Lightweight version of π-Cipher

Hristina Mihajloska, Mohamed El Hadedy, Danilo Gligoroski and Kevin Skadron
Department of Computer Science
University of Virginia
July 20, 2015
Goal: Efficient and flexible crypto processing

- ARX – simple operations and fast primitives
- Crypto operations increasingly common
  - Power, performance benefits from specialization

Outline

- ARX Programmable Processing Element
- Custom ARX for Pi-Cipher
- Comparison
- Conclusion
CAESAR competition

- Second round 29 candidates
  - mostly AES based
  - other sponge/permutation based
  - just three of them are ARX based

- π-Cipher is one of the ARX based candidates
π-Cipher

- An authenticated encryption cipher with associated data

- Designers:
  - Danilo Gligoroski, NTNU
  - Hristina Mihajloska, FINKI
  - Simona Samardziska, FINKI
  - Haakon Jacobsen, NTNU
  - Mohamed El-Hadeedy, VU
  - Rune Erlend Jensen, NTNU
π-Cipher

Main design goals:

- High security
- Simplicity (ARX design)
- Suitable for HW and SW
- Parallel
- Incremental
- Tag second-preimage resistant
π-Cipher

- The main role in the security and design has permutation function (π-function)
- Only uses Add, Rotate and XOR operations
- 4 rounds
  - each round has 8 ARX operations blocks (* op.)
  - every ARX block consists of 52 ARX operations
- Different word sizes
  - π16-Cipher – lightweight version
π16-Cipher

- In total:
  - 1664 ARX operations (for one π-func)
  - 128 bits of memory for * operation
  - 512 bits of memory for round constants
- Additional:
  - 64 bits for the counter
  - 256 bits for the state
ARX Programmable Processing Element

- Flexible & Simple

- ALU – supports different widths
  - Four 16-bit ALU
  - Two 32-bit ALU
  - One 64-bit ALU

![Diagram of ARX Programmable Processing Element](image-url)
## Arithmetic and Logic Unit

### Arithmetic and Logic Unit Truth Table

<table>
<thead>
<tr>
<th>Acc</th>
<th>Modes</th>
<th>Operations</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst (6)</td>
<td>Inst (5)</td>
<td>Inst (4)</td>
<td>Inst (3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Diagram of the Arithmetic and Logic Unit (ALU)
ARX Programmable Processing Element

- Flexible & Simple

- Rotator
  - Four 16-bit ROL
  - Two 32-bit ROL
  - One 64-bit ROL
## Rotator

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mode</th>
<th>RC3</th>
<th>RC2</th>
<th>RC1</th>
<th>RC0</th>
<th>Data width</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROR (1 → 15)</td>
<td>00001010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>64 Bits Mode</td>
</tr>
<tr>
<td>ROR (16 → 31)</td>
<td>01101010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (16 → 31)</td>
<td>01101010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (32 → 47)</td>
<td>10001010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (48 → 63)</td>
<td>11101010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (1 → 15)</td>
<td>00011010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>32 Bits Mode</td>
</tr>
<tr>
<td>ROR (1 → 15)</td>
<td>00011010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (16 → 31)</td>
<td>01111010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (16 → 31)</td>
<td>01111010101</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
<tr>
<td>ROR (1 → 15)</td>
<td>00000000000</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>16 Bits Mode</td>
</tr>
<tr>
<td>ROR (1 → 15)</td>
<td>00000000000</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td>1 → F</td>
<td></td>
</tr>
</tbody>
</table>

![Rotator Diagram](image)

**Modes** | **RC3** | **RC2** | **RC1** | **RC0** | **ALU** | **Coefficient Memory**

- Modes: 2 bit
- RC3, RC2, RC1, RC0: 4 bits each
- ALU: Mode (3 bits), OP (2 bits), R/W (2 bits), ADDRW (6 bits), ADDRB (6 bits), ADDRA (6 bits)
ARX Programmable Processing Element

512 byte coeff memory

VLIW

Processing Element

- Modes
- Rotator
- RC3: 11 bits
- RC2: 4 bits
- RC1: 4 bits
- RC0: 4 bits
- ALU
- ACC: 2 bits
- Mode: 3 bits
- OP: 2 bits
- RW: 6 bits
- ADDRW: 6 bits
- ADDRB: 6 bits
- ADDRA: 6 bits
**ARX Programmable Processing Element**

- Dual-ported instruction memory allows:
  - A new program to be loaded while the current program progresses
  - Or an early start on a new program while the rest of the program is still loading
  - Or seamless processing of a program that cannot entirely fit into the on-chip memory.
ARX Programmable Processing Element

Processing Element

Instruction RAM
Custom ARX Implementation of Pi-Cipher

- Single Width ARX
- Double-Width ARX
- Quad-width ARX

<table>
<thead>
<tr>
<th>μ-operation for 64-bit words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: ( X = (X_0, X_1, X_2, X_3) ) and ( Y = (Y_0, Y_1, Y_2, Y_3) ) where ( X_i ) and ( Y_i ) are 64-bit variables.</td>
</tr>
<tr>
<td>Output: ( Z = (Z_0, Z_1, Z_2, Z_3) ) where ( Z_i ) are 64-bit variables.</td>
</tr>
<tr>
<td>Temporary 64-bit variables: ( T_0, \ldots, T_{11} ).</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\mu - transformation \ for \ X: \\
T_0 & \leftarrow ROL{T}^3(0x0F0E8E4E2E1D8D4D2 + X_0 + X_1 + X_2); \\
1. & \quad T_1 \leftarrow ROL{T}^{19}(0x8D1CCACAC66C5C3B8 + X_0 + X_1 + X_3); \\
2. & \quad T_2 \leftarrow ROL{T}^{13}(0x2B4B2B1ACAAA9A6A5 + X_0 + X_2 + X_3); \\
3. & \quad T_3 \leftarrow ROL{T}^{27}(0xA39C9A99695938E + X_1 + X_2 + X_3); \\
\end{align*} \]

\[ \begin{align*}
\nu - transformation \ for \ Y: \\
T_0 & \leftarrow ROL{T}^{11}(0x0D8B87787472716C + Y_0 + Y_1 + Y_2); \\
1. & \quad T_1 \leftarrow ROL{T}^{19}(0x6A563665976C35CA59 + Y_0 + Y_2 + Y_3); \\
2. & \quad T_2 \leftarrow ROL{T}^{27}(0x5655534ED4B473C + Y_0 + Y_1 + Y_2); \\
3. & \quad T_3 \leftarrow ROL{T}^{27}(0xA39C9A99695938E + Y_1 + Y_2 + Y_3); \\
\end{align*} \]

\[ \begin{align*}
\sigma - transformation \ for \ both \ \mu(X) \ and \ \nu(Y): \\
Z_0 & \leftarrow T_3 + T_8; \\
Z_1 & \leftarrow T_5 + T_9; \\
Z_2 & \leftarrow T_7 + T_{10}; \\
Z_3 & \leftarrow T_8; \\
\end{align*} \]
Custom, Quad ARX Implementation
## Implementations & Comparison

<table>
<thead>
<tr>
<th></th>
<th>PPE</th>
<th>Single Width</th>
<th>Double Width</th>
<th>Quad Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>1.20 Gbps</td>
<td>3.57 Gbps</td>
<td>3.68 Gbps</td>
<td>4.34 Gbps</td>
</tr>
<tr>
<td>Area (Slices)</td>
<td>227</td>
<td>132</td>
<td>154</td>
<td>266</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>250</td>
<td>371</td>
<td>324</td>
<td>347</td>
</tr>
<tr>
<td>Throughput/Area</td>
<td>5.4</td>
<td>27.7</td>
<td>24.5</td>
<td>16.7</td>
</tr>
</tbody>
</table>

**ARX Performance (16-Wide Pi-Cipher)**

Xilinx Virtex 7 FPGA
## Implementations & Comparison

<table>
<thead>
<tr>
<th></th>
<th>PPE</th>
<th>Single Width</th>
<th>Double Width</th>
<th>Quad Width</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>1.17 Gbps</td>
<td>3.09 Gbps</td>
<td>3.68 Gbps</td>
<td>4.22 Gbps</td>
</tr>
<tr>
<td><strong>Area (Slices)</strong></td>
<td>227</td>
<td>445</td>
<td>447</td>
<td>634</td>
</tr>
<tr>
<td><strong>Frequency (MHz)</strong></td>
<td>250</td>
<td>254</td>
<td>243</td>
<td>245</td>
</tr>
<tr>
<td><strong>Throughput/Area</strong></td>
<td>5.3</td>
<td>7.1</td>
<td>8.4</td>
<td>6.8</td>
</tr>
</tbody>
</table>

**ARX Performance (64-Wide Pi-Cipher)**

Xilinx Virtex 7 FPGA
Performance

PPE

- **Pros**: It is a *programmable* element which can be used to implement different algorithms based on the ARX paradigm
- **Pros**: It can support different word sizes
- **Pros**: Duplicating the PPE can achieve 75% of the 64-bit custom core’s throughput, but with greater flexibility
Performance

PPE

- **Cons**: The custom hardware implementations are much more efficient than the PPE (Area & Performance)

- **Cons**: The PPE design we have so far is handicapped because it uses a native 64-bit ALU, which we suspect lowers the achievable frequency by increasing the critical path
Future Work

- Further improve efficiency of the PPE
- Implement more algorithms with the PPE & benchmark against custom hardware
Thank you

Q&A