Symmetric Block Ciphers Based on Group Bases

Valér Čanda¹, Tran van Trung¹, Spyros Magliveras², and Tamás Horváth³
¹ Institute for Experimental Mathematics, University of Essen, Elberstrasse 29, 45326 Essen, Germany
valer.trung@exp-math.uni-essen.de
² Department of Mathematical Sciences, Florida Atlantic University
777 Glades Rd, Boca Raton, FL 33431, U.S.A.
spyros@fau.edu
³ Secunet AG, Steubenstrasse 53, 45138 Essen, Germany
horvath@secunet.de

Abstract
We introduce a new family of symmetric block ciphers based on group bases. The main advantage of our approach is its full scalability. It enables us to construct, for instance, a trivial 8-bit Caesar cipher as well as a strong 256-bit cipher with 512-bit key, both from the same specification. We discuss the practical aspects of the design, especially the choice of carrier groups, generation of random group bases and an efficient factorization algorithm. We also describe how the cryptographic properties of the system are optimized, and analyze the influence of parameters on its security. Finally we present some experimental results regarding the speed and security of concrete ciphers from the family.

Keywords: symmetric cipher, block cipher, scalability, security, effectiveness, binary group, group basis.

1 Introduction
A good block cipher should possess several properties. In addition to security and efficiency, which are essential, there are other important attributes like generality, scalability and theoretical foundations. In what follows we discuss these properties in more detail.

A block cipher can be characterized by two basic parameters: the block length \( n \) and the key length \( k \), both expressed as a number of bits. For each of the \( 2^k \) possible keys, the cipher defines a bijective mapping between the \( 2^n \) plaintext blocks and the \( 2^n \) ciphertext blocks. As the plaintext and ciphertext spaces are usually the same, we can view an \( n \)-bit block cipher as defining a permutation on a set of \( 2^n \) elements for each possible key. A simple key-indexed lookup table containing all 64-bit numbers in random order would implement a very strong 64-bit block cipher, without any additional algorithm. Unfortunately, such an implementation would take so much memory, that it would not be applicable for any practical use. Almost all modern block ciphers simulate such a large lookup random table using smaller tables (S-Boxes) in combination with other transformations. The goal is to make the dependence between the plaintext, ciphertext and key so complex that it is virtually indistinguishable from the random case.

As \( 2^n \) elements can be permuted in \( 2^n! \) different ways, a “perfect” \( n \)-bit block cipher could accept keys of length up to \( \lfloor \log_2(2^n!) \rfloor \) bits. This means a 1683-bit key for \( n = 8 \) and approximately a \( 10^{21} \)-bit key for \( n = 64 \). Of course, no one needs such long keys and it would be extremely impractical to use
them. These large numbers show, however, the strong potential of block ciphers and the restricted
generality of current systems which use by design a fixed key-length or fixed S-boxes. These ciphers are
in our opinion not flexible enough. They are constrained to one specific configuration and the functions
defined by them might be much too far from ideal random permutations. Cryptanalytic attacks such
as differential or linear cryptanalysis attempt to exploit any biases or regularities inherent in these
functions.

Another frequent drawback of block ciphers is a small or totally missing scalability. Because of
the unprecedented growth rate of computer power available to the public, it is highly desirable to
have choices for some basic parameters of the cipher. If our ciphers were fully scalable, we could just
adapt the values of these parameters, when some new, amazing breakthrough in processor or memory
technology occurs. The values of the parameters $n$ and $k$ could be easily changed without a complete
redesign of the cipher and we would not be forced to throw away all research on the properties of the
cipher, starting again from “ground zero”.

For example, the block length of DES is 64 bits. If we wish to create a 128-bit version of DES,
we would have to design new, larger S-Boxes. As the design of good S-Boxes is by far a non-trivial
task [1], the properties of the new DES could be quite different. This, in fact, would be a totally new
cipher.

Another cipher - IDEA [2] - has a very plain structure and its block length might be doubled
simply by increasing the length of each of the four subblocks from 16 to 32 bits. The fact, however,
that $2^{16} + 1$ is a prime number is essential to the functionality of IDEA. Since $2^{32} + 1$ is not a prime,
the double version would not work well.

Each new cipher should be studied extensively, perhaps for several years, before it is deemed
trustworthy and is presented for widespread use. If the cipher is based on a strong theoretical foun-
dation, we can gain a better understanding of possible failures, cryptanalytic attacks, etc., and we
have stronger tools with which to analyze the new algorithm. Therefore, a cipher based on strong
mathematical foundations will either be rejected outright, or if it appears workable, there should be a
reasonable chance for it to have provable reliability and trustworthiness.

All in all, we think that an ideal cipher should not only be secure and fast but also theoretically
well-founded, general, and scalable. In this paper we present a new family of fully scalable block

ciphers which is quite general. This approach enables the construction of a range of ciphers, from a
tiny toy cipher to a large, secure one. The idea on which the encryption is based is a mapping of group

elements between two random group bases. A subject which does not know the two secret bases is
not able to recover the mapping. We discuss the selection of suitable carrier groups, the generation of
random group bases which enable an efficient factorization and the optimization of the cryptographic
properties of the system. Finally, we discuss the security, speed and memory requirements of a concrete
software implementation.

2 The Principle of Encryption

The ciphers we propose utilize group theory [3], [4]. Although we focus our attention on permutation
groups, it is also possible to construct a cryptosystem based on any carrier group in other representation
forms. By a permutation of $n$ symbols we understand a bijective function $p : Z_n \rightarrow Z_n$. The
Cartesian representation of $p$ is the vector $[p(0), p(1), \ldots, p(n-1)]$. The basic notion needed for the
ciphers proposed in this paper is the idea of a Group Basis.

**Definition 2.1 Group Basis**

Let $G$ be a finite group. A group basis for $G$ is an ordered collection $\beta = (B_0, B_1, \ldots, B_{w-1})$ of ordered
subsets \( B_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,r_i-1}) \) of \( G \) such that each element \( p \in G \) can be expressed uniquely as a product of the form:

\[
p = b_{0,x_0} \cdot b_{1,x_1} \cdots b_{w-1,x_{w-1}}, \quad b_{i,x_i} \in B_i
\]

The \( B_i \) are called the blocks of \( \beta \), the vector of block lengths \( r = (r_0, r_1, \ldots, r_{w-1}) \) is called the type of \( \beta \) and the number \( w \) the dimension of \( \beta \). Each \( p \in G \) corresponds to a unique index vector \( x = (x_0, x_1, \ldots, x_{w-1}) \), where \( x_i \in \mathbb{Z}_{r_i} \). The space of all index vectors is \( X = \mathbb{Z}_{r_0} \times \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_{w-1}} \). The index set \( X \) has cardinality \( |X| = r_0 \cdot r_1 \cdots r_{w-1} = |G| \).

A basis \( \beta \) describes a bijective mapping \( \tilde{\beta} : X \to G \) as follows:

\[
\tilde{\beta}(x) = \tilde{\beta}(x_0, x_1, \ldots, x_{w-1}) = b_{0,x_0} \cdot b_{1,x_1} \cdots b_{w-1,x_{w-1}} = p.
\]

When computing \( p = \tilde{\beta}(x) \) we say that \( p \) is composed from factors \( b_{i,x_i} \). Computing the inverse function \( x = \tilde{\beta}^{-1}(p) \) is called factorizing \( p \) with respect to \( \beta \). For more discussion of group bases see [5] and [6].

For the rest of this paper our groups will be permutation groups, i.e. subgroups of the full symmetric group \( S_n \) of some degree \( n \).
Example 2.1 A Group Basis for $G = S_3$
$G = \{[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]\}$

\[
\begin{array}{c|c}
& \beta \\
B_1 & \begin{bmatrix} [0, 2, 1] & b_{1,1} \\
[0, 1, 2] & b_{1,0} \\
\end{bmatrix} \\
B_0 & \begin{bmatrix} [1, 2, 0] & b_{0,2} \\
[2, 0, 1] & b_{0,1} \\
[0, 1, 2] & b_{0,0} \\
\end{bmatrix}
\end{array}
\]

$w = 2, r = (3, 2)$,
$p = [1, 0, 2] = [2, 0, 1] * [0, 2, 1] = b_{0,1} * b_{1,1}$,
$\beta(p) = (1, 1) = e$

One can think of a group basis as a kind of \(w\)-dimensional discrete coordinate system as illustrated on the Figure 2.1. The six permutations of $G$ might be seen as points in a 2-dimensional space. Any one of the six points can be expressed as a unique sum\(^1\) of two points, one from each axis. The two axes, the first with three and the second with two points, correspond to the two blocks of $\beta$.

The crucial property of group bases from the cryptographic point of view is that there is an enormous number of different group bases for a given group. We denote the set of all bases that generate $G$ by $B_G$. For example, the tiny group $S_3$ of our example has 924 different bases.\(6!\) of them are one-dimensional bases of type (6) and\(2!2^3.3!+3!2^2.2!\) of them are two-dimensional bases of types (2,3) and (3,2). Later we will describe how all these bases can be generated. The most basic version of a secret-key cryptosystem based on group bases is defined as follows:

**Definition 2.2** A Block Cipher Based on Group Bases

Let $G$ be a finite group, called the carrier group. Let $\lambda : \mathbb{Z}_{|G|} \rightarrow G$ be any fixed bijective function. The plaintext and ciphertext spaces for the cipher are the same: $P = C = \mathbb{Z}_{|G|}$. The key space is the set $K = B_G \times B_G$.

Let $k = (\beta_1, \beta_2), k \in K$ be a secret key. Let $x \in P$ be a plaintext and $y \in C$ the corresponding ciphertext. The encryption function $e_k : P \rightarrow C$ is defined by the rule

$$y = e_k(x) = \lambda^{-1}(\beta_2(\beta_1^{-1}(\lambda(x))))$$

and the decryption function $d_k : C \rightarrow P$ is defined as

$$x = d_k(y) = \lambda^{-1}(\beta_1(\beta_2^{-1}(\lambda(y))))$$

In other words, we take two random group bases for $G$, $\beta_1$ and $\beta_2$, and each time we want to encrypt some $p \in G$, we have to find such $p' \in G$ which has the same coordinates in $\beta_2$ as $p$ has in $\beta_1$. The function $\lambda$ only defines a unique numbering of the group elements.

Again, for a better visualization, we take a simple example with geometric coordinates (Example 2.2). There are 16 numbered points in the space, thus we can encrypt and decrypt the plaintexts and ciphertexts from $\mathbb{Z}_{16}$. Suppose, we want to encrypt plaintext point 14. First we find the coordinates of the point 14 with respect to basis $\beta_1$. The corresponding index vector is (2,3). Now we compose the point, which has the same coordinates in $\beta_2$, this gives us point 10. Therefore $e_{(\beta_1, \beta_2)}(14) = 10$.

The complete table for $e_k$ is displayed on the right hand side.

---

\(^1\)Addition of points in this discrete geometry is defined by means of vectors as:

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2 \mod 3, y_1 + y_2 \mod 2).$$

Note that while this is a commutative operation, operation $\ast$ in $S_3$ is not. This is only an illustrative example.
Example 2.2 A Mapping of Points Between Two Bases

In a real-world application the dependencies become much more complex. A 64-bit cipher can use for instance two 8-dimensional bases with 256 elements on each “axis”. Moreover, the carrier group will not necessarily be commutative.

3 Implementation Aspects

3.1 The Carrier Group \( G \) and Function \( \lambda \)

The size of the plaintext and ciphertext space depends directly on the order of the carrier group \( G \). We are only interested in groups whose order is a power of two, the so called 2-groups or binary groups. More precisely, we should have \( |G| = 2^n \) for a natural number \( n \), because only ciphers whose blocks fit exactly in \( k \) bytes are interesting for a practical use. Note that \( |S_m| = m! \neq 2^n \) for any \( m > 2 \). Therefore the symmetric group \( S_m \) is not suitable for a carrier group.

3.1.1 Group \( \mathbb{Z}_2^n \)

The simplest available 2-group is the elementary abelian group \( \mathbb{Z}_2^n = \mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \mathbb{Z}_2 \). It contains the permutations of \( 2n \) symbols in form \( p = [a_0, a_1, \ldots, a_{2n-1}] \) where for each pair of symbols \( a_{2k}, a_{2k+1} \), \( k \in \mathbb{Z}_n \) either \( a_{2k} = 2k \) and \( a_{2k+1} = 2k + 1 \), or else \( a_{2k} = 2k + 1 \) and \( a_{2k+1} = 2k \).

The permutations of \( \mathbb{Z}_2^n \) can be represented very efficiently with our so called compact representation. The compact representation of a \( p = [a_0, a_1, \ldots, a_{2n-1}] \), \( p \in \mathbb{Z}_2^n \) is the binary vector \( x = (x_0, x_1, \ldots, x_{n-1}) \) where \( x_i = 0 \) if and only if \( a_{2i} = 2i \). Otherwise \( x_i = 1 \). In other words, the \( i \)-th bit of the compact representation indicates, whether the elements \( a_{2i} \) and \( a_{2i+1} \) have been swapped or not. In terms of memory requirements the compact representation is optimal, as it is impossible to represent the \( 2^n \) elements in less than \( n \) bits. Another benefit of the compact representation is that it makes it possible to multiply permutations very fast. Note that if \( x_1 \) is the compact representation of \( p_1 \) and \( x_2 \) of \( p_2 \), then \( x_1 \text{ XOR } x_2 \) is the compact representation of the product \( p_1 * p_2 \).

The operation \(*\) in \( \mathbb{Z}_2^n \) is commutative and linear. Last but not least, the compact representation of the permutations fulfills the role of the function \( \lambda \) from Definition 2.2. If we consider the vector \((x_0, x_1, \ldots, x_{n-1})\) as a binary representation of a natural number, we have a unique numbering of all permutations in the group.

The group basis for \( \mathbb{Z}_2^n \) of the form \( \alpha = (A_0, \ldots, A_{n-1}) \), where each block \( A_i \) contains two permutations in compact representation, the identity \((0, \ldots, 0)\) and a single swap on the \( i \)-th place \((0, \ldots, 0) \{i\} (0, \ldots, 0)\), is called the canonical basis for \( \mathbb{Z}_2^n \). The one-element set \( c_i = \{i\} \) is called the set of key bit positions for block \( A_i \).
3.1.2 Group $\mathcal{H}_s \times \mathcal{H}_1$

In contrast with $\mathbb{Z}_2^n$, the most complex 2-group is $\mathcal{H}_s$, the largest binary subgroup of $S_n$. When $n = 2^s$, the order of $\mathcal{H}_s$ is $2^{2^s-1}$. $\mathcal{H}_s$ is also known as the Sylow 2-subgroup of $S_{2^n}$.

**Definition 3.1** Sylow 2-subgroup $\mathcal{H}_s$ of the symmetric group $S_n$, $n = 2^s$.

The group $\mathcal{H}_s$ is defined recursively as follows:

- $\mathcal{H}_1 = \mathbb{Z}_2$
- $\mathcal{H}_s = (\mathcal{H}_{s-1} \times \mathcal{H}_{s-1}) \cdot \mathbb{Z}_2$, for $s > 1$.

The permutation representation $\mathcal{T}_s$ of the $\mathbb{Z}_2$ appearing in $\mathcal{H}_s$, contains two permutations of $2^s$ elements, the identity $e$ and the involution $r_s$, which swaps the two halves $\{0, 1, \ldots, 2^{s-1} - 1\}$ with $\{2^{s-1}, \ldots, 2^s - 1\}$, each of length $2^{s-1}$. For example $\mathcal{T}_1 = \{(0, 1), (1, 0)\}$, $\mathcal{T}_2 = \{[0, 1, 2, 3], [2, 3, 0, 1], [3, 2, 0, 1], [3, 2, 1, 0]\}$, etc.

**Example 3.1** $\mathcal{H}_s$ and $\alpha_s$ for $s = 1, 2, 3$.

- $\mathcal{H}_1 = \mathcal{T}_1 = \{(0, 1), (1, 0)\}$

Using the trivial representation $\mathcal{H}_s$, the subgrup $\mathcal{H}_s$ can be rewritten as $\mathcal{H}_s = \mathcal{H}_{s-1} \times \mathcal{H}_{s-1}$.

Each $\mathcal{H}_s$ has a unique **canonical basis** $\alpha_s$ which contains $2^s - 1$ blocks each consisting of two permutations. Each block $A_i$ has one key bit position $c_i = \{i\}$.

<table>
<thead>
<tr>
<th>$\alpha_0$</th>
<th>$\alpha_1$</th>
<th>$\alpha_2$</th>
<th>$\alpha_3$</th>
<th>$\alpha_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Here, $I_s$ is defined as $(0, 1, \ldots, 2^s - 1)$, $\hat{I}_s = 2^s + I_s = (2^s, 2^s + 1, \ldots, 2^{s+1} - 1)$ and $J_s$ denotes a $(2^{s+1} - 2) \times 2^s$ array each row of which is equal to $I_s$. Analogously, $\hat{J}_s = 2^s + J_s$.

**Definition 3.2** The compact representation of elements in $\mathcal{H}_s$.

Let $h$ be a permutation from $\mathcal{H}_s$. The binary vector $\hat{\alpha}_s^{-1}(h) = (x_0, x_1, \ldots, x_{2^w-1})$, $w = 2^s - 1$, is called the **compact representation** of $h$.

Again, the compact representation is optimal in terms of memory requirements. In general, we can say that each $h \in \mathcal{H}_s$ can be uniquely represented by a $(2^s - 1)$-bit binary number. The multiplication
of permutations in $H_s$ is a non-linear and non-commutative operation. It can be performed directly and efficiently in the compact representation [6].

As already mentioned, the preferred order of the carrier group should be a number in form $2^{8k}$, where $k \in \mathbb{N}$. However, the order of $H_s$ is $2^{2^n-1}$ and $2^n - 1 \neq 8k$. Thus the real-world ciphers will be based on a group whose compact representation is one bit longer. This can simply be achieved by using a slightly modified group $H_s \times H_1$ instead of $H_s$. The compact representation grows by one bit to the desired $2^n$ and the multiplication stays in principle the same as in $H_s$, only the highest bit must be handled (xor-ed) separately. The multiplication operation continues to be non-commutative and non-linear. From now on we suppose that all permutations and all group bases are stored and manipulated only in the compact representation.

We have presented the two most extreme examples for permutation 2-groups, the simplest $\mathbb{Z}_2^2$, which is commutative, and the most complex $H_s \times H_1$. In principle any other 2-group can be used in an appropriate representation. New 2-groups for our cryptographic purposes can be constructed from the available ones by taking wreath products, direct products, extensions and their combinations [6].

### 3.2 Key Generation

A key for our cryptosystem consists of two randomly chosen group bases. This approach ensures an extremely high upper limit of the scalable key space. Because the bases can hardly be entered manually by the user, we need a mechanism for generating random bases. Possibly, in cases where a fixed key length is expected, the bases could be generated from a binary key of fixed length or from a pass-phrase, in conjunction with the use of a pseudorandom number generator, which in turn is based on a subsystem implementing a fixed version of our system.

In general, not every basis enables a fast factorization. An efficient factorization algorithm is only known for so called transversal group bases [5], [6]. Therefore we want to generate only bases of this kind. The Basis Generation Algorithm (BGA) starts from the canonical basis $\alpha$ and carries out the following four steps:

1. The commutative block shuffle operation randomly changes the order of the blocks by multiple swaps of two adjacent blocks. Two blocks $B_i = (b_{i,0}, \ldots, b_{i,r_i})$ and $B_{i+1} = (b_{i+1,0}, \ldots, b_{i+1,r_{i+1}})$ can be swapped only if $b_{i,j} \ast b_{i+1,j} = b_{i+1,j} \ast b_{i,j}$ for $j \in \mathbb{Z}_{r_i}$ and $k \in \mathbb{Z}_{r_{i+1}}$.

2. The block fusion operation replaces two randomly chosen, adjacent blocks $B_i = (b_{i,0}, \ldots, b_{i,r_i-1})$ having the set of key bit positions $c_i$ and $B_j = (b_{j,0}, \ldots, b_{j,r_j-1})$, $j = i + 1$, having the key bit positions $c_j$ by a single longer block $B'_i = B_i \ast B_j = (b_{i,m} \ast b_{j,n} : m \in \mathbb{Z}_{r_i}, n \in \mathbb{Z}_{r_j})$ having the key bit positions $c'_i = c_i \cup c_j$. Note that block fusion changes the type of the basis from $r = (r_0, r_1, \ldots, r_i, r_{i+1}, r_{i+2}, \ldots, r_{w-1})$ to $r' = (r_0, r_1, \ldots, r_i \ast r_{i+1}, r_{i+2}, \ldots, r_{w-1})$ and decreases the dimension of the basis from $w$ to $w - 1$.

3. The randomization operation replaces each $b_{i,j} \in B_i$, $i \in \{1, 2, \ldots, w - 1\}$, $j \in \mathbb{Z}_{r_i}$ by $b'_{i,j} = b_{i,j} \ast \prod_{k=0}^{l-1} b_{k,l_k}$, where $l_k \in \mathbb{Z}_{r_k}$ is chosen randomly for every combination of $i$, $j$ and $k$.

4. The element shuffle operation randomly changes the order of the elements within each block.

Each step can be skipped or carried out several times. If $\beta \in B_G$ then each $\beta'$ generated from the $\beta$ by any combination of these steps is also in $B_G$. Moreover, BGA preserves transversality, so all bases generated from transversal $\alpha$ will enable a fast factorization. For instance, the basis $\beta_2$ in Example 2.2 was created from $\beta_1$ by a block shuffle (the axes $B_0$ and $B_1$ are exchanged) and an element shuffle
(the indices of the points on each axis are shuffled). Block fusion and randomization were not applied there.

The complete key generation scheme from the pass-phrase to the pair of group bases might look as follows:

**Figure 3.1 Key generation**

[Diagram of key generation process]

A $k$-bit hash value is extracted from the pass-phrase which was entered by the user. For example a Cyclic Redundancy Code [7] with a primitive polynomial of degree $k+1$ might be used for obtaining the $k$-bit hash. Optionally, the $k$-bit binary key $K$ can be generated or entered directly. The length of $K$ is freely scalable, theoretically up to several tens of thousands of bits. In practice, lengths of about 64 to 256 bits will be used. Key $K$ is passed as a seed to a pseudo-random number generator which delivers pseudo-random numbers to the BGA. The generator PRNG is a sensitive part of an implementation and must be chosen very carefully. (See also Section 4.1.)

3.3 Fast Factorization

Suppose $G$ is a 2-group, $|G| = 2^n$, and $\beta = (B_0, \ldots, B_{w-1})$ a transversal, $w$-dimensional basis of $G$ of type $r = (r_0, \ldots, r_{w-1})$. Each block $B_i = (b_{i,0}, \ldots, b_{i,r_i-1})$ contains $r_i$ permutations, where $r_i = 2^{m_i}$. The set $c_i = \{c_{i,1}, \ldots, c_{i,m_i}\}$, contains the key bit positions for $B_i$. Let $K_{B_i}: 2^n \rightarrow 2^{m_i}$ be a function which extracts the key bits from a binary vector, $K_{B_i}(a_0, \ldots, a_{n-1}) = (a_{c_{i,1}}, \ldots, a_{c_{i,m_i}})$.

The factorization of a permutation $p \in G$ is performed level-wise. First, the highest coordinate $x_{w-1}$ is obtained from $p_w = p$ as described below, then the intermediate result $p_{w-1} = p_w \cdot b_{w-1,x_{w-1}}^{-1}$ is passed to the lower level and the process continues in the same way until the lowest level, where an $x_0$ is obtained and $p_0 = p_1 \cdot b_{0,x_0}^{-1}$ is equal to the identity permutation in $G$.

Let $p_i = (a_0, \ldots, a_{n-1}) \in 2^n$ be an input to a factorization step. The index $x_{i-1}$ is obtained as $x_{i-1} = F_{i-1}(K_{B_{i-1}}(p_i))$, where $F_i : 2^{m_i} \rightarrow 2^{m_i}$ is a bijection such that $F_i(k) = j$ if and only if $K_{B_i}(b_{i,j}) = k$.

3.4 Extensions of the Basic System

The cryptosystem introduced in Definition 2.2 demonstrates the basic principle of encryption based on group bases, the mapping of elements from one basis to another. However, even if we use a non-commutative carrier group with multi-dimensional bases, the cryptographic properties of this mapping will not be sufficient. In the following we present two effective techniques which extend the basic setup and improve the confusion as well as the diffusion [8] of the cipher.

3.4.1 Bit Reversing

During encryption a permutation $p \in G$ is factorized with respect to the first basis $\beta$ and the resulting coordinates $x_i$ are passed to the composition in the second basis $\beta'$. Let us suppose for a moment, that both $\beta$ and $\beta'$ are of the same type, this makes the problem more obvious.

Because both bases are randomized, we can consider each factorization and composition level as a kind of an S-Box, which increases the confusion. As shown on the left side of Figure 3.2, each of
the indices $x_i$ has been influenced by a different number of S-Boxes. While $x_0$ passed all eight, $x_3$ went only through two S-Boxes. That means that some parts of the information contained in $x$ have been “scrambled” much less than other ones. This is an undesirable property, because “parts of the information” are not fully protected. Moreover, if $G$ is a commutative group (such as $\mathbb{Z}_2^2$), a large part of the ciphertext will not depend on $x_3$ at all. So diffusion is also reduced.

**Figure 3.2 Effect of Bit Reversing**

For this reason, we propose a bit reversing of the index vector $x$ before the start of composition. Note that bit reversing is better than a simple index vector component reversing, because the bases are not necessarily of the same type. The new encryption and decryption functions are:

$$y = e_k(x) = \lambda^{-1}(\beta_2(R(\beta_1^{-1}(\lambda(x)))))$$

$$x = d_k(y) = \lambda^{-1}(\beta_1(R(\beta_2^{-1}(\lambda(y)))))$$

where the function $R : 2^n \rightarrow 2^n$ reverses the order of the bits of a binary vector. $R(b_0, b_1, \ldots, b_{n-1}) = (b_{n-1}, b_{n-2}, \ldots, b_0)$, $b_i \in \{0, 1\}$. The effect of $R$ is illustrated on the right hand side of Figure 3.2. Now every index passes exactly five S-Boxes resulting in a balanced confusion of all components. The length of the index vectors $x = x_0||x_1||\ldots||x_{w_1-1}$ and $y = y_0||y_1||\ldots||y_{w_2-1}$ is the same $|x| = |y| = n$, even if the bases $\beta$ and $\beta'$ are not of the same type. So bit reversing can be used in this case as well.

### 3.4.2 Non-Linear Diffusive Transformation

At each factorization level the input $p_i$ is divided by a factor $b_{i-1}, x_{i-1}$ from the current basis block. Only a small part, namely the key bits, of $p_i$ determines which factor will be taken. Because multiplication and division of permutations in the compact form of $\mathbb{Z}_2^n$ are defined as a simple bit-wise XOR, a change of a single non-key bit of $p_i$ affects only a single bit of $p_i$. Consequently diffusion at each factorization step is weak. Even worse, factorization in $\mathbb{Z}_2^n$ is a linear function. Although factorization in $H_s$ is not linear and its diffusion is the best among all 2-groups, it is still not strong enough from the cryptographic point of view. This is because the higher order bits of a product depend only on the higher order bits of multiplicands.

Fortunately, both the weak diffusion and linearity can be compensated by a simple extension of group bases. Figure 3.2 shows the idea on a geometric analogy to the group bases of $\mathbb{Z}_2^4$. Basis $A$
was obtained from the canonical $\alpha_4$ by two fusing $B_0$ with $B_1$ and $B_2$ with $B_3$. An element shuffle of $A$ resulted in basis $B$ and applying a further randomization created $C$. In all three cases the factor $B_1[x_1]$ of any point depends only on its vertical position, the horizontal position does not play any role. The points having the same factors in $B_1$ lie on parallel horizontal lines (surfaces).

**Example 3.2 Linear And Non-Linear Bases**


cases $A, B, C$ of our example, $KB_1 : 2^4 \rightarrow 2^2$ extracted two key bits from a 4-bit word (compact representation of $\bar{p} \in \mathbb{Z}_2$), $KB_1(a_0, a_1, a_2, a_3) = (a_2, a_3)$, and $F_1 : 2^2 \rightarrow 2^2$ found the appropriate factor in $B_1$, $F_1(a_2, a_3) = x_1$. In the extended version (cases $D, E, F$) $x_1$ depends on all four bits of $p$. In case $C$, $T_1 : 2^3 \rightarrow 2^2$ is defined by $T_1((a_0, a_1, a_2, a_3) = (a_0, a_1) + (a_2, a_3)$, in case $D$ by $T_1((a_0, a_1, a_2, a_3) = (a_0, a_1) \ XOR (a_2, a_3)$ and in case $E$ as $T_1((a_0, a_1, a_2, a_3) = \text{ROTL}(a_0, a_1) + (a_2, a_3)$. Many other functions are also possible. In general, the hash function $T$ should possess at least the following four properties:

- each bit of the output should be dependent on each bit of the input
- each output bit should be balanced\(^2\)
- the function should not be linear
- composite function $SB(p, T(p))$ must be invertible

However, one can also use stricter criteria, similar to those used in the construction of an $n \times m_i$ S-Box [1]. When using a proper $T$, the avalanche effect in the cipher will be very strong, because every single factorization or composition step ensures that the avalanche criterion is fulfilled. In [12] author defines the term *excess avalanche factor* for iterative ciphers. If we think of every factorization or composition step of our non-iterative ciphers as a "round", then the value of an analogue of the EAF will be equal to $w_1 + w_2$, where $w_i$ are the dimensions of used bases.

\(^2\)An output bit is balanced if $n_0 \equiv n_1$, where $n_0$ is the number of inputs for which the output bit is equal to 0.
4 Security Aspects

The security, speed and memory requirements of our ciphers depend strongly on the concrete configuration. The most important parameters are:

- the order of the carrier group (affects the block length),
- the extent of block fusion (affects the size of “S-Boxes” and the number of “rounds”),
- the function \( T \) used,
- and the randomness of the group bases.

In implementations, where i) the carrier group has large order (e.g. \( 2^{128} \)), ii) the extent of block fusion is reasonably large (e.g. 12 key bits per block), iii) a sensible non-linear \( T \) was chosen and iv) the bases were generated directly from some physical source of “true” random numbers, we can probably relax.

4.1 The Pseudo-Random Number Generator

A key in our cryptosystem consists of two secret group bases, alternatively speaking, of two sets of several, large, key dependent S-Boxes of special structure. As the algorithm itself is simple and public, a possible attack would try to reconstruct the bases, using a chosen plaintext attack or similar techniques.

If an implementation uses a \( \text{PRNG} \) for generating the bases, the properties of the \( \text{PRNG} \) are crucial for the security of the cipher. The number of possible initial states of the generator, i.e. the size of the generator’s seed, must be reasonably high, because it directly bounds the real key space of the cryptosystem, which must be exhausted in a brute force attack. Of course, we cannot use a simple 32-bit linear congruential generator, unless we want to construct a weak cipher. The size of the generator’s seed is just one of the many measures that determine the quality of the \( \text{PRNG} \) from a cryptographic point of view. The \( \text{PRNG} \) needs to pass non-trivial randomness tests, like the Maurer Test [9], the Diehard suite [10], etc. Otherwise some attacks based on dependencies within the bases might be possible.

In our opinion, the lagged Fibonacci generator with Lüschers’s approach [11] is a proper example of an acceptable \( \text{PRNG} \). For instance, using lags (37,100) with a word length of 32 bits, the generator passes all statistical tests, the size of its seed can be scaled up to 3200 bits and the period of the generated sequence is \( 2^{131} \). These values can be further improved by changing the lags.

Finally, a comment should be made about a brute force attack on a cipher using \( \text{BGA} \). The time needed for generating bases (usually less than one second) is negligible for the legal user, who generates the key once, but it is a big problem for an attacker, who tries all possible keys. When trying, say, \( 2^{64} \) different keys, with a delay of 500 ms per key, a brute-force attack is infeasible.

4.2 Block Fusion

The average length of blocks is also very important for the security of the system. Let \( 2^n \) be the order of the group \( G \) and let \( x \) be a divisor of \( n \). When \( \text{BGA} \) merges \( x \) adjacent blocks \( k \cdot x, k \cdot x + 1, \ldots, k \cdot x + x - 1 \), of \( a_n \), \( k \in [0, \frac{n}{x} - 1] \), we say that a block fusion to extent \( x \) was performed. \( (x \) is equal to the number of key bits per block.) The number of adjacent blocks merged needs not necessarily be constant for all fused \( x \)-tuples. In this case the average fusion extent can be computed by \( x = \frac{2^n}{w} \), where \( w \) is the dimension of the basis after block fusion.

For instance the canonical basis \( a_{64} \) has 64 blocks with two permutations in each block. Each permutation in the compact form is 64 bits long, so the whole basis fits in 1 KB of memory. If
we perform a fusion to extent 4, we obtain a basis with 16 blocks of 16 permutations (2 KB), a
fusion to extent 8 creates 8 blocks of 256 permutations (16 KB), etc. The fusion to extent 64, would
result in one block of $2^{64}$ permutations (227 TB). Of course, the higher the extent of block fusion,
the more secure the cipher. In the extreme case $x = n$ we obtain a full random permutation of $2^n$
elements, which is the strongest n-bit cipher available. On the other hand, the memory requirements
are growing exponentially with $x$ and the quality of the PRNG becomes more critical. The more PRNs
are generated, the higher the probability, that some weakness of the PRNG might be exploited. For
the reasons above a tradeoff between security and memory load must be found. The values between
8 and 16 key bits per block would be appropriate for practical use.

5 Experimental results

We implemented a scalable software version of the proposed algorithm. We used two carrier groups,
the $\mathcal{H}_s \times \mathcal{H}_1$, supporting block lengths 8, 16, 32, 64, 128 and 256 bits, as well as the $\mathbb{Z}_2^n$, supporting
all block lengths from 32 to 512 bits, divisible by 32. We used a fixed key length of 128 bits given by
the number of possible initial states of the PRNG.

5.1 Throughput

Both algorithms were implemented in C++ and tested on a Pentium II machine running at 350 MHz.
As expected, the first carrier group was less suitable for a software implementation. The multiplication
of permutations from $\mathcal{H}_s \times \mathcal{H}_1$ in the compact representation is a bit-oriented recursive algorithm not
very well supported by the instruction set of the processors. To make the factorization faster, we
precomputed the inverses of all permutations in the group bases. So we actually stored four instead
of two bases. The required memory space was about 90 KB. The throughput of the 64-bit version
without transformation $T$ was about 75 KB/s and with a simple transformation only 50 KB/s. When
we used the Cartesian representation of permutations, speeds rose to 275 KB/s without a $T$, and
100 KB/s with a transformation. The memory requirements were about 900 KB. Even if some tighter
optimization techniques were to improve the speeds by a factor of 2 to 4, the values achieved by the
software implementation can not be considered as very satisfactory. A simplified hardware version
of the algorithm with its own special multipliers running at a clock rate of 45 MHz achieves speeds
above 20 MB/s according to [6], so the group $\mathcal{H}_s \times \mathcal{H}_1$ is definitely more suitable for a hardware
implementation.

Our second implementation used $\mathbb{Z}_2^n$ as carrier group. The multiplication of permutations in this
commutative group is much faster than in $\mathcal{H}_s \times \mathcal{H}_1$. We used a simplified NFS without block shuffle and
with the fixed fusion length 8 blocks, which made the factorization even more efficient. A non-linear
transformation $T$ was used at each level of the factorization and composition operations. The 64-bit
version occupied 18 KB and encrypted at a rate of 2 MB per second. The 128-bit version with memory
requirements 69 KB achieved about 1.5 MB/s and the 256-bit version ran at 1 MB/s occupying 270 KB
of memory. Again some speed improvements by a factor of 2 to 3 might be possible after a strong
optimization effort. These results confirmed that $\mathbb{Z}_2^n$ is much more efficient than $\mathcal{H}_s \times \mathcal{H}_1$, at least in
software.

5.2 Randomness

As we do not yet know of any special attacks against the new cipher, we used a general statistical
approach to estimate the quality of encryption. We encrypted a large amount of highly redundant
non-periodic data (e.g. a sequence of blocks, containing n-bit binary representation of a counter
sequence 0, 1, 2, ...), and tested the output for randomness. The idea of the testing approach is the following: The cipher must provide strong diffusion, so even small changes between the adjacent input blocks must result in big and random looking changes in the output blocks. Further, the cipher must provide a strong confusion, so a systematic and highly redundant input sequence must be encrypted into a sequence which cannot be distinguished from a true random one by any statistical tests. The output sequences were tested by the DieHard suite of statistical tests [10]. The tests were carried out for many different keys.

The data were encrypted using the carrier group \( \mathbb{Z}_2^n \) and the bases were generated with the lagged (37, 100) Fibonacci Generator with Lütcher's approach. The fixed number of key bits per block was set to 8. Here is a C-like definition of one of the non-linear transformations used, \( T : 2^n \rightarrow 2^8 \), \( n = 8k, k \in \mathbb{N} \):

```c
byte I(vector p)
byte sum = 0;
for i=0 to n/8-1
    sum = rotr3(sum + p[i]);
return sum;
```

The \( p[i] \) are the 8-bit segments (bytes) of the n-bit binary vector \( p \) and the function \( \text{rotr3} \) performs a 3-bit right rotation of an 8-bit value.

Each test from the DieHard suite evaluates the quality of the input with a so called p-value, \( p \in [0,1] \). Good results should lie between 0.001 and .999. The tests with results below \( 10^{-6} \) or above \( 1 - 10^{-6} \) are considered as failed. However, one must keep in mind that even a true random number generator generates sometimes a sequence, which "fails" the test, since all sequences, even the "less random" ones, appear with the same probability.

We carried out 4400 tests for each configuration and counted the number of significant results among these tests. A result was considered as significant (or suspect), if the value \( p \) was below 0.001 or above 0.999. The average ratio of significant results, measured by our cipher, was 0.0024 for block length 64 bits and 0.0025 for 128 bits. In our opinion the results can be considered as satisfactory. For instance, the cipher IDEA, which is regarded as one of the most secure 64-bit ciphers today, achieved on average 0.0029 of significant results by the same test. An output of a simple linear congruential generator produced 0.5036 of significant results.

6 Conclusions

We have presented a family of block ciphers based on group bases. Our approach enables to design a simple weak cipher, which can be deeply analyzed and examined, as well as a large, strong one. Even the full symmetric group of degree \( 2^n \) can be realized from the same specification.

In contrast to Feistel networks, our cipher is not iterative. Instead of several repetitions of a uniform round, a specific number of factorization and composition steps are carried out. The security can be scaled through the average fusion extent instead of the number of rounds. The group bases used by our cipher are random and key-dependent. They can be viewed as a set of S-Boxes with a special structure.

The block length, key length and security level of the ciphers are scalable. Some other components of the cipher, which affect the speed and memory requirements, are also variable. The system has been optimized for maximal confusion, diffusion and non-linearity. The results of statistical tests were very satisfactory. Nevertheless, the cryptosystem is still too new to allow us to make strong statements about its security. Some attacks based on the special structure of the group bases may be possible as well as attacks targeting the special properties of used PRNG. The presence of a mathematical foundation lets us hope that a deeper theoretical analysis of the cryptosystem will be possible.
References


Appendix - Encryption Example

This appendix is intended as an extension to sections 3.3 and 3.4.2 for people who want to implement our cipher.

The carrier group of the presented cryptosystem is $\mathbb{Z}_2^n$ for $n = 16$. Both group bases were constructed from $\alpha_{16}$ by a simplified version of $\delta$GA. The block shuffle was skipped. The block fusion was performed with constant extent 4 and a full randomization and full element shuffle were carried out.

A block of a transversal basis contains three kinds of bits. The dark gray ones are completely random. The light gray ones - the key bits - represent a random permutation of all 4-bit values for each block. The white bits are always zeros and therefore do not necessarily need to be stored in memory.

If the block fusion had not been skipped, the three areas (dark, light, white) would not be continuous. The columns of the blocks would be permuted by a randomly chosen fixed permutation for all blocks of one basis.

In our example the plaintext $p = 10010111110010111$ is encrypted into the corresponding ciphertext $c = 00000101110011001$.\n
The functions $T: 2^n \rightarrow 2^4$ and $T_{inv}: 2^4 \rightarrow 2^n$ are defined as follows:

```c
byte T(vector p) { byte sum = 0; for (i=0 to n/4-1) sum = a * (sum * p[i]); return sum; }
byte T_{inv}(vector p) { byte sum = 0; for (i=0 to n/4-2) sum = a * (sum * p[i]); return b * (p[n/4-1] - sum); }
```

where the constant $a = 13$ is relatively prime to $n$ and $b = a^{-1}(\text{mod} 16) = 5$.

The function $\oplus$ reverses the bits of an operand and the function $\oplus$ performs a bit-wise XOR of its two operands.