CEG5010: A Massively Parallel RC4 Encryption Engine

Overview

- Introduction
- Algorithm
- Architecture
- Implementation
- Results
- Conclusion
FPGAs as cryptographic accelerators

- To date application to cryptanalysis limited: high cost and low density
- Recent improvements make FPGAs increasingly suitable for cryptanalysis
- High density (58k~4M system gates)
- High on-chip memory bandwidth
- Algorithm hardwired (more parallelism)
- Reconfigurable for new algorithms

RC4 Algorithm – Overview

- A stream encryption algorithm
- Single key for both encryption and decryption
- A PRNG uses the key as seed
- Key size ranges from 40 to 256 bits
- S[] state memory, K[] is the key
RC4 Algorithm – Key Scheduling

```c
key_scheduling() {
    /* initialisation */
    for i = 0 to 255
        s[i] = i;

    /* scrambling */
    j = 0;
    for i = 0 to 255
    {  
        j = j + K[i % k_size] + S[i];
        swap S[i] and S[j];
    }
}
```

RC4 Algorithm – PRNG

```c
PRNG() {
    i = 0;
    j = 0;
    while not end of stream
    {  
        i = (i + 1) % 256;
        j = (j + S[i]) % 256;
        swap S[i] and S[j];
        t = S[i] + S[j];
        ct[i] = pt[i] xor S[t];
    }
}
```
Known Plain Text Attack

- Both plain text and cipher text are known
  - e.g. email has headers from which plain text can be determined
- Could be adapted for a cipher text only attack
  - e.g. look for 7-bit ascii characters in decrypted text
Key Search

Actually used key = “abcde”
Starting base key = 00000

key = 00000 00001 00004 00005
RC4_0 RC4_1 RC4_4 RC4_5
found = false false false false
base key = base key + N

key = abcda abcdb abcdab abcd
RC4_0 RC4_1 RC4_4 RC4_5
found = false false true false

return key = abcda + 4 = abcde

Key Search

keysearch()
    k = 0; cxp = pt xor ct;
    forever {
        for i = 0 to N-1 (in parallel){
            found = rc4(cxp, k + i)
            if (found(i))
                return k + i;
        }
        k = k + N;
    }
RC4 Core Structure

- Key array, $K[i]$, indexed by counter $i$
- S-block, $S[i]$, dual port Block RAM
- 3 x 8-bit adders, for
  - $j = j + S[i] + K[i]$
  - $t = S[i] + S[j]$
- 2 x 8-bit registers, for $j$ and $t$
- 3 clock cycles for each iteration in both phases
- S-block is re-initialized for next key in key scheduling

RC4 Core Structure – Key Scheduling

\[ j = pre_j + S[i] + K[i \% k\_size] \]
\[ swap S[j] S[i] \]
RC4 Core Structure – PRNG

$$ j = \text{pre}_j + S[i] $$

swap $S[j]$ $S[i]$  

$$ t = S[j] + S[i] $$

RC4 key re-initialization

- Re-initialize one half while scrambling key in another
- BlockRAM has 4096 bits (dual port)
- Only 2048 bits are need for S-block
- Split BlockRAM into 2 halves
- Saves 256 clock cycles
Parallel Key Search Datapath

- 96 RC4 cores on a single chip
- Single control unit
- Global $i$-counter, local $j$-counter
- Each core has an offset $\{0 \ldots 95\}$ and a copy of local key
  (local key = global key + offset)
- When core finds the key, core’s offset and global key are sent to host
Parallel Key Search Control

- Dispatch the global key to cores
- Start cores to decrypt the message
- All cores work synchronously
- If no valid key is found, update global key by adding 96 and loop
- If a valid key is found, stop all cores and report result to host

Host Interface

- Host sends expected results
- Host sends starting global key
- FSM starts, host waits
- Valid key found, FSM stops
- Host reads back the global key and core’s offset
- Can start from any location in the key space
Implementation – Core Design

- Each core associated with a Block RAM
- A single core can fit into a 6x4 CLB array
- All core signals are locally routed
- For local key, 40-bit adder with carry chain use a column of 20 slices

Implementation – Core Design

- Logic consumed by key byte selection
  - K[i % k_size]
- TBUFs are used instead
- Wired OR all outputs
- Eliminated 8 5-to-1 MUXs
  - Saves 8 slices for each core
  - Saves 768 slices in design
Implementation – Parallel Design

- Structural design with reusable core module
- Uses RLOC attribute for placement of cores
- The control unit resides in the center area
- Buffers are added for large fanout signals
- Floor planning tools used to optimize the design
Implementation – VHDL

- After stating constraints in VHDL, most parts of the design are fixed
- Floorplanner used to adjust the I/O and control units
- Hard macro generated from FPGA editor (dramatically reduces place and route time)

Results – Platform

- Chip: Xilinx VirtexE (XCV1000E-6)
- Board: Pilchard (memory slot based card)
- Tools: Xilinx ISE4.1i
- Host: PIII 800MHz
- OS: Linux
Results – Area Utilization

<table>
<thead>
<tr>
<th>Resources</th>
<th>Use Count</th>
<th>Utilization Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLLs</td>
<td>1 out of 4</td>
<td>25%</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>96 out of 96</td>
<td>100%</td>
</tr>
<tr>
<td>SLICEs</td>
<td>5178 out of 12288</td>
<td>42%</td>
</tr>
<tr>
<td>TBUFs</td>
<td>4608 out of 12544</td>
<td>36%</td>
</tr>
</tbody>
</table>

Results – Timing Spec.

- Individual core frequency: ~100MHz
- Max. design speed: ~80MHz
- Reported by Xilinx TRACE tools
- Key search engine frequency: 50MHz
- Interface frequency (system memory bus): 100MHz
- Critical path is: from key through two adders to Block RAM inputs
Results – Performance

• 256 iterations for key scheduling
• 8 iterations for PRNG
• 3 clock cycles for each iteration
• 20ns for each clock cycle
⇒ 15us for a single key (single core)
⇒ 6,000,000 keys/s (single FPGA)
⇒ 50hr for 40-bit key search (single FPGA)

Results – Performance

• Time to complete an encryption

<table>
<thead>
<tr>
<th>Platform</th>
<th>Frequency (MHz)</th>
<th>Time (us)</th>
<th>Normalized time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Ultra IlI</td>
<td>400</td>
<td>49.4</td>
<td>299</td>
</tr>
<tr>
<td>SGI R12000A</td>
<td>400</td>
<td>11.3</td>
<td>68.6</td>
</tr>
<tr>
<td>Intel P4</td>
<td>1500</td>
<td>9.62</td>
<td>58.3</td>
</tr>
<tr>
<td>This work</td>
<td>50</td>
<td>0.165</td>
<td>1</td>
</tr>
</tbody>
</table>
Results – Performance

Summary of approach

- Parallelism in RC4 core allows several operations to be completed in a single cycle
- On-chip resources were used to achieve a very low latency, high bandwidth memory interface
- Dual-ported memory for higher memory transfer efficiency
- Floorplanning was used to minimize interconnect delays
- A large number of the decryption cores were used in parallel
Further Speedups

- Slightly faster part would allow us to operate at 100MHz (2x improvement)
- Parts with more Block RAMs would enable us to increase number of cores
  - e.g. virtex EM XCV812E has 280 Block RAMs so we could fit 3x more cores (3x improvement)
  \[ \Rightarrow \text{Together would make a 6x improvement} \]
- It may also be possible to utilize the unused slices to add more cores (currently only 42% of slices are used)

Conclusion

- Described a high performance RC4 core which can perform an encryption in 15us
- Integrated 96 cores on a single FPGA
- 58x faster than 1.5GHz Pentium 4
- With improving density and on-chip memory, FPGAs are becoming increasingly suitable for cryptanalysis applications