AES Key Agility Issues in High-Speed IPsec Implementations

Doug Whiting* Bruce Schneier† Steve Bellovin‡

May 15, 2000

Abstract

Some high-speed IPsec hardware systems need to support many thousands of security associations. The cost of switching among different encryption keys can dramatically affect throughput, particularly for the very common case of small packets. Three of the AES finalists (Rijndael, Serpent, and Twofish) provide very high key agility, as is required for such applications. The other two candidates (MARS, RC6) exhibit low key agility and may not be appropriate for use in such equipment.

Keywords: cryptography, block cipher, AES, key agility, IPsec, performance.

1 Introduction

The ultimate winner of the AES "contest" will be used in many different applications, with widely varying cost and performance constraints. IPsec will almost certainly adopt AES, dramatically speeding up software implementations over the currently predominant algorithm (triple-DES).

In high-speed routers and other networking boxes that apply IPsec [KA98c, KA98a, MG98a, MG98b, MD98, KA98b, Pip98, MSST98, HC98, GK98, TDG98, PA98] to aggregated traffic, hardware encryption is almost always necessary to meet performance objectives. For some applications, such equipment may have to handle thousands or tens of thousands of security associations. In such environments, the cost of switching between encryption keys for different security associations may be a significant issue, particularly for the important case of small packets (e.g., 64 bytes).

This paper attempts to quantify the key agility of each AES candidate algorithm and assess the performance impact of this metric on high-performance IPsec equipment.

2 Why IPsec?

Although encryption can be used in many places in the Internet, the focus in this paper is on IPsec because of its unique characteristics. First, as noted above, it is often employed in contexts where hardware implementations are useful. Second, it is in some sense an extreme case in that it requires more key agility than most other Internet applications.

Most Internet encryption is used above TCP. SSL, widely used in Web browsers, is one example. Typically, the host needs to complete TCP processing before any cryptographic chips can be invoked. (There have been many attempts to build front-end processors for TCP; i.e., RFCs 928 and 929. In most cases, these have failed, because the complexity of the host-to-front-end protocol has been comparable to that of TCP/IP itself.) However, requiring a separate pass over the data by the cryptographic processor would place an extra load on main memory. Furthermore, given that SSL is typically used only for electronic commerce, the actual amount of data transmitted in

*Hi/fn, Inc., 5973 Avenida Encinas Suite 110, Carlsbad, CA 92008, USA; dwhiting@hifn.com
†Counterpane Internet Security, Inc., 101 E Minnehaha Parkway, Minneapolis, MN 55419, USA; schneier@counterpane.com
‡AT&T Labs Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07974, USA; smb@research.att.com

any given session is relatively small. Thus, there is not much data over which to amortize the cost of the hardware setup operations.

Encrypted e-mail is even less suited to hardware processing. The application itself must extract the ciphertext before any possible decryption can take place, or package the resulting ciphertext after encryption. Use of outboard hardware would thus require an extra kernel interaction (and possibly data copy) to do any cryptographic operations in hardware. Furthermore, e-mail should generally be encrypted on the end user's system, rather than on a gateway, suggesting little need for key agility.

By contrast, IPsec was designed for hardware implementation. There are already commercially available Ethernet controllers that incorporate IPsec. There are also gateway products today that run at tens of megabits per second, while handling thousands of security associations, and new products under design will support much higher throughput and SA count (see next section).

The gateways are also a test of key agility. In a typical remote user scenario, all traffic from every remote user, to any destination whatsoever, will pass through the gateway. The gateway thus sees all traffic patterns, and all mixes of packet sizes. Furthermore, gateways of this type are inherently multiuser. Depending on the number of simultaneous users and the link speed, rapid key switching would be needed.

Although they will not be analyzed in detail here, other gateway-like scenarios will present similar challenges. One is ATM-based virtual private networks, which are typically very high speed. A second is the interface between voice-over-IP products and the current switched phone network; draft standards from at least one group require that this traffic be encrypted over the Internet. Both of these situations may be even more demanding, because they use small packets exclusively.

3 The Environment

In IPsec systems, the first step in a typical processing path for a packet is for a CPU or a packet processor to examine the packet header and make a policy decision as to which Security Association (SA), if any, the packet "belongs." Once the SA determination is made, the packet is passed to the crypto subsystem (if required), where the IPsec header manipulation and crypto transforms are applied. The SA index is used by the crypto subsystem to load from memory any SA-specific context, including sequence number, Security Parameters Index (SPI), transform type, and key material.

Due to the large number of SAs supported in some equipment, this context cannot reasonably fit on a single chip, so it typically resides in a large memory (e.g., 128 Mbytes) external to the crypto chip(s). Such chips are available from several companies, such as Hi/fn, Rainbow Technologies, Broadcom, and IRE. These chips can perform the entire header manipulation and crypto processing tasks, including triple-DES and HMAC-SHA/MD5, all in a single pass over the packet, at full-duplex speeds of hundreds of megabits per second. In the coming few years, these speeds will approach and exceed the gigabit/second range.

Empirically, to a first order approximation, all IP packets are either large (1500 bytes) or small (less than 128 bytes). Various empirical studies have been done, showing distribution histograms with large spikes at both ends of the spectrum, with relatively small background counts in the packet sizes [Abb99], as Table 1 illustrates. This distribution makes sense because of the Ethernet packet size limit (about 1500 bytes) on one hand, and because of the many small packets used for sending keystrokes and protocol acknowledgements on the other. In general, the majority of packets are small, but the majority of bytes are in large packets. In any case, the small-packet throughput of an IPsec system is normally used as the key comparative performance benchmark by all equipment vendors.

4 The Issue

Since small-packet performance is so critical, if an IPsec algorithm for encryption or MAC requires a large context, and if one cannot load that context quickly enough, the performance of the entire system may suffer. For example, given a 64-byte packet, the typical context size for an ESP tunnel transform using triple-DES and HMAC-SHA is on the order of 100 bytes. This includes 24 bytes for triple-DES key, 8 bytes for IV (if saved from packet to packet within the SA), 40 bytes for the precomputed HMAC inner and outer loop values, and 20–30 other bytes, including protocol, SPI, sequence number, tunnel header information (e.g., IP addresses), and other per-session configuration information (e.g., tunnel vs. transport mode, byte counts for session lifetime, etc.). In such systems, these parameters may all vary from SA to SA and must thus be loaded as

Packet Size	Probability Distribution	Bandwidth Distribution
32	0.000	0.0000
64	0.489	0.0602
96	0.055	0.0102
128	0.012	0.0030
160	0.006	0.0018
192	0.006	0.0022
224	0.006	0.0026
256	0.005	0.0025
288	0.016	0.0089
320	0.009	0.0055
352	0.008	0.0054
384	0.006	0.0044
416	0.005	0.0040
448	0.004	0.0034
480	0.004	0.0037
512	0.003	0.0030
544	0.010	0.0105
576	0.068	0.0754
1024	0.023	0.0453
1536	0.253	0.7480

Table 1: Size Distribution of Internet Packets

each packet is processed. Thus, the time for loading the session context is comparable to the time required for reading and writing the packet (e.g., 64 bytes in, about 100 bytes out). Some of the context (e.g., sequence number) needs to be updated when the packet processing has been completed, but the byte counts are roughly balanced, to the first order. The packet may or may not go through the same memory as the session contexts, depending on the architecture and performance requirements.

Ideally, these numbers should not change dramatically for ESP using the AES algorithm, compared to triple-DES. For example, using triple-DES, the subkeys from round to round are easily computed on the fly directly from the 24-byte key material itself. There is no need to perform any precomputation on the key material on a per-packet basis, so encryption or decryption can begin immediately upon loading of the key. Assuming a 128-bit AES key size, the total context for AES encryption should ideally remain roughly comparable to that of triple-DES, at 16-32 bytes (16 for key, possibly 16 for IV). However, if the key schedule cannot be easily computed on the fly, the transfer count could change significantly, affecting performance.

There are several solutions to such a problem, each with its own cost in hardware and/or complexity. The simplest solution is to always precompute the entire key schedule when the SA is established and load it from memory each time the session context is accessed. Because memory bandwidth is almost always a performance constraint in such systems, depending on the size of the subkey material, this approach probably requires widening the memory path or adding a separate memory just for subkeys to maintain a given performance level. The expanded subkey size is usually at least an order of magnitude larger than the raw key itself (e.g., 160+ bytes versus 16 bytes). Since most of the memory in such crypto subsystems is dedicated to session contexts, any such multiplier on the session context size impacts memory cost almost linearly, compared to algorithms that can perform on-the-fly subkey generation. Thus, such solutions are expensive, not only in terms of the cost of the memory chips, but also in pin count (which translates directly to dollars) and in board area.

It is also possible to have an off-chip memory with an on-chip cache holding the expanded key schedule for recently used SAs. There are several problems with this approach, in addition to the intrinsic design complexity and unpredictable performance of caches. First, the size of the on-board cache is related to the number of SAs supported, and thus would necessarily be quite large to achieve reasonable cache "hit" rates for systems with tens of thousands of SAs. In such instances, it is likely that the on-chip area dedicated to such a cache would be as large as or larger than the rest of the crypto logic, thus significantly increasing the system cost. Also, the size of the cache is dictated by the key agility to first order. In other words, for an algorithm with high key agility, the key could be stored unexpanded in the cache (e.g., 16 bytes), while for an algorithm with low key agility the expanded key (e.g., 150+ bytes) must be stored. Suffice it to say that relative factors of ten in on-chip memory size can rarely be ignored!

Another solution is to pipeline the subkey computation on the chip, loading the 128-bit key and expanding it in on-chip memory for the "next" packet. Assuming that the key schedule computation could be completed in less than the processing time for a small packet (e.g., 64 bytes, or four blocks), this approach should not affect throughput, although it may add considerably to the gate count of the chip. Unfortunately, as will be shown below, this assumption is not true for two of the AES candidate ciphers. Equally problematic is the fact that such a change modifies the underlying chip architecture by adding an entirely new stage in the pipeline that is not required for current algorithms such as triple-DES. Such a change can be extremely costly in terms of design complexity and time-to-market.

Ideally, adding AES to an existing IPsec chip should involve a relatively simple change, not radical architectural modifications. To be fair, changing from triple-DES to AES does require a change due solely to the different block size, and this is not insignificant. However, the block size change only affects the bundling of data within the crypto processing pipeline, not the overall relative flow of session context and packets.

5 The Ciphers

Three of the AES finalist candidates, Rijndael [DR98], Serpent [ABK98], and Twofish [SKW+98, SKW+99a], can easily perform on-the-fly subkey generation in hardware. The key schedules of Rijndael and Serpent are stateful in progressing from round to round. Thus, in order to perform decryption on the fly, both require a one-time computation of the key schedule to the end of the cipher so that the key schedule can be run backwards from the end point, using only 128 bits of key material state. However, this obstacle is easily overcome by performing the one-time precomputation of the key schedule when the SA is initially established, and placing the "decrypt" key starting point value in the session context. Depending on whether encryption or decryption is needed, only the forward or backward subkey (16 bytes for a 128-bit key) starting point would be loaded from memory. The Twofish key schedule is entirely stateless from round to round, so no such initialization is required. All three of these ciphers can be dropped into an existing architecture without modification.

However, MARS [BCD+98] and RC6 [RRS+98] do not have on-the-fly key schedules. The subkey material for the two algorithms totals 160 and 176 bytes, respectively. Thus, solely from a memory transfer count, a simple approach would nearly double the memory bandwidth requirements, compared to triple-DES or Rijndael, Serpent, or Twofish. On the other hand, if a pipelined approach is used, it appears that cost goes up significantly and performance may still be affected, as shown below.

For MARS, if a separate key schedule pipeline stage is used, that pipeline stage must have its own S-box. Since each S-box requires a large (16 Kbit) ROM, this change is a non-trivial addition to the logic cost of MARS, which has already been shown to be by far the largest among the AES finalists [WBRF00, IKM00]. Also, the time required for precomputing the MARS subkeys is fairly large. Assuming that a single core round of MARS can be run in one clock and that two rounds of the mixing layer can be run in one clock (to go faster requires a second S-box, further increasing the gate count), then a single MARS block can be encrypted or decrypted in 24 clocks. By contrast, the key schedule of MARS (assuming a single S-box) requires *at least* 270 clocks, assuming that two linear transformation steps can be performed per clock. Even if the linear transformation were free, the key schedule requires 240 clocks, or the time required to encrypt 10 blocks (i.e., a 160-byte packet!). In other words, a pipelined approach is actually not sufficient to hide the key schedule time of MARS. Because the key schedule is inherently sequential, even if dramatically more hardware were thrown at the problem (e.g., two S-boxes for the key scheduler), the key schedule speed could not be significantly increased relative to the speed of the encryption itself.

For RC6, if a separate key schedule pipeline stage is used, the additional logic required is non-trivial (e.g., another 32-bit variable rotator or two), increasing cost. Also, the number of clocks required to run the full key schedule is about 180, assuming that the key schedule logic can perform two variable rotates in a single clock. Assuming that a round of RC6 runs in one clock, the block encrypt time is 20 clocks, so the key schedule requires 9 block timings, corresponding to a 144-byte packet.

For RC6 and MARS, it appears that the only viable approach to maintaining small-packet IPsec performance with high key agility is to use more and/or wider memory, and even that may still hurt performance somewhat. Further, using these algorithms with pre-expanded keys more than doubles the session context size (100 bytes to 250+ bytes) compared to other algorithms (triple-DES, Rijndael, Serpent, Twofish). In many applications, all off-chip key material must be stored in encrypted form, and the performance overhead of decrypting the expanded key may be significant. Since the session contexts generally occupy the lion's share of the memory, it is also fair to say that the cost disadvantage is very significant for these two algorithms. For large packets, the performance issue is much less noticeable, because the cost of loading the key schedule from memory is amortized over many more blocks of data. However, the memory cost problem does not disappear.

6 Conclusion

RC6 and MARS do not have on-the-fly key schedules, raising cost and lowering performance, calling into question their suitability for certain high-performance IPsec hardware environments. If all the AES candidates had such limitations, it could perhaps be argued that the associated additional cost and complexity involved in using these algorithms in such systems is justifiable. However, Rijndael, Serpent, and Twofish all have on-the-fly key schedules that work very well in such environments, fitting easily into existing architectures without significantly affecting cost or performance.

7 Acknowledgments

The authors would like to thank Steve Kent, who contributed to this paper. Also, the authors would like to thanks the Extended Twofish Team for their comments and support.

References

[Abb99] S. Abbott, "Architectures for Supporting Hardware Cryptographic Engines," 1999 RSA Conference, Jan 1999.

[ABK98] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," NIST AES Proposal, Jun 1998.

[BCD+98] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic, "MARS — A Candidate Cipher for AES," NIST AES Proposal, Jun 1998.

[DR98] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, Jun 1998.

[GK98] R. Glenn and S. Kent, "The NULL Encryption Algorithm and its Use with IPsec," RFC 2410, Nov 1998.

[HC98] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," RFC 2409, Nov 1998.

[IKM00] T. Ichikawa, T. Kasuya, M. Matsui. "Hardware Evaluation of the AES Finalists," AES3 Conference, April 2000.

[KA98a] S. Kent and R. Atkinson, "IP Authentication Header, RFC 2402, Nov 1998.

[KA98b] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 2406, Nov 1998.

[KA98c] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Nov 1998.

[MD98] C. Madson and N. Doraswamy, "The ESP DES-CBC Cipher Algorithm with Explicit IV," RFC 2405, Nov 1998.

[MG98a] C. Madson and R. Glenn, "The Use of HMAC-MD5-96 Within ESP and AH," RFC 2403, Nov 1998.

[MG98b] C. Madson and R. Glenn, "The Use of HMAC-SHA-1-96 Within ESP and AH," RFC 2404, Nov 1998.

[MSST98] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)," RFC 2408, Nov 1998.

[PA98] R. Pereira and R. Adams, "The ESP CBC-mode Cipher Algorithms," RFC 2452, Nov 1998.

[Pip98] D. Piper, "The Internet IP Security Domain of Interpretation for ISAKMP," RFC 2407, Nov 1998.

[RRS+98] R. Rivest, M. Robshaw, R. Sidney, and Y.L. Yin, "The RC6 Block Cipher," NIST AES Proposal, Jun 1998.

[SKW+98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher," NIST AES Proposal, Jun 1998.

[SKW+99a] 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.

[TDG98] R. Thayer, N. Doraswamy, and R. Glenn, "IP Security Document Roadmap," RFC 2411, Nov 1998.

[WBRF00] B. Weeks, M. Bean, T. Rozylowicz, C. Ficke (NSA), "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms," AES3 Conference, April 2000.