UNIVERSITÄT KARLSRUHE

FAKULTÄT FÜR INFORMATIK

Postfach 6980, D 7500 Karlsruhe

More Efficient Software Implementations of (Generalized) DES

Andreas Pfitzmann, Ralf Aßmann

Institut für Rechnerentwurf und Fehlertoleranz
Fakultät für Informatik
Universität Karlsruhe

Interner Bericht 18/90
May 1990

More Efficient Software Implementations of (Generalized) DES

Andreas Pfitzmann, Ralf Aßmann
Institut für Rechnerentwurf und Fehlertoleranz, Fakultät für Informatik, Univ. Karlsruhe,
Postfach 6980, D-7500 Karlsruhe 1, Federal Republic of Germany

Interner Bericht 18/90
May 1990

Abstract

By preserving the macro structure of the Data Encryption Standard (DES), but by allowing the user to choose

1. 16•48 independent key bits instead of generating them all using only 56 key bits,
2. arbitrary substitutions $S_1, ..., S_8$ and
3. arbitrary permutations IP and P, and
4. an arbitrary expanding permutation E,

we obtain a very general and presumably much stronger cipher called *generalized DES*, or *G-DES* for short. A cipher having the first three extensions is called *G-DES with non-arbitrary E*.

We choose, in an unorthodox way, from some well known equivalent representations of G-DES and some well suited table combinations and implementations. Concatenations of substitutions and permutations are precomputed and tabulated. Since direct tabulation of e.g. a permutation of 32 bits requires 2^{32} entries of 4 bytes each, which clearly exceeds the main memories of today, the big table is split into smaller ones that permute disjoint and compact parts of the input bits at the appropriate positions. To compute an entry in the big table, the corresponding entries in the smaller tables are ORed.

For some specific expanding permutations (including the original E in DES), the expense of this permutation can be reduced drastically: Only copy, rotate, and AND with a mask of a register is necessary, if the bits in the register and the tables of the substitutions are ordered appropriately. Since this is the only way to achieve better performance for DES than for G-DES we know, it does not seem to make sense to implement anything more narrow in software than *G-DES with non-arbitrary E*.

This is an extended and revised version of a paper presented at SECURICOM 90.

Using these techniques, we get by far the fastest software implementations of DES (more specifically: G-DES with non-arbitrary E) and G-DES known to us. The same DES and G-DES implementations, partly in MC68000 assembler language, achieve

87 and 68 kbit/s on an Apple Macintosh Plus (MC68000, 7.83 MHz),
104 and 81 kbit/s on an Apple Macintosh SE (MC68000, 7.83 MHz),
208 and 162 kbit/s on an Apple Macintosh Portable (MC68000, 15.67 MHz),
445 and 327 kbit/s on an Apple Macintosh II (MC68020, 15.67 MHz),
487 and 366 kbit/s on an Apple Macintosh IIx, IIcx, or SE/30 (MC68030, 15.67 MHz),
722 and 546 kbit/s on an Apple Macintosh IIci (MC68030, 25 MHz, no cache board),
829 and 597 kbit/s on an Apple Macintosh IIci (MC68030, 25 MHz, with 64 Kbyte cache board)

in ECB and CBC mode using less than 110 and 430 Kbytes main memory. OFB(64) mode is between 19 and 28% faster because we can save IP, except for the first time. Conversely, if CBC mode is used for authentication, we can save IP^{-1} except for the last block. This increases speed by between 11 and 12%.

To avoid unnecessary IP and IP^{-1} executions and to enable multiple encryption in all modes of operation, our implementation supports multiple encryption, which may be necessary if (standard) DES is used. To give an example: triple encryption with G-DES in ECB and CBC mode is performed with 28 kbit/s, 32 kbit/s, 64 kbit/s, 140 kbit/s, 154 kbit/s, 230 kbit/s, and 254 kbit/s, respectively, for the 7 Macintosh configurations mentioned above.

Our table implementation makes it possible to save key EORing (Exclusive OR = addition mod 2) in each round by precomputing a key-specific table for every (or in some cases only every second) round. This costs 1 Mbyte main memory and increases cipher speed by 9% on the Macintosh Plus and by 7% on the Macintosh II, respectively. For MC680xy machines without a data cache or memory management unit, or if their capacity sufficiently exceeds 1 Mbyte, this is the next step in a speed vs. main memory trade-off.

Memory requirements can be reduced by not copying bits which are input to one combined S-box. For G-DES with non-arbitrary E, this reduces the use of main memory to less than 300 Kbytes and increases cipher speed by 16% on the Macintosh Plus, SE or Portable and by 20% both on the Macintosh II and IIx, IIcx or SE/30 and by 17% on the Macintosh IIci without cache board.

Table of Contents

Table of Tables

Table of Figures

1 Introduction and overview

Cryptography is a necessary tool for the privacy and authenticity of data transmitted over networks or stored in computers, cf. e.g. [DaPr_89]. Since networks, and especially radio networks, grow and link more and more computers, the need for secure, compatible, and efficiently implemented cryptographic tools becomes more and more urgent.

The contribution of this paper to the solution of these problems is twofold:

- §2 argues for the generalization of the Data Encryption Standard (DES) to obtain a more secure but nevertheless compatible cryptosystem, called *generalized DES* (G-DES).
- §§3-7 and §9 show how to implement G-DES very efficiently in software. §8 describes how an 18 to 26% increase in throughput can be gained by restricting G-DES slightly (no longer are all possible values of the expansion permutatation E allowed, but only a class of values including the original value in DES).

Since our implementation of G-DES with non-arbitrary E is – as far as we know – the fastest of all software implementations of DES (and, as its name suggests, comprises DES) on personal computers, it might be interesting even to those who think that DES is appropriately secure for their purposes. Additionally, the description of our implementation in §§4-8 is the only public, comprehensive and comparing description of efficient DES software implementations we know of. §3 reminds us what is already known, in §4 we justify the selection of implementation languages and target machines, §5 describes the often published and used, but inefficient straightforward implementation, §6 describes the efficient table look-up implementation, §7 explains how key-specific tables can be used for a further speed-up, and §8 describes how to make use of a special property of the expanding permutation E. §9 compares the performance of our implementations with the good ones known to us, which results in a rather complete justification – as far as performance of software implementations is concerned – of our generalization of DES in §2.

In the following, we assume that the reader is familiar with DES and its operating modes as described in the literature, e.g. [DaPr_89, Denn_82, MeMa_82].

2 Motivation

First we explain why and how DES should be generalized. Thereafter, we remark on how the generalized DES should and should not be used. After discussing the possible expense of implementing generalized DES in hardware, we give our reasons why the generalization should be implemented primarily in software.

2.1 Why generalized DES ?

For more than a decade, the security of DES has been questioned for two reasons [Denn_82, DaPr_89]:

- the short key of 56 bits;
- the secrecy of the design criteria for the specific values of the S-boxes and, to a lesser degree, the permutations.

Since for political reasons [Kola_85, Ath1_86, Horg_86, WaPP_87, CACM6_87, Prei_88], no official successor is in sight [Pric_88], we propose a series of successors based on the following three conservative principles:

a) Each successor should be upward compatible with the previous ones.
b) Each successor should be efficiently implementable both in software and hardware.
c) The security of each successor should be based on the wealth of cryptologic experience with DES.

Since the strategies for extending the useful lifetime of DES recommended in [Carr_87] do not basically change the key length of 56 bits, we consider them to be insufficient. In our view, four successors of DES are evident:

1. Instead of using 56 key bits to generate 16 subkeys of 48 bits each as in DES (see §3 Fig. 1: Standard Model), adopt the proposal of IBM [Mass_88 p. 541], Hellman [Yasa_76], Morris [Morr_78], Berson [Ber1_83], and Luby and Rackoff [LuRa_88] to allow 16•48 independently generated key bits.
2. Additionally, allow the user to specify arbitrary substitutions $S_1, ..., S_8$.
3. Additionally, allow the user to specify arbitrary permutations IP and P.
4. Additionally, allow the user to specify an arbitrary expanding permutation E.

All these extensions have been part of folklore for many years and therefore are not new in themselves, but proposing them as a series of successors and as standards is – as far as we know. While 1. is called modified DES (MDES) in [LuRa_88], we call a cipher having all four extensions *generalized DES* (G-DES).

Some remarks concerning the length of G-DES keys are in order:

Even if straightforward and redundant coding is used, a G-DES key comprises at most 16•48 (key bits) + $8 \cdot 2^6 \cdot 4$ ($S_1, ..., S_8$) + $48 \cdot \lceil \mathrm{ld}\ 32 \rceil$ (E) + $32 \cdot \lceil \mathrm{ld}\ 32 \rceil$ (P) + $64 \cdot \lceil \mathrm{ld}\ 64 \rceil$ (IP) = 768 + 2048 + 240 + 160 + 384 = 3600 bit. Since the first use of DES in 1977, storage and transmission capacity have been increased by far more than 2 magnitudes, thus in 1990 the relative expense of a 3600 bit key is less than that of a 56 bit key in 1977. And storage and transmission capacity will further increase after 1990.

Since 1977, cheap, powerful and convenient key transport devices have been produced, e.g. so called super smart card, i.e. chipcards with keyboard and display.

In most situations, the values of the substitutions and permutations are clear from the context, i.e. normally the key length is 768 bit. These fit well into a single block of an asymmetric cryptosystem (several of which have been published and used for key-distribution since 1977, e.g. RSA) if its blocklength is chosen with appropriate security margins (a number of 106 digits has been factored recently and 10 additional digits represent an increase in computing power of almost a factor of 10 [LeMa1_89]).

If 768 key bits are too many, it is straightforward to agree on a key scheduling scheme to expand a shorter key to 768 bit, as noted even in 1982 [Hell_82].

2.2 How to use generalized DES and how not

While it is generally assumed that G-DES is much more secure than DES if used as a symmetric cryptosystem, this is not the case if G-DES is used to construct a *one-way function* in the simple way it is widely done using DES [LuR1_88]: The argument of the one-way function is taken as DES key, which is used to encrypt a fixed and publicly known plaintext. The resulting ciphertext is the image of the one-way function.

If G-DES is used instead of DES, it is very easy to adjust the last 2•48 key bits (out of the 16•48 of G-DES) so that the fixed and publicly known plaintext encrypts under the key in the given block of ciphertext. There are three obvious possibilities to overcome this "limitation": The trivial possibility is to

- use the key scheduling of DES for this occasion (i.e. use 56 key bits to generate 16 subkeys of 48 bits each).

The two nontrivial possibilities, which maintain the keylength of 16•48 bit, but deserve further evaluation, are either

- to apply G-DES not only once, but a constant number of times (e.g. 16) to the plaintext or

- to use e.g. 16 different fixed and publicly known blocks of plaintext and 16 given blocks of ciphertext.

The first possibility can be broken by effort in the order of 2^{56}, i.e. search through the keyspace of DES, which works deterministically. The second possibility can be broken by effort in the order of 2^{64}, i.e. search through the keyspace of G-DES till the ciphertext is obtained by chance, which works only probabilistically. The third possibility hopefully requires effort in the order of 2^{768}, i.e. hopefully cannot be broken in the next centuries. (After having written this paper, we obtained [LuRa_89], where the third possibility is described on page 157. Merkle describes in [Merk1_89] another method of using DES as one-way hash function: He uses 120 bits as input – 64 plaintext and 56 key bits. But applying this method to G-DES instead of DES yields a one-way function which is as insecure as the one described above.)

For G-DES, all *symmetric modes of operation* for DES may be used and should be supported by an implementation.

For privacy (without authentication), e.g.
ECB mode (Electronic CodeBook),
CBC mode (Cipher Block Chaining),
CFB mode (Cipher FeedBack, parameters: number of used and fed back bits, ≤64), or
OFB mode (Output FeedBack, parameter: number of used bits, ≤64, feedback = 64)
may be used as described in [DaPr_89 pp. 79-98, Denn_82, MeMa_82].

For authentication (without privacy), e.g.
CBCauth mode (Cipher Block Chaining for authentication), or
CFBauth mode (Cipher FeedBack for authentication),
may be used as described in [DaPr_89 pp. 121-122].

For authentication and privacy within one encryption pass, e.g.
PCBC mode (Plain Cipher Block Chaining := EOR current plaintext and modulo 2^{64} sum of previous plaintext and ciphertext before encrypting the result)
may be used as described in [MeMa_82 pp. 69-71]. (We use "EOR" as an abbreviation of "Exclusive OR" which is the same as "bitwise addition mod 2".)

Please note that a feedback of less than 64 bits in OFB mode is not advisable [DaPa_83] and therefore not allowed in ISO 8372. Hence, if implementations allow a feedback of less than 64 bits in OFB mode, for a feedback of 64 bits they should at least aim to optimize by saving IP with the exception of the first time.

Conversely, if CBCauth mode is used, we can save IP^{-1} except for the last block.

2.3 What is the expense of implementing generalized DES ?

While hardware implementations with G-DES extension 1 will usually be neither more complex nor slower than those of (standard) DES, this will probably not be the case for extensions 2, 3 and 4: Arbitrary substitutions require e.g. RAMs instead of ROMs and the access time of RAMs is usually a little bit slower. Arbitrary permutations require significantly more wiring, which increases the required chip area and therefore decreases the possible clock rate.

In the following, we will concentrate on showing that the first three extensions can be implemented in software without degrading speed.

2.4 Why software implementations of generalized DES ?

As mentioned, a software implementation offers almost unlimited flexibility.

For most applications, an efficient software implementation will be fast enough – as we shall see. With the increasing computing power of laptops, personal computers and workstations, this class of applications is growing continually.

For some potential users of cryptography, it may be impossible for monetary or political reasons to obtain suitable cryptographic hardware. Especially the former will be true as long as G-DES is *not* a de facto standard.

3 What can be learned about software implementations of DES from the literature ?

In dozens of papers and books, you can read something like

> *It is straightforward to implement EORing (exclusive OR = addition mod 2) and the S-boxes efficiently, since EOR instructions on bitstrings are available on most processors (or can be emulated by AND, OR, NOT instructions without too much expense) and how to implement an (S-box) table look-up is well known. But implementing the permutations is cumbersome and results in bad performance.*

In many of them, you can find implementation guidelines or program listings which substantiate this (these inappropriate implementation techniques are described in §5).

Due to linearity one of the two permutations in each of the 16 rounds of DES can be saved (more precisely: brought next to another permutation, whose concatenation can be precomputed and therefore is a single permutation at runtime) as explained in [DDFG_84 pp. 181-185]. The original description of (G-)DES and two possibilities to save a permutation, the 32-bit and the 48-bit model, are shown in Fig. 1.

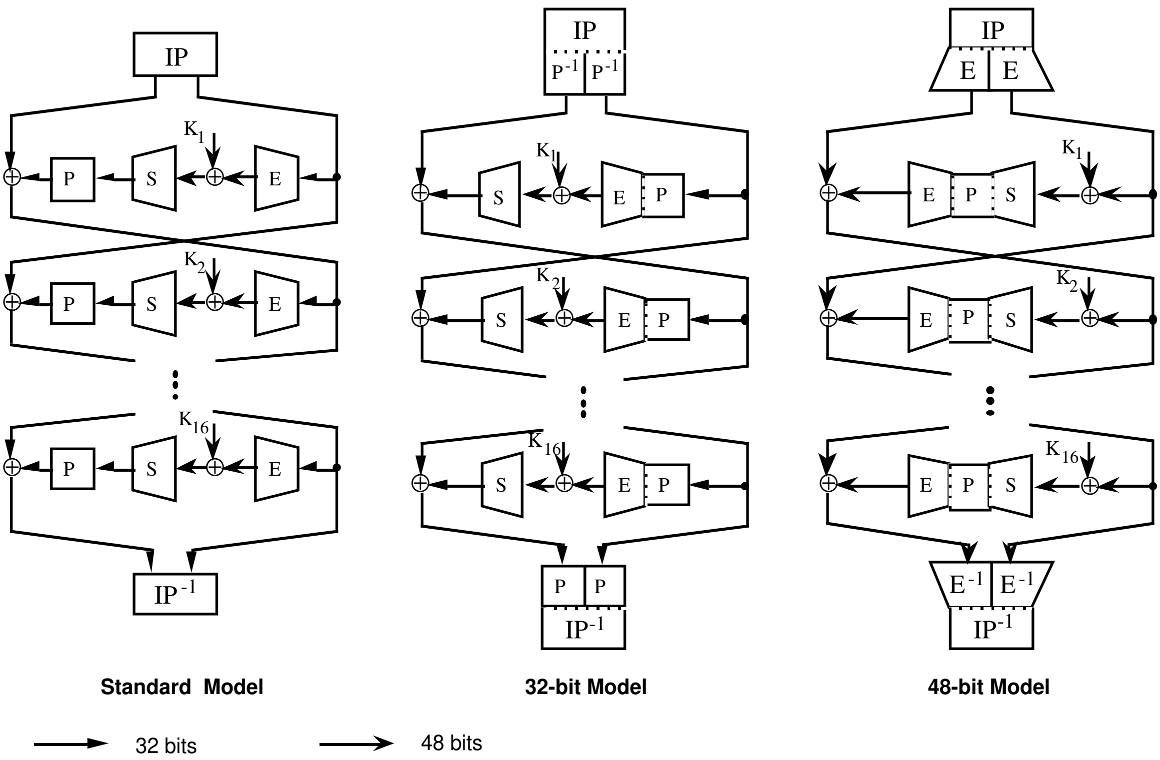


Fig. 1: Three models of G-DES

Only few references to programs which achieve reasonable performance (and therefore must efficiently implement the permutations) could be found: [WiHi_80, DDGH_85, Bras_88 p. 17]. But nothing is said there how it is done (we strongly suspect as explained in §6). After we had written this paper, we received another reference to two programs [Seel_89 p. 701]. Both seem to use the techniques explained in §6.

4 Choosing implementation languages and target machines

In the following subsections, we briefly describe a portable reference version written in standard PASCAL, explain why we decided to use assembler language for all time critical parts in most other versions, give our reasons for using the MC68000 assembler language, and finally remark on the portability of these program versions.

4.1 A (standard) PASCAL reference version

For conformance testing and low performance applications, we have implemented a highly portable version of the 32-bit model in (standard) PASCAL. The bitstrings are represented as boolean arrays and the permutations are implemented as bitwise assignment statements. For an n bit permutation, n assignment statements are used. By using another PASCAL program, which on input of the substitutions and permutations, outputs a PASCAL program with the corresponding assignments, the user has, at the expense of compilation, a complete G-DES implementation [Aßma_89]. Independent of whether the user chooses his "personal" substitutions or permutations, this program achieves *5 kbit/s* on the Apple Macintosh II, using 24 Kbytes main memory.

Here, and where nothing else is stated explicitly, throughput is given for ECB mode and data that is in the main memory. The reason for the latter is that the performance of the G-DES program, and not that of the hard disk or floppy disk, combined with the operating system, is to be measured. CBC and PCBC mode have roughly the same throughput as ECB. In most cases, the throughput on the Macintosh Plus and the Macintosh IIx, the exact throughput of the other modes of operation, as well as the throughput for double and triple encryption, can be found in §9.1 Table 1. There, the interested reader will find a short characterization of the machines as well.

This and all our other implementations described in §6 and §8.1 support multiple encryption to avoid unnecessary IP and IP^{-1} executions and to enable multiple encryption in all modes of operation. Multiple encryption may be necessary if (standard) DES is used.

4.2 Why parts in assembler language ?

Since the G-DES definition and structure is bit-oriented, and portable high level programming languages usually hide all bit-oriented instructions of the target machines (e.g. PASCAL "abstracts away" EOR instructions on bitstrings, cf. §3), and since we are interested in what can really be achieved by a software implementation, we decided not to limit ourselves: For performance reasons, we implement all time critical parts of our G-DES program in assembler. All other parts are written in PASCAL, as stated in §4.1.

4.3 Why on MC680xy processors ?

After the decision to use assembler language, we had to decide which processor or family of processors to choose as target machines. We chose the family of MC680xy processors for two reasons:

- The technical reason is that the almost orthogonal instruction set of the MC68000 is a good basis to derive *efficient* implementations for other processors, even those that have no orthogonal instruction set. The reverse would not be as easy.
- Another reason is that personal computers and workstations based on MC680xy processors are cheap, widely available and easy to use.

Both reasons recommend against the use of special instructions of the MC68020 or MC68030, i.e. those instructions, the MC68000 does not offer. At first sight, this seems to be a severe limitation, which contradicts our aim to find out what can really be achieved by a software implementation. But at a closer look, there is no place where these special instructions would significantly help.

4.4 What about portability ?

Since less than 10% of the code of our application programs is time critical, coding only these 10% in MC68000 assembler language (which is an orthogonal one) and having them and the rest written in (standard) PASCAL achieves a reasonable degree of portability of all these versions described in §§5-9.2. Of course the PASCAL version described in §4.1 is very easily portable but only suitable for low performance applications.

In the meantime, a portation to intel 8086, suitable for e.g. IBM PC, XT, AT and PS/2 has been finished by another group of Karlsruhe University. It will be described briefly and compared in §9.3

5 "Test single bits" implementations of the permutations

A "test single bits" implementation of a permutation tests each input bit separately and sets the corresponding output bit accordingly. For efficiency, all output bits are first set to the same value by none or at most a few instructions. Thereafter it is only necessary to set bits if their value is not the preset one.

Using a "naive" implementation, i.e. a short assembler coded loop that interprets tables representing IP, E, P and IP^{-1}, more than 7 instructions are needed to permute a bit. The resulting throughput of this "naive" implementation, which uses 7 Kbytes of main memory, is *14.5 kbit/s* on the Macintosh II.

If the loop is made over the input bits (and not, as the form of the tables defining DES suggests, over the output bits), the following optimization can be used: If the permutation is an expanding one, all "duplicates" of each input bit are set together which saves bit tests. In the evaluation of permuted choices (e.g. E^{-1} in Fig. 1, 48-bit model), input bits without depending output bits can be skipped.

By additionally

- implementing the 8 S-boxes as four groups of two S-boxes using 16 Kbytes instead of 256 bytes [DDGH_85 p. 146] and by
- using the 32-bit model of DES (Fig. 1) [DDFG_84 p. 183] to save the permutation P and by
- implementing the permutations $P^{-1}\circ IP$, $E\circ P$, and $IP^{-1}\circ P$ using direct operands, which, on average, yields an expense of 2.5 instructions per permuted bit,

the throughput is *68 kbit/s* on the Macintosh II. 29 Kbyte main memory are used.

As in the PASCAL version described in §4.1, by using a PASCAL program which on input of the substitutions and permutations, outputs assembler source code with the corresponding S-boxes and direct operands, the user has, at the expense of compilation, a complete G-DES implementation [Aßma_89].

Since we did not find further improvements conforming to the strategy "test single bits", we changed to "table look-up".

6 "Table look-up" implementations of the permutations

The key idea to achieve much more efficient software implementations of G-DES is to tabulate the permutations. In §6.1, we describe how this can actually be done, i.e. within the limits of given main memories. In §6.2 and §6.3, we describe in depth various implementations of the 32-bit and 48-bit models, respectively.

6.1 How to tabulate permutations and substitutions followed by permutations within the limits of given main memories

Direct tabulation of e.g. an expanding permutation of 32 to 48 bits requires 2^{32} entries of 48 bits each, which clearly exceeds the main memories available on personal computers today. Therefore, the big table is split into smaller ones in the following way: All smaller tables have entries of 48 bits each and permute disjoint and compact parts of the input bits at the appropriate position(s). Those bits that are 1 in the relative address of an entry are set at the appropriate position(s) in the entry, whereas all other bits are 0. To compute an entry in the big table, the corresponding entries in the smaller tables are ORed, see Fig. 2.

Since there is at most one 1 at any bit position in all entries in the small tables (implementing an entry in the big table), it is the same whether the entries in the smaller tables are ORed or EORed (exclusive OR = addition mod 2). This is appropriate if the EOR instruction is faster than the OR instruction.

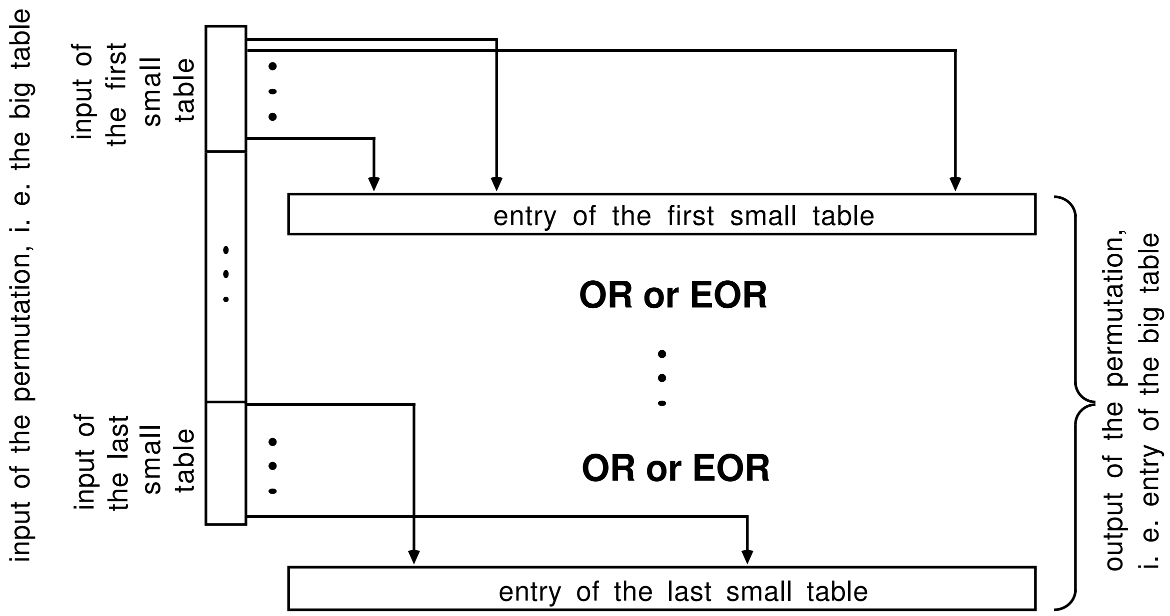


Fig. 2: How to implement big permutations by small tables

Permutations can be computed using the following steps:

1. Split the input of the permutation into smaller parts (they need not be of the same length).
2. Repeat steps 3 and 4 for all parts of the input.
3. Use the part of the input to compute – if necessary – an index to the corresponding small table.

4. Fetch the indexed entry from the small table and combine it with the intermediate result by ORing or EORing.

If the AND instruction is faster than both the OR and the EOR instruction, it may be appropriate to invert all bits in the entries in the smaller tables and AND the entries.

To keep the presentation as simple as possible, neither possibility is further discussed in the rest of the paper.

It is worth noting that this implementation technique is not only capable of efficiently handling permutations. Without additional expense, it can handle a concatenation of substitutions and permutations, iff the input is split in step 1 into smaller parts, without splitting the input of substitutions.

Additionally, it is very practical that changing the values of S-boxes or permutations does not require re-compilation, but only the change of tables, which is much faster and more user-friendly.

6.2 32-bit model

By

- implementing the 8 S-boxes as four groups of two S-boxes [DDGH_85 p. 146] and by
- using the 32-bit model of DES (Fig. 1) to save the separate evaluation of the permutation P and by
- using 20 small tables with 2^8 entries each for $P^{-1} \circ IP$, $E \circ P$, and $IP^{-1} \circ P$ together,

the throughput is *235 kbit/s* on the Macintosh II.

Each entry in $E \circ P$ comprises 64 bits, because the 4•12 = 48 bits are distributed over 4•16 bits in such a way that the 4 S-boxes with 2^{12} entries each can be adressed directly with 16 bits without bit shifts. The entries in $P^{-1} \circ IP$ and $IP^{-1} \circ P$ are of course both 64 bits as well. To avoid byte shifts, two groups of S-boxes have entries of 8 bits, the other two have entries of 16 bits where the low order byte is garbage (and therefore does not occupy additional main memory because it can be used for an entry in the 8-bit tables). The 4 tables are used as follows: First an entry of 16 bits is MOVEd into the lower half of the corresponding 32-bit register, thereafter the garbage is overwritten by moving an entry of 8 bits into the low order byte of the same register. After SWAPing the two halves of the register, the procedure is repeated to assemble the common result of the four groups of two S-boxes. All tables together demand 56 Kbytes of main memory.

This implementation of the 32-bit model requires not only expensive bit manipulation (i.e. shifts) to address the table representing $P^{-1} \circ IP$, but also to address the tables representing $E \circ P$ and $IP^{-1} \circ P$. While the first is unavoidable because it seems to be unreasonable to expect external data to be delivered in a *noncompact form*, the second can be avoided by expanding the internal data format from 32 to 64 bits. The disadvantage is that the entries in the tables $P^{-1} \circ IP$ and S are twice as long, and that twice as many instructions are needed to OR the entries in $P^{-1} \circ IP$ and an additional SWAP to assemble the results of the four tables representing S. This possibility motivates the avoidance of internal bit manipulations, i.e. shifts, by noncompact internal data representation.

Experiments demonstrate that a compromise between both pure solutions achieves even better performance: the entries in $P^{-1} \circ IP$ are not expanded, but the representation of the two halves of 32 bits each is expanded immediately thereafter. This version achieves a throughput of *245 kbit/s* on the Macintosh II. The program uses tables adding up to 72 Kbytes of main memory and occupies itself ≈10 Kbytes.

By implementing $E \circ P$ by 2 tables each with 2^{16} entries of 64 bits, the throughput increases to *315 kbit/s* on the Macintosh II, but all tables together now demand 1088 Kbytes of main memory.

6.3 48-bit model

Since in 4•16 = 64 bits we can represent 4•12 = 48 bits as well as 4•8 = 32 bits, and since the 48-bit model (Fig. 1) allows us to tabulate E∘P∘S in one big table (broken down into smaller ones as explained in §6.1), we decided to implement the noncompact form idea in the 48-bit model.

Using 8 tables with 2^8 entries of 128 bits each for E∘IP, 4 tables with 2^{12} entries of 64 bits each for E∘P∘S, and 8 tables with 2^{12} entries of 64 bits each for IP^{-1}∘E^{-1}, the program achieves *327 kbit/s* on a Macintosh II. All tables together demand 416 Kbytes of main memory. The program itself occupies ≈10 Kbytes.

Since
- the performance of this 48-bit model implementation is even better than that of the 1088 Kbytes version of the 32-bit model, and
- using larger tables for P^{-1}∘IP or E∘IP, respectively, helps the 48-bit model implementations more than any 32-bit model implementation,

the hint "The 48 bit model ... is very useful on computers with words of 48 bits." [DDGH_85 p.146] can be extended to at least some 32-bit machines (and even pseudo 32-bit machines such as the Macintosh Plus, using a MC68000 processor with an external data bus of 16 bits only, cf. Table 1).

Please note that all machines we experimented with have enough registers to hold all intermediary results of the G-DES algorithm until they are no longer needed. Whether the 48-bit model is useful for 32-bit machines that have significantly less than 8 data registers (as the MC680xy has) is another question.

In OFB mode, the initial permutation (i.e. P^{-1}∘IP or E∘IP respectively) can be saved with the exception of the first time, and the relative expense of IP is significantly higher in the 48-bit model implementations than in the 32-bit model implementations. The same is true in CBCauth mode with respect to IP^{-1}. Therefore, the performance advantage is even understated by the values of ECB mode given above, cf. §9.1 Table 1.

6.4 How to further increase speed ?

In all versions of §6, speed can be increased by about *2%* by freezing the instruction cache of the MC68020 or MC68030 after the inner loop has been completed for the first time. The inner loop comprises 2 G-DES rounds, since 2 rounds fit well even into the 256 byte instruction caches of the MC68020 and MC68030, but if it should comprise only one round, it would include 2 additional MOVEs, which would be unsuitable on all MC680xy.

Conversely, by including 4, 8, or even 16 (the latter means unfolding the loop) G-DES rounds, performance on the MC68000, which has no instruction cache, would increase a little bit. But since the instruction caches of the MC68020 and MC68030 only comprise 256 bytes, which is not even enough for the initial permutation and the inner loop comprising 2 G-DES rounds, to say nothing about the inverse initial permutation, this would cause a noticeable decrease on these processors.

So to include 2 rounds is the best compromise today. Since the MC68040 will have an instruction cache of 4 Kbyte, it may be appropriate to revise our decision at a later date.

On some processors, reordering the sequence of instructions (of course within the limits of data dependency) changes the runtime significantly. But nothing is said about this in the manuals of the MC680xy, and on our Macintoshs, corresponding experiments did not show any significant changes.

A natural question to ask is how further performance enhancements can be achieved by using more main memory.

It is straightforward to use bigger tables to implement P^{-1}∘IP or E∘IP of the 32-bit or 48-bit model, respectively. Using 4 tables with 2^{16} entries of 64 or 128 bits respectively, is not possible on today's personal computers, but on those of tomorrow. Using e.g. 5 tables, 4 with 2^{13}, 1 with 2^{12} entries (totalling to 576 Kbyte and resulting in 960 Kbytes for all tables) does not use too much memory even today, but requires more complex bit extractions since the extracted quantities do not fit byte boundaries. These bit extractions must either be programmed explicitly using MC68000 instructions, or bit field moves only available on the MC68020 and MC68030 must be used. The first severely limits the improvement in performance (*338 kbit/s* are obtained on a Macintosh II), the second does not adhere to the arguments of §4.3. In any case, the gain is limited because it does not concern the inner loop of G-DES, and in OFB(64) there is no asymptotic gain at all.

To use 3 tables to implement E∘P∘S (2 with 2^{18} entries and 1 with 2^{12} entries of 64 bits each) also requires too much memory on today's personal computers. But it will become affordable in the near future and concerns the inner loop of G-DES. The possible improvement in performance is at most 25%, and much less on a MC680xy processor, because addresses cannot be handled as conveniently without using additional registers.

A further and rather unconventional possibility with the same possible improvement in performance is discussed in the next section.

7 Precomputing key-specific tables

To save the EORing of a subkey in each round, we consider four possibilities of combining the tables introduced in §6 and the subkeys. The first two possibilities combine the tables of a permutation, e.g. E, and EORing of subkeys which follows immediately after. The third possibility combines the tables of the substitutions and the preceding EORing of subkeys. The fourth possibility (§7.5) combines both approaches. The first and fourth possibility might be applied in practice to implement G-DES.

7.1 EORing the subkeys to one of the smaller tables

The EORing of the key to the result of the permutation E in each round can be saved by precomputing key-specific tables: Instead of ORing the entries in the smaller tables implementing a bigger one, the entries can be EORed, since there is at most one 1 at any bit position of all entries of the small tables which implement an entry of the big table. If this is done at en- and decryption time, the key can be EORed to one of the smaller tables beforehand, see Fig. 3. Of course, this small table must now be specific for each round.

Since the MC680xy has no EOR instruction allowing a register and a main memory location as operands and the register as the target, we have to simulate this instruction by a MOVE followed by an EOR.

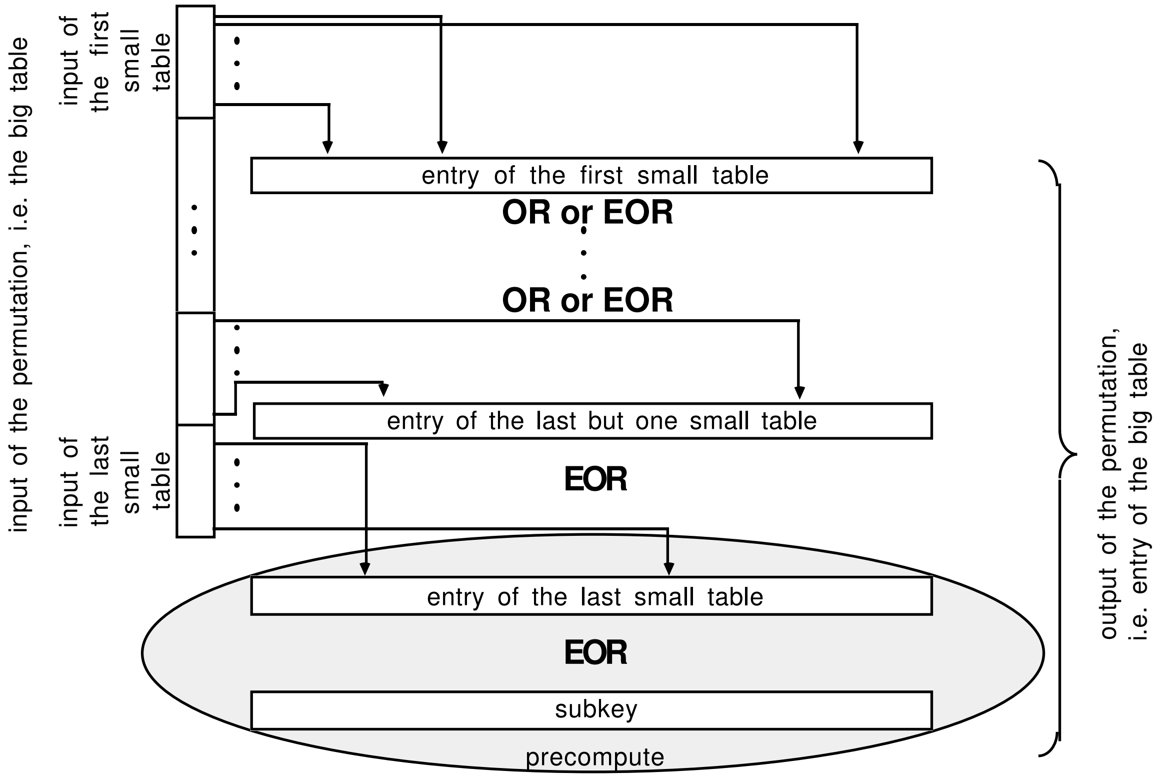


Fig. 3: Using key-specific tables: the big table represents keyEORing∘E

7.2 EORing to the appropriate bits of all smaller tables

It is possible to trade ORing of all table entries for more expensive key-specific table precomputation: instead of EORing the key to the entries in a single small table, each key bit is EORed to the entries of the corresponding small table (Fig. 4). The price is that now *all* small tables are specific to a G-DES round.

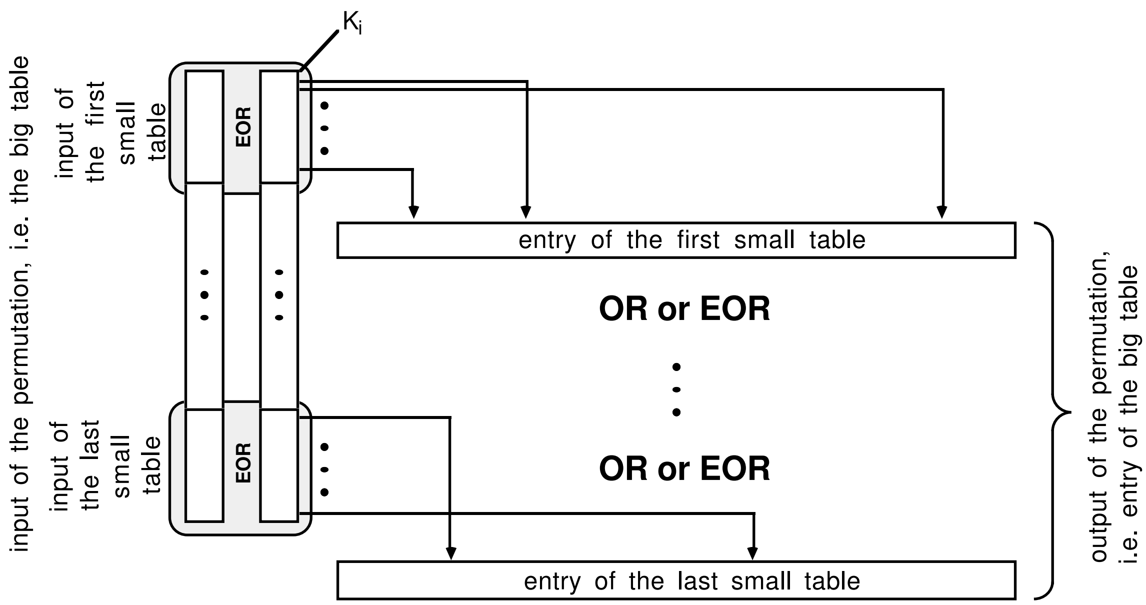
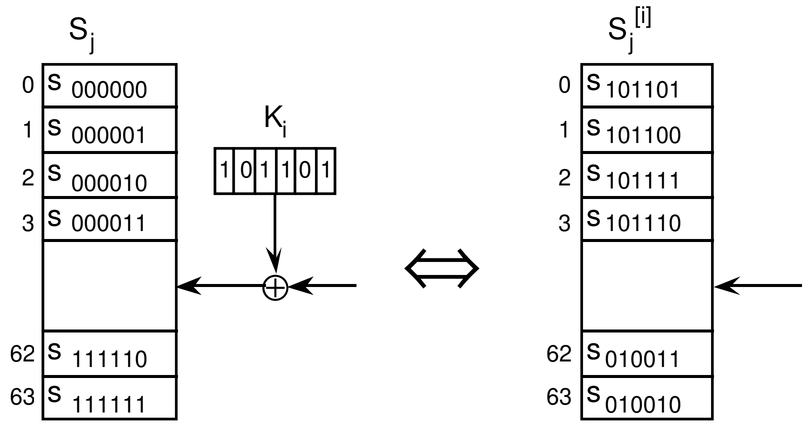


Fig. 4: EORing key-bits to the entries in the appropriate small tables: the big table represents keyEORing∘E

7.3 Reordering the S-box entries with respect to the subkeys

Precomputing cannot only combine the tables of a permutation and the subkeys, it can also combine substitutions and the subkeys. This is done by reordering the entries of the tables representing the substitutions, as shown in Fig. 5.



$S_j^{[i]}$ means that the entries in the S-box S_j are reordered with respect to the subkey K_i.
This is done by interchanging s_m with $s_{m \oplus K_i}$.

Fig. 5: Reordering of the S-boxes

7.4 Moving the subkeys to make use of §7.1 and §7.2 directly

On the one hand, the possibilities of precomputing key-specific tables described in §7.1 and §7.2 can be applied directly in the standard and the 32-bit model of G-DES (cf. Fig. 1). But this is uninteresting, since either the attainable throughput is significantly lower than that of the 48-bit model without key-specific tables, or, if the last version of §6.2 (i.e. 2 "small" tables for E∘P) is taken as basis, the necessary tables are tremendously greater for approximately the same throughput.

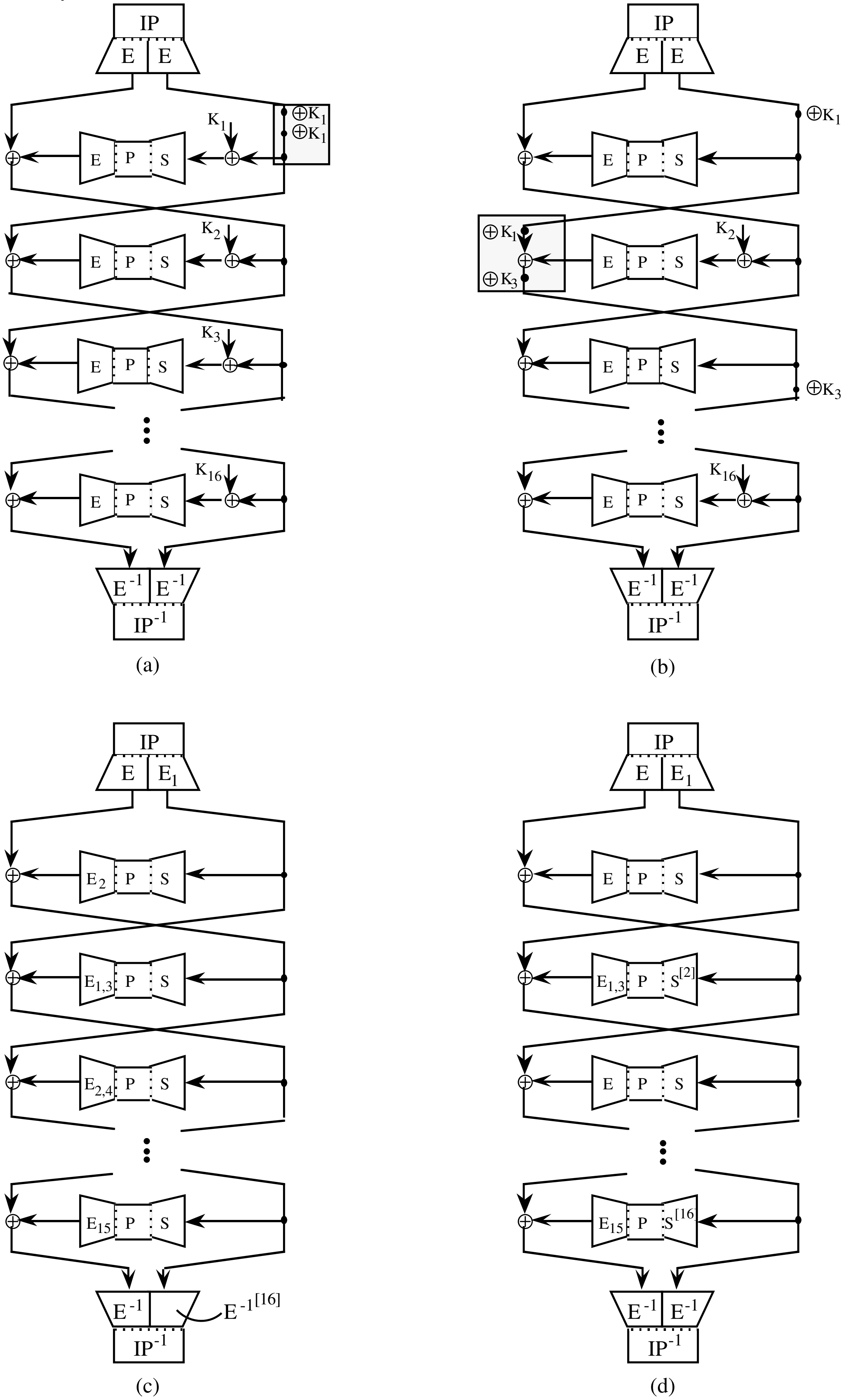
On the other hand, application of §7.1 or §7.2 in the 48-bit model requires shifting of the key-addition to appropriate places, which is explained in Fig. 6 (a) - (c).

§7.3 can be applied directly in all three models of G-DES, but requires quite a bit of additional main memory. It will be demonstrated in §7.5, that the same gain in throughput can be achieved using only half as much additional main memory.

If key-specific tables (described in §7.1 and §7.2) are used in our implementation of the 48-bit model (described in §6.3), in round j (j=1,...,16) the subkeys K_{j-1} and K_{j+1} (where K_0 and K_{17} are both zero) are EORed, see Fig. 6 (c). The subkey K_1 is EORed to the right half of the entries of IP and half the entries of IP^{-1} are reordered with respect to the subkey K_{16}.

If this is done as described in §7.1, 15 additional tables with 2^{12} entries of 64 bits are needed. All tables together now demand 896 Kbytes instead of 416 Kbytes main memory. This is much less than if tables with 2^{18} entries are used and there is not as much additional addressing expense so that the performance could increase by nearly 25% – if the MC680xy had an EOR instruction allowing a register and a main memory location as operands and the register as the target. As noted in §7.1, we have to simulate this instruction by a MOVE followed by an EOR and the attained performance of *335 kbit/s* on the Macintosh II is somewhat disappointing!

If precomputation is done as described in §7.2, we do not need the additional instruction, but the memory increase is 4•15 = 60 additional tables with 2^{12} entries each in the example given above. We will see in §7.5, how the additional instruction can be avoided using only half as much additional main memory.



$S_j^{[i]}$ means that the entries in the S-box S_j are reordered with respect to the subkey K_i;

$E_{i,j}$ means that the subkeys K_i and K_j are EORed to the entries in the table E.

Fig. 6: Combining tables and subkeys in every second round

7.5 Integrating the keys of two rounds in one table

The expense of straight application of §7.2 or §7.3 can be reduced by integrating the keys of two rounds into one round by using the technique described for permutations, and precomputing for the substitutions as well as described in §7.3. Then, only each second round needs key-specific tables, as is derived in Fig. 6 (d). This accounts for 8 additional big tables. In our implementation, this means 32 additional small tables, adding up to 1 Mbyte additional main memory requirement. Nevertheless, this is, as far as we know, the best that can be achieved on the MC68000 processor for G-DES.

The precomputation of the 8 round-specific tables takes less than *0.5 s*, the throughput is *348 kbit/s* on the Macintosh II.

8 Using specific values of DES for performance

There are several proposals in the literature how to use specific values of DES to improve the performance of a DES software implementation. But only one, proposed recently by David C. Feldmeier [Feld_89] and described in §8.1, allows increased throughput.

In §8.2 we describe how this proposal can be combined with precomputed key-specific tables.

8.1 Making use of a special property of the permutation E

David C. Feldmeier [Feld_89 §4.1] uses a special property of the expansion permutation E of DES. A motivation (and necessary, but not sufficient precondition) for his proposal is that the expansion permutation E of DES uses each input bit for at most two output bits (Fig. 7), thus making 2 copies of the input produces each output bit of E. Whether the output bits are in an order which can be used efficiently is another question. But hopefully this can be solved efficiently: Any bit order can be obtained for free at runtime by precomputing, i.e. permuting the bits of the entries of the tables representing IP and P and by permuting the entries of IP^{-1}, cf. Fig. 1, Standard Model. But no matter how we reorder, one restriction by the permutation E of DES cannot be overcome: Since the same bits are used in S-boxes which are next to each other, it follows that such S-boxes must be addressed using different copies. This prevents S-boxes which are next to each other from being combined into one big table.

Feldmeier assumes that the 8 S-boxes are implemented by 4 tables of 2 S-boxes each. He gives the bit order depicted in Fig. 7 so that the S-boxes S_i and S_{i+2} (i = 1, 2, 5, 6) can be combined. This order allows a very efficient implementation of E by

- making 2 copies of the register holding the input,
- rotating one copy by 8 bits,
- ANDing both copies with a mask. This is to zero both the two least significant bits (entries are 4 bytes each) and the two most significant bits of the four 16-bit S-box addresses.

Using Fig. 7, the patient reader can convince himself easily that this really works.

On the MC68000, the low order 16 bits can be used to directly address a table implementing 2 S-boxes with 2^{12} entries each of 4 bytes. Thereafter, the low and high order 16 bits can be interchanged by a SWAP instruction, which is one of the fastest instructions of the MC68000. Now the next table can be addressed directly.

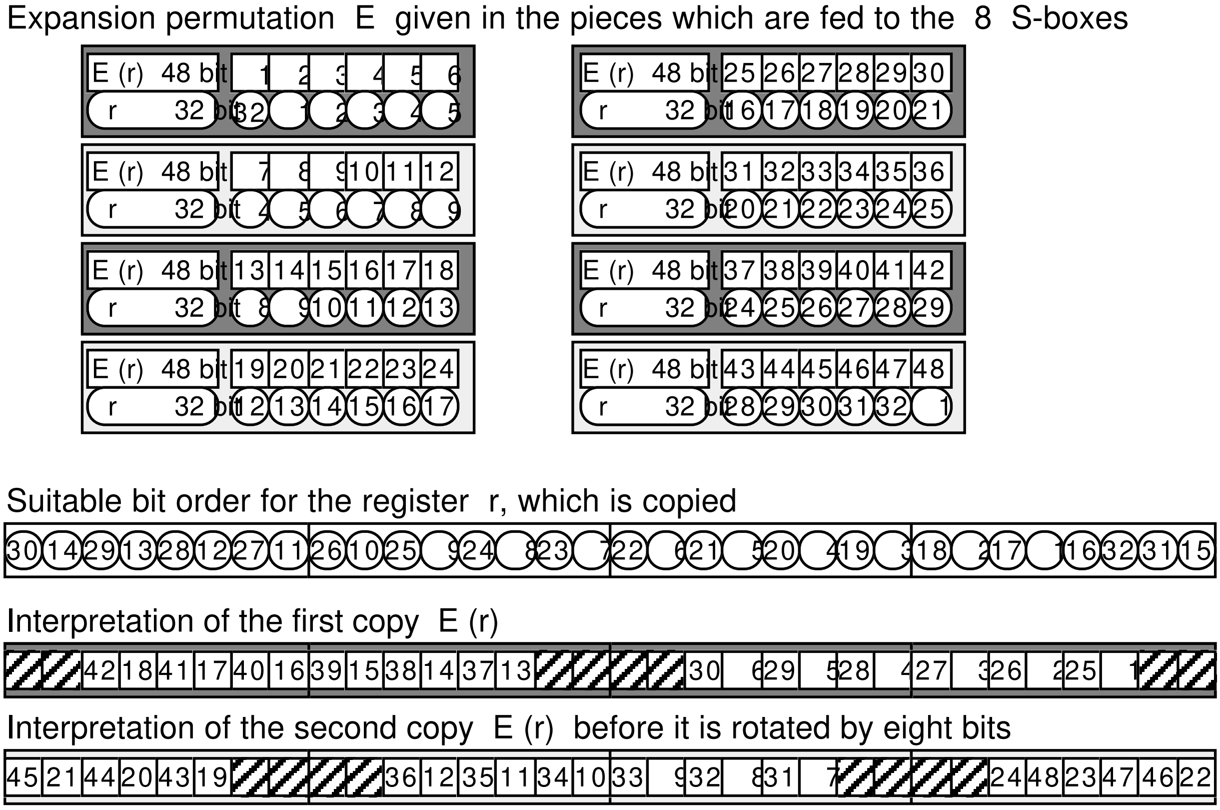


Fig. 7: How to implement the expansion permutation E of DES by register copy, rotate, and AND with mask if the S-boxes are implemented as 4 tables with 2^{12} entries of 4 bytes

As noted above, the property of E to output each input bit at most twice is necessary, but not sufficient, for this efficient implementation to work. It is easy to construct an expanding permutation E, which requires 7 instead of 2 copies of the input: Think of the copies as nodes of a graph, and of the bits, which are put out twice, as edges. If an input bit is output to two different S-boxes, both S-boxes must be addressed using different copies. Now consider the complete graph with 7 nodes. Each node is coincident with 6 edges, which is possible since each S-box has 6 input bits. There are 7•6/2 = 21 edges, which is possible since there are 32 input bits. Therefore, this is an example where 7 copies are required. It is another matter, that this example seems not to make sense cryptographically, since the eighth S-box can only use 6 of the remaining 32-21 = 11 bits. But the latter argument does not apply to the complete graph with 6 nodes, which would make the application of Feldmeier's proposal inappropriate as well.

Of course there are values for E for which copying does not work at all, e.g. feeding the same bit twice to the same S-box. But this example definitely does not make any sense cryptographically!

Summing up, we do not know whether a restriction of G-DES to those E, where Feldmeier's proposal works with 2 copies, called *G-DES with non-arbitrary E*, is a severe one. So we have to leave this as an open problem.

However, for applications where this restriction does not matter, throughput increases and memory requirements are reduced significantly: By adapting the first program described in §6.2 we obtain a throughput of *445 kbit/s* on the Macintosh II using 96 Kbytes of main memory for tables, i.e. 8 with 2^8 entries of 8 bytes for IP and 8 with 2^8 entries of 8 bytes for IP^{-1}, and 4 tables with 2^{12} entries of 4 bytes for the S-boxes.

It is interesting for use in the future (i.e. when more main memory is available) that Feldmeier's proposal can be adapted to other S-box table sizes. Any expanding permutation E which can be implemented in the way described in Fig. 7 can be implemented the same way if the S-boxes are implemented as 2 tables with 2^{24} entries, or roughly the same way if the S-boxes are implemented as

2 tables with 2^{18} and 1 table with 2^{12} entries. Fig. 8 and Fig. 9 give examples of suitable bit orders. In the latter case, the use of bit field moves, which are not available on the MC68000 but on the MC68020 and MC68030 might yield better performance than copy, rotate, and AND with mask.

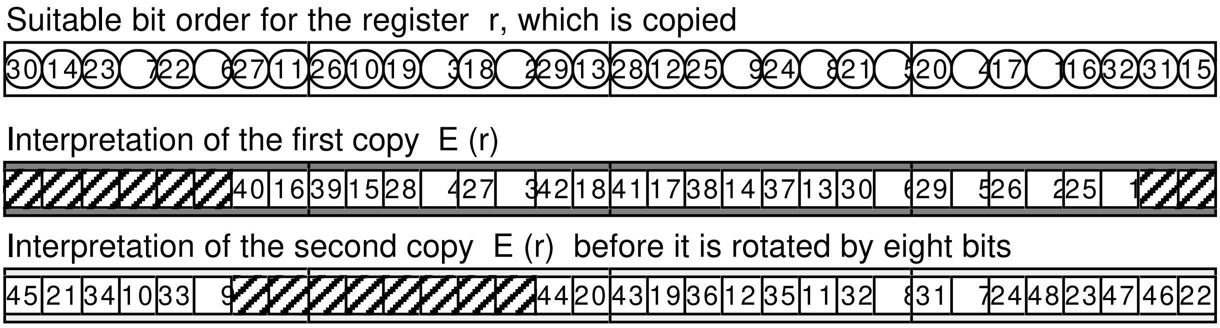


Fig. 8: How to implement the expansion permutation E of DES by register copy, rotate, and AND with mask if the S-boxes are implemented as 2 tables with 2^{24} entries of 4 bytes

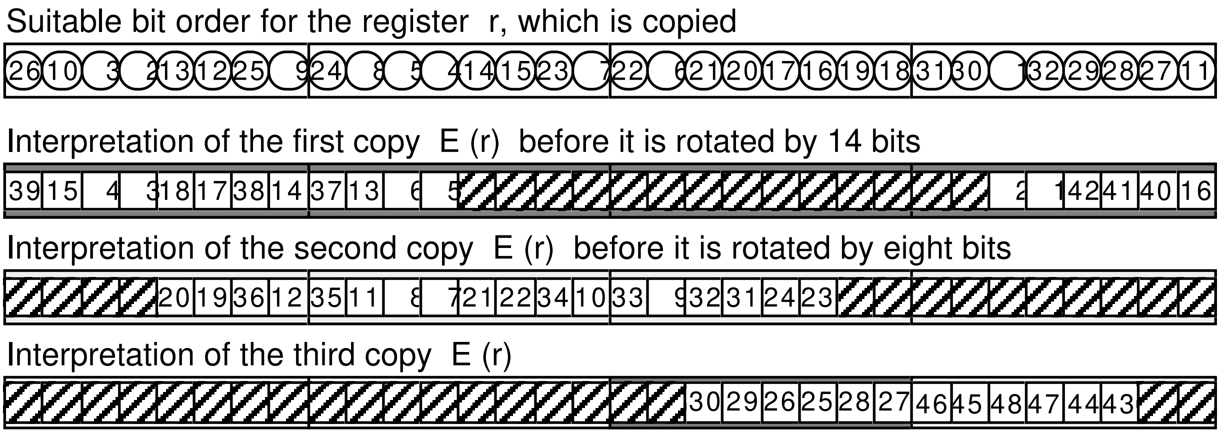


Fig. 9: How to implement the expansion permutation E of DES by register copy, rotate, and AND with mask if the S-boxes are implemented as 2 tables with 2^{18} and 1 table with 2^{12} entries of 4 bytes

8.2 Combining this with precomputed key-specific tables

As in §7, it is interesting to see which improvement in throughput can be gained by precomputing key-specific tables and using more memory. There are two interesting possibilities:

The first is to reorder *all* S-boxes in each round as described in §7.3. This costs additionally $15 \cdot 4 \cdot 2^{12} \cdot 4$ bytes of main memory, and results in a program using tables of 1056 Kbytes but encrypting *499 kbit/s* on the Macintosh II.

The second is to reorder only *half* the S-boxes, e.g. the ones addressed with the second copy, and EORing half the key for the first copy in the round before, i.e. in P. The price for saving half the additional memory is that the last entry of P must now be EORed as described in Fig. 3. Described in terms of the MC680xy, this costs an additional MOVE, cf. §7.1.

For the first round there are two possibilities to incorporate the subkey. One is to EOR it to the last table of IP and EOR the last entry at runtime. A faster approach is to reorder in the first round all S-boxes, and to correct the "error" which is incurred by reusing half the S-boxes of the first round over and over again by appropriate precomputation, i.e. EORing half the first subkey to the last tables of P. After all these precomputations, the resulting program would use tables of 576 Kbytes.

Please note that the second possibility, which works fine for G-DES with non-arbitrary E, is not possible for G-DES in general: Assume that a bit is replicated 6 times and fed to 6 different S-boxes. Then it has to be changed in up to 5 cases, which clearly prevents half the S-boxes from being unchanged in each round. On the other hand, the second possibility does not depend on the specific implementation technique used for E.

Like in §7.5 we can save half the round-specific tables by loss of generality – but while in §7.5 this loss only means restriction to the 48-bit model of G-DES, here it means a restriction to a non-arbitrary E.

8.3 Reducing memory requirements

If S-boxes are combined, two bits of the 32-bit block are fed twice in such a box (assuming we use the standard expansion E and combine two S-boxes which follow each other). If these boxes are reordered with respect to the subkeys (cf. §7.3), we do not need this duplication at runtime. We therefore expand every 8 bits to 10 instead of 12 bits, which results in a memory requirement of 256 Kbytes instead of 1 MByte for the S-boxes. This means 298 Kbytes are needed for a program, which encrypts *537 kbit/s* on the Macintosh II. This way, combining S-boxes and subkeys becomes the fastest version even on MC68030-machines.

A further advantage of this idea is the possibility of combining 4 S-boxes into one big one with the main memory available on PCs in the near future. As we need 6 bits to address one S-box and 4 additional bits for every one that is combined with it, for 4 combined S-boxes we only need 18 bits to address them. This results in a memory requirement of

$$16 \text{ (rounds)} \bullet 2 \text{ (S-boxes/round)} \bullet 2^{18} \text{ (entries/S-box)} \bullet 4 \text{ (bytes/entry)} = 2^{25} \text{bytes} = 32 \text{ MBytes}$$

instead of

$$2 \bullet 2^{24} \text{ (entries/S-box)} \bullet 4 \text{ (bytes/entry)} = 2^{27} = 128 \text{ Mbytes}$$

using 2 key-independent S-boxes. Today, even 32 Mbytes is surely too much for a normal application, but looks very interesting for the near future.

Like in §8.2 we can half the memory requirements for the S-boxes, however, this version is normally a bit slower, because the 68000 processors lack the possibility to EOR a memory content to a register. Measurements prove this, as this program, which needs 178 Kbytes of main memory encrypts only *523 kbit/s* on the Macintosh II. (An exception is the Macintosh IIci with a cacheboard of 64 Kbytes, where this version is the fastest one – it encrypts *831 kbit/s*.)

8.4 Omitting IP and IP^{-1}

At last it is interesting to note that if one is willing to omit the cryptographically insignificant permutations IP and IP^{-1}, the performance advantage of the program versions of §§8.1 and 8.2 compared with those of the §§6 and 7 is the same, i.e. the expense if IP is only roughly halved. In the standard model with the implementation trick suggested by Feldmeier, we have to reorder the bits in the beginning and re-reorder them in the end. In the 32-bit model, P^{-1} at the beginning and P at the end are still necessary. In the 48-bit model, it is E at the beginning and E^{-1} at the end.

9 Comparison of existing software and hardware implementations

9.1 Implementation techniques on MC680xy processors

"Test single bits" implementations are inferior to "table look-up" implementations with respect to performance. By using an appropriate table size for the implementation, "table look-up" can be adapted to the requirements of the application, but only in tight limits: To achieve twice as much speed, about 100 times as much main memory is needed in the area of interest on today's personal computers and workstations.

The 48-bit model is probably more suitable for implementation even on some 32-bit and pseudo 32-bit machines, cf. Table 1.

The performance of the assembler program according to §7.5, i.e. precomputing key-specific tables and integrating the keys of two rounds in one table, is surprisingly poor on the Macintosh IIx. We suppose, the reason for this is that the tables summing up to 1416 Kbyte cannot be addressed at once by the memory management unit (MMU) which is part of the MC68030. The MMU can only hold 22 descriptors at a time, and each descriptor can be used to address at most 32 Kbyte. Therefore at any moment, the MMU can at most address 22•32 Kbyte = 704 Kbyte (without loading a "new" and overwriting an "old" descriptor).

The values for the Macintosh SE and Portable are not given explicitly in Table 1 since for CPU-intensive tasks, the Macintosh SE is uniformly 20% faster than the Macintosh Plus [Loeb_87 p. 202] and the Macintosh Portable is twice as fast as the SE [ThHa_89 p. 108].

The right column gives the values measured on the Macintosh IIx or IIcx which have exactly the same performance. The Macintosh SE/30 is even a bit faster due to the lack of NuBus logic. The Macintosh IIci is, without an installed cache board, about 55% faster [ThHa_89 p. 108] (our measurements suggest 49%). Installation of a cache board increases the throughput by another 20 to 30% (our measurements suggest 14 to 22%). But in contrast to all other ratings, the latter will not be a uniform one.

The decisions

- to implement not only DES but G-DES (or in other words: not to try to use structures in the specific values of the permutations) and
- to use instructions of the MC68000 only (cf. §4.3)

turned out to be good. That way, we could concentrate on the essentials and obtain, to the best of our knowledge, by far the fastest implementation of DES on a personal computer. Incorporating a recent proposal by Feldmeier (cf. §8.1) to make use of a special property of the expanding permutation E of DES was easy and increased throughput by 20 to 75%. Since this is the only way to achieve better performance for DES than for G-DES we know, it does not seem to make sense to implement anything more narrow in software than G-DES with non-arbitrary E.

After we have seen what can really be achieved by a software implementation, we have to address again implementations in portable high level programming languages (cf. §4.2).

The implementation techniques described in §§6-7 could be used to tune our PASCAL reference version (§4.1) considerably. But without requiring unreasonably large main memories, at most a factor 8 in speed can be gained. Additionally, one might be tempted to use PASCAL extensions provided by many implementations, e.g. EOR instructions on bitstrings.

executed on program version memory requirem.	mode of operation	Mac Plus 68000, 7.83 MHz max. 32-bit opds 16-bit data bus 2.5 Mbyte memory			Mac II 68020, 15.67 MHz max. 32-bit opds 32-bit data bus 256 byte instr.cache 5 Mbyte memory			Mac IIx, IIcx, SE/30 68030, 15.67 MHz max. 32-bit opds 32-bit data bus 256 byte instr.cache 256 byte data cache 5 Mbyte memory		
from memory to memory										
§4.1	ECB	1.1	0.6	0.4	5.0	2.6	1.7	6.3	3.3	2.2
G-DES	CBC	1.0	0.6	0.4	4.1	2.2	1.5	5.4	3.0	2.1
PASCAL	CFB(8,8)	0.1	0.1	0.1	0.5	0.3	0.2	0.6	0.4	0.3
(32-bit model,	OFB(64)	0.9	0.5	0.4	3.8	2.1	1.5	5.0	2.9	2.0
assign single bits)	CBCauth	1.0	0.6	0.4	4.4	2.4	1.7	5.6	3.1	2.1
24 Kbyte	PCBC	1.0	0.6	0.4	4.0	2.2	1.5	5.3	3.0	2.1
§5	ECB		20			68			75	
G-DES	CBC		20			68			75	
Assembler	CFB(8,8)		2			8			9	
(32-bit model,	OFB(64)		20			68			75	
test single bits)	CBCauth		20			68			75	
29 Kbyte	PCBC		20			68			75	
§6.2	ECB	55	31	21	245	138	96	271	152	106
G-DES	CBC	55	30	21	248	138	97	273	152	106
Assembler	CFB(8,8)	7	4	3	31	17	12	34	19	13
(32-bit model,	OFB(64)	62	33	22	284	149	102	310	163	111
table look-up)	CBCauth	59	32	22	268	144	99	295	158	109
82 Kbyte	PCBC	55	30	21	245	139	97	270	152	106
§6.3	ECB	68	39	28	327	195	140	366	216	154
G-DES	CBC	68	39	28	331	196	141	367	217	154
Assembler	CFB(8,8)	8	5	3	41	24	17	46	27	19
(48-bit model,	OFB(64)	81	43	29	424	225	155	461	245	168
table look-up)	CBCauth	72	40	28	367	209	147	408	229	161
426 Kbyte	PCBC	68	39	28	329	196	140	366	216	154
§7.5	ECB		74			348			254	
G-DES	CBC		73			356			254	
Assembler	CFB(8,8)		9			44			31	
(48-bit model,	OFB(64)		88			453			296	
table look-up)	CBCauth		78			391			291	
1426 Kbyte	PCBC		73			354			253	
§8.1 **G-DES w.**	ECB	87	50	35	445	262	188	487	286	204
non-arbitrary E	CBC	86	49	35	455	266	190	494	289	205
Assembler	CFB(8,8)	11	6	4	56	33	24	61	36	26
(standard model,	OFB(64)	97	53	37	545	297	205	589	320	221
table look-up)	CBCauth	99	54	37	559	301	207	600	323	222
106 Kbyte	PCBC	86	49	35	453	264	189	495	288	205
§8.2 **G-DES w.**	ECB		96			499			326	
non-arbitrary E	CBC		95			506			328	
Assembler	CFB(8,8)		12			63			41	
(standard model,	OFB(64)		110			627			375	
table look-up)	CBCauth		110			638			382	
1066 Kbyte	PCBC		95			502			328	

§8.3 **G-DES w.**	ECB	101	537	584
non-arbitrary E	CBC	101	537	579
Assembler	CFB(8,8)	12	66	72
(standard model,	OFB(64)	117	658	704
table look-up)	CBCauth	117	673	714
298 Kbyte	PCBC	101	543	589
from disk to disk		external HD 20	internal SCSI (40)	internal SCSI (80)
§9.2	ECB	37	266	292
G-DES	CBC	38	261	287
Assembler	CFB(8,8)	8	36	42
(48-bit model,	OFB(64)	41	323	348
table look-up)	CBCauth	51	320	348
426 Kbyte	PCBC	38	258	288

In all experiments, we used original Apple hardware exclusively, as software finder D1-6.1 and system D1-6.0.3, the assembler of the Macintosh 68000 Development System, and the Pascal-compiler THINK's Lightspeed Pascal 2.01.

Table 1: Encryption performance of our program versions on different Macintoshs in kbit/s. The entries are of the form single encryption { | double encryption | triple encryption }

9.2 Integration in an application program

We have integrated all described program versions in an application program working with files on hard disk or floppy disk. While the versions in §§4-5 are only slightly slower when working with the hard disk than using the main memory only, this is not true for the versions in §§6-8. This is quite satisfying because it suggests that our fast software implementations of G-DES (and G-DES with non-arbitrary E) start to reach the other system limits. The version described in §6.3 achieves *261 kbit/s* on a Macintosh II in encrypting a file of half a Mbyte in CBC mode. The whole application program uses less than 530 Kbytes main memory, including 31 Kbytes I/O-buffers. Since it is inconvenient to transport lengthy tables on floppy disk or by electronic mail, the program builds up the lengthy tables at runtime, which lasts between *0.5 s* (on a Macintosh IIci with 64Kbyte cacheboard) and *4.0 s* (on a Macintosh Plus). On a Macintosh II it needs *1.2 s*, the version described in §7 needs *2.2 s*, the one described in §8.1 needs *1.0 s* and the one described in §8.3 needs *1.3 s*. The size of the program including all compact tables and two character-oriented user interfaces for the novice and experienced user is less than 70 Kbytes.

Relatively much time has been spent defining and implementing a moderately user friendly, character-oriented interface for our application program. We decided not only to allow direct entry of 16•48 key bits, but also entry of 56 key bits (a normal DES key) as well as the use of arbitrary long keys using the technique recommended in [Gran_87]. All sorts of keys can be entered both as extended ASCII-characters and in hexadecimal form.

Currently, we are working on an integration of RSA [RSA_78] for key-distribution and GMR [GoMR_88] for digital signatures.

9.3 Derivation of an efficient implementation for intel 8086

For an application where only DES was needed, Manfred Mühlan has ported our G-DES program to intel 8086, suitable for e.g. IBM PC, XT, AT and PS/2 [Mühl_89]. The performance is given in Table 2. Since Manfred Mühlan restricted himself to the usage of intel 8086 instructions only (as we did to the MC68000), his program is executable on all intel 80x8y processors. But since the intel 8086 offers no 32-bit operands (nor does the intel 80286), the 32-bit data bus of the intel 80386 is badly used. Therefore, it is much more appropriate to use special instructions of the intel 80386, than it is to do the same concerning the MC68020 or MC68030.

Manfred Mühlan misused the registers ES, DI, SI of the intel 8086 to hold as many intermediate results as possible.

- This "emulation" of the MC68000 data registers (where we had no reason to misuse address registers to compensate for the lack of data registers), that
- the intel 8086 has an EOR instruction allowing a register and a main memory location as operands and the register as the target (we missed this instruction on the MC68000 in all our program versions, and it became apparent in §7.1) and that
- the instruction code of the intel 8086 is more dense than that of the MC68000

are the three reasons why Manfred Mühlan's program performs so well on the intel 80286 compared with the MC68000. (Please note that Manfred Mühlan did not implement the 48-bit model. So *77.1 kbit/s* on the IBM AT stand against 55 kbit/s•1.2 = 66 kbit/s on the comparable Macintosh SE. For a few other performance comparisons between Macintosh SE and IBM AT cf. [Loeb_87 p. 203].)

As far as we have insight into Manfred Mühlan's work, which has been tutored by Michael Spreng, its expense and results strongly confirm our claims in §4 and §9.1.

Please note that the restriction to G-DES with non-arbitrary E (cf. §8.1) would significantly improve performance. Using the ratio of of the Macintosh Plus (87/55 = 1.58) we extrapolate 77.1 kbit/s•1.58 = 122 kbit/s on the IBM AT.

executed on program version memory requirem.	mode of operation	IBM AT i80286, 8 MHz max. 16-bit opds 16-bit data bus	Zenith Supersport i80286, 12 MHz max. 16-bit opds 16-bit data bus	Toshiba T5100 i80386, 16 MHz max. 32-bit opds 32-bit data bus
from memory to memory				
Assembler	ECB	77.1	90.1	148.9
(32-bit model,	CBC	72.2	83.8	136.5
table look-up)	CFB(8,8)	8.5	10.1	16.3
82 Kbyte	OFB(64)	69.2	82.4	132.4
from disk to disk		80 ms access time	28 ms access time	29 ms access time
Assembler	ECB	64.6	77.1	128.5
(32-bit model,	CBC	61.1	72.2	119.2
table look-up)	CFB(8,8)	8.3	9.9	16.0
82 Kbyte	OFB(64)	59.6	72.4	122.5

Table 2: Encryption performance (in kbit/s) for single encryption of Manfred Mühlan's G-DES program on different PC-compatibles. IP is not saved in OFB mode.

9.4 Comparisons with other fast DES software implementations

In [WiHi_80], the performance of two software implementations of DES is given.

Gumpertz's requires 2.5 Kbyte for the program and 150 bytes for the data. Its performance is given in figures, so even the second decimal of the numbers we give may be wrong: intel 8080 (2MHz) *3.5 kbit/s*, TI TMS9900 (3MHz) *15 kbit/s*, DEC VAX 11/780 *150 kbit/s*, DEC KL-10 *250 kbit/s*, IBM 370/168 *1300 kbit/s*, Cray-1 *8500 kbit/s*.

Herbert Bright's program Desqik encrypts *304 kbit/s* on machines in the Amdahl 470V/6 class.

In a recent survey [Bras_88 p. 17], two examples are given: *160 kbit/s* on a DEC VAX 11/780 and *20 kbit/s* on an IBM PC (intel 8088, 4.77 MHz, 8-bit data bus).

In 1986, P. Karn and J. Gillogly published a DES program in USENET, which is written in C and uses the techniques described in §6 to implement the standard model of DES: IP and IP^{-1} are tabulated in 16 tables of 2^4 entries each, E is programmed with bit field moves, and the concatenation of S and P is tabulated in 8 tables of 2^6 entries each. Oliver Kowalski, GMD, St. Augustin, compiled and executed it and obtained *29.84 kbit/s* on DEC VAX 11/780 and *84.88 kbit/s* on SUN 3/50 (MC68020, 16.67 MHz).

If you take it with a pinch of salt, the following comparison can be drawn:

We can extrapolate the performance of the DES implementation on an IBM PC to an IBM AT: 20 kbit/s•(8 MHz / 4.77 MHz)•(16-bit data bus / 8-bit data bus) = 67 kbit/s. Hereby we assume that the overestimation by the factor quotient of the data busses (not in all cases are really 2 bytes to move between the processor and the main memory) and the underestimation of neglecting that some instructions possibly require fewer cycles on a later designed processor cancel out.

(This assumption matches the performance comparisons given in [Fing_85 p. 274] very well: For the two computation intensive benchmarks, the performance ratios between an IBM AT (6 MHz) and an IBM PC are 191s/80s = 2.39 and 69s/27s = 2.56. Using our assumption, we calculate (6 MHz / 4.77 MHz)•(16-bit data bus / 8-bit data bus) = 2.52.)

If this assumption holds, Manfred Mühlan's G-DES implementation, which achieves 77.1 kbit/s (and could achieve approx. 122 kbit/s, cf. §9.3), compares very favorably with the best known implementation on IBM PC according to [Bras_88]. Since making good implementations for IBM PC etc. is surely challenging for many people, this also suggests that our design decisions and the hints on how to implement DES or G-DES are on the right track.

9.5 A VLSI standard cell implementation of MDES

The statements of §2.3 on the expense of hardware implementations of G-DES have in part been validated by a VLSI standard cell implementation directed by our colleagues Thomas Kropf and Oliver Haberl.

Until now, a MDES-chip, which is able to cipher 40 Mbit/s internally, but due to its bytewise and asynchronous, two-way handshaked input/output operates with 10 Mbit/s externally only, has been designed, produced and tested in a student's project of limited scale [BeFG_89]. It comprises 7155 gate equivalents, is implemented using 2 µm technology on an area of 47 mm^2, and can be operated with a maximum clock rate of 10 MHz.

9.6 Comparisons with fast DES hardware implementations

According to the literature, DES-chips encrypting 15 Mbit/s [Abbr_84, WiHi_80] are available for sale, and those encrypting 32 Mbit/s are announced [VHVD_88].

The MDES-chip of §9.5 has an internal ciphering rate, which compares favorably with that of the DES-chips. If a sophisticated controller would be added to the MDES-chip (as described for the DES-chip in [VHVD_88]) and its input/output made 4 bytes wide and possibly synchronous, the internal ciphering rate could be offered externally nearly in full.

10 Implications for the usage of high quality encryption

In §2.1, we mentioned that political reasons, i.e. the aim to hinder high quality encryption at least on a mass basis, brought about the standardization of cryptosystems being abandonned. The reason for the cipher boards, notably the NSA, to achieve this and to control the production or at least the export of cryptographic hardware, is that otherwise gathering intelligence information by tapping long haul communication would become impossible as far as message contents are concerned [PfWa_86].

The performance of our G-DES programs demonstrates that at least for long haul networks, software implemented high quality encryption is fast enough for nearly all applications, if powerful personal computers or workstations are used. Since the dissemination of cryptographic software cannot be controlled, either the production of personal computers has to be stopped or cryptosystems can (and for other reasons: should) be standardized as planned a few years ago!

Acknowledgment

We thank Manfred Böttger, Prof. Winfried Görke, Andreas Ort, Birgit Pfitzmann, and Michael Waidner for a lot of useful comments and stimulating discussions.

References

Abbr_84 C. R. Abbruscato: Data Encryption Equipment; IEEE Communications Magazine Vol. 22, Nu. 9, September 1984, pp. 15-21.

Aßma_89 Ralf Aßmann: Effiziente Software-Implementierung von verallgemeinertem DES, Diploma-Thesis, Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, February 1989.

Ath1_86 Tom Athanasiou: Encryption: Technology, Privacy, and National Security; Technology Review, Cambridge, Mass., Aug./Sept. 1986, pp. 57-66.

BeFG_89 Wilfried Beller, Jürgen Frößl, Thomas Giesler: Spezifikation und Implementierung eines erweiterten DES-Algorithmus als VENUS-Standardzellenchip; Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, 1989.

Ber1_83 Thomas A. Berson: Long Key Variants of DES; Crypto 82, Plenum Press, New York 1983, pp. 311-313.

Bras_88 Gilles Brassard: Modern Cryptology - A Tutorial; LNCS 325, Springer-Verlag, Berlin 1988.

CACM6_87 General News and Notes: Computer Security Act Stresses Encryption Standard; Communications of the ACM, Vol. 30, Nu. 6, June 1987, p. 572.

Carr_87 John M. Carroll: Strategies for Extending the Useful Lifetime of DES; Computers & Security Vol. 6, Nu. 4, August 1987, pp. 300-313.

DaPa_83 D. W. Davies, G. I. P. Parkin: The Average Cycle Size of the Key Stream in Output Feedback Encipherment; Proc. Burg Feuerstein 1982, LNCS 149, Springer-Verlag, Berlin, 1983, pp. 263-279.

DaPr_89 D. W. Davies, W. L. Price: Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer; (2nd ed.) John Wiley & Sons, New York 1989.

DDFG_84 Marc Davio, Yvo Desmedt, Marc Fosséprez, René Govaerts, Jan Hulsbosch, Patrick Neutjens, Philippe Piret, Jean-Jacques Quisquater, Joos Vandewalle, Pascal Wouters: Analytical characteristics of the DES; Crypto '83, Plenum Press, New York 1984, pp. 171-202.

DDGH_85 Marc Davio, Yvo Desmedt, Jo Goubert, Frank Hoornaert, Jean-Jacques Quisquater: Efficient hardware and software implementations for the DES; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, pp. 144-146.

Denn_82 Dorothy E. Denning: Cryptography and Data Security; Addison-Wesley, Reading, Mass.; 1982; Reprinted with corrections, January 1983.

Feld_89 David C. Feldmeier: A High–speed Crypt Implementation; Crypto '89, August 20-24 1989, Abstracts, 51-64.

Fing_85 Alan Finger: IBM PC AT (System Review); Byte Vol. 10, Nu. 5, May 1985, pp. 270-277.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. Vol. 17, Nu. 2 , 1988, pp. 281-308.

Gran_87 Lynn Grant: DES Key Crunching for Safer Cipher Keys; SIGSAC, Security, Audit & Control Review Vol. 5, Nu. 3, Summer 1987, pp. 9-16.

Hell_82 Martin E. Hellman: Cryptographic Key Size Issues; digest of papers compcon spring 1982, February 22-25, pp. 130-131.

Horg_86 John Horgan: NSA explains encryption program to IEEE Privacy Subcommittee; The Institute, IEEE, Vol. 10, Nu. 9, September 1986, p. 2.

Kola_85 Gina Kolata: NSA to Provide Secret Codes; Science Vol. 230, October 1985, pp. 45-46.

LeMa1_89 Arjen K. Lenstra, Mark S. Manasse: Factoring – where are we now?; IACR Newsletter, A Publication of the International Association for Cryptologic Research, Vol. 6, Nu. 2, June 1989, pp. 4-5

Loeb_87 Laurence H. Loeb: The Macintosh SE (System Review); Byte Vol. 12, Nu. 9, August 1987, pp. 201-205.

LuRa_88 Michael Luby, Charles Rackoff: How to construct pseudorandom permutations from pseudorandom functions; SIAM J. Comput. Vol. 17, Nu. 2, 1988, pp. 373-386.

LuR1_88 Michael Luby, Charles Rackoff: A study of password security; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, pp. 392-397.

LuRa_89 Michael Luby, Charles Rackoff: A study of password security; Journal of Cryptology Vol. 1, Nu. 3, 1989, pp. 151-158.

Mass_88 James L. Massey: An Introduction to Contemporary Cryptography; Proceedings of the IEEE Vol. 76, Nu. 5, May 1988, pp. 533-549.

MeMa_82 Carl H. Meyer, Stephen M. Matyas: Cryptography - A New Dimension in Computer Data Security; (3rd printing) John Wiley & Sons, 1982.

Merk1_89 Ralph C. Merkle: One Way Hash Functions and DES; Crypto '89, August 20-24 1989, Abstracts, 407-421.

Morr_78 Robert Morris: The Data Encryption Standard – Retrospective and Prospects; IEEE Communications Society Magazine Vol. 16, Nu. 6, Nov. 1978, pp. 11-14.

Mühl_89 Manfred Mühlan: Portierung, Optimierung und Integration eines Kryptographieprogrammes in eine Software-Umgebung; Studienarbeit am Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1989.

PfWa_86 Andreas Pfitzmann, Michael Waidner: Networks without user observability – design options; Eurocrypt '85, LNCS 219, Springer-Verlag, Heidelberg 1986, pp. 245-253; Computers & Security Vol. 6, Nu. 2, 1987, pp. 158-166.

Prei_88 Ralph J. Preiss: Classification of 'sensitive' information is once again in civilian hands; Computer, IEEE, Vol. 21, Nu. 3, March 1988, p. 124.

Pric_88 Wyn L. Price: Standards for Data Security – A Change of Direction; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, pp. 3-8.

RSA_78 R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM Vol. 21, Nu. 2, February 1978, pp. 120-126.

Seel_89 Donn Seeley: Password Cracking: A Game of Wits; Communications of the ACM Vol. 32, Nu. 6, June 1989, pp. 700-703.

ThHa_89 Tom Thompson, Frank Hayes: The Portable and the Powerful; Byte Vol. 14, Nu. 10, October 1989, pp. 98-108.

VHVD_88 I. Verbauwhede, F. Hoornaert, J. Vandewalle, H. De Man: Security considerations in the design and implementation of a new DES chip; Eurocrypt '87, LNCS 304, Springer-Verlag, Heidelberg 1988, pp. 287-300.

WaPP_87 Michael Waidner, Birgit Pfitzmann, Andreas Pfitzmann: Über die Notwendigkeit genormter kryptographischer Verfahren; Datenschutz und Datensicherheit Nu. 6, June 1987, pp. 293-299.

WiHi_80 Deborah Williams, Harvey J. Hindin: Can software do encryption job?; Electronics July 3, 1980, pp. 102-103.

Yasa_76 E. K. Yasaki: Encryption Algorithm: Key Size is the Thing; Datamation Vol. 22, Nu. 3, 1976, pp. 164-166.