On The Twofish Key Schedule

Bruce Schneier* John Kelsey† Doug Whiting‡ David Wagner§ Chris Hall¶
Niels Ferguson ||

July 15, 1998

Abstract

Twofish is a new block cipher with a 128 bit block, and a key length of 128, 192, or 256 bits, which has been submitted as an AES candidate. In this paper, we briefly review the structure of Twofish, and then discuss the key schedule of Twofish, and its resistance to attack. We close with some open questions on the security of Twofish's key schedule.

1 Introduction

NIST announced the Advanced Encryption Standard (AES) program in 1997 [NIST97a]. NIST solicited comments from the public on the proposed standard, and eventually issued a call for algorithms to satisfy the standard [NIST97b]. The intention is for NIST to make all submissions public and eventually, through a process of public review and comment, choose a new encryption standard to replace DES.

Twofish is our submission to the AES selection process. It meets all the required NIST criteria—128-bit block; 128-, 192-, and 256-bit key; efficient on various platforms; etc.—and some strenuous design requirements, performance as well as cryptographic, of our own.

Twofish was designed to meet NIST's design criteria for AES [NIST97b]. Specifically, they are:

- A 128-bit symmetric block cipher.
- Key lengths of 128 bits, 192 bits, and 256 bits.
- No weak keys.
- Efficiency, both on the Intel Pentium Pro and other software and hardware platforms.
- Flexible design: e.g., accept additional key lengths; be implementable on a wide variety of platforms and applications; and be suitable for a stream cipher, hash function, and MAC.
- Simple design, both to facilitate ease of analysis and ease of implementation.

A central feature of Twofish's security and flexibility is its key schedule. In this paper, we will briefly review the design of Twofish, and then discuss the security features of the key schedule. The remainder of the paper is as follows: First, we discuss the specific design of Twofish. Next, we analyze the Twofish key schedule in some detail. Finally, we point out some open questions with respect to the key schedule. Note that for space reasons, this paper does not include a complete discussion of the Twofish design. Instead, we refer the reader to `http://www.counterpane.com`.

*Counterpane Systems; 101 E Minnehaha Parkway, Minneapolis, MN 55419, USA; schneier@counterpane.com.
†Counterpane Systems; kelsey@counterpane.com.
‡Hi/fn, Inc., 5973 Avenida Encinas Suite 110, Carlsbad, CA 92008, USA; dwhiting@hifn.com.
§University of California Berkeley, Soda Hall, Berkeley, CA 94720, USA; daw@cs.berkeley.edu.
¶Counterpane Systems; hall@counterpane.com.
||Counterpane Systems; niels@counterpane.com.

2 Twofish

Figure 1 shows an overview of the Twofish block cipher. Twofish uses a 16-round Feistel-like structure with additional whitening of the input and output. The only non-Feistel elements are the 1-bit rotates. The rotations can be moved into the F function to create a pure Feistel structure, but this requires an additional rotation of the words just before the output whitening step.

The plaintext is split into four 32-bit words. In the input whitening step, these are XORed with four key words. This is followed by sixteen rounds. In each round, the two words on the left are used as input to the g functions. (One of them is rotated by 8 bits first.) The g function consists of four byte-wide key-dependent S-boxes, followed by a linear mixing step based on an MDS matrix. The results of the two g functions are combined using a Pseudo-Hadamard Transform (PHT), and two keywords are added. These two results are then XORed into the words on the right (one of which is rotated left by 1 bit first, the other is rotated right afterwards). The left and right halves are then swapped for the next round. After all the rounds, the swap of the last round is reversed, and the four words are XORed with four more key words to produce the ciphertext.

More formally, the 16 bytes of plaintext $p_0, \ldots, p_{15}$ are first split into 4 words $P_0, \ldots, P_3$ of 32 bits each using the little-endian convention.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \qquad i = 0, \ldots, 3$$

In the input whitening step, these words are XORed with 4 words of the expanded key.

$$R_{0,i} = P_i \oplus K_i \qquad i = 0, \ldots, 3$$

In each of the 16 rounds, the first two words are used as input to the function F, which also takes the round number as input. The third word is XORed with the first output of F and then rotated right by one bit. The fourth word is rotated left by one bit and then XORed with the second output word of F. Finally, the two halves are exchanged. Thus,

$$\begin{aligned}
(F_{r,0}, F_{r,1}) &= F(R_{r,0}, R_{r,1}, r) \\
R_{r+1,0} &= \mathrm{ROR}(R_{r,2} \oplus F_{r,0}, 1) \\
R_{r+1,1} &= \mathrm{ROL}(R_{r,3}, 1) \oplus F_{r,1} \\
R_{r+1,2} &= R_{r,0} \\
R_{r+1,3} &= R_{r,1}
\end{aligned}$$

for $r = 0, \ldots, 15$ and where ROR and ROL are functions that rotate their first argument (a 32-bit word) left or right by the number of bits indicated by their second argument.

The output whitening step undoes the 'swap' of the last round, and XORs the data words with 4 words of the expanded key.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \qquad i = 0, \ldots, 3$$

The four words of ciphertext are then written as 16 bytes $c_0, \ldots, c_{15}$ using the same little-endian conversion used for the plaintext.

$$c_i = \left\lfloor \frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8 \qquad i = 0, \ldots, 15$$

2.1 The Function

The function F is a key-dependent permutation on 64-bit values. It takes three arguments, two input words R_0 and R_1, and the round number r used to select the appropriate subkeys. R_0 is passed through the g function, which yields T_0. R_1 is rotated left by 8 bits and then passed through the g function to yield T_1.

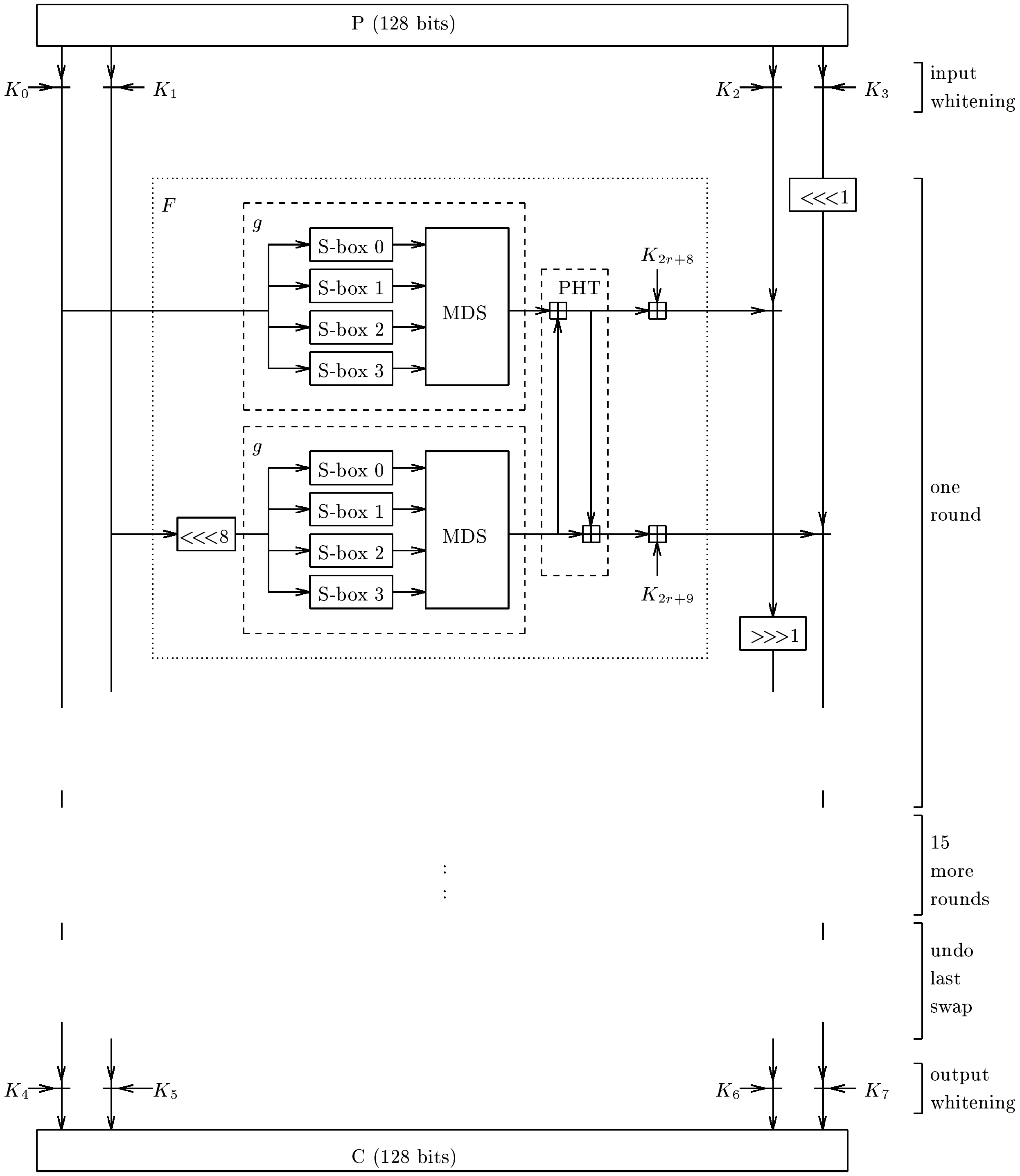

Figure 1: Twofish

The results T_0 and T_1 are then combined in a PHT and two words of the expanded key are added.

$$\begin{aligned} T_0 &= g(R_0) \\ T_1 &= g(\mathrm{ROL}(R_1, 8)) \\ F_0 &= (T_0 + T_1 + K_{2r+8}) \bmod 2^{32} \\ F_1 &= (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32} \end{aligned}$$

where (F_0, F_1) is the result of F. We also define the function F' for use in our analysis. F' is identical to the F function, except that it does not add any key blocks to the output. (The PHT is still performed.)

2.2 The Function

The function g forms the heart of Twofish. The input word X is split into four bytes. Each byte is run through its own key-dependent S-box. Each S-box is bijective, takes 8 bits of input, and produces 8 bits of output. The four results are interpreted as a vector of length 4 over $\mathrm{GF}(2^8)$, and multiplied by the 4×4 MDS matrix (using the field $\mathrm{GF}(2^8)$ for the computations). The resulting vector is interpreted as a 32-bit word which is the result of g.

$$\begin{aligned} x_i &= \lfloor X/2^{8i} \rfloor \bmod 2^8 \qquad i = 0, \ldots, 3 \\ y_i &= s_i[x_i] \qquad i = 0, \ldots, 3 \\ \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} &= \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & \text{MDS} & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \\ Z &= \sum_{i=0}^{3} z_i \cdot 2^{8i} \end{aligned}$$

where s_i are the key-dependent S-boxes and Z is the result of g. For this to be well-defined, we need to specify the correspondence between byte values and the field elements of $\mathrm{GF}(2^8)$. We represent $\mathrm{GF}(2^8)$ as $\mathrm{GF}(2)[x]/v(x)$ where $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ is a primitive polynomial of degree 8 over $\mathrm{GF}(2)$. The field element $a = \sum_{i=0}^{7} a_i x^i$ with $a_i \in \mathrm{GF}(2)$ is identified with the byte value $\sum_{i=0}^{7} a_i 2^i$. This is in some sense the "natural" mapping; addition in $\mathrm{GF}(2^8)$ corresponds to a XOR of the bytes.

2.3 The Key Schedule

The key schedule has to provide 40 words of expanded key $K_0, \ldots, K_{39}$, and the 4 key-dependent S-boxes used in the g function. Twofish is defined for keys of length $N = 128$, $N = 192$, and $N = 256$. Keys of any length shorter than 256 bits can be used by padding them with zeroes until the next larger defined key length.

We define $k = N/64$. The key M consists of $8k$ bytes $m_0, \ldots, m_{8k-1}$. The bytes are first converted into $2k$ words of 32 bits each

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0, \ldots, 2k-1$$

and then into two word vectors of length k.

$$\begin{aligned} M_e &= (M_0, M_2, \ldots, M_{2k-2}) \\ M_o &= (M_1, M_3, \ldots, M_{2k-1}) \end{aligned}$$

A third word vector of length k is also derived from the key. This is done by taking the key bytes in groups of 8, interpreting them as a vector over $\mathrm{GF}(2^8)$, and multiplying them by a 4×8 matrix derived from a

Reed-Solomon code. Each result of 4 bytes is then interpreted as a 32-bit word. These words make up the third vector.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & \text{RS} & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

$$S_i = \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}$$

for $i = 0, \ldots, k-1$, and

$$S = (S_{k-1}, S_{k-2}, \ldots, S_0)$$

Note that S lists the words in "reverse" order. For the RS matrix multiply, $\mathrm{GF}(2^8)$ is represented by $\mathrm{GF}(2)[x]/w(x)$, where $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ is another primitive polynomial of degree 8 over GF(2). The mapping between byte values and elements of $\mathrm{GF}(2^8)$ uses the same definition as used for the MDS matrix multiply.

2.3.1 Additional Key Lengths

Twofish can accept keys of any byte length up to 256 bits. For key sizes that are not defined above, the key is padded at the end with zero bytes to the next larger length that is defined. For example, an 80-bit key $m_0, \ldots, m_9$ would be extended by setting $m_i = 0$ for $i = 10, \ldots, 15$ and treating it as a 128-bit key.

2.3.2 The Function h

The function h takes two inputs—a 32-bit word X and a list $L = (L_0, \ldots, L_{k-1})$ of 32-bit words of length k—and produces one word of output. This function works in k stages. In each stage, the four bytes are each passed through a fixed S-box, and XORed with a byte derived from the list. Finally, the bytes are once again passed through a fixed S-box, and the four bytes are multiplied by the MDS matrix just as in g. More formally: we split the words into bytes.

$$\begin{aligned} l_{i,j} &= \lfloor L_i / 2^{8j} \rfloor \bmod 2^8 \\ x_j &= \lfloor X / 2^{8j} \rfloor \bmod 2^8 \end{aligned}$$

for $i = 0, \ldots, k-1$ and $j = 0, \ldots, 3$. Then the sequence of substitutions and XORs is applied.

$$y_{k,j} = x_j \qquad j = 0, \ldots, 3$$

If $k = 4$ we have

$$\begin{aligned} y_{3,0} &= q_1[y_{4,0}] \oplus l_{3,0} \\ y_{3,1} &= q_0[y_{4,1}] \oplus l_{3,1} \\ y_{3,2} &= q_0[y_{4,2}] \oplus l_{3,2} \\ y_{3,3} &= q_1[y_{4,3}] \oplus l_{3,3} \end{aligned}$$

If $k \geq 3$ we have

$$\begin{aligned} y_{2,0} &= q_1[y_{3,0}] \oplus l_{2,0} \\ y_{2,1} &= q_1[y_{3,1}] \oplus l_{2,1} \\ y_{2,2} &= q_0[y_{3,2}] \oplus l_{2,2} \\ y_{2,3} &= q_0[y_{3,3}] \oplus l_{2,3} \end{aligned}$$

In all cases we have

$$\begin{aligned}
y_0 &= q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}] \\
y_1 &= q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}] \\
y_2 &= q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}] \\
y_3 &= q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}]
\end{aligned}$$

Here, q_0 and q_1 are fixed permutations on 8-bit values that we will discuss shortly. The resulting vector of y_i's is multiplied by the MDS matrix, just as in the g function.

$$\begin{aligned}
\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} &= \begin{pmatrix} . & \cdots & . \\ \vdots & \text{MDS} & \vdots \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \\
Z &= \sum_{i=0}^{3} z_i \cdot 2^{8i}
\end{aligned}$$

where Z is the result of h.

2.3.3 The Key-dependent S-boxes

We can now define the S-boxes in the function g by

$$g(X) = h(X, S)$$

That is, for $i = 0, \ldots, 3$, the key-dependent S-box s_i is formed by the mapping from x_i to y_i in the h function, where the list L is equal to the vector S derived from the key.

2.3.4 The Expanded Key Words K_j

The words of the expanded key are defined using the h function.

$$\begin{aligned}
\rho &= 2^{24} + 2^{16} + 2^8 + 2^0 \\
A_i &= h(2i\rho, M_e) \\
B_i &= \mathrm{ROL}(h((2i+1)\rho, M_o), 8) \\
K_{2i} &= (A_i + B_i) \bmod 2^{32} \\
K_{2i+1} &= \mathrm{ROL}((A_i + 2B_i) \bmod 2^{32}, 9)
\end{aligned}$$

The constant ρ is used here to duplicate bytes; it has the property that for $i = 0, \ldots, 255$, the word $i\rho$ consists of four equal bytes, each with the value i. The function h is applied to words of this type. For A_i the byte values are $2i$, and the second argument of h is M_e. B_i is computed similarly using $2i + 1$ as the byte value and M_o as the second argument, with an extra rotate over 8 bits. The values A_i and B_i are combined in a PHT. One of the results is further rotated by 9 bits. The two results form two words of the expanded key.

2.3.5 The Permutations q_0 and q_1

The permutations q_0 and q_1 are fixed permutations on 8-bit values. They are constructed from four different 4-bit permutations each. We have investigated the resulting 8-bit permutations, q_0 and q_1, extensively, and believe them to be at least no weaker than randomly selected 8-bit permutations.

3 Analysis of The Key Schedule

The key schedule has been designed to provide resistance to attack, while also providing a great deal of flexibility in implementation. For example, after S has been computed from M_e, M_o, all remaining key scheduling can be done "on the fly" during encryption. This allows for very low-memory implementations, and for implementations with excellent key agility. In implementations with more memory, all the subkeys can be precomputed for improved performance. In implementations with still more memory, such as on modern high-end processors with a reasonable RAM cache size, the effects of the key-dependent S-boxes and the MDS matrix multiply can be precomputed, reducing the work per g computation to four table lookups and three XORs.

Note that S is only half the size of the key. This was done so that precomputation of the S-boxes and MDS matrix multiply would be sufficiently fast, and so that low-memory implementations would not have to take too large a performance hit. This means that the g function is slightly different for longer keys than for shorter keys.

3.1 Byte Sequences

The subkeys in Twofish are generated by using the h function, which can be seen as four key-dependent S-boxes followed by an MDS matrix. The input to the S-boxes is basically a counter. In this section we analyze the sequences of outputs that this construction can generate.

All key material is used to define key-dependent S-boxes in h, which are then used to derive subkeys. Each S-box gets a sequence of inputs, $(0, 2, 4, \ldots, 38)$ or $(1, 3, 5, \ldots, 39)$. The S-box generates a corresponding sequence of outputs. The corresponding outputs from the four S-boxes are combined using the MDS matrix multiply to produce the sequence of A_i and B_i words, and those words are processed with the PHT (with a couple of rotations thrown in) to produce a pair of subkey words. Analyzing these byte sequences thus gives us important insights about the whole key schedule.

We can model each byte sequence generated by a key-dependent S-box as a randomly selected non-repeating byte sequence of length 20. This allows us to make many useful predictions about the likelihood of finding keys or pairs of keys with various interesting properties. Because we will be analyzing the key schedule using this assumption in the remainder of this section, we should discuss how reasonable it is to treat this byte sequence as randomly generated. We have not found any statistical deviations between our key-dependent S-boxes and the random model in any of our extensive statistical tests.

We are looking at 20-byte-long sequences of distinct bytes. There are $256!/236!$ of those sequences, which is close to 2^{159}.

3.2 Equivalent S-box Keys

We have verified that there are no equivalent S-box keys that generate the same sequence of 20 bytes. In the random model, the chance of this happening for the $N = 256$ case is about $2^{63} \cdot 2^{-159} = 2^{-96}$. This is the chance of such equivalent S-boxes existing at all. In fact, we recently completed an exhaustive search demonstrating that no pair of key inputs to an S-box produces identical S-box entries.

3.3 Byte Difference Sequences

Let us consider the more general problem of how to get a given 20-byte difference sequence between a pair of S-boxes. Suppose we have two S-boxes, each defined using 32 bits of key material, which are not equal, but which must be chosen to give us a given difference sequence in the XOR of their byte sequences. We can estimate the probability of a pair of 4-byte inputs existing with the desired XOR difference sequence as $2^{63} \cdot 2^{-159} = 2^{-96}$. Note that this is the probability that such a pair of inputs exists, not the probability that a random pair of keys will have this property.

3.4 The and Sequences

¿From the properties of the byte sequences, we can discuss the properties of the A and B sequences generated by each key M.

$$A_i = \mathrm{MDS}(s_0(i, M), s_1(i, M), s_2(i, M), s_3(i, M))$$

Since the MDS matrix multiply is invertible, and since i is different for each round's subkey words generated, we can see that no A or B value can repeat itself.

Similarly, we can see from the construction of h that each key byte affects exactly one S-box used to generate A or B. Changing a single key byte always alters every one of the 20 bytes of output from that S-box, and so always alters every word in the 20-word A or B sequence to which it contributes.

Consider a single byte of output from one of the S-boxes. If we cycle any one of the key bytes that contributes to that S-box through all 256 possible values, the output of the S-box will also cycle through all 256 possible values. If we take four key bytes that contribute to four different S-boxes, and we cycle those four bytes through all possible values, then the result of h will also cycle through all possible values. This proves that A and B are uniformly distributed for all key lengths, assuming the key M is uniformly distributed.

3.5 Difference Sequences in and

Let us also consider difference sequences. If we have a specific difference sequence we want to see in A, we are faced with an interesting problem: since the MDS matrix multiply is XOR-linear, each desired output XOR from the matrix multiply allows only one possible input XOR. This means that:

1. A zero output XOR difference in A can occur *only* with a zero output XOR difference in all four of the byte sequences used to build A.
2. Only 1020 possible output differences (out of the 2^{32}) in A_i can occur with a single "active" (altered) S-box. Most differences require all four S-boxes used to form A_i to be active.
3. Each desired output XOR in A requires a specific output XOR in each of the four byte sequences used to form A. This means that getting any desired difference sequence into all 20 A_i values requires getting a desired XOR sequence into all four 20-byte sequences. (Note that if the desired output XOR in A_i is an appropriate value, up to three of the four byte sequences can be identical without much trouble, simply by leaving their key material unchanged.) As mentioned above, this is very unlikely to be possible for a randomly chosen difference pattern in the A sequence. (There are of course difference sequences of A_i's that can occur.)

The above analysis is of course also valid for the B sequence.

3.6 The Sequence ($_{2i}$ $_{2i+1}$)

As A_i and B_i are uniformly distributed (over all keys), so are all the K_i. As all pairs (A_i, B_i) are distinct, all the pairs (K_{2i}, K_{2i+1}) are distinct, although it might happen that $K_i = K_j$ for any pair of i and j.

3.7 Difference Sequences in the Subkeys

Difference sequences in A and B translate into difference sequences in (K_{2i}, K_{2i+1}). However, while it is natural to consider A and B difference sequences in terms of XOR differences, subkeys can reasonably be considered either as XOR differences or as differences modulo 2^{32}. Thus, we may discuss difference sequences:

$$\begin{aligned} D[i, M, M^*] &= K_{i,M} - K_{i,M} \\ X[i, M, M^*] &= K_{i,M} \oplus K_{i,M} \end{aligned}$$

where the difference is computed between the key value M and M^*.

3.8 XOR Differences in the Subkeys

Each round, the subkeys are added to the results of the PHT of two g functions, and the results of those additions are XORed into half of the cipher block. An XOR difference in the subkeys has a fairly high probability of passing through the addition operation and ending up in the cipher block. (The probability of this is determined by the Hamming weight of the XOR difference, not counting the highest-order bit.) However, to get into the subkeys, a XOR difference must first pass through the first addition.

Consider

$$\begin{aligned} x + y &= z \\ (x \oplus \delta_0) + y &= z \oplus \delta_1 \end{aligned}$$

Let k be the number of bits set in δ_0, not counting the highest-order bit. Then, the highest probability value for δ_1 is δ_0, and the probability that this will hold is 2^{-k}. This is true because addition and XOR are very closely related operations. The only difference between the two is the carry between bit positions. If flipping a given bit changes the carry into the next bit position, this alters the output XOR difference. This happens with probability $1/2$ per bit. The situation is more complex for multiple adjacent bits, but the general rule still holds: for every bit in the XOR difference not in the high-order bit position, the probability that the difference will pass through correctly is cut in half.

For the subkey generation, consider an XOR difference, δ_0, in A. This affects two subkey words:

$$\begin{aligned} K_{2i} &= A_i + B_i \\ K_{2i+1} &= \mathrm{ROL}(A_i + 2B_i, 9) \end{aligned}$$

where the additions are modulo 2^{32}. If we assume these XOR differences propagate independently in the two subkeys (which appears to be the case), we see that this leads to an XOR difference of δ_0 in the even subkey word with probability 2^{-k}, and the XOR difference $ROL(\delta_0, 9)$ in the odd subkey with the same probability. The most probable XOR difference in the round's subkey block thus occurs with probabiity 2^{-2k}. A desired XOR difference sequence for all 20 pairs of subkey words is thus quite difficult to get to work when $k \geq 3$, assuming the desired XOR difference sequence can be created in the A sequence at all.

When the XOR difference is in B, the result is slightly more complicated; the most probable XOR difference in a round's pair of subkey words may be either $2^{-(2k-1)}$ or 2^{-2k}, depending on whether or not the XOR difference in B covers the next-to-highest-order bit.

An XOR difference in A or B is easy to analyze in terms of additive differences modulo 2^{32}: an XOR difference with k active bits has 2^k equally likely additive differences. Note that if we have a additive difference in A, we get it in both subkey words, just rotated left nine bits in the odd subkey word. Thus, k-bit XOR differences lead to a given additive difference in a pair of subkey words with probability 2^{-k}. (The rotation does not really complicate things much for the attacker, who knows where the changed bits are.)

Note that when additive subkey differences modulo 2^{32} are used in an attack, they survive badly through the XOR with the plaintext block. We estimate that XOR differences are much more likely to be directly useful in mounting an attack.

3.9 Key-dependent Characteristics and Weak Keys

The concept of a key-dependent characteristic seems to have been introduced in [BB93] in their cryptanalysis of Lucifer, and also appears in [DGV94a] in an analysis of IDEA.[1] The idea is that certain iterative properties of the block cipher useful to an attacker become more effective against the cipher for a specific subset of keys.

A differential attack on Twofish may consider XOR-based differences, additive differences, or both. If an attacker sends XOR differences through the PHT and subkey addition steps, his differential characteristic probabilities will be dependent on the subkey values involved. In general, low-weight subkeys will give an

[1] See [Haw98] for further cryptanalysis of IDEA weak keys.

attacker some advantage, but this advantage is relatively small. (Zero bits in the subkeys improve the probabilities of cleanly getting XOR-based differential characteristics through the subkey addition.) Since there appears to be no special way to choose the key to make the subkey sequence especially low weight, we do not believe this kind of key-dependent differential characteristic will have any relevance in attacking Twofish.

A much more interesting issue in terms of key-dependent characteristics is whether the key-dependent S-boxes are ever generated with especially high probability differential or high bias linear characteristics. The statistical analysis presented earlier shows that the best linear and differential characteristics over all possible keys are still quite unlikely.

Note that the structure of both differential and linear attacks in Twofish is such that such attacks appear to generally require good characteristics through at least three of the four key-dependent S-boxes (if not all four), so a single high-probability differential or linear characteristic for one S-box will not create a weakness in the cipher as a whole. Our statistical testing has allowed us to estimate that few or no keys result in a single S-box with a differential characteristic of probability higher than 24/256 for any length key, and with a linear characteristic with bias higher than 108/256. These probabilities do not allow for practical differential or linear attacks. Further, for an attacker to mount a differential or linear attack, it appears to be necessary to get very high-probability differential or linear characteristics in all four S-boxes at once.

3.10 Related-key Cryptanalysis

Related-key cryptanalysis [Bih94, KSW96, KSW97] uses a cipher's key schedule to break plaintexts encrypted with related keys. In its most advanced form, differential related-key cryptanalysis, both plaintexts and keys with chosen differentials are used to recover the keys. This type of analysis has had considerable success against ciphers with simplistic key schedules—e.g., GOST and 3-Way [DGV94b]—and is a realistic attack in some circumstances. A conventional attack is usually judged in terms of the number of plaintexts or ciphertexts needed for the attack, and the level of access to the cipher needed to get those texts (e.g., known plaintext, chosen plaintext, adaptive chosen plaintext); in a related-key attack, we must add the requirement for encryptions to occur under two different, but related, keys.

3.11 Resistance to Related-key Slide Attacks

A "slide" attack occurs in an iterated cipher when the encryption of one block for rounds 1 through n is the same as the encryption of another block for rounds $s+1$ to $s+n$. An attacker can look at two encryptions, and can slide the rounds forward in one of them relative to another. S-1 [Anon95] can be broken with a slide attack [Wag95a]. Travois [Yuv97] has identical round functions, and can also be broken with a slide attack. Conventional slide attacks allow one to break the cipher with only known- or chosen-plaintext queries; however, as we shall see next, there is a generalization to related-key attacks as well.

Related-key slide attacks were first discovered by Biham in his attack on a DES variant [Bih94]. To mount a related-key slide attack on Twofish, an attacker must find a pair of keys M, M^* such that the key-dependent S-boxes in g are unchanged, but the subkey sequences slide down one round. This amounts to finding, for each of the eight byte-permutations used for subkey generation, a change in the keys such that:

$$s_i(j, M) = s_i(j + 2s, M^*)$$

for n values of j. In total, this requires $8n$ of these relations to hold.

Let us look in more detail for a fixed key M. Let $m \in \{5, \ldots, 8\}$ be the number of S-boxes used to compute the round keys that are affected by the difference between M and M^*. Observe that $m \geq 5$ due to the restriction that S cannot change and the properties of the RS matrix that at least 5 inputs must change to keep the output constant. There are at most $\binom{8}{m}2^{32m-128}$ possible choices of M^*. We have a total of nm 8-bit relations that need to be satisfied. The expected number of M^* that satisfy these relations is thus $\binom{8}{m} \cdot 2^{-8nm+32m-128}$. For $n \geq 4$ this is dominated by the case $m = 5$; we will ignore the other cases for now. So for each M we can expect about 2^{38-40n} keys M^* that support a slide attack for $n \geq 4$. This means that any specific key is unlikely to support a slide attack with $n \geq 4$. Over all possible key pairs, we expect

$2^{293-40n}$ pairs M, M^* for which a slide of $n \geq 4$ occurs. Thus, it is unlikely that a pair exists at all with $n \geq 8$.

3.11.1 Resistance to Related-key Differential Attacks

A related-key differential attack seeks to mount a differential attack on a block cipher through the key, as well as or instead of through the plaintext/ciphertext port. Against Twofish, such an attack must control the subkey difference sequence for at least the rounds in the middle. For the sake of simplifying discussions of the attack, let us consider an attacker who wants to put a chosen subkey difference into the middle twelve rounds' subkeys. That is, he wants to change M to M^*, and control $D[i, M, M*]$ for $i = 12..35$. At the same time, he needs to keep the g function, and thus the key S, from changing. All else being equal, the longer the key, the more freedom an attacker has to mount a related-key differential attack. We thus will assume the use of 256 bit keys for the remainder of this section. Note that a successful related key attack on 128 or 192 bit keys that gets only zero subkey differences in the rounds whose subkey differences it must control translates directly to an equivalent related key attack on 256 bit keys.

Consider the position of the attacker if he attempts a related-key differential attack with different S keys. This must result in different g outputs for all inputs, since we know that there are no pairs of S values that lead to identical S-boxes. Assuming the pair of S values does not lead to linearly-related S-boxes, it will not be possible to compensate for this change in S with changes in the subkeys in single rounds. The added difficulty is approximately that of adding 24 active S-boxes to the existing related-key attack. For this reason, we believe that any useful related-key attack will require a pair of keys that keeps S unchanged.

3.11.2 The Zero Difference Case

The simplest related-key attack to analyze is the one that keeps both S and also the middle twelve rounds' subkeys unchanged. It thus seeks to generate identical A and B sequences for twelve rounds, and thus to keep the individual byte sequences used to derive A and B identical.

The RS code used to derive S from M strictly limits the ways an attacker can change M without altering S. The attacker must try to keep the number of active subkey generating S-boxes as low as possible, since each active S-box is another constraint on his attack. The attacker can keep the number of active S-boxes down to five without altering S, and so this is what he should do. With only the key bytes affecting these five subkey generation S-boxes active, he can alter between one and four bytes in all five S-boxes; the nature of the RS matrix is that if he needs to alter four bytes in any one of these S-boxes, he must alter bytes in all five. In practice, in order to maximize his control over the byte sequences generated by these S-boxes, he must alter four bytes in all five active S-boxes.

To get zero subkey differences, the attacker must get zero differences in the byte sequences generated by all five active S-boxes. Consider a single such byte sequence: The attacker tries to find a pair of four-byte key inputs such that they lead to identical byte sequences in the middle twelve rounds, which means the middle twelve bytes. There are 2^{63} pairs of key inputs from which to choose, and about 2^{95} possible byte sequences available. If the byte sequences behave more-or-less like random functions of the key inputs, this implies that it is extremely unlikely that an attacker can find a pair of key inputs that will get identical byte sequences in these middle twelve rounds. We discuss this kind of analysis of byte sequences in section 3.1. From this analysis, we would not expect to see a pair of keys for even one S-box with more than eight successive bytes unchanged, and we would expect even eight successive bytes of unchanged byte sequence to require control of all four key bytes into the S-box. We would expect a specific pair of key bytes to be required to generate these similar byte sequences.

To extend this to five active S-boxes, we expect there to be, at best, a single pair of values for the twenty active key bytes that leave the middle eight subkeys unchanged.

3.11.3 Other Difference Sequences

An attacker who has control of the XOR difference sequences in A_i, B_i does not necessarily have great control over the XOR or modulo 2^{32} difference sequence that appears in the subkeys.

First, we must consider the context of a related-key differential attack. The attacker does not generally know all of the key bytes generating either A_i or B_i. Instead, he knows the XOR difference sequence in A_i and B_i.

Consider an A_i value with an XOR difference of δ. If the Hamming weight of δ is k, not including the high-order bit, then the best estimate for the XOR difference that ends up in the two subkey words for a given round generally has probability about 2^{-2k}. (Control of the A_i, B_i XOR difference sequence does not make controlling the subkey XOR differences substantially easier.)

Consider an A_i value with an XOR difference of δ. If the Hamming weight of δ is k, then the best estimate for the modulo 2^{32} difference of the two subkey words for a given round has probability about 2^{-k}.

This points out one of the difficulties in mounting any kind of successful related-key attack with nonzero A_i, B_i difference sequences. If an attacker can find a difference sequence for A_i, B_i that keeps $k = 3$, and needs to control the subkey differences for twelve rounds, he has a probability of about 2^{-72} of getting the most likely XOR subkey difference sequence, and about 2^{-36} of getting the most likely modulo 2^{32} difference sequence.

3.11.4 Probability of a Successful Attack With One Related-Key Query

We consider the use of the RS matrix in deriving S from M to be a powerful defense against related-key differential attacks, because it forces an attacker to keep at least five key generation S-boxes active. Our analysis suggests that any useful control of the subkey difference sequence requires that each active S-box in the attack have all four key bytes changed.

Further, our analysis suggests that, for nearly any useful difference sequence, each active S-box in the attack has a specific pair of defining key bytes it needs to work. At attacker specifying his key relation in terms of bytewise XOR has five pairs of sequences of four key bytes each, which he wants to get. This leaves him with a probability of a pair of keys with his desired relation actually leading to the desired attack of about 2^{-115}, which moves the attack totally outside the realm of practical attacks.

So long as an attacker is unable to improve this, either by finding a way to get useful difference sequences into the subkeys without having so many active key bytes, or by finding a way to mount related-key attacks with different S values for the different keys, we do not believe that any kind of related key differential attack is feasible.

Note the implication of this: Clever ways to control a couple extra rounds' subkey differences are not going to make the attacks feasible, unless they also allow the attacker to use far fewer active key bytes. For reference, note that with one altered key byte per active subkey generation S-box, the attacker ends up with a 2^{-39} probability that a pair of related keys will yield an attack; with two key bytes per active S-box, this increases to 2^{-78}; with three key bytes per active S-box, it increases to 2^{-117}. In practice, this means that any key relation requiring more than one byte of key changed per active S-box appears to be impractical.

3.12 Conclusions

Our analysis suggests that related-key attacks against the full Twofish are not workable. Note, however, that we have spent less time working on resistance to chosen key attacks, such as will be available to an attacker if Twofish is used in the straightforward way to define a hash function. For this reason, we recommend that more analysis be done before Twofish is used in the straightforward way as a hash function, and we note that it appears to be much more secure to use Twofish in this way with 128-bit keys than with 256-bit keys, despite the fact that this also slows the speed of a hash function down by a factor of two.

4 Open Questions

Several questions remain open regarding the strength of the Twofish key schedule. These include for following:

1. We have discussed differential related key attacks within a certain set of assumptions, including the assumption that the subkey generation mechanism has certain more-or-less random properties. We do

not have a stronger argument than our intuition and statistical tests that this is the case. A proof or stronger argument in either direction would be of great interest.

2. We have done some analysis (not reflected here for space reasons) on partial chosen key attacks on Twofish. Still remaining are issues raised by the desire to use Twofish in some Davies-Meyer hashing mode. What kind of collision resistance might we expect in this case.

3. We have assumed that the derivation of q_0 and q_1 introduces no weaknesses. Further analysis of this construction, as well as our larger S-box construction methods, would be of interest.

4. We have discussed related-key slide attacks. There are many other ways to reorder the round subkeys. Do any of these ways lead to attacks on the cipher?

References

[Anon95] Anonymous, "this looked like it might be interesting," sci.crypt Usenet posting, 9 Aug 1995.

[BB93] I. Ben-Aroya and E. Biham, "Differential Cryptanalysis of Lucifer," *Advances in Cryptology — CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 187–199.

[Bih94] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Journal of Cryptology*, v. 7, n. 4, 1994, pp. 229–246.

[Bih95] E. Biham, "On Matsui's Linear Cryptanalysis," *Advances in Cryptology — EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 398–412.

[DGV94a] J. Daemen, R. Govaerts, and J. Vandewalle, "Weak Keys for IDEA," *Advances in Cryptology — EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 159–167.

[DGV94b] J. Daemen, R. Govaerts, and J. Vandewalle, "A New Approach to Block Cipher Design," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 18–32.

[Haw98] P. Hawkes, "Differential-Linear Weak Key Classes of IDEA," *Advances in Cryptology — EUROCRYPT '98 Proceedings*, Springer-Verlag, 1998, pp. 112–126.

[KSW96] J. Kelsey, B. Schneier, and D. Wagner, "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 237–251.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner, "Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA," *Information and Communications Security, First International Conference Proceedings*, Springer-Verlag, 1997, pp. 203–207.

[NIST97a] National Institute of Standards and Technology, "Announcing Development of a Federal Information Standard for Advanced Encryption Standard," *Federal Register*, v. 62, n. 1, 2 Jan 1997, pp. 93–94.

[NIST97b] National Institute of Standards and Technology, "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)," *Federal Register*, v. 62, n. 117, 12 Sep 1997, pp. 48051–48058.

[Wag95a] D. Wagner, "Cryptanalysis of S-1," sci.crypt Usenet posting, 27 Aug 1995.

[Yuv97] G. Yuval, "Reinventing the Travois: Encrytion/MAC in 30 ROM Bytes," *Fast Software Encryption, 4th International Workshop Proceedings*, Springer-Verlag, 1997, pp. 205–209.