Analyzing the Hardware Costs of Different Security-Layer Variants for a Low-Cost RFID Tag*

Thomas Plos 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

Abstract. Radio-frequency identification (RFID) technology is the enabler for the future Internet of Things (IoT) where security will play an important role. In this work, we evaluate the costs of adding different security-layer variants that are based on symmetric cryptography to a low-cost RFID tag. In contrast to related work, we do not only consider the costs of the cryptographic-algorithm implementation, but also the costs that relate to protocol handling of the security layer. Further we show that using a tag architecture based on a low-resource 8-bit microcontroller is highly advantageous. Such an approach is not only flexibility but also allows combining the implementation of protocol and cryptographic algorithm on the microcontroller. Expensive resources like memory can be easily reused, lowering the overall hardware costs. We have synthesized the security-enabled tag for a 130 nm CMOS technology, using the cryptographic algorithms AES and NOEKEON to demonstrate the effectiveness of our approach. Average power consumption of the microcontroller is 2 µW at a clock frequency of 106 kHz. Hardware costs of the security-layer variants range from about 1100 GEs using NOEKEON to 4500 GEs using AES.

Keywords: Low-cost RFID tag, 8-bit microcontroller, AES, NOEKEON, security layer, low power consumption.

1 Introduction

Over the last years, radio-frequency identification (RFID) technology has found its way into many applications of our daily life. The integration of RFID functionality into the latest smart phones (*e.g.* Nexus S, Blackberry Bold 9900) emphasizes the relevance of this technology. An upcoming application that relies on RFID technology is the Internet of Things (IoT). The vision of the future IoT is that every object has communication capabilities by equipping it with

* 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 European Commission through the ICT programme under contract ICT-2007-216676 (ECRYPT II).

M. Rajarajan et al. (Eds.): SecureComm 2011, LNICST 96, pp. 426–435, 2012.

an RFID tag. An important aspect of the IoT is security [5]. Equipping every object with a tag presumes that they are cheap in price, making the integration of security a challenging task.

A typical RFID system mainly consists of three components: a back-end database, a reader, and one or more tags. The reader is connected to the back-end database and communicates with the tags contactlessly by means of a radio-frequency (RF) field. A tag is a small microchip attached to an antenna that receives its data and probably the clock signal from the RF field emitted by the reader. So-called passive tags also receive their power from the RF field.

The emergence of the IoT will not only pave the way for new applications but will also require to have additional functionality available on the tags. Such additional functionality comprises for example file management and security features, which increases the control complexity on the tag. Today's RFID tags use state machines fixed in hardware for handling their control tasks. As soon as the control complexity increases, the state-machine approach is no longer practical and even inefficient. Using a microcontroller approach instead that is more flexible seems to be favorable [13,14]. Having a microcontroller on the tag for handling the control tasks, allows reusing it for computing cryptographic algorithms that are necessary for the security features.

Our contribution in this paper is twofold and deals with the integration of security on low-cost RFID tags. Firstly, we analyze the benefits of having a combined implementation of protocol handling and cryptographic algorithm on a microcontroller. We demonstrate this by using a synthesizable 8-bit microcontroller that is optimized for low-resource usage. Secondly, we define three different security-layer variants using the block ciphers AES and NOEKEON and evaluate the hardware costs introduced by them. In contrast to related work, not only the costs of the cryptographic-algorithm implementation alone are considered, but also the costs that arise from protocol handling of the security layer. Our results underline that protocol handling constitutes a significant cost factor and must not be neglected.

The remainder of this paper is structured as follows. In Section 2 we present a system overview of our low-cost tag. Section 3 gives details about the deployed security-layer variants and Section 4 describes the concept for realizing them on the tag. The implementation results are provided in Section 5. Conclusions are drawn in Section 6.

2 System Overview

RFID tags consist of a small microchip attached to an antenna. The microchip contains an analog front-end and a digital part. Complexity of the digital part ranges from simple state machines with a small EEPROM for storing its unique identifier (UID), to contactless smart cards with powerful microcontrollers and special coprocessors. Powerful microcontrollers as they are found in contactless smart cards are not suitable for our low-cost tag. They consume too much power and require too much hardware resources. Hence, we are using a self-designed

8-bit microcontroller for our tag that is optimized for low-resource usage. A preliminary version of the microcontroller has been published in [12].

Our tag uses the ISO14443 standard for communication and operates in the high frequency range at a carrier frequency of 13.56 MHz. Low-level functionality is implemented according to ISO14443-3 [6]. High-level functionality is implemented according to ISO14443-4 [9] and uses a block-transmission protocol for exchanging application data as specified in ISO7816-4 [7]. The digital part of our tag mainly consists of five components: the framing logic, the low-resource 8-bit microcontroller, the bus arbiter, the EEPROM, and the true-random number generator (TRNG). The framing logic is connected to the analog front-end and provides a byte interface to the microcontroller. Low-level commands that are time critical are directly handled by the framing logic, commands on higher level are forwarded to the microcontroller. The 8-bit microcontroller is the central part of our security-enabled tag and controls all other components of the digital part through an Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB) [1]. The APB is managed by the bus arbiter.

High-level protocol functionality of the tag, including commands for security and file-management operations, as well as the cryptographic algorithm itself are entirely implemented in the program memory of the microcontroller. Hence, there is no dedicated coprocessor that handles encryption or decryption of data as typically found in the design of security-enabled tags. Random data that is required for security operations is generated within the TRNG and transferred to the memory of the microcontroller over the APB. The EEPROM is divided into files and is used for storing configuration data of the tag, the UID, the cipher key, and user data. Files are handled through file-management operations that allow selecting a file, reading from a file, or writing to a file. Depending on the file, different access rights are granted.

3 Security Layer

Two security services have been selected for implementation on our tag to quantify the costs of adding security functionality. The two security services are tag authentication and reader authentication. Tag authentication ensures originality of the tag to prevent simple cloning of. Reader authentication ensures originality of the reader to restrict access to certain resources on the tag.

The security services are based on a challenge-response protocol using symmetric cryptography as defined in ISO9798-2 [8]. We have selected two different cryptographic algorithms for the security services: AES [11] and NOEKEON [2]. Both algorithms are block ciphers with a block size of $n = 128$ bits. Selecting two different block ciphers allows analyzing their influence on the overall implementation costs. AES has been chosen because it is standardized and provides high security. NOEKEON has been selected since it provides a good trade off between security and resource usage (encryption and decryption function of NOEKEON can be implemented with very little overhead). Using symmetric cryptography requires that reader and tag share a secret key K. The key can be stored on

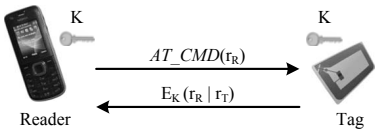

Fig. 1. Tag authentication

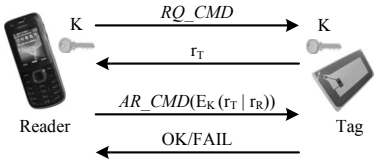

Fig. 2. Reader authentication

the tag, for example, during a personalization phase that is performed within a protected environment (*i.e.* it can be assumed that there is no adversary).

Tag Authentication. The basic principle of tag authentication is illustrated in Figure 1. The reader sends a randomly-selected challenge r_R with a length of $\frac{n}{2}$ bits through a tag-authenticate command (*AT_CMD*) to the tag. After receiving r_R from the reader, the tag generates itself a random number r_T of the same length, and encrypts the concatenation of the two random numbers $r_R \mid r_T$ under the secret key K. The encrypted value is then sent to the reader, which can decrypt it with its secret key. If both reader and tag use the same secret key, the decrypted value will contain the random number r_R that has initially been selected by the reader, and the tag is treated as authentic.

Reader Authentication. The second security services is reader authentication, which is depicted in Figure 2. The reader sends a request command (*RQ_CMD*) to the tag, which in turn generates a random number r_T with a length of $\frac{n}{2}$ bits that is transmitted to the reader. It is important to note that the tag has to store r_T internally to be able to verify later whether the reader is authentic or not (consumes $\frac{n}{2}$ bits of memory). After receiving r_T the reader generates its own random number r_R (also with a length of $\frac{n}{2}$ bits), and encrypts the concatenation of the two random values $r_T \mid r_R$ (position of random numbers is interchanged compared to tag authentication) using its secret key K. As next step, the encrypted value is transmitted through a reader-authenticate command (*AR_CMD*) to the tag, which decrypts the value using its secret key. When both reader and tag use the same secret key K, the decrypted value will contain the random number r_T initially selected by the tag, and the reader is treated as authentic. Alternatively, the reader can also decrypt $r_T \mid r_R$ instead of encrypting it. This has the advantage that the tag only needs to support encryption and not encryption and decryption, which makes for block ciphers like AES a significant difference in terms of resource usage. The tag finalizes the authentication step by sending a message to the reader with the status of the authentication process (*OK* or *FAIL*).

Security-Layer Variants. For a detailed analysis of the costs caused by adding a security layer to our tag, three security-layer variants are considered. The first variant (named *Variant 1* in the following) only supports tag authentication. Thus, the tag needs to implement the encryption function of the block cipher

and to handle one additional command. This is the least-expensive scenario. The second variant (*Variant 2*) realizes both services tag authentication and reader authentication. For reader authentication, the alternative method previously described is used, where the reader decrypts the value $r_T \mid r_R$. Hence, the tag needs only to implement the encryption function of the block cipher. Three additional reader commands have to be handled by the tag and memory for storing r_T inside the tag has to be provided. The third security-layer variant (*Variant 3*) is the most expensive one. Tag and reader authentication are supported. As in case of *Variant 2*, three additional reader commands need to be handled and memory inside the tag has to be reserved for storing r_T. However, the important difference to *Variant 2* is that the reader-authentication approach is used that requires the tag to support also the decryption function of the block cipher. In order to prevent potential attacks on protocol level such as reader-impersonation, every tag should use a different secret key K. Further, the tag accepts an *AR_CMD* only if it directly follows a *RQ_CMD* (*i.e.* using an *AT_CMD* after the *RQ_CMD* aborts the reader-authentication process).

4 Concept for Implementing the Security-Layer Variants

The way we implement the security-layer variants on our tag differs from the traditional approach typically found in related work, where protocol handling and cryptographic algorithm are implemented separately. There, protocol handling is implemented in a control state machine fixed in hardware and the cryptographic algorithm is implemented within a coprocessor that is highly optimized for low-resource usage. A schematic view of this approach is given in Figure 3. As already shown in various publications, for example in the work of Yan *et al.* [13] and Yu *et al.* [14], using a programable controller for handling complex control tasks on RFID tags is advantageous. The design becomes more flexible, easier to maintain, and faster to adapt.

Our tag uses also a programable approach for handling the complex parts of the protocol (high-level protocol). Complex parts of the protocol include for example: reconstructing chained reader commands, handling file-access commands, and managing configuration-parameters of the tag. Moreover, when adding a security layer, control complexity further increases. Generation of random values has to be triggered and the values have to be transferred to concerning locations in memory. Encryption and decryption of data has to be initiated and results have to be checked. Combining the security layer with existing tag functionality like handling file-access commands and managing configuration parameters also increases control complexity. Hence, we only use a fixed state machine in hardware (called framing logic) for time-critical commands that require low control complexity (low-level protocol) and whose functionality is typically fixed. Complex protocol parts are processed by an 8-bit microcontroller optimized for low-resource usage, which can be reused for computing cryptographic algorithms as well. A schematic view of this combined approach is presented in Figure 4. The program code of the microcontroller contains both the implementation of

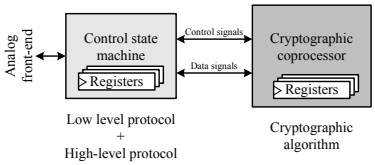

Fig. 3. Traditional approach where protocol handling and cryptographic algorithm are implemented separately

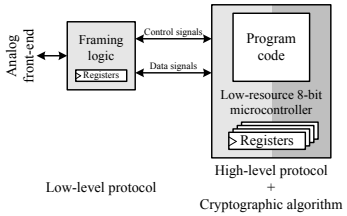

Fig. 4. Combined approach where high-level protocol and cryptographic algorithm are handled by a low-resource microcontroller

the high-level protocol and the cryptographic algorithm. Another benefit of this combined approach is the easier and more efficient reuse of costly resources like memory (registers of the microcontroller).

5 Implementation Results

We have implemented the three security-layer variants previously described using the block ciphers AES and NOEKEON, respectively. For each block cipher, various versions with different optimization targets are used. Implementation results are given for a 130 nm CMOS process technology [3] after place and route using Cadence RTL compiler and involve all components of the tag's digital part excluding TRNG and EEPROM.

Central element of our tag is a synthesizable 8-bit microcontroller optimized for low-resource usage. The microcontroller is based on a Harvard architecture using an 8-bit wide data memory (register file) and a 16-bit wide program memory (program ROM). Depending on the targeted application, up to 64 registers can be included into the register file (specified during synthesis). The program ROM is realized as look-up table and contains the instructions that the microcontroller should execute. Size of the program ROM is also flexible and can be at maximum 128 kB. Synthesizing the microcontroller core (control unit, program counter, and arithmetic-logic unit (ALU)) without register file and program ROM for a 130 nm process technology results in a chip area of 1067 GEs. A preliminary version of the microcontroller has been published in [12] to which we refer for more details.

5.1 Implementation Results of AES and NOEKEON

The two block ciphers AES and NOEKEON have been used for realizing the security-layer variants described in Section 3. For each cipher, three different optimization targets have been used: *fast*, *balanced*, and *small*. The target *fast* aims for shortest execution time of the cipher by using techniques like code duplication and loop unrolling, *balanced* provides a good trade off between execution

Table 1. Implementation results of the block ciphers AES and NOEKEON

Algorithm	Optimization target	Encryption	Decryption	Code size	Utilized registers
		[clock cycles]	[clock cycles]	[bytes]	-
AES	*fast*	3149	4570	2034	39
	balanced	3369	5101	1816	39
	small	5104	8286	1602	39
AES (encr. only)	*fast*	3070	n/a	1050	39
	small	4270	n/a	858	39
NOEKEON	*fast*	3817	3785	980	35
	balanced	5839	5824	532	25
	small	7563	7546	414	23
NOEKEON (encr. only)	*fast*	3805	n/a	652	35
	small	7553	n/a	382	23

time and code size, and *small* is optimized for minimal code size where as many operations as possible are handled through function calls that can be reused. Encryption function and decryption function of both ciphers are implemented. Moreover, for security-layer variants *Variant 1* and *Variant 2*, also encryption-only versions of the two algorithms are realized (with targets *fast* and *small*). Data that needs to be encrypted or decrypted is located in the register file of the microcontroller. The cipher key is stored in the EEPROM and has to be loaded each time during processing of data.

A summary of the implementation results is presented in Table 1. The AES implementations used in this work are similar to the ones published in [12]. In contrast to AES, NOEKEON requires only bit-wise Boolean operations and cyclic shifts which can be implemented with compact code size. No large look-up tables are required. We are using NOEKEON in indirect mode that applies an additional key schedule to increase resistance against related-key attacks. The key schedule in indirect mode can be precomputed, since the operation is independent of the processed data and all rounds use the same key. Hence, a lot of computation time can be saved when storing the precomputed working key in the EEPROM instead of the original cipher key.

5.2 Implementation Results of the Security-Layer Variants

Adding security to our tag influences mainly register-file size and ROM size of the microcontroller. For simplification, costs introduced by the TRNG and through storing additional data like the cipher key in the EEPROM are neglected. These costs are independent of the selected security-layer variant and the chosen block cipher.

Our tag with advanced file-management functionality utilizes 45 8-bit registers in the register file and 2214 bytes of code in the ROM for high-level protocol handling. Synthesizing the microcontroller with this configuration for our 130 nm target technology results in a chip size of roughly 9 kGEs (after place and route).

Only 9 of the 45 registers are permanently used for handling the protocol (*e.g.* to store status of tag and parameters). The remaining 36 registers are used for temporarily storing data (*e.g.* to reassemble chained reader commands) and can be reused when computing cryptographic algorithms. Since the computation of AES on our microcontroller requires 39 registers, only 3 additional registers are necessary when combining the computation of protocol and cryptographic algorithm. When using NOEKEON, no additional registers are necessary. Even the "largest" NOEKEON version consumes only 35 registers and fits within the 36 registers that can be reused from protocol handling.

When selecting a security layer based on *Variant 2* or *Variant 3* that involves reader authentication, additional registers are required for storing the random number r_T. Since r_T has a length of $\frac{n}{2} = 64$ bits, 8 additional registers are necessary. As a result, the total number of utilized registers increases to a maximum of 56 registers when reader authentication is supported and AES is used, and 53 registers when NOEKEON is used.

For determining the overall costs of the different security-layer variants, not only the size of the register file but also the size of the ROM has to be considered. ROM size is influenced by the security-layer variants through two parameters: the implementation of the block cipher and handling of the additional reader commands. Information about the code size of the different block-cipher implementations have already been given in Section 5.1. The required code size for handling the additional reader commands depends on the security-layer variant and ranges from 250 bytes for *Variant 1* to 460 bytes for *Variant 2*.

Synthesizing our tag with the different security-layer variants for a 130 nm process technology gives actual numbers about the area requirements in hardware. The register file of the microcontroller is built up with latches to minimize chip area. The ROM of the microcontroller is implemented as look-up table which gets mapped by the synthesis tool to an unstructured mass of standard cells. Detailed synthesis results after place and route obtained with Cadence RTL compiler are provided in Table 2. The least-expensive security-layer variant, which is *Variant 1* with the code-size optimized version of NOEKEON, results in an area overhead of 1074 GEs. The most-expensive security-layer variant, which is *Variant 3* with the speed-optimized version of AES, leads to an overhead of 4465 GEs.

When considering only the area requirement of the block-cipher implementation, AES encryption function and decryption function can be realized with 2772 GEs. Implementing the encryption-only version costs less than 1600 GEs. This is a consequence of heavily reusing registers that are normally utilized for handling the protocol. The so far smallest AES coprocessor implementation has been reported by Feldhofer *et al.* [4] and consumes about 3400 GEs. The smallest encryption-only version of AES, recently published by Moradi *et al.* [10], has a size of 2400 GEs. NOEKEON comes at much lower costs. The smallest version of NOEKEON containing encryption and decryption function counts 751 GEs. Comparison with related work is difficult since we could not find any published low-resource hardware implementation of NOEKEON.

Table 2. Overhead costs introduced by the different security-layer variants

Security layer		Protocol costs			Block-cipher costs			Total costs
Variant	**Algorithm**	**Registers**	**Code size**	**Total**	**Registers**	**Code size**	**Total**	
		-	[bytes]	[GEs]	-	[bytes]	[GEs]	[GEs]
	AES							
Variant 1	*fast*	0	250	**500**	3	1050	**1614**	**2115**
	small	0	250	**500**	3	858	**1517**	**2017**
Variant 2	*fast*	8	460	**1257**	3	1050	**1678**	**2935**
	small	8	460	**1257**	3	858	**1615**	**2872**
Variant 3	*fast*	8	452	**1165**	3	2034	**3300**	**4465**
	balanced	8	452	**1165**	3	1816	**2981**	**4146**
	small	8	452	**1165**	3	1602	**2772**	**3937**
	NOEKEON							
Variant 1	*fast*	0	250	**500**	0	652	**887**	**1387**
	small	0	250	**500**	0	382	**574**	**1074**
Variant 2	*fast*	8	460	**1283**	0	652	**1041**	**2323**
	small	8	460	**1283**	0	382	**660**	**1943**
Variant 3	*fast*	8	452	**1191**	0	980	**1545**	**2736**
	balanced	8	452	**1191**	0	532	**883**	**2074**
	small	8	452	**1191**	0	414	**751**	**1942**

Costs introduced by handling the additional reader commands and potentially storing the random number r_T range from 500 GEs to 1283 GEs. Although often neglected in related work, handling the protocol part of the security layer constitutes a significant portion of the overall costs and can even be the dominating factor. An example is *Variant 2* with the code-size optimized version of NOEKEON, where 66 % of the overhead costs are caused by the implementation of the protocol.

Simulating our microcontroller with the most-expensive security-layer variant (*Variant 3* with speed-optimized version of AES) gives an average power consumption of 2 µW at a clock frequency of 106 kHz and a voltage of 1.2 V. This value is very low since the microcontroller is highly optimized for low power consumption. Another advantage that arises from the combined implementation of protocol handling and cryptographic algorithm on the microcontroller is that no additional power is consumed for handling the security layer. When using a dedicated coprocessor, additional power would be required during computation of the cryptographic algorithm.

6 Conclusion

In this work we have evaluated the hardware overhead that arises from integrating different security-layer variants into a low-cost RFID tag. The security-layer variants are based on the cryptographic algorithms AES and NOEKEON.

We have used a combined implementation of high-level protocol handling and cryptographic algorithm on a low-resource 8-bit microcontroller. This combined approach provides high flexibility and allows reusing registers of the microcontroller that are only temporarily used during protocol handling. In that way AES encryption function can be implemented with an overhead of about 1600 GEs and NOEKEON encryption function with an overhead of about 600 GEs when using a 130 nm CMOS technology. The microcontroller has a power consumption of 2 µW at a clock frequency of 106 kHz. Total costs of the security-layer variants range from 1100 GEs to 4500 GEs and consider also the protocol handling of the security layer. Protocol handling can make up a significant part of the total costs and must not be neglected.

References

1. ARM Ltd. AMBA Advanced Microcontroller Bus Architecture Specification (1997), http://www.arm.com
2. Daemen, J., Peeters, M., Assche, G.V., Rijmen, V.: Nessie proposal: NOEKEON (2000), http://gro.noekeon.org/Noekeon-spec.pdf
3. Faraday Technology Corporation. Faraday FSA0A_C 0.13 μm ASIC Standard Cell Library (2004), http://www.faraday-tech.com
4. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES Implementation on a Grain of Sand. IEEE Proceedings on Information Security (1) (October 2005)
5. Giusto, D., Iera, A., Morabito, G., Atzori, L.: The Internet of Things - 20th Tyrrhenian Workshop on Digital Communications. Springer, Heidelberg (2010)
6. International Organization for Standardization (ISO). ISO/IEC 14443-3: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part3: Initialization and Anticollision (2001)
7. ISO/IEC. 7816-4: Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 4: Interindustry commands for interchange (1995)
8. ISO/IEC. 9798-2: Information technology – Security techniques – Entity authentication – Mechanisms using symmetric encipherment algorithms (1999)
9. ISO/IEC. 14443-4: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part4: Transmission Protocol (2008)
10. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
11. National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard (November 2001)
12. Plos, T., Groß, H., Feldhofer, M.: Implementation of Symmetric Algorithms on a Synthesizable 8-Bit Microcontroller Targeting Passive RFID Tags. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 114–129. Springer, Heidelberg (2011)
13. Yan, H., Jianyun, H., Qiang, L., Hao, M.: Design of low-power baseband-processor for RFID tag. In: International Symposium on Applications and the Internet Workshops (SAINT 2006). IEEE Computer Society (January 2006)
14. Yu, Y., Yang, Y., Yan, N., Min, H.: A Novel Design of Secure RFID Tag Baseband. In: RFID Convocation (2007)