# Notes on boomerang attack on COCONUT98 and 16-round Khufu

Project of Roman Dovgard in cryptanalysis course given by Alex Biryukov
at the Weizmann Institute of Science, Israel.

September 19, 2000

# 1 Introduction

We present a technique of cryptanalysis called boomerang attack that was invented by David Wagner and was used to cryptanalyze COCONUT98, Khufu, CAST-256 and FEAL. We show how it can be used to break COCONUT98 and 16-round Khufu. Boomerang attack is a differential style attack that uses differential characteristics of half of the cipher. COCONUT98 cipher was invented by Serge Vaudenay and was proven to be secure against differential attacks. This security proof was based on decorrelation techniques and assumed that attacker should use differential characteristic for whole cipher. This was key point of boomerang attack. Boomerang attack uses only halfway differential characteristics, and so security proof of COCONUT98 against it failed.

This paper is organized as follows. In section 2 we describe boomerang attack in general lines. In section 3 we describe COCONUT98 cipher. In section 4 we describe good differential characteristics for parts of COCONUT98 cipher. In section 5 we describe boomerang attack on COCONUT98. In section 6 we describe Khufu cipher. In section 7 we describe boomerang attack on Khufu cipher. In section 8 (summary) we note all subtle details and differences between description of boomerang attacks in paper [1] and our practical applications of these attacks (including our new observations). Finally, in section 9 we describe how to use our program, which performs cryptanalysis described in the paper. Appendix describes more precisely field $GF(2^{64})$ arithmetic used in description of COCONUT98 cipher (section 3).

# 2 Boomerang attack in general

In this section we describe boomerang attack in general lines (without any specific cipher in mind). Suppose we have some cipher whose encryption function $E$ has form $E = E_1 \circ E_0$. Also, suppose we have following differentials for halfway cipher with good probabilities:

1. $\Delta \to \Delta^*$ by $E_0$
2. $\nabla \to \nabla^*$ by $E_1^{-1}$
3. $\Delta^* \to \Delta$ by $E_0^{-1}$

for some differences $\Delta$, $\nabla$, $\Delta^*$ and $\nabla^*$. One can form a "quartet" as follows:

- Choose randomly distributed plaintext $P$.
- Define other plaintext $P'$ by $P' = P \oplus \Delta$.
- Encrypt plaintexts $P$ and $P'$ and get ciphertexts $C$ and $C'$ respectively.
- Decrypt ciphertexts $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$ and get plaintexts $Q$ and $Q'$ respectively.

Such a quartet is said to be right if characteristic 1 holds for $P$ and $P'$, characteristic 2 holds for $C$ and $D$ and $C'$ and $D'$, and finally characteristic 3 holds for $E_0(Q)$ and $E_0(Q')$. As we described procedure of generation of $P, P', C, C', D, D', Q, Q'$ it's clear that in right quartet we'll have equations

- $E_0(P) \oplus E_0(P') = \Delta^*$
- $E_0(P) \oplus E_0(Q) = \nabla^*$
- $E_0(P') \oplus E_0(Q') = \nabla^*$

and so following equation holds:

$$E_0(Q) \oplus E_0(Q') = (E_0(Q) \oplus E_0(P)) \oplus (E_0(P) \oplus E_0(P')) \oplus (E_0(P') \oplus E_0(Q')) = \nabla^* \oplus \Delta^* \oplus \nabla^* = \Delta^*$$

If characteristic 3 holds for $E(Q)$ and $E(Q')$, then we'll have just same difference between $Q$ and $Q'$ as difference between $P$ and $P'$.

# 3 COCONUT98 cipher

COCONUT98 is Feistel cipher with 64-bit block size that uses 256-bit key $K = (K_1, \dots, K_8)$ where $K_i's$ are 32-bit sub-keys. First four subkeys are used in key schedule algorithm as follows:

$$k_1 = K_1,\ k_2 = K_1 \oplus K_3,\ k_3 = K_1 \oplus K_3 \oplus K_4,\ k_4 = K_1 \oplus K_4,$$

$$k_5 = K_5,\ k_6 = K_2 \oplus K_3,\ k_7 = K_2 \oplus K_3 \oplus K_4,\ k_8 = K_2 \oplus K_4.$$

Last four keys are used in decorrelation module. The cipher is depicted in figure 1:

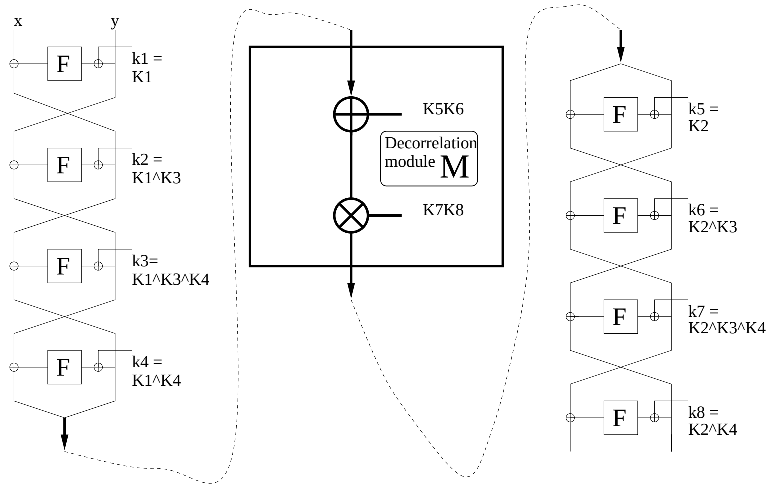


Figure 1: COCONUT98 cipher.

Here $F$ stands for the following function ($\ll$ stands for left shift like in C++):

$$F(z) = \phi\Big(\big(\phi(z) \ll 11\big) + c \bmod 2^{32}\Big)$$

where

$$\phi(z) = z + 256 \cdot S(z \bmod 256) \bmod 2^{32}$$

In the functions above, $S$ is $8 \to 24$ bit S box and $c$ is a 32-bit constants. This S box and constant $c$ are described precisely in Vaudenay's paper [2] (they look rather arbitrary - they're obtained from the first bits of expansion of $\frac{1}{e}$ in binary representation). In our experiments we generated S box table at random (however we took precise constant $c$). Bold lines indicate 64-bit values. Finally bold $\oplus$ stands for 64-bit XOR and bold $\otimes$ stands for 64-bit multiplication over $GF(2^{64})$ (see appendix). This multiplication is performed in Galois field with $2^{64}$ elements. We look at this field in polynomial representation over $GF(2)$ with irreducible polynomial $x^{64} + x^{11} + x^2 + x + 1$ (all this bold picture is called decorrelation module). As in any Feistel cipher, decryption is performed like encryption, just with reverse key schedule (inverting decorrelation module is also not a problem because field multiplication and addition are invertible). Although 4-round Feistel cipher is weak by itself, decorrelation module is "spoiling" the differences and thus is core (nut) of a cipher.

# 4 Differential characteristics for COCONUT98

Let $e_j = 2^j$, $0 \le j \le 31$, be 32-bit difference in $j^{\text{th}}$ bit. We look at $j$'s which are neither in the set $\{0, \dots, 7\}$ nor in the set

$$\{0, \dots, 7\} \gg 11 = \{21, \dots, 28\}$$

(i.e. $j$'s in the set $J = \{8, \dots, 20, 29, 30, 31\}$). For these $j$'s we have characteristic $e_j \to e_{j+11}$ for $F$ function with probability close to half. The reason is that for these $j$'s difference isn't supposed to enter $0 \dots 7$ dangerous window of neither first nor second $\phi$ and so $\phi$ will either expand our 1-bit difference (due to carry caused by adding constant S box factor) or keep it as it is. This observation is true also for another application of $\phi$ and the constant $c$ and so we'll be just interested whether all these three factors together cause a carry or not. Probability of such a carry is approximately half. There is some subtle point in the analysis above. Carry in first two additions (first S box and the constant $c$) can cause difference in the window of the second $\phi$. However this is not a problem for the most $j$'s in $J$ as they're quite far away (we look on right to left distance only as carry goes to the left) from the window $\{21, \dots, 28\}$ of the second $\phi$ and carry which can be caused must reach from $j$ to this window. As long carries have low probability for most of $j$'s in $J$ we have no problem. For most unlucky bit $j = 20$ David Wagner empirically found that this probability is 0.38 and so even for near $j$'s we're supposed to be in good shape. Also one can look (actually we don't need this here) on characteristic $e_j \oplus e_k \to e_{j+11} \oplus e_{k+11}$ where $j, k \in J$ are distinct. Similarly, this characteristic has probability close to $\frac{1}{4}$ (we have to worry here about two carries).

These high probability characteristics for $F$ cause good characteristics for halfway of the cipher (actually for both halves in both directions) which is used in boomerang attack.

# 5 Boomerang attack on COCONUT98

Main problem with COCONUT98 is decorrelation module. Although decorrelation module is affine (i.e. $M(x) \oplus M(y) = M(x \oplus y)$ for every $x$ and $y$ in $GF(2^{64})$; this is basic field distributivity property) and so we have always characteristic $\delta \to M(\delta)$ (where $M(\delta)$ is fixed!) with probability 1, we cannot mount any traditional differential attack on this cipher. This is because when $K_5, \ldots, K_8$ are unknown we don't know $M(\delta)$ and so cannot use this characteristic. Actually the problem is even more serious. First, even if we knew $M(\delta)$, it could be the case that for every nice $\delta$ (i.e. $\delta$ with low hamming weight) $M(\delta)$ isn't nice at all, and so no good characteristic for whole cipher can be found. Second, even much more serious problem, is that Serge Vaudenay proved (and here comes into the picture his decorrelation theory which was a main point in design of this cipher) that any characteristic $\delta \to \delta^*$ has averaged (w.r.t. to unknown key values $K_5, \ldots, K_8$) probability $\frac{1}{2^{64}-1}$. So, in other words, there is no hope for standard differential attack on this cipher.

However, using boomerang attack, we can exploit good differentials for first and last four rounds (which we denote by $\Psi_0$ and $\Psi_1$ respectively) of the cipher together with probability 1 characteristic $\nabla^* \to M^{-1}(\nabla^*)$ for $M^{-1}$ where output difference is unknown to the attacker. In order to see this we represent this cipher as $E_1 \circ E_0$ where $E_0$ stands for first four rounds and $E_1$ stands for decorrelation module combined with last four rounds. This can be done as follows: probability that quartet generated by procedure described above is a potentially right quartet (i.e. quartet in which main criteria $P \oplus P' = Q \oplus Q'$ holds) is exactly

$$p = \sum_{w \oplus x \oplus y \oplus z = 0} P\Big[\Delta \to w \text{ by } E_0\Big] \cdot P\Big[\nabla \to x \text{ by } E_1^{-1}\Big] \cdot P\Big[\nabla \to y \text{ by } E_1^{-1}\Big] \cdot P\Big[z \to \Delta \text{ by } E_0^{-1}\Big]$$

Now suppose that we have three characteristics (1, 2 and 3) that have probability much better than all the others <u>together</u>. In this case probability that potentially right quartet is right w.r.t. this characteristics is close to 1 and so

- We can estimate success probability (i.e. probability of potentially right quartet) by

$$p = P\Big[\Delta \to \Delta^* \text{ by } \Psi_0\Big] \cdot P\Big[\Delta^* \to \Delta \text{ by } \Psi_0^{-1}\Big] \cdot P\Big[\nabla \to \nabla^* \text{ by } \Psi_1^{-1}\Big]^2$$

  This in turn can be rewritten as

$$p = P\Big[\Delta \to \Delta^* \text{ by } \Psi_0\Big]^2 \cdot P\Big[\nabla \to \nabla^* \text{ by } \Psi_1^{-1}\Big]^2$$

  because

$$P\Big[\Delta \to \Delta^* \text{ by } \Psi_0\Big] = P\Big[\Delta^* \to \Delta \text{ by } \Psi_0^{-1}\Big]$$

  holds for standard differentials (it may <u>not</u> hold for truncated differentials as we show later). Last fact can be shown as follows:

$$P\Big[\Delta \to \Delta^* \text{ by } \Psi_0\Big] = \frac{\#\{x \text{ s.t. } \Psi_0(x) \oplus \Psi_0(x \oplus \Delta) = \Delta^*\}}{\#\{x\}}$$

$$P\Big[\Delta^* \to \Delta \text{ by } \Psi_0^{-1}\Big] = \frac{\#\{x \text{ s.t. } \Psi_0^{-1}(x) \oplus \Psi_0^{-1}(x \oplus \Delta^*) = \Delta\}}{\#\{x\}}$$

  As we're working with 64-bit blocks in both sides of $\Psi_0$ we have the same $\#\{x\}$ in the both denominators and so we just need to establish equality of numerators. However their equality is clear due to bijection $\Psi_0$ from $\{x \text{ s.t. } \Psi_0(x) \oplus \Psi_0(x \oplus \Delta) = \Delta^*\}$ to $\{x \text{ s.t. } \Psi_0^{-1}(x) \oplus \Psi_0^{-1}(x \oplus \Delta^*) = \Delta\}$.

- When we have potentially right quartet we can suppose (with high probability) that it's right quartet w.r.t. our differentials. In this case we can assume that our specific characteristics happened in both $P, P'$ and $Q, Q'$ pairs.

Now we show how to break this cipher in more details. We choose some $j \in I$ (where subset $I$ of $J$ will be defined later) and define $\Delta = \nabla = (e_{j+11}, e_j)$. There is characteristic with good probability that has zero difference after the first round on the left side (and all other characteristics have relatively small probabilities - however together they have enough probability to cause some noise in this attack). Now we generate many quartets with these $\Delta$ and $\nabla$ while success criteria is that difference between $Q$ and $Q'$ is $\Delta$. Once we have many right pairs and some guessed key, we can check that our key admits zero output difference after the first round on the left side of the block for all right pairs. If this is right we consider this key as candidate for true key. For each one of these eight $j$'s we generate 10 right quartets (i.e. 20 right pairs) that filter out many keys and keep the correct key (probability of noise is not too high and so the right key survives the filtering). We use these $8 \cdot 10 = 80$ generated right quartets in order to break both $k_1$ and $k_8$ (we generate right pairs for $k_1$ and $k_8$ recovery from the same right quartets). Empirically we see that such 80 right quartets cost us around few (1,2,3) millions encryption/decryption trials and so probability that quartet is right is around $\frac{4 \cdot 80}{1000000} = \frac{1}{3125}$ (1000000 encryption/decryption trials are equivalent to $\frac{1000000}{4}$ quartets, which should be divided by 80 due to our need in $80 = 8 \cdot 10$ right quartets). In fact this number 3125 is similar to empiric averages in the table 2.

For the rest of the attack (we recover farther $k_2$ and $k_3$ using already recovered $k_1$; rest of the keys are easily obtained due to weaknesses in the key schedule algorithm) we use slightly less right quartets and we see (also empirically) that this costs us small number of encryption/decryption trials (probability of success increases because number of rounds decreases).

Suppose we have some $j \in J$ and suppose that $e_j \to e_{j+11}$ by the first $F$ function. Then the second round is bypassed with probability 1 (just zero input difference becomes zero output difference). We consider subset $I$ of $J$ which consists of such indices that don't enter the dangerous window in first four rounds. As second round is trivially out of consideration and third round is equivalent to first one (if we bypassed the first round correctly) we must worry only about first and fourth rounds. In the fourth round we have input difference $e_{j+11}$ and so we want both $e_j$ and $e_{j+11}$ to be in $J$. So we are interested in following set:

$$I = J \cap (J \lll 11) = \{8, \dots, 20, 29, 30, 31\} \cap \{29, 30, 31, 0, \dots, 9, 18, 19, 20\} =$$

$$\{29, 30, 31, 8, 9, 18, 19, 20\}$$

For each $j$ in this set (and only for $j$'s in this set) we suppose to have nice (with small hamming weight) difference after the first four rounds. If $j \notin I$ difference after the fourth round (on the left side of the block) will look like some random garbage. As we have characteristic $\nabla \to \nabla^*$ for $E_1^{-1}$ with good probability we expect to have same difference between the pair $E_1^{-1}(C)$ and $E_1^{-1}(C')$ and the pair $E_1^{-1}(D)$ and $E_1^{-1}(D')$. So difference between $E_1^{-1}(D)$ and $E_1^{-1}(D')$ will have same random garbage as difference between $E_1^{-1}(C)$ and $E_1^{-1}(C')$.

This random garbage enters the window in the fourth round in reverse direction and so if values in forward and backward directions were different (thus picking different entries from the table) we would have completely different decryptions for $E_1^{-1}(D)$ and $E_1^{-1}(D')$ and so we would never get difference $\Delta$ between $Q$ and $Q'$. So in order to get right quartet we need to have same values (indexing same entries from the S box), in forward and backward directions and this is indeed the case. When there is no decorrelation module there will be same values because $\Delta^*$ will be equal to $\nabla^*$ and so values will be just swapped (as it happens in Khufu - look there for more detailed analysis). However we have decorrelation module and so (as $\Delta^*$ came from above without passing through decorrelation module and $\nabla^*$ came from below passing through decorrelation module we have $\Delta^* \neq \nabla^*$ and so there is no reason for same values). Hence with decorrelation module attack won't work if we take $j \notin I$.

So we always take always $j \in I$. Probability of right quartet for fixed $j$ can be estimated as follows. In one quartet we have four identical characteristics, and in each characteristic we use twice $e_j \stackrel{F}{\to} e_{j+11}$ (in first and third rounds). So probability of characteristic is

$$P[e_j \stackrel{F}{\to} e_{j+11}]^8 \tag{1}$$

Probabilities $P[e_j \stackrel{F}{\to} e_{j+11}]$ for different values of $j \in J$ (for $j \notin J$ these values were checked to be zero) are shown in table 1. Table 2 gives us results found using expression (1) and some more empiric results about success probabilities for different values of $i \in I$.

In theory key filtration routine has a $\frac{1}{2}$ chance of filtering wrong keys for each right quartet. In practice more careful approach is required since some "wrong" quartets have non-negligible probability to survive the test (these don't follow the specified characteristic, but follow some other low hamming weight characteristic). For example they allow carry in the first round and thus such quartet will filter out the correct key.

The other problem is that of clustering: we can't divide the key space till we arrive at a single correct key value - some small cluster of wrong key values (differ in a particular byte) always survives all the quartet filtration.

These differences between theory and practice caused us to use multiple $j$'s instead of single $j$ (as suggested in [1]) in order to decrease the cluster size.

When there is a decorrelation module the attack doesn't give us correct $k_1$ and $k_8$ (and so attack fails). In fact, (this information was found empirically; here situation is similar for $k_1$ and $k_8$ and so we claim same thing about both of them) it gives correct key bits except for the bits of second byte from the right. This was found empirically as follows: we ran key search from 0 up to 0xFFFFFFFF and got key value 0xC0F70091 and after that ran key search from 0xFFFFFFFF down to 0 and got key value 0xC0F73E91 (all this with randomly chosen seed 48). As correct key value was 0xC0F70491 we conclude that attack finds all bytes correctly except for second byte from the right (so 256 possibilities are still left). Anyway, we can perform the rest of key recovery, but this will give us 256 factor for the rest of the attack (we need to duplicate attack on $k_2$ 256 times - if program finds $k_2$ we suppose that $k_1$ was found correctly and only then we proceed to $k_3$ recovery).

| $j \in J$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| Probability | 0.49 | 0.5 | 0.5 | 0.5 | 0.49 | 0.49 | 0.48 | 0.49 |
| $j \in J$ | 16 | 17 | 18 | 19 | 20 | 29 | 30 | 31 |
| Probability | 0.48 | 0.46 | 0.43 | 0.37 | 0.48 | 0.43 | 0.37 | 0.49 |

Table 1: Probabilities of characteristics of the F function for different values of $j \in J$ (we used arbitrarily chosen seed 74 while calculating this table).

| $i \in I$ | 8 | 9 | 18 | 19 | 20 | 29 | 30 | 31 | average |
|---|---|---|---|---|---|---|---|---|---|
| Semi-empiric with seed 44 | 242 | 244 | 791 | 1946 | 158 | 625 | 3353 | 243 | 950 |
| Empirical with seed 44 | 3083 | 2779 | 4671 | 12344 | 772 | 2295 | 16020 | 2266 | 5529 |
| Semi-empiric with seed 45 | 229 | 256 | 686 | 2358 | 471 | 656 | 2612 | 151 | 927 |
| Empirical with seed 45 | 2103 | 2303 | 3566 | 14933 | 4675 | 4588 | 14952 | 1327 | 6056 |
| Semi-empiric with seed 23 | 297 | 214 | 800 | 2378 | 333 | 740 | 2732 | 444 | 992 |
| Empirical with seed 23 | 4189 | 3844 | 8374 | 13068 | 4825 | 7777 | 18825 | 3706 | 8076 |
| Semi-empiric with seed 26 | 250 | 233 | 681 | 2934 | 407 | 807 | 2634 | 281 | 1028 |
| Empirical with seed 26 | 3652 | 6389 | 6911 | 17757 | 3345 | 8581 | 8386 | 2764 | 7223 |
| Semi-empiric with seed 74 | 272 | 223 | 791 | 2703 | 354 | 835 | 2450 | 298 | 991 |
| Empirical with seed 74 | 1664 | 2868 | 5169 | 12645 | 2717 | 3162 | 14484 | 1826 | 5567 |

Semi-empiric—calculated via expression (1), where probabilities of characteristics of $F$ were calculated using simulation of $F$ function. Empirical—calculated by inspection of program execution.

Table 2: One over probabilities of COCONUT98 right quartets for different values of $i \in I$.

# 6 Khufu with 16-rounds

Khufu cipher [3] is depicted in figure 2[1]. For each one of the two S boxes we have eight rounds. Each box indicates 8-bits. These two S boxes take us from 8 bits to 32 bits in such a way that each byte is a permutation. This cipher decomposes naturally to $E_1 \circ E_0$ where $E_0$ is part which uses the first S box and $E_1$ is part which uses the second S box. So the keys of this cipher are $k_1, k_2, k_3, k_4$ and the two S boxes.

# 7 Boomerang attack on 16-round Khufu

Here we describe how boomerang attack can be used to break 16-round Khufu without auxiliary keys. We fix some 8-bit constant $a$. For this constant we have characteristic (with truncated differentials)

$$\Delta = (0,0,0,a,b,c,d,e) \to (0,0,0,a,0,0,0,0) = \Delta^*$$

for forward direction with probability $2^{-32}$ (we have to guess correctly in $(b,c,d,e)$ output difference after first S box) and differential

$$\Delta^* = (0,0,0,a,0,0,0,0) \to (0,0,0,a,b,c,d,e) = \Delta$$

for inverse direction with probability 1 (this is because difference on the left side $(0,0,0,a)$ will propagate upwards). Also we use characteristic

$$\nabla = (0,0,0,a,0,0,0,0) \to (0,0,0,a,f,g,h,i) = \nabla^*$$

of probability 1 for $E_1^{-1}$. We choose quartet according to these $\Delta$ and $\nabla$ and say that it's potentially right if difference between $Q$ and $Q'$ is $(0,0,0,a)$ in the left side of the block (i.e. if $Q \oplus Q' \in \Delta$). If we didn't bypass the first round successfully probability of any characteristic will be extremely low and so we can approximate success probability by probability of such characteristic and suppose that any potentially right quartet is right w.r.t. to our choice of characteristics. Therefore success probability is

$$p = \sum_{w \oplus x \oplus y \oplus z = 0} P\Big[\Delta \to w \text{ by } E_0\Big] \cdot P\Big[\nabla \to x \text{ by } E_1^{-1}\Big] \cdot P\Big[\nabla \to y \text{ by } E_1^{-1}\Big] \cdot P\Big[z \to \Delta \text{ by } E_0^{-1}\Big] \approx$$

$$P\Big[\Delta \to \Delta^* \text{ by } E_0\Big] \cdot P\Big[\Delta^* \to \Delta \text{ by } E_0^{-1}\Big] \cdot P\Big[\nabla \to \nabla^* \text{ by } E_1^{-1}\Big]^2 \cdot$$

$$P\Big[w \oplus x \oplus y \in \Delta^* \text{ s.t. } w \in \Delta^*, x, y \in \nabla^*\Big] = 2^{-32}$$

The last step follows from calculation of probabilities above and from the fact that

$$P\Big[w \oplus x \oplus y \in \Delta^* \text{ s.t. } w \in \Delta^*, x, y \in \nabla^*\Big] = 1$$

This fact holds because blocks $(e,f,g,i)$ in both $x$ and $y$ are the same. The last statement can be proved as follows. We'll denote left side of a block by $l(\cdot)$ and right side by $r(\cdot)$. Differences between

[1] In real Khufu byte-rotations are different between rounds, however this doesn't affect attacks described here. We also skip the details of the key schedule since they are irrelevant to this presentation.

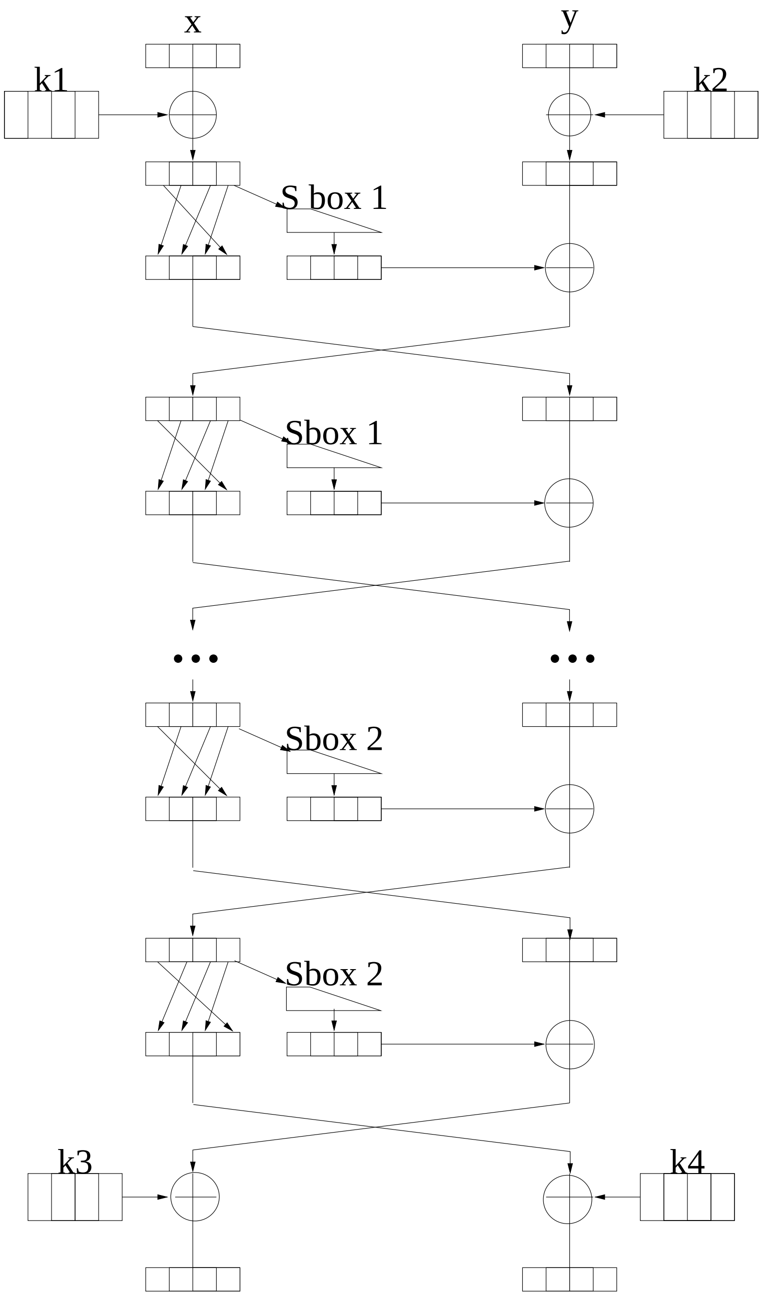


Figure 2: Khufu cipher.

$$l(E_0(Q)) \text{ and } l(E_0(P))$$

$$l(E_0(P)) \text{ and } l(E_0(P'))$$

$$l(E_0(P')) \text{ and } l(E_0(Q'))$$

are all $(0,0,0,a)$ and so

$$l(E_0(Q)) = l(E_0(P'))$$

$$l(E_0(P)) = l(E_0(Q'))$$

So left halves of pairs $(P,Q)$ and $(Q',P')$ are the same (and thus they index the same entries from the S box) and so output difference of the S boxes in first rounds are the same for both pairs. Hence right halves have to be the same (as they're equal to these output differences; this always happens because we started from difference $\nabla = (0,0,0,a,0,0,0,0)$ in both ciphertext pairs).

Now we choose a pool of nearly (actually we found that slightly larger number 80000 gives better performance) $2^{16}$ plaintexts $P_i = (L, R_i)$ and $P'_j = (L', R'_j)$ where $L, L'$ are fixed with difference $(0,0,0,a)$. For each such plaintext we decrypt $D = C \oplus \Delta$ where $C$ is corresponding ciphertext (and get plaintext $Q$). Then we sort the pool of ciphertexts $C'_j$. Then, using binary search, we "intersect" $\{C_i\}$ with $\{C'_j\}$ in that sense that we record all pairs $(i,j)$ s.t. $Q_i \oplus Q'_j = \Delta$. These pairs indicate for us potentially right pairs. Empirically we see that this procedure should give us few potentially right pairs (although there is probability 0.2 or 0.35 (respectively to 80000 or $2^{16}$ pool size) to get no potential right pairs at all - in this case we just give up; as probability of right quartet is $2^{-32}$ and we look on $\approx 2^{32}$ pairs and so it's natural to have few right pairs on average). After we have potentially right pair $(P,P')$ we can construct more $2^{10}$ (and even $2^{24}$ - but we don't need so much) pairs (that will be right if original pair was right as we'll bypass successfully the first round in forward direction and all other characteristics hold with probability 1) by looking on $Q, Q'$ corresponding to $(P,P') \oplus (\alpha,0,0,0,0,0)$ where $\alpha$ is any random 24-bit number. Now we check that these constructed $Q, Q'$ satisfy $Q \oplus Q' \in \Delta$ and in this way we filter out more wrong pairs. So, if there was right pair in the list, we find this right pair with corresponding to it $2^{10}$ more right pairs. For all these right pairs we have high probability characteristic $\Delta \to \Delta^*$ in mind and so with high probability all these right pairs will bypass first round with zero difference on the left side of the block and so for each such pair we'll get equation of the type (reason that we chose pair $Q, Q'$ (and even didn't think of $P, P'$) is that all this $2^{10}$ pairs $P, P'$ cause the same equation on $S_1$ (because their five last entries are constant)) $S_1(r) \oplus S_1(s) = t$ for $r$, $s$ and $t$ dictated by this right pair ($r$ and $s$ are right bytes in left halves of the plaintexts in both pairs, while $t$ is difference in right half between the two plaintexts).

Now, using these equations we want to recover $S_1$ up to 32-bit output XOR. The main pitfall is that $r$ and $s$ aren't arbitrary (in fact if they were we would be able to recover $S_1$ with some small constant multiple of 256 equations). In fact they're related via equation $r \oplus t = a$ because input difference in any right pair (and even potentially right pair) is $(0,0,0,a)$ in left half of the block. To overcome this problem we repeat the whole process with $a = e_j$ where $0 \le j \le 7$ (here we can save $\frac{1}{4}$ trials by taking the same pool of $P_i$'s). These $a$'s are linearly

| Pool size | Probability of right quartet | | |
|---|---|---|---|
| | empiric | half empiric | theoretic |
| $2^{16}$ | 0.65 | 0.65 | 0.63 |
| 80000 | 0.77 | 0.78 | 0.77 |
| $2^{17}$ | 1.00 | 0.99 | 0.98 |

empiric—was estimated while analyzing program performance on 100 trials, half empiric—calculated empirically using birthday paradox simulation procedure, theoretic—calculated using theoretic birthday paradox argument.

Table 3: Probabilities of Khufu right quartets for different pool sizes.

independent and so knowledge of $S_1(x) \oplus S_1(y)$ for any $x$ and $y$ with difference $a$ for some any such $a$ will be sufficient to recover $S_1$ up to 32-bit output XOR.

In the same way we recover $S_2$ up to 32-bit output XOR. This time we go from below upwards. Empirically we saw that number of chosen plaintext/ciphertext trials is $4 \sim 5$ million. Simple theoretical calculation shows that it's in fact similar to what we should get. Dividing by two (as we have two S boxes) and multiplying by $\frac{4}{3}$ (as we saved this factor with choice of the same $P$'s for all different $a$'s) we get around $\frac{8}{3}$ millions. We have 7 $a$'s and so we divide this quantity by 7 and get $\frac{8}{21}$ millions (so this is empirical estimate per one $a$). For each $a$ we have two pools each with 80000 elements and so $4 \cdot 80000 = 320000$ trials. Probability of success of such pool (i.e. probability that we'll have element in first pool and element in second pool s.t. their right half block sum is certain correct value is empirically found to be $\approx 0.8$; actually it can be calculated easily via a birthday paradox - we can reformulate the problem as we're looking for intersection of two sets, but we prefer to check empirically to be on the safe side) and so per one $a$ we use in average $\frac{10}{8} \cdot 320000 = 400000$ trials. We see that practical estimate $\frac{8}{21}$ millions and theoretical estimate 400000 are almost identical and so all results are natural. A more accurate estimates of probability of right quartet can be found in table 3.

### 7.1 Finding the last 32-bits of the S boxes

Now we try to recover 32-bit XORs of $S_1$ and $S_2$. We'll deal with $S_1$ (the same way we can deal with $S_2$). Lets denote output 32-shift of $S_1$ by $c = c_1c_2c_3c_4$ and treat the cipher as we XOR constant $c$ after each implication of $S_1$ (and so $S_1$ is totally known to the attacker). First we'll show how to recover $c_4$. We take some relation $S_1(x) \oplus S_1(y) = z$ that we already assess from the boomerang attack. We look on differentials $a, b, c, d, e$ where $a = x \oplus y$ and $(b, c, d, e) = z$. We take any 32-bit values $x$ and $x'$ with difference $(e, b, c, d)$ and denote their $S_1$ output difference[2] by $(f, g, h, i)$. Also we take arbitrary 32-bit values $y_k$ and $y'_k$ with difference $(f, g, h, i \oplus a)$ where $0 \le k \le 255$ and rightmost byte of $y_k$ is $k$. Thereafter we define $P = (x, y)$ and $P' = (x', y')$. For this pair we observe differential $(0, 0, 0, a, b, c, d, e)$ at the second round. This allows us, using boomerang idea, to check whether input difference in the left half of the third round is zero. As input difference $a$ can cause output difference $(b, c, d, e)$ (as $S_1(x) \oplus S_1(y) = z$), trying all the $k$'s we'll find a pair for which there is zero

[2] We know $S_1$ and so we know this output difference.

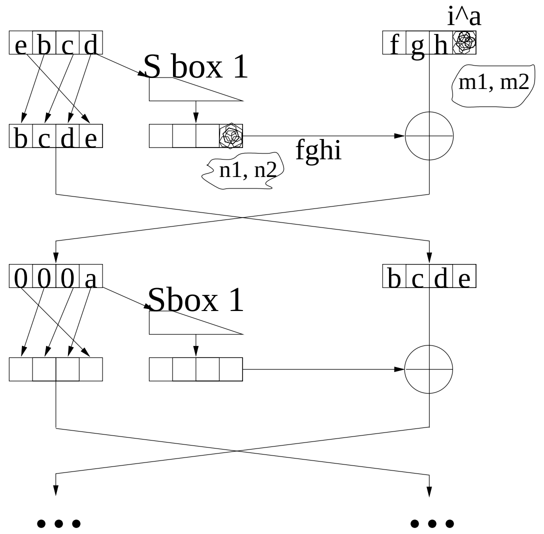


Figure 3: Schematic attack on $c_4$

in the left half of the third round (because when we go among all the $k$'s we try all inputs to $S_1$ in the second round). Moreover, with high probability, there are only two relations of the type $S_1(x') \oplus S_1(y') = z$ with $x' \oplus y' = a$ (these are relations $S_1(x) \oplus S_1(y) = z$ and $S_1(y) \oplus S_1(x) = z$; there are only $2^8$ output differences of pairs $(x', y')$ where $x' \oplus y' = a$ in the domain of $2^{32}$ possible output differences - so probability of more than two such relations is $\approx 2^{-24}$). Hence, with high probability, we'll find $k$ for which inputs in the second round are $(x, y)$ or $(y, x)$. Therefore we'll be able to form equation (where $m_1, m_2, n_1, n_2$ are 8-bit values in the both encryptions that are depicted in the figure)

$$\{m_1 \oplus n_1 \oplus c_4, m_2 \oplus n_2 \oplus c_4\} = \{x, y\}$$

where the only unknown is $c_4$. So we're able to recover $c_4$ up to XOR with $a$. Now, trying another $a$, we identify $c_4$ uniquely (so we used $512 = 2 \cdot 256$ quartets to recover $c_4$—i.e. we use 2048 encryption/decryption trials for $c_4$ recovery).

After we know $c_4$ first round is almost transparent (OK we didn't recover $c_1, c_2$ and $c_3$ yet) and we can arrange the same thing we arranged in first two rounds in rounds 2 and 3. We know output difference of S box in first round. Also we can know output difference of S box in second round (this is because we know exactly input to this S box; the last holds because we recovered $c_4$ which was the only obstacle to this task). So we can find right quartet. However we see that this whole story doesn't help at all. This is because $c_1, c_2, c_3$ don't enter the third S box. So we proceed farther. We want to do this for third and fourth rounds. As only byte of $c$ entering first three S boxes is $c_4$ (OK this automatic: if this wasn't the case we would deal with earlier rounds) we know inputs to first three S boxes and so can predict output differences there. Hence we can arrange difference $(0, 0, 0, a, b, c, d, e)$ for arbitrary $a, b, c, d, e$ in the fourth round. After recovering some right quartet (all this story should be similar to what we had for $c_4$ recovery) we can form equation on $S_1$ which will contain $c_4$ and $c_1$ (it's clear why $c_4$ will enter; $c_1$ will enter either because it is carried from the first S box passing through rotation in the middle). This way we recover $c_1$. So with 2048 trials we recover $c_1$.

Same way we recover $c_2$ and $c_3$ and so in total we use $4 \cdot 2048 = 2^{13}$ trials to recover $c$. Therefore, in total, for both S boxes we should have $2^{14}$ trials in two 32-bit shifts recovery (we see that it's much less than what we needed in boomerang attack and so we can ignore this in our analysis of data complexity).

# 8 Summary

In this paper we analyzed boomerang attack suggested in [1] through applying it on COCONUT98 and Khufu ciphers. For COCONUT98 we saw some differences between theory and practice. For Khufu we saw some incompleteness in the analysis which however are not fatal. Purpose of this summary is to describe precisely our empirical results and new observations.

Applying boomerang attack on COCONUT98 we saw that single $j = 31$ (as was suggested) isn't sufficient, since filtration of wrong keys based on carry is not perfect. When COCONUT98 without decorrelation module is considered, attack begins to work only when we use 6-8 $j$'s. When usual COCONUT98 is considered, not all wrong keys are filtered even if we take all $j$'s in $I$ (big unfiltered cluster of around 256 wrong keys still survives the filtering). Still our improved filter is capable of finding 24 key bits of 32. We believe that better analysis and more careful filtration is possible. Also we note that filtering is limited since with some probability correct key also can be filtered out.

In Khufu, original paper broke two S boxes up to two 32-bit output constant XORs (one for each S box) and under assumption that there are no auxiliary keys at the beginning and at the end of the cipher. In subsection 7.1 we described cryptanalysis of these output XORs. Ideas presented in this subsection can be used to cryptanalyze auxiliary keys as well.

# 9 Program

We wrote a program "boomerang.cpp" that performs the cryptanalysis described above. To compile this program (under UNIX) do

```
g++ -O4 boomerang.cpp
```

After that, there are three possibilities to run this program:

1. 
   ```
   a.out COCONUT98 seed
   ```
   This option will run boomerang attack on COCONUT98. Parameter "seed" will be used to initialize random number generators which are used in the program (same holds for Khufu). Vary DECORRELATION_MODULE definition in order to switch from COCONUT98 with and without decorrelation module. Also vary SEARCH_RANGE definition in order to have debugging or real version of the key search (debugging version will just search the key in some region around the true key). Don't forget to disable DEBUG definition (otherwise program will check some auxiliary procedures). Finally there is an option PRINT that enables some useful debugging "printf's" in the program.

2. 

a.out Khufu seed

This option will run boomerang attack on Khufu. Here we break $S_1$ and $S_2$ up to 32-bit shift (we don't implement $c_4$ recovery) under assumption that we have cipher without auxiliary keys $k_1$, $k_2$, $k_3$ and $k_4$.

3. 

a.out Birthday seed

This will check birthday paradox, while averaging the result among "seed" trials.

# References


[1] D. Wagner, *The Boomerang Attack*, FSE'99, Springer-Verlag, 1999.

[2] S. Vaudenay, *Provable Security for Block Ciphers by Decorrelation*, STACS'98, Springer-Verlag LNCS 1373, 1998.

[3] R. C. Merkle, *Fast software encryption functions*, In Advances in Cryptology: Proceedings of CRYPTO'90, pages 476–501. Springer-Verlag, Berlin, 1991.

[4] L.R. Knudsen, *Truncated and Higher Order Differentials*, Fast Software Encryption, 2nd international Workshop Proceedings, Springer-Verlag, 1995.


# Appendix: $GF(2^{64})$ operations

Every element in $GF(2^{64})$ (Galois field with $2^{64}$ elements) is represented as a polynomial over $GF(2)$ of degree at most 63. Addition $\oplus$ is just XOR of coefficients. Multiplication $\otimes$ is polynomial multiplication (with coefficients over $GF(2)$) followed by taking the result modulo irreducible polynomial $m(x) = x^{64} + x^{11} + x^2 + x + 1$. In other words $a \otimes b = r$ where $a \cdot b = q \cdot m + r$ with $\deg r < \deg m = 64$. In decorrelation module described in section 3 we need to find inverse polynomial corresponding to the polynomial of the key in order to perform decryption. Inverse element can be found via extended Euclidean algorithm, which we describe below. Suppose we have some $a \neq 0$ in the field. Then extended Euclidean algorithm, when applied on pair $(a, m)$, gives us pair $(x, y)$ such that

$$x \cdot a + y \cdot m = \gcd(a, m)$$

As $m$ is irreducible and $\deg a < \deg m$ we have $\gcd(a, m) = 1$ and so

$$x \cdot a + y \cdot m = 1.$$

Taking modulo $m$ we obtain

$$x \cdot a \equiv 1 (\bmod\ m)$$

and so

$$x \otimes a = 1$$

i.e. $x = a^{-1}$.

# Appendix: Extended Euclidean algorithm

Input: polynomials $a$ and $b$. Output: polynomials $x$ and $y$ such that $\gcd(a, b) = x \cdot a + y \cdot b$. Algorithm:

Step 1. Let $x = v = 1$ and $y = u = 0$.

Step 2. Determine polynomials $q$ and $r$ such that $a = q \cdot b + r$ and $\deg r < \deg b$.

Replace (simultaneously)

$a$ by $b$ and $b$ by $r$,

$x$ by $u$ and $y$ by $v$,

$u$ by $x - q \cdot u$ and $v$ by $y - q \cdot v$.

Step 3. Repeat Step 2 until $b$ equals 0.

Step 4. Return x and y.