Enhancing CAESAR Hardware API Support for Lightweight Architectures

Panasayya Yalla\textsuperscript{1}  \textbf{Jens-Peter Kaps}\textsuperscript{1}  Fabrizio De Santis\textsuperscript{2}  Michael Tempelmeier\textsuperscript{2}

\textsuperscript{1}Cryptographic Engineering Research Group (CERG)  
http://cryptography.gmu.edu  
Volgenau School of Engineering, GMU, USA

\textsuperscript{2}Technische Universität München, Munich, Germany

DIAC 2016: Directions in Authenticated Ciphers 2016
Support for CAESAR Hardware API

- Specification of standard hardware Application Programming Interface (API)
- Implementer’s Guide
- Development Package including VHDL code for high-speed implementations
  - Pre- and PostProcessors to handle the protocol
  - Universal Padding Unit

No VHDL Code for Lightweight Implementations
- Protocol handling is an additional burden
- Makes debugging more difficult
Goals for Support of Lightweight Implementations

- Reduce the burden on the designer
- Fully compliant with the CAESAR API
- Support CAESAR API’s bus widths for lightweight implementations of 8, 16, and 32 bits
- Clear separation of Communication Protocol and Algorithm
- Low area footprint:
  - Avoid duplication of elements between protocol handling and algorithm
- Minimize overhead:
  - Not a ‘one-size-fits-all’ solution ⇒ needs tweaking
  - No universal padding unit ⇒ designer can choose most efficient way to implement
Main Components

- Lightweight PreProcessor:
  - Handles protocol
  - Provides signals needed for padding
  - Separate state machines for $sdi$ and $pdi$
  - 16 bits counters for segment lengths

- CipherCore (not provided):
  - Cipher function
  - Padding
  - PISO and SIPO if I/O width different than CAESAR API

- By-Pass FIFO: Stores and passes-on header information and Tag to PostProcessor

- PostProcessor
  - 16 bits counter for output segment length
  - Tag verification module
CAESAR Lightweight Block Diagram

Differences to HS are shown in blue.
CAESAR LW Block Diagram with PISO, and SIPO

**AEAD**
- sdi_data
- sdi_valid
- sdi_ready

**CipherCore**
- key
- Key PISO
- key_piso_valid
- key_piso_ready
- bdo
- bdo SIPO
- bdo_valid
- bdo_ready

**Cipher Function**
- end_of_block
- bdi_valid
- bdi_piso_valid
- bdi_ready
- bdi_valid_bytes
- bdi_type
- decrypt
- decrypt_in
- decrypt_out
- tag_verified
- end_of_block

**Data PISO**
- bdi_size
- bdi_eot
- bdi_eoi
- bdi_pad_loc
- bdi_valid
- bdi_ready
- bdi_partial
- bdi_valid_bytes
- cmd
- cmd_valid
- cmd_ready
- dout_valid
- dout_ready
- din_valid
- din_ready
- do_valid
- do_ready
- do_valid
- do_data
- do_ready

**Pre Processor**
- Pre
- w/8+1
- w/8
- Pre
- w/8+1

**Post Processor**
- Post
- w/8+1
- w/8+1

**Tag Comparator**
- Tag
- Comparator
- cmd
- cmd_valid
- cmd_ready
- end_of_block
- do_data
- do_valid
- do_ready

**Optional**
- Additional components and signals for protocol support.
Differences between High-Speed vs Lightweight Packages

Lightweight Development Package

- Supports bus widths 8, 16, and 32 bits
- Assumes that Padding is performed in CipherCore
- Sets the width of all data buses between modules PreProcessor, CipherCore, FIFO, and PostProcessor equal to bus width \( w \)
- Moves the Tag comparison to the PostProcessor (when possible)
- Does not provide data storage other than for control signals in the PreProcessor
Protocol: Instruction

16-bit Instruction with \( w=16 \)

<table>
<thead>
<tr>
<th>Opcode or Status</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>4</td>
</tr>
<tr>
<td>LSB</td>
<td>12</td>
</tr>
</tbody>
</table>

16-bit Instruction with \( w=8 \)

<table>
<thead>
<tr>
<th>Opcode or Status</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>4</td>
</tr>
<tr>
<td>MSB</td>
<td>4</td>
</tr>
<tr>
<td>LSB</td>
<td>2</td>
</tr>
</tbody>
</table>

States for Processing Instruction

** Opcode Description **

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>Authenticated Encryption (ENC)</td>
<td>1110</td>
<td>Success</td>
</tr>
<tr>
<td>0011</td>
<td>Authenticated Decryption (DEC)</td>
<td>1111</td>
<td>Failure</td>
</tr>
<tr>
<td>0100</td>
<td>Load Key (LDKEY)</td>
<td>Others</td>
<td>Reserved</td>
</tr>
<tr>
<td>0111</td>
<td>Activate Key (ACTKEY)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** MSB **

- \( w=8 \)
- \( w=16, 32 \)
Protocol: Segment Header

32-bit Header

With \( w = 8 \), and 16
Case Study

- Implementation of Ketje-Sr with integrated support of CAESAR API
- Implementation of Ketje-Sr using new CAESAR lightweight development package
  ⇒ Determine overhead of CAESAR LW package
- Implementation of Ascon using CAESAR LW package
  ⇒ Using CAESAR LW package on "unknown" algorithm

Target Devices for Benchmarking

<table>
<thead>
<tr>
<th>Vendor</th>
<th>FPGA Family</th>
<th>Device</th>
<th>Tool</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx</td>
<td>Spartan6</td>
<td>xc6slx16csg324-3</td>
<td>Xilinx ISE 14.7</td>
<td>ATHENa</td>
</tr>
<tr>
<td></td>
<td>Artix7</td>
<td>xc7a100tcsg324-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Altera</td>
<td>Cyclone IV</td>
<td>ep4ce22f17c6</td>
<td>Quartus Prime 16.0.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cyclone V</td>
<td>5ceba4f23c7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ketje

- Ketje is based on round reduced Keccak-$f$ called MonkeyWrap.
- Each round of Keccak-$p^*$ consists of five steps $\theta$, $\rho$, $\pi$, $\chi$, and $\iota$.
- In $\theta$ step, each bit in the state is Xored with two other bits from two different columns.
- The state bits are rotated for each lane using one of the 25 different offsets in $\rho$ step.
- Lanes are rearranged in $\pi$, integer multiplication in $\chi$.
- The last step is $\iota$, where a round constant is added.
Ketje-Sr Datapath

Port–A
RAM
Port–B

<<<1

reg–A

Rho

rcon

RAMK1 (MSB)

RAMK2 (LSB)

reg–K

do_data
sdi_data
pdi_data

RAM: Port-A (Read/Write); Port-B (Read only)
We implemented a Ketje-Sr using a 16-bit datapath and interface.

Datapath is the same for integrated CAESAR API support and using CAESAR LW package.

State is stored in a dual-port memory (RAM) with one read/write and one read-only ports.

To reduce the complexity of padding for key, the key size is fixed to 128-bits.

Two memory units (RAMK1, and RAMK2) with pre-stored values and a register (reg-K) for key storage and KeyPack operations.

Padding for message and AD using multiplexers.

Needs 160 clock cycles to process a 32-bit block.

\[ TP = \frac{32}{160} \times F \]
Ascon is a permutation based authenticated cipher.

Ascon-128, and Ascon-128a - two variants with block sizes of 64 and 128 respectively.

In each round, three sub transformations called constant-addition, substitution, and linear diffusion.

Constant-addition is the first operation in the round, where a constant is added to one of the five words. Twelve round constants are used.

Substitution layer uses 5x5 S-boxes.

Linear diffusion layer for diffusion across each of the five 64-bit words using circular shifts and an XOR.
Ascon Datapath

- do_data
- pdi_data
- sdi_data

Diagram showing data paths and components such as RAM, sbox, and rcon.
A 80-bit datapath is used in this design and a 32-bit interface.

- The state is stored in a shift-register, which either shifts by 64 bits or 80 bits.
- 80-bit shifts are used for substitution operation as 64 is not a multiple of 5 (5x5 S-boxes).
- 5x5 S-Boxes are implemented using look-up tables.
- The round constants are generated using two 4 bit registers and adders.
- Key is stored in a RAM
- Two 5-to-1 multiplexers are used to perform circular shifts in linear diffusion step (LDiff)

\[ TP = \frac{32}{54} \times F \]
## Implementation Results on Xilinx Spartan 6 FPGA

<table>
<thead>
<tr>
<th>Design</th>
<th>Slices</th>
<th>LUTs</th>
<th>FFs</th>
<th>Freq [MHz]</th>
<th>TP [Mbps]</th>
<th>TP/Area [Mbps/slice]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ketje-SR¹</td>
<td>140</td>
<td>436</td>
<td>98</td>
<td>122.4</td>
<td>24.48</td>
<td>0.17</td>
</tr>
<tr>
<td>Ketje-SR²</td>
<td>155</td>
<td>450</td>
<td>114</td>
<td>120.1</td>
<td>24.03</td>
<td>0.16</td>
</tr>
<tr>
<td>Overhead</td>
<td>15</td>
<td>14</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ascon²</td>
<td>203</td>
<td>645</td>
<td>393</td>
<td>137.5</td>
<td>154.25</td>
<td>0.76</td>
</tr>
<tr>
<td>Joltik³</td>
<td>168</td>
<td>534</td>
<td>381</td>
<td>200.0</td>
<td>426.67</td>
<td>2.54</td>
</tr>
<tr>
<td>ASCON⁴</td>
<td>202</td>
<td>540</td>
<td>383</td>
<td>231.0</td>
<td>1,852.80</td>
<td>9.17</td>
</tr>
</tbody>
</table>

¹ ⇒ Integrated CAESAR API; ² ⇒ CAESAR LW Package; ³ ⇒ No compliance with CAESAR API (non-GMU design); ⁴ ⇒ CAESAR HS Package (tweaked, non-GMU design);

- Using CAESAR LW Package leads to a small area increase.
- Three separate counters for sdi, pdi and do buses are used for simplicity and parallel operation.
- Counter for sdi can be dropped if cipher core provides end_of_key signal.
Results for Ascon with $w=32$ on Spartan 6.

- Supporting CAESAR API using LW Package leads to a small area increase.
- Supporting CAESAR API using HS Package leads to a larger area increase.
CAESAR LW Package allows for bus widths of 8 and 16 bits, which are not currently supported by CAESAR HS Package.

Using CAESAR LW Package leads to a small area increase over integrated designs, however this can be easily mitigated.

CEASAR HS Package leads to a much larger area increase than the LW Package as it converts between the word width and block width data buses.

The CAESAR LW-Package reduces the design time for LW implementations.

The CAESAR LW Package will be included in the next release of the Development Package for the CAESAR Hardware API.

The usage will be documented in the next release of the Implementer’s Guide to the CAESAR Hardware API.