Hardware Accelerator for Stream Cipher Spritz

by
Debjyoti Bhattacharjee and Anupam Chattopadhyay

School of Computer Science and Engineering (SCSE)
26-July-2016
OUTLINE

• Overview and Introduction to Spritz
• Design decisions for Spritz component functions
• Hardware implementation
• Performance Evaluation and Benchmarking
• Conclusion
Overview

• RC4 - The dominant stream cipher
  • e-commerce
  • communication protocols such as, WEP, TLS

• Spritz - A new stream cipher.
  • Proposed by the author of RC4 [Ron Rivest and Jacob Schuldt]
  • Design is based on Cryptographic Sponge construction
  • Permits use in different modes

Series of vulnerabilities that have been pointed out in recent past !!!

Makes it an attractive design choice for security protocols.
Introduction to Spritz

All value in Spritz are modulo-$N$

Spritz State consists of -

• six one byte registers $i$, $j$, $k$, $w$, $z$ and $a$
• array $S$ of length $N$ which stores a permutation of $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$

The cryptographic key $K$ is a byte-array of length $L$. 
Spritz top-level functions

INITIALIZESTATE
initializes the state of Spritz to a standard state.

ABSORB
takes a variable length input $I$ and updates the state of Spritz based on the input.

For every $\text{floor}(N/2)$ nibbles absorbed,
SHUFFLE is invoked which whips, crushes, whips, crushes and finally whips again.

ABSORBSTOP absorbs a special stop symbol.
It is used to separate various inputs being absorbed.

SQUEEZE- the main output function of Spritz,
Produces $r$-output bytes, where $r$ is an input to the function.
Using top-level functions, InitializeState, Absorb, AbsorbStop and Squeeze, Spritz operates in various modes.
OUTLINE

• Overview and Introduction to Spritz
• Design decisions for Spritz component functions
• Hardware implementation
• Performance Evaluation and Benchmarking
• Conclusion
**ABSORB**Byte**(b)**

- **ABSORB**Nibble**(LOW(b))** and **ABSORB**Nibble**(HIGH(b))** performed in a single cycle.
- The target address for the swaps in the corresponding **ABSORB**Nibble**(x)** is computed along with incrementing the value of state register **a**.
- Double swaps performed - Read-After-Write**(RAW)** dependencies considered.
**AbsorbByte(b)**

- Higher number of AbsorbNibble (b) **not** performed simultaneously
- Negative impact on critical path due to additional circuitry to resolve dependencies
- Cost of additional memory ports to allow concurrent access.
UPDATE() 

• Three stage pipeline
• Stage 1 has a data dependency with the Stage 3 of previous iteration
  - one cycle stall required.
**DRIP()**

- First three stages identical to that of UPDATE
- Can share hardware with UPDATE
- Stage 4-6 used to update register $z$
- **Two cycle stall** required due to read dependency of Stage 4 on Stage 6 of previous iteration.
**WHIP(x)**

- $x$ iterations operation UPDATE are performed.
- In the subsequent cycle, state register $w$ is incremented by 2, - since $N$ is a power of 2
CRUSH()

• The data at location $S[v]$ is swapped with data at location $S[N-1-v]$ if the condition $S[v] > S[N-1-v]$ is satisfied.

• Well defined memory access patterns
CRUSH() 

- There are no data dependencies between the iterations of the loop.
- Theoretically possible to unroll the loop up to floor(N/2) to complete in a single clock cycle.
- For practical hardware implementation, assuming four read and four write ports,
  - Two iterations can be performed simultaneously.
SHUFFLE()

1. Invoke UPDATE 2N times, followed by incrementing \( w \) by 2.
2. Invoke CRUSH N/4 times, with parameter \( v \) ranging from 0 to N/2, incremented by two in each iteration.
3. Same as Step 1
4. Same as Step 2
5. Same as Step 1
6. Set state register \( a \) to 0 in the cycle immediately after the cycle in which \( w \) has been incremented by 2 in Step 4.
**InitializeState()**

- Assuming 4 write ports, the state array \( S \) can be initialized in \( N/4 \) cycles.
- Simultaneously, the state registers
  - \( i, j, k, z, a \) can be initialized to zero
  - \( w \) is initialized to one.
OUTLINE

• Overview and Introduction to Spritz
• Design decisions for Spritz component functions
• Hardware implementation
• Performance Evaluation and Benchmarking
• Conclusion
Hardware Implementation

- Verilog description was used for implementation.
- The array $S$, of size 256 bytes, has been implemented as an array of master-slave flip-flops.
- $func\_sel$ chooses the operation to be performed
- $data\_in$ is used to supply required data to the accelerator.
- $shuffle\_on$ is set to high when SHUFFLE has be invoked in the next cycle.
- $resume$ is used to flag that operation ABSORBBYTE can resume after SHUFFLE.
OUTLINE

• Overview and Introduction to Spritz
• Design decisions for Spritz component functions
• Hardware implementation
• Performance Evaluation and Benchmarking
• Conclusion
Performance evaluation

Synthesis tool for FPGA-platform : Xilinx ISE 14.7
Target device : Virtex-7 (device: xc7vx330t-3ffg1157)
Timing analysis tool : Xilinx static timing analysis tool.

Throughput computation parameters: A 16-byte key is set up followed by encryption of a 10 kilobyte message to compute the throughput.

<table>
<thead>
<tr>
<th></th>
<th>ASIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock frequency(MHz)</td>
<td>1150</td>
<td>Frequency(MHz)</td>
</tr>
<tr>
<td>Throughput(Mbps)</td>
<td>2780.14</td>
<td>Throughput(Mbps)</td>
</tr>
<tr>
<td>Combinatorial Area(KGE)</td>
<td>119.52</td>
<td>#slice registers</td>
</tr>
<tr>
<td>Non-combinatorial Area(KGE)</td>
<td>11.19</td>
<td>#slice LUTS</td>
</tr>
<tr>
<td>Total Area (KGE)</td>
<td>130.71</td>
<td>#occupied slices</td>
</tr>
<tr>
<td>Area Efficiency(Mbps/KGE)</td>
<td>21.27</td>
<td></td>
</tr>
<tr>
<td>Energy Efficiency(pJ/bits)</td>
<td>5.834</td>
<td></td>
</tr>
</tbody>
</table>
Comparison against software implementation

<table>
<thead>
<tr>
<th></th>
<th>Software</th>
<th>Hardware [ASIC]</th>
<th>Hardware [FPGA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUEEZE</td>
<td>94.69 Mbps</td>
<td>3066.67 Mbps</td>
<td>172.6 Mbps</td>
</tr>
<tr>
<td>ABSORB</td>
<td>5.62 Mbps</td>
<td>360.12 Mbps</td>
<td>20.24 Mbps</td>
</tr>
</tbody>
</table>

32.38X speed-up for the SQUEEZE.
64.07X speed-up for the ABSORB function.
Benchmarking against other stream ciphers and hash-function accelerators

- Spritz fares rather poorly in comparison to
  - lightweight stream ciphers
  - high-speed stream cipher candidates (e.g., Sosemanuk).

- Large area overhead due to the state array S
- Nested memory accesses required for computation of UPDATE and OUTPUT

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Technology Node (nm)</th>
<th>Area (KGE)</th>
<th>Throughput (Gbps)</th>
<th>TpA (Gbps/KGE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sosemanuk [12]</td>
<td>130</td>
<td>95.74</td>
<td>66.56</td>
<td>0.695</td>
</tr>
<tr>
<td>RC4 [2]</td>
<td>130</td>
<td>59.93</td>
<td>10.0</td>
<td>1.667</td>
</tr>
<tr>
<td>Grain128 [19]</td>
<td>130</td>
<td>3.2</td>
<td>14.48</td>
<td>4.525</td>
</tr>
<tr>
<td>MICKEY [20]</td>
<td>130</td>
<td>5.0</td>
<td>0.41</td>
<td>0.082</td>
</tr>
<tr>
<td>Trivium [21]</td>
<td>130</td>
<td>4.9</td>
<td>22.3</td>
<td>4.551</td>
</tr>
<tr>
<td>Keccak256 [22]</td>
<td>130</td>
<td>50.0</td>
<td>43.01</td>
<td>0.860</td>
</tr>
</tbody>
</table>
Throughput Comparison with Existing designs for Large Messages

• *cpb* of ABSORB reaches 25.5 for long *absorb text*
  • 2.3x slower compared to the SHA-3 standard hash function Keccak
  • 16x compared to the software implementation of Spritz.

• *cpb* of SQUEEZE function reaches a value of 3 for arbitrarily long messages
  • 6x slower than the best reported implementation of RC4

*This slowdown is directly caused by the nested calls to the storage for the output function of Spritz.*
OUTLINE

• Overview and Introduction to Spritz
• Design decisions for Spritz component functions
• Hardware implementation
• Performance Evaluation and Benchmarking
• Conclusion
Conclusion

• Spritz is a new stream cipher proposed as a replacement for RC4.

• We explored the design points of Spritz considering
  - a high performance custom hardware architecture
  - minimize its cycles per byte.

• significant speed-up compared to the basic, un-optimized software implementation.
  – For both ASIC and FPGA implementations.

• In terms of area-efficiency, Spritz fares worse compared to the prominent stream ciphers and hash functions.
References

Thank you