Performance

- There are as many different measures of “performance” as there are platforms to measure it on.
- As a standard, AES will have to perform on all of them.
- We concentrate on the common ones and the general ones.
How the Candidates Approached Key Lengths and Performance

- Some algorithms are slower for larger keys.
- Some algorithms have slower key setup for larger keys.
- Some algorithms have slower key setup AND encryption for larger keys.
- Some algorithms have constant speeds and key setup for all keys.
- One algorithm has slower key setup for smaller keys!!!
**Speed on Different Processors**

- Processor architectures stick around forever.
  - The lesson of the past twenty years is that this high-end always gets better, but the low end never goes away.
- The AES standard will have to work on all processors: small 8-bit embedded CPUs and smart cards, 32-bit CPUs and smart cards, 64-bit CPUs, etc., etc., etc.
- Performance on the low end is much more important than performance on the high end.

**Languages**

- Performance is only important in assembly language.
- It makes no sense to compare performance in C or Java.
  - Any application which has speed as a requirement will code the encryption algorithm in assembly.
  - An encryption algorithm is an ideal piece of code to hand optimize.
  - Optimized assembly implementations of AES will be available on the Internet.
- If performance is critical, it will be in assembly.
32-Bit Comparisons

- 32-bit machines will be used forever.
- The Intel Pentium Pro/II architecture has some oddities not present in other 32-bit processors, either low-end processors or other high-end processors.
- Most important is performance on generic 32-bit processors.

## Pentium/ Pro/ II Comparison

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Key Setup</th>
<th>Encrypt Pentium Pro C (clocks)</th>
<th>Encrypt Pentium Pro C (clocks)</th>
<th>Encrypt Pentium Pro ASM (clocks)</th>
<th>Encrypt Pentium Pro ASM (clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast-256</td>
<td>4300</td>
<td>660</td>
<td>600*</td>
<td>600*</td>
<td></td>
</tr>
<tr>
<td>Crypton</td>
<td>955</td>
<td>476</td>
<td>345</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>DEAL</td>
<td>4000*</td>
<td>2600</td>
<td>2200</td>
<td>2200</td>
<td></td>
</tr>
<tr>
<td>DFC</td>
<td>7200</td>
<td>1700</td>
<td>750</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>2100</td>
<td>120</td>
<td>440</td>
<td>410*</td>
<td></td>
</tr>
<tr>
<td>Fling</td>
<td>1386000</td>
<td>2600</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>HPC</td>
<td>120000</td>
<td>1600</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Lok97</td>
<td>7500</td>
<td>2150</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>MagnaIA</td>
<td>50</td>
<td>6600*</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>4400</td>
<td>590</td>
<td>320*</td>
<td>550*</td>
<td></td>
</tr>
<tr>
<td>RC6</td>
<td>1700</td>
<td>260</td>
<td>230</td>
<td>700*</td>
<td></td>
</tr>
<tr>
<td>Rijndael</td>
<td>850</td>
<td>440</td>
<td>291</td>
<td>520</td>
<td></td>
</tr>
<tr>
<td>SAFERt</td>
<td>4000</td>
<td>1400</td>
<td>800*</td>
<td>1100*</td>
<td></td>
</tr>
<tr>
<td>Serpent</td>
<td>2500</td>
<td>1030</td>
<td>900*</td>
<td>1100*</td>
<td></td>
</tr>
<tr>
<td>Twofish</td>
<td>8600</td>
<td>400</td>
<td>238</td>
<td>290</td>
<td></td>
</tr>
</tbody>
</table>

AES candidates’ performance with 128-bit keys on Pentium-class CPUs
**Things to Note**

- Performance varies greatly.
- Some algorithms depend heavily on the particular details of the 32-bit CPU, while others are largely CPU-independent.
- Note that these speeds are for 128-bit keys.

---

**Bulk Encryption versus Real Speed**

- These speeds are for encryption, and do not take into account key setup.
- For bulk encryption this is a reasonable simplification, but not for smaller messages.
- We looked at total performance (key setup + encryption) for different message sizes, for the fastest algorithms (plus Serpent).
## Clock Cycles, Pentium

<table>
<thead>
<tr>
<th>Text Size (bytes)</th>
<th>Crypton</th>
<th>E2</th>
<th>Mars</th>
<th>RC6</th>
<th>Rijndael</th>
<th>Serpent</th>
<th>Twofish</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>73</td>
<td>100</td>
<td>260</td>
<td>146</td>
<td>59</td>
<td>205</td>
<td>175</td>
</tr>
<tr>
<td>32</td>
<td>49</td>
<td>63</td>
<td>147</td>
<td>95</td>
<td>39</td>
<td>137</td>
<td>119</td>
</tr>
<tr>
<td>64</td>
<td>37</td>
<td>44</td>
<td>91</td>
<td>69</td>
<td>30</td>
<td>103</td>
<td>91</td>
</tr>
<tr>
<td>128</td>
<td>30</td>
<td>35</td>
<td>63</td>
<td>57</td>
<td>25</td>
<td>86</td>
<td>70</td>
</tr>
<tr>
<td>256</td>
<td>27</td>
<td>30</td>
<td>48</td>
<td>50</td>
<td>22</td>
<td>77</td>
<td>48</td>
</tr>
<tr>
<td>512</td>
<td>26</td>
<td>38</td>
<td>41</td>
<td>47</td>
<td>21</td>
<td>73</td>
<td>38</td>
</tr>
<tr>
<td>2^16</td>
<td>25</td>
<td>27</td>
<td>38</td>
<td>45</td>
<td>21</td>
<td>71</td>
<td>31</td>
</tr>
<tr>
<td>2^32</td>
<td>25</td>
<td>26</td>
<td>36</td>
<td>45</td>
<td>20</td>
<td>70</td>
<td>25</td>
</tr>
<tr>
<td>2^64</td>
<td>25</td>
<td>26</td>
<td>35</td>
<td>44</td>
<td>20</td>
<td>69</td>
<td>22</td>
</tr>
<tr>
<td>2^128</td>
<td>24</td>
<td>26</td>
<td>35</td>
<td>44</td>
<td>20</td>
<td>69</td>
<td>20</td>
</tr>
<tr>
<td>2^256</td>
<td>24</td>
<td>26</td>
<td>34</td>
<td>44</td>
<td>20</td>
<td>69</td>
<td>19</td>
</tr>
</tbody>
</table>

Clock cycles, per byte, to key and encrypt different text sizes on a Pentium

---

## Clock Cycles, Pentium Pro/II

<table>
<thead>
<tr>
<th>Text Size (bytes)</th>
<th>Crypton</th>
<th>E2</th>
<th>Mars</th>
<th>RC6</th>
<th>Rijndael</th>
<th>Serpent</th>
<th>Twofish</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>70</td>
<td>100</td>
<td>246</td>
<td>118</td>
<td>53</td>
<td>193</td>
<td>132</td>
</tr>
<tr>
<td>32</td>
<td>46</td>
<td>63</td>
<td>133</td>
<td>67</td>
<td>36</td>
<td>125</td>
<td>93</td>
</tr>
<tr>
<td>64</td>
<td>34</td>
<td>44</td>
<td>76</td>
<td>41</td>
<td>27</td>
<td>90</td>
<td>73</td>
</tr>
<tr>
<td>128</td>
<td>28</td>
<td>35</td>
<td>48</td>
<td>28</td>
<td>23</td>
<td>73</td>
<td>64</td>
</tr>
<tr>
<td>256</td>
<td>25</td>
<td>30</td>
<td>34</td>
<td>22</td>
<td>20</td>
<td>65</td>
<td>48</td>
</tr>
<tr>
<td>512</td>
<td>23</td>
<td>28</td>
<td>27</td>
<td>19</td>
<td>19</td>
<td>61</td>
<td>33</td>
</tr>
<tr>
<td>2^16</td>
<td>22</td>
<td>27</td>
<td>24</td>
<td>17</td>
<td>19</td>
<td>58</td>
<td>25</td>
</tr>
<tr>
<td>2^32</td>
<td>22</td>
<td>26</td>
<td>22</td>
<td>16</td>
<td>18</td>
<td>57</td>
<td>20</td>
</tr>
<tr>
<td>2^64</td>
<td>22</td>
<td>26</td>
<td>21</td>
<td>16</td>
<td>18</td>
<td>57</td>
<td>18</td>
</tr>
<tr>
<td>2^128</td>
<td>22</td>
<td>26</td>
<td>20</td>
<td>16</td>
<td>18</td>
<td>57</td>
<td>17</td>
</tr>
<tr>
<td>2^256</td>
<td>22</td>
<td>26</td>
<td>20</td>
<td>16</td>
<td>18</td>
<td>56</td>
<td>17</td>
</tr>
<tr>
<td>2^512</td>
<td>22</td>
<td>26</td>
<td>20</td>
<td>16</td>
<td>18</td>
<td>56</td>
<td>16</td>
</tr>
</tbody>
</table>

Clock cycles, per byte, to key and encrypt different text sizes on a Pentium Pro/II
**Things to Note**

- Algorithms settle down pretty quickly:
  - For a 1K message, speeds are within 15% of fastest speeds.
  - Fastest algorithms for small blocks are Rijndael and Crypton.
  - Note these speeds are for 128-bit keys: Rijndael will be slower with larger keys.

**Hash Functions**

- Block ciphers can be used as hash functions.
- Hash function constructions require one key setup and one encryption per block hashed.
Hash-Function Comparison

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Hash Speed Pentium Pro ASM (clocks)</th>
<th>Hash Speed Pentium ASM (clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast-256</td>
<td>283*</td>
<td>283*</td>
</tr>
<tr>
<td>Crypton</td>
<td>46*</td>
<td>49*</td>
</tr>
<tr>
<td>DEAL</td>
<td>349*</td>
<td>349*</td>
</tr>
<tr>
<td>DFC</td>
<td>245*</td>
<td>1*</td>
</tr>
<tr>
<td>E2</td>
<td>100*</td>
<td>100*</td>
</tr>
<tr>
<td>Frog</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>HPC</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Loki97</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Magenta</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Mars</td>
<td>246*</td>
<td>260*</td>
</tr>
<tr>
<td>RC6</td>
<td>118*</td>
<td>146*</td>
</tr>
<tr>
<td>Rijndael</td>
<td>32*</td>
<td>34*</td>
</tr>
<tr>
<td>SAFER+</td>
<td>193*</td>
<td>222*</td>
</tr>
<tr>
<td>Serpent</td>
<td>193*</td>
<td>205*</td>
</tr>
<tr>
<td>Twofish</td>
<td>132</td>
<td>175</td>
</tr>
</tbody>
</table>

Hash-function performance, per byte, of AES candidates (128-bit key) on Pentium and Pentium Pro/II

Hash Functions and Key Schedules

- Encryption algorithms do not automatically make good hash functions; they must be analyzed.
- Simple key schedules are much efficient, but may also be much less secure.
- Like all measures in this paper, these ignore security.
Biham has invented this measure in an attempt to “normalize” the submissions.

He takes his estimate of the number of rounds that is secure, and then adds a standard two cycles.

This metric is not necessarily useful or interesting.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Rounds</th>
<th>Minimal Secure Rounds</th>
<th>MSR Encrypt Pentium Pro ASM (clocks)</th>
<th>MSR Encrypt Pentium ASM (clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash-256</td>
<td>48</td>
<td>40</td>
<td>500*</td>
<td>500*</td>
</tr>
<tr>
<td>Crypton</td>
<td>12</td>
<td>11</td>
<td>316</td>
<td>358</td>
</tr>
<tr>
<td>DEAL</td>
<td>6</td>
<td>9</td>
<td>3300</td>
<td>3300</td>
</tr>
<tr>
<td>DFC</td>
<td>8</td>
<td>9</td>
<td>844</td>
<td>7</td>
</tr>
<tr>
<td>E2</td>
<td>12</td>
<td>10</td>
<td>342*</td>
<td>342*</td>
</tr>
<tr>
<td>Frog</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>HPC</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Loka97</td>
<td>16</td>
<td>&gt;36</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Magenta</td>
<td>6</td>
<td>&gt;10</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Mars</td>
<td>32</td>
<td>20</td>
<td>200*</td>
<td>344*</td>
</tr>
<tr>
<td>RC6</td>
<td>20</td>
<td>20</td>
<td>250</td>
<td>700*</td>
</tr>
<tr>
<td>Rijndael</td>
<td>10</td>
<td>8</td>
<td>233</td>
<td>256</td>
</tr>
<tr>
<td>SAFER+</td>
<td>8</td>
<td>7</td>
<td>700*</td>
<td>963*</td>
</tr>
<tr>
<td>Serpent</td>
<td>32</td>
<td>17</td>
<td>478*</td>
<td>584*</td>
</tr>
<tr>
<td>Twofish</td>
<td>16</td>
<td>12</td>
<td>194</td>
<td>218</td>
</tr>
</tbody>
</table>

Minimum secure round performance of AES candidates with 128-bit keys on Pentium-class CPUs.
Things to Note

- Twofish and Rijndael are the fastest.
- E2 and Mars are also fast.
- RC6 is fast on the Pentium Pro/II only.

64-Bit CPUs

- Again, algorithms that depend heavily on processor architecture are hurt on 64-bit CPUs.
- Our data is for the Dec Alpha.
- DFC is fastest, followed by Rijndael, Twofish, and HPC.
- We have some performance comparison's on the PA-RISC and Merced architectures. These will be discussed during the rump session.
### DEC Alpha Comparison

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast-256</td>
<td>600</td>
</tr>
<tr>
<td>Crypton</td>
<td>408</td>
</tr>
<tr>
<td>DEAL</td>
<td>2528*</td>
</tr>
<tr>
<td>E2</td>
<td>471</td>
</tr>
<tr>
<td>Frog</td>
<td>7</td>
</tr>
<tr>
<td>HPC</td>
<td>376</td>
</tr>
<tr>
<td>LOK97</td>
<td>?</td>
</tr>
<tr>
<td>Magenta</td>
<td>?</td>
</tr>
<tr>
<td>Mars</td>
<td>478</td>
</tr>
<tr>
<td>RC6</td>
<td>467*</td>
</tr>
<tr>
<td>Rijndael</td>
<td>340*</td>
</tr>
<tr>
<td>SAFER+</td>
<td>656</td>
</tr>
<tr>
<td>Serpent</td>
<td>915</td>
</tr>
<tr>
<td>Twofish</td>
<td>360*</td>
</tr>
</tbody>
</table>

AES candidate performance on the DEC Alpha

### Smart Cards

- Relative performance on 32-bit smart cards is approximately the same as on the Pentium.
- We concentrated on 8-bit smart cards.
- Numbers in the various papers are not good comparisons, because the assumptions vary greatly.
- Someone needs to code the leading candidates on several standard smart-card chips.
Smart Cards (cont.)

- Memory requirements are essential.
  - Most smart cards sold have 128 to 265 bytes of RAM.
  - All of this RAM is not available to the encryption engine.
- This is not a temporary problem; requirements to fit in a very small software footprint will always be there.
- High end smart cards will get better, but the low end will just get cheaper.

Smart Card RAM Requirements

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Smart Card RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast-256</td>
<td>60*</td>
</tr>
<tr>
<td>Crypton</td>
<td>52*</td>
</tr>
<tr>
<td>DEAL</td>
<td>50*</td>
</tr>
<tr>
<td>DRL</td>
<td>200</td>
</tr>
<tr>
<td>E2</td>
<td>300</td>
</tr>
<tr>
<td>Frog</td>
<td>2300+</td>
</tr>
<tr>
<td>HPC</td>
<td>?</td>
</tr>
<tr>
<td>Loko97</td>
<td>?</td>
</tr>
<tr>
<td>Magenta</td>
<td>?</td>
</tr>
<tr>
<td>Mars</td>
<td>195*</td>
</tr>
<tr>
<td>RCA</td>
<td>210*</td>
</tr>
<tr>
<td>Rijndael</td>
<td>52</td>
</tr>
<tr>
<td>SAFER+</td>
<td>50*</td>
</tr>
<tr>
<td>Serpent</td>
<td>50*</td>
</tr>
<tr>
<td>Twofish</td>
<td>60</td>
</tr>
</tbody>
</table>

AES candidates' smart card RAM requirements
Things to Note

- Some AES submissions CANNOT fit on small smart cards: DFC, E2, Mars, RC6. Frog cannot fit on any smart cards.

Hardware Performance

- We did not try to count gates for the different submissions.
- We concentrated on switching speeds in hardware applications.
- An algorithm should encrypt two blocks with two keys in no more time than it takes to encrypt two blocks with the same key.
## Hardware Key-Context RAM Requirements

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Key Context RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast-256</td>
<td>0</td>
</tr>
<tr>
<td>Crypton</td>
<td>0</td>
</tr>
<tr>
<td>DEAL</td>
<td>0</td>
</tr>
<tr>
<td>EDFC</td>
<td>0</td>
</tr>
<tr>
<td>E2</td>
<td>256</td>
</tr>
<tr>
<td>Frog</td>
<td>2300+</td>
</tr>
<tr>
<td>HPC</td>
<td>?</td>
</tr>
<tr>
<td>Loka97</td>
<td>?</td>
</tr>
<tr>
<td>Magenta</td>
<td>?</td>
</tr>
<tr>
<td>Mars</td>
<td>160</td>
</tr>
<tr>
<td>RC6</td>
<td>176</td>
</tr>
<tr>
<td>Rijndael</td>
<td>0</td>
</tr>
<tr>
<td>SAFER+</td>
<td>0</td>
</tr>
<tr>
<td>Serpent</td>
<td>0</td>
</tr>
<tr>
<td>Twofish</td>
<td>0</td>
</tr>
</tbody>
</table>

Hardware key-context RAM requirements

## Algorithm-Specific Comments
**CAST-256**

- Fits in small smart cards; on-the-fly key schedule generation hurts performance.

**Crypton**

- 32bit: Uniform across CPUs
- Fits in small smart cards.
- Most hardware-friendly algorithm.
- Most hash-function friendly algorithm.
**DEAL**

- Performance of DES.
- Fits on small smart cards.

---

**DFC**

- 32 bit: Multiplication over $2^{64}+13$ slow; hurts performance. Performance strongly depends on CPU.
- Can fit on small smart cards with significant performance penalties.
- Fastest on 64-bit CPUs.
- Key schedule makes decryption slower.
**E 2**

- 32 bit: Uniform across CPUs.
- Expanded key cannot fit on small smart cards.

**Frog**

- VERY slow key schedule.
- Expanded key cannot fit on any smart card.
HPC

- Heavy use of 64-bit operations hurt performance on other CPUs.
- Expanded key cannot fit on small smart cards.

Loki97

- Use of bit-level permutations hurts performance on all CPUs.
- Large tables makes it hard to fit on smart cards; expanded key cannot fit on small smart cards.
**Magenta**

- Slowest of all the candidates.
- Fits on small smart cards.

---

**Mars**

- 32 bit: Use of data-dependent rotations and modular multiplications hurts performance on most CPUs.
- 64-bit: Again, the use of data-dependent rotations and modular multiplications hurts performance.
- Expanded key cannot fit on small smart cards.
### RC6

- **32 bit:** Use of data-dependent rotations and modular multiplications hurts performance on most CPUs.
- **64-bit:** Again, the use of data-dependent rotations and modular multiplications hurts performance. (A 600 MHz Alpha runs RC6 at a slower absolute speed than a 400 MHz Pentium II.)
- Expanded key cannot fit on small smart cards.

### Rijndael

- **32 bit:** Uniform across CPUs.
- Fits on small smart cards.
- Very fast on 64-bit CPUs.
- Efficient in hardware.
- Most efficient across all platforms.
SAFER+

  Uniform across CPUs.
- Fits on small smart cards.

Serpent

- C performance closest to ASM performance.
- Fits on small smart cards.
**Twofish**

- Fits on small smart cards; performance improvements on larger smart cards.
- Efficient in hardware.

---

**Conclusions**

- Draw your own.