A Small Initial Investigation into a Key-dependent Random Permutation with an Example in PRESENT

D. Dhebar[*] and S. Jassim
University of Buckingham, Dept. of Applied Computing
Buckingham MK18 1EG, United Kingdom
[*]*Email: 0109340 [AT] buckingham.ac.uk*

Abstract— **In this paper we demonstrate a way to create a random permutation component directly from a symmetric key; we call this the *Bipartite Graph Function* (BGF). The idea is that the BGF will inherit the entropy of the key and therefore produce unpredictable permutations. We first show, by a few small initial trials, that the BGF retains the expected number of fixed points as for any random permutation. Then, as a secondary idea, we explain how one might use the BGF to replace a permutation in a block cipher and chose PRESENT to demonstrate an example. Several small trials that focussed on different Hamming distances showed BGF-PRESENT compared well with PRESENT. Indeed, both ciphers produced near ideal Hamming distances of *$n/2 = 32$* in all trials. The BGF may therefore be a useful tool to protect ciphers that have weak *fixed* permutations that can be exploited by attacks such as the *statistical saturation attack*.**

***Keywords:* random permutations, symmetric ciphers, bipartite graphs, PRESENT**

I. INTRODUCTION

In modern ciphers the only component that must necessarily remain secret is the *encryption key*, KE [1]. To help ensure this secrecy, encryption keys should be the result of approved *random bit generators* (RBG), such as those described by NIST, and therefore hard to predict [2]. In this paper we used a similar key called the *rearrangement key*, KR, (also the result of an approved RBG) to create a random permutation component called the *Bipartite Graph Function* (BGF). The idea is that the BGF should inherit the same, or nearly the same, entropy from KR and therefore create unpredictable permutations.

Block ciphers typically meet their security needs through the use of iterated rounds in which each includes a layer that aims to *confuse* and *diffuse* the input message and the key. The use of substitution boxes, often the only non-linear component in a cipher, is responsible for adding confusion, while the permutation layer is responsible for adding diffusion.

The purpose of the permutation layer is to diffuse the input such that after only a few rounds every bit will depend on every previous one. This ensures the desirable property that a small change in the input will effect a large change in the output; ideally a 1-bit change in the input will cause a change in about half the output bits. Indeed, a good cipher will typically achieve full diffusion after relatively few rounds. For example, AES-128 achieves full diffusion after just two of its ten rounds [3]. This required property highlights the importance of a carefully described permutation layer and for this reason we first show that the BGF retains the expected number of fixed points of a random permutation, i.e. 1. If the number of fixed points were too high this would render the BGF useless as a means to cause diffusion.

We then do a small initial investigation into how the BGF may work in practice by comparing PRESENT (original) with BGF-PRESENT, i.e. PRESENT with the permutation layer replaced with the BGF. For any good cipher a good avalanche effect is a result of both a good substitution layer *and* a good permutation layer working in harmony. Therefore, changing the permutation layer in a cipher may adversely disturb its avalanche effect, so in this initial investigation we focus on comparing the avalanche effect of PRESENT and BGF-PRESENT to observe if there are any immediate concerns.

The rest of the paper is organised as follows: In section II we review related research work on randomising cryptographic components of existing symmetric ciphers and section III describes the process of constructing a BGF, including some analysis. In section IV we briefly describe PRESENT and our variant, BGF-PRESENT while section V provides the results of four small trials. Section VI offers conclusions and further work.

II. RELATED WORK

The idea of random S-boxes and permutations in symmetric ciphers goes back some thirty years and there is still much interest in them. In the next two paragraphs we look at some of the work on random S-boxes and then some of the work on random permutations.

Biham and Shamir looked at randomising the contents, and the order, of DES S-boxes only to show that both made DES weaker [4]. In 1993 Schneier published Blowfish [5], a 16-round Feistel cipher that uses 64-bit input blocks, four 32-bit key-dependent S-boxes and keys up to 448 bits in length. However, despite this a reduced-rounds variant of Blowfish could be susceptible to certain weak keys because they might generate weak S-boxes [6]; a weak key is one that creates an S-box that has a collision, i.e. two distinct inputs to an S-box produce the same output. Even though such attacks are not effective against the standard Blowfish algorithm, it shows that key-dependent aspects and the sheer size of a key space are not guarantees against weaknesses. Five years after Blowfish was published Schneier et al. released Twofish [7], the successor to Blowfish and the only one from five finalists for the *Advanced Encryption Standard contest* that included random features by way of key-dependent S-boxes [8]. Abd-ElGhafar et al. (2009) recommends adding dynamism to AES by using the stream cipher RC4 to generate a random key stream that in turn produces the S-box [9]. Hosseinkhani et al. (2012) [10] and Mahmoud E. et al. (2013) [11] put forward similar ideas that include a key-dependent S-box for AES, yet more recently Tiessen, Knudsen et al. (2015) showed how AES with a secret S-box is weak to attacks when reduced to six rounds [12].

Ruby and Rackoff (1988) showed how to construct pseudorandom permutations from pseudorandom functions [13], and Even and Mansour (1991) created a cipher from a single pseudorandom permutation and only two subkeys [14]. However, Daeman (1991) quickly demonstrated its *severe limitations* [15] and further weaknesses were found by Dunkelman et al. (2012) [16]. No matter, other ways of using random permutations persisted. Hall et al. (1998) [17] looked at building pseudorandom functions from pseudorandom permutations and, more recently, Kuppusamy et al. (2014) [18] proposed a means of creating a table of permutations created by a key-pair K1 and K2. However, this paper showed some poor avalanche effects. For example, two inputs differing by one bit resulted in ciphertexts that also differed by only one bit. Borgoff, Knudsen et al. (2011) offers an attack on PRESENT-like ciphers with secret S-boxes and even extend the idea to cases where both the S-boxes and permutations are chosen uniformly at random [19]. The attacks were effective against PRESENT-like ciphers, with 16 secret S-boxes, up to 28 rounds. For example, they attacked the cipher Maya, Gomathisankaran and Lee (2009), and recovered all 16 S-boxes even though all were key-derived and therefore secret [20]. However, once the random S-boxes and permutation were chosen, they remained fixed thus making differential-type attacks easier to carry out. In 2010 Knudsen, Leander, et al. proposed the ultra-lightweight PRINTcipher that includes a key-dependent permutation. Every set of three bits that enter an S-box are randomly permuted but thereafter remain the same for each round, and even then not all permutations are available [21].

The difference in our proposal is that the permutations will change randomly and dynamically for *each* round, not just for each full encryption, and are entirely dependent on M, KE and KR (see *Fig. 5*). Indeed, the BGF may offer greater resistance against the statistical saturation attack. In the case of PRESENT, this attack exploits a weakness in its permutation layer. As just one example, of the sixteen bits that enter S-boxes 5, 6, 9 and 10, only eight bits are sent to other S-boxes by the permutation layer. Therefore, fixing sixteen bits means eight of those bits *will be known at the very same input of the next round* [22]. By exploiting this and using the statistical saturation attack an attacker can break up to fifteen rounds of PRESENT using about 2^{36} plaintext-ciphertext pairs. The permutation in PRESENT also leads to other attacks such as linear cryptanalysis. For example, Cho (2010) demonstrated a way to recover the 80-bit key of PRESENT for up to 26 of the 31 rounds [23]. However, if the permutation were random for each round, as for BGF-PRESENT, such attacks may be infeasible.

III. THE BIPARTITE GRAPH FUNCTION (BGF)

The Bipartite Graph Function (BGF) is a random permutation function that accepts inputs of various lengths and, in this paper, is dependent on the symmetric key KR. We use the expression *n-BGF* where *n* refers to the number of objects being permuted. Moreover, we use a *KR* that is independent of KE, but this may not be a necessity. The required length of KR depends on the degree of security necessary. However, it would be sensible to make KR as short as possible (i.e. such that security is not compromised) so as not to burden key management unnecessarily. We have chosen KR to be 64 bits as this is the same length as the encryption key for PRESENT.

A. *Creating a permutation from the BGF*

Consider a complete bipartite graph $K_{n,n}$ with vertex sets |K| = |L| and where K = {k_0, k_1, … , k_{n-1}} and L = {0, 1, …, n-1} (*Fig. 1*). A *BGF graph* is a subset of $K_{n,n}$ where exactly one edge from each vertex in **K** joins a vertex in **L**. Hence, it is permissible for more than one vertex in **K** to join the same vertex in **L**, but not the other way around, i.e. it is comparable to a surjective function. The BGF of interest is in fact a directed graph from **K** to **L**, but for clarity we omit the arrows. Succinctly, a BGF graph is *a directed bipartite graph without multiple edges*.

Now imagine each vertex $k_i \in \mathbf{K}$ can accept some input element $m_i \in \mathbf{m}$ (where **m** is a finite set) that can traverse any edge incident with its vertex and move to any vertex $l_i \in \mathbf{L}$. For example, if input m_0 traverses edge {k_0, 0} it will move straight down to the vertex parallel with its original position. However, if it traverses edge {k_0, 1} it will move to a position parallel with vertex k_1. In this way any m_i can traverse an edge that would place it parallel with any k_i. For example, if we randomly select edges {k_0, 2}, {k_1, 1}, {k_2, 0} and {k_3, 1} we end up with a random permutation of the inputs. See *Fig.2* in which the non-traversed edges have been removed for clarity.

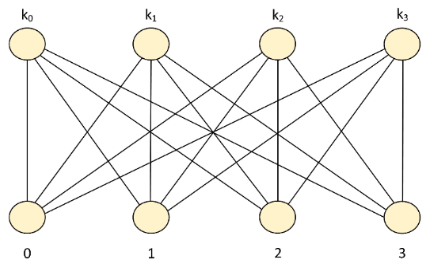

Figure 1: Complete Graph $K_{4,4}$

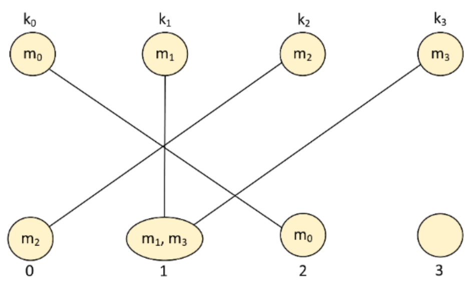

Figure 2: A random permutation of inputs m_0 to m_3

Therefore we can use a BGF graph to create a random permutation of a finite set **m** = $\{m_0, m_1, \ldots, m_{n-1}\}$. By using two-line notation, we obtain a permutation π: $\boldsymbol{m} \rightarrow \boldsymbol{m}$:

$$\pi = \begin{pmatrix} m_0 & m_1 & \Lambda & m_{n-1} \\ \pi(m_0) & \pi(m_1) & \Lambda & \pi(m_{n-1}) \end{pmatrix}$$

Note that when two or more inputs go to the same vertex in **L**, the natural order takes precedence and hence the resulting permutation from the BGF graph depicted in *Fig. 2* is:

$$\pi = \begin{pmatrix} m_0 & m_1 & m_2 & m_3 \\ m_2 & m_1 & m_3 & m_0 \end{pmatrix}$$

However, while graphs offer clear visual examples of the BGF, they soon become cumbersome (and indeed *unclear*) as *n* becomes larger. Therefore, we shall now introduce a more formal description using matrices.

The adjacency matrix of a bipartite graph is given as:

$$A = \begin{pmatrix} 0_{r,r} & B \\ B^T & 0_{s,s} \end{pmatrix},$$

where ***B*** is an $r \times s$ matrix (sometimes called the *bi-adjacency matrix*), $\boldsymbol{B^T}$ is the transpose of ***B*** and $\boldsymbol{0}_{r,r}$ and $\boldsymbol{0}_{s,s}$ are the zero matrices of $r \times r$ and $s \times s$ respectively. However, we need only matrix ***B*** to represent the graph. Furthermore, the BGF graph is always square (i.e. $r = s$) so we denote the matrix of the BGF graph as:

$$B = \begin{pmatrix} a_{1,1} & a_{1,2} & \Lambda & a_{1,n} \\ a_{2,1} & a_{2,2} & \Lambda & a_{2,n} \\ \mathrm{M} & \mathrm{M} & & \mathrm{M} \\ a_{n,1} & a_{n,2} & \Lambda & a_{n,n} \end{pmatrix},$$

$$a_{i,j} = \begin{cases} 1 \text{ if the corresponding edges are connected} \\ 0 \text{ if the corresponding edges are not connected} \end{cases}$$

For a BGF graph, the rows and columns of its corresponding bi-adjacency matrix, $\mathbf{B}_{\mathrm{KR}}$, are represented by the vertices in the sets **K** and **L** respectively. For example, *Fig. 3* is the BGF bi-adjacency matrix of *Fig. 2*:

$$B_{KR} = \left(\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ k_0 & 0 & 0 & 1 & 0 \\ k_1 & 0 & 1 & 0 & 0 \\ k_2 & 1 & 0 & 0 & 0 \\ k_3 & 0 & 1 & 0 & 0 \end{array}\right)$$

Figure 3: Bi-adjacency matrix of BGF graph in Fig. 2

Choosing desired edges at random requires a reliable and convenient method, so we use *KR* to meet this need. Consider an 8-bit *KR* where each pair of bits k_i refer to the rows of $\mathbf{B}_{\mathrm{KR}}$ and where the denary expression of each k_i refer to the columns. For example:

$$KR = (k_3\, k_2\, k_1\, k_0) = (01-00-01-10) = 1\,0\,1\,2$$

$$\Rightarrow B_{KR} = \left(\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ k_0 & 0 & 0 & 1 & 0 \\ k_1 & 0 & 1 & 0 & 0 \\ k_2 & 1 & 0 & 0 & 0 \\ k_3 & 0 & 1 & 0 & 0 \end{array}\right) \qquad (1)$$

We can now multiply (1) by the input vector **m** to produce the permutation $\pi(\boldsymbol{m})$:

$$\boldsymbol{mB_{KR}} = \boldsymbol{m}(\pi) \qquad (2)$$

Where **m** is the input row vector of *n* elements to be permutated by the bi-adjacency matrix $\mathbf{B}_{\mathrm{KR}}$ and $\mathbf{m}(\pi)$ is the resulting row vector permutation.

In conventional matrix operations the result of multiplying such a matrix by a vector might include noughts and *addition* symbols. However, we are interested only in the objects being permuted so we discard them from the result. For example, if we use (2) with the same $\boldsymbol{B_{KR}}$ described in (1), then:

$$(m_0\, m_1\, m_2\, m_3)B_{KR} = (m_2\ \{m_1 + m_3\}\ m_0\ 0) \Rightarrow (m_2\ m_1\ m_3\ m_0)$$

B. BGF analysis

For an n-BGF we require a KR of minimum length $n\log_2(n) = N$ bits with bit-partitions of length $\log_2(n)$. In our case, a 64-BGF requires KR to be a minimum 384 bits long with bit-partitions of length 6, so we need a key expansion; see (5). However, since the total number of permutations of *n* objects is *n!* there will be many KRs that produce the same permutation. We know this since $n! < n^n$. We omit the proof here, but this can easily be shown using Sterling's approximation for factorials and setting $n\log_2(n) = N$ and with 2^N written in terms of *n*.

All KRs that produce the same permutation are considered to be of the same class. The number of KRs in each class can be found using the formula for *combinations with replacements*:

$$C^R_{(n,r)} = \binom{n+r-1}{r} \tag{3}$$

where n is the number of possibilities and r the number selected.

As n increases the number of certain KRs in a class will also increase rapidly, thus making them more likely to be used to create a permutation. In other words, a certain number of *distinct* keys will produce the same permutation. For example, a 4-BGF permutes 4 elements which gives only $4! = 24$ permutations, even though a 4-BGF comes from an 8-bit KR for which there is a total of $2^8 = 256$ keys. We can find how the 24 classes of KRs are shared among the 256 keys using (3) and fixing $r = 4$ and substituting $n = 4$, $n = 3$, $n = 2$, and $n = 1$ respectively. The results tell us that one class/perm is produced by 35 distinct KRs, eleven different classes/perms are each produced by 15 distinct KRs, another eleven different classes/perms are each produced by 5 distinct KRs and one class/perm is produced by 1 distinct KR. This means, for instance, in the worst case there is a 13.7 per cent chance (35/256) of a random 8-bit KR being one of those in the KR class of 35 keys. However, as n becomes larger this *worst-case* percentage decreases rapidly. For example, for a 32-BGF the worst case is $2^{60}/2^{160} = 2^{-100}$. No matter, we can easily check if two distinct KRs are of the same class. There are $n!$ ways to permute n objects and therefore $n!$ distinct $n \times n$ permutation matrices **P**. Moreover, since there are $n!$ key classes KR, there must be one, and only one, **P** for each class. If two distinct permutation matrices were in the same class, this would mean they would both produce the same permutation, which is impossible. Therefore, if we have two keys KR and KR*, we can check if they are from the same class by observing their bi-adjacency matrices and comparing them with their corresponding permutation matrices **P** and **P***. We can easily find a corresponding matrix **P** by observing the two-line notations of **m**(π) and **m***(π). For example, $\boldsymbol{B}_{KR}$ from (1) gives:

$$\pi = \begin{pmatrix} m_0 & m_1 & m_2 & m_3 \\ m_3 & m_0 & m_1 & m_2 \end{pmatrix}$$

Therefore, each input m_i corresponds to row k_i in **P** and each **m**(π) to the column. For instance, in the two-line notation above m_0 maps to m_3 and therefore **P** has a 1 in row k_0, column 3. Continuing in this way we obtain:

$$P = \left(\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ k_0 & 0 & 0 & 0 & 1 \\ k_1 & 0 & 1 & 0 & 0 \\ k_2 & 1 & 0 & 0 & 0 \\ k_3 & 0 & 0 & 1 & 0 \end{array}\right)$$

Similarly, we can deduce **P***. If **P** = **P*** then KR and KR* are in the same class and if **P** ≠ **P*** then KR and KR* are in distinct classes.

IV. PRESENT AND BGF-PRESENT

In this section we shall demonstrate a possible application of the BGF by using it as a replacement for the permutation layer in the lightweight cipher PRESENT. However, we emphasise this as a theoretical example, not necessarily as a recommendation. Indeed, the extra coding would inevitably undo its *lightweight* properties. No matter, PRESENT is susceptible to the statistical saturation attack and multivariate linear cryptanalysis because of its fixed permutation layer. Therefore, there is good reason to consider replacing it with a stronger one.

A. PRESENT cipher overview

PRESENT is a lightweight substitution-permutation network that takes 64-bit input blocks and uses an 80-bit or 128-bit encryption key, KE. In this work we consider only the 80-bit KE. There are 31 full rounds with a final XOR to complete a full encryption. PRESENT uses a fixed 4-bit S-box and a fixed bitwise 64-bit permutation. KE is used to create 32 subkeys, one for each of the 31 rounds and one more for the final XOR operation. PRESENT is one of only two lightweight ciphers specified by ISO/IEC [24].

B. BGF-PRESENT

BGF-PRESENT is exactly the same as PRESENT except the permutation layer is replaced with the BGF (see *Fig. 5*). When using the BGF for BGF-PRESENT we require a subkey KRr for each round r. These subkeys are used to create the BGF of corresponding rounds, so we refer to a BGF perm of round r as BGF-r, e.g. BGF-3 refers to the BGF perm in round 3. The key schedule for KRr is given below:

$$\begin{aligned} &KR1 = KR \oplus KE1; \\ &KR2 = KR1 \oplus KE2; \\ &\vdots \\ &KRr = [KR(r-1) \oplus KEr] \end{aligned} \tag{4}$$

C. Creating a BGF for BGF-PRESENT

To create a BGF we first need to expand KRr to its minimum bit length $N = n\log_2 n$, where n is the length of KRr, and then partition it into $x = N/n$ blocks. In this paper we chose to further apply a distinct and simple bit-shift to each partition (5).

For BGF-PRESENT $n = 64$ so we expand KRr from 64 bits into $64\log_2(64) = 384$ bits. We then partition it into $x = N/n = 384/64 = 6$ blocks and apply bit-shifts of 0, 1, 2, 3, 4 and 5:

$$E(KR1) = KR1 \| KR1 >>> 1 \| \ldots \| KR1 >>> 5 \tag{5}$$

D. Outline of the key register for the BGF (see also Fig. 4)

1) *A 64-bit KR is XORed with KE to give a 64-bit subkey KR1.*
2) *KR1 is expanded and partitioned by (5) and used to create BGF-1.*
3) *KR1 is XORed with KE2 to produce KR2 (4).*

4) *KR2 is expanded and partitioned by (5) and used to create BGF-2.*
5) *KR2 is XORed with KE3 to produce KR3. Continue in this way up KR31.*
6) *C31 is XORed with KR to produce a new KR ready for a new encryption.*

The internals of BGF-PRESENT, including the key register of KR, is explained below (see also *Fig. 5*).

E. BGF-PRESENT outline

1) *A 64-bit input M is XORed with a 64-bit subkey KE1 (from the PRESENT encryption key)*
2) *The 64-bit output from 1. enters the S-box layer (the usual PRESENT S-box).*
3) *The output from 2. is randomly permuted by the BGF-1 layer.*
4) *The output of the BGF-1 layer completes one encryption round to give ciphertext C1.*
5) *The new input, C1, is XORed with KE2 and enters the S-box as before and the output of this is permutated by BGF-2. Continue in this way until BGF-31 gives C31.*
6) *XOR C31 with KE32 to produce the final output ciphertext C.*

Decryption works by applying the scheme in reverse order. Users will have access to KE and KR and therefore each BGF-r can easily be obtained from (4).

F. Analysis of BGF-PRESENT

The PRESENT cipher has already undergone extensive testing and is clearly strong against cryptanalytic attacks. However, as described in Section II, it could be stronger against the statistical saturation attack (and others) that exploit its weak permutation layer. The random permutation provided by the BGF should, in theory, remove this weakness because any bit output from an S-box has the potential to move to any of the available sixty-four positions for each round. Therefore, the fixed and predictable patterns will no longer occur. Nonetheless, we must not assume that solving one problem will not lead to others. For example, S-boxes and permutations work in harmony, so changing one of them may create new weakness that were not there originally. In particular, we must be careful that the BGF does not introduce weaknesses to differential and linear cryptanalysis. In theory, this should not be the case because the S-box remains unchanged and collecting input-out pairs of differentials will be much harder; the BGF-PRESENT produces unique outputs even when two or more inputs are the same. In a future paper we will look at this more closely, but here we look at how the BGF behaves as a random function independently and then how it might work as a replacement for the permutation in the cipher PRESENT.

V. Results

In this section we offer the results of our investigation. In Trial 1 we check the number of fixed points of permutations created by the BGF. If there were a notably high number, the BGF may have to be reviewed of even discarded, so we did this one first. In Trials 2-4 we investigated how the BGF may influence the avalanche effect of BGF-PRESENT, including comparisons with PRESENT original.

A. Trial 1: Testing the BGF for fixed points

For any random permutation the expected number of fixed points is 1 [**25**]. However, we are using partitions of bits from a random key and then using that to create a random permutation function, the BGF. This raises the question, *does the expected number of fixed points change unfavourably by the very process of creating the BGF?* To test this, we tried 100 random 64-bit permutations obtained from *Random.org* because they produce random numbers based on atmospheric noises, i.e. non-deterministic [**26**]. We also tested a further 100 random 64-bit permutations that were each 1-bit different from the initial 100 permutations. See *Table 1* at the end of this document.

From this small test of 200 random permutations the results appeared favourable since most fixed points (about 80 per cent) were in the range 0–4 and none were higher than 7. Therefore, we continued with further tests that would offer some idea on how the BGF might work in practice.

All good ciphers will obtain full diffusion in relatively few rounds and with fixed components this can be shown precisely. However, a random permutation will, by its very nature, be unpredictable. Therefore, since good diffusion offers a good avalanche effect, we checked for this in several ways to see if there were any immediate and obvious problems when we replace the permutation of PRESENT with the BGF. Moreover, since we are using *KR* as the means to *directly* control the initial permutation, our trials pay particular attention to this key.

B. Trial 2: BGF-PRESENT$_{KR}$ versus BGF-PRESENT$_{KR'}$

This trial consisted of forty pairs of tests (i.e. eighty in total) where we observed the Hamming distances, denoted by *d(,),* of the corresponding output ciphertexts. Each pair of tests included one encryption with KR and another with KR’ such that *d(KR, KR’) = 1* or where *KR’ = KR-compliment*. Each M, KE and KR were carefully selected and included all-zeros and particular patterns such as 0x0f…0f and 0x0…0f…f. The aim was to observe if there were any noticeable patterns between KR and its ciphertext C, and KR’ and its ciphertext C’. Our results produced an average *d(C, C’) = 29.2*. However, we noticed the keys *KR = 00…00* and *KR’ = 00…01* typically resulted in weaker d(C, C’). When we discount these potentially weak keys, the average changed to *d(C, C’) = 32.4*.

C. Trial 3: BGF-PRESENT (KR fixed) versus PRESENT

In this trial we compared fifty encryptions using BGF-PRESENT against fifty using PRESENT. The encryption key was fixed at *KE = 0* for all tests and for BGF-PRESENT we fixed *KR = 0x0123456789abcdef*. The *hand-picked* inputs M were distinct for each pair of fifty encryptions. The purpose of this experiment was to compare the Hamming distances (HD in the table) between M and the outputs of each complete round

C_r, where *r* is the round number; see *Table 2* at the end of this document. The average Hamming distances for the final outputs C (after 32 rounds) were almost the same for each cipher. For PRESENT *d(M, C) = 32.1* and for BGF-PRESENT *d(M, C) = 31.6* and therefore both ciphers were almost exactly the desired average of 32. Indeed, the tabulated results show the Hamming distances of BGF-PRESENT were almost the same as for PRESENT throughout the various round-ranges tested and again both ciphers were almost an ideal 32. In addition, the standard deviations (SD in the table) of BGF-PRESENT were slightly better (though not significantly) than those of PRESENT as they were a little closer to their average.

D. *Trial 4: BGF-PRESENT versus PRESENT with random M, K and KR*

In this trial we used 100 random 64-bit inputs M and 100 random 80-bit encryption keys KE. For BGF-PRESENT we also used 100 random 64-bit keys KR. All were obtained from *Random.org*.

As for *Trial 2*, the purpose of this experiment was to compare the Hamming distances between M and the outputs of each complete round C_r, where *r* is the round number. For each round of outputs C_r, the Hamming distances for BGF-PRESENT and PRESENT were very similar, especially when considering the same average ranges as shown in *Table 1*. The average Hamming distances for the final output C (after 31 rounds and the final XOR operation) were almost the same for each cipher. For PRESENT *d(M, C) = 31.7* and for BGF-PRESENT *d(M, C) = 32.3*.

VI. CONCLUSIONS AND FURTHER WORK

This paper explored a way to use a random symmetric encryption key as a means to create a random permutation function called the BGF. We showed by a small experiment that the BGF retained the expected number of fixed points as for any random permutation and therefore warrants further experiments. We also showed by several different experiments that replacing the permutation layer in PRESENT with the BGF did not adversely affect the Hamming distances. Indeed, the results showed a near perfect Hamming distance of n/2 in nearly all cases. Moreover, BGF-PRESENT has so far shown to offer good diffusion even though its permutation layer is not fixed. Therefore, we theorised that BGF-PRESENT may offer greater protection against attacks, such as the statistical saturation attack, that target the weaknesses in the permutation of PRESENT. These results will prompt us to do further similar experiments, with a *much* larger data set and with different model ciphers, for a future paper.

Further still, we theorised that BGF-PRESENT might be better resistant to differential-type attacks because identical inputs will not produce identical outputs and therefore differentials between rounds will not propagate in a predictable way as they do for ciphers with fixed components. However, this requires a separate and detailed investigation. Perhaps an even simpler cipher, one that focuses on random permutations, such as the Even-Mansour cipher [14] with the BGF, would be an ideal model to check for resistance to differential and linear cryptanalysis specifically.

Naturally, simple ways to improve the security of a cipher are to increase the key length and/or the number of rounds. Yet longer keys will further add to the burden of key management and more rounds may even offer *less* security [27]. Therefore, using key-created components that change dynamically could lead the way to longer use of keys (i.e. less frequent key changes) without adversely affecting security even, perhaps, with fewer rounds. No doubt, a scheme that uses KE XORed with an IV (rather than KR) to create the BGF would also be interesting future work. This will make for a more efficient scheme whilst using only one symmetric key.

REFERENCES

[1] Kerckhoffs, A. (1883) La cryptographie militaire. J sci militaires IX:5–38, 161–191. [http://www.petitcolas.net/fabien/kerckhoffs/]

[2] Barker, E. and Kelsey, J., 2015. NIST Special Publication 800-90A Revision 1. tech. rep., National Institute of Standards and Technology.

[3] Daemen, J. and Rijmen, V., 2013. The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media, pp. 41.

[4] Biham, E. and Shamir, A. "Differential Cryptanalysis of DES-like Cryptosystems," Journal of Cryptology, Vol.4, pp.3–72, (1991).

[5] Schneier, B.: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

[6] Orhun, K. and Cevat, M. A New Class of Weak Keys for Blowfish. FSE 2007.

[7] Schneier, B. et al.: Twofish: A 128-bit Block Cipher (1998) [https://www.schneier.com/paper-twofish-paper.pdf].

[8] (NIST), N. I. (1997). ANNOUNCING REQUEST FOR CANDIDATE ALGORITHM NOMINATIONS FOR THE ADVANCED ENCRYPTION STANDARD. Retrieved November 2017 from http://csrc.nist.gov/edgehive/aes/pre-round1/aes_9709.htm.

[9] Abd-ElGhafar, A.R., Diaa, A. and Mohammed, F., 2009, May. Generation of AES key dependent S-boxes using RC4 algorithm. In 13th International Conference on Aerospace Sciences & Aviation Technology (pp. 26-28).

[10] Hosseinkhani R., Haj Seyyed Javadi H. Using Cipher Key to Generate Dynamic S-box in AES Cipher System. International Journal of Computer Science and Security (IJCSS), Volume 6, issue 1. (2012).

[11] Eman Mohammed Mahmoud et al. Dynamic AES-128 with Key-dependent S-box. International Journal of Engineering Research and Applications (IJERA), Vol. 3, Issue 1, Jan-Feb (2013), pp.1662-1670.

[12] Tiessen, T., Knudsen, L.R., Kölbl, S. and Lauridsen, M.M., 2015, March. Security of the AES with a Secret S-Box. In International Workshop on Fast Software Encryption (pp. 175-189). Springer, Berlin, Heidelberg.

[13] Luby, M. and Rackoff, C., 1988. How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing, 17(2), pp.373-386.

[14] Even, S. and Mansour, Y., 1991, November. A construction of a cipher from a single pseudorandom permutation. In International Conference on the Theory and Application of Cryptology (pp. 210-224). Springer, Berlin, Heidelberg.

[15] Daemen, J., 1991, November. Limitations of the Even-Mansour construction. In International Conference on the Theory and Application of Cryptology (pp. 495-498). Springer, Berlin, Heidelberg.

[16] Dunkelman, O., Keller, N. and Shamir, A., 2012, April. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In Eurocrypt (Vol. 7237, pp. 336-354).

[17] Hall, C., Wagner, D., Kelsey, J. and Schneier, B., 1998. Building prfs from prps. In Advances in Cryptology—CRYPTO'98 (pp. 370-389). Springer Berlin/Heidelberg.

[18] Kuppusamy, Arulmani, Swaminathan Pitchai Iyer, and Kannan Krithivasan. "Two-key dependent permutation for use in symmetric cryptographic system." Mathematical Problems in Engineering 2014 (2014).

[19] Borghoff, J., Knudsen, L.R., Leander, G., Thomsen, S.S.: Cryptanalysis of PRESENT-like Ciphers with Secret S-Boxes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 270–289. Springer, Heidelberg (2011)].

[20] Gomathisankaran M., Lee, R.B.: Maya: A Novel Block Encryption Function. In: Proceedings of International Workshop on Coding and Cryptography (2009), (February 14, 2010).

[21] Knudsen, L.R., Leander, G., Poschmann, A. and Robshaw, M.J., 2010, August. PRINTcipher: A Block Cipher for IC-Printing. In CHES (Vol. 6225, pp. 16-32).

[22] Collard, B. and Standaert, F.X., 2009, April. A Statistical Saturation Attack against the Block Cipher PRESENT. In CT-RSA (Vol. 5473, pp. 195-210).

[23] In CT-RSA (Vol. 5473, pp. 195-210). (Cho, J.Y., 2010, March. Linear Cryptanalysis of Reduced-Round PRESENT. In CT-RSA (Vol. 5985, pp. 302-317).

[24] Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007).

[25] Grinstead, C.M. and Snell, J.L., 2012. Introduction to probability. American Mathematical Soc., pp. 81.

[26] Haahr, M. and Haahr, S., 2005. Retrieved September 2017 from https://www.random.org/bytes/

[27] Bhaumik, R., Dutta, A., Guo, J., Jean, J., Mouha, N. and Nikolić, I., 2015. More Rounds, Less Security? (Doctoral dissertation, Inria Paris Rocquencourt).

Table 1 Fixed points of 200 BGFs

	Number of fixed points from 100 trials							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Random bit-string 1 (non-deterministic)	12	33	24	18	9	3	0	1
Random bit-string 2 (1-bit different from *Random bit-string 1*)	15	35	22	18	8	1	0	1

Table 2 Average Hamming distances over ranges of rounds

	PRESENT		**BGF-PRESENT**	
Rounds	*HD*	*SD*	*HD*	*SD*
1-8	31.9	3.7	31.9	4.2
1-16	31.9	3.9	32.0	4.1
1-24	31.8	3.9	31.9	4.1
1-32	31.9	4.0	32.0	4.1

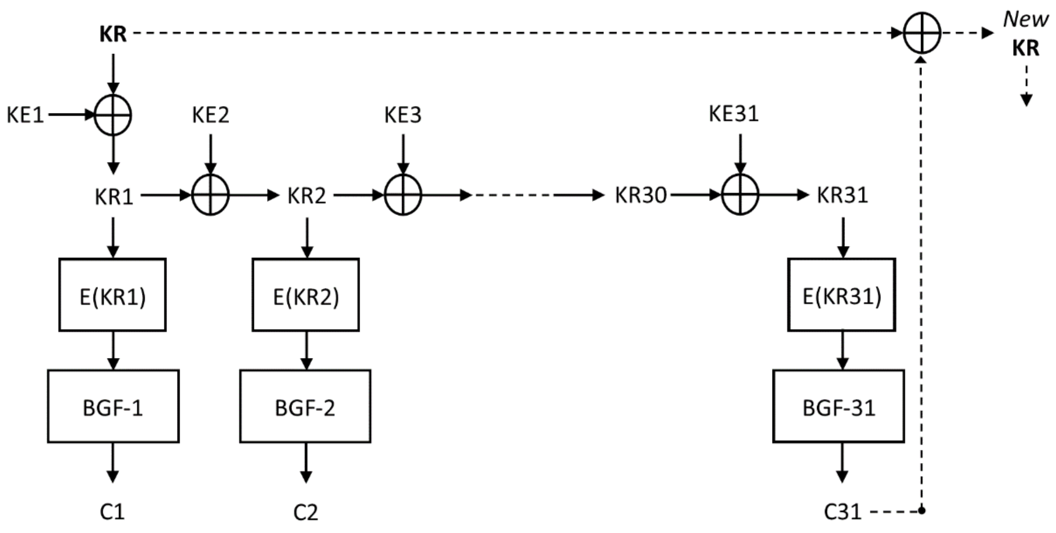

Figure 4: Key register for BGF-PRESENT

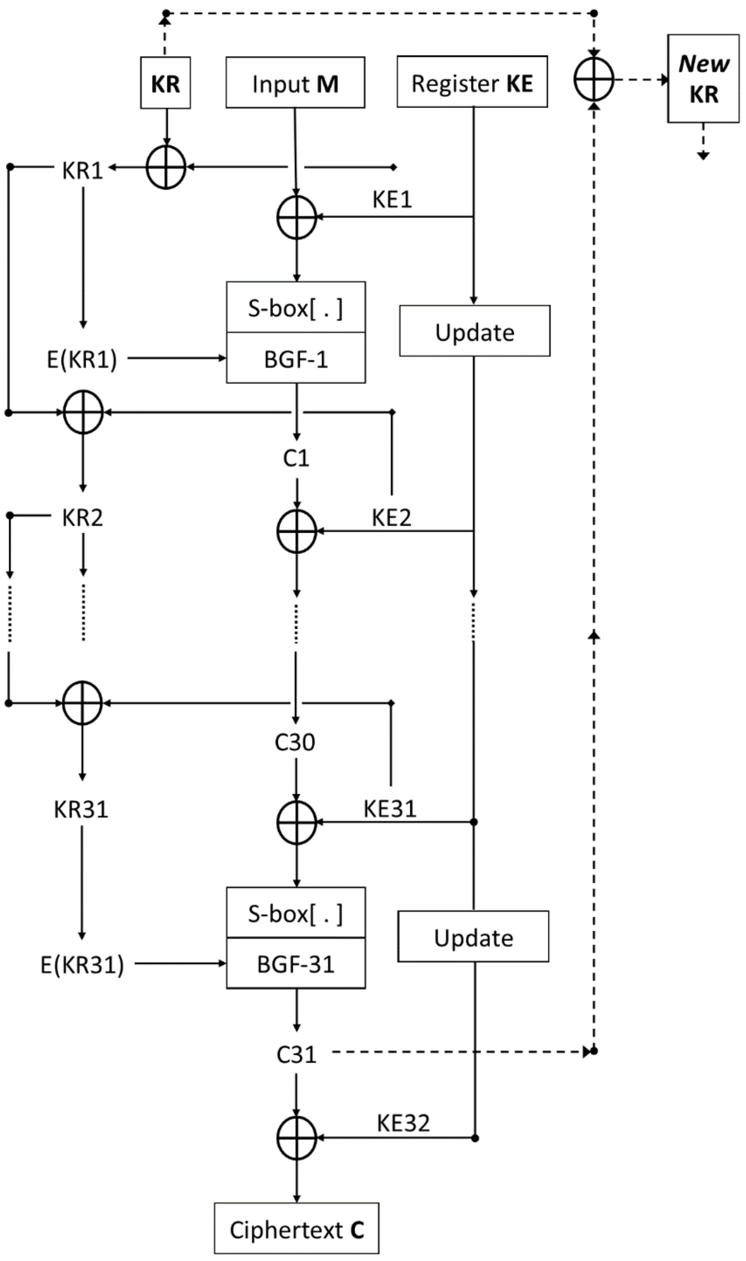

Figure 5: BGF-PRESENT with round-function components