GMU Hardware API for Authenticated Ciphers

Ekawat Homsirikamol,
William Diehl, Ahmed Ferozpuri,
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

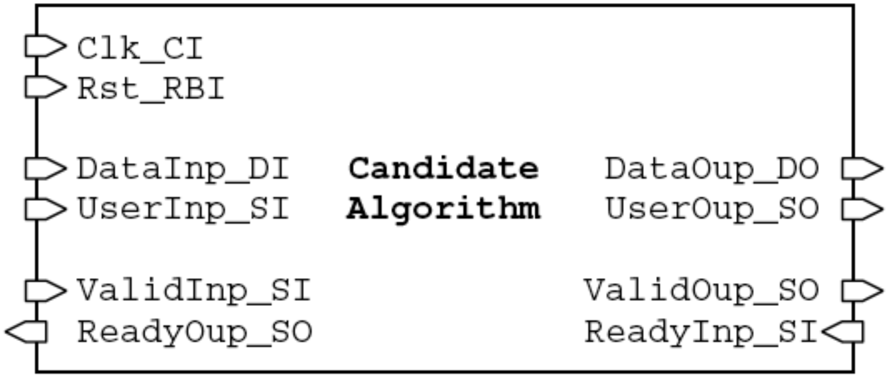
Proposed Features (2)

- 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

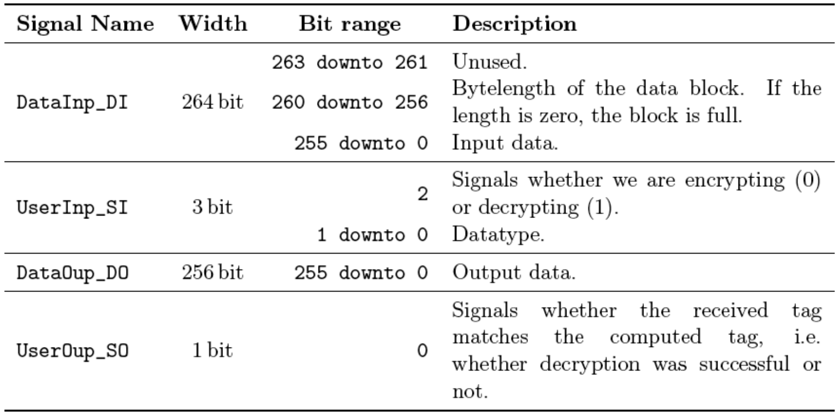
- **Popular general-purpose interfaces**
  - **ARM: AXI4, AXI4-Lite, AXI4-Stream** (Advanced eXtensible Interface)
  - **IBM: PLB** (Processor Local Bus), **OPB** (On-chip Peripheral Bus)
  - **Altera: Avalon**
  - **Xilinx: FSL** (Fast Simplex Link)
  - **Silicore Corp.: Wishbone** (used by opencores.org)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



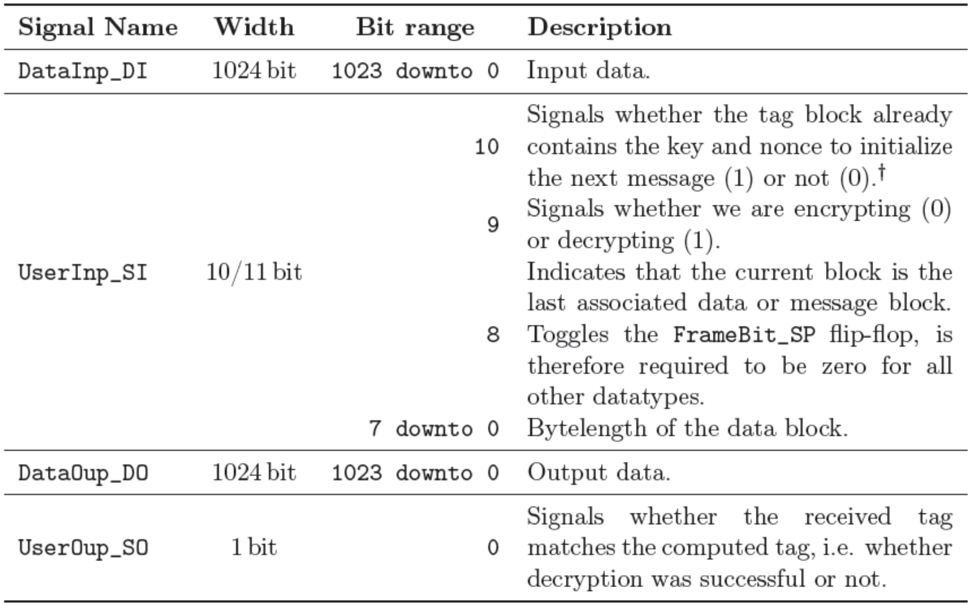
Clk_CI
Rst_RBI
DataInp_DI
UserInp_SI
ValidInp_SI
ReadyOup_SO
Candidate Algorithm
DataOup_DO
UserOup_SO
ValidOup_SO
ReadyInp_SI

Tiaoxin-346



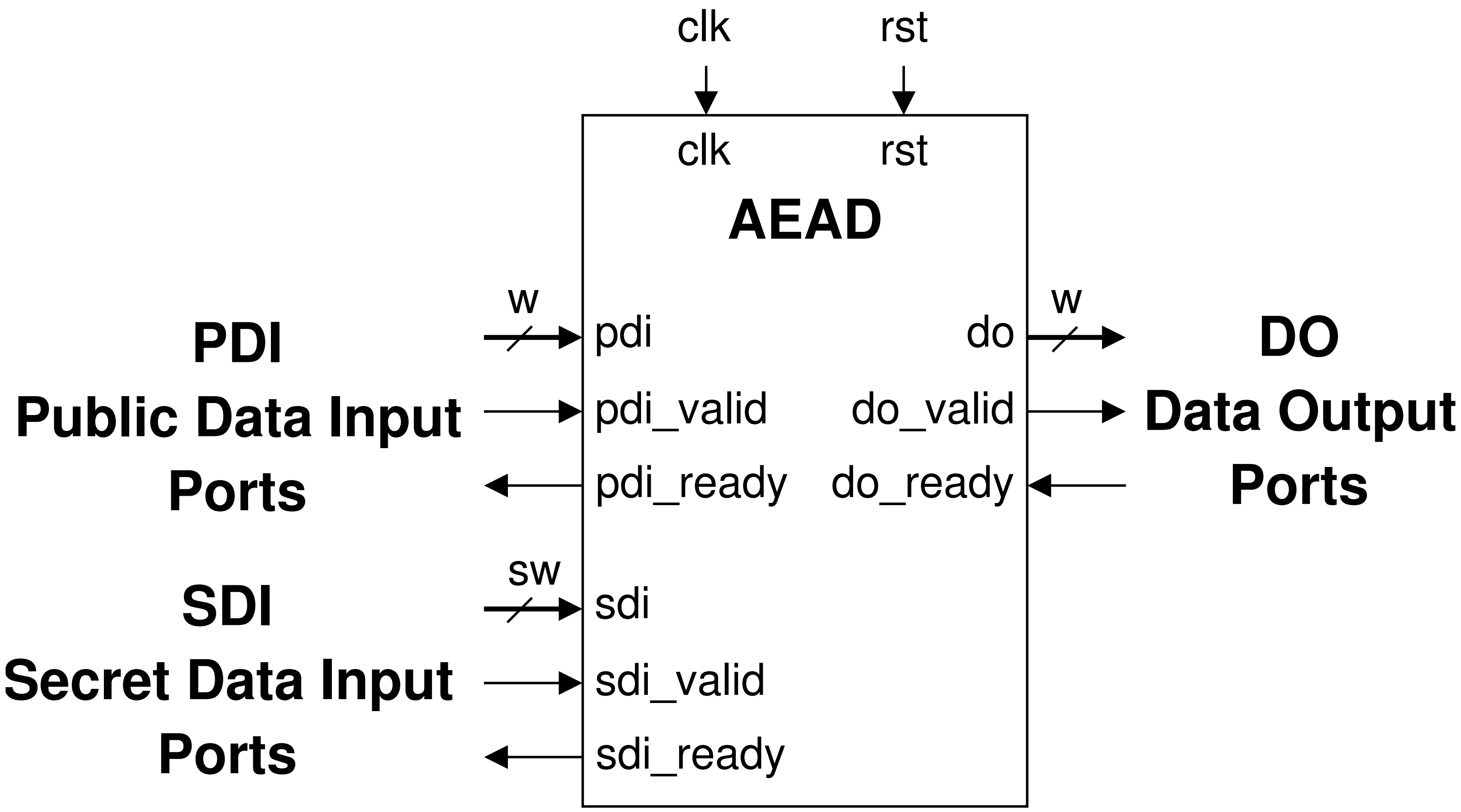
Signal Name	Width	Bit range	Description
DataInp_DI	264 bit	263 downto 261	Unused.
		260 downto 256	Bytelength of the data block. If the length is zero, the block is full.
		255 downto 0	Input data.
UserInp_SI	3 bit	2	Signals whether we are encrypting (0) or decrypting (1).
		1 downto 0	Datatype.
DataOup_DO	256 bit	255 downto 0	Output data.
UserOup_SO	1 bit	0	Signals whether the received tag matches the computed tag, i.e. whether decryption was successful or not.

ICEPOLE

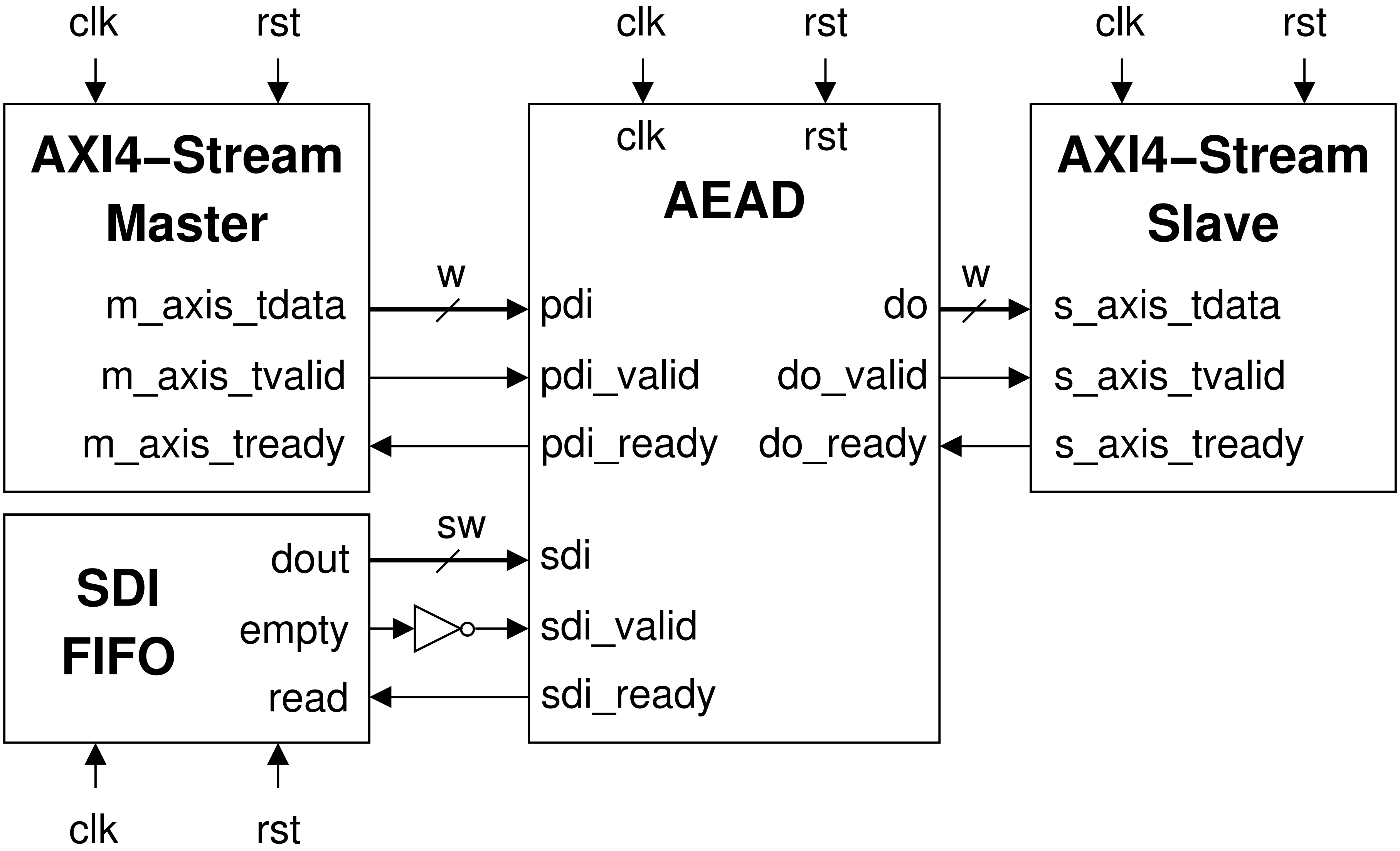


Signal Name	Width	Bit range	Description
DataInp_DI	1024 bit	1023 downto 0	Input data.
UserInp_SI	10/11 bit	10	Signals whether the tag block already contains the key and nonce to initialize the next message (1) or not (0).†
		9	Signals whether we are encrypting (0) or decrypting (1).
		8	Indicates that the current block is the last associated data or message block. Toggles the FrameBit_SP flip-flop, is therefore required to be zero for all other datatypes.
		7 downto 0	Bytelength of the data block.
DataOup_DO	1024 bit	1023 downto 0	Output data.
UserOup_SO	1 bit	0	Signals whether the received tag matches the computed tag, i.e. whether decryption was successful or not.

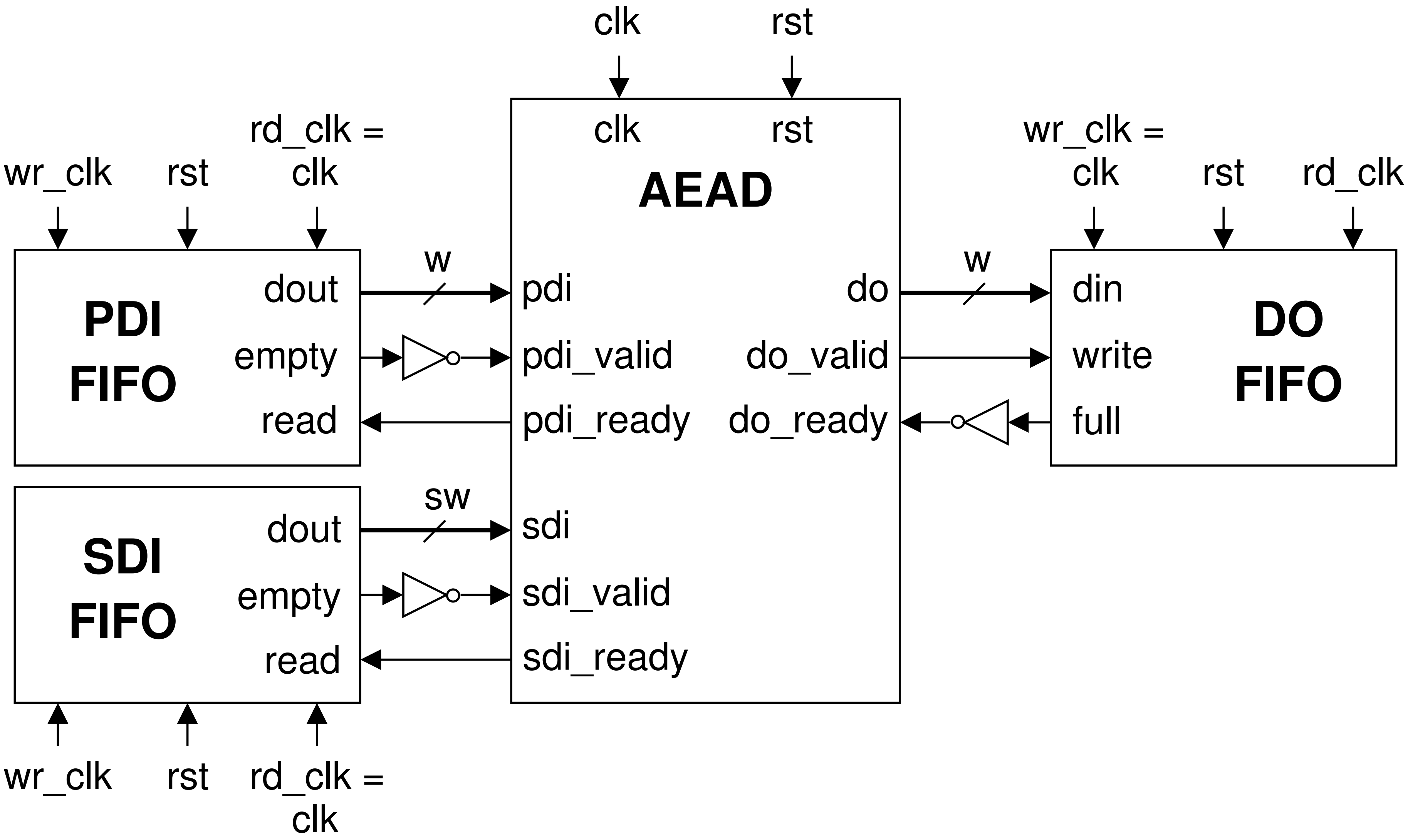
AEAD Interface



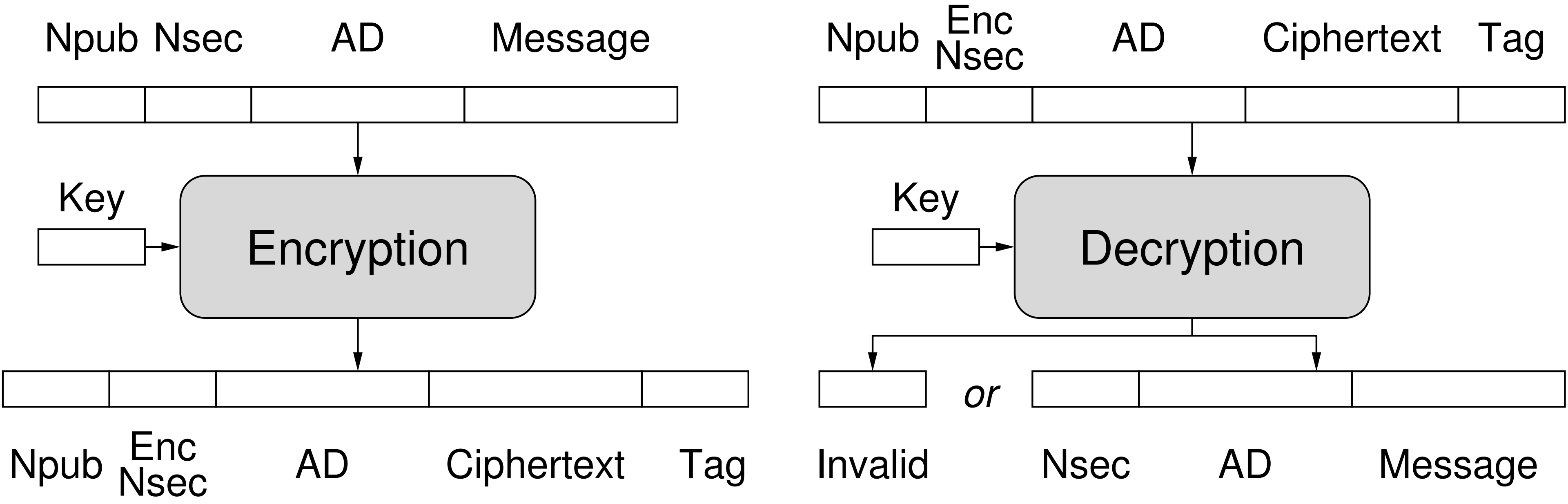
Typical External Circuits (1) – AXI4 IPs



Typical External Circuits (2) - FIFOs



Input and Output of an Authenticated Cipher



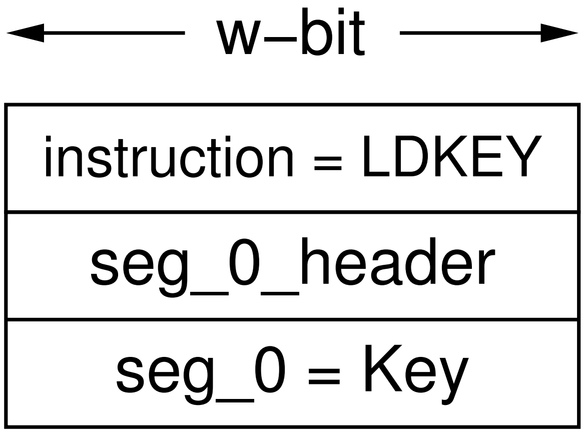
Npub (Public Message Number), typically Nonce

Nsec (Secret Message Number) [supported by few algorithms]

Enc Nsec – Encrypted Secret Message Number

AD – Associated Data

Format of Secret Data Input



w–bit

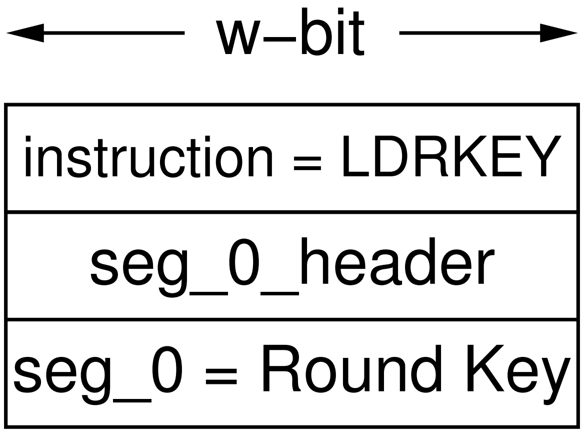
instruction = LDKEY

seg_0_header

seg_0 = Key

Loading Main **Key**

for HW implementations
with Key Scheduling



w–bit

instruction = LDRKEY

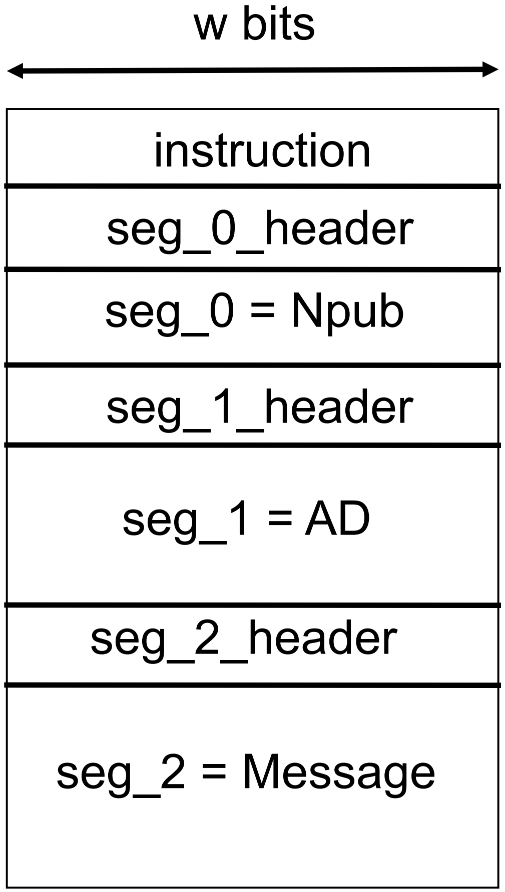
seg_0_header

seg_0 = Round Key

Loading **Round Keys**

for HW implementations
without Key Scheduling

Format of Public Data Input



w bits

instruction

seg_0_header

seg_0 = Npub

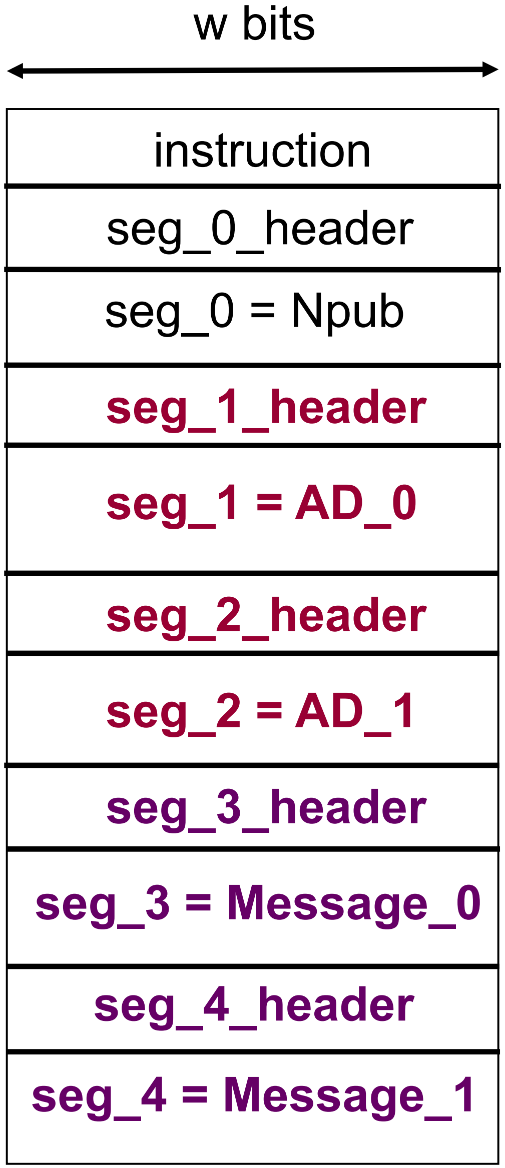
seg_1_header

seg_1 = AD

seg_2_header

seg_2 = Message

OR



w bits

instruction

seg_0_header

seg_0 = Npub

seg_1_header

seg_1 = AD_0

seg_2_header

seg_2 = AD_1

seg_3_header

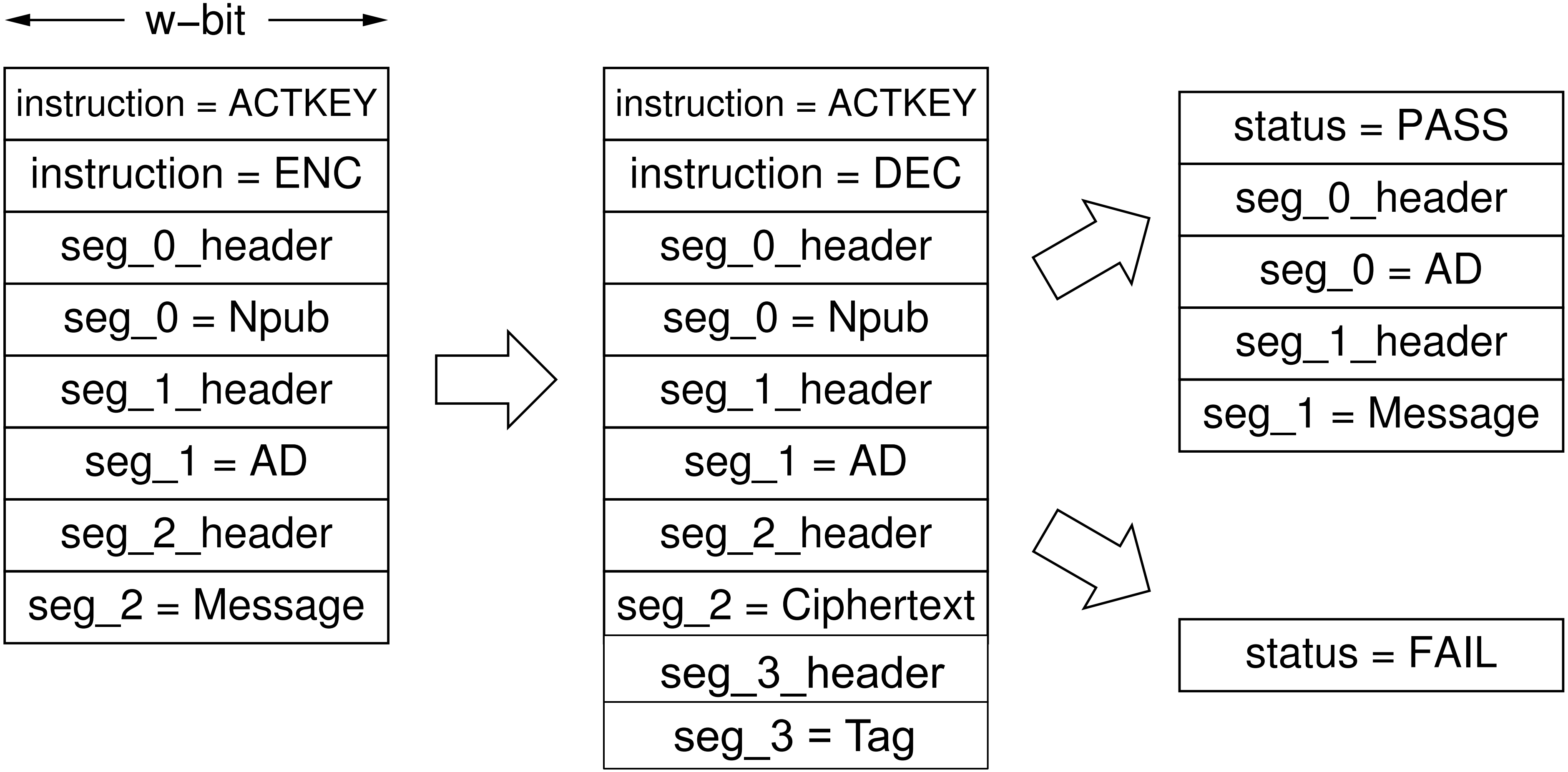
seg_3 = Message_0

seg_4_header

seg_4 = Message_1

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

Format of Public Data Input for Ciphers without Nsec



instruction = ACTKEY
instruction = ENC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Message

instruction = ACTKEY
instruction = DEC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Ciphertext
seg_3_header
seg_3 = Tag

status = PASS
seg_0_header
seg_0 = AD
seg_1_header
seg_1 = Message

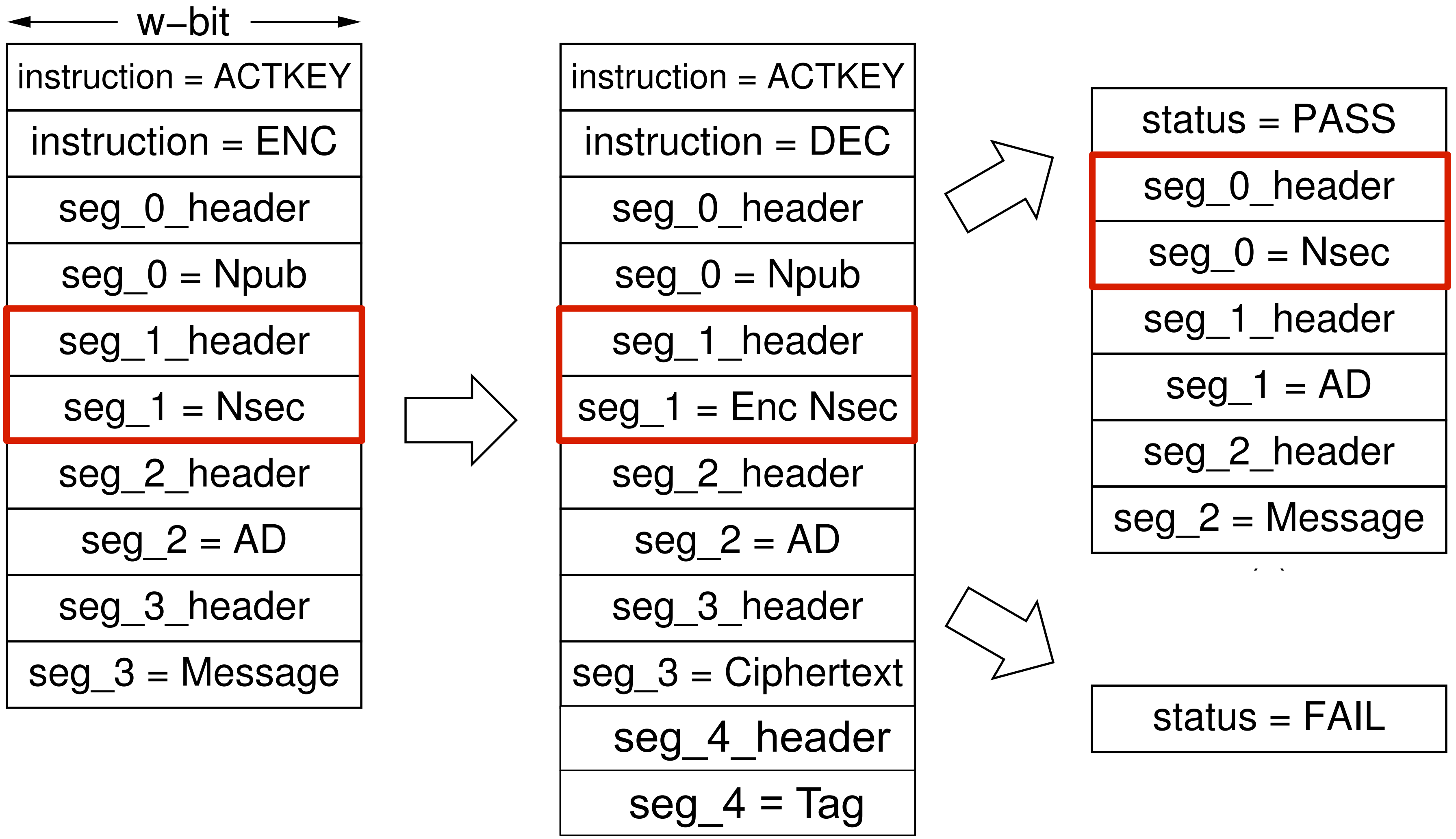
status = FAIL

PDI for encryption

DO for encryption = PDI for decryption

DO for decryption

Format of Public Data Input for Ciphers with Nsec



w–bit

instruction = ACTKEY
instruction = ENC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = Nsec
seg_2_header
seg_2 = AD
seg_3_header
seg_3 = Message

PDI for encryption

instruction = ACTKEY
instruction = DEC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = Enc Nsec
seg_2_header
seg_2 = AD
seg_3_header
seg_3 = Ciphertext
seg_4_header
seg_4 = Tag

DO for encryption =
PDI for decryption

status = PASS
seg_0_header
seg_0 = Nsec
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Message

status = FAIL

DO for decryption

Instruction/Status Word Format

MSB LSB

Msg ID	0000	Opcode or Status	Key ID
8	4	4	8

Divided into $\lceil 24/w \rceil$ 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)

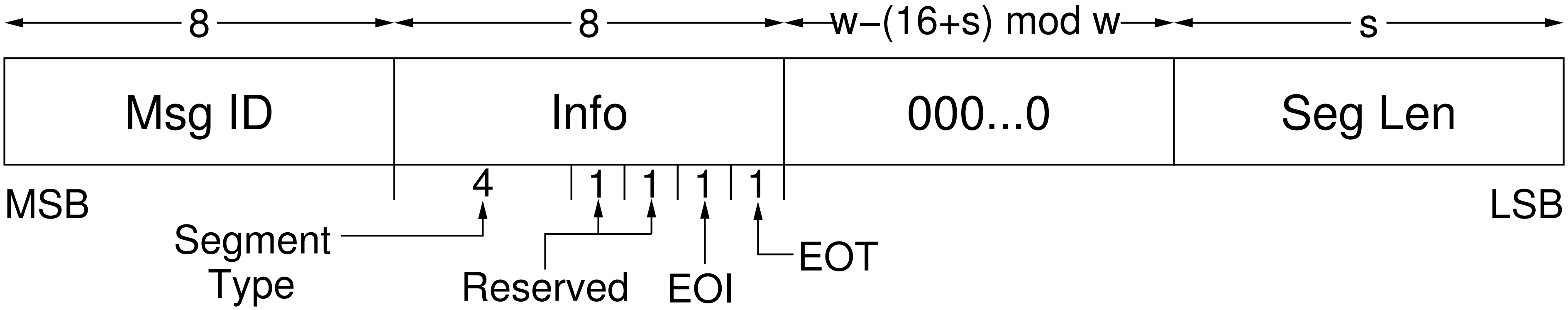
Status:

1110 – Pass

1111 – Fail

Others – Reserved

Segment Header Format



Divided into ⌈(16+s)/w⌉ words, starting from MSB

EOI = 1 if the last segment of input
0 otherwise

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

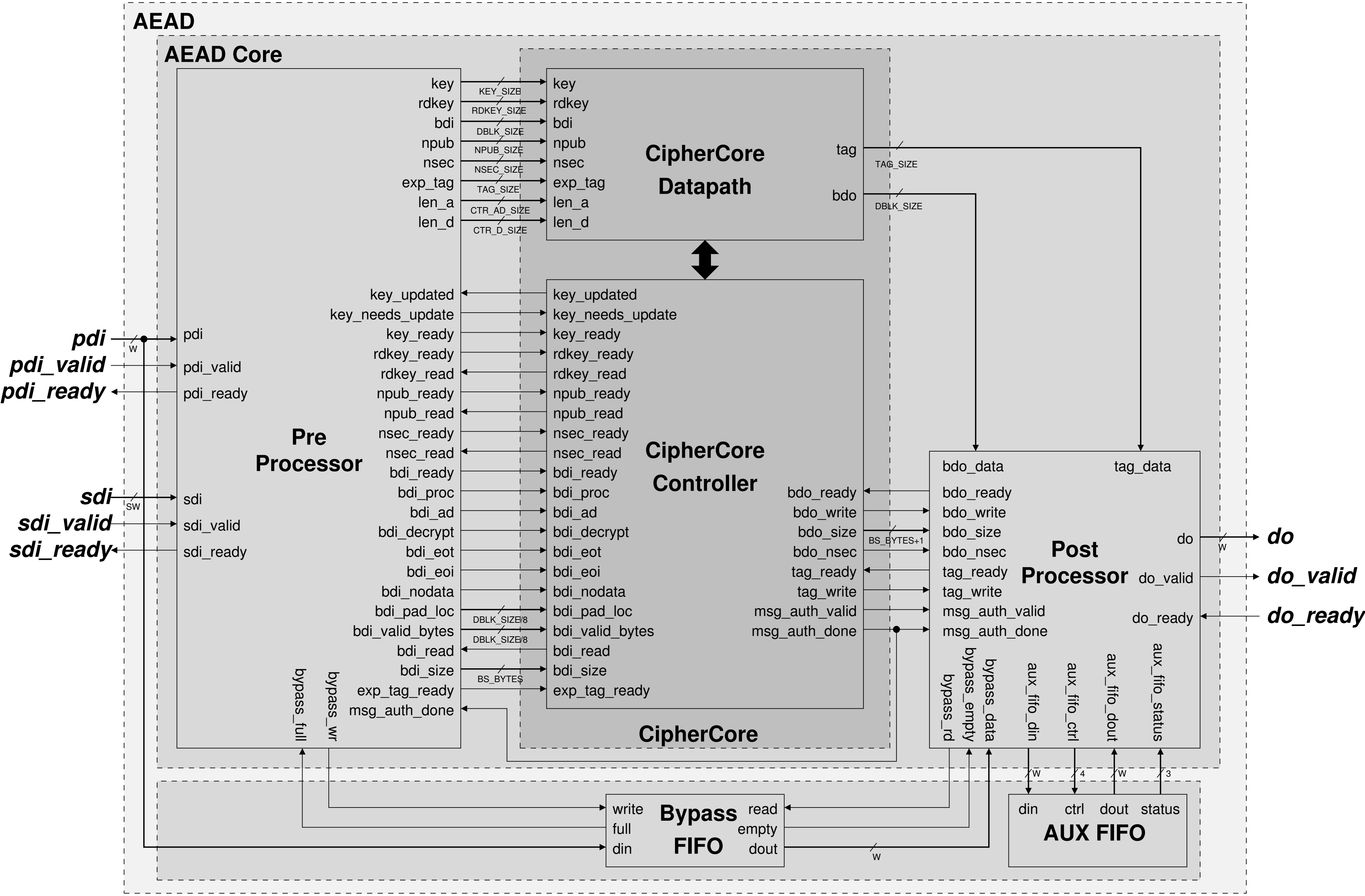
Segment Type:

0000 – Reserved	0100 – Ciphertext
0001 – Npub	0101 – Tag
0010 – AD	0110 – Key
0011 – Message	0111 – Round Key
1000 – Nsec	1001 – Enc Nsec

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)

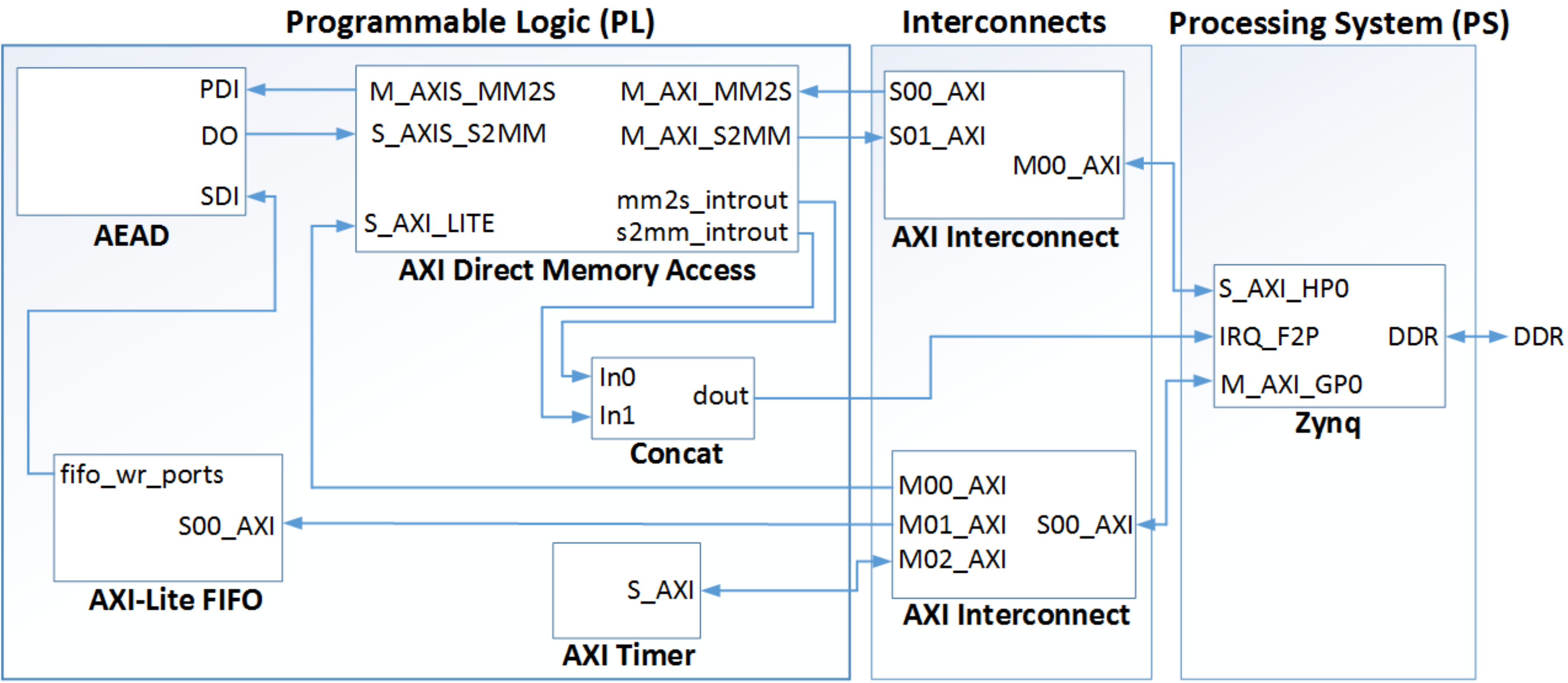
Features:

- 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

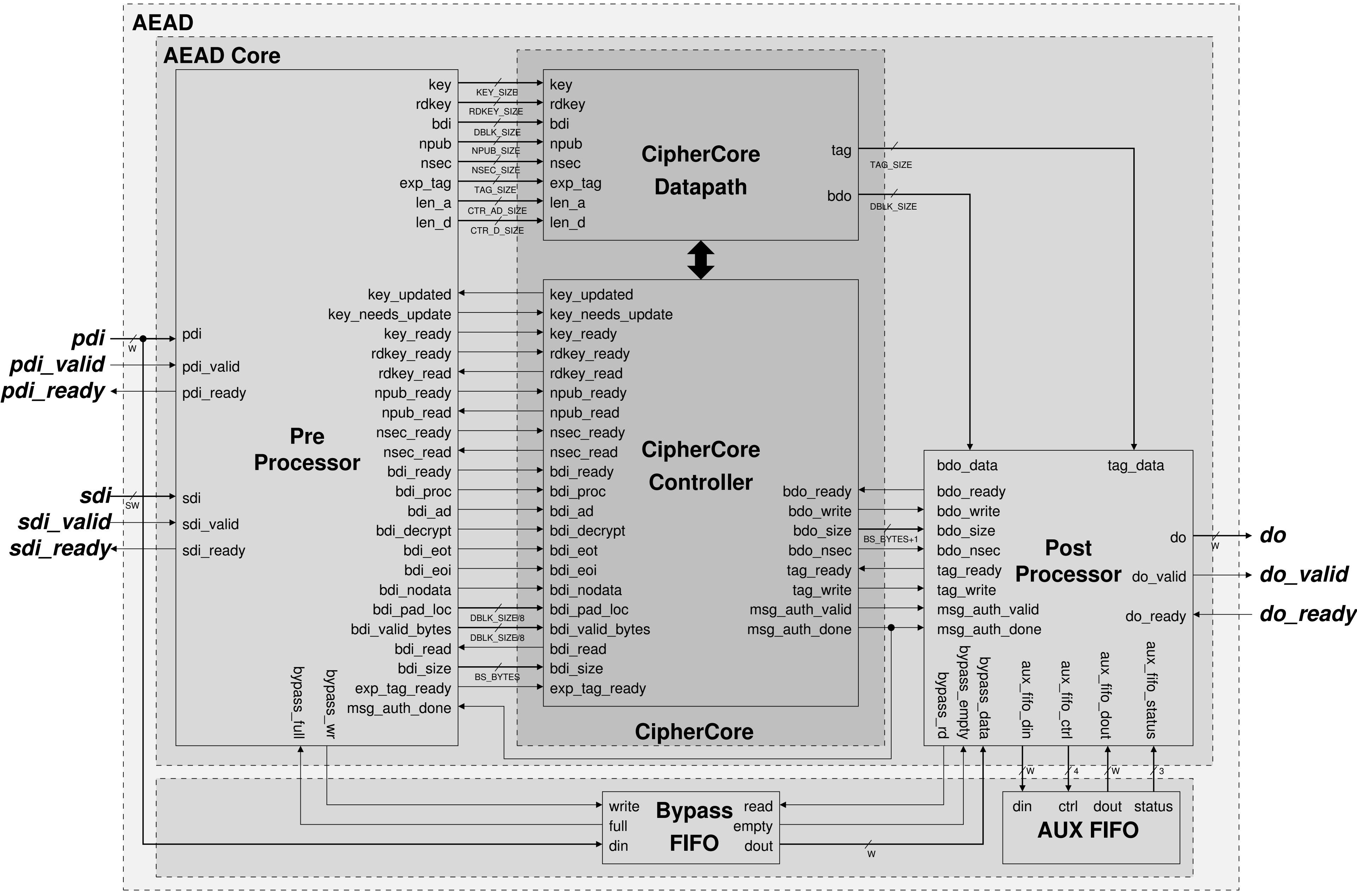
- Additional support provided for designers of Cipher Cores of CAESAR candidates based on AES and Keccak
- Fully verified VHDL codes, block diagrams, and ASM charts of
  - AES
  - Keccak-F Permutation
- All resources made available at the GMU ATHENa website

https://cryptography.gmu.edu/athena

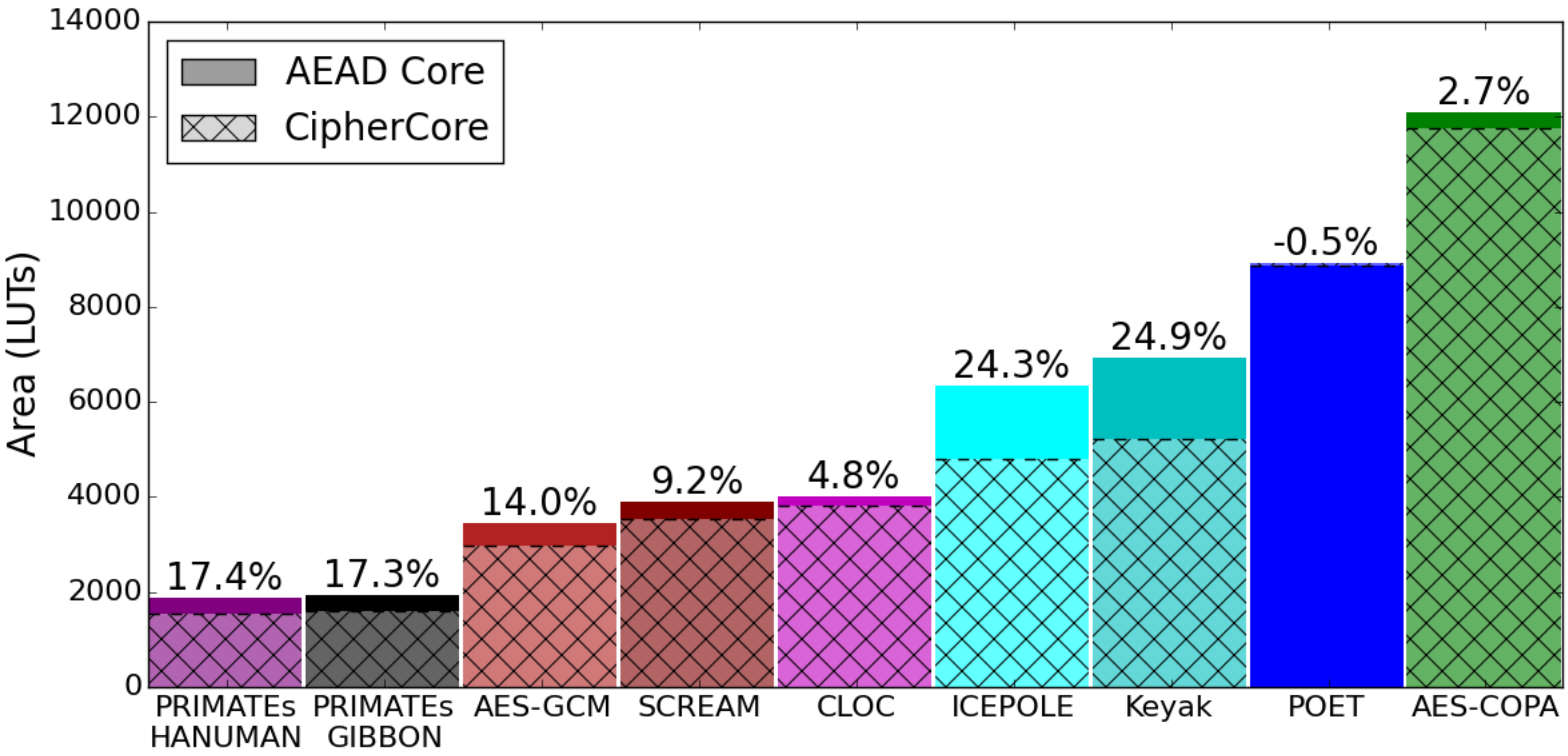
Generation of Results

- 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

Block Diagram of AEAD



AEAD Core vs. CipherCore Area Overhead for Virtex-6



$$\text{Overhead} = \frac{\text{LUT(AEAD\_Core)-LUT(CipherCore)}}{\text{LUT(AEAD\_Core)}} \times 100\%$$

ATHENa Database of Results for Authenticated Ciphers

- Available at

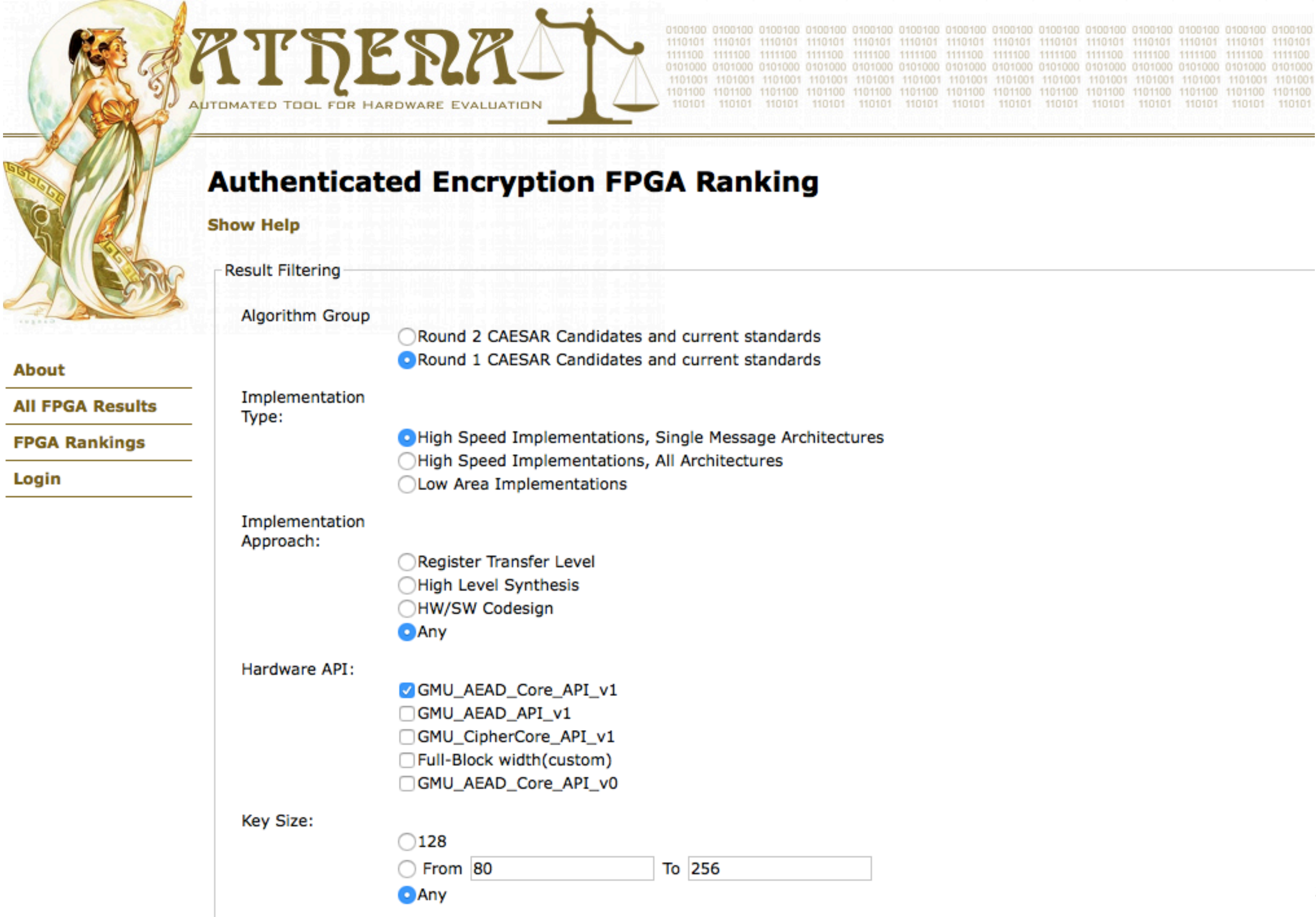
  http://cryptography.gmu.edu/athena

- 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

Ranking View (1)



ATHENA

AUTOMATED TOOL FOR HARDWARE EVALUATION

Authenticated Encryption FPGA Ranking

Show Help

- **About**
- **All FPGA Results**
- **FPGA Rankings**
- **Login**

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

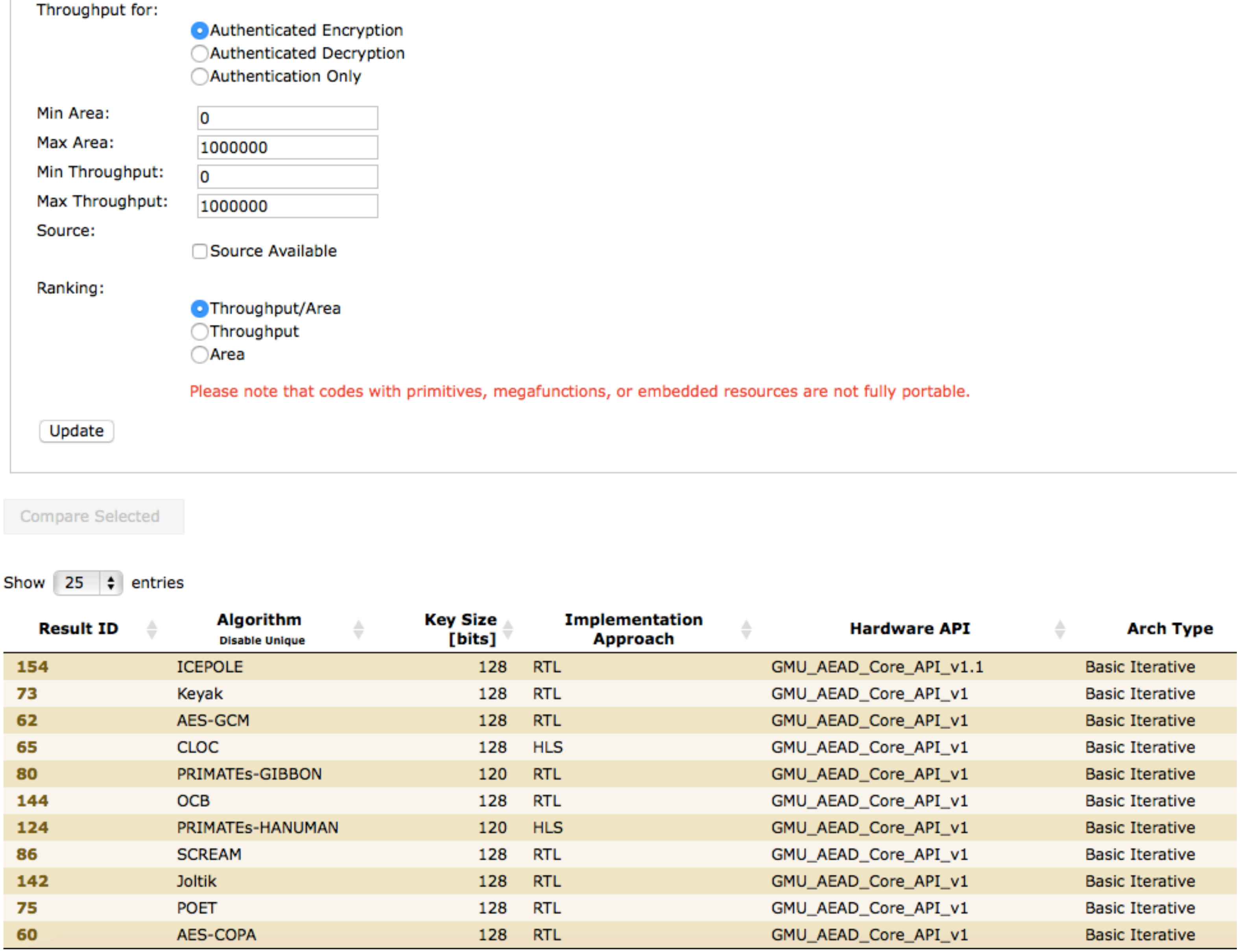
Hardware API:

☑ 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)



Throughput for:

- ● Authenticated Encryption
- ○ Authenticated Decryption
- ○ Authentication Only

Min Area: 0

Max Area: 1000000

Min Throughput: 0

Max Throughput: 1000000

Source:

□ Source Available

Ranking:

- ● Throughput/Area
- ○ Throughput
- ○ Area

Please note that codes with primitives, megafunctions, or embedded resources are not fully portable.

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

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