New Results on the Twofish Encryption Algorithm

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

February 1, 1999

Abstract

Twofish is a 128-bit block cipher submitted as an AES candidate. We provide several new results, continuing the research in [SKW+98a, SKW+99b]. 1) We provide new performance numbers, including: faster encryption and decryption on the Pentium Pro/II, faster key setup on the Pentium and Pentium Pro/II in assembly language, large-RAM implementations on 32-bit CPUs, Alpha performance, more implementation options on smart cards, and a low-gate-count hardware implementation. 2) In the initial Twofish paper [SKW+98a], we gave initial estimates of an upper bounds on the probability of a 12-round differential. These results used an imperfect model of Twofish. We present an improved model, and show that any 12-round differential characteristic has a probability of at most $2^{-102.8}$. 3) We show that each distinct Twofish key generates a unique sequence of subkeys K_i, and each round function F is unique for a distinct value of the S bits used to generate the S-boxes. Thus, no two distinct keys result in an identical sequence of round functions.

1 Improved Twofish Implementations

This section discusses improvements in the performance of Twofish [SKW+98a, WS98, SKW+99b].

1.1 Pentium and Pentium Pro/II Performance

Table 1 gives new assembly-language performance for Twofish on the Pentium and Pentium Pro/II. We have sped up the fastest assembly-language implementations.

1.2 Pentium Pro/II Large Memory Implementations

For machines with sufficient RAM and a good memory cache subsystem, large precomputed tables can be used to reduce the key setup time for Twofish even further. For example, in compiled, full, or partial keying modes, the first two levels of q_0 and q_1 lookups with one key byte can be precomputed for all four S-boxes, requiring 256 Kbytes of table (four tables of 64 Kbytes each). This approach saves roughly 2000 clocks per key setup on the Pentium Pro in assembly language; details are shown in Table 2. For instance, the compiled mode key setup for 128-bit keys on a Pentium Pro can be reduced from 8700 clocks to 6500 clocks. Unfortunately, the savings on Pentium and Pentium MMX CPUs seems to depend on the performance of the L2 cache subsystem (which is included in the Pentium Pro and thus is more predictable); the gain seems to range from 500 clocks down to nothing. Implementing this "big table" version in C also leads to savings of about 1000 clocks per key setup on the Pentium Pro, depending on the quality of the compiler; again, Pentium performance gains are minimal.

For the ultimate in key agility, a full 256 Mbytes of precomputed tables could comprise all four S-boxes for the final two stages of q_0, q_1, covering all 2^{16} key byte possibilities for the 128-bit key case, and including the MDS matrix multiply. With a good memory subsystem, such a version should cut

[*] 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`.

Processor	Lang	Keying Option	Code Size	Clocks to Key			Clocks to Encrypt		
				128	192	256	128	192	256
PPro/II	ASM	Comp.	9000	8600	11300	14100	258	258	258
PPro/II	ASM	Full	8500	7600	10400	13200	315	315	315
PPro/II	ASM	Part.	10700	4900	7600	10500	460	460	460
PPro/II	ASM	Min.	13600	2400	5300	8200	720	720	720
PPro/II	ASM	Zero	9100	1250	1600	2000	860	1130	1420
Pentium	ASM	Comp.	9100	12300	14600	17100	290	290	290
Pentium	ASM	Full	8200	11000	13500	16200	315	315	315
Pentium	ASM	Part.	10300	5500	7800	9800	430	430	430
Pentium	ASM	Min.	12600	3700	5900	7900	740	740	740
Pentium	ASM	Zero	8700	1800	2100	2600	1000	1300	1600

Table 1: Twofish ASM Performance with Different Key Lengths and Options

Processor	Lang	Keying Option	Code Size	Clocks to Key			Clocks to Encrypt		
				128	192	256	128	192	256
PPro/II	ASM	Comp.	271,200	6500	9200	11900	285	285	285
PPro/II	ASM	Full	270,600	5300	8000	11000	315	315	315
PPro/II	ASM	Part.	272,900	2600	5300	8200	460	460	460
PPro/II	MS C	Full	273,300	7300	11200	15700	600	600	600

Table 2: Twofish Performance with Large Fixed Tables

another 1000 clocks or so out of the above key-setup times. Clearly, this is a fairly expensive solution (at least with today's technology), but it illustrates the flexibility of Twofish very nicely.

1.3 Smart Card Performance

Table 3 gives additional performance data for the 6805 smart card CPU. (See also [SKW+99a].) The code size includes both encryption and decryption.[1] The block encryption and decryption times are almost identical. If only encryption is required, minor improvements in code size and speed can be obtained. The only key schedule precomputation time required in this implementation is the Reed-Solomon mapping used to generate the S-box key material S from the key M, which requires slightly over 1750 clocks per key. This setup time can be made considerably shorter at the cost of two additional 256-byte ROM tables. It should also be observed that the lack of a second index register on the 6805 has a significant impact on the code size and performance, so a different CPU with multiple index registers (e.g., 6502) might be a better fit for Twofish.

1.3.1 RAM Usage

For any encryption algorithm, memory usage can be divided into two parts: that required to hold the expanded key, and that required as working space to encrypt or decrypt text (including the text block). In applications where a smart card holds a single key for a long period of time, the key can be put into EEPROM or even ROM, greatly reducing RAM requirements. Most applications, however, require the smart card to encrypt using session keys, which change with each transaction. In these situations, the expanded key must be stored in RAM, along with working space to perform the encryption.

Twofish—the 128-bit key version—can be implemented in a smart card in 60 bytes of RAM. This includes the text block, key, and working space. If a slightly expanded key (16 bytes of the key plus another 8 bytes of the Reed-Solomon results (S)) can be stored in ROM or EEPROM, then Twofish can be implemented in only 36 bytes of RAM. In either case, there is zero key-setup time for the next encryption operation with the same key.[2]

Larger key sizes require more RAM to store the larger keys: 36 bytes for 192-bit keys and 48 bytes for

[1] For comparison purposes: DES on a 6805 takes about 1 Kbyte code, 23 bytes of RAM, and 20000 clock cycles per block.
[2] All of our implementations leave the key intact so that it can be used again.

RAM, ROM, or EEPROM for Key	Working RAM	Code and Table Size	Clocks per Block	Time per Block @ 4MHz
24	36	2200	26500	6.6 msec
24	36	2150	32900	8.2 msec
24	36	2000	35000	8.7 msec
24	36	1750	37100	9.3 msec
184	36	1900	15300	3.8 msec
184	36	1700	18100	4.5 msec
184	36	1450	19200	4.8 msec
1208	36	1300	12700	3.2 msec
1208	36	1100	15500	3.9 msec
1208	36	850	16600	4.2 msec
3256	36	1000	11900	3.0 msec

Table 3: Twofish Performance, with a 128-Bit Key, on a 6805 Smart Card

256-bit keys. If these applications can store key material in ROM or EEPROM, then these key lengths can be implemented on smart cards with only 36 bytes of RAM. All of this RAM can be reused for other purposes between block encryption operations.

For smart cards with larger memory to hold key-dependent data, encryption speed can increase considerably. This is because the round keys can be precomputed as part of the expanded key, requiring a total of 184 bytes of key memory. As shown in Table 3, this option nearly halves the encryption time. If the smart card has enough additional memory available to hold 1 Kbyte of precomputed S-box in either RAM, ROM, or EEPROM (for a total of 1208 bytes), performance improves further. Finally, as shown in the final row of Table 3, if the entire precomputed S-box plus MDS table can be held in memory (3256 bytes), the speed can again be increased slightly more. It should be noted that some of these "large RAM" implementations save 512 bytes of code space by assuming that certain tables are not required in ROM, with the entire precomputation being instead performed on the host that sets the key in the smart card. If the smart card has to perform its own key expansion the code size will increase. This increase has its own space/time tradeoff options.

This flexibility makes Twofish well-suited for both small and large smart card processors: Twofish works in the most RAM-poor environments, while at the same time it is able to take advantage of both moderate-RAM cards and large-RAM cards.

1.3.2 Encryption Speed and Key Agility

On a 6805 with only 60 bytes of RAM, Twofish encrypts at speeds of 26500 to 37100 clocks per block, depending on the amount of ROM available for the code. On a 4 MHz chip, this translates to 6.6 msec to 9.3 msec per encryption. In these implementations, the key-schedule precomputation time is minimal: slightly over 1750 clocks per key. This setup time could be cut considerably at the cost of two additional 512-byte ROM tables, which would be used during the key schedule.

If ROM is expensive, Twofish can be implemented in less space at slower speeds. The space–speed tradeoffs are of two types: unrolling loops and implementing various lookup tables. By far, the latter has the larger impact on size and speed. For example, Twofish's MDS matrix can be computed in three different ways:

- Full table lookups for the multiplications by `EF` and `5B`. This is the fastest, and requires 512 bytes of ROM for tables.
- Single table lookup for the multiplications by α^{-1}. This is slower, but only requires 256 bytes of ROM for the table.
- No tables, all multiplies done with shifts and XORs. This is the slowest, and the smallest.

Longer keys are slower, but only slightly so. For the small memory versions, Twofish's encryption time per block increases by less than 2600 clocks per block for 192-bit keys, and by about 5200 clocks per block for 256-bit keys. Similarly, the key schedule precomputation increases to 2550 clocks for 192-bit keys, and to 3400 clocks for 256-bit keys.

As shown in Table 3, in smart card CPUs with sufficient additional RAM storage to hold the entire set of subkeys, the throughput improves significantly, although the key setup time also increases. The time savings per block is over 11000 clocks, cutting the block encryption time down to about 15000 clocks; i.e., nearly doubling the encryption speed. The key-setup time increases by roughly the same number of clocks, thus making the key-setup time comparable to a single block encryption. This approach also cuts down the code size by a few hundred bytes. It should be noted further that, in fixed-key environments, the subkeys can be stored along with the key bytes in EEPROM, cutting the total RAM usage down to 36 bytes while maintaining the higher speed.

As another tradeoff, if another 1 Kbyte of RAM or EEPROM is available, all four 8-bit S-boxes can be precomputed. Clearly, this approach has relatively low key agility, but the time required to encrypt a block decreases by roughly 6000 clocks. When combined with precomputed subkeys as discussed in the previous paragraph, the block encryption time drops to about 12000 clocks, which translates to nearly three times the best speed for "low RAM" implementations. In most cases, this approach would be used only where the key is fixed, but it does allow for very high throughput. Similarly, if 3 Kbytes of RAM or EEPROM is available for tables, throughput can be further improved slightly.

The wide variety of possible speeds again illustrates Twofish's flexibility in these constrained environments. The algorithm does not have one speed; it has many speeds, depending on available resources.

1.3.3 Code Size

Twofish code is very compact: 1760 to 2200 bytes for minimal RAM footprint, depending on the implementation. The same code base can be used for both encryption and decryption. If only encryption is required, minor improvements in code size can be obtained (on the order of 150 bytes). The extra code required for larger keys is fairly negligible: less than 100 extra bytes for a 192-bit key, and less than 200 bytes for a 256-bit key.

Observe that it is possible to save further ROM space by computing q_0 and q_1 lookups using the underlying 4-bit construction. Such a scheme would replace 512 bytes of ROM table with 64 bytes of ROM and a small subroutine to compute the full 8-bit q_0 and q_1, saving perhaps 350 bytes of ROM; unfortunately, encryption speed would decrease by a factor of ten or more. Thus, this technique is only of interest in smart card applications for which ROM size is extremely critical but performance is not. Nonetheless, such an approach illustrates the implementation flexibility afforded by Twofish.

1.4 Performance on the Alpha

The 64-bit Alpha 21164 CPU can run up to 600 MHz using only a 0.35 micron CMOS process, compared to the 0.25 micron technology used in a Pentium II. The Alpha is widely regarded as the fastest general purpose processor available today. Its architecture and performance are expected to remain at the leading edge of technology for the foreseeable future. It has a 4-way superscalar architecture, which is fairly close in many respects to a Pentium II. Twofish should run on an Alpha in roughly the same number of clocks as on a Pentium Pro (i.e., 300).

1.5 Hardware Performance

Table 4 gives hardware size and speed estimates for the case of 128-bit keys. The first line is new. The first line of the table is a "byte serial" implementation. It uses one clock per S-box lookup, and four clocks per h function (including the MDS). We allow two clocks for the PHT and key addition. With four h functions per round, each round requires 18 clock cycles.

2 Empirical Verification of Twofish Key Uniqueness Properties

In the Twofish encryption key schedule for an N-bit key ($N = 128, 192, 256$), three different sets of key material are used, each consisting of $N/2$ bits, as described in [SKW+98a, SKW+98b]. Two of these sets, M_e and M_o in the notation of the paper, consist of the even and odd 32-bit words of key material, respectively, and are used to generate the round subkeys K_j. The third set, S, is derived by applying a Reed-Solomon parity code matrix (RS) to the entire key, and the bits of S are used to define the four key-dependent S-boxes s_i of Twofish. Any two of these three sets is sufficient to determine the entire key.

Given the structures used for generating subkeys and S-boxes, it appears intuitively that each distinct Twofish key results in a distinct cipher. In this paper we describe some empirical work performed to verify certain uniqueness properties of Twofish keys that strongly support our intuition [WW98]. In particular, we have proven that:

Gate Count	h Blocks	Clocks/ Block	Interleave Levels	Clock Speed	Throughput (Mbits/sec)	Startup Clocks
8000	0.25	324	1	80 MHz	32	20
14000	1	72	1	40 MHz	71	4
19000	1	32	1	40 MHz	160	40
23000	2	16	1	40 MHz	320	20
26000	2	32	2	80 MHz	640	20
28000	2	48	3	120 MHz	960	20
30000	2	64	4	150 MHz	1200	20
80000	2	16	1	80 MHz	640	300

Table 4: Hardware Tradeoffs (128-bit Key)

- No two distinct keys produce an identical sequence of subkeys K_j. In fact, we actually have proven an even stronger result; namely, each distinct M_e results in a unique sequence of A_i values, and each distinct M_o results in a unique sequence of B_i values.
- Each distinct value for S results in a unique round function F. This fact splits all keys into equivalence classes of size $2^{N/2}$ with respect to the round functions.

Given these two results, it is clear that no two keys produce the same sequence of round functions. Unfortunately, this result stops short of proving conclusively that each distinct Twofish key results in a distinct cipher, since it is not impossible theoretically that two ciphers with different round functions could produce an identical cipher. However, such an occurrence seems incredibly unlikely.

2.1 Distinct Subkey Sequences

Let us examine the A_i sequences used in generating subkeys. Similar arguments apply to the B_i sequence, and the same empirical verification has been performed for both sequences. Note that, since the operations used to compute K_{2i} and K_{2i+1} from A_i and B_i are reversible, proving that the A_i and B_i sequences are distinct is more than sufficient to prove that the K_j sequences are distinct.

Note that $A_i = h(2\rho i, M_e)$, where in this case the h function consists of applying the MDS matrix multiplication to the four values $(y0, y1, y2, y3)$ obtained by running the value i through $1+N/64$ levels of q_0 and q_1, XORing with bits from M_e. Since the MDS matrix is nonsingular, it is easily seen that, for example, a unique sequence for the y_0 values results in a unique sequence for the A_i values, with similar results for y_1, y_2, and y_3.

The sequence of y_0 values for 256-bit keys is

$$q_1[q_0[q_0[q_1[q_1[i] \oplus k_3] \oplus k_2] \oplus k_1] \oplus k_0]$$

where the k_j are different bytes from M_e. Smaller keys result in similar equations with fewer key bytes, so the analysis is almost identical but the empirical search time is much smaller. The question at hand is whether two distinct sets of four key bytes can result in an identical sequence of twenty y_0 values. In fact, we exclude the input and output whitening values to concentrate solely on the round subkeys, so only sixteen values are included in our test (one per round). Clearly, there are 2^{32} such sequences. However, this number can be reduced for the purposes of our search by noting that, since the "outer" mapping q_1 is bijective, it can be removed from the equation, leaving us with

$$q_0[q_0[q_1[q_1[i] \oplus k_3] \oplus k_2] \oplus k_1] \oplus k_0$$

Now the outer XOR term k_0 can be effectively removed by creating a related sequence with only fifteen values, where the y_0 values for $i = 5..19$ are XORed with the $i = 4$ value. The k_0 terms thus cancel out, so there are only 2^{24} equivalence classes of sequences, speeding up the search dramatically. If all these sequences (each with 15*8 = 120 bits) are unique, then the y_0 sequence is also guaranteed to be unique.

For 192-bit keys, the y_0 sequence is

$$q_1[q_0[q_0[q_1[i] \oplus k_2] \oplus k_1] \oplus k_0]$$

and for 128-bit keys, it is

$$q_1[q_0[q_0[i] \oplus k_1] \oplus k_0].$$

In a manner identical to that discussed for 256-bit keys, the outer q_1 permutation can be removed for these smaller keys, and the outer k_0 term can similarly be removed by creating the related XOR sequence. The number of the remaining sequences is

2^{16} and 2^8, respectively. All these sequences can then be compared across key sizes (with a total of $N_s = 2^{24} + 2^{16} + 2^8$ sequences) to verify uniqueness.

A computer search has been performed over y_0, y_1, y_2, and y_3, for all sequences across all key lengths. It turns out that only 64 bits were needed to distinguish the sequences. This fact is actually quite encouraging, since there are 120 bits in each sequence, giving some heuristic comfort that the sequences are in fact quite different. Each search requires slightly more than 128 Mbytes memory (i.e., $8N_s$ bytes) to hold all the sequences, which are then sorted and compared to adjacent values to guarantee uniqueness. The test was run on a Pentium computer with only 32 Mbytes of memory, so the search was actually performed in multiple passes, running through all the N_s values several times, on each pass selecting only those values falling into certain bins. For example, using approximately 8 Mbytes of memory, there are sixteen passes, with the m^{th} pass discarding all sequence values for which a fixed four-bit field of the sequence is not equal to m. In addition, the same test was run for the B_i using M_o, with similar results.

The definitive result from these tests is that there are no two distinct keys of any size for which the same sequence of A_i and B_i values is obtained. Thus, the round subkey sequence K_j is unique across keys.

2.2 Distinct Round Functions

The round function F takes a 64-bit input and produces a 64-bit output, and F is characterized by the four S-boxes s_i and the round keys K_{2r+8}, K_{2r+9}. The S-boxes depend on $N/2$ bits of key material S. Our test for distinctness of the round functions concentrates on only a single bit of the output, which will be sufficient to prove uniqueness.

In particular, consider the value

$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

where $T_0 = g(R_0)$ and $T_1 = g(\mathrm{ROL}(R_1, 8))$. The g function involves an MDS matrix multiply, which uses the XOR operation. Hence, it is difficult to analyze the full F_1 value, because of the interaction between operations of different algebraic groups (i.e., XOR and addition). However, if we examine only the least significant bit (lsb) of F_1, we can ignore carries, so the operation can be analyzed entirely using XOR. Note that this bit does not depend on R_1, due to the multiplication by two in the PHT. Further, the MDS matrix element that maps the S-box output is a simple linear transformation (i.e., multiplication by a GF(256) field element). Thus, for each S-box, the effect of the final fixed permutation (q_0 or q_1) of the S-box and the MDS multiply on the lsb of F_1 is a simple fixed mapping from eight bits to one bit.

Now, consider two keys with distinct values for S. Since the S values are different, at least one of the four S-boxes must have different key material under the two different keys. To prove uniqueness, we simply fix the inputs to the remaining three S-boxes, so that their XOR "contribution" to the lsb of F_1 is fixed. Up to a fixed XOR constant (based on K_{2r+9} and the other three S-boxes), we are then left with a simple function involving a single S-box that maps eight bits to one bit, with $N/8$ bits of key material used in the S-box. We can remove dependence on the fixed constant by constructing sequences consisting of the XOR of two bits in the S-box/MDS lsb output sequence, similar to the method discussed in the previous section. If this sequence of bits is unique across all $2^{N/8}$ possible key material values for the S-box, then the F function is unique with respect to that S-box. If all four S-boxes are unique in this way, then the F function is unique for each distinct value of S.

To remove the dependence on the constant bit, we performed our search on a modified sequence in which each bit was the XOR of two lsbs of the S-box/MDS output values. Since the s_i mapping has 256 inputs, this limits the sequence to only 255 values, which is still easily sufficient to distinguish between F functions. Let us first consider the 256-bit key case, which obviously involves the longest search. Note that, for example,

$$s_0(x) = q_1[q_0[q_0[q_1[q_1[x] \oplus k_3] \oplus k_2] \oplus k_1] \oplus k_0]$$

where the k_j bytes are a subset of the bytes in S. There are 2^{32} such functions, but, unfortunately, since we are dealing only with the lsb, there is no obvious method to cut down the search time by a factor of 256, as we were able to do in the previous section. Similar sequences were produced for the smaller key sizes, with all sequences for each S-box combined across key sizes in the test, for a total of $N_F = 2^{32} + 2^{24} + 2^{16}$ sequences per S-box.

To avoid a "birthday surprise" collision with N_F sequences, we require more than 64 bits of each sequence. Even with only 64 bits, however, a total of over 32 GB of memory would be required, a size far beyond the budget of this experiment. Thus, this test was also performed in multiple passes, with each pass generating all N_F sequences but discarding those values not falling into the selected bin for the particular pass, as in the previous section. It was found empirically that the performance time was roughly proportional to the number of passes; in other words, the sort/compare time for a given pass was considerably shorter than the $O(2^{32})$ time requires to generate the "filtered" list. For example, on a Pentium computer with slightly over 256MB of available RAM, 128 passes are required, which empirically were completed for a single S-box in slightly less than three days on a 200 MHz Pentium. Each sequence value in the list consisted of 64 bits, with additional sequence bits used to filter out values not to be used in a given pass. For a 128-pass test, this means that seven extra bits were used to help avoid a birthday surprise collision. The filtering code (in C) was carefully optimized so that most of the values to be filtered on each pass were quickly rejected. Since 127 of every 128 values were filtered out on each pass, this simple optimization sped up performance considerably, without requiring particularly fast generation of the 64-bit sequence values to be included.

The test was run on several Pentium computers with various amounts of RAM over a period of about ten days. The results showed that, for each of the four S-boxes, all N_F mappings have a unique lsb sequence. Thus, each F function is unique for each distinct value of S.

2.3 Conclusion and Future Work

We have empirically proven that every Twofish key leads to a unique set of round constants, and that every string S used to define the S-boxes results in a unique F-function. Although these properties seemed intuitively true, having an exhaustive proof is nonetheless reassuring. Given the structure of Twofish, it appears unlikely that there are any weak keys or significant problems with related keys.

3 Upper Bounds on Differential Characteristics in Twofish

The original Twofish report [SKW+98a] contained an initial analysis of the feasibility of a differential attack against the Twofish cipher. In this paper we will investigate differential attacks against Twofish further [Fer98b]. We assume familiarity with both Twofish and differential cryptanalysis.

The results of [SKW+98a] are not firm, as the model used to estimate the best differential is only an approximation of Twofish. We started a project to investigate differential attacks against Twofish further. This paper is a status report of our results to date. We expect to continue this work and achieve significant improvements over our current results.

The first choice we have to make in differential cryptanalysis is what type of differences to use. Twofish contains S-boxes, an MDS matrix multiply, addition modulo 2^{32}, XORs, and rotations. There are two types of differences that we think could be useful: an XOR difference, and a difference mod 2^{32}. When we use an XOR difference we have to use approximations for the S-boxes and the additions modulo 2^{32}; when we use a differences modulo 2^{32} we have to use approximations for the S-boxes, the MDS matrix multiply, the XORs, and the rotations.

The XOR and addition operations are fairly closely related, and either operation can be approximated with reasonable success in the group of the other operation. In the comparison we will ignore the S-boxes; we assume they are equally hard to approximate in each of the groups. An XOR differential has to approximate two addition operations in each round. A additive differential has to approximate the MDS matrix, a single XOR, and a rotation. We estimate that it is about as difficult to approximate an addition for an XOR differential as it is to approximate an XOR for an additive differential. The rotation seems to be somewhat easier to approximate for an additive differential than an XOR operation. So if we ignore the MDS matrix, it would seem that additive differentials are more attractive.

For our analysis, the MDS matrix multiply is best written as a linear function: each output bit is the XOR of several input bits. This is very easy from the point of view of an XOR difference; no approximation is necessary and any given input difference leads to precisely one output difference. For an additive difference this is much harder. There do not seem to be any good approximations of the MDS matrix for an additive difference. Therefore, we estimate that XOR-based differentials are much more effective than additive differentials. In the rest of this paper we will only look at XOR based differentials.

3.1 Notation

We use the definitions and symbols of the Twofish report [SKW+98a, SKW+99b]. Let $\mathcal{B}$ be the set of all possible byte values. Let G be the F-function without the key-dependent S-boxes. Thus G consists of two MDS matrix multiplies, the PHT, and the subkey addition.

3.2 Differentials of the S-boxes

In this section we look at differential characteristics of the S-boxes. Each S-box consists of a sequence of q-mappings and XORs with a key byte. For q_0 and q_1, the probability of each differential can easily be computed by trying all possible pairs of inputs.

We define $p_i(a,b)$ to be the probability that q_i has an output difference of b, given an input difference of a. In other words:

$$p_i(a,b) := \Pr_{x \in \mathcal{B}}[q_i(x \oplus a) = q_i(x) \oplus b] \qquad i = 0, 1$$

We now look at the first two stages of an S-box. This consists of a q-mapping, followed by an XOR with a key byte, followed by another q-mapping. As usual, we assume uniform random distributions of the input values and the key bytes. We define $p_{ij}(a,b)$ to be the probability that this construction gives an output difference b given an input difference a, where i is the number of the first q-mapping, and j is the number of the second q-mapping. It is easy to see that

$$p_{ij}(a,b) \quad = \quad \sum_{d \in B} p_i(a,d) p_j(d,b) \qquad (1)$$

for $i,j \in \{0,1\}$, and we can extend this definition to arbitrary long chains of q-mappings and key-byte XORs. In general it holds that

$$p_{ij\ldots m}(a,b) = \sum_{d \in B} p_i(a,d) p_{j\ldots m}(d,b)$$

for $i,j,\ldots,m \in \{0,1\}$. This allows us to compute the exact probabilities for each of the S-boxes in Twofish. Table 5 gives the probabilities of the best differential of each of the S-boxes for each of the key lengths. From this point of view the S-boxes are very good; there are no high-probability differentials. (Note that the average differential probability is $1/255 = 1.0039 \cdot 2^{-8}$ as we know that the S-boxes are permutations and thus the output differential 0 does not occur in non-trivial cases. The best differential probability must be at least as large as the average.)

Note that the numbers in this table hold only when the key bytes are chosen at random. If we try a differential many times, each time with random input and key byte values then we expect to get the numbers in the table. However, for any particular set of key bytes there are differentials with a much higher probability (as shown in [SKW+98a]). Our computations are no longer valid because, for any fixed key byte, the differential probabilities of p_i and p_j in equation 1 are no longer independent of each other.

Twofish uses the same S-boxes in each round. When analyzing a multi-round differential characteristic, the differential probabilities of each of the round functions are not independent, either. This makes the analysis of the probability of a differential characteristic more difficult.

3.3 Differentials of F

The function F takes a 64-bit input and produces a 64-bit output. Thus there are a total of about 2^{128} possible differentials. It is clearly not possible to compute or list all of them. To alleviate this problem we will group the differentials in sets, and for every set compute upper bounds on the probability of the differentials in that set.

We split the 2^{128} different differential patterns into a number of subsets. The input difference is classified by the set of input-bytes that are non-zero. There are 256 different classifications of input differences. The output difference too are classified by the set of output-bytes that are non-zero. We group differentials with the same input and output classification in the same set. There are therefore 2^{16} different sets of differentials, each containing between 1 and 255^{16} elements.

We will construct differentials of F in two steps. First we use a differential approximation of the S-boxes, and then we use an approximation of the differentials of G.

3.3.1 Differentials of G

The MDS matrix multiply is purely linear, and thus creates no problem for our differential. The PHT and key addition use addition modulo 2^{32} as basic operation. This makes the differentials non-trivial. A theoretical analysis of differential probabilities is difficult as the probabilities at the result are not independent of each other. We therefore chose to use numerical simulation to establish bounds on the differential probability.

We are trying to derive an upper bound on differential probabilities. Therefore, we are interested in finding good bounds for the most likely differentials of G. In [SKW+98a] it is shown that for any 128-bit key, the best differential probability of an S-box is 18/256. If we look only at the S-boxes, then the most likely differentials occur when there are a low number of active S-boxes. The most important task is thus to find good bounds on the differential characteristics of G for differentials with a low number of active S-boxes.

	128-bit key	192-bit key	256-bit key
Sbox 0	$1.0649 \cdot 2^{-8}$	$1.0084 \cdot 2^{-8}$	$1.0043 \cdot 2^{-8}$
Sbox 1	$1.0566 \cdot 2^{-8}$	$1.0087 \cdot 2^{-8}$	$1.0043 \cdot 2^{-8}$
Sbox 2	$1.0533 \cdot 2^{-8}$	$1.0097 \cdot 2^{-8}$	$1.0045 \cdot 2^{-8}$
Sbox 3	$1.0538 \cdot 2^{-8}$	$1.0088 \cdot 2^{-8}$	$1.0044 \cdot 2^{-8}$

Table 5: Best Differential Probabilities of the S-boxes

We performed numerical simulations of differentials of G. Given an input difference, we generated n random input pairs with that difference and applied the G function, using random keys. We collected the output differences and counted how often each of them appeared. Due to limited resources we could only do this analysis for moderately large n.

From this data, we would like to derive a bound on the differential probability. Let us assume a specific differential occurs k times out of n tries. It is obviously not a good idea to use k/n as a bound on the differential probability. Most possible differences occur 0 times, but we should not assume that they have a 0 probability. If we knew the distribution of the probability we could give some meaningful bound; for example, saying that the probability is less than x with a confidence level of 1%. However, in our case we do not know the distribution of the differential probabilities, and it would be dangerous to assume one. We can, however, reverse the process.

Let us assume that a specific differential has a probability p. If we try the input difference n times, we expect to find this differential around $p \cdot n$ times. The number of times this differential is actually observed is binomially distributed. Let X be a stochastic variable that represents the number of times the differential is observed. We have

$$\Pr(X = k) = \binom{n}{k}(1-p)^{n-k}p^k \qquad k = 0, \ldots, n$$

From this distribution we can derive a bound on the lower tail of the binomial distribution [Fer98a]:

$$\Pr(X \leq k) < \Pr(X = k)\frac{p(n-k+1)}{(p \cdot n - k) + p}$$

for $k \leq p \cdot n$. Given a probability p for the differential we can say that we have an unlikely event if the differential occurs k times and $\Pr(X \leq k) < \gamma$ where γ would be a small number. This is a normal test for statistical significance.

We use the following rule to derive a bound on a differential that occurs k out of n times. We use a probability p such that $\Pr(X \leq k) < \gamma$ for some global parameter γ (typical values for γ are 0.05 or 0.01). Of course, we try to choose p as low as possible given this condition. This will overestimate p for most differentials, but underestimate the actual p in a few cases.

We ran these simulations for all input differences with a low enough number of active S-boxes. For every differential that we tried we estimated the probability using this rule. For each set of differentials with the same input and output characterization we computed the maximum estimated probability. For each differential there is a small chance that we underestimates the probability. However, it is far less likely that we underestimated the maximum probability of a set of differentials. For our maximum to be too low we have to have underestimated the probability of the most likely differential. This by itself is rather unlikely. Not only that, we cannot significantly overestimate the probability of any differential with a probability close to the most likely differential. We therefore feel confident that these approximations are reasonable, and that they most likely will result in our overestimating the actual differential probability quite significantly.

For differentials with too many active S-boxes (for which we did not run the simulations) we simply use an upper bound of 1 on the probability of a differential of G.

To improve efficiency we generate our input differences using a straightforward structure. This improves our performance and allows us to increase the number of samples that we make. However, the differentials that we try are no longer independent of each other. We have observed that the use of structures significantly increases the peaks in the bounds. The smaller the structures that we use, the lower the maximum probability bound tends to be. Therefore, we try to reduce the use of structures. We hope our next software version will allow us to eliminate structures altogether.

Apart from these numerical results, we know that certain differential patterns cannot occur. For example, if the input difference is restricted to the first input word of the G, then the output difference must have active bits in both output words. Similarly, if the input difference is restricted to the second input word of G, then the output difference must have active bits in both halves, except when the output of the MDS matrix has a difference of 0x80000000. In this special case, we know that all four S-boxes in this half must be active (otherwise, more than 1 byte in the output of the MDS matrix must change). Our software generates all these impossible differential patterns and sets the differential probabilities of the associated sets to zero.

3.3.2 Differentials of F

Given the results from the last section, we can now create a table of upper bounds on the differential probabilities of differentials of F. For each set of differentials we know how many active S-boxes there are. Let σ be the maximum probability of a differential of an S-box. We can now bound the probability of each set of differentials by multiplying the bounds that we found in the previous section on the set by the proper power of σ.

The value σ can be set in various ways. We know that most S-boxes have a best differential probability of 12/256. For the time being we will use this value for σ. Other values, especially larger ones, will be discussed later.

3.4 Differentials of the Round Function

Once we have derived bounds on the differentials of F we can do the same for the round function. The differential pattern at the start of the round is characterized by 16 bits, each bit indicating whether the differential pattern in the corresponding byte is nonzero. Given the characterization of the differential pattern at the input of the round, we know exactly which S-boxes are active. We can generate a list of all suitable differential patterns of F with their associated probability bound. Each of these differentials is combined with the other half of the input differential using the rotate and XOR operations. Each choice of F differential set leads to several possible output differential patterns as the rotate and XORs can lead to different output characterizations, depending on the exact differential.

For example, let us look at a differential pattern of 0110 in a 32-bit word. This pattern indicates that only the middle two bytes of the 4-byte word contain active differential bits. After a left rotation, the possible output differential patterns are: 0100, 0110, 1010, 1100, and 1110. XORing two differential patterns can similarly lead to a list of possible results. If we XOR two words, one with a differential pattern of 0101 and one with a differential pattern of 0011, then the possible result patterns are 0110 and 0111.

For each input differential pattern, we can go through all possible F differential patterns, and generate all possible output patterns that can arise. For each possible output pattern, we keep track of the largest upper bound that we generate this way. This produces an upper bound for each of the 2^{32} possible input/output differential patterns of the round function.

3.5 Multi-round Patterns

The simplest way of generating multi-round patterns would be to use the list of 2^{32} possible round patterns and a standard search algorithm. We use an algorithm that is somewhat more efficient than that. There are 2^{16} possible differential patterns after r rounds, as each of the 16 data bytes can have a zero or nonzero difference. For each of the 2^{16} possible patterns we store an upper bound on the probability of any characteristic that has this difference pattern after r rounds. Furthermore, we store the list of differential patterns of F, and a precomputed table of how the rotates and XORs can propagate patterns. For each difference pattern after $r+1$ rounds, we use this data to compute an upper bound on the probability of a differential characteristic that has this pattern after $r+1$ rounds.

Given the output pattern of the round in question, we know the first half of the input pattern. This leaves us with 256 possible differential input patterns, and 256 possible differential patterns of F. Each of the 2^{16} possible combinations is tried to see whether it can yield the required output differential pattern. The process can be speeded up by traversing either the F output patterns or the input patterns in decreasing order of probability and using some simple cut-off logic.

3.6 Results

The results depend on the parameters used to estimate the differential probabilities of G, and the values of γ and σ.

Our current results use $n = 2^{11}$ tries for all differentials with 1 active S-box, and $n = 2^8$ tries for all differentials with 2 active S-boxes. The structure size is 8 and 16, respectively. We use $\gamma = 0.05$, and $\sigma = 12/256$. The full Twofish cipher has 16 rounds. We assume that an adversary can somehow bypass the first round, and can mount a 3R-attack. We thus look at the best 12-round differential characteristic.

With these parameters we found an upper bound on a 12-round differential characteristic of $2^{-102.8}$. This puts a differential attack against Twofish well outside the practical realm.

This upper bound is pessimistic in the following areas:

- The best differential pattern used three active S-boxes in four of the 12 rounds. The probability of passing a differential with three active S-boxes through G is currently taken to be 1. This is clearly overly optimistic, especially since the differential pattern used has both a low input and a low output weight. We believe that extending our simulations to all differentials with three active S-boxes will yield a significant further reduction in probability.

- Many of the rounds in the best differential pattern use fancy transformations of the difference pattern by the rotations. This is to be expected of our algorithm, but any non-trivial transformation poses serious restrictions on the actual difference patterns of that word. This makes it much less likely that our upper bound can actually be approached by an actual differential.

- Our estimates are based on the maximum probability of groups of differentials. It is not clear at all that there exists a differential that has a probability that even approaches our upper bound.

3.7 Other Problems for the Attacker

To create an attack, the attacker has to choose a specific differential characteristic. That characteristic uses certain specific differences of each of the S-boxes. To get anywhere near the bound all of these differences need to have a probability close to our σ. We chose σ equal to the probability of the best differential of most S-boxes. However, a specific differential will not have the same probability under all keys. If the S-box keys are not known, the attacker has two options. First, he can guess the S-box key bits, and construct a differential characteristic based on that assumption. To achieve good differential probabilities in enough S-boxes, he will have to guess the keys of at least two S-boxes (between 32 and 64 bits, depending on the key size). Alternatively he can try to find a differential that works for all keys. As we saw in section 3.2 this leads to very low differential probabilities.

3.8 Best S-box Differential

We use $\sigma = 12/256$. While we know that there are keys for which the best S-box differential has probability $18/256$ for a 128-bit key (and even higher for larger key sizes), those higher probabilities only occur for a small subset of the keys. We need to address the question of how much we are willing to pay in the size of the keyspace the attack is effective on to get a higher probability. If we have an attack that works for 2^{-28} of the key space, how much more efficient should the attack be before it is a better choice for the attacker?

The most natural way to decide this is to look at the expected work for the attacker before recovering a single key. We assume that there are enough keys to attack, and optimize the attacker's strategy to find any one key with the least amount of work. This is a reasonable way of looking at the attacker's problem. After all, we know there is an attack that is effective on a subset of 2^{-64} of the keys with 2^{64} work: a simple exhaustive search of the subset of that size will do. With such an brute-force attack on a subset of the keys, the expected amount of work before a key is found remains the same.

Let us now look at our Twofish differentials. Suppose we want to use a differential of S-box 0 with probability $14/256$; this is possible for about 1 in 8 of all possible keys. We have restricted ourselves to 1/8th of the set of keys, so the workload of our attack should be reduced by at least a factor of 8 for this to be worthwhile. We ran our search for the best differential characteristic pattern again where S-box 0 had a best differential probability of $14/256$. The resulting differential probability was 4.6 times higher than the result with $\sigma = 12/256$. Is this worth it?

Let us assume a differential has a probability that depends on the key. We have a list of (p_i, k_i) where the probability of the differential is at most p_i for a fraction k_i of all keys. The expected workload of the attacker to get a single right pair is $1/p_i$ for a fraction k_i of the key space, and thus $\sigma k_i/p_i$ when taken over all keys. The workload is at least $\max k_i/p_i$. This corresponds to the workload of an attack with a differential with probability $\min p_i/k_i$. In our situation the minimum occurs when we use the S-box approximation with probability $12/256$. (Using the figures from Table 3 in [SKW+98a], we find that p_i/k_i reaches its minimum at $p_i = 12/256$.) As we currently ignore the $1/k_i$ term, the actual "effective" probability of a real differential is lower than the bound that we have derived.

We conclude that using a higher probability than $12/256$ for an S-box approximation is not worth the loss in key space on which the approximation holds. Thus the bounds we presented earlier hold, and are in fact pessimistic.

3.9 Other Variants

As an experiment we ran the same analysis for Twofish with the 1-bit rotations removed. This makes our approximations match the behavior of the differential much better. Our results give an upper bound on the probability of a 12-round differential characteristic of $2^{-104.1}$.

This bound is not much better than we have for full Twofish. However, the sequence of differential patterns that achieves this bound uses far more approximations of F that have three active S-boxes. In all cases it uses differential characteristics of G that have three active input bytes and only a few active output bytes. In practice, such differentials of G will have a far lower probability than the upper bound of 1 that we currently use. Therefore, we expect that our bound can be improved to beyond 2^{-128}.

We have no reason to believe that the 1-bit rotations make Twofish stronger against a differential attack. They were conceived to break up the byte-level structure, but they do not require a separate approximation or increase the avalanche effect of the cipher. We think it is unlikely that the full Twofish has a differential characteristic that is significantly more likely than the version without rotations.

3.10 Further Work

We will continue our analysis work to improve the bound and our understanding of the intricacies of Twofish. We have several areas that we plan to improve.

3.10.1 Improved G Differential Estimates

An obvious way to improve our overall bound is to improve our bounds on the differentials of G. We hope to be able to do this in the near future. A larger sample-size will improve the accuracy of our estimates. Extending our computations to differentials of G with three active S-boxes should give a great improvement.

3.10.2 More Accurate Patterns

Our current pattern-representation is somewhat coarse. We group differentials only by which bytes contain active bits. Apart from the first and final round, all internal differential patterns in our best result have at most four active bytes (out of 16). A more fine-grained grouping of the differentials could lead to a better upper bound.

For example, there are only a few G differences with one active input byte that have a relatively high probability. Instead of grouping these into the sets, we could treat them separately. This would ensure that our algorithm doesn't magically transform the output of the high-probability difference pattern to one with fewer active bytes by the rotation. The characterization could be extended with special cases for differences that have only a few active nibbles. We expect that this will result in more S-boxes being needed for a full differential characteristic, and thus a lower bound.

3.10.3 Improved Treatment of S-box Differentials

There is still room to improve our approximations of the S-boxes. We can, for example, compute the best differential approximation for each of the output differences separately. This can then be combined with the analysis of G to get a better bound on differentials of F.

The data on the best S-box differentials in [SKW+98a] is merged for all the S-boxes. We plan to test the differentials again and collect information for each S-box separately.

We will also improve our handling of the variation of differential probability over the keyspace. This will also result in a better bound.

3.10.4 Additive Differentials

We would like to take a closer look at additive differentials modulo 2^{32}. Although we do not expect these to be more useful, it would be nice to derive some bound in that case too.

3.11 Conclusion

For practical purposes, Twofish is immune to differential cryptanalysis. We have shown that any 12-round differential has a probability of at most $2^{-102.8}$. This bound is far from hard, and we expect that any real differential has a much smaller probability.

The Twofish structure is not easy to analyze. The mixing of various operations makes it hard to give a clean analysis and forces us to use approximation techniques. Some aspects, such as the rotates, make the analysis considerably harder and force us to use less accurate approximations, while there is no a priori reason to assume that the rotations would have any significant influence on the differential probabilities.

One can argue whether a cipher with a structure that is easier to analyze would be preferable. On the one hand, a structure that allows easier analysis makes it easier to rule out certain attacks. On the other hand, the very structure that makes it easy to analyze might be used in a future attack. Although differential attacks were obviously considered during the design, Twofish was not specifically strengthened against differential attacks, or designed to allow a simple upper bound on differential probabilities to be derived. This is a result of the design philosophy of Twofish. It was not optimized specifically against known attacks; it is a conservative design that tries to resist both known and unknown attacks.

References

[Fer98a] N. Ferguson, "Bounds on the tail of binomial distributions", research notes, 1998.

[Fer98b] N. Ferguson, "Upper Bounds on Differential Characteristics in Twofish," Twofish Technical Report #1, Counterpane Systems, `http://www.counterpane.com/twofish-differential.html`, Aug 1998.

[SKW+98a] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher," NIST AES Proposal, Jun 98. `http://www.counterpane.com/twofish.html`

[SKW+98b] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "On the Twofish Key Schedule," *Proceedings of the 1998 SAC Conference*, Springer-Verlag, 1998, to appear.

[SKW+99a] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish on Smart Cards," *Proceedings of CARDIS 98*, Springer-Verlag, to appear.

[SKW+99b] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *The Twofish Encryption Algorithm: A 128-bit Block Cipher*, John Wiley & Sons, 1999.

[WS98] D. Whiting and B. Schneier, "Improved Twofish Implementations," Twofish Technical Report #3, Counterpane Systems, `http://www.counterpane.com/twofish-speed.html`, 2 Dec 1998.

[WW98] D. Whiting and D. Wagner, "Empirical Verification of Twofish Key Uniqueness Properties," Twofish Technical Report #2, Counterpane Systems, `http://www.counterpane.com/twofish-keys.html`, 22 Sep 1998.