The Saturation Attack – a Bait for Twofish

(September 14, 2000)

Stefan Lucks*

Theoretische Informatik
University of Mannheim, 68131 Mannheim, Germany
`lucks@th.informatik.uni-mannheim.de`

Abstract. We introduce the notion of a "saturation attack" and present attacks on reduced-round versions of the Twofish block cipher. Our attack for all generic key sizes of Twofish (i.e., for 128-bit, 192-bit and 256-bit keys) improves on exhaustive key search for seven rounds of Twofish with full whitening, and for eight rounds of Twofish without whitening at the end. The core of the attack is a a key-independent distinguisher for six rounds of Twofish. The distinguisher is used to attack up to 7 rounds of Twofish with full whitening and and 8 rounds of Twofish with prewhitening only – half of the cipher. The attacks take up to 2^{127} chosen plaintexts (half of the codebook!) and are 2–4 times faster than exhaustive search.

1 Introduction

Modern b-bit block ciphers often use permutations $p : \{0,1\}^w \rightarrow \{0,1\}^w$ with $w < b$ as building blocks. E.g., p may be an S-box, a round function, or a group operation where one of the operands is constant. The constant may be unknown to the cryptanalyst, e.g. as a part of the (round) key. We regard the input for p as a data channel. For the cryptanalyst, p may be known or unknown, and the cryptanalysts may be unable to determine the input for p. A *"saturation attack"* is based on the idea of choosing a set of $k * 2^w$ plaintexts such each of the 2^w inputs for p occurs exactly k times. In this case, we say that the data channel into p is *"saturated"*. A saturation attack exploits the fact that if the input for p is saturated, then the output from p is saturated, too.

The "miss in the middle" attacks from [1] are rudimentarily related to satutation attacks, exploiting the fact that given two inputs $x \neq y$ for a permutation p one gets two outputs $p(x)$ and $p(y)$ with $p(x) \neq p(y)$.

While the name "saturation attack" is new, such attacks have been known before. E.g., the "Square attack" (which was developed as a dedicated attack on the block cipher Square [3]) is a saturation attack. It also works for other Square-like ciphers such as the AES candidate Crypton [9, 10] and the AES finalist Rijndael [4]. All these ciphers are 128-bit block ciphers with 8-bit data channels. The attack starts with a set of 2^8 plaintexts with one saturated channel,

---

* Supported by German Science Foundation (DFG) grant KR 1521/3-2.

while all the other channels are constant. After two rounds, all 16 data channels are saturated. After three rounds, the saturation property is likely to have been lost, but a simple linear relationship between the values of a data channel allows distinguishing the three-round output from random. The best currently known attacks on Crypton [2] and Rijndael [6] are extensions of the Square attack.

This paper shows that saturation attacks are a useful tool for ciphers which are definitely not Square-like. We concentrate on Twofish [13], another AES finalist. So far, the authors of Twofish published some preliminary cryptanalytic results [14, 5] themselves, a key separation property has been identified for Twofish [11, 12, 7], and some observations on the generation of the Twofish S-Boxes and on differential cryptanalysis have been made [8].

1.1 Notation

We will use the notion of a "multiset" to describe a w-bit data channel. A multiset with $k * 2^w$ entries is *"saturated"*, if every value in $\{0,1\}^w$ is found exactly k times in the multiset. If $k = 1$, a saturated multiset is the set $\{0,1\}^w$.

In the context of this paper, a data channel is always 32-bit wide, and we call a value in a data channel a *"word"*. We interchangeably view a word x as a 32-bit string $x = (x_{31}, \dots, x_0) \in \{0,1\}^{32}$ and as an unsigned integer $x = \sum_i x_i * 2^i$. The addition of values in a data channel is thus addition mod 2^{32}. We write "$x \lll b$" for the rotation of the word x by b bits to the left, and "$x \ggg b$" for rotation to the right. E.g. $(x \lll b) \ggg b = x$ for all x and b, and $(x_{31}, x_{30}, \dots, x_1, x_0) \lll 1 = (x_{30}, \dots, x_1, x_0, x_{31})$. $\mathrm{LSB}(x) = x \bmod 2$ denotes the *"least significant bit (LSB)"* of x, and $\mathrm{LSB}^1(x) = \mathrm{LSB}(x \operatorname{div} 2)$ denotes the 2nd-least significant bit. Similarly, we define the *"most significant bit (MSB)"*: $\mathrm{MSB}(x) = \mathrm{LSB}(x \lll 1)$. If the multiset M denotes a data channel, the bits at the LSB-position of M are *"balanced"* if $\bigoplus_{m \in M} \mathrm{LSB}(m) = 0$. For technical reasons, we also consider *"semi-saturated"* data channels. The multiset M is semi-saturated if one bit of M is constant and each of the 2^{31} remaining values for M appears exactly $2k$ times in M.

2 A Description of Twofish

In this section, we describe the structure of Twofish. We omit many details, concentrating on the properties of Twofish which are relevant for our attack.

2.1 The Main Operations of Twofish

Twofish is based on the following operations:

Whitening. A 128-bit block of Twofish is decomposed into words $a_0, \dots, a_3 \in \{0,1\}^{32}$. The Twofish whitening operation is the XOR of four key words $K_{j+\delta} \in \{0,1\}^{32}$ to the words a_j: $b_j := a_j \oplus K_{j+\delta}$ for $j \in \{0, \dots, 3\}$, as described in Figure 1.

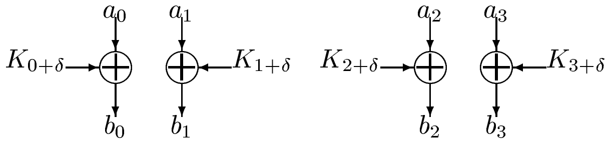

Fig. 1. The Twofish Whitening Operation.

Application of the round function. The i-th round function F_i takes two words $a, b \in \{0,1\}^{32}$ and computes a pair of words $(a', b') = F_i(a, b) \in (\{0,1\}^{32})^2$. The round function F_i is defined by two round keys K_{2i+2} and K_{2i+3} and two functions $G_1, G_2 : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$ (see also Figure 2):

$$a' := G_1(a) + G_2(b) + K_{2i+2}, \quad \text{and} \quad b' := G_1(a) + 2G_2(b) + K_{2i+3},$$

The functions G_1 and G_2 are key-dependent, but do not depend on i. Given the round function's results a' and b', the remaining two words $c, d \in \{0,1\}^{32}$ come into play:

$$x := (a' \oplus c) \lll 1, \quad \text{and} \quad y := b' \oplus (d \ggg 1).$$

Except for the rotate operations, Twofish works like a Feistel cipher.

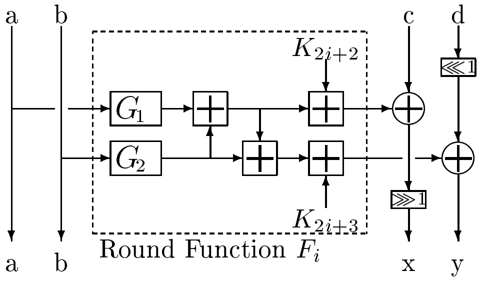

Fig. 2. The Application of the Twofish Round Function.

Our attack greatly depends on the functions G_1 and G_2 to be permutations over $\{0,1\}^{32}$. It also helps a little bit that G_1 and G_2 are almost the same function: $G_2(x) = G_1(x \lll 8)$. Except for that, the internal structure of G_1 and G_2 is not relevant for us.

The swap. The quadruple (a, b, c, d) is replaced by (c, d, a, b). See Fig. 3.

2.2 The Basic Structure of Twofish

Twofish uses a 16-round Feistel structure with two additional one-bit rotates in each round, pre-whitening before the first round and post-whitening after the last round. Twofish works as follows:

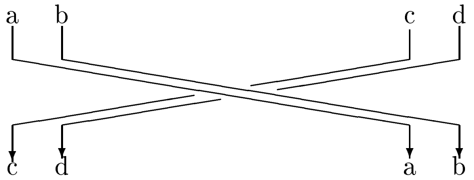

Fig. 3. The Twofish Swap.

1. Generate the key-dependent S-boxes, which define the functions G_1 and G_2.
2. Generate four subkey words $K_0, \ldots K_3 \in \{0,1\}^{32}$ for the pre-whitening, two subkey words K_{2i+2}, K_{2i+3} for each round and another four subkey words $K_{36}, \ldots, K_{39}$ for the post-whitening.
3. Given a plaintext block, do the pre-whitening.
4. For $i := 1$ to 15 do
   (a) Apply the round function F_i.
   (b) Do the swap.
5. Apply the last round function F_{16} (no swap in the final round).
6. Do the post-whitening.

The first two of the above steps constitute the "key schedule" described below. Note that we can obviously generalise the Twofish structure to r rounds, where the loop in step 4 is iterated $r-1$ times.

2.3 The Twofish Key Schedule

A Twofish key consists of 128, 192, or 256 bit[1]: $2k$ words $M_0, \ldots, M_{2k-1} \in \{0,1\}^{32}$ with $k \in \{2,3,4\}$, organised as two vectors $M_e = (M_0, M_2, \ldots, M_{2k-2})$ and $M_o = (M_1, M_3, \ldots, M_{2k-1})$. A third vector $S = (S_0, S_1, \ldots, S_{k-1})$ is derived from M_e and M_o by using techniques from the theory of Reed-Solomon codes. Given any two of the three vectors M_e, M_o and S, the third one is easy to find.

With these three vectors, the "three halfs of a Twofish key", we can do the first two steps of the abovely described structure:

1. The vector S is used to determine the internal S-boxes and thus the functions G_1 and G_2. Note that S is a k-word vector, while the key consists of $2k$ words or $64k$ bit.
2. The 40 subkey words $K_0, \ldots, K_{39}$ are defined by using functions h_e and h_o and by doing 20 *"subkey generation"* steps ($j \in \{0, \ldots, 19\}$):

$$\begin{aligned} A_j &:= h_e(j, M_e); & K_{2j} &:= A_j + B_j; \\ B_j &:= h_o(j, M_o); & K_{2j+1} &:= (A_j + 2B_j) \lll 9. \end{aligned}$$

---

[1] These are the three generic key lengths of Twofish. Other keys of less than 256 bit are padded to the next generic length by appending zeros.

3 Distinguishers for Twofish

A good block cipher is supposed to behave like a random permutation for anyone not knowing the secret key. In this section, we describe distinguishers. I.e., given a well-chosen set of plaintexts, we find properties in the corresponding set of ciphertexts which are unlikely in the case of a random permutation. This holds for reduced-round versions of Twofish under arbitrary keys.

3.1 A Four-Round Distinguisher

Consider 2^{32} plaintexts $(\alpha_0, \alpha_1, A, \alpha_3)$, where α_0, α_1, and α_3 are three arbitrary 32-bit constants and A is the set of all 2^{32} words. The pre-whitening changes this set of texts to $(\beta_0, \beta_1, A, \beta_3)$ with new constants β_i.

Given this set of texts as the input for the first round, the input for the round function F_1 is constant: (β_0, β_1). By (γ_0, γ_1) we denote the output of F_1, which then generates the texts $(\beta_0, \beta_1, A, \gamma_3)$ with $\gamma_3 = (\beta_3 \lll 1) \oplus \gamma_1$. (Note that $A = \{a_i\} = \{(a_i \oplus \gamma_0) \ggg 1\}$.) The swap changes these texts to $(A, \gamma_3, \beta_0, \beta_1)$.

In the second round, the 2^{32} inputs for the round function are (A, γ_3). The round function generates the pairs (b_i, c_i) with $b_i = G_1(a_i) + G_2(\gamma_3) + K_6$ and $c_i = G_1(a_i) + 2G_2(\gamma_3) + K_7$ for $a_i \in A$. The sets $B = \{b_i\}$ and $C = \{c_i\}$ are saturated, just like A. Applying the round function here means XORing the constant β_0 to the values of B, followed by a rotation, and XORing $\beta_1 \lll 1$ to C. Neither operations changes the saturated sets B and C. We get 2^{32} texts (A, γ_3, B, C), where A, B, and C are saturated. By the swap, we get texts (B, C, A, γ_3).

The 2^{32} inputs for the third round function are of the form (B, C) with saturated B and C. Since both G_1 and G_2 are permutations, $G_1(b_i) \neq G_1(b_j)$ and $G_2(c_i) \neq G_2(c_j)$ for $b_i, b_j \in B$, $c_i, c_j \in C$, and $i \neq j$. Let $d_i = G_1(b_i) + G_2(c_i) + K_8$ and $e_i = G_1(b_i) + 2G_2(c_i) + K_9$. The 2^{32} outputs of the round function are of the form (D, E), with the multisets $D = \{d_i | 0 \leq i < 2^{32}\}$ and $E = \{e_i | 0 \leq i < 2^{32}\}$. Neither D nor E is likely to be saturated. However, we are still able to observe a weaker property: Since $\sum_{b_i \in B} b_i = \sum_{c_i \in C} c_i = \sum_{0 \leq i < 2^{32}} i \equiv 2^{31} \bmod 2^{32}$:

$$\sum_{0 \leq i < 2^{32}} d_i \equiv 2^{31} + 2^{31} + 2^{32} * K_8 \equiv 0 \pmod{2^{32}},$$

$$\sum_{0 \leq i < 2^{32}} e_i \equiv 2^{31} + 2 * 2^{31} + 2^{32} * K_9 \equiv 2^{31} \pmod{2^{32}},$$

thus $\sum d_i \equiv \sum e_i \equiv 0 \pmod{2}$ – i.e., the LSBs of D and E are *balanced.*

Applying the round function means to evaluate 2^{32} pairs (f_i, g_i) with $f_i = f_i' \ggg 1$, $f_i' = a_i \oplus d_i$, and $g_i = (\gamma_3 \lll 1) \oplus e_i$. Define the multisets $F = \{f_i\}$, $F' = \{f_i'\}$, and $G = \{g_i\}$. We observe: The bits at the LSB-positions of both F' and G are balanced, and, due to the rotate, the bits at the MSB-position of F are balanced. Hence, the third round generates 2^{32} texts of the form (B, C, F, G), which are then swapped to (F, G, B, C).

The inputs for the fourth round are of the form (F, G). We write $(?, ?)$ for the outputs. Applying the round function gives us 2^{32} texts $(F, G, ?, ?)$. If this is followed by a swap, we finally get $(?, ?, F, G)$.

Having chosen 2^{32} plaintexts, we can check the balancedness of the ciphertexts bits at the two positions determined by the MSB of F and the LSB of G. Whatever the keys are, four rounds of Twofish always pass this test – even the post-whitening cannot destroy the balancedness. But a random permutation only passes this test with about a 25% probability.

3.2 Another Four-Round Distinguisher

Our second distinguisher works quite similarly to the first one. We start with 2^{32} plaintexts of the form $(\alpha_0, \alpha_1, \alpha_2, A)$ with arbitrary constants α_i. The pre-whitening changes the constants and we get texts $(\beta_0, \beta_1, \beta_2, A)$. After the first round, including the swap, these are $(\gamma_2, A, \beta_0, \beta_1)$.

In the second round, the inputs to the round function are of the form (γ_2, A), where $A = \{0, 1\}^{32}$ is a saturated set and γ_2 is constant. The round function generates the pairs (b_i, c_i) with $b_i = G_1(\gamma_2) + G_2(a_i) + K_6$ and $c_i = G_1(\gamma_2) + 2G_2(a_i) + K_7$ for $a_i \in A$. Now the set $B = \{b_i\}$ is saturated like A, but the multiset $C^* = \{c_i\}$ isn't. Instead, it is semi-saturated with a constant $\mathrm{LSB}(c_i) = \gamma^* \in \{0, 1\}$ for all $c_i \in C^*$: $\gamma^* = \mathrm{LSB}(G_1(\gamma_2)) \oplus \mathrm{LSB}(K_7)$. We apply the round function by adding some constants to the elements of B and C^*, and by then rotating the elements of B. The results are a saturated set B and a semi-saturated set C^*, as before. After the swap, we have texts of the form (B, C^*, γ_2, A).

In the third round the 2^{32} inputs for the round function are of the form (B, C^*). Consider the round function's outputs (d_i, e_i) with $d_i = G_1(b_i) + G_2(c_i) + K_8$ and $e_i = G_1(b_i) + 2G_2(c_i) + K_9$. Since $B = \{b_i\}$ is saturated, so is $\{G_1(b_i)\}$, and especially

$$\sum_{0 \le i < 2^{32}} G_1(b_i) \equiv 0 \pmod 2.$$

Since C^* is semi-saturated, it has 2^{31} different values, each repeated exactly twice. The same holds for the 2^{32} values $G_2(c_i)$ (with $c_i \in C^*$), hence

$$\sum_{0 \le i < 2^{32}} G_2(c_i) \equiv 0 \pmod 2.$$

Thus, both multisets $D = \{d_i\}$ and $E = \{e_i\}$ are balanced:

$$\sum_{0 \le i < 2^{32}} d_i \;=\; \sum_i G_1(b_i) + \sum_i G_2(c_i) + 2^{32} * K_8 \equiv 0 \pmod 2$$

and

$$\sum_{0 \le i < 2^{32}} e_i \;=\; \sum_i G_1(b_i) + 2 * \sum_i G_2(c_i) + 2^{32} * K_9 \equiv 0 \pmod 2.$$

By applying the round function and swapping, we get 2^{32} texts of the form (F, G, B, C^*). The bits at the LSB-position of G are balanced, as are the bits at the MSB-position of F (due to the one-bit rotate). The fourth round makes this $(F, G, ?, ?)$, and if we do the swap we get texts of the form $(?, ?, F, G)$.

A random permutation passes the corresponding test only with a probability of about 0.25.

3.3 An Extension to Five Rounds

Next, we show how to extend the distinguisher from Section 3.2 to five rounds. Let α an arbitrary 32-bit constant and c^* an arbitrary 1-bit constant. We choose all 2^{95} plaintexts of the form (α, a_i, b_j, c_k), with $c_i \operatorname{div} 2^{31} = c^*$. We write (α, A, B, C^+) for these 2^{95} texts. Note that the multisets A and B are saturated and the multiset C^+ is semi-saturated. The pre-whitening changes the constant α to β, and the constant c^* to γ^*, but leaves A and B saturated and C^+ semi-saturated with a constant MSB. We still have 2^{95} distinct input texts (β, A, B, C^+) for the first round.

Let $(e_i, f_i) = F_1(\beta, a_i)$ with $a_i \in A$. We can write $e_i = \beta_e + G_2(a_i)$ and $f_i = \beta_f + 2G_2(a_i)$, for some constants β_e, β_f. Hence the outputs of F_i consist of pairs (E, F^*) with saturated E and semi-saturated F^*. Set $\beta^* = f_i \bmod 2$ for the constant LSB of the values $f_i \in F$.

For every value $a_i \in A$ there are 2^{63} pairs (b_i, c_i) with a constant bit $\gamma^* = c_i \operatorname{div} 2^{31} = \mathrm{MSB}(c_i)$. We can fix any constants $\gamma_2, \gamma_3 \in \{0, 1\}^{32}$ with $\gamma_3 \bmod 2 = \gamma^* \oplus \beta^*$ and find pairs (b_i, c_i) in (B, C^*) such that $(e_i \oplus b_i) \ggg 1 = \gamma_2$ and $f_i \oplus (c_i \lll 1) = \gamma_3$ holds for every a_i. (Note that the MSB of c_i is the LSB of $c_i \lll 1$.)

Now the 2^{95} input texts (β, A, B, C^*) can be separated into 2^{63} disjoint groups of 2^{32} texts, determined by the pair (γ_2, γ_3) of constants, such that after applying the first round functions all texts in the same group are of the form $(\beta, A, \gamma_2, \gamma_3)$. The swap changes these to $(\gamma_2, \gamma_3, \beta, A)$.

For each such group, applying the four-round distinguisher from Section 3.2 would result in a set of 2^{32} ciphertexts $(?, ?, F, G)$, where the ciphertext bits at the LSB-position of G and at the MSB-position of F are balanced. Now, we do not know which ciphertexts belong into which group, but if these bits for each group are balanced, then so are all 2^{95} such bits. Five rounds of Twofish always pass this test, while a random permutation passes it with about 25% probability.

The same technique can also be applied to the distinguisher from Section 3.1. Here, we need 2^{96} plaintexts of the form (α, A, B, C) with constant α. A random permutation passes the corresponding test with about 25% probability.

3.4 An Extension to Six Rounds

To attack six rounds, we choose 2^{127} plaintexts (a_i, b_i, c_i, d_i), (half of the codebook (!)), where $b_i \operatorname{div} 2^{31} = \mathrm{MSB}(b_i)$ is fixed to an arbitrary constant. Our plaintexts are of the form (A, B^+, C, D), where A, B, and D are saturated multisets, and B^+ is a semi-saturated one.

Our choice of plaintexts ensures that for each of the 2^{63} left-side pairs (a_i, b_i) all 2^{64} right-side pairs (c_i, d_i) exist. Neither the pre-whitening nor the application of the first round function change this property. By the swap we get 2^{127} texts (C, D, A, B^+) as the input for the second round. For each 32-bit constant α we get a group of 2^{95} texts (α, D, A, B^+). These are 2^{32} disjoint groups which are the kind of input we need for the 5-round distinguisher.

After six rounds of Twofish, we get 2^{127} ciphertexts $(?, ?, F, G)$ with balanced bits at two positions. A random permutation does satisfy this with about 25% probability.

3.5 Distinguishers: Summary

In Table 1 we summarise the distinguishers we have found. We describe which section the distinguisher was described in, the number r of Twofish rounds the attack works for, the chosen plaintexts required (how they look like and how many we need), and the probability for a random permutation to pass the test. All tests are one-sided, i.e. r rounds of Twofish pass the test with probability 1.

Section	Rounds r	Chosen Plaintexts Form	Chosen Plaintexts Number	Probability
3.1	4	$(\alpha_0, \alpha_1, A, \alpha_3)$	2^{32}	25%
3.2	4	$(\alpha_0, \alpha_1, \alpha_2, A)$	2^{32}	25%
3.3	5	(α, A, B, C^+)	2^{95}	25%
3.3	5	(α, A, B, C)	2^{96}	25%
3.4	6	(A, B^+, C, D)	2^{127}	25%

Table 1. Distinguishers for Twofish.

4 Finding the Key

In modern cryptanalysis one often uses a distinguisher for some rounds of a product cipher to find the key: Guess some key bits for one or more additional rounds and exploit the distinguishing property to falsify wrong key guesses. This is, what we do below, concentrating on using the six-round distinguisher.

4.1 The Basic Technique

Consider seven rounds of Twofish. Let 2^{127} plaintexts be chosen as required for the six-round distinguisher. After six rounds (including the swap), we have 2^{127} text quadruples (a_i, b_i, c_i, d_i) of 32-bit words. We have two distinguishing properties: the bits at the LSB-position of the words d_i are balanced, and the bits at the MSB-position of the c_i-words are balanced. We start with concentrating on the first distinguishing property.

The XOR over the 2^{127} LSBs $\delta_i = d_i \bmod 2$ of d_i is:

$$\delta^* = \bigoplus_{0 \le i < 2^{127}} \delta_i = 0.$$

After seven rounds, we have 2^{127} ciphertext quadruples (w_i, x_i, y_i, z_i), and we cannot count on balanced ciphertext bits. Note that the seventh round uses the keys K_{16} and K_{17}, and the post-whitening keys denoted by $K_{18}, \ldots, K_{21}$. Figure 4 visualises the last round, including the post-whitening.

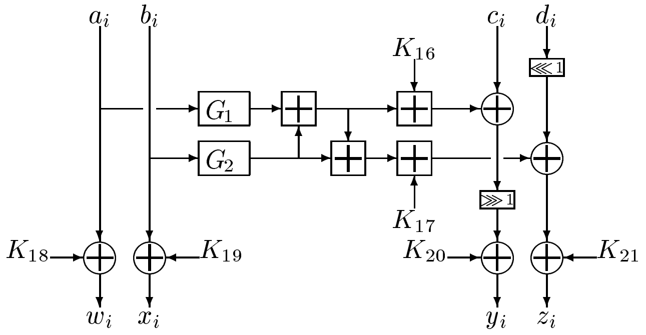

Fig. 4. The Seventh Round and the Post-Whitening.

Rewrite δ_i by $\delta_i = \delta_{w,i} \oplus (\delta^{\mathrm{c}}_{w,i} * \mathrm{LSB}(K_{17})) \oplus \delta_{x,i} \oplus \delta_{z,i} \oplus \mathrm{LSB}^1(K_{17})$ with

$$\begin{aligned}
\delta_{w,i} &= \mathrm{LSB}^1(G_1(w_i \oplus K_{18})) \\
\delta^{\mathrm{c}}_{w,i} &= \mathrm{LSB}(G_1(w_i \oplus K_{18})), \\
\delta_{x,i} &= \mathrm{LSB}(G_2(x_i \oplus K_{19})), \\
\delta_{z,i} &= \mathrm{LSB}^1(z_i) \oplus \mathrm{LSB}^1(K_{21}).
\end{aligned}$$

The necessity to consider the 2nd-least significant bits $\mathrm{LSB}^1(\ldots)$ is due to the last round's one-bit rotate: $\mathrm{LSB}(d_i) = \mathrm{LSB}^1(d_i \lll 1)$. Note that the value $(\delta^{\mathrm{c}}_{w,i} * \mathrm{LSB}(K_{17})) \in \{0, 1\}$ specifically deals with the carry bit generated at the LSB-position.

We can evaluate the bit δ_i by *partial* decryption. Since we are rather interested in the bit $\delta^* = \bigoplus_i \delta_i$, we rewrite δ^* by

$$\delta^* = \delta^*_w \oplus (\delta^{*\mathrm{c}}_w * \mathrm{LSB}(K_{17})) \oplus \delta^*_x \oplus \delta^*_z$$

with

$$\delta^*_w = \bigoplus_{0 \le i < 2^{127}} \delta_{w,i} = \bigoplus_i \left(\mathrm{LSB}^1(G_1(w_i \oplus K_{18})) \right),$$

$$\begin{aligned}
\delta_w^{*\mathrm{c}} &= \bigoplus_{0 \le i < 2^{127}} \delta_{w,i}^{\mathrm{c}} = \bigoplus_i \left(\mathrm{LSB}(G_1(w_i \oplus K_{18}))\right), \\
\delta_x^* &= \bigoplus_{0 \le i < 2^{127}} \delta_{x,i} = \bigoplus_i \mathrm{LSB}(G_2(x_i \oplus K_{19})), \\
\delta_z^* &= \bigoplus_{0 \le i < 2^{127}} \delta_{z,i} = \bigoplus_i \left(\mathrm{LSB}^1(z_i) \oplus \mathrm{LSB}^1(K_{21})\right) = \bigoplus_i \mathrm{LSB}^1(z_i).
\end{aligned}$$

Assume an adversary to know (or to have guessed) the S-boxes, i.e., to know the functions G_1 and G_2. Given 2^{127} ciphertexts (w_i, x_i, y_i, z_i) we can compute the bits δ_w^*, $\delta_w^{*\mathrm{c}}$, δ_x^* and δ_z^* independently.

For δ_z^* just count how often one of the 2^{127} bits $\mathrm{LSB}^1(z_i)$ is one – δ_z^* is just this number (modulo 2). Regarding δ_x^* we just need to consider all words x_i which appear an odd time as a part of a ciphertext (w_k, x_i, y_k, z_k). These are at most 2^{32} words x_i. Given these and the key word K_{19} we just count mod 2 how often $\mathrm{LSB}(G_2(x_i \oplus K_{19}))$ is one. Computing δ_w^* and $\delta^{*\mathrm{c}}$ can be done similarly.

4.2 Attacking Seven Rounds with full Whitening

In this section we describe and analyse the seven-round attack. Consider the following algorithm.

0. Choose 2^{127} plaintexts as required for the six-round distinguisher. Ask for the corresponding ciphertexts (w_i, x_i, y_i, z_i), $0 \le i < 2^{127}$.
1. For all $W \in \{0,1\}^{32}$: count (mod 2) the ciphertexts (W, x_i, y_i, z_i).
2. For all $X \in \{0,1\}^{32}$: count (mod 2) the ciphertexts (w_i, X, y_i, z_i).
3. Evaluate δ_z^* by counting the bits $\mathrm{LSB}^1(z_i)$ of the words z_i.
4. Guess the key-dependent vector S, defining the functions G_1 and G_2.
5. Consider the 2^{32} one-bit counters for ciphertexts (w_i, X, y_i, z_i). For every key $K_{19} \in \{0,1\}^{32}$ evaluate δ_x^*. Write $\delta_x^*(K_{19})$ for the results.
6. Consider the 2^{32} one-bit counters for ciphertexts (W, x_i, y_i, z_i). For every choice of K_{18} evaluate δ_w^* and $\delta_w^{*\mathrm{c}}$. Write $\delta_w^*(K_{18})$ and $\delta_w^{*\mathrm{c}}(K_{18})$ for the results.
7. The subkey triple $(\mathrm{LSB}(K_{17}), K_{18}, K_{19}) \in \{0,1\} \times \{0,1\}^{32} \times \{0,1\}^{32}$ is *"valid for S"* if and only if

$$\delta_w^*(K_{18}) \oplus (\delta_w^{*\mathrm{c}}(K_{18}) * \mathrm{LSB}(K_{17})) \oplus \delta_x^*(K_{19}) = \delta_z^*.$$

How expensive is this algorithm? Let $K \in \{128, 192, 256\}$ be the key size, i.e. each of the "key halfs" M_e, M_o, and S consists of $K/64 \in \{2,3,4\}$ words. Obviously, we need 2^{127} chosen plaintexts, half of the codebook. Each of the steps 1–3 requires 2^{127} very simple operations (incrementing a one-bit counter). Step 4 requires to guess $K/2$ key bits. Each of the following steps is repeated $2^{K/2}$ times:

- Note that we can compute and store a table with 2^{32} entries for the function G_1. Then for each key K_{19}, step 5 requires an average of 2^{31} very simple operations (essentially table look-ups).

- Since $G_2(x) = G_1(x \ggg 8)$, we can reuse the table for G_1 for the function G_2. Then for Each key K_{18}, step 6 requires $2 * 2^{32}$ very simple operations, on the average.

Hence this algorithm requires about $3 * 2^{(K/2)+64}$ *"very simple"* operations. Since such an operation is much faster than a single Twofish encryption, the attack is much faster than exhaustive key search for $K \geq 128$.

The algorithm allows us to filter out half of the keys. (I.e., about 50% of all random permutations pass the test.) So a key finding attack based on the algorithm has to exhaustively search the remaining half of the key space, requiring an expected number of 2^{K-2} Twofish encryptions.

4.3 Attacking Eight Rounds Without Post-Whitening

The previous attack can be modified to work for eight rounds of Twofish without post-whitening. Let the usual 2^{127} plaintext be chosen. After six rounds, including the swap, we have 2^{127} quadruples (a_i, b_i, c_i, d_i) of 32 bit words, and

$$\delta^* = 0 = \bigoplus_{0 \leq i < 2^{127}} \mathrm{LSB}(d_i).$$

Applying the seven round functions changes these to (a_i, b_i, e_i, f_i), the swap to (e_i, f_i, a_i, b_i), and the last round to (e_i, f_i, g_i, h_i). See Figure 5. Note that K_{18} and K_{19} are no longer part of the post-whitening key, but constitute the round key for round eight.

Set $x_i = G_1(e_i) + G_2(f_i)$ and $y_i = G_1(e_i) + 2G_2(f_i)$. For every key-dependent k-word vector S we know a mapping from the 2^{64} pairs (e_i, f_i) to the pairs (x_i, y_i). So we can virtually blow up the 2^{127} ciphertext quadruples (e_i, f_i, g_i, h_i) to six-tuples $(e_i, f_i, x_i, y_i, g_i, h_i)$. Computing the mapping requires about the time of 2^{64} one-round decryptions.

The attack on eight rounds of Twofish works quite similarly to the previous attack. We decompose σ^* into key-dependent bits $\delta_a^*(K_{18})$, $\delta_a^{*c}(K_{18})$, $\delta_b^*(K_{19})$ and δ_f with

$$\begin{aligned}
\delta_a^*(K_{18}) &= \bigoplus_i \left(\mathrm{LSB}^1(G_1(a_i))\right) \\
&= \bigoplus_i \left(\mathrm{LSB}^1(G_1((x_i + K_{18}) \oplus (g_i \lll 1)))\right), \\
\delta_a^{*c}(K_{18}) &= \bigoplus_i (\mathrm{LSB}(G_1(a_i))) \\
&= \bigoplus_i (\mathrm{LSB}(G_1((x_i + K_{18}) \oplus (g_i \lll 1)))), \\
\delta_b^*(K_{19}) &= \bigoplus_i (\mathrm{LSB}(G_2(b_i)))
\end{aligned}$$

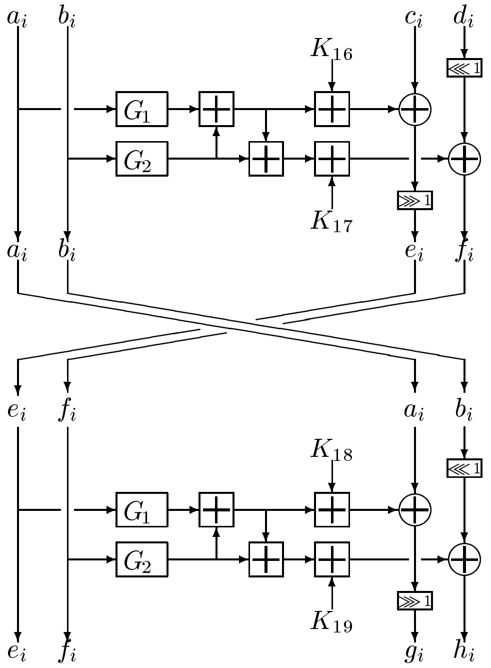

Fig. 5. Round Seven and Eight Without Post-Whitening.

$$= \bigoplus_i \left(\mathrm{LSB}(G_2(((y_i + K_{19}) \oplus h_i) \ggg 1)))\right),$$

and $$\delta_f^* = \bigoplus_i \left(\mathrm{LSB}^1(f_i)\right).$$

As above, a subkey triple $(\mathrm{LSB}(K_{17}), K_{18}, K_{19}) \in \{0,1\} \times \{0,1\}^{32} \times \{0,1\}^{32}$ is *"valid for S"* if and only if

$$\delta_a^*(K_{18}) \oplus (\delta_a^{*c}(K_{18}) * \mathrm{LSB}(K_{17})) \oplus \delta_b^*(K_{19}) = \delta_f^*.$$

As before, the basic idea for the attack is to run a filtering algorithm to sort out 50% of the keys, and then to exhaustively search the remaining half of the keys. The filtering is *much* faster than exhaustive key search, key finding with using this attack takes the equivalent of about 2^{K-2} encryptions and hence is twice as fast as exhaustively searching the entire key space.

4.4 An Improvement for 256-bit Keys

Reconsider seven rounds of Twofish and 2^{127} plaintexts chosen as as before, as described in Section 4.2 and Figure 4. So far, we only have used one distinguishing property: the bits at the LSB-position of the words d_i are balanced. Thus we could filter out half of the keys. Using the second property too, we can filter out 75 % of the keys: the bits at the MSB-position of the c_i-words are balanced.

Set $u_i = G_1(w_i \oplus K_{18}) + G_2(x_i \oplus K_{19})$ and $v_i = u_i \oplus K_{16}$. Then $c_i = v_i \oplus ((y_i \oplus K_{20}) \ggg 1)$. The bits at the MSB-position of c_i are balanced, i,e.,

$$\bigoplus_{0 \le i < 2^{127}} \mathrm{MSB}(c_i) = 0 \quad \Longleftrightarrow \quad \bigoplus_{0 \le i < 2^{127}} \mathrm{MSB}(v_i) = \bigoplus_{0 \le i < 2^{127}} \mathrm{MSB}(y_i \lll 1).$$

Consider running the following algorithm:

0. Choose 2^{127} plaintexts as before and ask for the ciphertexts (w_i, x_i, y_i, z_i).
1. For all $(W, X) \in (\{0,1\}^{32})^2$: count (mod 2^{32}) the ciphertexts (W, X, y_i, z_i).
2. Evaluate δ_y^{**} by counting the bits $\mathrm{MSB}(y_i \lll 1)$
3. Guess the key-dependent vector S, defining the functions G_1 and G_2.
4. For each of the 2^{64} key pairs (K_{18}, K_{19}), and every $u_i \in \{0,1\}^{32}$:
   Use the 2^{64} counters from Step 1 to compute counters $\mathrm{cnt}[u_i, K_{18}, K_{19}]$ for the values $u_i = G_1(w_i \oplus K_{18}) + G_2(x_i \oplus K_{19})$.
5. For each of the 2^{32} keys K_{16}, compute

   $$\delta_v^{**}(K_{18}, K_{18}, K_{20}) = \bigoplus_i \mathrm{MSB}((u_i + K_{16}) * \mathrm{cnt}[u_i, K_{18}, K_{19}]).$$

6. The triple $(K_{16}, K_{18}, K_{19}) \in (\{0,1\}^{32})^3$ is *"good for S"* if and only if

   $$\delta_v^{**}(K_{18}, K_{18}, K_{20}) = \delta_y^{**}.$$

Similar to Section 4.2, the above algorithm requires 2^{127} chosen plaintexts, and each of the steps 1–2 requires 2^{127} very simple operations (incrementing a 32-bit counter). Step 4 requires to guess $K/2$ key bits, and both the steps 4 and 5 are repeated $2^{K/2}$ times. Step 4 deals with 2^{64} counters and 2 key words, and step 5 deals with 2^{32} values u_i and 3 key words. Hence, both steps require 2^{128} very simple operations.

The entire algorithm requires about $2 * 2^{(K/2)+64+64} = 2^{(K/2)+129}$ *"very simple"* operations and is clearly faster than exhaustive key search for $K \geq 256$.

The same technique works for the attack on eight rounds of Twofish without post-whitening, too. So for $K = 256$ a key finding attack based on checking subkeys being both *valid* and *good* requires computation time equivalent to about 2^{K-3} Twofish encryptions.

4.5 Finding the Keys: Summary

In Table 2 we summarise our key finding attacks and compare them with exhaustive key search. Note that the running time depends on the number $K \in \{128, 192, 256\}$ of key bits, and we use one encryption as unit of time. It turns out that our attacks are 2–4 times faster than exhaustive key search, at the expense of a huge amount of plaintexts to be chosen.

Attack	Plaintexts	Rounds	Key Bits	Whitening		Running Time
				Pre-	Post-	
Section 4.2	2^{127} chosen	7	$K \geq 128$	yes	yes	2^{K-2}
Section 4.2 / 4.4	2^{127} chosen	7	$K = 256$	yes	yes	2^{K-3}
Section 4.3	2^{127} chosen	8	$K \geq 128$	yes	no	2^{K-2}
Section 4.3 / 4.4	2^{127} chosen	8	$K = 256$	yes	no	2^{K-3}
ex. key search	≈ 3 known	any	any	any	any	2^{K-1}

Table 2. Key Finding Attacks for Twofish.

5 Conclusion

At present, this paper describes the best known attack on the AES finalist Twofish. We are able to break up to 8 rounds of Twofish with one-sided whitening faster than exhaustively searching the key would require, though our attacks are only 2–4 times faster than exhausive key search. Since Twofish is a 16-round cipher with twosided whitening, we are able to penetrate exactly one half of Twofish. This still leaves Twofish with a reasonable security margin.

An interesting side-note is that the one-bit rotate operations prove to be useful – a variant of Twofish without these rotates would allow to enhance our attacks somewhat.

Finally, the paper demonstrates the usefulness of saturation attacks.

Acknowledgements

Supported by Counterpane and Hi/fn, the author participated at the "Third Twofish Retreat" in Provo in the state of Uta, USA. He thanks the workshop participants for their encouragement in considering saturation attacks for Twofish.

References

1. E. Biham, A. Biryukov, A. Shamir, "Miss in the Middle Attacks on IDEA and Khufru", *Fast Software Encryption 1999*, Springer LNCS 1636, pp. 124–138.
2. C. D'Halluin, G. Bijnens, V. Rijmen, B., Preneel, "Attack on six round of Crypton", *Fast Software Encryption 1999*, Springer LNCS 1636, pp. 46–59.
3. J. Daemen, L. Knudsen, V. Rijmen: "The block cipher Square", *Fast Software Encryption 1997*, Springer LNCS 1267, pp. 149–165.
4. J. Daemen, V. Rijmen, "AES proposal: Rijndael" (2nd version), *AES Submission* [15].
5. N. Ferguson, "Impossible differentials in Twofish", Twofish Technical Report #5, 1999 [15].
6. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, D. Whiting, "Improved Cryptanalysis of Rijndael" *Fast Software Encryption 2000*, to appear.
7. J. Kelsey, "Key Separation in Twofish", Twofish Technical Report #7, 2000 [15].

8. L. Knudsen, “Trawling Twofish (revisited)” May 15, 2000, [15].
9. C. H. Lim, “Crypton: a new 128-bit block cipher”, *AES Submission* [15].
10. C. H. Lim, “A revised version of Crypton – Crypton V 1.0 – ”, Fast Software Encryption 1999, Springer LNCS 1636, pp. 31–45.
11. F. Mirza, S. Murphy, “An Observation on the Key Schedule of Twofish”, In: *The 2nd Advanced Encryption Standard Conference*, pages 151–154, NIST, April 1999 [15].
12. , S. Murphy, “The Key Separation of Twofish”, 2000 [15].
13. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, “Twofish: A 128-bit Block Cipher”, *AES Submission* [15].
14. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, *“The Twofish Encryption Algorithm”*, Wiley, 1999.
15. “`http://d-ngm---t`