Implementation of Symmetric Algorithms on a Synthesizable 8-Bit Microcontroller Targeting Passive RFID Tags

Thomas Plos, Hannes Groß, and Martin Feldhofer

Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Thomas.Plos,Martin.Feldhofer}@iaik.tugraz.at,
Hannes.Gross@student.tugraz.at

Abstract. The vision of the secure Internet-of-Things is based on the use of security-enhanced RFID technology. In this paper, we describe the implementation of symmetric-key primitives on passive RFID tags. Our approach uses a fully synthesizable 8-bit microcontroller that executes, in addition to the communication protocol, also various cryptographic algorithms. The microcontroller was designed to fulfill the fierce constraints concerning chip area and power consumption in passive RFID tags. The architecture is flexible in terms of used program size and the number of used registers which allows an evaluation of various algorithms concerning their required resources. We analyzed the block ciphers AES, SEA, Present and XTEA as well as the stream cipher Trivium. The achieved results show that our approach is more efficient than other dedicated microcontrollers and even better as optimized hardware modules when considering the combination of controlling tasks on the tag and executing cryptographic algorithms.

Keywords: passive RFID tags, 8-bit microcontroller, symmetric-key algorithms, low-resource hardware implementation.

1 Introduction

Radio frequency identification (RFID) is the enabler technology for the future Internet-of-Things (IoT), which allows objects to communicate with each other. The ability to uniquely identify objects opens the door for a variety of new applications. Some of these applications like proof-of-origin of goods or privacy protection require the use of cryptographic algorithms for authentication or confidentiality.

An RFID system typically comprises three components: an RFID reader, a back-end database, and one or more RFID tags. The reader is connected to the back-end database and communicates with the tag contactless by means of an RF field. The so-called tag is a small microchip attached to an antenna that receives the data and probably the clock signal from the RF field. Passive tags also receive their power supply from the RF field. Supplying the tag from the RF field strongly limits the power consumption of the tag (typically, around 30 µW are available). Moreover, passive RFID tags are produced in high volume and need to be cheap in price. In order to keep the price of the tag low, its microchip must not exceed a certain size in terms of silicon area (typically, a whole tag has a size of 20000 gate equivalents). These two constraints make it a challenging task to implement cryptographic security on passive RFID tags.

For a secure system, not only the reader and the back-end database need to provide cryptographic security, but also the tags have to perform cryptographic operations. During the last years, a lot of effort was made by the research community to bring cryptographic security to RFID tags. The most prominent attempts among others are for example symmetric schemes like the Advanced Encryption Standard (AES) [11, 15], or asymmetric schemes like Elliptic Curve Cryptography (ECC) [2, 27]. All these attempts use dedicated hardware modules that are highly optimized for a specific cryptographic algorithm. Moreover, they do not consider the increased controlling effort that comes along with adding security to RFID tags. Even without adding security, a substantial part of the chip size is consumed by the controlling unit of the tag that handles the communication protocol. The work of Yu *et al.* [29] states that about 7500 gate equivalents (GEs) (one GE is the silicon area required by a NAND gate) are consumed for only handling the protocol.

Several tasks have to be accomplished by the control unit of a tag when using a simple challenge-response protocol to verify the authenticity of the tag. First, the control unit of the tag needs to generate random data and combine it with the challenge from the RFID reader. Second, random data and challenge need to be provided to the cryptographic hardware module and processing of data has to be started. Finally, the processed data needs to be transferred to the RFID reader. Other security services like mutual authentication or secure key update are even more demanding in terms of controlling effort.

Today's RFID tags have their control unit implemented as a finite-state machine (FSM) in hardware. However, increased controlling effort is easier to handle with a simple microcontroller. Particularly, microcontrollers are more flexible and allow faster integration of new functionalities, which reduces the time to market. Moreover, when using the microcontroller for control tasks, it seems reasonable to reuse it also for computing cryptographic algorithms. This reuse enables a better utilization of resources like the memory, which can help to reduce the chip size of the tag and in turn also the costs.

In this work, we present a synthesizable hardware implementation of a simple 8-bit microcontroller that is suitable to handle complex control tasks. The microcontroller fulfills the fierce requirements of passive RFID tags regarding power consumption and chip size. Five symmetric-key algorithms are implemented on this microcontroller, which are: the Advanced Encryption Standard (AES), the Scalable Encryption Algorithm (SEA), Present, the Extended Tiny Encryption Algorithm (XTEA), and Trivium. The performance of our implementations is evaluated and compared with results from implementations on other dedicated microcontroller platforms. Finally, the hardware costs that are added due to the implementation of the cryptographic algorithms are compared with the costs of stand-alone hardware modules. The results clearly show that the implementations on our microcontroller have lower costs than the dedicated hardware modules. For example, AES encryption and decryption comes at cost of less than 3000 GEs on our microcontroller.

The remainder of this paper is organized as follows. Section 2 describes the synthesizable 8-bit microcontroller that is used for implementing the cryptographic algorithms. In Section 3, a short overview of the selected algorithms is given, followed by the implementation results of the algorithms in Section 4. Discussion of the results with respect to passive RFID tags is done in Section 5. Conclusions are drawn in Section 6.

2 Description of the Synthesizable 8-Bit Microcontroller

The overhead that comes along with adding security to RFID tags in terms of controlling effort is often underestimated. Increased controlling effort is easier to accomplish with a simple microcontroller than with dedicated finite-state machines. In the following, we present a simple 8-bit microcontroller that is suitable to handle such complex control tasks. Moreover, it is a fully synthesizable microcontroller (available in VHDL) and it fulfills the fierce requirements of passive RFID tags.

Our 8-bit microcontroller uses a Harvard architecture with separated program memory and data memory. Such an architecture has the advantage that the program memory can have a different word size (16-bit) than the data memory (8-bit). The microcontroller supports 36 instructions and is a so-called Reduced Instruction Set Computer (RISC). The instructions have a width of 16 bits and can mainly be divided into three groups: logical operations like AND or XOR, arithmetic operations like addition (ADD) and subtraction (SUB), and control-flow operations like GOTO, CALL, and branching.

The main components of the microcontroller are the read-only memory (ROM), the register file, the program counter, the instruction decode unit, and the arithmetic-logic unit (ALU). An overview of the microcontroller is presented in Figure 1. The ROM contains the program of up to 4096 instructions and is realized as look-up table in hardware. It gets mapped to an unstructured mass of standard cells by the synthesis tool. Unused program space reduces the chip size of the microcontroller which makes our approach flexible and efficient. The register file is the data memory of the microcontroller and consists of at most 64 8-bit registers. If not all registers are needed by an application (during protocol execution and cryptographic calculations), unused registers can be omitted reducing the overall size of the microcontroller. This flexibility is only possible with a customizable processor architecture. Instructions are executed within a two-stage pipeline that consists of a fetch and a decode/execute step. In the first stage, the instruction that is addressed by the 12-bit program counter is loaded from the ROM into the instruction decode unit. In the second stage, the instruction is decoded by the instruction decode unit and executed by the ALU. Afterwards, the program counter

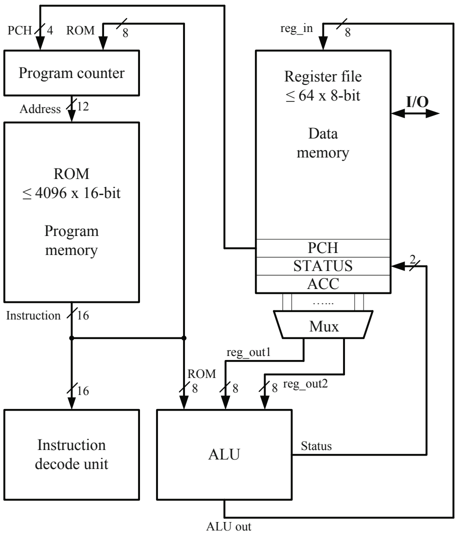
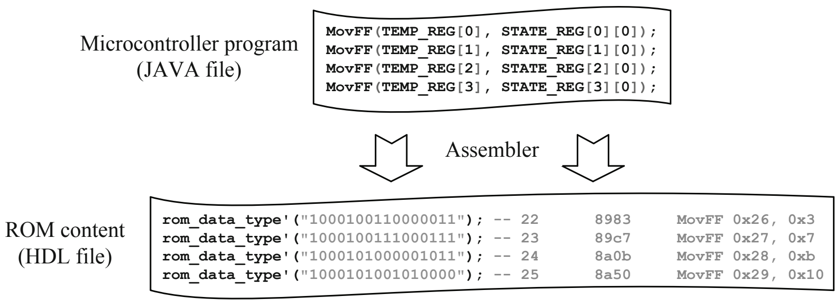


Fig. 1. Overview of the 8-bit microcontroller

is updated. The program counter contains a call stack that allows up to three recursive subroutine calls. All instructions are executed within a single clock cycle, except control-flow operations which require two clock cycles.

There are two types of registers in the register file of the microcontroller: special-purpose registers and general-purpose registers. The special-purpose registers involve an accumulator register (ACC) for advanced data manipulation, a status register (STATUS) that gives information about the status of the ALU (*e.g.* carry bit after addition), a program-counter register (PCH) for addressing the higher 4 bits of the program counter, and input/output (I/O) registers. The I/O registers are used for accessing external devices. In case of RFID tags, external devices can be the digital front-end for sending and receiving byte streams or the EEPROM. The I/O registers are also used for reacting on external events via busy waiting, since interrupts are not supported by the microcontroller. General-purpose registers are used for arbitrary data manipulations and temporarily storing data.

We have developed a self-written instruction-set simulator and an assembler for implementing the microcontroller program that is stored in the ROM. Both programs are written in JAVA and allow a fast and easy way of program development. The simulator supports a single-step mode and gives access to the internal state of the microcontroller. This makes debugging and testing of the program very convenient. After simulation, the assembler is used to generate a file with the content of the ROM. Figure 2 presents a code snippet of a microcontroller program written in JAVA with resulting ROM content after applying the assembler. The file with the ROM content is directly integrated into the hardware-description language (HDL) model of the microcontroller. This HDL



```
MovFF(TEMP_REG[0], STATE_REG[0][0]);
MovFF(TEMP_REG[1], STATE_REG[1][0]);
MovFF(TEMP_REG[2], STATE_REG[2][0]);
MovFF(TEMP_REG[3], STATE_REG[3][0]);
```

```
rom_data_type'("1000100110000011"); -- 22      8983     MovFF 0x26, 0x3
rom_data_type'("1000100111000111"); -- 23      89c7     MovFF 0x27, 0x7
rom_data_type'("1000101000001011"); -- 24      8a0b     MovFF 0x28, 0xb
rom_data_type'("1000101001010000"); -- 25      8a50     MovFF 0x29, 0x10
```

Fig. 2. Code snippet of a microcontroller program written in JAVA with resulting ROM content after applying the assembler

model is then synthesized with Cadence RTL Compiler for a 0.35 µm CMOS technology using a semi-custom design flow. The synthesis results in Table 1 show that the microcontroller can be implemented within less than 5600 gate equivalents (GEs), excluding the ROM. The by far largest part is the register file (for 64 x 8-bit) with 4582 GEs. As already mentioned, our approach allows to flexibly reduce the instantiated number of registers. Moreover, we performed power simulations with Synopsys Nanosim. The simulations show a mean power consumption of 10.3 µA for the microcontroller when operating at 100 kHz clock frequency and 3 V supply voltage. The power consumption can be further reduced by switching to a more recent CMOS technology or by reducing the supply voltage.

Both chip area and power consumption of the microcontroller fulfill the requirements of passive RFID tags. Hence, when using the microcontroller for handling the control tasks of the tag, it seems reasonable to reuse it also for computing the cryptographic algorithms. In order to address this issue, different cryptographic algorithms are implemented on the microcontroller. The following sections contain a short description of the selected cryptographic algorithms and evaluate their implementations with respect to execution time and code size.

Table 1. Synthesis results of the microcontroller excluding the ROM

Component	**Chip area**	
	[GEs]	[%]
Program counter with call stack	463	8.3
ALU	265	4.7
Register file (64 x 8 bit)	4582	81.9
Instruction decode unit	284	5.1
Total	**5594**	**100.0**

3 Overview of the Selected Cryptographic Algorithms

Several cryptographic algorithms that could be interesting for RFID applications were selected for implementation on the microcontroller. Main selection criteria were adequate security of the algorithm and moderate resource usage. We selected four block ciphers and one stream cipher for evaluation. The block ciphers are: AES, XTEA, Present, and SEA. The stream cipher is Trivium. Table 2 provides an overview of the selected algorithms and compares some of their properties like key size, block size, and number of rounds.

Table 2. Comparison of the selected cryptographic algorithms

Algorithm	**Key size** [bits]	**Block size** [bits]	**Number of rounds** per block
Block ciphers			
AES-128	128	128	10
$\text{SEA}_{96,8}$	96	96	93
Present-80	80	64	31
XTEA	128	64	64
Stream ciphers			/128 bits
Trivium	80	-	128

The Advanced Encryption Standard (AES) is the successor of the Data Encryption Standard (DES) and was introduced by the National Institute of Standards and Technology (NIST) in 2001 [21]. AES uses a so-called substitution-permutation network (SPN) and works on a fixed block size of 128 bits. We selected AES-128 for our implementations, which has a key length of 128 bits and uses 10 rounds. AES-128 is treated as very secure since it is widely deployed and well researched. The best known attack on AES-128 applies on 8 out of 10 rounds and was published in 2009 [14].

SEA stands for Scalable Encryption Algorithm and was developed by Standaert *et al.* in 2006 [25]. The algorithm is a so-called Feistel block cipher and was designed for resource-constrained devices such as microcontrollers that have only a limited instruction set and little memory available. As the name implies, SEA is scalable. This means that parameters like the word size b, the width n of the key and the plaintext, and the number of rounds n_r can be adjusted for the requirements of the target device. As suggested by the designers, we selected for implementation on our 8-bit microcontroller (word size b=8) n=96 bits and n_r=93 rounds ($\text{SEA}_{96,8}$). SEA is a rather new algorithm and not as well researched as other block ciphers. Hence, further analysis of its security is still necessary.

Present is another lightweight block cipher suitable for implementation on constrained devices. It uses a substitution-permutation network (SPN) like AES and was introduced by Bogdanov *et al.* in 2007 [3]. Present operates on 64-bit data blocks and supports key lengths of 80 bits (Present-80) and 128 bits (Present-128). In our implementations we selected Present-80 since it is most interesting for low-resource implementations. Present-80 uses 31 rounds. The best known attack on Present-80 applies on 26 out of the 31 rounds and was recently published by Cho in [6].

The Extended Tiny Encryption Algorithm (XTEA) was published in 1997 [22] and is the successor of the Tiny Encryption Algorithm (TEA). XTEA is a 64-bit block cipher with a Feistel structure that iterates a simple round function over a number of 64 rounds. The key used by XTEA has a length of 128 bits. Encryption and decryption operation of XTEA have a similar structure allowing a rather compact implementation of the block cipher. The best known attack on XTEA is a so-called related-key rectangle attack that addresses 36 rounds out of the 64 rounds [16].

Trivium is a hardware-oriented stream cipher developed by De Cannière *et al.* [4]. Although the stream cipher is optimized for hardware designs, it provides also low-resource usage in software implementations. Trivium follows a very simple design strategy and allows to generate up to 2^{64} bits of key stream from an 80-bit initial value (IV) and an 80-bit secret key. Trivium operates on a 288-bit internal state, which needs to be initialized first before generation of the key stream is started. There exist several attacks on reduced variants of Trivium as described in [8] and [28]. However, there is no successful attack against the full version of Trivium.

Implementation results of the five selected algorithms are presented in the next section, followed by comparing the results with implementations on other microcontroller platforms.

4 Implementation Results

This section presents the implementation results of the cryptographic algorithms on our microcontroller. All implementations were done in our own assembler environment and optimized towards three targets: execution time, code size, and efficiency. In order to determine the efficiency of an implementation, a scaling factor $s = 10^8$ is divided by the product of execution time in clock cycles and code size in bytes (s is used to obtain easier-manageable values). As mentioned in the previous section, we selected four block ciphers and one stream cipher for implementation. For the block ciphers both encryption and decryption were implemented. When the block ciphers need to compute round keys this is done on-the-fly during the encryption or the decryption routine. Moreover, we compare our results with implementations on other platforms like AVR microcontrollers from Atmel, PIC microcontrollers from Microchip, 68HC08 microcontrollers from Motorola, or 8051 microcontrollers from various manufacturers. The latter are based on a so-called Complex Instruction-Set Computer (CISC) architecture where the execution of a single instruction (a machine cycle) typically requires several clock cycles. This makes a comparison with 8051 microcontrollers difficult, since depending on the manufacturer, the number of clock cycles per instruction can vary. An overview of our implementation results is given in Table 3, which contains also results from implementations on the other microcontroller platforms.

4.1 AES-128

AES-128 was the first cryptographic algorithm implemented on our microcontroller. Round keys are computed on-the-fly. Encryption and decryption operation were implemented. Decryption consumes significantly more execution time than encryption, since the last round key need to be computed at the beginning. S-box operation and inverse S-box operation are realized as look-up tables with 256 entries each. The AES implementation with the best efficiency requires only 3304 clock cycles for encryption and only 5037 clock cycles for decryption (both values already include the key schedule). Code size of this version is 1940 bytes. A more compact implementation that extensively uses function calls saves 200 bytes of code, which comes at the prize of a significantly longer execution time, 5064 clock cycles for encryption and 8226 clock cycles for decryption. The speed-optimized version of AES uses code duplication and requires only 3084 clock cycles for encrypting a data block and 4505 clock cycles for decrypting a data block. This moderate speed up causes the code-size to increase to 2158 bytes. Regardless of the optimization target, 39 registers are used by our implementations.

AES-128 is widely deployed and many implementations of this algorithm for various microcontroller platforms are available. In Table 3 we list some of them, and compare them with our results. We also added two encryption-only versions of our implementations, one optimized for speed and one optimized for code size, to provide better comparability with related work where the decryption operation is omitted. When comparing with implementations on AVR or PIC microcontrollers, our versions are not only faster but also more compact in code size, leading to a much better efficiency. At first glance, the situation looks different for our encryption-only versions. There, the AES implementations on the 8051 microcontrollers seem to provide better efficiency. However, it has to be noted that the performance numbers of the 8051 microcontrollers are related to machine cycles (a machine cycle requires typically several clock cycles).

4.2 SEA

The simple structure of SEA allows a rather straight-forward implementation on the microcontroller. Encryption and decryption operation of SEA are quite similar and can efficiently be combined in a single function. In that way, a very compact implementation of SEA is obtained that requires only 332 bytes of code. Encryption or decryption of a 96-bit data block lasts 14723 clock cycles each with this version. When optimizing the implementation towards efficiency, execution time is reduced to 8597 clock cycles, by using 488 bytes of code. The speed-optimized implementation of SEA is only about 500 clock cycles faster and uses separate functions for encryption and decryption. However, this minor speed up increases the code size by nearly 300 bytes. All implementations of SEA utilize 12 registers each.

Table 3 gives an overview of the results and compares them with implementations on other microcontroller platforms. Our implementations have a

Table 3. Implementation results of the cryptographic algorithms and comparison with related work

Algorithm	Platform	Target	Code size	Encryption		Decryption	
			[bytes]	clock cycles	efficiency	clock cycles	efficiency
			Block ciphers				
AES-128	**This work**	**size**	**1704**	**5064**	**11.6**	**8226**	**7.1**
	This work	**eff.**	**1940**	**3304**	**15.6**	**5037**	**10.2**
	This work	**speed**	**2158**	**3084**	**15.0**	**4505**	**10.3**
	AVR [24]	-	3410	3766	7.8	4558	6.4
	AVR [10]	-	2606	6637	5.8	7429	5.2
	PIC [19]	-	2478	5273	7.7	7041	5.7
AES-128 (encr. only)	**This work**	**size**	**918**	**4192**	**26.0**	**-**	-
	This work	**speed**	**1110**	**3004**	**30.0**	**-**	-
	8051 [7]	-	1016	3168[1]	31.1[1]	-	-
	8051 [7]	-	826	3744[1]	32.3[1]	-	-
	8051 [7]	-	768	4065[1]	32.0[1]	-	-
	68HC98 [7]	-	919	8390	13.0	-	-
$SEA_{96,8}$	**This work**	**size**	**332**	**14723**	**20.5**	**14723**	**20.5**
	This work	**eff.**	**488**	**8597**	**23.8**	**8597**	**23.8**
	This work	**speed**	**786**	**8053**	**15.8**	**8053**	**15.8**
	AVR [24]	-	2132	9654	4.9	9654	4.9
	AVR [25]	-	386	17745	14.6	17745	14.6
	AVR [9]	-	834	9658	12.4	9658	12.4
	8051 [9]	-	604	8250[1]	20.1[1]	8250[1]	20.1[1]
Present-80	**This work**	**size**	**920**	**28062**	**3.9**	**60427**	**1.8**
	This work	**eff.**	**1148**	**15042**	**5.8**	**17677**	**4.9**
	This work	**speed**	**2146**	**8958**	**5.2**	**11592**	**4.0**
	AVR [23]	-	2398	9595	4.3	9820	4.2
	AVR [23]	-	1474	646166	0.1	634614	0.1
	AVR [10]	-	936	10723	10.0	11239	9.5
XTEA	**This work**	**size**	**504**	**17514**	**11.3**	**19936**	**10.0**
	This work	**eff.**	**820**	**7786**	**15.7**	**8928**	**13.7**
	This work	**speed**	**1246**	**7595**	**10.6**	**8735**	**9.2**
	AVR [24]	-	1160	6718	12.8	6718	12.8
	8051 [18]	-	542	6954[1]	26.5[1]	7053[1]	26.2[1]
	PIC [20]	-	962	7408	14.0	7408	14.0
	Stream ciphers			Initialization		/128 bits	
Trivium	**This work**	**size**	**332**	**85697**	**3.5**	**9488**	**31.7**
	This work	**eff.**	**726**	**40337**	**3.4**	**4448**	**31.0**
	This work	**speed**	**1226**	**39833**	**2.0**	**4112**	**19.8**
	AVR [1]	-	424	775726	0.3	85120	2.8

good efficiency, which is even better than on the 8051 microcontroller, whose execution time is indicated in machine cycles.

[1] The values for the 8051 microcontrollers refer to machine cycles. Typically, a machine cycle requires several clock cycles.

4.3 Present-80

Both encryption and decryption operation of Present-80 were implemented. Round keys are computed on-the-fly. As in case of AES-128, decryption operation requires significantly longer than encryption operation since the last round key needs to be computed at the beginning. The most compact implementation of Present-80 uses two look-up tables with 16 entries each, one for the S-box operation and one for the inverse S-box operation. This results in a code size of 920 bytes, allowing encryption of data within 28062 clock cycles, and decryption of data within 60427 clock cycles. A more efficient implementation uses two additional look-up tables with 16 entries each to speed-up the S-box and inverse S-box operation. This implementation requires 15042 clock cycles for encryption and 17677 clock cycles for decryption. Code size increases to 1148 bytes. The fastest version of Present-80 uses two big look-up tables with 256 entries each, performing the S-box operation on a whole byte. In that way, execution time reduces to 8958 clock cycles for encryption and 11592 clock cycles for decryption. Code size significantly increases to 2146 bytes. All versions of Present-80 require a total number of 30 registers each.

An overview of the implementation results of Present-80 is provided in Table 3. A comparison with implementation results on AVR devices shows that our speed-optimized version allows encryption within less clock cycles and that our code-size optimized version is even more compact. However, the implementation of [10] provides better efficiency.

4.4 XTEA

XTEA has a Feistel structure just like SEA. Thus, similar optimization strategies can be applied. Again, we implemented both encryption and decryption operation of the cipher. The most compact version needs 504 bytes of code and requires 17514 clock cycles for encryption and 19936 clock cycles for decryption. The efficiency-optimized version reduces the execution time to 7786 clock cycles for encryption and 8928 clock cycles for decryption. Code size is nearly doubled and increases to 820 bytes. The fastest version of XTEA uses code duplication and provides only a minor speed-up of about 200 clock cycles, while spending more than 400 bytes of additional code. The register usage of the XTEA implementations is between 23 registers for the most compact version, and 27 registers for the fastest version. This low values result from the simple structure of the key schedule.

Table 3 compares our results of XTEA with implementations on other 8-bit microcontroller platforms. The implementations on the AVR microcontroller and on the PIC microcontroller are a bit faster than our speed-optimized version. This is a consequence of the limited instruction set of our microcontroller, which prevents from efficiently adding 32-bit words. Nevertheless, our microcontroller achieves compact code size with slightly better efficiency for encryption.

4.5 Trivium

Trivium is the last algorithm that was implemented on the microcontroller. Although Trivium is a hardware-oriented design, it can be implemented in a very compact way in software. The code-size optimized version of Trivium generates on key-stream bit per iteration and requires only 332 bytes of code. Execution time results in 85697 clock cycles for initialization and 9488 clock cycles for generating 128 bits of key stream. The efficiency-optimized version generates 8 key-stream bits per iteration. This noticeably reduces execution time by doubling the code size. The speed-optimized version of Trivium generates 16 key-stream bits per iteration. Such an approach only slightly improves execution time by significantly increasing code size. All our versions of Trivium require 39 registers.

The results of Trivium are listed in Table 3 together with an implementation on an AVR microcontroller. The implementation of Trivium on the AVR microcontroller is done in C, leading to poor performance values compared to our versions.

4.6 Summary of Implementation Results

The results above clarify that our microcontroller allows implementing the selected cryptographic algorithms in a very compact and efficient way. Our implementations of AES-128, SEA, Present-80, and Trivium are faster than on the other compared 8-bit microcontroller platforms. Except in the case of AES, our implementations are also the most compact ones.

Comparing the performance numbers of the cryptographic algorithms shows that AES has by far the shortest execution time, but also requires most code size. When looking at code size, SEA and Trivium are the algorithms that can be implemented with minimum number of bytes. However, the initialization phase of Trivium takes exceptionally long. All these results are used in the next section to give actual values for the hardware costs that arise from implementing the algorithms on the microcontroller.

5 Discussing the Costs of Integrating the Implemented Algorithms on Passive RFID Tags

Two important constraints need to be considered when implementing cryptographic algorithms on passive RFID tags: power consumption and chip size. Power consumption affects the read range of the tag while chip size affects the costs of the tag. First, the power consumption of our microcontroller is more or less independent of the number of instructions present in the synthesized ROM. Thus, increasing the number of instructions by implementing cryptographic algorithms will not increase the power consumption. Second, the chip size of our microcontroller is not fixed, rather it is mainly defined by the size of the register file and by the size of the synthesized ROM. Depending on the application, the register file can contain up to 64 8-bit registers. These registers are already used by the microcontroller for handling the control tasks. Reusing them for computing cryptographic algorithms introduces no additional hardware costs. Hence, the only factor that influences the chip size of the microcontroller when implementing cryptographic algorithms is the code size. For this reason, we only consider the code size of the implemented algorithms as cost factor. Less attention is drawn on the execution speed of the algorithms, since RFID tags typically have enough time for the computations and only need to handle little data.

Actual values for chip-size increase were determined by implementing the cryptographic algorithms on our microcontroller platform. These values are obtained by synthesizing the program code of the implementations described in the previous section. Synthesis was done for a 0.35 µm CMOS technology using a semi-custom design flow with Cadence RTL Compiler. Table 4 presents the synthesis results, by bringing code size of each implementation in relation with chip area. Code size is given in terms of bytes and chip area is given in terms of gate equivalents (GEs). The chip area of the implementations ranges from 745 GEs for the code-size optimized version of Trivium to 3273 GEs for the speed-optimized version of AES.

Looking at the synthesis results brings up an interesting observation that concerns the area efficiency of the implemented algorithms. The area efficiency in terms of bits per GE is not constant but strongly varies and mainly depends on two factors. First, the area efficiency is improved when the code size of an implementation increases. For example, the speed-optimized version of AES with 2158 bytes of code has an area efficiency of 5.3 bits/GE, but the code-size optimized version of Trivium with 332 bytes of code has only 3.6 bits/GE. This varying area efficiency is caused by the synthesis tool, which can better optimize larger look-up tables. However, it is not intended that the algorithm implementations are used on their own, but together with the implementation of the communication protocol. This leads to a larger overall code size, which finally improves the area efficiency. Second, implementations with a lot of redundancy in the code reach even a much better area efficiency. An example for such an implementation is the speed-optimized version of Present which reaches 8.0 bits/GE. In this version, the 4-bit S-box is replicated 16 times to achieve faster execution of the algorithm (code duplication). Although this replication significantly increases code size, the chip area is only moderately increased since the synthesis tool removes redundancies in the resulting look-up table.

Table 4 gives not only an overview of the synthesis results, but also compares them with the area requirements of stand-alone hardware modules. In almost all cases, the hardware modules require more chip area than the implementations on our microcontroller (only code size is treated as cost factor since the register file is reused). Even the speed-optimized versions, which have the highest area requirements are smaller. The hardware implementation of Present-80 is the only exception. It consumes about 300 GEs less than the most compact version on the microcontroller. Looking at the results of AES shows that an implementation on the microcontroller supporting encryption and decryption can be realized within

Table 4. Synthesis results of the algorithm implementations on the microcontroller and comparison with dedicated hardware modules

Algorithm	Platform	Target	Code size	Area	Area efficiency
			[bytes]	[GEs]	[bits/GE]
Block ciphers					
AES-128	**This work**	**size**	**1704**	**2911**	**4.7**
	This work	**efficiency**	**1940**	**3130**	**5.0**
	This work	**speed**	**2158**	**3273**	**5.3**
	Feldhofer [13]	-	-	3400	-
AES-128 (encr. only)	**This work**	**size**	**918**	**1755**	**4.2**
	This work	**speed**	**1110**	**1871**	**4.7**
	Hämäläinen [15]	-	-	3100	-
$SEA_{96,8}$	**This work**	**size**	**332**	**786**	**3.4**
	This work	**efficiency**	**488**	**1083**	**3.6**
	This work	**speed**	**786**	**1619**	**3.9**
	Mace [17]	-	-	3758	-
Present-80	**This work**	**size**	**920**	**1399**	**5.3**
	This work	**efficiency**	**1148**	**1763**	**5.2**
	This work	**speed**	**2146**	**2139**	**8.0**
	Poschmann [23]	-	-	1075	-
XTEA	**This work**	**size**	**504**	**1230**	**3.3**
	This work	**efficiency**	**820**	**1718**	**3.8**
	This work	**speed**	**1246**	**2507**	**4.0**
	Feldhofer [12]	-	-	2636	-
Stream ciphers					
Trivium	**This work**	**size**	**332**	**745**	**3.6**
	This work	**efficiency**	**726**	**1476**	**3.9**
	This work	**speed**	**1226**	**2228**	**4.4**
	Feldhofer [12]	-	-	2390	-

less than 3000 GEs. This is even less area than the smallest encryption-only AES hardware module consumes.

Particularly for AES there exist several other approaches that try to minimize the costs of implementing the algorithm on a microcontroller. For example, microcontrollers with AES-specific design like the AESMPU [5] or microcontrollers with instruction-set extensions (ISE) [26]. Although both examples only need about half the code size of our AES implementations they are less flexible. The lack of flexibility comes from the AES-specific hardware parts that are used by both approaches. These parts need to be removed (redesign of the microcontroller on HDL level necessary) when implementing other cryptographic algorithms. Otherwise, the AES-specific parts will unnecessarily increase the chips size of the microcontroller. Moreover, the AESMPU is not designed for low-resource usage since it precomputes all round keys and stores them in its internal memory (176 8-bit registers required). This enormous memory usage makes the AESMPU inapplicable for passive RFID tags. ISE are more attractive than the AESMPU in terms of resource usage.

The ISE consume about 1100 GEs for the AES-specific hardware parts and require 840 bytes of code for implementing encryption and decryption. When assuming an area efficiency of 3.7 bits/GE (realistic for 840 bytes), the ISE will end up with roughly the same hardware costs as our most compact AES implementation. When using a more optimistic value of 4.8 bits/GE, the size of the ISE approach will be about 400 GEs smaller. This is not that much compared to the overall size of the AES implementation, but comes at cost of less flexibility. Moreover, the AES-specific hardware parts will slightly increase the overall power consumption of the microcontroller. Nevertheless, ISE are much faster. They allow to encrypt or decrypt a block of data within less than 1500 clock cycles. Thus, when the execution time is an important factor for an application, using ISE is beneficial. Due to the flexibility of our synthesizable microcontroller, it should not be too much effort to integrate also ISE if required in order to achieve an additional speed up.

The results of our algorithm implementations let us come to two important conclusions. First, our microcontroller platform that is mainly intended for simple control tasks on RFID tags, allows also to efficiently implement cryptographic algorithms like AES, Present, or XTEA. Second, the additional hardware costs that are introduced by implementing cryptographic algorithms on our synthesizable microcontroller are in almost all cases lower (except in case of Present) than by using stand-alone hardware modules. The additional hardware costs are only affected by the code size of the algorithm. The data memory in the register file is already used by the microcontroller for handling control tasks and will not result in additional hardware costs (Note that this statement is only correct in an environment where the microcontroller is used anyway and the question how much does security cost arises). This makes our synthesizable microcontroller a resource saving and flexible concept to bring cryptographic security to passive RFID tags.

6 Conclusion

In our work, we showed a very efficient concept of reusing a dedicated 8-bit microcontroller for the implementation of symmetric-key algorithms. The microcontroller, which is highly optimized for controlling tasks like protocol execution, is synthesizable and optimized concerning low chip area and low power consumption. It is also flexible concerning the program-memory size and the number of used registers. We evaluated the block ciphers AES, SEA, Present and XTEA as well as the stream cipher Trivium with respect to program size and required number of clock cycles. Our findings clearly show that the implemented microcontroller is more efficient than other dedicated microcontrollers and outperforms even optimized hardware modules when considering the reuse of the microcontroller for protocol execution tasks.

Acknowledgements

This work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems under the Project Number 820843 (Project CRYPTA) and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

[1] AVR-Crypto-Lib, http://www.das-labor.org/wiki/AVR-Crypto-Lib/en
[2] Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: Public-Key Cryptography for RFID-Tags. In: Workshop on RFID Security 2006 (RFIDSec 2006), July 12-14, Graz, Austria (2006)
[3] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurinand, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007) ISBN 978-3-540-74734-5
[4] Canniére, C.D., Preneel, B.: TRIVIUM Specifications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030 (April 2005), http://www.ecrypt.eu.org/stream
[5] Chia, C.-C., Wang, S.-S.: Efficient Design of an Embedded Microcontroller for Advanced Encryption Standard. In: Proceedings of the 2005, Workshop on Consumer Electronics and Signal Processing, WCEsp 2005 (2005), http://www.mee.chu.edu.tw/labweb/WCEsp2005/96.pdf
[6] Cho, J.Y.: Linear cryptanalysis of reduced-round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
[7] Daemen, J., Rijmen, V.: AES proposal: Rijndael. First AES Conference (August 1998)
[8] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
[9] EFTON s.r.o. Implementing SEA on x51 and AVR, http://www.efton.sk/crypt/sea.htm
[10] Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., Uhsadel, L.: A Survey of Lightweight-Cryptography Implementations. IEEE Design & Test of Computers - Design and Test of ICs for Secure Embedded Computing 24(6), 522–533 (2007) ISSN 0740-7475
[11] Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
[12] Feldhofer, M., Wolkerstorfer, J.: Hardware Implementation of Symmetric Algorithms for RFID Security. In: RFID Security: Techniques, Protocols and System-On-Chip Design, pp. 373–415. Springer, Heidelberg (2008)
[13] Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES Implementation on a Grain of Sand. IEE Proceedings on Information Security 152(1), 13–20 (2005)
[14] Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. Cryptology ePrint Archive, Report 2009/531 (2009), http://eprint.iacr.org/

[15] Hämäläinen, P., Alho, T., Hännikäinen, M., Hämäläinen, T.D.: 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, August 30-September 1, pp. 577–583. IEEE Computer Society, Los Alamitos (2006)

[16] Lu, J.: Related-key rectangle attack on 36 rounds of the XTEA block cipher. International Journal of Information Security 8, 1–11 (2009)

[17] Mace, F., Standaert, F.-X., Quisquater, J.-J.: ASIC Implementations of the Block Cipher SEA for Constrained Applications. In: Munilla, J., Peinado, A., Rijmen, V. (eds.) Workshop on RFID Security 2007 (RFIDSec 2007), Malaga, Spain, July 11-13, 2007, pp. 103–114 (2007)

[18] Pavlin, M.: Encription Using Low Cost Microcontrollers, `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.5755/&rep=rep1&type=pdf`

[19] Microchip Technology Inc. AN821: Advanced Encryption Standard Using the PIC16XXX (June 2002), `http://ww1.microchip.com/downloads/en/AppNotes/00821a.pdf`

[20] Microchip Technology Inc. AN953: Data Encryption Routines for PIC18 Microcontrollers (January 2005), `http://ww1.microchip.com/downloads/en/AppNotes/00953a.pdf`

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

[22] Needham, R.M., Wheeler, D.J.: Tea extensions. Technical report, Computer Laboratory, University of Cambridge (October 1997)

[23] Poschmann, A.Y.: Lightweight Cryptography - Cryptographic Engineering for a Pervasive World. PhD thesis, Faculty of Electrical Engineering and Information Technology, Ruhr-University Bochum,Germany (Februrary 2009)

[24] Rinne, S., Eisenbarth, T., Paar, C.: Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers (June 2007), `http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/publications/conferences/lw_speed2007.pdf`

[25] Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: a Scalable Encryption Algorithm for Small Embedded Applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006)

[26] Tillich, S., Herbst, C.: Boosting AES Performance on a Tiny Processor Core. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 170–186. Springer, Heidelberg (2008)

[27] Tuyls, P., Batina, L.: RFID-Tags for Anti-counterfeiting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 115–131. Springer, Heidelberg (2006)

[28] Vielhaber, M.: Breaking one.fivium by aida an algebraic iv differential attack. Cryptology ePrint Archive, Report 2007/413 (2007), `http://eprint.iacr.org/`, `http://eprint.iacr.org/`

[29] Yu, Y., Yang, Y., Yan, N., Min, H.: A Novel Design of Secure RFID Tag Baseband. In: RFID Convocation, Brussels, Belgium (March 14, 2007)