The Hasty Pudding Cipher: One Year Later

Rich Schroeppel, Puddingmeister

rcs@cs.arizona.edu

June 12, 1999

Abstract: I argue that the Hasty Pudding Cipher should be chosen

as part of the Advanced Encryption Standard because it is the

fastest cipher for bulk encryption on 64-bit hardware! HPC is

twice as fast as any other AES candidate. I report recent work

with HPC, including some new Pentium assembly language timings,

and the fix for David Wagner's equivalent key discovery.

0. The Hasty Pudding Cipher: A One Paragraph Case

The most important future application for encryption will be for

video communications, on stock hardware, which will be 64-bit

machines. The Hasty Pudding Cipher is the fastest cipher for

bulk encryption on 64-bit machines. Using a 512-bit blocksize,

HPC is twice as fast as the closest competitor, DFC, and thrice as

fast as the other candidates. Performance in video applications

is so important that HPC should be the primary AES choice.

1. Introduction

NIST is running a competition [1] to choose a new block cipher to

replace DES. The minimum NIST requirements are for a block cipher

capable of encrypting 128-bit data blocks, with keysizes of 128,

192, and 256 bits. Extra credit is given for flexibility, such as

other keysizes and blocksizes. It is desired that the winning

candidate run well on as wide a range of platforms as possible.

The algorithm definition languages are ANSI C and Java, and the

principal test platform is a 200MHz Pentium Pro, a 32-bit machine,

using the C compiler in the Borland C++ environment.

The First AES Conference [2] was held in August 1998. Fifteen

candidate ciphers were presented.

I submitted the Hasty Pudding Cipher [3]. HPC is the first

Omni-Cipher: It can encrypt any blocksize with any keysize. This

is achieved by sacrificing design elegance, and substituting extra

mixing. The keysize is decoupled from the rest of the cipher by

having an intermediate fixed-size key-expansion table. The key is

hashed to make the table, which is then used in subsequent

encryptions. An optional secondary key (the SPICE) of up to 512

bits is also used for encryption. Concealment of the spice is

optional. HPC offers a security level of 400 bits.

Section 2 of this paper reviews HPC performance. Section 3

discusses Dave Wagner's discovery that HPC has some equivalent

keys. Section 4 reviews the security of HPC. Section 5 predicts

the future. Section 6 explains why HPC is the best cipher for the

predicted future. Section 7 reviews some objections made to HPC.

Section 8 further predicts the future.

2. HPC Performance

2.1. HPC on the Alpha

HPC performs very well on the DEC Alpha, a 64-bit architecture.

For 128-bit blocks, HPC ranks second behind DFC. For 512- and

1024-bit blocks, HPC is twice as fast as DFC and three times as

fast as the other candidates.

On a 300 MHz DEC Alpha, the 512-bit blocksize runs at nearly

250 MHz, making it good for bulk encryption. For direct

comparison with other AES entries, the 128-bit blocksize runs at

91 MHz. The 64-bit blocksize runs at 45 MHz, comparable to DES,

and three times as fast as 3-DES. It will be a good DES plugin

replacement when the option of changing to a longer blocksize is

unavailable.

All of the Alpha code is written in C; the timings use the GCC

compiler. Fixing the blocksize and unwinding the rounds makes a

big improvement in run time; eliminating the spice doesn't help

much. Note the big jump in performance (from 145 to 229 MHz) of

the extended cipher with a fixed 1024-bit blocksize. A similar

improvement can be expected for any large fixed blocksize.

(These timings were posted on the HPC web page in December 1998.)

300 MHz DEC Alpha (64-bit words)

blocksize time time/bit data rate

bits microseconds nanoseconds MHz

tiny

1 1.83 1830 .55

2 2.80 1400 .71

3 2.51 837 1.20

4 2.58 645 1.55

5 3.24 648 1.54

6 5.10 850 1.18

7 7.55 1079 .93

8-10 6.80 680-850 1.18-1.47

11-15 6.20 413-564 1.77-2.42

16-21 4.97 237-311 3.22-4.23

22-31 4.65 150-207 4.73-6.66

32f 4.251 133 7.53

32-35 4.40 126-138 7.27-7.95

short

36-64 1.68 26.3-46.7 21.4-38.1

64 f 1.569 24.5 40.8

64 fu 1.423 22.2 45.0

medium

65-128 1.59 12.4-24.5 40.9-80.5

128 f 1.538 12.0 83.2

128 fu 1.402 11.0 91.3

128 fus 1.44 11.3 88.9

long

129-192 1.71 8.9-13.3 75.4-112

193-256 1.89 7.4-9.8 102-135

256 f 1.688 6.6 152

257-320 2.06 6.4-8.0 125-155

321-384 2.22 5.8-6.9 145-173

385-448 2.38 5.3-6.2 162-188

449-512 2.49 4.9-5.5 180-206

512 f 2.295 4.5 223

512 fu 2.076 4.1 247

extended

513 3.79 7.4 135

1024 7.08 6.9 145

1024 f 4.481 4.4 229

2048 15.1 7.4 136

4096 31.1 7.6 132

10000 87.3 8.7 115

100000 868 8.7 115

1000000 9.43 msec 9.4 106

10000000 149.8 msec 15.0 66.8

key setup 79.5

spice setup .01

Notes:

F The routine has been specialized to a fixed blocksize.

S The spice is shortened to 64 bits.

U The inner loop is unwound.

Decryption time is about the same as encryption time.

In support of my contention that HPC is the fastest cipher, I've

reproduced part of a table from page 55 of the Proceedings of the

Second AES Conference [5]. The table reports measured speeds of

the AES candidates on a 64-bit machine, a 500 MHz DEC Alpha

21164a. The entries are the number of clocks to encrypt one

128-bit block.

clocks/block

Cast256 749

Crypton 499

Deal 2752

DFC 323

E2 587

Frog 2752

HPC 402 (should be 420)

Loki97 2356

Magenta 5074

Mars 507

RC6 559

Rijndael 490

Safer+ 1502

Serpent 998

Twofish 490

In terms of clock ticks per bit encrypted, DFC is 2.52, HPC is

3.28, and Twofish & Rijndael are tied for third place at 3.83.

For bulk encryption, HPC can be used with a 512-bit blocksize,

taking only 623 clocks/block, or 1.22 clocks/bit. This is more

than twice as fast as DFC, and more than thrice as fast as all the

other AES candidates.

How much data must be encrypted to amortize the key-setup cost?

HPC key-setup takes 23,850 clocks. Comparing DFC

(2.52 clocks/bit) versus HPC (1.22 clocks/bit), HPC gains

1.30 clocks/bit. Assuming DFC key-setup is negligible, HPC pulls

ahead after 18,400 bits, or 2300 bytes. Compared with the other

AES candidates, HPC needs 1150 bytes (or less) to catch up.

2.2. HPC on the Pentium

The Pentium x86 architecture presents problems for 64-bit ciphers,

particularly the Hasty Pudding Cipher. The instruction set is

reasonable: for the most part, operations with 64-bit quantities

take only twice as long as with 32-bit quantities. There are

minor drawbacks: Addition of 64-bit numbers takes only two

instructions, but the second instruction can only be executed in

one of the two arithmetic pipelines. The real problem is that

there are so few registers. After setting aside the stack

pointer, the remaining seven 32-bit registers can hold only 3.5

64-bit quantities. On the bright side, this presents near-

infinite possibilities for tinkering with the code, trying to

remove three cycles here for a cost of two there.

A second problem is that, to get the most out of the architecture,

the programmer must use assembly language: C has no provision for

the programmer to talk to the compiler about carry bits, double-

length quantities, or long shifts. In addition, the programmer

must perform sleight of hand with the registers, which is unfair

to the compiler.

The GCC compiler provides the "long long" datatype for 64-bit

integers, and it works well on another 32-bit machine, the Sparc.

Unfortunately, the Pentium GCC has serious optimization problems

with long longs-- it produces mathematically correct code, but

half the instructions are pointless moves.

In any case, NIST required that the cipher be defined in ANSI C,

which doesn't recognize a 64-bit integer data type. The ANSI C

version of HPC uses macros for all the arithmetic operations, and

suffers accordingly: again, mathematically correct results, but

with a significant time and code-size penalty.

Apparently there are better compilers available; Brian Gladman's

results [6,7] are much better than my C results.

2.2.1. New timings for Pentium Pro assembly language

To get an idea of the compiler-induced penalty, I wrote assembly

code for the Pentium Pro to encrypt blocksizes of 128 and 512

bits. I also wrote an assembly-code version of the key-setup

inner loop. The (much better) results are included in the table

below. Reflecting the assembly language advantage over C, my

times for the 128-bit blocksize and for key-setup are somewhat

better than Brian Gladman's. My code is available to anyone

inside the US cryptographic curtain.

Brian Gladman reports 120750 clocks for key setup, 1450 to

encrypt, and 1575 to decrypt. Using assembly language, I've

improved the key setup by 25% to 96000, and encryption by 30% to

1008. (Decryption should be virtually the same as encryption.

This aspect of Gladman's timings are puzzling: he finds a spread

of 5% in decryption times for different key sizes, whereas the

decryption time is actually independent of key size.)

This still leaves HPC in the middle of the pack with respect to

execution time, moving up only a couple of places in Gladman's

ranking. If we go to 512-bit blocks, things improve considerably:

A 512-bit block takes only 1765 clocks, equivalent to to 441 for

a 128-bit quarter block. This would move HPC up to position 5 in

Gladman's table, about 85% as fast as Rijndael, Mars, and Twofish,

but only 60% as fast as RC6.

200 MHz Pentium Pro (using GCC)

blocksize time time/bit data rate

bits microseconds nanoseconds MHz

tiny

1 8.0 8000 .125

2 14.4 7200 .139

3 12.2 4067 .246

4 14.1 3525 .283

5 33.6 6720 .148

6 50.2 10040 .119

7 85.0 12143 .082

8-10 74.0 7400-9250 .108-.135

11-15 64.5 4030-5864 .171-.233

16-21 51.7 2462-3231 .309-.406

22-31 47.5 1532-2159 .463-.653

32-35 42.7 1220-1334 .749-.820

short

36-64 23.0 359-639 1.57-2.78

medium

65-128 a 6.5 51-100 10.0-19.7

65-128 19.8 155-305 3.28-6.46

128 fs 14.4 113 8.89

128 fg 7.3 57 17.5

128 uaf 5.04 39.4 25.4

long

129-192 18.5 96-143 6.97-10.4

193-256 20.1 79-104 9.60-12.7

257-320 23.0 72-89 11.2-13.9

321-384 24.0 63-75 13.4-16.0

385-448 27.7 62-72 13.9-16.2

449-512 29.0 57-65 15.5-17.7

512 uaf 8.83 17.2 58.0

extended

513 39.5 77 13.0

1024 76.1 74 13.5

2048 167 82 12.3

4096 345 84 11.9

10000 871 87 11.5

100000 8.92 msec 89 11.2

1000000 87.7 msec 88 11.4

10000000 961 msec 96 10.4

key setup 802

key setup 604 (Gladman)

key setup 480 assembly

spice setup .1

Notes:

A Assembly code.

F The routine has been specialized to a fixed blocksize.

G Brian Gladman, C code.

S The spice is shortened to 64 bits.

U The inner loop is unwound.

Decryption time is about the same as encryption time.

The new key-setup algorithm is about the same as the old.

Compared to RC6 (the fastest Pentium cipher), HPC-128 is about 25%

as fast. HPC-512 is much better, about 60% as fast as RC6.

2.3. HPC in Java

Alan Folmsbee [8] compared the Java implementations of all fifteen

AES candidates. He evaluated security (avalanche), memory use

(RAM & ROM), and speed on the NIST Monte Carlo & Known Answer

tests. He weighted the factors 4 (avalanche), 3 (RAM), 2 (ROM),

1 (MCT), 1 (KAT), and dotted weights with ranks to develop an

overall point score.

His graphs of avalanche versus round number for all the candidates

are excellent. HPC ranked second in avalanche, behind the ultra-

conservative Serpent. This is in keeping with the HPC design

philosophy, which substitutes surplus mixing for detailed

analysis. As Gladman had already noted, HPC-128 achieves full

mixing in one round. To actually get anything to measure,

Folmsbee divided one HPC round into tenths.

HPC ranked in the mid-range (5th and 7th) on the speed tests,

probably because Java is friendly to 64-bit integers. The test

machine was a 200 MHz UltraSparc with 256 MB of memory.

Overall, Folmsbee ranked HPC 10th. HPC fared badly on RAM (15KB,

14th) and ROM (45KB, 14th). I'm somewhat puzzled by the RAM

numbers, since the key-expansion array is only 2KB, and there are

only ten or twenty integer variables. I can't complain about the

rank, since HPC is near the top in dynamic memory use. Wrt the

ROM size, the entire HPC specification is only 49KB, and it

includes a fair amount of explanatory text. I suspect that

Folmsbee simply used the out-of-the-box Java implementation of

HPC, which includes a user interface, test code, a file encryption

program, demos, etc. Even so, the rank seems about right, since

HPC is more complicated than the other ciphers.

I disagree with the weighting of the factors: Any computer owner

would gladly trade 100KB of memory for a 25% speed improvement.

Today's PC programs are so bloated that the memory used in all of

the AES candidates is so small as to be irrelevant. All the

ciphers together total about .2% of the test machine memory. For

a Java evaluation, these factors should weigh 0. (I doubt

developers gave much thought to minimizing program size-- it's

seldom important.) As an extreme example, Folmsbee ranks SAFER+

third overall: it ranks 9th in avalanche, and 10th in both speed

tests, but is 1st in RAM size and 4th in ROM size. This is silly.

I recalculated the ranking, dropping the RAM and ROM factors; it

becomes RC6, SERPENT, HPC, MARS, E2. I don't think this means

much: Security should be mostly a binary attribute. The Java

speeds reflect the code turned in to meet a requirement, and would

improve with some optimization work. In any case, the eventual

encryption inside Java will be a subroutine, not a block of Java

code.

2.4. Timing Summary

The takeaway message on timing is that HPC is 2-3 times faster

on 512-bit blocks versus 128-bit blocks. On the Alpha, this

moves it from a strong second place to runaway first place. On

the Pentium, it moves up to fifth place. The long blocksize isn't

a serious drawback-- a minimum ethernet packet is about 512 bits;

a typical network data packet is a kilobyte or more.

3. Equivalent Keys in HPC

In March 1999, David Wagner found a way to make equivalent keys in

HPC [9,10]. He discovered that, about 1/256 of the time, 25% of

the bits of a 128-bit HPC key could be modified and produce the

same key-expansion array as the unmodified key. Since HPC is

driven from the array, different keys can make equal encryptions.

Carl D'Halluin, Gert Bijnens, Bart Preneel, and Vincent Rijmen

followed up on this, and found that longer keys could have even

more free bits [11]. This violates the normal understanding of

cipher keys: changing the key, even by only one bit, should give

completely different encryptions.

In the inner mixing loop of the key expansion function, the first

statement is

s0 ^= (KX[i] ^ KX[(i+83)&255]) + KX[s0&255] /\* lossy sometimes \*/

KX is the key-expansion array, and s0 is one of the state

variables of the mixing algorithm. Wagner's observation is that,

when the statement is executed, if "s0 & 255" happens to equal

"i", two of the references to the KX array will fetch the same

value, and some of the bits in KX[i] have no influence on the

value of s0. KX[i] is later overwritten. If the happenstance

equality occurs during the first mixing pass over the

key-expansion array, the KX[i] bits which don't influence s0

become "free bits". The corresponding bits in the key can be set

arbitrarily.

3.1. The Key Expansion Tweak

Fortunately, the flaw is easy to fix. [12] One new line of code

does the trick. The statement

s2 += KX[i] /\* fix Wagner equivalent key problem \*/

is added to the inner mixing loop immediately after the first

statement (above). The fix carries the influence of KX[i] into

mixing state variable s2. The repair is cheap, adding less than

3% to the key expansion time.

4. Security of the Hasty Pudding Cipher

Hasty Pudding has a stronger security goal that the other AES

candidates, 400 bits. (Roughly, this means the opponent can only

score by exhaustive search, or at least 2^400 effort.) This is

achieved with a lot of internal state and by having the mixing

functions create "state-bloom".

Suppose an opponent has a plaintext-ciphertext pair, and is trying

to learn something about the key expansion table. He begins by

guessing some of the internal state of an encryption, and then

filling in the computation both forward and backward as necessary

to match the known plaintext and ciphertext. State-bloom means

that he has to keep extending his guess to keep the computation

going, so that eventually he must guess at least 400 bits of

internal state to make the plaintext-ciphertext connection.

Typically he must guess a large number of values from the

key-expansion table. Each encryption references the KX table at

least 25 times, representing 1600 unknown random bits. [The

opponent might hope that many of the references coincide; I've

taken care to frustrate this possibility.]

An additional strength of HPC is its irregularity.

All the successful analyses of DES have proceeded by analyzing a

few rounds, and extending the analysis to the full cipher. Within

48 hours of Skipjack's public release, Biham had analyzed a

slightly modified version. His attack took advantage of

Skipjack's regularity. HPC doesn't allow this because every round

is a little different: this thwarts differential and linear

cryptanalysis, since a differential (or a linear combination) that

works on one round will fail on the next. Irregularity is a

unique feature of HPC. MARS goes a little in this direction: it

uses two kinds of rounds, and the designers feel this gives them

extra protection against yet-to-be-discovered cryptanalytic

techniques.

4.1. Mixing Analysis

Only HPC-128 received any analytical attention.

Brian Gladman evaluated diffusion for each AES candidate. [7]

He finds HPC has total diffusion in one round.

Alan Folmsbee made a similar evaluation, with the same result. [8]

Folmsbee graphed avalanche versus rounds for each AES cipher.

To have something to show for HPC, he divided the round into

tenths. By his avalanche scoring, HPC ranks second only to

Serpent in total avalanche.

Biham [13] proposed to evaluate "equal security" versions of each

cipher, by determining how many rounds of each cipher were

required for some constant security level, and scaling the time

accordingly. He declined to evaluate the required number of

rounds for HPC, remarking "We also did not analyze HPC, and cannot

predict the minimal number of rounds. We expect that this lack in

predicting the minimal number of rounds of these two ciphers will

not affect the choice of the final AES cipher." He then places

HPC at the bottom of his Table 4, of proposed minimal-round

variants. Unsurprisingly, Serpent ranks at the top of his table.

Avalanche isn't everything, but it has the virtue of being a

reasonably objective measurable criterion. A cipher with

insufficient avalanche will be insecure.

4.2. Novelty: Threat or Menace?

There are two philosophies regarding the AES competition: One

says that new designs should be encouraged, while the other says

that new designs cannot be regarded as safe. The people

submitting old designs subscribe to the second philosophy. One

unfortunate side effect of the AES will be the chilling of

interest in new ciphers: with DES replaced, the market for new

product will dry up. Presumably this will also focus analytical

work on the selected cipher, to the exclusion of new ciphers.

5. The LONG View

The nominal design lifetime for the AES cipher is 20 years.

NIST publicity describes AES as "A Crypto Algorithm for the

Twenty-First Century". Given the inertia of an installed base,

the AES algorithm may well become "A Crypto Algorithm for the

Third Millenium -- All of It".

Consider DES, which is being slowly and reluctantly phased out.

The original planned lifetime for DES was only ten years. When

that period expired, the US Govt certified DES for an additional

five years. DES is very much still with us; even the banks have

made do with triple-DES. The cipher will likely still be in use

ten years from now, more than triple the original plan. If we

look at what actually is forcing DES to be replaced, we come to an

interesting conclusion: Differential Cryptanalysis and Linear

Cryptanalysis have nibbled away at the edges, but they each require

more than a trillion plaintexts to achieve anything. DES was

wounded by the Internet cracking crowd, but the killing blow was

John Gilmore's DES cracker. People don't seem to mind if NSA

reads their email, but the prospect of John Gilmore doing it is

disconcerting. DES has expired because it has a short key length,

a problem recognized from day one.

From this perspective, the lifetime of AES is forever: Although

Marvin, with a brain the size of a small planet [14], may aspire

to someday search a 128-bit keyspace, no conceivable development

in device technology can search 2^256 keys.

It's clear that the AES should not be chosen based on what runs

best on 32-bit machines, but based on machines likely to become

available in the future. We can already see a little way down

that murky road: The next generation of processors will have

64-bit words. We can't particularly count upon fast 32-bit

multiplication, or on fast 32-bit rotate instructions, or on fast

processing of byte-sized quantities. Unfortunately, most of the

AES entries rely on one or more of these operations. HPC is

designed to take advantage of 64-bit words, and makes only minor

use of unfavorable instructions.

32-bit machines will not disappear the moment Intel's 64-bit

Merced chip hits the market. They will take years to phase out,

and will survive forever in some niches. Even 8-bit machines are

still in use! But within a few years, most computers sold for the

home will use 64-bit technology. These machines will be the ones

that protect our network traffic. These machines must perform

well with the chosen AES cipher.

6. The Case for the Hasty Pudding Cipher

The most important future application of encryption will be for

video communications, on stock hardware. Users will disable

encryption if it significantly impacts video quality, so the

encryption must be fast, the faster the better.

Steve Kent, in his public comments [15], emphasizes that the

principal future use of cryptography will be in the high-bandwidth

domain, and that cheap smart cards aren't worth using as an AES

constraint.

HPC is the runaway performance winner on 64-bit architectures.

It's twice as fast as DFC, the other 64-bit entry, and three times

as fast as the other entries.

HPC is not the fastest algorithm on 32-bit machines, but it's good

enough -- Pentium performance is about 60% of RC6, and 85% as fast

as the next three ciphers. Since the AES may well last for a

century or more, it's silly to optimize for today's most popular

32-bit hardware. Tomorrow's 64-bit hardware is already in the

on-deck circle. None of the 32-bit AES algorithms improves

significantly on 64-bit machines; several are actually slower on

same-clock-speed comparisons.

HPC has perfect parallelism: the blocks can be encrypted

independently, because of the spice. Contrast with CBC mode,

which is inherently serial: the output of one encryption must be

available before the next block can be processed.

6.1. Variable Length is Good!

HPC's variable-length block feature promotes new uses of

encryption, allowing portions of a data record or packet header

to be encrypted even when the format doesn't allow for expansion.

No serious block cipher has ever been offered that can encrypt

variable-length blocks. [16,17] This is peculiar, because almost

no one wants to encrypt 128-bit quantities. The fixed-length

block cipher is always used as a building block for encrypting

longer or shorter pieces of data, with standard extension methods

such as Cipher Block Chaining (CBC).

The advantages of a standard fixed blocksize are that it's easy to

design hardware, the software can be elegant, and there's hope of

mathematical analysis. These advantages would be compelling, if

they were free.

However, there are significant costs: It's cheaper to encrypt

long blocks directly, rather than using CBC with a fixed-length

cipher. All-at-once long block encryption is mixed better than

CBC: every bit of the plaintext block affects every bit of the

ciphertext. With CBC, later blocks don't affects earlier blocks.

The standard constructions for encrypting short blocks offer two

unappetizing choices: (a) Expand the block to 128 bits. (b) Use

a simple xor or Caesar encryption. The problems are (a) Data

expansion isn't always an option, and (b) Xor/Caesar encryption

allows an opponent to edit the ciphertext and produce predictable

changes in the deciphered plaintext.

HPC's Spice is an important protection for short-block encryption:

The spice can be different for every bit field, preventing

dictionary attacks.

The no-expansion property of HPC is enabling technology for

another new feature: the ability to encrypt non-integer

blocksizes, where the size of the set of encrypted objects is not

a power of 2. This includes, for example, decimal digits, and

days of the year. HPC provides for keyed (and spiced) "random"

permutations of arbitrary sets. This allows for completely new

applications of encryption. It's also very useful for adding

encryption to legacy systems, since the encryption code can be

grafted onto existing data formats and software, rather than

requiring redesign around the encryption format. Encrypted data

can be printed without deranging the print code. Since only some

fields of data within a record are altered, encryption can be

conditional; the record format provides no clue that a field is

encrypted. Another kind of conditional encryption is also

available: some values within a field may be encrypted, while

others are unaltered.

Summing up, direct encryption of variable-length fields is a

better approach than using extension methods with a fixed-length

cipher.

6.2. Advantages of the Hasty Pudding Cipher

HPC is much faster than other ciphers on 64-bit machines.

HPC is infinitely parallelizable.

HPC can encrypt bit fields without expansion.

HPC can encrypt integer ranges that are not exact powers of two.

HPC can permute arbitrary sets.

HPC can conditionally encrypt field values.

HPC can encrypt 64-bit quantities such as DES blocks.

HPC spice is useful in database applications, and for subkeying.

HPC can encrypt entire files as a unit. Every plaintext bit

affects every ciphertext bit.

No other cipher provides these capabilities. The Hasty Pudding

Cipher is a superior solution to the real engineering problem of

encryption.

6.3. An Unpopular Position

My arguments for HPC have so far not persuaded the majority of the

cryptographic community. It's worth pointing out where I part

company from the rest of the AES cipher designers:

o Irregular rounds are good. Every round of HPC is a little bit

different. This prevents differential & linear cryptanalysis

based on a few rounds from being extended to the whole cipher.

o Use extra mixing, rather than trying to analyze exactly how

much is necessary.

o A variable length cipher is worth having, and a fixed blocksize

is mistaken.

o Encrypting bit-fields is useful.

o Encrypting integer ranges is useful.

o A secondary key provides useful flexibility-- avoidance of CBC,

for example. The concealment-optional feature gives additional

design space for applications.

o We can't count on multiplication instructions, or rotates, or

byte operations in future architectures.

o Trading extra memory & key-setup cost for encryption speed is

good.

o Optimizing for 64-bit architectures makes sense today.

Disregard the installed Pentium base.

o Elegance in cipher design is a trap to be avoided.

7. Some Objections Answered.

In this section, I review some of the criticisms that have been

made about HPC.

Objection: Speed. HPC is generally regarded as slow.

This is because the test machine is a Pentium. HPC shines on the

Alpha, while most of the AES candidates don't. Some AES

candidates execute more slowly on an Alpha than equivalent MHz

Pentium! [18] HPC does even better on long block sizes. I

emphasized this in my AES1 presentation, and posted more Alpha

timings in December 1998. Most evaluations ignored other

blocksizes. A 128-bit block is too small for good bulk

encryption.

Objection: Slow key-setup.

This isn't important. Some authors defined realistic encryption

tasks that amortized key-setup over a reasonable amount of

encryption work, and concluded that key-setup time was irrelevant

to the speed rankings.

There's one \*advantage\* to having a slow key setup: An opponent

will take longer to search a small key space.

Objection: HPC has poor key agility.

Key agility is the ability to switch keys quickly. HPC does badly

by this criterion, since it needs 1000-2000 bytes of material to

amortize the key-setup cost.

Sometimes the need for key agility can be met by changing the

spice, an instantaneous operation. In other situations, such as

setting up a network connection, or processing a transaction, the

key-setup and symmetric encryption times are insignificant in the

face of glacial public-key operations and communications latency.

Key agility is only rarely important.

Objection: HPC is a mediocre hash, because key-setup is slow.

Since none of the candidate AES algorithms is faster than SHA,

hash performance is a tertiary factor.

HPC has two hashing modes available that the other ciphers don't:

(a) The quantity to be hashed can be set into the spice, and a 0

of length equal to the required hash is encrypted.

(b) A long object-to-hash can be concatenated with itself,

encrypted, and trimmed to the required hash size. This works

because every bit of the plaintext affects every bit of the

ciphertext.

Objection: HPC is unanalyzable.

At one level, this is silly: HPC has perfectly measurable mixing,

correlations, etc. It \*is\* hard to track the effect of a

plaintext bit-change beyond one round, but that's good, not bad.

At a deeper level, this complaint is about judgment: is it better

to use a cipher that is understandable in terms of currently known

analytical techniques, and judged immune, or a cipher that is

intractable to current techniques? Which is more at risk from an

as-yet-unknown analysis method? Absent hard information, the

arguments here get pretty metaphysical. I think the complexity of

HPC, and the variation in the rounds are defensive advantages

against unknown attacks.

Objection: HPC won't run in Smart cards.

There \*are\* smart cards with enough oomph to run HPC, but they

aren't at the cheap end of the spectrum. One possible approach

would be to fix the primary key in the card, perhaps using a

different KX table for each card; the spice would then provide the

necessary connection to connection variability.

When the AES criteria were being developed, there was a notion

that a tamper-resistant smart card might contain a secret unknown

to the owner, and that the AES algorithm should protect this

secret. However, it has become apparent that all of the ciphers

are probably vulnerable to key extraction from captured smart

cards: Cheap smart cards are vulnerable to manipulation and fancy

test equipment. Tamper resistance is expensive.

Objection (several): Spice is a bad idea.

Specifically: Spice is a waste of cycles, noone will use it.

I mentioned several uses in my AES1 talk: it replaces CBC,

allowing parallel encryption and decryption; it prevents the

various splicing attacks that CBC is prone to; it allows random

access edits in the middle of a database without chaining-matchup

problems from adjacent records. It can be used to give a

different encryption for every block. It's useful in databases

where you want one key for the database, but every record to

encrypt differently. It's handy when encrypting small bit-fields,

making sure that every bit-field gets a different encryption.

The direct cost of including spice in the encryption is minimal.

There is an indirect cost, which is that the subciphers must

always do enough mixing that all 512 bits of the spice are deeply

involved and well mixed. This doesn't matter for the longer

blocksizes, but has a small impact on performance of the 128-bit

blocksize. On the other hand, we want lots of mixing.

Specifically: I don't see a specific attack but don't like the

idea of an opponent using the spice to probe around

inside the cipher. [19]

The HPC documentation warns against allowing the opponent to mount

a chosen-spice attack. If this is actually a possibility, the

duplicate-shortened spice of the next reply may help.

Ordinarily the spice values are defined by the application

designer, and an opponent has no influence over the spice.

Specifically: Someone might capture a tamperproof device

containing HPC and a protected primary key, but be

able to probe the device by loading different

spices into it and watching how the encryption

changes, and thus discover the protected primary

key.

(I find this scenario hard to credit.) One counter-measure would

be to design the device to use multiple copies of a shortened

spice, say four copies of 128-bits. Then the prober can't exert

enough control to learn anything. Another alternative is to hash

the spice before using it; then the prober can't do useful

experiments.

Objection: HPC is a new design, and can't be trusted.

This is really a philosophy problem, not subject to argument.

However, HPC is built from the same computer instructions that

people have been using to make ciphers for forty years, used in

pretty much the same way. For the most part, I've reused ideas

that have been around a long time.

Objection: HPC has no theory.

The theory is that enough mixing does the job. The mixing is

measurable, and equals or exceeds most of the other candidates.

Objection: The key-expansion array is too big.

Two kilobytes or even ten is irrelevant on any conventional

computer, or any hardware device with significant memory. Every

on-chip cache can hold the 2KB required for a single subcipher.

Even Palm Pilots come with megabytes these days. The KXA will be

a problem for cheap smart-cards. This is insufficient reason to

select a cipher that's inferior on 64-bit machines.

Objection: Code size.

Binary code size is completely irrelevant for today's computers.

Objection: HPC is complicated and hard to implement.

HPC is easier to implement and test than, say, bignumber division,

which is widely used. Very few people write their own encryption

routines. Almost everyone will use an implementation written by

someone else. The worldwide cost of ten independent

implementations pales beside the economic gain from fast bulk

encryption.

Objection: HPC compiles badly with several compilers & platforms.

This doesn't matter. As Schneier points out, tuned assembly code

will be used whenever speed is important.

Objection: Odd-sized blocks won't decode properly. [20]

This is incorrect. The fragment register operations are fully

reversible. Decryption has been verified to operate correctly.

The delivered version of HPC contains test code for blocksizes

1-1200 bits, and the test only takes a few minutes to run.

Objection: When encrypting odd-sized blocks, the fragment isn't

mixed enough. The cipher could be attacked by

assuming intermediate values for the fragment. [21]

In fact, the fragments are thoroughly mixed. Assuming

intermediate values for the fragment won't help the attacker,

because of the state-bloom: He must still guess too much

additional internal state to link the internal encryption steps

together.

Objection: Any of the AES candidates can act as a DES replacement

and encrypt 64-bit quantities. HPC isn't unique in

this regard.

This is wrong. The objector planned to append 64 bits of 0 to the

plaintext to make up 128 bits, and use the first 64 bits of the

ciphertext as the encrypted DES equivalent. I'm not sure how he

planned to decrypt.

Objection: Any block cipher can be used to encrypt bit-fields.

HPC isn't unique in this regard.

The objector usually plans to use his block cipher as a random

number generator, and to xor a few bits with the plaintext

bit-field. This is ok in exactly the same circumstances as a

one-time pad is acceptable: not very often. It allows an opponent

to make changes in the ciphertext and know how the deciphered

plaintext will be affected. It's completely unsuitable for a

database, where the key is used multiple times. It also makes a

crummy base encryption for the most interesting application of

bit-field encryption, encrypting non-power-of-two integer ranges.

Objection: Any cipher can be used to encrypt integer ranges that

are not powers of two. HPC isn't unique in this

regard. [22]

The offered encryption uses the cipher to generate a random number

(modulo N, the non-power-of-two) which is then added to the

plaintext (mod N) -- a Caesar cipher. This has the same

one-time-pad problems as the bit-field scheme above: An opponent

can edit the ciphertext with known affects on the decrypted

plaintext.

Objection: Non-powers-of-two won't encrypt or decrypt properly;

the algorithm might loop.

There's a theorem guaranteeing the algorithm won't loop, and

will encrypt and decrypt correctly. Moreover, if the bit-field

encryption is a random permutation, the derived integer-range

encryption is also a random permutation.

The Unstated Objection: Ugliness. HPC is prettier than Emacs,

but not by much.

Appearance doesn't matter, in the face of the advantages.

Encryption is an engineering challenge, not a beauty contest.

The overriding criterion is getting the job done, not how pretty

the solution is.

7.1. Some good things have been said about HPC.

Don Johnson [23] argued that HPC should make the shortlist on the

grounds that the design is different from the other ciphers. This

is exactly the reason that other people think it should not be

considered -- philosophy makes a big difference in risk estimates.

John Callas, in a public comment [24], remarked that HPC was the

only philosophically thought-provoking cipher, since it raised the

issue of encrypting bit-fields.

Brian Gladman remarked that HPC and DFC should not be ruled out,

since they will improve on 64-bit machines. Several authors point

out that DFC and HPC are hard to evaluate because they use 64-bit

designs.

7.2. Outrageous Fortune

I'm a big fan of the one-big-file approach to programming.

There's less to lose track of. The standalone version of HPC

doesn't even have a .h file. NIST required one, based on their

template, so I set up the submission accordingly. As Murphy would

have it, hpc.h was accidentally omitted from the code distribution

cdrom. NIST included a two-page paper version with the cdrom, but

they couldn't make it available on their web page because of US

export rules. Many folks weren't able to compile HPC.

I gave out around twenty copies of the HPC program. The majority

of inquiries were motivated by either curiosity or completeness.

One requestor said he wanted to use something different, on safety

grounds. I turned away a few foreign inquiries.

7.3. Miscellaneous Comments

NIST got a late start on replacing DES, and is now working on a

very tight schedule. In consequence, the evaluations were rushed.

Many evaluators chose to focus only on the leading candidates.

Six months is hardly enough time to analyze one cipher, let alone

fifteen. In some cases, the evaluation papers have been posted

early, allowing time for discussion and corrections before the

final papers were presented.

All of the evaluations were conscientious, even heroic, attempts

by the various investigators to measure the ciphers, but each

limited his effort in various ways. Brian Gladman, who

implemented all fifteen candidates from the specifications must

especially be commended for yeoman service.

One-size-fits-all seems to be too wide a range for a single AES

winner to encompass: Several of the submitters have suggested

having multiple winners. Others have suggested dropping smart

cards from the evaluation, on the grounds that they constrain the

design too much. (HPC does very well on 64-bit machines, but

badly on smart cards.) NIST has reserved for itself the option to

do anything at all, including to select multiple winners, to not

select a winner, or even to select an unsubmitted cipher.

Computer manufacturers don't usually put special cryptographic

instructions on the chip. We must work with whatever's good for

general purpose computing. For future hardware, we may count on

the instruction set including Add, Complement, And, and fixed-size

Shift. These are crucial for the computer to do its job.

Everything else is on the table in the battle for speed. HPC uses

a few variable-sized shifts, but that's the only significant

departure from this limited instruction repertoire.

8. Prospects

After the AES process is completed, whichever cipher(s) are

selected will become the standards for global encryption. All the

ciphers perform pretty much the same function, so the losers will

fade into obscurity, joining Marx in the dustbin of history, and,

like Marx, recalled occasionally for target practice. Perhaps the

Hasty Pudding Cipher will escape this fate because it provides

functionality beyond the other AES candidates.

9. References

[1] The NIST AES web page is at http://www.nist.gov/aes.

The web page has pointers to the fifteen AES candidates.

[2] First AES Conference, August 20-22, 1998, Ventura, California.

[3] The writeup for HPC, including overview, specification, test

cases, and operating instructions, is on my web page,

http://www.cs.arizona.edu/~rcs/hpc. This information is also

available on the first NIST cdrom, which I believe is freely

distributed. The code is available from me by email within

the US and Canada. The code (except for the hpc.h file) is

also on the second NIST cdrom, available from NIST, with an

export license required.

[4] Second AES Conference, March 22-23, 1999, Rome, Italy.

[5] Olivier Baudron, Henri Gilbert, Louis Granboulan, Helena

Handschuh, Antoine Joux, Phong Nguyen, Fabrice Noilhan, David

Pointcheval, Thomas Pornin, Guillaume Poupard, Jacques Stern,

Serge Vaudenay, Report on the AES Candidates, AES2.

[6] Brian Gladman, Implementation Experience with AES Candidate

Algorithms, AES2. Gladman implemented all 15 ciphers from

the specifications. A superb work.

[7] More of Gladman's measurements and his cipher implementations

are at http://www.seven77.demon.co.uk/aes.htm.

[8] Alan Folmsbee, AES Java Technology Comparisons, AES2.

[9] David Wagner, private email message, March 13, 1999.

[10] David Wagner, "Equivalent Keys in HPC", presented at the rump

session, AES2, Rome, March 22, 1999.

[11] Carl D'Halluin, Gert Bijnens, Bart Preneel, Vincent Rijmen,

Equivalent keys of HPC,

http://www.esat.kuleuven.ac.be/~rijmen/pub99.html.

[12] The tweak for HPC is at

http://www.cs.arizona.edu/~rcs/hpc/tweak.

[13] Eli Biham, A Note on Comparing the AES Candidates, AES2.

[14] Douglas Adams, The Hitchhiker's Guide to the Galaxy, 1974.

[15] Steve Kent, AES public comments, April 19, 1999.

[16] Stefan Lucks, BEAST: A fast block cipher for arbitrary

blocksizes, http://th.informatik.uni-mannheim.de/m/lucks/

papers/BEAST.ps.gz. Although intended for encrypting

arbitrarily long blocks, this theoretically would also work

for short blocks.

[17] D. Johnson, S. Matyas, M. Payravian, Encryption of long

blocks using a short-block encryption procedure, Nov. 96,

http://stdsbbs.ieee.org/groups/1363/index.html.

[18] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner,

Chris Hall, Niels Ferguson, Performance Comparison of the AES

Submissions, AES2.

[19] Don Coppersmith, message to the HPC thread of the NIST AES

Bulletin Board, August 28, 1998.

[20] Brian Snow raised this objection in the questions after my

AES1 presentation.

[21] Audience objection raised during my AES1 presentation.

[22] Peter Gutmann and Bruce Schneier, sci.crypt newsgroup,

messages dated Aug 25 and Aug 30, 1998, Subject "Re: Live

from the First AES Conference".

[23] Don Johnson, Future Resiliency: A Possible New AES Evaluation

Criterion, AES2 rump session.

[24] John Callas, AES public comments, Feb 1, 1999, on the NIST

AES web page.