Twofish Technical Report #3

Improved Twofish Implementations

Doug Whiting* Bruce Schneier†

December 2, 1998

Abstract

We provide new performance numbers for Twofish. Improvements include: 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.

Keywords: Twofish, cryptography, AES.

Current web site: `http://www.counterpane.com/twofish.html`

1 Pentium and Pentium Pro/II Performance

Table 1 gives new assembly-language performance for Twofish on the Pentium and Pentium Pro/II. The improvement in this table is that we have sped up the key setup times.

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 another 1000 clocks or so out of the above key-setup times. Clearly, this is a fairly expensive solution (at

*Hi/fn, Inc., 5973 Avenida Encinas, Suite 110, Carlsbad, CA 92008, USA; dwhiting@hifn.com.
†Counterpane Systems, 101 E Minnehaha Parkway, Minneapolis, MN, 55419, USA; schneier@counterpane.com.

Processor	Lang	Keying Option	Code Size	Clocks to Key 128	192	256	Clocks to Encrypt 128	192	256
PPro/II	ASM	Comp.	9000	8700	11500	14200	285	285	285
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 128	192	256	Clocks to Encrypt 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

least with today's technology), but it illustrates the flexibility of Twofish very nicely.

3 Smart Card Performance

Table 3 gives additional performance data for the 6805 smart-card CPU. 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.

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 1K 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 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 kilobyte 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.

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 1K bytes 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, 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 3K bytes 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.

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.

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).

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.

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)