From: "mrobshaw" <mrobshaw@supanet.com>
To: <AESround2@nist.gov>
Subject: comments
Date: Mon, 15 May 2000 22:41:10 +0100
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.2314.1300
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300

Dear Sirs,

Please find attached some comments on the suitability of RC6 for the AES.

Yours faithfully,

Matt Robshaw

The Case for RC6 as the AES

Ronald L. Rivest[1], M.J.B. Robshaw[2], and Y.L. Yin[3]

[1] M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. rivest@mit.edu
[2] 88 Hadyn Pk. Rd., London, W12 9AG, UK. mrobshaw@supanet.com
[3] NTT Multimedia Communications Laboratory, 250 Cambridge Ave., Palo Alto, CA 94306, USA. yiqun@nttmcl.com

1 Introduction

Over the past few weeks an enormous amount of information has been presented to those following the AES effort. At a conference it is often the headline claims that are remembered. A more detailed view takes time to come out.

While we will clearly consider the security implications of what was said at the third AES conference in New York, much of this note will concentrate on the huge variety of performance-related information that was presented. We will take some time to look at the data not just at face value, but also to see what trends are emerging for the future. It's interesting to discover that this can give a very different picture.

Our significant regret is the very limited amount of time that we have had after the 3rd AES conference to prepare this document. The more we have looked at recent controversial issues, the more we have found to question. Much is riding on the success of the AES. After such a long time and such an enormous effort, sound-bites and casual summaries seem to be a little out of place. Instead, the cryptographic community owes itself the luxury of looking long and hard at the details.

2 Implementation and performance issues

Many issues were raised recently about the suitability of RC6 in different implementation environments. We start by responding to some of the criticism we received and then highlight many of the successes for RC6 that have been forgotten, overlooked, or are quite simply taken for granted.

2.1 Key agility and IPsec

While "key agility" is one of the properties being considered during the AES process, it is not so easy to assess its actual importance in current and prospective applications. Doing so requires a broad understanding of the system being considered, an understanding of the statistics of the traffic being encrypted, and perhaps a little mathematical queuing theory as well.

Of course, it is easy to say that "shorter key setup times are better, all other things being equal." But usually all other things *aren't* equal, and so one must consider the tradeoffs more explicitly.

Key schedule security. ¿From a security viewpoint, a key setup that provides more thorough mixing of the key bits can provide greater protection against possible vulnerabilities due to key separation (as for Twofish [35, 28]), and vulnerabilities due to overly simple dependencies between round keys (as for Rijndael [19]). The cryptographic advantages of a strong key setup routine can be very significant. Given that security is a more important criterion for AES than efficiency, our above maxim should perhaps be reworded as "stronger key setup routines are better, all other things being equal."

Furthermore, the protection purchased by strong key setup is "cheap," as it is paid for only once per message to be encrypted, rather than once again for every block of the message. For example, if one were to double the number of rounds of encryption per block, this would incur a *multiplicative* penalty of a factor of two for encrypting a message, whereas a strong key setup routine only incurs an *additive* penalty paid at the beginning.

Key schedule performance. Turning now to the secondary criterion of efficiency, we see that the obvious parameters that are relevant are the relative speed of key setup versus encryption and the length of the message to be encrypted. It may be the case that a stronger key setup enables an encryption algorithm to run particularly quickly (as for RC6), so we need to determine how to assess the combination of the somewhat longer key setup time with a somewhat faster encryption rate.

Most often, there is not just a single message to be encrypted, but a stream of packets (e.g. IP packets) to be encrypted. The distribution of packet lengths then becomes the most relevant parameter. For Internet traffic Claffy [10] quotes the following saying.

> *More than half the packets are mice, more than half the payload are elephants.*

So, it is well-known that while "most packets are mice" (i.e., most packets are small), it is the case that "most of the payload are elephants" (i.e., most of the data occurs in large packets). A fairly realistic and plausible model of Internet traffic would have 60% of the packets having length 40 bytes, 30% of the traffic having around 560 bytes, and 10% of the traffic having length 1500 bytes. While sixty percent of the packets have length 40, the average packet size has length 342 bytes.

Understanding such statistics is important, since the average time to encrypt an Internet packet is the same as the time spent to encrypt a packet of average size, by the standard theorem of probability theory known as *linearity of expectation*. This assumes that the time to encrypt the packet is the key setup time plus the time needed to encrypt each block of data of the packet. *Thus, for Internet traffic, it would be reasonable to compare encryption algorithms based on the total time they require to encrypt a block of size 342 bytes.*

The throughput of an algorithm is directly related to the speed at which an algorithm can encrypt a packet of average size (e.g. 342 bytes).

While this is a good first-order rule of thumb, there are other thoughts or considerations that may come to mind. For example, Steve Bellovin [7] stated concerns that for VPNs (virtual private networks) using IPsec, an encryption unit using an algorithm with longer key setup (but presumably shorter average time to encrypt) might have difficulty when the input contained a sequence of short packets that required different keys. Perhaps the encryption unit would "fall behind" in such circumstances as too many key setups had to be performed.

The key to resolving this concern is to realize that routers and VPN units are normally built with surprisingly large buffers (or queues), in order not to interfere with the TCP/IP window size algorithm used for controlling information flow. The standard rule of thumb is that such a unit should have a buffer capable of holding 100 milliseconds of traffic [3]. If you have a router or VPN box working at T3 rates (45 Mbit/sec), this unit would normally have a buffer capable of holding around 560 Kbytes, or some 1600 average Internet packets. This buffer smoothes out the variability of packet arrival times and packet sizes. The encryption unit, which encrypts faster on the average than the transmission rate, can "work ahead" on the queue of packets waiting to go out. The queue size is so large that the law of large numbers takes over, and all that really matters is the length of time taken to encrypt a packet of average size.

As an aside we note that, of course, if the queue empties out, then this argument doesn't apply. But in that case the channel is not being fully utilized anyway, so throughput doesn't matter, and the extra latency of key setup for an incoming short packet—on the order of microseconds at most—is negligible compared to the transmission time (light travels less than a mile in a microsecond, and takes a couple of *milliseconds* to get from Boston to San Francisco).

So the above argument shows that for VPN's or other encryption units handling IP traffic, the most relevant parameter is the speed at which a packet of average size can be encrypted.

There are other techniques that would typically be employed in such circumstances as well. For example, by using extra memory, one can cache previously computed tables of round keys, and avoid key setup altogether when a new packet has the same key as a previous packet. Bellovin [8] has analyzed some typical VPN traffic, and found that *a cache of 100 round key setups enables over 99.9% of the key setups to be skipped altogether!* Note that for RC6, which has a round key table of 176 bytes, 100 keys occupy only 17.6 Kbytes, which is a trivial amount of memory compared to the normal packet buffers mentioned above. A "hit rate" of 99.9% means a "miss rate" of 0.1%, so that key setup only needs to be performed 0.1% of the time. In this case, one should divide the normal key setup time by a factor of 1000 to obtain the "effective key setup time." (Interestingly, such a standard "round key caching" technique yields the most benefits for algorithms, such as RC6, that have relatively long key setup times, and yield little or no benefit for algorithms that have don't put any effort into key setup.) In any case, key setup time becomes effectively irrelevant once caching of round keys is used.

One may wonder about other classes of data traffic, such as ATM traffic, that

has only short packets. The most important point here is that ATM networks are at the *transport* layer of the network protocol stack, whereas encryption is normally done at the *network* layer. ATM networks may be used to handle encrypted traffic, but the encryption is not done at the ATM packet (transport) level, but at the higher IP (network) level. Thus, our previous arguments still apply. (In any case, it seems likely that ATM networks are being eliminated in favor of networks that run IP packets directly over the lower link-level connections.)

In summary, it seems that using a strong key setup routine incurs little performance penalty per se for the most common applications (e.g. IP traffic), as long as the time to encrypt a packet of average size (including key setup) is good.

Finally we might also note that encrypting according to the IPsec protocol will typically increase the length of the packet being encrypted by 32–44 bytes (the former with no authentication, the latter with authentication), in order to accommodate the new IPsec headers, IVs, padding, authentication field, etc. The time taken to transmit these additional fields provides additional "slack time" for performing key setup. The effective data rate (as seen by the encryption unit) is noticeably less than the actual data rate (as seen on the encrypted channel). While interesting, this may be less significant than some of our other observations.

Conclusion. There are many good security reasons to use a key schedule that offers significant mixing of the user-supplied key data prior to encryption. This might involve some additional cost. But while it was claimed that this would have a significant impact on applications such as IPsec, a close analysis suggests that a longer key setup in fact has very few, if any, practical implications.

Indeed, it seems that the extra security offered by a more thorough key schedule makes it far more appealing as an AES algorithm than a light key schedule for which there may well be vulnerabilities.

2.2 Block ciphers, modes, and future processors

As well as the choice of block cipher, the way it is used will have a great effect on the rate of encryption.

> *High performance applications, such as high speed network encryption, will require the increase in output, and as a result, often focus on a non-feedback mode of operations such as counter mode to obtain performance.*
> – Weeks, Bean, Rozylowicz, Ficke [49]

> *However we believe that for many applications which require high encryption rates, non-feedback modes [...] will be the modes of choice. Note that the Counter mode grew out of the need for high speed encryption of ATM networks which required parallelization of the encryption algorithm.*
> – Elburt, Yip, Chetwynd, Paar [18]

[...] using current standards does not permit to fully utilize the performance advantage of the hardware implementations of secret key cryptosystems, based on parallel processing of multiple blocks of data.
– Gaj, Chodowiec [20].

Certainly most commentators conclude that for high performance applications non-feedback modes will be preferred. As such, when estimating the throughput of a cipher one should also consider the mode that the cipher will be used in. Is this likely to make a difference? It can make an enormous difference.

On advanced CPUs the relative performance of candidates may differ widely between feedback and non-feedback (or interleaved) modes.
– Clapp [11]

Following the standardization of DES there followed a FIPS document on the modes of DES. NIST has already announced its intention to do the same with the AES. Thus, in looking at the AES candidates it is not sufficient to look at their performance in isolation from a potential mode of use. Even more importantly, we should not restrict ourselves to looking solely at the four modes standardized for use with DES. New applications require new modes and there are very pressing reasons to consider the suitability of a cipher to modes like counter mode [33] and interleaved modes. These are the modes that are going to be used for the fastest applications, where performance matters.

Thus, merely comparing the algorithms on the performance of a single-block encryption does not give the full picture. And since many modern processors will permit the simultaneous encryption of multiple blocks we'll see that it is in these cases that RC6 truly comes into its own.

2.3 Performance on the IA-64

In [51] a careful assessment was made of the performance of the different AES finalists on the IA-64 and PA-RISC processor. Due to limited time we will only consider the IA-64 here.

The times for encrypting a single block on the IA-64 for Mars, RC6, Rijndael, Serpent, and Twofish respectively have been given as 511, 490, 125, 565, and 182 cycles [51]. We have a slight improvement on this for RC6 and our current best assembly implementation of a single-block encryption with RC6 is 470 cycles [53].

The surprisingly slow encryption performance of RC6 is due to the very poor support provided in the IA-64 for a 32-bit integer multiplication. Either this is performed via a series of 16-bit multiplications within the parallel instruction set, or it is performed within a separate computational unit for which there are delays in delivering and retrieving information. This presents a very strange evolutionary step, given the excellent support for this operation within the IA-32 architectures. Perhaps it is better viewed as a design anomaly rather than being truly indicative of future trends. For this reason, even the crude estimates for the (as yet hypothetical) IA-64^{++} given in [51] are more than illustrative. They

in fact represent the possibilities for RC6 if chosen as the AES. Certainly there is nothing at all intrinsic to the design of RC6 that makes it inherently slow on future architectures.

Yet even on the IA-64 we have not heard the full story.

Encrypting multiple blocks and non-feedback modes. Let us consider the case of processing multiple blocks simultaneously on the IA-64. Even though there is a delay in getting data to and from the multiplication operation in the IA-64, the multiplication itself takes one cycle. Thus, when multiple encryptions are processed they can be scheduled so that the arrival and departure of data to the multiplication operation is staggered but the multiplication operation is kept continually busy. To do this, all that is needed is a register-rich environment, as is provided by the IA-64. The results of our implementations on the IA-64 when processing 1, 2, 4, and 8 blocks simultaneously are given in Table 1.

number of blocks processed simultaneously	1	2	4	8
cycles	470	1001	766	1178
cycles/block	470	418	309	199

Table 1. Cycle counts for encrypting multiple blocks with RC6 on the IA-64. These are actual performance figures from verified implementations [53]. Except for the case of a single block which is hand-optimized assembly, the other figures used optimizing tools.

The figures for more than a single block were obtained via optimizing tools. They might be improved by hand-optimization or via any improvements in the performance of the tools available. In addition substantial effort was made to keep the size of the code reasonable. The performance figures presented in [51] were "optimized for performance, not code size or table size".

When using an algorithm for bulk encryption, when speed matters, it is accepted that a non-feedback mode would be preferred. When we compare the performance of the different AES finalists in a non-feedback mode, where the simultaneous processing of multiple-blocks is available, then we see that even on the IA-64, the performance of RC6 can compete with that of Twofish and when fully optimized might compete with that of Rijndael. Don't forget that this is on the IA-64 on which RC6 is particularly penalized. On some other 64-bit platforms it out-performs other finalists by far.

Non-feedback modes and other finalists. How do the other finalists compare when taking advantage of non-feedback modes? There is a general answer that applies to modern processors and a second additional issue that is specific to the IA-64.

Consider the issue specific to the IA-64 first. Using non-feedback mode allows for a straightforward design for 32-bit multiplication on the IA-64, since the latencies for non-feedback blocks can be overlapped. Other algoithms that pay performance penalties other than for latency will benefit far less from such optimizations. We are also able to make some other gains on top of that (which would potentially apply to the hypothetical IA-64^{++} or its equivalent) and this a result of a more general phenomenon that applies to many modern processors.

Whenever the issue of non-feedback modes is considered, we see that Rijndael typically gains the least, if anything at all. Twofish sometimes gains a little. In general terms this is because Rijndael and Twofish tend to offer more opportunities for parallelism during the encryption of a single block. Thus most quoted performance figures will already exploit what parallelism is available in modern processors. By contrast, the data-flow in RC6 is so simple that there are few opportunities for parallelism in a single block. At a sound-bite level, its performance appears to suffer. But once we move beyond that and realize that for performance-critical uses we would use a non-feedback mode, then the opportunities for parallelism with RC6 are no longer overlooked but instead become a winning factor.

Simultaneous encryption with different keys. The figures given in Table 1 used the same encryption key for each of the simultaneous streams. But many of the same observations can be used when we consider encrypting different blocks with different encryption keys. However we would need to be careful to ensure that all the subkeys are available as needed. Rough estimates suggest that we could encrypt with two different encryption keys but we would interleave the processing of each encryption strand twice. This would give roughly the same performance as that of encrypting four blocks simultaneously.

Even though our focus is on the high-performance non-feedback modes, our comments apply to any mode that allows the processing of multiple blocks. When different keys are accommodated it is possible to add interleaved modes of encryption and applications such as encrypting/decrypting data and simultaneously computing/verifying a MAC.

CBC + MAC. Consider the example of encrypting/decrypting a stream of data and simultaneously computing/verifying a MAC. The MAC might be computed using cipher block chaining and the encryption and MAC keys would be different. A first-cut prototype in "C" gave us an encryption-only routine that (on an IA-64 clocked at 500 Mhz) encrypted at 8.43 MBytes/sec and provided an encryption + MAC scheme at 7.48 MBytes/sec. The absolute speed is not that important. Rather we see that computing the MAC cost only around 10% in terms of throughput instead of the 100% penalty that might be anticipated[4].

[4] Such an overhead would refer to the typical technique of computing the encryption and the MAC separately. New research suggests new modes of use can achieve the same goal but use very different techniques [26].

2.4 Other processors

We have not had time to look at other platforms in any way near the same kind of depth as we did for the IA-64. Instead we can relate the experience of other implementers.

Alpha 21264. In [50] the AES finalists are compared on the Alpha 21264. Estimates are given for the time required to encrypt two blocks simultaneously and they show excellent performance for RC6. Being able to process more strands, as we could on the IA-64, would give even more opportunities for speed-up.

> *RC6 has the most potential for parallelism when multiple streams are processed on the same processor simultaneously in a single thread.*
> – Weiss and Binkert [50].

Note that if different keys were used then RC6 would really come into its own. If an algorithm offers no significant advantages when processing multiple blocks, then in a in a typical encryption + MAC application its performance would halve since a block would have to be processed twice; once for encryption and once for computing the MAC. Yet for RC6 the cycle count would remain around at 210 cycles per block perhaps with a small ($\approx 10\%$) overhead.

TriMedia. Clapp [11] reports on the possible implications of using non-feedback modes with the TriMedia CPU. The conclusions for this processor follow the same trends as we see for other advanced CPUs. When high-speed encryption is an important factor, non-feedback modes would typically be used. And when non-feedback modes are used, RC6 is typically the best performer—often by some considerable margin.

> *RC6 shows the greatest benefit from interleaved modes, considerably outperforming the other candidates.*
> – Clapp [11].

2.5 FPGA and hardware

At the 3rd AES conference, the last significant area of performance was considered. This was the issue of hardware performance, both in reconfigurable hardware (FPGAs) and also in custom and semi-custom ASIC designs. The biggest difficulty here is in the somewhat incomplete nature of the information available. Typically the cryptographic community, including ourselves, is more familiar with software implementation. As a consequence it can be hard to really highlight the significant issues from the papers that were presented.

FPGAs. Two papers on FPGA implementations [18, 20] seem to offer contradictary evidence while a third [48] provided detailed and informative observations though they weren't the results of actual implementations.

Since Serpent is bit-wise oriented, one would typically expect it to do well in FPGA and hardware implementations. This generally seems to be the case. There is however some confusion; in [20] we see that the measure of speed/area puts Serpent fourth among the finalists whereas in [18] it is first. Similar confusions reign for Twofish. In [18] Twofish appears to have the best speed/area measure whereas in [20] Twofish is the worst among the finalists. Clearly there are some discrepancies!

Even though RC6 does not outperform the other finalists on FPGAs as it might in software, it still performs very well. This seems to be somewhat contrary to public perception after the recent AES conference. And while it offers very good performance, it does so with exceptionally compact implementation. According to [20] Mars, Rijndael and Serpent all suffer in this regard.

> *For the low-cost, medium-size family of Xilinx FPGA devices, XC4000, only two ciphers, Twofish and RC6, were able to fit within the largest device from this family.*
> – Gaj and Chodowiec [20].

Note further that if both encryption and decryption are to be supported then Rijndael and Serpent once again suffer badly. The design of RC6 means that many of the encryption and decryption resources can be shared.

Even though Weaver and Wawrzynek [48] have concerns about the key schedule used in RC6—concerns that we have addressed in Section 2.1—they comment on the remarkable versatility of RC6.

> *The ability to reasonably reduce the hardware requirements without sacrificing too much performance is present, a useful feature when a low-cost implementation is desired.*
> – Weaver and Wawrzynek [48].

With regards to raw performance it is interesting to note that Elbirt et al. [18] place Twofish last (though they didn't implement Mars) both in a feedback and non-feedback mode. Serpent is placed first in both modes (as would be expected). For Rijndael and RC6 however the situation is interesting. For the feedback mode Rijndael outperforms RC6. But for implementations where speed really matters, when non-feedback modes would be used, RC6 outperforms Rijndael. Note the factor increase in performance speed between feedback and non-feedback modes as described by [18] and presented in Table 2.

Again, the trend is clear. RC6 gains significantly when compared to other finalists when we consider high-speed applications and the use of non-feedback modes. Rijndael typically offers the least gain.

	Mars	*RC6*	*Rijndael*	*Serpent*	*Twofish*
feedback mode (Mbit/sec.)	n/a	126.5	300.1	444.2	119.6
non-feedback mode (Gbit/sec.)	n/a	2.40	1.94	4.86	1.59
factor increase	n/a	19	6	11	13

Table 2. The factor increase in the encryption speed of the different algorithms in an FPGA implementation [18] when moving from feedback to high-performance non-feedback mode.

ASICs. With regards to custom and semi-custom ASIC designs, the information available once again gives a broad picture but few details. The different papers [25, 49, 42] appear to use different techniques and to have different aims.

Serpent and Rijndael tend to outperform the other finalists [25, 49]. In [43] Schneier describes Twofish as being good in hardware and RC6 poor, yet in terms of encryption speed there does not appear to be much between them and in [49] the pipelined mode of RC6 appears to outperform Twofish with what appears to be roughly comparable area. In other situations the positions are reversed.

Certainly RC6 and Mars seem to be somewhat hit by the use of the multiplication. (We should note that RC6 actually requires only a squaring operation.) Yet, in [25] the possibility of highly optimized multiplication circuits is mentioned and an estimate made for the performance of RC6 and Mars under such conditions (they would then both out-perform Twofish). However it is not quite clear what the financial cost would be of doing this. We are yet to be convinced that it would be prohibitively expensive to make such optimizations and we suspect that as a fraction of the total design and layout costs for a custom ASIC design, optimizations of any of the AES finalists would be relatively minor.

To summarize. The results available for hardware implementation are new and evolving. Existing papers on FPGA implementations seem at times to contradict each other. As more information becomes available we can expect the picture to stabilize somewhat. Even so, RC6 performs well in these summaries. Never the top performer, it is always a good performer. It offers remarkable flexibility in implementation [18, 48], it has excellent area requirements, and it provides good performance with excellent gains when we move to high-speed, non-feedback modes [18].

The information we have on ASIC designs is very new and also somewhat incomplete. There is very little detail available and in such a short time it is hard to really understand the true significance of recently announced data.

Our views on hardware implementations though remain unchanged. We believe that performance in software is a far more important criteria for the AES algorithm. We further believe that once the decision to make a dedicated hardware design has been undertaken, then the cost of significantly optimizing any

of the AES finalists will be a small fraction of the overall cost.

2.6 Smart card performance and the ARM processor

Strange claims appeared in Schneier's *Cryptogram* [43] newsletter for April 15, 2000. Among them was the often repeated refrain that RC6 doesn't fit on a cheap smart card. As has been made clear many times over, this is not true. RC6 fits in smart cards with as little as 128 bytes of RAM. This was demonstrated over a year ago at the 2nd AES conference in Rome [27].

Second was the strange claim that RC6 was too slow on ARM chips. Indeed the performance on ARM chips is very important and likely to become increasingly so. Yet, two independent studies [23, 34] have already shown that RC6 in fact offers the fastest encryption speed of any of the AES finalists on the ARM chip!

Hacez et al. [23] shows that in raw encryption speed RC6 is twice as fast as Rijndael, and more than ten times faster than Twofish! Messerges [34] shows that RC6 is once again the fastest and that Twofish is even slower than Mars and Serpent, being nearly three times slower than RC6. In fact, Hacez et al. [23] comment on the suitability of RC6 to the ARM processor.

> *RC6 was beyond a doubt the easiest candidate to implement on a 32 bits (sic) machine, as is illustrated by its incredibly short code [...]. On a speed point of view, RC6 is impressive too.*
> – Hacez, Koeune, and Quisquater, [23]

General smart card performance. It is accepted that smart cards won't be used for bulk encryption. At most a few blocks of data will be processed. On low-end smart cards, the performance of RC6 is adequate. Nothing more. But as we move to higher-end cards, including those that offer ARM chips and other advanced processors, the advantages of RC6 outweigh those of all other finalists. This is where smart cards will be heading over the life-time of the AES. This is where the advantages of RC6 will tell.

2.7 Other important environments

In among the whirlwind of new results, important information was often overlooked or forgotten. We provide a reminder here.

Java. The simplicity of a cipher is most acutely reflected in the Java performance of a cipher. This is in terms of code-size, performance, and potentially most critically, the amount of dynamic RAM used during the encryption process. With the increased importance of the Internet and its extension to mobile devices, the performance of the finalist in Java could well be vital. While there may well be many small processors in the coming years [46] many of them will in fact be Java-based, for instance in set-top boxes.

[...] while Java will hardly be the language of choice for high load servers it may well be the choice for medium load servers and especially clients. Add to that handheld and other small devices and performance in Java becomes an issue.
– Sterbenz and Lipp [47].

In every study [16, 17, 47] RC6 offers excellent performance in Java. Not just in raw encryption speed but perhaps more importantly in the amount of memory that is required. Considering Table 1 in [47] we see that RC6 has the smallest code size by a factor of three over the next closest finalists, and also uses the least amount of dynamic RAM.

The size of Java code and the amount of memory available in small portable devices could be an important issue for Java implementations. So comparing algorithms by looking at the raw encryption speed doesn't always give the full picture. One very crude attempt to take into account the penalty associated with using the different finalists is to add the memory requirements (both class file size and memory useage) in bytes m for the different algorithms—note however that this may well be implementation dependent so the measure we are about to derive should be viewed as guidance only. We then divide this quantity into the encryption speed e (where this is measured in bits/sec. for a 128-bit key). By doing this we get an indication of the performance that can be offered by a finalist when also taking into account some of the associated penalties. The figures we obtain are given here and the raw data was taken from [47]. Note that a higher value to e/m means that the cipher is more suited to the Java environment.

	Mars	*RC6*	*Rijndael*	*Serpent*	*Twofish*
e/m	1,469	11,093	753	878	1,173

Table 3. Encryption speeds for 128-bit keys divided by the memory requirements for a Java implementation of the different finalists [47]. The larger the value, the greater the indication that the algorithm may be more suitable for Java.

We are the first to say that this is not such a reliable measure and too much shouldn't be read into it. Nevertheless it is interesting since it illustrates the same general trends that we see in other implementation areas. The low score of several finalists shows that they obtain their fast performance at the expense of increased memory. This could be a considerable penalty in certain applications.

The suitability of RC6 for Java is readily apparent. So is its simplicity.

Although the least time was spent on optimizing RC6 it still comes out as the fastest algorithm on almost all platforms.
– Sterbenz and Lipp [47].

DSPs. One anticipated future trend is the growth of the market for DSPs and/or microprocessors with DSP capability. A recent *Business Week* article [37] quotes the claim that "By 2010, every microprocessor will have DSP." Schneier and Whiting [46] make similar suggestions.

> *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.*
> – Schneier and Whiting [46].

RC6 not only performs very well on processors of this type [52], but gains its impressive performance without the use of look-up tables which place additional burdens on memory.

> *Embedded system applications have often memory constraints.*
> – Wollinger, Wang, Guajardo and Paar [52].

As in the case of Java, we can calculate an index of suitability for the different finalists on the TMS320C6201 DSP. The raw data for this calculation comes from [52]. In fact, performance data is given there for both single-block and multi-block modes, something we have already considered with regards to the IA-64 and other advanced processors. To estimate one of the penalties in using an algorithm we might sum up the memory requirements m and divide this into the encryption speed e measured in Kbits/sec for the 128-bit key case. (It seems that the figure for Serpent in multi-block mode could be improved.) Again we must stress that this is not such a reliable measure and it is merely indicative of some general algorithm attributes. However it does give some indication of the relative suitability of the different finalists to the DSP environment, while also taking account of some of the associated penalties.

	Mars	*RC6*	*Rijndael*	*Serpent*	*Twofish*
e/m single-block mode	11	152	6	8	96
e/m multi-block mode	14	211	6	6	88

Table 4. Encryption speed for 128-bit keys divided by the memory requirements for a DSP implementation of the different AES finalists [52]. The larger the value, the greater the indication that the algorithm may be more suitable for DSPs.

Once again (see Table 4) RC6 compares very well to the other finalists. Note further the exceptional advantages that RC6 offers in multi-block applications and therefore in very high-speed, streaming applications. Once again, we see that the headline performance of some finalists is obtained at a cost.

32-bit. As is well known, RC6 offers the best performance on the NIST reference platform and on the full range of modern 32-bit Intel processors. Encryption times on the Pentium II are now 223 cycles per block [2].

Most of today's high-end computing base is deployed in PC's either in the workplace or at home, and these are 32-bit machines. When we couple the needs of greater processing power with the inevitable drop in prices of 32-bit processors, it is very clear that the mobile computing device market, including smart card applications, will inevitably shift to a 32-bit oriented processor base. This trend may take a few years to come to fruition, but its results are likely to be with us for the 20 or 30 years that might be required for the AES.

2.8 Simplicity is important

Hand-optimized assembly code will offer the best algorithm performance on any processor. Yet often developers will use portable code in a higher-level language and compile it for the environment of use. Under such circumstances the simplicity of a cipher is very important since it allows a compiler to produce well-optimized code. This means that good performance can be achieved without time-consuming and costly hand optimizations. Readers should be especially wary of techniques such as self-modifying code [46] or other unorthodox coding techniques that have been used to give good performance figures. This can lead to portability problems.

> *The Twofish algorithm compiles on the SGI using the MIPSpro compiler, but results in a Bus Error and a core dump when the* `blockEncrypt()` *and* `blockDecrypt()` *functions are invoked. This appears to be a problem with how the compiler is handling byte alignment in the optimized code.*
> – Bassham [5].

> *[...] self-modifying code and key-specific static areas is generally considered to be a bad programming practice.*
> – Aoki and Lipmaa [2].

2.9 Implementation conclusions

By any criteria, RC6 is the most suited of all the AES finalists to modern 32-bit processors, DSPs, ARM processors, and the Java environment. Together, these platforms cover the majority of today's important and growing application areas. These are likely to be the most important platforms for the lifetime of the AES.

On modern processors, including 64-bit designs, the performance of RC6 can be exceptional. Even on the IA-64 with its surprising design features, in high-performance non-feedback modes RC6 performs as well as any other finalist. On other modern processors, these high-performance non-feedback modes allow RC6 to outperform all other finalists by a significant margin.

Much was made at the 3rd AES conference in New York of the key agility of RC6. Yet we have shown in Section 2.1 that claims of the importance of key

agility, for instance to IPsec, have been greatly exaggerated. Instead we believe that the key schedule of RC6 is in fact a very attractive feature of the cipher. Not only is it immensely secure, surely the greatest priority for the AES, but we will see in Section 3.2 that its flexibility is particularly useful when it comes to hashing.

In fact much of the performance of RC6 is down to its flexibility and the exceptionally light demands the algorithm makes. Looking at the papers on FPGA implementation that were presented at the recent AES conference, it is enlightening to see that RC6 actually performs very well. Somehow at the conference we were left with the impression that this might be otherwise. The only area where RC6 appears not to perform as well as some of the other AES finalists is in the area of semi-custom and custom designed ASIC. At this early stage we would like to see more detailed information before jumping to conclusions. However we believe that it is in software where the performance of an algorithm matters most. In hardware design, more effort can be made on optimizing whichever of the AES finalists are chosen. Yet even at this early stage, with the first incomplete sets of information available, the performance of RC6 is good when the multiple Gigabit rate is readily achieved [49].

3 Versatility

One of the early stated aims of the AES process was that the final cipher be "simple and versatile". RC6 is clearly the finalist that most fully supports this goal.

3.1 Parameter choices

RC6 is fully parameterized; the number of encryption rounds, the size of the encryption key (not just the three must-support values of 128, 192, and 256 bits), and the block size can all be easily and readily changed. This kind of flexibility is an integral design feature. For most of the other finalists it is not at all clear how a change to the block size, or how the use of an extremely long encryption key, would be accommodated.

These could be important considerations. For some applications, a developer may wish to call on a 64-bit block cipher perhaps as a drop-in replacement to DES. With RC6 as the AES, such a variant is readily described. Not so with any of the other AES finalists. At the other extreme, it is possible that in the near future a 256-bit hash value will be preferred. RC6 is flexible enough in the algorithm specifications to give 256-bit hash values directly. Some other finalists, particularly Mars and Twofish, are so complicated that it is unclear how different block sizes would be defined. Instead more complicated constructions would be required to derive 256-bit hash values.

Another important property of having a fully-parameterized algorithm is that small-scale versions of the algorithm can be devised that retain some of the properties of the larger-scale cipher. This facilitates analysis and experimentation and helps to provide a quick and accurate security analysis of the cipher.

3.2 RC6 is well-suited to hashing

Considering the role of a block cipher for hashing focuses attention on the key schedule of a cipher. We will demonstrate that the key schedule of RC6 has some nice properties that make it suitable for hashing. One way to use a block cipher as a hash function is typically referred to as Davies-Meyer hashing [33]. In the following sections we will use this as an example that illustrates some of the potential flexibility of RC6 and its key schedule.

Davies-Meyer hashing. Suppose that we have a b-bit block cipher that uses a k-bit key. If we wish to hash a message m we divide the message into k-bit blocks, padding as necessary to complete an integral number of blocks $m_0, \ldots, m_{a-1}$. A constant b-bit initial value IV is specified. The hash output is then given by H_a where $H_0 = IV$ and for $0 \leq i \leq a-1$ we have that $H_{i+1} = \mathrm{Enc}_{m_i}(H_i) \oplus H_i$. At each iteration of the hash function k bits of the m bits of message are processed. The output is b bits in length. Denote the time to perform key setup with a k-bit key as T^k_{setup} and the time to encrypt a single b-bit block as T^b_{encrypt}. The time required to hash a message of m bits is $\lceil \frac{m}{k} \rceil \times (T^k_{\mathrm{setup}} + T^b_{\mathrm{encrypt}})$. Provided T^k_{setup} remains the same as we increase the key length k, we will improve hashing performance.

Increasing the key-length. We will first consider the Davies-Meyer construction with the key-size k set to either 128 and 256 bits while the block-size b is set to 128 bits. As a starting point the time to encrypt using Mars, RC6 and Rijndael on the Pentium Pro is given as 306, 223, and 237 cycles respectively [2] for 128-bit keys. For Serpent we refer to [45] where 900 cycles is claimed.

A rough and conservative estimate (particularly for RC6) of the factor increase in the time required for key setup when compared to the encryption of one block might be 10, 6, 1, and 2 for Mars[5], RC6[6], Rijndael, and Serpent respectively. This factor was estimated by considering several sources [22, 4, 45]. For the Mars figure we slightly decrease the derived factor to take into account the improved performance of the new keying option.

For Rijndael we observe that key setup and encryption times for 256-bit keys will be approximately 1.4 times the time obtained for 128-bit keys since extra rounds are needed. Finally for Twofish we need to refer to the designers own work [44]. The designers suggest the need to use the so-called *zero-keying* option for hashing applications and give the cycle counts of $1250 + 860$ for keying and encrypting one block on the Pentium Pro with 128-bit keys and $2000 + 1420$ for 256-bit keys.

Since the specifications for Mars and RC6 allow for keys longer than 256 bits we will then explore the full potential of the AES candidates for hashing. We

[5] This estimate agrees with documentation provided by the Mars designers [9].

[6] An assembly implementation of the RC6 key schedule requires 1100 cycles [14] giving a factor of 4.5.

will refer to this as extended Davies-Meyer techniques. The results we obtain are presented in Table 5. For Mars we take keys up to 448 bits and for RC6 we take keys up to 1024 bits. It would also be interesting to consider how the algorithms compare on platforms other than the NIST reference platform where RC6 already has a performance advantage.

	Mars	*RC6*	*Rijndael*	*Serpent*	*Twofish*
Davies-Meyer					
Key length $k = 128$, Pentium Pro *hashing rate (cycles/byte)*	210	98	30	169	132
Key length $k = 256$, Pentium Pro *hashing rate (cycles/byte)*	105	49	21	85	107
Extended Davies-Meyer techniques					
Pentium Pro hashing rate (cycles/byte)	60	13	21	85	107

Table 5. Estimated cycle counts for three different hash function constructions using the AES finalists where k denotes the key size in bits. The figures for this table were taken from the best known assembly figures for the NIST reference platform and other documentation where possible. The lower line gives the estimated cycle counts on the Pentium Pro when taking advantage of the added flexibility of the Mars and RC6 key schedules.

Discussion. Schneier et al [45] give a comparison of hashing performance. Their results are broadly in-line with the first row of entries in Table 5. However they stopped short of extending these results to a Davies-Meyer construction with a key size of 256 bits. It seems that hashing rates comparable to those obtained by today's dedicated hash functions could be possible with RC6.

Finally we note that the RC6 key schedule can take keys up to 2048 bits in length. No matter how tempting it might be to get truly exceptional performance, such very long keys would not be recommended in this role since the final few bytes of the text that would be input as key material might not receive sufficient mixing. However keys up to 1024 bits in length would be a perfectly reasonable choice. In addition, the design of the key schedule is such that even when taking a key of length 1024 bits, there is no increase in the time for key setup or encryption over that observed for 128- and 256-bit keys.

Conclusion. In this section we have considered the versatility of RC6. We see that the full parameterization of the cipher is an important and very useful attribute. It provides implementation flexibility and also has a very significant

effect on how easy it is to analyze a cipher. Furthermore, we have considered the suitability of the AES finalists to the role of hashing.

It is important in choosing the final AES algorithm that we fulfill one of the stated aims of the NIST AES process, and that is to choose an algorithm that is "simple and versatile".

4 Security

Despite the good performance of RC6, and its exceptional versatility, we believe that it is in terms of security that RC6 is the best suited finalist to become the AES.

Security is a difficult attribute to quantify. There are the headline figures in different "attacks" but how is their impact to be assessed?

The simplicity of a cipher is an essential and inseparable part of the security of a cipher. Only by making a cipher simple can cryptanalysts be challenged into looking at it. A simple cipher is one that is easily described and readily remembered. It will, as a direct result, be analyzed and scrutinized widely [6, 12, 13, 21, 32]. Not only will it receive the greatest quantity of analysis but it will also receive the most accurate analysis. When comparing the security of ciphers we must recall that the true security of a cipher depends on

- the amount of cryptanalytic scrutiny received,
- the accuracy of existing cryptanalysis,
- the ease with which verifying experiments can be conducted on a cipher,
- the amount of earlier cryptanalytic work that can be used in the assessment of the cipher,
- and, the accuracy of the designers initial estimates.

On all counts RC6 is by far the most suited finalist. A cipher we cannot understand or accurately analyze is not necessarily secure.

4.1 The state of RC6

During the design of RC6 we performed what we believe to be the most accurate assessments of the security of any of the AES finalists [12]. RC6 is not so complicated that approximating models have to be introduced to obtain results (as with MARS [9] and Twofish [44]). Instead we were able to get a surprisingly accurate assessment of the strength offered by RC6 using direct analysis[7]. In this way we were able to make a careful decision on the number of rounds for RC6 so that we delivered good performance once our security goals had been attained.

In the case of Mars and Rijndael new attacks [41, 19] have improved on the work of the designers. Yet it is a vindication of our approach that when

[7] Indeed, analysis is facilitated since it is easy to define simplified and small block-size variants of RC6 which allow for more extensive analysis and experimentation.

other techniques are applied, as was done by Knudsen and Meier [32] (and also Baudron et al. [6, 21]), they give surprisingly similar results to those provided by our own analysis. This isn't a "small margin for security". Rather it is a carefully assessed, and remarkably accurate, margin for security.

As well as being earned, some faith in a cipher can be inherited. The time for assessment of the finalists throughout the AES process has been a little less than two years. By building on the knowledge of earlier ciphers we gain insight into the security of a new cipher. It is clear that RC6 was designed in the light of experience[8] gained with RC5 [39]. And not only with regards to the structure of the round function. We decided to choose a key schedule for RC6 that was identical to that for RC5. No other AES finalist uses a key schedule that has been open to public analysis for nearly six years. Given the problems some finalists have in the key schedule, either with key separation in the case of Twofish [35] or with related-key attacks in the case of Rijndael [19], this is a very important attribute.

Kelsey et al. [31] discuss cryptanalytic progress and emphasize the need to be cautious when choosing the AES. We re-affirm this view and stress that a well-studied cipher, one that is very closely related to a previous cipher that has already received more than five years of analysis, and one that has received the most accurate security analysis, seems to be best-suited to this goal.

It is impossible to predict what new attacks will appear in the coming years. Choosing a cipher for which there has been accurate and tested analysis seems to be the best way to ensure that we satisfy even the most demanding security requirements for the lifetime of the AES and beyond.

5 Conclusions: The case for RC6

The three most important attributes of the final AES are security, performance, and versatility. With RC6 we achieve all three goals.

- Existing analysis on RC6 is not only by far the most extensive of any of the finalists, it is also the most accurate and the most detailed.
- RC6 offers exceptional performance where performance matters, and gives the best all-round suitability for many modern applications.
- RC6 is so simple that the full details of the cipher can be recalled at will. Through this simplicity we have developed a truly versatile cipher.

For these reasons we believe that RC6 is ideally suited to be the final AES.

References

1. R. Anderson, E. Biham, and L.R. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard.

[8] It would not be an idle claim to observe that RC5 is one of the most scrutinized of modern ciphers.

2. K.Aoki and H. Lipmaa. Fast implementations of the AES candidates. Proceedings of 3rd AES conference, New York, pages 106-120, April 2000.
3. H. Balakrishnan. E.E.C.S., M.I.T. Personal communication. May, 2000.
4. L.E. Bassham III. Efficiency Testing of ANSI C Implementations of Round 1 Candidate Algorithms for the Advanced Encryption Standard. August 9, 1999. National Institute of Standards and Technology. Available from `csrc.nist.gov/encryption/aes`.
5. L.E. Bassham III. Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard. National Institute of Standards and Technology. Proceedings of 3rd AES conference, New York, pages 136-148, April 2000.
6. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern and S. Vaudenay. Report on the AES candidates. Proceedings of 2nd AES conference, Rome, pages 53-67, March 1999.
7. S. Bellovin. Comments made at the rump session of 3rd AES conference, New York. April 13, 2000.
8. S. Bellovin. Posting made, April 26, 2000, to `cryptography@c2.net` and `ipsec@lists.tislabs.com`.
9. C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. Matyas, L. O'Conner, M. Peyravian, D. Safford, and N. Zunic. MARS - a candidate cipher for AES. June 10, 1998.
10. K. Claffy. Posting made, April 27, 1998, and archived at `http://www.cctec.com/maillists/nanog/historical/9804/msg00553.html`.
11. C. Clapp. Performance of AES candidates on the TriMedia VLIW Media-processor. Available from `csrc.nist.gov/encryption/aes`.
12. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. The security of RC6. Available from `www.rsasecurity.com/rsalabs/aes/`.
13. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. Improved analysis of some simplified variants of RC6. In L. Knudsen, editor, Fast Software Encryption, Lecture Notes in Computer Science Volume 1626, pages 1-15, Springer-Verlag, 1999.
14. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. Some comments on the first round AES evaluation of RC6.
    Available from `www.rsasecurity.com/rsalabs/aes/`.
15. J. Daemen and V. Rijmen. AES Proposal: Rijndael. June 11, 1998.
16. J. Dray. Report on the NIST Java AES candidate algorithm analysis. November 8, 1999. National Institute of Standards and Technology. Available from `csrc.nist.gov/encryption/aes`.
17. J. Dray. Report on the NIST Java AES candidate algorithm analysis. National Institute of Standards and Technology. Proceedings of 3rd AES conference, New York, pages 149-160, April 2000.
18. A. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists. Proceedings of 3rd AES conference, New York, pages 13-27, April 2000.
19. N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. In B. Schneier, editor, Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag. To appear.
20. K. Gaj and P. Chodowiec. Comparison of the hardware performance of the AES candidates using reconfigurable hardware. Proceedings of 3rd AES conference, New York, pages 40-54, April 2000.

21. H. Gilbert, H. Handschuh, A. Joux, and S. Vaudenay. A statistical attack on RC6. In B. Schneier, editor, Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag. To appear.
22. B. Gladman. Implementation experience with AES candidate algorithms. Proceedings of 2nd AES conference, Rome, pages 7-14, March 1999.
23. G. Hachez, F. Koeune, and J.J. Quisquater. cAESar results: Implementaion of four AES candidates on two smart cards. Proceedings of 2nd AES Candidate Conference, Rome, pages 95-108, March 1999.
24. S. Halevi. Suggested "tweaks" for the MARS cipher. Submitted to NIST at the end of Round 1 evaluation. Available via `csrc.nist.gov`.
25. T. Ichikawa, T. Kasuya, and M. Matsui. Hardware evaluation of the AES finalists. Proceedings of 3rd AES conference, New York, pages 279-285, April 2000.
26. C. Jutla. Encryption Modes with Almost Free Message Integrity. Preprint.
27. G. Keating. Performance analysis of AES candidates on the 6805 CPU core. Proceedings of 2nd AES conference, Rome, pages 109-114, March 1999. Available from `www.ozemail.com.au/ geoffk/aes-6805`.
28. J. Kelsey. Key separation in Twofish. Twofish technical report #7. Counterpane. April 7, 2000.
29. J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In B. Schneier, editor, Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag. To appear.
30. J. Kelsey and B. Schneier. MARS Attacks! Preliminary cryptanalysis of reduced-round MARS variants. Proceedings of 3rd AES conference, New York, pages 169-194, April 2000.
31. J. Kelsey, N. Ferguson, B. Schneier, and M. Stay. Cryptanalytic progress: Lessons for AES. Counterpane. May, 2000.
32. L.R. Knudsen and W. Meier. Correlations in reduced-round RC6. In B. Schneier, editor, Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag. To appear.
33. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography. 1997. CRC Press.
34. T. Messerges. Securing the AES finalists against power analysis attack. In B. Schneier, editor, Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag. To appear.
35. F. Mirza and S. Murphy. An observation on the key schedule of Twofish. Proceedings of 2nd AES Candidate Conference, Rome, pages 151-154, March 1999.
36. 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. National Institute of Standards and Technology. August 9, 1999. Available from `csrc.nist.gov/encryption/aes`.
37. O. Port. Chips for the Post-PC era. *Business Week*, Annual Special Issue, Page 96, March 27, 2000.
38. B. Preneel, V. Rijmen, and A. Bosselaers. Recent developments in the design of conventional cryptographic algorithms. In B. Preneel and V. Rijmen, editors, *State of the Art in Applied Cryptography*, Lecture Notes in Computer Science Volume 1528, pages 105-130, Springer-Verlag, 1998.
39. R.L. Rivest. The RC5 encryption algorithm. In B. Preneel, editor, Fast Software Encryption, Lecture Notes in Computer Science Volume 1008, pages 86-96, Springer-Verlag, 1995. Available at `theory.lcs.mit.edu:80/~rivest/`.

40. R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 Block Cipher. v1.1, August 20, 1998. Available at `www.rsasecurity.com/rsalabs/aes/`
41. M.J.B. Robshaw and Y.L. Yin. Potential flaws in the linear cryptanalysis of MARS. Extended abstract available at `www.rsasecurity.com/rsalabs/aes/`.
42. A. Satoh, N. Ooba, K. Takano, and E. D.'Avignon. High-speed Mars hardware. Proceedings of 3rd AES conference, New York, pages 305-316, April 2000.
43. B. Schneier. Crypto-gram. Counterpane. April 15, 2000.
44. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit Block Cipher. 15 June, 1998.
45. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Performance comparison of the AES submissions. Proceedings of 2nd AES Candidate Conference, Rome, pages 15-34, March 1999.
46. B. Schneier and D. Whiting. A performance comparison of the five AES finalists. Proceedings of 3rd AES conference, New York. Pages 123-135. April 2000.
47. A. Sterbenz and P. Lipp. Performance of the AES candidate algorithms in Java. Proceedings of 3rd AES conference, New York. Pages 161-165. April 2000.
48. N. Weaver and J. Wawrzynek. A comparison of the AES candidates amenability to FPGA implementation. Proceedings of 3rd AES conference, New York. Pages 28-39. April 2000.
49. B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke. Hardware performance simulations of Round 2 Advanced Encryption Standard algorithms. Proceedings of 3rd AES conference, New York, pages 286-304. April 2000.
50. R. Weiss and N. Binkert. A comparison of AES candidates on the Alpha 21264. Proceedings of 3rd AES conference, New York. Pages 75-81. April 2000.
51. J. Worley, B. Worley, T. Chrstian, and C. Worley. AES finalists on PA-RISC and IA-64: Implementations & Performance. Proceedings of 3rd AES conference, New York. Pages 57-74. April 2000.
52. T. Wollinger, M. Wang, J. Guajardo, and C. Paar. How well are high-end DSPs suited for the AES algorithms? Proceedings of 3rd AES conference, New York. Pages 94-105. April 2000.
53. D. Young. Personal communication. RSA Security, Inc. May 2000.

This article was processed using the LaTeX macro package with LLNCS style