New Designs in Lightweight Symmetric Encryption

C. Paar, A. Poschmann*, **and M.J.B. Robshaw**

Abstract In this article, we consider new trends in the design of ultra-lightweight symmetric encryption algorithms. New lightweight designs for both block and stream ciphers as well as the underlying hardware design rationale are discussed. It is shown that secure block ciphers can be built with about 1,500 gate equivalences and, interestingly, it seems that modern lightweight block ciphers can have similar hardware requirements to lightweight stream ciphers.

1 Introduction

The bulk of cryptographic work is done using symmetric primitives. While we might appeal to asymmetric cryptography to establish a shared key [39], cryptographic data processing is almost always done using a symmetric encryption, authentication, or hashing algorithm.

In this article, we will consider some new trends in the design of lightweight symmetric encryption algorithms. There are two distinct types of symmetric encryption; *stream ciphers* and *block ciphers* and the essential difference between them can be described as follows:

- A block cipher transforms blocks of *plaintext* into *ciphertext* under the action of a *key*. This is typically a relatively complicated transformation but, apart from the reused key, the encryption of one block is independent of another.
- A stream cipher generates a *keystream* by sampling a constantly evolving *cipher state*. The state is typically initialized under the action of a *key* and an *initialization vector*. The sampling operation and the operation used to update the state are

A. Poschmann
Horst Görtz Institute for IT Security, Embedded Security Group (COSY), Ruhr-Universität Bochum, Germany
e-mail: poschmann@crypto.rub.de

usually computationally lightweight. The *plaintext stream* is then encrypted by combining it directly – typically using bitwise exclusive-or – with the keystream to give the *ciphertext stream*

Stream ciphers themselves can be divided into *synchronous* and *self-synchronizing* stream ciphers. For the first, the cipher state is updated independently of the generated ciphertext. For the second, the self-synchronizing stream cipher state update includes the generated ciphertext. The two types of stream cipher have very different error-propagation and synchronization properties [39] but, for the purposes of this article, we need no more detail. The vast majority of contemporary proposals are synchronous, but (secure) self-synchronizing stream ciphers appear to be rather difficult to design [17].

It is well established that a block cipher can be used to give a stream cipher. The NIST modes of operation provide three ways of doing this and are known as the *cipher feedback*, *output feedback*, and *counter* modes [42]. Interestingly, the cryptographic folklore suggests that stream ciphers of a dedicated design should be more efficient than block ciphers and, therefore, more efficient than stream ciphers based on block ciphers. Such an advantage might manifest itself in increased encryption speeds, or more compact and power-efficient implementations. However, as stream cipher cryptanalysis and block cipher design have advanced, this advantage has been somewhat eroded. This is something we will return to in our conclusions.

Since there is an algorithmic distinction between block and stream ciphers, we will address the two primitives separately. First, we will consider the state of the art in low-cost block cipher design. Then we will consider low-cost stream cipher designs. Since block ciphers can be used to give stream ciphers, the most efficient block cipher proposals will, in some sense, set the bar against which the most efficient stream cipher proposals should be compared. To set the stage, we will consider some of the basic building blocks for cryptographic primitives and compare their efficiency in hardware.

Before starting out we mention some particular considerations that apply to deployments in constrained environments such as low-cost tags for RFID applications. Very often, such applications require only a moderate level of security and reflect the very limited financial gains available to an attacker. The security demanded for typical industry applications such as electronic commerce or internet communication may not be suitable for some constrained devices and, since increased security levels translate directly into more physical space in silicon and increased deployment costs, security in excess of what is required is both costly and unwelcome. An appropriate security level will only be revealed by risk assessment and cost–benefit analysis, but 80-bit security may well be adequate in many such applications.

Generally speaking, applications for constrained devices are unlikely to require the encryption of large amounts of data. Implementations can therefore be optimized for the space they occupy or the power they consume without too much practical impact. So while security will often be the main consideration for some cryptographic primitive, the physical space required for an implementation will typically be the primary physical consideration, closely followed by peak and average power consumption, and timing requirements being a less-important third metric. Interestingly, the lack of large amounts of encrypted data helps reduce exposure to a range of attacks that manipulate time–memory–data trade-offs [1, 9, 11]. Further savings can be made in some applications when the cryptographic key is fixed at the time of device manufacture. In such cases there would be no need to rekey a device which rules out both a range of key manipulation attacks [7] as well as the consumption of additional resources.

2 Hardware Efficiency of Cryptographic Building Blocks

Hardware efficiency can be measured in many different ways; the length of the critical path (or maximum frequency), latency, clock cycles, power/energy consumption, throughput, and area requirements all have a significant influence on the viability of an implementation.

One particular problem in passive RFID applications is that the tags face strict power constraints. A rule-of-thumb is that the power consumption should be less than 15µW and the power consumption is given by the voltage times current consumption. For chips built in CMOS[1] technology the power consumption is the sum of two parts: static and dynamic power consumption. The static power consumption is roughly proportional to the area, i.e., the larger the area the higher the power consumption. The dynamic part is proportional to the switching activity, which is proportional to the operating frequency.

To lower power consumption, RFID applications are typically clocked at a low frequency, e.g., 100 kHz or 500 kHz. In this frequency range the static power consumption is dominant. RFID applications usually have harsh cost constraints and the silicon area of the chip is directly proportional to the cost. Therefore, a good way to minimize both the cost and the power consumption is to minimize the area requirements. It has become common to use the term *hardware efficient* as a synonym for small area requirements.

Area requirements are usually measured in μm^2, but this value depends on the fabrication technology and the standard cell library. In order to compare the area requirements independently it is common to state the area as *gate equivalents* (GE). One GE is equivalent to the area which is required by the two-input NAND gate with the lowest driving strength of the appropriate technology. The area in GE is derived by dividing the area in μm^2 by the area of a two-input NAND gate.

2.1 Architecture Strategies

Generally speaking, there are three major hardware architecture options for block ciphers: parallel (loop unrolled), round-wise, and serial. A *parallel*, or loop unrolled,

[1] *Complementary Metal Oxide Semiconductor*, the most widely used technology.

block cipher implementation performs several round operations of the encryption/decryption process within one clock cycle. Usually parallel implementations are *pipelined*, i.e., registers are inserted in the critical path so as to increase the maximum clock frequency. While parallel implementations have high throughput rates, this is rarely the focus for RFID applications. Rather, the high area and power demands mean that parallel implementations of block ciphers and stream ciphers are rarely suited for passive RFID applications.

In a *round-wise* implementation, one round function of a block or a stream cipher is processed within one clock cycle. The decreased throughput comes at the benefit of decreased area and power consumption. From a low power and low area perspective, round-wise implementations are best suited for stream ciphers and make a reasonable option for block ciphers. For example PRESENT [13] has been implemented in a round-wise manner.

To lower power consumption and area requirements, implementations can be *serialized*; here only a fraction of one round is processed in a clock cycle. Up to a certain point this strategy can significantly decrease the area and the power consumption and the impressive results by Feldhofer et al. on the AES [41] are achieved by serialization [23]. However, it might not always be a suitable implementation strategy since the savings can sometimes be canceled by the overheads in additional control logic. Nevertheless, from a low-power and low-area perspective, serial implementations appear to be best suited for RFID-like implementations in the case of block ciphers. The natural way of implementing stream ciphers is in a bit serial fashion.

2.2 Internal State Storage

Ciphers have an internal state which we might refer to as *cipher state* and *key state*. When a block cipher is used, the cipher state is initialized by the plaintext (or ciphertext) and modified under the action of the *key* (and therefore the key state). When a stream cipher is used, the cipher state is initialized by the *initialization value* and the *key*. Stream ciphers then use the initialized cipher state to output the keystream. Block ciphers have a fixed number of rounds and the final internal state serves as the ciphertext. Note that independent of the implementation strategy, see above, the internal cipher state has to be saved at each round.

In software environments kilobytes of RAM and ROM are available. In low-cost tag applications this is not the case. Although most RFID tags have a memory module, for cryptographic algorithms there is only the barest minimum of storage capacity available. Furthermore, read and write access to the memory module (usually EEPROM) is very power consuming. As a consequence it is preferable to store all intermediate values and variables in registers rather than in external memory.

Registers typically consist of *flipflops*. Compared to other standard cells, flipflops have a rather high area and power demand. For example, when using the *Virtual Silicon* (VST) standard cell library based on the *UMC L180* 0.18μ *1P6M Logic process*

(UMCL18G212T3), flipflops require between 6 and 12 GE to store a single bit. As a consequence, to store an internal state of say 144 bits (64-bits block state and 80-bits key state), at least 864 GE are required. Storage of the internal state typically accounts for at least 50% of the total area and power consumption. Therefore stream and block cipher implementations for low-cost tag applications should aim to minimize the storage required.

2.3 Combinatorial Elements

The term *combinatorial elements* includes all the basic Boolean operations such as NOT, NAND, NOR, AND, OR, and XOR. It also includes some basic logic functions such as multiplexers (MUX). The gate count for these basic operations is typically independent of the library used. For the *Virtual Silicon* (VST) standard cell library based on the *UMC L180* 0.18μ *1P6M Logic process* (UMCL18G212T3) the figures for two-input gates with the lowest driving strength is given below. Note that in hardware XOR and MUX are rather expensive when compared to the other basic Boolean operations.

Gate	NOT	NAND	NOR	AND	OR	XOR	MUX
GE	0.5	1	1	1.33	1.33	2.67	2.67

2.4 Feedback Shift Registers

A common building block for stream ciphers is the *Feedback Shift Register* (FSR). An FSR inputs and outputs one bit per cycle and the input bit is a function of the previous state. Depending on the feedback function FSRs are either *Linear Feedback Shift Registers* (LFSR) or *Non-linear Feedback Shift Registers* (NFSR).

The hardware implementation of a bit-wise LFSR will consist of flipflops, to hold the register state, and XOR gates to compute the feedback. An LFSR is a reasonably hardware efficient building block. The feedback path, which often consists of a moderate number of binary XOR gates, will typically account for a few dozens GE, while the shift register consisting of flipflops cause a larger gate count. It is important to mention that while hardware efficiency will strive to minimize both the size of the register state and the number of XORs in the feedback path (sometimes called *feedback taps*), a cipher design using short registers or very few feedback taps may become susceptible to cryptanalysis.

NFSRs might use more complex Boolean functions or even *substitution boxes* (see below) in the feedback function. These tend to have a higher gate count then a series of XORs but it is the size of the register, and therefore the number of flipflops, that accounts for the bulk of the area. Therefore the hardware complexity of NFSRs is typically only slightly higher than that of LFSRs.

2.5 Confusion and Diffusion

Shannon [45] was the first to formalize the ideas of *confusion* and *diffusion* as two attractive properties in the design of a secure cipher. In practice, almost all block ciphers are product ciphers, i.e., they are based on subsequent operations of confusion and diffusion. In a block cipher, confusion is often identified with a substitution layer (see below) while diffusion is usually identified with a permutation or "mixing" layer. In reality is not always easy to separate and identify the components that contribute to confusion or diffusion.

Some ciphers use arithmetic operations as a diffusion and confusion technique, but this can significantly increase the area and power consumption. Arguably the most common confusion method is based on S-boxes (see Sect.2.6). A small change in the input to an S-box leads to a complex change in the output. In order to spread these output changes over the entire state quickly, a dedicated diffusion layer has to be applied. The classical way of doing this is to use bit *permutation*. In hardware, bit permutations can be realized with wires and no transistors are involved. They are therefore a very efficient component. Note that more complex diffusion techniques, such as the mix-column layer used in the AES, are also possible. Even though they have cryptographic advantages, they come at a higher hardware cost.

2.6 S-box Design

Many block ciphers, and some stream ciphers, use S-boxes to introduce nonlinearity. In software S-boxes are often implemented as *look-up tables* (LUT). In hardware these LUT can have a large area footprint[2] or they pose technological problems since a mix of combinatorial logic and ROM cannot always be easily achieved with a standard hardware design flow. Hence a purely combinatorial realization is often more efficient.

If combinatorial implementations do not exploit any internal structure in the S-box, then the area requirements will grow rapidly with the number of input and output bits. The more output bits an S-box has, the more Boolean equations will be required. And the more input bits an S-box has, the more complex these equations are likely to be. An interesting interaction between cryptography and hardware implementation can be observed here: In order to withstand differential and linear cryptanalysis [8, 38], high nonlinearity of S-boxes is required, which directly translates into a high gate count. A close look on the hardware efficiency of the S-boxes in AES [41], DES [40], and PRESENT [13] illustrates this.

AES uses a bijective 8-bit S-box, i.e., eight input bits are mapped to eight output bits. In [47], the hardware properties of several implementations of AES S-boxes, each illustrating different design goals, are compared. It turns out that the AES S-box

[2] Note that LUTs with a large memory footprint in software can be vulnerable to side-channel attacks based on *cache misses*.

realized as Boolean logic requires about 1,000 GE while there is no implementation that requires less than 300 GE. These figures also include the inverse S-box.

DES uses eight different S-boxes that map six input bits to four output bits. In [35], the authors state that in their DES ASIC design the S-boxes require in total 742 GE. However, taking into account that Boolean terms can be shared between the eight different S-boxes, it is not surprising that the area requirements for a single 6-bit to 4-bit S-box typically is around 120 GE. This can also be observed in implementations of DESXL and DESL, which will be introduced below. Both algorithms use 6-bit to 4-bit S-boxes but, in contrast to DES, a single S-box is repeated eight times. Therefore only one instance of the S-box has to be implemented in a serialized design, which requires 128 GE.

In [34], the area requirements of so-called *SERPENT-type* S-boxes are described. These are a special subset of 4-bit to 4-bit S-boxes fulfilling certain criteria and the authors found that the area requirements for this type of S-box varied between 21 and 39 GE. As an example, PRESENT uses a single, bijective 4-bit to 4-bit S-box which can be implemented with 21 GE. However, in [13], the authors state that a single S-box requires 28 GE. This deviation is caused by the fact that synthesis results depend heavily on the technology of the standard cells that are used.

3 Lightweight Block Ciphers

A block cipher can be viewed as a family of permutations indexed by a key k. For a block cipher that operates on b-bit blocks, the permutations are of the set of all b-bit inputs. There is a wide variety of design philosophies for block ciphers and the state of the art is well advanced. All the block ciphers of interest to us in this chapter are *iterated* and consist of the repeated application of a *round function*. At each round some key-related information is used to influence the computation, and this key information is derived from the user-supplied key k using a key schedule.

The computation in a single round usually follows one of two topologies. These have been termed a *Feistel* cipher or a *substitution–permutation network* which we will denote by SPN. This article is not an appropriate place to discuss block cipher design in detail, but since the choice of topology has some influence on the efficiency of an implementation, we briefly distinguish between them.

Suppose that we denote the cipher state at the start of round i by $L_i||R_i$ where $||$ denotes bitwise concatenation, and the round key to be used as k_{i+1}. If we further denote the round function in a Feistel cipher by f and the round function in an SPN cipher by g, then loosely speaking we have the following equations for a single round of encryption where the two arguments to f and g are the cipher state and the round key:

$$\text{FEISTEL CIPHER:}\quad L_{i+1} = R_i \quad R_{i+1} = L_i \oplus f(R_i, k_{i+1})$$

$$\text{SPN CIPHER:}\quad L_{i+1}||R_{i+1} = g(L_i||R_i, k_{i+1}).$$

At first sight, if our goal is compact hardware implementation then there appear to be two major advantages of Feistel ciphers when compared to SPN ciphers:

1. The round function f would be identical for encryption and decryption
2. Only part of the cipher state – one half in a classical Feistel cipher – is processed each round

The first property suggests that hardware implementations of Feistel ciphers would reuse the same datapath for encryption and decryption, with only the control logic being adapted. The second property suggests that fewer gates will be required to realize one round of encryption since only part of the cipher state is processed. However, it is notable that the many important block cipher today, e.g., the *Advanced Encryption Standard* (AES) [41] and the most compact block cipher, PRESENT [13], are both SPN ciphers. So do Feistel networks really hold an intrinsic advantage?

It appears not. The first potential advantage of a Feistel cipher, given above, is rarely relevant since for many tag-based applications decryption is not required. For example, when used for authentication in a challenge–response protocol the block cipher needs only to be used in the encryption direction. Also, if a block cipher were to be used as a stream cipher, e.g., when operating in counter or output feedback mode [42] then again, it is only used in the encryption direction. As for the second advantage given above, while it is true that the function f in a Feistel cipher might be more compact than the function g in a substitution–permutation network, this is a little misleading. Feistel ciphers will probably require more encryption rounds to achieve the same level of mixing as an SPN cipher and this can lead to a significant increase in execution time and energy requirements. But, in addition, Feistel ciphers also require additional gates *after* the application of f to mix the untransformed state with the transformed state. Usually the bitwise-XOR is used for this which costs approximately 2.5–3 GE per bit. This is an implementation overhead not required by SPN ciphers and further suggests that the minimum datapath for a serialized implementation is likely to be better with an SPN cipher than with a Feistel cipher. However, it should be noted that the state is the overwhelming proportion of the space requirements for both types of cipher. Thus any difference in the implementation of the round functions is likely to have a limited overall effect.

3.1 State of the Art

There has been an increased interest in the design of lightweight block ciphers. The benchmark implementation against which all others should be measured is the implementation of the AES by Feldhofer et al. [23]. This works show that it is possible to implement the AES in about 3,400 GE, a significant achievement. Yet this is well above the amount of space that we would like to devote to an encryption primitive in many RFID applications. At the same time, the AES arguably offers more security in an RFID-based application than we really need. A new dedicated design might provide a more suitable cost/security trade-off.

Table 1 Comparison of some particularly compact block cipher designs

	Key size	Block size	Cycles per block	Throughput at 100 kHz (kbps)	Logic process	Area GE	Area Rel.
DES [35]	56	64	144	44.4	0.18 µm	2,309	1.47
AES-128 [22]	128	128	1,032	12.4	0.35 µm	3,400	2.17
DESL [35]	56	64	144	44.4	0.18 µm	1,848	1.18
DESX [35]	184	64	144	44.4	0.18 µm	2,629	1.67
DESXL [35]	184	64	144	44.4	0.18 µm	2,168	1.38
PRESENT-80 [13]	80	64	32	200	0.18 µm	1,570	1
PRESENT-128 [13]	128	64	32	200	0.18 µm	1,886	1.20

A look at some older ciphers is quite illuminating. It is well known that DES was designed with hardware efficiency in mind, and DES still has very competitive hardware implementation properties. Implementations of around 3,000 GE [48] exist while a serialized implementation can be realized with around 2,300 GE [35]. The key length of DES limits its usefulness in many applications and makes proposals such as DESXL (2,168 GE) of some considerable interest [35]. This will be discussed further below. Implementation requirements for the *Tiny Encryption Algorithm* TEA [49, 50] are not known, but a crude estimate is that TEA needs at least 2,100 GE while XTEA needs at least 2,000 GE. These are "back-of-an-envelope" figures where we assume that a 32-bit bitwise exclusive-or requires 80 GE, a 32-bit integer addition requires 148 GE, and a 192-bit flipflop requires 1,344 GE. All these estimated figures do not take into account control logic which might significantly increase the required area. Four dedicated proposals for low-cost implementation are MCRYPTON [37], HIGHT [29], SEA [46], and CGEN [44], though the latter is not primarily intended as a block cipher. MCRYPTON has a precise hardware assessment and requires 2,949 GE, HIGHT requires around 3,000 GE while SEA with parameters comparable to PRESENT requires around 2,280 GE. All these figures are given in Table 1.

3.2 Two Dedicated Proposals: DESXL and PRESENT

3.2.1 DESXL

DESXL [35] is, as the name suggests, based on the *Data Encryption Standard* (DES) [40]. DES is a 64-bit block cipher with a 56-bit key. Both its history and structure are well known and details of the algorithm can be found in [40]. Unlike many modern ciphers DES was designed with good hardware properties in mind. However, even when adopted in the mid-1970s, DES was criticized for its short key length of 56 bits and this has only become a more pressing problem over the years [33]. Therefore, DESX was proposed by Rivest as a DES variant with higher resistance to brute-force attacks. This involves a process of prewhitening and postwhitening. DESX encryption using the 184-bit key $k||k_1||k_2$ can be described as

$$\mathrm{DESX}_{k||k_1||k_2}(x) = k_2 \oplus \mathrm{DES}_k(k_1 \oplus x).$$

The effectiveness of this simple technique was demonstrated by Kilian and Rogaway [32].

DESXL is derived from DESX but has two modifications. First the initial and final permutations (IP and IP^{-1}) in DES are omitted. Second, and more crucially, the eight original S-boxes of DES are replaced by a single S-box that is used eight times. The new S-box was chosen by randomly generating all S-boxes that fulfilled the original DES criteria with some additional conditions being added [35]. The goal was a single S-box that offers greater resistance to attacks such as differential and linear cryptanalysis than the original eight S-boxes of DES. The S-box is given below and more details are available in [35].

DESL (and DESXL) S-box: S															
14	5	7	2	11	8	1	15	0	10	9	4	6	13	12	3
5	0	8	15	14	3	2	12	11	7	6	9	13	4	1	10
4	9	2	14	8	7	13	0	10	12	15	1	5	11	3	6
9	6	15	5	3	8	4	11	7	1	12	2	0	14	10	13

3.2.2 Implementation of DESXL

Since DESXL is built around DES, an implementation of DES optimized for constrained environments is needed. An example is given in Fig. 1. This design, which is presented in [35], consists of five core modules: *mem_left*, *mem_right*, *keyschedule*, *controller*, and *sbox*.

Controller. The controller manages all control signals in the ASIC based on a finite state machine.

Keyschedule. This module generates the round keys of DES and consists of a 56-bit register, an input multiplexer, and an output multiplexer.

mem_left. This module consists of eight 4-bit registers, each composed of D-flipflops. Here the memory modules were designed as a shift register so that the output of a 4-bit block can be used as the new input to the following block. At the end of the chain, the current 4-bit block is provided and can be processed without an additional output multiplexer. This results in a saving of 48 GE.

mem_right. This module is similar to the *mem_left*. It consists of eight 4-bit wide registers, but it has different input and output signals. Instead of a 4-bit wide output it has a 6-bit wide output which accounts for the *expansion* function in DES. This design in a shift register manner saves even more area (72 GE) than in the *mem_left* module because, in this case, a 6-bit wide output multiplexer can be saved. Altogether 120 GE can be saved using this memory design when compared to regular approaches.

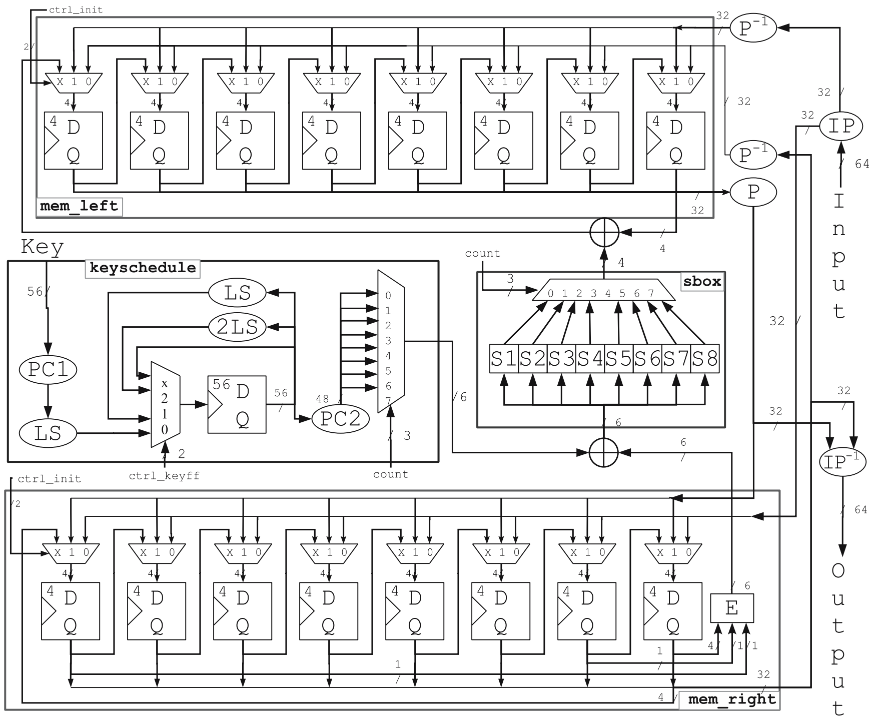


Fig. 1 Datapath of a serialized DES ASIC

sbox. For DES this module consists of the eight DES S-boxes and an output multiplexer. The S-boxes are realized in combinatorial logic, i.e., as a sum of products [20].

In the description that follows, it is assumed that the reader is familiar with the specifications of DES [40]. The 56-bit *key* is stored in the key flipflop register, after the PC1 and LS1 permutations have been applied, while the plaintext is mixed using the DES *initial permutation* and split into two 32-bit inputs to the modules *mem_left* and *mem_right*, respectively. The input of *mem_left* is modified by the inverse of the *P* permutation and stored in the registers of the modules *mem_left* and *mem_right* in one cycle. The output of the last register in *mem_right* is both stored in the first register of *mem_right* and expanded to six bits. After a bitwise exclusive-or operation with the appropriate block of the current round key, the expanded value is processed by the *sbox* module, which is selected by the *count* signal provided by the *controller* module. The result is bitwise exclusive-ored with the output of the *mem_left* module and stored in the first flipflop of the *mem_left* module. This is repeated eight times until all 32 bits of the right half are processed. By reducing the datapath from a 32-bit bus to a 4-bit bus, only $(6 \times 10) + (4 \times 10) = 100$ transistors (25 GE) are needed for the bitwise exclusive-or operations, compared to $(48 \times 10) + (32 \times 10) = 800$ (200 GE) transistors in a nonserial design. This saving comes at the cost of two additional multiplexers, one for the round key (72 GE) and one for the S-box output (48 GE). However, the second multiplexer is avoided in the final specification of DESXL.

Once all eight 4-bit blocks of both halves have been processed, they are concatenated to two 32-bit wide outputs of the modules *mem_left* and *mem_right*. The output of the module *mem_left* is transformed by the *P* permutation and stored as the new content of the *mem_right* module, while the output of the *mem_right* module is stored as the new content of the *mem_left* module. This execution flow repeats for another 15 rounds. Finally, both outputs from *mem_left* and *mem_right* are concatenated to give a 64-bit wide output and after IP^{-1} the ciphertext is generated.

The results of implementing this DES architecture are given in [35, 43] and summarized in Table 1. The registers comprise the majority of the chip size (33.78%), followed by the S-boxes (32.11%) and multiplexers (31.19%). Since the chip size of registers and multiplexers cannot be reduced it is natural to consider the space occupied by the S-boxes. And since there are no better logic minimizations of the original DES S-boxes, the designers of DESXL decided to use a new, single S-box repeated eight times. This resulted in a proposal called DESL.

For implementation, the main difference between DESL and DES lies in the f-function. The eight original DES S-boxes are replaced by a single S-box (see Table in Sect. 3.2.1) which is repeated eight times. This has implications for the design of the *sbox* module. As can be seen in Fig. 2, in a serialized design the S-box module is dramatically simplified. Another minor difference is that DESL omits the initial permutation IP and its inverse IP^{-1} for the sake of simplicity.

Adding the prewhitening and postwhitening to these algorithms obviously has an impact on the space required. Since we assume that all keys have to be stored on the RFID-tag in a nonvolatile memory and both the prewhitening and the postwhitening key never change, no additional flipflops are required for this operation. Therefore only two additional XOR-gates of 64 bits are required to perform the prewhitening and postwhitening. The gate count difference between DESXL and DESL (and also between DESX and DES) is 320 additional GEs. Figure 2 depicts the datapath of a serialized DESXL ASIC.

For the implementation of all these variants [35, 43] *Synopsys Design Vision V-2004.06-SP2* was used to map the design to the *Artisan UMC* 0.18μ*m L180 Process* 1.8-*Volt Sage-X Standard Cell Library* and *Cadence Silicon Ensemble 5.4* for the Placement & Routing-step. *Synopsys NanoSim* was used to simulate the power consumption of the back-annotated verilog netlist of the ASIC. The implementation of DES requires 2,309 GE and 144 clock cycles are required to encrypt one 64-bit block of plaintext. For one encryption at 100 kHz the average current consumption is 1.19 µA and the throughput reaches 5.55 kB s^{-1}. For the serialized DESL ASIC implementation, the area requirement was 1,848 GE and, again, 144 clock cycles were needed to encrypt one 64-bit block of plaintext. For one encryption at 100 kHz the average current consumption was 0.89 µA with throughput reaching 5.55 kB s^{-1}.

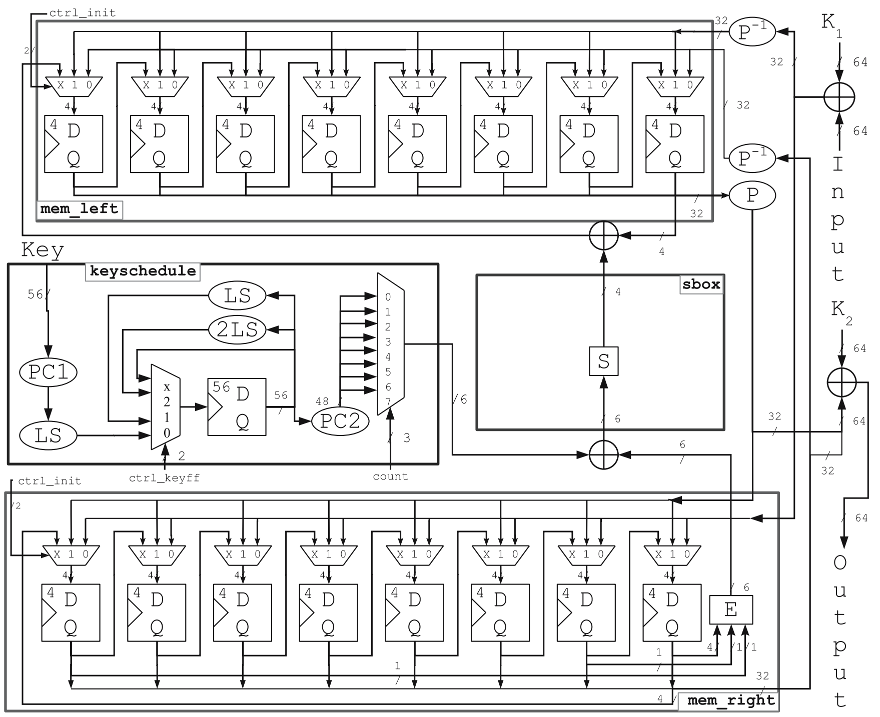


Fig. 2 Datapath of a serialized DESXL ASIC

3.2.3 Description of PRESENT

The block cipher PRESENT was designed with security, efficient implementation, and simplicity in mind. PRESENT is a 64-bit SPN block cipher with an 80-bit key. This is sometimes referred to as PRESENT-80 to differentiate it from PRESENT-128 which uses 128-bit keys. Encryption and decryption with PRESENT have roughly the same physical requirements and the encryption subkeys can be computed *on-the-fly*. PRESENT is described in pseudocode in Fig. 3 while details and design rationale can be found in [13]. The topology over two rounds is illustrated in Fig. 4.

PRESENT uses a single 4-bit to 4-bit S-box which is applied 16 times in parallel in each round. This was a direct consequence of the pursuit of hardware efficiency. Since a bit permutation is used as a linear diffusion layer, AES-like diffusion techniques [15] were not an option for PRESENT. Therefore some additional conditions were placed on the S-boxes to improve the so-called *avalanche of change*. Despite this, the S-box is particular well suited to efficient hardware implementation and is given below in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

```
generateRoundKeys()
**for** $i = 1$ to 31 **do**
    addRoundKey(STATE,$K_i$)
    sBoxLayer(STATE)
    pLayer(STATE)
**end for**
addRoundKey(STATE,$K_{32}$)
```

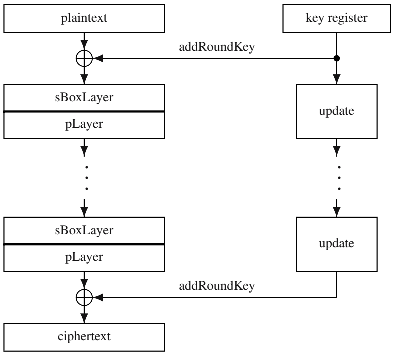


Fig. 3 A top-level algorithmic description of PRESENT

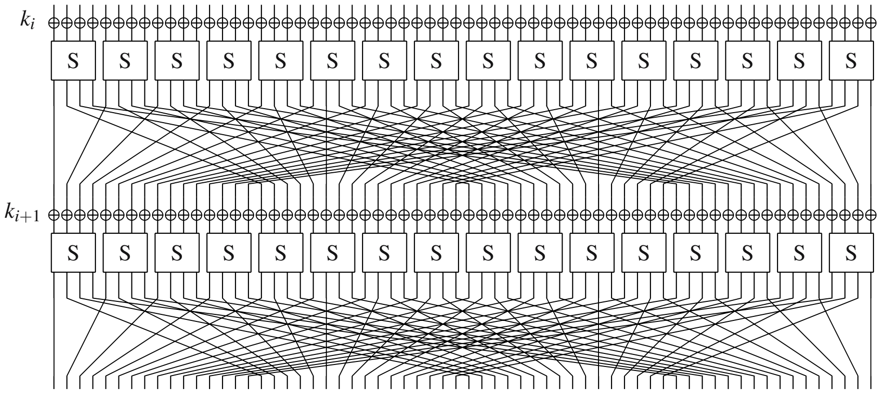

k_i
S S S S S S S S S S S S S S S S
k_{i+1}
S S S S S S S S S S S S S S S S

Fig. 4 Two round topology of PRESENT

3.2.4 Implementation of PRESENT

PRESENT-80 was implemented in VHDL and synthesized for the *Virtual Silicon* (VST) standard cell library based on the *UMC L180* 0.18μ *1P6M Logic process*. The authors used *Mentor Graphics Modelsim SE PLUS 5.8c* for simulation and *Synopsys Design Compiler* version *Y-2006.06* for synthesis and power simulation [13]. Figure 5 shows the datapath of an area-optimized encryption-only PRESENT-80, which performs one round in one clock cycle, i.e., a 64-bit width datapath. The implementation requires 32 clock cycles to encrypt a 64-bit plaintext with an 80-bit key, occupies 1,570 GE and has a simulated power consumption of 5µW.

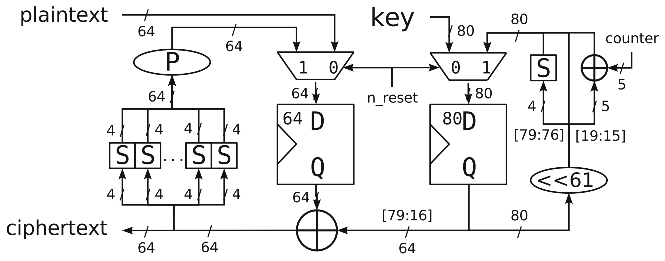


Fig. 5 The datapath of an area-optimized version of PRESENT-80

Module	GE	(%)	module	GE	(%)
Data state	384.39	24.48	KS: key state	480.49	30.61
s-Layer	448.45	28.57	KS: S-box	28.03	1.79
p-Layer	0	0	KS: Rotation	0	0
Counter: state	28.36	1.81	KS: counter-XOR	13.35	0.85
Counter: combinatorial	12.35	0.79	Key-XOR	170.84	10.88
Other	3.67	0.23			
			sum	*1,569.93*	*100*

The bulk of the area is occupied by flipflops for storing the key and the data state, followed by the S-layer and the key-XOR. Bit permutations are simple wiring and will increase the area only when the implementation is taken to the place and route-step. The main goal of the implementation was a small footprint in hardware, however, it has also been synthesized in a power-optimized implementation. For an additional 53 GE the power consumption is only 3.3 μW. Estimates also suggest that PRESENT-128 would occupy an approximate area of 1,886 GE. Beside a very small footprint PRESENT has a rather high throughput giving good energy-per-bit.

4 Lightweight Stream Ciphers

As was mentioned in Sect. 1, it is often suggested that stream ciphers of a dedicated design might be well suited to exacting conditions. Perhaps they offer particularly aggressive performance on some platform or perhaps they offer a particularly compact footprint in hardware. While the advanced state of block cipher research is beginning to challenge this view – it is not clear that a well-designed block cipher might not outperform a stream cipher in both regards – there have been considerable recent advances in the design of stream ciphers.

A stream cipher generates a keystream by sampling an evolving cipher state. Typically, for a dedicated stream cipher, the state is initialized under the action of a secret key k and an *initialization vector* (IV) v. By using an initialization vector different keystreams can be generated without a change of key and there is a mechanism – sending a new public value v – by which sender and receiver can synchronize with one another. Depending on the design of the stream cipher, state initialization might be closely related to keystream generation or it might be an entirely different process. It is well known that a keystream generated by a stream cipher with a finite cipher state must (eventually) repeat and its *period* must be sufficiently large. In practice, many stream ciphers come with an explicit upper-bound on the amount of keystream that can be generated after which the initialization vector, the key, or both are changed.

4.1 State of the Art

In contrast to block ciphers, the field of stream ciphers is very fragmented and there are no stream ciphers with the international profile of DES and the AES. As a consequence, the state of the art of the design and analysis of stream ciphers is not as well-developed as that of block ciphers. At first sight this might be a little surprising. After all, there is a very rich theory [36] surrounding the use of *Linear Feedback Shift Registers* (LFSRs) and for many years these have been used in the construction of stream ciphers. Yet, in some sense this has helped lead to the fragmentation of the field. Since stream ciphers can be built component-wise using these building blocks, it can be tempting for application developers to design subtly different stream ciphers that can be deployed in a proprietary manner. Despite this fragmentation, however, there are two stream ciphers of particular note. Somewhat ironically, given the reputation of stream ciphers for compact implementation, these two ciphers both occupy more space than most contemporary block ciphers.

The first of these two stream ciphers has been, and continues to be, used in many applications. RC4 was designed by Rivest in the mid-1990s and was provided as a proprietary cipher by RSA Data Security. The adoption of RC4 in industry has been widespread, e.g., [16, 30], and even though its confidentiality was compromised nearly a decade ago it remains, at core, a sound cipher. That said, it is showing its age and a variety of recommendations on how best to use the cipher help to protect the user from some unfortunate irregularities in the keystream. But independently of that, and in the applications of interest to us here, RC4 is completely unsuitable for environments with very restricted space. It should be noted that RC4 was primarily designed as a software-friendly cipher. The second stream cipher of note, SNOW 2.0 [18], is ISO-standardized and sports a contemporary design that meets modern performance requirements. While SNOW 2.0 might not be found in many products or applications, its success has lead to the design of variants such as SNOW 3G. However, SNOW 2.0 is also unsuitable for applications when space is limited. Both RC4 and SNOW 2.0 have a very large cipher state and rough estimates suggest that they would occupy around 12,000 and 7,000 GE, respectively.

To look for more compact stream ciphers we might turn to a host of proposals based on LFSRs and these typify the usual industry designs of the 1980s and 1990s. With LFSRs, however, it is difficult to achieve good performance, compact design, and good security at the same time. One of the most prominent stream ciphers of the era was A5/1. Its ensemble of three LFSRs with an irregular clocking mechanism require a very satisfying 1,000 GE [3], though unfortunately its cryptanalysis was equally impressive [10]. The cipher E0 [12] used in Bluetooth is another prominent shift-register-based stream cipher and estimates for its implementation suggest that over 1,600 GE are required.

The difficulty of designing a secure stream cipher is, to a great extent, a function of the way it operates. As well as a key the cipher typically uses an initialization value. Thus the cipher can be repeatedly initialized with different IVs that are known to an attacker while the secret key remains unchanged. Also, the key and IV are typically used to initialize the state of the stream cipher, and after this time the state evolves without any influence from the secret key. These are very special attributes to a cipher and it is not surprising that they lead to very special design demands. These properties have also lead to a series of time–memory–data trade-offs giving us lower bounds on the sizes of the state of a stream cipher. They also help us understand better the relationship between the sizes of the key, the IV, and the process of state initialization [1, 11, 19].

4.2 The eSTREAM Project

The eSTREAM project [17] is part of the ECRYPT Network of Excellence, and the goal of the project is to deliver a small portfolio of promising stream ciphers. The project is expected to end by May 2008.

At the start, the eSTREAM project identified stream ciphers for use in two very different ways. The first, labeled Profile 1, was for stream ciphers that could provide fast throughput in software. The second, labeled Profile 2, is of more interest to us here and required that stream ciphers be suitable for use in highly constrained environments. For Profile 2 (compact hardware) submissions, the must-satisfy values for the key and the IV lengths were 80 bits and at least one of 32 or 64 bits, respectively. Note that these values reflect the reasonable belief that the security level can, to some extent, be compromised so as to gain an implementation advantage.

The last round of the eSTREAM project featured 16 ciphers, eight for software and eight for hardware. At the time of writing, all have resisted cryptanalysis though some have been modified from their original submission. For entry in the final round of eSTREAM the main criterion after security for the hardware ciphers was the amount of space required for an implementation. For all eight final Profile 2 submissions there was good evidence that their implementation would require less space than an implementation of the AES and, in particular, the AES implementation of Feldhofer et al. that requires around 3,400 GE [22]. It appears that the candidates fall into three rough bands as shown in Table 2 : those that are slightly better than the AES, those that are certainly better than the AES, and two that appear to offer an exceptional performance profile.

Cryptanalysis may well take its toll on some of these finalists, but the space requirements for Grain v1.0 and Trivium are striking. When looking beyond the space

Table 2 Approximate gate count of some Profile 2 stream ciphers of the eSTREAM project

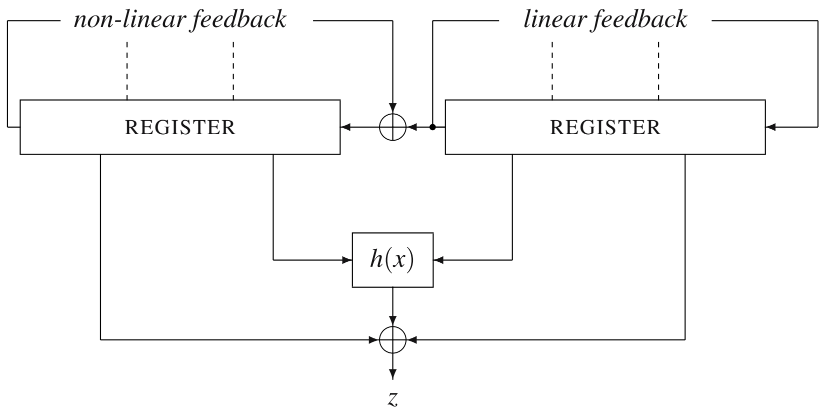
Algorithm	≈GE	Algorithm	≈GE	Algorithm	≈GE
Mickey v2.0 [2]	3,400	Decim v2.0 [4]	3,000	Grain v1.0 [28]	1,300
Pomaranch v3.0 [31]	3,300	Edon80 [24]	2,900	Trivium [14]	2,300
F-FCSR-H v2.0 [6]	3,200				

requirements we also find that these two algorithms are particularly amenable to low-cost hardware implementation; they offer great flexibility in their implementation so as to get a wide-range of performance metrics [25] as well as low-power implementations [21].

4.3 Two Dedicated Proposals: Grain and Trivium

4.3.1 Grain

Despite its version number Grain v1.0 [28] is the second version of the cipher; the first was broken [5] during the first phase of the eSTREAM project. Grain v1.0 consists of two feedback shift registers, one of which uses linear feedback while the other uses nonlinear feedback. Grain v1.0 offers 80-bit security, a level of security that is widely viewed as appropriate for the lower value applications often associated with RFID tag-based deployments. There is a variant – referred to as Grain-128 [27] – that offers 128-bit security as the name implies. A schematic overview of Grain v1.0 is given below.



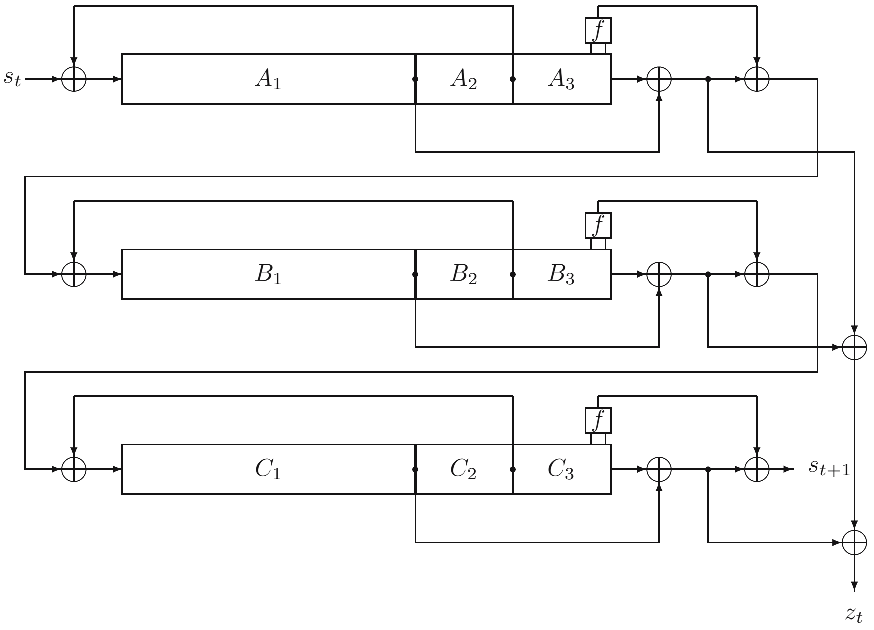
Each register is 80 bits in length and the function h is a boolean function that takes input bits from the two registers. It has been carefully chosen to provide particular cryptographic properties; more details can be found in [26, 28].

The hardware performance of Grain v1.0 has been studied closely in a variety of papers. One particularly nice feature of the algorithm, and one that is shared to some extent by Trivium, is the wide-ranging performance profile. In [25], for instance, it is shown that implementations oriented toward RFID tags, where reduced space is the driving metric, offer implementations requiring around 1,200 GE. Figures in [21] are derived from implementations that strive to minimize energy consumption; here the size might increase to 3,360 GE but with an electric current of merely 0.8 µA when clocked at 100 kHz.

4.3.2 Trivium

A second promising candidate from the eSTREAM project is Trivium [14]. Like Grain v1.0, it offers a wide rage of implementation options. A schematic overview of Trivium is given below.

Trivium uses three shift registers of lengths 93, 84, and 11, 1 respectively. From each register, internal bits are used as feedforward and feedback into the updating of the state, more details can be found in [14]. This bit extraction naturally divides each register into three parts and we have the following values which define the position of the bit extraction; $A_1 = 66$, $A_2 = 3$, $A_3 = 24$, $B_1 = 69$, $B_2 = 9$, $B_3 = 6$, $C_1 = 66$, $C_2 = 21$, and $C_3 = 24$. The function f takes the third- and second-to-last bits of each register, say x and y, and outputs the bitwise AND so $f(x,y) = xy$.



The total register size in Trivium is 288 bits and this translates into a larger implementation than Grain v1.0. Implementations oriented toward RFID tags in [25], for instance, offer implementations requiring around 2,300 GE, but the structure of Trivium allows for greater parallelization and hence a greater throughput. Figures in [21] are derived from implementations that strive to minimize energy consumption; here the size might increase to around 3,090 GE but with a current consumption of 0.68 μA when clocked at 100 kHz.

5 Conclusions

In this article, we have considered some innovative approaches in the design of low-cost symmetric encryption algorithms. This field of research is very active and much progress is being made in the design and implementation of both compact block and compact stream ciphers.

However, it is interesting to ask whether we really need both types of algorithms? When looking at the performance of Grain v1.0 and Trivium, one is tempted to say "yes." We get what appears to be very good encryption at roughly half the cost in space of using the AES. However, this is not necessarily a fair comparison since the AES was designed to be suitable for both software and hardware and offers three very high levels (128-, 192- and 256-bit) of security. By contrast Grain v1.0 and Trivium were exclusively designed with hardware implementation in mind and intended to offer only a single level of 80-bit security. It seems that the block cipher PRESENT offers a better point of comparison, but the space requirements for PRESENT are around 1,500 GE (see earlier). This is very close to those for Grain v1.0 and Trivium.

To be fair, a little more space is required to accommodate counter mode (say) if we were to use PRESENT as a stream cipher, but even so it is a reasonable indication that all three contemporary ciphers of a dedicated design require similar amounts of space. Perhaps this is not too surprising. Any given security level gives immediate requirements on the amount of state that we need. This means that we need to be using around 900 GE as the working space for any symmetric cipher that aims to offer, say, 80-bit security. We then need to add the space required for the different cipher operations and it seems that optimized ciphers (either stream or block) are able to make do with between 300 and 500 GE as an operational overhead.

Whether or not the specific proposals of Grain v1.0, Trivium, or PRESENT survive the efforts of cryptanalysts, there is no real reason to suppose that a compact block cipher necessarily requires more space than a compact stream cipher offering comparable security. To the authors of this article, the distinguishing feature between block and stream cipher implementation is unlikely to be the space required. Instead, any significant implementation difference may turn out to be the levels of parallelism and opportunities for low-energy optimization that are afforded. Depending on the design, these aspects appear to be more easily exploited with a stream cipher than with a block cipher. In the end, this may turn out to be the feature that distinguishes the two classes of symmetric encryption most when considering their design and implementation. Providing convincing evidence to the contrary is left as an open problem for the reader.

References

1. S. Babbage. A Space/Time Trade-off in Exhaustive Search Attacks on Stream Ciphers. *IEE European Convention on Security and Dectection*, 408, 1995
2. S. Babbage and M. Dodd. *MICKEY 2.0*, 2006 Available via `www.ecrypt.eu.org/stream`
3. L. Batina, J. Lano, N. Mentens, S. Berna Örs, B. Preneel, and I. Verbauwhede. Energy, Performance, Area Versus Security Trade-offs for Stream Ciphers. *State of the Art of Stream Ciphers 2004 (SASC 2004)*, Workshop Record, pp. 302–310, 2004. Available via `www.ecrypt.eu.org/stream`
4. C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. *DECIM v2.0*, 2006. Available via `www.ecrypt.eu.org/stream`
5. C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In M. Robshaw, editor, *Proceedings of FSE 2006*, volume 4047 of LNCS, pp. 15–29, Springer, Berlin, 2006
6. T. Berger, F. Arnault, and C. Lauradoux. F-FCSR-H v2.0. Available via `www.ecrypt.eu.org/stream`
7. E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In T. Helleseth, editor, *Proceedings of Eurocrypt'93*, volume 765 of LNCS, pp. 398–409, Springer, Berlin, 1994
8. E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In *Proceedings of CRYPTO*, pp. 487–496, 1992. Also available via `citeseer.ist.psu.edu/biham93differential.html`
9. A. Biryukov and A. Shamir. Cryptanalytic Time/Memory Trade-offs for Stream Ciphers. In T. Okamoto, editor, *Proceedings of Asiacrypt 2000*, volume 1976 of LNCS, pp. 1–13, Springer, Berlin, 2000
10. A. Biryukov, A. Shamir, and D. Wagner. Real-Time Cryptanalysis of A5/1 on a PC. In B. Schneier, editor, *Proceedings of FSE 2000*, volume 1978 of LNCS, pp. 37–44, Springer, Berlin, 2000
11. A. Biryukov, S. Mukhopadhyay, and P. Sarkar. Improved Time-memory Trade-offs with Multiple Data. In B. Preneel and S. Tavares, editors, *Proceedings of SAC 2005*, volume 3897 of LNCS, pp. 110–127, Springer, Berlin, 2005
12. S.I.G. Bluetooth *Specification of the Bluetooth System*, 2003. Available via `www.bluetooth.org/spec`, version 1.2
13. A. Bogdanov, G. Leander, L.R. Knudsen, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT – An Ultra-Lightweight Block Cipher. In *Proceedings of CHES 2007*, volume 4727 of LNCS, pp. 450 – 466, Springer, Berlin, 2007
14. C. de Cannière and B. Preneel. *Trivium*. Available via `www.ecrypt.eu.org/stream`
15. J. Daemen and V. Rijmen. *The Design of Rijndael*, Springer, Berlin, 2002
16. T. Dierks and C. Allen. *The TLS Protocol*. Available via `www.ietf.org/rfc/rfc2246.txt`
17. ECRYPT Network of Excellence. *The Stream Cipher Project: eSTREAM*. Available via `www.ecrypt.eu.org/stream`
18. P. Ekdahl and T. Johansson. A New Version of the Stream Cipher SNOW. In K. Nyberg and H. Heys, editors, *Proceedings of SAC 2002*, volume 2595 of LNCS, pp. 47–61, Springer, Berlin, 2002
19. H. Englund, M. Hell, and T. Johansson. A Note on Distinguishing Attacks. In T. Helleseth, P. Kumar, and O. Ytrehus, editors, *Proceedings of 2007 IEEE Information Theory Workshop on Information Theory for Wirless Networks*, pp. 87–90, 2007

20. *Espresso*. Available via `http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm`
21. M. Feldhofer. Comparison of Low-Power Implementations of Trivium and Grain. *State of the Art of Stream Ciphers 2007 (SASC 2007)*, Workshop Record, February 2007. Available for download via `http://www.ecrypt.eu.org/stream/`
22. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the *AES* algorithm. In M. Joye and J.-J. Quisquater, editor, *Proceedings of CHES 2004*, volume 3156 of LNCS, pp. 357–370, Springer, Berlin, 2004
23. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *Information Security, IEE Proceedings*, 152(1): 13–20, 2005.
24. D. Gligoroski, S. Markovski, L. Kocarev, and M. Gusev. *Edon80*. Available via `www.ecrypt.eu.org/stream`
25. 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`
26. M. Hell. *On the Design and Analysis of Stream Ciphers*. PhD Thesis, Lund University, 2007
27. M. Hell, T. Johansson, A. Maximov, and W. Meier. A Stream Cipher Proposal: Grain-128. In *IEEE International Symposium on Information Theory – ISIT 2006*, 2006. Also available via `www.ecrypt.eu.org/stream`
28. M. Hell, T. Johansson, and W. Meier. *Grain – A Stream Cipher for Constrained Environments*, International Journal of Wirelerss and Mobile Computing, 2(1): 86–93, 2007. Available via `www.ecrypt.eu.org/stream`
29. 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, *Proceedings of CHES 2006*, volume 4249 of LNCS, pp. 46–59, Springer, Berlin, 2006
30. IEEE. *802.11 LAN/MAN Wireless LANS*, 2007. Available via `standards.ieee.org/getieee802/`
31. C. Jansen, T. Helleseth, and A. Kholosha. *Pomaranch v3.0.* Available via `www.ecrypt.eu.org/stream`
32. J. Kilian and P. Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*, 14(1): 17–35, 1996. Available for download at `citeseer.ist.psu.edu/article/kilian96how.html`
33. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA – A Cost-Optimized Parallel Code Breaker. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2006, Yokohama, Japan*, Springer, Berlin, 2006
34. G. Leander and A. Poschmann. On the Classification of 4-Bit S-boxes. In C. Carlet and B. Sunar, editors, *Proceedings of WAIFI 2007*, volume 4547 of LNCS, Springer, Berlin, 2007
35. G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lighweight DES Variants. In *Proceedings of Fast Software Encryption 2007 – FSE 2007*, volume 4593 of LNCS, pp. 196–210, Springer, Berlin, 2007
36. R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, Cambridge, MA Revised edition, 1994
37. C. Lim and T. Korkishko. mCrypton – A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In M. Yung, J. Song, and T. Kwon, editor, *Workshop on Information Security Applications – WISA'05*, volume 3786 of LNCS, pp. 243–258, Springer, Berlin, 2005
38. M. Matsui. Linear Cryptanalysis of DES Cipher. In T. Hellenseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume of 0765 LNCS, pp. 286 – 397, Springer, Berlin, 1994
39. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, First edition, 1996
40. National Institute of Standards and Technology. *Data Encryption Standard (DES)*. Federal Information Processing Standards (FIPS) Publication 46-3, October 1999

41. National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. Federal Information Processing Standards (FIPS) Publication 197, November 2001. Available via `csrc.nist.gov`
42. National Institute of Standards and Technology. *SP800-38A: Recommendation for Block Cipher Modes of Operation*. Available via `csrc.nist.gov`, December 2001
43. A. Poschmann, G. Leander, K. Schramm, and C. Paar. New Lighweight Crypto Algorithms for RFID. In *Proceedings of The IEEE International Symposium on Circuits and Systems 2007 – ISCAS 2007*, pp. 1843–1846, 2007
44. M.J.B Robshaw. Searching for Compact Algorithms: CGEN. In P.Q. Nguyen, editor, *Proceedings of Vietcrypt 2006*, volume 4341 of LNCS, pp. 37–49, Springer, Berlin, 2006
45. C.E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4): 656–715, 1949
46. 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*, volume 3928 of LNCS, pp. 222–236, Springer, Berlin, 2006
47. S. Tillich, M. Feldhofer, and J. Großschädl. Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-Box. In *Proceedings of Embedded Computer Systems: Architectures, Modeling, and Simulation – SAMOS 2006*, volume 4917 of LNCS, pp. 457 – 466, Springer, Berlin, 2006
48. I. Verbauwhede, F. Hoornaert, J. Vandewalle, and H. De Man. Security and Performance Optimization of a New DES Data Encryption Chip. *IEEE Journal of Solid-State Circuits*, 23(3): 647–656, 1988
49. D. Wheeler and R. Needham. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *Proceedings of FSE 1994*, volume 1008 of LNCS, pp. 363–366, Springer, Berlin, 1994
50. D. Wheeler and R. Needham. TEA Extensions. October 1997. Available via `www.ftp.cl.cam.ac.uk/ftp/users/djw3/` (Also Correction to XTEA. October, 1998)