PRINTcipher: A Block Cipher for IC-Printing

Lars Knudsen[1], Gregor Leander[1], Axel Poschmann[2,⋆] and Matthew J.B. Robshaw[3]

[1] Technical University Denmark, DK-2800 Kgs. Lyngby, Denmark
[2] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[3] Orange Labs, Issy les Moulineaux, France
{Lars.R.Knudsen,G.Leander}@mat.dtu.dk
aposchmann@ntu.edu.sg, matt.robshaw@orange-ftgroup.com

Abstract. In this paper we consider some cryptographic implications of *integrated circuit (IC) printing*. While still in its infancy, IC-printing allows the production and personalisation of circuits at very low cost. In this paper we present two block ciphers PRINTcipher-48 and PRINTcipher-96 that are designed to exploit the properties of IC-printing technology and we further extend recent advances in lightweight block cipher design.

Keywords. symmetric cryptography, block cipher, IC-printing, hardware implementation

1 Introduction

New technologies open new applications and often bring challenging new problems at the same time. Most recently, advances in device manufacture have opened the possibility for extremely low-cost RFID tags. However, at the same time, their exceptional physical and economic constraints mean that we must leave behind much of our conventional cryptography. This has spurred the development of the field of lightweight cryptography.

This paper considers another technological advance, that of *integrated circuit printing* or *IC-printing*. Using silicon inks, circuits can quite literally be printed onto a range of materials using high-definition printing processes. The technology remains in its infancy and its true potential is yet to be fully understood. But the claimed advantages include the ability to print on to thin and flexible materials and, since the conventional fabrication process is by-passed, to be much cheaper than silicon-based deployments [19]. Since the main driver for IC-printing is economic, the typically-cited areas of application overlap closely with the typical domains for lightweight cryptography. Indeed, one of the oft-stated applications of IC-printing is in the fabrication of cheap RFID tags [13]. Therefore there is much in common between some of the techniques proposed for conventional RFID tags and those that will be used on printed tags.

However IC-printing has some interesting properties and these allow us to take a fresh look at our cryptography and to see how it might be adapted to this new field. In this paper, therefore, we consider the task of adding some simple security functionality to a printed tag, and following what has now become a reasonably well-trodden path, we start out with the design of a block cipher.

Block ciphers make a natural starting point for several reasons. Not only can they be used in many different ways, but as a community we feel somewhat more at ease with their design and analysis. That said, for such extreme environments as IC-printing, we are working right at the edge of established practice and we are forced to consider and highlight some interesting problems. This is the purpose behind the block cipher PRINTcipher.

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

2 Design approach to PRINTcipher

Just as for other constrained environments, the size of implementation will be a dominant issue in IC-printing. Our work will therefore have close links with other block cipher work in the field of lightweight cryptanalysis. In fact our starting point for the work in this paper will be the block cipher PRESENT [1] which appears to offer a range of design/implementation trade-offs. However we will re-examine the structure of PRESENT in the particular context of IC-printing.

Conceptually we can imagine that within a block cipher we need an "encryption computation" and a "subkey computation". For the first, there are limits to the short-cuts we can make since we are constrained by the attentions of the cryptanalyst. This means, for the most part, that proposals for a given security level and a given set of block cipher parameters would occupy pretty much the same space. If we wanted to reduce the space occupied by an implementation then we would most likely reduce the block size, something that has been proposed independently elsewhere [2]. However, for the "subkey computation" things are a little different and exactly how a key should be used is not always clear. This highlights two separate issues.

The first issue is whether a key is likely to be changed in an application. In fact there is probably not too much debate about this issue and many commentators over the years [1,21] have made the point that for RFID applications it is very unlikely that one would want to change the key. Indeed some other RFID implementation work [20] has demonstrated that the overhead in supporting a change of key can be significant.

The second issue is the exact form of the key schedule. Some block ciphers, *e.g.* IDEA [14], have a very simple key schedule in which subkeys are created by sampling bits of the user-supplied key. This is, in effect, the approach used in the KTANTAN family of ciphers. The advantage of this approach is that no working memory is needed for the subkey computations. Other lightweight block ciphers have some key schedule computation, *e.g.* PRESENT, while another proposal CGEN [21] proposes to use no key schedule; the user-supplied key is used without any sampling or additional computation.

Returning to the situation at hand, conventional silicon manufacturing uses lithographic techniques to massively duplicate an implementation across a silicon wafer. This gives the economy of scale to offset the fabrication costs but at the same time requires that all implementations are identical. In this paper, we take advantage of the properties of IC-printing to propose another approach. Regular IC manufacture requires all versions of the cipher to be identical and so while a specific tag can be personalised with a unique key, this is a post-fabrication step. With a printer, however, there is essentially no cost in changing the circuit that is printed at each run. This means that part—or all—of the secret key can be embedded into the algorithm description. The algorithms that appear on different printed labels will be subtly different from one another.

The PRINTcipher family was designed with this approach in mind. PRINTcipher-48 is a 48-bit block cipher which uses a fixed 48-bit secret key and derives an additional 32 bits of security via the secret algorithm variability. Different trade-offs can be established either reducing the effective security, say to 64 bits, and/or independently increasing the block size to 96 bits. In fact this is a particularly useful block size since it matches the length of an *electronic product code* (EPC) [5]. However we will tend to concentrate our attentions in this paper on two proposals PRINTcipher-48 and PRINTcipher-96. Given the amount of work in the area of block ciphers, some points of similarity with other proposals in the literature are inevitable. For instance, 3-bit S-boxes have been used in 3-way [4] and the Scaleable Encryption Algorithm (SEA) [26] while key-dependent algorithm features have appeared in a variety of block ciphers including Blowfish [24], Twofish [25], and GOST [10].

3 PRINTcipher-48 and PRINTcipher-96

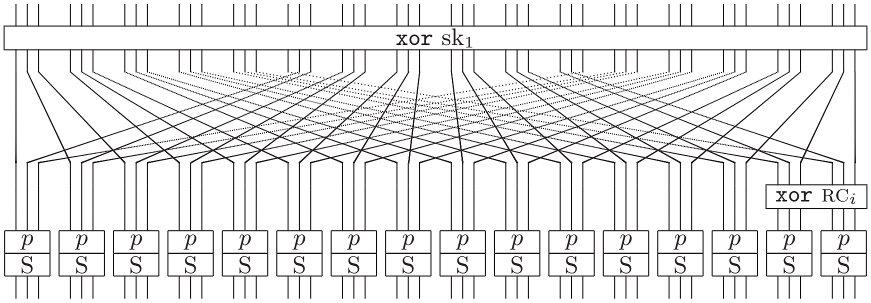


Fig. 1. One round of PRINTcipher-48 illustrating the bit-mapping between the 16 3-bit S-boxes from one round to the next. The first subkey is used in the first xor, the round counter is denoted RC$_i$, while key-dependent permutations are used at the input to each S-box.

PRINTcipher is a block cipher with b-bit blocks, $b \in \{48, 96\}$, and an effective key length of $\frac{5}{3} \times b$ bits. The essential structure of PRINTcipher is that of an SP-network with $r = b$ rounds. It follows that PRINTcipher-48 operates on 48-bit blocks, uses an 80-bit key and consists of 48 rounds while PRINTcipher-96 operates on 96-bit blocks, uses a 160-bit key and consists of 96 rounds. Each round of encryption consists of the following steps:

1. The cipher state is combined with a round key using bitwise `exclusive-or` (`xor`).
2. The cipher state is shuffled using a fixed linear diffusion layer.
3. The cipher state is combined with a round constant using bitwise `xor`.
4. The three-bit entry to each S-box is permuted in a key-dependent permutation layer.
5. The cipher state is mixed using a layer of $\frac{b}{3}$ non-linear S-box substitutions.

Key xor. The current state of the cipher is combined using bitwise `xor` with an b-bit subkey sk$_1$. This subkey is identical in all rounds.

Linear diffusion. The `pLayer` is a simple bit permutation that is specified in the following way. Bit i of the current state is moved to bit position $P(i)$ where

$$P(i) = \begin{cases} 3 \times i \bmod b - 1 & \text{for } 0 \leq i \leq b-2, \\ b - 1 & \text{for } i = b - 1. \end{cases}$$

Round counter RC$_i$. The round counter RC$_i$ for $1 \leq i \leq r$ is combined using `xor` to the least significant bits of the current state. The values of the round counter are generated using an n-bit shift register ($n = \lceil \log_2 r \rceil$) in the following way. Denote the state of the register as $x_{n-1}||\ldots||x_1||x_0$ and compute the update as follows:

$$\begin{aligned} t &= 1 + x_{n-1} + x_{n-2} & \\ x_i &= x_{i-1} & \text{for } n-1 \geq i \geq 1 \\ x_0 &= t & \end{aligned}$$

The shift register is initialised to all zeros, *i.e.* `000000` or `0000000`, and is then incremented at the start of every round. The round counter RC_i takes the current value of the register $x_{n-1}||\dots||x_1||x_0$.

Keyed permutation. Each set of three bits, namely the input bits to each of the S-boxes, are permuted among themselves. For each of the $\frac{b}{3}$ S-boxes the permutation can be the same or different and it is chosen in a key-dependent manner from a set of four. However for each S-box the same permutation—once chosen—is used in the same position in every round. In other words, $\frac{b}{3}$ permutations (of three bits) are picked from a set of four in a key-dependent manner. This gives $4^{b/3}$ possible mini-permutation layers which is equivalent to $\frac{2}{3} \times b$ key bits.

sBoxLayer. A single 3- to 3-bit S-box is applied $\frac{b}{3}$ times in parallel. For the sBoxLayer the current state is considered as $\frac{b}{3}$ 3-bit words, each word is processed using the same S-box, and the next state is the concatenation of the outputs. The action of the S-box is given by the following table.

x	0	1	2	3	4	5	6	7
$S[x]$	0	1	3	6	7	4	5	2

3.1 Deriving the permutations from the user key

The $\frac{5}{3} \times b$-bit user-supplied key k is considered as consisting of two subkey components $k = \mathrm{sk}_1||\mathrm{sk}_2$ where sk_1 is b bits long and sk_2 is $\frac{2}{3} \times b$ bits long. The first subkey is used, unchanged, within the `xor` layer of each and every round.

The second subkey sk_2 is used to generate the key-dependent permutations in the following way. The $\frac{2}{3} \times b$-bits are divided into $\frac{b}{3}$ sets of two bits and each two-bit quantity $a_1||a_0$ is used to pick one of four of the six available permutations of the three input bits. Specifically, the three input bits $c_2||c_1||c_0$ are permuted to give the following output bits according to the value of $a_1||a_0$. Of course one can combine the bitwise permutation with the fixed S-box to give, conceptually, four virtual S-boxes. These are given below and testvectors for both PRINTcipher variants can be found in the appendix.

$a_1\|\|a_0$	
00	$c_2\|\|c_1\|\|c_0$
01	$c_1\|\|c_2\|\|c_0$
10	$c_2\|\|c_0\|\|c_1$
11	$c_0\|\|c_1\|\|c_2$

x	0	1	2	3	4	5	6	7
$V_0[x]$	0	1	3	6	7	4	5	2
$V_1[x]$	0	1	7	4	3	6	5	2
$V_2[x]$	0	3	1	6	7	5	4	2
$V_3[x]$	0	7	3	5	1	4	6	2

3.2 Security goals

Our security goals behind PRINTcipher are the usual security claims for a block cipher with the operational parameters of PRINTcipher. Note that in the case of PRINTcipher-48 even though we have a regular sized 80-bit key, we only have a 48-bit block cipher and this greatly limits the opportunities for an attacker.

We follow much of the established literature on lightweight cryptography and do not consider side-channel attacks in this paper. While this is certainly a factor for consideration, typical applications are very low-cost and the potential gains for an attacker are minor. Even in a relatively well-developed field such as RFID tags for the supply chain it is not clear what level of protection is really appropriate for most deployments of lightweight cryptography. For IC-printing this is even more unclear, and there are some concerns that are particular to IC-printing for which appropriate precautions will likely be needed, such as the use of opaque masks to shield the circuit from simple inspection. Note that, shielding protection is not exclusively an issue for PRINTcipher where the key is part of the circuit, but also for more standard ciphers where the key is stored in memory, as it is in principle possible to inspect memory in similar ways (see for example [23]) .

Where we differ from some other work in the field, however, is that for PRINTcipher we are not particularly concerned by related-key attacks. This is not because we believe that PRINTcipher is in some way particularly vulnerable to them (see Section 4.3 for details). Instead it is because we believe that related-key attacks are so alien to the intended use of PRINTcipher that there is no point in considering them. Recall that a (printed) device will be initialised with a key in a random way. To mount a related-key attack one has to somehow find a pair of deployed devices that, by chance, satisfy a stated condition. We consider this to be an entirely unrealistic threat.

3.3 Some features of the design

During the design of PRINTcipher there were some interesting choices to make. Certainly, to improve the implementation efficiency we required that each round was identical, even as far as having an identical subkey in each round. However having the same round key in every round meant that we were restricted to 48-bit keys. So to increase the effective key length we used some additional permutation steps that could be key-dependent. Permutations cost nothing in hardware and, for our application of IC-printing, they incur effectively no additional cost during the printing of the cipher. It can be shown that there are no equivalent keys in the sense that there are no two pairs of subkey components $(\mathrm{sk}_1, \mathrm{sk}_2)$ that will yield the same round function. Note that since every round is identical—to the point of having the same round key—we needed to introduce a round-dependent value to prevent slide attacks and this was done using a shift register-based counter as outlined above.

The S-box The 3-bit S-box that we chose is optimal with respect to linear and differential properties. However we cannot avoid the existence of single-bit to single-bit differences or masks and so our specific choice of S-box minimizes there occurrence. That is, for a given single-bit input difference (resp. mask) exactly one single-bit output difference (resp. mask) occurs with non-zero probability (resp. non-zero bias). We generated all 3-bit S-boxes with this property and it turns out that there are exactly 384 such S-boxes in total.

Clearly, permuting the input bits and (`xor`) adding constants before or after the S-box preserves the desired properties. Up to these changes, there is only one possible choice of S-box, *i.e.* all 384 S-boxes fulfilling the desired criteria can be constructed from any one of them by permuting the input bits and adding constants before and after the S-box (indeed $384 = 6 \cdot 2^3 \cdot 2^3$).

Thus in the design of PRINTcipher there is, in effect, only one suitable choice of S-box. Choosing any other of the 384 possible S-boxes would result in the same cipher for a different key, up to an additional `xor` with a constant to the plaintext and the ciphertext. More formally, given two S-boxes S_0, S_1 out of the 384 possible choices and any key $(\mathrm{sk}_1, \mathrm{sk}_2)$ there exist a key $(\mathrm{sk}'_1, \mathrm{sk}'_2)$ and constants c_1, c_2 such that

$$\text{PRINT}\textsc{cipher}_{S_0,\mathrm{sk}_1,\mathrm{sk}_2}(p) = \text{PRINT}\textsc{cipher}_{S_1,\mathrm{sk}'_1,\mathrm{sk}'_2}(p \oplus c_1) \oplus c_2$$

for any plaintext p.

Those observations imply another interesting property of the S-box of PRINTcipher. Namely, instead of permuting the input bits of the S-box one could permute the output bits of the S-box and `xor` suitable constants before and after the S-box. More precisely, denoting the PRINTcipher S-box by S, for any bit permutation P, there exist constants c and d such that

$$S(P(x)) = P(S(x \oplus c)) \oplus d \quad \forall x.$$

Note that, while this might give some freedom in implementing the cipher we did not see any security implications of this.

The bit permutation We choose the permutation so as to give the potential for full dependency after a minimal number of rounds, *i.e.* after $4 = \lceil \log_3 48 \rceil$ rounds. Note that in general, given an SP-network with block size b and s bit Sboxes, where s divides b, it can be shown that the bit permutation

$$P(i) = \begin{cases} s \times i \bmod b-1 & \text{for } 0 \leq i \leq b-2, \\ b-1 & \text{for } i = b-1. \end{cases}$$

provides optimal diffusion in the sense that full dependency is reached after $\lceil \log_s b \rceil$ rounds. The bit permutation – or rather its inverse – used for the block cipher PRESENT is a special case of this general result.

4 Security Analysis

In this section we analyze the security of our proposal with respect to the main cryptanalytic methods known. Though we focus on PRINTcipher-48, the security analysis can be easily extended to PRINTcipher-96.

4.1 Differential and linear characteristics

Let p be the probability of a linear characteristic, then define the correlation of the linear characteristic as $q = (2p-1)^2$ [18]. As mentioned above, the S-box in PRINTcipher was chosen with good differential and linear properties. These properties are inherited by the other three virtual S-boxes, and so if we combine the key-dependent permutation with the S-box operation any differential characteristic over any S-box has a probability of at most 1/4, and any linear characteristic over any S-box has a correlation of at most 1/4.

Any characteristic over s rounds of PRINTcipher would have at least one active S-box per round. Consequently, an s-round differential characteristic will have a probability of at most 2^{-2s} and any s-round linear characteristic will have a correlation of at most 2^{-2s}. Thus, conventional differential and linear characteristics are unlikely to play a role in the cryptanalysis of PRINTcipher with the specified 48 respectively 96 rounds.

We furthermore experimentally checked for differential effects, i.e., the probability of differentials compared to the probability of characteristics. Consider the following one-round iterative characteristic (octal representation):

$$(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ ) \rightarrow (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ ).$$

Only the S-box in the least significant bits is active. This characteristic has probability 1/4 when the active S-box is V_0 or V_1. The iterative characteristic above has an expected probability of 2^{-24} for 12 rounds.

We implemented experiments with 20 keys, each randomly chosen but such that the S-box in the least significant bits is either V_0 or V_1. For each key we generated 2^{28} pairs of texts of the above difference. The number of pairs of the expected difference after 12 rounds of encryption was 16.6 on the average, where 16 is expected for the characteristic. In similar tests over 14 rounds using 2^{30} pairs, the average number of pairs obtained was 4.5 on average, where 4 was expected. Here the expected probability of the iterative characteristic is 2^{-28}. These tests suggest that there is no significant differential effect for the characteristic. Computing the exact differential effect for a characteristic over many more rounds of PRINTcipher is a very complex task. However since the probability of the iterative characteristic is very low, *e.g.* 2^{-80} for 40 rounds, we expect that good probability differentials are unlikely to exist for PRINTcipher.

4.2 High order differentials and algebraic attacks

The algebraic degree of the S-box is 2 and due to the large number of 48 rounds we expect the total degree of the cipher to be close to the maximum. This assumption is supported by the following experiments. It is well-known that for a function of algebraic degree d, a d^{th}-order differential will be a constant, and the value of a $(d+1)^{\text{st}}$-order differential will be zero. Consequently, if a d^{th}-order differential over s rounds for one key is not zero, then the algebraic degree of this encryption function is at least $d-1$. For seven rounds of PRINTcipher and for ten randomly chosen keys we computed the values of two different 25^{th}-order differentials. In all cases the values were nonzero. The experiments suggest that the algebraic degree of PRINTcipher reaches its maximum after much less than the specified 48 rounds. Due to this observations, we expect PRINTcipher to be secure against higher order differential attacks.

Regarding the so-called algebraic attacks, first observe that there exist quadractic equations over all 3-bit S-boxes, also those of PRINTcipher. Therefore, the secret key of one particular encryption can be described as the solution to a number of quadratic equations. However such a system of equation for PRINTcipher will be huge because of the large number of rounds, and with the techniques known today, there is not much hope that such systems can be solved in time faster than simply trying all values of the key. Moreover, the key dependent permutations potentially make the resulting systems of equation even more complex and harder to solve.

4.3 Related-key attacks

As stated above, we consider related-key attacks to be an entirely unrealistic threat. However, in the spirit of academic completeness, we consider the issue here.

The four S-boxes in PRINTcipher are closely related. As an example, S-box 0 and S-box 1 produce the same output for each of four inputs and similarly for S-boxes 2 and 3 and

Table 1. Promising trails of different sizes.

S-boxes in the trail	# of bits in the trail	Ratio
$\{0, 1, 5\}$	5	5/9
$\{0, 1, 5, 15\}$	7	7/12
$\{4, 10, 12, 14, 15\}$	9	9/15
$\{0, 1, 2, 5, 6, 7\}$	12	12/18
$\{3, 8, 9, 10, 11, 13, 15\}$	14	14/21
$\{0, 1, 4, 5, 10, 12, 14, 15\}$	18	18/24

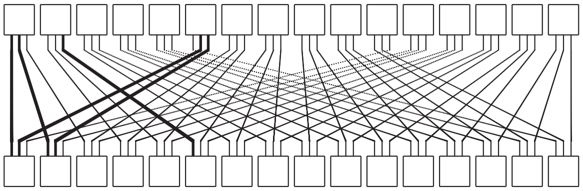
for S-boxes 4 and 5. Consider two keys different only in the selection of one S-box, say, the leftmost one. Assume further that one key selects S-box V_0 and the other key selects S-box V_1. It follows that for one round of encryption, the encryption function induced by the two keys will be equal for half the inputs. Consequently, the encryption functions over s rounds can be expected to produce identical ciphertexts for one in 2^s texts.

There are other related keys. Consider two keys different only in `xor` halves and only in the input to one S-box. For such two keys it may be possible to specify a keyed differential characteristic where the differences in the texts are canceled by the differences in the `xor` key in every second round. If in all other rounds it is assumed that there is only one active S-box and that the difference in the inputs equal the difference in the outputs, then one gets an s-round differential characteristic of probability 2^{-s} (for even s).

The observations in this section can potentially be used to devise related-key attacks which could recover a key for PRINTcipher using a little less than 2^b texts. It is clear, however, that if the keys of PRINTcipher are chosen uniformly at random it is very unlikely that one would find keys related as described above.

4.4 Statistical Saturation Attacks

Statistical saturation attacks have been presented in [3] and successfully applied to round-reduced versions of present. The key idea for statistical saturation attacks is to make use of low diffusion trails in the linear layer of present. As PRINTcipher uses a very similar linear layer, it seems natural that the attack applies to reduced round versions of PRINTcipher as well. We identified low diffusion trails for any number of S-boxes involved, see Table 1 for examples of the most promising ones using up to eight S-boxes in a trail. One example of such a low diffusion trail is given below.



As explained in [3] increasing the number of S-boxes in the trail makes estimating the complexity of the attack very complicated. Thus, in our experiments we focused only on the case of three active S-boxes in the trail. All four possible trails gave very similar results. We estimated the bias for 50 randomly chosen keys for up to 10 rounds and for 20 randomly chosen keys for up to 15 rounds. Table 2 and Figure 2 show the squared euclidian distance between the distribution in the trail and the uniform distribution. The data complexity for attacking

Table 2. Estimated squared distance ($\log_2$) for low diffusion trails with ratio 5/9.

S-boxes in the trail	$\{0,1,5\}$	$\{2,6,7\}$	$\{10,14,15\}$	$\{8,9,13\}$
Round 1	0	0	0	0
Round 2	-3.72	-3.72	-3.74	-3.73
Round 3	-6.67	-6.74	-6.89	-6.65
Round 4	-9.14	-8.98	-9.19	-9.05
Round 6	-13.08	-13.17	-13.17	-13.10
Round 8	-16.92	-17.25	-16.96	-17.10
Round 10	-21.02	-20.88	-20.87	-21.03
Round 12	-25.38	-25.33	-24.82	-24.93
Round 14	-28.72	-28.94	-29.19	-28.94
Round 16	-32.83	-33.05	-33.27	-33.00

$r+3$, *resp.* $r+4$ rounds, depending on how many key bits are guessed, is approximately the reciprocal of the squared euclidian distance for r rounds. While our experiments are certainly limited, the results strongly suggest that no more than 30 rounds of PRINTcipher can be broken using this attack.

Fig. 2. The estimated squared bias with the number of rounds on the x-axis and the $\log_2$ of the squared bias given on the y-axis.

5 Implementation Results

To demonstrate the efficiency of our proposal we have implemented both PRINTcipher variants in VHDL and used *Synopsys DesignVision 2007.12* [27] to synthesize them using the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18μm 1P6M* logic process and has a typical voltage of 1.8 Volt [29].

Before presenting the results we stress the unique deployment environment offered by IC-printing. While our implementation efforts allow us to obtain a reasonable estimate of the space required, in terms of gate equivalents (GE), for an IC-printing implementation of PRINTcipher, any attempts to compare the likely power consumption with other implementations of lightweight cryptography are not just difficult (as is usually the case), but they are essentially meaningless. For this reason our performance results and comparisons will concentrate on the space occupied by an implementation.

Figure 3 depicts two architectures that were implemented: a serialized one with a datapath of 3-bits and a round-based one with a datapath of 48 or 96 bits. Components that contain mainly sequential logic are presented in rectangles while purely combinational components are presented in ovals.

The first serialized implementation of PRINTcipher-48 used a *finite state machine* (FSM) that required 120 GE out of which 95 GE were occupied by two arithmetic counters: 59 GE were occupied by the 6-bit round counter and additional 36 GE were required for a 4-bit counter to keep track of the 3-bit chunks of the serialized state. Similar to KATAN [2] we replaced the arithmetic round counter by a shift register-based counter, which saved 28 GE (or 47%) while having better distribution properties. The second counter was also replaced

(a) serial 3-bit datapath.

(b) round-based 48/96-bit datapath.

Fig. 3. Two architectures for PRINTcipher.

by a register-based counter which decreased the gate count by another 12 GE (35%). Finally we completely omitted the FSM and replaced it with some combinatorial gates to generate the control signals required (*e.g.* for the MUX). In total, by omitting the FSM and optimizing the control logic, we were able to save 54 GE (45%).

As part of our quest for a minimal S-box, we used the Boolean minimization tool `BOOM II` [7,8] to obtain the boolean functions of all 48 S-box variants that can be generated from a 3-bit S-box, by permuting the output bits and XORing a hardwired constant. Our synthesis results show that the results vary between 10.67 and 12 GE, and we chose a minimal S-box.

In order to be able to present a detailed break down for each component of PRINTcipher (see the accompanying table), we advised the compiler to *compile simple*, *i.e.* to keep the hierarchy of the components. The smallest area footprint is achieved, however, if the compiler uses the *compile ultra* command, which allows the merging and optimization of different components simultaneously. Since the key `xor` is hardwired, the area requirements for the `KeyXOR` component are dependent on the Hamming weight of the key. The implementation figures of Figure 4 used a key with Hamming weight 24, thus the area footprint of a serialized implementation of PRINTcipher-48 is bounded by 386 GE and 418 GE for keys with Hamming weight 0 and 48, respectively (694 GE and 758 GE for PRINTcipher-96). The results show

PRINTcipher-n		$n = 48$		$n = 96$	
		serial	round	serial	round
cycles		768	48	3072	96
throughput @100 KHz (Kbps.)		6.250	100	3.125	100
compile ultra	sum	402	503	726	967
compile simple	sum	411	528	733	1011
sequential:	State	288	288	576	576
	genRC$_i$	31	31	36	36
	NLFSR	23	0	30	0
combinational:	MUX	7	0	7	0
	KeyXOR	16	16	32	32
	pLayer	0	0	0	0
	RC XOR	16	16	19	19
	sBoxLayer	11	171	11	342
	control	12	4	15	4
	other	7	2	7	2

Fig. 4. Implementation figures for PRINTcipher.

that both PRINTcipher variants scale nicely; by spending more area for additional S-boxes, the throughput can be scaled (nearly) linearly. At this point it is noteworthy to highlight the significant overhead (43 GE or 10.5%) that is required for additional control logic in a serialized PRINTcipher-48 implementation. This shows that it is hard to gain further area reductions. Furthermore, note that a 6-bit `xor` with the round constant RC_i requires the same area as the 48-bit hardwired `xor` with a key with a typical Hamming weight of 24.

While observing our earlier caveats about the use of power estimates in the context of IC-printing, we did make some measurements for the likely power consumption of more conventional silicon-based implementations. We used *Synopsys PowerCompiler* version *A-2007.12-SP1* [28] to estimate the performance of our implementations. Measurements using the smallest wire-load model (10K GE) at a supply voltage of 1.8 Volt and a frequency of 100 KHz suggested a power consumption below 2.6 μW; a good indication that all PRINTcipher variants are well-suited to demanding applications including printed passive RFID tags. It is a well-known fact that at low frequencies, as typical for low-cost applications, the power consumption is dominated by its static part, which is proportional to the amount of transistors involved. Furthermore, the power consumption strongly depends on the used technology and greatly varies with the simulation method. Thus we refer to the area figures (in GE) as the most important measure and to have a fair comparison we do not include the power values in Table 3.

Table 3. Hardware implementation results of some symmetric encryption algorithms.

		key	block	cycles/	Throughput	Tech.	Area
Algorithm		size	size	block	(@100 KHz)	[μm]	[GE]
			Stream Ciphers				
Trivium	[9]	80	1	1	100	0.13	2,599
Grain	[9]	80	1	1	100	0.13	1,294
			Block Ciphers				
PRESENT	[22]	80	64	547	11.7	0.18	1,075
SEA	[17]	96	96	93	103	0.13	3,758
mCrypton	[16]	96	64	13	492.3	0.13	2,681
HIGHT	[12]	128	64	34	188	0.25	3,048
AES	[6]	128	128	1,032	12.4	0.35	3,400
AES	[11]	128	128	160	80	0.13	3,100
DESXL	[15]	184	64	144	44.4	0.18	2,168
KATAN32	[2]	80	32	255	12.5	0.13	802
KATAN48	[2]	80	48	255	18.8	0.13	927
KATAN64	[2]	80	64	255	25.1	0.13	1054
KTANTAN32	[2]	80	32	255	12.5	0.13	462
KTANTAN48	[2]	80	48	255	18.8	0.13	588
KTANTAN64	[2]	80	64	255	25.1	0.13	688
PRINTcipher-48		80	48	768	6.25	0.18	402
PRINTcipher-48		80	48	48	100	0.18	503
PRINTcipher-96		160	96	3072	3.13	0.18	726
PRINTcipher-96		160	96	96	100	0.18	967

Table 3 compares a selection of lightweight block and stream cipher implementations that have been optimized for a minimal area footprint. It can be seen that the serialized implementation of PRINTcipher requires the least amount of area for its block and key sizes (402 GE). Moreover, spending additional 100 GE (or 25%) the throughput can be increased 16 fold to 100 Kpbs at a frequency of 100 KHz, while still having a remarkably small area footprint. The resulting throughput per area ratio of 198.8 Kpbs per GE is even suited for high-speed applications though our main focus is on a low area footprint.

It is noteworthy to stress that we designed PRINTcipher to be secure even in the absence of a key schedule. This allows for significant area savings, because no flipflops to store the key state are required. Of course one could hardwire all the roundkeys for any cipher with a key schedule and, theoretically, this would allow for similar savings. In practice, however, this is not the case. Since all low-area implementations are serialized or round-based designs, one needs complex additional logic to select the right roundkey or even the right part of the roundkey. For a serialized AES for example, one would need a 128-bit wide 11-to-1 MUX to select the correct roundkey plus an 8-bit wide 16-to-1 MUX to select the right chunk of the roundkey. Our experiments reveal that a 128-bit wide 8-to-1 MUX already consumes 1276 GE, which makes it more efficient to store the key state in flipflops (768 GE) than to hardwire the roundkeys.

Though they have not been the focus of our design, for those interested in software implementations we estimate the performance of PRINTcipher on a 64-bit platform to be around 5-10 times slower than an optimized AES implementation: merging the permutation and using 6-bit S-boxes could give an implementation with 9-12 cycles per round. With 48 rounds this amounts to 72-95 cycles per byte while AES runs in 10-20 cycles per byte.

6 Conclusions

In this paper we have considered the technology of IC-printing and we have seen how it might influence the cryptography that we use. In particular we have proposed the lightweight block cipher PRINTcipher that explicitly takes advantage of this new manufacturing approach. Naturally it must be emphasized that PRINTcipher-48 is intended to be an object of research rather than being suitable for deployment. It is also intended to be a spur to others who might be interested in considering this new technology. Certainly we believe that the properties of IC-printing could be an interesting line of work and we feel that it helps to highlight several intriguing problems in cryptographic design, most notably how best to use a cipher key.

References

1. A. Bogdanov, L.R. Knudsen, G. Leander , C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. present - An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Proceedings of CHES '07*, volume 4727 of LNCS, pages 450–466. Springer, 2007.
2. C. de Cannière, O. Dunkelman, and M. Knezević. KATAN and KTANTAN–A Family of Small and Efficient Hardware-Oriented Block Ciphers. In C. Clavier and K. Gaj, editors, *Proceedings of CHES '09*, volume 5747 of LNCS, pages 272–288. Springer, 2009.
3. B. Collard and F.-X. Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. In *Topics in Cryptology — CT-RSA 2009*, LNCS, volume 5473,pages 195–211.
4. J. Daemen, R. Govaerts, and J. Vandewalle. A new approach to block cipher design. In R. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9–11, 1993,* volume 809 of *Lecture Notes in Computer Science,* Springer-Verlag, pages 18–32, 1994.

5. EPCglobal. Organisation information available at www.epcglobal.com.
6. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *Information Security, IEE Proceedings*, 152(1):13–20, 2005.
7. P. Fišer and J. Hlavička. BOOM - A Heuristic Boolean Minimizer. *Computers and Informatics*, 22(1):19–51, 2003.
8. P. Fišer and J. Hlavička. Two-Level Boolean Minimizer BOOM-II. In *Proceedings of 6th Int. Workshop on Boolean Problems – IWSBP'04*, pages 221–228, 2004.
9. T. Good and M. Benaissa. Hardware Results for Selected Stream Cipher Candidates. State of the Art of Stream Ciphers 2007 (SASC 2007), Workshop Record, February 2007. Available via `www.ecrypt.eu.org/stream`.
10. GOST. Gosudarstvennyi standard 28147-89, *cryptographic protection for data processing systems*. Government Committee of the USSR for Standards, 1989 (in Russian).
11. P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *DSD*, pages 577–583, 2006.
12. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems — CHES 2006*, number 4249 in Lecture Notes in Computer Science, pages 46–59. Springer-Verlag, 2006.
13. Kovio. Company information available via www.kovio.com.
14. X. Lai, J. Massey, and S. Murphy. Markov ciphers and differential cryptanalysis. In D. Davies, editor, *Advances in Cryptology - EUROCRYPT'91,* volume 547 of *Lecture Notes in Computer Science,* Springer-Verlag, pages 17–38, 1992.
15. G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lightweight DES Variants. In *Proceedings of Fast Software Encryption 2007 – FSE 2007*, volume 4593 of *LNCS*, pages 196–210. Springer-Verlag, 2007.
16. C. Lim and T. Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In J. Song, T. Kwon, and M. Yung, editors, *Workshop on Information Security Applications — WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, 2005.
17. F. Mace, F.-X. Standaert, and J.-J. Quisquater. ASIC Implementations of the Block Cipher SEA for Constrained Applications. In *RFID Security — RFIDsec 2007, Workshop Record*, pages 103 – 114, Malaga, Spain, 2007.
18. M. Matsui. New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptanalysis. In D. Gollmann, editor, *Proceedings of FSE 1996*, volume 1039 of *LNCS*, pages 205–218. Springer-Verlag, 1996.
19. PolyIC. Information available via www.polyIC.com.
20. A. Poschmann, M.J.B. Robshaw, F. Vater, and C. Paar. Lightweight Cryptography and RFID: Tackling the Hidden Overheads. In D. Lee and S. Hong, editors, *Proceedings of ICISC '09*, Springer, to appear.
21. M.J.B Robshaw. Searching for Compact Algorithms: CGEN. In P. Nguyen, editor, *Proceedings of Vietcrypt 2006*, volume 4341 of *LNCS*, pages 37–49. Springer-Verlag, 2006.
22. C. Rolfes, A. Poschmann, G. Leander, and C. Paar. Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In G. Grimaud and F.-X. Standaert, editors, *Smart Card Research and Advanced Application — CARDIS 2008*, volume 5189 of *Lecture Notes in Computer Science*, pages 89–103. Springer-Verlag, 2008.
23. D. Samyde, S. Skorobogatov, R. Anderson, and J. Quisquater. On a New Way to Read Data from Memory. *SISW '02: Proceedings of the First International IEEE Security in Storage Workshop*, pages 65–69. IEEE Computer Society, 2002.
24. B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In R. Anderson, editor, *Proceedings of FSE 1993*, volume 809 of *LNCS*, pages 191–204. Springer-Verlag, 1994.
25. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, Hall, and N. Ferguson. Twofish: A 128-bit block cipher. Submitted as candidate for AES. Available via www.nist.gov/aes.
26. F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In J. Domingo-Ferrer, J. Posegga, and D. Schreckling, editors, *Smart Card Research and Applications, Proceedings of CARDIS 2006*, LNCS, volume 3928, pages 222–236, Springer-Verlag.
27. Synopsys. Design Compiler User Guide - Version A-2007.12. Available via http://tinyurl.com/pon88o, December 2007.
28. Synopsys. Power Compiler User Guide - Version A-2007.12. Available via http://tinyurl.com/lfqhy5, March 2007.
29. Virtual Silicon Inc. 0.18 μm VIP Standard Cell Library Tape Out Ready, Part Number: UMCL18G212T3, Process: UMC Logic 0.18 μm Generic II Technology: 0.18μm, July 2004.

Appendix A: Testvectors

Table 4. Testvectors for PRINTcipher-96 in hexadecimal notation.

	Testvector 1	*Testvector 2*
plaintext	5A97E895A9837A50CDC2D1E1	A83BB396B49DAA6286CD7834
key	953DDBBFA9BF648FF6940846	D83F1CEF1084E8131AA14510
permkey	70F22AF090356768	62C67A890D558DD0
ciphertext	45496A1283EF56AFBDDC8881	EE5A079934D98684DE165AC0
	Testvector 3	*Testvector 4*
plaintext	5CED2A5816F3C3AC351B0B4B	61D727437449984269OCA3CC
key	EC5ECFEF020442CF3EF50B8A	2F3F647A9EE6B4B5BAF0B173
permkey	68EA816CEBA0EFE5	A07CF36902B48D24
ciphertext	7F49205AF958DD440ED35D9E	3EB4830D385EA369C1C82129

Table 5. Sequence of RC_i for PRINTcipher-96 in hexadecimal notation.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
RC_i	01	03	07	0F	1F	3F	7E	7D	7B	77	6F	5F	3E	7C	79	73	67	4F	1E	3D	7A	75	6B	57
i	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
RC_i	2E	5C	38	70	61	43	06	0D	1B	37	6E	5D	3A	74	69	53	26	4C	18	31	62	45	0A	15
i	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
RC_i	2B	56	2C	58	30	60	41	02	05	0B	17	2F	5E	3C	78	71	63	47	0E	1D	3B	76	6D	5B
i	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
RC_i	36	6C	59	32	64	49	12	25	4A	14	29	52	24	48	10	21	42	04	09	13	27	4E	1C	39

Table 6. Cipher example for PRINTcipher-48 in hexadecimal notation.

plaintext	key	permkey	ciphertext
4C847555C35B	C28895BA327B	69D2CDB6	EB4AF95E7D37

Rd.	RC	keyAddition	pLayer	RC XOR	S-box perm.	S-box
1	01	8E0CE0EFF120	ED9921498D92	ED9921498D93	ED92A24B0AE3	5B12FB6E89BE
2	03	999A6ED4BBC5	A9DE9DEC68E1	A9DE9DEC68E2	65BF1EEC6991	C765F5585F59
3	07	05ED60E26D22	0D8345DB891C	0D8345DB891B	0D88C67B886B	1B0F85D50E66
4	0F	D987106F3C1D	90FA448917F7	90FA448917F8	517A442917F8	7DA8472D9C90
5	1F	BF20D297AEEB	EAEB7C66A29B	EAEB7C66A284	E76AFCC6C484	46D997A676C7
6	3E	8451021C44BC	84015030F6C4	84015030F6FA	4800D09277B9	6C0198FFAD51
7	3D	AE890D459F2A	A21FEB888A8E	A21FEB888AB3	621FEB2A0CB3	C394A62F8AEE
8	3B	011C3395B895	21B2166079C3	21B2166079F8	21D815C079F8	234E1CA02E90
9	37	E1C6891A1CEB	D21646B4A739	D21646B4A70E	D21C45B6464D	BF9A4493FC4C
10	2F	7D12D129CE37	7E6B4E01D46B	7E6B4E01D444	BE6ACD033304	8BD98C02E3C7
11	1E	495119B8D1BC	343C0771F644	343C0771F65A	385607D37729	344C02BEADE1
12	3C	F6C497049F9A	F273FBB01388	F273FBB013B4	F279FBB014F4	5FAE969C17AF
13	39	9D26032625D4	80C9572711F0	80C9572711C9	4122D78591CA	61B39AE7108B
14	33	A33B0F5D22F0	8284BE2EFCA6	8284BE2EFC95	43053D8EFAA6	6287D4FA28FD
15	27	A00F41401A86	8A120A28096C	8A120A28094B	46180988094B	679E09EC0F0E
16	0E	A5169C563D75	C2B7C50CF1F1	C2B7C50CF1FF	C35DC60E71FF	A2CA851BA092
17	1D	604210A192E9	323008758223	32300875823E	325008D7043D	3FC008B28614
18	3A	FD489D08B46F	F2FDC6148E49	F2FDC6148E73	F377C5160F33	5EAC841385EE
19	35	9C2411A9B795	A0F94B631543	A0F94B631576	6172CBC19375	C1A38EA3132C
20	2B	032B1B192157	00A437063C6F	00A437063C44	01443704DB04	01F62A04C9C7
21	16	C37EBFBEFBBC	F5B6BF73FFF0	F5B6BF73FFE6	F9DD3FD3FFD5	574BD2BD249C
22	2C	95C3470716E7	8851DEB480FF	8851DEB480D3	4431DDB601A3	6460B493817E
23	18	A6E82129B305	A3912B930C43	A3912B930C5B	6390AB318B2B	C110E63F09E6
24	30	039873853B9D	09B23FE05AC3	09B23FE05AF3	05D83FE03DB3	074E12406B6E
25	21	C5C687FA5915	D41397D93571	D41397D93550	D81997795360	B41F5AD5C338
26	02	7697CF6FF143	7ED5B38D45BF	7ED5B38D45BD	BF35B32D22FE	8AE76E39B395
27	05	486FFB8381EE	792C13768B7E	792C13768B7B	B4C613D68D7B	90FC1EB20B16
28	0B	52748B08396D	50D63316C741	50D63316C74A	913C3316A749	FDEA2E123D09
29	17	3F62BBA80F72	436F7F579428	436F7F57943F	82EEFF57923F	E25592711212
30	2E	20DD07CB2069	028092DCCF17	028092DCCF39	0301117E2E7A	0281D9CBB453
31	1C	C0094C718628	B804C809AA06	B804C809AA1A	7405480B4C29	D007080EFA21
32	38	128F9DB4C85A	6466A2C53BAC	6466A2C53B94	A86D21655CE4	8C5BF9C5CBBF
33	31	4ED36C7FF9C4	3D9FA1BD64F6	3D9FA1BD64C7	3D9FA2BD6387	2B157B89D342
34	23	E99DEE33E139	FF8C9581FB17	FF8C9581FB34	FF8716237C74	490DDD22AA6F
35	06	8B8548989814	A81E24C03544	A81E24C03542	641E24605341	C4143FC04301
36	0D	069CAA7A717A	4595318DFF18	4595318DFF15	8994B22F7E66	EF16EB3AA47D
37	1B	2D9E7E809606	2B3DDCC04968	2B3DDCC04973	26D7DC602973	264CB7C03F2E
38	36	E4C4227A0D55	9303519D3551	9303519D3567	5288D23D5357	7E0F9B29C31A
39	2D	BC870E93F161	A6DD91C4A137	A6DD91C4A11A	6B3792664069	CEED5BC7F061
40	1A	0C65CE7DC21A	6C0D981B378E	6C0D981B3794	AC079819D6E4	980D70174DBF
41	34	5A85E5AD7FC4	5DDAEBE505C6	5DDAEBE505F2	9DBB6BE503F1	EB69264582A9
42	29	29E1B3FFB0D2	63B816FF349E	63B816FF34B7	A3D215FDD2B7	81421C4B42EA
43	12	43CA89F17091	549426F93823	549426F93831	991425F95832	F5963C55CE2B
44	24	371EA9EFFC50	67D7664D5DB2	67D7664D5D96	ABBCE54D3AE5	8D6BBC79E9BC
45	08	4FE329C3DBC7	351F2FFE007F	351F2FFE0077	389EAFFC8137	3494E24881EA
46	11	F61C77F2B391	BBF1BB697911	BBF1BB697900	77F1BBC97840	D12156ADAE40
47	22	13A9C3179C3B	68527682BA9F	68527682BABD	A4387522DCBE	846E6C224AD5
48	04	46E6F99878AE	5DB722F2A768	5DB722F2A76C	9DDCA1F2C75C	EB4AF95E7D37