Software Optimizations for DES

21

Shay Gueron and Regev Shemy

Abstract

This paper describes some software optimizations for the classical Data Encryption Standard (DES) cipher DES applicable for modern processor architectures that have SIMD instructions. Performance is gained by processing several messages in parallel, compared to processing single messages serially. An added value that the proposed optimizations offer is that the resulting implementations are also side channel protected, unlike other implementations that are found in open source libraries. For comparison, when measured on the latest Intel server processor (Architecture Codename Skylake), our side channel safe implementation is 3.2× faster than that of OpenSSL.

Keywords

Component · DES · DOCSIS · IPSEC · Software optimizations · 3-DES · Side-channel protection

21.1 Introduction

The proliferation of the block cipher AES [1] together with the dedicated processor instruction that speed up AES encryption by more than an order of magnitude, make AES the most ubiquitous cipher in use. Nevertheless, some legacy systems that use outdated ciphers still need support. We are interested here in the classical Data Encryption Standard (DES) [2], and its 3-DES [3] variant, used in either ECB or CBC modes of operation [4], that are still used in some systems. One example is DOCSIS (Data Over Cable Service Interface Specifications) [5].

S. Gueron (✉)
Department of Mathematics, University of Haifa, Haifa, Israel

R. Shemy
Intel Corporation, Israel Development Center, Haifa, Israel

In general, DES is practically replaced by the more modern cipher AES. Thus, software performance of DES/3-DES is an attractive target for optimization only for better support of legacy systems that still use DES. We note that the performance of AES is accelerated by processor instructions AES-NI [6, 7], which are by now ubiquitous. This makes DES is significantly slower than AES. For example, on the latest architecture Intel Xeon Phi Processor 7230, the DES performance, in CBC mode is *42.77* cycles per byte (C/B hereafter). By comparison, on the same platform, AES (in CBC mode) performs at *2.63* C/B. Similarly, in ECB mode, DES performs at 40.9 C/B, and AES performs at *0.63* C/B. These were measured for a message of 8 KB, using the latest version 1.0.2k of OpenSSL [8].

We point out that the block size of DES is *64* bits (whereas AES operates on blocks of *128* bits). As such, the number of blocks that can be encrypted using the same key is limited to at most 2^{32}. Furthermore, Some cryptanalytic results on DES are reported in [9–14].

In this paper, we present a new method for software implementation of DES in ECB and CBC modes, when multiple messages are processed in parallel. We show that our implementation also has the security advantage of being resistant to side channel attacks. Nonetheless, while our solution described on DES, it is relevant for 3-DES, which is more practical and used today in a real system.

21.2 Preliminaries

DES is defined in the specifications [2–4]. We describe it only briefly here, as needed for the rest of the paper.

DES is a block cipher that operates on a block of *64* bit, using a cipher key of *56* bits (technically, the key is embedded in a 64-bit container, where the *8* extra bits are

S. Latifi (ed.), *Information Technology – New Generations*, Advances in Intelligent Systems and Computing 738,
https://doi.org/10.1007/978-3-319-77028-4_21

used as parity bits). The construction is called a Feistel cipher.

Let X be a plaintext block, let Y be the resulting ciphertext, and let K be the key. DES consists of *16* identical rounds (Gj), where, for $j = 1, 2, \ldots, 16$, round j uses round key K_j. Each round key K_j has *48* bits, and is derived from the cipher key K, using some key schedule procedure. The key schedule is independent of the processing of the data, and is not described here. The input X goes through a sequence *19* back-to-back transformations, each one producing a 64-bit output from a 64-bit input. The input for each transformation is the output of the previous one, where X counts as the first input, and Y is the output of the last transformation (IP^{-1} in our case), as follows:

$$Y = IP^{-1} \circ T \circ G_{16} \circ G_{15} \circ \cdots \circ G_1 \circ IP\,(X)$$

The *3* "outer" transformations (IP, IP^{-1},T) do not depend on the key: IP, called "initial permutation", is a fixed (known), IP^{-1} is its inverse, and T is a transformation that switches the position of the left *32* bits of its state, with that of the right *32* bits. To describe the rounds, let L_{j-1} and R_{j-1} denote the left and right halves of the input to round G_j, and let Lj and Rj denote the output of the round. Then,

$$L_j = R_{j-1}, \quad R_j = L_{j-1} \oplus f\left(R_{j-1}, K_j\right)$$

where f denotes the "Core Function" of DES. It operates on two inputs: one (the right half of the state) of *32* bits and the other (K_j) of *48* bits, and outputs *32* bits. It consists of *3* elements: Expansion (E), Substitution (S) and Permutation (P). The Expansion is a fixed function that takes a 32-bit input and expands it to *48* bits. The function computes the 48-bit result of $E(R_{j-1}) \oplus K_j$. These *48* bits are viewed as *8* 6-bit elements, which pass through *8* (different) S-boxes. Each of these S-boxes represents some nonlinear function, which is a lookup table that maps a *6* bits input to a *4* bits output. The details of all the functions and transformations are found in the above specifications. CBC mode processes a plaintext message of m blocks $P_1, P_2, \ldots P_m$ and produces m ciphertext blocks $C_1, C_2, \ldots, Cm$ by $Cj = DES\,(Pj \oplus C_{j-1})$, $j = 1,2, \ldots, m$, where, by definition, $C_0 = IV$.

We point out the fact that side channel attacks that exploit information from memory access patterns. Therefore, if the DES S-boxes are lookup tables that reside in memory, and software accesses directly, the resulting implementation is considered susceptible to side channel attacks. Indeed, this is the case with the OpenSSL implementation. The same is true, of course, for AES, the presence of AES-NI eliminates the need to use lookup tables for AES (Fig. 21.1).

21.3 Multi Block Approach for DES

Processing multiple independent messages in parallel, using modern SIMD architectures, improves the resulting performance significantly. Examples with some cryptographic algorithms are shown in [15–17] for hashing. Using SIMD is not the only way to introduce software pipelining in order to turn latency bounded computations to throughput bounded computations. For example, consider AES-CBC encryption, which is essentially a serial mode. References [6, 7] show how to process multiple messages in parallel with this mode (in the presence of AES-NI), to get the performance of parallelizable modes of operation. Reference [18, 20] shows a technique that gains a *3*× speedup factor for CRC32 computations when a single message is broken (logically) to three chunks, computations are done on these chunks independently so that the processor's pipeline is filled up, and

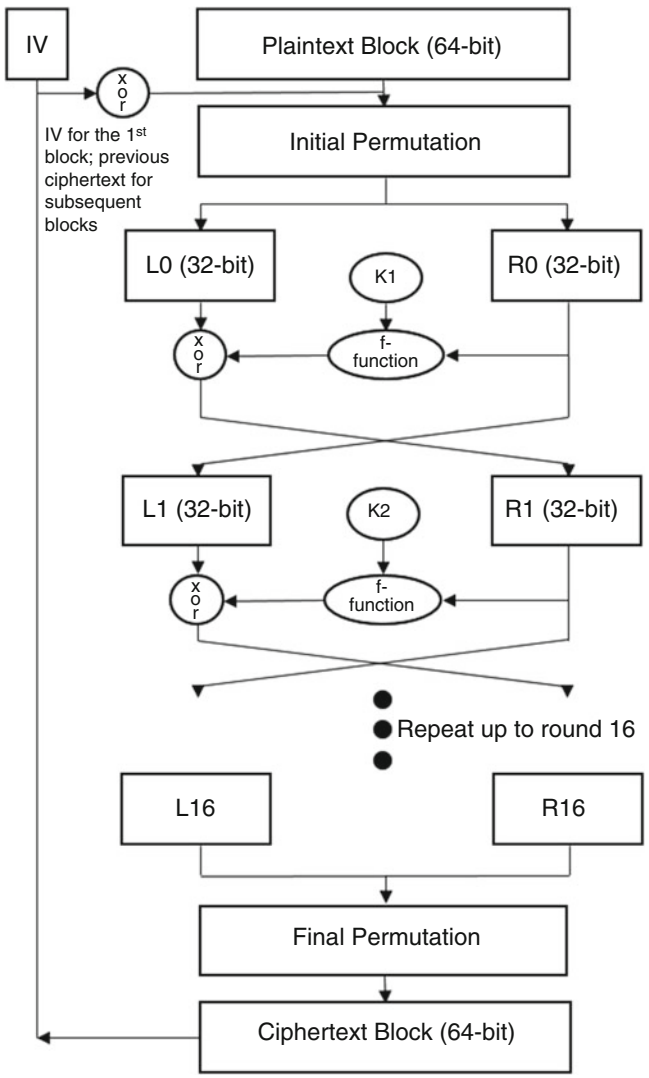

Fig. 21.1 Outline: DES in CBC mode

in the end, the three results are combined to a single CRC32 value by using some mathematical transformation.

We show here how to apply this approach, which we call "Multi Block DES", for DES (and 3-DES) cipher. Our method processes several messages in parallel, at the cost of some added overhead of "transposing" the inputs when they are consumed from memory, in order to make them ready for processing with SIMD instructions. Then, the SIMD capabilities of modern processors can be leveraged. The overall result is significant performance gains.

Transposing: To illustrate the transposing overhead, consider a single message M, pointed to by the pointer p. To encrypt M in CBC mode, it is possible to consume consecutive blocks (of *64* bits) and process them. Now, consider two message M1, M2, with two pointers p1, p2. Suppose that we wish to encrypt both messages in parallel, using a SIMD architecture with 128-bit registers (xmm's). This requires placing two 64-bit blocks B1, B2, of M1, M2, respectively, in a single xmm register, say xmm2. B1 and B2 are read (loaded) from different pointers, into registers, say xmm0, xmm1, populating the low halves of these registers. Subsequently, we need to appropriately shuffle the two 64-bit contents of xmm0, xmm1 into the single xmm2. The following is a code sequence is an example:

```
vmovdqu (%rsi), %xmm0
vmovdqu (%rdi), %xmm1
vpshufd $0x4e, %xmm1, %xmm1
vpblendd $0x0c, %xmm0, %xmm1, %xmm2
```

Of course, a more efficient implementation would read two blocks from each pointer, into xmm1, xmm2, and shuffle the contents of these registers into two xmm registers with a similar software flow. The following is a code sequence is an example.

```
vmovdqu (%rsi), %xmm0
vmovdqu (%rdi), %xmm1
vpunpcklqdq %xmm0, %xmm1, %xmm2
vpunpckhqdq %xmm0,%xmm1, %xmm3
```

In the context of DES, the algorithm actually processes two halves of the block independently. Denote B1 = [b1, a1] and B2 = [b2, a2], where a1, a2, b1, b2 are the 32-bit halves. An efficient implementation would place these halves in two xmm registers, xmm0, xmm1, as

xmm0 = [0, 0, b2, b1], xmm1 = [0, 0, a2, a1]

Obviously, it is more efficient to read in *4* blocks and deposit populate the respective halves in *2* xmm registers, in a way that populates them entirely (Fig. 21.2).

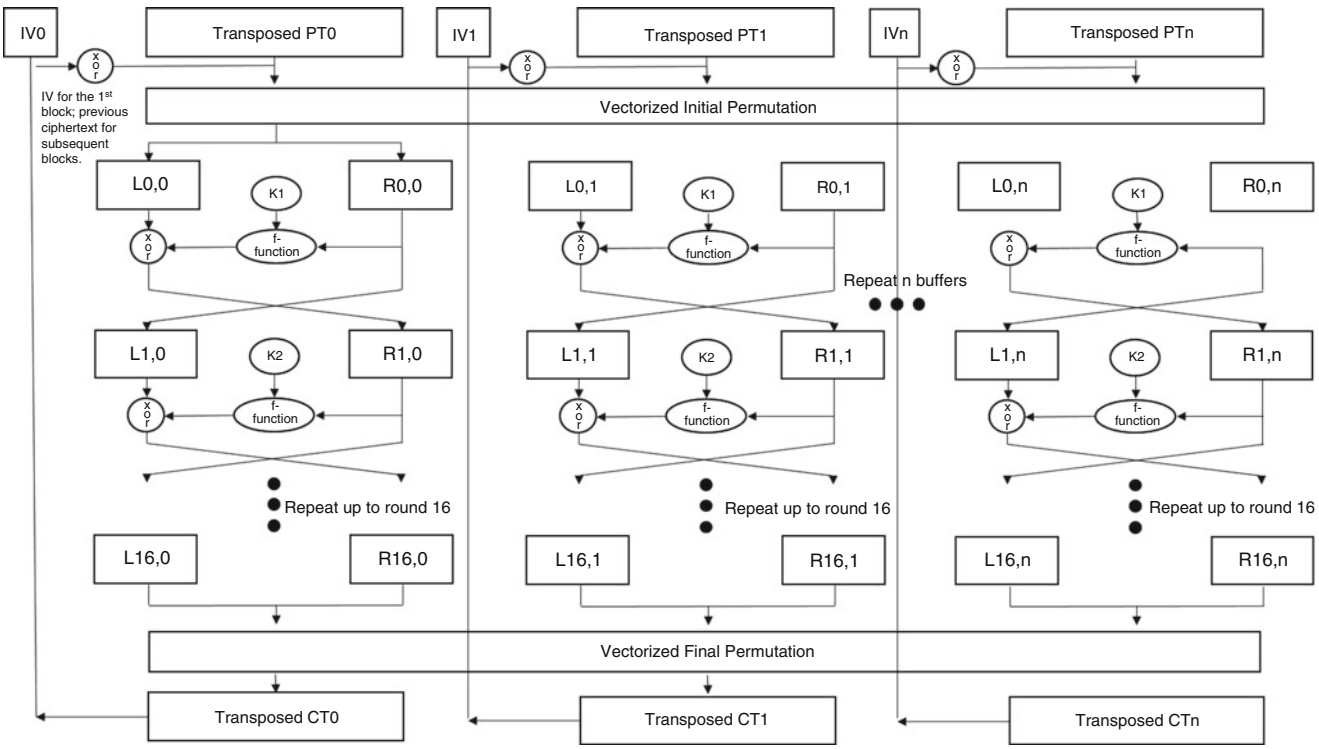

Fig. 21.2 Outline: DES in CBC mode, with a Multi Block implementation

```
vmovdqu (%rsi), %xmm0
vmovdqu (%rdi), %xmm1
vpunpckldq %xmm0, %xmm1, %xmm2
vpunpckhdq %xmm0, %xmm1, %xmm3
```

SIMD processing: Having the input organized in halves blocks on several registers, allows the encryption process to be done in parallel using SIMD instructions. Implementing the Initial Permutation and Final Permutation on the registers is easily done using shift, logical AND and XOR instructions done using SIMD on double word chunks (32- bit, half block data). The "f" core function implementation consists of three main phases. The first phase, E-phase is done using similar SIMD instructions. Both S-phase and P-phase, requires constants to be tailored to their use model. S-phase includes substation of each 6-bit of input to 4-bit according to the 6-bit element of the block. In order to prevent each element of each block from accessing S-box table serially, we load part of the S-box table to a register, permuting several elements of several different blocks in parallel. Result register of this permutation, is eventually blended in using mask registers, deciding whether the permutation was done with the correct part of the S-box table. Iterating through all part of the S-box table, resulting in a final result registers, with all results blended between all result registers, covering all S-box table constants. This technique, available using Intel AVX512 Extension. Using AVX512, we are able to process up to 8 halves blocks in a single register, storing each 6-bit element, in a single bytes, eventually permuting 8 different elements of 8 different halves blocks. Using AVX512 registers, this permutation can load big portions of the S-box table, and within 16 iterations of permutations, whole S-box table is fully loaded the needed permutations covering all options of input result with output register of the S-phase. This process can be parallelized with more registers doing same permutations, over the loaded S-box part each time. This process done with preparing the S-box table of constants to fit for the SIMD instructions used accordingly. After S-phase is done, using of P-box prepared constants in advanced and additional SIMD instructions implementing P-phase in parallel to all halves blocks.

This approach eliminates the needed memory accesses depending on input, removing all branches or cache accesses that relies on any secret information, supplying resilience for software side channel attacks. In comparison to different Multi Blocks approaches implemented on other schemes, this solution eliminate the memory access dependency as part of the encryption algorithm itself, using the wide registers to gain both performance and security features. This is enabled due to the total small size of the S-box compared to the registers size.

Remark 1: The technique detailed here is using Intel latest AVX512 Extension, found on architecture codename Skylake Server. This technique could be downgraded to fit for AVX2 extension, first introduced on Intel architecture codename Haswell. Using AVX2, will result in less parallel streams processed, bigger number of iterations done on processing S-phase of the "f" core function and additional costs. These additional costs withdraw the use of DES CBC mode of operation using Multi Block approach from performance view. Additional security benefit is till gained and describes later on this paper.

Remark 2: The use of AVX512, as similar to remark 1, is first introduced on Intel server processors (Intel Architecture Codename Skylake). Additional future AVX512 instructions, announced to public, including AVX512 VBMI instructions. These future instructions will give extra performance boost to the technique used in S-phase using AVX512 instructions, allowing byte permutations over AVX512 register (512 bit).

Remark 3: The method described in this section us referred to DES CBC mode of operation using Multi Block approach. Similar implementation, excluding the transpose costs, could be implemented on DES ECB mode of operation for single input data stream. DES ECB mode of operation will require additional instructions splitting the blocks in halves and uniting them in the end, though, this implementation will result in better performance boost than CBC Multi Block, due to the redundancy of the transpose costs compared to added instructions.

21.4 Results

This section reports the performance results of our study. For this study, we wrote new optimized code the algorithms discussed above. The measure workload was the encryption of 8 KB inputs, and we compare the performance to that of the legacy code as appears in OpenSSL (version 1.0.2k). We report results for AVX2 and for AVX512 based implementations. The experiments were carried out on an Intel server processer, Intel Xeon Processor E3-1230-v5 (Architecture codename Skylake), that supports all the architectural extensions mentioned above.

The Results are shown in Fig. 21.3. A slowdown factor of $0.79\times$ is achieved by using the AVX2 solution. With AVX512, additional ISA and wider registers result in positive speedup factor, reaching $3.2\times$. The comparison is made against the legacy code, used in the recent OpenSSL version (1.0.2k). All experiments were conducted using Intel Xeon PHI Processor 7230, configured with static frequency of 1.3 GHz. We note that our code executes in constant time: it

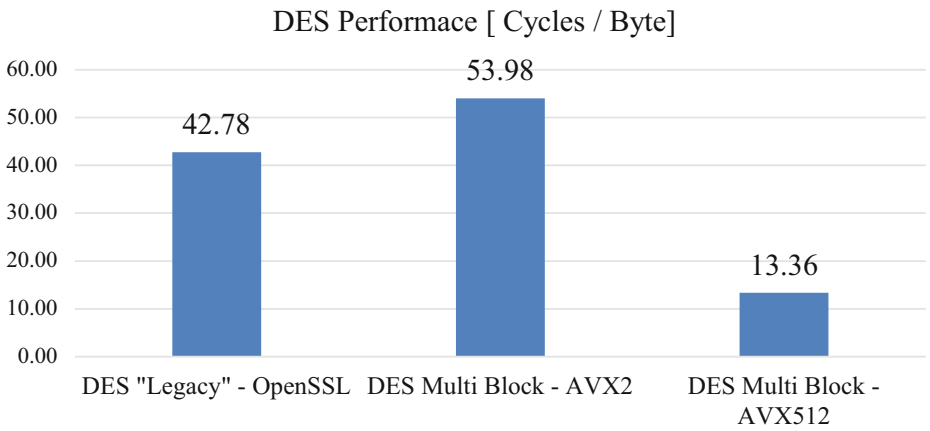

Fig. 21.3 Optimized DES implementations through Multi Block processing. Measurements taken on input messages of size *8192* Bytes (total). The results of code that leverages AVX2 and AVX512 are compared to the results of the code implemented on OpenSSL (1.0.2k)

	OpenSSL DES	AVX2 Multi Block DES	AVX512 Multi Block DES
Speedup	1x	0.79x	3.2x

does not use lookup tables for computing the results of the S-boxes. It is therefore side channel protected. Additional DES Multi Block AVX2 solution can be implemented, gaining extra performance speedup but lacks side channel protection and therefore is not discussed on this paper.

21.5 Conclusions

The solution described here, provides a new implementation for the legacy cipher DES. It offers a side channel protected implementation which is *3.2*× faster than the (unprotected) implementation of OpenSSL. Our code was published and is now part of the Intel IPSEC MB public code library [19].

The new approach can be adapted and adjusted to fit for wider area of new solution and defense mechanisms for protecting versus software side channel attacks, eventually providing a new methodology of using the updated hardware and architectures used in the market.

Acknowledgements This research was supported by the Israel Science Foundation (grant No. 1018/16), by the BIU Center for Research in Applied Cryptography and Cyber Security, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by the Center for Cyber Law & Policy at the University of Haifa.

References

1. National Institue of Standards and Technology, Advance Encryption Standard (AES), FIPS Publication 197, November, (2001), https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf
2. National Burea of Standards, Data Encryption Standard, U.S, Department of Commerece, FIPS pub. 47, January, (1977)
3. National Institue of Standards and Technology, Data Encryption Standard (DES), FIPS Publication 46–3, October, (1999)
4. National Institue of Standards and Technology, DES Modes of Operation, FIPS publications 81, December, (1980), https://csrc.nist.gov/csrc/media/publications/fips/81/archive/1980-12-02/documents/fips81.pdf
5. DOCSIS 3.1 Specfication, September, (2017), https://apps.cablelabs.com/specification/CM-SP-MULPIv3.1
6. S. Gueron, Intel Advanced Encryption Standard (AES) instructions set (Rev. 3), Intel Software Network, (2010), http://software.intel.com/en-us/articles/advanced-encryption-standard-aes-instructions-set/
7. S. Gueron, Intel's New AES Instructions for Enhanced Performance and Security. Fast Software Encryptiom, 16th International Workshop (FSE 2009). Lecture Notes in Computer Science: 5656, (2009), pp. 51–66
8. OpenSSL: The Open Source toolkit for SSL/TLS, project webpage, http://www.openssl.org
9. E. Biham, A. Biryukov, An improvement of Davies' attack on DES. J. Cryptol. **10**(3), 195–206 (1997)
10. E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
11. E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard* (Springer, Berlin, 1993). ISBN 0-387-97930-1, ISBN 3-540-97930-1
12. J. Kelsey, B. Schneier, D. Wagner, C. Hall, Side channel cryptanalysis of product ciphers, in *Proc. European Symp. Research in Computer Security (ESORICS '98)*, (1998), pp. 97–110
13. P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems, in *Adavance in Cryptology–CRYPTO '96 Proceedings*, (Springer, 1996), pp. 104–113
14. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, M. Schimmler, *How to Break DES for Euro 8,980, 2nd Workshop on Special-purpose Hardware for attacking Cryptographic Systems—SHARCS 2006*, Cologne, Germany, April, 2006
15. S. Gueron, V. Krasnov, Simultaneous hashing of multiple messages. J. Inf. Secur. **3**, 319–325 (2012)
16. S. Gueron, R. Shemy, [OpenSSL Patch]: Accelerating Multi (MB) CBC SHA256 on architectures that support AVX512 instructions set, January, (2016), http://openssl.6102.n7.nabble.com/openssl-org-4221-PATCH-Accelerating-Multi-Block-MB-CBC-SHA256-on-architectures-that-support-AVX512-it-td62058.html

17. S. Gueron, R. Shemy, [OpenSSL Patch]: Multi Block (MB) SHA 512 for x86_64 Architectures that support AVX2/ AVX512 instructions set, February, (2016), http://openssl.6102.n7. nabble.com/openssl-org-4307-PATCH-Multi-Block-MB-SHA512- for-x86-64-Architectures-that-support-AVX2-AVX512-instrt-td63716.html
18. S. Gueron, Speeding up crc32c computations with intel crc32 instructions. Inf. Process. Lett. **112**(5), 179–185 (2012)
19. Intel IPSEC MB Library, https://github.com/01org/intel-ipsec-mb
20. S. Gueron, V. Krasnov, Fast implementation of AES-CRT mode for AVX capable x86-64 processors, March, (2013), http://rt.openssl.org/Ticket/Display.html?id=3021&user=guest&pass=guest