Parallel Synchronous Code Generation for Second Round Light Weight Candidates

Pantea Kiaei, Archanaa S. Krishnan, Patrick Schaumont

Worcester Polytechnic Institute
Virginia Tech
Outline

1. Background
2. Timing side-channel
3. Timing aspects of AEAD ciphers
4. Predictable-time solutions
5. Parallel synchronous programming
6. Results
Background

• AEAD algorithm
  • Confidentiality
  • Integrity
  • Authenticity

• Implementation
  • Side-channel analysis
  • Fault attacks

• Cryptanalysis on the algorithm
  • Timing analysis
  • Power analysis
  • Fault injection
  • …
Timing attacks

• Sources of non-constant run-time:
  • Algorithm
  • Data-dependent run-time
  • Memory hierarchy
    • Cache-hit vs. cache-miss
  • Shared resources

• Timing side-channel
  • Algorithm
    • Table lookups
  • Memory hierarchy
    • Last Level Cache (LLC): Flush+Reload [1], Prime+Probe [2]

AEAD ciphers

- Timing side-channel
  - Recognize different phases
    - Facilitate power SCA and fault injection
  - Gain information about the internal state
Predictable-time solutions

• Bitslicing
  • Every thing computed
    ➢ no table lookups
  • Data scattered over many locations
    ➢ no memory hierarchy timing leakage
  • Normal control logic

• Parallel Synchronous Programming [3]
  • Implements FSMD in software
  • Adds the control logic into the bitslicing
    ➢ no algorithm-related unpredictability

Parallel Synchronous Programming (PSP)

- PSP kernel
  - Generated by automated tool (PSPCG)
  - Equivalent to one clock cycle of FSMD

- PSP wrapper
  - Calls the PSP kernel

```c
void kernel(..., MDTYPE* done) {
    ...
    NOT1(rst, n01_);
    XOR2(fsmd_reg[0], CNT[0], n02_);
    XOR2(CNT[1], fsmd_reg[1], n03_);
    OR2(n02_, n03_, n04_);
    ...
    DFF(clk, n00_[0], fsmd_reg[0]);
    DFF(clk, n00_[1], fsmd_reg[1]);
    DFF(clk, n00_[2], fsmd_reg[2]);
    DFF(clk, n00_[3], fsmd_reg[3]);
}
```

```c
int main() {
    // prepare inputs in bitsliced format
    ...
    // reset:
    ...
    // keep calling the kernel until all calculations complete:
    while (done != 0xffffffff) {
        kernel(..., &done);
    }
    return 0;
}
```
PSP implementation of LWC candidates

- Predictable run-time
  - The longest calculation among the parallel runs
- Indistinguishable AEAD phases

<table>
<thead>
<tr>
<th>Slices</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice 1</td>
<td>$P_{1,1}$</td>
<td>$P_{1,2}$</td>
<td>$P_{1,3}$</td>
<td>$P_{1,4}$</td>
<td>$P_{1,5}$</td>
<td>$P_{1,6}$</td>
<td>$P_{1,7}$</td>
<td>$P_{1,8}$</td>
</tr>
<tr>
<td>slice 2</td>
<td>$P_{2,1}$</td>
<td>$P_{2,2}$</td>
<td>$P_{2,3}$</td>
<td>$P_{2,4}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slice 3</td>
<td>$P_{3,1}$</td>
<td>$P_{3,2}$</td>
<td>$P_{3,3}$</td>
<td>$P_{3,4}$</td>
<td>$P_{3,5}$</td>
<td>$P_{3,6}$</td>
<td>$P_{3,7}$</td>
<td></td>
</tr>
<tr>
<td>slice 4</td>
<td>$P_{4,1}$</td>
<td>$P_{4,2}$</td>
<td>$P_{4,3}$</td>
<td>$P_{4,4}$</td>
<td>$P_{4,5}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time Steps (psp function calls)

- 'done'
  - 0x0
  - 0x0
  - 0x0
  - 0x2
  - 0xA
  - 0xA
  - 0xE
  - 0xF
Implementations

• Implemented PSP version of a few second round LWC candidates
  • Not a comparison among the candidates

• Compared the PSP version and normal implementation provided by submissions

• Compared the PSP version and bitsliced implementation generated by the Usuba compiler
Results: PSP vs. reference implementation

<table>
<thead>
<tr>
<th>AEAD</th>
<th>Implementation</th>
<th>Code size(B)</th>
<th>Cycles/byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascon-128</td>
<td>Reference</td>
<td>15,709</td>
<td>2,751</td>
</tr>
<tr>
<td></td>
<td>PSP</td>
<td>210k</td>
<td>6,855</td>
</tr>
<tr>
<td>WAGE-AE-128</td>
<td>Reference</td>
<td>24,439</td>
<td>9,305</td>
</tr>
<tr>
<td></td>
<td>PSP</td>
<td>47,304</td>
<td>21,032</td>
</tr>
</tbody>
</table>

Table 2: Overhead of complete PSP implementations of Ascon-128 and WAGE-AE-128 when compared with reference implementation. The measurements were computed for processing 8B of plaintext and 8B of associated data.
Results: PSP vs. bitsliced – code size

<table>
<thead>
<tr>
<th>Cipher/permutation Implementation</th>
<th>Code size</th>
<th>Cycle count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(B)</td>
<td>(cycles)</td>
</tr>
<tr>
<td><strong>ACE permutation</strong></td>
<td>Reference</td>
<td>14,937</td>
</tr>
<tr>
<td></td>
<td>Usuba</td>
<td>115k</td>
</tr>
<tr>
<td><strong>ACE-Æ-128</strong></td>
<td>PSP</td>
<td>84,280</td>
</tr>
</tbody>
</table>

Table 3: Code size and cycle count comparison of ACE-Æ-128 PSP core implementation with ACE permutation reference implementation and its Usuba bitsliced implementation.
Results: PSP vs. bitsliced – logic

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Code Generator</th>
<th>AND</th>
<th>ORR</th>
<th>EOR</th>
<th>MVN</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascon-$p^{12}$</td>
<td>Usubac</td>
<td>3840</td>
<td>0</td>
<td>16128</td>
<td>4657</td>
<td>24625</td>
</tr>
<tr>
<td></td>
<td>PSPCG</td>
<td>6381</td>
<td>6034</td>
<td>1669</td>
<td>1324</td>
<td>15408</td>
</tr>
<tr>
<td>GIFT-128</td>
<td>Usubac</td>
<td>3840</td>
<td>1280</td>
<td>14080</td>
<td>1360</td>
<td>20560</td>
</tr>
<tr>
<td></td>
<td>PSPCG</td>
<td>756</td>
<td>418</td>
<td>262</td>
<td>137</td>
<td>1573</td>
</tr>
</tbody>
</table>

Table 4: Comparison of number of instructions resulting from Usubac and PSPCG
Results: PSP vs. bitsliced – memory spill

Table 5: Register spill of the bitsliced code generated by Usubac

<table>
<thead>
<tr>
<th>Cipher</th>
<th>AND</th>
<th>ORR</th>
<th>EOR</th>
<th>MVN</th>
<th>MOV</th>
<th>LDR</th>
<th>STR</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE permutation</td>
<td>64</td>
<td>0</td>
<td>1168</td>
<td>832</td>
<td>11229</td>
<td>1469</td>
<td>25</td>
<td>86.04%</td>
</tr>
<tr>
<td>GIFT-128</td>
<td>192</td>
<td>64</td>
<td>704</td>
<td>66</td>
<td>6725</td>
<td>2041</td>
<td>18</td>
<td>89.54%</td>
</tr>
</tbody>
</table>

Table 6: Register spill of the PSP code generated by PSPCG

<table>
<thead>
<tr>
<th>Cipher</th>
<th>AND</th>
<th>ORR</th>
<th>BIC</th>
<th>EOR</th>
<th>ORN</th>
<th>MVN</th>
<th>MOV</th>
<th>LDR</th>
<th>STR</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascon-$p^{12}$</td>
<td>1732</td>
<td>1296</td>
<td>281</td>
<td>808</td>
<td>1265</td>
<td>123</td>
<td>4904</td>
<td>10277</td>
<td>3862</td>
<td>77.57%</td>
</tr>
<tr>
<td>GIFT-128</td>
<td>530</td>
<td>57</td>
<td>30</td>
<td>58</td>
<td>54</td>
<td>2</td>
<td>120</td>
<td>1415</td>
<td>971</td>
<td>77.42%</td>
</tr>
</tbody>
</table>
Conclusion

• Naïve implementations of AEAD ciphers can result in exploitable timing side-channel
• Parallel synchronous programming can prevent timing side-channel of AEAD ciphers
• Parallel synchronous programs in contrast to bitslicing include the control logic therefore are more secure while also being more compact
Thank you