A Performance Comparison of the Five AES Finalists

Bruce Schneier
Counterpane Internet Security, Inc.
3031 Tisch Way, Suite 100PE
San Jose, CA 95128

Doug Whiting
Hi/fn, Inc.
5973 Avenida Encinas, Suite 110
Carlsbad, CA 92008

15 March 2000

1 Introduction

In 1997, NIST announced a program to develop and choose an Advanced Encryption Standard (AES) to replace the aging Data Encryption Standard (DES) [NIS97a,NIST97b]. They solicited algorithms from the cryptographic community, with the intent of choosing a single standard. Fifteen algorithms were submitted to NIST in 1998, and NIST chose five finalists in 1999 [NBD+99]. NIST's plans to choose one (or more than one) algorithm to become the standard in 2000.

NIST's three selection criteria are security, performance, and flexibility. This paper is primarily about the latter two criteria, in light of information regarding the first criterion. In Section 3, we discuss the software performance of the five AES finalists on a variety of CPUs. Detailed information can be found in [SKW+99b], and is not repeated here.

Specifically, we compare the performance of the five AES finalists on a variety of common software platforms: current 32-bit CPUs (both large microprocessors and smaller, smart card and embedded microprocessors) and high-end 64-bit CPUs. Our intent is to show roughly how the algorithms' speeds compare across a variety of CPUs.

There has been considerable discussion in the community about taking the submissions and modifying the number of rounds to increase security. The effect of this is that successful cryptanalytic attacks against a submission do not necessarily knock it out of the running, but instead decrease its performance (by forcing the number of rounds to increase). In [Bih98,Bih99], the author suggests comparing "minimal secure variants" as a way to normalize algorithms.

In Section 4, we give the maximum rounds cryptanalyzed for each of the algorithms, and re-examine all the performance numbers for these variants. We believe that this provides a fairer way of comparing the different algorithms, and the design decisions the different teams made. We then compare the algorithms again, using the minimal secure variants as a way to more fairly align the security of the five algorithms.

2 Performance as a Function of Key Length

The speed of MARS, RC6, and Serpent are independent of key length. That is, the time required to set up a key and encrypt a block of text is the same, regardless of whether the key is 128, 192, or 256 bits long. Twofish encrypts and decrypts at a speed independent of key length, but takes longer to set up longer keys.[1] Rijndael both encrypts and decrypts more slowly for longer keys, and takes longer to set up longer keys. The results are summarized in Table 1.

Algorithm Name	Key Setup	Encryption
MARS [BCD+98]	constant	constant
RC6 [RRS+98]	constant	constant
Rijndael [DR98]	increasing	128: 10 rounds 192: 20% slower 256: 40% slower
Serpent [ABK98]	constant	constant
Twofish [SKW+98, SKW+99a]	increasing	constant

Table 1: Speed of AES Candidates for Different Key Lengths

[1] Twofish is more flexible than this chart implies. There are implementations where the encryption speed is different for different key lengths, but they are only suitable for encrypting small text blocks [SKW+98a,WS98,SKW+99a].

In our first performance comparison [SKW+99b], we concentrated on key setup and encryption for 128-bit keys. In this paper, we look at all three key lengths.

3 Software Performance

Efficiency on 32-bit CPUs is one of NIST's stated performance criteria. Unfortunately, modern architecture of modern CPUs is so complicated that there is no single performance number that can be used as a basis for comparison.

Today, most high-end microprocessors use 32-bit architectures. These microprocessors range from the Intel Pentium family to embedded CPUs like the ARM family to 32-bit smart card chips. Since all of the AES finalists use 32-bit word sizes, it is not surprising that they are most efficient on these architectures.

The performance space covered by 32-bit CPUs is actually very large, from a 386 or a 68000 or an embedded RISC CPU on the low end, through the various CPUs in the Pentium family. Each CPU chip has its own set of strengths and weaknesses, including clock frequency, cache size and architecture, instruction pipelining (scalar vs. superscalar), etc. Each feature can have significant impact on the speed with which an algorithm can be executed. Even within the same family, major differences in the relative efficiency of certain instruction types can be manifest. As a particular case in point, the Pentium CPU is relatively quite slow at performing multiplies and variable rotates compared to the Pentium Pro/II/III CPUs. This is of interest because two of the AES candidates rely heavily on such operations. It would seem encouraging that the trend is toward better performance on these opcodes in later generations, but in fact future generation processors (such as the IA-64 family) seem to revert back to relatively slow performance for these instructions. The point here is that relative performance of the AES algorithms may vary dramatically on various CPUs. For example, RC6 is the fastest algorithm on the Pentium II/III family by a small margin, but is less half the speed of the fastest candidates on the Pentium and the PA-RISC (and, reportedly, on the Intel Itanium, whose architecture is heavily influenced by PA-RISC). It is not clear exactly what conclusion to draw from such anomalies, particularly since all the candidates are quite fast compared to Triple-DES. In any case, great care should be taken not to assign rankings based solely on a single generation of a single CPU family, which may have a somewhat idiosyncratic performance.

Over the next several years, 64-bit CPUs will become the default microprocessor in desktop computers. These include the Intel Itanium (formerly the Merced) and McKinley, and the DEC Alpha. These microprocessors will first be designed into high-end servers and workstations, and will eventually migrate to commodity desktop computers.

3.1 Language Choice

The language of implementation should not matter in theory; i.e., the relative performance of the algorithms in various languages should not depend on the language. In practice, this is at least partially true. For example, some algorithms are very "compiler friendly" and perform quite nicely in high-level languages. To some extent, this is dependent on the language itself, such as the lack of a rotation primitive in C, but to a larger extent it depends on the compiler. The ratio between speeds of Serpent in C and assembler seems to be close to 1:1 on most platforms, while for other algorithms it is between 1.5:1 and 2:1. Thus, Serpent looks relatively much better in C than in assembler. The Java performance seems frankly so dependent on the compiler that the mere phrase "Java performance" seems like an oxymoron. Ultimately, the choice of language may shift the relative performance numbers across algorithms by up to a 2:1 ratio. As long as the metric of choice is somewhat independent to such a range, language really doesn't matter. However, if the ultimate 10% or 50% performance really matters, the algorithm of choice will always be written in assembly—encryption is a discrete and well-defined chunk of code with a single entry point that needs to run fast, a perfect example of something that should be coded in assembly—so those are the best numbers to use in attempting to get precise comparisons.

3.2 Comparing Software Performance

The tables in this section compare the five AES finalists on different CPUs. We either used the data in the candidate's AES submission documentation or, where such data was unavailable or unreliable, calculated our own or used the data from other people's implementations [Gla98,Alm99,BGG+99,Gra00,Lip00]. Where we were unsure if a particular optimization technique would work, we gave the algorithm the benefit of the doubt. When there were multiple sources of data, we took the fastest. We believe this is a fair comparison of the algorithms' speeds.

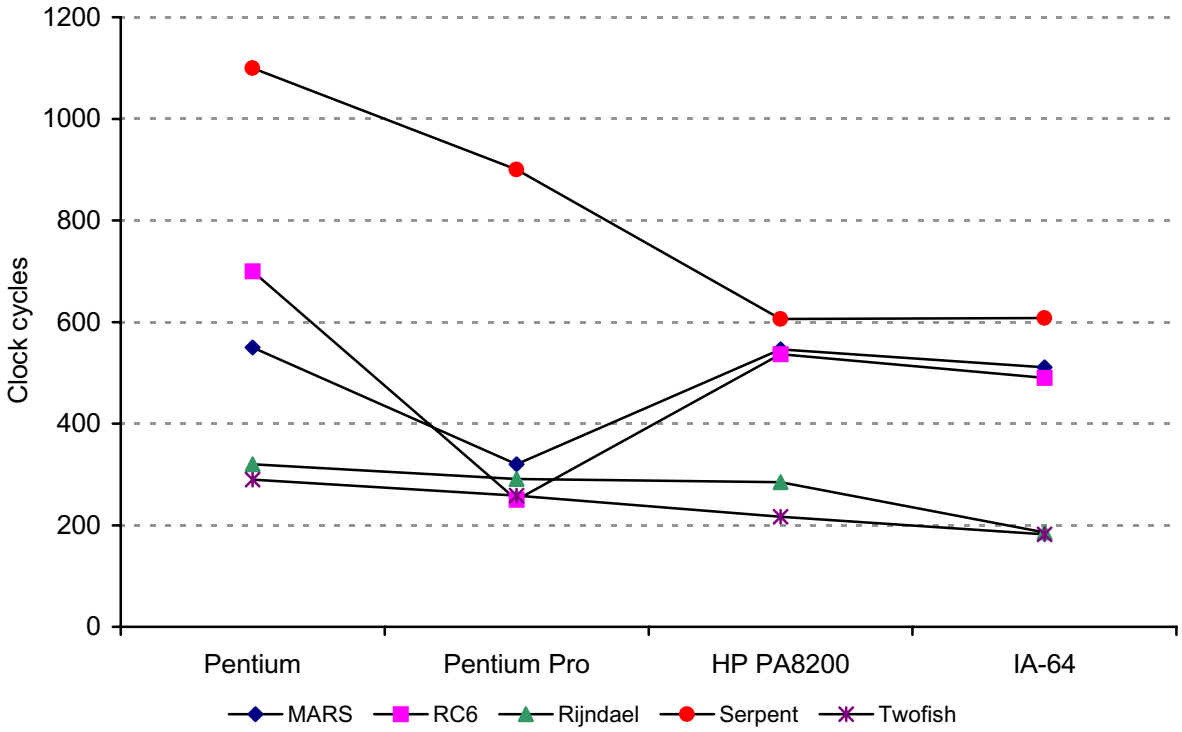

Figure 1: Encryption Speeds for 128-bit Keys in Assembly

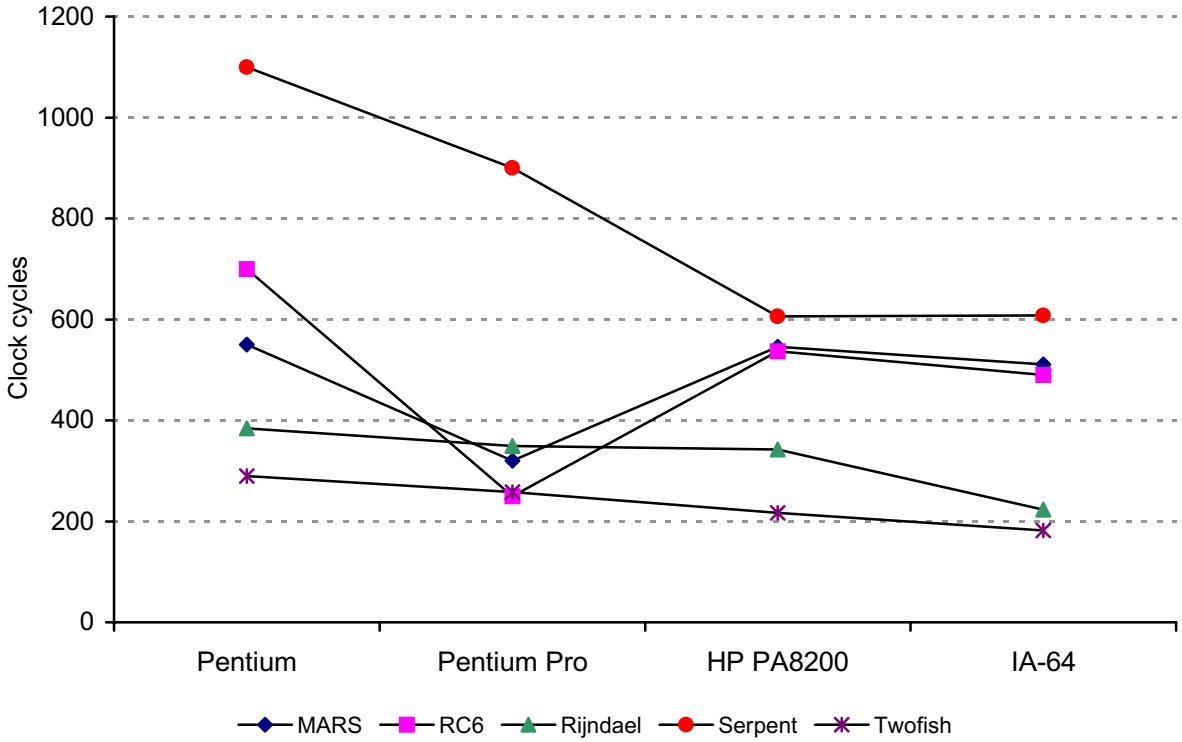

Figure 2: Encryption Speeds for 192-bit Keys in Assembly

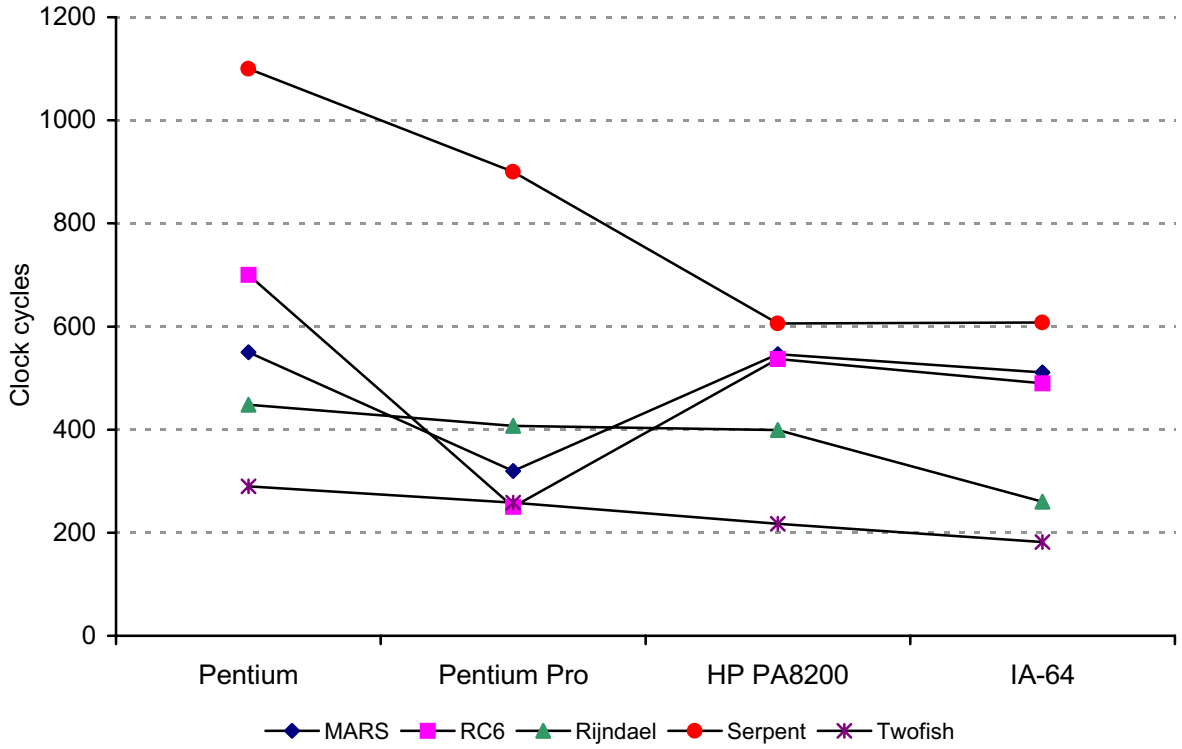

Figure 3: Encryption Speeds for 256-bit Keys in Assembly

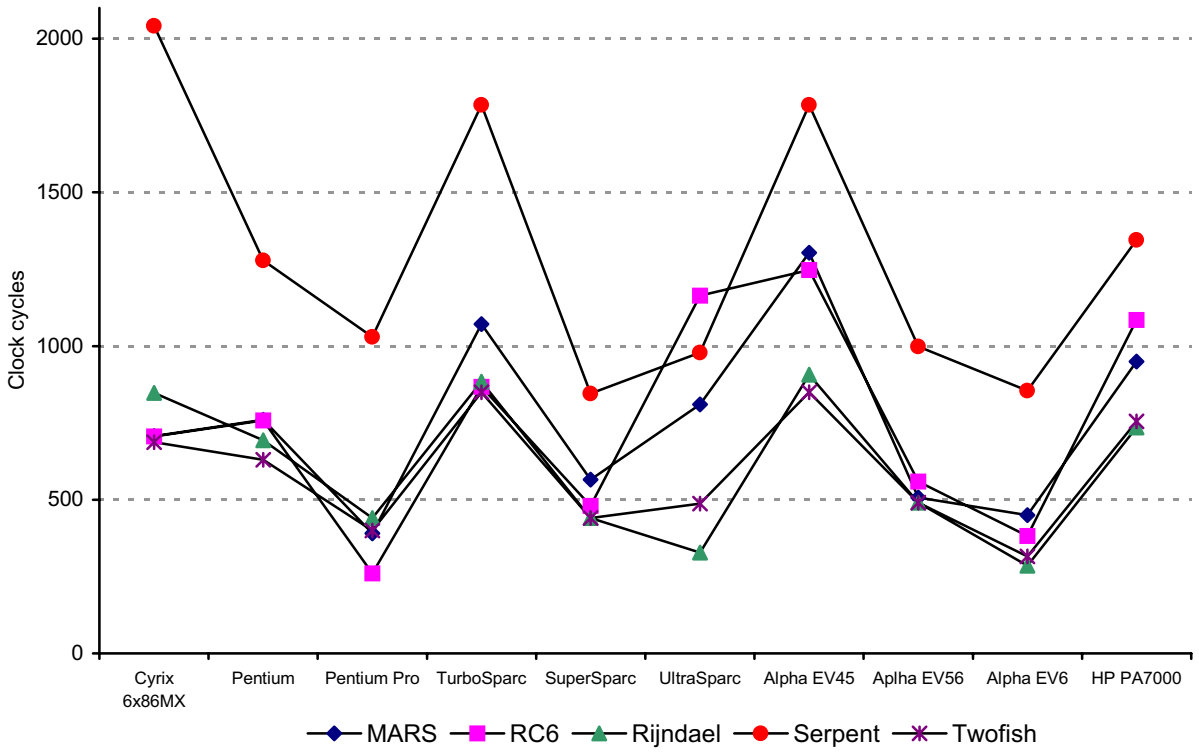

Figure 4: Encryption Speeds for 128-bit Keys in C

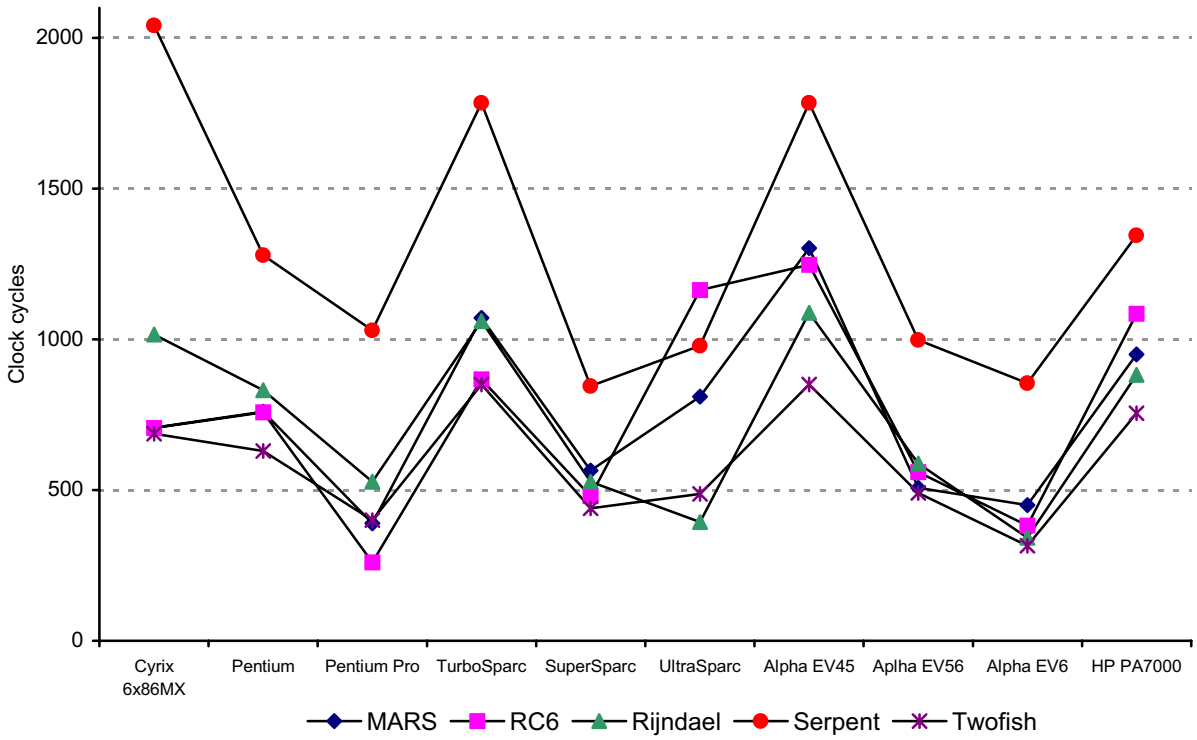

Figure 5: Encryption Speeds for 192-bit Keys in C

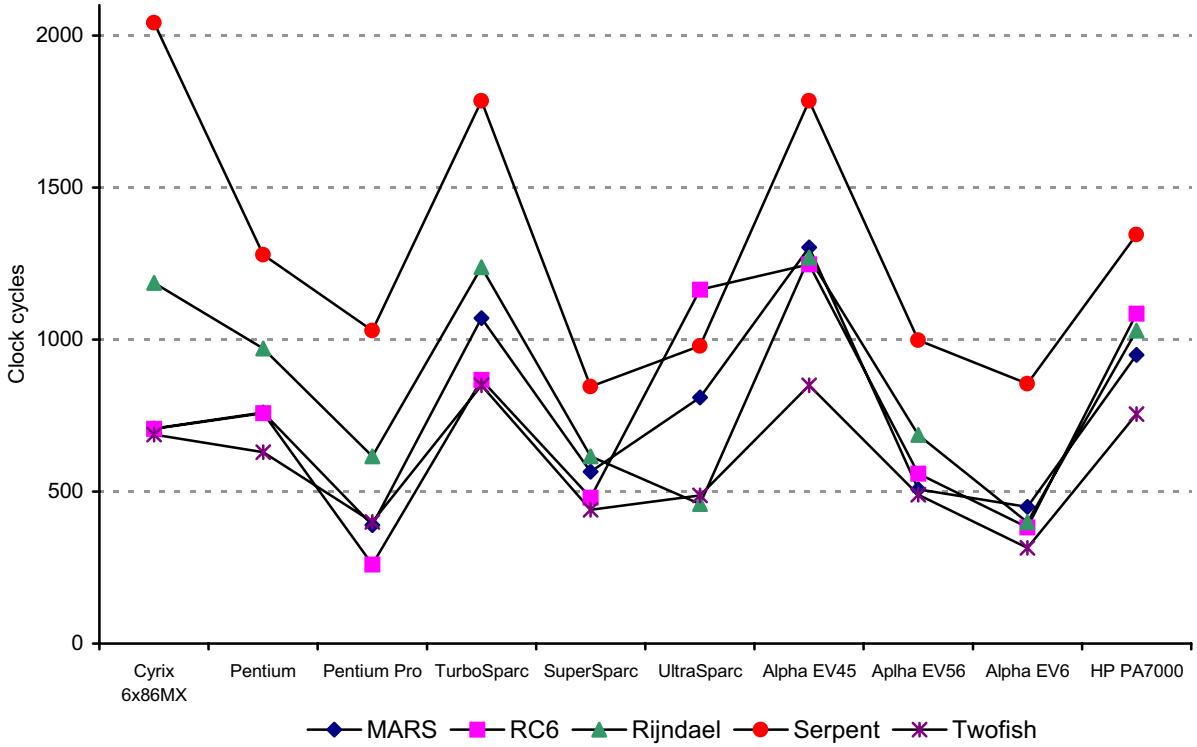

Figure 6: Encryption Speeds for 256-bit Keys in C

Interestingly enough, with the exception of MARS and RC6 on Pentium-Pro–class CPUs, the relative performance of the different algorithms was fairly constant. MARS and RC6 are exceptions because they rely heavily on data-dependent rotations and 32-bit multiplications. These operations are not part of the standard RISC instruction-set core, and their performance varies heavily with CPU [SW97]. Normally these two operations are very slow, but on the Pentium Pro, Pentium II, and Pentium III they are fast. Hence, these two algorithms are relatively faster on these CPUs than on both more and less sophisticated CPUs.

For 128-bit keys, Rijndael and Twofish are the fastest algorithms, MARS and RC6 are in the middle, and Serpent is slowest. For larger key lengths, Rijndael gets progressively slower; in some implementations it becomes slower than MARS. These trends hold true in both C and assembly language, although Serpent tends to produce C code that is closer to its assembly-language performance than the others.

3.2.1 Key Setup Plus Encryption

For encryption of short blocks of plaintext, performance is a function of both encryption speed and key setup speed. Much less data is available on key setup on various platforms, but we can estimate from the data we have. We use the key setup estimates from [SKW+98b]. Note that the numbers for Twofish take advantage of its flexibility in key setup vs. encryption.

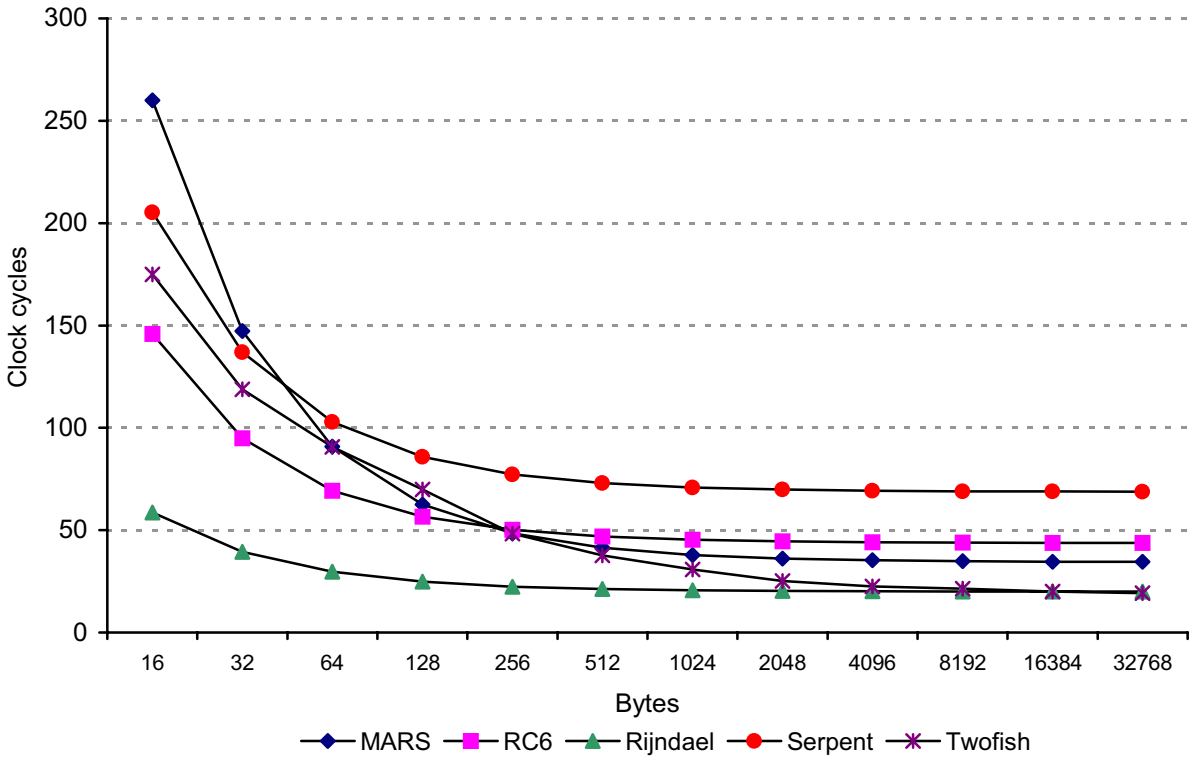

Figure 7: Key Setup and Encryption Rate, per Byte, for 128-bit Keys on a Pentium in Assembly

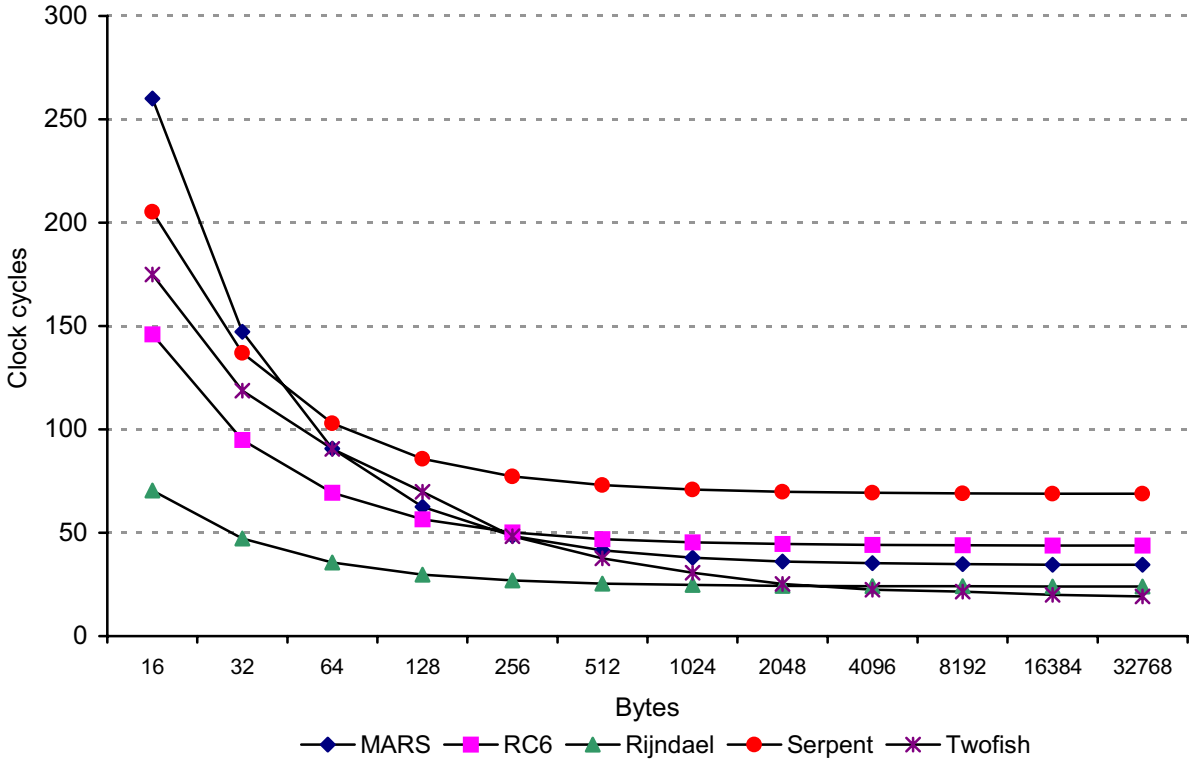

Figure 8: Key Setup and Encryption Rate, per Byte, for 192-bit Keys on a Pentium in Assembly

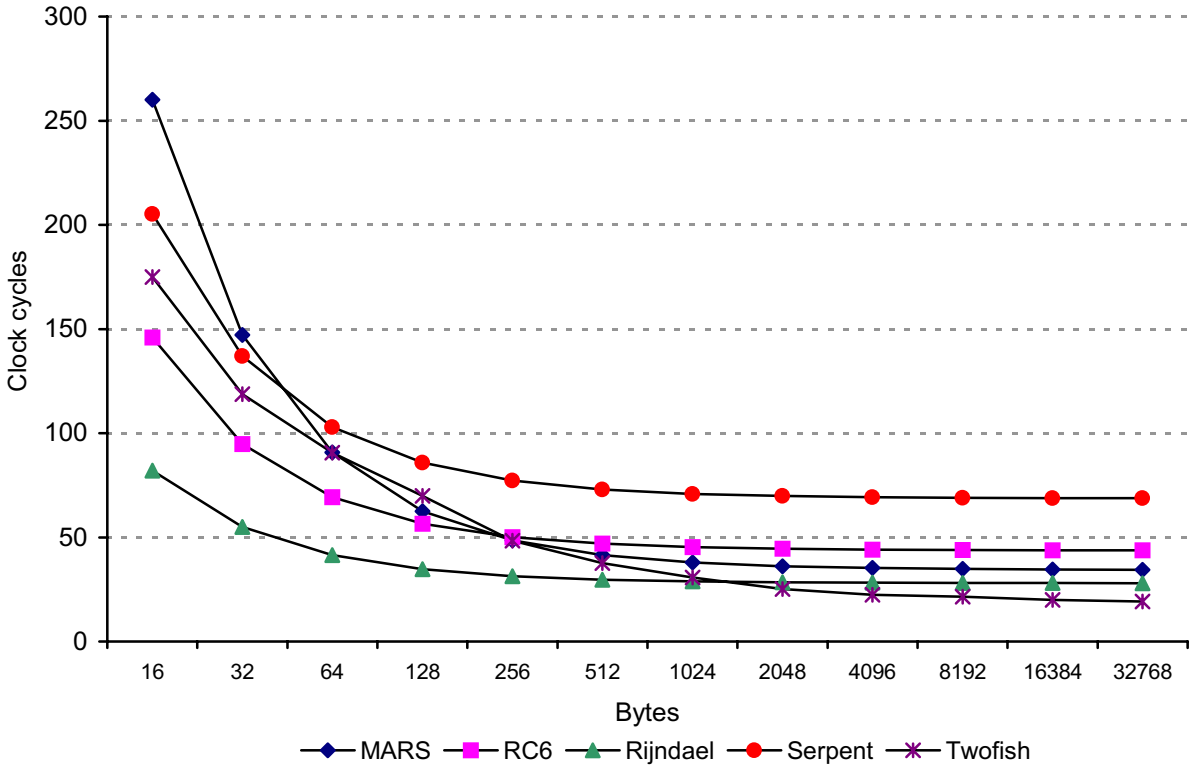

Figure 9: Key Setup and Encryption Rate, per Byte, for 256-bit Keys on a Pentium in Assembly

Rijndael is by far the best algorithm for encrypting small blocks, although the algorithms quickly normalize to their "natural" ordering. The 16-byte measure is of particular interest, because that is the speed of the algorithm if used as a hash function.

4 Software Performance of "Maximal Insecure Variants"

In [Bih98,Bih99], Biham introduced the notion of comparing the algorithms based on their "minimal secure variants." Different design teams were more or less conservative than each other; the number of rounds in their final submissions was not a fixed percentage of the number of rounds they could successfully cryptanalyze. Biham tried to normalize the algorithms by determining the minimal number of rounds that is secure (either as described by the designers or other cryptographers, or Biham's "best guess"), and then added a standard two cycles.

In his comments to NIST [Knu99], Lars Knudsen presented another rule of thumb for changing the number of rounds of the different algorithms: "Let *r* be the maximum number of rounds for which there is an attack faster than exhaustive key search. Choose *2r* rounds for the cipher." This rule gives us a new, although similar, measure of comparison.

For this comparison, we leave the scaling factor for another discussion, and simply compare the maximal number of rounds for which the best cryptanalytic attack is less complex than a 256-bit brute-force search; call this the "Maximal Insecure Variant." For Table 2, we use the best published cryptanalytic results as explained below.

Algorithm Name	Rounds
MARS	9 of 16
RC6	15 of 20
Rijndael	8 of 14
Serpent	9 of 32
Twofish	6 of 16

Table 2: Maximal Insecure Variants

If there is a weak key class, we count the attack if the probability of finding the weak key times the complexity of the attack is no more than 2^{256}. If there is a related-key attack, we make a similar calculation.

- MARS is complicated because there are four different types of round functions. The MARS design team believes that the cryptographic strength of the algorithm is in the "core"; hence, we concentrate on those rounds. There is an 11-round (of 16 total) attack of the MARS core [KKS00a]. There is also an attack against the cipher with the four different round functions symmetrically reduced from 8 rounds to 3 [KS00]. We make the somewhat arbitrary decision to take a scaling "halfway" between those two results: 9 rounds.
- RC6 has an attack against 15 rounds [KM00]. This attack also applies to a weak key class; the attack works for 1 in 2^{60} keys, and the complexity of the attack is 2^{170}. Hence, this attack counts by our definition. The RC6 designers estimate that 16 rounds is attackable, although they give no concrete attack.
- Rijndael has a distinguishing attack against 8 rounds [FKS+00a].
- Serpent has a distinguishing attack against 9 rounds [KKS00a,KKS00b]. The authors estimate that the longest variant that is not as secure as exhaustive search is 15 rounds, although they have no attack.
- Twofish has attacks against 6 rounds. The related-key attacks discussed in [SKW+98,SKW+99a] do not work [FKS+00b].

Unfortunately, these comparisons are fundamentally flawed, because they unfairly benefit algorithms that have been cryptanalyzed the least. Despite almost two years in the public, there has been little cryptanalysis of the five algorithms. We urge caution in reading too much into these numbers, and would like to see further cryptanalysis of the five algorithms.

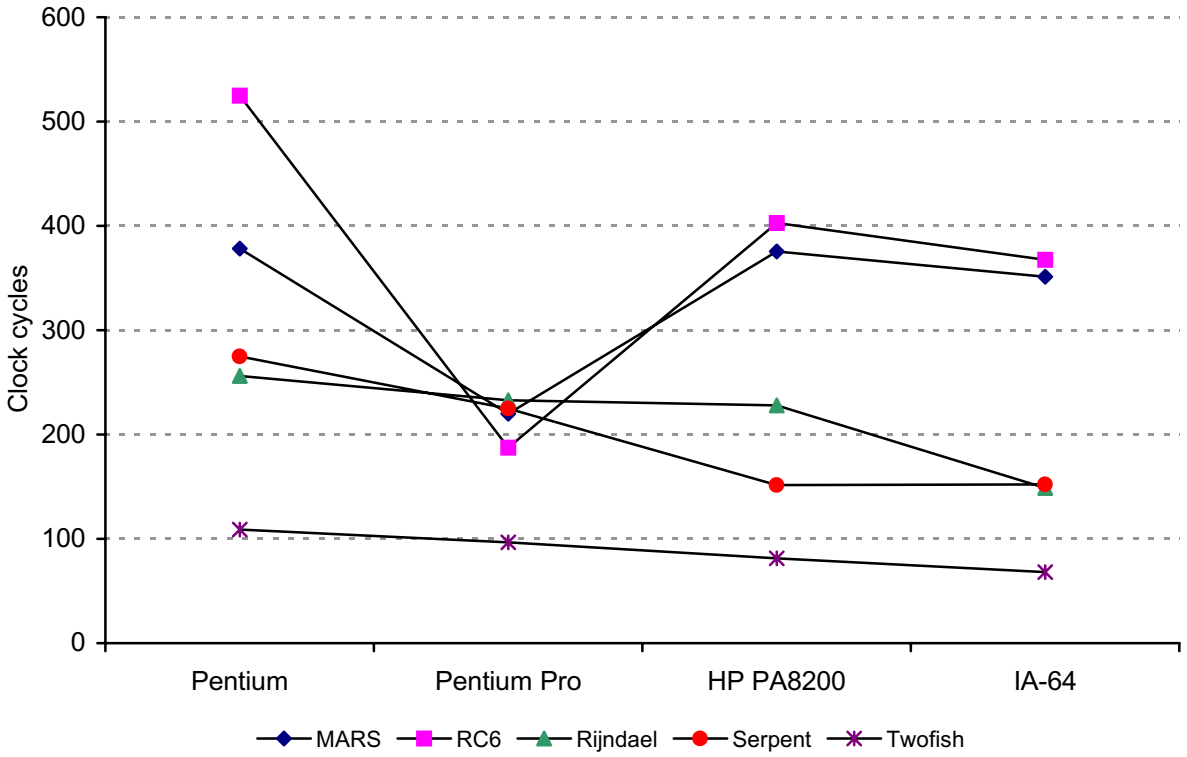

Figure 10: Encryption Speeds for the Minimal Secure Variant In Assembly

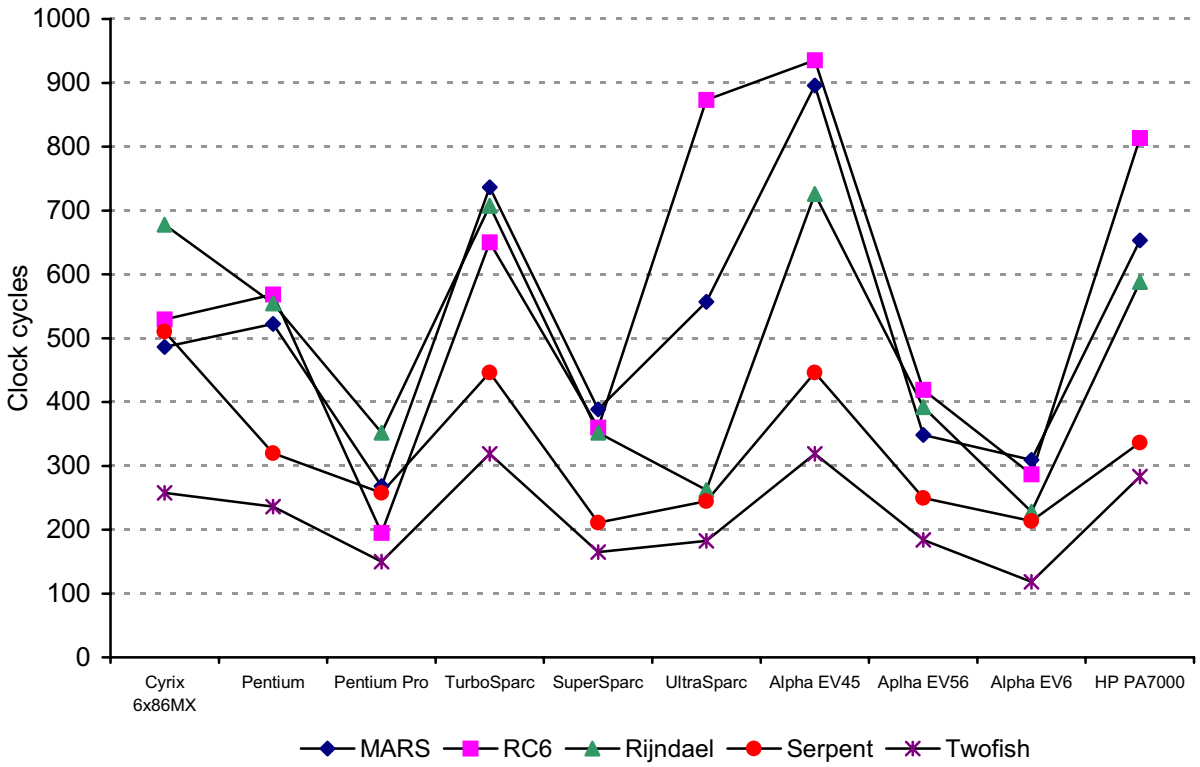

Figure 11: Encryption Speeds for the Minimal Secure Variant in C

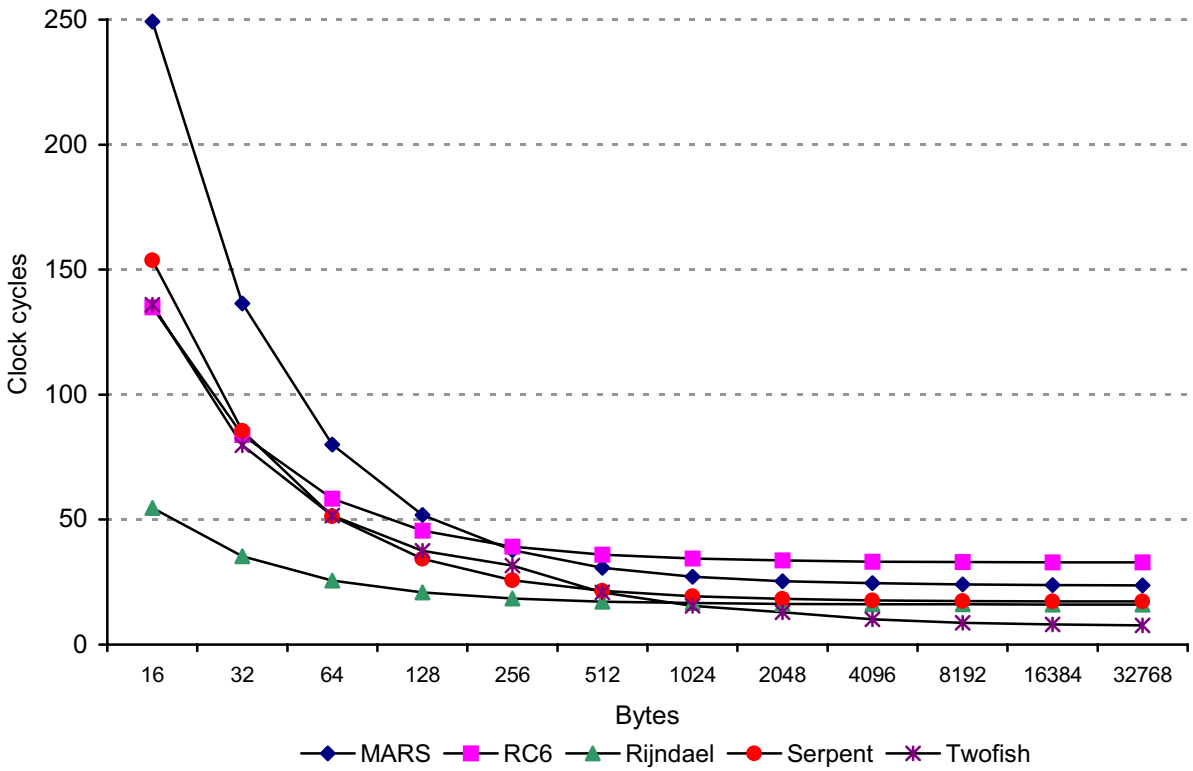

Figure 12: Key Setup and Encryption Rate, per Byte, for the Minimal Secure Variant on a Pentium in Assembly

Of course we are not recommending using these round numbers for the algorithms; any AES selection would obviously scale these numbers by a small linear factor. But the relative speeds would be independent of this scaling, and are hence relative to the selection process.

5 Performance on Memory-Limited 8-bit Smart Cards

As discussed in [SKW+99b], 8-bit implementations tend to be concerned more with fit than with performance. That is, RAM requirements are more important than clock speed.

Most commodity 8-bit smart card CPUs today include from 128 to 256 bytes of on-board RAM. Each CPU family typically contains members with more RAM and a correspondingly higher cost. RC6 has no effective way to compute subkeys on the fly, thus requiring that all the subkeys be precomputed whenever a key is changed. The RC6 subkeys consume from 176 bytes of "extra" RAM—more than is available on many commonly used CPUs. Although some CPUs include enough RAM to hold the subkeys, it is often unrealistic to assume that such a large fraction of the RAM can be dedicated solely to the encryption function. In a smart-card operating system, for example, encryption is a minor part of the system and won't be able to use more than half of the available RAM. Obviously, if an algorithm does not fit on the desired CPU, with its particular RAM/ROM configuration, its performance on that CPU family is irrelevant.

For some of the candidates, the performance or RAM requirements can depend on whether encryption or decryption is being performed. It is tempting to consider only one of the two operations, using the argument that the smart card can perform the more efficient side of the operation, and the terminal (with the faster, larger, and more RAM-endowed CPU) can perform the less efficient side. Experience shows that this does not work. Many smart card terminals contain a secure module; in many cases this secure module is itself a smart card chip. In several applications, it is a requirement that two smart cards execute a protocol together, and many existing protocols use both encryption and decryption on the same smart card.

Table 3 compares the RAM requirements for the different AES submissions. This table is identical to that in [SKW+99b], with the exception of MARS.

Algorithm Name	Smart Card RAM (bytes)
MARS	100
RC6	210
Rijndael	52
Serpent	50
Twofish	60

Table 3: AES Candidates' Smart Card RAM Requirements

With its new key schedule "tweak," MARS has on-the-fly subkey generation and appears to require 60 bytes of subkey RAM. When added to the 16 bytes of plaintext and 16 bytes of key, plus other scratch variables, it appears that about 100 bytes of RAM are required. This amount of RAM is much more reasonable than the pre-tweak version, although it is still nearly twice the size of Rijndael, for example.

The new key schedule does allow MARS to fit on small 8-bit CPUs. It should be noted, however, that in this case, the entire rather complex key schedule is recomputed for each block processed, slowing down performance by probably at least a factor of four over a precomputed key schedule

These numbers assume that the key must be stored in RAM, and that the key must still be available after encrypting a single block of text. The results seem to fall into two categories: algorithms that can fit on any smart card (less than 128 bytes of RAM required), and algorithms that can fit on higher-end smart cards (between 128 and 256 bytes of RAM required).[2]

Even if an algorithm fits onto a smart card, it should be noted that the card functionality will include much more than block encryption, and the encryption application will not have the card's entire RAM capacity available to it. So, while an algorithm that requires about 200 bytes of RAM can theoretically fit on a 256-byte smart card, it probably won't be possible to run the smart card application that calls the encryption. For many applications, a RAM requirement of more than 64 bytes just isn't practical.

Given all these considerations, the only algorithms that seem to be suitable for widespread smart card implementation are Rijndael, Serpent, and Twofish.

6 Conclusions

The principal goal guiding the design of any encryption algorithm must be security. If an algorithm can be successfully cryptanalyzed, it is not worth using. In the real world, however, performance and implementation cost are always of concern. For most operational systems, encryption is simply another feature that must be incorporated into a design, and must be traded off with other features. Efficiency, whether software encryption speed, hardware gate count, or hardware key-setup speed, may mean the difference between using AES or a home-grown cipher. Therefore, AES must be efficient.

Efficiency means many different things, depending on context. Efficiency may mean bulk encryption speed in software. Encryption may mean key-setup time in hardware. Efficiency may mean hardware gate count, or speed as a hash function, or speed for short messages on 8-bit smart card CPUs. As a standard, AES must be efficient in all of these meanings, since it will be used in all of these applications.

The most obvious conclusion that can be drawn from this exercise is that it is very difficult to compare cipher designs for efficiency, and even more difficult to design ciphers that are efficient across all platforms and all uses. It's far easier to design a cipher to be efficient on one platform, and then let the other platforms come out as they may. In the previous sections, we have tried to summarize the efficiencies of the AES candidates against a variety of metrics. The next thing to do is would be to assign a numerical score to each metric and each algorithm, then weights to each of the metrics, and finally to calculate an overall score for the different algorithms. While appearing objective, this would be more subjective than we want to be; we leave it as an exercise for the reader.

The performance comparisons will most likely leave NIST in a bit of a quandary. The easiest thing for them to do would be to decide that certain platforms are important and others are unimportant, and to choose an AES candidate that is efficient only on the important platforms. Unfortunately, AES will become a standard. This means AES will have to work in a variety of current and future applications, doing all sorts of different encryption tasks. Specifically:

[2] We do not consider the solution of using EEPROM to store the expanded key, as this is not practical in many smart card protocols that require the use of a session key. All algorithms have significantly reduced RAM requirements if this solution is used.

- AES will have to be able to encrypt bulk data quickly on top-end 32-bit and 64-bit CPUs. The algorithm will be used to encrypt streaming video and audio to the desktop in real time.
- AES will have to be able to fit on small 8-bit CPUs in smart cards. To a first approximation, all DES implementations in the world are on small CPUs with very little RAM. They are in burglar alarms, electricity meters, pay-TV devices, and smart cards. Certainly, some of these applications will use 32-bit CPUs as those get cheaper, but that simply means that there will be another set of even smaller 8-bit applications. These CPUs will not go away; they will only become smaller and more pervasive.
- AES will have to be efficient on the smaller, weaker 32-bit CPUs. Smart cards won't be getting Pentium-class CPUs for a long time. The first 32-bit smart cards will have simple CPUs with a simple instruction set.
- AES will have to be efficient in hardware, in not very many gates. There are lots of encryption applications in dedicated hardware: contactless cards for fare payment, for example.
- AES will have to be key agile. There are many applications where small amounts of text are encrypted with each key, and the key changes frequently. IPsec is an excellent example of this kind of application. This is a very different optimization problem than encrypting a lot of data with a single key.
- AES will have to be able to be parallelized. Sometimes you have a lot of gates in hardware, and raw speed is all you care about.
- AES will have to work on DSPs. Sooner or later, your cell phone will have proper encryption built in. So will your digital camera and your digital video recorder.
- AES will need to work as a hash function. There are many applications where DES is used both for encryption and authentication; there just isn't enough room for a second cryptographic primitive. AES will have to serve these same two roles.

Choosing a single algorithm for all these applications is not easy, but that's what we have to do. And when AES becomes a standard, customers will want their encryption products to be "buzzword compliant." They'll demand it in hardware, in desktop computer software, on smart cards, in electronic-commerce terminals, and in other places we never thought it would be used. Anything chosen as AES has to work in all those applications.

7 Authors' Biases

As authors of the Twofish algorithm, we cannot claim to be unbiased commentators on the AES submissions. However, we have tried to be evenhanded and fair. Many of the performance numbers in this paper are estimates; we simply did not have time to code each submission in optimized assembly language. As more accurate performance numbers appeared, we have updated the tables in this paper. We will continue to do so.

References

ABK98 R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," NIST AES Proposal, Jun 98.

Alm99 K. Almquist, "AES Candidate Performance on the Alpha 21164 Processor," version 2, posted to sci.crypt, 4 Jan 1999.

BGG+99 O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Puopard, J. Stern, and S. Vaudenay, "Report on the AES Candidates," *Second AES Candidate Conference*, 1999.

Bih98 E. Biham, "Design Tradeoffs of the AES Candidates," invited talk presented at ASIACRYPT '98, Beijing, 1998.

Bih99 E. Biham, "A Note Comparing the AES Candidates," revised version, comment submitted to NIST, 1999.

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 98.

DR98 J. Daemen and V. Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, Jun 98.

FKS+00a N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved Cryptanalysis of Rijndael," *Fast Software Encryption, 7th International Workshop,* Springer-Verlag, 2000, to appear.

FKS+00b N. Ferguson, J. Kelsey, B. Schneier, D. Whiting, "A Twofish Retreat: Related-Key Attacks Against Reduced-Round Twofish," Twofish Technical Report #6, http://www.counterpane.com/twofish-related.html, Feb 00.

Gla98 B. Gladman, "AES Algorithm Efficiency," http://www.seven77.demon.co.uk/aes.htm, 1 Dec 98.

Gra00 L. Granboulan, "AES: Timings of the Best Known Implementations," http://www.dmi.ens.fr/~granboul/recherche/AES/timings.html, 15 Jan 00.

KM00 L. Knudsen and W. Meier, "Correlations in RC6," *Fast Software Encryption, 7th International Workshop,* Springer-Verlag, 2000, to appear.

Knu99 L. Knudsen, "Some Thoughts on the AES Process," comment submitted to NIST, 15 April 1999.

KS00 J. Kelsey and B. Schneier, "Mars Attacks! Cryptanalyzing Reduced-Round Variants of MARS," ," *Third AES Candidate Conference,* 2000, to appear.

KSS00a J. Kelsey, T. Kohno, and B. Schneier, "Amplified Boomerand Attacks Against Reduced-Round MARS and Serpent," *Fast Software Encryption, 7th International Workshop,* Springer-Verlag, 2000, to appear.

KSS00b T. Kohno, J. Kelsey, and B. Schneier, "Preliminary Cryptanalysis of Reduced-Round Serpent," *Third AES Candidate Conference,* 2000, to appear.

Lip00 H. Lipmaa, "AES Ciphers: Speed," http://home.cyber.ee/helger/aes/table.html, 15 Jan 00.

NBD+99 J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback, "Status Report on the First Round of the Development of the Advanced Encryption Standard," *Journal of Research of the National Institute of Standards and Technology,* v. 104, n. 5, Sept.–Oct. 1999, pp. 435–459.

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

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

PRB98 P. Preneel, V. Rijmen, and A. Bosselaers, "Principles and Performance of Cryptographic Algorithms," *Dr. Dobb's Journal*, v. 23, n. 12, 1998, pp. 126–131.

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

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 98.

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.

SKW+99b B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Performance Comparison of the AES Submissions," *Second AES Candidate Conference*, 1999.

WS98 D. Whiting and B. Schneier, "Improved Twofish Implementations," Twofish Technical Report #3, Counterpane Systems, 2 Dec 98.