Available online at www.sciencedirect.com

ScienceDirect

INTEGRATION, the VLSI journal 40 (2007) 479–489

INTEGRATION the VLSI journal

www.elsevier.com/locate/vlsi

A fast pipelined multi-mode DES architecture operating in IP representation☆

Sylvain Guilley[a,b,*], Philippe Hoogvorst[a,b], Renaud Pacalet[a,c]

[a]*GET/Télécom Paris, CNRS LTCI (UMR 5141), Département communication et électronique, France*
[b]*46 rue Barrault, 75634 Paris Cedex 13, France*
[c]*Institut Eurecom BP 193, 2229 route des Crêtes, F-06904 Sophia-Antipolis Cedex, France*

Received 21 December 2005; received in revised form 2 May 2006; accepted 30 June 2006

Abstract

The Data Encryption Standard (DES) is a cipher that is still used in a broad range of applications, from smartcards, where it is often implemented as a tamper-resistant embedded co-processor, to PCs, where it is implemented in software (for instance, to compute `crypt(3)` on UNIX platforms). To the authors' knowledge, implementations of DES published so far are based on the straightforward application of the NIST standard. This article describes an innovative architecture that features a speed increase for both hardware and software implementations, compared to the state of the art. For example, the proposed architecture, at constant size, is about twice as fast as the state of the art for 3DES-CBC. The first contribution of this article is an hardware architecture that minimizes the computation time overhead caused by key and message loading. The second contribution is an optimal chaining of computations, typically required when "operation modes" are used. The optimization is made possible by a novel computation paradigm, called "IP representation".

Keywords: Data Encryption Standard (DES); Triple-DES; Modes of operation; Pipeline; IP representation

1. Introduction

The Data Encryption Standard, DES, is a block product cipher algorithm promoted by the NIST. The latest version of the standard is known as FIPS 46-3 [1], and includes the definition of "triple DES". The "DES modes of operation", standardized in FIPS 81 [2], is a companion document devoted to the description of the secure use of DES when the messages to encrypt are longer than 8 bytes.

Since its inception, DES has been used pervasively by many applications that require data confidentiality. However, from year 2001, DES has been superseded by the Advanced Encryption Standard (AES) [3]. But in practice, a lot of hardware or software applications still resort to DES.

The DES algorithm turns a 64-bit confidential data block, nicknamed *plaintext*, into another 64-bit data block, nicknamed *ciphertext*, using a standardized bijection parametrized by a 56-bit secret, nicknamed *key*. The bijection DES_k is crafted in such a way it is almost impossible to retrieve the plaintext from the ciphertext without the knowledge of the key k. The bijection can be inverted: this operation is called *decipherment* and noted DES_k^{-1}. When it is not relevant whether the algorithm performs encipherment or decipherment, the neologism "*cipherment*" is employed instead.

Several attacks against the plain DES version were published. They can basically be classified into two categories: *algorithmical* and *physical* attacks.

Algorithmical attacks are also referred to as cryptanalysis [7,8]. Those analyzes are somehow unrealistic, since a large amount of $\{\text{plaintext}, \text{ciphertext}\}$ couples must be

☆This work has been partly funded by the Conseil Régional "Provence Alpes Côte d'Azur", the STMicroelectronics AST Division (Rousset, France) and the French Ministry for Research, through ACI "Sécurité Informatique" MARS ⟨ http://www.comelec.enst.fr/recherche/mars/⟩.

*Corresponding author. GET/Télécom Paris, CNRS LTCI (UMR 5141), Département communication et électronique, France. Tel.: +33 145818333; fax: +33 145804036.

E-mail address: sylvain.guilley@enst.fr (S. Guilley).

0167-9260/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2006.06.004

intercepted. The exhaustive search of the key [9] has been *publicly* feasible since 1977, as proved by the RSA Laboratory's "DES Challenge II" being won in 1997 in 39 days by a network of computers running the distributed application DESCHALL and in 1998 in 3 days by a dedicated machine built by the Electronic Frontier Foundation (EFF). Other methods to speed-up the search using pre-computated data sets have been put forward [10].

To counteract those attacks, variants of the DES were proposed. We list below three of the most widespread ones:

(1) Modes of operation allow a message consisting of several 64-bit blocks to be ciphered in chain. The idea is that the knowledge of each of the 64-bit ciphertext blocks actually depends on the corresponding plaintext block, also of some, if not all, of the previous ones, and of an initialization vector (IV). The standardized modes of operation are ECB, CBC, CFB and OFB [2]; they were reinforced by ISO/IEC 10116 [5]. ECB and CBC are *block-ciphers*, whereas CFB and OFB are *stream-ciphers*. The latter two are actually defined in the K-bit version, $1 \leqslant K \leqslant 64$. As the $K = 64$ version is the most efficient in terms of throughput, it is usually the sole version to be implemented (refer for instance to `openssl`[11]). Because of the "short cycle property", NIST explicitly does not support $K < 64$ for OFB [12, p. 13].

(2) TDEA (informally called "triple-DES" or "3DES") is described in the annex of the DES standard [1, p. 22]. Three 64-bit keys $k_i, i \in \{0, 1, 2\}$ are used instead of one. The encipherment consists in computing $\mathrm{DES}_{k_2} \circ \mathrm{DES}_{k_1}^{-1} \circ \mathrm{DES}_{k_0}$, whereas decipherment is $\mathrm{DES}_{k_0}^{-1} \circ \mathrm{DES}_{k_1} \circ \mathrm{DES}_{k_2}^{-1}$, where "$\circ$" denotes the composition operator. Triple DES is customarily used with two keys [6,4,13] (i.e. $k_0 = k_2$). Notice that when the three keys are taken equal, $k_0 = k_1 = k_2$, triple DES actually computes plain DES, which guarantees the backward compatibility of 3DES engines.

(3) DESX [14] is a data whitening technique, proposed by Ron Rivest. It consists in adjoining two 64-bit blocks, `in_white` and `out_white`, to the key. The key `in_white` is used to *exclusive-or* (i.e. `XOR`) the plaintext prior to starting DES and `out_white` to `XOR` the result after the cipherment.

Those variants can of course be combined at will. For instance, triple-DES using two keys in CBC mode is often used to encipher long messages.

Physical attacks are the most recent threats against DES and its variants. The side-channel attacks, such as DPA (differential power analysis [15]) or EMA (electromagnetic analysis [16]), allow to retrieve the keys by the analysis of the physical emanation of the device while it is handling the key. Partial side-channel information, such as the Hamming weight of key chunks or key-dependent correlations between two small chunks of data, suffice to recover the full key, provided enough measurements can be performed. The faults injection attacks [17] consist in either perturbing transiently the circuit or to damage it to enhance other attacks. Algorithmical counter-measures (modes of operation, 3DES or DESX) do not protect against physical attacks. Both side-channel and fault attacks can be thwarted, with more or less success, by using leakage-proof logic and adequate sensors, for instance.

In this paper, we describe an architecture able to compute DES and its variants efficiently. More precisely, the described architecture can compute: DES in ECB, CBC, 64-bit CFB and 64-bit OFB, as well with simple or triple DES using two keys. The cryptanalytic strength of the variant as well as the security of its implementation against physical attacks is out of the scope of this paper.

The rest of the article is organized as follows. Section 2 discusses the DES datapath optimization: a hardware pipelined architecture is presented. Section 3 applies to both software (SW) and hardware (HW) implementations. It introduces the so-called "IP representation" computational framework, which allows to optimally chain DES computations. In Section 4, the gain of proposed architecture over state-of-the-art architectures is discussed. Finally, Section 5 summarizes the paper.

2. DES datapath improvement thanks to a generalized pipelining

In the DES algorithm, the control is independent of the data. It is thus safe to consider the design of the datapath and the control finite state machine (FSM) as two distinct tasks. This section is devoted to the datapath. The control is further studied in Section 3.

2.1. Straightforward DES

The inputs of the DES algorithm are two 64-bit blocks, the plaintext and the key. The two operands cannot be loaded in the DES operator in one go, since data provided by processors are typically on $n = 8$, 16 or 32 bits. In the rest of the article, we assume that the DES co-processor is fed by an $n = 8$-bit wide data bus. This figure corresponds to the case of an embedded system built around a microcontroller, as depicted in Fig. 1.

Most of DES implementations elude the question of the connexion to an $n < 64$ wide bus [18–20]. Some implementations, such as [21], use $n = 32$, but do not take advantage of the architectures presented in this paper. Other implementations use $n = 64$ and focus on achieving highest possible throughputs. For the processing core not to starve, the data must be input and output as 64-bit blocks at every clock cycle. For instance, 12 Gbps [23] and even 21.3 Gbps [24] DES-ECB encryptors/decryptors cores have been reported. Regarding 3DES, a 7.36 Gbps ($>21.3/3$ Gbps) implementation is described in [25]. Nevertheless, those high-throughput cores are I/O-intensive (thus limited by the communication rate) and thus are not suitable to be embedded in a resource-limited embedded system, such as a smartcard.

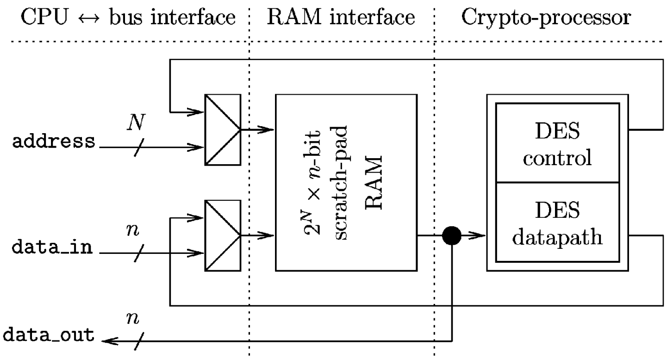

Fig. 1. System-on-chip environment for a VLSI version of the DES co-processor. Typical values for the bus widths are $n = 8$ and $N = 8$.

The knowledge of the DES algorithm internals is not required to explain the rationale of the three $n = 8$-bit architectures discussed in this paper. Only the following facts are indeed relevant for the coming analysis:

- DES is a Feistel cipher, which means that the message is divided into two halves (L and R), among which only L undergoes a logical operation dependent on the some bits of the round key, R being left untouched. Then the two halves are swapped, and the process is iterated sixteen times. After the last round, L and R are not swapped.
- Before any processing, the message bits are shuffled, using a permutation called IP. At the end of the Feistel scheme, the message is de-shuffled by the inverse permutation $\mathrm{FP} \doteq \mathrm{IP}^{-1}$.
- Only 56 bits of the key are used. As justified in the standard [1, p. 1], every byte of the key has a parity bit, chosen so that the Hamming weight of every byte of the key is odd. In a similar way to the message, the key bits are initially shuffled, using the permutation PC_1. The key is modified at each round, by a transformation known as "key schedule", consisting in one or two left shifts, LS (resp. right shifts, RS) for encipherments (resp. decipherments), followed by a permutation called PC_2. Every sub-key is designated by the term "CD" (for cipher/decipher). The shifts are designed in such a way that CD is back to its initial value after a full encipherment (16 rounds). They are implemented by a 2×2 input multiplexor (4 → 1 MUX). However, when enciphering, the initial value to be presented at PC_2 is $\mathrm{LS}(k)$, whereas when deciphering, bare $\mathtt{Id}\,(k) \doteq k$ is to be used instead. Given that a "*general purpose*" DES module is designed to both encipher and decipher, both PC_1 and $\mathrm{LS} \circ \mathrm{PC}_1$ must be computed in parallel.

As a result, a straightforward implementation of DES requires the following sequential resources:

(1) one 64-bit register (named LR in [1, p. 11]) to hold the ciphertext and to store the 16 intermediate messages, and

(2) one 56-bit register (named CD in [1, p. 19]) to hold the key stripped off its parity bits and to store the 16 round sub-keys.

Without any additional registers, the storage of the plaintext in LR and of the key in CD requires a demultiplexing logic, illustrated in Fig. 2. For the sake of clarity, the control part has been omitted in Fig. 2: the multiplexors and the key schedule logic are *implicitly* commanded externally.

The schematics follow those conventions:

- sequential gates, flip-flops (DFFs) in our case, are represented as boxes (■),
- combinatorial gates are represented as boxes with round corners (▢ or ■),
- permutation-only gates, such as IP or buses merge (⋎) or split (⋏), are hollow, whereas
- gates made up of logic have a solid background,
- datapath forks are represented with solder dots (•) and
- when some bits are useless, they are disposed of (⊠).

The entire DES design is made up of bit shuffling dataflow primitives (permutations, multiplexors and flip-flops), with the exception of the round logic. This fact is depicted in Fig. 3, where the critical path of the datapath is highlighted. Notice that the key schedule is not on the critical path: this is made possible by the fact that the 4 → 1 multiplexor that chooses between $\{\mathrm{LS}^{1,2}, \mathrm{RS}^{1,2}\}$ prepares the sub-key for the next round, and not for the current one. The typical resource utilization in the straightforward architecture of Fig. 2 is illustrated in Table 1.

Registers LR and CD must be loaded sequentially. In a pipelined architecture, the use of "enable" signals on the DFFs can usually be avoided. It is possible to use none, if the key is loaded first into CD, because there is a way to keep it "apparently" still. As $\mathrm{LS}^2 = \mathrm{LS} \circ \mathrm{LS}$, $\mathrm{RS}^2 = \mathrm{RS} \circ \mathrm{RS}$ and

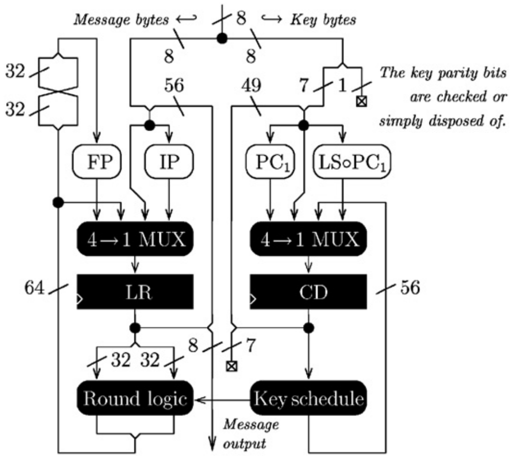

Fig. 2. Straightforward architecture for a DES datapath, equipped with demultiplexing logic to load the message and the key one byte at the time.

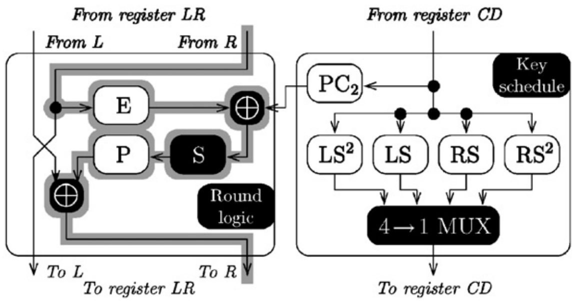

Fig. 3. DES round and key schedule combinatorial logic. The critical path LR → round logic → LR is highlighted.

Table 1
Resources area (μm^2) in Fig. 2, synthesized at 400 MHz in a 130 nm ASIC low-leakage technology

Datapath				Control
Round logic	Rest: dataflow logic			
S + XOR	Permutation	MUX	DFF	FSM
8482	0	7193	3437	5075

$\mathrm{LS} \circ \mathrm{RS} = \mathrm{RS} \circ \mathrm{LS} = \mathtt{Id}$, it is easy to control the key in such a way it is unchanged before and after the LR loading. In the sequel, we assume that the transformation is $\mathrm{LS}^4 \circ \mathrm{RS}^4$. As for LR, it never has to maintain its state more than one clock cycle. The same remark will hold for the refinements carried out on this straightforward architecture, because they are “pipelined”: data (other than the key) flows continuously through the datapath, without having to wait at any time. In addition, the datapath need not be initialized: this yields more compact code (SW) or implementation area (HW).

The straightforward pipeline is thus initially busy during $64/n = 8$ clock cycles to load the key into CD. During another eight clock cycles, the key is applied $\mathrm{LS}^4 \circ \mathrm{RS}^4$, whilst the first message block is loaded into LR. Then the DES engine can start the 16 iterations. The next eight clock cycles are devoted to flushing the result out.

In the straightforward scheme of Fig. 2, every computation has an overhead in execution time due to data loading/unloading in the LR or in the CD register. The evaluation of the architecture throughput does not take into account the key loading, because most applications use only one key, loaded once for many consecutive cipherments (the case of 3DES is detailed later on in Section 3.2). The loading stage consumes $64/n = 8$ cycles, and monopolizes the LR or the CD registers, so that it is impossible to parallelize a loading with a DES cipherment (16 cycles). Then the message must be output, which requires another $64/n = 8$ cycles. Notice that for read and write accesses to be done in parallel, two random access memories (RAMs) must be connected to the DES engine. In terms of memory usage, it is however optimal to use one single RAM, since every computation result can be written over the original message. Thus, the maximum throughput is one encipherment per $8 + 16 + 8$ clock cycles (2.0 bit/clock.)

The straightforward architecture suffers two drawbacks, that impede the cryptoprocessor performances:

(1) The DES cannot perform cipherments whilst new blocks m_{i+1} are read and processed blocks $\mathrm{DES}(m_i)$ are written out.
(2) The LR register is preceded by multiplexors, that increase the critical path.

The next section describes and motivates a novel pipelining scheme, where the data can be both input and output byte by byte, in parallel with DES cipherments.

2.2. DES datapath fast pipelining

The drawbacks put forward in the previous section can be overcome by a more elaborate pipelining scheme of the DES cryptoprocessor. The principle is to parallelize the message inputs and outputs with the DES algorithm. A comparison between the so-called *iterative* and *pipeline* architectures of DES inner-loop is discussed in [22, p. 589]. The difference is that an iterative DES engine processes one cipherment at the time, whereas a pipeline DES engine can process many—up to 16—at the same time. In all the architectures presented in this paper, DES is computed iteratively. However, the outside view of the DES engine is more like a pipeline: data are not input and output monolithically, but rather byte by byte. It must be clear that, throughout this paper, the term “pipeline” refers to the way the data are loaded and unloaded.

A 64-bit register, called IF (because of its role of InterFace between the 8-bit inputs and the 64-bit blocks involved within DES), is added to the DES cryptoprocessor.

IF is designed to have two possible sources: it can input either individual bytes or 64-bit blocks. In the first case, the output of IF is shift by 8 bits to make room for the incoming byte, to be concatenated with the others already collected. The byte that has been “shift-out” is not lost: it is available at the 8-bit output of the pipeline. In the second case, a 64-bit block, such as the result of the DES computation, is latched into IF, in a view to being output byte by byte. In the meantime, the whole content of IF can be transferred to LR, so that the DES datapath is ready to follow up on another cipherment.

In fact, the same IF register can be reused to manage the 8-bit ↔ 64-bit conversion for both LR and CD. Fig. 4 illustrates that the pipeline is generalized to cover both the round logic and the key schedule.

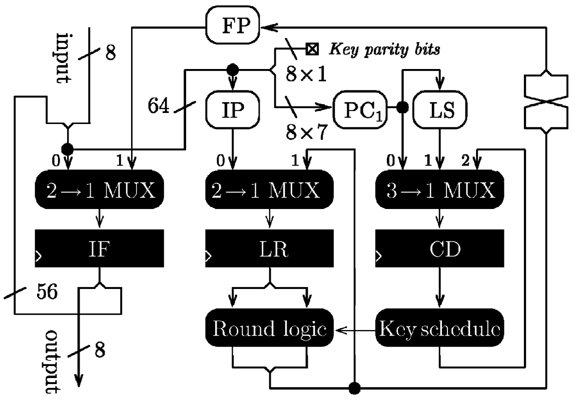

Fig. 4. Proposed pipelined DES 8-bit datapath for ECB cipherments.

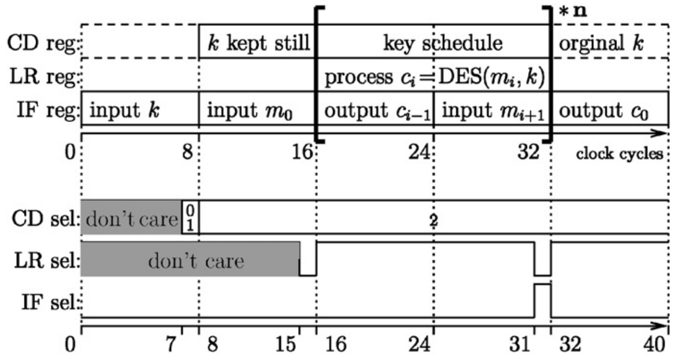

Fig. 5. Pipeline (cf. Fig. 4) steps involved in ECB cipherments $i = 0, 1, \ldots, n-1$. *Upper part*: registers content (c_{-1} = '–' is "don't care" data). *Lower part*: multiplexors selection signals.

A more detailed description of the pipelined process is given below and illustrated in Fig. 5 for DES-ECB encipherment with one key:

1–7: During seven clock periods, the seven first bytes of the key k are loaded, side by side, into IF.

8: The blocks comprised into the last byte of the key $k[56, 63]$, concatenated with the already loaded seven others $k[0, 7] \| \cdots \| k[48, 55]$, is then loaded into CD, using selection 0 (when deciphering) or selection 1 (when enciphering).

9–15: During the seven following clock periods, the message m_0 is built-up into IF.

16: The message m_0, now complete, is transferred into LR. In the meantime, k is kept still in CD, which is possible, as shown in Section 2.1. Incidentally, the result $\mathrm{DES}_k(m_{-1})$ of the previous computation—if any—is latched into IF.

17–24: The next eight cycles are devoted to the output of an hypothetical $c_{-1} \doteq \mathrm{DES}_k(m_{-1})$, byte by byte ($c_{-1}[8 \cdot i, 8 \cdot (i+1)[$, $i \in [0, 8[$), from IF. In the present case, c_{-1} is a "don't care" result. However, starting from clock cycle 33, relevant c_i, $i \geqslant 0$ are delivered byte by byte from IF. Concomitantly, the first eight rounds of DES are executed.

25–31: Whilst DES rounds are computed, a new 64-bit block of data is loaded (as already seen at clock cycles 9–15).

32: DES has finished the 16 rounds. The result is latched into IF. Simultaneously, a new 64-bit block of data is loaded into LR.

33–40: While DES starts the second cipherment, IF outputs c_0. The scheduling scheme goes on, with a periodicity of 16 clock cycles.

In practice, the pipeline is connected to a scratch-pad RAM. The pipeline reads from (cycles 1–8, 9–16, 25–32) and writes to (cycles 17–24, 33–40) the RAM on disjoint time slots. Therefore, a single-port RAM (the less expensive type of RAM) is perfectly suitable. The throughput of the DES pipelined operator is 64-bit per 16 clock periods (4.0 bit/clock). The input and output latencies are equal to eight cycles (as in Section 2.1, we ignore the key initial loading).

By the same token, the pipelined architecture improves the datapath speed. In the straightforward implementation, the LR register has four input sources:

(1) the input byte concatenated with the previous register content shifted by 8 bits to build the plaintext up;
(2) the same block, but passed through IP, to start the computation;
(3) the end of the round data, reinjected into LR for the next round;
(4) the same block, swapped and passed through FP.

As already shown in Fig. 2, a 4 → 1 multiplexor, to choose between those four sources, directly precedes LR.

In the pipelined architecture, IP is performed concomitantly with the collection of the plaintext constitutive bytes. It does not slow down the computation, because in a hardware implementation, IP requires no logic: it is a mere reordering of wires. Consequently, LR has only two possible inputs in the pipelined architecture; the 4 → 1 multiplexor is replaced by a 2 → 1. This optimization is crucial, since this multiplexor is on the critical path (LR → round logic → LR, as highlighted in Fig. 3).

3. Optimal SW/HW partition to realize all DES variants

3.1. IP representation

The notations used in this section are inspired from `openssl` [11] internals:

- `des_encrypt1` is the full DES,
- `des_encrypt2`$\doteq$IP $\circ$ `des_encrypt1` $\circ$ FP is DES, without IP nor FP.

Functions `des_encrypt{ 1,2}` (m, k, enc) take three arguments: a message m, a key k and a Boolean enc, specifying whether to encrypt ($enc = 1$) or decrypt ($enc = 0$).

For any function set $f_i : [0:63] \mapsto [0:63]$, the following property holds:

$$\prod_i (\mathrm{FP} \circ f_i \circ \mathrm{IP}) = \mathrm{FP} \circ \left(\prod_i f_i \right) \circ \mathrm{IP},$$

$$\text{where } \prod_{i=i_{\min}}^{i=i_{\max}} f_i \doteq f_{i_{\max}} \circ \cdots \circ f_{i_{\min}}, \tag{1}$$

because FP $\circ$ IP is the identity function.

This property allows the chaining of DES operations without caring for IP and FP permutations. The "IP representation" computational framework consists in using the `des_encrypt2` primitive instead of `des_encrypt1`, the IP (resp. FP) being called only once at the beginning (resp. at the end) of the computation. Eq. (1) can be applied to the following DES variants:

- $f_i = \texttt{des\_encrypt2}(m_i, k, enc)$ (ECB and ECB^{-1});
- $f_i = \texttt{des\_encrypt2}(m_i, enc?k_i : k_{2-i}, (enc + i)\%2), \quad \forall i \in \{0, 1, 2\}$ (triple-DES on one block m; $m_0 = m$ and $m_{i+1} = f_i(m_i)$, the output being m_3);
- $f_i = \begin{cases} \texttt{des\_encrypt2}(m_i \oplus f_{i-1}, k, 1) & \text{if } enc = 1, \\ \texttt{des\_encrypt2}(m_i, k, 0) \oplus f_{i-1} & \text{if } enc = 0 \end{cases}$ CBC and CBC^{-1}, with $f_{-1} = \mathrm{IV}$);
- $f_i = \texttt{des\_encrypt2}(f_{i-1}, k, 1) \oplus m_i$ (64-CFB and 64-CFB^{-1}, with $f_{-1} = \mathrm{IV}$);
- $f_i = \texttt{des\_encrypt2}(f_{i-1} \oplus m_{i-1}, k, 1) \oplus m_i$ (64-OFB and 64-OFB^{-1}, with $f_{-1} \oplus m_{-1} = \mathrm{IV}$).

In software implementations, IP is not free as in hardware, because bits cannot be arbitrarily moved within or between words. In `openssl`, IP and FP are implemented using 32-bit registers in $5 \times (3\,\texttt{XOR} + 2\,\texttt{SHIFT} + 1\,\texttt{AND}) = 30$ operations.

DES `des_{en,de}crypt3` function performs triple DES on one block of plaintext. It is the only function from `openssl` that takes advantage of the optimization provided by the computation in the IP representation (1). All other functions, especially chained DES, are thus inefficient.

3.2. Multi-mode pipelined DES datapath operating in "IP representation"

The pipeline described in Section 2.2 (see Fig. 4) is not designed to chain cipherments. However, it can be enhanced to cope with triple-DES and all modes of operation. The rationale is to add two inputs to the LR multiplexer:

(1) the result of the previous DES, which allows triple-DES and also OFB (where the series $\{\mathrm{DES}^i(\mathrm{IV})\}_{i \geqslant 0}$ is to be computed);
(2) *idem*, but `XOR`ed with the new message, which allows CBC and CFB chained modes.

The new inputs to LR are compatible, provided that they are in the IP representation. It basically means that inputs to DES must be previously IPed and that output of DES to be recycled must not be FPed. Additionally, the IF register must be able to latch the `XOR` between the new message and the current result, which is required by the stream modes (i.e. CFB and OFB) of DES. Those constraints lead to the versatile version of the pipelined DES datapath represented in Fig. 6. By default, the multiplexor in front of IF (resp. LR) selects the input 0 (resp. 1). At the end of every cipherment (i.e. every 16 clock periods), the multiplexers choose another input, as shown in Table 3.

The realization of triple DES requires a special schedule. The 3DES-ECB is illustrated in Fig. 7. The IF and CD registers sample their default inputs, selection 1 for IF and 0 for CD (corresponding to the ECB and ECB^{-1} lines in Table 3). The scheme for 3DES of Fig. 7 can be combined with the modes of operation. It suffices that the data to be output by IF and sampled into LR have non-default origins documented in Table 3 every 3 × 16 clock periods.

In the case of 3DES with two keys (k_0 and k_1, $k_2 = k_0$), it is noticeable that the computation never stalls. As a matter of fact, the key for the first of the three DES is already

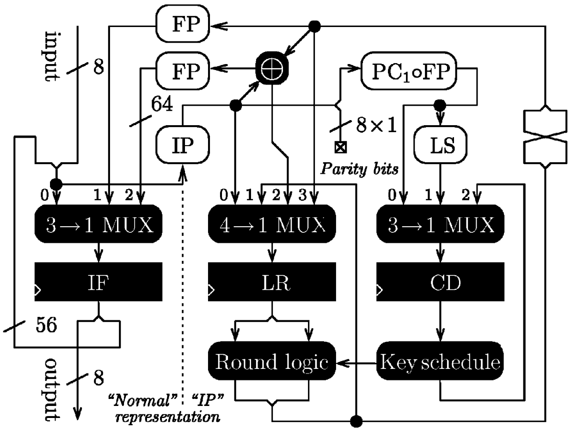

Fig. 6. Proposed multi-modes pipelined DES datapath operating in "IP representation".

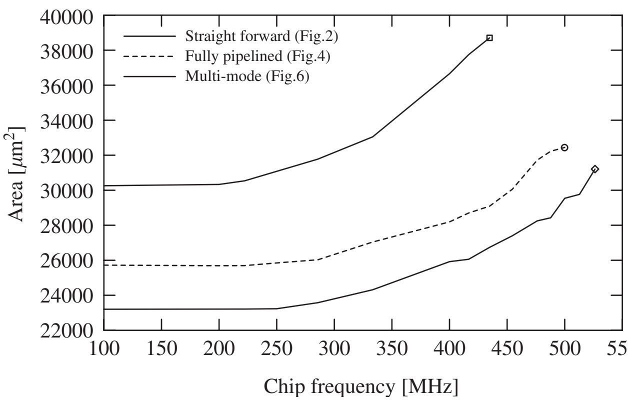

Fig. 7. Register contents when the pipeline is configured for 3DES encipherments with two keys k_0 and k_1, possibly chained $i \in [0 : n[$ times (in which case the indicated clock cycles must be added the offset $i \times 48$).

present in CD, since the last the key was $k_2 = k_0$. Consequently a new message m_i can be loaded instead, and the next computation can follow seamlessly.

Finally, the hardware is also able to realize some non-standard operations, such as "cascade-encryptions" [27, p. 234] (used in `crypt(3)`) or "multiple-encryption modes of operation" [27, p. 237] (e.g. triple-inner-CBC), with the minor limitations explained in the next section.

3.3. SW/HW trade-offs

The proposed pipelined architecture of Fig. 6 is versatile, since all modes of operation can be fit. Nevertheless, this architecture suffers three drawbacks, discussed in the following three sections.

3.3.1. Realization of 3DES with three different keys

In 3DES with three keys, it would be necessary to load the first key k_0 and the new message block m_i at the same time. However, the RAM delivering the data is single-port and there is a single IF register. As there is a contention, the two loadings must be done sequentially. As CD can keep a key globally unchanged during 8 clock cycles, it is loaded first. During the extra eight clock cycles required to load m_i, the pipeline stalls, because it is starving data. Triple-DES with three keys can thus be used with modes of operation, but it is the only exception where the cipherments do not chain gracefully.

3.3.2. Realization of CBC^{-1}

As already indicated in Table 3, CBC cannot be deciphered directly. The reason is that to retrieve plaintext block m_i, the following XOR must be computed: $m_i = \mathrm{DES}^{-1}(c_i) \oplus c_{i-1}$. Unfortunately, the XOR right-hand side c_{i-1} has already been consumed by the pipeline (to compute $\mathrm{DES}^{-1}(c_{i-1})$) when it is needed again. Re-fetching the ciphertext c_{i-1} in memory would require to freeze the pipeline during 8 clock cycles, which is not desirable.

The first workaround is to implement CBC^{-1} by EBC^{-1}, which yields m_0, $m_0 \oplus m_1$, $m_1 \oplus m_2$, etc. instead of m_0, m_1, m_2, etc. The processor can afterwards compute (in software) the XOR between the couples in the memory `ram[0:N[` to retrieve the correct plaintext. An example programme is listed below:

```
register char tmp0, tmp1;
for(register char i = 0; i<8; ++i ){
   tmp0 = ram[i];//The 1st block is only read
   for(register size_t j = 1; j<N; ++j){
         tmp1=ram[j*8+i];//Read jth block
         ram[j*8+i]=tmp0^tmp1;//Write jth block
         tmp0 = tmp1;
   }
}
```

The second workaround we propose is the smartest, because it does not require any post-processing in software. It consists in adapting the control to decipher the blocks c_i in reverse order. If we note $c'_i \doteq c_{N-1-i}$, then $m_i = \mathrm{DES}^{-1}(c'_{i-1}) \oplus c'_i$, for $i \in]N:0]$, is computable by the multi-mode architecture of Fig. 6. It is the same configuration as CFB^{-1} with full feedback, but with the key schedule set to <u>decipher</u>.

3.3.3. Using CBC) and CBC^{-1} *with an IV*

At last, CBC and CBC^{-1} modes cannot be used with an IV. The IV should indeed be loaded, kept in some register (say LR) while the first block m_0 is built-up into IF. The computation could then start with the first operand $\mathrm{IV} \oplus m_0$. However, this scenario also implies that LR has an enable, which we explicitly want to avoid.

A first solution relies on the software. The task simply consists in XORing the first block prior to calling an encipherment or after a decipherment.

A second solution consists in adding an initialization procedure, during which $\mathrm{DES}^{\pm 1}(\mathrm{IV})$ is computed. Then, every message to cipher is simply prepended $\mathrm{DES}^{\pm 1}(\mathrm{IV})$. For long messages, this overhead in processing time becomes negligible.

A third solution implies to increase the DES engine area. The datapath is augmented with an 8-bit XOR operator that would compute "input $\oplus$ output" (with the notations of Fig. 6). This result would be injected into the multiplexor in front of the IF register. It is a design choice to decide whether it is worth implementing this minor hardware feature that complexifies both the datapath and the control (since the IF multiplexor has a new input).

4. Performance evaluation of the proposed architecture

4.1. Implementation in FPGA and ASIC

The three architectures discussed in this paper, namely the "straightforward" (Fig. 2), "pipelined" (Fig. 4) and "multi-mode" (Fig. 6) have been captured using VHDL. They have been synthesized in an FPGA technology (for prototyping) and in an ASIC "low-leakage" 130 nm technology (for production).

The FPGA front-end was Mentor Graphics Precision Synthesis and the back-end Xilinx ISE. The performances are given in Table 2 for the Virtex 4vfx12sf363-12.

In terms of speed, the "straightforward" architecture is the fastest and the "multi-mode" is the slowest.

Table 2
Resources area and maximum frequency of the three proposed architectures implemented in a Xilinx Virtex-4

Architecture	Number of instances	Number of DFFs	Frequency (MHz)
"Straightforward" (Fig. 2)	1445	209	211
"Pipelined" (Fig. 4)	1454	259	202
"Multi-mode" (Fig. 6)	1957	276	144

The ASIC tool-chain for the tape-out of the embedded 8-bit DES blocks was Cadence `pks_shell` for the front-end (synthesis) and `SOC/Encounter` for the back-end (place-and-route). The synthesis results, for both the control and the datapath, are given in Fig. 8. The control is dimensioned to interface with a 256-byte single port RAM. The straightforward architecture is the most compact and the multi-mode is the largest. The design maximum frequencies are 540 MHz (straightforward DES), 500 MHz (pipelined DES), 435 MHz (multi-mode DES). The pipelined DES does not reach the same frequency as the straightforward DES because its more complex control limits its speed. The multi-mode DES datapath is more sophisticated, which explains why it cannot reach frequencies as high as the two other architectures. The maximum frequency of the proposed architectures are fairly high for an embedded system. The architectures can be adapted to an external datapath width of $n = 16$ (resp. $n = 32$) bits, in which case two (resp. four) rounds can be computed within one clock period. This new architecture will run at a maximum speed roughly half (resp. four times less) Table 3.

However, the cipherment throughput is the highest for the pipelined architecture in ECB mode, and for the multi-mode in all the other modes and triple DES. Table 4 shows the throughput of some modes. It should be noted that neither the straightforward nor the pipelined architectures are designed to handle modes of operation or triple-DES. The chaining operation must thus artificially be performed in SW. An estimation of the code for such an operation is given below:

(1) Read `ram[i]` (1 *clock cycle*)
(2) Read `ram[i+8]` (1 *clock cycle*)
(3) Compute `ram[i] XOR ram[i+8]` (1 *clock cycle*)
(4) Write `ram[i+8]` (1 *clock cycle*)

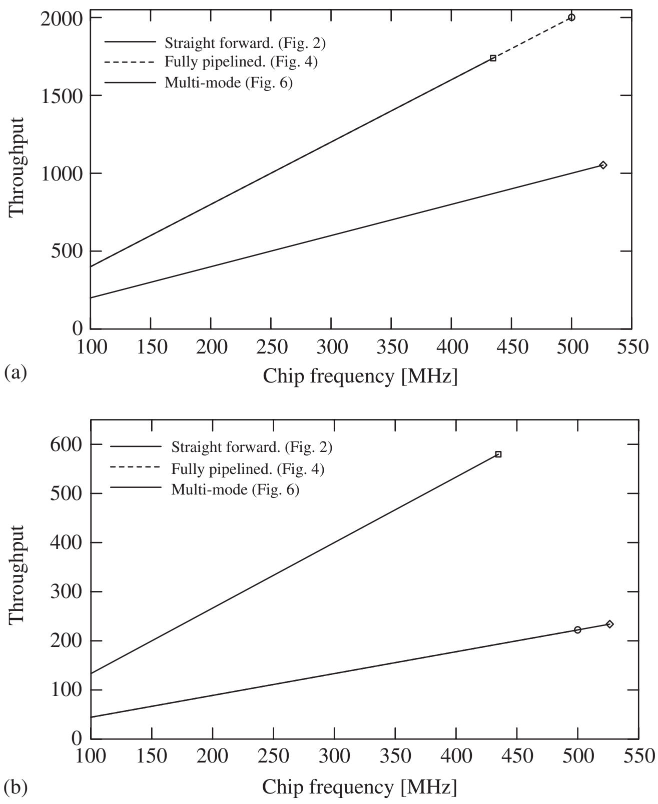

Fig. 8. Synthesis results for the three architectures.

Table 3
Selected signals at the beginning of each DES chained with modes of operation

Mode	IF MUX	LR MUX	Built upon
ECB	1	0	DES
ECB^{-1}	1	0	DES^{-1}
CBC	1	2	DES
CBC^{-1}	–	–	–
CFB	2	2	DES
CFB^{-1}	2	0	DES
$\mathrm{OFB} = \mathrm{OFB}^{-1}$	2	$0, 3, 3, \ldots$	DES

Table 4
Throughput in bit/clock of some modes of the three studied implementations of DES

	Straightforward	Pipelined	Multi-mode
DES-ECB	2.000	4.000	4.000
DES-CBC	1.000	1.000	4.000
3DES-ECB	0.571	0.571	1.333
3DES-CBC	0.444	0.444	1.333

This fragment must be repeated eight times, which leads to a total of $8 \times 4 = 32$ clock cycles. This evaluation is optimistic, because it does not take into account the context switch. It is also unrealistic, since the processor should not be disturbed by the computation internal details. The throughput figures given for straightforward or the pipelined are thus only indicative.

The maximum throughputs are also shown graphically in Fig. 9(a) for DES-ECB and in Fig. 9(b) for 3DES-CBC. It is also interesting to compare the throughputs of an ASIC design with the one of a personal computer (PC). The maximum throughput for 3DES-CBC attained by the multi-mode architecture is 580 Mbit/s, while a 3.2 GHz PC is only able to encrypt at 200 Mbit/s (result of `openssl speed des`).

However, achieving high throughput would be needless if the area overhead is getting too large. For most modes of operation, the parallelization of the cipherments is impossible, due to data dependencies between the consecutive blocks. Still, $\mathrm{ECB}^{\pm 1}$, CBC^{-1} and CFB^{-1} can indeed be parallelized. In those cases, the throughput can be multiplied by the instantiation of multiple engines operating concurrently. Therefore, in Fig. 10, the throughput divided by the area is plot. At constant area, the multi-mode architecture of DES remains the fastest.

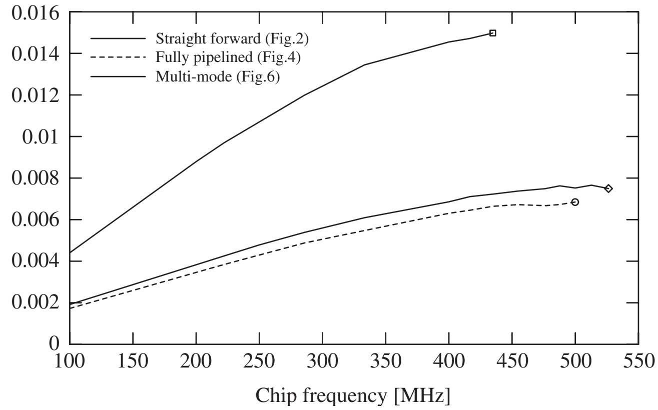

Fig. 9. Throughput (in 10^6 bit/s) of the three solutions in (a) DES-ECB and (b) 3DES-CBC.

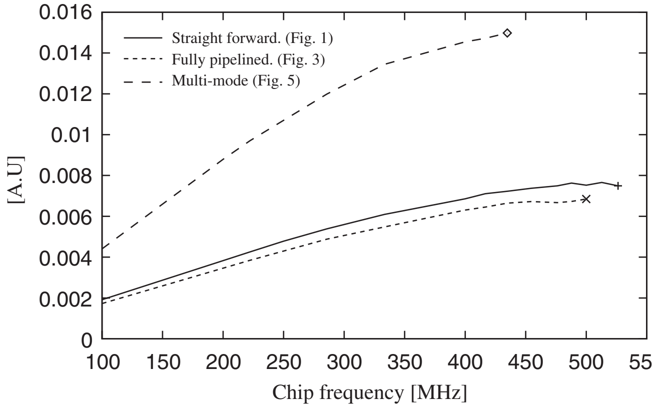

Fig. 10. Comparative efficiency of 3DES-CBC (in Mbit/s/μm^2) for the three proposed architectures.

The DES module after automatic place-and-route by SOC/Encounter is shown in Fig. 11. It happens that the synthesizer was optimistic: static timing analysis performed on the final layout at 95% placement density reports, after post-route resynthesis and in-place optimization, a maximal frequency of 286 MHz (versus 435 MHz predicted by the logical synthesizer). This limitation is in practice not deterrent, since 256 bytes embedded RAMs in 130 nm technology cannot work above 333 MHz without violating either hold or setup times.

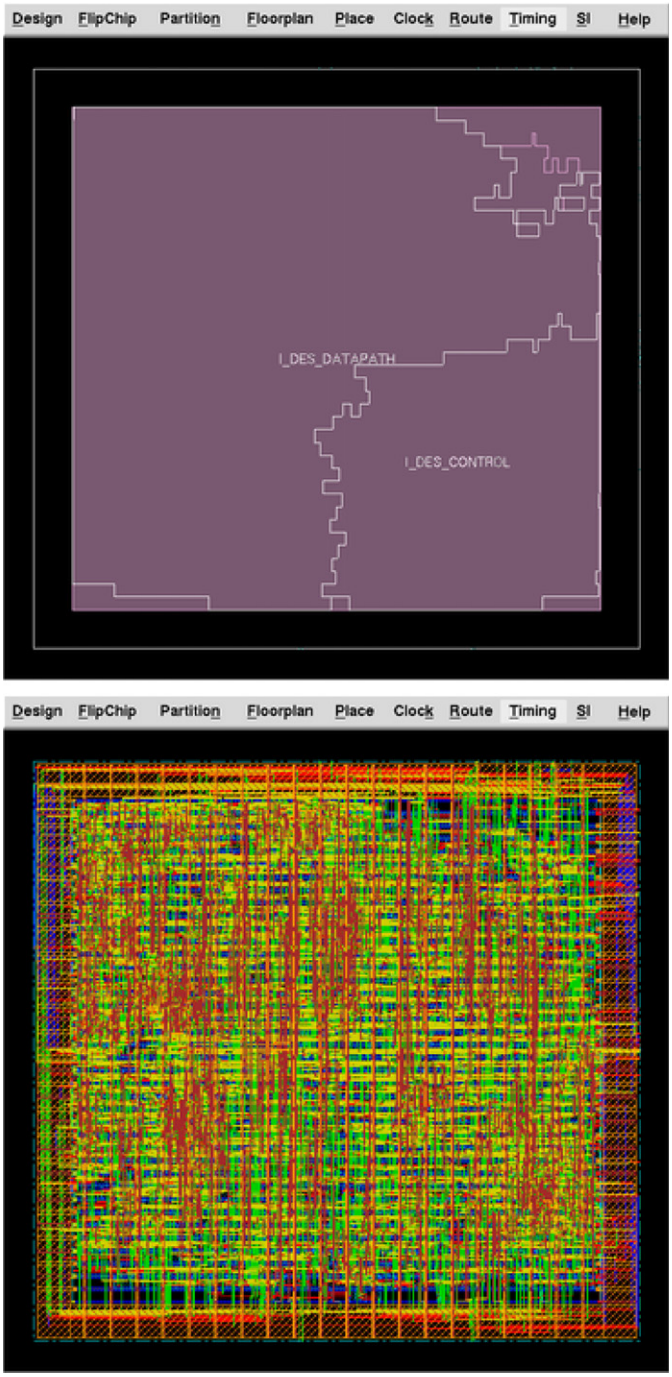

Fig. 11. The multi-mode DES after place-and-route in 130 nm technology. *Top*: datapath/control partitioning; *bottom*: final layout.

4.2. *Comparison with other fast and versatile implementations of DES*

A "Cryptographic Reuse Library" based on static genericity is described in [26]. It contains synthesizable algorithms commonly used in cryptography, each of which can be used either as such, or wrapped into a module that enables modes of operations, or further wrapped into an interface module that adapts throughputs and latencies to match that of the environment. Although the methodology has not been applied to DES in [26], it could be extended to support this algorithm. The features of this "Cryptographic Reuse Library" are those we present in this paper. However, as the mode of operation and interface wrappers involved in the library are not aware of the algorithm internals, the resulting block is necessarily sub-optimal. The approach used in the "multi-mode" architecture (Fig. 6) is to merge the two abovementioned wrappers into the algorithm datapath itself. This allows the "multi-mode" architecture to work without dead cycles at a constant throughput. This prominent feature is a valuable characteristic of the multi-mode architecture: the I/Os are equipartitioned during the processing of the DES algorithm. However, this design solution is specific to DES, and probably does not extend to other algorithms.

Some architectural innovations are described in [24] regarding the round logic of DES. The frontier between the consecutive rounds i and $i+1$ is dissolved in order to balance the critical path between $L_i \rightarrow R_{i+1}$ and $R_i \rightarrow L_{i+1}$. The transformation yields an overall decrease of the critical path length, at the cost of an increase of the latency (the apparent number of rounds rises from 16 up to 21 or 37) and of a particularization of the first and last rounds. These modifications are not a burden when a pipelined implementation is targeted. However, they are deterrent for the architectures presented in this paper, because the data processing is kept iterative.

5. Conclusion

Two architectural innovations, namely the I/O and processing pipelining and the use of "IP representation", allow to improve the design of DES 8-bit implementations. The proposed architecture supports all modes of operation and triple DES with two keys. The VLSI hardware implementation can take advantage of both methods, whereas software implementations can only benefit from the "IP representation". The pipelining strategy consist in parallelizing the data inputs and outputs with the processing. It also enables shorter clock periods, due to the elimination of the some multiplexors on the critical path. The IP representation enables optimized chaining. These optimizations allow to accelerate DES operations in smartcards or in embedded systems or to speed-up DES-cracking machines [22,24].

References

[1] NIST/ITL/CSD, FIPS PUB 46-3: Data Encryption Standard (DES), ⟨http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf⟩, October 1999.

[2] NIST/ITL/CSD, FIPS 81, DES Modes of Operation, ⟨http://www.itl.nist.gov/fipspubs/fip81.htm⟩, December 1980.

[3] NIST/ITL/CSD, FIPS PUB 197: Advanced Encryption Standard (AES), ⟨http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf⟩, November 2001.

[4] NIST/ITL/CSD, FIPS PUB 180-2: Secure Hash Standard (SHA), August 2002.

[5] ISO/IEC International Standard 10116, Information Technology, Modes of Operation for an n-bit Block Cipher Algorithm, 1991.

[6] American National Standards Institute, ANSI X9.17 "Key Management." (see also [A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, October 1996, 816p. [ISBN: 0-8493-8523-7 (Fifth Printing, August 2001)]], p. 173].)

[7] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems, J. Cryptol. 4 (1) (1991) 3–72.

[8] M. Matsui, Linear cryptanalysis method for DES cipher, in: T. Helleseth (Ed.), Proceedings Eurocrypt'93 Lecture Notes in Computer Science, vol. 765, Springer, Berlin, 1994, pp. 386–397.

[9] Electronic Frontier Foundation, Secrets of Encryption Research, Wiretap Politics & Chip Design, 1998, ISBN: 1-56592-520-3.

[10] J.-J. Quisquater, F.-X. Standaert, Exhaustive Key Search of the DES: updates and refinements ⟨http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/⟩, 2005.

[11] Eric Young, <eay@cryptsoft.com>, DES ASM and C implementation in openssl, ⟨http://www.openssl.org/source/⟩, 1995.

[12] NIST/ITL/CSD, modes of operation validation system (MOVS): requirements and procedures, ⟨http://csrc.nist.gov/publications/nistpubs/800-17/800-17.pdf⟩, 1998.

[13] R. Merkle, M. Hellman, On the security of multiple encryption, Commun. ACM 24 (7) (1981) 465–467.

[14] J. Kilian, P. Rogaway, How to protect DES against exhaustive key search (an analysis of DESX), J. Cryptol. 14 (1) (2001) 17–35.

[15] P. Kocher, J. Jaffe, B. Jun, Differential power analysis: leaking secrets, in: Proceedings of CRYPTO'99, Lecture Notes in Computer Science, vol. 1666, Springer, Berlin, 1999, pp. 388–397.

[16] K. Gandolfi, C. Mourtel, F. Olivier, Electromagnetic analysis: concrete results, in: Proceedings of CHES'01, Lecture Notes in Computer Science, vol. 2162, Springer, Berlin, 2001, pp. 251–261.

[17] E. Biham, A. Shamir, Differential Fault Analysis on Secret Key Cryptosystems, vol. 1294, 1997, pp. 513–525.

[18] Helion Technology, datasheet—high performance DES and triple DES core for ASIC, ⟨http://www.heliontech.com/downloads/des_asic_helioncore.pdf⟩, 2003.

[19] Sci-worx, datasheet—DES/triple DES (high performance), ⟨http://www.sci-worx.com/Data_Encryption_Standard_DES.150.0.html⟩, 2001.

[20] F. Bouesse, M. Renaudin, B. Robisson, E. Beigné, P.-Y. Liardet, S. Prevosto, J. Sonzogni, DPA on quasi delay insensitive asynchronous circuits: concrete results, in: XIX Conference on Design of Circuits and Integrated Systems, 2004.

[21] ATMEL, Datasheet—Triple Data Encryption Standard (TDES), ⟨http://www.atmel.com/dyn/resources/prod_documents/6150s.pdf⟩, 2005.

[22] R. Clayton, M. Bond, Experience using a low-cost FPGA design to crack DES keys, in: Cryptographic Hardware and Embedded Systems (CHES'02), Lecture Notes in Computer Science, vol. 2523, 2002, pp. 579–592.

[23] S. Trimberger, R. Pang, A. Singh, A 12 Gbps DES encryptor/decryptor core in an FPGA, in: Proceedings of CHES 2000, Lecture Notes in Computer Science, vol. 1965, August 2000, pp. 156–163.

[24] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, J.-D. Legat, Efficient uses of FPGAs for implementations of DES and its experimental linear cryptanalysis, IEEE Trans. Comput. 52 (4) (2003).

[25] P. Kitsos, S. Goudevenos, O. Koufopavlou, VLSI implementations of the triple-DES block cipher, in: Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS'03), United Arab Emirates, December 2003.

[26] A. Schubert, R. Jährig, W. Anheier, Cryptography reuse library, in: Forum on Design Languages (FDL'99), Lyon, France, August 1999.

[27] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, October 1996, 816p. [ISBN: 0-8493-8523-7 (Fifth Printing, August 2001)].

Sylvain Guilley had been awarded the diploma of the french École Polytechnique (X1997) and of the french École Nationale Supérieure des Télécommunications (ENST 2002). He passed the competitive exams to enter the inter-ministerial body of the telecommunications in 2000. In 2002, he received the M.S. of quantum physics from the Université Pierre et Marie Curie (ENS / Paris 6 University). Since 2002, he is associate professor with the Digital Electronic System group at Télécom Paris. His research interest is the security of security hardware.

Philippe Hoogvorst graduated from École Normale Supérieure, Paris. He got his Ph.D. in 1974 for a study on languages without assignments. He was one of the creators of the "Laboratoire d'Informatique Expérimentale of the École Normale Supérieure". He defended a "thése d'État" in 1983 on the same subject as the Ph.D. He worked for 4 years in the consumer electronic domain by Philips on the development of 16-bit micro-controllers. He was then interested in massively parallel computers. In 1994, he joined the cryptographic team of the École Normale Supérieure, where he worked on the factorisation of integers, ending in the factorisation of a 100 decimal digits integer, which was the biggest result which could be achieved, given the available computing power at the ENS at that time. He moved in 1999 the École Nationale Supérieure des Télécommunications, where he has been working on video compression standards, MPEG-4 and MPEG-7 for 4 years before coming back to a subject linked to security: the defense of cryptographic devices against the so-called "side-channel attacks". Philippe Hoogvorst is currently researcher at the CNRS and detached to the http://www.ltci.enst.fr/ {LTCI/ UMR 5141}.

Renaud Pacalet received his M.S. from the École Nationale Supérieure des Télécommunications in 1988. From 1993 to 1995 he worked on various industrial projects as engineer at Télécom Paris. From 1996 to 2003 he was responsible for the "integrated systems" group at Télécom Paris. From 2003 on, he created and now leads the "Systems on Chip" laboratory of Télécom Paris at Sophia-Antipolis. His research interests include the methods and tools for the specification and the validation of integrated systems. He also works in the field of the assistance for refinement and for the model abstractions of digital integrated systems. This knowledge is applied to the hardware securization of integrated systems: resistance against side-channel attacks, privacy and integrity of memory buses.