Session 6:

"AES Issues" Panel

AES and Future Resiliency: More Thoughts And Questions

By Don B. Johnson
djohnson@certicom.com
March 10, 2000

Introduction

In a paper submitted previously, the author argued that a new AES evaluation criterion of future resiliency be added, defined as the ability to respond to the uncertain future and that this criterion could best be met by NIST selecting multiple disparate AES winners. This paper continues that discussion by providing more rationale. It also asks some questions and explores possible outcomes of the AES process.

Summary From Previous Paper

The author's previous paper closed with this summary:
"NIST should carefully examine the various classification schemes that have been made and endeavor to choose the AES second round finalist candidates considering that it is a worthwhile goal to try to ensure that differing design approaches are included. This is because of reasons of future resiliency, extending cryptographic knowledge, Super AES, crypto toolbox philosophy, possible patent complications, target diffusion, avoidance of artificial tiebreakers, recognition of the problem being multidimensional with imperfect information, and the constraints of other standards organizations. That is, in selecting the handful of AES second round finalists, disparity of design approaches is to be desired over conformity."

For further explanation of these rationales, see the paper at the NIST AES website at www.nist.gov/aes. These rationales continue to be valid in the discussion regarding whether there should be multiple AES winners or not.

NIST is to be congratulated for their selection of AES finalists as they do represent a disparate selection from among the submitted AES candidates. Furthermore, the inventors of each finalist algorithm represent a significant portion of the skills in the cryptographic community. As the AES winner(s) must give a royalty-free license (if the algorithm is patented), perhaps the main rationale to participate in

the AES process is the recognition one receives, with the winner(s) getting major “bragging rights.” Another way to look at the NIST AES finalist selection is that NIST has put “five cats in a bag” to see who survives as each submitting group is highly motivated to find chinks in the armor of the other AES finalist algorithms. Better to find out now rather than later.

I) Additional Rationale for Multiple Algorithms

Space Probe Scenario

A reason to consider multiple winners is that sometimes one needs to use hardware for performance reasons, but the hardware is difficult or impossible to change once deployed. Consider a commercial space probe [JC]. Once it arrives at its destination, it must be essentially self-sufficient. Calling it back is out of the question. However, backup circuitry is a normal part of its design and this flexibility could be extended to include a backup symmetric cryptographic algorithm. As these types of projects might take years or decades, such an algorithm backup is simply prudent.

AES Selection Time

Another factor that should be considered by NIST is the amount of time that was taken by the AES process. If a sole AES winner were to prove unfortunate for some reason, then it could take many years to determine a substitute. It has been said that three months is considered an Internet year. The time needed to do another AES process may not meet the requirements of the market.

Infrastructure Overoptimization

As we saw with the deployment of DES, the selection of one algorithm by NIST meant that best-practices resulted in the use of that **one** algorithm. For much of the life of DES, there was no pressing need for vendors to try to design systems to support multiple symmetric cryptographic algorithms, DES was it. With DES the only choice, this simplified things for a vendor. However, we see today that this simplification resulted in a deployed infrastructure where there are concerns that some portions are now vulnerable to a determined attack.

Einstein is reputed to have said, “One should try to make things as simple as possible, but no simpler.” Even the selection by NIST of as few as two winners will mean that vendors will need to design in flexibility of algorithm choice in some products and provide for the possibility of algorithm replacement in others, rather than overoptimize as was done in the case of DES.

NIST as AES Architect

NIST is overseeing the AES process. As such, NIST is the architect of the AES process, that is, it is creating the AES design architecture. There are two fundamental responsibilities of an architect, as follows:

1) Specify **enough** detail to allow others to proceed.
2) Know what **not to specify** to allow creativity and flexibility in others.

In this AES architect role, NIST should follow the general principle of “If in doubt, don’t.” NIST/NSA can and should make “apparent health” statements on the security of the AES finalists. NIST can and should make decisions about which AES finalist algorithms are suitable for government use, using whatever additional criteria (if any) besides security that NIST deems appropriate. This is ALL that NIST should try to do. NIST should resist the temptation to try to solve potential challenges resulting from the existence of multiple algorithms, such as the need to negotiate algorithms or the need for a vendor or market segment to select the most appropriate algorithm.

NIST or the Marketplace?

Asking NIST to select a sole AES winner means that one believes this decision is appropriate for top-down decision-making, as in a command economy or an army. A top-down methodology is appropriate when any decision is better than no decision (e.g., traffic lights) or when a decision must be made quickly (e.g., a battle). However, simply as a matter of information flow, all the relevant information cannot be expected to be available to the responsible top-level decisionmaker. The marketplace (bottom-up decision-making) has been shown to be much more responsive and adaptable than a command economy. This is because each economic entity or group of entities makes decisions based on its own information and needs.

So one question that NIST needs to ask itself is does it see the AES process (that is, the development of commercially-appropriate symmetric cipher or ciphers) as needing a top-down decision to be made or does it believe that the marketplace is the most appropriate place for this decision to be made. The marketplace has a way of determining what is appropriate; if there is truly one finalist that is superior in many ways, it does not need NIST's selection of it as the winner to emerge as the winner in the marketplace. However, there is a real concern that NIST could make a suboptimal choice due to insufficient information. In this case, "hands off" is the wisest course of action.

NIST needs to resist the temptation to make a decision in an area beyond their (or anyone's) competence. The round 2 discussion issues asking questions about how to assess speed versus security margin, need for low-end flexibility, and hardware versus software performance indicate that NIST recognizes its lack of certainty in these areas. This is fundamentally because there are no obviously single correct answers to these questions. Different applications may require different answers to these questions. NIST should make a virtue of its (really, everyone's) ignorance and not attempt to decide these unanswerable questions one way or the other, but let others make each decision that is most appropriate for them.

Bias?

It may not be politically correct to say so, but NIST should understand that any counsel given it might be biased; this might be especially true of counsel from submitters of algorithms. This is not necessarily a bad thing, the submitters of the AES finalists have very high cryptographic skills and it is certain that the submitters made their decisions after thinking long and hard about the problem. It is just that each submitter naturally thinks their beliefs are correct.

For example, it would be no surprise that a designer of a very flexible algorithm might think that flexibility is an important AES criterion. That is likely one of the reasons the submitted algorithm was made flexible in the first place, so that it would have an advantage when compared with other AES candidates.

The point is that if NIST were to announce they are seeking a single winner then this (in turn) results in a ranking of finalists, just as identifying any other AES criterion as critically important would also potentially rank the finalists. However, note that if an algorithm is truly more flexible than another, it still stands a greater chance of being used in the "marketplace" selection process mentioned above. That is, any advantages of an algorithm remain advantages; by selecting multiple winners and relying on the marketplace, NIST is not required to try to determine which advantages are more important than others.

II) Some Questions

Quantum Computers

One big question regarding the future is whether or not quantum computers are feasible and if they are, what effect they will have on cryptography. An arbitrary bitsize quantum computer (assuming it can be built) allows a square root attack on a symmetric cipher. The possibility of this provides some justification for the larger AES keysizes; a 256-bit symmetric cipher would take 2**128 quantum operations to exhaust the key space.

However, an interesting question is whether there is some limit in practice to the number of bits of a quantum computer. Many researchers suspect this is the case, that quantum decoherence will prove insurmountable for some number of quantum bits.

In terms of AES this question becomes: if one can only build an x-bit quantum computer, how much does this help in attacking each AES finalist? As all block ciphers are composed of smaller chunks, how might these chunks interact with a quantum computer? This possibility can be termed a partial quantum attack. And of course, an adversary could construct many quantum computers to run the attack in parallel, assuming this would help. So the question is: "How does a parallel partial quantum computer affect the ability to attack the AES finalists?"

As an example, DES is composed of a 56-bit key. A 56-bit quantum computer should be able to attack the DES. However, the DES design is such that each of the sixteen rounds uses a 48-bit key. This

suggests the possibility that a 48-bit quantum computer might somehow be able to be used to successfully attack the DES. The question of how the AES finalists stack up in relation to parallel partial quantum computers is a critical question to be answered. NIST should step up to this analysis if it is not forthcoming from the research community. No final AES decision should be made without some exploration of the expected effects of this possibility.

Random Cipher

It is clear that a random cipher for a certain blocksize is the unrealizable ideal. This is a cipher that selects a random choice for the output block for each input block, the key providing an index into a set of random selections. There is no structure that is able to be attacked by an adversary. The best attack is key exhaustion, which is the goal of any symmetric cipher. It is also clear that such a ideal block cipher is totally impractical as the space needed is totally infeasible. However, one would like any particular block cipher to "appear" to be ideal to an adversary. That is, even though the structure is known to an adversary, this structure does not allow any shortcuts to be made. A critical question is whether a AES finalist appears "random." There are many established randomness tests. Any deviation from random is a cause for concern.

Another related important question is at what point do degenerate forms of a finalist not appear random. For example, a finalist may have 20 rounds. It is important to know if the output after 4 rounds appears random or if it takes 8 or even 16 rounds. This is important as it gives an indication of the margin of safety built into the cipher. It is obvious that a round of cipher A cannot be considered equivalent to a round of cipher B but this type of analysis allows one to at least map some internals of one algorithm to another for comparison purposes.

Knowing what to do with this analysis is more problematical. Regardless, this is an important data point. If I know that cipher A is essentially as fast as cipher B, but that cipher A results in random-appearing output after 5 of 16 rounds and cipher B results in random-appearing output after 8 of 12 rounds, then cipher A may be the more conservative choice in some sense. But NIST should be wary of this analysis, one can simply add more rounds at a performance cost.

Should a cipher be rewarded (or penalized) for minimizing overhead? Should a cipher be rewarded (or penalized) if it has “more” rounds? This means that (apparent) security and performance are very closely tied together.

Combined Attacks

In the real world, the adversary is able to combine the effects of various attacks. Even if each attack results in only a relatively small advantage that is not relevant when considered by itself, a combination of attacks may accumulate to result in a feasible attack. For this reason, any discovered theoretical advantage for an adversary attacking an AES finalist (no matter how apparently small) is a concern.

III) Thoughts on the AES Finalists

Following are some thoughts on the AES finalists. It should be recognized that these ideas are tentative and subject to improvement and correction. Of course, the detection of any security flaw in a finalist would have a major impact. Each finalist algorithm can be seen as a statement by the designers regarding not only one way to solve the various tradeoffs of the AES puzzle, but also as how the designers see the future. It is hoped that these thoughts on the finalists are used by NIST in the spirit in which they are given, as food for thought.

MARS

MARS was designed with some thought to try to avoid potential future attacks, especially in its heterogeneous structure, a keyed-core surrounded by unkeyed forwards- and backwards-mixing functions. The unkeyed mixing functions cost time and space, but their inclusion seemed prudent to the designers and worth the cost. The core “mixing” function uses addition, multiplication, fixed and data-dependent rotations, and an S-Box (straightforward substitution cipher). The designers responded to criticism to improve the performance of MARS by using the “tweak” allowed by NIST.

From a perspective on the future, the designers of MARS believed the best way to handle uncertainty was to use many different techniques using a cost/benefit analysis. The MARS design is the

most different of the Feistel cipher finalists. Another way to look at MARS is that IBM is a large organization which had many people with good ideas trying to get them incorporated into the IBM submission. This can be seen in the number of authors of the MARS paper.

From a perspective of future resiliency, the inventors of MARS thought that a heterogeneous structure was important.

RC6

RC6 was built from a heritage of RC5 and was designed to be fast and simple to describe. The core ideas of RC6 came from RC5, which was designed by one person, as such it represents a unity of design approach. In many scenarios, RC6 is the fastest AES finalist. The pseudocode for RC6 is very straightforward with basic operations defined on 32-bit words; the RC6 pseudocode is the shortest of all finalists. It uses addition, multiplication, data-dependent rotations and substitution to do the cryptographic "mixing." RC6 can be seen as an example of building a performance-optimized cipher on the idea of data-dependent rotations. The challenge for the designers of RC6 is to show that their design is not **too simple**. For example, comparing RC6 to MARS, MARS adds more complexity to its specification to try to provide more mixing.

Indeed, the "Correlations in RC6" paper by Knudsen and Meier (available at www.nist.gov/aes) indicate that reduced rounds of RC6 do not appear random. The observation by Saarinen in the NIST RC6 forum on finding "almost equivalent" keys in RC6 suggests other possible concerns. These ideas hint that RC6 may be on the edge of security.

From a future resiliency perspective, the designers of RC6 believed that parameterization was paramount. In this way, if a certain number of rounds was found to be weak, this number could be adjusted upwards.

Rijndael

Rijndael does not use a Feistel structure, rather it uses a matrix structure where the cryptographic mixing involves byte substitution, row shifting and column multiplication. Rijndael has the most

different structure when compared with the other AES finalists. It can be implemented using byte operations and is therefore very flexible.

From a future resiliency perspective, the designers of Rijndael were willing to go in new directions and wanted high flexibility in implementation.

Serpent

Serpent is a conservative design and deliberately tries to build on the vast amount of information relating to DES. Serpent is also the **slowest** of the five AES finalists on most platforms. Being the slowest, the challenge for the designers of Serpent is to try to show how the other finalist algorithms cut corners in ways that Serpent did not (that is, the additional performance cost should be justified). For example, suppose that NIST gave all five AES finalists "certificates of apparent security," it is not clear what symmetric algorithm niche would best be filled by use of Serpent, as opposed to one of the other finalists. Of course, a specific implementation might find that Serpent is the fastest method, if the instructions it uses are fast and the instructions that other methods use are slow.

The designer's of Serpent have presented an "equivalent rounds" analysis of the AES candidates and tried to show how Serpent uses more rounds than might be thought needed as a safety margin. Yet the designers did not officially change the specification of Serpent (even though they knew that there were many other faster AES candidates) so they must believe they have good reasons for designing it as they did. Serpent and RC6 appear to have opposite design philosophies in this area of tradeoff between security margin and performance.

From a future resiliency perspective, the designers of Serpent decided to use more rounds and affect performance to try to achieve a higher security margin of safety. This means Serpent may have some performance concerns, at least when compared with the alternatives.

Twofish

Twofish is a byte-oriented Feistel cipher with great flexibility of implementation, allowing a wide range of time/space tradeoffs. Many research reports have been written on various aspects of Twofish, which give confidence in its security. There was also a cost/benefit analysis done by the designers to decide which operations to use.

From a future resiliency perspective, Twofish's goals were security and implementation flexibility.

IV) Possible Outcomes

Does NIST want the fastest cipher? ... the cipher with the largest safety margin? ... the cipher with the most flexibility? ... the cipher with the most disparate instructions? ... the most Feistel-like cipher? ... the cipher with the most disparate design? Single or multiple winners? ... some other criteria? The point is that different answers to each question can lead to a different ordering of the AES finalists. Furthermore, any selection by NIST indicates in a backwards fashion which criteria they decided was more important than others. As one example, comparing MARS and Serpent, are more rounds or different rounds the better way to address having a sufficient safety margin? As another example, comparing MARS and RC6, are many different ideas or unity of design the better way to design a cipher?

The problem for NIST is not that there are no answers, it is that there are **too many** rational answers. Barring a security flaw, any of the AES finalists could be justified as being the sole winner simply by NIST adopting the corresponding design philosophy behind the winner as its own. NIST should resist any temptation to do this. Rather, as each submission has a different design philosophy, NIST should accept the implication that there was no obvious single all-around best solution. NIST should accept this implicit "higher-level" statement from the submitters and agree with them (as a group) that there is no single all-around best answer.

Strictly speaking, NIST's AES mandate is to select a winner or winners that is/are suitable for use by the US Federal government to protect sensitive non-classified data. Following the historical pattern of DES, it is also expected that NIST/NSA will issue a statement that the winner(s) is/are suitable for the intended purpose. Historically, it

was this endorsement that gave confidence to other groups, such as the American Bankers Association, to also endorse DES, which in turn led to DES becoming the most-deployed commercial cryptographic algorithm.

Now, some 25 years after DES, we see the endorsement by NIST of 3 families of asymmetric cryptographic algorithms in the revision of FIPS 186; namely, those based on the difficulty of integer factorization, the normal discrete logarithm, and the elliptic curve discrete logarithm. This allows the advantages of each method to determine the way asymmetric cryptography rolls out in the future. That is, NIST recognizes that there are multiple answers to the asymmetric cryptography question.

This author hopes that similar rationale will prevail among the NIST AES selection team regarding the symmetric cryptography question. While this author believes that the best outcome of the AES process is a handful of winners which lets the marketplace determine each algorithm's niche, it is realized that not all others share this opinion.

Ranking?

NIST should realize its decision is not restricted between having one AES winner and having multiple winners, it could also decide to have a ranking among multiple winners. As an example, NIST might specify that algorithm A is the primary winner and algorithm B is the backup. In this example, an implementation would be expected to either implement algorithm A (if resources are constrained) or both algorithms A and B (if resources are available). This seems much preferable to declaring a single AES winner, although inferior to selecting multiple co-equal winners.

Multiple Endorsement?

Another alternative is that regardless whether one or multiple winners (ranked or not) are selected by NIST for use by the US Federal government, NIST/NSA could issue health statements that certain finalists meet their intended security goals. This would at least allow other standards bodies to negotiate with increased confidence for the rights to an endorsed algorithm, if that algorithm better met their needs. For example, NIST might say that algorithm A wins (for US

Federal government use), but also issue a NIST/NSA report that algorithms A, B, and C meet their intended security goals.

Just to be clear on this point, if all five AES finalists have no known security weaknesses, then all five finalists should be giving a "certificate of health" regardless of the decision regarding the number or specific selection of AES winner(s) for approval for US Federal government use.

Acknowledgements

The author wishes to thank Certicom for providing the environment in which to write this paper.

Reference

[JC] Jerry Coffin in a post on sci.crypt on AES mentioned that satellites were an example where hardware was infeasible to change once deployed and saw this as a reason to have multiple winners.

Biography

Don B. Johnson is Director of Cryptographic Standards for Certicom, is a member of Certicom Research, and sits on the Advisory Board of the Standards for Efficient Cryptography Group (SECG). He participates in ISO SC27, ANSI X9, IEEE P1363 and other standards bodies. He has over 40 patents and patent applications in the area of cryptography. He was the editor of the X9.62 Elliptic Curve Digital Signature Algorithm (ECDSA) standard.

The Effects of Multiple Algorithms in the Advanced Encryption Standard

Ian Harvey, nCipher Corporation Ltd, 4th January 2000

Abstract

This paper presents a discussion of the issues relating to the selection of encryption algorithms in practical situations. An AES standard which recommends multiple algorithms in a variety of ways is discussed, and it is shown that this can present an overall advantage.

Introduction

The Advanced Encryption Standard aims to become the first choice for most situations requiring a block cipher. To do this it has to satisfy a wide variety of requirements for - amongst other things - security, performance, and resource constraints.

Each of the current five candidate algorithms for AES satisfies a different balance of these constraints; the 'best' algorithm depends on circumstances, which are impossible to know beforehand. Furthermore, one of the principal requirements - that of security of the algorithm - cannot easily be measured; subjective judgements therefore must be made (based, for instance, on notions of 'safety margin' or 'conservative design') which may prove to be inaccurate or irrelevant in years to come.

In the absence of a precise definition of what constitutes the 'best' algorithm, or accurate means even to measure this, any choice of algorithm is somewhat arbitrary. In security terms this seems needlessly risky, and it has been suggested that avoiding the need for a single final algorithm would have advantages.

In the sections below, we discuss the factors which affect algorithm choice, and then examine the effect on these of an AES which offers multiple algorithms.

Factors in algorithm choice

Many factors may be involved in the selection of an algorithm. In most cases, one single factor is overwhelmingly the most important, and often some are of little or no importance.

Aside from performance issues, which are discussed later, criteria for algorithm selection include:

- **Security against theoretical attacks**

The reputation of an algorithm is frequently a major factor in its selection, both in terms of the design of the algorithm, and the extent to which it has been studied for cryptanalysis. A theoretical attack does not have to become practical before the cipher is rendered commercially unusable (for instance, liability insurance on a system using it may become void).

It is to be expected that no candidate with a known theoretical weakness will be given final recommendation within AES. Furthermore, the effect of the AES 'brand name' will

be to concentrate research into the selected algorithms; this will (all being well) improve their reputation as time passes.

- **Security of implementations**

Some of the most practicable attacks of recent years have been directed against particular implementations of algorithms, rather than their theoretical definitions. These include timing attacks, power-analysis attacks, and fault-induction attacks.

In general, defending against these attacks is done when the implementation is designed, using a range of proprietary techniques. Some algorithms may have features which make them particularly difficult to defend, but it is generally not possible simply to define a 'good' feature set. Selection of an algorithm to resist implementation attacks can often only be done when the threat model has been decided, and little generalised guidance can be given.

- **Cost of implementation**

The effort required to correctly implement a given algorithm is frequently a major issue. For software implementations, the factors which affect this include:
- availability of reference implementations in a given language
- ease of understanding and adapting the reference implementation
- complexity of any optimisations required for optimum performance, and
- the ease and completeness of correctness testing.

For hardware implementation, important factors are:
- the complexity of the cipher
- the clarity of the cipher's given description
- availability of test vectors sufficient to give complete coverage.

Many developers will want to choose AES on the grounds of easy access to good reference material.

- **Architectural implications**

The precise functional 'shape' of an algorithm will have an impact on the way systems and protocols are designed to accommodate the algorithm. The block size and key size are the principal parameters, and fortunately all AES candidates are required to be compatible here.

However, additional parameters or features offered by particular algorithms - variable number of rounds, keys of other than standard length - may create additional complexities. Where it is not possible to adapt existing protocols to deal with these parameters, behaviour is often implied. This can be a major cause of interoperability problems - something which must not be allowed to bedevil the AES.

- **Legal issues**

Patent, copyright, and export-control issues affect no area of computing more than cryptography. Commercial developers will accept some licensing costs, but only up to a point - many of the potentially superior alternatives to DES (e.g. IDEA or RC5) have not been deployed widely, mainly for cost reasons. Free software developers are often unable to accept any restrictions on algorithm use.

One of the goals of the AES process is to produce a cipher which can be deployed universally, and a leading reason for choosing it will be freedom from legal impediments.

Performance issues

Every situation has its own unique performance criteria, which are invariably a trade-off between system requirements and speed, given 'enough' security. There are three main categories:

- **Best ideal-case speed**

The highest bit rate is required, irrespective of implementation complexity. The platform for deployment can be chosen (or at least unsuitable ones eliminated). Typically this means an algorithm which does well when hand-optimised in assembler for a modern processor, or can use parallelism in a large ASIC.

This type of performance is required by high-end hardware manufacturers, software developers who choose to target few platforms, and users who can choose platforms for best performance.

- **Best worst-case speed**

An acceptable bit rate is required, on a wide variety of platforms, or on a relatively non-standard platform. There should be no platforms on which speed is significantly lower than an alternative algorithm.

This type of performance is required by software developers who target a broad range of platforms, and is often associated with good speed available from a portable C implementation. It is also of significance to manufacturers (of e.g. embedded systems) whose choice of platform determined by other factors and cryptography is a secondary consideration.

- **Minimum implementation size**

Bit rate is not important, but constraints are placed on the resources required: gate count, code size or table size.

This type of performance is required by manufacturers of embedded systems for mass deployment, where unit hardware cost is critical. However, this group typically has much less need of interoperability outside the embedded application. The main reason to choose AES in this case will be the brand-name security.

It should be noted that performance and available resources will increase dramatically over time, but resistance to attacks will decrease. A standard intended for the long term should favour security over performance or resource requirements.

Approaches towards multiple algorithms

An Advanced Encryption Standard may be proposed which recommends more than one algorithm. There are a number of ways in which this might be done.

Multiple algorithms may be made optional; they must however be specified in such a way that any conforming AES implementation can interoperate with any other.

In some situations, both encryption and decryption can be controlled by one party, but in others they are controlled by separate parties and the choice of algorithm must be a mutually acceptable one. Any two such implementations must therefore share an algorithm, and the AES recommendations must guarantee this. The following approaches will be suggested:

A. All AES implementations must include all algorithms.

B. All AES implementations must include one primary algorithm, and a choice of secondary algorithms (possibly also ranked in order). Implementations will include secondary algorithms if it is to their advantage.

C. Given a set of N algorithms, an AES implementation must include at least N/2+1 algorithms from that set - this ensures any two implementations have at least one algorithm in common. Implementations will choose the subset of algorithms that best fulfils their requirements.

Theoretical Security / Implementation Security

Properly managed, multiple algorithm choice should enhance security. Should one algorithm fall to cryptanalysis, a second choice will already be available to provide backup. Also, given a choice, an developer will be able to pick the algorithm which best resists implementation attacks in the available technology.

Improperly managed, multiple algorithm choice will detract from security. If the choice of algorithm can be subverted in a given protocol, an attacker will be able to pick the easiest target.

Approach A is the most robust; two communicating parties can negotiate the 'strongest' algorithm and it will be used. Should an algorithm be broken, it is simply removed and the next best is selected.

Approach B allows implementers to choose to implement the secondary algorithms if they require a fallback. If secondary algorithms are broken first, all systems can revert to the primary algorithm, but if this is broken, some systems may not have an alternative. This scheme is therefore as resilient as approach A, except where systems omit the secondary algorithms due to cost considerations. In this approach, it would be prudent to choose a primary algorithm with a good security 'safety margin'.

Approach C, would in theory allow implementers to implement their choice of the 'strongest' algorithms, and two communicating parties would agree on at least one they considered secure. However, should *any* algorithm subsequently be broken, some combinations of communicating parties will be left unable to communicate. Also, this relies on the implementers holding opinions about algorithm security, which is contrary to the spirit of a security standard.

In practice, this makes approach C less secure than a single-algorithm selection.

The beneficial effect of an Advanced Encryption Standard concentrating cryptanalytic efforts will be diluted if too many algorithms are chosen. Approach B will present a primary target for research, and is best in this regard.

Cost of implementation

Any approach that mandates more than one algorithm to be implemented will raise the development costs proportionally. Approach A particularly, and to a lesser extent C, have the most impact. Where development costs are the overriding concern, approach B is as good as a single-algorithm selection.

The cost of implementation can be reduced dramatically, however, given good reference materials to accompany the standard. It is to be hoped that the various software implementations made available during the selection process will also be available to accompany the standard itself. This will strongly reduce the cost of producing a correct implementation.

It may be necessary to rewrite the descriptions of one or more algorithms to use a consistent set of terminology - particularly, for instance, with respect to bit-numbering and byte-ordering conventions.

Architectural implications

Multiple algorithms, and the process required to select one, will undoubtedly add to the architectural changes required for the new standard. In many situations, where a negotiation of cipher suite is already part of the protocol, this will have minimal additional impact.

Approaches A and B allow the selection to be made fixed, but approach C necessitates some form of negotiation, and this may be impossible in some circumstances.

The issue of additional algorithm parameters needs careful consideration. The table below gives some potential variation in parameters for each of the current AES candidates:

Cipher	Variations
MARS	Key size 4-39 32-bit words
RC6	Word size *w* , no. of rounds *r*, key size 0-255 bytes
Rijndael	Block length of 128, 192 or 256 bits
Serpent	Key size 0..256 bits
Twofish	Key size 0..32 bytes

All block ciphers can accommodate a 128-bit block, and 128, 192 and 256-bit keys as per the AES requirements, but beyond this the functionality differs substantially. In fact no two candidates have exactly the same set of allowed keys. It is poor software engineering practice to expose this to the user of the cipher; any variant in algorithm should *exactly* match the functional interface of the other, including rejecting the same set of invalid keys.

Similarly, if any variations in the number of rounds is proposed to allow a speed/security trade-off to be chosen by the user, it is poor design to let the user

choose the number of rounds directly. Apart from the dangerous possibility of a round-by-round attack, it requires the user to know 'good' and 'bad' values for each algorithm. An acceptable solution would be to allow, say, three security levels - 'minimum', 'medium', and 'maximum', which is translated to a number of rounds appropriate to the algorithm in use.

Any AES which recommends more than one algorithm must address these issues, to remove ambiguity and promote interoperability.

Legal Issues

Clearly, multiple algorithms may increase the legal complications for a developer. In an ideal case, *any* algorithm offered as an option in the final AES will be free for use without restriction. As a minimum, sufficient algorithms should be available to construct a conforming implementation, without any patent or similar restriction.

Approach A requires all algorithms to have no restrictions; approach C requires the majority to have no restrictions, and approach B requires the primary algorithm alone to have no restrictions.

Performance - ideal case

Multiple algorithms give the best opportunity to maximise absolute speed, especially as evolving technology changes the balance between operations in different algorithms.

Approach A is good for this; the communicating parties will negotiate the fastest algorithm, and this is guaranteed to be available.

Approach B has some merit; the primary algorithm may not be the fastest on the chosen platform, but the implementer can add the secondary algorithms if they improve speed. In the ideal case, both communicating parties will do this and speed is maximised.

Approach C also allows implementers to choose the subset of algorithms which are most efficient on the chosen platform. Two communicating parties should then be able to pick their mutually fastest choice.

Performance - best worst-case

This benefits greatly from a choice of algorithms. Most worst-cases will be a particular feature of an algorithm which behaves poorly on a particular platform, and often any alternative will help.

The same strategy for choosing algorithms can be adopted as in the 'ideal case' scenario, with similar results. The worst case in Approach B is when the primary algorithm has poor performance on a given platform, and one of the communicating parties does not support any secondary algorithms. This is still an improvement, however, because it will only occur in those few cases where there is an overwhelming need for cost saving.

Minimum size requirements

This is impeded by the requirement for multiple algorithms; any standard which mandates more than one to be implemented will severely impact costs for low-end system developers.

To a certain extent this can be mitigated where two algorithms share common large functional blocks, or memory requirements which can be overlaid. To demonstrate this, the table below summarises the major functional requirements for each AES candidate. For comparison, triple-DES is also listed.

The table sizes given are 'minimum' requirements, with a 128-bit key, and will not be for the most efficient possible implementation; the RAM size given does not include that required for working purposes. The ROM size may be misleading as it does not include code size, which is in some cases traded off against lookup table size.

Candidate	ALU Operations					Table size (bytes)	
	logic / fixed shift	add/sub	data-dep shift	GF (2^P) ops	mult	ROM (S-boxes, etc.)	RAM (key schedule, etc.)
MARS	X	X	X		X	2048	160
RC6	X	X	X		X	none	176
Rijndael	X			X		512	16
Serpent	X					128	32
Twofish	X	X		X		64	24
Triple-DES	X					256	24

So it can be seen, for instance, that adding RC6 to a chip designed to implement MARS would have relatively little impact, but adding it to one optimised for Rijndael might be difficult.

Approach A is the worst of all worlds for minimum-size implementations. Approach C is better (only the smallest algorithms should be selected) but would still be typically double the best-case cost. Approach B allows very resource-limited implementations to implement solely the primary algorithm, and is as good as the single-algorithm case.

Summary of results

The effect of the various possibilities for a multiple-algorithm standard can be summarised in the table below, where "++" indicates the most positive impact, "0" indicates no impact compared to a single-algorithm standard, and "--" is the most negative impact.

Category	A	B	C	Notes
Security	++	+	--	1
Impl. Cost	--	0	-	2
Architecture	-	-	--	2
Legal issues	--	0	-	2
Best-case speed	++	++	++	
Worst-case speed	++	+	++	
Minimum size	--	0	-	

Notes:

1. Based on the ability of the standard to continue given failure of a cipher.
2. Can be mitigated by a good standardisation process.

Conclusions

A number of approaches to specifying multiple algorithms have been presented. This suggests that approach B - to specify a required 'primary' algorithm and one or more optional 'secondary' algorithms - has advantages over other approaches, and allows potential speed and security improvements over a single algorithm selection.

This approach means that outright performance can be eliminated from the criteria for primary algorithm selection - this can be left for the secondary algorithm. Security (or in practice, safety margin and conservative design) should be the primary algorithm's main requirement, followed by a modest resource requirement for minimum-size implementations.

Similarly, resource requirements can be ignored when making the secondary algorithm selection; implementers seeking a lowest-cost solution can simply omit these. An algorithm more aggressively optimised for performance is ideal here.

Provided that the legal and functional differences between the algorithms are mitigated by a well-written standard, there is no reason that this approach should not offer the best of all worlds.

Session 7:

"ASIC Evaluations / Individual Algorithm Testing"

Hardware Evaluation of the AES Finalists

Tetsuya ICHIKAWA* Tomomi KASUYA** Mitsuru MATSUI**

* Kamakura Office, Mitsubishi Electric Engineering Company Limited
ichikawa@harriet.mee-unet.ocn.ne.jp
** Information Technology R&D Center, Mitsubishi Electric Corporation
kasuya@iss.isl.melco.co.jp, matsui@iss.isl.melco.co.jp

1. Introduction

This report describes our evaluation results of implementing hardware of the AES finalists, concentrating on 128-bit key version, using Mitsubishi Electric's 0.35 micron CMOS ASIC design library. Our goal is to estimate the "critical path length" of data encryption /decryption logic and key setup time of key scheduling logic for each algorithm, which corresponds to the fastest possible encryption speed in feedback modes of operation such as CBC etc. To achieve this, we wrote fully loop-unrolled codes in Verilog-HDL language without introducing pipeline structure that blocks the feedback.

We first tried to investigate the evaluation environments to be used in NSA, especially the hardware design library, since NSA is expected to join the Round Two hardware analysis as has been shown in the NIST AES homepage [NIST (1998)]. However, after communicating NIST and MOSIS, we found that the library is an internal 0.5 micron standard cell library that is not available outside NSA, and a non-proprietary version of the library has not been developed. We therefore decided to analyze the AES finalists using Mitsubishi Electric's CMOS ASIC design library, whose information is publicly available in [MITSUBISHI (1997)].

Our simulation results show that Rjindael is the fastest as expected and it is even faster than DES, and Serpent is the next. Twofish, Mars and RC6 are slower than Triple-DES. We should note that since we used a general ECA (embedded cell array) library without applying special performance optimization techniques, these algorithms that heavily use arithmetic operations could be much faster if we introduce more expensive semi- or full-custom designs. However our analysis also indicates that even such designs are not expected to give a significant impact to change the ranking of the critical path length.

2. The AES Finalists

NIST announced the five AES finalists, in August 1999. This section briefly summarizes these algorithms, mainly data encryption operations, from hardware viewpoint.

2.1 Mars

Mars supports 128-bit blocks and a variable key size from 128 bits to 448 bits. It is designed to take advantage of the powerful operations supported on today's computers [Burwick et. al. (1999)].

The encryption part of Mars, which is composed of four kinds of round functions, is performed as follows. We have also listed major components that have an impact in hardware performance.

-The initial key addition
  4 additions mod 2^{32}.
-The unkeyed forward mixing (8 rounds)
  2 additions mod 2^{32}, and 4 look-up tables with 8bit-input/32bit-output.
-The keyed forward transformation (8 rounds)
  6 additions and 2 multiplications mod 2^{32}, and 4 data-dependent rotations.
-The keyed backwards transformation (8 rounds)

6 additions and 2 multiplications mod 2^{32},
and 4 data-dependent rotations.
-The unkeyed backwards mixing (8 rounds)
2 subtractions mod 2^{32}, and 4 look-up tables with 8bit-input/32bit-output.
-The final key addition
4 subtractions mod 2^{32}.

It seems that the heavy use of arithmetic operations, especially multiplications and additions mod 2^{32}, makes hardware slower and larger unless they are specially designed in a transistor level.

2.2 RC6

RC6 has three variable parameters, i.e., the number of rounds, the data block size, and the key size up to 2040 bits. The proposed version in AES has 20 rounds with a total of 4 additions (subtractions) mod 2^{32} before and after the round functions [Rivest (1998)], [RSA (1998)]. The major hardware components in the round function are as follows:

2 additions and 2 multiplications mod 2^{32},
2 data-dependent rotations.

These operations are well supported and fast on modern microprocessors, but expensive in hardware, especially multiplications and additions mod 2^{32}, make hardware slower and larger unless they are specially designed in a transistor level.

2.3 Rijndael

Rijndael also has a variable block length and a variable key length. The block length and the key length can be independently specified to 128, 192 or 256 bits. The proposed number of rounds in AES is 10, 12 and 14 when the key length is 128 bits, 192 bits and 256 bits, respectively [J.Daemen and V.Rijmen (1998)].
The round function of Rijndael in 128-bit blocks is composed of four distinct invertible transformations as follows:

-The ByteSub transformation
16 lookup tables with 8bit-input/output.
-The ShiftRow transformation
no hardware operations.
-The MixColumn transformation
logical AND and XOR operations.
-The AddRoundKey transformation
logical XOR operations.

Before the first round, the AddRoundKey transformation is also performed, and in the final round, the MixColumn transformation is omitted.
The basic components of Rijndael are logical operations and lookup tables; the latter is actually a composite function of an inversion over GF(2^8) with an affine mapping. Hence the structure of Rijndael is expected to be suitable for hardware implementation.

2.4 Serpent

Serpent has a 32-round SP-network structure with initial and final permutations, whose round function consists of 32 lookup tables with 4-bit input/output, logical and rotate shifts, and XOR operations [Anderson, Biham and Knudsen (1998)], [Biham (1997)].
These components are suitable for hardware implementation; particularly the small table size is expected to make hardware sufficiently small and fast.

2.5 Twofish

Twofish has a 16-round Feistel-like structure with an additional whitening of the input and output that consists of XOR operations. The major hardware components of the round function are as follows:

n lookup tables with 8-bit input/ output,
4 additions mod 2^{32},
logical AND and XOR operations,

The lookup tables can be also generated from another smaller 8 lookup tables with 4-bit input/output, and ***n*** is 12, 16 or 20 when the key length is 128, 192 and 256, respectively.
Twofish is not using particularly heavy operations in hardware, but its critical path is not short because, for instance, the number of cascaded 8x8 lookup tables is 48, where that for Rijndael is 10 when the key length is 128 [B.Schneier et. al. (1998)].

3. Design Policy

Our purpose is to evaluate the fastest possible encryption speed of the AES finalists using the existing hardware library under

fair conditions. To achieve this and also to complete the analysis in our limited time scale and resources, we designed the 128-bit key version for each candidate on the basis of the following criteria and conditions:

1. We fully unrolled the loop in the encryption and decryption logic and the key scheduling logic to achieve the fastest possible speed (throughput). In practice, the loop structure is commonly used in order to reduce hardware size, but generally makes the hardware slower because additional setup -time and hold-time is required for the loop registers, which is usually not negligible. Note that we therefore did not take a special effort to reduce hardware size.

2. We assume that all subkey bits are stored in subkey registers before an encryption operation begins. Also we have inserted another 128-bit resister to hold a block of ciphertext as shown in Figure 3.1, where we define the critical encryption and decryption path as the time required for all output bits of the encryption and decryption logic to reach the output registers under the fixed (sub)key value.

3. We did not introduce pipeline architecture; i.e., we did not insert any additional intermediate registers in the encryption and decryption logic. This is because the pipeline architecture makes the ECB mode faster but also blocks feedback modes of operations such as CBC. In other words, our hardware model encrypts one block plaintext data in one cycle.

4. We did not use a special optimization technique to design lookup tables in hardware. This means that the performance of the lookup tables heavily depends on optimization capability of the logic synthesis tool. In practice, as will be shown in the next section, the output of the synthesis tool seems to have reasonably optimized the lookup tables (not very slow).

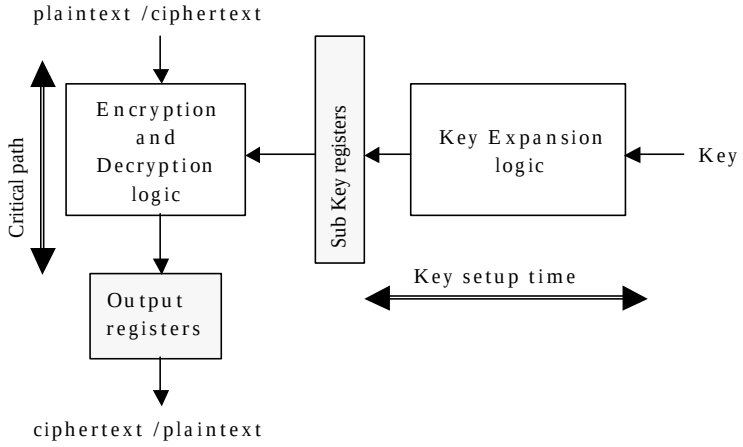


Figure 3.1 The hardware structure

5. Our design environment is as follows:

language:	Verilog-HDL
simulator:	Verilog-XL
design library:	Mitsubishi 0.35micron CMOS ASIC Library
logic synthesis:	Synopsys Design Compiler version 1998.08

For arithmetic operations such as additions, subtractions and multiplications, we used faster ones in the library of Synopsys Design Ware Basic Library [Synopsys (1998)]. Also, we adopted the WORST case hardware

conditions for evaluation. The worst case speed is a guaranteed speed of a given circuit, which is commonly used in real products. We think that the TYPICAL case evaluation is too optimistic to apply to a real ASIC hardware.

4. Evaluation Results

The results of our hardware evaluation of the five finalists are presented in Table 4.1. The fastest algorithm in terms of the critical path between plaintext and ciphertext is Rijndael, which is an only algorithm faster than DES. The second fastest algorithm is Serpent, which is twice faster than triple-DES but still much slower than Rijndael (approximately half). The speed of Twofish is almost the same as that of triple-DES, but Mars and RC6 are further slower; Rijndael is approximately ten times faster than RC6.

On the other hand, for the key setup time, Twofish is fastest, consuming only 5% of the critical path of its encryption procedure. Note however that the key setup time of DES and Triple-DES is almost nothing in hardware. Rijndael and Serpent have approximately 85%, while the key scheduling logic of Mars and RC6 is more than three times slower than their encryption.

Figures 4.1 and 4.2 show more detailed breakdowns of hardware components on the critical path of each algorithm, where the horizontal line of Figure 4.2 is normalized to show proportion of each component .

Mars has 16 multiplications, 26 additions/ subtractions, 15 lookup tables (specifically 11 S0's and 4 S1's) and 9 data-dependent rotations on its critical path, where all arithmetic operations are taken on mod 2^{32}. As shown in the figures, the multiplications occupy 63% of the critical path, 13% for additions/subtractions, and 9% for the lookup tables.

RC6 has 20 multiplications, 21 additions and 20 data-dependent rotations on its critical path, where all arithmetic operations are also taken on mod 2^{32} As shown in the figures, the multiplications occupy 77% of the critical path, 13% for additions/subtractions, and 8% for the data dependent rotations.

The critical path of Rijndael is not in the encryption but in the decryption procedure since the InvMixColumn function, which is an inverse of the MixColumn function, is a bit slower than the MixColumn function due to more complex constant values. On the critical path, a total of 10 InvByteSub functions (table lookups) occupy 48% of the entire decryption time, and a total of 9 InvMixColumn functions have 43%.

It is easy to see that the critical path of Serpent has 32 lookup tables and 31 linear transformations (XOR's and shifts). Our analysis shows that the linear transformations of Serpent are more expensive than its lookup tables; the former is 36% while the latter is 45%. In a logical sense, the lookup tables and the linear transformations must exhaust the critical path; however Figure 4.2 exhibits other factors that occupy a total of 19%. This is mainly because the design compiler has automatically inserted driver gates in order to supply sufficient fan-out counts, which reflects the fact that an output bit of a lookup table of Serpent has many "branches" that reach many different lookup tables in the next round. This is part of design criteria of Serpent.

It is also easily seen that the critical path of Twofish have 48 lookup tables --- specifically 16 q0's and 32 q1's, which is not a trivial fact ---, 16 MDS's (linear transformations) and 32 additions mod 2^{32}. The dominant part is the lookup tables, which occupy 53%, but also time for additions is not negligible (28%).

5. Discussions and Conclusions

The performance of Mars and RC6 heavily depends on the speed of the multiplication circuits mod 2^{32}. Our evaluation results show that the average time for the multiplication is around 23ns, which is six to eight times slower than the addition circuit mod 2^{32}, which takes around 3ns.

This also shows that by using highly optimized multiplication circuits in a transistor level, these algorithms are expected to be much faster. For this topic, see [Hagi (1998)] for instance. Now as an example, let us assume, in Mars and RC6, the 32-bit multiplication can work at the same speed as the 32-bit addition. We see that still the critical path of (the modified) Mars and RC6 is approximately 250 and 200ns, respectively. Also, we should notice that a full-custom solution is generally process-dependent and hence is not an inexpensive solution in practice.

Another speeding-up possibility is to optimize a lookup table. The average time for one lookup table for each algorithm is 3.2ns for Rijndael (8x8), 1.5ns for Serpent (4x4), 3.5ns for Twofish (8x8) and 3.5ns for Mars (8x32), respectively. Twofish will be most rewarded for the efforts of optimizing the lookup tables. However, the optimization will not lead to a significant impact to affect the ranking of the five finalists.

In this paper, we did not take efforts to reduce the size (area) of each algorithm since we adopted a full loop unrolling in order to evaluate the fastest possible encryption speed. Appendices 1 and 2 show the information of the size of each algorithm with the detailed breakdowns, which we will not discuss here. How to reduce the gate size is another practical topic to be pursued.

References

[NIST (1998)]:
http://csrc.nist.gov/encryption/aes/aes_home.htm

[MITSUBISHI (1997)]:
Mitsubishi Electric America, Inc.,
"0.35um CMOS ASIC DATA BOOK ", 1997.

[Burwick et.al. (1999)]:
http://www.research.ibm.com/security/Mars.html
See also
http://csrc.nist.gov/encryption/aes/round2/AESAlgs/MARS/mars-int.pdf

[Rivest (1998)]:
R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, "The RC6 Block Cipher," 1998.

[RSA(1998)]:
http://www.rsasecurity.com/rsalabs/aes/index.html

[J.Daemen and V.Rijmen (1998)]:
J. Daemen and V. Rijmen, "AES Proposal: Rijndael," Document vers on 2, Date: 03/09/99.
http://www.esat.kuleuven.ac.be/~rijmen/rijndael

[Biham (1997)]:
E Biham, "A Fast New DES Implementation in Software", in Fast Software Encryption - 4th International Workshop, FSE '97, Springer LNCS v 1267,pp 260-271.

[Anderson, Biham and Knudsen (1998)]:
R. Anderson, E. Biham and L. R. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," 1998.
http://www.cl.cam.ac.uk/~rja14/serpent.html

[B.Schneier et. al. (1998)]:
B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Fergusen, "Twofish: A 128-Bit Block Cipher," June 15, 1998.
http://www.counterpane.com/twofish.ps.zip

[Synopsys (1998)]:
Synopsys Inc. ,"Design Ware Foundation Quick Reference Guide ", Aug.1998.

[Hagi (1998)]:
Y.Hagihara, et. al., "A 2.7ns 0.25um CMOS 54x54b Multiplier", ISSCC Digest of Tech. Papers, pp296-297, Feb.1998.

Table4.1 Hardware evaluation results

Algorithm name	area [Gate]			Key setup time[ns]	Critical-path[ns]	Throughput [Mbps]
	Encryption & Decryption	Key Schedule	Total			
DES	42,204	12,201	54,405	-	55.11	1161.31
Triple-DES	124,888	23,207	148,147	-	157.09	407.4
MARS	690,654	2,245,096	2,935,754	1740.99	567.49	225.55
RC6	741,641	901,382	1,643,037	2112.26	627.57	203.96
Rijndael	518,508	93,708	612,834	57.39	65.64	1950.03
Serpent	298,533	205,096	503,770	114.07	137.4	931.58
Twofish	200,165	231,682	431,857	16.38	324.8	394.08

Figure 4.1 Critical Path of the Finalists(1)

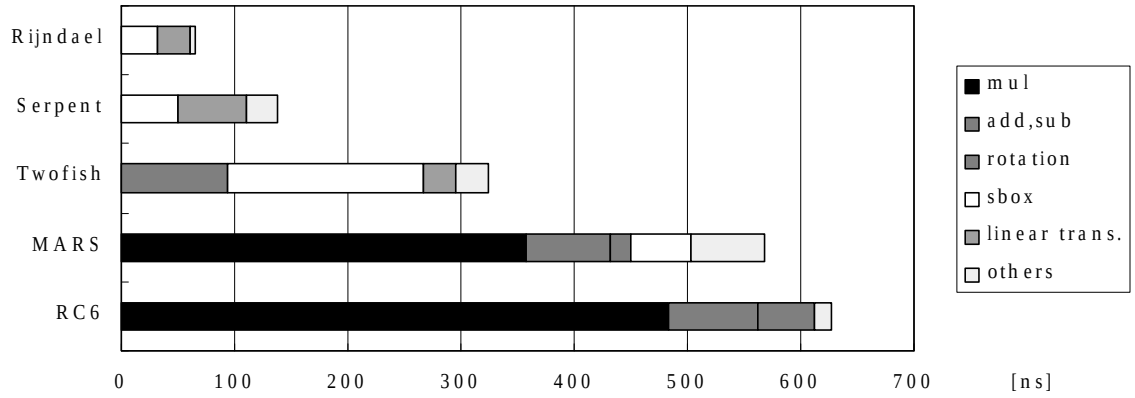
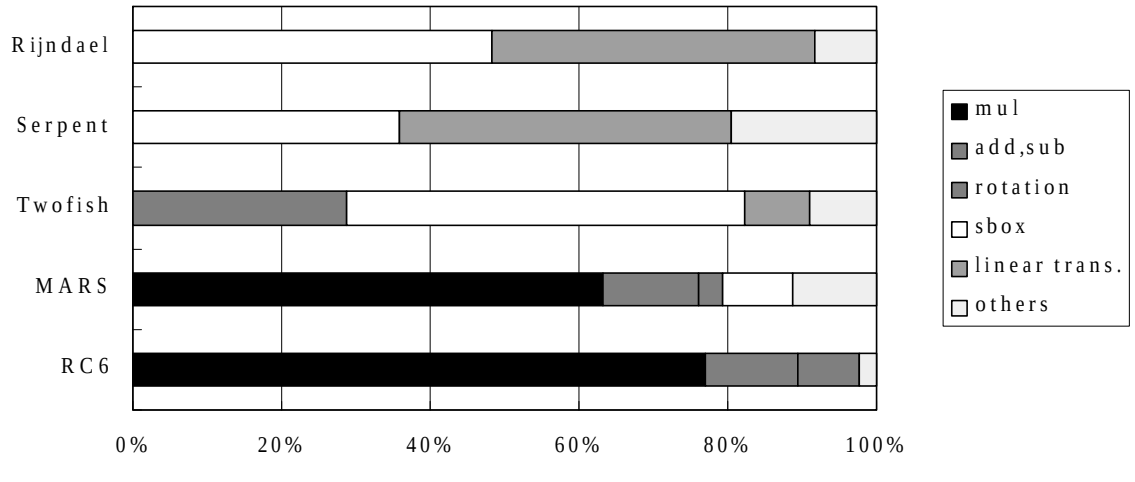
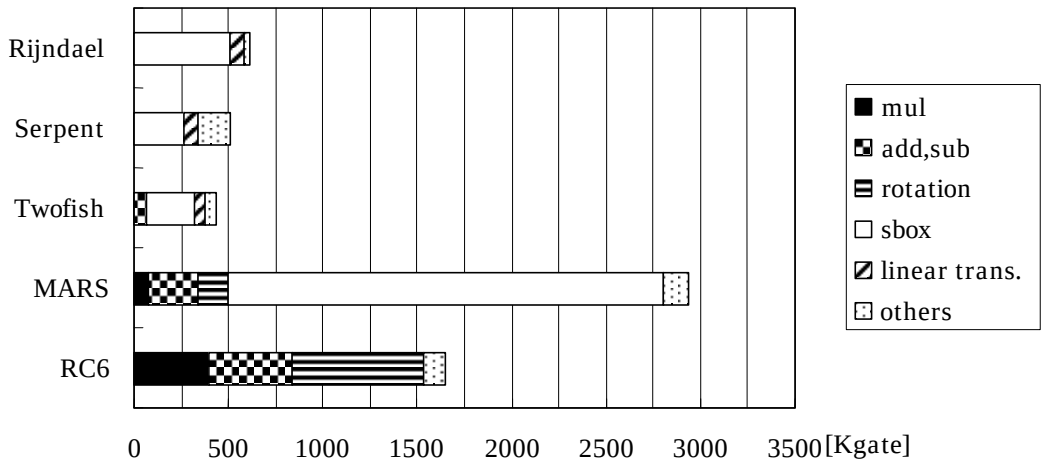


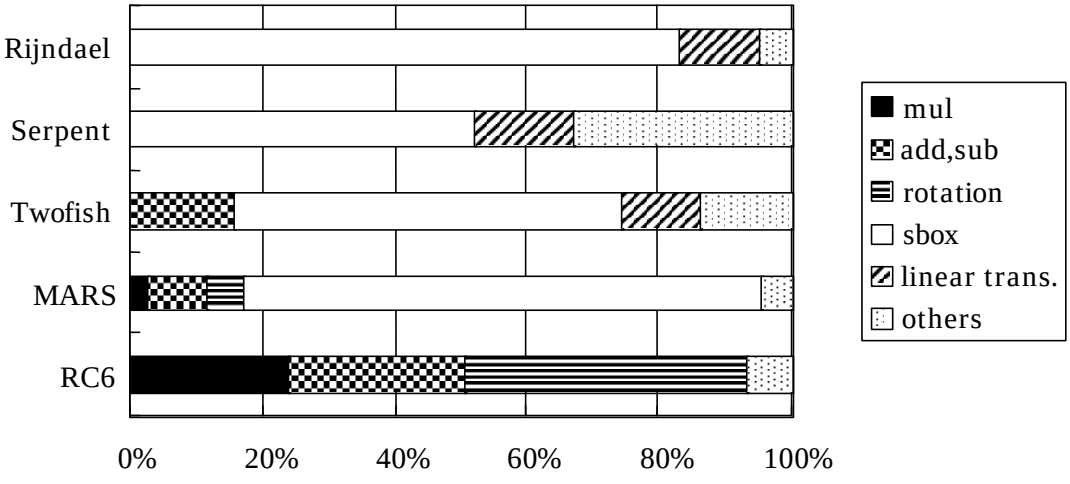
Figure 4.2 Critical Path of the Finalists(2)



Appendix 1: Area Size of the Finalists(1)



Appendix 2: area size of the Finalists(2)



Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms

Bryan Weeks, Mark Bean, Tom Rozylowicz, Chris Ficke

National Security Agency

1 Abstract

The National Security Agency (NSA) is providing hardware simulation support and performance measurements to aid NIST in their selection of the AES algorithm. Although much of the Round 1 analysis focused on software, much more attention will be directed towards hardware implementation issues in the Round 2 analysis. As NIST has stated, a common set of assumptions will be essential in comparing the hardware efficiency of the finalists. This paper presents a technical overview of the methods and approaches used to analyze the Round 2 candidate algorithms (MARS, RC6, RIJNDAEL, SERPENT and TWOFISH) in CMOS-based hardware. Both design procedures and architectures will be presented to provide an overview of each of the algorithms and the methods used. To cover a wide range of potential hardware applications, two distinct architectures will be targeted for comparison, specifically a medium speed, small area iterated version and a high speed, large area pipelined version. The standard design approach will consist of creating hardware models using VHDL and an underlying library of cryptographic components to completely describe each algorithm. Once generated, the model can be verified for correctness through simulation and comparison to test vectors, and synthesized to a common CMOS hardware library for performance analysis. Hardware performance data will be collected for a variety of design constraints for each of the algorithms to ensure a wide range of measured data. A summary report of the findings will be presented to demonstrate algorithm performance across a wide range of metrics, such as speed, area, and throughput. This report will provide a common baseline of information, which will enable NIST and the community to compare the hardware performance of the algorithms relative to one another.

2 Introduction

The National Security Agency (NSA) agreed to provide technical support to the National Institute of Standards and Technology (NIST) in the form of an analysis of the hardware performance of the Round 2 Advanced Encryption Standard (AES) algorithm submissions. This analysis consisted of the design, coding, simulation and synthesis of the five algorithms using the procedure outlined below. Throughout this evaluation, NSA has taken care to assure that best design practices were used and that all algorithms received equal treatment. No attempt was made to optimize any particular design, but care was taken to find the best configuration for each algorithm. Cross-validation measures during design and simulation were used to overcome the subjective effects of the design process and to ensure that all designs receive the same amount of attention. The results of this analysis should provide an accurate measure of the hardware performance of each algorithm relative to the others. Undoubtedly more optimized (and hence better performing) implementations of these algorithms can be designed, so the individual score of any particular algorithm is not very valuable outside the context of this environment. The point of this analysis is to provide a controlled setting in which a meaningful comparison can be made.

Based on a mathematical description of the Round 2 algorithms, and C code reference models when necessary for clarification, NSA designers fully described each of the algorithm submissions in a hardware modeling language. A review by a team of design engineers followed the initial design stage to reduce the effects of coding style on performance. Using commercially available analysis, simulation and synthesis tools, NSA design engineers have performed simulations to produce performance estimates based on each of the hardware models. In order to provide a wider perspective on the performance of the algorithms, two different architectures or applications were simulated for each algorithm: an iterative version to provide a medium speed operation at minimal area/transistor count, and a pipelined version to provide optimum speed operation, but at the cost of a larger area. This report is a summary of the performance of the Round 2 AES candidate algorithms, and will compare and contrast the results of the analysis.

3 Hardware Design Background

3.1 Design Guidelines

For this analysis effort, one of the main goals was to provide an unbiased comparison of the algorithms in hardware, specifically in Application Specific Integrated Circuits (ASICs). To that end, the overhead found in typical hardware implementations, such as a robust user-interface, was minimized to reduce the impact on the overall performance of the algorithms. The user-interface is the Input/Output (I/O) connections and logic needed to take the plaintext and key and present them to the algorithm, and take the output ciphertext and present it off the chip. All inputs and control signals were registered in a common interface in order to provide uniformity across all of the algorithms, with fixed setup and hold times identical for all algorithms. A wide variety of architectures could be used to implement a given algorithm. In order to restrict all possible choices and yet capture valuable data points, two fundamental architectures were chosen: iterative and pipelined. All algorithms were designed in each architecture style. There are several variations on these approaches, including multiple copies of an iterative implementation for parallel processing, a partially pipelined implementation, or a combination of these hybrids (multiple copies of a partially pipelined implementation). The approach chosen will depend on the needs of the system, but these variations will likely result in performance within the ranges given by the iterative and fully pipelined implementations. However, these optimizations were beyond the scope of this study.

3.1.1 Target Applications

3.1.2 Iterative Architecture

The iterated approach to implementing the algorithm focuses on providing a medium to low speed version of the algorithm, with efforts placed on limiting the physical size of the hardware. In this instance of the algorithm, one step is performed per clock period, with the output of the previous step being used as the input to the next step. Data is only placed on the output after the required number of algorithm rounds has been completed.

3.1.3 Pipelined Architecture

The pipelined approach to implementing an algorithm centers on providing the highest throughput to the design, sacrificing area to obtain the level of performance needed. In the case of pipelining, all of the steps in computing the algorithm are cascaded into a single design, with each stage feeding the next stage. The latency remains the same as in the iterated case, but the throughput is increased significantly as new data is placed on the output on every clock cycle. Pipelining has been shown to be an effective method of dramatically increasing the throughput capabilities of a given algorithm. However, it comes at the expense of limiting the number of cryptographic modes that can be supported at the maximum throughput rate. For example, since the latency of an encryption cycle remains the same as an iterative case, there is no throughput advantage when using feedback modes such as Cipher Block Chaining (CBC). High performance applications, such as high speed network encryption, will require the increase in throughput, and as a result, often focus on a non-feedback mode of operation such as counter mode to obtain performance.[1]

3.2 Parameter Description

There are many design parameters that can be reported for each design implementation. Some parameters will have much more significance in a given application or environment than others. This evaluation reports on these parameters as a method of comparison among the five algorithms, and does not claim that any single parameter has been fully optimized. The following is a description of the parameters being reported. Some have a direct impact or relation to performance metrics (e.g. throughput) and some are simply a function of the algorithm itself (e.g., I/O requirements). Algorithm performance in each of the evaluation categories will be documented for each algorithm submission.

3.2.1 Area

As an estimate based on an available MOSIS library, the results of the synthesis area reporting will consist of pre-layout area estimates of the algorithm. Although potentially different from a post-layout estimate, the area reported

by Synopsys will provide a relative comparison of each of the algorithm submissions. Generally, the two varieties of architectures-– iterative and pipeline -– will be on the extremes of area with the iterative being the smallest, and the pipelined being the largest.

3.2.2 Throughput

In most cases, throughput is directly proportional to area; as area decreases, throughput decreases. As with area, the iterative and pipelined architectures will report the extremes of throughput. Iterative architectures will have much lower throughput rates since there is a minimum amount of hardware, and it is re-used on multiple clock cycles of execution. Thus, the throughput is limited by the amount of hardware reuse. More specifically, it is limited by the number of rounds in a codebook algorithm. On the other hand, a pipelined architecture dedicates hardware for performing all calculations in any given clock cycle. This maximizes throughput by allowing data to be written and read from the device on every clock. In this case, throughput is a function of the worst-case delay in any one given stage of the algorithm. Throughput will be reported for both iterative and pipelined architectures.

3.2.3 Transistor Count

Transistor count is a more specific measure than area and is often more useful. While transistor count is somewhat dependent on the design library being used, it is a useful method of comparing the algorithms since they were compiled using the same library. In addition, the transistor count will be a more useful figure than area when estimating programmable logic implementations since these devices typically report the number of useable gates (which is also directly related to transistor count). Based on the synthesized netlist (from Synopsys), an additional report describing the number of transistors required to implement the algorithm will be provided.

3.2.4 Input/Outputs (I/O) Required

With the goal of consistency among algorithms, the I/O was fixed identically for all algorithms. However, since this parameter is highly useful to hardware designs, it will still be reported.

3.2.5 Key Setup Time

The key setup time refers to the amount of time required before subkey expansion is ready to execute. Some algorithms use the user-supplied key directly in the subkey expansion thereby reducing the key setup time to zero. Others require some pre-calculation or translation of the key prior to subkey expansion steps. Key setup times will be examined to assess the overhead of each algorithm in establishing a usable key.

3.2.6 Algorithm Setup Time

Similar to key setup time, the algorithm setup time reports the minimum amount of time before an algorithm is ready to process data. Time to create look-up tables, etc. will fall in this category. None of the evaluated algorithms contained an algorithm setup time greater than zero.

3.2.7 Time to Encrypt One Block

This paramter will address minimum latency times for each of the algorithm submissions. The time to encrypt one block, measured in nanoseconds, is a function of two parameters: the worst-case path delay between any two registers, and the number of rounds in the algorithm.

3.2.8 Time to Decrypt One Block

As above, this parameter will address minimum latency times for each of the algorithm submissions. Decryption does not always require identical processing as encryption. Therefore, the time required to decrypt one block is reported.

3.2.9 Time to Switch Keys

Originally, this parameter was included as a measure to encompass both key setup time and algorithm setup time overhead. However, since none of the evaluated algorithms contained an algorithm setup time, this parameter is identical to key setup time. Therefore, it will not be reported further in this document.

4 Methodology

4.1 Standard Design Flow

The design process followed a common methodology used by ASIC designers. The process started with the documentation supplied by the algorithm authors and was completed with a gate-level schematic, which included the performance metrics data. A complete ASIC development would require physical layout and fabrication. These steps were beyond the scope of this effort. However, the performance metrics data obtained here closely matches that which would be found from actual fabrication and testing. Previous efforts using these tools have correlated estimated performance from the schematic to the actual testing. Figure 1 shows the steps in the design flow.

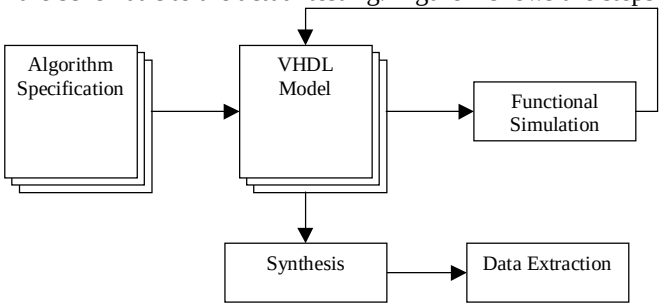


Figure 1 Standard Design Flow

4.1.1 VHDL code generation

VHSIC Hardware Description Language (VHDL)

VHDL modeling is analogous to programming simulations in C code and follows much of the same syntax. However, unlike a behavioral description of the algorithm, VHDL (IEEE 1076) specifies how the algorithm will be implemented in hardware. Using this hardware language, NSA designers fully described the hardware necessary to implement each of the algorithm submissions. Performance metrics, such as speed, area, etc. (see below) can be estimated from the hardware description using available analysis and computer aided design (CAD) tools.

There are different styles in which to code VHDL models, offering various levels of abstraction. For this evaluation, the designers used the register transfer logic (RTL) coding style. For this style, the placement of registers and corresponding logic between registers is chosen by the designer and is determined at the VHDL code level. There are many different methods for identifying an optimized placement of registers. Ideally, there would be an equal amount of logic delay between registers for all stages of the design. However, in order to simplify the design cycle and to be consistent among all algorithms, the designers chose a common placement of registers, even if this placement is not fully optimized. Specifically, the output of each “round” (as defined by the algorithm authors) is registered for both a key schedule round and an algorithm round.

4.1.2 Simulation and Verification

NSA followed the design phase with a functional VHDL simulation of the designs using the Synopsys VHDL System Simulator (VSS) to verify the correct operation of the algorithm. The test vectors submitted to NIST for each algorithm were applied to assure that the design was working as intended. Specifically, the Variable Key and Variable Text tests were performed for each algorithm implementation and mode (e.g., iterative encrypt, pipelined decrypt, etc.). The modeled algorithm output was also compared with the C code model supplied to provide an added assurance that the simulation was operating as expected.

4.1.3 Code review

NSA had one or more engineers design the VHDL for each algorithm submitted. Initial hardware designs were straightforward implementations of the core algorithm. Following completion of each initial design, an informal group of engineers met to review and provide feedback for the design. Improvements and alternatives to the initial design were examined to determine potential benefits from differing architecture approaches (area compression, pipelining, etc.). Variants of the design that improve the performance of the algorithm were then programmed for comparison.

4.1.4 Synthesis

Gate-level synthesis of the algorithm utilized the Synopsys Design Compiler to produce a functionally equivalent schematic in hardware. A MOSIS-specific technology library was used to generate a gate-level schematic of the design and provide more accurate area and timing estimates, as if the design were to be implemented in an integrated circuit (IC). The MOSIS library is based on a publicly available fabrication facility's model of a specific CMOS process, thus giving real performance metrics for an available ASIC line. The VHDL model can be re-targeted to any supported hardware or field programmable gate array (FPGA) design libraries.

The synthesis process can generate a wide range of implementations depending on the constraints provided to the synthesis tool. For example, one implementation may minimize area while another may minimize delay time. In hardware synthesis, the two fundamental parameters are time and area. These parameters are directly related. As delay time decreases, area increases. Timing and area curves that further illustrate this point are shown in subsequent sections. The constraints provided for each algorithm synthesis routine were maintained consistently. Therefore, differences among algorithm synthesis results will be a function of the logic required (algorithm specific) and the synthesis tool's ability to meet the given constraints.

4.1.5 Documentation

In addition to a summary report containing performance data, both design notebooks and VHDL documentation will be provided to NIST for evaluation. The design notebooks will contain reporting information for all of the hardware data that was collected, with all algorithms, designs, and architectures represented. The VHDL models and their testbenches (for simulation verification) will also be included.

4.2 Synthesis Analysis

4.2.1 Function Characterization

Although the hardware design of the algorithms followed a top-down approach, the synthesis portion of the analysis proceeded from the bottom of the design up through the hierarchy. In order to obtain an accurate picture of the performance of the sub-blocks and functions in this type of analysis, a sweep was performed on each of the functional blocks to graphically depict performance versus design constraints. Specifically, the timing constraints, such as output delay and clock frequency, of each of the blocks were varied to observe the performance output of the block. The results of the sweep make up the characterization for that particular block. All subsequent blocks of the hierarchy will be analyzed using these methods.

4.2.2 Cryptographic Library

With characterization curves for each of the sub-blocks complete, five speed grade implementations were selected to cover the performance range of the block. A variety of key performance points were selected to reflect requirements for both high speed and small area. Figure 2 shows typical timing and area curves following a sweep of maximum delay time constraints. These curves allow design engineers to select specific implementations of a given function. Specifically, five implementations, or speed grades, were chosen for each function. In this example, the five selected implementations are noted in the figure, and they represent one minimum time delay, one minimum area, and three other points that have desired characteristics such as large area savings for a small increase in delay time.

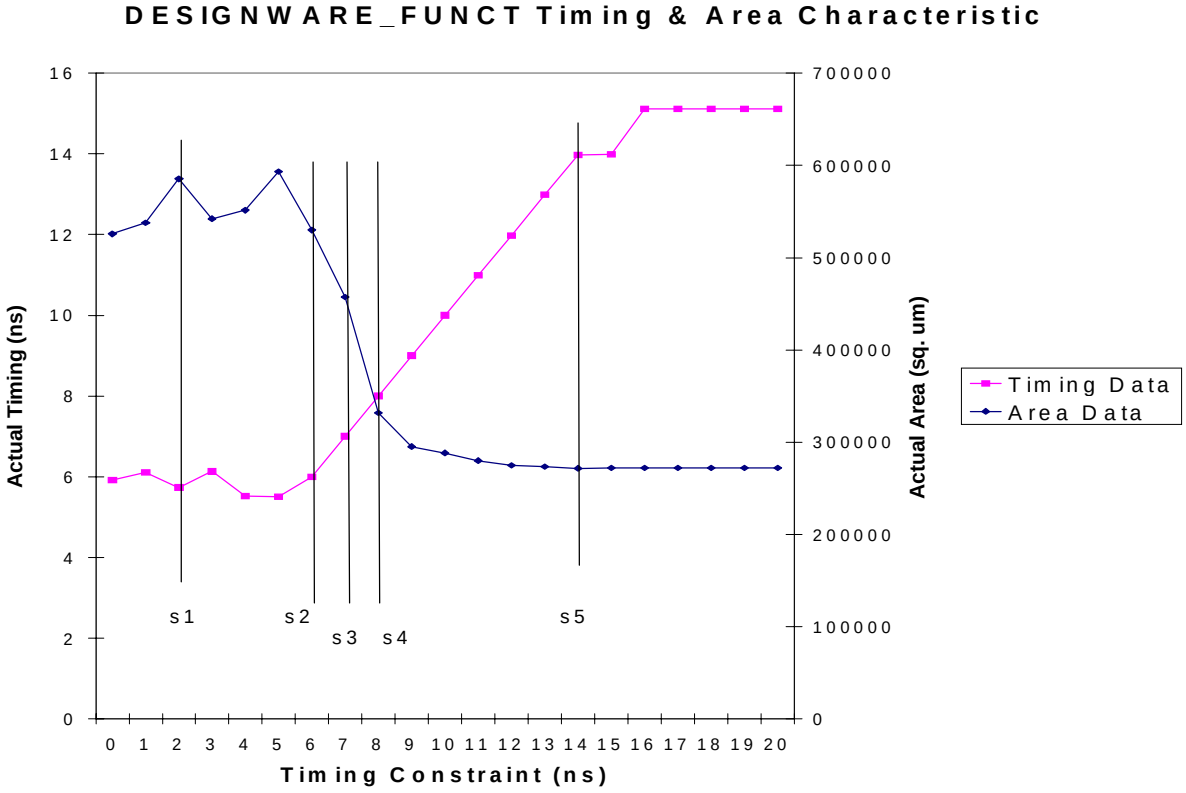


Figure 2 Sample Function Sweep

The five implementations were selected only for the functions of each of the algorithms, and then assembled into a cryptographic library. Each library contained implementations of all the functions required to build the given algorithms.

4.2.3 Block Level Characterization

Continuing with the bottom up approach, the higher level blocks underwent the same performance sweep as described for the function level. The design constraints were varied across the entire range of the block to fully describe the performance curve of the block. At each iteration of the synthesis process, components (e.g., functions) were selected from the cryptographic library based on the required speed and performance. Performance curves at the block level encompassed components of several different speed grades depending on design constraints. At the top level, the characterization curve reflected the performance of the entire design across a wide range of design constraints.

5 General Architecture Approach

5.1 Top Level Architecture

The design of each algorithm started with a common top level architecture that is well suited for virtually any codebook algorithm. The generalized top level architecture consists of an Interface, an Algorithm block, a Key Schedule block and a Controller. Figure 3 shows a block diagram of the architecture.

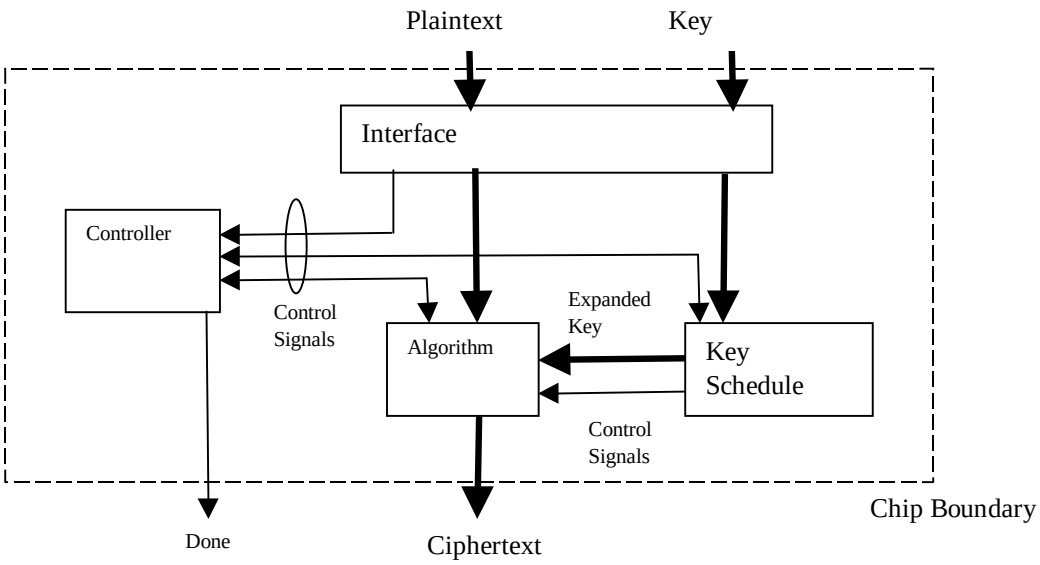


Figure 3 Top Level Architecture

The Interface serves to register all data inputs. This is consistent with the hardware design methodology of placing registers at the chip boundary, thus minimizing strict setup-and-hold timing requirements. In some cases, the interface also provides minimum functionality, such as padding keys when appropriate. The Key Schedule performs the generation of subkeys to be used in by the Algorithm block. This includes any required key setup as well as the expansion itself. The Algorithm performs the actual encryption or decryption of data provided from the Interface using the subkeys from the Key Schedule. For iterative implementations, the Algorithm and Key Schedule blocks implement a single round with internal feedback datapaths; whereas the pipelined implementations expand these sections to include as many implementations of a round as required by the particular algorithm. Finally, the Controller provides any necessary control signals for maintaining proper synchronization among the various blocks.

6 Algorithm Evaluation

For each of the algorithms, a description of how it was architected for both the pipelined and iterated cases is given. Any nuances of how the rounds were simulated and the key schedule implemented are also given along with specific examples of approaches to reduce redundancy or streamline the design. Each algorithm section then provides block level results of timing constraints versus both chip area and timing in both the iterative and pipelined cases. A table of performance parameters is then provided for four different key sizes, 128 bit key, 192 bit key, 256 bit key, and a hybrid that combines all three key sizes in one key schedule that can be controlled for any particular key size. In some cases, the combined three-in-one key schedule must make compromises to achieve the greater degree of flexibility. Each of the performance parameters is described in more detail in Section 7, along with comparisons across the five algorithms.

Following the architecture for each of the algorithms, each section will provide a summary of the results of the hardware analysis for the individual algorithm. In an effort to save space, the timing and area graphs will be presented for only the combined case which contains all three key sizes in one implementation. Both pipelined and iterated cases will be covered. However, the complete report and design workbooks will contain graphs for all key sizes and contains a much more complete data set. The corresponding tables will capture key performance data points for all key size implementations. *

* Note: At time of publication, not all information was available for every parameter and for every algorithm. Due to some unforeseen difficulties in the amount of time for simulation, some information on area, transistor count and key setup times was not available. This was especially true for simulating the larger blocks in the pipelined cases and for the various key sizes. In addition, certain information for MARS and RC6 was being finalized at time of publication, so is not included in this version. Incomplete data in the following sections are indicated with asterisks.

Complete data for the performance curves and tables of key parameters will be provided on the NIST web site and at the conference.

6.1 MARS

6.1.1 Architecture

The MARS algorithm requires several different types of rounds[2]. Specifically, there are unkeyed forward mixing, keyed forward transformation, keyed backwards transformation and unkeyed backwards mixing rounds, as well as pre-addition and post-subtraction. The mixture of keyed and unkeyed rounds resulted in the requirement for complex control and data flow operations between the Key Schedule and Algorithm blocks. Specifically, a complex control situation results from the fact that subkeys are required immediately for the pre-addition stage, whereas the next subkeys are not required until the eight unkeyed forward mixing rounds are completed. This architecture presented some unique timing and data synchronization issues.

6.1.1.1 Pipelined Key Schedule

As with all pipelined implementations, the subkey from each round must be registered. However, for MARS, the subkeys are not utilized on consecutive clock cycles. Therefore, additional pipelined storage is necessary. The updated key schedule of MARS following AES Round 1 allowed for separating the 40 subkeys into groups of 10. The VHDL model takes advantage of this operation by adding pipelined storage for groups of 10 subkeys, only as long as necessary, rather than creating pipelined storage for all 40 subkeys. This reduced the total number of registers required. Additionally, the pipelined registers are controlled by a latch signal rather than updating on every clock. Again, this reduced the number of registers by removing redundancy.

6.1.1.2 Pipelined Algorithm

Relative to the intricacies of the key schedule, the pipelined algorithm implementation is straightforward. It consists of six different types of rounds, each one with its own registered output: one key addition, eight unkeyed forward mixings, eight keyed forward transformations, eight keyed backwards transformations, eight unkeyed backwards mixings and one key subtraction. This makes a total of 34 rounds to complete the algorithm.

6.1.1.3 Iterative Key Schedule

The MARS algorithm key schedule generates 10 subkeys at a time. Therefore, the traditional iterative methodology of a single round implementation for generating a single subkey (or set of subkeys as required by a single algorithm round) did not apply. Instead, a single round implementation per 10 subkeys was generated resulting in a key expansion round iterated four times for one encryption cycle. This presents some additional logic overhead for iterative applications in that a “round” generates 10 subkeys simultaneously rather than the exact amount needed by the algorithm at a given stage. (In the case of MARS, two 32-bit subkeys are required per keyed round in the cryptographic core.) In addition to the subkey expansion overhead, there is a storage overhead for the remaining subkeys. Also, decryption requires a full expansion of subkeys prior to beginning data processing. Therefore, the full set of 40 subkeys is stored in registers.

6.1.1.4 Iterative Algorithm

The iterative algorithm is consistent with the pipelined algorithm in its relative simplicity when compared to the key schedule. There is a single register for all rounds. The input to the register depends on the round number. For example, the input for the first round of encryption is the key addition round result; for the second round it is the unkeyed forward mixing round result and so on.

Subkeys are presented to the algorithm block as an array of all 40 subkeys. This differs from other iterative algorithm implementations that present only one subkey at a time. The rationale for this design was to eliminate duplicate logic in both the Key Schedule block and Algorithm block. Due to the timing gaps in the application of subkeys and the fact that all 40 subkeys are generated prior to decryption processing, it was considered advantageous to allow a 40 element bus to connect the two blocks.

6.1.2 MARS Top Level Results

6.1.2.1 Timing and Area

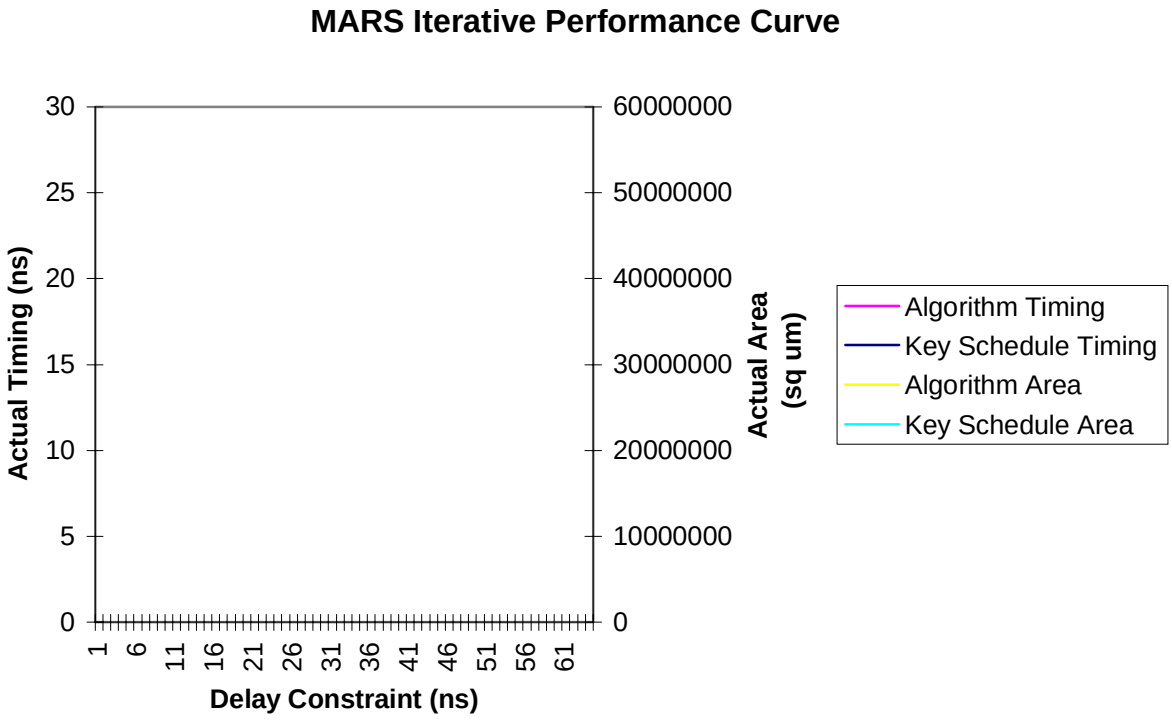


Figure 4

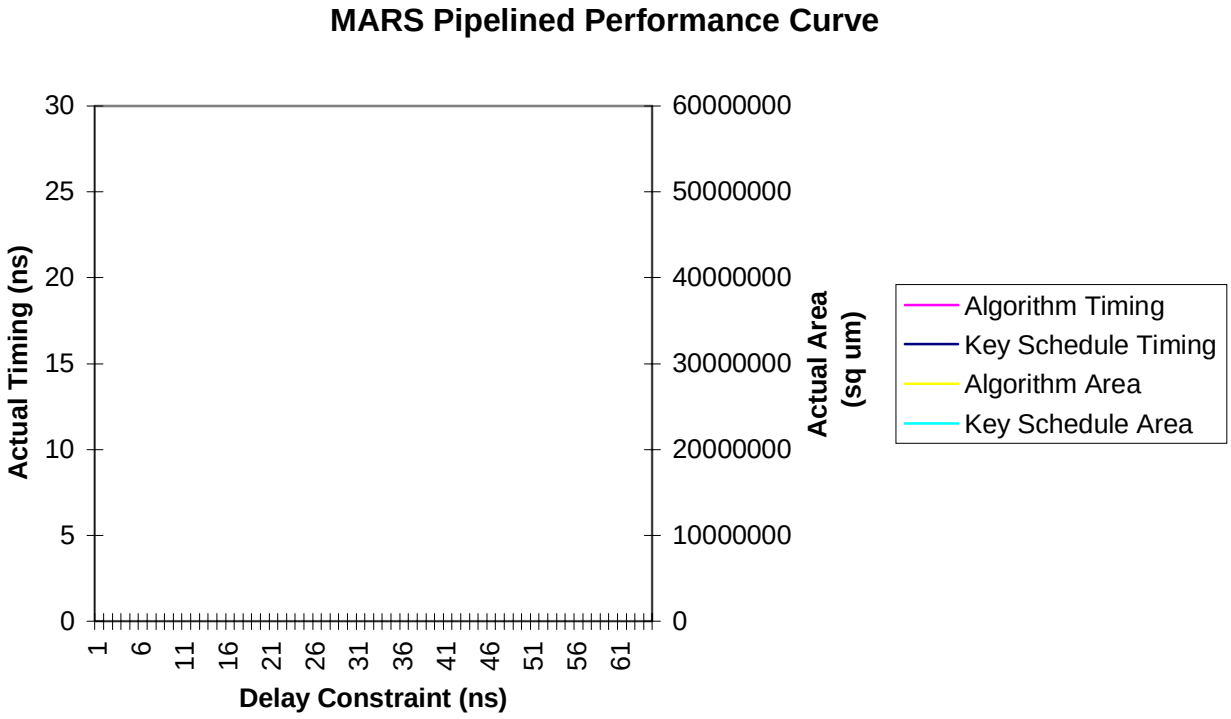


Figure 5

6.1.2.2 Key Parameters

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	*	*	*	*
Transistor Count	*	*	*	*
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	*	*	*	*
Key Setup Time Encrypt (ns)	*	*	*	*
Key Setup Time Decrypt (ns)	*	*	*	*
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	*	*	*	*
Time to Decrypt One Block(ns)	*	*	*	*

Table 1 MARS Summary

6.2 RC6

6.2.1 Architecture

The following provides a high level description of the major blocks in the RC6 algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook[3].

6.2.1.1 Pipelined Key Schedule

The RC6 key schedule is pipelined using a slightly different method than the other algorithms. Since a significant number of computations for the key schedule are required before any expanded keys are generated, the architecture takes advantage of the run-up by performing the expansion at the start of the pipeline. Only a single copy of the expansion hardware is required, but additional registering is needed to maintain the keys on a time dependent basis, discarding keys from previous stages (i.e., the keys have already been used). Keys are then passed from register to register to follow the data in the pipeline.

6.2.1.2 Pipelined Algorithm

The algorithm "unrolls" the stages of the algorithm into a pipeline, following the algorithm description for function ordering and naming conventions. Combination functions are used to perform cases where distinct operations need to be performed in encrypt and decrypt. For example, the pre-add will contain both addition and subtraction to accommodate both cases. A similar condition exists in the algorithm round function, with slightly different functions needed for encrypt and decrypt. However, synthesis optimization can take advantage of common operations, such as the multiply, to reduce the total number of operators needed.

6.2.1.3 Iterative Key Schedule

The iterative key schedule is designed to perform a single round of expansion per clock. Expanded keys are fed to the algorithm block after the controller initiates a start signal. However, the key setup has been designed to compute single or multiple steps of the run-up in a single clock, depending on the performance needed by the rest of the system. A load cryptovariable (i.e., load key) signal from the controller will initiate the key setup. Once complete, the expansion can be started.

6.2.1.4 Iterative Algorithm

The RC6 iterative algorithm closely reflects the pipelined version. The same round instance is called repeatedly to process input data. The encrypt and decrypt are symmetrical with respect to operations performed in a similar manner (e.g., pre-add, round, post-add), so the same block can be called without additional overhead.

6.2.2 RC6 Top Level Results

6.2.2.1 Timing and Area

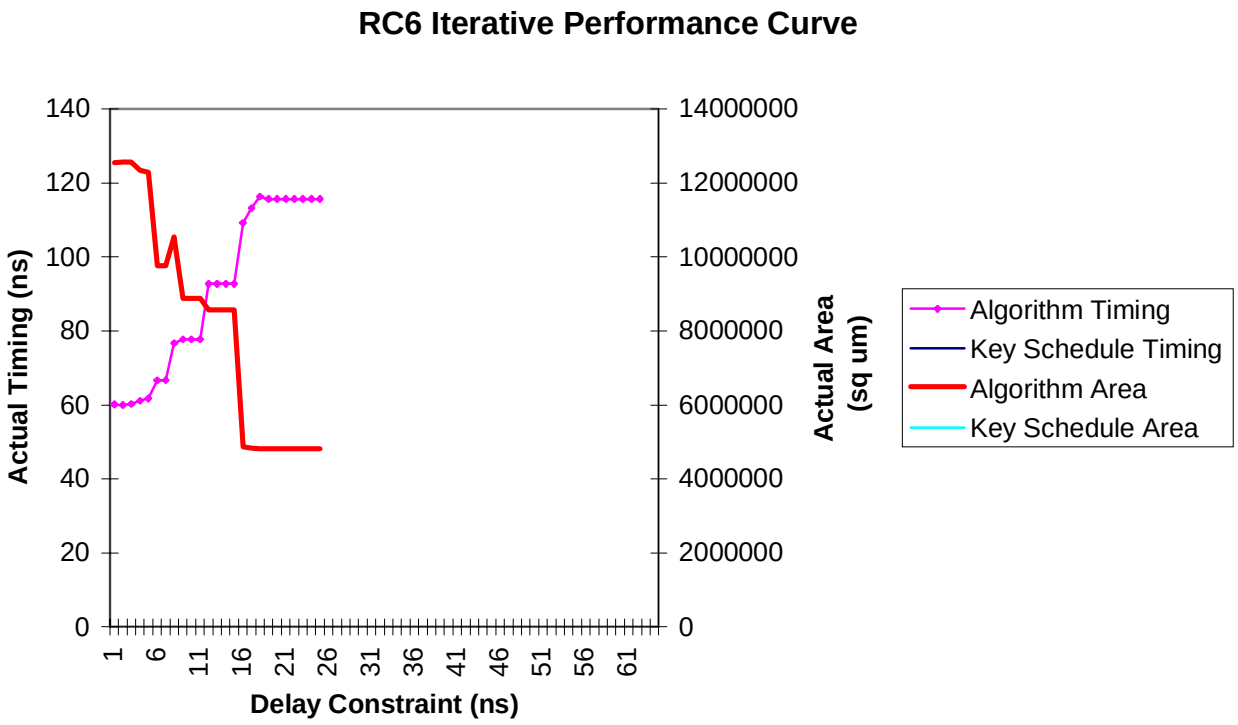


Figure 6

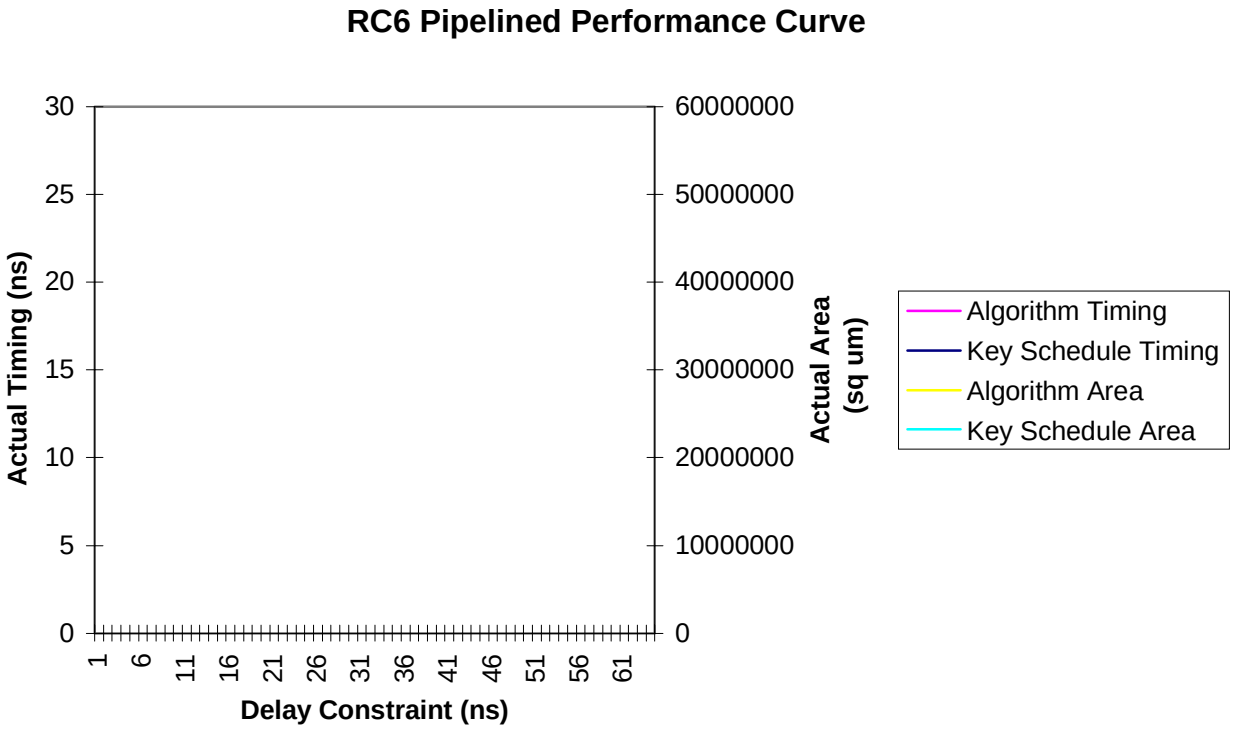


Figure 7

6.2.2.2 Key Parameters

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	*	*	*	*
Transistor Count	*	*	*	*
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	*	*	1192.00	2171.00
Key Setup Time Encrypt (ns)	*	*	*	*
Key Setup Time Decrypt (ns)	*	*	*	*
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	*	*	1179.2	2146.8
Time to Decrypt One Block(ns)	*	*	1179.2	2146.8

Table 2 RC6 Summary

6.3 RIJNDAEL

6.3.1 Architecture

The following provides a high level description of the major blocks in the RIJNDAEL algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook[4].

6.3.1.1 Pipelined Key Schedule

The RIJNDAEL key schedule is based on a sliding window approach as described in the algorithm specification. Multiple key sizes are based on the n-1 element and the n-k element (32 bit word organized), where k is 4,6, or 8, depending on key size. The key expansion is a linear combination of the elements, so a similar function can be used on the decrypt function to “unexpand” the keys in a reverse direction. Such an approach allows for an increase in the key agility without sacrificing significant amounts of area to store all of the expanded keys.

The encryption expansion can start immediately, with the first words of the initial key being used as expanded key. The setup time for this case is zero. During the decryption, the key is expanded to the last key, stored, and then the pipeline is run to create the previous expanded key until the last decrypt key is generated, which is the initial key. Keys are generated at a rate of four 32 bit words per round, regardless of key size, to keep up with the requirements of the algorithm block. Additional registers are used to maintain sufficient previous keys to generate the next four words of expanded key.

Keys are pulled from the bank of registers which make up the sliding window. S-Boxes are re-used, without a performance penalty, to minimize the size impact of having additional S-Boxes.

6.3.1.2 Pipelined Algorithm

The RIJNDAEL algorithm pipeline consists of a sequential mapping of the steps of the algorithm to registered stages in hardware. Each stage reflects a single round of the algorithm. The primary advantage to pipelining in this manner is the significant increase in throughput. RIJNDAEL was architected such that both the encrypt and decrypt functions could be performed with the same pipeline. This approach needed a static pipeline that could perform both functions, so the algorithm round functions contained in the package will serve a dual role by providing cases for encrypt and decrypt within the same function. The pipeline structure reflects changes in direction, such as requiring a pre-add on the encryption (first round) versus decryption requiring a post-add on the last round.

6.3.1.3 Iterative Key Schedule

The iterative version of the key schedule focuses on reducing the area of the key expansion, so only a single copy of the expansion is maintained. For encryption, as in the pipelined case, the expansion starts immediately, with no key setup required. The keys are expanded every round, producing the four 32 bit words of key required. As each new key is created and stored, the old key is overwritten.

In the case of decryption, the algorithm requires a setup time to effectively run the algorithm to the last key. This serves as the starting point for all decryptions using that key. This value will also be stored so it can be referenced on each new decryption to eliminate key setup for every new decryption.

6.3.1.4 Iterative Algorithm

The algorithm block uses the same functionality as described in the pipeline but does not re-use some of the combination functions used to construct the pipeline. Instead, the function calls are made explicitly, depending on encryption/decryption to provide the widest possible range of hardware re-use. The function calls in the encrypt and decrypt directions are not symmetrical. The algorithm processes the state data on each round, performing only one step of the algorithm per round.

6.3.2 RIJNDAEL Top Level Results

6.3.2.1 Timing and Area

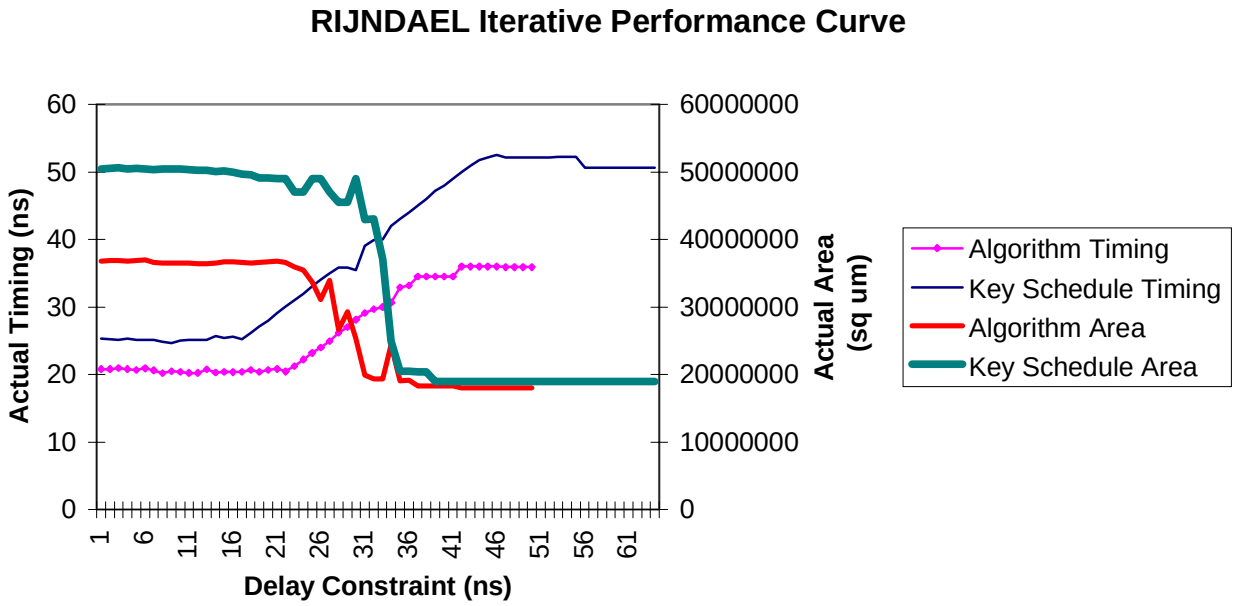


Figure 8

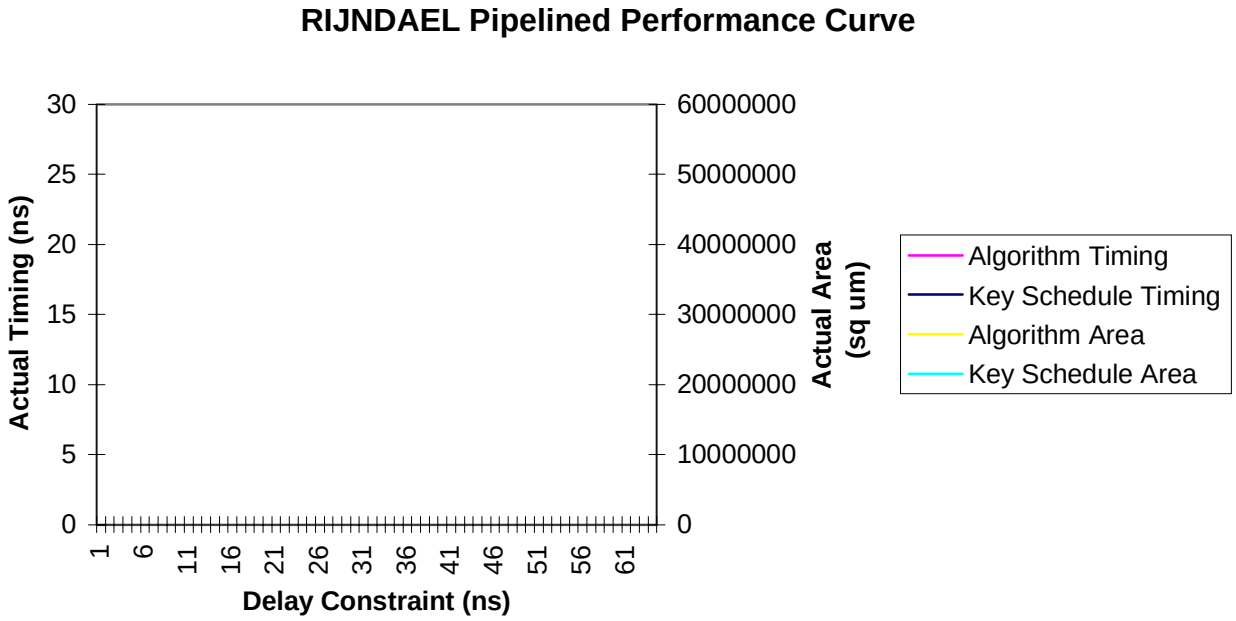


Figure 9

6.3.2.2 Key Parameters

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	37034346.00	81661400.00	*	*
Transistor Count	*	*	*	*
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	371.06	519.48	4060.00	5163.00
Key Setup Time Encrypt (ns)	0.00	0.00	0.00	0.00
Key Setup Time Decrypt (ns)	246.4	344.96	0	277.92
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	493.8	346.36	247.4	346.36

Table 3 RIJNDAEL Summary

6.4 SERPENT

6.4.1 Architecture

The following provides a high level description of the major blocks in the SERPENT algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook[5].

6.4.1.1 Pipelined Key Schedule

The SERPENT algorithm implements a simple expansion function for the key scheduling. The exclusive-or based function allows for quick computation and does not require key setup in the encrypt direction. Pipelining is maximized as this approach utilizes a sliding window approach, where only a small number of previous expanded keys are needed to compute the next sub-keys. However, for decryption, a key setup time is required to compute the starting point for the key expansion, which is the last set of W registers. The decrypt pipeline computes the previous set of W registers based on the current set, as the exclusive-or based expansion can be reversed easily. To save storage in this design, the keys are computed at each stage, with the decrypt case requiring a block of logic at the beginning to find the last subkeys.

The SERPENT pipelined key schedule provides two successive keys to each round of the algorithm on expansion. The algorithm will select the correct key based on the current encryption/decryption mode. The additional key allows for the rounds that require two keys to operate.

6.4.1.2 Pipelined Algorithm

The pipelined SERPENT algorithm block contains a structural model of the unraveled rounds of the algorithm. Four distinct functions are needed to implement both the encrypt and decrypt operations. The core algorithm round functions are the same for 30 rounds of the algorithm, with an internal mux/demux to select the encrypt or decrypt mode. The first two rounds of encrypt and last two rounds of decrypt distinguish the cases where the pipeline is re-routed. The encrypt will bypass the two special rounds of the decrypt while the decrypt will bypass the two special rounds of encrypt. The latency will remain the same as no extra rounds are added. The pipeline will select and re-route based on the current mode of encryption or decryption.

6.4.1.3 Iterative Key Schedule

The SERPENT iterative key schedule uses a single copy of the expansion function to generate the sub-keys, one at a time. Area can be significantly reduced using the same hardware repeatedly. Additional key setup will be required in the decrypt direction to allow for the run-up to the last key of the expansion.

6.4.1.4 Iterative Algorithm

The iterative algorithm uses the same functions as the pipeline, with the same round instance referenced repeatedly to perform the main processing of the algorithm. The special case rounds are selected by the state machine within the iterative block to determine encrypt/decrypt direction, and consequently, which pre/post add functions to perform.

6.4.2 SERPENT Top Level Results

6.4.2.1 Timing and Area

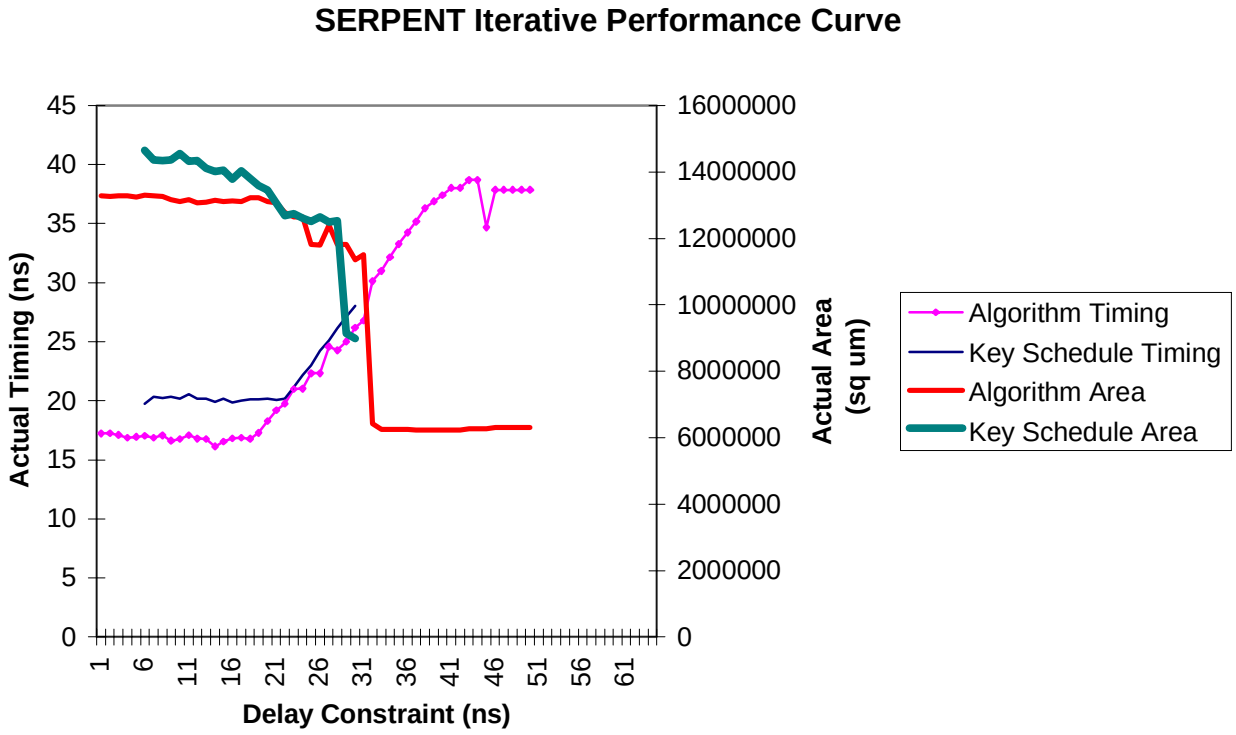


Figure 10

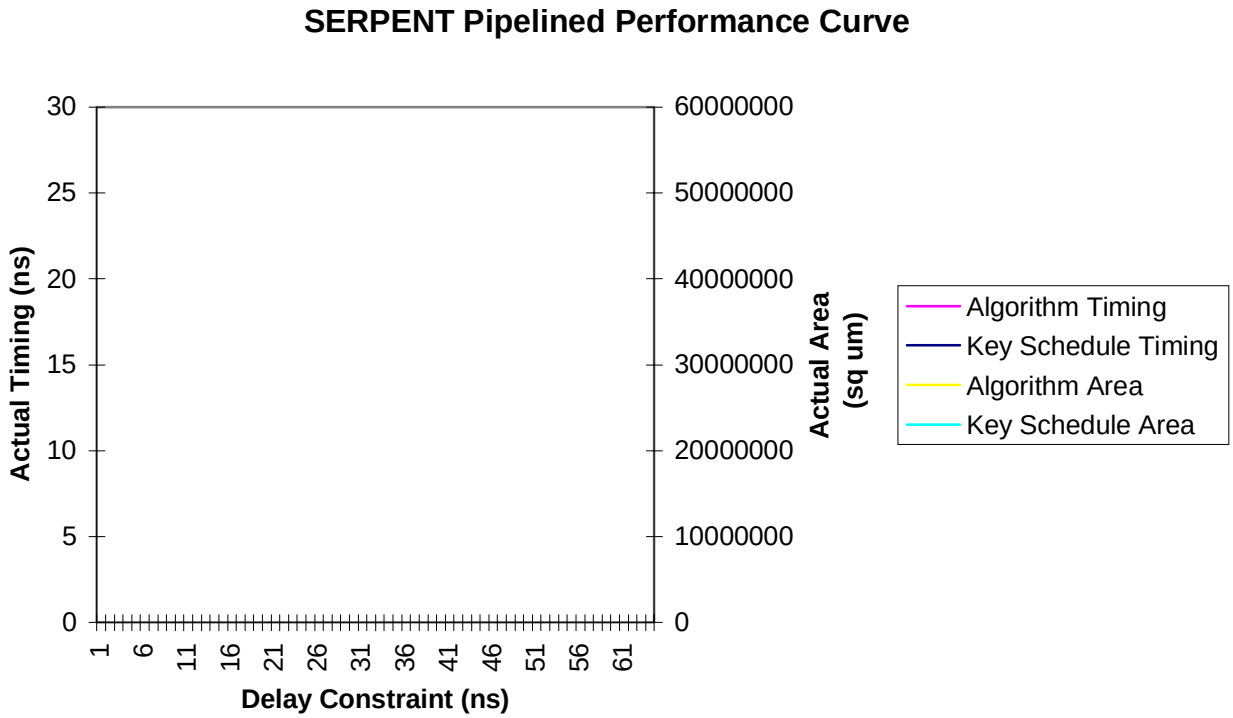


Figure 11

6.4.2.2 Key Parameters

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	*	*	*	*
Transistor Count	*	*	*	*
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	*	202.33	5298.01	8030.11
Key Setup Time Encrypt (ns)	19.77	*	6.74	11.76
Key Setup Time Decrypt (ns)	672.18	*	212.55	365.58
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	632.64	*	510.08	773.12
Time to Decrypt One Block(ns)	632.64	*	510.08	773.12

Table 4 SERPENT Summary

6.5 TWOFISH

The following provides a high level description of the major blocks in the TWOFISH algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook[6].

6.5.1 Architecture

A useful property of the TWOFISH architecture was the relatively large amount of re-use of design blocks. Both the Key Schedule and Algorithm utilized many of the same functions. While this does not result directly in a direct increase in performance, since key expansion and encryption are performed in parallel, it does simplify the hardware coding process. As stated, common coding techniques and processes were used for developing each algorithm design resulting in areas available for improvement in a more highly optimized design. In the case of TWOFISH, a smaller design could be created by taking advantage of the function re-use. However, as with most hardware trade-offs, this area optimization would come at the expense of performance and complex control mechanisms.
Another feature of TWOFISH is the lack of initial key runup prior to subkey expansion. In addition, the key schedule is not a feed-forward design. Each round of key schedule is independent of the previous round. This unique characteristic allowed for a Key Schedule that does not require a setup time for either encryption or decryption.
In the TWOFISH algorithm, the first step of encryption is a pre-whiten function, In hardware, this is simply an exclusive-OR. The pre-whiten step is performed during the same clock cycle as the first subkey expansion which generates the pre-whiten subkey. This was possible because the XOR function did not create a critical path concern since the main algorithm rounds incorporate an integer addition that is more complex. The result was the ability to load data and key in the same clock cycle, thereby reducing the overall time for encryption by one clock cycle.

6.5.1.1 Pipelined Key Schedule

In order to allow for either encryption or decryption, both pre-add and post-add subkeys are generated during the first pipeline stage. The post-add key is buffered through a pipelined delay until needed in the final processing step. Also, since one of the input parameters is the round number which is fixed for a given pipelined round there is an optimization or pre-calculation in each pipelined round.

6.5.1.2 Pipelined Algorithm

The algorithm is an efficient unrolling of stages because encryption and decryption are nearly identical. In addition, the symmetry allows for similar processing in both the encrypt and decrypt directions.

6.5.1.3 Iterative Key Schedule

As in the pipelined key schedule, the iterative design requires buffering of the post-add subkey until it is needed in the final processing step. However, this buffering is not required to be implemented in pipelined stages. The key schedule round is generalized such that the round number is not a fixed constant as in the pipelined case. This does not allow synthesis optimization of each round, but does save area since only one hardware block is instantiated.

6.5.1.4 Iterative Algorithm

The differences between encryption and decryption are minor such that the additional hardware to support either process in a single round adds minimal area.

6.5.2 TWOFISH Top Level Results

6.5.2.1 Timing and Area

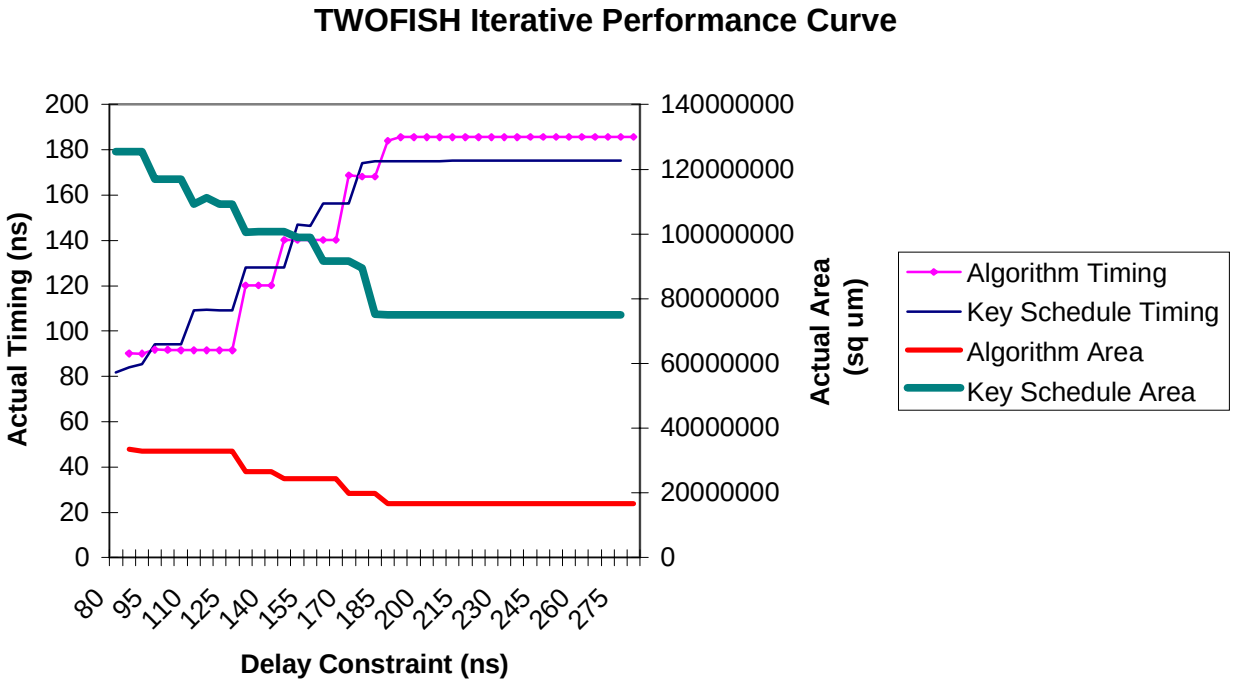


Figure 12

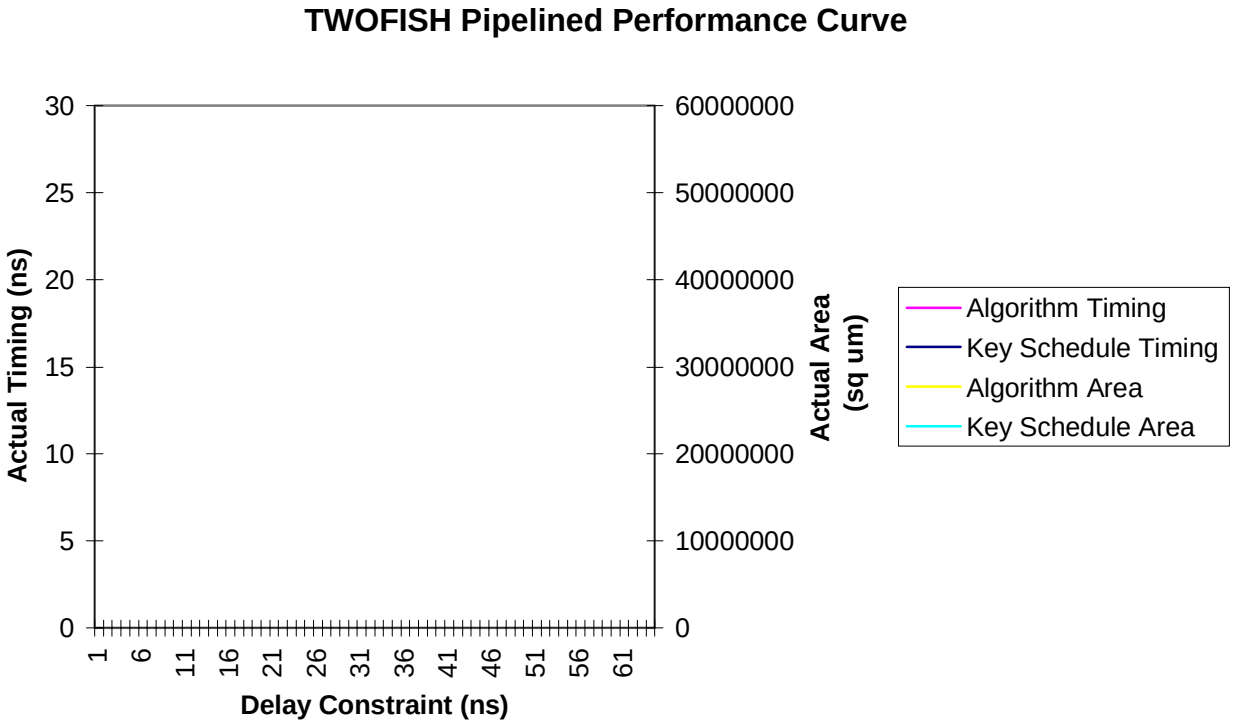


Figure 13

6.5.2.2 Key Parameters

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	91686840	158300076	*	*
Transistor Count	*	*	*	*
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	38.29	79.00	*	1445.55
Key Setup Time Encrypt (ns)	0	0	0	0
Key Setup Time Decrypt (ns)	0	0	0	0
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1620.18	3342.6	1593.9	*
Time to Decrypt One Block(ns)	1620.18	3342.6	1593.9	*

Table 5 TWOFISH Summary

7 Performance Results

A table summarizing the results and performance metrics is given below for algorithm comparison. These comparison values are given only for the combined key size implementation, which implements a selectable 128 bit, 192 bit, and 256 bit key in the same implementation.

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	*	*	*	*	*
Transistor Count	*	*	*	*	*
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	*	519	*	202	79
Key Setup Time (ns)	*	*	*	*	*
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	*	494	*	633	1620
Time to Decrypt One Block(ns)	*	494	*	633	1620

Table 6 Iterated Summary

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	*	*	*	*	*
Transistor Count	*	*	*	*	*
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	*	5163	2171	8030	1445
Key Setup Time (ns)	*	*	*	*	*
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	*	247	1179	510	1594
Time to Decrypt One Block(ns)	*	247	1179	510	1594

Table 7 Pipelined Summary

8 Summary

This paper has presented an overview of the methods and architectures used for the AES hardware comparison. The primary characteristics used for design tradeoffs in hardware engineering are area and timing. As such, each algorithm was examined from the standpoint of minimum area (iterative architecture) and maximum throughput

(pipelined architecture). Further, statistics and data based on area and timing were emphasized and illustrated for each algorithm.
The results (in Section 7) show vital parameters for both the iterative and pipelined architectures of each algorithm that can be used to evaluate relative performance. The designs were not optimized for any one parameter, but rather they serve as a good testbench scoring of all the algorithms relative to one another, given the same commonly used hardware design practices and procedures. Key performance data points to highlight are minimum transistor count and maximum throughput. *
It should be emphasized that any data point based on a single parameter (e.g. transistor count or throughput) is a relatively narrow view of the algorithm's overall performance or rating. For this reason, there was no attempt to rank algorithms in order. Rather, it is left to the cryptographic community to establish a consensus of the most important parameters – in combination or alone – and to draw appropriate conclusions from the data provided herein.

* Note: Because incomplete information was available at publication time, additional results will be updated and provided to the community through NIST as the parameter information is filled in for all algorithms.

9 Acknowledgements

The authors would like to thank others at NSA for providing support to this project and in editing this report, namely Mr. Jeff Ingle. The authors also appreciate the opportunity NIST gave in encouraging this analysis, since they feel that high performance implementations for high-speed networks are an important aspect of the AES competition.

10 References

[1] W. Semancik, L. Mercer, T. Hoehn, G. Rowe, M. Smith-Luther, R. Agee, D. Fowlkes, and J. Ingle , "Cell Level Encryption for ATM Networks and Some Results from Initial Testing," , *DoD Fiber Optics Conference*, March 1994.

[2] 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," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[3] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6™ Block Cipher," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[4] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[5] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[6] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

High-Speed MARS Hardware

Akashi Satoh[†], Nobuyuki Ooba[†], Kohji Takano[†], Edward D'Avignon[††]
[†]IBM research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14, Shimotsuruma,
Yamato-shi, Kanagawa 242-8502, Japan
{akashi, ooba, chano}@jp.ibm.com
[††]IBM Corporation, Poughkeepsie, NY 12601, USA
davignon@us.ibm.com
March 15, 2000

Abstract. High-speed MARS encryption/decryption hardware was developed using a 0.18μm IBM CMOS technology. In order to boost performance, a special adder and multiplier was designed by optimizing the adder block structure and interconnections between adder cells using signal delay profiles. A description of the hardware including block diagrams and data flow diagrams is presented. One of the most critical portions of the design is the special adder and multiplier. The design philosophy and tradeoffs used in these pieces are discussed. Finally, performance and size estimates are presented along with the rationale behind them. The design achieves 677Mbit/s data rate for encryption when using cipher block chaining and 1.28Gbit/s for decryption and other encryption modes in 13.8Kgates + 2.25Kbyte SRAM.

1. Introduction

MARS [1] is a symmetric-key block cipher, supporting 128-bit blocks and a variable key size. It is designed to take advantage of the powerful operations supported by today's computers, resulting in a much improved security/performance tradeoff over existing ciphers. We developed high-speed MARS hardware for use when additional performance or security is required over a software implementation. Since MARS uses 32-bit multiplications and additions in conjunction with S-box lookups, it is essential for MARS hardware to have a high-speed multiplier and adder. The key to realizing high-speed arithmetic circuits is to first break one operation into parallel sub-operation blocks, then precisely adjust and control the number of signal delays from each block. We developed an automatic circuit generation program, which optimizes the parallel block structure and the wiring interconnection by using the signal delay profiles. A high-speed adder with the combination of carry-skip [2] and carry-select [3] techniques designed for an RSA encryption LSI [4] was implemented in the final stage of the multiplier. These arithmetic circuits boost the speed of MARS hardware while maintaining compact silicon area.

In this paper, we first show the data path level design of the MARS hardware with an overview of the MARS algorithm and how encryption and decryption are performed. Next, we discuss the techniques that apply to the adders and multipliers to realize the high-speed MARS computation. Finally, we give estimated performance results and the size of the MARS hardware.

2. MARS Algorithm and Hardware Architecture

2.1. Hardware Block Diagram

We designed the MARS hardware entirely from the gate level to the chip level, so that it is ready for chip fabrication. Figure 1 shows the block diagram of the hardware. It has a chip external bus, which consists of a 32-bit data bus, a 10-bit address bus, four control signals, and a clock, to interface with external logic, such as a CPU. Through the bus, the external logic will read and write message data and the key. The hardware has a forward/backward mixer, a cryptographic core for MARS encryption/decryption, and a key expander for key setup. During those operations, two S-boxes and key storage are accessed. Each S-box is a 32-bit × 256-word SRAM. The key storage is a 32-bit × 64-word SRAM.

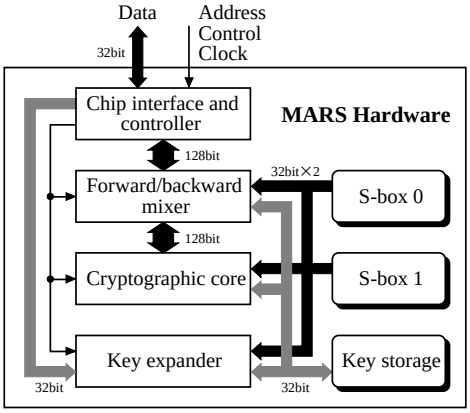


Figure 1. Block diagram of MARS hardware.

2.2. Encryption Procedure

The MARS encryption procedure has three phases: 8-round "forward mixing," 16-round "cryptographic core," and 8-round "backward mixing," as shown in Figure 2. Figure 3 shows the type-3 Feistel network structure of MARS. A 128-bit plain text block is divided into four 32-bit data words M_0, M_1, M_2, M_3, and encrypted as four words D_0, D_1, D_2 and D_3. In the figure, $\oplus$ denotes XOR, "<<<*n*" and ">>>*n*" denote *n*-bit cyclic left and right rotations, respectively. The lower 32 bits of the results of addition, subtraction and multiplication are used; the higher bits are discarded. MARS uses S-box (32-bit × 512-word table) lookups in the key setup, encryption, and decryption procedures. The S-box is composed of two 256-entry tables S0 (the first 256 words) and S1 (the last 256 words), used in the forward and backward mixing phases. The decryption procedure is the inverse of the encryption operation, and all circuits shown in this paper are used for both procedures by switching selectors in the data paths.

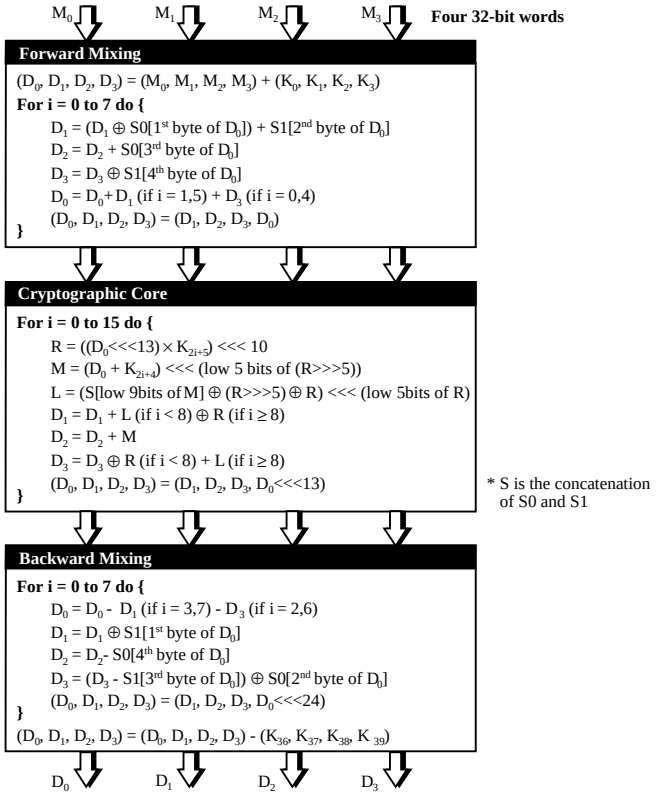


Figure 2. MARS encryption procedure.

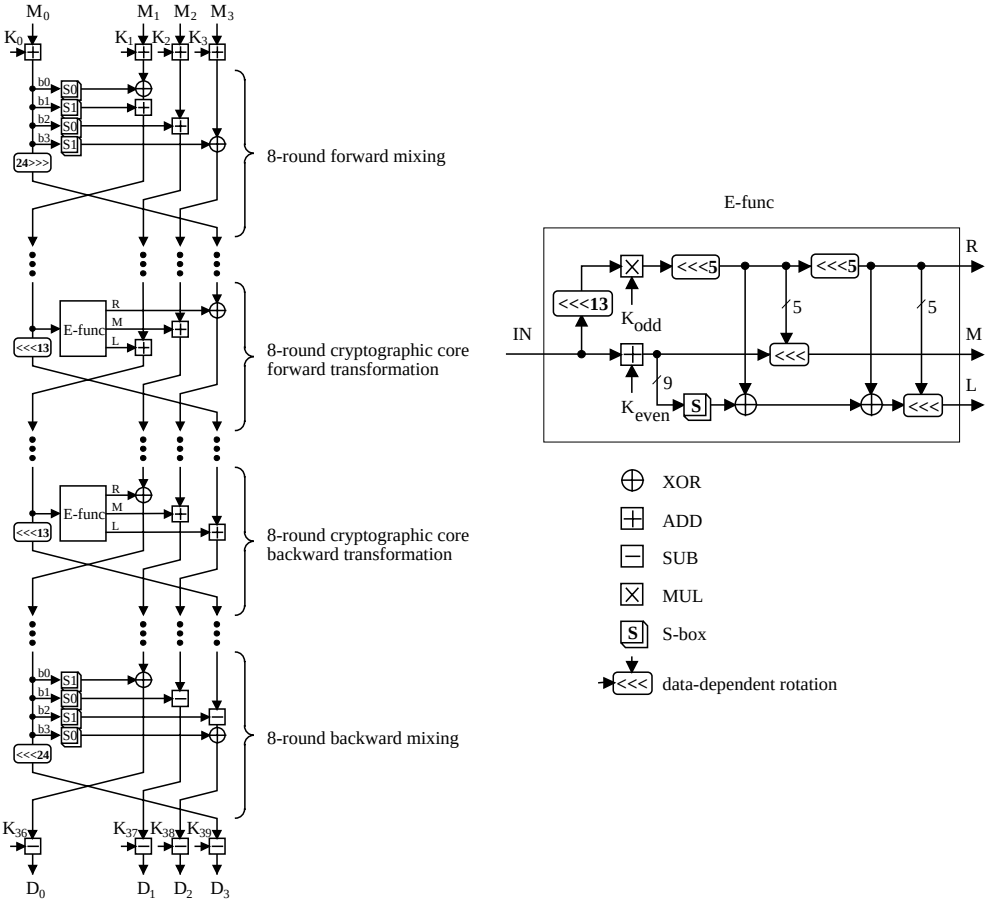


Figure 3. Type-3 Feistel network structure.

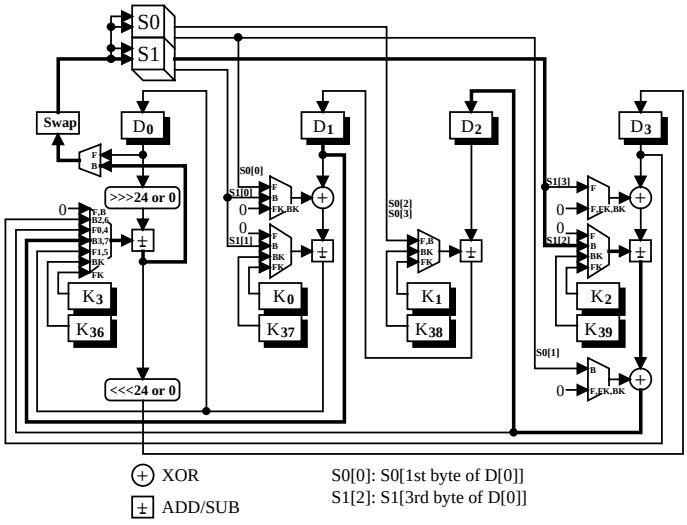


Figure 4. Forward / backward mixing data paths.

Figure 4 shows the circuit block diagram of the forward and backward mixing data paths. This circuit is also shared by the encryption and decryption procedures. Switching the selectors changes the order of the operations. The S-boxes, S0 and S1, are implemented by three-port SRAMs, one port for the write and two ports for the read operations. The thick lines show the critical path for the backward mixing process, which contains subtraction, S-box, subtraction, and XOR operations in order. Two sets of key registers $K_{0\text{-}3}$ and $K_{36\text{-}39}$ are dedicated to this mixing operation, and eight key words are copied from the 32-bit × 40-word expanded keys stored in SRAM "K".

This circuit block is used 9 times in the forward mixing mode, then one cycle is required to add the sub-keys $K_{0\text{-}3}$ to the data $D_{0\text{-}3}$, and then 8 times in the rounds of mixing operation. The backward mixing operation takes 9 cycles.

Figure 5 is the block diagram of the cryptographic core (Feistel network) data path. The thick lines specify the critical path. It consists of a multiplier, two XORs, a conditional rotator, an adder and a selector. The S-box read operation is executed in parallel with the multiplication, so that the memory access time does not affect the critical path. The S-box shares the SRAM used for the forward and backward phases shown in Figure 4. The cryptographic core operation uses this circuit in the 8-round keyed forward transformation followed by the 8-round keyed backward transformation. The cryptographic core requires 16 cycles for each 128-bit block encryption.

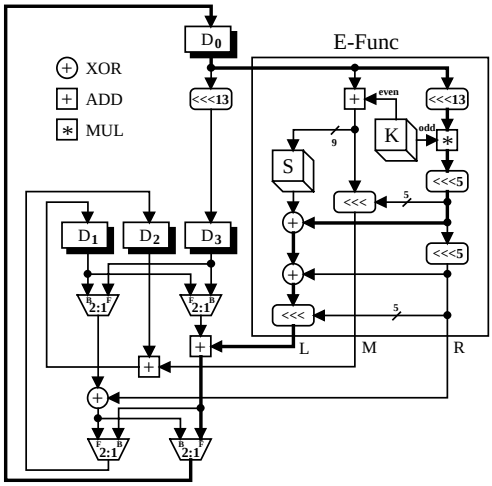


Figure 5. Cryptographic core data path.

The cryptographic core and the forward/backward mixer can operate simultaneously on separate 128-bit blocks when four-port (one for write and three for read) SRAM is used as the S-box. A 128-bit bus connection can swap data between these two circuits without additional cycles. If we share the circuits of Figure 4 with forward and backward mixing operations to save hardware resources, 18 cycles are required for one set of encryption procedures. A timing chart for this case, which is suitable for electronic codebook (ECB) encryption mode and all decryption modes, is given in Figure 6 (a). The data throughput of this architecture is 128 bits / 18 cycles. For cipher block chaining (CBC) encryption mode, the encrypted data D in the previous cycle is required before starting the current encryption of block M. In this case, the mixing phases cannot be pipelined with the cryptographic core. CBC operations require 34 cycles, with the throughput becoming 128 bits / 34 cycles. The timing chart for cipher block chaining encryption mode is shown in Figure 6 (b).

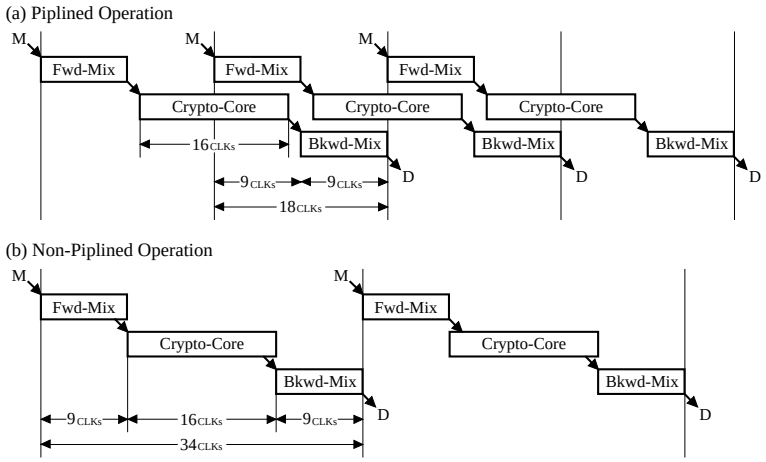


Figure 6. Timing chart of MARS encryption.

2.3. Key Expansion

The key expansion procedure, shown in Figure 7, expands the user-supplied key array, k_0, …, k_{n-1}, into a 40-word internal key array, K_0, …, K_{39}. The range of *n* is from 4 to 14 32 bit words, that is, MARS supports user key lengths from 128 bits to 448 bits. In the figure, bit-wise OR and AND are denoted by $\vee$ and $\wedge$, respectively. The block diagram of the key expander data paths is shown in Figure 8. The major components of the key expansion circuit are a barrel rotator, two registers, an adder, and multiplexers. The key storage "K" is implemented using a three-port SRAM. It is capable of one write and two read operations in parallel. We designed the key expander with a small number of latches in order to keep it small in size. The temporary storage T, which is used during the key expansion procedure, is implemented in the SRAM. For this reason, the key storage has 64 entries of 32 bits data. Key expansion takes 752 to 848 cycles depending on the value of the key.

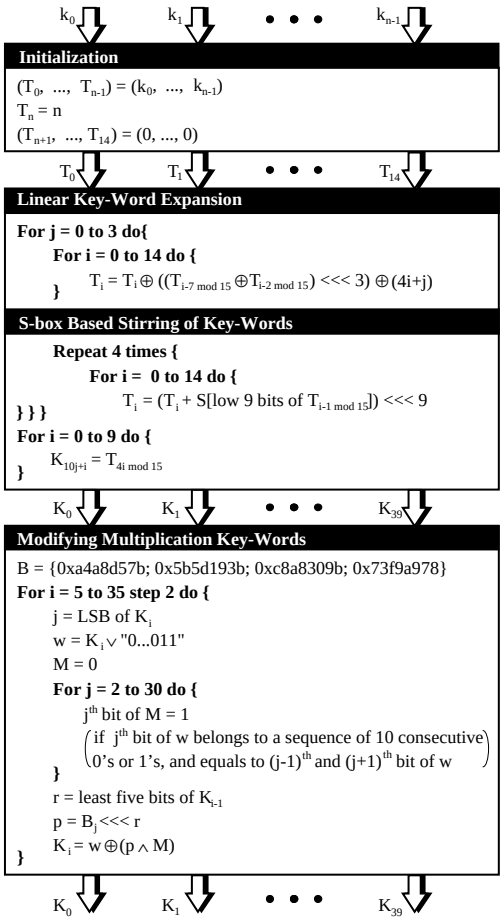


Figure 7. Key expansion procedure.

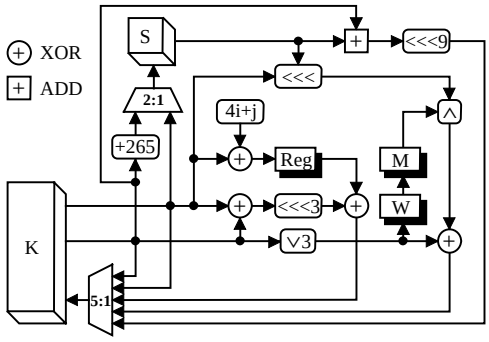


Figure 8. Key expander data path.

3. High-Speed Adder and Multiplier

3.1. High-Speed Adder

In this section, we first explain the design of a high-speed adder employing a combination of carry-skip [2] and carry-select [3] techniques used in the RSA encryption LSI [4]. This adder is used in the E-function and in the last stage of the multiplier. It is one of the most critical parts affecting MARS hardware performance.

Figure 9 shows the basic structure of the adder. It consists of ripple-carry adder blocks where each successive block is one bit longer than the block immediately below along with a carry-skip path jumping over each adder block. The delays in the ripple-carry adders and the carry-skip path are well balanced so that every carry propagates from the LSB to the MSB without waiting for the results from the other blocks. To simplify the figure, a full adder cell FA is used in the first bit of each adder block. It can be replaced by a half adder cell in the actual implementation.

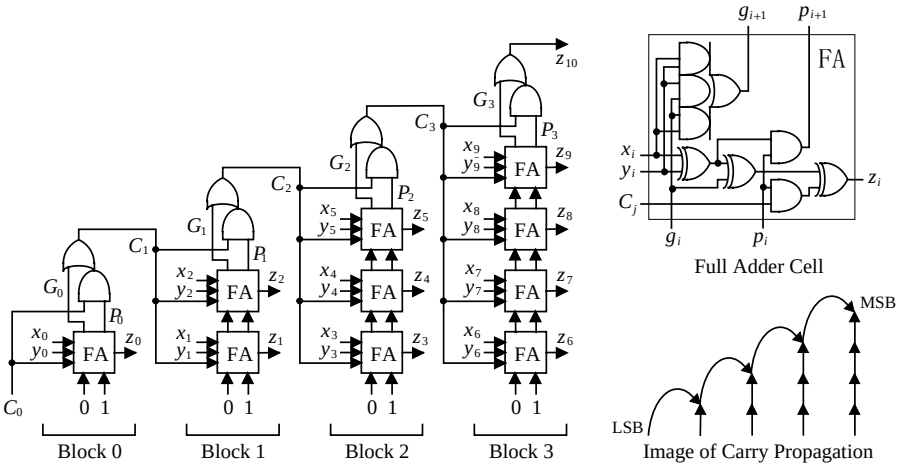


Figure 9. High-speed adder.

If two or more bits of x_i, y_i and g_i are '1' in the i-th full-adder cell, carry $g_{i+1} = 1$ is generated and fed to the next cell. The cell never generates a carry if both x_i and y_i are '0', regardless of the input g_i. If $g_{i+1} = 0$, either x_i or y_i is '0' and the other is '1', it will generate a carry if carry $C_j = 1$ comes up from the lower ripple-carry adder block j-1. For example, when (x_3, x_4, x_5) = (1, 1, 1) and (y_3, y_4, y_5) = (0, 0, 0), block 2 does not generate carries g_4, g_5, g_6 (= G_2). However, if the carry $C_2 = 1$ reached the block, the carry output C_3 immediately becomes '1.' This means that the carry C_j can skip over the blocks one after another by pre-calculating a condition between x_i and y_i in each adder block j. The condition is defined by

$$P_j = \prod_i x_i \oplus y_i = 1, \text{ where } \oplus \text{ is XOR.}$$

By making the adder block size bigger toward the MSB side, the propagation time of P_j and C_j are equalized, and therefore the total delay time is minimized. Output z_i initially holds a sum as if the block carry C_j is 0, and is inverted by the XOR gate if $C_j = 1$ comes up later.

3.2. High-Speed Multiplier

A standard n-bit × n-bit multiplier gives a 2n-bit result by repeatedly summing up the n-bit partial product rows. The multiplier used in MARS is not required to calculate the higher half of the result, as shown in Figure 10, so it is faster and smaller than standard multipliers. The high-speed techniques described in this section, however, can be applied to any multiplier. Figure 11 shows a Wallace tree [5], which is an adder cell array commonly used in a multiplier to reduce the number of partial product rows. The tree takes three rows and produces one carry row and

one sum row, so the full adder cell, FA, is called "3:2 compressor." This reduction is repeated until there are only two partial product rows, which are added together with a high-speed carry-propagation adder. Several tree architectures, which use 4:2, 6:2 and 9:2 compressors, were proposed [6][7] to optimize the critical path of this tree, but these compressors basically consist of 3:2 compressors. Booth encoding [8] is widely used to reduce the number of partial products, but it is a kind of 4:2 compression technique and does not change the tree structure. Oklobdzija et al [9] suggested that not all inputs and outputs from a compressor contribute equally to the delay, and the difference in using 4:2 and higher order compressors is not in the structure of the compressor but in the way they are interconnected.

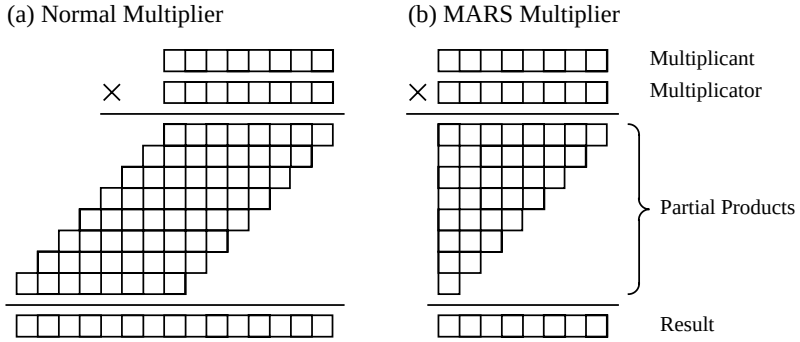


Figure 10. Partial products in MARS multiplier (n = 8).

In Figure 11, the input signals x and y of the full adder FA pass through two XORs to the output s, but the input c_i goes through only one XOR gate. The full-adder FA and half-adder HA located at the later stage of the tree are shaded in the figure. The delay profile of the tree is shown with the same shading. Here, all the XOR, NAND and AND gates are assumed to have the same propagation delay. The two signals fed into the adder at the bit-5 location come from the third-stage half adders marked with '*,' but the right signal arrives earlier than the left one. In addition, the propagation delay from an input to an output varies with the types of gate and input pin locations. For example, AND usually operates faster than XOR, and NAND is faster than AND. For that reason, we developed an optimal Wallace tree generation program in consideration of six delay propagation paths of a full adder (combination of the three inputs to two outputs) and four paths of a half adder, based on a 0.18μm IBM CMOS standard cell library.

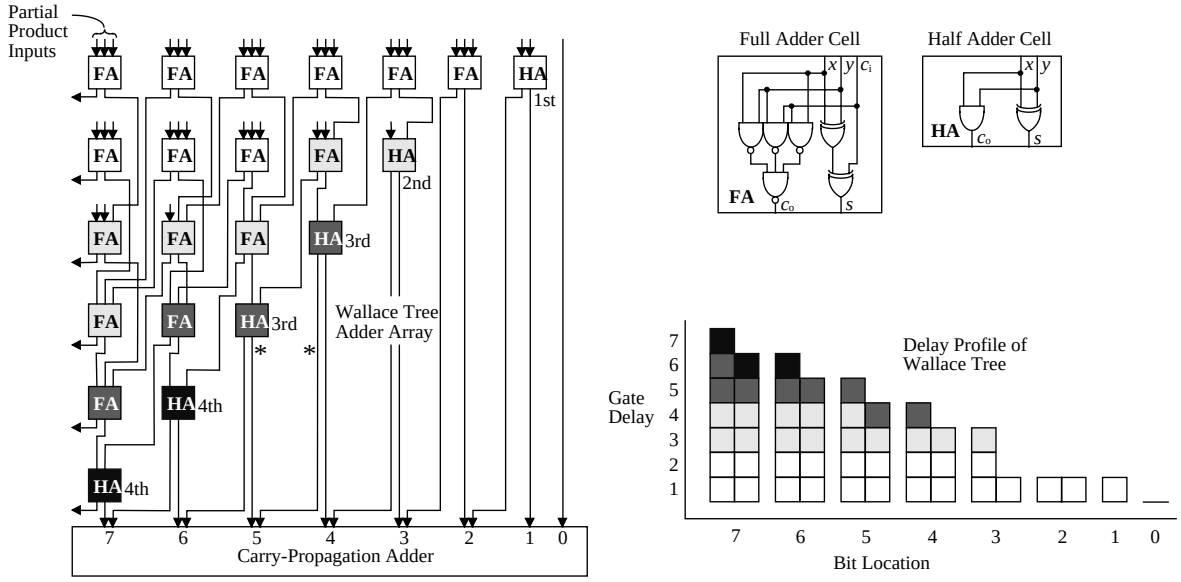


Figure 11. MARS multiplier using Wallace tree and its delay profile (n = 8).

Since the delay of the final carry-propagation adder is an addendum to the Wallace tree delay, the adder should have the optimized carry-propagation path for the tree delay profile. At the same time, we should consider the adder structure to determine the tree interconnection. Figure 12 shows the carry-propagation path in high-speed adders with equal and non-equal input signal arrival profiles. Both adders are identical to the one shown in Figure 9. The adder exhibits the best performance for the equal input profile (a). In case (b), the carry skipping over the adder blocks, though carry generator $C_{GEN,}$ has to wait until the carries propagates from the ripple-carry adder blocks. This is due to the slow input signals. In other words, an adder which is faster than the input delay slope is not needed. A simple ripple-carry adder can run fast enough in this case. The input signals at bit 4, 8 and 9 arrive very quickly, but these fast inputs also waste time waiting for the carry propagation from the next adder cells. To optimize performance of the multiplier, we have to make the positive delay slope gentle, and make the top of the hill as low as possible in the Wallace tree. This is achieved by optimizing the connection between the full adder and half adder gates according to their pin-to-pin internal delay profiles.

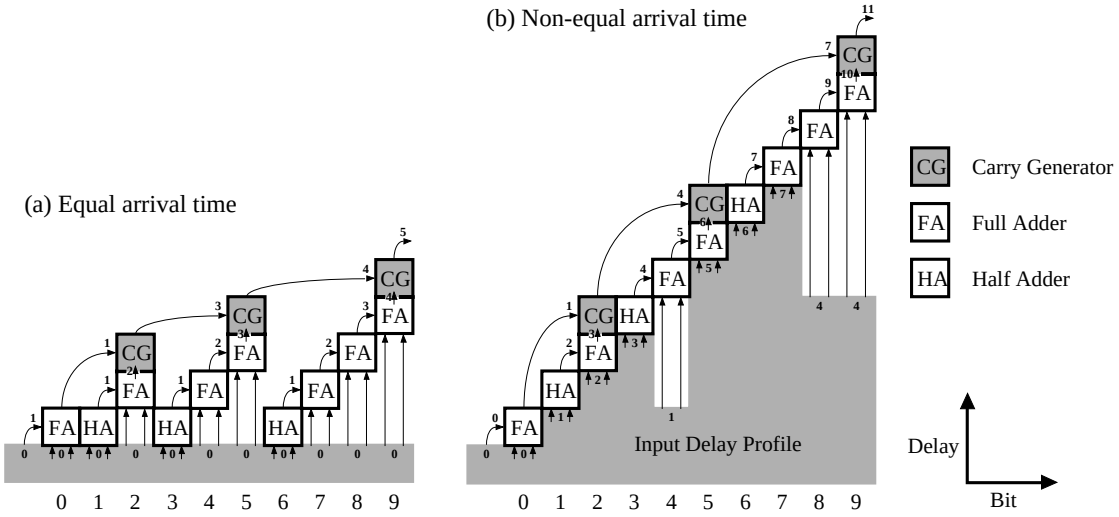


Figure 12. Carry propagation in high-speed adder on equal and non-equal input signal arrival profiles.

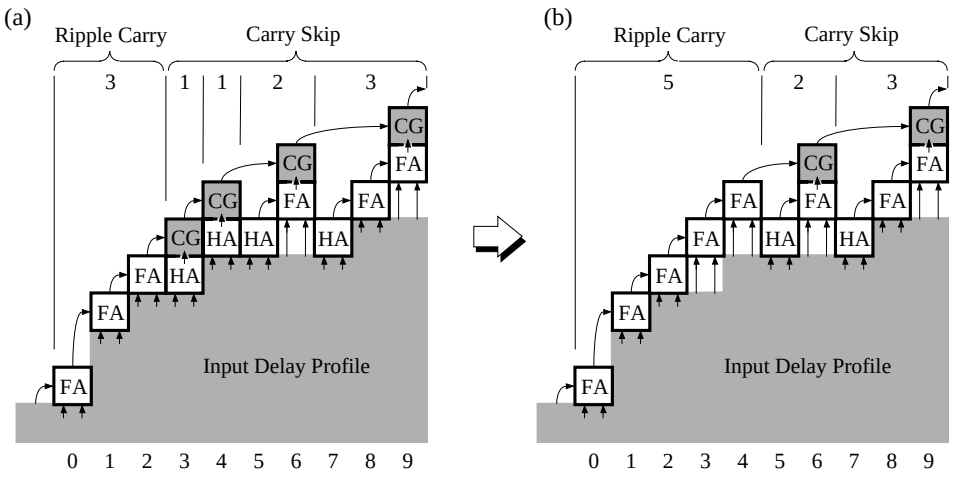


Figure 13. Adder selection over input delay profile.

Figure 13 shows an example adder structure with a positive delay slope profile. From bits 0 to 2, the slope is steeper than one FA delay, so a ripple carry adder is chosen for this part. A carry skip adder with bit blocks 1-1-2-3 is used after bit 3, in example (a). The operation of one half adder HA with a carry generator C_{GEN} is identical to that of one full adder FA, so they are replaced in example (b) to simplify the structure.

In Figures 12 and 13, the input delay time in each bit location is defined by a multiple of one adder cell delay, thus it is not difficult to optimize the adder structure. Actually, as shown in Figure 14(a), there is slack time between the input signals C and P into C_{GEN} at the 9th bit. In case (a), C is generated earlier than P, and waits for the arrival of P at C_{GEN}. When we move the location of C_{GEN} one bit left (to bit 8) so that the carry C does not waste time, the signal P has to wait instead. It is not clear which choice is better until the final adder cell is placed, and we have to choose the right combination of bit locations where C_{GEN}s are placed. In case (a), the output signal delay from the C_{GEN} is longer than that of case (b), but the carry C reaches a higher bit. We should keep the slope of the carry path over the adder blocks as gentle as possible. Therefore, the Wallace tree generator should employ a structure that has smaller value of delay/bit shown as the slope of triangles in the figure. If the structures (a) and (b) have the same slope, then we chose the former because it has higher probability to have fewer C_{GEN} cells.

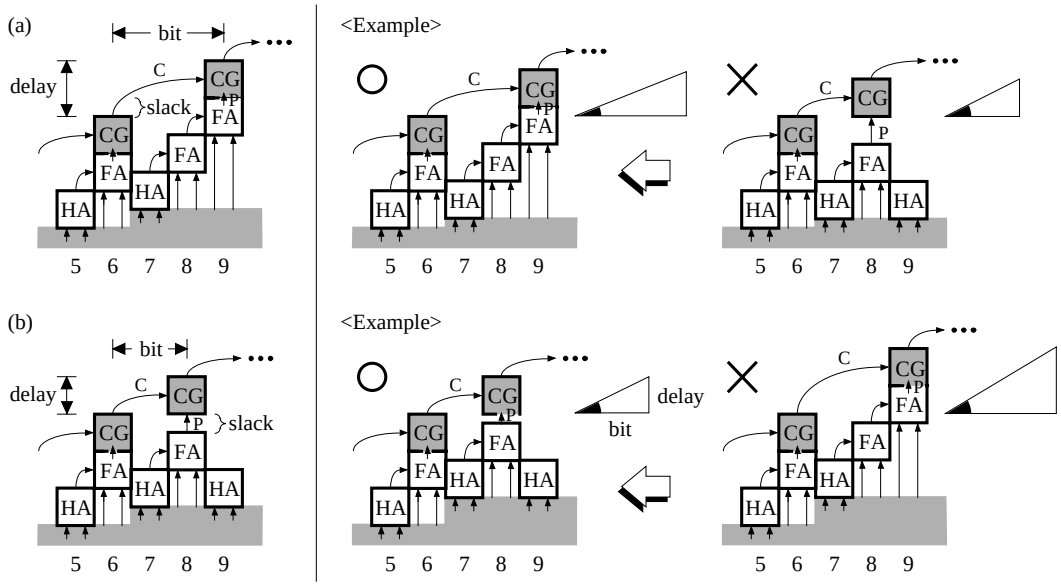


Figure 14. Carry propagation block design.

Figure 15 shows the actual delay profile of the MARS multiplier using 0.18μm IBM copper CMOS technology under nominal (V_{DD}=1.8V, Temp=25°C and L_{eff}=0.11μm) and worst case (V_{DD}=1.65V, Temp=125°C and L_{eff}=0.14μm) conditions. The output delay from the Wallace tree has an almost perfect gentle positive slope. The delay line that looks like a saw blade shows the ripple carry adder blocks. The carry skips over them smoothly. As a result, using the techniques described in this chapter realize a 2.32ns (nominal) to 3.41ns (worst) operation for the 32-bit MARS multiplier with a compact size of 3.2Kgates.

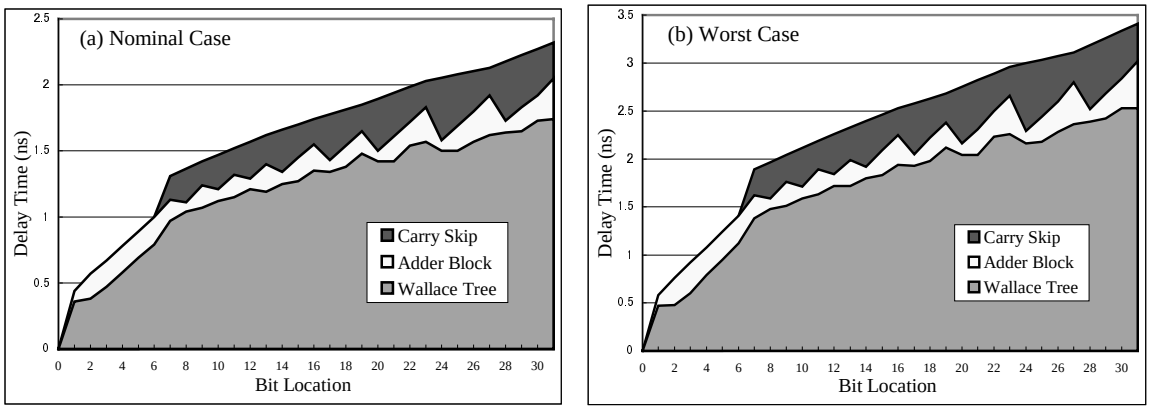


Figure 15. Actual delay profile of the MARS multiplier.

4. Performance Evaluation

We designed MARS hardware including an external interface, the control logic, and the calculation core. All the design files are written in VHDL 93. We synthesized the design using a 0.18μm IBM copper CMOS standard cell technology and evaluated its performance and size.

Table 1 shows the gate size of the logic where one gate is the size of a 2-input NAND. The memory area for the key register and S-box is shown in Table 2. We get data throughputs of 128bits / 34cycles for CBC encryption and 128bits / 18cycles for all decryption and other encryption modes, assuming a four-port SRAM implementation. However, the area of a four-port SRAM becomes larger than that of the logic part. If we do not need the high data rate, the area can be greatly reduced by using fewer-port SRAMs and a mask ROM. A two-port SRAM (one for read and one for write) for the key register halves the memory area while adding only one additional cycle for one 128-byte block encryption process. If the S-box is implemented with a single-port memory or a ROM, the cycles for the forward/backward mixing increase to 34, then the throughput of all encryption and decryption modes becomes 128 bits / 50 cycles.

The critical path delays in the forward/backward mixer and the cryptographic core under nominal and worst case conditions are shown in Figure 16. The longer delay of the cryptographic core, 5.57ns, determines the operation frequency of 180MHz (= 1 / 5.57ns) (122MHz worst case). As a result, we get a maximum data throughput of 677Mbit/s (459Mbits/s worst case) for CBC encryption and 1.28Gbit/s (867Mbits/s worst case) for other modes. All decryption modes achieve maximum throughput of 1.28Gbit/s (867Mbits/s worst case). The throughput and gate sizes for other memory implementations are summarized in Table 3.

Table 1. Logic area

Circuit Block	Gate Size
Key Expansion	2.2K
Enc/Dec Controller	4.5K
Enc/Dec Data path	6.1K
Interface + Memory Controller	1.0K
Total	13.8K

Table 2. Memory area

Function	Type	Gate Size
Key Register (256bytes)	3-port SRAM	6.8K
	2-prot SRAM	4.8K
S-box (2Kbytes)	4-port SRAM	46.2K
	3-port SRAM	30.8K
	1-port SRAM	15.4K
	ROM	6.3K

Table 3. Performance of each implementation

Memory Type		Total Gate Size (Mem+Logic)	Throughput			
Key	S-box		CBC Encryption		Other Encryption and All Decryption Modes	
			Nominal Case	Worst Case	Nominal Case	Worst Case
3-port	4-port	66.8K	677Mbit/s (34cycles)	459Mbit/s	1.28Gbit/s (18cycles)	867Mbit/s
3-port	3-port	51.4K	677Mbit/s (34cycles)	459Mbit/s	677Mbit/s (34cycles)	459Mbit/s
3-port	1-port	36.0K	460Mbit/s (50cycles)	312Mbit/s	460Mbit/s (50cycles)	312Mbit/s
2-port	1-port	34.0K	451Mbit/s (51cycles)	306Mbit/s	451Mbit/s (51cycles)	306Mbit/s
2-port	ROM*	24.9K	263Mbit/s (51cycles)	263Mbit/s	263Mbit/s (51cycles)	263Mbit/s

* 105MHz operation limited by ROM performance

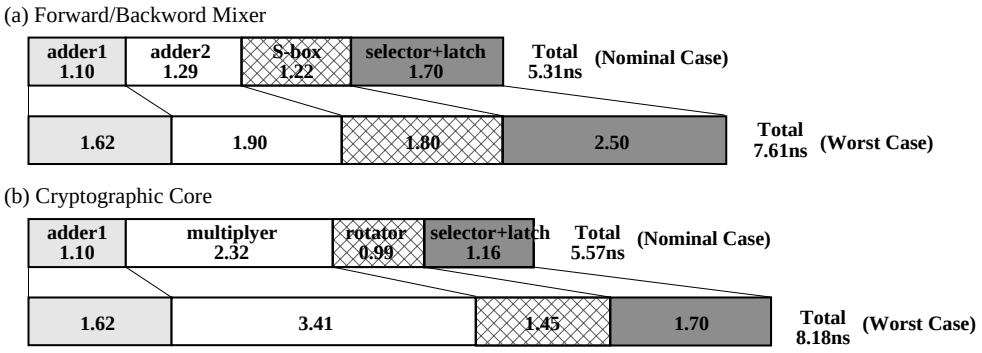


Figure 16. Critical path delay.

The technology chosen for the above estimations is a low cost copper CMOS technology several generations behind the state of the art CMOS technology. As such the performance cannot be directly compared with that of software running on today's high performance microprocessors. If built using a newer CMOS technology the performance can be expected to improve by approximately 60%.

5. Conclusion

We designed MARS hardware and estimated its size and performance. Since the MARS algorithm uses 32-bit additions and multiplications, its performance is highly dependent on the hardware design of the adder and multiplier. We designed multipliers and adders, which fully take into account the carry propagation delay. This work demonstrates that MARS can be implemented efficiently in hardware, both in terms of area and performance. We believe the design point chosen is a reasonable tradeoff of area vs. performance. We do not claim that this is the highest performance MARS design possible. Other tradeoffs may yield faster hardware implementations. Considering the size and performance in both hardware and software along with the very high security, we believe MARS is well suited to serve as the Advanced Encryption Standard algorithm.

Acknowledgement

The authors are grateful to Ms. Carolynn Burwick, Dr. Don Coppersmith, Dr. Mike Matyas, Mr. Hideto Niijima, and Mr. Nev Zunic for their reviews and comments. We also would like to thank Mrs. Akemi Ogura for supporting us generously with CAD tool operation.

References

[1] 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," http://csrc.nist.gov/encryption/aes/round2/AESAlgs /MARS/mars.pdf, Aug. 1999.

[2] M. Lehman and N. Burla: "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units," IRE Trans. Elec. Comput., vol. EC-10, pp. 691-698, Dec. 1961.

[3] O. J. Bedrij: "Carry-Select Adder," IRE Trans. Elec. Comput., vol. EC-11, pp. 340-346, June 1962.

[4] A. Satoh, Y. Kobayashi, H. Niijima, N. Ooba, S. Munetoh, and S. Sone: "A High-Speed Small RSA Encryption LSI with Low Power Dissipation," LNCS 1396, pp. 174-187, 1997.

[5] C. S. Wallace: "Suggestion for a Fast Multiplier," IEEE Trans. Computers, vol. 13, no. 2, pp.14-17, Feb. 1964.

[6] A. Weinberger: "4:2 Carry Save Adder Module," IBM Technical Disclosure Bulletin, vol. 23, Jan. 1981.

[7] P. Song and G. De Michelli: “Circuit and Architecture Trade-Offs for High Speed Multiplication,” IEEE J. Solid State Circuits, vol. 26, no. 9, Sept. 1991.

[8] A. D. Booth: “A Signed Binary Multiplication Technique,” Quarterly J. Mechanical Applications in Math, vol. 4, part 2, pp. 236-240, 1951.

[9] V. G. Oklobdzija, D. Villeger and S. S. Liu: “A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach,” IEEE Trans. on Comp., vol. 35, no. 3, pp. 294-305, Mar. 1996.

Speeding up Serpent

Dag Arne Osvik *

March 15, 2000

Abstract

We present a method for finding efficient instruction sequences for the Serpent S-boxes. Current implementations need many registers to store temporary variables, yet the common x86 processors only have 8 registers, of which even fewer are available for computations. The instructions are also destructive, replacing one input with the output. Alternative versions of the S-box instructions are presented, requiring only 5 registers and also utilizing parallelism. Speedup of C language implementations of 24% is shown on the Pentium Pro Processor, and 42% on the Pentium, both compared to the previously fastest known implementation of Serpent.

1 Introduction

The main aspect of the finalists for the Advanced Encryption Standard is the security level they provide, especially against already known attack methods. Another aspect is the encryption speed they allow in different applications. The goal of this work has been to find ways to improve the execution speed of the Serpent algorithm on the x86 processors, including use of two-way parallel execution.

Serpent[1], being an SP-network (it consists of substitutions and permutations), has two major parts; the S-boxes and the linear transformation. The latter has a simple structure, and is well suited for manual optimization. The S-boxes are 16-element permutations, and are performed in a bit parallel (also known as bitslice) style by simple boolean operations.

2 The problem

The x86 processors, which can be found in nearly every personal computer, have some clearly distinguishing features when compared to more modern architectures. One of these is the small number of registers, only 8. Another is the instruction set, where almost all instructions always modify one of their input registers.

*University of Bergen, Department of Informatics, N-5020 Bergen, Norway. Email address: osvik@ii.uib.no

3 Previous work

Other efforts on optimizing Serpent have centered on the more purely mathematical problem of lowering the number of boolean operations needed to express the S-boxes [2]. Thus those essential properties of the x86 processors have been ignored. The result is a high so-called 'register pressure', meaning compilers have to put temporary variables in memory, issuing load and store instructions in addition to the actual computation. The compiler also gets the job of copying values when needed. One note is appropriate here, though; lowering the number of operations is a much better approach for RISC processors than it is for x86, as RISC instructions don't have to destroy an input value, and those processors typically have 32 registers, making register pressure a non-issue. A comparison of my results to those others (on x86) is given in a later section.

4 Our approach

One possible approach to solving a computational problem is to consider all possible computations, ordered by their length. Searching to the depth needed to find complete solutions in the case of Serpent S-boxes is infeasible using this simple approach, so we need substantial improvements.

4.1 Serpent S-boxes

The Serpent S-boxes are 16-element permutations, implying that they belong to a somewhat special subset of functions in $\{Z_{16} \to Z_{16}\}$. Now, every number from 0 to 15 can be represented by a 4-digit binary number, so these functions map 4 input bits to 4 output bits. They can also be split into 4 functions mapping 4 input bits to 1 output bit, just like any 4-bit number may be split into 4 separate bits. Now recall that any function can be uniquely specified by telling its output value for every allowed input value. In the case of 4-to-1 bit functions this is simply a list of 16 binary digits, given some ordering of the input values.

4.2 Finding solutions

We need some way to transform any 4 input bits into the corresponding 4 output bits using only those instructions available in the x86 instruction set, and in a bit parallel way. We'll use S_2 as an example:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_2(x)$	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2

Now rewrite x and $S_2(x)$ in binary:

x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$S_{2,3}$	1	0	0	1	0	1	1	1	1	0	1	0	0	1	0	0
$S_{2,2}$	0	1	1	0	0	1	0	1	1	0	1	1	0	0	1	0
$S_{2,1}$	0	1	1	0	1	0	1	1	0	0	1	0	0	1	0	1
$S_{2,0}$	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0

Each column in this table contains the bits of some value for x, as well as the bits of the corresponding $S_2(x)$. The set of all columns contains all possible values for x. The number of columns is thus determined by the number of possible inputs, and is *not* related to the word length of any processor.

If we find a way of combining the x_i rows by boolean operations so that we get the $S_{2,i}$ rows, then applying those operations to the bits of an input value x is equivalent to looking up $S_2(x)$. To see how this is actually done, we will look at the execution of an instruction sequence for S_2.

The x86 instructions usable for the S-boxes are these:

Instruction	Effect	C expression
and a, b	$a := a \cdot b$	a &= b
or a,b	$a := a + b$	a |= b
xor a,b	$a := a \oplus b$	a ^= b
not a	$a := a \oplus 1$	a = ~a
mov a, b	$a := b$	a = b

Suppose we have 5 registers, named $r_0, \ldots, r_4$, available for our computations, and 4 of them initially contain our 4 input bits (r_i contains x_i, $0 \leq i \leq 3$). As r_4 is not an input register, we just ignore its previous contents. Thus we have this initial state:

r_4																
r_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
r_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
r_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
r_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The instruction sequence found by the search program (with two-way parallelism shown) is this:

mov r4, r0	and r0, r2
xor r0, r3	xor r2, r1
xor r2, r0	or r3, r4
xor r3, r1	xor r4, r2
mov r1, r3	or r3, r4
xor r3, r0	and r0, r1
xor r4, r0	xor r1, r3
xor r1, r4	not r4

Executing the first line of instructions makes the modifications $r_4 := r_0$; $r_0 := r_0 \cdot r_2$, giving us this new state:

r_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
r_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
r_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
r_0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1

Next, we perform $r_0 := r_0 \oplus r_3$; $r_2 := r_2 \oplus r_1$.

r_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
r_2	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
r_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
r_0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	0

Now things get more interesting. Notice the values in the r_2 row after $r_2 := r_2 \oplus r_0$; $r_3 := r_3 + r_4$.

r_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r_3	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
r_2	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0
r_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
r_0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	0

r_2 is now the same as $S_{2,0}$, one of our wanted output bits.
Executing the next three lines of instruction pairs, we reach this state:

r_4	0	1	1	0	1	1	0	0	1	0	0	1	0	0	1	1
r_3	0	1	1	0	1	0	1	1	0	0	1	0	0	1	0	1
r_2	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0
r_1	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	0
r_0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0

Now r_3 is the same as $S_{2,1}$. The next two lines complete the work:

$r_4 = S_{2,3}$	1	0	0	1	0	1	1	1	1	0	1	0	0	1	0	0
$r_3 = S_{2,1}$	0	1	1	0	1	0	1	1	0	0	1	0	0	1	0	1
$r_2 = S_{2,0}$	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0
$r_1 = S_{2,2}$	0	1	1	0	0	1	0	1	1	0	1	1	0	0	1	0
r_0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0

Thus we have a way of applying the function S_2, using only boolean operations with 1-bit input values. Now remember that the columns were initially just a list of possible input values. So the operations performed are actually independent of the number and contents of the columns. So we may now e.g. extend our table to 32 columns and allow any contents in each of the columns. Then, when the operations are performed, they perform S_2 32 times in parallel. This is exactly what we do on a processor with 32-bit registers.

The search for such solutions basically tries all possible instruction sequences of a given length, looking for rows equal to those of the S-box wanted. Shorter sequences are generally preferred, so we start with a small length, progressing to longer ones when no solution is found. To search for sequences capable of parallel execution, like the one above, we require that an instruction not read the output of an earlier instruction on the same line. It may write to an input register of an earlier instruction, though, as that will in no way affect the outcome of the other instruction.

4.3 Optimizations

Below are short descriptions of the most important optimizations of the search algorithm. Almost all of these avoid removing solutions without keeping an equivalent solution.

- Recursion stops when the register contents can no longer generate a permutation.
- When two instruction sequences are identified as being equivalent, we remove one of them from the search.
- No instruction other than *mov* may make a register contain a copy of the value in another register.
- Unread registers may not be written to by the *mov* instruction.
- Negated registers (those last modified by a *not* instruction) are marked as such, and may not again be negated until they have been read.
- Lookahead functions efficiently calculate a set containing all values reachable in one or two cycles.
- The search is narrowed by requiring an increasing number of result values in the registers as the search goes deeper. This constraint is important for deep searches, but its most strict variant (increasing the required number

as soon as there is at least one sequence that reached it) often drops better solutions, and should be relaxed by postponing the requirement by one or two cycles.

- The instructions are limited to using only 5 registers.

First experiences using the search program with 7 registers available showed most solutions using 6 of those, while others only used 5 registers. Further testing always provided solutions using 5 registers whenever a 6 register solution was found. Given the reduced complexity of the search, and the advantages of having the S-boxes do all their computations in only 5 registers, I chose to limit the search accordingly.

5 Results

The S-box functions chosen from the search results have the properties shown in the table. Cycle count is for running these on processors like the Pentium, with two integer execution units running in parallel.

Function	Instructions	Cycles	Registers
S_0	18	9	5
S_1	18	10	5
S_2	16	8	5
S_3	19	10	5
S_4	20	10	5
S_5	19	10	5
S_6	18	10	5
S_7	20	11	5
S_0^{-1}	19	11	5
S_1^{-1}	19	11	5
S_2^{-1}	19	10	5
S_3^{-1}	18	10	5
S_4^{-1}	20	11	5
S_5^{-1}	19	10	5
S_6^{-1}	17	9	5
S_7^{-1}	19	10	5

The low register pressure of these functions makes their compiled code completely free from loads and stores. So we only load input data and round keys, and store the result. Except for the round key loads, no memory operations are issued during encryption. This is completely different from the S-boxes used in the AES submission package[1], as well as those found by Gladman and Simpson [2], which depend heavily on memory for storage during encryption. Also, the memory footprint of the encryption routines themselves is much reduced; a fully inlined encryption requires less than 4 kilobytes.

6 Optimized implementations

Due to the problem of making C compilers schedule instructions properly for the Pentium, the S-box instructions were also incorporated into assembly routines for Serpent encryption and decryption. The result was then manually tuned for this processor (which may make it slower on other processors). The implementation was made with these constraints:

- The stack pointer register is reserved for its normal use.
- Make the routines suitable as plug-in replacements for the C routines in the AES submittal of Serpent, allowing easy testing.
- One register contains a pointer to the round key table.

Keeping the stack and key table pointers, instead of using them as general purpose registers, allows multiple simultaneous use of the routines, such as in multithreaded environments.

A new set of four round keys is loaded 33 times during an encryption or decryption. Reserving a register to point to the key table avoids having to reload the pointer every time. The ideal solution for performance is to put the round keys on the stack or in a fixed location, as that would free up the key pointer (round keys would be fetched using the stack pointer). But, since the pointer to the round key table is a parameter to the routines we replace, it is needed.

Given these limitations, we still have those five registers needed for the S-boxes, plus one free for whatever use we might have for it, like early loading of a round key. This gives the opportunity to exploit the parallelism of the Pentium nearly to its full extent, thus usually executing two instructions per cycle (some instructions can only execute one at a time). The benefit of one more free register, as could be gained by fixing the location of round keys, will thus be minimal.

7 Performance comparison

Speed testing was done on these computers:

Processor	Clock speed	RAM size	OS
486 SX	33 MHz	20 MB	Linux 2.0
Pentium	100 MHz	64 MB	Linux 2.0
(Dual) Celeron	333 MHz	256 MB	Linux 2.2

The following tables give a comparison of the different implementations of Serpent on these computers. My speed figures in Mbit/s are scaled to given clock speeds, assuming all memory operations are performed in level 1 caches. In the case of Pentium Pro, I compare against the best of Gladman's most

recent numbers. On the others, numbers are compared to those reported by Granboulan [3], using Gladman's code.

- 486 DX/2-50

Implementation	Encryption		Decryption	
	Mbit/s	cycles	Mbit/s	cycles
Gladman's code	0.48	12900		
Osvik	3.8	1650	3.8	1660

- Pentium 90

Implementation	Encryption		Decryption	
	Mbit/s	cycles	Mbit/s	cycles
AES submission	7.17	1605	5.88	1956
Gladman's code	8.56	1290		
Osvik	12.7	907	12.7	905
Osvik, asm	14.4	800		

- Pentium Pro 200

Implementation	Encryption		Decryption		Key setup
	Mbit/s	cycles	Mbit/s	cycles	cycles
AES submission	21.8	1170	20.6	1301	
Gladman	27.0	945	26.9	951	1290
Osvik	33.7	759	33.2	770	1106

The compiler used to compile both my own and the AES submission C code is PentiumGCC version 2.95.2. For my own code, I used the options "-O -mpentium -fPIC -fomit-frame-pointer" on Pentium and "-O2 -mpentium -fPIC -fomit-frame-pointer" on PPro. For the AES submission code I used "-O -mpentium". Other optimization settings I tried reduced the speed achieved. All times are measured including parameter passing, function call and return from the function. Timings on the 486 are not nearly as accurate as the others, as it does not have a cycle counter.

Note: the figures quoted above are for Gladman's results in C using a static array of round keys which frees up an extra register. This only allows multiple concurrent encryptions when they all use the same key. His C++ code, which does not have this limitation, shows a 3% performance reduction.

8 Future directions

- My implementations may be further tuned - actually, I expected the Pentium assembly implementation to come close to 735 cycles for encryption.

While trying to manually optimize the encryption, I found the Pentium to be very touchy regarding tight dependencies involving rotation instructions. Given the Pentium processor's slowdown when executing such instruction sequences, 735 cycles seems to be unreachable. Still, faster S-boxes might exist, as my search has not been exhaustive.

- The key setup function can generate the encryption code with round keys embedded directly in the instructions, thus removing the load instructions and saving upto 66 cycles on the Pentium. This will increase key setup time, though.

- 3-way parallelism on x86 (AMD Athlon). This only requires a (theoretically) simple extension of my current search program. The curious can quite easily verify that S_6^{-1} and S_7^{-1} both can execute in 7 cycles with up to 3 instructions/cycle, as opposed to 9 and 10 cycles on Pentium/.../Pentium III.

- Hardware implementations have a natural emphasis on parallelism. Preliminary results in this area look extremely promising; given 3-input nand and nor gates, and (at most) 2-input versions of other gates, all S-boxes can be performed with a gate depth of only 3. Combined with a depth of 4 for the linear transformation and 1 for key mixing, this indicates that several Gb/s should be possible in CBC mode with common technology. If we can also add 3-input (n)xor, the gate depth of one round is reduced to no more than 5.

- The instruction sets of RISC processors may be viewed as a set of gates from which we can build wide S-box functions. Their lack of 3-input logical operations raises the maximum gate depth needed to 4. That is, given enough parallelism on a RISC (or EPIC) chip, all S-boxes have solutions requiring no more than 4 cycles to execute. This hardware-style RISC optimization will be further investigated in the near future.

9 Acknowledgements

I would like to thank my supervisor, Lars R. Knudsen, for proposing this project, and for his advice and criticisms regarding this article.

My parents and some of my friends and fellow students have been helpful in various ways. Most of the search and timing tests where performed on Odd Egil Nerland's computers. Gisle Sælensminde has made an Ada implementation using my S-box functions, and in the process he made a Python script automating inlining of the encryption functions. This was necessary to avoid stressing the GCC register allocator with these rather intricate S-boxes. My C and assembly implementions were also made using slightly modified versions of his script, saving much work and time.

References

[1] RJ Anderson, E Biham, LR Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard"

[2] BR Gladman:
http://www.btinternet.com/~brian.gladman/cryptography_technology/

[3] L Granboulan:
http://www.dmi.ens.fr/~granboul/recherche/AES/timings.html

[4] Intel Corporation, "Intel Architecture Optimization Manual", Order Number 242816-003, 1997.

Appendix

Below are all the S-box functions selected from the search results. The functions expect their input values to be in r0 .. r3, ordered from least to most significant bit. The contents of r4 are ignored. Output values are given in the registers listed at the bottom of each table, again ordered from least to most significant bit.

S_0						S_0^{-1}					
r3	^=	r0	r4	=	r1	r2	=~	r2	r4	=	r1
r1	&=	r3	r4	^=	r2	r1	|=	r0	r4	=~	r4
r1	^=	r0	r0	|=	r3	r1	^=	r2	r2	|=	r4
r0	^=	r4	r4	^=	r3	r1	^=	r3	r0	^=	r4
r3	^=	r2	r2	|=	r1	r2	^=	r0	r0	&=	r3
r2	^=	r4	r4	=~	r4	r4	^=	r0	r0	|=	r1
r4	|=	r1	r1	^=	r3	r0	^=	r2	r3	^=	r4
r1	^=	r4	r3	|=	r0	r2	^=	r1	r3	^=	r0
r1	^=	r3	r4	^=	r3	r3	^=	r1			
						r2	&=	r3			
						r4	^=	r2			
r1, r4, r2, r0						r0, r4, r1, r3					

S_1						S_1^{-1}					
r0	$=\tilde{}$	r0	r2	$=\tilde{}$	r2	r4	=	r1	r1	$\hat{}=$	r3
r4	=	r0	r0	&=	r1	r3	&=	r1	r4	$\hat{}=$	r2
r2	$\hat{}=$	r0	r0	|=	r3	r3	$\hat{}=$	r0	r0	|=	r1
r3	$\hat{}=$	r2	r1	$\hat{}=$	r0	r2	$\hat{}=$	r3	r0	$\hat{}=$	r4
r0	$\hat{}=$	r4	r4	|=	r1	r0	|=	r2	r1	$\hat{}=$	r3
r1	$\hat{}=$	r3	r2	|=	r0	r0	$\hat{}=$	r1	r1	|=	r3
r2	&=	r4	r0	$\hat{}=$	r1	r1	$\hat{}=$	r0	r4	$=\tilde{}$	r4
r1	&=	r2				r4	$\hat{}=$	r1	r1	|=	r0
r1	$\hat{}=$	r0	r0	&=	r2	r1	$\hat{}=$	r0			
r0	$\hat{}=$	r4				r1	|=	r4			
						r3	$\hat{}=$	r1			
r2, r0, r3, r1						r4, r0, r3, r2					

S_2						S_2^{-1}					
r4	=	r0	r0	&=	r2	r2	$\hat{}=$	r3	r3	$\hat{}=$	r0
r0	$\hat{}=$	r3	r2	$\hat{}=$	r1	r4	=	r3	r3	&=	r2
r2	$\hat{}=$	r0	r3	|=	r4	r3	$\hat{}=$	r1	r1	|=	r2
r3	$\hat{}=$	r1	r4	$\hat{}=$	r2	r1	$\hat{}=$	r4	r4	&=	r3
r1	=	r3	r3	|=	r4	r2	$\hat{}=$	r3	r4	&=	r0
r3	$\hat{}=$	r0	r0	&=	r1	r4	$\hat{}=$	r2	r2	&=	r1
r4	$\hat{}=$	r0	r1	$\hat{}=$	r3	r2	|=	r0	r3	$=\tilde{}$	r3
r1	$\hat{}=$	r4	r4	$=\tilde{}$	r4	r2	$\hat{}=$	r3	r0	$\hat{}=$	r3
						r0	&=	r1	r3	$\hat{}=$	r4
						r3	$\hat{}=$	r0			
r2, r3, r1, r4						r1, r4, r2, r3					

S_3						S_3^{-1}					
r4	=	r0	r0	|=	r3	r4	=	r2	r2	$\hat{}=$	r1
r3	$\hat{}=$	r1	r1	&=	r4	r0	$\hat{}=$	r2	r4	&=	r2
r4	$\hat{}=$	r2	r2	$\hat{}=$	r3	r4	$\hat{}=$	r0	r0	&=	r1
r3	&=	r0	r4	|=	r1	r1	$\hat{}=$	r3	r3	|=	r4
r3	$\hat{}=$	r4	r0	$\hat{}=$	r1	r2	$\hat{}=$	r3	r0	$\hat{}=$	r3
r4	&=	r0	r1	$\hat{}=$	r3	r1	$\hat{}=$	r4	r3	&=	r2
r4	$\hat{}=$	r2	r1	|=	r0	r3	$\hat{}=$	r1	r1	$\hat{}=$	r0
r1	$\hat{}=$	r2	r0	$\hat{}=$	r3	r1	|=	r2	r0	$\hat{}=$	r3
r2	=	r1	r1	|=	r3	r1	$\hat{}=$	r4			
r1	$\hat{}=$	r0				r0	$\hat{}=$	r1			
r1, r2, r3, r4						r2, r1, r3, r0					

S_4						S_4^{-1}					
r1	$\hat{}=$	r3	r3	$=\tilde{}$	r3	r4	$=$	r2	r2	$\&=$	r3
r2	$\hat{}=$	r3	r3	$\hat{}=$	r0	r2	$\hat{}=$	r1	r1	$\mid=$	r3
r4	$=$	r1	r1	$\&=$	r3	r1	$\&=$	r0	r4	$\hat{}=$	r2
r1	$\hat{}=$	r2	r4	$\hat{}=$	r3	r4	$\hat{}=$	r1	r1	$\&=$	r2
r0	$\hat{}=$	r4	r2	$\&=$	r4	r0	$=\tilde{}$	r0	r3	$\hat{}=$	r4
r2	$\hat{}=$	r0	r0	$\&=$	r1	r1	$\hat{}=$	r3	r3	$\&=$	r0
r3	$\hat{}=$	r0	r4	$\mid=$	r1	r3	$\hat{}=$	r2	r0	$\hat{}=$	r1
r4	$\hat{}=$	r0	r0	$\mid=$	r3	r2	$\&=$	r0	r3	$\hat{}=$	r0
r0	$\hat{}=$	r2	r2	$\&=$	r3	r2	$\hat{}=$	r4			
r0	$=\tilde{}$	r0	r4	$\hat{}=$	r2	r2	$\mid=$	r3	r3	$\hat{}=$	r0
						r2	$\hat{}=$	r1			
r1, r4, r0, r3						r0, r3, r2, r4					

S_5						S_5^{-1}					
r0	$\hat{}=$	r1	r1	$\hat{}=$	r3	r1	$=\tilde{}$	r1	r4	$=$	r3
r3	$=\tilde{}$	r3	r4	$=$	r1	r2	$\hat{}=$	r1	r3	$\mid=$	r0
r1	$\&=$	r0	r2	$\hat{}=$	r3	r3	$\hat{}=$	r2	r2	$\mid=$	r1
r1	$\hat{}=$	r2	r2	$\mid=$	r4	r2	$\&=$	r0	r4	$\hat{}=$	r3
r4	$\hat{}=$	r3	r3	$\&=$	r1	r2	$\hat{}=$	r4	r4	$\mid=$	r0
r3	$\hat{}=$	r0	r4	$\hat{}=$	r1	r4	$\hat{}=$	r1	r1	$\&=$	r2
r4	$\hat{}=$	r2	r2	$\hat{}=$	r0	r1	$\hat{}=$	r3	r4	$\hat{}=$	r2
r0	$\&=$	r3	r2	$=\tilde{}$	r2	r3	$\&=$	r4	r4	$\hat{}=$	r1
r0	$\hat{}=$	r4	r4	$\mid=$	r3	r3	$\hat{}=$	r4	r4	$=\tilde{}$	r4
r2	$\hat{}=$	r4				r3	$\hat{}=$	r0			
r1, r3, r0, r2						r1, r4, r3, r2					

S_6						S_6^{-1}					
r2	$=\tilde{}$	r2	r4	$=$	r3	r0	$\hat{}=$	r2	r4	$=$	r2
r3	$\&=$	r0	r0	$\hat{}=$	r4	r2	$\&=$	r0	r4	$\hat{}=$	r3
r3	$\hat{}=$	r2	r2	$\mid=$	r4	r2	$=\tilde{}$	r2	r3	$\hat{}=$	r1
r1	$\hat{}=$	r3	r2	$\hat{}=$	r0	r2	$\hat{}=$	r3	r4	$\mid=$	r0
r0	$\mid=$	r1	r2	$\hat{}=$	r1	r0	$\hat{}=$	r2	r3	$\hat{}=$	r4
r4	$\hat{}=$	r0	r0	$\mid=$	r3	r4	$\hat{}=$	r1	r1	$\&=$	r3
r0	$\hat{}=$	r2	r4	$\hat{}=$	r3	r1	$\hat{}=$	r0	r0	$\hat{}=$	r3
r4	$\hat{}=$	r0	r3	$=\tilde{}$	r3	r0	$\mid=$	r2	r3	$\hat{}=$	r1
r2	$\&=$	r4				r4	$\hat{}=$	r0			
r2	$\hat{}=$	r3									
r0, r1, r4, r2						r1, r2, r4, r3					

S_7						S_7^{-1}					
r4	=	r1	r1	|=	r2	r4	=	r2	r2	^=	r0
r1	^=	r3	r4	^=	r2	r0	&=	r3	r4	|=	r3
r2	^=	r1	r3	|=	r4	r2	=~	r2	r3	^=	r1
r3	&=	r0	r4	^=	r2	r1	|=	r0	r0	^=	r2
r3	^=	r1	r1	|=	r4	r2	&=	r4	r3	&=	r4
r1	^=	r0	r0	|=	r4	r1	^=	r2	r2	^=	r0
r0	^=	r2	r1	^=	r4	r0	|=	r2	r4	^=	r1
r2	^=	r1	r1	&=	r0	r0	^=	r3	r3	^=	r4
r1	^=	r4	r2	=~	r2	r4	|=	r0	r3	^=	r2
r2	|=	r0				r4	^=	r2			
r4	^=	r2									
r4, r3, r1, r0						r3, r0, r1, r4					

Session 8:

"Algorithm Submitter Presentations"

Submitter Statements

IBM Comments

Third AES Conference
April 13, 2000

**Don Coppersmith, Rosario Gennaro, Shai Halevi, Charanjit Jutla,
Stephen M. Matyas Jr., Mohammad Peyravian, David Safford, Nevenko Zunic**

Introduction:

All five of the AES finalist candidates are solid ciphers, with no known weaknesses. It seems likely that any of the candidates would make a good standard. In this short paper, we summarize some qualitative and objective comments on the finalists, and make recommendations for final selection.

General Comments on the Candidates

MARS

MARS has one of the widest security margins, both in terms of number of rounds, and in terms of diversity (as its security relies on a combination of several different "strong operations" and on a heterogeneous structure). MARS is the only candidate with a heterogeneous structure, which was a deliberate design feature to help resist unknown attacks. Also, the design of the round function in MARS lends itself to analysis. In particular, a nearly complete characterization is known for the differential behavior of the round function, and independent analysis has been published.

At the same time, MARS is also a very fast cipher. In fact, in some of the measurements, MARS posted the fastest C and Java benchmarks. In Gladman's C benchmarks, MARS average performance across all key sizes was second only to RC6.

One concern raised about MARS was that it was hard to implement on memory constrained environments. In response to this criticism, the key schedule was tweaked prior to round 2, significantly reducing memory requirements.

Another criticism raised about MARS concerned its complexity. We feel that this was partly due to our extremely detailed presentation and analysis of the algorithm. We subsequently released a simplified description including simplified pseudocode which fits on a single page, (which is included later in this paper). In addition, using implementation lines as a complexity measurement, MARS is ***less*** complex than Twofish, Rijndael, and Serpent.

RC6

RC6 has a simple, elegant round function, and it is the fastest cipher in speed tests. A possible concern about RC6 is that its round function may be "too simple". Specifically, the combination of multiplications and rotation, although providing some excellent properties, is

a "single point of failure" in RC6 (as it does not use S−boxes). Also, RC6 seems to have the lowest security margin of the candidates in terms of number of rounds.

Rijndael

Rijndael is a fast cipher, which is very flexible for implementation. It is important to note that its speed on 256 bit keys is lower than MARS or Twofish.

Rijndael has a round function which is hard to analyze, and a key schedule that makes it easier to mount power attacks. Also, the fact that the round function can be expressed as only a few simple algebraic operations makes one wary of potential algebraic attacks against it.

The structure of Rijndael and Square is new, and not fully understood. In "The Block Cipher Square", Daemen, Knudsen, and Rijmen presented an attack unique to the Square structure, which caused them to increase the number of rounds. The existence of attacks unique to Square call into question Rijndael's long term resistance.

Rijndael's mode with only 10 rounds has a relatively low security margin.

Serpent

Serpent has very wide security margins in terms of number of rounds, and very strong mixing. On the down side, it is quite slow, and it also has a key schedule that makes power attacks easier to mount. As there are other candidates with good security margins, and much faster performance, we feel that Serpent is too slow.

Twofish

Twofish is a flexible cipher in terms of implementation tradeoffs, and it is also one of the fastest ciphers (except for its key−schedule). It has good security margins, and reasonable complexity.

A concern about Twofish is that it is very hard to analyze its security. Its round function was engineered to provide flexibility, rather than to facilitate analysis. Indeed, although a lot of effort has already been invested in its analysis, it is safe to say that the exact properties of the round function are not very well understood. Moreover, the reliance on key dependent S−boxes which are not generated pseudorandomly, makes the analysis even harder.

Another drawback of the key dependent S−boxes is that they are inherently more costly. In Twofish this extra cost can be shifted between the key−setup and the cipher, but nonetheless it is always there. Finally, the key schedule of Twofish makes power attacks easier, since the entire key can be deduced from only the initial whitening key.

Complexity/Size of the Candidates

As mentioned earlier, MARS is actually not a complex algorithm. One way to measure complexity is to count lines needed to implement the cipher. Here are some measurements of Gladman's C code implementations, which can be used to compare complexity:

Cipher	Lines	LOC	Statements
RC6	116	71	86
MARS	424	298	249
Twofish	496	346	224
Rijndael	449	282	212
Serpent	623	479	620

(*Lines* counts the lines in the implementation, including comments and blanks; *LOC* (lines of code) counts only lines with statements, and *statements* counts the number of C statements.) As expected, RC6 is significantly simpler. Surprisingly, Serpent is significantly more complex to implement. MARS, Twofish, and Rijndael fall in the middle, with comparable complexity. In addition, to show the conceptual simplicity of MARS, here is the entire pseudocode for MARS encryption in 30 lines, (counting comments and blank lines).

```
// Forward Mixing
(A,B,C,D) = (A,B,C,D) + (K0,K1,K2,K3)
For i = 0 to 7 {
  B = (B ^ S0[A]) + S1[A>>>8]
  C = C + S0[A>>>16]
  D = D ^ S1[A>>>24]
  A = (A>>>24) + B(if i=1,5) + D(if i=0,4)
 (A,B,C,D) = (B,C,D,A)
}

// Keyed Transformation and E−Function
For i = 0 to 15 {
  R = ((A<<<13) * K[2i+5]) <<< 10
  M = (A + K[2i+4]) <<< (low 5 bits of (R>>>5))
  L = (S[M] ^ (R>>>5) ^ R) <<< (low 5 bits of R)
  B = B + L(if i<8) ^ R(if i>=8)
  C = C + M
  D = D ^R(if i<8) + L(if i>=8)
 (A,B,C,D) = (B,C,D,A<<<13)
}

// Backward Mixing
For i = 0 to 7 {
  A = A − B(if i=3,7) − D(if i=2,6)
  B = B ^ S1[A]
  C = C − S0[A<<<8]
  D = (D − S1[A<<<16]) ^ S0[A<<<24]
 (A,B,C,D) = (B,C,D,A<<<24)
}
(A,B,C,D) = (A,B,C,D) − (K36,K37,K38,K39)
```

Performance, Complexity, and Relative Security Margin

In this section we have collected and summarized some measurements of performance, and complexity, and estimates of security margin. For performance, we use Gladman's C code results [1]. Note that Rijndael's performance varies based on key size. While other papers have analyzed the candidates on other platforms, only performance on the NIST selected reference platform has received adequate analysis and review, so we use those numbers here.

As a simple complexity measurement, we count lines in Gladman's C implementations [2]. As these have all been written by the same person to the same API, with the same style, the line counts indicate relative complexity. For security margin, we use Biham's analysis [3] of rounds divided by minimum secure rounds, to get a ratio, in which large numbers represent higher (better) margins.

Cipher	Speed(Mb/sec)	Setup(Clocks)	Lines	Security Margin
RC6	94.2	1875	116	**1.0**
Mars	69.4	2134	424	1.6
Twofish	68.8	**8493−15616**	496	1.6
Rijndael	50.5−70.3	207−1983	449	1.3−1.8
Serpent	**26.7**	1296	**623**	1.9

In this table, we have **highlighted** values that are less competitive compared to the other candidates. This table makes clear the tradeoffs between speed and margins. RC6 is fastest, with the lowest margin. Serpent is slowest with the highest margin. The Serpent code is surprisingly more complex than the others, while RC6 is, as expected, the simplest code, with the others comparable between the extremes.

Recommendation Summary

RC6 is an elegant, fast, and well analyzed cipher, and would normally be considered the obvious best candidate, but for a standard that is supposed to last twenty years, its security margin is perhaps a bit too close to the edge. If only one candidate is chosen, RC6 is perhaps a bit risky.

Of the other ciphers, Serpent is too slow. Rijndael's structure is new and less well understood, and it has a slight disadvantage in performance with large keys. The security of Twofish is difficult to analyze, given its key dependent S−box, and it has a slight disadvantage in key setup performance. Since MARS has well understood and analyzed components, has a solid security margin, is fast, and does not have the large key or key setup performance problems, it is the best choice.

Should two candidates be selected, we feel that RC6 would be the obvious second choice, since the risk from its low margin would be much less of an issue, given the existence of the other cipher to fall back on. Its simplicity and tiny size make it very easy to add as a second cipher to any implementation.

References:

1. http://www.btinternet.com/~brian.gladman/cryptography_technology/aes2/ index.html
2. http://www.btinternet.com/~brian.gladman/cryptography_technology/aes2/aes.r2.algs.zip
3. http://www.cs.technion.ac.il/~biham/Reports/aes−comparing−revised.ps.gz

RC6 as the AES

Ronald L. Rivest[1], M.J.B. Robshaw[2], and Yiqun Lisa 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 Laboratories, 250 Cambridge Ave., Palo Alto, CA 94306, USA. yiqun@nttmcl.com

Introduction

After more than a year of design and nearly two years of scrutiny, the process to choose the Advanced Encryption Standard is drawing to a close. We are now left with five designs that would each be a good choice as the final AES. These five ciphers have radically different design philosophies and they have very different security and performance properties. No one cipher sticks out as being the natural choice in all respects.

During the design of RC6 our pragmatic aim was to satisfy as many goals as possible while keeping the cipher simple. Only by keeping a cipher simple can one achieve a well-understood level of security, good performance, and a versatility of design that makes the cipher highly adaptable to future demands.

We believe that we have been successful in this approach and developments over the last two years have only served to strengthen our views. We believe that RC6 would make an excellent choice as the final AES.

Security through simplicity

Despite the talk of "margins for security" and "fair" or "minimal" round assessments, the most important measure of the likely security of a cipher is quite simply the amount of scrutiny it has received. Yet it is not clear how much attention the different ciphers have received. Cryptanalysts have full-time jobs teaching in a university or working on a range of unrelated industry projects. Very few, if any, will have looked at more than two finalists in any depth, let alone all five.

A simple cipher is one that is easily described and readily remembered. It will, as a direct result, be analyzed and scrutinized widely [2, 4, 5, 8, 11]. Not only will it receive the greatest quantity of analysis - it will also receive the most accurate analysis. During the design of RC6 we performed what we believe to be one of the most accurate assessments of the security of any of the AES finalists [4]. RC6 is not so complicated that approximating models have to be introduced (as with MARS [3] and Twofish [17]). Instead we were able to get a remarkably accurate view of the strength offered by RC6 using direct analysis[4]. In this way we were

[4] Since it is easy to define simplified and small block-size variants of RC6, the cryptanalyst can perform far more extensive analysis and experimentation.

able to make a careful decision on how many rounds RC6 should have so that we delivered good performance once our security goals had been attained. In the case of some finalists new attacks have improved on the work of the designers. Yet it is a vindication of our approach that when other techniques are applied, as was done by Knudsen and Meier [11] (and also Baudron et al. [2]), 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. Clearly RC6 was designed in the light of experience gained with perhaps the most studied modern cipher, RC5 [14]. 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 [12] or with related-key attacks in the case of Rijndael [7], this is a very important attribute.

The AES effort is so important that we should not be relying on crude and subjective metrics for our decisions. The process of subtracting some arbitrary number of rounds from the number of proposed rounds - arbitrary numbers that might in one case be taken from the designers documentation and in another from direct independent analysis - can be a misleading way of comparing the AES finalists. To quote [18]: "These comparisons are fundamentally flawed, because they unfairly benefit algorithms that have been cryptanalyzed the least." Instead, 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 analytic work that can be used in the assessment of the cipher, and,
- the accuracy of the designers initial estimates.

We believe that on all counts RC6 is most suited to be chosen as the AES.

Performance through simplicity

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. Here RC6 typically offers exemplary performance. Some restricted devices that are currently quite widely deployed are 8-bit based. These might include a relatively insignificant fraction of mobile devices, but would most likely be smart cards. However, 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.

With regards to very cheap smart-cards with old 8-bit processors, it has already been observed [9] that such very cheap smart-cards are vulnerable to system attacks and are inherently insecure. Such insecurities would apply to any of the AES finalists. As a result we should be careful that we do not place too much weight on the performance of a cipher in an environment that is both insecure now and obsolete (perhaps even non-existent) in a few years time. Nevertheless such processors are currently deployed and the AES may well be desired in such applications. The first question we should ask is whether performance is an important issue in such situations? What applications are going to be used on such cheap 8-bit smart cards? Certainly they won't require bulk encryption - at most a few blocks of data will be processed. So, the performance of any of the five AES finalists is going to be adequate.

On a separate issue it is repeatedly claimed (almost to the point of folklore and most surprisingly in [18]) that an implementation of RC6 requires at least 176 bytes of RAM. Yet Keating [10] has already shown that this is not the case and that RC6 can be implemented in around 120 bytes of RAM. So we can conclude that all the AES finalists can be implemented, and can be expected to offer adequate performance, on cheap (insecure) low-end smart cards.

Looking to future architectures, fine-grained estimates today of performance on future architectures really don't seem to be terribly useful. Technology evolves in unpredictable ways (for instance the growing significance of DSPs) and it seems likely that technology will evolve to best support whichever of the AES finalists is chosen. Instead, experience in the area of 32-bit processors shows that there is nothing intrinsically unsuitable about any of the five finalists for future architectures and future designs can be expected to devote significant support to providing the best possible performance from the final AES.

We provide some additional observations.

- 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 or lengthy code that tries to choose among a dozen different optimization strategies.
- 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 [18] many of them will in fact be Java-based, for instance in set-top boxes.

– One possible future trend is the growth of the market [13] for DSPs and/or microprocessors with DSP capability. RC6 not only performs very well on processors of this type [19], but gains its impressive performance without look-up tables which provide additional burdens on memory requirements.

We believe that excellent performance of RC6 on 32-bit processors, the close convergence in performance between simple compiled code and hand-optimized assembly, and outstanding performance in Java and in DSP environments, all make RC6 ideally suited to be chosen as the AES.

Versatility through simplicity

One of the early stated aims of the AES process was that the final cipher be "simple and versatile". For RC6 these were design goals.

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 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. At the other extreme, it is possible that in the near future a 256-bit hash value will be preferred. The most natural way to do this when using an AES candidate as the basis for a hash function would be to change the block-size.

As another example of the flexibility of RC6, the key schedule allows for very long keys (for example up to 1024 bits) to be used without a compromise to performance. This is not that important for encryption, but it does provide extraordinary improvements to the performance of the Davies-Meyer hashing mode [16]; potentially to the point of providing hashing performance comparable to that offered by dedicated hash functions.

Simplicity and versatility go hand-in-hand. Once again, we believe that RC6 would be the most suited finalist to become the AES.

Conclusions

The three most important attributes of the final AES are security, performance, and versatility. With RC6 we achieve all three goals. RC6 is so simple that the full details of the cipher can be recalled at will. Through simplicity we have developed a truly versatile cipher. We have also developed a cipher that offers exceptional performance, and gives the best all-round suitability in Java with all the implications this holds for future applications. Most importantly, though, 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.

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.
2. 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. In Proceedings of *The Second AES Candidate Conference*, pages 53–67. March 22-23, 1999.
3. 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.
4. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. The security of RC6. Available from `www.rsasecurity.com/rsalabs/aes/`.
5. 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.
6. J. Daemen and V. Rijmen. AES Proposal: Rijndael. June 11, 1998.
7. N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. Preprint.
8. H. Gilbert, H. Handschuh, A. Joux, and S. Vaudenay. A statistical attack on RC6. Preprint.
9. S. Halevi. Suggested "tweaks" for the MARS cipher. Submitted to NIST at the end of Round 1 evaluation. Available via `csrc.nist.gov`.
10. G. Keating. Performance analysis of AES candidates on the 6805 CPU core. In Proceedings of *The Second AES Candidate Conference*, pages 109–114. March 22-23, 1999. Available from `www.ozemail.com.au/ geoffk/aes-6805`.
11. L.R. Knudsen and W. Meier. Correlations in RC6. Preprint.
12. F. Mirza and S. Murphy. An observation on the key schedule of Twofish. Proceedings of the Second AES Candidate Conference, pages 151-154.
13. O. Port. Chips for the post-PC era. *Business Week*, Annual Special Issue, page 96, March 27, 2000.
14. 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 from `theory.lcs.mit.edu:80/~rivest/`.
15. R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 Block Cipher. v1.1, August 20, 1998. Available from `www.rsasecurity.com/rsalabs/aes/`
16. M.J.B. Robshaw. Hashing with the AES finalists. Preprint.
17. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit Block Cipher. 15 June, 1998.
18. B. Schneier and D. Whiting. A performance comparison of the five AES finalists. Preprint.
19. T. Wollinger, M. Wang, J. Guajardo, and C. Paar. How well are high-end DSPs suited for the AES algorithms? Preprint.

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

Rijndael for AES

Joan Daemen, Proton World, daemen.j@protonworld.com
Vincent Rijmen, KULeuven, vincent.rijmen@esat.kuleuven.ac.be

1. Introduction

In this document we give a short overview of the reasons why Rijndael should be selected as the AES. We have divided our arguments into four categories:

- **Security:** Rijndael has the same objective security level as the other finalists, and can easily be implemented in a secure way.
- **Efficiency:** Rijndael has a large "performance margin" compared to the other candidates.
- **Design philosophy:** The clear design has many advantages: easy implementable on a wide range of platforms, easy to get confidence in the claimed security level, ...
- **Extensions:** Rijndael is easily extendable to other key and block lengths.

Finally, we discuss the issue of multiple AES algorithms.

2. Security

2.1 Objectively demonstrable security

Until now, for none of the 5 AES finalists, an attack has been published that demonstrates a weakness inherent in the design. Hence, from a cryptanalytical point of view, all 5 ciphers are equivalent.

2.2 Suitability for secure implementation

In software, Rijndael can be implemented using the operations bitwise XOR, table-lookup and 8-bit shifts. Serpent requires no table-lookups but more general shifts and rotations and bitwise boolean operations.

Twofish additionally requires 32-bit addition and both MARS and RC6 even require 32-bit multiplication and shifts over data-dependent off-sets. The presence of these operations makes the latter three algorithms harder to implement in a secure way on smart cards [DaRi99].

2.3 Adding rounds

For all well-designed block cipher, the complexity of published cryptanalytic attacks increases with the number of rounds in the cipher. This has already been taken into account in the Rijndael design: the increasing number of rounds for increasing key lengths assures a growing security marging against cryptanalytic attacks.

In fact, the number of rounds is a parameter that can be increased further, *without a need for any additional specifications*. In applications where the confidence in Rijndael's security doesn't match the importance of the confidentiality/integrity, or in the hypothetical case that an effective attack on Rijndael would be published, a Rijndael version with an increased number of rounds can be used.

3. Relative efficiency

The relative efficiency of the different finalists can be shown by comparing optimal implementations on several platforms. Given the fact that the different design teams have taken different security margins, the question rises how to compare the algorithms on equal footing. One approach is to determine a minimum number of rounds that has to be used in order to resist currently known attacks, and to add some rounds extra [Bi99]. Unfortunately, not all ciphers have been subjected to the same amount of study. Furthermore, there is no consensus on how many rounds one should add to get an adequate security margin. For instance, how should the added security of an extra round of a (generalised) Feistel cipher be compared with a round of an S-P-network ?

On the other hand, the performance of all the algorithms has been evaluated on many different platforms, and all algorithms got their fair share of attention. Therefore we propose to compare the other AES finalists to Rijndael variants with an adapted number of rounds, such that both algorithms execute in the same time. In Table 1 we list for each AES finalist the number of Rijndael rounds (including the implied round key generation) that can be executed in the same time. Nominally, Rijndael has 10 rounds (for 128-bit keys).

We consider the following platforms:

- **Pentium II/Pro:** representative processor for PCs today;
- **Motorola 6805:** representative processor for smart cards today.

Moreover, we give numbers for different amounts of data treated with the same key:

- **many blocks:** indicative if the same key is used for a considerable amount of data (say at least some Kbytes).
- **4 blocks:** indicative if AES is used to secure a small amount of data. In most financial transactions the amount of data that is subject to a MAC is indeed below 64 bytes. This includes electronic purse, debit/credit and ticketing transactions that will be used in timing-critical applications such as public transport and toll-road payment automation.
- **1 block:** relevant if AES is used as the compression function of a hash function, for PIN code encipherment/decipherment or for session/instance key derivation (in smart card, terminal and/or Host security module) typical for payment systems.

Processor	# blocks	source	DES***	MARS	RC6	Serpent	Twofish
Pentium II/Pro	many	[Li00]	-	13	9	38	12*
	4	[Co99]	-	28	15	33	27
	1	[Co99]	-	46	22	36	25
Motorola 6805**	many	[Ke99]	30	30	28	110	23
	4	[Ke99]	32	52	45	107	23
	1	[Ke99]	37	114	91	100	22

Table 1 Number of rounds in Rijndael, given the same number of cycles

* The Twofish design team measures the performance of Twofish with code that has the used key compiled into the executable. We use the code by Aoki and Lipmaa, slightly slower than the self-modifying (!) code by the Twofish team.

****** For Twofish, only the results of the designers are available. For MARS and RC6 we use the implementations for smartcards with massive RAM available. For Rijndael, we average cipher and inverse cipher speed.

******* For DES, the number of blocks is doubled as the block length is only 64 bits

4. Design philosophy

In the Rijndael design, we have tried to keep everything as simple as possible. Complexity has been added only when necessary to thwart attacks. One example is the key schedule, that is very simple and efficient compared to that of other AES finalists.

Other “simplicity” properties include:

- Symmetry in the round transformation and across the rounds,
- Orthogonality of components,
- Absence of arithmetic operations.

These properties lead to a number of advantages that are treated in the following sections.

MARS and Twofish, on the contrary, have both a very complex round function, with many different operations. According to the documentation given by the respective design teams this is partly due to the fact that during the design, whenever complexity could be added ‘at no additional cost', it was added. ‘At no additional cost' should be understood as `no additional cost *on the Pentium Pro*'. On other ‘unknown' platforms [Cl99], these extra operations could be cheaply available, or not.

Serpent introduced asymmetry across the rounds by adopting 8 different S-boxes and asymmetry in the round transformation by having shift (instead of cyclic shift) operations. RC6 has a reasonably symmetrical design. However, it still mixes XOR and arithmetic addication operations and it uses 32-bit multiplication.

Another important advantage of Rijndael is that it was designed right from the start to support 128 bit block lengths. Twofish and RC6 on the other hand, are obviously upgrades from their 64-bit predecessors, respectively Blowfish and RC5, and this shows in the design.

4.1 Symmetry

There is only a single S-box, since until now, no advantage has been demonstrated for the use of different S-boxes (as in Serpent, Twofish and MARS). This S-box is applied in parallel to all state bytes. Similarly, the linear transformation and the round key addition treat all state bytes in the same way and have rotational symmetry. The round function is the same for the complete cipher execution (unlike Serpent and MARS) as the differences in the round keys are considered to introduce sufficient asymmetry. This gives Rijndael the following advantages:

- **Parallelism:** among the finalists, Rijndael is by far the best suited to be implemented on processors with a parallel architecture[Cl99], that is expected to be the architecture of the future (Merced, McKinley, …). Moreover, a dedicated hardware implementation in which the Rijndael round is fully hardwired can give very high speed thanks to its short critical path[DaRi98].

- **Compactness:** the single S-box and the simplicity of the linear transformation allow to code Rijndael in a small number of bytes, relevant on smart cards. Moreover, a minimal dedicated hardware implementation of Rijndael can be built by hardwiring a single S-box and a single 4-byte to 1-byte linear transform[DaRi98].
- **Absence of arithmetic operations:** the description of Rijndael does not make any (hidden) assumptions on the coding of integers as a sequence of bits. One of the advantages of this is that Rijndael is immune for so-called big endian/little endian confusion and conversion problems.

4.2 Orthogonality of the components

In Rijndael, the round function is composed of a number of components each with their own contribution: S-boxes for non-linearity, round key XORing for key dependence and asymmetry, byte transposition for inter-word diffusion and an MDS transform for intra-word inter-byte diffusion. This design feature allows to get more easily a view on the security of the algorithm.

We have provable lower bounds for linear and differential probabilities based on the interaction of these components. These proofs make use of only a few macroscopic properties of the components and leave a lot of freedom on how these properties are actually attained. The advantage of this modular approach is that components may be replaced without affecting these lower bounds as long as the macroscopic properties hold. For example, in the hypothetical case that an attack would be launched that makes use of some specific property of the current S-box, it could be replaced by another one without affecting the lower bounds.

For the other AES finalists, the interaction between the different components is intricate and much harder to analyse and the act of replacing a single component turns a lot of the analysis performed obsolete.

4.3 Confidence

As a consequence of its clarity of design and good performance results, Rijndael attracted by far the most attention from cryptanalysts outside the design team. Although the other finalists seem to have been analyzed quite thoroughly by their own designers, history has shown that `friendly' cryptanalysis is not as effective. A number of attacks on reduced versions has been published. We can conclude that Rijndael has a sufficient security margin, and do so with a high level of confidence.

5. Extensions

Rijndael is the only AES finalist that supports other block lengths than 128 bits, namely 192 bits and 256 bits. Moreover, extensions are defined for all combination of block lengths and key lengths between 128 and 256 bits in steps of 32 bits [DaRi98].

The added value of the longer block lenghts is that the cipher can be used as the compression function of a collision-resistant iterated hash function. Note that a length of 128 bits was considered to be insufficient for SHA-1.

6. Multiple algorithms

The technical reasons for having multiple algorithms for the AES would be the fact that a single algorithm cannot be efficiently and securely implemented on all target platforms, or to have a backup in case the primary algorithm has been broken.

If Rijndael is chosen as the AES, there is no need for an alternative algorithm for the first reason as Rijndael is very efficient on all target platforms. Of course, if MARS or RC6 would be chosen, smart card application developers will see their performance and RAM availability go down and will tend to stick to good old Triple-DES if no alternative AES is available.

In practice, “having a backup in case the primary algorithm is broken” is a very expensive and cumbersome undertaking. It implies coding, testing and integrating both the primary and the backup algorithms in all products and applications where this backup is really taken seriously. If Rijndael is chosen as AES, the “backup” could be a Rijndael version with the number of rounds doubled. In this respect it is worth while to consider the actual risk. For the current standard DES, the most practical attack to date is exhaustive key search, an attack that was already known before its publication. The more sophisticated attacks, such as linear and differential cryptanalysis are very interesting and have learnt us a lot on how to design ciphers, but are no threat in the real world. The design teams of the AES finalist algorithms know their literature and have all used the experience obtained from analysing DES, FEAL, IDEA, … to build their ciphers. Hence, although new attacks may always be found, we think it is unlikely that they will be a security threat in real-world applications, whatever choice is made among the finalists.

7. References

[Bi99] E. Biham, "A note on comparing the AES candidates", AES 2.

[BAK98] E. Biham et al., “Serpent, a proposal for the Advanced Encryption Standard”, AES 1.

[Cl99] C. Clapp, “Instruction-level parallelism in AES candidates”, AES 2.

[Co99] B. Schneier et al., “Performance comparison of the AES submissions”, AES 2.

[DaRi98] J. Daemen and V. Rijmen, “Rijndael”, AES 1. Updated version from `http://www.esat.kuleuven.ac.be/~rijmen/rijndael`

[DaRi99] J. Daemen and V. Rijmen, “Resistance against implementation attacks: a comparative study of the AES proposals”, AES 2.

[Ke99] G. Keating, “Performance analysis of AES candidates on the 6805 CPU core”, AES 2. Updated version from `http://www.ozemail.com.au/~geoffk/aes-6805/`.

[IBM98] C. Burwick et al., “Mars - a candidate cipher for AES”, AES 1.

[RC98] R. Rivest et al., “The RC6™ block cipher”, AES 1.

[Co99] B. Schneier et al., “Twofish, a block encryption algorithm”, AES 1.

[Li00] H. Lipmaa, AES cipher performance cross-table, available at `http://home.cyber.ee/helger`

The Case for Serpent

Ross Anderson, Eli Biham and Lars Knudsen

24th March 2000

Summary

Serpent should be chosen because it is the most secure of the AES finalists. Not only does it have ample safety margin, but its simple structure enables us to be sure that none of the currently known attacks will work. It is also simple to check that an implementation is correct. Although Serpent is not as fast as the other finalists on the 200 MHz Pentium machine used for round 1 benchmarking, this disadvantage largely disappears when we consider the likely platforms and applications of the 21st century. In hardware, for example, Serpent has easily the best performance, while on IA64 it's second.

1 Security

The most important requirement is stated succinctly in the AES announcement [7]: *'The security provided by an algorithm is the most important factor in the evaluation.'*

From the day in September 1997 when we started designing Serpent, we asked ourselves what protection requirements we were trying to meet. We concluded that AES needed to last for a useful service lifetime plus a human lifetime after that. That means at least a century. So we like the AES motto of a *'crypto algorithm for the twenty-first century'*. Also, if Moore's Law runs out sometime this century, then the AES might never be replaced. So the selectors should consider how their choice will look in the twenty-second century and beyond.

1.1 Advances in mathematics

An algorithm may break if someone comes up with a powerful new theory. We do not believe that the history of cryptanalysis is over. Although we have no real idea what the next hundred (or five hundred) years of mathematics will bring, there are three things we can do to future-proof a design.

First, a block cipher should be simple and easy to analyse. The DES algorithm had such a complex description that until the late 1980's no-one appears to have tried seriously to attack it. When they did, differential [5] and then linear [9] attacks were found – both of which can now be explained to bright students in a single 50-minute lecture.

Second, a block cipher should have more rounds than are needed to block today's attacks. Improvements in cryptanalysis usually increase the number of rounds required.

Third, a block cipher should use only well understood primitives. S-boxes and SP-networks have been around for over a quarter of a century, so it is less likely that surprising new attacks will be found on them.

Serpent was designed with all these considerations firmly in mind.

1.2 Engineering issues

Moore's Law may be the most obvious interaction between crypto security and engineering. But assurance is at least as important. If Moore's law continues, then 128-bit keys will be vulnerable in about a century; but many systems fail right now because of design and implementation errors.

Complicated algorithms are hard to implement correctly, and it is harder still to prove implementations to be correct. Serpent's simple design makes verification easier. It is so simple that it can be optimised in high level languages such and C and Ada. So a developer can avoid many of the errors that creep into assembly language routines.

Many secure systems are also vulnerable because of poor random number generators, memory remanence or other engineering failures (e.g., [3]). These risks provide an even more compelling argument for 256-bit keys than either Moore's Law or quantum computers. It would be nice if implementation failures became less common over time, but experience suggests the contrary. As systems get more complex, there are more things to go wrong.

1.3 Public confidence

Ciphers can also be damaged through erosion of public confidence.

Recall the effect which the invention of differential and then linear cryptanalysis had on the standing of DES. Neither of these attacks is practical: there are no DES applications known to us where an opponent might get hold of 2^{40} texts. Indeed, a prudent designer would normally never use any key for a 64-bit block cipher to encipher more than 2^{32} texts. Yet despite the discoverers' strenuous efforts to keep the story straight, differential and linear attacks became translated in the public mind to 'DES has been broken'. It's imprudent to expect the public to distinguish between practical attacks and 'certificational' attacks – attacks which require infeasibly large amounts of data or effort.

We have often been asked why, given that Serpent is secure today with at most 16 rounds, we do not allow 16 rounds – at least for 128-bit keys. The answer is this. Having experienced what happened to DES, we are concerned that, in perhaps 50 years' time, advances in mathematics will lead to a certificational attack on 16-round Serpent. As the other AES finalists have no more margin of safety than 16-round Serpent, they run a similar risk. (That is why we believe that they should have more rounds, rather than Serpent having less.) We think such an attack on Serpent is unlikely. But 'unlikely' isn't enough; the AES algorithm should have the highest achievable level of design assurance.

So we believe that the Advanced Encryption Standard should be 32-round Serpent with 256-bit keys. If people want to use less than 256 bits, or less than

32 rounds, then they should do so only with good reason, and understand that the two issues are orthogonal. The threats against 128-bit 32-round Serpent and 16-round 256-bit Serpent are different.

2 Performance

Many superficial analyses of the AES finalists have concluded that Serpent is half the speed of the other candidates, because we used twice as many rounds as we needed to. This is not accurate.

The three most important aspects of performance are hardware complexity, software speed and memory cost. We have already discussed memory usage extensively in [2]; this is the critical parameter for embedded and smartcard applications. Serpent does extremely well here. We will spend the rest of this section discussing hardware and software.

First, Serpent is the best of the AES finalists in hardware – even with the full 32 rounds. An independent team produced implementations for the Xilinx XCV1000 FPGA of RC6, Rijndael, Serpent and Twofish[1]. Serpent was the only finalist for which a fully pipelined implementation could be fitted into a single chip. Serpent was also by far the fastest, achieving a throughput of 5.04 Gbit/sec, versus 2.40 Gbit/sec for RC6, 1.94 Gbit/sec for Rijndael and 1.71 Gbit/sec for Twofish [6]. An NSA study of ASIC costs predicts 8.03 Gbit/sec for Serpent versus 5.163 for Rijndael, 2.171 for RC6 and 1.445 for Twofish [12].

Second, several AES finalists are heavily optimised for encrypting very large files on the Pentium II. But in most applications, key agility matters more, and this isn't likely to change any time soon.

Gigabit networks already demand encryption of ATM cell streams. This often won't be done in the end systems, as people rely increasingly on boundary control devices such as firewalls or guards to create virtual private networks. This is likely to mean changing the key every three blocks.

In low cost embedded systems, key changes are already common. In [2] we described a typical fielded electronic purse system where each transaction involved ten key set-ups and fourteen block cipher operations.

So we believe that most real applications will have one key change every 1–5 encryptions, and suggest for simplicity's sake that the benchmark should be the one natural in ATM networks, namely the average cost of one key change plus three block cipher operations. On this benchmark, Serpent does not badly across a wide range of platforms, especially the IA-64 architecture which will almost certainly be the standard for the next generation of PCs. According to engineers from Hewlett Packard, the relevant figures are [13]:

	MARS	RC6	Rijndael	Serpent	Twofish	Serpent is:
IA64	2965	3051	504	**2269**	2991	**2nd**
PA-RISC	3409	2686	666	**2415**	3453	**2nd**

[1] Although this team did not implement MARS, there seems no reason to suppose that MARS would do any better than RC6

The above figures are the average clock cycle costs, over encryption and decryption, of one key setup plus three block cipher operations. Even on Pentium, using this benchmark, Serpent is the third fastest algorithm when one combines the published cycle count figures from Gladman [8] and Osvik [11], and fourth fastest combining Worley et al [13] and Osvik. It's second and third respectively with Osvik's latest figures (2531 cycles on a K7). We hope to have stable and comparable figures by the May 15th deadline. NIST's results also show Serpent doing well on Ultrasparc II [4] (though unfortunately without clock cycle counts).

One of the main things to emerge from the extensive testing of round 2 finalists is that some algorithms achieve high throughput at the cost of slow key setup, while others are reasonably key agile. We believe that very many application designers will prefer the latter.

Another point is that some algorithms achieve high software throughput at the cost of high hardware complexity. We believe that the AES should have a simple hardware implementation.

We are not trying to claim that Serpent is the fastest algorithm. Speed was not the primary goal of the AES competition, and we designed Serpent according to the specification from NIST. What we do say is that Serpent's security was not bought at an unacceptable price in speed.

3 Miscellaneous

Much has been written recently about power analysis. One of us is currently doing an implementation of all five finalists on an 8051-based smartcard with no specific power analysis defences. As the bulk of the work is being done by students, full results aren't expected until the end of the academic year. But from what's known so far, we don't expect that any one finalist will be much superior to any other: just that the attack techniques will differ.

The likely solution to power analysis is hardware engineering, and a strong contender is dual-rail logic in which the current drawn is independent of the data. One of us is involved in such a project [10]. Dual-rail design is easier where one only has to worry about the simple logical operations used in Serpent, rather than operations with carry, and especially multiplications. So choosing Serpent as the AES will make the smartcard designer's job easier.

Finally, the claim that Serpent's whole key schedule has to be worked out in advance for decryption is incorrect. It is not necessary to apply the S-boxes during the forward computation.

4 Conclusion

Serpent should be chosen as the Advanced Encryption Standard. It's the fastest algorithm in hardware, and the second fastest in software on the IA-64 architecture. Above all, Serpent should be chosen because it's the most secure of the candidates.

References

1. RJ Anderson, E Biham, LR Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard', submitted to NIST as an AES candidate. A short version of the paper appeared at the AES conference, August 1998; both papers are available at `http://www.cl.cam.ac.uk/~rja14/serpent.html`
2. RJ Anderson, E Biham, LR Knudsen, "Serpent and Smartcards" in *Cardis 98*, Springer Verlag (2000) pp 257–264; also available at `http://www.cl.cam.ac.uk/~rja14/serpent.html`
3. RJ Anderson, MG Kuhn, "Low Cost Attacks on Tamper Resistant Devices" in *Security Protocols – Proceedings of the 5th International Workshop* (1997) Springer LNCS vol 1361 pp 125–136
4. LE Bassham III, "Efficiency Testing of ANSI C implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard", to appear in the proceedings of the 3rd AES Candidate Conference
5. E Biham, A Shamir, *'Differential Cryptanalysis of the Data Encryption Standard'* (Springer 1993)
6. AJ Elbirt, W Yip, B Chetwynd, C Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", to appear in the proceedings of the 3rd AES Candidate Conference
7. "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)", in *Federal Register* September 12, 1997 (Volume 62, Number 177), pp 48051–48058
8. B Gladman, "Implementation Experience with AES Candidate Algorithms", in *Proceedings of the 2nd AES Candidate Conference* (NIST, 1999) pp 7–14
9. M Matsui, "Linear Cryptanalysis Method for DES Cipher", in *Advances in Cryptology — Eurocrypt 93*, Springer LNCS v 765 pp 386–397
10. SW Moore, RJ Anderson, MG Kuhn, "Improving Smartcard Security using Self-timed Circuit Technology", Fourth ACiD-WG Workshop, Grenoble, ISBN 2-913329-44-6, 2000
11. DA Osvik, "Speeding Up Serpent", to appear in the proceedings of the 3rd AES Candidate Conference
12. B Weeks, M Bean, T Rozylowicz, C Ficke, "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms", to appear in the proceedings of the 3rd AES Candidate Conference
13. J Worley, B Worley, T Christian, C Worley, "AES Finalists on PA-RISC and IA64: Implementations and Performance", to appear in the proceedings of the 3rd AES Candidate Conference

Comments on Twofish as an AES Candidate

Bruce Schneier* John Kelsey† Doug Whiting‡ David Wagner§ Niels Ferguson¶

March 24, 2000

1 Introduction

In 1996, the National Institute of Standards and Technology initiated a program to choose an Advanced Encryption Standard (AES) to replace DES. Four years later, NIST is about to choose that standard. We, the authors of the Twofish algorithm, would like to express our continued support for Twofish.

2 Twofish

Twofish is our submission to the AES process. Since first proposing the algorithm in 1998, we have continued to perform extensive analysis of the cipher: both cryptanalysis and performance analysis. We believe that Twofish is the best AES candidate of the five finalist algorithms.

Security: Twofish was designed primarily with security in mind. To date the Twofish round function has proven to be the strongest round function of any of the finalists, with the best known attack being on 6 rounds of Twofish compared to at least 9 rounds for any of the other finalists.

Performance: Twofish is routinely one of the fastest AES candidates; it was designed to have good performance on a variety of hardware and software platforms, instead of being optimized for a single platform. Although Twofish is not the easiest algorithm to implement or optimise, it is amongst the fastest algorithms on virtually every platform when properly implemented.

Flexibility: Twofish is unique in its implementation flexibility. The algorithm can be optimized for bulk encryption, key agility, low gate count, high gate count, or any combination of factors. All of these implementations are completely interoperable.

More interesting than these individual measures is the security/performance ratio of Twofish. Looking at the five algorithms in this manner—normalizing to the largest number of rounds cryptanalyzed is a good metric—Twofish far surpasses the other four finalists.

3 Discussion

The AES process has worked even better than expected. Today we have five good algorithms, and any of the designs would make a good AES standard. (We would recommend increasing the number of rounds for RC6 from 20 to 32, and the number of rounds in Rijndael from 10/12/14 to 18, to get at least a 2x security

*Counterpane Internet Security, Inc., 3031 Tisch Way, 100 Plaza East, San Jose, CA 95128, USA; schneier@counterpane.com.
†Counterpane Internet Security, Inc. kelsey@counterpane.com.
‡Hi/fn, Inc., 5973 Avenida Encinas Suite 110, Carlsbad, CA 92008, USA; dwhiting@hifn.com.
§University of California Berkeley, Soda Hall, Berkeley, CA 94720, USA; daw@cs.berkeley.edu.
¶Counterpane Internet Security, Inc. niels@counterpane.com.

margin—number of rounds greater than the maximum number of rounds that can be cryptanalyzed—as recommended by Lars Knudsen.)

Two of the finalists, MARS and RC6, are not well-suited certain applications, most notably small-memory implementations (e.g., smart cards) and highly key-agile systems (e.g., IPsec). Any one of the other three algorithms—Rijndael (with the extra rounds), Serpent, or Twofish would make an *excellent* standard.

Of the five finalists, Twofish has the best speed/security-margin tradeoff, as well as the most flexibility. With security and speed being the most important criteria (certainly the most talked-about), we believe that Twofish is the best single finalist.

4 More Information

More information on Twofish can be found on the Twofish Web site, at `http://www.counterpane.com/twofish.html`.

NOTES