confident that by investigating the different arrival times of single bits in OpB_i and by using more detailed timing information, our leakage model can be refined.

For many practical implementations without point randomization, finding those intermediate values would be sufficient. However, in the context of this paper (with point randomization) finding the intermediate values is only a preparational step for the following attack scenarios.

8.1 Correlate with an Arithmetic Combination of Intermediates.

In Section 7, a direct correlation of the power trace on two consecutive rounds was performed. By changing the order of the operands of the multiplication this attack can be prevented. In many cases there is still a link between consecutive rounds when the result of an arithmetical combination of several operands $OpB^1, OpB^2, \ldots$ in round i is used as operand in round $i+1$. If the arithmetical combination $f(\cdot)$ as well as a set of operands $OpB^1, OpB^2, \ldots$ for round i is known, a set of used operands $F = f(OpB^1, OpB^2, \ldots)$ for round $i+1$ can be calculated. Feeding $f(\cdot)$ with every possible combination of $OpB^1, OpB^2, \ldots$ and performing the correlation $d(hd(F), R_{i-1})$ the size of the possible key candidates can be decreased. The complexity of this attack highly depends on the number of operands N_{OpB} used as arguments for $f(\cdot)$ as well as on d. N_{OpB} as well as d have an influence on the number of possible results for F. If e.g. $f(\cdot)$ is a squaring function ($N_{OpB} = 1$) and $d \leq 2$ the attack can be performed within acceptable bounds. Furthermore it has to be considered that if some bits in F are wrong there might still be a significant peak in the correlation plot, so there is no need to find the exact value of F. This fact also decreases the attack complexity.

8.2 Attack Several Intermediates Simultaneously.

An enhancement of the previous scenario is to attack $(OpB^1, OpB^2, \ldots)$ as well as $F = f(OpB^1, OpB^2, \ldots)$ simultaneously. Let us assume the Hamming distances of F and $OpB^1, OpB^2, \ldots$ are known, so only a limited number of combinations for F and $OpB^1, OpB^2, \ldots$ fulfill the equation $F = f(OpB^1, OpB^2, \ldots)$. Although we did not test it, we are confident that by attacking different intermediates simultaneously the search space $N_{solutions}$ can be reduced by a certain degree.

8.3 Find the x-Coordinate.

If an attacker can reveal the exact values for X_i (projective X coordinate in round i) and Z_i, she can calculate the x-coordinate of the currently processed point. Even if X_i and Z_i have been randomized and multiplied with a random λ this attack works. In the case of $X_r = X_i \cdot \lambda$ and $Z_r = Z_i \cdot \lambda$,

$$x_i = X_r \cdot Z_r^{-1} = (\lambda X_i) \cdot (\lambda Z_i)^{-1} = X \cdot Z^{-1} \tag{5}$$

can be recovered. Assuming a scalar multiplication $Q = k \times P$ is performed, small multiples of P can be precalculated and compared with x_i which is revealed

in each round. Thus step by step all bits of k can be revealed. Even if there is an error in round i and x_i cannot be matched, x_{i+1} can be used to identify more key bits at once.

8.4 Undo the Projective Coordinate Randomization.

In order to perform a projective coordinate randomization, each coordinate is multiplied with a random number λ. For that operation usually the same finite-field multiplier is used as the finite-field multiplier used during a point multiplication. So in the case of $X_r = X \cdot \lambda$ and λ is used as OpB, the randomization factor can be found. By knowing λ many attack scenarios on a device doing a point multiplication become feasible.

9 Conclusion

Nowadays the community knows that no implementation is believed to be secure as long as it has not been attacked and investigated in sufficient detail. In this paper, we attacked a protected ECC implementation by using only a single power trace. So even the popularly used ECDSA and the Diffie-Hellman key exchange algorithms are vulnerable. The corollary one should take away from this paper is that a designer must not only consider a simple difference-of-means attack but also be aware of more advanced and unexpected attack scenarios such as the here shown custom correlation attack or the attack on all processed intermediate values. But how should any designer be aware of future, currently unknown attacks?

Acknowledgments

The research described in this paper has been supported, in parts, by the European Commission through the ICT Program under contract ICT-SEC-2009-5-258754 TAMPRES, and by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

References

1. H. Aigner, H. Bock, M. Hütter, and J. Wolkerstorfer. A Low-Cost ECC Coprocessor for Smartcards. In M. Joye and J.-J. Quisquater, editors, *CHES 2004, Proceedings*, volume 3156 of *LNCS*, pages 107–118. Springer, 2004.
2. T. Akishita and T. Takagi. Power Analysis to ECC Using Differential Power Between Multiplication and Squaring. In *CARDIS 2006*, volume 3928 of *LNCS*, pages 151–164, April 2006.
3. F. Amiel, B. Feix, M. Tunstall, C. Whelan, and W. Marnane. Distinguishing Multiplications from Squaring Operations. In R. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 346–360. Springer, 2009.

4. L. Batina, N. Mentens, S. B. Örs, and B. Preneel. Serial Multiplier Architectures over GF(2^n) for Elliptic Curve Cryptosystems. In *IEEE Mediterranean Electronical Conference – MELECON 2004*, pages 779–782. IEEE, May 2004.
5. L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks – ESAS 2006, Revised Selected Papers*, volume 4357, pages 6–17, Berlin Heidelberg, 2006. Springer-Verlag.
6. H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008, July 2008.
7. D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
8. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Horizontal Correlation Analysis on Exponentiation. In M. Soriano, S. Qing, and J. López, editors, *Information and Communications Security*, volume 6476 of *LNCS*, pages 46–61. Springer, 2010.
9. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
10. J.-F. Dhem, F. Kœune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A Practical Implementation of the Timing Attack. In J.-J. Quisquater and B. Schneier, editors, *CARDIS 1998, Proceedings*, number 1820 in LNCS, pages 167–182. Springer, 1998.
11. H. Eberle, N. Gura, S. C. Shantz, and V. Gupta. A Cryptographic Processor for Arbitrary Elliptic Curves over GF(2^m). In E. Deprettere, S. Bhattacharyya, J. Cavallaro, A. Darte, and L. Thiele, editors, *Application-Specific Systems, Architectures, and Processors – ASAP 2003*, pages 444–454, June 2003.
12. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of 2007 IEEE International Symposium on Circuits and Systems*. IEEE, IEEE, May 2007.
13. C. H. Gebotys and R. J. Gebotys. Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002, Revised Papers*, volume 2523 of *LNCS*, pages 114–128. Springer, 2003.
14. J. Großschädl. A Bit-Serial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m). In Ç. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001, Proceedings*, volume 2162, pages 202–219. Springer Berlin / Heidelberg, May 2001.
15. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
16. J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. volume 28, pages 100–108. Blackwell Publishing for the Royal Statistical Society, 1979.
17. C. Herbst and M. Medwed. Using Templates to Attack Masked Montgomery Ladder Implementations of Modular Exponentiation. In K.-I. Chung, M. Yung, and K. Sohn, editors, *Workshop on Information Security Applications – WISA 2008, Proceedings*, volume 5379 of *LNCS*, pages 1–13. Springer, Februar 2008.

18. N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE Transactions on Computers*, 59(6):795–807, 2010.
19. M. Kirschbaum and T. Popp. Evaluation of Power Estimation Methods Based on Logic Simulations. In K. C. Posch and J. Wolkerstorfer, editors, *Proceedings of Austrochip 2007, October 11, 2007, Graz, Austria*, pages 45–51. Verlag der Technischen Universität Graz, October 2007. ISBN 978-3-902465-87-0.
20. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO 1996, Proceedings*, number 1109 in LNCS, pages 104–113. Springer, 1996.
21. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, pages 388–397, 1999.
22. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security – RFIDSec 2006*, 2006.
23. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
24. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^{m}) without Precomputation. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 316–327. Springer, 1999.
25. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer, 2007. ISBN 978-0-387-30857-9.
26. M. Medwed and E. Oswald. Template Attacks on ECDSA. In K.-I. Chung, M. Yung, and K. Sohn, editors, *Workshop on Information Security Applications – WISA 2008, Pre-Proceedings*, pages 14–27, 2008.
27. B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In G. I. Davida and Y. Frankel, editors, *Information Security Conference – ISC 2001, Proceedings*, volume 2200 of *LNCS*, pages 324–334. Springer, 2001.
28. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.
29. NXP. Jcop 41 v2.3.1 java card, 2007.
30. G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In Ç. K. Koç and C. Paar, editors, *CHES 2000, Proceedings*, volume 1965 of *0302-9743*, pages 41–56, London, UK, August 2000. Springer.
31. S. B. Örs, E. Oswald, and B. Preneel. Power-Analysis Attacks on FPGAs – First Experimental Results. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *CHES 2003, Proceedings*, volume 2779 of *LNCS*, pages 35–50. Springer, 2003.
32. E. Oswald. Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002, Revised Papers*, volume 2523 of *LNCS*, pages 82–97. Springer, 2003.
33. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In M. Joye and J.-J. Quisquater, editors, *CHES 2004, Proceedings*, volume 3156 of *LNCS*, pages 92–106. Springer, August 2004.
34. W. Pan and W. P. Marnane. A Correlation Power Analysis Attack against Tate Pairing on FPGA. In *Reconfigurable Computing: Architectures, Tools and Applications – ARC 2011, Proceedings*, volume 6578 of *LNCS*, pages 340–349. Springer-Verlag, 2011.
35. Side-channel attack standard evaluation board. The SASEBO Website. `http://staff.aist.go.jp/akashi.satoh/SASEBO/en/index.html`.

36. C. D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems CHES 2001*, volume 2162 of *LNCS*, pages 286–299. Springer, 2001.
37. C. D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In M. Joye and J.-J. Quisquater, editors, *CHES 2004, Proceedings*, volume 3156 of *LNCS*, pages 191–204. Springer, 2004.
38. M. F. Witteman, J. G. van Woudenberg, and F. Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In S. B. Heidelberg, editor, *Topics in Cryptology CT-RSA*, pages 77–88, 2011.
39. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto, July 13-15, 2005, Graz, Austria*, pages 78–91, 2005.

A Used Double-And-Add Formula

In this paper we used a slight modification of the Montgomery Ladder by López and Dahab. Algorithm 1 shows three iterations of the used double-and-add algorithm for three consecutive key bits. The modification from the original formula can be found in line 6. Here we swapped the order of x and Z_1, which is allowed according to the law of commutativity. During the first two iterations (left and middle column), an identical key bit is handled. So there is only a correlation when the constant c is used. During the second and third iteration, in which the key bit differs, Z_1 is used multiple times. Consequently in Figure 4 three easily distinguishable peaks occur. A correlation of c and Z_1 can be observed.

Algorithm 1 López and Dahab round operations with key bits (0-0-1).

Ensure: $P_1' \leftarrow P_1 + P_2$.	**Ensure:** $P_1' \leftarrow P_1 + P_2$.	**Ensure:** $P_2' \leftarrow P_2 + P_1$.
Ensure: $P_2' \leftarrow 2 \cdot P_2$.	**Ensure:** $P_2' \leftarrow 2 \cdot P_2$.	**Ensure:** $P_1' \leftarrow 2 \cdot P_1$.
Point Addition	Point Addition	Point Addition
1: $X_1 \leftarrow X_1 \cdot Z_2$	1: $X_1 \leftarrow X_1 \cdot Z_2$	1: $X_2 \leftarrow X_2 \cdot$ **Z_1**
2: $Z_1 \leftarrow Z_1 \cdot X_2$	2: $Z_1 \leftarrow Z_1 \cdot X_2$	2: $Z_2 \leftarrow Z_2 \cdot X_1$
3: $T_1 \leftarrow X_1 \cdot Z_1$	3: $T_1 \leftarrow X_1 \cdot Z_1$	3: $T_1 \leftarrow X_2 \cdot Z_2$
4: $Z_1 \leftarrow Z_1 + X_1$	4: $Z_1 \leftarrow Z_1 + X_1$	4: $Z_2 \leftarrow Z_2 + X_2$
5: $Z_1 \leftarrow Z_1 \cdot Z_1$	5: $Z_1 \leftarrow Z_1 \cdot Z_1$	5: $Z_2 \leftarrow Z_2 \cdot Z_2$
6: $X_1 \leftarrow x \cdot Z_1$	6: $X_1 \leftarrow x \cdot$ **Z_1**	6: $X_2 \leftarrow x \cdot Z_2$
7: $X_1 \leftarrow X_1 + T_1$	7: $X_1 \leftarrow X_1 + T_1$	7: $X_2 \leftarrow X_2 + T_1$
Point Doubling	Point Doubling	Point Doubling
8: $X_2 \leftarrow X_2 \cdot X_2$	8: $X_2 \leftarrow X_2 \cdot X_2$	8: $X_1 \leftarrow X_1 \cdot X_1$
9: $Z_2 \leftarrow Z_2 \cdot Z_2$	9: $Z_2 \leftarrow Z_2 \cdot Z_2$	9: $Z_1 \leftarrow Z_1 \cdot$ **Z_1**
10: $T_1 \leftarrow Z_2 \cdot$ **c**	10: $T_1 \leftarrow Z_2 \cdot$ **c**	10: $T_1 \leftarrow Z_1 \cdot$ **c**
11: $Z_2 \leftarrow Z_2 \cdot X_2$	11: $Z_2 \leftarrow Z_2 \cdot X_2$	11: $Z_1 \leftarrow Z_1 \cdot X_1$
12: $T_1 \leftarrow T_1 \cdot T_1$	12: $T_1 \leftarrow T_1 \cdot T_1$	12: $T_1 \leftarrow T_1 \cdot T_1$
13: $X_2 \leftarrow X_2 \cdot X_2$	13: $X_2 \leftarrow X_2 \cdot X_2$	13: $X_1 \leftarrow X_1 \cdot X_1$
14: $X_2 \leftarrow X_2 + T_1$	14: $X_2 \leftarrow X_2 + T_1$	14: $X_1 \leftarrow X_1 + T_1$

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

(date) (signature)