Optimizing AES for Embedded Devices and Wireless Sensor Networks

Shammi Didla
Aaron Ault
Saurabh Bagchi

The Center for Wireless Systems and Applications at Purdue University

Goals

- Software-based AES-128 at Zigbee bitrate (250 kbps)
- Minimal code space (ROM) and memory usage (RAM)
- 16-bit, 8MHz microcontroller platform: MSP430
- Written in C for maximum portability
AES Review

- 10 rounds:
  - Substitute bytes
  - Shift rows
  - Mix Columns
  - Add round key
- Can pre-compute round keys or compute on-the-fly

AES Review - 2

- Encryption:
  - SubBytes: look-up table
  - ShiftRows: rotate bytes in each of 4 “rows”
  - MixColumns: matrix multiplication in GF(2^8)
  - AddRoundKey: XOR with round key
- Key Expansion:
  - SubWord: same as SubBytes, but operates on key
  - RotWord: cyclical byte-level shift
Brian Gladman’s Reference

- Included with initial AES proposal
- “Low-resource”
  - All math reduced to XOR and Look-up Tables
  - Combines MixColumns with SubBytes
  - Combines MixColumns with ShiftRows
- Tuning options:
  - HAVE_MEMCPY: uses built-in memcpy
  - HAVE_UINT32: allows 32-bit data types
  - VERSION_1: uses local buffers in functions instead of pointers

Optimization

- -O3 option to GCC
  - unpredictable effect on some manual optimizations
  - All compiler optimizations on
- Specialization (SPECIAL)
  - limit to only AES-128
- Varying Data Type Size (DATASZ)
  - Allows up to 64-bit variables for XOR operations
- Function Inlining (INLINE)
  - Reduce overhead and stack size
Optimization - 2

- Loop unrolling (UNROLL)
  - Common optimization to reduce loop index maintenance costs
- Reducing Memory Moves (REDMEM)

Optimization - 3

- Eliminate local buffers (LOCBUF)
  - Use pointers instead of copying
- Use of Global Key Schedule (GLOB)
  - Store round keys globally
- On-the-fly Key Generation (OTFK)
  - Recompute round keys as they are needed
- 16-bit Memory Writes in MixColumns
  - Change uint8_t buffers to uint16_t
Experiment

- Softbaugh DZ1611 Zigbee Demo Board
- ROM (Code) Size
  - msp430-objdump
- RAM (Memory) Usage
  - msp430-gdb printing stack pointer
- Execution Time
  - Set I/O line on start, clear on end
  - Measure on oscilloscope

Results - Without -O3

- Version 12 is best:
  - SPECIAL
  - DATASZ(64)
  - INLINE
  - REDMEM
  - LOCBUF
  - GLOB
  >> 200 Kbps
Results With -O3

- Version 12 is best:
  - SPECIAL
  - DATASZ(64)
  - INLINE
  - REDMEM
  - LOCBUF
  - GLOB

>> 286 Kbps!

Other Implementations

- Improved both speed and code size (RAM unknown)
- Note that our versions seem to have varied significantly from published numbers in some cases

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Reference paper</th>
<th>Measured ROM Usage</th>
<th>Published ROM Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[6]</td>
<td>5968 bytes</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>[12]</td>
<td>6780 bytes</td>
<td>12616 bytes</td>
</tr>
<tr>
<td>3</td>
<td>[14]</td>
<td>6848 bytes</td>
<td>3322 bytes</td>
</tr>
<tr>
<td>4</td>
<td>[16]</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>5</td>
<td>Our implementation</td>
<td>5160 bytes</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Hardware AES

- As expected, performs much faster (579 kbps for standalone encryption)
- Complex, poorly documented for CC2420
- Offers standalone encryption, but not decryption
- “Stuck” with this chip: any future changes require change in hardware
- Used oscilloscope to time various operations:

<table>
<thead>
<tr>
<th>Process</th>
<th>Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing to the CC2420 RAM</td>
<td>94.40</td>
</tr>
<tr>
<td>Issuing the encrypt/decrypt command to the CC2420</td>
<td>6.40</td>
</tr>
<tr>
<td>Wait for encryption module to complete processing by requesting status byte</td>
<td>18.40</td>
</tr>
<tr>
<td>Read from the CC2420 RAM</td>
<td>102.40</td>
</tr>
</tbody>
</table>

Conclusion

- Catalogued ideal optimizations
- Measurement metrics
- Improvement over previous implementations

>> Final code runs at 286 Kbps with 5190 bytes ROM and 260 bytes RAM
Future Work

- Develop a full security suite in C for MSP430/CC2420 with AES-128 as core
- Perform similar optimizations for AES-256
- Deployment in safety and security applications at Purdue’s Ross-Ade football stadium.