GMU Hardware API for Authenticated Ciphers

Ekawat Homsirikamol,
William Diehl, Ahmed Ferozpurri,
Farnoud Farahmand,
Malik Umar Sharif, and Kris Gaj
George Mason University
USA

http://cryptography.gmu.edu
https://cryptography.gmu.edu/athena
GMU CAESAR Team

“Ice”
Homsirikamol

Will
Diehl

Ahmed
Ferozpuri

Farnoud
Farahmand

Umar
Sharif
Motivation

• Only Software API defined in the CAESAR Call for Submissions
• Hardware API can have a high influence on Area and Throughput/Area ratio of all candidates
• Hardware API typically much more difficult to modify than Software API
• Tentative deadline for second-round Verilog/VHDL: 2015.12.15
• Without a comprehensive hardware API, the comparison of existing and future implementations highly unreliable and potentially unfair
• Need for a uniform hardware API, endorsed by the CAESAR Committee, and adopted by all future implementers
Proposed Features (1)

- Inputs of arbitrary size in bytes (but a multiple of a byte only)
- Size of the entire message/ciphertext does not need to be known before the encryption/decryption starts (unless required by the algorithm itself)
- Independent data and key inputs
- Wide range of data port widths, $8 \leq w \leq 256$
- Simple high-level communication protocol
- Support for the burst mode
- Possible overlap among processing the current input block, reading the next input block, and storing the previous output block
Proposed Features (2)

• storing decrypted messages internally, until the result of authentication is known
• support for encryption and decryption within the same core, but only one of these two operations performed at a time
• ability to communicate with very simple, passive devices, such as FIFOs
• ease of extension to support existing communication interfaces and protocols, such as
  • AMBA-AXI4 from ARM - a de-facto standard for the System-on-Chip buses
  • PCI Express – high-bandwidth serial communication between PCs and hardware accelerator boards
Previous Work

• Popular general-purpose interfaces
  • ARM: AXI4, AXI4-Lite, AXI4-Stream (Advanced eXtensible Interface)
  • IBM: PLB (Processor Local Bus), OPB (On-chip Peripheral Bus)
  • Altera: Avalon
  • Xilinx: FSL (Fast Simplex Link)
  • Silicore Corp.: Wishbone (used by opencores.org)

• Hardware APIs used during the SHA-3 Contest
  • GMU, Virginia Tech, University College Cork, etc.

• Interfaces used so far in the CAESAR competition
  • minimalistic, algorithm specific
  • AXI4-Stream-based proposed by ETH
    (non-uniform, algorithm-specific user and data ports)
**ETH Interface Conventions**
*MS Thesis of Cyril Arnould, March 2015*

### Tiaoxin-346

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Width</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataInp_DI</td>
<td>264 bit</td>
<td>263 downto 261</td>
<td>Unused. Byte length of the data block. If the length is zero, the block is full.</td>
</tr>
<tr>
<td>UserInp_SI</td>
<td>3 bit</td>
<td>2</td>
<td>Signals whether we are encrypting (0) or decrypting (1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 downto 0</td>
<td>Datatype.</td>
</tr>
<tr>
<td>DataOut.DO</td>
<td>256 bit</td>
<td>255 downto 0</td>
<td>Output data.</td>
</tr>
<tr>
<td>UserOut.SO</td>
<td>1 bit</td>
<td>0</td>
<td>Signals whether the received tag matches the computed tag, i.e. whether decryption was successful or not.</td>
</tr>
</tbody>
</table>

### ICEPOLE

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Width</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataInp_DI</td>
<td>1024 bit</td>
<td>1023 downto 0</td>
<td>Input data.</td>
</tr>
<tr>
<td>UserInp_SI</td>
<td>10/11 bit</td>
<td></td>
<td>Signals whether the tag block already contains the key and nonce to initialize the next message (1) or not (0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Signals whether we are encrypting (0) or decrypting (1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Indicates that the current block is the last associated data or message block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Toggles the FrameBit_SP flip-flop, is therefore required to be zero for all other datatypes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bytelenth of the data block.</td>
</tr>
<tr>
<td>DataOut.DO</td>
<td>1024 bit</td>
<td>1023 downto 0</td>
<td>Output data.</td>
</tr>
<tr>
<td>UserOut.SO</td>
<td>1 bit</td>
<td>0</td>
<td>Signals whether the received tag matches the computed tag, i.e. whether decryption was successful or not.</td>
</tr>
</tbody>
</table>
AEAD Interface

PDI
Public Data Input Ports

SDI
Secret Data Input Ports

AEAD

w
\[\text{sdi} \rightarrow \\
\text{sdi\_valid} \rightarrow \]
\[\text{sdi\_ready} \\
\leftrightarrow \]

\[\text{clk} \\
\text{rst} \rightarrow \]

\[\text{pdi} \rightarrow \]
\[\text{pdi\_valid} \rightarrow \]
\[\text{pdi\_ready} \rightarrow \]
\[\text{do} \rightarrow \]
\[\text{do\_valid} \rightarrow \]
\[\text{do\_ready} \rightarrow \]

w
\[\text{clk} \rightarrow \]
\[\text{rst} \rightarrow \]

DO
Data Output Ports
Typical External Circuits (1) – AXI4 IPs

AXI4–Stream Master
- m_axis_tdata
- m_axis_tvalid
- m_axis_tready

SDI FIFO
- dout
- empty
- read

clk \downarrow \downarrow \begin{array}{l} \text{clk} \\ \text{rst} \end{array}

AEAD
- pdi
- pdi_valid
- pdi_ready
- do
- do_valid
- do_ready

AXI4–Stream Slave
- s_axis_tdata
- s_axis_tvalid
- s_axis_tready

clk \downarrow \downarrow \begin{array}{l} \text{clk} \\ \text{rst} \end{array}
Typical External Circuits (2) - FIFOs

PDI FIFO
- dout
- empty
- read

SDI FIFO
- dout
- empty
- read

AEAD
- clk
- rst
- pdi
- pdi_valid
- pdi_ready
- do
- do_valid
- do_ready

DO FIFO
- din
- write
- full
Input and Output of an Authenticated Cipher

Npub (Public Message Number), typically Nonce
Nsec (Secret Message Number) [supported by few algorithms]
Enc Nsec – Encrypted Secret Message Number
AD – Associated Data
Format of Secret Data Input

<table>
<thead>
<tr>
<th>w-bit</th>
<th>instruction = LDKEY</th>
<th>seg_0 = Key</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>w-bit</th>
<th>instruction = LDRKEY</th>
<th>seg_0 = Round Key</th>
</tr>
</thead>
</table>

Loading Main **Key**

for HW implementations **with** Key Scheduling

Loading **Round Keys**

for HW implementations **without** Key Scheduling
# Format of Public Data Input

Single segment or **multiple segments** per data type (AD and/or Message)

<table>
<thead>
<tr>
<th>w bits</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_0 = Npub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1 = AD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2 = Message</td>
<td></td>
</tr>
</tbody>
</table>

OR

<table>
<thead>
<tr>
<th>w bits</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_0 = Npub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1 = AD_0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2 = AD_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_3_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_3 = Message_0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_4_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_4 = Message_1</td>
<td></td>
</tr>
</tbody>
</table>
Format of Public Data Input for Ciphers without Nsec

PDI for encryption

DO for encryption = PDI for decryption

DO for decryption
Format of Public Data Input for Ciphers with Nsec

- **w-bit**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Segment Header</th>
<th>Segment Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTKEY</td>
<td>seg_0</td>
<td>Npub</td>
</tr>
<tr>
<td>ENC</td>
<td>seg_1</td>
<td>Nsec</td>
</tr>
<tr>
<td>DEC</td>
<td>seg_0</td>
<td>Npub</td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>Enc Nsec</td>
</tr>
<tr>
<td></td>
<td>seg_2</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_3</td>
<td>Message</td>
</tr>
<tr>
<td>ENC</td>
<td>seg_0</td>
<td>Nsec</td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_2</td>
<td>Message</td>
</tr>
</tbody>
</table>

**PDI for encryption**

**DO for encryption = PDI for decryption**

**DO for decryption**
## Instruction/Status Word Format

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Msg ID</strong></td>
<td><strong>0000</strong></td>
</tr>
</tbody>
</table>

Divided into $\lceil \frac{24}{w} \rceil$ words, starting from MSB

### Opcode:
- 0010 – Authenticated Encryption (ENC)
- 0011 – Authenticated Decryption (DEC)
- 0100 – Load Key (LDKEY)
- 0101 – Load Round Key (LDRKEY)
- 0111 – Activate Key (ACTKEY)

### Status:
- 1110 – Pass
- 1111 – Fail
- Others – Reserved
Segment Header Format

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>0001</td>
<td>Npub</td>
</tr>
<tr>
<td>0010</td>
<td>AD</td>
</tr>
<tr>
<td>0011</td>
<td>Message</td>
</tr>
<tr>
<td>1000</td>
<td>Nsec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>Ciphertext</td>
</tr>
<tr>
<td>0101</td>
<td>Tag</td>
</tr>
<tr>
<td>0110</td>
<td>Key</td>
</tr>
<tr>
<td>0111</td>
<td>Round Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Enc Nsec</td>
</tr>
</tbody>
</table>

- **Msg ID**: 8 bits
- **Info**: 8 bits
- **000...0**: 4 bits
- **Seg Len**: 4 bits

**Segment Type**

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>0001</td>
<td>Npub</td>
</tr>
<tr>
<td>0010</td>
<td>AD</td>
</tr>
<tr>
<td>0011</td>
<td>Message</td>
</tr>
<tr>
<td>1000</td>
<td>Nsec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>Ciphertext</td>
</tr>
<tr>
<td>0101</td>
<td>Tag</td>
</tr>
<tr>
<td>0110</td>
<td>Key</td>
</tr>
<tr>
<td>0111</td>
<td>Round Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Enc Nsec</td>
</tr>
</tbody>
</table>

EOI = 1 if the last segment of input
0 otherwise

EOT = 1 if the last segment of its type
(AD, Message, Ciphertext),
0 otherwise

Divided into \([16+s]/w\) words, starting from MSB
Universal Testbench & Automated Test Vector Generation

• Universal Testbench supporting any authenticated cipher core following GMU AEAD API

• Change of cipher requires only changing test vector file

• A Python script created to automatically generate test vector files representing multiple test cases
  • Encryption and Decryption
  • Empty Associated Data and/or Empty Message/Ciphertext
  • Various, randomly selected sizes of AD and Message/Ciphertext
  • Valid tag and invalid tag cases

• All source codes made available at GMU ATHENA website
Block Diagram of AEAD
PreProcessor and PostProcessor for High-Speed Implementations (1)

PreProcessor:

• parsing segment headers
• loading and activating keys
• Serial-In-Parallel-Out loading of input blocks
• padding input blocks
• keeping track of the number of data bytes left to process

PostProcessor:

• clearing any portions of output blocks not belonging to ciphertext or plaintext
• Parallel-In-Serial-Out conversion of output blocks into words
• formatting output words into segments
• storing decrypted messages in AUX FIFO, until the result of authentication is known
• generating an error word if authentication fails
PreProcessor and PostProcessor for High-Speed Implementations (2)

Features:

• Ease of use
• No influence on the maximum clock frequency of AEAD (up to 300 MHz in Virtex 7)
• Limited area overhead
• Clear separation between the AEAD Core unit and internal FIFOs
  • Bypass FIFO – for passing headers and associated data directly to PostProcessor
  • AUX FIFO – for temporarily storing unauthenticated messages after decryption

Benefits:

• The designers can focus on designing the CipherCore specific to a given algorithm, without worrying about the functionality common for multiple algorithms
• Full-block width interface of the CipherCore
Test of Compatibility with AXI4 IP Cores

Correct operation verified and performance measured experimentally using the ZedBoard based on Xilinx ZYNQ XC7Z020 All Programmable SoC
AES & Keccak-F Permutation VHDL Codes

- Additional support provided for designers of Cipher Cores of CAESAR candidates based on AES and Keccak
- Fully verified VHDL codes, block diagrams, and ASM charts of
  - AES
  - Keccak-F Permutation
- All resources made available at the GMU ATHENa website
  https://cryptography.gmu.edu/athena
Generation of Results

- Generation of results possible for
  - CipherCore – full block width interface, incomplete functionality
  - AEAD Core - recommended
  - AEAD – difficulty with setting BRAM usage to 0 (if desired)

- Use of wrappers:
  - Out-of-context (OOC) mode available in Xilinx Vivado (no pin limit)
  - Generic wrappers available in case the number of port bits exceeds the total number of user pins, when using Xilinx ISE
  - GMU Wrappers: 5 ports only (clk, rst, sin, sout, piso_mux_sel)

- Recommended Optimization Procedure
  - ATHENa for Xilinx ISE and Altera Quartus II
  - 25 default optimization strategies for Xilinx Vivado
AEAD Core vs. CipherCore Area Overhead for Virtex-6

**Overhead** = \( \frac{\text{LUT(AEAD\_Core)} - \text{LUT(CipherCore)}}{\text{LUT(AEAD\_Core)}} \times 100\% \)
ATHENa Database of Results for Authenticated Ciphers

- Available at http://cryptography.gmu.edu/athena

- Developed by John Pham, a Master’s-level student of Jens-Peter Kaps

- Results can be entered by designers themselves. If you would like to do that, please contact us regarding an account.

- The ATHENa Option Optimization Tool supports automatic generation of results suitable for uploading to the database
# Authenticated Encryption FPGA Ranking

**Result Filtering**

**Algorithm Group**
- Round 2 CAESAR Candidates and current standards
- Round 1 CAESAR Candidates and current standards

**Implementation Type**
- High Speed Implementations, Single Message Architectures
- High Speed Implementations, All Architectures
- Low Area Implementations

**Implementation Approach**
- Register Transfer Level
- High Level Synthesis
- HW/SW Codesign
- Any

**Hardware API**
- GMU_AEAD_Core_API_v1
- GMU_AEAD_API_v1
- GMU_CipherCore_API_v1
- Full-Block width(custom)
- GMU_AEAD_Core_API_v0

**Key Size**
- 128
- From 80 to 256
- Any
## Ranking View (2)

<table>
<thead>
<tr>
<th>Result ID</th>
<th>Algorithm</th>
<th>Key Size [bits]</th>
<th>Implementation Approach</th>
<th>Hardware API</th>
<th>Arch Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>154</td>
<td>ICEPOLE</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1.1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>73</td>
<td>Keyak</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>62</td>
<td>AES-GCM</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>65</td>
<td>CLOC</td>
<td>128</td>
<td>HLS</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>80</td>
<td>PRIMATEs-GIBBON</td>
<td>120</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>144</td>
<td>OCB</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>124</td>
<td>PRIMATEs-HANUMAN</td>
<td>120</td>
<td>HLS</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>86</td>
<td>SCREAM</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>142</td>
<td>Joltik</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>75</td>
<td>POET</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>60</td>
<td>AES-COPA</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
</tbody>
</table>

Please note that codes with primitives, megafonctions, or embedded resources are not fully portable.
Database of Results

Ranking View:

Supports the choice of

I. Hardware API (e.g., GMU_AEAD_Core_API_v1, GMU_AEAD_API_v1, GMU_CipherCore_API_v1)

II. Family (e.g., Virtex 6 (default), Virtex 7, Zynq 7000)

III. Operation (Authenticated Encryption (default), Authenticated Decryption, Authentication Only)

IV. Unit of Area (for Xilinx FPGAs: LUTs vs. Slices)

V. Ranking criteria (Throughput/Area (default), Throughput, Area)

Table View:

- more flexibility in terms of filtering, reviewing, ranking, searching for, and comparing results with one another
Conclusions

• Complete Hardware API for authenticated ciphers, including
  • Interface
  • Communication Protocol
• Design with the GMU hardware API facilitated by
  • Detailed specification (ePrint 2015/669)
  • Universal testbench and Automated Test Vector Generation
  • PreProcessor and PostProcessor Units for high-speed implementations
  • Universal wrappers and scripts for generating results
  • AES and Keccak-F Permutation source codes and diagrams
  • Ease of recording and comparing results using ATHENa database
• GMU proposal open for discussion and possible improvements through
  • Better specification
  • Better implementation of supporting codes
Extensions of the Current Hardware API (under development)

- support for two-pass algorithms
- pipelining of multiple streams of data
- detection and reporting of input formatting errors
- accepting inputs with padding done in software
Thank you!

Comments?

Questions?

Suggestions?

http://cryptography.gmu.edu/athena
Choose: Download

http://cryptography.gmu.edu