From: "Brian Gladman" <brian.gladman@btinternet.com>
To: <AESround2@nist.gov>
Subject: AES Comment
Date: Thu, 16 Mar 2000 14:42:19 -0000
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.2919.6700
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2919.6700

A paper is attached as an AES comment.

Brian Gladman

Some Informal Reflections on Rijndael and Twofish

By Dr. B. R. Gladman, Worcester, UK, March 2000

Introduction

During the implementation of Rijndael [1] and Twofish [2] with pre-computed key schedules the author has been struck by similarities between the round functions of these algorithms when implemented in this way. This paper considers these functions informally from a security perspective and concludes by raising a question about the relative security strengths of the two algorithms.

In reading this paper it is important to understand that Twofish provides considerable implementation flexibility that allows optimisation depending on the expected size of the encryption task. Because similarities between the Rijndael and Twofish round functions are evident with pre-computed key schedules it is inevitable that this paper will focus on this mode of operation. Nevertheless, it should be remembered that Twofish can be implemented in other ways that will often reduce some of the costs discussed here.

The Round Functions

In what follows the sixteen 8-bit bytes of the cipher state block will be designated `s00` to `s15`; each 32-bit word in this cipher state will be designated by the range of bytes it contains using, for example, the notation: `s08:11`. Left and right circular rotations will be designated by `<<<` and `>>>` respectively. A superscript ‘`'`’ after a variable will denote an updated state variable that is not used on the right hand side of subsequent expressions during the calculation of the round function (i.e. the value is stored at the end of the round).

The operation that is common to both Rijndael and Twofish when implemented for bulk encryption takes 4 of the 16 bytes of the cipher state and mixes these to form a 32-bit output using the operation:

```
mix(p, q, r, s) = T0[p] ^ T1[q] ^ T2[r] ^ T3[s]
```

where `p`, `q`, `r` and `s` are the chosen bytes and `T0`, `T1`, `T2` and `T3` are four tables that each contain 256 32-bit words. This function updates one of the four 32-bit words in the cipher state using four selected state bytes as input. By applying this function four times the four 32-bit words in the cipher state can hence be updated, this being one round for Rijndael and one cycle – two Feistel rounds – for Twofish. The two algorithms use different byte selections and different updating orders; they also employ different key mixing approaches as shown in the following C pseudo-code:

```
rijndael_round(s00:15, *ksch)
{
    s00:03'  = mix(s00, s05, s10, s15)
    s04:07'  = mix(s04, s09, s14, s03)
    s00:03′ ^= *ksch++
    s04:07′ ^= *ksch++
    s08:11'  = mix(s08, s13, s02, s07)
    s12:15'  = mix(s12, s01, s06, s11)
    s08:11′ ^= *ksch++
    s12:15′ ^= *ksch++
}
```

```
twofish_cycle(s00:15, *ksch)
{
    x      = mix(s00, s01, s02, s03)
    y      = mix(s07, s04, s05, s06)
    s08:11 = [s08:11 ^ (x + y + *ksch++)] >>> 1
    s12:15 = [s12:15 <<< 1] ^ (x + 2 * y + *ksch++)
    x      = mix(s08, s09, s10, s11)
    y      = mix(s15, s12, s13, s14)
    s00:03 = [s00:03 ^ (x + y + *ksch++)] >>> 1
    s04:07 = [s04:07 <<< 1] ^ (x + 2 * y + *ksch++)
}
```

The Twofish cycle is applied 8 times whereas the Rijndael round is applied 10, 12 or 14 times for 128, 192 or 256 bit keys respectively. Each of these functions takes a 16 byte input cipher state and 16 bytes of the key schedule (produced from the key) and mixes these to produce the 16 bytes of the output state. The tables for the mix() function in Rijndael are fixed but those in Twofish are key dependent. However, in what follows these two functions will be compared for fixed tables since the security properties provided by key expansion will be considered separately later.

For Rijndael each byte in the output state is dependent on 4 bytes of the input state and on 1 byte from the key schedule.

The situation for Twofish is more complex. Bytes in the upper half of the state block are mixed with values that depend on all the bytes in the lower half. This means that they depend on 9 of the 16 input bytes. Because of the additions and the rotations in the PHT stage there are some additional dependencies but these will be ignored. Output state bytes in the lower half of the state depend on all 16 bytes of the input state. On average, therefore, each output byte is dependent on at least 12 of the 16 input bytes. The situation for key schedule bytes is also more complex because the keys are added rather than XOR’d so that carries between bytes create additional dependencies. But ignoring these nuances once again, the bytes in the lower half of the state block depend on 1 byte in the key schedule while bytes in the upper half depend on 9 bytes of the schedule.

This is summarised in the following table.

Average number of input bytes on which bytes in the output cipher state depend	Input cipher state bytes (average)	Key schedule bytes (average)
Rijndael	4	1
Twofish	12	5

In addition the use of key dependent tables in the Twofish round function provide even greater diffusion of the key into the output state.

This analysis suggests that the Twofish cycle is more effective than the Rijndael round in diffusing the input state and key schedule bytes across the bytes of the output state. We would thus expect Rijndael to need more rounds than Twofish and this is, indeed, reflected in the two designs. But the ratio of rounds – 10:8 – seems somewhat less than might be expected from this diffusion comparison alone.

A naïve calculation of overall diffusion as the product of 'single round diffusion count' multiplied by the number of rounds gives a ratio of 12:5 (Twofish/Rijndael) but this is really meaningless since composition of diffusion between rounds is certainly far from linear. In consequence determining whether the increased number of Rijndael rounds is sufficient to offset what appears to be a somewhat stronger Twofish cycle is a matter for cryptanalysts.

In performance terms, inspection of these functions suggests that the processing cost of a Rijndael round and a Twofish cycle will be quite similar but with a disadvantage for Twofish as a result of its more complex key mixing and PHT operations. This is confirmed by practical results for the two algorithms:

Cycles/block (bulk encryption, 128-bit keys)	Twofish	Rijndael	Cost (Twofish Cycle/Rijndael Round)
Pentium II, C (Gladman)	373	363	1.30
Pentium II, Assembler (Lipmaa)	286	237	1.50
Merced (Whiting)	232	170	1.70
McKinley (Whiting)	181	126	1.80

The values given in the last three lines are those provided by Helger Lipmaa at [3]. The round cost ratio takes account of the different numbers of rounds or cycles involved. Hence while the Twofish cycle is more effective in diffusion terms, it is also about 50% more expensive in processing cost. Taking account of the different number of rounds and cycles this suggests that for software based bulk encryption Rijndael will typically be around 20% faster than Twofish for 128-bit keys, about the same for 192-bit keys and 20% slower for keys of 256 bits.

Key Expansion

Although the byte mixing functions used by Rijndael and Twofish are the same, they operate on different bytes and, more importantly, they use tables that are generated in entirely different ways. In Rijndael these tables are fixed in design whereas in Twofish they are key dependent. Moreover the generation of the two key schedules are also very different with Rijndael employing a very simple key expansion process which contrasts sharply with the much more complex mechanism used by Twofish. The latter also uses key dependent tables for the mix() function and hence provides for a stronger influence of the key on the encryption process.

The end result is a radical difference in the cost of the two key schedules (as used here the term 'key schedule' for Twofish includes the generation of its key dependent S-Boxes):

Key Expansion Cost for Bulk Encryption	Encrypt (Forward)	Decrypt (Inverse)
Rijndael (Pentium II, C, 128-bit keys)	213	1338
Twofish (Pentium II, C, 128-bit keys)	8530	8590

The disparity in the cost of pre-computed key schedules – 40:1 for encryption and 6.4:1 for decryption – is striking and suggests that the two design teams must have taken very different views of the security properties required of key expansion. This disparity is particularly large for encryption because Rijndael has been specifically designed to make this operation very efficient in the knowledge that a number of applications and block chaining modes do not require the inverse cipher. Since the round functions for the two algorithms appear to be much closer to each other in security performance, it seems reasonable to conclude that either the Rijndael key expansion is under-engineered or that of Twofish is over-engineered (or both).

This is an important issue since the cost difference is significant. In view of the similarities evident in the two round/cycle functions it seems rather unlikely that their security 'strengths' will differ by more than 2:1 and we might hence expect that key schedules designed to match the security of these functions would have costs that differ by much less than 6.4:1 or even 40:1. Of course, for bulk encryption the impact of key expansion cost declines as the number of blocks increases but if the simpler and faster key expansion used in Rijndael is sufficient to achieve the required level of security, this algorithm will have a considerable performance advantage in tasks involving encryption for small and medium numbers of blocks.

But this paper has so far considered implementations tuned for bulk encryption whereas Twofish can be implemented in different ways to give a much lower encryption cost for small numbers of blocks. Figures given at http://www.counterpane.com/twofish_key_setup.html suggest a Twofish single block encryption cost of 2110 cycles for an assembler implementation, made up of 1250 cycles for key expansion and 860 cycles for encryption. Equivalent results provided by Helger Lipmaa at [3] suggest that a reasonable estimate for Rijndael in similar circumstances would be 427 cycles consisting of 190 cycles for key expansion and 237 cycles for encryption (190 cycles for key expansion is an assembler estimate based on a cost of 213 cycles in C). While a lot smaller than 40:1, this shows that Rijndael is still a factor of five faster than Twofish for encryption, which remains a large performance advantage (the advantage for decryption is much less – about 1.4:1).

With an encryption cost penalty of somewhere between 5:1 and 40:1, it would be hard not to conclude that Twofish key expansion should be more robust in security terms than that of Rijndael. But since this can have a big impact on performance, it is important to understand if this is necessary in order to meet AES security requirements. If Rijndael key expansion is ‘good enough’ to meet AES needs, does this mean that Twofish is over-engineered? On the other hand, if Twofish has set the right level of security performance, is Rijndael key expansion too simple and hence vulnerable to attack? These are important questions in the final months prior to the selection of the AES finalist algorithm(s).

Conclusion

The Twofish cycle and Rijndael round functions possess similarities that suggest that their respective security strengths and encryption/decryption speeds will be significantly, but not radically, different when considered with fixed tables. The Twofish round function appears to be stronger than that of Rijndael but this is offset by the additional number of rounds that Rijndael employs.

But the costs of key expansion in software for pre-computed key schedules differ by a factor of 40:1 for encryption (6.4:1 for decryption) and such a large disparity raises the question of whether Rijndael is under-engineered and/or Twofish is over-engineered in this respect. For single block encryption the cost ratio is less extreme but still gives Rijndael a 5:1 (1.4:1 for decryption) advantage over Twofish on the Pentium platform.

The aim of this paper is not to show preference for either algorithm but rather to promote discussion of what might be a significant trade-off between security and performance. If this is suggestive of over-engineering in Twofish then this just illustrates a conservative algorithm design. If, however, this is indicative of under-engineering in Rijndael, this would be a serious concern since algorithm security is paramount.

The author is not a cryptanalyst but would like to hear more from the cryptanalytic community on the mismatch in the complexity of the Rijndael and Twofish key expansion processes in the light of round functions which, when considered with fixed tables, appear to be closer in terms of their security performance.

Acknowledgements

The author would like to thank Bruce Schneier, David Wagner, Niels Ferguson, John Kelsey, Doug Whiting, Vincent Rijmen and Joan Daemen for their most helpful comments on earlier versions of this paper.

References

[1] AES Proposal: Rijndael, by Joan Daemen and Vincent Rijmen, available at: http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

[2] Twofish: A 128-Bit Block Cipher, by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson, available at: http://www.counterpane.com/twofish.html.

[3] AES Algorithm Performance comparison compiled by Helger Lipmaa at: http://home.cyber.ee/helger/aes/table.html.