Security-enabled NFC Tag with Flexible Architecture Supporting Asymmetric Cryptography

Thomas Plos, Michael Hutter, Martin Feldhofer, Maksimiljan Stiglic, and Francesco Cavaliere

***Abstract*—This article presents the design and implementation of a complete Near-Field Communication (NFC) tag system that supports high-security features. The tag design contains all hardware modules required for a practical realization, which are: an analog 13.56 MHz Radio-Frequency Identification (RFID) front-end, a digital part including a tiny (programmable) 8-bit microcontroller, a framing logic for data transmission, a memory unit, and a crypto unit. All components have been highly optimized to meet the fierce requirements of passively powered RFID devices while providing a high level of flexibility and security. The tag is fully compliant to the NFC Forum Type-4 specification and supports the ISO/IEC 14443A (layer 1-4) communication protocol as well as block transmission according to ISO/IEC 7816. As security features, it supports encryption and decryption using the Advanced Encryption Standard (AES-128), the generation of digital signatures using the Elliptic Curve Digital Signature Algorithm (ECDSA) according to NIST P-192, and it includes several countermeasures against common implementation attacks such as side-channel attacks and fault analyses. The chip has been fabricated in a 0.35 µm CMOS process technology and requires 49 999 GEs of chip area in total (including digital parts and analog front-end). Finally, we present a practical realization of our design that can be powered passively by a conventional NFC-enabled mobile phone for realizing proof-of-origin applications to prevent counterfeiting of goods or to provide location-aware services using RFID technology.**

***Index Terms*—8-bit microcontroller, AES, ECDSA, elliptic curve cryptography, embedded system, implementation security, NFC, RFID, VLSI design.**

I. Introduction

Radio-Frequency Identification (RFID) is a wireless communication technique that has become increasingly important in the last decade. Applications like electronic passports, logistics, animal identification, and car immobilizers make already use of this technology. A widely-used data-transmission standard that bases on RFID technology is Near-Field Communication (NFC). With the integration of NFC functionality into the latest generation of mobile phones (Samsung Galaxy Nexus, HTC Ruby) a further spreading of RFID technology is expected, paving the way for new applications. These new applications will have increased demand concerning the functionality provided by the RFID system, especially in context of security and privacy.

In a typical RFID system, a reader (e.g., a mobile phone) and a tag communicate contactlessly by means of a radio-frequency (RF) field. Most of the tags (more than 2 billions have been sold in 2011) are so-called passive tags that also receive their power supply from the RF field. A passive tag is basically a microchip attached to an antenna. The simple design of passive tags allows them to be produced at low cost, which is important for applications where large numbers of tags are required.

Tags used in future RFID applications will have to provide additional functionality such as security and data-storage features. Moreover, the design of the tags must get more flexible to allow an easier adaption for new applications. Achieving these goals for passive tags by keeping the overhead in terms of power consumption (passive operation) and silicon area (directly influences the tag price) as low as possible is highly challenging.

A lot of effort has been made by the research community to allow cryptographic services on resource-constrained RFID tags. The most prominent services are strong authentication using for example symmetric primitives like the Advanced Encryption Standard (AES) [1], [2] or asymmetric primitives like Elliptic Curve Cryptography (ECC) [3], [4]. Especially the integration of asymmetric schemes is a big challenge for passive RFID tag designs as they need more resources (computational effort, memory, etc.) than symmetric schemes.

When tags have to handle additional functionality, also their control complexity increases. Today's RFID tags use state machines fixed in hardware for handling their control tasks. However, this approach is no longer practical and even inefficient when the control complexity increases. Using a microcontroller approach instead is favorable and provides much more flexibility. Moreover, having a microcontroller on the tag for handling the control tasks, allows also reusing it for computing cryptographic operations.

In this article, we present the design and implementation of a security-enabled NFC tag with flexible architecture. The so-called CRYPTA tag (CRYPTA stands for ***Cryp**tographic Pro-tected **T**ags for new RFID **A**pplications*) operates at a carrier frequency of 13.56 MHz and works fully passively. We target a low-area design that requires as little resources as possible such that the tag production does not exceed the practical limits of a possible commercial launch. The security-enabled NFC tag has a size of less than 50 kGEs and supports strong

Manuscript received April 4, 2012; revised August 1, 2012; accepted October 18, 2012. This work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems (Project CRYPTA, Project Number 820843) and by the European Commission through the ICT program under contract ICT-2007-216676 (ECRYPT II).

Thomas Plos and Michael Hutter are with Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria, (e-mail: {Thomas.Plos, Michael.Hutter}@iaik.tugraz.at).

Martin Feldhofer was with Graz University of Technology when the work was done and is now with NXP Semiconductors Austria, Austria, (e-mail: martin.feldhofer@nxp.com).

Maksimiljan Stiglic and Francesco Cavaliere are with AMS AG, Austria, (e-mail: {Maksimiljan.Stiglic, Francesco.Cavaliere}@ams.com).

 DOI: http://dx.doi.org/10.1109/TVLSI.2012.2227849

authentication features that base on the AES-128 (symmetric cryptography) as well as on digitally signing of data using the Elliptic Curve Digital Signature Algorithm (ECDSA) over the prime field GF(p192) (asymmetric cryptography). The low-area goals have been achieved by heavily reusing existing hardware components like a common 8-bit microcontroller or a common memory. Passive operation of the tag with conventional NFC-enabled mobile phones allows realizing security-related NFC/RFID applications. Besides this, we also present a fully working prototype sample of our design fabricated on a 0.35 µm CMOS process technology.

Our work contains multiple contributions that relate to the field of secure tag design, which are:

- First low-resource RFID tag that supports asymmetric cryptography.
- First combined low-area implementation of ECDSA and AES.
- Flexible tag design based on a microcontroller for protocol handling and steering of the cryptographic module (including a design flow for program development).
- First low-resource RFID tag with countermeasures against implementation attacks.
- First prototype chip of a proof-of-origin tag.
- Consideration of the whole tag life cycle including: production, personalization, and user application.

Among the contributions listed above, describing the design of a complete system including all hardware components required for a practical chip fabrication of a security-enabled tag (including EEPROM and analog frond-end which are often omitted in related work) is indeed the most valuable one. Moreover, we provide details of the design at a level that is hardly available in published literature.

The remainder of this article is organized as follows. Section II provides an overview of the CRYPTA tag and describes the supported functionality. Detailed information about the tag architecture is given in Section III and an explanation of the design flow for program development of the microcontroller is provided in Section IV. Implementation results and a description of a prototyping sample are presented in Section V. Conclusions are drawn in Section VI.

II. The CRYPTA Tag

This section gives a brief overview of the CRYPTA tag and its main components. We describe the functionality that is provided by the tag, explain which parts are realized in hardware and which in software, and outline the typical life cycle of the tag.

An overview of the architecture of the CRYPTA tag is given in Figure 1. The tag mainly consists of an analog front-end and a digital part. The analog front-end is connected to an antenna and is responsible for demodulating and modulating data, extracting the power supply, and providing a stable clock signal as well as a reset signal. Attached to the analog front-end is the digital part which processes the received (demodulated) data, performs the requested actions, and prepares the data for the tag response. The digital part consists of: a low-resource 8-bit microcontroller, a framing logic (FL), a crypto unit (CU),

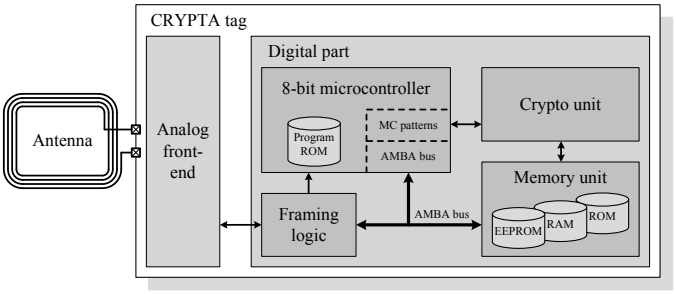

Fig. 1. Overview of the CRYPTA tag's architecture.

and a memory unit. Central element is the microcontroller that steers all operations on the tag. The microcontroller has its program stored in an internal Read-Only Memory (ROM) and communicates via an Advanced Microcontroller Bus Architecture (AMBA) bus with the framing logic and the memory unit. The framing logic is connected to the analog front-end and provides a byte interface for the microcontroller. Moreover, the framing logic also handles low-level commands (basic tag functionality) that are time-critical. High-level commands (advanced tag functionality) that have increased control complexity are handled by the microcontroller. Cryptographic operations such as signing of messages or encrypting of data are processed within the crypto unit that is accessed by the microcontroller via micro-code patterns. Volatile memory (RAM) for storing temporary results, non-volatile memory (EEPROM) for permanently storing data in files, and memory for storing constants (ROM) are located in the memory unit.

A. Standard Compliance

The tag is compliant to NFC Forum Type-4 specification [5] and uses the ISO/IEC 14443A protocol standard for communication. Basic tag functionality covering tag initialization and anticollision is implemented according to ISO/IEC 14443-3 [6]. Advanced tag functionality including security and file-management commands is implemented according to ISO/IEC 14443-4 [7] and bases on a block-transmission protocol. The block-transmission protocol uses Application Protocol Data Units (APDUs) for exchanging data as defined in ISO/IEC 7816-4 [8].

Advanced tag functionality of our tag involves security and file-management features that rely on six APDU commands. The security features cover tag authentication and reader authentication based on challenge-response protocols using the commands: INTERNAL_AUTHENTICATE, GET_CHALLENGE, and EXTERNAL_AUTHENTICATE. The file-management features allow accessing the files stored in the EEPROM and use the commands: SELECT_FILE, READ_BINARY, and UPDATE_BINARY. Tag authentication allows an RFID reader to verify the authenticity of the tag and can either be done symmetrically using AES [9] or asymmetrically using ECDSA [10]. Reader authentication allows the tag to verify the authenticity of the reader and is only done symmetrically using AES. We use 12 files to store, e.g., encryption keys, configuration parameters, data required for NFC compatibility, and user data. Depending on the file and

the configuration parameters of the tag, different read and write access to the files is granted (e.g., reader has to authenticate before reading from a file). A detailed description of the commands supported by the CRYPTA tag can be found in [11].

B. Splitting Functionality into Hardware and Software

Integrating security and file-management features to a tag significantly increases the control complexity. Data has to be transmitted from one component to another. Commands that are split into several blocks (i.e., chaining of data) need to be reconstructed. Moreover, commands have to be handled according to their parameters, the configuration of the tag as well as the current tag state. A tag architecture that is based on a microcontroller can better cope with such an increased control complexity than a conventional state-machine approach. However, when using a microcontroller, the fierce requirements of passive RFID tags in terms of chip area and power consumption have to be fulfilled. Consequently, only a very simple microcontroller with small chip size can be deployed. In order to keep the power consumption low, the microcontroller should be clocked with the lowest-possible clock frequency.

Processing all control tasks with the microcontroller would result in a rather high clock frequency due to the short tag-response time during initialization and anticollision phase (basic tag functionality). In order to address this aspect, basic tag functionality is directly handled by a dedicated hardware circuit (framing logic). Since controlling complexity of basic tag functionality is low, implementation in hardware is achievable. Moreover, basic tag functionality is independent of the overlaying application data and consequently does not affect the flexibility of the design. Advanced tag functionality on the contrary leads to high control complexity but has relaxed timing requirements that make an implementation in software (i.e., as program) on the microcontroller favorable.

C. Life Cycle of the CRYPTA Tag

We describe the typical life cycle of the CRYPTA tag by using the example of proof-of-origin as target application. The life cycle involves three main steps: tag production, personalization, and user application.

1) Tag Production: The first step in the life cycle is the production of the tag. There, the chip manufacturer writes a unique identifier (UID) and a temporary AES key to the EEPROM of the tag. The UID is fixed and cannot be changed afterwards. The temporary AES key is sent together with the tag to the product manufacturer.

2) Personalization: With the temporary AES key, the product manufacturer can access the EEPROM of the CRYPTA tag and can start its personalization. During personalization phase, the temporary AES key is replaced with the one selected by the product manufacturer. For ECDSA, a private key and a public-key certificate (in fact the dynamic part of a X.509 certificate) are stored on the tag. Moreover, read and write access to the different files in the EEPROM is defined and configuration parameters are set. The personalization phase is finished by enabling a special lock bit that prevents a further personalization of the tag afterwards (cf. Section III-D2).

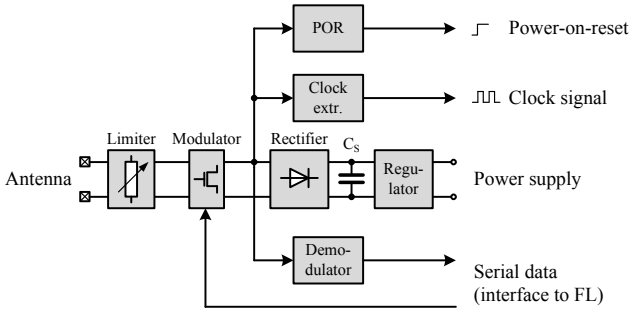

Fig. 2. Basic blocks of the analog front-end.

3) User Application: After personalization, the tag can be attached to the product that needs to be protected against counterfeiting. When a user buys the product together with the tag and wants to check its proof-of-origin (i.e., whether the product is genuine or not), one can download, e.g., an application from the web page of the manufacturer and run it on an NFC-enabled mobile phone. When touching the tag with the mobile phone, the application verifies first the validity of the public-key certificate on the tag, before starting with the authentication via ECDSA afterwards. When the certificate validation and the authentication step succeed, the product can be treated as genuine.

III. Tag Architecture

In the following, we describe the individual hardware components of the CRYPTA-tag architecture in detail, which are: analog front-end, framing logic, 8-bit microcontroller, and crypto unit.

A. Analog Front-End

The analog front-end extracts the tag's power supply from the RF field and provides an interface for the digital part (data, clock, reset). Main building blocks of the analog front-end, as shown in Figure 2, are: a limiter, a rectifier, a storage capacitor C_S, a regulator, a power-on-reset (POR) circuit, a clock-extraction circuit, a demodulator, and a modulator. The analog front-end is connected to a coil antenna that is receptive for the 13.56 MHz RF field emitted by the reader. In order protect the input of the analog front-end from too large voltages at the antenna, a limiter is used. The limiter starts drawing current when the antenna voltage increases (similar to a shunt regulator). For extracting the power supply of the tag, the voltage from the antenna is first fed into the rectifier and buffered by a storage capacitor before it goes through the regulator that keeps the supply voltage at a constant value. When the supply voltage is sufficiently large, a reset signal is released by the power-on-reset circuit. This reset signal activates the other components of the analog front-end and also the digital part. The clock signal for the digital part is directly extracted from the RF field via the clock-extraction circuit. Hence, the tag operates synchronously with the RF field. For receiving data from the reader, a demodulator is used. The demodulator has an envelope detector integrated as the reader data is amplitude modulated on the RF signal. The output of

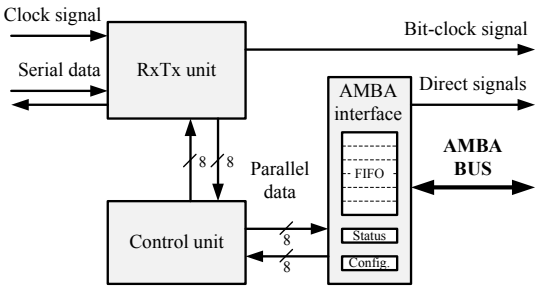

Fig. 3. Overview of the framing-logic architecture.

the demodulator provides a clean digital signal that can be further processed by the framing logic. For transmitting data to the reader that comes from the framing logic, a modulator is used. The modulator switches an impedance in step with the data that needs to be transmitted, resulting in a classical load modulation.

B. Framing Logic

The framing logic is some kind of serial-to-parallel interface that handles also basic tag functionality. Figure 3 sketches the architectural overview of the framing logic with the following main blocks: receive-and-transmit (RxTx) unit, control unit, and AMBA interface. The RxTx unit is the interface between the serial data signals of the analog front-end and the parallel data signals of the control unit. Additionally, the RxTx unit receives a clock signal from the analog front-end, which is used to extract a bit-clock signal that is provided to the microcontroller and the other components of the tag's digital part. For a default data rate of 106 kbit/s, the resulting bit-clock signal has a frequency of 106 kHz. Incoming serial data from the analog front-end is first sampled by the RxTx unit, decoded into bits, transformed to byte data, and checked for integrity (parity bits and CRC). Byte-level data coming from the control unit is appended with a checksum, encoded, and then transmitted bit by bit to the analog front-end. The RxTx unit is also responsible for proper timing of the tag response, which needs to be transmitted within certain time slots. The control unit steers the RxTx unit as well as the AMBA interface and handles also the initialization and anticollision phase of the tag (basic tag functionality). Commands that relate to advanced tag functionality are not handled by the control unit and are directly forwarded to the AMBA interface instead. The AMBA interface places this data into a so-called first-in first-out (FIFO) buffer (stores up to 6 bytes) that is accessed by the microcontroller over the AMBA bus. The buffer decouples the communication between control unit and microcontroller. When data coming from the microcontroller needs to be transmitted by the framing logic it is first placed in the FIFO buffer and then forwarded by the control unit to the RxTx unit.

For connecting the framing logic with the AMBA bus the AMBA interface is used. Although the data width of the AMBA bus is 16 bits, only the lower 8 bits are used by the framing logic, since it operates on byte level. The AMBA interface also contains a status register that provides

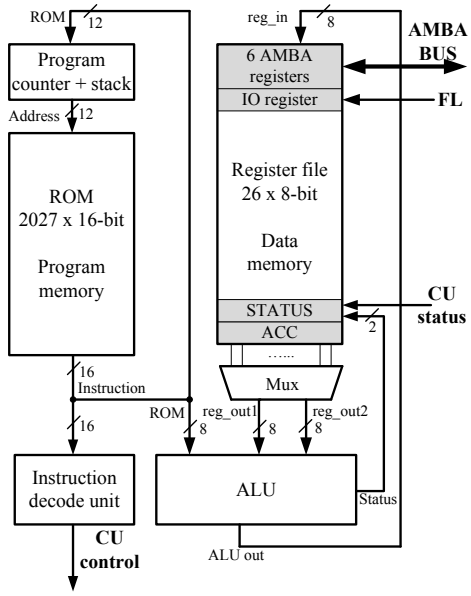

Fig. 4. Overview of the microcontroller architecture.

information about the internal state of the framing logic and a configuration register that allows the microcontroller to adjust various parameters. Both registers can be accessed by the microcontroller via the AMBA bus. Besides the AMBA bus, some additional direct signals are shared between framing logic and microcontroller to speed up communication (e.g., actual number of utilized bytes in the FIFO buffer).

C. 8-Bit Microcontroller

Our 8-bit microcontroller targets at low chip area and low power consumption for replacing conventional state machines that make a design inflexible and modifications very costly. Contactless smart cards in comparison have often 32-bit controllers integrated that can perform rather expensive operations [12], [13] but have high resource usage (area/power consumption). Our implemented microcontroller combines the advantages of hardwired state machines and complex controllers. It keeps the design programmable while consuming only a limited amount of hardware resources. The microcontroller steers all other modules of the tag via a memory-mapped AMBA bus or direct interfacing. Moreover, it is fully synthesizable for standard-cell technology but using an integrated program ROM macro is also possible.

An overview of the microcontroller architecture is depicted in Figure 4. The design uses a Harvard architecture which has the advantage that data memory (8 bits) and program memory (16 bits) can have different word sizes. The microcontroller supports 31 instructions which can be divided into four groups: logical operations (AND, XOR), arithmetic operations (ADD, SUB), control-flow operations (CALL, RET), and an operation for executing micro-code patterns (MICRO instruction). In order to reduce overhead no interrupts are supported which means that polling has to be implemented when waiting for an event.

There are several reasons why using an 8-bit datapath width for the microcontroller of our CRYPTA tag is beneficial.

First, using a smaller datapath width reduces the area of the microcontroller core and lowers also the power consumption. Although a larger datapath width (e.g., 16 or 32 bits) typically allows a more efficient handling of data (i.e., with less instructions) this holds no longer true when using a microcontroller mainly for control tasks. Second, the deployed block-transmission protocol (ISO/IEC 7816-4) operates on byte level, making it the natural choice to select an 8-bit datapath width. Using a smaller datapath width (e.g., 4 bits) will unnecessarily increase code size when handling protocol commands on byte level (e.g., when checking a command sequence).

The main components of the microcontroller are the register file, the program counter (PC), the program memory, the arithmetic-logic unit (ALU), and the instruction decode unit. The register file contains the data memory and consists of 26 8-bit registers. Although potentially 32 registers are addressable we reduced the size to 26 (minimum number of registers required for handling the protocol) which reduces the overall chip size and emphasizes the flexibility of our approach (saves about 65 GEs per register). The register file contains a set of general-purpose registers for storing variables and the internal state as well as special-purpose registers. These special-purpose registers (accumulator (ACC), status register (STATUS), 6 AMBA registers, IO register) are used for advanced data manipulation, status information like carry or the external status of a device, the AMBA bus access, and for the direct access of information from the framing logic and the crypto unit.

Instructions are executed within a two-stage pipeline that consists of a fetch and a decode-execute step. First, the instruction that is addressed by the 12-bit program counter is loaded from the program ROM into the instruction decode unit. Then the instruction is decoded by the instruction decode unit and executed by the ALU. Finally the program counter is updated. The program counter contains a call stack that allows up to four recursive subroutine calls. All instructions are executed within a single clock cycle, except control-flow operations (2 cycles if branch is taken) and the execution of micro-code patterns (depends on the pattern that is executed). The ROM contains the program of up to 4 096 instructions and is realized as look-up table in hardware. The ROM is also flexible where we instantiate only 2 027 instructions in the current design.

D. Crypto Unit

The crypto unit provides the following cryptographic services: digitally signing of data using the National Institute of Standards and Technology (NIST) recommended elliptic curve NIST P-192 [10], encryption and decryption using AES-128 [9], and hashing of data using SHA-1 [14]. We selected the NIST P-192 curve and AES-128 since both algorithms are standardized and have been analyzed over more than 12 years. They have proven to resist various attacks and provide a high level of security. Moreover, standardized algorithms are easier to integrate into existing infrastructures (e.g., when using X.509 certificates or when using software bundles that already support the curve NIST P-192 and AES-128). The NIST P-192 curve is the smallest recommended NIST elliptic curve over prime fields, which is important to minimize the amount of resources needed on the tag. Note that using an elliptic curve with a finite-field size of 160 bits or less (i.e., a security level of 80 bits or less) is no longer recommended by organizations such as the NIST or the Standards for Efficient Cryptography Group (SECG). Choosing a larger curve, e.g., NIST P-224 or P-256, would need more hardware resources (especially more memory capacity) and would require more time to generate a digital signature.

During the design phase of the project, we decided to reduce the overall area requirements by reusing hardware components like the memory and the controller for all implemented algorithms. ECDSA dominates the memory requirements and the controlling effort. In order not to increase the overall area requirements of the tag, AES and SHA-1 share as many resources (e.g., registers, finite-field multiplier) as possible with ECDSA. Furthermore, for implementing the higher-level cryptographic protocols we made use of the microcontroller. The protocols we have implemented are, e.g., the ECDSA authentication protocol according to ISO/IEC 9798-3 and ISO/IEC 9798-2 for mutual authentication using AES.

In order to improve the performance of cryptographic computations, we further decided to follow a micro-code control paradigm where low-level cryptographic instructions are implemented in eight distinct ROM tables. These so-called micro-code patterns are executed by a pattern sequencer that is invoked by the microcontroller using a dedicated instruction-set extension (the MICRO instruction). We implemented several microprograms, for example, for SHA-1, Montgomery multiplication and inversion, modular addition, subtraction, multiplication, modular reduction, and AES encryption and decryption. Each microprogram requires different instructions, instruction widths, and lines of code. SHA-1, e.g., needs only 9 bits for the instructions whereas modular arithmetic requires 18 bits. Therefore, we implemented each microprogram in eight different ROM tables that are different in length and width which lowers the overall area requirements.

Next to the micro-code pattern sequencer, the crypto unit consists of a datapath that realizes the basic operations for ECDSA, AES, and SHA-1, as well as a common memory unit which are described in the following subsections.

1) Datapath for ECDSA, AES, and SHA-1: The datapath of the crypto unit is shown in Figure 5. Basically, it consists of an ECDSA and AES datapath as well as a common 40-bit register that is used as accumulator during ECDSA execution and as an intermediate storage register for AES. In order to find the optimal datapath width for our processor, we made use of a high-level model written in Java. The model implements the datapath of ECDSA (including SHA-1) and AES as hierarchical functional blocks (a very hardware-near implementation). The model allows to vary different parameters like the datapath width. Each functional block of the model counts the number of needed clock cycles so that the execution time of the implementation can be roughly estimated for a given clock frequency. As a result, it showed that for small datawidths of, for example, 8 bits or less the runtime of ECDSA exceeds several million clock cycles. Hence, the

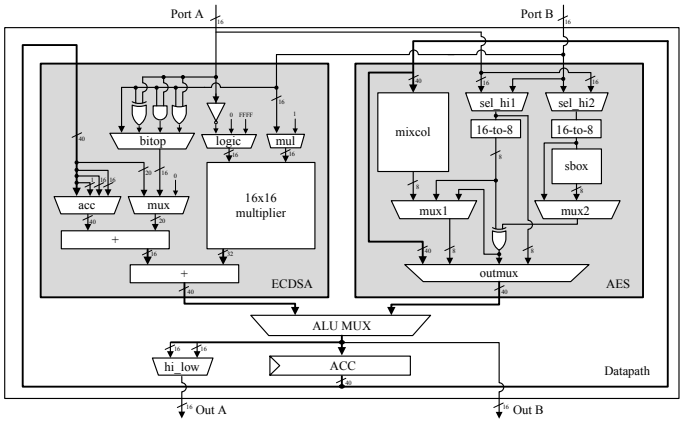

Fig. 5. Datapath of the crypto unit supporting ECDSA, AES, and SHA-1.

computation of ECDSA would require several seconds when using a clock frequency of 1 MHz. For a 16-bit datawidth, we estimated the calculation to take around 0.8 seconds, and for larger datawidths of 32 bits and more, the estimated execution time is reduced to only a few hundred milliseconds and even below. In general, the faster the implementation the broader is the range of possible RFID applications. While there exist applications where long response times are acceptable, e.g., in cases with continuous (authenticated) inventory requests in ware houses, there exist many applications where short response times are mandatory, e.g., authentication of products in logistics or in access control. However, choosing a larger datawidth requires more hardware resources even though it can process the operations much faster. In view of hardware resources, the dominant part (especially for ECDSA) is the hardware multiplier. We therefore implemented and synthesized different hardware multipliers and compared the area requirements. It showed that if the size of the datapath is doubled, the required area for the multiplier is increased by a factor of 4. In detail, an 8-bit multiplier needs about 380 GEs, a 16-bit multiplier needs 1 600 GEs, and a 32-bit multiplier needs already about 6 700 GEs. Due to these outcomes, we decided to implement a 16-bit datapath which, in our case, is a good trade-off between area and required speed.

Figure 5 further shows two separated ALUs for ECDSA and AES whereby in the ECDSA part a 16×16-bit multiplier and two 40-bit adders build the central components. For ECDSA, we implemented a multiply-accumulate architecture that allows multiplication and addition in the same clock cycle as proposed by J. Großschädl [15] and also practically applied by D. Hein [16]. Furthermore, we integrated the logical operations of AND, OR, and XOR in the ECDSA datapath which are also the main operations in SHA-1. ECDSA can reuse these operations, e.g., to extract individual bits of the secret nonce used during the scalar multiplication. So, sharing these resources is obvious and recommended to minimize the area requirements of the tag. The ALU of the AES mainly consists of an AES S-box and a MixColumns multiplier. This architecture has been taken from the low-power AES implementation of M. Feldhofer et al. [17]. In addition, we decided to separate the AES datapath into two 8-bit operations. This allows to reuse the remaining 8 bits to implement countermeasures against implementation attacks. In fact, we implemented dummy AES rounds and shuffling of bytes in the AES state. For further information and details about the crypto unit, the implementation results, and a comparison with related work, we refer to [18], [19].

2) Memory Unit: The memory unit consists of three memory types that are RAM, ROM, and EEPROM. They are addressable using a 16-bit linear dual-port memory space. The 128×16-bit dual-port RAM has been realized using a dedicated macro block. This significantly reduces the chip area and production costs, respectively, as compared to a standard-cell based RAM. A dual-port RAM showed to be advantageous since it allows reading of two words within one clock cycle. Also writing into one port and reading from the other is possible. This fact decreases the execution time of modular multiplication (which is the main finite-field operation in ECDSA) significantly. Also the size of the RAM, namely 128×16 bits, is advantageous since the datapath has also a width of 16 bits. Another important fact is that our chip manufacturer is in possession of a RAM macro with that size (RAM macros may not be available for other exceptional sizes). The RAM memory is used for: ECDSA that needs 7×192 bits for calculating the point multiplication, one 192-bit value that is needed to store the message that has to be signed, and one 192-bit value that is needed to store the ephemeral key k. Additionally, we reserved 192 bits for storing the seed that is used in both ECDSA and AES to generate the needed random numbers. Our prototype uses a Pseudo Random Number Generator (PRNG) to derive the random numbers from the seed. When used in a commercial product, the PRNG should be replaced by a True Random Number Generator (TRNG) to ensure a maximum level of security.

Next to the RAM macro, we made use of a dedicated EEPROM macro. The EEPROM stores non-volatile data like the ECDSA private key, the public-key certificate, the AES secret key, and potentially other user-specific data up to 4 kbits, which can be written in a personalization phase or during the protocol execution. The EEPROM also supports a so-called one-time programmable mode where bits that are set once can no longer be cleared afterwards. This is used for example to prevent erasing of lock bits that were previously set. Note that most of the related work does not consider an EEPROM, which is in fact an important real-world requirement. Last but not least, we implemented a ROM circuit that stores 128 16-bit constants like ECC parameters, SHA-1, and AES constants. In contrast to the RAM and EEPROM macro, it has been implemented as an unstructured mass of standard cells.

3) Implementation-Attack Countermeasures: We integrated several countermeasures in our design to thwart against common implementation attacks [20]. For ECDSA, we made use of the Montgomery scalar multiplication algorithm [21] that provides implicit protection against Simple Power Analysis (SPA) attacks. Furthermore, we randomized the projective coordinates of each elliptic-curve point to resist against Differential Power Analysis (DPA) attacks [22]. Finally, after scalar multiplication, the resulting point is checked to be a valid point on the elliptic curve which protects against most of the known

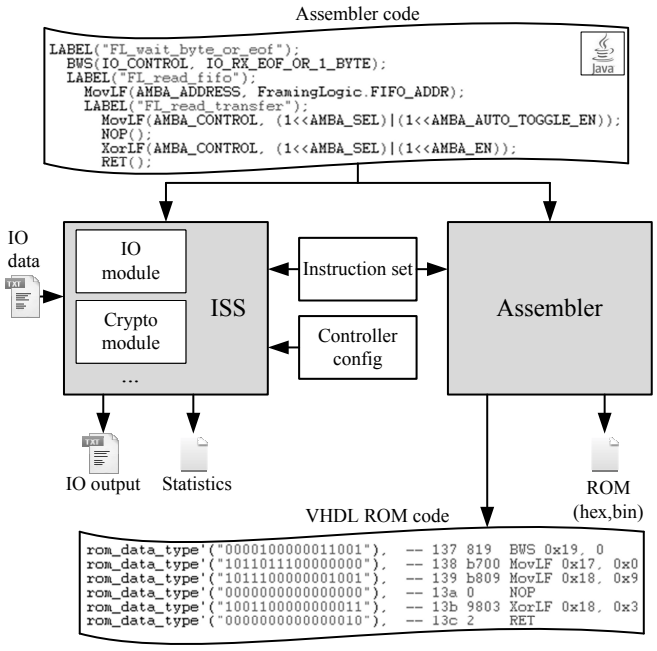

Fig. 6. Design flow for program development.

fault attacks [18], [19].

For AES, we integrated dummy operations (the number of dummy operations can be configured before synthesis) and shuffled the bytes in the 128-bit state. These hiding countermeasures make attacks on AES less attractive since the effort to perform an attack significantly increases with the number of added dummy operations. Practical measurements, where we performed electromagnetic-analysis attacks on AES, have shown that the number of needed measurements is increased from 246 (for the unprotected AES version) to more than 16 millions (for the protected AES version) [23].

IV. Design Flow for Program Development

The code development for the microcontroller makes our design approach very flexible. We have implemented a self-written tool chain that provides instruction-set simulation and assembler functionality. An overview of the design flow for code development is depicted in Figure 6. The program itself is written in assembler style but uses Java syntax that is actually a Java file. This avoids that we have to write parsing functionality and we can use Java for preprocessing, constant definitions and the like. Both the instruction-set simulator (ISS) and the assembler use a common instruction-set architecture definition (also based on Java). Further controller configuration that defines for example the available number of registers, the stack depth etc. are used in the simulator only.

The simulator additionally allows to integrate models of IO modules and other components like cryptographic circuits to enable simulation of the whole system (e.g., to generate also test data for Hardware Description Language (HDL) simulation). The simulation provides features like single-step mode, statistical data on the simulation run, and gives access to the internal state of the microcontroller. This makes debugging and testing of the program very convenient. Whenever the developed program is working in the simulator we use the assembler tool for code generation. The assembler is used to transform code from assembly language to a binary representation based on the instruction set (also dissolves addresses of labels). As the first and most important output it generates HDL code of the ROM as a look-up table ,which can be subsequently used for synthesis or for HDL simulation. Furthermore, it provides the data in a hex-file format and in a representation used for ROM-macro implementation.

TABLE I
Area of Chip Components.

Component	GEs	%
Analog front-end	8 100	16.20
Framing logic	2 663	5.33
8-bit microcontroller		
Instruction decode unit, ALU, and PC	945	1.89
Register file (26 × 8-bit)	1 693	3.38
Program ROM (2 027 × 16-bit)	6 764	13.53
Bus arbiter	319	0.64
Crypto unit		
Micro-code pattern sequencer	3 880	7.76
Datapath (ECDSA, AES, and SHA-1)	3 608	7.22
Memory unit		
EEPROM (256 × 16-bit)	12 700	25.40
ROM (crypto-unit constants)	600	1.20
RAM macro (128 × 16-bit)	8 727	17.45
Total	**49 999**	**100.00**

V. Implementation Results

We have implemented our flexible tag platform in VHDL and designed it towards low resource usage and low power consumption, i.e., by applying clock gating and operand-isolation techniques.

We implemented our design in a 0.35 µm CMOS technology using a semi-custom design flow with Cadence RTL Compiler as synthesis tool. Table I shows the chip-area results in terms of gate equivalents (GEs). In total, the chip needs 49 999 GEs including analog front-end, framing logic, microcontroller, bus arbiter, crypto unit, and memory. About 21 % (10 763 GEs) are needed for the RFID analog front-end and the framing logic. The microcontroller needs around 19 % including instruction unit, ALU, PC, register file (about 65 GEs per register), and program ROM. The datapath and the pattern sequencer of the crypto unit take about 15 % of the chip area, i.e., 7 488 GEs (this number does not include the ROM for ECDSA, AES, and SHA-1 program and the needed constants). The highest amount of resources is required for the memory, i.e., about 44 % of the total area, which equals to 22 027 GEs. By far smallest component is the bus arbiter (responsible for the AMBA bus), consuming less than 1 % of the total area.

The RFID front-end is clocked with 106, 212, or 448 kHz according to the specified data rate. The crypto unit can be clocked at higher frequencies (0.847, 1.7, 3.3, or 6.68 MHz) in order to improve the performance (configured in an EEPROM register during tag personalization). At a frequency of 1.7 MHz, a digital signature can be generated within 505 ms, i.e., 863 109 clock cycles. Hashing a message needs 2.15 ms (3 639 clock cycles) and AES needs 2.66 ms (no dummy rounds) and 9.16 ms (10 dummy rounds applied) which corresponds to 4 529 and 15 577 clock cycles, respectively. At the

TABLE II
DISTRIBUTION OF ROM CODE WITH RESPECT TO TAG FUNCTIONALITY.

Tag functionality	Code size [Instructions]	[%]
Protocol		
Generic subroutines	312	15.40
Block transmission	499	24.60
File management	331	16.30
Security features	119	5.90
Crypto services		
ECDSA-P192	487	24.00
AES-128	223	11.00
SHA-1	56	2.80
Total	**2 027**	**100.00**

highest frequency of 6.68 MHz, the ECDSA module needs 127 ms for generating a digital signature, which is sufficient for most applications having stringent response-time requirements.

A. Program ROM

After developing and evaluating the program of the microcontroller with the Java-based instruction-set simulator described in Section IV, the assembler was used to transform the assembly code into synthesizable VHDL ROM code. Proper operation of the whole tag has been further verified through simulations with Cadence NC Sim and through tests on an FPGA RFID tag prototype that can communicate with different reader devices, cf. [24]. The final ROM code for the microcontroller contains 2 027 instructions (equals 4 054 bytes of code). Subroutine calls are used whenever possible to keep code size small. Table II shows the distribution of the ROM code with respect to tag functionality. Most instructions of the ROM code, about 25 %, are only used for handling the block-transmission protocol. Around 15 % of the instructions are utilized for generic subroutines that provide a basic set of functions that are reused multiple times (e.g., routines for accessing the AMBA bus). File management and security features require about 22 %. The program part for steering the crypto unit needs 766 instructions, corresponding to about 38 % of the total program ROM (24 % for ECDSA-P192, 11 % for AES encryption/decryption, and 2.8 % for SHA-1).

Most of the instructions stored in the ROM relate to protocol handling, illustrating the high control complexity of our tag design. However, also the code used for steering the crypto unit comprises mainly control instructions (e.g., for executing micro-code patterns). Analyzing the code in the ROM in detail shows that about 60 % of the instructions are control operations (CALL, RET, BNZ, MICRO). Only 10 % of the instructions relate to pure data-flow oriented operations between one or two registers (XOR, ADD, ROT). The rest of the instructions belongs to operations between constants in ROM and registers, e.g., immediate load and compare instructions (MOVLF, XORLF).

B. Power Consumption

Power simulations of the system were conducted with the transistor-level SPICE simulator Synopsys Nanosim. The simulation for the microcontroller shows a mean power consumption of only about 10 µA for the 0.35 µm CMOS process technology when powered with a supply voltage of 2 V and using a clock frequency of 106 kHz, i.e., for a default data rate of 106 kbit/s. When higher data rates are selected, the power consumption increases accordingly (linearly with data rate). The crypto unit, in contrast, consumes about 485 µA as total mean current measured at 847 kHz, i.e., the lowest frequency for the crypto unit. More than 40 % of that power is due to the memory unit which is heavily used during scalar multiplication. The datapath unit needs about 24 %, the clock tree requires approximately 16 % [18].

Note that the overall power consumption of the system is already quite low due to low-power design techniques like clock gating and operand isolation. It meets the power requirements of most HF RFID systems and can be applied in different RFID or NFC applications. However, the power-consumption value can be even further decreased by moving towards a more-advanced CMOS process technology, e.g., 0.18 µm or 0.13 µm. Using these technologies, the reading distance becomes even better and can be applied, e.g., in long range ISO/IEC 15693 applications.

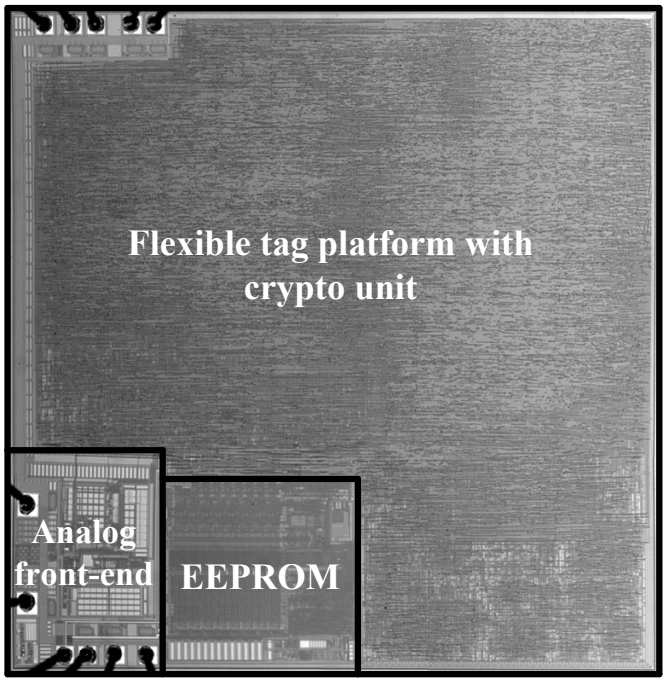

Fig. 7. Photo of the manufactured RFID tag-prototype chip.

C. An RFID-Tag Prototyping Sample

We manufactured our RFID-tag implementation on a multi-project wafer (MPW) using the 0.35 µm CMOS process technology C35b4 from AMS AG [25]. For ease of testability, a small serial debug interface has also been added that allows detailed analysis of the analog front-end and the EEPROM (e.g., reading/writing arbitrary values from/to EEPROM). A photo of the manufactured chip is shown in Figure 7.

After production, the chip has been integrated into a ceramic package and soldered on a small printed circuit board (PCB)

Fig. 8. Proof-of-origin application using our RFID-tag prototyping sample and the Google Nexus S mobile phone.

to allow tests with real-world RFID-reader devices. The PCB contains an antenna with 4 windings that is connected to the analog front-end of the chip. An adjustable capacitor is used for matching of antenna and analog front-end. Figure 8 shows a photo of the PCB with the packaged chip. We successfully tested the RFID-tag sample with different commercially available RFID readers including mobile devices featuring NFC capabilities. Using the Google Nexus S, for example, the tag can be powered fully passively and can reliably communicate with the phone up to 3 centimeters (at data rates up to 424 kbit/s and frequencies up to 6.68 MHz for the crypto unit). Using our flexible tag platform, different RFID and NFC applications have been realized such as proof-of-origin authentication to thwart against counterfeiting goods or to generate location-aware signatures to prove that a person or object has been at a certain location in a specific moment in time. Several press releases have been published that demonstrate these demo applications, see for example [26] or [27].

D. Comparison with Related Work

Comparing our results with related work is rather difficult as only a handful of publications exist that deal with implementing security-enabled tags. Moreover, authors often give only a vague description of their designs regarding implementation details and provided functionality. A. S. Man et al. [28] and A. Ricci et al. [29], e.g., present tag designs for the ultra-high frequency (UHF) range that contain an AES-128 implementation. The AES implementations used by them have an area requirement of about 6-7 kGEs. Moreover, the two tag designs cover only the baseband part, i.e., the digital circuit without EEPROM and analog front-end. A design that is better comparable to our work is the one of J.-W. Lee et al. [30]. The authors present an NFC tag including EEPROM (4 kbits, i.e., same size as ours), analog front-end, and cryptographic unit with AES-128. Their NFC tag has a similar size (i.e., around 50 kGEs) than our design, but supports neither asymmetric cryptography (or SHA-1) nor has it countermeasures against implementation attacks integrated. This illustrates the advantage of our design concept that provides not only high flexibility but also very low resource usage when considering all the implemented features.

VI. Conclusion

In this article, we presented a flexible NFC-tag architecture that provides enhanced security features using symmetric as well as asymmetric cryptography. As a main contribution, the work describes an entire "real-world" RFID system including all hardware components needed for a practical chip fabrication. During the work, several outcomes have been obtained. First, our design shows that significant resources can be saved by applying a microcontroller-based architecture instead of using a finite-state machine based controlling. The reason lies in the fact that the controller can be simply reused by many hardware components such as the crypto unit or the RFID framing logic that would require more area when implemented as individual hardware modules. For example, AES encryption and decryption has been realized with an area overhead of only 2 387 GEs, which is lower than existing low-area AES implementations. Furthermore, SHA-1 needs only 889 GEs because of reusing available memory and microcontroller components of the entire system. Next to these outcomes, we made the experience that it is favorable to reuse the microcontroller for RFID protocol handling, e.g., handling ISO/IEC 14443 layer 4. This can be completely realized as a micro program, which reduces further chip-area requirements while increasing flexibility and assembly-based implementation convenience. Finally, we have practically proven our design by fabricating the system as a prototyping sample that demonstrates the feasibility of a full-blown RFID/NFC tag supporting ISO/IEC 14443A layer 1-4, NFC Forum Type-4 features (including NDEF support), a flexible (programmable) 8-bit microcontroller, memory (RAM, ROM, and EEPROM), analog front-end, and strong cryptography (ECDSA and AES) for less than 50 kGEs.

As future work, we plan to further analyze our design regarding enhanced implementation attacks such as side-channel analysis and fault attacks. Moreover, we plan to implement additional demo applications to verify the applicability of our tag in different security-related scenarios.

Acknowledgment

The authors would like to thank Johannes Wolkerstorfer, Manfred Aigner, Jörn-Marc Schmidt, and Nikolaus Ribic for their contributions within this project and for very fruitful discussions.

References

[1] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems using the AES Algorithm," in *CHES 2004, Proceedings.*, vol. 3156. Springer, August 2004, pp. 357–370.

[2] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," in *DSD 2006, Proceedings.* IEEE CS, September 2006, pp. 577–583.

[3] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, "Public-Key Cryptography for RFID-Tags," in *RFIDSec 2006, Proceedings.*, 2006, pp. 1–16.

[4] P. Tuyls and L. Batina, "RFID-Tags for Anti-counterfeiting." in *CT-RSA 2006, Proceedings.*, D. Pointcheval, Ed., vol. 3860. Springer, 2006, pp. 115–131.

[5] NFC Forum, "NFC Forum Type 4 Tag Operation - Technical Specification," Available online at http://www.nfc-forum.org/specs, NFC Forum, March 2007.

[6] International Organization for Standardization (ISO), "ISO/IEC 14443-3: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part3: Initialization and Anticollision," Available online at http://www.iso.org, 2001.
[7] ——, "ISO/IEC 14443-4: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part4: Transmission Protocol," Available online at http://www.iso.org, 2008.
[8] ——, "ISO/IEC 7816-4: Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 4: Interindustry commands for interchange," Available online at http://www.iso.org, 1995.
[9] National Institute of Standards and Technology (NIST), "FIPS-197: Advanced Encryption Standard," November 2001, available online at http://www.itl.nist.gov/fipspubs/.
[10] ——, "FIPS-186-3: Digital Signature Standard (DSS)," 2009, available online at http://www.itl.nist.gov/fipspubs/.
[11] T. Plos and M. Feldhofer, "Hardware Implementation of a Flexible Tag Platform for Passive RFID Devices," in *DSD 2011, Proceedings.* IEEE CS, August 2011, pp. 293–300, iSBN 978-1-4577-1048-3.
[12] Infineon Technologies AG., "Security and Chip Card ICs SLE 88CFX4000P," Available online at http://www.ic-on-line.cn/iol/datasheet/sle88cfx4000p_1310434.pdf, 2003.
[13] NXP Semiconductors., "LPC1000(L) - 32-bit MCU," Available online at http://www.nxp.com, 2011.
[14] National Institute of Standards and Technology (NIST), "FIPS-180-3: Secure Hash Standard," October 2008, available online at http://www.itl.nist.gov/fipspubs/.
[15] J. Großschädl, "A Bit-Serial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)," in *CHES 2001, Proceedings*, vol. 2162. Springer, May 2001, pp. 202–219.
[16] D. Hein, J. Wolkerstorfer, and N. Felber, "ECC is Ready for RFID - A Proof in Silicon," in *SAC 2008, Proceedings.* Springer, September 2008, pp. 401–413.
[17] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand," *IEEE Proceedings on Information Security*, vol. 152, no. 1, pp. 13–20, October 2005.
[18] M. Hutter, M. Feldhofer, and J. Wolkerstorfer, "A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES like Sardines," in *WISTP 2011, Proceedings.*, vol. 6633. Springer, 2011, pp. 144–159.
[19] M. Hutter, M. Feldhofer, and T. Plos, "An ECDSA Processor for RFID Authentication," in *RFIDsec 2010, Proceedings.*, vol. 6370. Springer, 2010, pp. 189–202.
[20] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer, 2007, iSBN 978-0-387-30857-9.
[21] P. L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, January 1987, iSSN 0025-5718.
[22] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO 1999, Santa Barbara, CA, USA*, 1999, pp. 388–397.
[23] T. Korak, T. Plos, and M. Hutter, "Attacking an AES-enabled NFC Tag - Implications from Design to a Real-World Scenario," in *COSADE 2012, Proceedings.*, 2012, pp. 17–32.
[24] M. Feldhofer, M. J. Aigner, M. Hutter, T. Plos, E. Wenger, and T. Baier, "Semi-Passive RFID Development Platform for Implementing and Attacking Security Tags," in *RISC 2010, Proceedings.*, 2010, pp. 1–6.
[25] AMS AG, "The AMS AG Website," Available online at http://www.ams.com/.
[26] RFID im Blick, "Schlüssel zur Authentizität," Available online at http://www.rfid-im-blick.de/images/stories/magazin/rib_flschungsschutz.pdf, August 2011.
[27] ORF, "Grazer Forscher entwickeln Echtheitsprüfer," Available online at http://stmv1.orf.at/stories/514992, May 2011.
[28] A. S. Man, E. S. Zhang, V. K. Lau, C. Tsui, and H. C. Luong, "Low Power VLSI Design for a RFID Passive Tag baseband System Enhanced with an AES Cryptography Engine," in *Eurasia 2007, Proceedings*. IEEE, September 2007, pp. 1–6.
[29] A. Ricci, M. Grisanti, I. De Munari, and P. Ciampolini, "Design of a 2uW RFID baseband processor featuring an AES cryptography primitive," in *ICECS 2008, Proceedings*, 31 2008-sept. 3 2008, pp. 376 –379.
[30] J.-W. Lee, D. H. T. Vo, S.-H. Hong, and Q.-H. Huynh, "A Fully Integrated High Security NFC Target IC Using 0.18 µm CMOS Process," in *ESSCIRC 2011, Proceedings*. IEEE, 2011, pp. 551–554.

Thomas Plos received the BSc and MSc degrees in telematics from Graz University of Technology (TU Graz) in 2004 and 2007, respectively. In 2011 he received the PhD degree in computer science from TU Graz. His research interests include digital VLSI design, information security, RFID technology, and side-channel analysis. Currently, he is a post-doctoral researcher at the Institute for Applied Information Processing and Communications (IAIK) at TU Graz.

Michael Hutter is a post-doctoral research assistant at Graz University of Technology in Austria. Since 2007, he has been working in the security group of the Institute for Applied Information Processing and Communications (IAIK). His main research interests include applied cryptography, RFID security and privacy, side-channel attacks, and fault analyses. He holds a PhD and MSc in Computer Science.

Martin Feldhofer received the MSc degree in telematics in 2003 and the PhD degree in computer science in 2008 both at Graz University of Technology. Currently he is working at NXP Semiconductors with the focus in passive UHF technology. His main research activity is in the area of secure RFID technology with the focus of hardware design.

Maksimiljan Stiglic graduated in 1983 at University of Ljubljana, Faculty of Electrical Engineering (title dipl.ing.). From 1983 to 1989 he worked as researcher at the Laboratory for Microelectronics at the University of Ljubljana, Faculty of Electrical Engineering. In parallel he studied and passed his master thesis (title Mag.) in Electrical Engineering in 1987. From 1989 to 2008 he worked for EM Microelectronic-Marin, Marin, Switzerland, as IC designer and project manager, specialized in RFID tag ICs. He joined AMS AG in August 2008.

Francesco Cavaliere graduated in 1989 at University of Rome "La Sapienza", Faculty of Electronic Engineering. From 1990 to 2009 he worked at Texas Instruments in Dallas, Texas, USA, as IC designer and project manager in different business units, with focus on digital and mixed-signal design for RF ICs. He joined AMS AG in March 2009.