New Results on Boomerang and Rectangle Attacks[*]

Eli Biham[†] Orr Dunkelman[‡] Nathan Keller[§]

Abstract

The boomerang attack is a new and very powerful cryptanalytic technique. However, due to the adaptive chosen plaintext and ciphertext nature of the attack, boomerang key recovery attacks that retrieve key material on both sides of the boomerang distinguisher are hard to mount. We also present a method for using a boomerang distinguisher, which enables retrieving subkey bits on both sides of the boomerang distinguisher. The rectangle attack evolved from the boomerang attack.In this paper we present a new algorithm which improves the results of the rectangle attack.

Using these improvements we can attack 3.5-round SC2000 with 2^{67} adaptive chosen plaintexts and ciphertexts, and 10-round Serpent with time complexity of $2^{173.8}$ memory accesses (which are equivalent to $2^{165.3}$ Serpent encryptions) with data complexity of $2^{126.3}$ chosen plaintexts.

1 Introduction

Differential cryptanalysis [3] is based on studying the propagation of differences through an encryption function. Since its introduction many techniques based on it were introduced. Some of these techniques, like the truncated differentials [12] and the higher order differentials [2, 12], are generalizations of the differential attack. Some other techniques like differential-linear attack [16] and the boomerang attack [21] use the differential attack as a building block.

The boomerang attack is an adaptive chosen plaintext and ciphertext attack. It is based on a pair of short differential characteristics used in a specially built quartet. In the attack a pair of plaintexts with a given input difference are encrypted. Their ciphertexts are used to compute two other ciphertexts according to some other difference, these new ciphertexts are then decrypted, and the difference after the decryption is compared to some (fixed known) value.

The boomerang attack was further developed in [11] into a chosen plaintext attack called the amplified boomerang attack. Later, the amplified boomerang attack was further developed into the rectangle attack [7].

In the transition from the boomerang attack to the rectangle attack the probability of the distinguisher is reduced (in exchange for easing the requirements from adaptive chosen plaintext and ciphertext attack to a chosen plaintext attack). The reduction in the distinguisher's probability results in higher data complexity requirements. For example, the data requirements for distinguishing a 2.5-round SC2000 [19] from a random permutation using a rectangle distinguisher is $2^{84.6}$ chosen plaintext blocks, whereas only $2^{39.2}$ adaptive chosen plaintext and ciphertext blocks are required for the boomerang distinguisher.

In this paper we present a method to retrieve more subkey bits in key recovery boomerang attacks. We also present a better algorithm to perform rectangle attacks. These improvements

[*]The work described in this paper has been supported by the European Commission through the IST Programme under Contract IST-1999-12324.

[†]Computer Science department, Technion - Israel Institute of Technology, Haifa 32000, Israel, biham@cs.technion.ac.il, http://www.cs.technion.ac.il/~biham/.

[‡]Computer Science department, Technion - Israel Institute of Technology, Haifa 32000, Israel, orrd@cs.technion.ac.il, http://vipe.technion.ac.il/~orrd/me/.

[§]Mathematics department, Technion - Israel Institute of Technology, Haifa 32000, Israel, nkeller@tx.technion.ac.il.

result in better key recovery attacks which require less data or time (or both) and are more effective. The improvement to the generic rectangle attack reduces the time complexity of attacking 10-round Serpent from 2^{217} memory accesses[1] to $2^{173.8}$ memory accesses which are equivalent to about $2^{166.3}$ 10-round Serpent encryptions. We also prove that these key recovery attacks succeed (with very high probability) assuming that the distinguishers are successful.

The paper is organized as follows: In Section 2 we briefly describe the boomerang and the rectangle attacks. In Section 3 we present our new optimized generic rectangle attack and analyze its application to generic ciphers and to SC2000 and Serpent. In Section 4 we present our optimized generic boomerang attack and analyze its application to both a generic cipher and real blockciphers like SC2000 and Serpent. Section 5 describes a new technique to transform a boomerang distinguisher into a key recovery attack that retrieves more subkey material. We summarize this paper and our new results in Section 6.

2 Introduction to the Boomerang and the Rectangle Attacks

2.1 The Boomerang Attack

The boomerang attack was introduced in [21]. The main idea behind the boomerang attack is to use two short differentials with high probabilities instead of one differential of more rounds with low probability. The motivation for such an attack is quite apparent, as it is easier to find short differentials with a high probability than finding a long one with a high enough probability.

We assume that a block cipher $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ can be described as a cascade $E = E_1 \circ E_0$, such that for E_0 there exists a differential $\alpha \rightarrow \beta$ with probability p, and for E_1 there exists a differential $\gamma \rightarrow \delta$ with probability q. The boomerang attack uses the first characteristic $(\alpha \rightarrow \beta)$ for E_0 with respect to the pairs (P_1, P_2) and (P_3, P_4), and uses the second characteristic $(\gamma \rightarrow \delta)$ for E_1 with respect to the pairs (C_1, C_3) and (C_2, C_4). The attack is based on the following boomerang process:

- Ask for the encryption of a pair of plaintexts (P_1, P_2) such that $P_1 \oplus P_2 = \alpha$ and denote the corresponding ciphertexts by (C_1, C_2).
- Calculate $C_3 = C_1 \oplus \delta$ and $C_4 = C_2 \oplus \delta$, and ask for the decryption of the pair (C_3, C_4). Denote the corresponding plaintexts by (P_3, P_4).
- Check whether $P_3 \oplus P_4 = \alpha$.

We call these steps (encryption, XOR by *delta* and then decryption) a $\delta-$shift, as each pair is encrypted, and its corresponding ciphertexts are shifted by δ, and finally decrypted.

It is easy to see that for a random permutation the probability that the last condition is satisfied is 2^{-n}. For E, however, the probability that the pair (P_1, P_2) is a right pair with respect to the first differential $(\alpha \rightarrow \beta)$ is p. The probability that both pairs (C_1, C_3) and (C_2, C_4) are right pairs with respect to the second differential is q^2. If all these are right pairs, then they satisfy $E_1^{-1}(C_3) \oplus E_1^{-1}(C_4) = \beta = E_0(P_3) \oplus E_0(P_4)$, and thus, with probability p also $P_3 \oplus P_4 = \alpha$. Therefore, the total probability of this quartet of plaintexts and ciphertexts to satisfy the boomerang conditions is $(pq)^2$. Therefore, $pq > 2^{-n/2}$ must hold for the boomerang distinguisher (and the boomerang key recovery attacks) to work.

The attack can be mounted for all possible β's and γ's simultaneously (as long as $\beta \neq \gamma$), thus, a right quartet for E is built with probability $(\hat{p}\hat{q})^2$, where:

$$\hat{p} = \sqrt{\sum_{\alpha \xrightarrow{\beta} \beta} \Pr{}^2[\alpha \rightarrow \beta]},$$

[1]In [7] it was claimed to be 2^{205} due to an error that occurred in the analysis.

and

$$\hat{q} = \sqrt{\sum_{\substack{\gamma \\ \gamma \to \delta}} \Pr{}^2[\gamma \to \delta]}.$$

We refer the reader to [21, 7] for the complete description and the analysis.

2.2 The Rectangle Attack

Converting adaptive chosen plaintext and ciphertext distinguishers into key recovery attacks pose several difficulties. Unlike the regular known plaintext, chosen plaintext, or chosen ciphertext distinguishers, using the regular methods of [3, 17, 12, 4, 5, 16] to use adaptive chosen plaintext and ciphertext distinguishers in key recovery attacks fail, as these techniques require the ability to directly control either the input or the output of the encryption function.

In [11] the amplified boomerang attack is presented. This is a method for eliminating the need of adaptive chosen plaintexts and ciphertexts. The amplified boomerang attack achieves this goal by encrypting many pairs with input difference α, and looking for a quartet (pair of pairs) for which, $C_1 \oplus C_3 = C_2 \oplus C_4 = \delta$ when $P_1 \oplus P_2 = P_3 \oplus P_4 = \alpha$. Given the same decomposition of E as before, and the same basic differentials $\alpha \to \beta, \gamma \to \delta$, the analysis shows that the probability of a quartet to be a right quartet is $2^{-(n+1)/2}pq$.

The reason for the lower probability is that no one can guarantee that the γ difference (in the middle of the encryption; needed for the quartet to be a right boomerang quartet) is achieved. The lower probability makes the additional problem (already mentioned earlier) of finding and identifying the right quartets even more difficult.

The rectangle attack [7] shows that it is possible to count over all the possible β's and γ's, and presents additional improvements over the amplified boomerang attack. The improvements presented in the rectangle attack improve the probability of a quartet to be a right rectangle quartet to $2^{-n/2}\hat{p}\hat{q}$.[2]

3 Improving the Rectangle Attack

In this section we improve the rectangle attack. We cannot increase the probability of a rectangle distinguisher over the probabilities given in [7]. However, we can improve the key recovery attack which exploits the rectangle distinguisher.

The main problem dealt in previous works is the large number of possible quartets. Unlike in the boomerang attack, in which the identification of possible quartets is relatively simple, it is hard to find the right quartets in the rectangle attacks since the attacker encrypts a large number of pairs (or structures) and then has to find the right quartets through analysis of the ciphertexts. As the number of possible quartets is quadratic in the number of pairs[3], and as the attacker has to test all the quartets, it is evident that the time complexity of the attack is very large.

In this section we present an algorithm which solves the above problem by exploiting the properties of a right quartet, and therefore tests only a small part of the possible quartets. The new algorithm is presented on a generic cipher with the following parameters: Let E be a cipher which can be described as a cascade $E = E_f \circ E_1 \circ E_0 \circ E_b$, and assume that E_0 and E_1 satisfy the properties of E_0 and E_1 presented in Section 2 (i.e., there exist differentials $\alpha \to \beta$ with probability p of E_0 and $\gamma \to \delta$ with probability q of E_1). An outline of such an E is presented in Figure 1. We can treat this E as composed of $E' = E_1 \circ E_0$ (for which we have a distinguisher) surrounded by the additional rounds of E_b and E_f. As mentioned in Section 2, for sufficiently high probabilities $\hat{p}, \hat{q}$, we can distinguish $E_1 \circ E_0$ from a random permutation using either a boomerang or a rectangle distinguisher. However, we also like to mount key recovery attacks on the full E.

[2]This is a lower bound for the probability. For further analysis we refer the reader to [7].

[3]There are about N^2 possible quartets if N is the number of ciphertexts (rather than $N^2/2$), since the quartet $[(x,y),(z,w)]$ is different from the quartet $[(x,y),(w,z)]$.

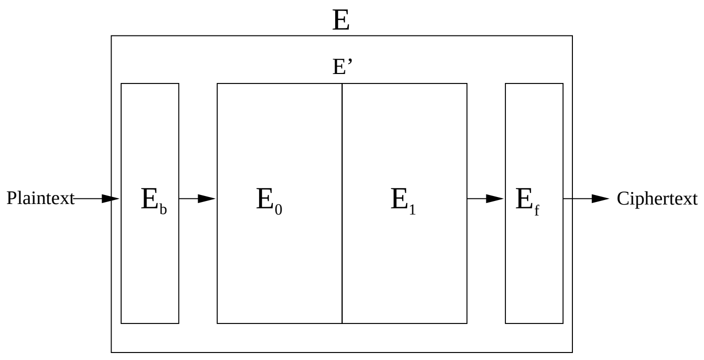

Figure 1: Outline of E

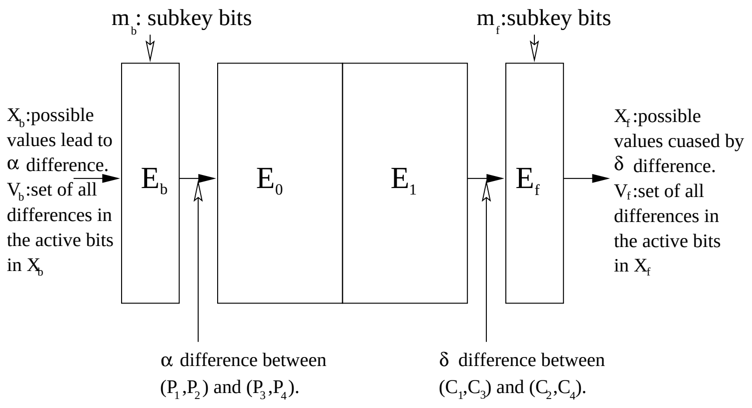
m_b: subkey bits
m_f:subkey bits
X_b:possible values lead to α difference. V_b:set of all differences in the active bits in X_b
E_b
E_0
E_1
E_f
X_f:possible values cuased by δ difference. V_f:set of all differences in the active bits in X_f
α difference between (P_1,P_2) and (P_3,P_4).
δ difference between (C_1,C_3) and (C_2,C_4).

Figure 2: The Notations Used in This Paper

The rectangle distinguisher parameters are α, δ, $\hat{p}$, and $\hat{q}$. Given these parameters, the rectangle distinguisher of the cipher $E' = E_1 \circ E_0$ can easily be constructed.

Before we continue we introduce some additional notations: Let X_b be the set of all plaintext differences that may cause a difference α after E_b. Let V_b be the space spanned by the values in X_b and denote its size 2^{r_b} (i.e., $r_b = \log_2 |V_b|$). Note that usually $n - r_b$ bits are set to 0 for all the values in V_b. Let $t_b = \log_2 |X_b|$ (t_b is not necessarily an integer). Let m_b be the number of subkey bits which enter E_b and affect the difference of the plaintexts by decrypting pairs whose difference after E_b is α, or formally

$$m_b = \left| \left\{ K' \middle| w(K') = 1 \text{ and } \exists K, x : \begin{array}{l} E_{b_K}^{-1}(x) \oplus E_{b_K}^{-1}(x \oplus \alpha) \neq \\ E_{b_{K \oplus K'}}^{-1}(x) \oplus E_{b_{K \oplus K'}}^{-1}(x \oplus \alpha) \end{array} \right\} \right|$$

where $w(x)$ denotes the hamming weight of x.

Similarly, let X_f is the set of all ciphertext differences that a difference δ before E_f may cause. Let V_f denote the space spanned by the values of X_f and denote $r_f = \log_2 |V_f|$. Let $t_f = \log_2 |X_f|$. Let m_f be the number of subkey bits which enter E_f and affect the difference when encrypting a pair with difference δ or formally

$$m_f = \left| \left\{ K' \middle| w(K') = 1 \text{ and } \exists K, x : \begin{array}{l} E_{f_K}(x) \oplus E_{f_K}(x \oplus \alpha) \neq \\ E_{f_{K \oplus K'}}(x) \oplus E_{f_{K \oplus K'}}(x \oplus \alpha) \end{array} \right\} \right|.$$

We outline all these notations in Figure 2.

Our new algorithm for using rectangle distinguisher in a key recovery attack is as follows:

1. Create $Y = \lceil 2^{n/2+2-r_b}/\hat{p}\hat{q} \rceil$ structures of 2^{r_b} plaintexts each. In each structure choose P_0 randomly and let $L = P_0 \oplus V_b$ be the set of plaintexts in the structure.

2. Initialize an array of $2^{m_b+m_f}$ counters. Each counter corresponds to a different guess of the m_b subkey bits of E_b and the m_f subkey bits of E_f.

3. Insert the $N = Y \cdot 2^{r_b}$ ciphertexts into a hash table according to the $n - r_f$ ciphertext bits that are set to 0 in V_f. If a pair agrees on these $n - r_f$ bits, check whether the ciphertext difference is in X_f.

4. For each collision (C_1, C_2) which remains, denote C_i's structure by S_{C_i} and attach to C_1 the index of S_{C_2} and vice versa.

5. In each structure S we search for two ciphertexts C_1 and C_2 which are attached to some other S'. When we find such a pair we check that the $P_1 \oplus P_2$ (the corresponding plaintexts) is in X_b, and check the same for the plaintexts which P_1 and P_2 are related to.

6. For all the quartets which passed the last test denote by (P_1, P_2, P_3, P_4) the plaintexts of a quartet and by (C_1, C_2, C_3, C_4) the corresponding ciphertexts. Increment the counters which correspond to all subkeys K_b, K_f (actually their bits which affect the α and δ differences, respectively) for which $E_{b_{K_b}}(P_1) \oplus E_{b_{K_b}}(P_2) = E_{b_{K_b}}(P_3) \oplus E_{b_{K_b}}(P_4) = \alpha$ and $E^{-1}_{f_{K_f}}(C_1) \oplus E^{-1}_{f_{K_f}}(C_3) = E^{-1}_{f_{K_f}}(C_2) \oplus E^{-1}_{f_{K_f}}(C_4) = \delta$.

7. Output the subkey with maximal number of hits.

The data complexity of the attack is $N = 2^{r_b} Y = 2^{r_b} \lceil 2^{n/2+2-r_b}/\hat{p}\hat{q} \rceil$ chosen plaintexts. The time complexity of Step 1 (the data collection step) is N encryptions. The time complexity of Step 2 is $2^{m_b+m_f}$ memory accesses in a trivial implementation and only one memory access using a more suitable data structures (like B-trees).

Step 3 requires N memory accesses for the insertion of the ciphertexts into a hash table (indexed by the $n - r_f$ bits which are set to 0 in V_f). The number of colliding pairs is about $N^2 \cdot 2^{r_f - n}/2$ as there are N plaintexts divided into 2^{n-r_f} bins (each bin correspond to a value of the $n - r_f$ bits). Note that we not necessarily use all the bins due to large memory requirements (i.e., we can hash only according the first 30 bits set to 0 in V_f). For each collision we check whether the difference of the ciphertexts of the colliding pair belongs to X_f. We keep all the 2^{t_f} values of X_f in a hash table, and thus, the check requires one memory access for each colliding pair. Out of the 2^{r_f} possible differences for a colliding pair, only 2^{t_f} differences are in X_f (i.e., can occur in a right quartet), and thus, about $N^2 \cdot 2^{t_f - n - 1}$ pairs remain. The time complexity of this step is $N + N^2 \cdot 2^{r_f - n - 1}$ memory accesses on average.

Step 4 requires one memory access for each pair which passes the filtering of Step 3. In a real implementation it is wiser to implement Step 4 as part of Step 3, but we separate these steps for the sake of simpler analysis. As there are $N^2 \cdot 2^{t_f - n - 1}$ such pairs, the time complexity of this step is on average $N^2 \cdot 2^{t_f - n - 1}$ memory accesses.

Step 5 implements a search for possible quartets. In a right quartet both (P_1, P_2) and (P_3, P_4) must satisfy that $E_b(P_1) \oplus E_b(P_2) = E_b(P_3) \oplus E_b(P_4) = \alpha$, and thus any right quartet must be combined from some $P_1, P_2 \in S'$ and $P_3, P_4 \in S''$ where S' and S'' are two (not necessarily distinct) structures. Moreover, a right quartet satisfies that $E_f^{-1}(C_1) \oplus E_f^{-1}(C_3) = E_f^{-1}(C_2) \oplus E_f^{-1}(C_4) = \delta$, and thus C_1 is attached to S_{C_3} and C_2 is attached to S_{C_4} and as P_3, P_4 are from the same structure then – $S_{C_3} = S_{C_4}$. Therefore, in each structure S we search for colliding attachments, i.e., pairs of ciphertexts in S which are attached to the same (other) structure $\tilde{S}$. There are about $N^2 \cdot 2^{t_f - n - 1}$ attachments (colliding pairs). The $N^2 \cdot 2^{t_f - n - 1}/Y = N \cdot 2^{t_f + r_b - n - 1}$ attachments in each structure vary over all the Y structures. We conclude that approximately $(N \cdot 2^{t_f + r_b - n - 1})^2/Y$ possible quartets are suggested in each structure (where a quartet corresponds to a pair of plaintexts from some structure attached to the same structure). As there are Y structures the total number of suggested quartets is $N^2 \cdot 2^{2t_f+2r_b-2n-2}$ and in each structure we need to examine approximately $N^2 \cdot 2^{2t_f+2r_b-2n-2}/Y$ quartets. We implement the test in the same manner as in Step 3, i.e., keeping a hash table H_S for each structure S and inserting each ciphertext C to H_{S_C} according to the index of the structure attached to C. Denoting the plaintexts of the suggested quartet by (P_1, P_2, P_3, P_4) and their corresponding ciphertexts by (C_1, C_2, C_3, C_4), we first check that $P_1 \oplus P_2 \in X_b$. This test requires one memory access for each possible quartet. If we assume that the 2^{t_b} values in X_b vary uniformly in V_b [4] then the probability that the value $P_1 \oplus P_2$ is in X_b is $2^{t_b-r_b}$. A quartet which fails this test can be discarded immediately. Therefore, out of the $N^2 \cdot 2^{2t_f+2r_b-2n-2}$ possible quartets only $N^2 \cdot 2^{2t_f+r_b+t_b-2n-2}$ quartets remain. As stated before, this filtering requires one memory accesses for each candidate quartet, thus the algorithm requires $N^2 \cdot 2^{2t_f+2r_b-2n-2}$ in total. We can discard more quartets by testing whether $P_3 \oplus P_4 \in X_b$. We conclude that this step requires $N^2 \cdot 2^{2t_f+2r_b-2n-2} \cdot (1 + 2^{t_b-r_b})$ memory accesses. The expected running time of Step 5 for all the data is $N^2 \cdot 2^{2t_f+2r_b-2n-2} \cdot (1 + 2^{t_b-r_b})$ memory accesses and about $N^2 \cdot 2^{2t_f+2t_b-2n-2}$ quartets remain after this step.

In Step 6 we try to deduce the right subkey from the remaining quartets. As stated before, a right quartet satisfies $E_b(P_1) \oplus E_b(P_2) = \alpha = E_b(P_3) \oplus E_b(P_4)$. Both pairs are encrypted by the same subkey, hence, a right quartet must agree on K_b (the m_b subkey bits which enter E_b and affect the output difference α). There are 2^{t_b} possible input differences that lead to α difference after E_b, therefore, $2^{m_b-t_b}$ subkeys on average take one of these values into a difference α. As each pair suggests $2^{m_b-t_b}$ subkeys, they agree on average on $(2^{m_b-t_b})^2/2(2^{m_b}) = 2^{m_b-2t_b-1}$ subkeys for E_b. We can find these options by keeping in a precomputed table either the possible values for any pair on its own, or for the whole quartet. Repeating the analysis for E_f, (C_1, C_3), and (C_2, C_4) we get about $2^{m_f-2t_f-1}$ subkeys suggestions from each quartet. Thus, each of the $N^2 \cdot 2^{2t_f+2t_b-2n-2}$ remaining quartets suggests $2^{m_b+m_f-2t_f-2t_b-2}$ possible subkeys. There are $2^{m_b+m_f}$ possible subkeys and $N^2 \cdot 2^{2t_f+2t_b-2n-2} \cdot 2^{m_b+m_f-2t_f-2_b-2} = N^2 \cdot 2^{m_b+m_f-2n-4}$ hits. The expected number of hits for a (wrong) subkey is about $N^2 \cdot 2^{-2n-4}$. Since $N \leq 2^n$ is the number of plaintexts the expected number of hits per wrong subkey is less than $2^{-4} = 1/16$, and we can conclude that the attack almost always succeeds in recovering subkey bits (since the number of expected hits for the right subkey is 4), or at least reduces the number of candidates for the right subkey. We can insert the $2^{m_b-t_b}$ subkeys suggested by (P_1, P_2) into a hash table, and for each subkey suggested by the pair (P_3, P_4) we can check whether it was already suggested. Doing the same for E_f we get that for each quartet we need $3 \cdot 2^{m_b-t_b} + 3 \cdot 2^{m_f-t_f}$ memory accesses. We can optimize this a little bit by storing in advance a table and a list for each difference in X_b and save the time of building the hash table. This method saves a 1/3 of the number of memory accesses. Using this method this step requires attack about $N^2 \cdot 2^{2t_f+2t_b-2n-3} \cdot (2^{m_b-2t_b} - 2^{m_f-2t_f}) = N^2 \cdot 2^{-2n-3} \cdot (2^{m_b+2t_f} + 2^{m_f+2t_b})$ memory accesses for the entire attack.

Step 7 requires $2^{m_b+m_f}$ memory accesses using a trivial implementation, which can be reduced to 1–4 memory accesses using a more efficient data structure (e.g., B-trees or dynamic hash tables).

Overall, this algorithm requires $N = 2^{r_b}Y = 2^{r_b}\lceil 2^{n/2+2-r_b}/\hat{p}\hat{q} \rceil$ chosen plaintexts, and time complexity of $N + N^2 \cdot 2^{r_f-n-1} + N^2 \cdot 2^{t_f-n} + N^2 \cdot 2^{2t_f+2r_b-2n-2} + N^2 \cdot 2^{t_b+t_f-2n-1} \cdot (2^{m_b+t_f} + 2^{m_f+t_b}) = N + N^2(2^{r_f-n-1} + 2^{t_f-n} + 2^{2t_f+2r_b-2n-2} + 2^{m_b+t_b+2t_f-2n-1} + 2^{m_f+2t_b+t_f-2n-1})$ memory accesses. The memory complexity is $N + 2^{t_b} + 2^{t_f} + 2^{m_b+m_f}$.

Table 1 summarizes the time complexity of each step and the number of plaintexts / pairs / quartets that remain after each step of the algorithm.

Using this algorithm we can break 3.5-round SC2000, using the following decomposition: E_b consists of the first S4 layer, the following 1.25 rounds are E_0, the next 1.25 rounds are E_1, and the final S4 layer is E_f. For this decomposition the following properties were presented in [9]: $r_b = r_f = m_b = m_f = 40, t_b = 27, t_f = 27.9, n = 128, \hat{p} = 2^{-8.96}, \hat{q} = 2^{-9.16}$. Thus, we conclude that the data complexity is $N = 2^{84.6}$ chosen plaintexts and that the time complexity is $2^{84.6}$ memory accesses, which slightly improves the results in [9].

We can also break 10-round Serpent using the following decomposition: E_b consists of round 0. The following 4 rounds are E_0, the next 4 rounds are E_1, and round 9 is E_f. For this decomposition

[4] If this is not the case we can present a better algorithm which exploits the non-uniformity.

Step No.	Short Description	Time Complexity	# of Remaining Plaintexts/ Pairs/Quartets
1	Data generation	N encryptions	N plaintexts
2	Subkey counters' init.	1 MA	No change
3	First filtering	$N + N^2 \cdot 2^{r_f - n - 1}$ MA	$N^2 \cdot 2^{t_f - n - 1}$ pairs
4	Suggesting quartets	$N^2 \cdot 2^{t_f - n - 1}$ MA	No change
5	Eliminating quartets	$N^2 \cdot 2^{2t_f + 2r_b - 2n - 2}$ MA	$N^2 \cdot 2^{2t_f + 2t_b - 2n - 2}$ quartets
6	Subkey detection	$N^2 \cdot 2^{t_b + t_f - 2n - 1}(2^{m_b + t_f} + 2^{m_f + t_b})$ MA	No change
7	Printing subkey	1–4 MA	No Change

MA - Memory Accesses

Table 1: The Rectangle Attack Steps and their Effect

the following properties are presented in [7]:[5] $r_b = m_b = 76, r_f = m_f = 20, t_b = 48.85, t_f = 13.6, n = 128, \hat{p} = 2^{-25.4}, \hat{q} = 2^{-34.9}$. Thus, we conclude that the data complexity is $N = 2^{126.3}$ chosen plaintexts and that the time complexity is $2^{173.8}$ memory accesses. Note that the time complexity of the attack presented in [7] is 2^{217} memory accesses using 2^{196} memory cells (or $2^{219.4}$ memory accesses with $2^{131.8}$ memory cells).

Note that we can use the above algorithm in several other scenarios. One of them is a chosen ciphertext scenario, where the above algorithm is applied on $E^{-1} = E_b^{-1} \circ E_0^{-1} \circ E_1^{-1} \circ E_f^{-1}$. The analysis of this case is the same as before as long as we replace in the above equations all the sub-scripts b by f and vice versa.

Sometimes, we might want to attack a cipher E with additional rounds only at one side, i.e., to divide E to $E = E_1 \circ E_0 \circ E_b$ (or $E = E_f \circ E_1 \circ E_b$). In this case our analysis still holds with $m_f = r_f = t_f = 0$.

4 A Key Recovery Attack Based on Boomerang Distinguisher

In this section we apply our ideas from the previous section to the boomerang attack. We generalized the results of [21, 9]. Like the rectangle attack, we have found that whenever the boomerang distinguisher succeeds then the key recovery attack also succeeds.

There are various standard techniques to use distinguishers for a key recovery attack [3, 17, 12, 4, 5, 16]. The basic idea is to try all subkeys which affect the difference (or the approximation) before and after the distinguishers (i.e., in E_b and E_f), and to deduce that the correct subkey is the one for which the statistical distinguisher has the best results. However, this basic idea can be very expensive in terms of time and memory complexities. Moreover, due to the adaptive chosen plaintext and ciphertext requirement, using a boomerang distinguisher in a key recovery attack can be done if either E_b or E_f is present but not when both exist.

As both boomerang and rectangle distinguishers exploit the same α and δ, we use the same notations of $E = E_f \circ E_1 \circ E_0 \circ E_b$, m_b , r_b , t_b, m_f, r_f, and t_f as in the earlier sections.

The generic boomerang attack on $E = E_1 \circ E_0 \circ E_b$ is as follows:

1. Initialize an array of 2^{m_b} counters. Each counter corresponds to a different guess of the m_b subkey bits of E_b.

2. Generate a structure F of plaintexts, choose P_0 randomly and let $F = P_0 \oplus V_b$ be the set of plaintexts in the structure.

3. Ask for the encryption of F and denote the set of ciphertexts by G.

4. For each ciphertext $c \in G$ compute $c' = c \oplus \delta$, and define the set $H = \{c \oplus \delta | c \in G\}$.

[5]As stated earlier, in [7] it was mistakenly claimed that $t_b = 64$. We use the correct value of 76, and derive the correct time complexity.

5. Ask for the decryption of H, and denote the plaintexts set by I.

6. Insert all the plaintexts in I into a hash table according to the $n - r_b$ plaintext bits which are set to 0 in V_b.

7. In case of a collision of plaintexts in the hash table:

   (a) Denote the plaintexts which collide in the hash table by (P_3, P_4), and test whether $P_3 \oplus P_4 \in X_b$. If this condition is satisfied denote the plaintexts from F which correspond to (P_3, P_4) by (P_1, P_2). Test whether $P_1 \oplus P_2 \in X_b$. If any of the tests fails, discard this quartet.

   (b) For a quartet (P_1, P_2, P_3, P_4) which passes the above filtering we check all possible K_b which enter E_b (actually its bits which affect the α) and increment the counters which correspond to the subkeys for which $E_{b_{K_b}}(P_1) \oplus E_{b_{K_b}}(P_2) = E_{b_{K_b}}(P_3) \oplus E_{b_{K_b}}(P_4) = \alpha$.

8. Repeat Steps 2–7 until a subkey is suggested 4 times.

Steps 2–5 perform a δ-shift on structures (and not on the pairs directly).

From the analysis in [21] and the properties of the algorithm it is evident that the data complexity of the attack is about $8(\hat{p}\hat{q})^{-2}$. However, we generate at least 2^{r_b+1} plaintexts and ciphertexts, and thus the data complexity of the attack is $N = \max\{2^{r_b+1}, 8(\hat{p}\hat{q})^{-2}\}$.

The time complexity of Step 1 is equivalent to 2^{m_b} memory accesses. However, we can keep the counters in more efficient data structures (like B-trees, or dynamic hash tables) for which Step 1 takes only one memory access.

In steps 2–5 we encrypt $N/2$ plaintexts, compute $N/2$ XOR operations and decrypt $N/2$ ciphertexts. Thus, the total time complexity of Steps 2–5 for the whole attack is N encryptions/decryptions.

We will now restrict our attention to the time complexity analysis for each F independently of other F's, as each execution of Steps 6–7 for a given structure is independent of the execution of these steps for other structures.

Inserting the 2^{r_b} plaintexts from I into the hash table according to $n - r_b$ bits[6] requires 2^{r_b} memory accesses and results in about $(2^{r_b})^2/(2 \cdot 2^{n-r_b}) = 2^{3r_b-n-1}$ collisions. Thus, the time complexity of Step 6 is 2^{r_b} memory accesses per structure, and therefore $N/2$ memory accesses for the entire attack.

The probability that a pair that differs in the r_b affected bits has a difference in X_b is $2^{t_b-r_b}$. In a right quartet both pairs have difference α after E_b. Each candidate quartet has to pass this filter twice. As stated before the probability that a pair passes this filter is $2^{t_b-r_b}$, therefore, a candidate quartet passes the filtering of Step 7(a) with probability $2^{2t_b-2r_b}$, and the expected number of memory accesses needed for this filtering is approximately $1 + 2^{t_b-r_b}$ memory accesses per candidate quartet. Since each collision in the hash table of Step 6 suggests a quartet, we have in this step about 2^{3r_b-n-1} memory accesses for each structure F and the expected number of remaining quartets is $2^{3r_b-n-1} \cdot (2^{t_b-r_b})^2 = 2^{2t_b+r_b-n-1}$ quartets. The most time consuming part of this step is the first filtering. for each structure I (and therefore, for each F) we need about 2^{3r_b-n-1} memory accesses, and in total for the entire $N/2^{r_b+1}$ structures this step requires $2^{3r_b-n-1} \cdot N/2^{r_b+1} = N \cdot 2^{2r_b-n-1}$ memory accesses.

Both pairs ((P_1, P_2) and (P_3, P_4)) are encrypted by the same subkey, hence, a right quartet must agree on K_b (the m_b subkey bits which enter E_b and affect the output difference α). There are 2^{t_b} possible input differences that lead to α difference after E_b, therefore, $2^{m_b-t_b}$ subkeys on average take one of these values into a difference α. As each pair suggests $2^{m_b-t_b}$ subkeys, they agree on $(2^{m_b-t_b})^2/2(2^{m_b}) = 2^{m_b-2t_b-1}$ subkeys for E_b on average. Like in the previous section, our analysis shows that the expected number of subkeys suggested by such a quartet is $2^{m_b-2t_b-1}$ and that it takes $2^{m_b-t_b+1}$ memory accesses for each quartet. Hence, the expected number of memory accesses in Step 7(b) is $2^{t_b+m_b+r_b-n}$ for each structure. We conclude that the total time complexity of Step 7(b) is expected to be $N \cdot 2^{t_b+m_b-n-1}$ memory accesses for the whole attack.

[6]Again, we can use less bits as keys for the hash table if the number of bins is much greater than the number of plaintexts which are inserted into the hash table.

Step No.	Short Description	Time Complexity	# of Remaining Plaintexts/ Pairs/ Quartets
1	Subkey counters' init.	1 MA	—
2+3	Data generation	$N/2$ encryptions	$N/2$ plaintexts
4	Data generation	$N/2$ MA	No change
5	Data generation	$N/2$ decryptions	N plaintexts
6	Finding possible quartets	$N/2$ MA	$N \cdot 2^{2r_b-n-2}$ quartets
7(a)	Eliminating quartets	$N \cdot 2^{2r_b-n-2}$ MA	$N \cdot 2^{2t_b-n-2}$ quartets
7(b)	Subkey detection	$N \cdot 2^{r_b+m_b-n-1}$MA	No change
8	Printing subkey	$N/2^{r_b+1}$ MA	No change

MA - Memory Accesses

Table 2: The Basic Boomerang Attack Steps and their Effect

Each structure F induces about $2^{2t_b+r_b-n-1} \cdot 2^{m_b-2t_b-1} = 2^{r_b+m_b-n-2}$ subkey hits. As there are $N/2^{r_b+1}$ structures, the total number of subkey hits is expected to be $N \cdot 2^{m_b-n-3}$, which are distributed over 2^{m_b} subkeys. Thus, the expected number of hits for each subkey is $N \cdot 2^{-n-3}$. As $N \leq 2^n$, the expected number of hits for a wrong subkey is less then 1/8 while the right subkey is expected to get 4 hits. This is sufficient for either recovering the right subkey, or to reduce the subkey candidates space by a very large factor.

In Step 8 we check whether one of the counters has the value of 4 (or more). This has to be done whenever we finish Step 7(b) for some F. In a trivial implementation this step would require 2^{m_b} memory accesses for each F. We can also implement this step as part of Step 7(b). Whenever a counter is increased we check that it has not exceeded 4. This method results in enlarging the time of Step 7(b). Using more appropriate data structures, we can perform the check once whenever we replace the F structure we work with, and thus, this step would cost us 1 memory accesses for each F, and for the entire attack $N/2^{r_b+1}$ memory accesses.

We conclude that the attack requires $N = \max\{2^{r_b+1}, 8(\hat{p}\hat{q})^{-2}\}$ adaptive chosen plaintexts and ciphertexts and time complexity of about $N/2 + N/2 + N \cdot 2^{2r_b-n-2} + N \cdot 2^{t_b+m_b-n-1} + N/2^{r_b+1} = N(1 + 2^{2r_b-n-2} + 2^{t_b+m_b-n-1})$ memory accesses.

Table 2 summarizes the time complexity of each step and the number of plaintexts / pairs / quartets that remain after each step of the algorithm.

Using this algorithm, we can break 3-round SC2000, using the following decomposition: E_b consists of the first S4 layer, the following 1.25 rounds are E_0 and the next 1.25 rounds are E_1. For this decomposition the following properties were presented in [9]: $r_b = m_b = 40, t_b = 27, n = 128, \hat{p} = 2^{-8.96}, \hat{q} = 2^{-9.16}$. Thus, we conclude that the data complexity of the attack is $N = 2^{41}$ adaptive chosen plaintexts and ciphertexts. The time complexity of the attack is about 2^{41} memory accesses.

We attack 9-round Serpent, using this algorithm and the following decomposition: E_b consists of round 0, the following 4 rounds are E_0, and the next 4 rounds are E_1. For this decomposition the following properties were presented in [7]: $r_b = m_b = 76, t_b = 48.85, n = 128, \hat{p} = 2^{-25.4}, \hat{q} = 2^{-34.9}$. The data complexity of the attack is $N = 2^{123.6}$ adaptive chosen plaintexts and ciphertexts, with time complexity of $2^{147.2}$ memory accesses. We can also attack rounds 1–9 of Serpent using the decomposition used in the previous section. In this attack we are attacking rounds 9–1 (i.e., the attack is on $E_0^{-1} \circ E_1^{-1} \circ E_f^{-1}$). For this decomposition we get: $r_f = m_f = 20, t_f = 13.6, n = 128, \hat{p} = 2^{-25.4}, \hat{q} = 2^{-34.9}$. Obviously the data complexity does not change, as we use the same underlying distinguisher. However, the time complexity of this attack drops to $2^{123.6}$ memory accesses (instead of $2^{147.2}$).

5 Enhancing the Boomerang Attack

In this section we present a new method to use the boomerang attack with both E_b and E_f. Recall that the main step of the boomerang attack is the δ-shift (encryption of each plaintext, XORing of the corresponding ciphertext with δ, and decryption of the outcome). Our method uses a generalization of the δ-shift.

The new algorithm to attack $E_f \circ E_1 \circ E_0 \circ E_b$ is as follows:

1. Initialize an array of $2^{m_b+m_f}$ counters. Each counter corresponds to a different guess of the m_b subkey bits of E_b and the m_f bits of E_f.
2. Generate a structure F of plaintexts, choose P_0 randomly and let $F = P_0 \oplus V_b$ be the set of plaintexts in the structure.
3. Ask for the encryption of F and denote the set of ciphertexts by G.
4. For each $c \in G$ and $\epsilon \in X_f$ compute $c' = c \oplus \epsilon$, and define the set $H = \{c \oplus \epsilon | \; c \in G \text{ and } \epsilon \in X_f\}$.
5. Ask for the decryption of H, and denote the plaintexts set by I.
6. Insert all the plaintexts in I into a hash table according to the $n - r_b$ plaintext bits which are set to 0 in V_b.
7. In case of a collision of plaintexts in the hash table:
   (a) Denote the plaintexts which collide in the hash table by (P_3, P_4), and test whether $P_3 \oplus P_4 \in X_b$. If this condition is satisfied denote the plaintexts from F which correspond to (P_3, P_4) by (P_1, P_2). Test whether $P_1 \oplus P_2 \in X_b$. If any of the tests fails, discard this quartet.
   (b) For a quartet (P_1, P_2, P_3, P_4) which passes the above filtering we obtain from a precomputed table the possible values for the m_b subkey bits which enter E_b and affect the α difference. We also obtain from a precomputed table the possible values for the m_f subkey bits which enter E_f and affect the δ difference. The specific implementation aspects of this step are described later on. We increment counters which correspond to subkeys K_b, K_f for which $E_{b_{K_b}}(P_1) \oplus E_{b_{K_b}}(P_2) = E_{b_{K_b}}(P_3) \oplus E_{b_{K_b}}(P_4) = \alpha$ and $E^{-1}_{f_{K_f}}(C_1) \oplus E^{-1}_{f_{K_f}}(C_3) = E^{-1}_{f_{K_f}}(C_2) \oplus E^{-1}_{f_{K_f}}(C_4) = \delta$.
8. Repeat Steps 2–7 until a subkey is suggested 4 times.

We call Steps 2–5 an ϵ-shift as each plaintext is encrypted, then shifted by all possible ϵ's and the values of the shifted ciphertexts are decrypted.

The time complexity of Step 1 is equivalent to $2^{m_b+m_f}$ memory accesses. However, we can keep the counters in a more efficient data structures (like B-trees, or dynamic hash tables), for which Step 1 takes only one memory access.

Each F induces a set of 2^{r_b} ciphertexts in G, and each ciphertext is shifted by 2^{t_f} possible values, hence $|H| = |I| = 2^{r_b+t_f}$. Even though we expand the number of possible quartets (by multiplying the size of I by 2^{t_f}), the number of right quartets does not change. Hence, we still need about $\lceil 8(\hat{p}\hat{q})^{-2}/2^{r_b+1} \rceil$ structures.

The data complexity of the attack is $N = 2^{r_b+t_f} \cdot \lceil 8(\hat{p}\hat{q})^{-2}/2^{r_b+1} \rceil$. However, we might get that $N > 2^n$. We can implement these cases in one of two ways. The first way is to ask for the encryption/decryption N (not necessarily different) oracle queries. The second way to implement this is to store the already encrypted/decrypted values in a table, and test for each encryption/decryption if it is already in the table, in order to save most of the encryptions/decryptions. This way the attack requires 2^n known plaintexts and N memory accesses.

Like in the previous section we perform the time complexity analysis for each F independently of other F's, as each execution of Steps 6–7 for a given structure is independent of the execution of these steps for some other structure.

Inserting $2^{r_b+t_f}$ values into a hash table according to $n-r_b$ bits requires $2^{r_b+t_f}$ memory accesses and results in about $(2^{r_b+t_f})^2/(2\cdot 2^{n-r_b}) = 2^{3r_b+2t_f-n-1}$ collisions (when a collision suggests a quartet). Thus, the time complexity of Step 6 is $2^{r_b+t_f}$ memory accesses per structure, and N memory accesses in total.

The probability that a pair that differs in the r_b affected bits has a difference α after E_b is $2^{t_b-r_b}$. In a right quartet both pairs have difference α after E_b. Each candidate quartet has to pass this filter twice. As stated earlier, the probability that a pair passes this filter is $2^{t_b-r_b}$, therefore, a candidate quartet passes the filtering of Step 7(a) with probability $2^{2t_b-2r_b}$, and the expected number of memory accesses needed for this filtering is approximately $1+2^{t_b-r_b}$ memory accesses per candidate quartet. We cannot discard quartets according to their differences after E_f due to the way we constructed these quartets. Since each collision in the hash table of Step 6 suggests a quartet, we have in this step about $2^{3r_b+2t_f-n-1}$ memory accesses for each structure F, and that the expected number of remaining quartets is $2^{3r_b+2t_f-n-1}\cdot(2^{t_b-r_b})^2 = 2^{2t_b+r_b+2t_f-n-1}$. Since there are $N/2^{r_b+t_f}$ structures we conclude that for the whole attack this step requires about $N\cdot 2^{2r_b+t_f-n-1}$ memory accesses.

We recover subkey material both in E_b and E_f. By repeating the analysis from Section 3, each remaining quartet suggests $2^{m_b+m_f-2t_b-2t_f-2}$ subkeys for E_b and E_f, and thus, we get $2^{r_b+m_b+m_f-n-3}$ hits (on average) from each structure and $N\cdot 2^{m_b+m_f-n-t_f-3}$ subkey hits in total. Like in Section 3, each quartet requires about $2^{m_b-t_b+1}+2^{m_f-t_f+1}$ memory accesses. We conclude that Step 7(b) requires $N\cdot 2^{t_b+t_f-n-1}\cdot(2^{m_b+t_f}+2^{m_f+t_b})$ memory accesses for the whole attack.

Note that we require that the total number of hits per subkey is less then 1. Otherwise, the signal to noise ratio becomes too small for the attack to be effective. As there are about $N\cdot 2^{m_b+m_f-t_f-n-3}$ subkey hits in total, the expected number of hits per subkey is about $N\cdot 2^{-t_f-n-3}$. Note that N might be bigger than 2^n but on the same time, $N\leq 2^{n+t_f}$ (because we take at most 2^n ciphertexts and shift them by 2^{t_f} values). As for the boomerang attack, we find that the number of hits per wrong subkey is $\leq 1/8$.

In Step 8 we check whether one of the counters has the value of 4 (or more). This has to be done whenever we finish Step 7(b) for some F. In a trivial implementation this step would require $2^{m_b+m_f}$ memory accesses for each F. We can also implement this step as part of Step 7(b). Whenever a counter is increased we check that it has not exceeded 4. This method results in enlarging the time of Step 7(b). Using more appropriate data structures, we can perform the check once whenever we replace the F structure we work with, and thus, this step would cost us 1 memory accesses for each F, and for the entire attack $N/2^{r_b+t_f}$ memory accesses.

The data complexity of the attack is $N = 2^{r_b+t_f}\cdot\lceil 8(\hat{p}\hat{q})^2/2^{r_b+1}\rceil$ adaptive chosen plaintexts and ciphertexts (and as stated earlier if $N>2^n$ we can replace it by 2^n known plaintexts using a table of size 2^n). The expected time complexity of the attack is $N+N+N\cdot 2^{2r_b+t_f-n-1}+N\cdot 2^{t_b+t_f-n-1}\cdot(2^{m_b+t_f}+2^{m_f+t_b})+N/2^{r_b+t_f} = N(2+2^{2r_b+t_f-n-1}+2^{m_b+t_b+2t_f-n-1}+2^{m_f+2t_b+t_f-n-1})$ memory accesses. The memory complexity of the attack is $2^{m_b+m_f}+2^{r_b+t_f}$.

Table 3 summarizes the time complexity of each step and the number of plaintexts / pairs / quartets that remain after each step of the algorithm.

We use the same decomposition of 3.5-round SC2000 as in Section 3, which is as follows: E_b consists of the first S4 layer, the following 1.25 rounds are E_0, the next 1.25 rounds are E_1, and the final S4 layer is E_f. For this decomposition the following properties were presented in [9]: $r_b = r_f = m_b = m_f = 40, t_b = 27, t_f = 27.9, n = 128, \hat{p} = 2^{-8.96}, \hat{q} = 2^{-9.16}$. Hence, the data complexity of the attack is $2^{67.9}$ adaptive chosen plaintexts and ciphertexts (this complexity can be reduced to 2^{67} by attacking E^{-1}). The attack requires $2^{67.9}$ memory cells (when we attack E^{-1} it requires only 2^{67} memory cells). The time complexity of the attack is $2^{68.9}$ memory accesses (the attack on E^{-1} requires 2^{68} memory accesses).

Decomposing 10-round Serpent like in Section 3: E_b consists of round 0, the following 4 rounds are E_0, the next 4 rounds are E_1, and round 9 is E_f. For this decomposition the following properties are presented in [7]: $r_b = m_b = 76, r_f = m_f = 20, t_b = 48.85, t_f = 13.6, n = 128, \hat{p} = 2^{-25.4}, \hat{q} = 2^{-34.9}$. The data complexity of the attack is $N = 2^{123.6+13.6} = 2^{137.2}$. As mentioned before, we can either treat this as $2^{137.2}$ queries to the encryption/decryption oracle (of course not distinct queries) or we can just ask the encryption/decryption of any plaintext/ciphertext we need, and store it in a table.

Step No.	Short Description	Time Complexity	# of Remaining Plaintexts/ Pairs/ Quartets
1	Subkey counters' init.	1 MA	—
2+3	Data generation	$N/2^{t_f}$ encryptions	$N/2^{t_f}$ plaintexts
4	Data generation	N MA	No change
5	Data generation	N decryptions	N Plaintexts
6	Eliminating wrong pairs	N MA	$N \cdot 2^{2r_b+2t_f-n-2}$ quartets
7(a)	Eliminating quartets	$N \cdot 2^{2r_b+t_f-n-1}$ MA	$N \cdot 2^{2t_b+2t_f-n-2}$ quartets
7(b)	Subkey detection	$N \cdot 2^{t_b+t_f-n-2}(2^{m_b+t_f}+2^{m_f+t_b})$ MA	No change
8	Printing subkey	$N/2^{r_b+t_f}$ MA	No change

MA - Memory Accesses

Table 3: The Steps of the Improved Boomerang Attack and Their Effect

The attack requires $2^{173.8}$ memory accesses.

6 Summary

This paper presents several contributions. The first contribution is an improved generic rectangle attack. In order to apply a rectangle key recovery attack, it is sufficient to provide several parameters of the attacked cipher (for example α, δ, etc.). The improved attack algorithm can attack 10-round Serpent with data complexity of $2^{126.3}$ chosen plaintexts and time complexity of $2^{173.8}$ memory accesses. This new result enables attacking 10-round Serpent with 192-bit subkeys. We also have shown that the algorithm is very successful and almost always reduces the number of candidate subkeys.

The second contribution is a generic boomerang key recovery attack. The attack uses similar techniques as in the rectangle key recovery attack and the result is an efficient algorithm for retrieving subkey material. In the analysis of this attack we found out that this attack almost always succeeds.

The third contribution is extending the generic boomerang key recovery attack to attack more rounds. This contribution allows for the boomerang attack to attack as many rounds as the rectangle attack despite its adaptive chosen plaintext and ciphertext nature. This allows to attack 10-round Serpent with the enhanced boomerang attack using $2^{137.2}$ adaptive chosen plaintexts and ciphertexts (or 2^{128} known plaintexts), and $2^{173.8}$ memory accesses.
In Table 4 we compare the requirements of the generic attacks, and in Table 5 we present our new results on Serpent and SC2000. For comparison, we also include the previous boomerang and rectangle results and the best known attacks against these ciphers.

References

[1] Ross Anderson, Eli Biham, Lars R. Knudsen, *Serpent: A Proposal for the Advanced Encryption Standard*, NIST AES Proposal, 1998.

[2] Eli Biham, *Higher Order Differential Cryptanalysis*, unpublished paper, 1994.

[3] Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[4] Eli Biham, Alex Biryukov, Adi Shamir, *Cryptanalysis of Skipjack reduced to 31 rounds*, Advances in Cryptology, proceedings of EUROCRYPT '99, Lecture Notes in Computer Science 1592, pp. 12–23, Springer-Verlag, 1999.

Attack	Rectangle (Section 3)	Boomerang (Section 4)	Enhanced Boomerang (Section 5)
Cipher's parts being attacked	$E_f \circ E_1 \circ E_0 \circ E_b$	$E_1 \circ E_0 \circ E_b$	$E_f \circ E_1 \circ E_0 \circ E_b$
Type of Attack	Chosen Plaintext	Adaptive Chosen Plaintext and Ciphertext	Adaptive Chosen Plaintext and Ciphertext
Data Complexity (N)	$\max\{2^{r_b}, 2^{n/2+2}/\hat{p}\hat{q}\}$	$\max\{8(\hat{p}\hat{q})^{-2}, 2^{r_b}\}$	$2^{t_f} \max\{2^{r_b}, 8(\hat{p}\hat{q})^{-2}\}$
Memory Accesses	$N^2(2^{r_f-n-1} + 2^{t_f-n} + 2^{2t_f+2r_b-2n-2} + 2^{m_b+2t_f+t_b-2n-1} + 2^{m_f+2t_b+t_f-2n-1}) + N$	$N(1 + 2^{2r_b-n-2} + 2^{t_b+m_b-n-1})$	$N(2 + 2^{2r_b+t_f-n-1} + 2^{m_b+t_b+2t_f-n-1} + 2^{m_f+t_f+2t_b-n-1})$
Memory Cells	$2^{m_b+m_f} + N$	$2^{r_b} + 2^{m_b}$	$2^{r_b+t_f} + 2^{m_b+m_f}$
Subkey Bits	$m_b + m_f$	m_b	$m_b + m_f$
Hits per Wrong Subkey	$\leq 1/16$	$\leq 1/8$	$\leq 1/8$

Table 4: Comparison of the Boomerang and the Rectangle Generic Attacks

Cipher	Attack	Number of Rounds	Complexity		
			Data	Time	Memory
SC2000	Rectangle – this paper	3.5	$2^{84.6}$ CP	$2^{84.6}$ MA	$2^{84.6}$
	Boomerang – this paper	3	2^{41} ACPC	2^{41} MA	2^{40}
	Boomerang – this paper	3.5	2^{67} ACPC	2^{67} MA	2^{67}
best	Linear [23]	4.5	$2^{104.3}$ KP	$2^{83.3}$ MA	2^{80}
Serpent	Amp. Boomerang[11]	9	2^{110} CP	2^{252} MA	2^{208}
	Rectangle[7]	10	$2^{126.8}$ CP	2^{217} MA	2^{192}
	Rectangle[7]	10	$2^{126.8}$ CP	$2^{219.4}$ MA	$2^{126.8}$
	Boomerang – this paper	9	$2^{123.6}$ ACPC	$2^{123.6}$ MA	$2^{21.5}$
	Boomerang – this paper	10	2^{128} KP	$2^{173.8}$ MA	2^{96}
	Rectangle – this paper	10	$2^{126.3}$ CP	$2^{173.8}$ MA	$2^{126.3}$
best	Linear [6]	11	2^{118} KP	2^{214} MA	2^{85}

MA - Memory Accesses
CP - Chosen Plaintexts, KP - Known Plaintexts
ACPC - Adaptive Chosen Plaintexts and Ciphertexts

Table 5: New Boomerang and Rectangle Results on SC2000 and Serpent

[5] Eli Biham, Alex Biryukov, Adi Shamir, *Miss in the Middle Attacks on IDEA and Khufu*, proceedings of Fast Software Encryption 6, Lecture Notes in Computer Science 1636, pp. 124–138, Springer-Verlag, 1999.

[6] Eli Biham, Orr Dunkelman, Nathan Keller, *Linear Cryptanalysis of Reduced Round Serpent*, proceedings of Fast Software Encryption 8, 2001, to appear. Available on-line at *http://vipe.technion.ac.il/~orrd/crypt/.*

[7] Eli Biham, Orr Dunkelman, Nathan Keller, *The Rectangle Attack – Rectangling the Serpent*, Advances in Cryptology, proceedings of EUROCRYPT 2001, Lecture Notes in Computer Science 2045, pp. 340–357, Springer-Verlag, 2001.

[8] CRYPTREC project - Evaluation of Cryptographic Techniques, *http://www.jpg.go.jp/security/enc/CRYPTREC/index-e.html.*

[9] Orr Dunkelman, Nathan Keller, *Boomerang and Rectangle Attacks on SC2000*, preproceedings of the NESSIE second workshop, 2001.
Available on-line at *http://vipe.technion.ac.il/~orrd/crypt/*.

[10] Louis Granboulan, *Flaws in Differential Cryptanalysis of Skipjack*, proceedings of Fast Software Encryption 8, 2001, to appear.

[11] John Kelsey, Tadayoshi Kohno, Bruce Schneier, *Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent*, proceedings of Fast Software Encryption 7, Lecture Notes in Computer Science 1978, pp. 75–93, Springer-Verlag, 1999.

[12] Lars Knudsen, *Truncated and Higher Order Differentials*, proceedings of Fast Software Encryption 2, Lecture Notes in Computer Science 1008, pp. 196–211, Springer-Verlag, 1995.

[13] Lars Knudsen, Håvard Raddum, *A First Report on Whirlpool, NUSH, SC2000, Noekeon, Two-Track-MAC and RC6*, NESSIE internal report NES/DOC/UIB/WP3/007/b, 2001. Available on-line at *http://www.nessie.eu.org/nessie/*.

[14] Lars Knudsen, Håvard Raddum, *A Differential Attack on Reduced-Round SC2000*, preproceedings of the NESSIE second workshop, 2001.

[15] Lars Knudsen, Matt J.B. Robshaw, David Wagner, *Truncated Differentials and Skipjack*, proceedings of Advances in Cryptology – CRYPTO '99, Lecture Notes in Computer Science 1666, pp. 165–180, Springer-Verlag, 1999.

[16] Suzan K. Langford, Martin E. Hellman, *Differential-Linear Cryptanalysis*, Advances in Cryptology, proceedings of CRYPTO '94, Lecture Notes in Computer Science 839, pp. 17–25, Springer-Verlag, 1994.

[17] Mitsuru Matsui, *Linear Cryptanalysis Method for DES Cipher*, Advances in Cryptology, proceedings of EUROCRYPT '93, Lecture Notes in Computer Science 765, pp. 386–397, Springer-Verlag, 1994.

[18] NESSIE - New European Schemes for Signatures, Integrity and Encryption.
http://www.nessie.eu.org/nessie/.

[19] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, Hidema Tanaka, *Specification and Supporting Document of the Block Cipher SC2000* submitted to the NESSIE project, 2000. Available at the NESSIE website - *http://www.nessie.eu.org/nessie/*.

[20] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, Hidema Tanaka, *The Block Cipher SC2000*, proceedings of Fast Software Encryption 8, 2001, to appear.

[21] David Wagner, *The Boomerang Attack*, proceedings of Fast Software Encryption 6, Lecture Notes in Computer Science 1636, pp. 156–170, Springer-Verlag, 1999.

[22] Hitoshi Yanami, Takeshi Shimoyama, *Differential and Linear Cryptanalysis of a Reduced-Round SC2000*, preproceedings of the NESSIE second workshop, 2001.

[23] Hitoshi Yanami, Takeshi Shimoyama, Orr Dunkelman, *Differential and Linear Cryptanalysis of a Reduced-Round SC2000*, these proceedings.