Int. J. Inf. Secur.
DOI 10.1007/s10207-014-0236-y

REGULAR CONTRIBUTION

Implementing public-key cryptography on passive RFID tags is practical

Alex Arbit · Yoel Livne · Yossef Oren · Avishai Wool

Abstract Passive radio-frequency identification (RFID) tags have long been thought to be too weak to implement public-key cryptography: It is commonly assumed that the power consumption, gate count and computation time of full-strength encryption exceed the capabilities of RFID tags. In this paper, we demonstrate that these assumptions are incorrect. We present two low-resource implementations of a 1,024-bit Rabin encryption variant called WIPR—in embedded software and in hardware. Our experiments with the software implementation show that the main performance bottleneck of the system is not the encryption time but rather the air interface and that the reader's implementation of the electronic product code Class-1 Generation-2 RFID standard has a crucial effect on the system's overall performance. Next, using a highly optimized hardware implementation, we investigate the trade-offs between speed, area and power consumption to derive a practical working point for a hardware implementation of WIPR. Our recommended implementation has a data-path area of 4,184 gate equivalents, an encryption time of 180 ms and an average power consumption of 11 μW, well within the established operating envelope for passive RFID tags.

Keywords RFID · Security · Supply chain

A. Arbit · Y. Livne · A. Wool
Cryptography and Network Security Lab, School of Electrical Engineering, Tel-Aviv University, Ramat Aviv,
Tel Aviv 69978 , Israel
e-mail: alexand5@eng.tau.ac.il

Y. Livne
e-mail: livneyoe@eng.tau.ac.il

A. Wool
e-mail: yash@eng.tau.ac.il

Y. Oren (✉)
Network Security Lab, Computer Science Department,
Columbia University, 1214 Amsterdam Avenue,
New York, NY 10027, USA
e-mail: yos@cs.columbia.edu

1 Introduction

1.1 Background

The electronic product code (EPC) system is one of the world's most ambitious pervasive computing projects. It aims to replace today's familiar 14-digit optical-scan universal product code bar codes with radio-frequency identification (RFID) tags operating in the ultra-high frequency (UHF) band, which are based on the EPC standard [1]. As noted in [2], the additional capabilities of EPC tags create considerable privacy issues which did not exist with optical bar codes. For example, it is possible to track individuals by placing EPC readers in multiple locations and searching for RFID tags carried by a person (for example on RFID-tagged clothes or banknotes) as he moves between them. Clearly, the EPC ecosystem will greatly benefit from the use of cryptography to protect the communications between the tag and the reader. However, adding cryptography to the EPC system is far from trivial.

There are several factors which make it extremely challenging to introduce security and privacy into an RFID environment. Most significantly, there is the issue of *power consumption*—EPC tags are passively powered by the RFID reader and, as such, have an extremely limited energy budget. Since the power available to the tag decreases in proportion to the square of its distance from the reader, increasing a tag's energy budget will force it to move closer to the reader and severely limit its usability. According to [3], the average power consumption of a typical UHF tag cannot exceed 30 μW. This limits both the circuit size of the device and its maximum clock rate. Another constraint is that of *gate count*—EPC tags are designed to cost only a few cents, imposing a severe limit on the chip area and thus on the gate count. According to [4], the overall gate budget of a passive RFID tag is on the order of 10,000 gate equivalents (GEs).

Because of these constraints, common wisdom holds that public-key cryptography is too expensive for such RFID tags [5]. Specifically, the perception is that full-strength cryptography is too slow and that it requires too much energy and too many gates. Hence, the vast majority of proposed security schemes for RFID systems rely exclusively on symmetric-key primitives [6]. However, RFID tags were shown to be vulnerable to reverse engineering, even by a moderately funded adversary [7]. This makes it extremely *problematic to store sensitive data* (such as symmetric encryption keys) on these tags, since the entire system can be compromised as soon as the secret key is recovered from even a single tag.

WIPR is an encryption scheme, first described in [8], which is designed to address all three of these challenges—power consumption, gate count and storage of sensitive data. WIPR has a very simple design, allowing its implementation to have both *low power consumption* and a *low gate count*. Significantly, since WIPR is an *asymmetric* (*public-key*) encryption scheme, *no sensitive data* need to be stored on the tag itself, dramatically reducing the damage caused by reverse engineering attacks. WIPR also enjoys a very large payload capacity, which enables a wide variety of applications, from supply-chain anti-counterfeiting to secure sensor networks.

1.2 Related work

The WIPR scheme is based on the randomized variant of the well-known Rabin cryptosystem [9], first discussed in [10]. This scheme's applicability to low-resource smart cards was explored in [11,12] and later [13]. The Rabin cryptosystem was first implemented in a low-resource setting by [5], but was found to be unsuitable for the ultra-low-resource RFID tags. Other public-key RFID contenders can be found in works such as [14,15], but these implementations generally require more gates than can fit in a low-cost tag or rely on uncommon features such as very large random sources. Several authentication protocols based on other light-weight primitives such as hash functions were also suggested in [16,17].

The ultra-low-resource implementation of the Rabin protocol presented in [8,18] replaces the long pseudo-random sequence, originally stored on EEPROM in [12], by a reversible stream cipher using less than 300 bits of RAM, with gate count estimate (based on partially simulating the data path) of around 5,000 gate equivalents. A proposed improvement, which claims reduced hardware requirements and protects against some attacks, was also presented in [19]. A prototype for a logistical system that uses WIPR is described in [20].

Several other works have also evaluated concrete low-resource implementations of public-key cryptography, as surveyed recently by Najera et al. [21]. In [22], Plos et al. present the design and implementation of a magnetically coupled near-field communication tag system supporting high-security features, including an elliptic curve digital signature system. The gate count of the complete device, including an analog front end, is 49,999 GEs. In [23], Wenger et al. evaluate the cost of adding support for elliptic curve cryptography to several popular microcontrollers using instruction set extensions. The gate cost of adding an ECC core to these microcontrollers was simulated and found to be between 6,140 and 18,700 GEs excluding RAM, and between 16,786 and 32,034 GEs including RAM. Other works, such as that of Batina et al. [24], propose additional public-key schemes suitable for RFID tags, but these works do not discuss complete implementations and as such are difficult to compare to our system.

1.3 Our contribution

In [8], Oren and Feldhofer presented a preliminary possible implementation of WIPR's data path and presented an estimate on the area and power consumption of a device built using this design. This implementation was improved in the work of [18], which also presented a deployment scenario for the WIPR scheme. However, the question of the scheme's practicality remained unresolved.

In this contribution, we present detailed software and hardware implementations of WIPR and use them to explore the technological design space and its limitations.

Our first implementation target was a slow microcontroller-based software implementation on a custom programmable RFID tag [25]. We used this implementation to experiment with the protocol, the air interface and the connection between the tag and the reader. We discovered that the main performance bottleneck was not the encryption time, but rather the EPC Class1 Generation2 (C1G2) air interface and the way the protocol was implemented in the reader.

Our second implementation target was a detailed ASIC implementation. We used this implementation to explore the design space of a hardware implementation of WIPR, which presents a trade-off between area, power, energy and time for encryption. Through extensive gate-level simulation, we identified a recommended working point within this design space which is fast-performing yet frugal enough, both in its area and in its power consumption, to fit into a passive supply-chain tag: Our recommended implementation has a data-path area of 4,184 GEs, an encryption time of 180 ms and an average power consumption of 11 μW, well within the established operating envelope for passive RFID tags.

1.4 Document structure

In Sect. 2, we describe the WIPR cryptographic scheme. In Sect. 3, we describe our embedded software implementation and experiments. In Sect. 4, we describe our detailed ASIC implementation. Finally, we conclude our paper in Sect. 5.

2 The WIPR cryptographic scheme

2.1 Theoretical basis

WIPR is a variant of the Rabin's encryption scheme presented in [9], first discussed in [10], which is provably as secure as factoring large numbers. In Rabin's scheme, the private key consists of two large prime numbers p and q. These are multiplied to form the public key $n = p \cdot q$. The plaintext P is typically generated from a shorter string (in our case an ID) by padding it with random bits until it is as long as n. To encrypt a plaintext P in this scheme, the sender calculates the ciphertext M as its square, reduced modulo n:

$$M = P^2 \pmod{n}$$

To decrypt a ciphertext, the receiver calculates the square roots of M modulo p and q, and then combines the resulting values using the Chinese Remainder Theorem [26, §2.4.3]. Each ciphertext has two possible roots modulo p and two roots modulo q ($\pm m \pmod{p}$ and $\pm m \pmod{q}$), leading to four possible plaintexts for each ciphertext. To allow the receiver to determine which of the four possible plaintexts is the correct one, the sender typically adds some redundancy to the message (in our case, the reader's challenge serves this purpose).

The encryption element of Rabin's scheme is relatively easy to implement, requiring only a single multiplication and modular reduction. However, modular reduction is a RAM-intensive process, a fact that limits the applicability of Rabin's algorithm to low-resource devices such as smart cards. To reduce the resource requirements of Rabin's scheme, Naccache in [11] and Shamir in [12] and later [13] suggested a RAM efficient variant, replacing the modular reduction step by an addition of a large random multiple of n, where the size of the random value r is at least 80 bits longer than the size of n (to have no detrimental effects on security):

$$M = P^2 + r \cdot n$$

The decryption algorithm is precisely identical to Rabin's original scheme. Shamir proved that the security of this resource-reduced scheme and the original Rabin scheme are equivalent. The reduced scheme is easier to implement since it has only multiplication operations and not modular reductions. In terms of space requirement, the problem of storing P^2 was replaced by the challenge of storing the large random number r. However, since r is written to only once per protocol execution [12], suggested that it should be stored in EEPROM, which is plentiful on smart cards, and not on the more scarce RAM. However, rewritable EEPROM is cheap on smart cards and prohibitively expensive on RFID tags, due to the high power cost of the write operation.

The final resource reduction in the Rabin scheme was presented in the WIPR scheme [8,18]. WIPR replaces r with the output of a low-resource reversible stream cipher. This cipher is implemented by creating a Feistel structure [27], a well-known cryptographic construct used in symmetric ciphers such as DES and TEA. To make use of this cryptographic building block to provide secure identification, a challenge-response construction was used, adding a reader-supplied random challenge to the plaintext P.

2.2 Protocol steps

Given the above description, following is an outline of the protocol steps:

1. **Setup:** The tag is provided with the public key n and a signed unique identifier ID. The reader is provided with the private key (p, q).
2. **Boot:** The reader generates a random bit string R_r, where $|R_r| = \alpha$. The tag generates two random bit strings R_{t1} and R_{t2}, where $|R_{t1}| = |n| - \alpha - |ID|$ and $|R_{t2}| = |n| + \beta$. and α, β are security parameters (both set to 80 in our implementation).
3. **Challenge**: The reader sends R_r to the tag.
4. **Response:** The tag generates a plaintext as follows: $P = R_r \# R_{t,1} \# ID$, where # denotes concatenation, and then transmits the following message:

$$M = P^2 + R_{t2} \cdot n$$

5. **Verification:** The reader uses the private key to decrypt M. There are four candidate decryptions, so the reader checks which of the four possible decryptions contain the value of the challenge R_r it sent to the tag. If such a plaintext is found, the reader outputs the value of ID. In all other cases, the authentication fails.

The WIPR protocol is based on public-key cryptography—the public key stored on the tag allows messages to be encrypted, but does not allow messages to be decrypted, even if those messages were previously transmitted by the same tag. In contrast, a system based on secret-key cryptography must use the same key both on the reader and on the tag, and this secret key can be used to encrypt and decrypt all messages. In such a scenario, capturing and reverse engineering a tag may compromise the entire authentication system. As discussed in [20], building a system around public- key cryptography provides additional security guarantees to the users of the system and dramatically simplifies the logistics involved with creating, distributing and deploying the tags.

3 Embedded software implementation

3.1 Objectives

WIPR was shown in [18] to have an acceptable gate count and power consumption, but the time presented in [18] was 600 ms per encryption, a delay which might be considered too much in a supply-chain scenario. Through the software implementation, we wanted to discover whether the cryptographic operation is indeed an inherent time bottleneck, or whether it can be sped enough to make the system usable. We also wanted to address the system issues and find out whether a practical public-key system can be created using today's hardware and standards.

3.2 Design

The system we built consists of an EPC C1G2-compliant RFID tag, an EPC C1G2-compliant RFID reader and two PC workstations.

The system setup is presented in Fig. 1. Our system used the UHF Demotag, a hardware prototyping platform developed by IAIK TU Graz. As stated in [25], the tag is battery-powered, but behaves like a fully passive tag in the reader field. It is fully compatible to ISO 18000-6c and EPC C1G2 standards. The tag is optimized for easy adaptability to allow fast development of prototypes. It features a ATMega128 microcontroller with JTAG and ISP interface for programming. An RS232 interface is available for configuration and logging. The front end consists of discrete devices on a PCB, with a PCB antenna that is tuned to 868 MHz. The tag is connected via a serial RS232 communication link to a Linux workstation running the CrossStudio for AVR embedded development environment by Rowley Associates, version 1.4. The firmware executes on power-on from the Atmega128's on-chip flash memory. As a reader, we chose the CAEN RFID DK828EU reader. It features a controller module with embedded EPC C1G2 reader firmware which is controlled via USB link by a Windows workstation running Matlab. The DK828EU reader conforms with European ETSI power requirements [28]. In our laboratory tests, we found that this reader has an average read rate of approximately 15 kbps, a fact which dominated the overall performance of our system. The IAIK SCA Toolkit provides the connection between the reader's software libraries and Matlab. Finally, an RFID wireless link is established between the Demotag and the reader.

Figure 2 demonstrates the full WIPR protocol flow through an EPC C1G2 air interface using standard EPC protocol commands. The reader first sends the standard INVENTORY command. WIPR tags do not respond to this command with the full EPC, which may be sensitive and should not be disclosed. Instead, the tag sends a special EPC value indicating that it is a WIPR tag and possibly disclosing a limited subset of the EPC which is sufficient for use with non-secure readers. To allow for a single WIPR tag to be successfully singulated when multiple WIPR tags are present, part of this special EPC value will be a random value computed on boot.

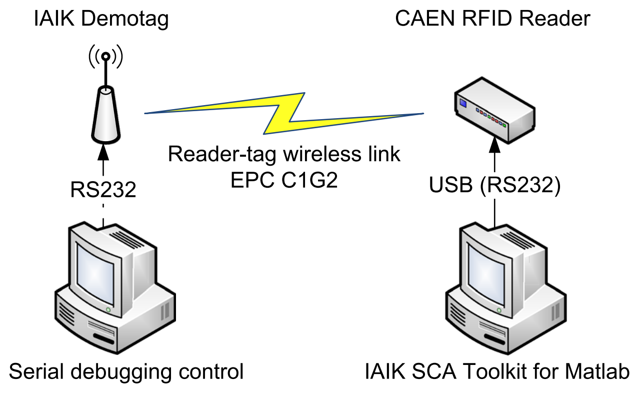


Fig. 1 System setup

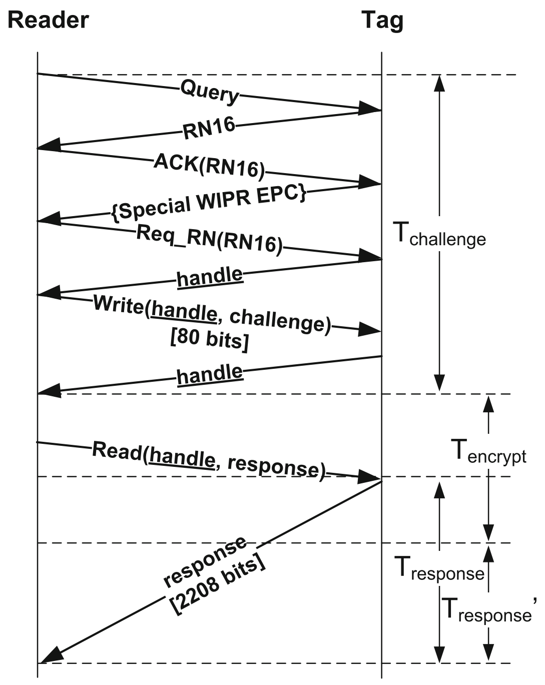


Fig. 2 The full WIPR implemented using mandatory C1G2 commands (based on [1], [annex E])

The reader then starts sending the 80-bit cryptographic challenge R_r. This operation is performed through the standard EPC C1G2 WRITE command. After the challenge is sent, the tag automatically encrypts its payload of data (consisting of its ID, the challenge and the locally generated random string R_{t1}) and places it in the SRAM buffer on the ATMega128 chip. Once the reader issues a standard BLOCK_READ command to the tag, the ciphertext is read out from the tag. The reader is free to initiate as many cycles of data transfer as it wishes between 1 and 138 16-bit words (the entire encrypted payload). As shown in the following subsection, larger block sizes result in a faster and more efficient data transfer.

It is important to note the three times marked in Fig. 2 as $T_{\text{challenge}}$, T_{encrypt} and T_{response}. While $T_{\text{challenge}}$ and T_{response} are determined by the speed of the link between the tag and the reader, T_{encrypt} is solely a function of the implementation quality of the WIPR algorithm. It can also be noted that only a part of T_{response} (marked as $T_{\text{response}'}$) happens after encryption is completed. As we discuss in the following subsection, this is due to a special property of the WIPR algorithm which allows for the ciphertext to be generated byte by byte.

3.3 Implementation

The tag is provided with a 1,024-bit public key n, which is stored in the tag's ROM and can be copied to the heap on boot to improve performance. The tag also stores its signed ID, which can be up to 864 bits long (for reference, a high-security ECDSA signature is 320 bits long). When issued with a fresh challenge R_r, the tag generates two random bit strings R_{t1} (between 80 and 1,024 bits) and R_{t2} (1,104 bits).

When the tag receives the challenge R_r sent by the reader, it stores it in heap memory. It then creates its response message $P = R_r\#R_{t1}\#ID$—i.e., R_{t1} is used as random padding to bring the plaintext to 1,024 bits. Beginning at the least significant byte, the encrypted message $M = P^2 + R_{t2} \cdot n$ is computed using multiplication by convolution. Note that there is no modular reduction, so the message M is 2,208 bits long. The response bytes are then stored in SRAM memory. The WIPR algorithm structure allows encryption in a byte by byte on demand fashion, supporting devices with limited memory and also allowing the response to be generated in the background.

Our software implementation of the WIPR scheme had a very minor effect on the resources of the IAIK Demotag. The code section of a firmware design with the complete WIPR implementation requires 33,540 bytes, only 7.5 % (2,534 bytes) more than the standard version of the firmware without WIPR support. WIPR uses only 660 bytes of the available 4 KB of SRAM in its most RAM-heavy implementation.

3.4 Evaluation

Three possible scenarios were evaluated: First we evaluated a naïve implementation which does not cache the values of P and R_{t2} values in SRAM prior to the multiplication by convolution, but instead recalculates them on demand. Next, we tried caching the value of P before convolution. Finally, we tried caching the values of both P and R_{t2}. As depicted in Fig. 3, caching data on the heap has a dramatic effect on the execution time. The first scenario required 7 s to encrypt. The second scenario (caching only P) took 1.18 s, while the third scenario (caching both values prior to the convolution) sped the calculation to 180 ms. The convolution was implemented using the ATMega128's built-in hardware multiplier for all scenarios.

Figure 4 shows the value of T_{response} as a function of the amount of bits accessed in each block read operation. Recall that the computed result of 2,208 bits is read from the tag in a sequence of BLOCK_READ operations, and the block size is an implementation parameter of the reader's software. If a single 16-bit word is read in every round trip, the 138 read commands issued by the reader take 6.5 s to transfer the entire payload. On the other hand, a block size of 34 bytes (272 bits, the maximum size supported by our laboratory setup) allows the same payload to be transferred in only 0.46 s using 8 block reads. Upon further investigation, we found that the system's bottleneck is concentrated in the CAEN reader firmware, which takes about 40 ms to perform a single read

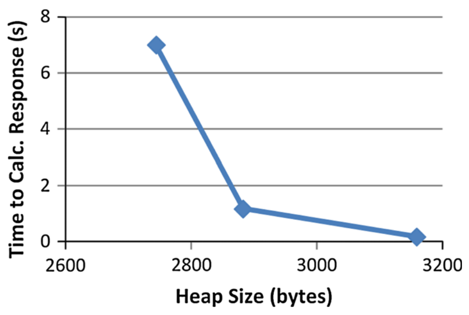


Fig. 3 T_{encrypt} as a function of heap size

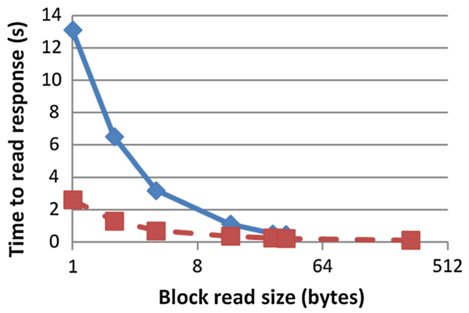


Fig. 4 T_{response} as a function of block read size. The *solid line* shows the measured time, while the *dotted line* is the calculated maximum

operation, regardless of the size of the data exchanged. This happens because the reader performs a fresh singulation protocol each time a tag is accessed, even if the tag is already in the SECURED state. The singulation process results in three unnecessary protocol round trips per command, dramatically reducing the I/O performance. The reader we used also powers up the radio circuit before each command and shuts it down again after the command concludes, further reducing performance. The dashed line in Fig. 4 shows an estimated performance of the same reader assuming the tag enters the read process powered on and singulated and that the reader does not repeat the singulation protocol between commands.

Table 1 estimates the values of T_{response} for a reader-tag link using an optimized EPC C1G2 flow. The estimation assumes the fixed cost of 40 ms related to powering up and singulating the tag was already incurred when the challenge was sent, so all the time incurred is related to the propagation delay of BLOCK_READ operations performed at 15 kbps. The current reader's configuration did not allow us to interfere with its order of execution or implement any protocol optimization.

3.5 Further optimizations

The results we measured are for a completely serialized operation, with the transmission of the ciphertext starting only after the last byte of ciphertext is calculated ($T_{\text{response}} = T_{\text{response}'}$). In addition, the current firmware of the Demotag supports writes of no more than 2 bytes and reads of no more than 34 bytes, resulting in 5 commands for writing the challenge and at least eight for reading the response. Finally, the off-the-shelf reader we evaluated communicates with tags in an inefficient way, as discussed previously. By implementing relatively minor tweaks to these limitations, we believe that the operation of the system can be dramatically improved. Table 2 shows the estimated performance gains of these optimization steps.

The first and immediate improvement could be achieved by better use of the air interface. By sending the challenge in a single 80-bit packet and keeping the tag in the SECURED state, we can reduce $T_{\text{challenge}}$ from 200 ms to an estimated 85 ms. Next, we can remove the unnecessary singulation steps by making sure the reader keeps the tag powered on and in the SECURED state throughout the response phase. In addition, we can pipeline the encryption and response transmission: Using WIPR, the tag can compute the ciphertext in 34-byte blocks and send them to the reader as soon as they are ready. The total time to perform the entire protocol in this case is equivalent to the time required to power on the tag and send it a challenge (85 ms), the time required for the tag to calculate the full response (180 ms) and the time required to send the final 34-byte chunk, which is ready only after encryption is finished (60 ms). Under these minor modifications, we estimate the entire protocol (including both identification and authentication) will take 325 ms.

For a more dramatic optimization, we can read the entire 276-byte response in a single read command which is issued immediately after the challenge is sent. This is possible since the tag can be designed to concurrently transmit the initial bytes of the ciphertext while it calculates the following ones. Since the data link takes only 112 ms to transfer 2,208 bits, the entire protocol time is dominated in this case by T_{encrypt}, leading to a total estimated time of 265 ms for the entire protocol.

Passive UHF tags communicate with the reader using modulated backscatter—instead of explicitly transmitting a signal back to the reader, the tag rapidly varies the impedance of its antenna, causing a variation in the phase or amplitude of the signal it reflects toward the reader [29]. Thus, in contrast to traditional radio-based systems, a passive UHF tag does not consume significantly more power while it is communicating with the reader. This property allows the tag to simultaneously encrypt and transmit without requiring a high peak power consumption.

3.6 Discussion

We consider the general-purpose 8-bit microcontroller present on the Demotag to be inherently slower than a custom designed ASIC implementation. Indeed, a naïve software implementation of the WIPR protocol which was functionally identical to the ASIC's implementation took an unacceptable 7 s to perform an encryption. However, as illustrated in Fig. 3, the addition of RAM significantly sped up the software implementation to the point that the entire encryption took 180 ms.

We found that the real bottleneck is in communication, with the dominant parameter being the number of round trips made by the reader. This problem is even more acute if the reader being used does not recognize the concept of sessions and repeats the singulation process with the tag every time it wishes to send it a command. It will be interesting to investigate whether other reader vendors handle multi-request ses-

Table 1 T_{response} as a function of block read size

Ciphertext bytes read per block	Measured T_{response} (s)	Estimated T_{response} (s)
1	13.1	1.02
2	6.5	0.57
4	3.2	0.34
14	1.1	0.18
28	0.52	0.15
34	0.46	0.14
276	Unsupported	0.12

Table 2 Performance of the complete WIPR protocol under various optimizations (all times are in ms)

Protocol Step	Current results	Partial pipelining	Full pipelining	Optimization step
$T_{\text{challenge}}$	200	85	85	Write all 80 bits of the challenge in a single round trip
T_{encrypt}	180	180	180	
T_{response}	460	180	112	Keep tag alive and singulated
T'_{response}	460	60	0	Pipeline encryption and transmission (via FIFO or via background calculation)
Total	840	325	265	

sions to a single tag more efficiently. If the tag can calculate the response bits faster than they are transmitted, optimal performance can be achieved by a pipeline design which transmits the ciphertext byte by byte as it is being generated within the context of a single large read command. This results in a very efficient performance and a saving of valuable RAM. Even when using minimal optimizations, the time required for the complete protocol is quite reasonable ($\approx$ 325 ms).

4 Detailed ASIC implementation

4.1 Objectives

In this part of the work, we wanted to test the feasibility of a realistic ASIC-based implementation of WIPR, beyond the sketches of [8, 18], and to evaluate whether indeed it fits the constraints of EPC C1G2 tags. Our first objective was to present a fully functional implementation of a WIPR tag in RTL, including data-path control logic and test-bench stimuli. The next objectives were to propose optimizations for gate cost and power consumption, implement and analyze the alternatives.

4.2 Design

4.2.1 Design flow and tool-chain

We used Cadence's Incisive tool suite version 11.10.006 [30] for compilation, elaboration, simulation and debug using the following commands—ncvhdl, ncvlog, ncelab, ncsim, irun. The RC tools-suite version 11.23.000 was used for synthesis and power analysis.

We selected TSMC's $TSMC65LP$ 65nm low-power process silicon process [31] due to our experience and its maturity and reliability. Virage [32] was selected to provide standard cell libraries for the above process.

The reference gate size (used to convert area to gate equivalents) for this technology is $1.8\,\mu\text{m} \cdot 0.8\,\mu\text{m} = 1.44\ \ \mu\text{m}^2$, and VDD of 1.08 V. For reference, dynamic power dissipation, a single data flip-flop of the simplest kind (positive-edge triggered, q-only) consumes an energy of 0.0188 pJ when clocked and both input (D) and output (Q) are toggling. Assuming that an RFID tag has an average power of 20 μW and a clock rate of 1 MHz, this allows for approximately 1,000 flip-flops to toggle every clock period.

4.2.2 Original hardware architecture of a WIPR tag

Our starting point was the hardware architecture first presented in [8] and [18], with chosen protocol parameters of $n = 1{,}024$, $\alpha = 80$, $\beta = 80$ to achieve an 80-bit security level, comparable with 1,024-bit RSA [33]. The properties and total resource requirements of this implementation sketch are presented in Table 3. Note that the numbers for area and power in this table refer to an implementation with a different process, standard cell libraries and tools, and are therefore not directly comparable with the implementation alternatives presented in this work.

The protocol requires two online multiplications: $M = P^2 + r \cdot n$. This multiplication step can readily be performed on a multiply-accumulate (MAC) register by convolution. Assuming a word size of 8 bits (byte), a single multiply-accumulate register can carry out this multiplication in about 2^{16} steps using 25 bits of carry memory (enough to accumulate 512 8-bit multiply operations). The ciphertext can be transmitted byte by byte (LSB first) as soon as it is computed, minimizing the need for intermediate registers. The data-path architecture is depicted in Fig. 5.

The public key (n) is selected as a composite number with a predefined upper half, thus reducing the ROM cost by half (see for example [34]), by setting the upper half to a value easily represented in hardware.

Table 3 Properties of the original ASIC design of WIPR, presented in [18]

Cipher strength	1,024 bits
Challenge size	80 bits
Response size	2,208 bits
Payload capacity	864 bits
Area (GE)	4,682
Total current draw (μA)	14.2

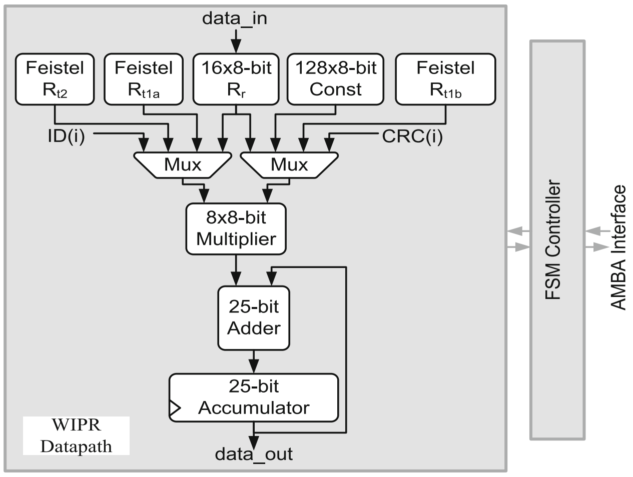


Fig. 5 Data-path architecture of WIPR

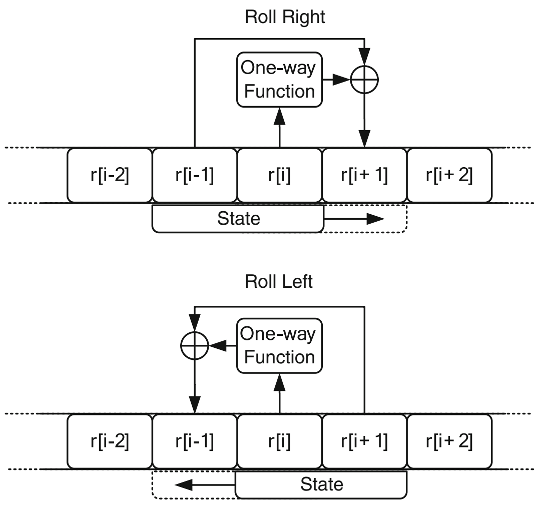


Fig. 6 Creating a reversible stream cipher using a Feistel structure and an arbitrary OWF

As suggested in [8], we replace the long random strings generated by the tag with pseudo-random outputs from a reversible stream cipher. Instead of storing the entire random string, we store short seed values (one for R_{t2} and two for each end of R_{t1}, denoted R_{t1a} and R_{t1b} in Fig. 5), and use the stream cipher operation to evolve them over time. Due to the sequential nature of accesses to the random strings, only a single "roll left" or "roll right" operation is required for each convolution step. The reversible stream cipher was implemented using a Feistel structure [27] and a representative one-way function (OWF), as shown in Algorithm 4.1 and Fig. 6.

Algorithm 4.1 Rolling algorithm used to create pseudo-random sequence

```
Roll Right:
    left_in <= right_out;
    right_in <= left_out xor oneway(right_out);
Roll Left:
    right_in <= left_out;
    left_in <= right_out xor oneway(left_out);
```

The random bit string R_r which is the challenge provided by the reader must be stored in a RAM due to the random access nature of the read transactions.

4.3 Implementation

The WIPR tag was implemented in RTL, written in the VHDL hardware description language. The design hierarchy of the WIPR tag includes a top level which is the test-bench stimuli, encapsulating the control logic FSM (finite state machine) which controls the data path through a common AMBA [35] wrapper. The data path itself has a lower hierarchy of modules—arithmetic (multiplier, adder, accu-

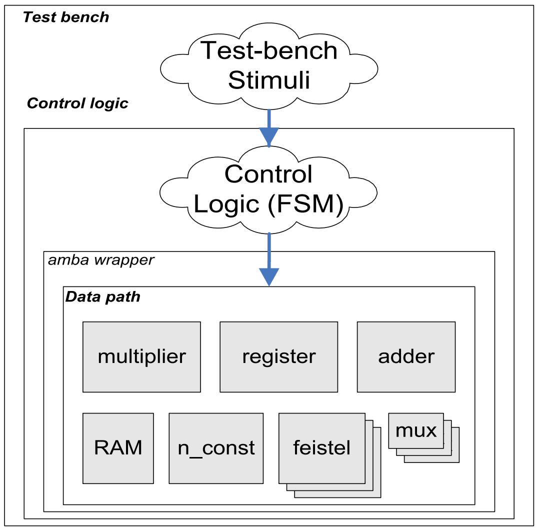


Fig. 7 Design hierarchy of the WIPR tag

mulator register), logic (multiplexers, free logic) and storage (RAM, n_const, Feistel). This hierarchy is depicted in Fig. 7.

The data-path module's interface which is controlled by the control logic includes the following types of ports: addresses (for controlling the various memory blocks), enable signals, select lines (for controlling the multiplexers), input buses for external data (challenge) and internal data (e.g., tag ID) and various controls such as shift and reset.

During the course of the RTL implementation, we needed to overcome three major issues for the design to work (before any optimization stage):

1. A single port RAM was not enough, due to the fact that at some steps of the calculation of P^2, different bytes of

R_r are required to be multiplied by each other. The trivial (though inefficient) solution is placing two identical instances of this single port RAM—one for each version of P. This solution was later optimized (see Sect. 4.4).
2. At some steps of the calculation of P^2, the strings R_{t1a} and R_{t1b} are required to be multiplied by each other, therefore should both move at the same step (either left or right). However, only a single Feistel logic module exists in the design, so they cannot both move at the same cycle. Adding another Feistel logic is a costly alternative; therefore, the control was altered to allow a two-cycle step only for those specific cases.
3. At each cycle, the Feistel logic outputs two 48-bit halves, but only a single byte from the Feistel state is fed to the multiplier. The function which reduces these two halves into a single byte must be symmetric such that it returns the same value even if the direction was flipped. We used the following symmetric function: $out = xor(left[47:40], right[47:40])$.

4.4 RTL optimizations

Given a functional, bit-accurate design which complies with the properties of the protocol, the next stage was optimizing it. The optimizations concentrated mainly, but not solely, on the data-path module. The first-order optimization parameter was area, while the second-order optimization parameter was power. Speed was not found to be a real constraint, as described below.

Three main improvements were introduced:

1. RAM reads—As mentioned above, the single RAM had to be duplicated for the design to be functional. Two main optimization alternatives were considered:
   (a) A two-cycle read step—each multiplication which requires two different bytes of R_r simultaneously will happen during two cycles, reading the multiplicand in the first cycle and reading the multiplier and multiplying it by the multiplicand in the second cycle. This solution requires some added complexity to the control logic, a few more cycles to the protocol and more importantly a temporary register to hold the multiplicand which was read at the first cycle. This implementation was not as efficient as the next one.
   (b) A dual-port-read RAM—allowing two cells (bytes) of the RAM to be read simultaneously through a double interface. Typical RAM architectures (SRAM, DRAM) do not allow parallel access to all their bit cells. However, since the RAM was small enough to be implemented with sequential logic (flip-flops), the double read interface was rather cheap—only another set of read multiplexers was required.
2. RAM writes—R_r is stored only once, at the initialization process of the protocol before calculations take place so a serial-in random-out implementation was found to be more efficient than the typical symmetric (read/write) RAM which was originally designed. There was no address required for write transactions as they entered the RAM serially, similar to a typical shift register. Also, a single write port is all that is needed and writes could be separated in time from reads, so the existing the read port can also serve as a bi-directional write port.
3. The security level required 80 bits, but in the original design, there were 16 bytes. Reducing it to 10 bytes saved valuable area (even though 10 is not a power of 2, so each read multiplexer still required a 4-bit select line).

To summarize, out of the several design alternatives, the chosen RAM architecture consisted of two parallel, random access read ports and one single serial write port as depicted in Fig. 8.

4.4.1 Clock gating

Clock gating is a popular technique for reducing dynamic power dissipation by adding more logic to a circuit to prune the clock tree. Pruning the clock disables portions of the circuitry so that the flip-flops in them do not have to switch states, thus do not consume dynamic power. Clock gating works by taking the enable conditions attached to registers, and uses them to gate the clocks. Clock gating can save significant die area as well as power, since it removes large numbers of multiplexers, or flip-flops with enable ports, replacing them with clock gating logic which is usually a dedicated optimized library cell.

The synthesis tool *rc* claims to identify these enable conditions automatically and replace them with CG cells. Therefore, our first step was having the tool perform its semi-automatic clock gating process, and indeed all the D-FF cells which included an enable port were converted to D-FF

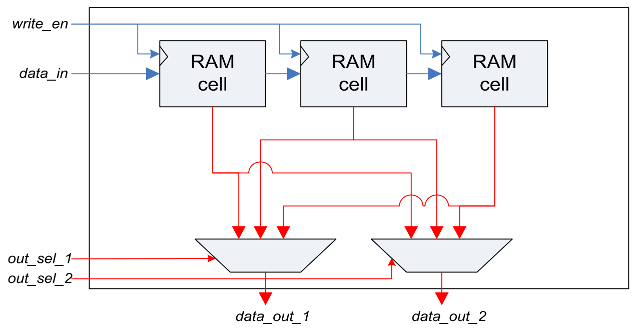


Fig. 8 Illustration of the selected RAM architecture (an example with three RAM cells). The *write-path* is indicated in *blue*. *Read-paths* are indicated in *red* (color figure online)

with no enable port. However, this semi-automatic process depends on the tool's static analysis of the design and does not take into account implicit information which the designer is aware of. For example consider the multiplexer implementation described in Algorithm 4.2:

Algorithm 4.2 Example of muxing between buses according to a select control signal

```
if (Rt1[a]_moves) then
    mux_select <= "00";
else if (Rt1[b]_moves) then
    mux_select <= "01";
else // select $R_{t2}$
    mux_select <= "10";
```

When both $R_{t1}[a]$ and $R_{t1}[b]$ do not move, the mux selects R_{t2} even when it does not need to move (when nothing moves). In that case, the tool lacks the explicit enable condition which can be automatically translated into clock gating logic when R_{t2} is not actually moving. We implemented manual clock gating to capitalize on this.

Another manual clock gating was explicitly implemented for the result register (accumulator), such that when the multiplication result equals 0 (or alternatively, when one of the multiplier's inputs equals 0), the accumulation register is not enabled.

4.4.2 Reset logic

Initially, some of the sequential logic had been given an asynchronous reset. However, functionally it is not necessary for the circuit to be reset in that manner, so all the flip-flops were eventually provided with a synchronous reset.

The accumulator register which had an asynchronous reset was upgraded to receive a synchronous reset through a reg-reset control signal initiated by the control logic, resulting in 13 % area decrease. More specifically, it allowed the synthesis to replace the $FDPRBQ$ library cells (D-Flip-Flop, positive-edge triggered, *lo-async-clear*, q-only) with $FDPQ$ cells (D-Flip-Flop, positive-edge triggered, q-only).

The Feistel states for $R_{t1}[a]$, $R_{t1}[b]$ and R_{t2} need also an initial seed value to start with. In our baseline design, this was implemented using flip-flops with asynchronous set/reset. We optimized the design via a control sequence which loads the random seed values into the Feistel states using existing data paths. These random data are loaded 48 bit per cycle over 6 cycles to the 3×96 bit Feistel state registers, through an input multiplexer which is already connected to the Feistel logic. This allowed to replace $FDPRBQ$ cells (lo-async-clear) and $FDPSBQ$ cells (lo-async-set) with $FDPQ$ (no async-set/clear) which translates to 13 and 17 % area reduction accordingly.

4.4.3 Move-flip Feistel architecture

Each of the strings $R_{t1}[a]$, $R_{t1}[b]$ and R_{t2} has an instantaneous Feistel state composed of two halves– right and left, 48 bit each. As the multiplications of the long strings are done in a convolutional manner over small chunks (a single byte each), the corresponding memory accesses to the long strings are of a sequential nature. Flipping the direction of movement (from right to left and vice versa) for a given string was initially performed inside the Feistel logic using a set of four 2:1 48-bit multiplexers to control which half is fed to which part of the logic. This baseline architecture is depicted in Fig. 9.

We observed that when a given Feistel state starts rolling in a certain direction, it keeps rolling that way until the current ciphertext byte is calculated, then flipping its direction and rolling the other way. We also notice that the rolling operation is completely symmetric. So, if we can flip directions cheaply, only once per ciphertext byte, and get rid of the large multiplexers we can save significant area and power.

This was the incentive to get rid of left–right architecture and replace it by a novel move-flip notion—a string moves in a certain direction (whatever that is) for many cycles and is then flipped in a single extra cycle. The calculations now take slightly longer due to the extra cycle per flip, but the extra logic for flipping directions is very cheap, much cheaper than the above-mentioned multiplexers. This new architecture is depicted in Fig. 10, which also presents the above-mentioned synchronous reset logic which feeds in the RAND_IN bus upon a reset condition.

The control logic was altered accordingly to provide the flip and move controls instead of the roll-right, roll-left controls.

4.5 Evaluation and discussion

4.5.1 Data analysis

The activity-based reports of the gate-level data-path module were examined and compared according to the three parameters (in descending priority order): area, power and speed. We compared three implementations:

1. **Baseline**—'naïve' implementation, based on the proof-of-concept implementation, after making the necessary fixes and additions to make it functionally correct and identical with the reference model.
2. **RTL optimized**—including optimizations which do not require knowledge of the WIPR protocol:
   (a) Semi-automatic clock gating using the *rc* tool
   (b) Simple dual-port RAM

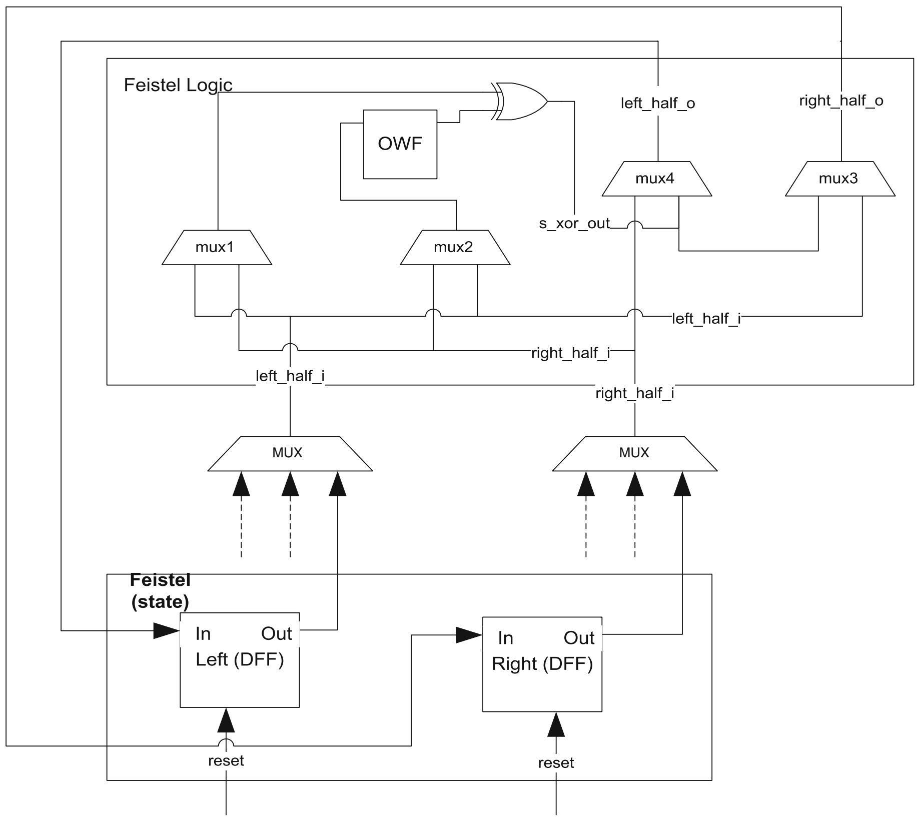


Fig. 9 Baseline architecture of Feistel state and logic

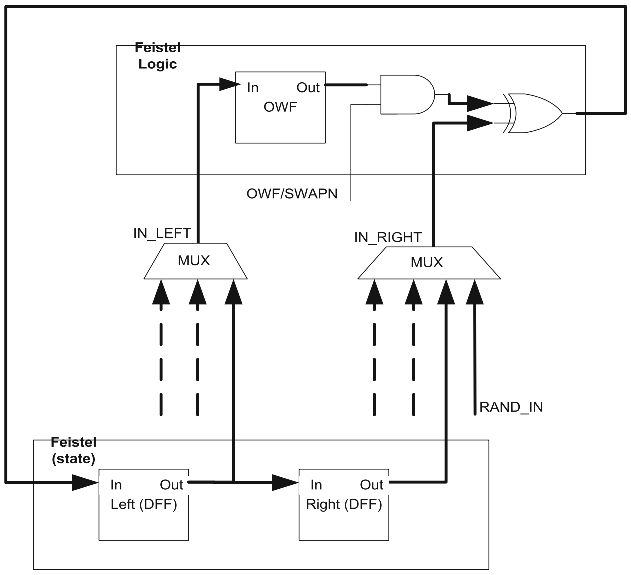


Fig. 10 New architecture of Feistel state and logic

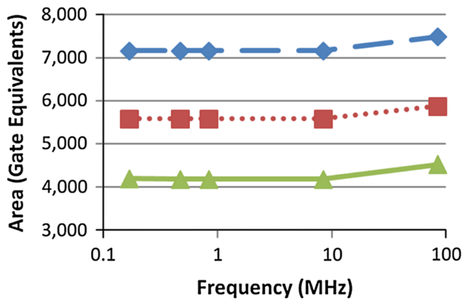


Fig. 11 Area as function of speed for the three optimization levels—baseline (*dashed*), RTL optimized (*dotted*) and fully optimized (*solid*)

Table 4 Summary of area for the three implementations

Area	Gate Equivalents	%
Baseline	7,160	100
RTL optimized	5,579	78
Fully optimized	4,184	58

3. **Fully optimized**—including all relevant optimizations, detailed in Sect. 4.4

The graphs in the following sub-sections present the area and power as function of speed for the three different levels of optimization.

4.5.2 Area improvements

Figure 11 and Table 4 show the area versus speed for the three implementations. Each step provided a 20–25 % improvement over the previous one with a bottom line of

Table 5 Breakdown of the data-path area for its composing sub-modules

Sub-module	Area (gate equivalents)			Fully optimized/baseline (%)
	Baseline	RTL optimized	Fully optimized	
R_{t2}Feistel state	767	579	495	65
$R_{t1}[a]$ Feistel state	767	579	495	65
$R_{t1}[b]$ Feistel state	771	579	495	64
Feistel logic + OWF	1,374	1,376	906	66
R_r Memory	2,381	1,365	710	30
Constant n	208	208	208	100
Multiplexers	99	99	99	100
Multiplier	402	402	402	100
Adder	115	115	115	100
Accumulator	203	203	184	91
Free logic	74	74	76	102
Total data-path area	7,160	5,579	4,184	58

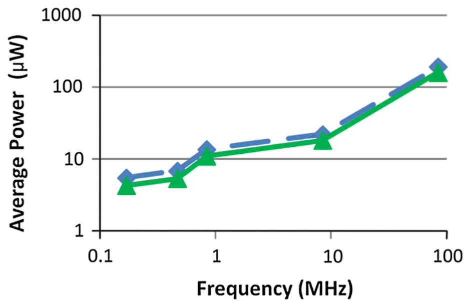


Fig. 12 Average power (static + dynamic) for two optimization levels—baseline (*dashed*) and fully optimized (*solid*)

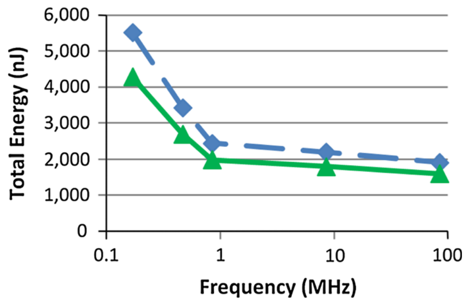


Fig. 13 Total energy consumption for two optimization levels—baseline (*dashed*) and fully optimized (*solid*)

4,184 gate equivalents, which stand for a 42 % improvement over the baseline implementation.

For a detailed analysis of the results, we observed the breakdown of the data-path design into its sub-blocks to see what is the improvement factor for each sub-module and validate it with our initial assumptions. The detailed list is shown in Table 5. This table shows that the pure sequential parts (the Feistel states and the accumulator) improved by 10–35 %, mainly due to clock gating and new reset logic. The Feistel logic (including the OWF) improved by 1/3, mainly due to the new move-flip architecture. The RAM improved significantly by 70 % due to the series of improvements detailed in Sect. 4.4, while the free logic and arithmetic operations did not improve at all as none of the applied methods was related to them.

As for speed dependency, when the speed is higher, the synthesis tends to use cells with larger drive strength which is also larger in size, thus increasing the area of the circuit. The maximum speed is then limited also by the driving strength of the library cells in hand. This explains the increase in area seen in Fig. 11 as the clock rate approaches 100 MHz.

4.5.3 Power/energy improvements and speed trade-offs

The next graphs show power and energy as function of speed. The measured power in Fig. 12 is the average combined (dynamic and static) power for the duration of the whole simulation (not instantaneous power). The measured energy in Fig. 13 is the total energy spent during the entire simulation. The performance of the RTL-optimized version is essentially equal to that of the fully optimized version and is omitted for clarity.

As mentioned in [36], the power dissipation of a digital circuit is determined by the following formula $P = P_d + P_s = C \cdot V^2 \cdot f + P_s$, where P_d is the dynamic power dissipation, P_s is the static power dissipation, f is the circuit frequency, V is the supply voltage and C is a process-dependent constant. Thus, if the dynamic power dissipation is much larger than the static power dissipation, which is typically the case when the circuit is operating, we can say that the total power dissipation is linear with the frequency. A second-order phenomenon is an increase in the static power when the dynamic power is high, due to temperature effects (heating causes more leakage).

The absolute numbers for our design are shown in the following results:

1. Energy consumption of 1.5–3 μJ in the interesting speed range (where area stays constant) and specifically 2 μJ for a clock frequency of 467 KHz, which corresponds to a protocol duration of 180 ms.
2. Power dissipation of less than 20 μW for clock frequencies below 800 KHz.
3. Current draw of 4.2 μA at 100 KHz, compared to 14.2 μA reported for a similar frequency in the proof-of-concept design of [18].

Comparing the three implementations led to the following observations. First, the average power and energy improvement for the fully optimized implementation over the baseline implementation are around 20 %. Second, it can be seen on the power graph (Fig. 12) that the power is linear with the frequency for all speed ranges, as expected. Note that the x-axis is logarithmic, and hence, a linear dependence appears as an exponential curve. Third, the energy is increasing with simulation duration as the static power (leakage) is accumulated in time, while the dynamic power contribution stays approximately the same.

4.5.4 Recommended working point

Given the above results, we can summarize:

1. Any speed below 10 MHz is slow enough not to incur in area penalty.
2. Any speed below 1 MHz is slow enough not to surpass the 30 μW power budget listed by [3], as seen in Fig. 12.

Our recommendation is to work in the 100 KHz–1 MHz frequency range, depending on the application. This translates to a protocol duration of 800–80 ms, correspondingly. In particular for a clock rate of 467 KHz, the total energy consumption is 2 μJ and the average power dissipation is 11 μW, values which were shown in [3] to be suitable for typical passive UHF RFID tags up to a range of 8.5 m.

The EPC standard establishes time constraints for protocol execution. For example, there is a T1 timing boundary, typically on the order of 20 μs, that establishes the maximum delay from the interrogator transmission to tag response. Designing a WIPR implementation that can perform an entire encryption within this duration would require a high clock rate and increased power consumption. To allow a WIPR-based tag to comply with the strict timing requirements of the EPC standard while remaining at a low clock rate, the WIPR protocol was designed to employ a challenge-response mechanism based on memory-mapped I/O [37]. Under this design, the WIPR challenge is written to the tag in one EPC command, while the response is read back in one or more additional commands. Thus, the WIPR tag can always precalculate a few bytes of its response and store them in RAM, making them immediately available to the reader—the first precalculation is performed immediately after the challenge has been written to the tag, and subsequent precalculations take place immediately after the tag has finished sending a ciphertext block to the reader. Our software implementation, which used this mechanism, was tested without issue against a standard EPC reader with standard timing parameters (see Sect. 3). As shown in Sect. 3.4, the amount of ciphertext bytes sent to the reader in each read operation has a direct effect on the overall throughput of the tag. Thus, a trade-off exists between the RAM consumption of the tag (and thus its overall chip area) and the tag's read rate.

5 Conclusions

Public-key cryptography was previously claimed to be impractical for RFID tags. The reasons for this claim were the high cost (in gate count and power consumption) of public-key encryption and its slow performance when compared to secret-key ciphers or hash functions. In our software implementation, we demonstrated that even on an inherently slow 8-bit microcontroller, encryption speed was not a bottleneck. We were able to run the entire encryption in 180 ms using only standard EPC commands.

We found that the real bottleneck is in communication, with the dominant parameter being the number of round trips made by the reader. This problem is even more acute if the reader being used does not recognize the concept of sessions and repeats the singulation process with the tag every time it wishes to send it a command. It will be interesting to investigate whether other reader vendors handle multi-request sessions to a single tag more efficiently. If the tag can calculate the response bits faster than they are transmitted, optimal performance can be achieved by a pipeline design which transmits the ciphertext byte by byte as it is being generated within the context of a single large read command.

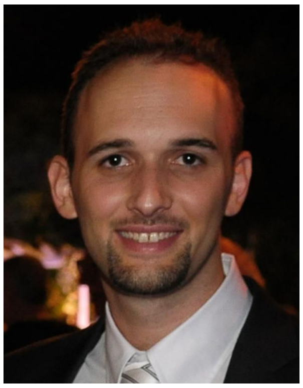
We also presented an optimized WIPR implementation which is small enough to fit on an RFID tag: Using a variety of hardware design optimization techniques, we were able to identify a working point that is well within a tag's power and area budgets, and is fast enough for the intended application. We conclude that the public-key approach is a viable design alternative for supply-chain RFID EPC tags.

Acknowledgments We thank the anonymous reviewers for their helpful and instructive comments.

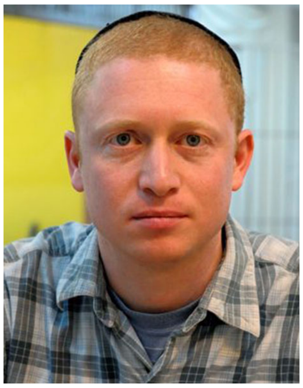
References

1. Epcglobal inc.: EPC radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 MHz–960 MHz, version 1.0.9. Sept (2005)
2. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and privacy aspects of low-cost radio frequency identification systems. In: Hutter D., Müller G., Stephan W., Ullmann M., (eds.) SPC, volume 2802 of Lecture Notes in Computer Science, pp. 201–212. Springer (2003)
3. Dobkin, D.M.: The RF in RFID, 2nd edn. UHF RFID in Practice, Newnes (2012)
4. Juels, A., Weis, S.A.: Authenticating pervasive devices with human protocols. In: Shoup, V. (ed.) Advances in Cryptology—CRYPTO 2005, Lecture Notes in Computer Science, vol. 3621, pp. 293–308. Springer, Berlin (2005)
5. Gaubatz, G., Kaps, J-P., Ozturk, E., Sunar, B.: State of the art in ultra-low power public key cryptography for wireless sensor networks. In: Third IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 146–150. (2005)
6. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: Quisquater J-J., Joye M. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2004: 6th International Workshop, LNCS, vol. 3156, pp. 357–370 Springer (2004)
7. Nohl, K., Plötz, H.: MIFARE—little security, despite obscurity. Technical report, 24th Chaos Communication Congress (2007)
8. Oren, Y., Feldhofer, M.: WIPR—public-key identification on two grains of sand. In: Dominikus S., (ed.) Workshop on RFID Security, pp. 15–27 (2008)
9. Rabin, M.O.: Digitalized signatures and public-key functions as intractable as factorization. (1979)
10. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
11. Naccache, D.: Method, sender apparatus and receiver apparatus for modulo operation. US Patent 5,479,511, 26 Dec (1995)
12. Shamir, A.: Memory efficient variants of public-key schemes for smart card applications. In: Advances in Cryptology-EUROCRYPT'94, pp. 445–449. Springer (1995)
13. Shamir, A.: SQUASH-a new MAC with provable security properties for highly constrained devices such as RFID tags. In: Fast Software Encryption, pp. 144–157. Springer (2008)
14. Finiasz, M., Vaudenay, S.: When stream cipher analysis meets public-key cryptography. In: Selected Areas in Cryptography, pp. 266–284. Springer (2007)
15. Furbass, F., Wolkerstorfer, J.: ECC processor with low die size for RFID applications. In: IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007. pp. 1835–1838. IEEE (2007)
16. Blass, E.-O., Kurmus, A., Molva, R., Noubir, G., Shikfa, A.: The f_f-family of protocols for RFID-privacy and authentication. IEEE Trans. Dependable Secur. Comput. **8**(3), 466–480 (2011)
17. Chien, H.-Y.: SASI: a new ultralightweight RFID authentication protocol providing strong authentication and strong integrity. IEEE Trans. Dependable Secur. Comput. **4**(4), 337–340 (2007)
18. Oren, Y., Feldhofer, M.: A low-resource public-key identification scheme for RFID tags and sensor nodes. In: Basin, D.A., Capkun, S., Lee, W. (eds.) WISEC, pp. 59–68. ACM, New York (2009)
19. Wu, J., Stinson, D.R.: How to improve security and reduce hardware demands of the WIPR RFID protocol. In: IEEE International Conference on RFID, 2009. pp. 192–199. IEEE (2009)
20. Arbit, A., Oren, Y., Wool, A.: A secure supply-chain RFID system that respects your privacy. Pervasive Computing, IEEE, Accepted for publication
21. Najera, P., Roman, R., Lopez, J.: User-centric secure integration of personal RFID tags and sensor networks. Secur. Commun. Netw. **6**(10), 1177–1197 (2013)
22. Plos, T., Michael, H., Feldhofer, M., Stiglic, M., Cavaliere, F.: Security-enabled near-field communication tag with flexible architecture supporting asymmetric cryptography. IEEE Trans. VLSI Syst. **21**(11), 1965–1974 (2013)
23. Wenger, E., Unterluggauer, T., Werner, M.: 8/16/32 shades of elliptic curve cryptography on embedded processors. In: Paul G., Vaudenay S., (eds.) INDOCRYPT, volume 8250 of Lecture Notes in Computer Science, pp. 244–261. Springer (2013)
24. Batina, L., Seys, S., Singelée, D., Verbauwhede, I.: Hierarchical ECC-based RFID authentication protocol. In: Juels A., Paar, C. (eds.) RFIDSec, volume 7055 of Lecture Notes in Computer Science, pp. 183–201. Springer (2011)
25. Aigner, M., Plos, T., Ruhanen, A., Coluccini, S.: Secure semi-passive RFID tags—prototype and analysis. Technical report, BRIDGE Project (2008)
26. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC, Boca Raton (1996)
27. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. **17**(2), 373–386 (1988)
28. Barthel, H.: UHF RFID regulations. http://www.oecd.org/sti/interneteconomy/35472969.pdf (2006)
29. Finkenzeller, K.: RFID Handbook : Fundamentals and Applications in Contactless Smart Cards and Identification. Wiley, New York (2003)
30. Cadence incisive tool suite. http://www.cadence.com/products/pages/default.aspx
31. TSMC65LP 65nm low-power process silicon process. http://www.tsmc.com/english/dedicatedFoundry/technology/65nm.htm
32. Virage logic standard cell libraries. http://www.synopsys.com/dw/ipdir.php?ds=dwc_standard_cell
33. Lenstra, A.K., Verheul, E.R.: Selecting cryptographic key sizes. J. Cryptol. **14**(4), 255–293 (2001)
34. Johnston, A.M.: Digitally watermarking rsa moduli. Cryptology ePrint Archive, Report 2001/013. http://eprint.iacr.org/2001/013 (2001)
35. Advanced microcontroller bus interface open specification. http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
36. Finkenzeller, K.: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-field Communication. Wiley, New York (2010)
37. Arbit, A., Oren, Y., Wool, A.: Toward practical public key anti-counterfeiting for low-cost EPC tags. In: 2011 International IEEE Conference on RFID, vol. 4, pp. 184–191 Orlando, USA (2011)

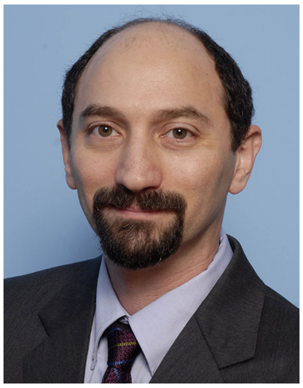
Alex Arbit is a Hardware & Electronics Engineer at Tel Aviv University. His research interests include real-world cryptography and low-resource cryptographic constructions for lightweight computers. Arbit is an MSc graduate in electrical engineering from Tel Aviv University.



Yoel Livne received his B.Sc. degree (Cum Laude) in Computer Science and Electrical Engineering from Tel Aviv University, Israel, in 2005. He received his M.Sc. degree in Electrical Engineering from Tel Aviv University, Israel, in 2013. He is currently a team leader of ASIC design for the physical layer of an advanced LTE baseband modem, working for Altair semiconductor, Israel, since 2006. His interests include logic design, computers architecture, digital communications, and digital signal processing.



Yossef Oren is a post-doctoral research scholar in the Department of Computer Science at Columbia University. His research interests include power analysis attacks and countermeasures, low-resource cryptographic constructions for lightweight computers, and real-world cryptography. Oren has a PhD degree in electrical engineering from Tel Aviv University.



Avishai Wool is cofounder of the AlgoSec Systems (formerly Lumeta) network security company and is an associate professor at Tel Aviv University's School of Electrical Engineering. His research interests include firewall technology, computer, network, and wireless security, smart card and RFID systems, and side-channel cryptanalysis. Wool has a PhD degree in computer science from the Weizmann Institute of Science, Israel. He is the creator of the AlgoSec Firewall Analyzer, a senior member of IEEE, and a member of the ACM and Usenix.