Lightweight Cryptography and RFID: Tackling the Hidden Overhead

Axel Poschmann[1], Matthew J.B. Robshaw[2], Frank Vater[3] and Christof Paar[4]
[1] Division of Mathematical Sciences, Nanyang Technological University
21 Nanyang Link, 637371 Singapore
[e-mail: aposchmann@ntu.edu.sg]
[2] Orange Labs, 38-40 rue du Général Leclerc, Issy les Moulineaux, France
[e-mail: matt.robshaw@orange-ftgroup.com]
[3] Innovations for High Performance Microelectronics, Frankfurt/Oder, Germany
[e-mail: vater@ihp-microelectronics.com]
[4] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
[e-mail: christof.paar@rub.de]
*Corresponding author: Matthew J.B. Robshaw

Received March 16, 2010; revised April 16, 2010; accepted April 17, 2010; published April 29, 2010

Abstract

The field of lightweight cryptography has developed significantly over recent years and many impressive implementation results have been published. However these results are often concerned with a core computation and when it comes to a real implementation there can be significant hidden overheads. In this paper we consider the case of cryptoGPS and we outline a full implementation that has been fabricated in ASIC. Interestingly, the implementation requirements still remain within the typically-cited limits for on-the-tag cryptography.

Keywords: Lightweight cryptography, RFID, cryptoGPS, ASIC

The research was supported in part by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03.

DOI: 10.3837/tiis.2010.04.002

1. Introduction

Radio-frequency identification (RFID) tags are becoming a part of our everyday life and a wide range of applications from the supply chain to the intelligent home are often described in the literature. Yet, at the same time, security and privacy issues remain a major issue, not least in the battle against counterfeit goods with pharmaceutical products and even engine components in the automotive and aeronautic industries at risk [1].

It has long been recognized that cryptographic techniques might be used to help alleviate these problems. However they have all too often been considered as too expensive to implement, or too unsuited to the enviroment of use. Over recent years this view has begun to change and there have been substantial advances in cryptographic design, for instance in new block ciphers such as PRESENT [2]. But as well as the advances we might have expected in symmetric cryptography- which is typically viewed as the lightweight choice - there has been a growing understanding of which asymmetric techniques are available and how they might best be implemented. Indeed, given the essential nature of an RFID-based deployment with many (potentially unknown) players being involved - i.e. an open rather than a closed system - lightweight public-key cryptography could be viewed as a particularly attractive technology. Some of the more recent implementation results in the literature have been very impressive. The oft-cited opinion is that there are around 2000-3000 gate equivalents (GE) available for on-tag security features,[1] and despite this representing a formidable challenge, several algorithms claim to achieve this.

In this paper we highlight a problem with many of these estimates and we observe that figures are often given for the cryptographic core of a computation. For instance, estimates for the feasibility of elliptic curve cryptography might consider just the elliptic curve operation while implementation results for cryptoGPS are focused on the protocol computations [3][4]. This means that when it comes to a real implementation there can be significant hidden overheads. The main purpose of this paper is to highlight this issue, but also to reexamine the case of one particular proposal, that of cryptoGPS. To do this we will describe a full implementation of cryptoGPS which includes all the additional functionality that would be required in a real deployment. Further, noting that implementation results for lightweight cryptography are often derived from an FPGA implementation or ASIC synthesis tools, we have gone one step further and we report on the results of the full ASIC fabrication of a fully-supported version of cryptoGPS.

2. Related Work

Over recent years a lot of work on public key cryptography for RFID tags has centered around elliptic curves. A comparison between different ECC implementations is not always easy because the choice of the underlying curve determines both the efficiency and security of the algorithm. However no implementation has been published so far that comes under 5000 GE which would, even then, be too great for passive RFID tags. Instead several elliptic curve implementations with a significantly lower security level than 80-bit exist, but their size lies in the range of 10 000 GE or above [5][6][7].

Gaubatz et al. have investigated the hardware efficiency of the NTRUencrypt algorithm with

[1] The gate equivalent (GE) is a unit of area and is equivalent to the physical space occupied by a logical NAND gate for the given manufacturing process.

the following parameter set (N, p, q) = (167, 3, 128) that offers a security level of around 57 bits [8][9][10]. Though their implementation requires only 2850 GE, it takes 29 225 clock cycles, which translates to 292 ms for the response to be computed at the typical clocking frequency of 100 KHz. Further, it is noteworthy that more than 80% of the area is occupied with storage elements and that already a bit serial datapath is used. This implies that the opportunities for future improvement are very limited. Oren et al. propose a public key identification scheme called WIPR [11]. Their ASIC implementation requires 5705 GE and 66 048 clock cycles, though a proposed optimization suggests a reduced area requirement of around 4700 GE [12]. In this paper, however, we will concentrate on the cryptoGPS scheme. The name GPS is derived from the inventors Girault, Poupard, and Stern, but the term cryptoGPS is increasingly used to avoid confusion with the geographical positioning system.

A description of the scheme and numerous variants can be found in [13][14][15]. It is standardised within ISO/IEC 9798-5 [16] and listed in the final NESSIE portfolio [17]. Some initial analysis of the ASIC implementation requirements for the elliptic-curve based variant of the cryptoGPS identification scheme are available [3][4]. There implementation estimates range between 300–900 GE, but they are only concerned with the core on-tag operation in cryptoGPS. A more complete implementation in the form of a fully-functioning FPGA prototype is described in [18]. But in moving from an FPGA implementation to a dedicated RFID-tag implementation there are many differences and complications to consider and this is one of the goals behind this paper.

2.1 This Paper

This paper is organized as follows. First we introduce the cryptoGPS identification scheme and we provide a summary of some of the optimizations that are available. Then we turn to the question of how an implementation would look in reality and what additional functionality - over and above the core cryptoGPS computations - would be required. In Section 3 we describe the engineering and design challenges that needed to be overcome in designing an ASIC that incorporates three different (two round-based and one serialized) variants of the cryptoGPS scheme. In Section 4.3 we discuss our results before we draw our conclusions in Section 5.

3. The cryptoGPS Identification Scheme

A public key identification scheme allows the possessor of a secret key to prove possession of that secret by means of an interactive protocol [19]. Thus, in the case of an RFID deployment, the tag would "prove" to a reader that it contains a tag-specific secret and the reader is thereby assured that the tag is genuine. Only a device possessing the key could provide the necessary responses. While at first sight this might appear to be quite a specialised functionality, for instance we don't have the conventional public key services of encryption or digital signatures[2], interactive identification schemes have been deployed widely. In particular the cryptoGPS scheme seems to allow a particularly compact implementation on the tag. This allows us to consider RFID tags with public key capability which can open up previously unavailable application areas.

3.1 Overview of cryptoGPS

There are many variants and optimizations of cryptoGPS. One variant uses RSA-like moduli but

[2] Identification schemes can be converted to signature schemes in a standard way [19] though some computational advantages can be lost.

here, and in **Fig. 1**, we illustrate the essential elements of cryptoGPS using elliptic curve operations. For the system as a whole there are the shared parameters of the elliptic curve C and a base point P on that curve. These are not required on the tag and so they do not impact our implementation. The cryptoGPS secret key s is stored on the tag and is assumed to be σ bits in length. The public key $V = -sP$ is an elliptic curve point and we assume that this is available to the reader by some mechanism. To take full advantage of the optimizations described in Section 2.2 the tag is required to support a pseudo-random generator (PRG) that uses a tag-specific secret key k. Note that k is required at initialisation to perform some pre-computation, but afterwards k is never needed outside the tag. Several parameter sizes need to be set and the appropriate choices will depend on the application and the security level. We have already mentioned σ which for a security level of 80 bits is set to $\sigma = 160$. The length of the challenge c from the reader to the tag will be denoted δ and the particular value will depend on different optimizations. The length of the pseudo-random numbers r_i will be denoted ρ and it is a requirement of cryptoGPS that we set $\rho = \sigma + \delta + 80$.

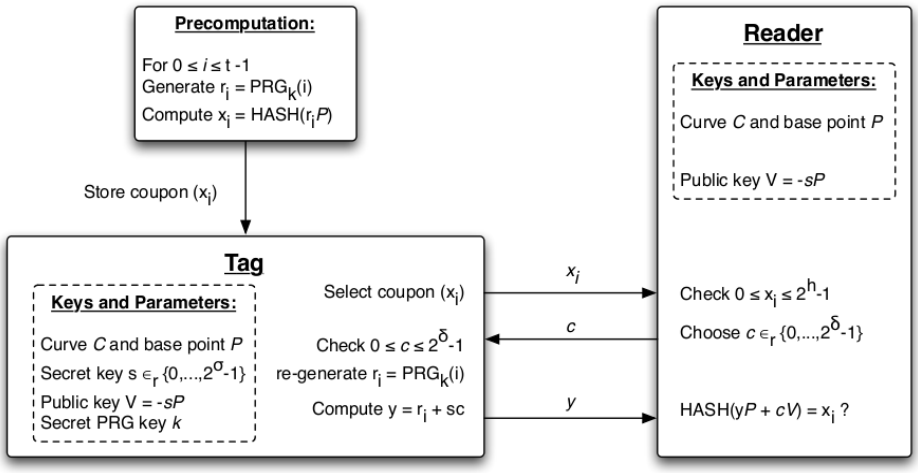


Fig. 1. An overview of the elliptic curve-based variant of cryptoGPS with most available optimizations implemented

3.2 Implementing cryptoGPS in Theory

Of particular practical interest are a series of optimizations designed to ease the computation and storage costs of cryptoGPS implementation.

- One important optimization is the use of coupons. In Girault describes a storage/computation trade-off for cryptoGPS that uses t coupons, each consisting of a pair (r_i , x_i) for $1 \le i \le t$ [20]. These coupons are stored on the tag before deployment. **Fig. 1** shows a general overview of the elliptic curve-based variant of cryptoGPS where both pre-computation and reader verification use a hash function HASH giving h-bit outputs. However when coupons are used neither the elliptic curve operation nor the hash function are needed on the tag.
- As a further improvement to the storage costs of coupons, we can generate the r_i using a keyed pseudo-random generator PRG_k as described in [16][21]. This is done at the time of tag manufacture, and then the necessary r_i can be re-computed on the tag at the time of verification.

– The on-tag computation $y = r_i + sc$ can be optimised by using what is termed a Low Hamming Weight (LHW) challenge [22]. This effectively turns the integer multiplication into a few simple integer additions.

3.3 Using Coupons

The combination of coupons and the LHW challenge lends cryptoGPS its advantageous performance. Yet coupons are not to everyone's taste.

The usual argument against coupons is the storage cost. Certainly the first generation UHF RFID tags being prototyped do not have enough memory to support coupons. However it is important to observe that current tags use old fabrication technology since this is the cheapest. As advances in digital architecture are incorporated, i.e. as tag manufacturers move towards the typical architectures we encounter in consumer devices today, there will be more room on the tag for enhanced digital functionality and memory. Interestingly, many use-cases would directly benefit from increased memory since this allows, for example, information to be added to the tag as it moves through the supply chain. Indeed, we see for certain niche applications that large amounts of memory is a top priority [23]. One can also consider more advanced application scenarios.

For instance, there appears to be no reason why coupons should necessarily be carried on the tag. Instead it is possible to envisage situations where coupons are delivered directly or cached on the interrogator/reading device perhaps along with the public key.

In such situations there are no additional on-tag memory requirements, though there could be some additional application-level issues to address in the management coupon use. However, even if we leave aside such advanced applications, the use of coupons ideally captures today's typical environment of use; we want aggressive and cheap performance on the tag and in most applications RFID tags will only be verified a moderate number of times, perhaps over several hops in the supply chain. After this the tag would be thrown away or deactivated as is currently recommended in a variety of policy statements on privacy.

3.4 Implementing cryptoGPS in Practice

In abstract terms, Section 2.2 gave an outline of how we would implement cryptoGPS. But these optimizations carry their own problems and it is a task of some difficulty to arrive at a good solution in practice.

Implementing the LHW challenge. In order to avoid the rather demanding ($\sigma \times \delta$)-bit multiplication that is required, it is possible to use a series of simple additions [22]. For this purpose it is required to turn the challenge c into a Low Hamming Weight (LHW) challenge such that at least $\sigma - 1$ zero bits lie between two subsequent 1 bits [22]. hen using binary representations of the multiplicands it is easy to see that multiplications can be performed using the basic Shift-And-Add multiplication algorithm [24]. When a bit of the input challenge c is 0, the multiplicand s is shifted to the left by one position. When the input challenge c is 1, the multiplicand s is shifted to the left and the result is added (with carry) to the multiplicand s. This way a complete multiplication can be reduced to simple shiftings and additions. Since in our case we use a low Hamming weight challenge that has all 1 bits at least $\sigma - 1$ zero bits apart, it is ensured that there is no overlap in subsequent additions of s. In other words s is never added more than once at the same time. In our implementation the secret is of size $\sigma = |s| = 160$ and the challenge c is of length $\delta = |c| = 848$ with a Hamming weight of 5. The specifications of cryptoGPS state that the parameters are typically set to $\rho = |r| = \sigma + \delta + 80$ and so for our chosen

values, achieving a probability of impersonation of 2^{-32} requires $\delta = 848$ bits [22] and this leads to $\rho = |r| = 160 + 848 + 80 = 1088$ bits. However 848 bits is quite a long challenge to transmit from the reader to the tag, and so work in has considered this issue [18][24]. In particular two encoding schemes have been proposed that require that we use only 40 bits to encode the complete 848-bit challenge c. We build on this work and in our implementation we will use a modified variant of the encoding scheme that was proposed for the 8-bit architecture in[22]. In particular it assumes that the challenge c is represented as five 8-bit chunks n_i so that $c = n_4||n_3||n_2||n_1||n_0$. Then, each n_i consists of the 5-bit number $c_{i,1}$ and the 3-bit number $c_{i,2}$, and so $n_i = c_{i,2}||c_{i,1}$ and these are used to encode the exact position of one of the five non-zero bits of the 848-bit low Hamming weight challenge. In particular, the positions $p_0,\ldots,p_4$ of the non-zero bits of the challenge c can be calculated using the following equations:

$$p_i = \begin{cases} 8 \times c_{0,1} + c_{0,2} & for\ i = 0 \\ 160 + 8 \times c_{i,1} + c_{i,2} & for\ 1 \le i \le 4 \end{cases} \quad (1)$$

Consider two example challenges $C_{comp,1}$ and $C_{comp,2}$. The all-zero compact transmitted challenge $C_{comp,1}$ gives the following $c_{i,1}$ and $c_{i,2}$, from which it is easy to compute $P(i)$ using Section 2.4.1

i	n_i	c_{i2}	c_{i1}	P (i)
0	0x00	000	00000	0
1	0x00	000	00000	160
2	0x00	000	0000	320
3	0x00	000	00000	480
4	0x00	000	00000	640

	n_4	n_3	n_2	n_1	n_0
$C_{comp,1}$ =	00	00	00	00	00

We can then recover the whole 848-bit challenge c as:[3]

	864	832	800	768	736	704	672
$C_{comp,1}$ =	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	640	608	576	544	512	480	448
	00000001	00000000	00000000	00000000	00000000	00000001	00000000
	416	384	352	320	288	256	224
	00000000	00000000	00000000	00000001	00000000	00000000	00000000
	192	160	128	96	64	32	0
	00000000	00000001	00000000	00000000	00000000	00000000	00000001

For the second example, set $C_{comp,2}$. as shown below, which leads to the associate values of $P(i)$:

	n_4	n_3	n_2	n_1	n_0
$C_{comp,2}$ =	44	E3	A2	C1	20

i	n_i	c_{i2}	c_{i1}	P (i)
0	0x20	001	00000	$8 \cdot 0 + 1 = 1$
1	0xC1	110	00001	$1 + 160 + 8 \cdot 1 + 6 = 175$
2	0xA2	101	00010	$175 + 160 + 8 \cdot 2 + 5 = 356$
3	0xE3	111	00011	$356 + 160 + 8 \cdot 3 + 7 = 547$
4	0x44	010	00100	$547 + 160 + 8 \cdot 4 + 2 = 741$

[3]Note that throughout this example we padded the challenge with 48 zeros to the left in order to gain a multiple of 64 ($848 + 48 = 896 = 14 \times 64$).

The associated challenge, in hexadecimal notation, is then given as:

$C_{comp2}=$

864	832	800	768	736	704	672
00000000	00000000	00000000	00000000	00000020	00000000	00000000
640	608	576	544	512	480	448
00000000	00000000	00000000	00000000	00000008	00000000	00000000
416	384	352	320	288	256	224
00000000	00000000	00000010	00000000	00000000	00000000	00000000
192	160	128	96	64	32	0
00000000	00008000	00000000	00000000	00000000	00000000	00000002

Using a PRG. Storing coupons cost memory and in both hardware and software implementations for embedded devices this can be a significant cost factor. Hence, the size of the coupons limits the number of available coupons for a given amount of memory or increases the cost. One approach uses a hash function to reduce the size of the x_i that need to be stored [21]. A second improvement is to observe that, above a certain threshold, it can be cheaper to implement a way of re-generating the r_i than to store them. The ISO standard 9798 suggests using a tag-specific keyed PRG for doing this [16]. While there are a variety of lightweight algorithms available we decided to use the lightweight block cipher PRESENT in an appropriate mode to regenerate the r_i [25][26]. The most efficient choice was to use the output feedback mode (OFB) for our cryptoGPS implementations [27]. Clearly care needs to be taken to manage the state of the cipher between calls to the tag to ensure that no repetitions in r_i are generated.

Summary. The following optimizations have been considered for this prototype:

1. Coupons are used to avoid hash and elliptic curve operations on the tag.
2. LHW challenges are used to reduce the on-tag ($\sigma \times \delta$)-bit multiplication to simple additions.
3. Compact encodings of the LHW challenge are used to reduce the transmission time.
4. A PRG is used to eliminate the need to store the r_i.

The implementations to be described in Sections 3.1 and 3.2 take the complete compact challenge c and a 64-bit initialization vector IV at the beginning of the computation. Though the secret s will be fixed in practical applications we also implemented a version with variable s. This gave us the flexibility for additional testing. The 64-bit IV was used to initialize a PRESENT-80 core in OFB mode. At the end of one run, *i.e.* after 17 complete iterations of PRESENT (since $17 \times 64 = 1088$), the ASIC outputs the internal state of the present core, allowing the state to be managed for the next run. In total, we implemented three different architectures.

1. One variant with a round-based PRESENT-80 core, an internal datapath of 8 bits and a fixed secret s. We refer to this variant as GPS-64/8-F and describe the implementation in Section 3.1.
2. A second variant uses a serialized PRESENT-80 core instead of a round-based one. For this variant it is advantageous to use an internal datapath of 4 bits. Again this was implemented with a fixed secret s. Details for the variant GPS-4/4-F are provided in Section 3.2.
3. A third variant returned to the round-based approach but allowed the secret s to be updated. This covers the few applications where one might envisage changing the key and it allows for some additional testing. This third variant, referred to as GPS-64/8-V, uses a

round-based PRESENT-80 core, an internal datapath of 8 bits and a variable secret *s*, see Sections 3.1.

4. Hardware Architectures of cryptoGPS

In this section we provide more details on the two round-based implementations, denoted cryptoGPS-64/8-F and cryptoGPS-64/8-V, before we describe the serialized implementation cryptoGPS-4/4-F. During our work the design of the prototype board posed several challenging limitations and these are discussed in Section 4.2.

As we will see, one issue is that the fabricated chips were mounted on a board and a microcontroller used to simulate the remaining parts of an RFID tag. These components needed to be synchronized and a handshake protocol was implemented. This is referred to in the sections that follow since we need to identify where this created a moderate performance overhead.

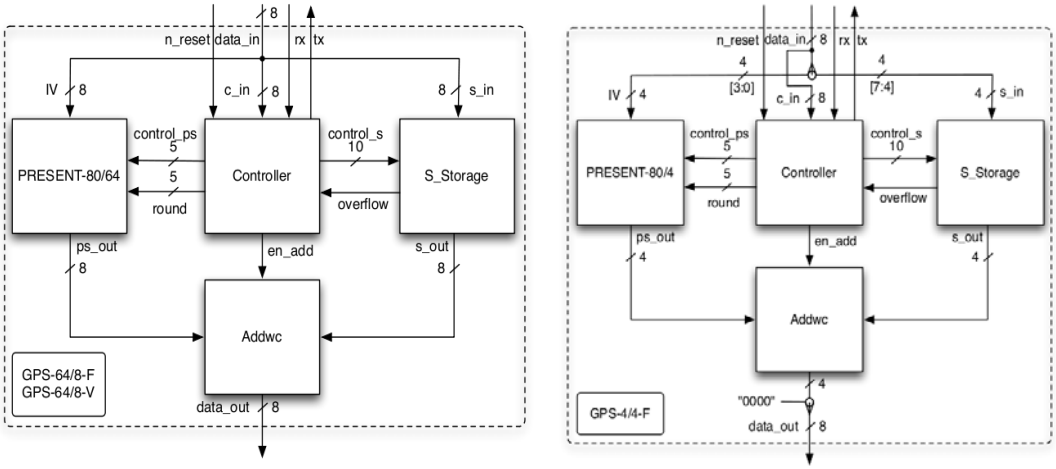


Fig. 2. Top-level architecture of the cryptoGPS-cores cryptoGPS-64/8-F and -64/8-V (left) and cryptoGPS-4/4-F (right)

4.1 Round-based Implementations

The architecture of cryptoGPS-64/8-F is depicted in Fig. 2. We use a round-based implementation of PRESENT, a `Controller` component, a full-adder component `Addwc` for the cryptoGPS computation, and `S_Storage` for holding the tag secret *s*. The variant cryptoGPS-64/8-V uses essentially the same architecture although the storage of *s* is handled differently. Here we describe these different components in detail and the relative space they occupy within the manufactured ASIC is nicely illustrated in **Fig. 12**.

The controller consists of four separate but interacting FSMs each one for the central control, I/O, `S_Storage`, and PRESENT (see **Fig. 3-6**). It requires 64 clock cycles to initialize the ASIC and to load the values IV, c_{in} , and s. In the round-based version it requires 32 cycles to create 64 pseudo-random bits using PRESENT and to add it with the appropriate chunk of the secret s. Due to the handshaking protocol, it then requires 64 cycles to output the result in 8-bit chunks. Since we have to compute 1088 bits, we have to repeat this procedure another 16 times. Finally, the internal state of PRESENT needs to be stored outside the ASIC so that it can be used as the new IV for the next iteration of cryptoGPS. In total, including I/O overhead, it takes $(17 \times (32 + 32)) + 32 = 1120$ clock cycles for one complete run of cryptoGPS. If we assume a more realistic scenario where the

cryptoGPS module is part of an integrated circuit, *i.e.* on an RFID tag, then there is no need for a handshaking protocol and only 724 cycles are required.

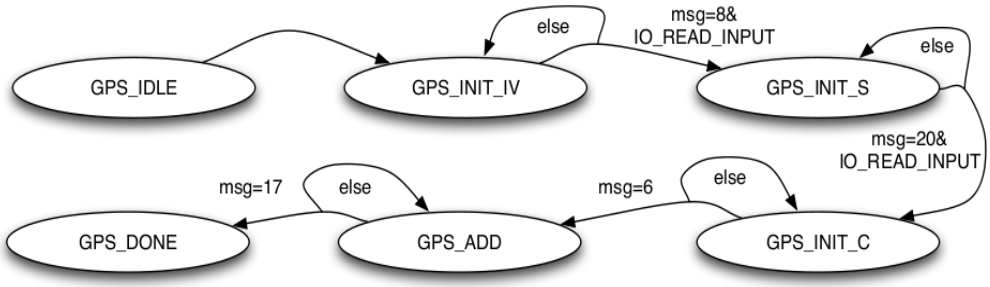


Fig. 3. Central FSM of all cryptoGPS variants

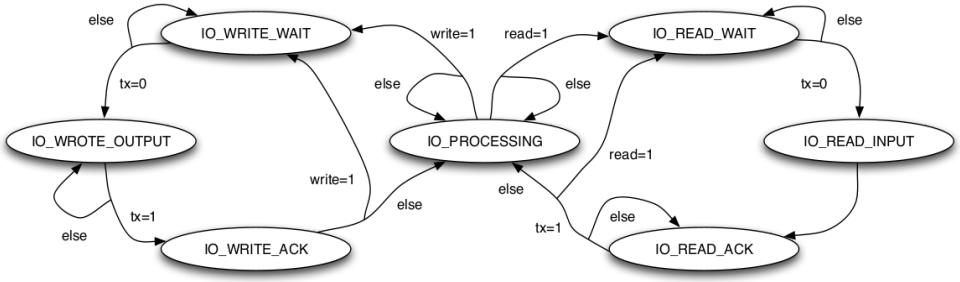


Fig. 4. I/O FSM of all cryptoGPS variants

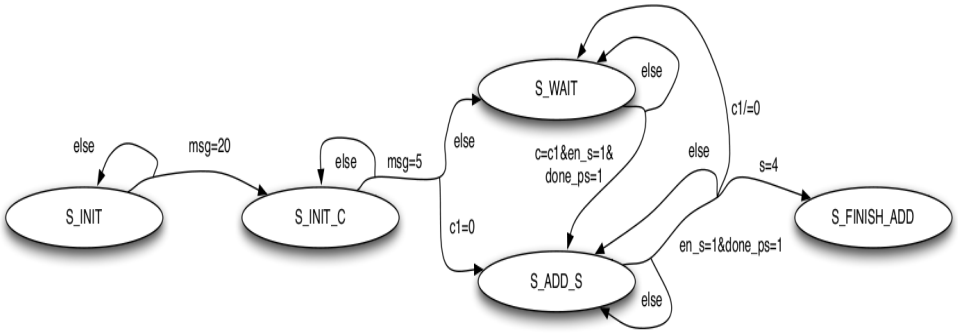


Fig. 5. FSM of the storage component of all cryptoGPS variants

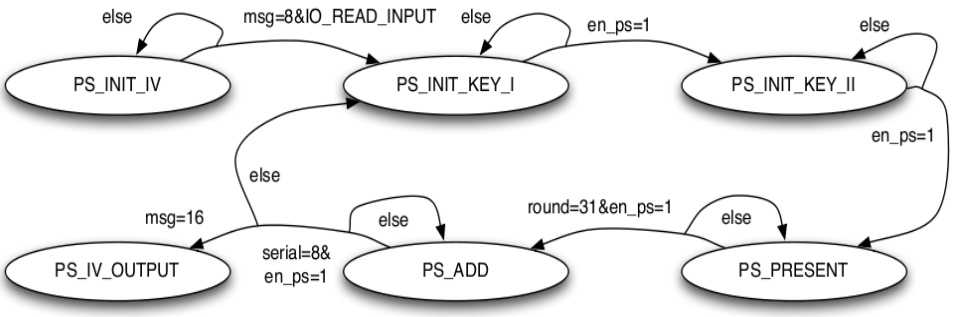


Fig. 6. FSM of the round-based present core of GPS-64/8-F and GPS-64/8-V

The Addwc component (see Fig. 7) consists of a flip-flop to store the carry bit and a ripple carry adder in order to keep the area requirements to a minimum. For the round-based variants GPS-64/8-F and GPS-64/8-V it has a datapath width of 8 bits, *i.e.* two 8-bit input values are added.

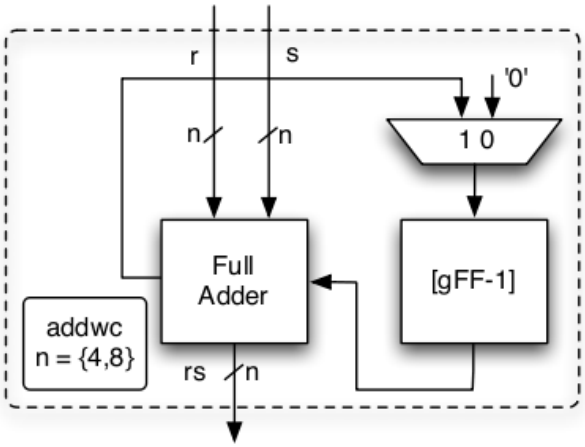


Fig. 7. Architecture of the adder component of all cryptoGPS variants

The architecture of the **S_Storage** component for a fixed secret *s* consists of an 8-bit AND gate, an 8-bit OR gate, a gated register with 8-bit input, and an 8-bit 20-to-1 MUX (see Fig. 8). These require 11, 11, 48 and 249 GE respectively, in total 319 GE. The appropriate 8-bit chunk of s is chosen by MUX and it is combined using AND with an 8-bit signal denoted *n_zero*. In fact *n_zero* is an eight-fold replication of a single bit and so *n_zero* can either be set to 00000000 or 11111111. This way the resulting value *a* is either set to 8-bits of *s* or 00000000 before being processed by the shifting component. To start, the input value *a* is appended to the string 00000000 to yield the intermediate state *b* and this is rotated by *c2* positions to the left. Since *c2* has three bits the shifting offset varies between 0 and 7. Finally it outputs two 8-bit values *c* and *d*, which consist of the eight most significant (*c*) and the eight least significant (*d*) bits of *b*, the internal state. The value *c* is stored in an 8-bit gated register and *d* is combined using OR with the output of the gated register.

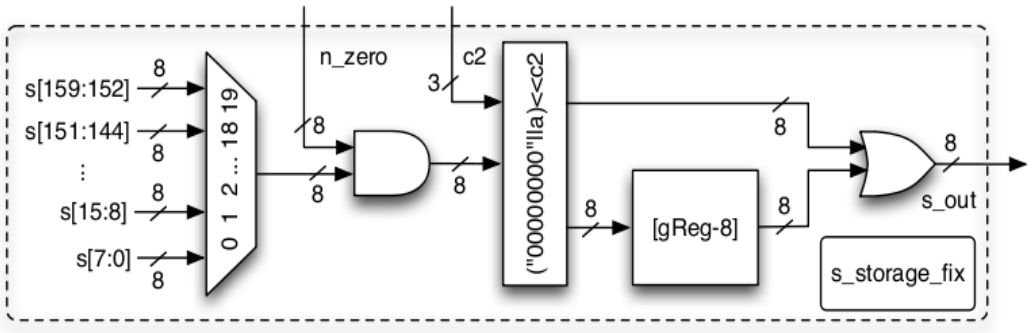


Fig. 8. Architecture of the storage component of GPS-64/8-F with a fixed secret s

Varying the secret *s*. To allow for additional testing we implemented one version of cryptoGPS with a key *s* that can be changed. This would not be the typical implementation in practice since the key for an RFID tag is normally set at the time of manufacture and cannot be changed. Adding this feature clearly imposes an additional cost: in our prototype the area overhead is 54%, mainly

due to the additional storage for the secret, but also due to a more complex finite state machine (see **Table 1**).

The `S_Storage` component that supports variable secrets *s* consists of an 8- bit 4-to-1 input MUX, an 8-bit 3-to-1 output MUX, an 8-bit AND, an 8-bit OR and 22 gated shifting registers that each store 8 bits (see **Fig. 9**). Twenty of these shifting registers are required to store the complete secret *s* while the remaining two are required to temporarily store the shifted values for the next addition cycle. This additional logic increases the area requirements for the `S_Storage` component more than ninefold to nearly 1500 GE making it too expensive for practical applications.

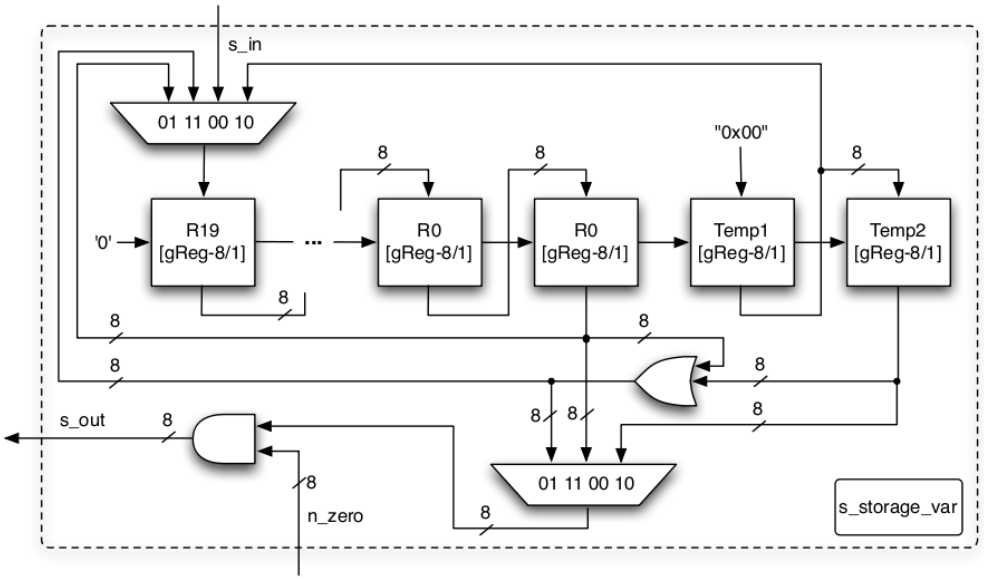


Fig. 9. Architecture of the storage component of GPS-64/8-v with a variable secret s

4.2 Serialized Implementations

To reduce the space demands we explored a serialized version of PRESENT-80 implementation (see **Fig. 2**). While the general form of the PRESENT and the `Addwc` components are relatively unchanged, the `Controller` and the `S_Storage` components are different and we describe them in more detail. Further, since the internal datapath of this variant is 4 bits, and since the outputs of the PRESENT, S_Storage, and Addwc components are 4-bits wide, the 4-bit output signal data_out is padded with 0000 to fit the 8-bit I/O interface.

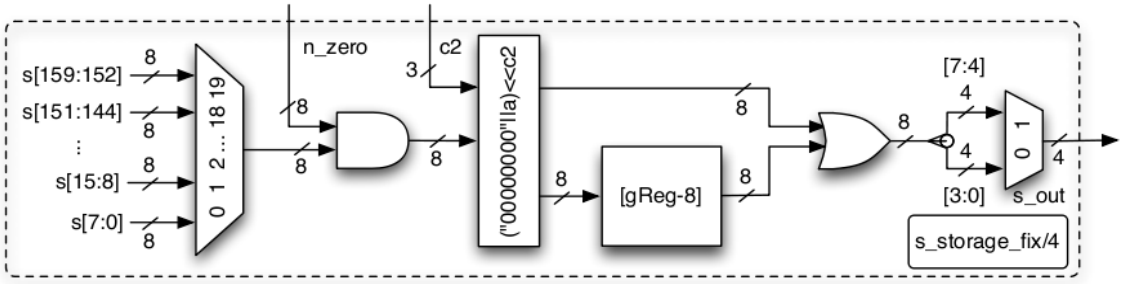


Fig. 10. Architecture of the storage component of GPS-4/4-F with a fixed secret s

Fig. 10 depicts the architecture of the `S_Storage` component for a fixed secret *s* and an internal 4-bit datapath. The main difference to the `S_Storage` component of the round-based variant cryptoGPS-64/8-F (see **Fig. 8**) is that it splits the 8-bit output value into two 4-bit chunks. Dependent on a counter value it outputs either the higher or the lower nibble.

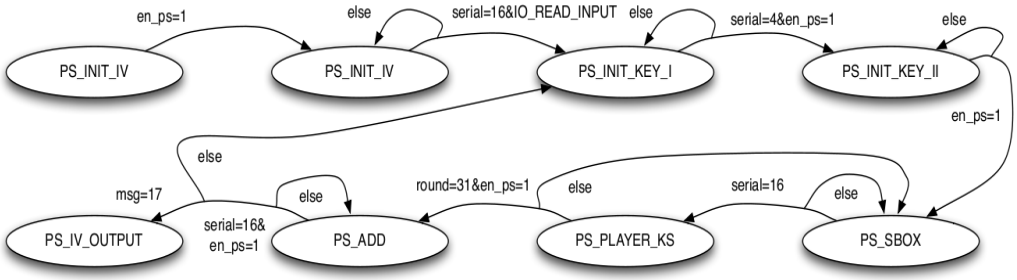


Fig. 11. FSM of the serialized present core of the GPS-4/4-F variant

Three out of four FSMs of the Controller module are similar to those used for the round-based variants. However the FSM of the serialized PRESENT-80 component is significantly more complex than a round-based implementation (see **Fig. 11**). It requires 64 clock cycles to initialize the ASIC and load the values *IV*, c_{in} and *s*. In the serialized version it requires 563 cycles to create 64 pseudo-random bits by the present component and to add it to the appropriate chunk of the secret *s*. Here we encounter an artificial delay since, due to the design of the board (see Section 4.2), it requires 64 cycles to output the result in 4-bit chunks. Since we have to compute 1088 bits, we have to repeat this procedure another 16 times. Finally the internal state of the present component has to be stored outside the ASIC as the new *IV* for the next iteration of cryptoGPS. So in total, including the I/O overhead, it takes $17 \cdot (527 + 64) + 64 = 10,111$ clock cycles for one complete run of cryptoGPS. Without the overhead this drops to 9, 319 cycles.

5. Implementation of cryptoGPS

ASIC fabrication is notoriously expensive and poses a formidable barrier. For our ASIC implementation of cryptoGPS we took advantage of the facilities provided by IHP Microelectronics[4] which offer so-called multi-design ASICs. Here different designs from different customers are bundled on the same wafer, and this permits significant cost savings for the production of the lithographic mask, which in turn allows us to fabricate designs for a very limited budget.

5.1 Communication Between ASIC and Board

One requirement of the shared design ASIC was that all variants have the same I/O pins. In order to have the possibility of using a small packaging we tried to use as few pins as possible. Beside the mandatory pins for power supply we decided to use the following 20 I/O pins: *clk*, *n_reset*, *rx* as the input channel and *tx* as the output channel of the ASIC for the I/O handshake protocol, *data_in* is used to load values in 8-bit chunks into the ASIC and *data_out* is used to output the result in 8-bit chunks.

Since the microcontroller (µC) is clocked independently from the ASIC, both components have

[4] Innovations for High Performance Microelectronics, Frankfurt/Oder, Germany.

to be synchronized when they are communicating. Therefore a handshake protocol with the following steps was implemented (see **Fig. 12**):

1. µC sets input data
2. wait until input data valid
3. µC sets *tx* to '0' indicating that input data are valid
4. wait until ASIC notices that input is valid (IO_READ_WAIT)
5. ASIC sets rx to '0' indicating that input is being read (IO_READ_INPUT)
6. ASIC reads input (IO_READ_INPUT)
7. ASIC sets *rx* to '1' indicating the successful read of input (IO_READ_ACK)
8. wait until µC notices that *rx* was set to '1'
9. µC sets *tx* to '1' thus finishing the input procedure
10. ASIC computes the response
11. ASIC sets *rx* to '0' indicating that output data are valid (IO_WRITE_WAIT)
12. wait until µC notices that output is valid (IO_WRITE_WAIT)
13. µC sets tx to '0' indicating that output is being read
14. µC reads output (IO_WROTE_OUTPUT)
15. µC sets tx to '1' indicating that the output was successfully read
16. wait until ASIC notices that tx was set to '1'
17. ASIC sets rx to '1' thus finishing the output procedure.

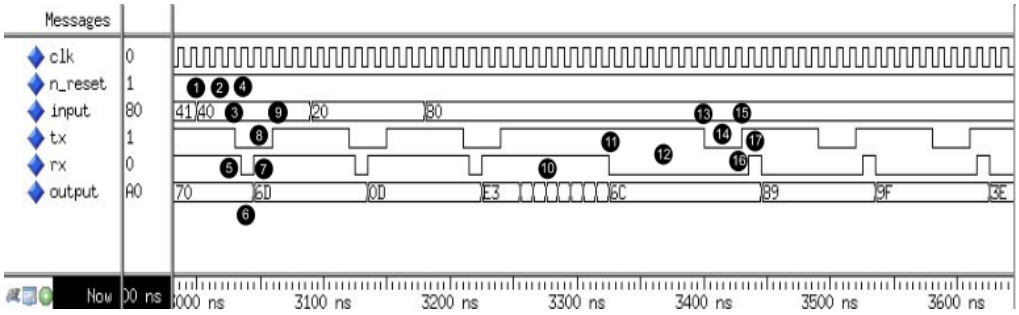


Fig. 12. Signal flow of the handshake protocol for communication between board and cryptoGPS ASIC

5.2 The cryptoGPS Proof-of-Concept Prototype Board

While the work in this paper demonstrates that a full implementation of cryptoGPS on an RFID tag is both feasible and, in terms of silicon, economically viable, our implementations still fall short of a fully functioning RFID tag. There is no radio/communication interface. This shortfall has no impact on the conclusions that can be drawn; indeed it serves to illustrate just how close to a prototype RFID tag we are. Nevertheless this communication with the outside world needs to be provided for testing and evaluation purposes. To achieve this the fabricated chips were mounted on a board and an ATMEL ATmega32a [28]. microcontroller, denoted µC, was used to simulate the remaining parts of an RFID tag. As such it provides the ASIC with the challenge c_{in} (and the secret *s* for the variant that allowed a variable secret) and receives the output of the ASIC. Since the microcontroller is clocked independently of the ASIC, these two components have to be synchronized when they are communicating. For this reason the handshake protocol described in Section 4.1 was implemented, and this lead to an increase of around 150 GE in the area requirements for the implementation. For the proof-of-concept prototype it was important to demonstrate the different functionalities of the cryptoGPS variants.

Therefore an external adapter provided a serial-to-USB interface for easy communication with a PC. The microcontroller converts the bit serial data stream from the serial interface to the 8-bit parallel I/O of the ASIC, and vice versa. **Fig. 14** depicts the layout of the prototype board and below in **Fig. 13** is a photograph shown.

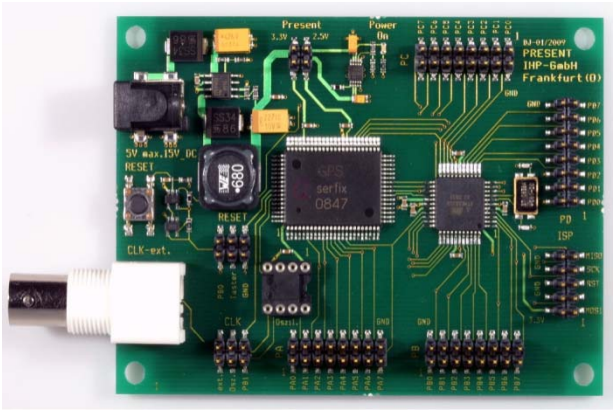


Fig. 13. Photograph of the prototype board

The ATMEL ATmega32a has a single power supply of 3.3 volts and the ASIC uses two different power supplies; one for the core (2.5 V) and one for the pads (3.3 V). This allows us to consider the power consumption of the cryptographic core without any influence of the pads. This is important since the cryptographic core would be integrated into a full custom design and directly connected to a main component. The ASIC design is in fact limited by the pads which means that the core itself occupies more space than is strictly required. The size of the die is 1, 372 × 1, 179 μm^2 yet the core itself requires only 445 × 645 μm^2 . After fabrication the die was put in a relatively large QFP-80 package, so as to be compatible with the test equipment at IHP.

5.3 Results and Discussion

For functional and post-synthesis simulation we used *Mentor Graphics Modelsim SE PLUS 6.3a* [29] while *Synopsys DesignCompiler version Z-2007.12-SP1* [30] was used to synthesize the designs to the IHP standard cell library *SESAME-LP2-IHP0.25UM*, which is compatible with the IHP 0.25 µm SGB25V process and has a typical voltage of 2.5 Volt [31].

Table 1 details the post-layout area requirements of every component of the three different architectures of cryptoGPS while **Table 2** provides area figures for comparison reasons for two different design steps: post-synthesis (*syn.*) and post-layout (*lay.*). As we can see flexibility comes at a high price; while the fixed secret variants of cryptoGPS can hardwire s and select the appropriate chunk with MUXes, a variant that allows s to change requires 160 additional flip-flops and a more complex finite state machine. Together this constitutes a significant overhead of 1, 550 GE (see **Table 1**). The area occupied by the different components of the cryptoGPS implementation are illustrated in Fig. 15. We can also see from **Table 2** that, for a single challenge, the round-based variants cryptoGPS-64/8-F and cryptoGPS-64/8-V require 724 clock cycles while the serialized variant cryptoGPS-4/4-F requires 9, 319 clock cycles.

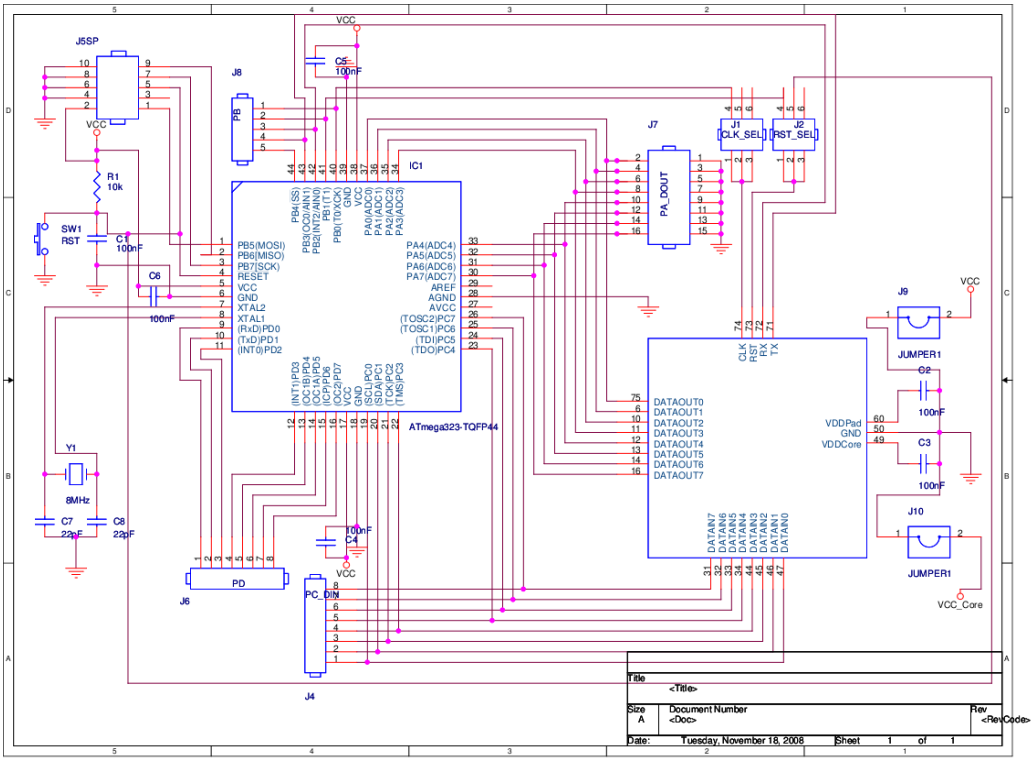


Fig. 14. Layout diagram of the cryptoGPS prototype board

Table 1. Breakdown of the post-layout implementation results of three different architectures of cryptoGPS

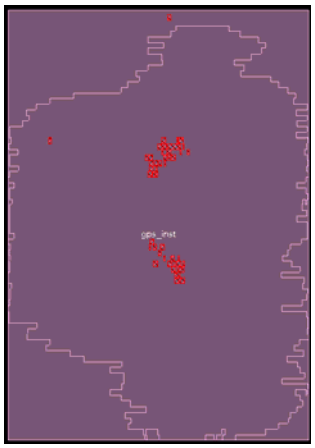
Component	PRESENT		Addwc		Controller		S_Storage		Sum
	[GE]	%	[GE]	%	[GE]	%	[GE]	%	[GE]
GPS-64/8-V	1751	39.5	67	1.5	1127	25.5	1483	33.5	4428
GPS-64/8-F	1751	60.9	60	2.1	905	31.5	159	5.5	2876
GPS-4/4-F	1200	50.0	35	1.5	905	37.7	263	11.9	2403

This is as one would expect, and at a frequency of 100 KHz this translates to 7.24 ms and 93.19 ms, both of which are well below the typical target of 200 ms. Since we omitted the timing overhead introduced by the handshaking protocol, these figures offer a realistic view of the timing demands of an embedded cryptoGPS core. Given that the processing time for serialized present is nearly 13 times longer than the round-based version it offers only a marginal benefit. Interestingly we observe that the post-synthesis area requirements are 3861, 2433, and 2143 GE depending on the variant.

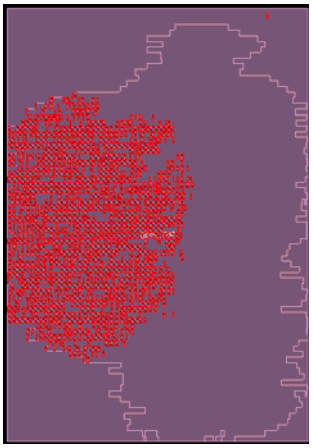
However filler cells, clock tree insertion and other layout overheads introduce a 12 to 18 % area increment and after manufacturing, these figures increase to 4428, 2876 and 2405 GE, respectively. Such an overhead is common and has been remarked on in other work [32]. Post-synthesis and post-layout current figures were simulated with *Synopsys DesignCompiler version Z-2007.12-SP1* and *Synopsys PrimePower* respectively. The results, ranging from 1.6 µA to 2.7 µA depending on the variant, indicate that cryptoGPS is well-suited for passive RFID tags.

Table 2. Post-synthesis and manufactured implementation results of three different architectures of cryptoGPS. We provide area figures for the two different design steps of post-synthesis (syn.) and post-layout (lay.). We also include figures for other low-cost asymmetric cryptographic implementations

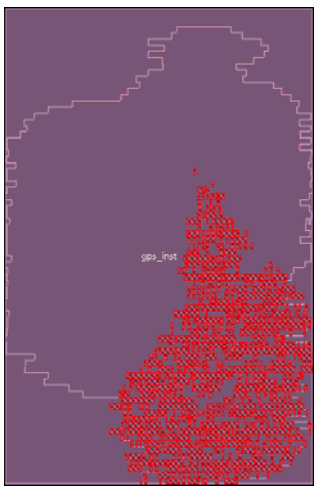
	Security level [bits]	Data path size	Cycles per block	Logic process	Design step	Area [GE]
GPS-64/8-V	80	8	724	0.25 IHP	syn.	3,861
					lay.	4,428
GPS-64/8-F	80	8	724	0.25 IHP	syn.	2,433
					lay.	2,876
GPS-4/4-F	80	4	9,319	0.25 IHP	syn.	2,143
					lay.	2,403
WIPR[28]	80	8	66,048	0.35 AMS	syn.	5,705
ECC-$(2^{67})^2$ [2]	67	1	418,250	0.25	syn.	12,944
ECC-112 [9]	56	1	195,264	0.35 AMI	syn.	10,113
NTRUencrypt [10]	57	1	29,225	0.13 TSMC	syn.	2,850



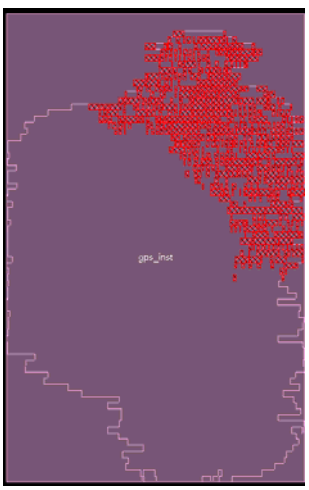
(a) *Addwc*(67 GE)



(b) PRESENT-80/64(1,751 GE)



(c) S_Storage(1,484 GE)



(d) Controller(1,127 GE)

Fig. 15. Area shares of single components within the GPS-64/8-V ASIC

6. Conclusions

In the field of lightweight cryptography hidden overheads are crucial. So while much attention is often focused on the headline implementation of the cryptographic core, additional mechanisms required to make the solution functional can be overlooked. In this paper we have made two contributions. The first is to highlight and quantify the unseen overheads for cryptoGPS. We have undertaken the design of a full version of the scheme yet the total costs still remain surprisingly modest; a fully-functioning version of cryptoGPS can be envisaged for 2000-3000 GE depending on the variant. The second contribution of the paper is to go through the full fabrication process and to produce a final functioning ASIC. This allows us to give increasingly accurate performance measurements, moving us one additional step closer to putting cryptography, indeed asymmetric cryptography, onto RFID tags.

Acknowledgements

We would like to thank Loïc Juniot, Marc Girault, Henri Gilbert, and Peter Langendörfer for their help and contributions.

References

[1] R. B. Handfield and E. L. Nichols, "Introduction to Supply Chain Management," *Prentice-Hall*, 1999.

[2] A. Bogdanov, G. Leander, L.R. Knudsen, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe, "An Ultra-Lightweight Block Cipher." *Lecture Notes in Computer Science*, vol.4727, pp. 450-466, 2007.

[3] M. McLoone and M. J. B. Robshaw, "Public Key Cryptography and RFID," *Lecture Notes in Computer Science*, vol.4377, pp.372-384, 2007.

[4] M. McLoone and M. J. B. Robshaw, "New Architectures for Low-Cost Public Key Cryptography on RFID Tags," in *Proc.of Int IEEE Conf. on Security and Privacy of Emerging Areas in Communication Networks* , pp.1827-1830, 2007.

[5] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, "An elliptic curve processor suitable for RFID-tags," Cryptology ePrint *Archive - Report 2006/227*, 2006. http:// eprint .iacr.org/.

[6] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight Cryptography Implementations," *IEEE Design & Test of Computers*, vol.24, no.6, pp.522-533, 2007.

[7] F. Fürbass and J. Wolkerstorfer, "ECC Processor with Low Die Size for RFID Applications," in *Proc. of The IEEE International Symposium on Circuits and Systems 2007*, pp.1835–1838, 2007.

[8] G. Gaubatz, J.-P. Kaps, and B. Sunar, "Public key cryptography in sensor networks — revisited. in C. Castellucia, H. Hartenstein," *Lecture Notes in Computer Science*, vol.3312 , pp.2-18, 2004.

[9] J. Hoffstein, J. Pipher, and J. Silverman," NTRU: A Ring-based Public Key Cryptosystem," *Lecture Notes in Computer Science*, vol.1423, pp.267-288, 1998.

[10] NTRU Corporation, NTRUencrypt. *http://www.ntru.com*.

[11] Y. Oren and M. Feldhofer, "WIPR – public-key identification on two grains of sand," 2008. http://iss.oy.ne.ro/WIPR.

[12] J. Wu and D. Stinson, "How to Improve Security and Reduce Hardware Demands of the WIPR RFID Protocol," in *Proc.of Int IEEE Conf.on RFID*, 2009.

[13] M. Girault. Self-certified public keys. In D. W. Davies, "Advances in Cryptology," *Lecture Notes in Computer Science*, vol.547, pp.490-497, 1991.

[14] M. Girault, G. Poupard, and J. Stern, "On the Fly Authentication and Signature Schemes Based on

Groups of Unknown Order," *Journal of Cryptology*, vol.19, pp.463-487, 2006.
[15] G. Poupard and J. Stern, "Security Analysis of a Practical on the fly Authentication and Signature Generation," *Lecture Notes in Computer Science*, vol.1403, pp.422-436, 1998.
[16] ISO/IEC 9798 Information technology–Security techniques–Entity authentication–Part 5: Mechanisms using Zero-Knowledge Techniques. http://www.iso.org/iso/iso_catalogue/catalogue_ tc /catalogue_detail.htm?csnumber=39720.
[17] IST-1999-12324, Final Report of European Project IST-1999-12324: New European Schemes for Signatures, Integrity, and Encryption (NESSIE), April 2004. http://www.cosic.esat.kuleuven.be/nessie.
[18] M. Girault, L. Juniot, and M. Robshaw, "The Feasibility of On-the-Tag Public Key Cryptography," in *Proc.of on RFID Security*, 2007.
[19] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography," *CRC Press*, 1996.
[20] M. Girault, "Low-Size Coupons for Low-Cost IC Cards," in *Proc. of the fourth working conference on smart card research and advanced applications on Smart card research and advanced applications*, pp.39-50, 2001.
[21] M. Girault and J. Stern, "On the Length of Cryptographic Hash-Values Used in Identification Schemes," *Lecture Notes in Computer Science*, vol.893, pp.202-215, 1994.
[22] M. Girault and D. Lefranc, "Public Key Authentication with One (Online) Single Addition," *Lecture Notes in Computer Science*, vol.3156, pp.967-984, 2004.
[23] FILRFID. Airbus chooses MainTag to Enable RFID in the A350, 2010. http://www.filrfid.org.
[24] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs," *Oxford University Press*, 1999.
[25] C. de Cannière and B. Preneel. "Trivium," *Lecture Notes in Computer Science,* vol.4986, pp.244-266, 2008.
[26] M. Hell, T. Johansson, and W. Meier, "The Grain Family of Stream Ciphers, *Lecture Notes in Computer Science,* vol.4986, pp.179–190, 2008.
[27] National Institute of Standards and Technology, "SP800-38A: Recommendation for Block Cipher Modes of Operation," 2001.
[28] Atmel Corporation. Datasheet of ATMega32a, 2003. http://atmel.com/dyn/resources/prod_documents/doc8155.pdf.
[29] Mentor Graphics Corporation. ModelSim SE User's Manual. http://www.model.com/resources/resources_manuals.asp.
[30] Synopsys. Design compiler user guide - version a-2007.12. https://solvnet.synopsys.com/dow_retrieve/A-2007.12/dcug/dcug.html, December 2007.
[31] Dolphin Integration, "Sesame-lp2 – description of the standard cells for the process IHP 0.25 µm – vic Specifications," 2005.
[32] M. Feldhofer, J. Wolkerstorfer and V. Rijmen, "AES Implementation on a Grain of Sand. Information Security," *IEE Proceedings on Information Security*, vol.152, no.1, pp.13-20, 2005.

Frank Vater received his master degree in information and media technology from the Brandenburg University of Technology at Cottbus (BTU) in 2007. After some preliminary work as student he joined the IHP in Frankfurt (Oder) in 2007. He worked in the area of efficient implementations of cryptographic algorithms in hardware. In this field he filed three patents and published five reviewed papers. He is currently member of the wireless sensor networks group where he works in the research of secure implementations and side-channel-resistance of integrated circuits.

Christof Paar has the Chair for Embedded Security at Ruhr-University Bochum, Germany, and is Affiliated Professor at the University of Massachusetts at Amherst. From 1994 to 2001 he was with Worcester Polytechnic Institute in Massachusetts. He co-founded, with Cetin Koc, the CHES (Cryptographic Hardware and Embedded Systems) workshop series. Christof's research interests cover fast software and hardware realizations of cryptography, physical security, penetration of real-world systems, trusted systems, and cryptanalytical hardware. He has over 150 peer-reviewed publications in applied cryptography, holds several patents, and is author of the textbook Understanding Cryptography.

Axel Poschmann is a post-doctoral research fellow with the Nanyang Technological University, Singapore. In 2009 he received his Ph.D. degree in Electrical Engineering from Ruhr University Bochum, Germany, where he also graduated as an IT security engineer (2005). In 2008 he received a Master degree in business studies from University Hagen, Germany. His primary research interest includes lightweight cryptography and side channel aspects for pervasive devices. He is also the co-editor of the ISO 29192-2 standard on Lightweight Cryptography - Part 2: Block Ciphers.

Matt Robshaw has a 1st class B.Sc. Hons from St. Andrews University and a Ph.D. from Royal Holloway, University of London. In 1993 he joined RSA Laboratories in California, and left in 1999 as the manager of the west coast office of RSA Laboratories and Principal Research Scientist. From 2000-2005 he was a member of the Information Security Group at Royal Holloway where he was Reader in Information Security then, in 2005, moved to France Telecom Research and Development, now called Orange Labs, where he is Senior Research Expert in cryptology. He has served on many program committees including Crypto, Eurocrypt, and FSE and has broad cryptographic research interests. His recent work has been particularly focused on the design, analysis, implementation, and deployment of symmetric cryptographic algorithms and lightweight cryptography. He is a co-designer of the AES finalist block cipher RC6 and the lightweight block cipher PRESENT. He is currently active in a variety of projects and standardization efforts and he is the author of numerous articles, papers, and patents.