A Survey of Lightweight-Cryptography Implementations

Thomas Eisenbarth
Ruhr University Bochum

Christof Paar and Axel Poschmann
Ruhr University Bochum

Sandeep Kumar
Philips Research Europe

Leif Uhsadel
Catholic University of Leuven

Editor's note:
The tight cost and implementation constraints of high-volume products, including secure RFID tags and smart cards, require specialized cryptographic implementations. The authors review recent developments in this area for symmetric and asymmetric ciphers, targeting embedded hardware and software.

—Patrick Schaumont, Virginia Tech

THE UPCOMING ERA of pervasive computing will be characterized by many smart devices that—because of the tight cost constraints inherent in mass deployments—have very limited resources in terms of memory, computing power, and battery supply. Here, it's necessary to interpret Moore's law differently: Rather than a doubling of performance, we see a halving of the price for constant computing power every 18 months. Because many foreseen applications have extremely tight cost constraints—for example, RFID in tetrapacks—over time, Moore's law will increasingly enable such applications. Many applications will process sensitive health-monitoring or biometric data, so the demand for cryptographic components that can be efficiently implemented is strong and growing. For such implementations, as well as for ciphers that are particularly suited for this purpose, we use the generic term *lightweight cryptography* in this article.

Every designer of lightweight cryptography must cope with the trade-offs between security, cost, and performance. It's generally easy to optimize any two of the three design goals—security and cost, security and performance, or cost and performance; however, it is very difficult to optimize all three design goals at once. For example, a secure and high-performance hardware implementation can be achieved by a pipelined, side-channel-resistant architecture, resulting in a high area requirement, and thus high costs. On the other hand, it's possible to design a secure, low-cost hardware implementation with the drawback of limited performance.

In this article, we present a selection of recently published lightweight-cryptography implementations and compare them to state-of-the-art results in their field. This survey covers recent hardware and software implementations of symmetric as well as asymmetric ciphers. We will discuss software and hardware implementations separately, because they have different and sometimes contrary characteristics. For example, bit permutations are virtually free in hardware, whereas in software they can significantly slow down implementations. Also, large substitution tables are often software friendly, but hardware realizations can be relatively costly. Finally, the evaluation metric is different: For software implementations, we compare both RAM and ROM requirements and the required number of clock cycles. For hardware implementations, we focus on the required chip size and the number of clock cycles. We don't compare power consumption for the hardware implementations, because different standard-cell technologies were used and estimates from simulating environments are not accurate. Software implementations let us achieve a rough estimate of power

0740-7475/07/$25.00 © 2007 IEEE Copublished by the IEEE CS and the IEEE CASS

IEEE Design & Test of Computers dtco-24-06-pos.3d 4/10/07 08:41:02 0 Cust # pos

consumption by multiplying the processing time by the average power consumption of the target device.

Another distinction is between symmetric and asymmetric ciphers, because the latter offer more security functionality and therefore have different application scenarios. Symmetric ciphers serve mainly for message integrity checks, entity authentication, and encryption, whereas asymmetric ciphers additionally provide key-management advantages and nonrepudiation. Asymmetric ciphers are computationally far more demanding, in both hardware and software. The performance gap on constrained devices such as 8-bit microcontrollers is huge. For example, an optimized asymmetric algorithm such as elliptic-curve cryptography (ECC) performs 100 to 1,000 times more slowly than a standard symmetric cipher such as the Advanced Encryption Standard (AES) algorithm, which correlates with a two- to three-orders-of-magnitude higher power consumption. Unlike block ciphers, which are well investigated and understood, stream ciphers have received little attention from the scientific community. Although this has recently started to change, we cite stream ciphers only for comparison. (The increasing interest in stream ciphers is apparent in projects such as eStream, within the European Network of Excellence in Cryptography, which aims to foster knowledge about stream ciphers.) The ''Related work'' sidebar summarizes some recent developments in lightweight cryptography.

Symmetric ciphers

Many works on symmetric ciphers have been published during the past two decades. Because most main applications of symmetric ciphers use software implementations, it's no surprise that nearly all algorithms—for example, the AES—have been developed with good software performance in mind. The paradigm shift that we foresee will likely lead to an increasing demand for lightweight ciphers that perform well in hardware. Therefore, we focus here on recently published works on ciphers that have been developed for minimal hardware requirements—namely, DESL[1] and Present.[2]

Hardware implementations of symmetric ciphers

The only well-established cipher designed with a strong focus on low hardware cost is the Data Encryption Standard (DES). Comparing a standard one-round implementation of AES and DES, we find that the latter consumes only about 6% of the logic resources of AES and has a shorter critical path. However, researchers have described a low-power, low-cost AES implementation that requires only 3,400 gate equivalents (GEs) and encrypts a plaintext within 1,032 clock cycles.[3] This impressive result seems to have achieved the limit in area minimization. This implementation, as well as the implementations of DESL and Present, features encryption only, because encryption is sufficient for many lightweight target applications, such as authentication with a challenge-response protocol.

DES. Inspired by the one-round implementation results of AES and DES, we implemented a serialized version of DES that processes 4-bit and 6-bit data words rather than those with 32 bits and 48 bits. Our implementation requires 2,310 GEs and encrypts a plaintext within 144 clock cycles.[1] To our knowledge, this is the smallest reported DES implementation, sacrificing throughput to achieve minimal area requirements. However, the 56-bit key limits the security provided. Brute-forcing this key space using software takes a few months and hundreds of PCs, but only a few days with a special-purpose machine such as Copacobana.[4] Hence, this implementation is relevant only for applications needing short-term security or where the values protected are relatively low. In certain low-cost applications, such a security level is adequate. When a higher security level is needed, so-called *key whitening*, can be added to standard DES, yielding DESX. The key-whitening technique requires only two additional XOR gates: one gate to add a prewhitening key to the plaintext before the cipher processes it, and another to add a postwhitening key to the resulting ciphertext. In the case of DES, this enlarges the key space from 56 bits to 184 bits. However, because of time-memory trade-offs (birthday attack), the security level of DESX is bounded by 118 bits.

DESL and DESXL. In our serialized DES implementation, substitution boxes (S-boxes) take up approximately 32% of the area. We can further decrease the gate complexity of DES by replacing the eight original S-boxes with a single new one, eliminating seven S-boxes as well as the multiplexer. This lightweight DES variant is called DESL and results in an approximately 20% smaller chip than DES (1,850 GEs versus 2,310 GEs). The S-box has been carefully selected and highly optimized, enabling DESL to resist common attacks

Related work

There are many recent symmetric ciphers—for example, Hight, Clefia, DESXL,[1] and Present[2]—with special implementation properties proposed. Hight was first presented at the 2006 Workshop on Cryptographic Hardware and Embedded Systems and was designed with good hardware performance in mind. In their paper, the authors provide hardware figures for a one-round implementation—that is, one round is performed in one cycle—and they conclude that Hight is well-suited for ubiquitous computing devices such as wireless sensor nodes and RFID tags. Their figures show that Hight requires approximately the same chip size as the Advanced Encryption Standard (AES) algorithm (3,048 versus 3,400 gate equivalents, or GEs) but is much faster. However, figures for implementations with a smaller footprint in hardware are not yet available. Clefia was designed with a broader application range in mind—to perform well in both hardware and software implementations. Two ciphers especially optimized for software architectures are the Tiny Encryption Algorithm (TEA) family and the International Data Encryption Algorithm (IDEA). They consist only of arithmetic operations on 16-bit words (IDEA) and 32-bit words (TEA). IDEA consists of addition, XOR addition, and multiplication. The TEA family uses only addition, XOR addition, and shifts. In both cases, each operation can be implemented efficiently on 8-bit platforms. Neither cipher uses a substitution box (S-box), so they don't need much memory. The Scalable Encryption Algorithm (SEA) can be parameterized according to processor size as well as plaintext size and key size; the goal is to enable efficient implementations on different platforms.

With respect to asymmetric algorithms on small processors, Gura et al.[3] make a key contribution, comparing RSA (Rivest, Shamir, Adleman) and elliptic-curve cryptography (ECC) on two different, commonly used 8-bit CPUs: AVR (the Atmel ATmega128 platform) and 8051 (the Chipcon CC1010 platform). They show that for the Atmel ATmega128 clocked at 8 MHz, a point multiplication using a 160-bit ECC $GF(p)$ standard curve required 0.81 second. A security equivalent 1,024-bit RSA encryption requires about 11 seconds. Consequently, this article considers only ECC in the asymmetric case. Although hyperelliptic curves hold promise too, their lack of standardization makes them less promising at the moment. Optimum extension fields (OEFs), which can be parameterized for small CPUs, offer an alternative method for fast ECC implementations.[4] Several hardware implementations for standardized ECC have been suggested, but few are aimed at low-end devices. Most implementations focus on speed and, owing to their huge area requirements, are suitable mostly for server-end applications only.[5]

References

1. G. Leander et al., "New Lightweight DES Variants," *Proc. Fast Software Encryption* (FSE 07), LNCS 4593, Springer-Verlag, 2007, pp. 196-210.
2. A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher," *Proc. Workshop Cryptographic Hardware and Embedded Systems* (CHES 07), LNCS 4727, Springer, 2007, pp. 450-466.
3. N. Gura et al., "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," *Proc. 6th Int'l Workshop Cryptographic Hardware and Embedded Systems* (CHES 04), Springer-Verlag, LNCS 3156, 2004, pp. 119-132.
4. A. Woodbury, D.V. Bailey, and C. Paar, "Elliptic Curve Cryptography on Smart Cards without Coprocessors," *Proc. 4th Working IFIP Conf. Smart Card Research and Advanced Applications* (CARDIS 2000), Kluwer Academic, 2000, pp. 71-92.
5. L. Batina et al., "Hardware Architectures for Public Key Cryptography Integration," *VLSI J.*, vol. 34, no. 6, 2003, pp. 1-64.

such as linear and differential cryptanalysis and the Davies-Murphy attack. Thus, DESL achieves a security level appropriate for many applications. Key whitening can be applied to strengthen the cipher, yielding the DESXL cipher, with a security level of approximately 118 bits. DESXL requires 2,170 GEs and encrypts a plaintext within 144 clock cycles. (Further details are available elsewhere.[1])

Present. Besides efficiently implementing or slightly modifying an established cipher, an alternative for lightweight cryptography is to design a new hardware-optimized cipher from scratch. We followed this approach when designing Present, an SPN-based (substitution permutation network) block cipher with 32 rounds, a block size of 64 bits, and a key size of 80 or 128 bits. The main design philosophy was

simplicity: No part of the cipher was added without a good reason, such as thwarting an attack. Figure 1 depicts Present's very simple design. Let's take the round counter XOR in the key scheduling as an example of how to thwart a whole class of attacks with minimal area overhead. Hardware designers favor repetition because it lets them reuse parts of the chip and hence reduce chip size. However, if the key-update functions are similar in each round, this property can be exploited by related-key attacks. Among the possibilities for achieving deviating round functions, we chose the one that is most hardware efficient and simple: adding a round-dependent constant in the key scheduling. Because a round counter is needed anyway, this constitutes no additional area requirements; the XOR requires only 13 GEs.

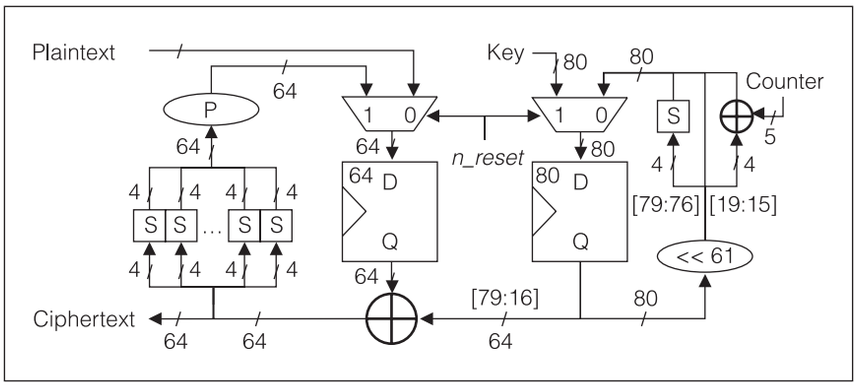

Figure 1. Data path of the Present implementation. (S: substitution layer; P: permutation layer.) (Source: Bogdanov et al.[2])

Present, like any other SPN, comprises three stages: a key-mixing step, a substitution layer, and a permutation layer. For the key mixing, we chose a simple XOR because this operation can be implemented efficiently in both hardware and software. The key schedule consists essentially of a 61-bit rotation together with an S-box and a round counter. (Present-80 uses a single S-box, whereas Present-128 requires two S-boxes.) The substitution layer comprises 16 S-boxes with 4-bit inputs and 4-bit outputs (4×4). We decided to use similar S-boxes in both the data path and the key scheduling because we learned from DESL that this can result in significant area savings when a serialized implementation is desired. The choice of 4×4 rather than 8×8 S-boxes was also hardware driven; 8-bit S-boxes require about 40 times more area than 4-bit S-boxes (1,000 GEs versus 25 GEs). However, 4-bit S-boxes must be selected very carefully because they are cryptographically weaker than 8-bit S-boxes. Nevertheless, through careful selection, it's possible to achieve an appropriate security level. The permutation layer is a very regular and simple bit transposition. It comes virtually free in hardware because it is realized by simple wiring and hence needs no transistors. The permutation layer ensures that an S-box's four output bits will be distributed to four distinct S-boxes in the following round, which ensures the avalanche effect. This is required to thwart linear and differential cryptanalyses. (Further details are available elsewhere.[2])

Table 1 compares the implementation results of various lightweight ciphers.

Software implementations of symmetric ciphers

The many design choices within the AES reflect the wide discussion of efficient symmetric cryptography for software implementations. Yet, for constrained devices in particular, designers must take into account the target platform's special properties when choosing cryptographic algorithms.

In many areas where cost and energy considerations dominate, computational power comes in the form of a small, inexpensive CPU. By a wide margin, 8-bit controllers have the largest share of the worldwide CPU market. These small microcontrollers are constrained in program memory (flash or ROM), RAM, clock speed, register width, and arithmetic capabilities. In this context, efficiency means more than simply throughput: Resources needed to implement a cipher should be kept small. In fact, in many situations, resource efficiency (measured mainly by memory consumption) is more critical than throughput, especially because many embedded applications encrypt only small payloads. Typically, these 8-bit microcontrollers offer as little as tens of kilobytes of program memory, and sometimes less than 1 Kbyte of SRAM; they usually operate at clock speeds of a few MHz.

Nevertheless, especially for battery-powered devices, low computational complexity can be of great value too because processing time directly correlates with power consumption. Modern microcontrollers can enter a variety of power-down and power-saving modes as soon as they have finished computation. Hence, a fast-executing algorithm can reduce energy consumption and lengthen the lifetime of a battery-powered device.

Table 1. Comparison of lightweight ciphers.

Cipher	Key bits	Block bits	Cycles per block	Throughput at 100 kHz (Kbps)	Logic process	Area (GEs)
Block ciphers						
Present	80	64	32	200.00	0.18 µm	1,570
AES	128	128	1,032	12.40	0.35 µm	3,400
Hight	128	64	34	188.20	0.25 µm	3,048
Clefia	128	128	36	355.56	0.09 µm	4,993
mCrypton	96	64	13	492.30	0.13 µm	2,681
DES	56	64	144	44.40	0.18 µm	2,309
DESXL	184	64	144	44.40	0.18 µm	2,168
Stream ciphers						
Trivium[5]	80	1	1	100.00	0.13 µm	2,599
Grain[5]	80	1	1	100.00	0.13 µm	1,294

*AES: Advanced Encryption Standard; DES: Data Encryption Standard; DESXL: lightweight DES with key whitening.

In this context, we compare the previously discussed ciphers that are primarily optimized for hardware with several software-friendly ciphers. For comparison, we added two software-oriented stream ciphers of the eStream project: Salsa20 and LEX.[6] The latter is a modified AES in which several bytes of the intermediate states are extracted and used as a key stream. Salsa20, like the Tiny Encryption Algorithm (TEA) and the International Data Encryption Algorithm (IDEA), is based on simple arithmetic operations. We chose these ciphers because their consumption of ROM and RAM is suited for small embedded processors. Stream ciphers usually have lengthy setup phases. LEX needs only one AES encryption for setup, and Salsa20's setup phase is even shorter. Hence, these ciphers can provide efficient encryption of the small payloads often found in embedded systems. All the discussed ciphers were implemented for 8-bit AVR microcontrollers. AVRs are a popular family of 8-bit RISC microcontrollers. The ATmega family offers 8 Kbytes to 128 Kbytes of flash memory and 1 Kbyte to 8 Kbytes of SRAM. The devices of the ATmega series have 32 general-purpose registers with a word size of 8 bits. Most of the microcontrollers' 130 instructions are one cycle, and the microcontrollers can be clocked at up to 16 MHz.

To keep the source code small, we used a straightforward approach for all our software implementations. Only the substitution tables are realized as lookup tables (LUTs), where applicable, because this provides an enormous speedup for reasonable memory consumption. Many fast software implementations of ciphers use larger LUTs to achieve a higher throughput. Unfortunately, this leads to an unacceptable increase in code size for many embedded applications. The LUTs are stored in the program memory (ROM). TEA, IDEA, and Salsa20 do not use substitution tables, which allows for a smaller code. For the inversion needed in IDEA, we used a slow but extremely small algorithm. This explains the small code as well as the huge discrepancy between encryption and decryption time.

The results of our implementations appear in Table 2. As expected, the software-oriented ciphers perform better on our platform. We had problems decreasing the code size of Hight, but it still shows good encryption performance. Although Present shows poor performance, its code, along with that of IDEA, is extremely small. LEX is a modified AES cipher, yet its code is smaller than that of AES because it lacks the decryption part.

Considering the trade-offs between security, cost, and performance, the stream ciphers seem to be a good choice. LEX and Salsa20 do well in both throughput and size, yet they are good choices only if the encrypted payload is sufficiently large. Otherwise, they produce a computational overhead because of their huge block length and their setup phase.

When code size is extremely critical, TEA, IDEA, and even Present seem to be reasonable choices. For most other cases, AES again shows its strength in software.

Asymmetric ciphers

Among public-key algorithms, there are three established families of practical relevance: ECC, RSA

Table 2. Comparison of software implementations of ciphers.

Cipher	Key size (bits)	Block size (bits)	Encryption (cycles/ block)	Throughput at 4 MHz (Kbps)	Decryption (cycles/ block)	Relative throughput (% of AES)	Code size (bytes)	SRAM size (bytes)	Relative code size (% of AES)
Hardware-oriented block ciphers									
DES	56	64	8,633	29.6	8,154	38.4	4,314	0	152.4
DESXL	184	64	8,531	30.4	7,961	39.4	3,192	0	112.8
Hight	128	64	2,964	80.3	2,964	104.2	5,672	0	200.4
Present	80	64	10,723	23.7	11,239	30.7	936	0	33.1
Software-oriented block ciphers									
AES	128	128	6,637	77.1	7,429	100.0	2,606	224	100.0
IDEA	128	64	2,700	94.8	15,393	123.0	596	0	21.1
TEA	128	64	6,271	40.8	6,299	53.0	1,140	0	40.3
SEA	96	96	9,654	39.7	9,654	51.5	2,132	0	75.3
Software-oriented stream ciphers									
Salsa20	128	512	18,400	111.3	NA	144.4	1,452	280	61.2
LEX	128	320	5,963	214.6	NA	287.3	1,598	304	67.2

*IDEA: International Data Encryption Algorithm; TEA: Tiny Encryption Algorithm; SEA: Scalable Encryption Algorithm.

(Rivest-Shamir-Adleman), and discrete logarithms. ECC is considered the most attractive family for embedded environments because of its smaller operand lengths and relatively lower computational requirements. ECC has been accepted commercially and has also been adopted by standardizing bodies such as the American National Standards Institute (ANSI), the IEEE, the International Organization for Standardization (ISO), the Standards for Efficient Cryptography Group (SECG), and the National Institute of Standards and Technology (NIST).

A lightweight elliptic-curve engine

Interest is growing in stand-alone asymmetric cipher engines in small, constrained devices for applications such as sensor networks and contactless smart cards (for example, e-passports). This interest is normally dictated by the need for better performance to satisfy a communication protocol or energy constraints.

Here, we present a hardware implementation of a low-area, stand-alone, public-key processor for standardized ECC curves, details of which can be found elsewhere.[7] We sacrifice flexibility to save area by setting the design to fit a specific standardized binary-field curve that is quite reasonable for constrained devices. Standardized binary fields that provide short-term security (113 bits), as well as fields that are required for medium-term security applications (193 bits), are implemented. For some constrained applications, 113-bit fields provide adequate security.

The main reason for choosing a binary field rather than a prime field is the carry-free arithmetic, which is well-suited for hardware implementations. A second reason is the simplified squaring structure, which is a central idea used in the algorithms chosen for the processor design.

Inversion. Itoh and Tsujii proposed the construction of an addition chain such that the inversion could be performed in $O(\log m)$ multiplications.[8] Although the algorithm was proposed for optimal normal-basis implementations, where squarings are almost free (cyclic rotations), the area requirement for the squaring structure in our implementation is within bounds but has the same timing efficiency of 1 clock cycle as in the normal basis. Our algorithm exploits the fact that squaring is very efficient in the standard basis as long as the field is fixed. It's easy to show that the inverse A^{-1} can then be obtained in $(\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1)$ multiplications and $(m-1)$ squarings using this addition chain, where H_w denotes the Hamming weight of the binary representation.

Point multiplication. We used a modified version of the Montgomery algorithm for implementing the point multiplication. We require one inversion and

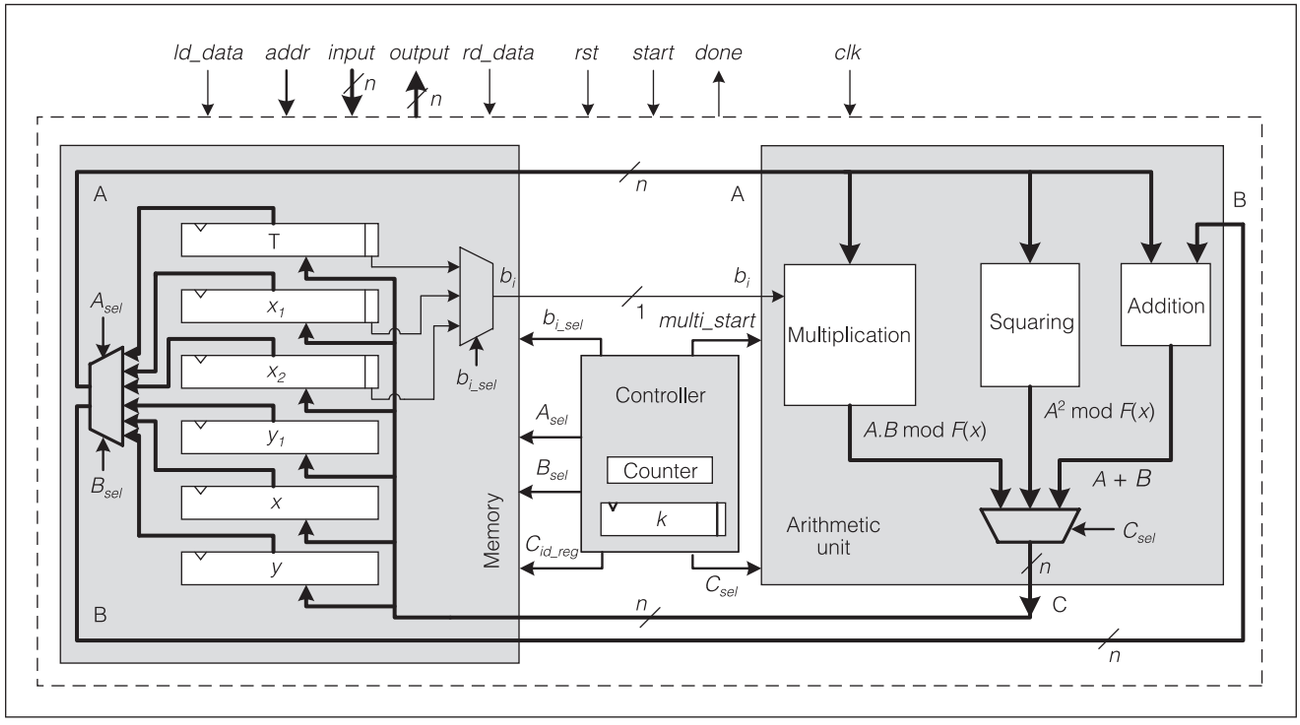

Figure 2. Area-optimized $GF(2^m)$ elliptic-curve cryptography (ECC) processor.

four multiplications in each iteration. The algorithm also lets us compute each iteration without requiring extra temporary memory locations, thus reducing area.

ECC processor design. The overall design appears in Figure 2. The three units—$GF(2^m)$ addition; $GF(2^m)$ multiplication, implemented as an MSB-first (most-significant-bit) multiplier; and $GF(2^m)$ squaring—are closely interconnected inside a single arithmetic unit sharing the common input data bus A. The adder needs an additional data bus B for the second operand, and the multiplier requires a single-bit b_i signal for the multiplicand. The operands are stored in the memory as registers (some of them as cyclic registers) with the output being selected for A, B, and b_i using multiplexers with control signals (A_{sel}, B_{sel}, and b_{i_sel}) from the controller. All the operand registers are connected in parallel to data bus C, with the appropriate register being loaded on the basis of the controller load signal C_{ld_reg}.

Inversion, as described, requires no additional hardware apart from the preexisting multiplying unit and squaring unit, with some additional control circuitry to enable loading the proper variables to the appropriate unit.

The implementation was synthesized for a custom ASIC design using AMI Semiconductor 0.35-micron CMOS technology. The designs have a total area ranging from 10 K GEs for a 113-bit field for short-term security to 18 K GEs for a 193-bit field for medium-term security applications. This implementation shows that an extremely small-area implementation of an ECC processor is possible in affine coordinates. Table 3 shows the ECC processor's total area in GEs and latency in clock cycles for a single scalar multiplication and compares them with those of other implementations.[9]

Hardware-software codesign for ECC

An ECC coprocessor, as we've defined it, can be small; it can also be prohibitively expensive for many pervasive applications and can be capable of performance that those applications don't need. Hardware assistance in the form of instruction set extensions (ISEs) is more favorable in such situations because the cost of extra hardware is quite low compared with that of a coprocessor. Here, we present an efficient ISE implementation for ECC that is a tightly coupled hardware and software codesign. As a first step, we used a software-only ECC implementation to identify the functional elements and code segments that would

Table 3. ECC processor performance for scalar multiplication.

Source	Field	Total area (GEs)	Technology (μm)	Frequency (MHz)	Time (ms)
This work	$GF(2^{113})$	10,113	0.35	13.560	14.4
	$GF(2^{131})$	11,970	0.35	13.560	18.0
	$GF(2^{163})$	15,094	0.35	13.560	31.8
	$GF(2^{193})$	17,723	0.35	13.560	41.7
Batina et al.	$GF(2^{67})^2$	12,944	0.25	0.175	2,390.0
	$GF(2^{131})$	14,735	0.25	0.175	430.0
Gaubatz et al.[10]	$GF(p_{100})$	18,720	0.13	0.500	410.5
Wolkerstorfer[11]	$GF(2^{191})$	23,000	0.35	68.500	6.7
Ötztürk et al.[12]	$GF(p_{166})$	30,333	0.13	20.000	31.9

provide efficiency gains if implemented as an ISE. Then, a hardware model of the new processor determined the effects of the new extension on such parameters as execution time, code size, and data RAM usage.

We used the AT94K family of field-programmable, system-level ICs as a development platform. This architecture integrates an AVR 8-bit microcontroller core and FPGA resources on a single chip. We chose the standardized 163-bit elliptic curve over $GF(2^m)$, as recommended by NIST and ANSI. We used scalar point multiplication over this curve for determining the benefits of the ISE.

The pure software implementation was done first. Table 4 includes the point arithmetic performance; it shows that $GF(2^m)$ multiplication is the most costly operation with respect to execution time and memory requirement. Moreover, in the point multiplication algorithms, field multiplications are extremely frequent and therefore constitute the bottleneck operation for ECC. A closer analysis of the multiplication block shows that most of the time was spent for load and store operations because the small number of registers available in the AVR processor could not hold the large operands. Therefore, a hardware extension for this functional block would potentially reduce the memory bottleneck and speed up the ECC.

We present a complete $GF(2^{163})$ multiplier as an ISE requiring the minimum possible area. We implemented two multiplier architectures that provide a trade-off between performance and the extra area requirement. The first is a 163 × 163 least-significant-bit-first multiplier. The multiplier requires 163 AND gates, 167 XOR gates, and 489 flip-flops. A 163 × 163 multiplication computes in 163 clock cycles, excluding data input and output. In our implementation, overhead from control and memory access leads to a total execution time of 313 clock cycles.

An additional trade-off between area and speed is possible using the second option, a digit-serial multipli-

Table 4. ECC scalar point multiplication performance at 4 MHz.

Field multiplier	Combinational logic blocks	Point multiplier	Time (s)	Code size (bytes)	Data RAM (bytes)	Precomputation (bytes)
Software multiplier		Binary	6.039	10,170	379	544
		NAF	5.031	10,654	379	
		Montgomery	4.140	8,208	358	
163 × 163 multiplier	245	Binary	0.264	2,936	294	0
		NAF	0.224	3,014	294	
		Montgomery	0.169	2,048	273	
163 × 163 4 digits	498	Binary	0.183	2,936	294	0
		NAF	0.115	3,014	294	
		Montgomery	0.113	2,048	273	

*NAF: nonadjacent form.

Table 5. Performance of ECC scalar multiplication in software.

Finite-field multiplication	Platform	Time (ms)	Time (10^3 cycles)
Binary multiplication, 160-bit prime field[14]	ATmega128L	810	6,480
Binary multiplication, 160-bit prime field[13]	ATmega128L	1,040	8,383
Binary multiplication, 134-bit OEF[15]	8,051	8,370	100,440
Window multiplication, 134-bit OEF[15]	8,051	1,830	21,960

*OEF: optimum extension field.

er. In a bit-serial multiplier, only one bit of the multiplicand is used in each iteration; however, in a digit-serial multiplier, multiple bits (equal to the digit size) are multiplied in each iteration. We use a digit size of 4, since it yields a good speedup without drastically increasing the area requirement. The total area for the multiplier is 652 AND gates, 684 XOR gates, and 492 flip-flops. A 163 × 163 multiplication with reduction requires 42 clock cycles. In our implementation, the control overhead results in a total of 193 clock cycles.

Our implementation results in Table 4 show that huge performance gains are possible in small 8-bit processors by introducing small amounts of extra hardware. The results show an increase in speed of one to two orders of magnitude for the ECC implementation. Hardware costs are in the range of 250 to 500 extra combinational logic blocks (CLBs). Also, code size and data RAM usage decrease. In an ASIC-based ISE, where the hardware is more tightly coupled, the control signal overhead can be considerably reduced, permitting better efficiency.

Software realization of ECC

Using a hardware-software codesign can substantially increase public-key performance with minimal area. However, in some situations public-key cryptography must be implemented purely in software because changes to the hardware aren't possible. Here, we describe an implementation that proves that public-key cryptography can indeed be used in low-end 8-bit processors to provide adequate security for low-end applications. We met these goals by leveraging the computational savings provided by ECC.

For the implementation, we chose Mica motes. They provide the 8-bit ATmega128L microcontroller, which comprises 128 Kbytes of in-system reprogrammable flash, 4 Kbytes of electrically erasable programmable ROM (EEPROM), and 4 Kbytes of internal SRAM. The microcontroller can handle up to 64 Kbytes of optional external memory space.

As we saw previously, the efficiency of the finite-field arithmetic, especially the field multiplication, determines an ECC's overall efficiency. Because asymmetric systems are as much as three orders of magnitude slower than symmetric systems, the main focus is on speed rather than code size. For the same reason, we implemented a curve with a fixed bit length (a standardized 160-bit curve chosen for security reasons). This seems to be a reliable trade-off between security and speed. We chose the curve secp160r1 (standardized by *Standards for Efficient Cryptography – SEC 2: Recommended Elliptic Curve Domain Parameters*, v1.0, Certicom Research, 2000) for our implementation.

ECC's core operation is the scalar multiplication $k \times P$, where k is an integer and P is a point on an elliptic curve. Depending on the protocol used, optimizations to the scalar multiplication are possible; for example, the elliptic-curve digital-signature algorithm (ECDSA) protocol doesn't require the y-coordinate. Not focusing on a specific protocol, we use more-general algorithms to speed up the curve arithmetic to make it reusable for many protocol-adapted curve implementations. Because optimizations in the prime field arithmetic will always improve the ECC system's performance, the main focus lies here.

Elliptic-curve arithmetic. The curve multiplication might be changed or modified for each protocol, so we implemented a simple binary left-to-right, double-and-add algorithm. We used Jacobian projective coordinates for the curve multiplication. This let us avoid frequent use of the inversion, which is costly in software. Only one inversion is needed at the end of the curve multiplication to transform back to affine coordinates. We used algorithms that mix affine and Jacobian projective coordinates to significantly increase speed. Storing one point in the SRAM requires 320 bits in affine coordinates and 480 bits in Jacobian projective coordinates. Hence, we traded SRAM for speed by applying mixed coordinates

Table 6. Overview of instructions used.

Operation	Clock cycles	Code size (bytes)	SRAM (bytes)
Multiplication	2,881	5,440	32 (+80)
Addition	168	530	16 (+60)
Subtraction	188	910	16 (+60)
Bisection	136	450	0 (+20)
Reduction	432	1,508	40 (+40)

(Jacobian projective and affine). One multiplication requires 8,383,488 clock cycles, translating to 1.04 seconds at 8 MHz. (Further implementation details are available elsewhere.[13]) In Table 5, we list several implementations for $k \times P$.

Prime field arithmetic. The prime field arithmetic must provide multiply, add, inversion, and reduction operations; subtract, halve, and square operations are optional. The latter three can also be realized by using the former operations, but with additional overhead. Hence, subtract and halve were also implemented. To keep the code size moderate, we did not implement the square operation.

Operations with the most potential for optimization are multiplication and reduction. Because the prime field is based on a pseudo-Mersenne prime, it's possible to reduce the prime field by mere shifts and adds. The prime field arithmetic is completely implemented in assembly, thereby minimizing code size and optimizing performance. The greatest computation cost lies in the 160-bit field multiplication. SRAM operations and the microcontroller's 8-bit multiply instruction both need two clock cycles, so we exclude precalculations. Furthermore, Karatsuba multiplication does not result in a good trade-off on this platform. In addition to the theoretical lower bound of 1,600 clock cycles, two bottlenecks occur here: SRAM access, because the operands do not fit completely in the registers; and carry handling, because the intermediate results need to be accumulated.

The hybrid multiplication[14] is a schoolbook variant, optimized for low SRAM access. The parameter d indicates how many registers are used to minimize SRAM access. Using more registers means fewer SRAM operations are required. We implemented the hybrid multiplication for $d = 5$, which requires about half of the microcontroller's registers. This way, the remaining registers can be used for efficient carry handling.[13] Table 6 shows the code size, the number of clock cycles, and the SRAM required for the various operations. The listed SRAM requirements are supplemented in parentheses by the memory needed for the operands. The code size of 5.44 Kbytes for the 160-bit multiplication is acceptable, as it is highly optimized for speed.

Established symmetric ciphers can be implemented with 3,400 GEs in hardware, and about 600 bytes of code in the software case. Specialized symmetric ciphers can further reduce the resource requirements of hardware implementations to as few as 1,570 GEs with reasonable performance. Contrary to common belief, stream ciphers do not offer a substantial advantage in resource-constrained applications for either hardware or software.

Asymmetric cryptography with a secure bit length realized in hardware still requires a significantly larger chip (at least 10,000 additional GEs) than symmetric cryptography, but it is already reasonably fast. Hardware-software codesign seems to produce the best trade-off between size and speed for many pervasive computing applications. Nevertheless, carefully optimizing algorithms in software enables microcontrollers to perform asymmetric operations in less than 1 second, and this is sufficient for some envisioned pervasive applications. ■

Acknowledgments

We thank Andrey Bogdanov, Lars Knudsen, Gregor Leander, Matt Robshaw, Yannick Seurin, and Charlotte Vikkelsoe for their contributions to the development of Present. We also thank Sören Rinne for his assistance with software implementations.

■ References

1. G. Leander et al., "New Lightweight DES Variants," *Proc. Fast Software Encryption* (FSE 07), LNCS 4593, Springer-Verlag, 2007, pp. 196-210.
2. A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher," *Proc. Workshop Cryptographic Hardware and Embedded Systems* (CHES 07), LNCS 4727, Springer, 2007, pp. 450-466.

3. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand," *IEE Proc*, vol. 152, no. 1, Oct. 2005, pp. 13-20.
4. S. Kumar et al., "Breaking Ciphers with COPACOBANA—A Cost-Optimized Parallel Code Breaker," *Proc. Workshop Cryptographic Hardware and Embedded Systems* (CHES 06), LNCS 4249, Springer, 2006, pp. 101-118.
5. T. Good and M. Benaissa, "Hardware Results for Selected Stream Cipher Candidates," Workshop Record State of the Art of Stream Ciphers (SASC 07), 2007, http://www.ecrypt.eu.org/stream/papersdir/2007/023.pdf.
6. G. Meiser et al., "Software Implementation of eSTREAM Profile I Ciphers on Embedded 8-bit AVR Microcontrollers," Workshop Record State of the Art of Stream Ciphers (SASC 07), 2007, http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/publications/conferences/sasc2007_117.pdf.
7. S. Kumar, *"Elliptic Curve Cryptography for Constrained Devices," doctoral dissertation*, Electrical Engineering and Information Sciences, Ruhr University Bochum: Germany, 2006.
8. T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Information and Computation*, vol. 78, no. 3, Sept. 1988, pp. 171-177.
9. L. Batina et al., "An Elliptic Curve Processor Suitable for RFID-Tags," Benelux Workshop Information and System Security (WISSec 06), 2006, http://eprint.iacr.org/2006/227.pdf.
10. G. Gaubatz et al., "State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks," *Proc. 3rd IEEE Int'l Conf. Pervasive Computing and Communications* (PERCOMW 05), IEEE CS Press, 2005, pp. 146-150.
11. J. Wolkerstorfer, "Scaling ECC Hardware to a Minimum," Cryptographic Advances in Secure Hardware (CRASH 2005), 2005, invited talk.
12. E. Öztürk, B. Sunar, and E. Savaş, "Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic," *Proc. 6th Int'l Workshop Cryptographic Hardware and Embedded Systems* (CHES 04), LNCS 3156, Springer-Verlag, 2004, pp. 92-106.
13. L. Uhsadel, A. Poschmann, and C. Paar, "Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes," *Proc. 4th European Workshop Security and Privacy in Ad hoc and Sensor Networks* (ESAS 07), LNCS 4572, Springer-Verlag, 2007, pp. 73-86.
14. N. Gura et al., "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," *Proc. 6th Int'l Workshop Cryptographic Hardware and Embedded Systems* (CHES 04), LNCS 3156, Springer, 2004, pp. 119-132.
15. A. Woodbury, D.V. Bailey, and C. Paar, "Elliptic Curve Cryptography on Smart Cards without Coprocessors," *Proc. 4th Working IFIP Conf. Smart Card Research and Advanced Applications* (CARDIS 00), Kluwer Academic, 2000, pp. 71-92.

Thomas Eisenbarth is a PhD candidate in the Department of Electrical Engineering at Ruhr University Bochum, where he is also a research assistant with the Communication Security Group of the Horst Görtz Institute for IT Security. His research interests include embedded security, efficient implementation of cryptographic algorithms, and physical security. Eisenbarth has an MS in electrical engineering and computer science from Ruhr University Bochum. He is a student member of the IEEE Computer Society and the International Association of Cryptologic Research (IACR).

Sandeep Kumar is a research scientist at Philips Research Europe. He contributed to the work described in this article while he was a research assistant working on his doctoral thesis at Ruhr University Bochum. His research interests include both hardware and software implementations on constrained devices of cryptographic systems—in particular, elliptic-curve cryptography—and his current focus is on hardware implementations of physically unclonable functions. Kumar has a PhD in electrical engineering and information sciences from Ruhr University Bochum. He is a member of the IACR.

Christof Paar is the chair of communication security in the Department of Electrical and Computer Engineering at Ruhr University Bochum. His research interests include physical security, cryptanalytical hardware, security in real-world systems, and efficient software and hardware implementations of cryptographic algorithms. Paar has a PhD in electrical engineering from the University of Essen. He is a member of the IEEE, the ACM, and the IACR.

Axel Poschmann is a PhD candidate in the Department of Electrical and Computer Engineering at Ruhr University Bochum, where he is also a research assistant with the Communication Security Group at the Horst Görtz Institute for IT Security. His research interests include lightweight cryptography, low-power and area-efficient implementations of cryptographic algorithms, and cryptography for wireless sensor networks. Poschmann has an MS in IT security from Ruhr University Bochum. He is a student member of the IEEE Computer Society and the IACR.

Leif Uhsadel is a PhD candidate with the Computer Security and Industrial Cryptography Group in the Department of Electrical Engineering at the Catholic University of Leuven. He contributed to the work described in this article while attending Ruhr University Bochum. His research interests include embedded security, efficient implementation of cryptographic algorithms, and side-channel attacks. Uhsadel has an MS in IT security from Ruhr University Bochum.

■ Direct questions and comments about this article to Axel Poschmann, Horst Görtz Institute for IT Security, Communication Security Group, Ruhr University Bochum, IC 4/134, Universitaetsstrasse 150, 44780 Bochum, Germany; poschmann@crypto.rub.de.