Chapter I

PERMUTATION OPERATIONS IN BLOCK CIPHERS

R. B. Lee [I.1] , [I.2], R. L. Rivest [I.3], M. J. B. Robshaw [I.4], Z. J. Shi [I.2], Y. L. Yin [I.2]

New and emerging applications can change the mix of operations commonly used within computer architectures. It is sometimes surprising when instruction-set architecture (ISA) innovations intended for one purpose are used for other (initially unintended) purposes. This chapter considers recent proposals for the processor support of families of bit-level permutations. From a processor architecture point of view, the ability to support very fast bit-level permutations may be viewed as a further validation of the basic word-orientation of processors, and their ability to support next-generation secure multimedia processing. However, bitwise permutations are also fundamental operations in many cryptographic primitives and we discuss the suitability of these new operations for cryptographic purposes.

I.1   INTRODUCTION

To support new user requirements such as digital multimedia processing and secure information processing, the basic operations supported within new generation processors might evolve. For a general-purpose microprocessor, it is desirable that any

I. PERMUTATION OPERATIONS IN BLOCK CIPHERS

added instructions have multiple uses, rather than being specific to only one algorithm or to one application. Since secure communications and networking have become critical features of many applications, it would seem to be advantageous for the architectural and cryptographic communities to explore the following questions. Are there instruction-set architecture (ISA) innovations that may occur in a widespread way that might also be used beneficially in the design of cryptographic algorithms? Alternatively, are there desirable instructions, perhaps motivated by the design of cryptographic algorithms, that might also be useful for other emerging applications? To begin exploring these questions, this chapter examines recently-proposed bit permutation operations from the perspective of cipher design and cryptanalysis. In addition to studying the cryptographic properties of such permutation operations in isolation, we consider their role in the design of new ciphers.

The contributions of this chapter are as follows. We examine the cryptographic properties of bit-level permutations in the construction of new ciphers or in strengthening existing ciphers. In particular, we study the cryptographic properties of the *group operation* GRP [13, 23], as well as OMFLIP [13, 27], which were recently identified for possible inclusion in future processor architectures. We investigate the properties of GRP and OMFLIP and consider how their inclusion within a cryptographic design might change the properties of the scheme. As a detailed example, we consider the implications of incorporating the GRP operation into a block cipher and discuss some of the issues that arise. Provided care is taken, it may be possible for the support of new operations to lead to new designs offering higher performance and reduced energy consumption; something which would be particularly important for constrained environments like hand-held devices. In Section I.2, we motivate the study of permutation operations from both an architectural and cryptographic viewpoint. We provide our design goals in Section I.3 and detailed definitions of bit permutation operations in Section I.4. In Section I.5, we analyze the cryptographic properties of GRP and, as an example, in Section I.6 we explore how one might use GRP in a variant of the block cipher RC5 [20]. Section I.7 concludes the chapter.

I.2 Motivation for New Permutation Operations

Bit-level permutation operations are very important from both an architectural and cryptographic point of view.

Architecturally, the ability to support very fast bit-level permutations may be the next step in the evolution of word-oriented processors to support new multimedia and secure information processing workloads. Bit level computation is used in Huffman encoding and decoding, for example, and general-purpose processors are optimized for word-oriented computation. Hence, their instruction set architecture (ISA) provides limited support for the manipulation of data items smaller than a word. Currently, only simple bit-level operations like logical operations and shifts are implemented in microprocessors. For multimedia processing, processor architectures have already incorporated the concept of subword parallelism [11, 12] where subwords are typically 8-bit pixels or 16-bit audio samples. A subword-parallel instruction performs the same operation on multiple pieces of data (sub-

words) packed in one or more registers [11]. Subword-parallel arithmetic operations can efficiently exploit the data parallelism in processing images, video, graphics and audio. Subword-parallel instructions—first introduced to accelerate multimedia in PA-RISC microprocessors [11, 12]—have now been added to all microprocessors [6, 8, 11, 12, 18, 19]. These ISA additions have swept the microprocessor industry in a matter of five years, demonstrating that new architectural features will be added to processors if they provide significant performance or other advantages at a very low cost. Subword permutation operations are often necessary to rearrange subwords into proper positions in registers so that subsequent operations can be applied to all subwords in parallel. As we decrease the size of the subword, we significantly increase the difficulty of achieving all possible permutations since the number of items to be permuted increases. Nevertheless, recent work [13, 23, 26, 27] has examined architectural solutions that can achieve any arbitrary permutation of both single-bit and multi-bit subwords packed in a register.

Cryptographically, bit-level operations are useful in the design of many algorithms, particularly block ciphers, stream ciphers, and hash functions. The design of the block cipher DES [16] is an important landmark in this regard. The security of many of these algorithms relies on what Shannon termed *confusion* and *diffusion* [22] which are typically attained by a judicious combination of simple operations. Bit-level permutations naturally provide certain effects which are not easily obtained through word-level operations. However, bit-level permutations tend to be slow on current programmable processors, since they have to be emulated using other instructions. While all processors implement add, subtract, logical, memory load and shift operations, the only bit-level permutations that might be routinely supported in microprocessors are *bitwise rotations* which form a very small subset of all possible bit-level permutations. Some processors support fixed bitwise rotations where the amount of rotation is specified at compile time; even fewer processors support data-dependent rotations (DDR) where the rotation amount is only available at execution time. DES [16] uses bit-level permutations which are very fast in special-purpose hardware, but inherently slow in software. While the few fixed permutations in DES can be sped up using table lookup techniques in software, it is not feasible to do this for all possible data-dependent permutations. In [13, 27] the use of OMFLIP to speed up the performance of fixed permutations within DES is explored.

More recent proposals for hash functions and encryption functions—including the new AES [17]—have demonstrated a move away from bit-level operations and toward a mix of word-oriented operations such as arithmetic and logical operations, as well as some form of table lookup accomplished with memory load instructions. Much of this, however, might be due to the currently poor support for bit-level permutations; currently no processors implement more general purpose bit-wise permutation instructions. Nevertheless, the role of bit-wise permutations remains fundamental and it is interesting to consider whether or not increased support for bit-level permutation operations might encourage their use in new cipher designs.

Finally, another interesting application of bit-level permutations is in the obfuscation of data [3] within tamper-resistant chips. The use of keyed bit-level per-

mutations can provide a mechanism to enhance the resistance of such hardware deployments to so-called "probing attacks". It would be interesting to consider the applications of the techniques we discuss in this chapter to this particular problem.

I.3 DESIGN GOALS FOR NEW PERMUTATION OPERATIONS

A permutation operation for our architectural and cryptographic needs should ideally satisfy the following goals:

- Goal 1: *Be general-purpose and flexible.* The new permutation operation should be general-purpose, rather than specific to a given algorithm. For example, the permutation operation might have uses in applications as diverse as multimedia applications, sorting applications, and cryptography.

- Goal 2: *Be easy to implement.* The new permutation operation should be easy to implement in a variety of processors, from high-performance microprocessors down to the simplest processors suitable for small information appliances and even smart cards. Since many of these processors have simple architectures, the new operation should ideally require no more than two source registers, and write to one destination register upon completion of execution. Ideally, the latency through the functional unit should allow the operation to execute in a single cycle. On the other hand, if the direct hardware support for the operation is not available, other instructions should be able to emulate the operation.

- Goal 3: *Have good cryptographic properties.* The new permutation operation should have good cryptographic properties, and be resistant to common cryptanalytic attacks as well as not opening new weaknesses.

To help judge how successful such new operations might be, we will use the *data-dependent rotation* (DDR) as a means for comparison. This operation has been used in the block cipher RC5 [20] and it has been widely studied from a cryptographic perspective. Like all the permutations considered in this chapter, the action of DDR is not fixed. Instead, the bits of a *control register* are used to specify the permutation to be applied to the bits in the *data register*. One potential weakness of DDR is that only the lower $\log_2(w)$ bits of the w-bit control register are used to effect the permutation. For convenience, $\log_2(w)$ will be denoted with $\lg(w)$ in this chapter. The potential weakness of DDR has been used to mount certain theoretical attacks on RC5 and so it seems that new permutation operations with more control bits might potentially be cryptographically useful.

I.4 PERMUTATION OPERATIONS: GRP AND OMFLIP

The general form of a permutation operation will be written as $Z = X \bullet Y$ where the bits (or subwords) of X are permuted according to the value of bits (or subwords) of Y. The data-dependent rotation (DDR) typically denoted as $Z = X \lll Y$

Y	1	0	0	1	1	0	1	0

X	a	b	c	d	e	f	g	h

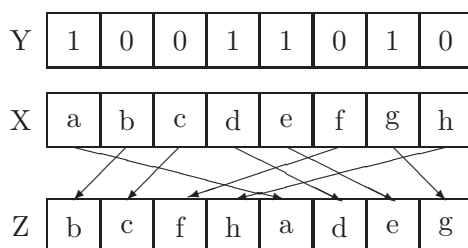| Z	b	c	f	h	a	d	e	g

Figure I.1. An 8-bit GRP operation

takes two operands X and Y, generating a result Z where all are w-bit words. The word X is rotated left by the amount specified in the lower $\lg(w)$ bits of Y. Several new permutation instructions such as PPERM [13], GRP [13, 23], CROSS [13], OMFLIP [13, 27], and BFLY [26] have been proposed for arbitrary bit-level permutations. However, we will restrict our attention to GRP and OMFLIP in this chapter.

I.4.1   Definition of GRP

The GRP operation will be written as $Z = X \triangleleft Y$ where the bits in X are divided into two groups depending on whether the corresponding bit in Y is 0 or 1. The two groups of bits are then placed next to each other in Z. The bits with a control bit of 0 are placed at the left end; the bits with a control bit of 1 at the right end. Fig. I.1 shows an example of an 8-bit GRP operation. Since the control bits of x_0, x_2, x_5, x_6 are 0, these four bits are placed at the left end in Z. The bits x_1, x_3, x_4, x_7 are placed at the right since their control bit has the value 1.

If the GRP operation is used in a cryptographic algorithm, the inverse operation, UNGRP for *ungroup*, may be needed for decryption. Here we give programmatic definitions of GRP and UNGRP. Let $X = x_{w-1} \ldots x_0$, $Y = y_{w-1} \ldots y_0$, and $Z = X \triangleleft Y = z_{w-1} \ldots z_0$ be w-bit words.

```
GRP ◁                               UNGRP

j = 0;                              j = 0;
for (i = 0;  i < w;  i = i + 1)     for (i = 0;  i < w;  i = i + 1)
    if (y_i = 1){                       if (y_i = 1){
       z_j = x_i;                          z_i = x_j;
       j = j + 1;}                         j = j + 1;}
for (i = 0;  i < w;  i = i + 1)     for (i = 0;  i < w;  i = i + 1)
    if (y_i = 0){                       if (y_i = 0){
       z_j = x_i;                          z_i = x_j;
       j = j + 1;}                         j = j + 1;}
```

I. PERMUTATION OPERATIONS IN BLOCK CIPHERS

I.4.2   Definition of OMFLIP

The OMFLIP operation will be written as $Z = X \diamond_{(\cdot,\cdot)} Y$. It is based on the omega-flip network, which concatenates `omega` *stages* with `flip` *stages*. In an `omega` or a `flip` stage, w input bits are divided into $w/2$ pairs. The two bits in a pair are mapped to two output positions, the destination order being determined by a single control bit. Consequently $w/2$ control bits are needed for $w/2$ data pairs in an `omega` or a `flip` stage.

At the input of an `omega` stage, bits i and $(i + w/2)$, $0 \le i < w/2$, form a pair and they are mapped to the two bit positions $2i$ and $(2i + 1)$. At the input of a `flip` stage, bits $2i$ and $(2i + 1)$, $0 \le i < w/2$, form a pair which is mapped to positions i and $i + w/2$. Clearly, a `flip` stage can be viewed as the inverse of an `omega` stage. The OMFLIP operation $Z = X \diamond_{(a_0,a_1)} Y$ uses two stages in an omega-flip network to permute the data bits X with Y specifying the control bits for the two stages. The subscript (a_0, a_1) represents a two-bit encoding (with `omega` being represented by 0 and `flip` by 1) that specifies which stages are used; they could be (`omega`, `omega`), (`flip`, `flip`), (`omega`, `flip`), or (`flip`, `omega`). Fig. I.2 shows an 16-bit omega-flip network that has two `omega` stages and two `flip` stages. It can be used to perform 16-bit OMFLIP operations, which may select any two stages and pass through the other two. Actually, each stage in the figure has pass-through paths, allowing bits to go through the stage without any position changes. The pass-through paths are not shown in the figure for illustrating better the essential paths in an `omega` or a `flip` stage. The programmatic definition of OMFLIP is given below. Let $X = x_{w-1} \ldots x_0$, $Y = y_{w-1} \ldots y_0$, and $Z = X \diamond_{(a_0,a_1)} Y = z_{w-1} \ldots z_0$ be w-bit words.

OMFLIP $\diamond_{(a_0,a_1)}$

```
j = 0;
for (i = 0;  i < 2;  i = i + 1)
```
$\quad$ `if` $(a_i = 0)\{$
$\qquad$ `for` $(j = 0;\ j < \frac{w}{2};\ j = j + 1)$
$\qquad\quad z_{2j} = x_j;$
$\qquad\quad z_{2j+1} = x_{j+\frac{w}{2}};$
$\qquad\quad$ `if` $(y_{j+\frac{iw}{2}} = 1)$
$\qquad\qquad$ `swap`$(z_{2j},\ z_{2j+1});$
$\quad \}$ `else` $\{$
$\qquad$ `for` $(j = 0;\ j < \frac{w}{2};\ j = j + 1)$
$\qquad\quad z_j = x_{2j};$
$\qquad\quad z_{j+w/2} = x_{2j+1};$
$\qquad\quad$ `if` $(y_{j+\frac{iw}{2}} = 1)$
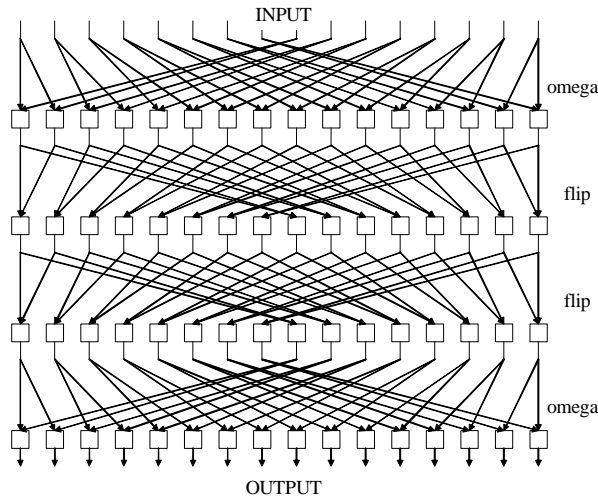$\qquad\qquad$ `swap`$(z_j,\ z_{j+\frac{w}{2}});\ \}$

Figure I.2. A 16-bit omega-flip network

I.4.3 Basic properties of GRP and OMFLIP

GRP can be used to simulate any bit permutation of a w-bit word with at most $\lg(w)$ steps [23]. It can also be used for multi-bit subword permutations and is useful for multimedia processing. It can achieve any one of $m!$ permutations of m subwords in at most $\lg(m)$ instructions, where m is the number of subwords. Here $m = w/k$, where w is the number of bits in a word, and k is the number of bits in a multi-bit subword. In addition, GRP is very useful for accelerating sorting algorithms, and can achieve a speedup of 10 or more when sorting a small set of integers [24].

OMFLIP has similar properties to GRP in terms of performing permutations of bits or multi-bit subwords that are stored in one word (or register). It can perform an arbitrary permutation of w bits with at most $\lg(w)$ steps and an arbitrary permutation of m multi-bit subwords with at most $\lg(m)$ steps. Any one of the $w!$ permutations can be achieved by simulating a full omega-flip network, which consists of $\lg(w)$ `omega` stages followed by $\lg(w)$ `flip` stages. Since an OMFLIP instruction performs the operation of two of these stages, a sequence of $\lg(w)$ OMFLIP instructions can achieve any arbitrary w-bit permutations.

Both GRP and OMFLIP are general-purpose permutation primitives useful in multimedia and security applications; hence, they satisfy Goal 1.

I.4.4 Implementation of GRP and OMFLIP

Both GRP and OMFLIP are easy to add to a typical processor since each requires reading only two source registers and writing one result register. This fits typical processor datapaths, instruction formats, and pipeline organizations. Other implementation issues like execution latency and size of the functional unit required are discussed below.

A hardware implementation of GRP given in [25] suggests that it takes slightly longer than a typical ALU (Arithmetic Logical Unit) latency. Since the latter is often

7

used to determine the cycle time of a processor, this means that a GRP operation will execute in one or two cycles, depending on the aggressiveness of the processor cycle time in the design with respect to the latency of the ALU. When implemented in a processor, the GRP functional unit may also be used to perform some other operations such as DDR. In a processor design where a GRP operation takes two cycles to complete, the GRP functional unit can easily be pipelined, if desired, so that a new GRP instruction can start every cycle. While the functional unit implementing a GRP operation is more complicated than an ALU, it is simpler than that needed for a MULTIPLY operation. On some processors such as Itanium [5, 8], the multiplications of large integers are intended to be performed with floating-point units, by first transferring the operands to floating-point registers, performing the multiplication, and transferring the result back. Hence, the cost of the MULTIPLY operation becomes even higher when it is mixed with other operations that are performed with integer units. Furthermore, a GRP operation takes only 1-2 cycles of execution latency compared to the 3-7 cycles needed by a MULTIPLY operation.

A hardware implementation of an OMFLIP instruction is much simpler than for GRP, and also simpler than for an ALU. An OMFLIP instruction will have a latency no longer than a typical ALU, and hence it can execute in a single cycle. Since the number of stages in an OMFLIP functional unit is fixed no matter how big w is, the size and latency advantages of the OMFLIP functional unit over the GRP functional unit increases as the number of bits, w, to be permuted increases.

OMFLIP definitely satisfies Goal 2 in terms of ease of implementation. GRP's implementation complexity is higher, but it has a latency much smaller than that of a MULTIPLY operation, with a smaller functional unit size. Indeed, GRP may be a simpler alternative than MULTIPLY for cryptography purposes. Hence, Goal 2 is reasonably well satisfied for both GRP and OMFLIP. In the next section, we show that GRP has better cryptographic properties than OMFLIP.

I.5   Cryptographic Properties of Permutations

We now discuss the cryptographic properties of permutation operations in the context of cipher design and analysis, and the satisfaction of Goal 3. We first give a brief overview of cryptographic algorithms and the role of bitwise permutations as a contribution to their security.

It is typical to classify cryptographic algorithms according to the way they use key information [15]. Public key algorithms use two keys; one is kept secret and the other—as the name implies—is made public. Such algorithms are not our concern here. Other algorithms require that the two participants in a cryptographic exchange share the same secret key. Encryption is provided by *block ciphers* and *stream ciphers* and authentication based on secret key techniques can be provided by *message authentication codes*. Finally, a class of algorithms known as *hash functions* are entirely keyless.

While public key algorithms are based on difficult problems in number theory and have a rich mathematical structure, secret-key algorithms and hash functions tend to be more *ad hoc* in design. The process to establish the new AES [17] was

notable for the wealth of new design and analysis techniques that were discussed at great length. The fields of stream ciphers, message authentication codes, and hash functions have not had comparable exposure, though many of the same design principles can often be applied in one way or another.

Indeed, the basic ideas of confusion and diffusion [22] that are so prominent in block cipher designs also appear elsewhere. Confusion might be viewed as a process by which small amounts of complex interaction are introduced locally, while diffusion can be viewed as the process by which this complexity is spread from being solely a local phenomenon. By alternating primitive functions that provide confusion and diffusion, the hope is that the final algorithm will exhibit globally complex, and cryptographically strong, behavior.

The common way to provide the diffusive elements of this process is to use a bitwise permutation, and the success of a cipher design can depend in a fundamental way on the properties of the permutation that is used.

I.5.1   GRP and OMFLIP as cryptographic primitives

There are many different ways of using a bitwise permutation in a cipher design. Frequently the permutation is fixed, as is the case in DES [16], and so it is straightforward to account for the behavior of the permutation in analysis. However, some recent designs have introduced the possibility of using a permutation that is variable and depends on the value of the data being encrypted. We have already mentioned one good example of this, the data-dependent rotation DDR. The operations we consider here, GRP and OMFLIP, might be viewed as being complementary to the DDR operation. With this in mind, we consider the role of these permutations in relation to some specific attacks on block ciphers. More particularly, we will consider their effect on two important kinds of block cipher attacks; *differential cryptanalysis* [1] and *linear cryptanalysis* [14].

Differential and linear cryptanalysis For differential cryptanalysis, the basic idea is that two plaintexts are chosen with a certain *difference* between them; the difference is typically measured by exclusive-or but for some ciphers an alternative measure can be more useful. These two plaintexts are enciphered to give two ciphertexts, and it is hoped that the difference between the outputs has a specific value with a better-than-average probability. Depending on the cipher and the analysis, the behavior of such differences and their evolution can be useful in deriving certain bits of the key. For linear cryptanalysis, the basic idea is to find relations among certain bits of plaintext, ciphertext, and the key that hold with a probability $p \neq 1/2$ (i.e., there is a bias of $|p-1/2| > 0$). Such a relation is called a linear approximation. As in differential cryptanalysis, we seek to exploit such non-ideal behavior and it may be possible to identify linear approximations that reveal information about the key.

An important feature that determines the possible success of differential and linear cryptanalysis is the speed with which the complexity of a difference or linear approximation increases as we try and keep track of such close relations during the

encryption process. For a good block cipher, the differences between related texts and the relation between bits of the same text should both become very complicated very quickly so that by the time the encryption process is concluded any statistical variations are smoothed out and there is no unusual behavior left for the cryptanalyst to exploit. The process by which this is achieved is often loosely referred to as the *avalanche of change* and the spread of change and the spread of effect and influence is often influenced by the role of permutations within the cipher design.

Differential and linear properties Here we consider the differential and linear properties of GRP, OMFLIP, and DDR. There are many differential characteristics and linear approximations for a given permutation operation, each holding with different associated probabilities. The most useful ones are typically those that are both simple and which hold with relatively large probabilities. The properties on DDR are mostly results that can be found in [4, 9], while the properties for GRP and OMFLIP are new results. The results in this chapter are necessarily preliminary results and concentrate on some of the simplest forms of cryptanalysis. In Section I.5.3 we take account of some more advanced considerations.

For differential cryptanalysis, we need to consider a pair of inputs and their corresponding output. Specifically, for $i = 1, 2$, let $Z_i = X_i \bullet Y_i$. We define the differences in the input and output to be $\Delta_X = X_1 \oplus X_2$, and $\Delta_Y = Y_1 \oplus Y_2$, and $\Delta_Z = Z_1 \oplus Z_2$. A differential characteristic of the permutation operation $Z = X \bullet Y$ is a triplet $(\Delta_X, \Delta_Y) \to \Delta_Z$, together with the probability p that the given triplet holds when the inputs are chosen at random. We let e_s denote the w-bit word which is zero except for a single one in bit position s. In our preliminary investigation, we will restrict our attention to single-bit differences and approximations.

The following differential characteristics of a permutation operation are often useful (we use Δ to denote a general difference which may be zero). The aim is to keep track of any changes induced during encryption and to keep the evolution of differences as simple as possible.

$$(A) \qquad (e_s, 0) \to e_t$$
$$(B) \qquad (0, e_t) \to \Delta$$
$$(C) \qquad (e_s, e_t) \to \Delta$$

Since Z is a permutation of the bits in X, we know that type (A) characteristics exist and their probabilities are easy to compute. The more interesting characteristics are type (B) and type (C) which depend on the input difference in the control bits Y. For these, we will compare the diffusion effect by computing the expected Hamming weight of the output difference Δ_Z. The three types of characteristics of different permutation operations and their associated probabilities or Hamming weights are shown in Table I.1. There, $E(\text{Hwt}(\Delta))$ denotes the expected value of $\text{Hwt}(\Delta)$, the Hamming weight of Δ, when inputs are chosen at random.

In linear cryptanalysis, we aim to exploit a linear relation among certain bits of the inputs and outputs. Specifically, if Γ and X are two binary vectors of length w, then their inner product, denoted by $\Gamma \cdot X$, is the parity of the bits in X specified

Table I.1. The propagation of differences across DDR, GRP, and OMFLIP

	Type (A) $(e_s, 0) \to e_t$	Type (B) $(0, e_t) \to \Delta$	Type (C) $(e_s, e_t) \to \Delta$
DDR	$p = \frac{1}{w} \; \forall s, t$	$\text{Hwt}(\Delta) = 0$ $\lg(w) \le t$ $E(\text{Hwt}(\Delta)) = \frac{w}{2}$ $0 \le t \le \lg(w) - 1$	$E(\text{Hwt}(\Delta)) = 1$ $\lg(w) \le t$ $E(\text{Hwt}(\Delta)) = \frac{w}{2}$ $0 \le t \le \lg(w) - 1$
GRP	$p \le (\frac{1}{2} + \frac{1}{2^w}) \; \forall s, t$	$E(\text{Hwt}(\Delta)) = \frac{w}{4} \; \forall s, t$	$E(\text{Hwt}(\Delta)) = \frac{w}{4} \; \forall s, t$
OMFLIP	$p \le \frac{1}{4} \; \forall s, t$	$\text{Hwt}(\Delta) \le 2 \; \forall s, t$	$\text{Hwt}(\Delta) \le 3 \; \forall s, t$

Table I.2. The propagation of linear approximations across DDR, GRP, and OM-FLIP

	Type (L) $(e_s, 0, e_t)$	Type (M) (e_s, e_u, e_t)
DDR	$|b| \le 1/(2w)$ Max. with $s = t = 0$	$|b| \le 1/(2w)$ Max. with $s = t = 0$
GRP	$|b| \le (1/4 + 1/2^{w+1})$ Max. with $s = t = 0$	$|b| \le (1/4 - 1/2^{w+1})$ Max. with $s = t = 0$
OMFLIP	$|b| \le 1/8$ Max. with $s = t = 0$	$|b| \le 1/8$ Max. with $s = t = 0$

by the non-zero entries in Γ. A linear approximation of the permutation $Z = X \bullet Y$ is therefore a triplet $(\Gamma_X, \Gamma_Y, \Gamma_Z)$ together with the probability p that the equation $(\Gamma_X \cdot X) \oplus (\Gamma_Y \cdot Y) = (\Gamma_Z \cdot Z)$ holds on random inputs. The bias b of the linear approximation is defined to be $|p - 1/2|$. For example, $(\Gamma_X, \Gamma_Y, \Gamma_Z) = (2^w - 1, 0, 2^w - 1)$ is a linear approximation that holds with probability $p = 1$ for any permutation operation, since the parity of all the bits in Z is always equal to the parity of all the bits in X; this approximation has a bias $b = 1/2$. We will consider restricted forms to the linear approximations, depending on whether any control bits Y are involved in the approximation. When Y is not involved, the simplest approximation takes the form of $(e_s, 0, e_t)$. This will be denoted type (L) and intuitively, the bias of such a linear approximation measures how uniformly the permutation moves the bits around (e.g. whether there is a bit position that tends to be fixed). When Y is involved, the simplest approximation, denoted with type (M), takes the form of (e_s, e_u, e_t). The bias of these approximations measures if the destination position of a bit in X highly depends on a single bit in Y. Ideally, the destination position of a bit in X depends on many bits in Y, and these bits are equally important to determining the position. The maximum bias for these approximations are listed in Table I.2.

I.5.2 Comparison between DDR, GRP, and OMFLIP

Even though all three permutations might appear to be doing the same thing—i.e. shifting bits—we see that the cryptographic properties can be very different. In

Table I.1 we see that GRP has differential properties that are generally similar to DDR, but suggest a better diffusive effect when there is a difference in any bit of the control word; i.e. for differentials of types (B) and (C). This might be interesting, since one potential weakness for DDR is that there is no bit-level diffusive effect when there is no difference in the lower $\lg(w)$ bits of the control word. This might be exploited by the cryptanalyst as we will see in Section I.6.1. Unfortunately the differential properties of OMFLIP for these simple characteristics are not too good; in all cases the diffusive effect is very limited. Turning to linear approximations, we see that for both GRP and OMFLIP, the maximum bias is quite large compared to that achieved with DDR when the word size w is sufficient large. Taken together these results suggest that OMFLIP is unlikely to bring any additional advantages over those provided by GRP and DDR; that GRP will perhaps not be particularly resistant to linear cryptanalysis on its own; but that GRP might complement DDR by providing additional resistance to differential cryptanalysis in the areas where diffusion using DDR might be controlled by an adversary. We will examine this combination of DDR and GRP in Section I.6.2.

I.5.3   Additional considerations

We have to caution the reader that the results presented in Section I.5.1 are basic results. They merely provide some evidence that one permutation might be better than another. It is quite natural to focus on single bit differences and approximations since they are typically the ones that are easier to handle in a cryptanalytic attack. However, when we introduce a new primitive operation we need to consider other issues as well.

As an example of this, we might consider two-bit differences and their propagation across the GRP permutation. We consider two triplets (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) with the following form, where we use $\{-b-\}$ to denote a $(w-2)$-bit string of some unknown value, and $\{-0-\}$ to denote a $(w-2)$-bit string of zeros. Let $X_1 = \{10 - b-\}$, $Y_1 = \{01 - c-\}$, $X_2 = \{01 - b-\}$, and $Y_2 = \{10 - c-\}$. Then $\Delta_X = \{11 - 0-\}$ and $\Delta_Y = \{11 - 0-\}$, yet $\Delta_Z = \{00 - 0-\}$. We have two two-bit input differences effectively producing the same output! This is quite an unusual effect, and additional analysis is required to fully appreciate the consequences of such bit-level interactions[I.5].

Another interesting consideration is the distribution of the permutations generated by these operations since there are some interesting links here with the shuffling of a deck of cards [7]. While DDR can only be used to generate a small fraction of bit-wise permutations, all of the resultant permutations are equally likely. When we turn to the GRP operation, however, while all permutations can conceivably be generated, in a single GRP operation there is a slight bias to the generation of the identity permutation; the probability for the identity permutation is $w/2^w$ for w bits while that for other permutations is $1/2^w$. The implications of this for the

---

[I.5]Preliminary study suggests that it could be difficult to use such difference propagations, but no general statements can be made in this regard.

suitability of GRP is unclear, but it suggests that a cautious approach needs to be taken.

On a more positive note, it is important to note there are also some constructive properties of permutations such as GRP that we have not explored. For instance, we have not considered the ability of GRP to change the "neighborhood" of bits in achieving any one of w permutations. Such properties may provide some additional cryptographic and architectural advantages when compared to the DDR permutation.

I.6 AN ILLUSTRATIVE EXAMPLE FOR CIPHER DESIGN

Analysis in Section I.5.1 demonstrated certain interesting properties of GRP. First, GRP uses all w bits of the control word, rather than only $\lg(w)$ bits as in DDR. Second, GRP appears to have properties that are complementary to DDR in terms of differential attacks; a difference in any bit of the control word should produce a large difference in the output. In this section, we will explore whether we can take advantage of these properties.

I.6.1 The block cipher RC5

When considering the possible impact of DDR and other permutations in cryptographic algorithms, a natural starting point is the block cipher RC5 [20]. This was designed to be extremely simple and this means that the effect of introducing DDR can be reasonably well measured. We give a very brief description of RC5. The initial secret key is used to generate a set of round keys $S[\cdot]$ that will be used in encryption. The $2w$-bit input to RC5 is divided into two words L_0 and R_0, each w bits long. The encryption process consists of $2r$ iterations of a simple round function. Each iteration is called a "half-round" and two iterations form a full round in RC5. The $2w$-bit ciphertext output from RC5 is given by $L_{2r}||R_{2r}$.

RC5 Encryption

$L_1 = L_0 + S[0]$;
$R_1 = R_0 + S[1]$;
`for` $(i = 2; \ i \le 2r; \ i = i + 1)$ {
    $L_i = R_{i-1}$
    $R_i = ((L_{i-1} \oplus R_{i-1}) \lll R_{i-1}) + S[i]$ }

Since its publication, RC5 has come under considerable scrutiny [2, 9, 10, 21] especially with regards to its extensive use of DDR. While no practical attack on RC5 has been found, studies provide some interesting theoretical attacks, mostly based on the fact that the "rotation amounts" used in the DDR will not depend on all bits of the control word. Therefore, it is interesting to consider whether the GRP operation might be used to complement the DDR operation that is already used in RC5. In [9] three types of single-bit characteristics are used to form a three-half-round characteristic that can be iterated for as many rounds as needed. In [10] these

Table I.3. Single-bit characteristics for GRP

Char.	Prob.	Prob. when $w = 32$
$(e_s, 0) \rightarrow e_t$	$p \leq 1/2$	2^{-1}
$(e_s, e_s) \rightarrow e_s$	$p = 1/2^{w-1}$	2^{-31}
$(0, e_s) \rightarrow 0$	$p = \frac{3^{w-1}}{2^{2w-2}}$	2^{-12}

characteristics were used in a more general way while [2] considered more general characteristics. However, all this work on RC5 helped to motivate the choice of differential characteristics that were studied in Section I.5.1.

I.6.2 A role for GRP in an RC5-variant

There are many possible ways to incorporate GRP into the round function of RC5. As a motivational example, we have chosen a way that incurs a minimal change to the original round function. This might make it easier to leverage the existing security analysis of RC5. We propose the following straw-man proposal for a round function for a RC5 variant that we refer to as RC5-GRP.

RC5-GRP Encryption

$L_1 = L_0 + S[0]$;
$R_1 = R_0 + S[1]$;
`for` $(i = 2; \ i \leq 2r; \ i = i + 1)$ {
$\qquad L_i = R_{i-1}$
$\qquad T = ((L_{i-1} \oplus R_{i-1}) \lll R_{i-1}) + S[i]$
$\qquad R_i = T \triangleleft R_{i-1}$ }

The round function of RC5-GRP is the same as that of RC5 except that the new operation GRP is performed at the end of the round, updating the value of R_i (again) using R_{i-1}. Thus the variable R_{i-1} that controls DDR is also used to control GRP. In [9], three single-bit characteristics—types (A), (B), and (C) from Table I.1—were used in the differential attack on RC5. When analyzing the security of RC5-GRP, we still use these three characteristics for DDR. In order to form an iterative characteristic across three half-rounds as in [9], a specific characteristic for GRP is needed to follow each of the characteristics for DDR. These characteristics for GRP are summarized in Table I.3. (One can see that these are special cases of the characteristics for GRP from Table I.1.)

When $w = 32$, the total differential probability of the three characteristics for GRP in Table I.3 is $2^{-1-31-12} = 2^{-44}$. It appears that adding GRP could have a significant effect on a specific class of one-bit differential characteristics. However, by considering the more sophisticated two-bit characteristics in Section I.5.3, it seems we might need to be more cautious and there appears to be a two-bit differential characteristic over two half-rounds of RC5-GRP that holds with probability around

2^{-16}. The resistance of RC5 to linear cryptanalysis is likely to be inherited by RC5-GRP, but with regards to more advanced differential attacks the full extent of any increased resistance still needs to be quantified.

According to [2] RC5 requires 18 rounds to be secure against advanced differential cryptanalysis. Based on our analysis on the reduction in differential probabilities for GRP, it would be interesting to know whether ten rounds (twenty half-rounds) of RC5-GRP would offer sufficient security. If this were the case, we might provide the following performance comparison between RC5 and RC5-GRP. RC5 has four basic operations in each half-round, while RC5-GRP has five. We will assume that all the operations are well-supported and that each operation (including the GRP operation implemented in processor hardware) takes one cycle. In this case, the total execution cycles for RC5 will be $(18 \times 2 \times 4) + 4 = 148$ cycles. The cycles required for RC5-GRP would be $(10 \times 2 \times 5) + 4 = 104$ cycles. Hence, for equivalent security, RC5-GRP would be faster than RC5. Also, since RC5-GRP requires only 66% of the computation cycles required for RC5, this will result in a significant reduction in energy consumption, prolonging the battery life of secure mobile devices. While this is almost certainly not the final word in the analysis of RC5-GRP, it does illustrate our larger point that low-level support of bit-level permutations might lead to simple enhancements of existing algorithms and the design of more efficient ciphers.

I.7 CONCLUSION

In this chapter, we proposed the study of new computer processor features that might have interesting cipher design implications. As a first step, we analyzed bit-level permutation operations and presented new results on the characterization of the permutation operations GRP and OMFLIP. We began to explore the cryptographic potential for the low-level support of bit-level permutations, and provided some basic initial analysis. This suggests that other proposals in the future may lead the way to increased performance and reduced energy consumption, an aspect of algorithm design that is increasingly important for battery-powered hand-held devices and sensors. However there remain significant open problems for future work. Some are specific to the particular permutation operations we have considered here, others are of a more general nature. However, we hope that the results and ideas in this chapter serve as an initial step in establishing a continuing dialog between the computer architecture and the cryptographic communities. This may lead to architectural and algorithmic innovations that would be immensely useful, not just for cryptographic applications, but in supporting the increasingly rapid evolution to pervasive networks and ubiquitous computing.

Bibliography

[1] E. Biham and A. Shamir. Differential cryptanalysis of the data encryption standard. In *Proceedings of Eurocrypt '98, LNCS(1403)*, pp. 85–99. Springer-Verlag, January 1998.

[2] A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. In *Proceedings of Eurocrypt '98, LNCS(1403)*, pp. 85–99. Springer-Verlag, 1998.

[3] E. Brier, H. Handschuh, and C. Tymen. Fast Primitives for Internal Data Scrambling in Tamper Resistant Hardware. In *Proceedings of CHES 2001, LNCS(2162)*, pp. 16–28. Springer-Verlag, 2001.

[4] S. Contini and Y. L. Yin. On Differential Properties of Data-Dependent Rotations and their use in MARS and RC6. In *Proceedings of Second AES Conference*, 2000.

[5] M. Cornea, J. Harrison, and P. T. Tang. *Scientific Computing on Itanium-based Systems*. Intel Press, 2002.

[6] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales. AltiVec Extension to PowerPC Accelerates Media Processing. *IEEE Micro*, 20(2):85–95, April 2000.

[7] C. Grinstead and J. Snell. *Introduction to Probability*. American Mathematical Society, Providence, Rhode Island, 1994.

[8] Intel Corporation. *IA-64 Application Developers Architecture Guide*. Intel Press, May 1999.

[9] B. Kaliski and Y. L. Yin. On differential and linear cryptanalysis of RC5. In *Advances in Cryptology – CRYPTO'95, LNCS(963)*, pp. 171–184. Springer-Verlag, 1995.

[10] L. R. Knudsen and W. Meier. Improved differential attacks on RC5. In *Advances in Cryptology – CRYPTO'96, LNCS(1109)*, pp. 216–228. Springer-Verlag, 1996.

[11] R. B. Lee. Accelerating multimedia with enhanced microprocessors. *IEEE Micro*, 15(2):22–32, April 1995.

[12] R. B. Lee. Subword parallelism in MAX-2. *IEEE Micro*, 16(4):51–59, August 1996.

[13] R. B. Lee, Z. Shi, and X. Yang. Efficient permutation instructions for fast software cryptography. *IEEE Micro*, 21(6):56–69, December 2001.

[14] M. Matsui. First experimental cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology – CRYPTO'94, LNCS(839)*, pp. 1–11. 1994.

[15] A. Menezes, P. van Oorschot, and S. Vanstone. *The Handbook of Applied Cryptography*. CRC Press, 1996.

[16] National Institute of Standard and Technology. *Data Encryption Standard (DES)*. FIPS 46-2, December 1993.

[17] National Institute of Standard and Technology. *Advanced Encryption Standard (AES)*. FIPS 197, November 2001.

[18] S. Obeman, G. Favor, and F. Weber. AMD 3Dnow! Technology: Architecture and Implementations. *IEEE Micro*, 19(2):37–48, April 1999.

[19] A. Peleg and U. Weiser. MMX Technology Extension to the Intel Architecture. *IEEE Micro*, 16(4):10–20, August 1996.

[20] R. L. Rivest. The RC5 encryption algorithm. In *Proceedings of Fast Software Encryption, LNCS(1008)*, pp. 86–96. Springer-Verlag, 1995.

[21] A. A. Selcuk. New results in Linear Cryptanalysis of RC5. In *Proceedings of the 5th Workshop on Fast Software Encryption, LNCS (1372)*, pp. 1–16. 1998.

[22] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[23] Z. Shi and R. B. Lee. Bit permutation instructions for accelerating software cryptography. In *Proceedings of the 11th International Conference on Application-Specific Systems, Architectures and Processors*, pp. 138–148, July 2000.

[24] Z. Shi and R. B. Lee. Subword sorting with versatile permutation instructions. In *Proceedings of the International Conference on Computer Design (ICCD 2002)*, pp. 234–241, September 2002.

[25] Z. Shi and R. B. Lee. Implementation complexity of bit permutation instructions. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, November 2003.

[26] Z. Shi, X. Yang, and R. B. Lee. Arbitrary bit permutations in one or two cycles. In *Proceedings of the 14th International Conference on Application-Specific Systems, Architectures and Processors*, pp. 237–247, June 2003.

[27] X. Yang and R. B. Lee. Fast subword permutation instructions using omega and flip network stages. In *Proceedings of the International Conference on Computer Design (ICCD 2000)*, pp. 15–22, September 2000.