ECE 545

Project Background

Fall 2015
Crypto 101
Cryptography is Everywhere

- Buying a book on-line
- Withdrawing cash from ATM
- Teleconferencing over Intranets
- Backing up files on remote server
Alice: I love you! Bob
Alice: I love you!

Bob
Basic Security Services (1)

1. Confidentiality

Bob → Alice

2. Message integrity

Bob → Alice

3. Message authentication

Bob → Alice
Bob

\( N \) \quad Message

\( K_{AB} \) \quad Cipher

\( N \) \quad Ciphertext

Alice

\( N \) \quad Ciphertext

\( K_{AB} \) \quad Cipher

\( N \) \quad Message

\( K_{AB} \) - Secret key of Alice and Bob
\( N \) – Nonce or Initialization Vector
Authentication
Message Authentication Code - MAC

Bob

Tag

K_{AB}

MAC

Message

K_{AB} - Secret key of Alice and Bob

Alice

Tag

MAC

Tag'

valid/invalid

=
Confidentiality & Authentication
Authenticated Ciphers

Bob

N  Message

\[ K_{AB} \]

Authenticated Cipher Encryption

N  Ciphertext  Tag

Alice

N  Ciphertext  Tag

\[ K_{AB} \]

Authenticated Cipher Decryption

invalid  or  Message

\[ K_{AB} \] - Secret key of Alice and Bob
N – Nonce or Initialization Vector
Confidentiality & Authentication

Authenticated Ciphers

Npub - Public Message Number
Nsec - Secret Message Number
Enc Nsec - Encrypted Secret Message Number
AD - Associated Data
K_{AB} - Secret key of Alice and Bob
Cryptographic Transformations
Most Often Implemented in Practice

Secret-Key Ciphers
- Block Ciphers
- Stream Ciphers
  - encryption

Hash Functions
  - message & user authentication

Public-Key Cryptosystems
  - digital signatures
  - key agreement
  - key exchange
Hash Function

It is computationally infeasible to find such $m$ and $m'$ that $h(m) = h(m')$.
Hash Functions in Digital Signature Schemes

Alice

Message

Hash function

Hash value

Signature

Public key cipher

Alice’s private key

Bob

Message

Signature

Hash function

Hash value 1

yes

Hash value 2

no

Public key cipher

Alice’s public key
Cryptographic Standards Before 1997

Secret-Key Block Ciphers
- IBM & NSA
  - 1977: DES – Data Encryption Standard
  - 1999: Triple DES

Hash Functions
- NSA
  - 1993: SHA-1 – Secure Hash Algorithm
  - 1995: SHA
  - 2003: SHA-2
  - 2005: SHA-2
Why a Contest for a Cryptographic Standard?

• Avoid **back-door** theories
• Speed-up the **acceptance** of the standard
• **Stimulate** non-classified research on methods of designing a specific cryptographic transformation
• **Focus** the effort of a relatively small cryptographic community
Cryptographic Standard Contests

1997–2017

15 block ciphers → 1 winner

34 stream ciphers → 4 HW winners + 4 SW winners

51 hash functions → 1 winner

57 authenticated ciphers → multiple winners
Cryptographic Contests - Evaluation Criteria

Security

Software Efficiency
- µProcessors
- µControllers

Hardware Efficiency
- FPGAs
- ASICs

Flexibility

Simplicity

Licensing
Specific Challenges of Evaluations in Cryptographic Contests

- Very wide range of possible applications, and as a result performance and cost targets
  - throughput: single Mbits/s to hundreds Gbits/s
  - cost: single cents to thousands of dollars
- Winner in use for the next 20-30 years, implemented using technologies not in existence today
- Large number of candidates
- Limited time for evaluation
- Only one winner and the results are final
Mitigating Circumstances

• Security is a primary criterion
• Performance of competing algorithms tend to very significantly (sometimes as much as 500 times)
• Only relatively large differences in performance matter (typically at least 20%)
• Multiple groups independently implement the same algorithms (catching mistakes, comparing best results, etc.)
• Second best may be good enough
AES Contest 1997-2000
Rules of the Contest

Each team submits

- Detailed cipher specification
- Justification of design decisions
- Tentative results of cryptanalysis
- Source code in C
- Source code in Java
- Test vectors
# AES: Candidate Algorithms

<table>
<thead>
<tr>
<th>Country</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>CAST-256, Deal</td>
</tr>
<tr>
<td>USA</td>
<td>Mars, RC6, Twofish, Safer+, HPC</td>
</tr>
<tr>
<td>Costa Rica</td>
<td>Frog</td>
</tr>
<tr>
<td>Germany</td>
<td>Magenta</td>
</tr>
<tr>
<td>Belgium</td>
<td>Rijndael</td>
</tr>
<tr>
<td>France</td>
<td>DFC</td>
</tr>
<tr>
<td>Israel, UK, Norway</td>
<td>Serpent</td>
</tr>
<tr>
<td>Korea</td>
<td>Crypton</td>
</tr>
<tr>
<td>Japan</td>
<td>E2</td>
</tr>
<tr>
<td>Australia</td>
<td>LOKI97</td>
</tr>
</tbody>
</table>
AES Contest Timeline

June 1998

**15 Candidates**
CAST-256, Crypton, Deal, DFC, E2, Frog, HPC, LOKI97, Magenta, Mars, RC6, Rijndael, Safer+, Serpent, Twofish,

Round 1

Security
Software efficiency

August 1999

**5 final candidates**
Mars, RC6, Twofish (USA)
Rijndael, Serpent (Europe)

Round 2

Security
Software efficiency
Hardware efficiency

October 2000

**1 winner:** Rijndael
Belgium
NIST Report: Security & Simplicity

- MARS
- Twofish
- Serpent
- Rijndael
- RC6
Efficiency in software: NIST-specified platform

200 MHz Pentium Pro, Borland C++

Throughput [Mbits/s]

- 128-bit key
- 192-bit key
- 256-bit key

- Rijndael
- RC6
- Twofish
- Mars
- Serpent
NIST Report: Software Efficiency
Encryption and Decryption Speed

32-bit processors
- high: RC6
- medium: Rijndael, Mars, Twofish
- low: Serpent

64-bit processors
- high: Rijndael, Twofish
- medium: Mars, RC6
- low: Serpent

DSPs
- high: Rijndael, Twofish
- medium: Mars, RC6
- low: Serpent
Efficiency in FPGAs: Speed

Xilinx Virtex XCV-1000

Throughput [Mbit/s]

- Serpent x8: 431
- Rijndael: 444
- Twofish: 414
- Serpent x1: 353
- RC6: 294
- Mars: 177
- University of Southern California
- Worcester Polytechnic Institute
- George Mason University
Efficiency in ASICs: Speed
MOSIS 0.5µm, NSA Group

Throughput [Mbit/s]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rijndael</td>
<td>606</td>
</tr>
<tr>
<td>Serpent x1</td>
<td>443</td>
</tr>
<tr>
<td>Twofish</td>
<td>202</td>
</tr>
<tr>
<td>RC6</td>
<td>202</td>
</tr>
<tr>
<td>Mars</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>57</td>
</tr>
</tbody>
</table>

- 128-bit key scheduling
- 3-in-1 (128, 192, 256 bit) key scheduling
Results for ASICs matched very well results for FPGAs, and were both very different than software.

Lessons Learned

Serpent fastest in hardware, slowest in software.
Lessons Learned

Hardware results matter!

Final round of the AES Contest, 2000

Speed in FPGAs
GMU results

<table>
<thead>
<tr>
<th>Speed [Mbit/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serpent</td>
</tr>
<tr>
<td>Rijndael</td>
</tr>
<tr>
<td>Twofish</td>
</tr>
<tr>
<td>RC6</td>
</tr>
<tr>
<td>Mars</td>
</tr>
</tbody>
</table>

Votes at the AES 3 conference

<table>
<thead>
<tr>
<th># votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rijndael</td>
</tr>
<tr>
<td>Serpent</td>
</tr>
<tr>
<td>Twofish</td>
</tr>
<tr>
<td>RC6</td>
</tr>
<tr>
<td>Mars</td>
</tr>
</tbody>
</table>
Limitations of the AES Evaluation

- Optimization for **maximum throughput**

- **Single** high-speed **architecture** per candidate

- **No use of embedded resources** of FPGAs (Block RAMs, dedicated multipliers)

- **Single FPGA family** from a single vendor: Xilinx Virtex
eSTREAM Contest 2004-2008
eSTREAM - Contest for a new stream cipher standard

PROFILE 1 (SW)

- Stream cipher suitable for software implementations optimized for high speed
- Key size - 128 bits
- Initialization vector – 64 bits or 128 bits

PROFILE 2 (HW)

- Stream cipher suitable for hardware implementations with limited memory, number of gates, or power supply
- Key size - 80 bits
- Initialization vector – 32 bits or 64 bits
# eSTREAM Contest Timeline

<table>
<thead>
<tr>
<th>Phase</th>
<th>PROFILE 1 (SW)</th>
<th>PROFILE 2 (HW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2005</td>
<td>23 Phase 1 Candidates</td>
<td>25 Phase 1 Candidates</td>
</tr>
<tr>
<td>July 2006</td>
<td>13 Phase 2 Candidates</td>
<td>20 Phase 2 Candidates</td>
</tr>
<tr>
<td>April 2007</td>
<td>8 Phase 3 Candidates</td>
<td>8 Phase 3 Candidates</td>
</tr>
<tr>
<td>May 2008</td>
<td>4 winners: HC-128, Rabbit, Salsa20, SOSEMANUK</td>
<td>4 winners: Grain v1, Mickey v2, Trivium, F-FCSR-H v2</td>
</tr>
</tbody>
</table>
Hardware Efficiency in FPGAs
Xilinx Spartan 3, GMU SASC 2007

Throughput [Mbit/s]

Area [CLB slices]
Lessons Learned

Very large differences among 8 leading candidates

~30 x in terms of area

~500 x in terms of the throughput to area ratio
SHA-3 Contest 2007-2012
NIST SHA-3 Contest - Timeline

51 candidates

Oct. 2008 → Round 1 → 14 candidates → July 2009

Round 2 → 5 candidates → Dec. 2010

Round 3 → 1 candidate → Oct. 2012
SHA-3 Round 2
Throughput vs. Area Normalized to Results for SHA-256 and Averaged over 11 FPGA Families – 256-bit variants
Throughput vs. Area Normalized to Results for SHA-512 and Averaged over 11 FPGA Families – 512-bit variants
Performance Metrics

Primary

1. Throughput

3. Throughput / Area

Secondary

2. Area

4. Hash Time for Short Messages (up to 1000 bits)
Overall Normalized Throughput: 256-bit variants of algorithms
Normalized to SHA-256, Averaged over 10 FPGA families
<table>
<thead>
<tr>
<th></th>
<th>256-bit variants</th>
<th></th>
<th>512-bit variants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Thr/Area</td>
<td>Thr</td>
<td>Area</td>
</tr>
<tr>
<td>BLAKE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CubeHash</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECHO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fugue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groestl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamsi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keccak</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luffa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shabal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skein</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SHA-3 Round 3
SHA-3 Contest Finalists
New in Round 3

- Multiple Hardware Architectures
- Effect of the Use of Embedded Resources (Block RAMs, DSP units)
- Low-Area Implementations
BLAKE-256 in Virtex 5

- $x_1$ – basic iterative architecture
- $x_k$ – unrolling by a factor of $k$
- $x_k$-PPL$n$ – unrolling by a factor of $k$ with $n$ pipeline stages
- /$k(h)$ – horizontal folding by a factor of $k$
- /$k(v)$ – vertical folding by a factor of $k$
256-bit variants in Virtex 5
512-bit variants in Virtex 5
256-bit variants in 4 high-performance FPGA families
512-bit variants in 4 high-performance FPGA families
## FPGA Evaluations

<table>
<thead>
<tr>
<th>Feature</th>
<th>AES</th>
<th>eSTREAM</th>
<th>SHA-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple FPGA families</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple architectures</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Use of embedded resources</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Primary optimization target</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput/Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experimental results</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Availability of source codes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Specialized tools</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
CAESAR Contest 2013-2017
Contest Timeline

- 2014.03.15: Deadline for first-round submissions
- 2014.04.15: Deadline for first-round software
- 2015.07.07: Announcement of second-round candidates
- 2015.12.15: Deadline for second-round Verilog/VHDL
- 2016.03.15: Announcement of third-round candidates
- 2016.12.15: Announcement of finalists
- 2017.12.15: Announcement of final portfolio
Cryptographic Standard Contests

- IX.1997: AES
- X.2000: 15 block ciphers → 1 winner
- I.2000: NESSIE
- XII.2002: CRYPTREC
- XI.2004: eSTREAM
- IV.2008: 34 stream ciphers → 4 HW winners + 4 SW winners
- X.2007: SHA-3
- X.2012: 51 hash functions → 1 winner
- XII.2017: CAESAR
- I.2013: 57 authenticated ciphers → multiple winners
Evaluation Criteria

- Security
- Software Efficiency
  - µProcessors
  - µControllers
- Hardware Efficiency
  - FPGAs
  - ASICs
- Flexibility
- Simplicity
- Licensing
Traditional Development & Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Manual Optimization

FPGA Tools

Netlist

Post Place & Route Results

Functional Verification

Timing Verification

Test Vectors
Extended Traditional Development & Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Automated Optimization

FPGA Tools

Netlist

Functional Verification

Xilinx ISE + ATHENA
Vivado + Default Strategies

Timing Verification

Test Vectors

Post Place & Route Results
Need for a Uniform Hardware API

- Software implementations compared using a uniform API, using the SUPERCOP software and eBACS framework
- 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
- No comprehensive hardware API proposed by other groups to date
- Comparison of existing and future codes highly unreliable and potentially unfair
- Need for a uniform hardware API, endorsed by the CAESAR Committee, and adopted by all future implementers
AEAD Interface

PDI
Public Data Input
Ports

SDI
Secret Data Input
Ports

DO
Data Output
Ports

AEAD

clk    rst

clk    rst

w

pdi

pdi_valid

pdi_ready

w

sdi

sdi_valid

sdi_ready

w

do

do_valid

do_ready
Typical External Circuits (1) – AXI4 IPs

AXI4-Stream Master

- m_axis_tdata
- m_axis_tvalid
- m_axis_tready

SDI FIFO

- dout
- empty
- read

clk
rst

AEAD

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

AXI4-Stream Slave

- s_axis_tdata
- s_axis_tvalid
- s_axis_tready

clk
rst

clk
rst

clk
rst

clk
rst
Typical External Circuits (2) - FIFOs

- **PDI FIFO**
  - wr_clk
  - rd_clk = clk
  - dout
  - empty
  - read

- **SDI FIFO**
  - wr_clk
  - rd_clk = clk
  - dout
  - empty
  - read

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

- **DO FIFO**
  - wr_clk = clk
  - rst
  - rd_clk
  - din
  - write
  - full
## Format of Public Data Input

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

OR

<table>
<thead>
<tr>
<th>w bits</th>
<th>instruction</th>
<th>seg_0_header</th>
<th>seg_0 = Npub</th>
<th>seg_1_header</th>
<th>seg_1 = AD_0</th>
<th>seg_2_header</th>
<th>seg_2 = AD_1</th>
<th>seg_3_header</th>
<th>seg_3 = Message_0</th>
<th>seg_4_header</th>
<th>seg_4 = Message_1</th>
</tr>
</thead>
</table>

Single segment or **multiple segments** per data type (AD and/or Message)
# Format of Data after Encryption and Decryption

## No Secret Message Number

### Before Encryption

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Segment Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTKEY</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td>ENC</td>
<td>seg_0</td>
<td>Npub</td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_2_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2</td>
<td>Message</td>
</tr>
</tbody>
</table>

(a) Before Encryption

### After Encryption / Before Decryption

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Segment Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTKEY</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td>ENC</td>
<td>seg_0</td>
<td>Npub</td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_2_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_2</td>
<td>Ciphertext</td>
</tr>
</tbody>
</table>

(b) After Encryption / Before Decryption

### After Decryption

<table>
<thead>
<tr>
<th>Status</th>
<th>Segment Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_0</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>Message</td>
</tr>
</tbody>
</table>

(c) After Decryption

<table>
<thead>
<tr>
<th>Status</th>
<th>Segment Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIL</td>
<td>seg_0_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_0</td>
<td>AD</td>
</tr>
<tr>
<td></td>
<td>seg_1_header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seg_1</td>
<td>Message</td>
</tr>
</tbody>
</table>

(d) After Decryption
Instruction and Status Word

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

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

Msg ID | Info | 000...0 | Seg Len
---|---|---|---
| 8 | 8 | \(w - (16 + s) \mod w\) | \(s\)

Segment Type:
- EOI = 1 if the last segment of input
- EOT = 1 if the last segment of its type

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 \(\lceil(16 + s)/w\rceil\) words, starting from MSB
## Format of Data after Encryption and Decryption

With Secret Message Number

<table>
<thead>
<tr>
<th>Before Encryption</th>
<th>After Encryption / Before Decryption</th>
<th>After Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instruction = ACTKEY</code></td>
<td><code>instruction = ACTKEY</code></td>
<td><code>status = PASS</code></td>
</tr>
<tr>
<td><code>instruction = ENC</code></td>
<td><code>instruction = DEC</code></td>
<td><code>seg_0_header</code></td>
</tr>
<tr>
<td><code>seg_0 = Npub</code></td>
<td><code>seg_0_header</code></td>
<td><code>seg_0 = Npub</code></td>
</tr>
<tr>
<td><code>seg_1_header</code></td>
<td><code>seg_1_header</code></td>
<td><code>seg_1 = Enc Nsec</code></td>
</tr>
<tr>
<td><code>seg_1 = Nsec</code></td>
<td><code>seg_1_header</code></td>
<td><code>seg_1 = AD</code></td>
</tr>
<tr>
<td><code>seg_2_header</code></td>
<td><code>seg_2_header</code></td>
<td><code>seg_2_header</code></td>
</tr>
<tr>
<td><code>seg_2 = AD</code></td>
<td><code>seg_2 = AD</code></td>
<td><code>seg_2 = Message</code></td>
</tr>
<tr>
<td><code>seg_3_header</code></td>
<td><code>seg_3_header</code></td>
<td><code>status = FAIL</code></td>
</tr>
<tr>
<td><code>seg_3 = Message</code></td>
<td><code>seg_3 = Ciphertext</code></td>
<td><code>After Decryption</code></td>
</tr>
</tbody>
</table>

(a) (b) (c) (d)
Format of Secret Data Input

<table>
<thead>
<tr>
<th>Instruction</th>
<th>seg_0_header</th>
<th>seg_0 = Key</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LDRKEY</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Round Keys calculated in hardware

<table>
<thead>
<tr>
<th>Instruction</th>
<th>seg_0_header</th>
<th>seg_0 = Round Key</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LDRKEY</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Round Keys calculated in software
GMU Hardware API 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)
- wide range of data port widths, $8 \leq w \leq 256$
- independent data and key inputs
- 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
GMU Hardware API 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 - a de-facto standard for the Systems-on-Chip buses
  • PCI Express – high-bandwidth serial communication between PCs and hardware accelerator boards
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 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
SIPO: Serial In Parallel Out
PISO: Parallel In Serial Out

![Diagram of PISO circuit](image-url)
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
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
  • 26 default optimization strategies for Xilinx Vivado
AEAD Core vs. CipherCore Area Overhead in Virtex 6

Overhead = \( \frac{\text{LUT}(\text{AEAD Core}) - \text{LUT}(\text{CipherCore})}{\text{LUT}(\text{AEAD Core})} \times 100\% \)
AEAD Core vs. CipherCore Area Overhead in Virtex 7

Overhead = \frac{LUT(AEAD\_Core) - LUT(CipherCore)}{LUT(AEAD\_Core)} \times 100\%
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 me regarding an account.

The ATHENa Option Optimization Tool supports automatic generation of results suitable for uploading to the database
### Ranking View (2)

**Throughput for:**
- Authenticated Encryption
- Authenticated Decryption
- Authentication Only

Min Area: 0
Max Area: 1000000
Min Throughput: 500
Max Throughput: 1000000
Source: Source Available

**Ranking:**
- Throughput/Area
- Throughput
- Area

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

[Update]

Show 25 entries

<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>68</td>
<td>ICEPOLE</td>
<td>128</td>
<td>RTL</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>73</td>
<td>Keysk</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>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>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>

Showing 1 to
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
Details of Result ID 97

**Algorithm**
- IV or Nonce Size [bits]: 96
- Transformation Category: Cryptographic
- Transformation: Authenticated Cipher
- Group: Standards
- Algorithm: AES-GCM
- Tag Size [bits]: 128
- Associated Data Support: -
- Key Size [bits]: 128
- Secret Message Number: -
- Secret Message Number Size [bits]:
- Message Block Size [bits]: 128
- Other Parameters: -
- Specification: SP-800-38D.pdf
- Formula for Message Size After Padding:

**Design**
- Design ID: 21
- Impl Approach: HLS
- Hardware API: GMU_AEAD_Core_API_V1
- Primary Optimization Target: Throughput/Area
- Secondary Optimization Target: -
- Architecture Type: Basic Iterative
- Description Language: VHDL
- Use of Megafunctions or Primitives: No
- List of Megafunctions or Primitives: -
- Maximum Number of Streams Processed in Parallel: 1
- Number of Clock Cycles per Message Block in a Long Message: 12
- Datapath Width [bits]: 128
- Padding: Yes
- Minimum Message Unit: -
- Input Bus Width [bits]: 32
- Output Bus Width [bits]: 32
## Comparison of Result #s 95 and 97

### Algorithm

<table>
<thead>
<tr>
<th>IV or Nonce Size [bits]:</th>
<th>96</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation Category:</td>
<td>Cryptographic</td>
<td>Cryptographic</td>
</tr>
<tr>
<td>Transformation:</td>
<td>Authenticated Cipher</td>
<td>Authenticated Cipher</td>
</tr>
<tr>
<td>Group:</td>
<td>Standards</td>
<td>Standards</td>
</tr>
<tr>
<td>Algorithm:</td>
<td>AES-GCM</td>
<td>AES-GCM</td>
</tr>
<tr>
<td>Tag Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Associated Data Support:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Key Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Secret Message Number:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Secret Message Number Size [bits]:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Message Block Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Other Parameters:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Specification:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Formula for Message Size After Padding</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Design

<table>
<thead>
<tr>
<th>Design ID:</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impl Approach:</td>
<td>RTL</td>
<td>HLS</td>
</tr>
<tr>
<td>Hardware API:</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>GMU_AEAD_Core_API_v1</td>
</tr>
<tr>
<td>Primary Optimization Target:</td>
<td>Throughput/Area</td>
<td>Throughput/Area</td>
</tr>
<tr>
<td>Secondary Optimization Target:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Architecture Type:</td>
<td>Basic Iterative</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>Description Language:</td>
<td>VHDL</td>
<td>VHDL</td>
</tr>
<tr>
<td>Use of Megafuntions or Primitives:</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>List of Megafuntions or Primitives:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Maximum Number of Streams Processed in Parallel:</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of Clock Cycles per Message Block in a Long Message:</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Datapath Width [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Padding:</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Minimum Message Unit:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Input Bus Width [bits]:</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td><strong>Comparison of Results #s 95 and 97</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device Vendor: Xilinx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Family: Virtex 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device: xc7vx485tfg1761-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption/Authentication Throughput [Mbits/s]: 3261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decryption/Authentication Throughput [Mbits/s]: 3261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication-Only Throughput [Mbits/s]: 3261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthesis Clock Frequency [MHz]: -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key Scheduling Time [ns]: -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requested Synthesis Clock Frequency [MHz]: -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requested Implementation Clock Frequency [MHz]: -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Clock Frequency [MHz]: 280.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Encryption/Authentication Throughput)/LUT [(Mbits/s)/LUT): 0.909</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Encryption/Authentication Throughput)/Slice [(Mbits/s)/Slice): 2.797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Decryption/Authentication Throughput)/LUT [(Mbits/s)/LUT): 0.909</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Decryption/Authentication Throughput)/Slice [(Mbits/s)/Slice): 2.797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Auth-Only Throughput)/LUT [(Mbits/s)/LUT): 0.909</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Auth-Only Throughput)/Slice [(Mbits/s)/Slice): 2.797</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resource Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLB Slices: 1166</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUTs: 3588</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flip Flops: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSPs: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRAMs: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Supporting Materials

• Design with the GMU hardware API facilitated by
  • Detailed specification
  • 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
  • Ease of recording and comparing results using ATHENa database
Expected by the end of Fall 2015

20+ RTL results
generated by 20+ ECE 545 students
C vs. VHDL: Comparing Performance of CAESAR Candidates Using High-Level Synthesis on Xilinx FPGAs

Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, and Kris Gaj
George Mason University
USA

http://cryptography.gmu.edu
https://cryptography.gmu.edu/athena
Remaining Difficulties of Hardware Benchmarking

- Large number of candidates
- Long time necessary to develop and verify RTL (Register-Transfer Level) Hardware Description Language (HDL) codes
- Multiple variants of algorithms (e.g., multiple key, nonce, and tag sizes)
- High-speed vs. lightweight algorithms
- Multiple hardware architectures
- Dependence on skills of designers
Ekawat Homsirikamol
a.k.a “Ice”

Working on the PhD Thesis entitled
“A New Approach to the Development of Cryptographic Standards Based on the Use of High-Level Synthesis Tools”
Potential Solution: High-Level Synthesis (HLS)

High Level Language (e.g. C, C++, Matlab, Cryptol) → High-Level Synthesis → Hardware Description Language (e.g., VHDL or Verilog)
Short History of High-Level Synthesis

Generation 1 (1980s-early 1990s): research period

Generation 2 (mid 1990s-early 2000s):
- Commercial tools from Synopsys, Cadence, Mentor Graphics, etc.
- Input languages: behavioral HDLs Target: ASIC

Outcome: Commercial failure

Generation 3 (from early 2000s):
- Domain oriented commercial tools: in particular for DSP
- Input languages: C, C++, C-like languages (Impulse C, Handel C, etc.), Matlab + Simulink, Bluespec
- Target: FPGA, ASIC, or both

Outcome: First success stories
Cinderella Story

AutoESL Design Technologies, Inc. (25 employees)

Flagship product:

AutoPilot, translating C/C++/System C to VHDL or Verilog

• Acquired by the biggest FPGA company, Xilinx Inc., in 2011
• AutoPilot integrated into the primary Xilinx toolset, Vivado, as Vivado HLS, released in 2012

“High-Level Synthesis for the Masses”
Our Hypothesis

- Ranking of candidate algorithms in cryptographic contests in terms of their performance in modern FPGAs will remain the same independently whether the HDL implementations are developed manually or generated automatically using High-Level Synthesis tools.

- The development time will be reduced by at least an order of magnitude.
Potential Additional Benefits

Early feedback for designers of cryptographic algorithms

- Typical design process based only on security analysis and software benchmarking
- Lack of immediate feedback on hardware performance
- Common unpleasant surprises, e.g.,
  - Mars in the AES Contest
  - BMW, ECHO, and SIMD in the SHA-3 Contest
Extended Traditional Development and Benchmarking Flow

1. Informal Specification
2. Manual Design
3. HDL Code
4. Option Optimization
5. FPGA Tools
6. Netlist
7. ATHENa
8. Functional Verification
9. Timing Verification

Post Place & Route Results

Test Vectors
Examples of Source Code Modifications

Unrolling of loops:

```c
for (i = 0; i < 4; i++)
    #pragma HLS UNROLL
    for (j = 0; j < 4; j++)
        #pragma HLS UNROLL
        b[i][j] = s[i][j];
```

Function Reuse:

```c
void KeyUpdate (word8 k[4][4],
               word8 round) {
    #pragma HLS INLINE
    ...
}
```

Flattening function's hierarchy:

```c
// (a) Before modification
for(round=0; round<NB_ROUNDS; ++round)
    {
        if (round == NB_ROUNDS-1)
            single_round(state, 1);
        else
            single_round(state, 0);
    }
```

```c
// (b) After modification
for(round=0; round<NB_ROUNDS; ++round)
    {
        if (round == NB_ROUNDS-1)
            x = 1;
        else
            x = 0;
        single_round(state, x);
    }
```
Our First Test Case

5 final SHA-3 candidates
Most efficient sequential architectures
GMU RTL VHDL codes developed during SHA-3 contest
Reference software implementations in C
   included in the submission packages

Hypotheses:

Ranking of candidates will remain the same
Performance ratios RTL/HLS similar across candidates
Manual RTL vs. HLS-based Results: Altera Stratix III
Manual RTL vs. HLS-based Results: Altera Stratix IV

RTL

HLS
Lack of Correlation for Xilinx Virtex 6

**RTL**

**HLS**
Hypothesis Check

Hypothesis I:

• Ranking of candidates in terms of throughput, area, and throughput/area ratio will remain the same
  TRUE for Altera Stratix III and Stratix IV
  FALSE for Xilinx Virtex 5 and Virtex 6

Hypothesis II:

• Performance ratios RTL/HLS similar across candidates

<table>
<thead>
<tr>
<th></th>
<th>Stratix III</th>
<th>Stratix IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>0.99-1.30</td>
<td>0.98-1.19</td>
</tr>
<tr>
<td>Area</td>
<td>0.71-1.01</td>
<td>0.68-1.02</td>
</tr>
<tr>
<td>Throughput</td>
<td>1.10-1.33</td>
<td>1.09-1.27</td>
</tr>
<tr>
<td>Throughput/Area</td>
<td>1.14-1.55</td>
<td>1.17-1.59</td>
</tr>
</tbody>
</table>
Correlation Between Altera FPGA Results and ASICS

Stratix III FPGA

ASIC

Normalized Throughput vs. Normalized Area for different algorithms on Stratix III FPGA and ASIC.
Our Second Test Case

- 8 Round 1 CAESAR candidates + current standard AES-GCM
- Basic iterative architecture
- GMU AEAD Hardware API
- Implementations developed in parallel using RTL and HLS methodology
- 2-3 RTL implementations per student, all HLS implementations developed by a single student (Ice)
- Starting point: Informal specifications and reference software implementations in C provided by the algorithm authors
- Post P&R results generated for
  - Xilinx Virtex 6 using Xilinx ISE + ATHENa, and
  - Virtex 7 and Zynq 7000 using Xilinx Vivado with 26 default option optimization strategies
- No use of BRAMs or DSP Units in AEAD Core
# Parameters of Authenticated Ciphers

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key size</th>
<th>Nonce size</th>
<th>Tag size</th>
<th>Basic Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Cipher Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-COPA</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>CLOC</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>POET</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>SCREAM</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>TLS</td>
</tr>
<tr>
<td><strong>Permutation Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICEPOLE</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>Keccak-like</td>
</tr>
<tr>
<td>Keyak</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>Keccak-f</td>
</tr>
<tr>
<td>PRIMATEs-GBBON</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>PRIMATE</td>
</tr>
<tr>
<td>PRIMATEs-HANUMAN</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>PRIMATE</td>
</tr>
</tbody>
</table>
## Parameters of Ciphers & GMU Implementations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Word Size, w</th>
<th>Block Size, b</th>
<th>#Rounds</th>
<th>Cycles/Block RTL</th>
<th>Cycles/Block HLS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block-cipher Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-COPA</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>CLOC</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>POET</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>SCREAM</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td><strong>Permutation Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICEPOLE</td>
<td>256</td>
<td>1024</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Keyak</td>
<td>128</td>
<td>1344</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>PRIMATEs-GIBBON</td>
<td>40</td>
<td>40</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>PRIMATEs-HANUMAN</td>
<td>40</td>
<td>40</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>
Datapath vs. Control Unit

Datapath

Data Inputs

Data Outputs

Control Inputs

Control Signals

Status Signals

Control Outputs

Determines

- Area
- Clock Frequency

Determines

- Number of clock cycles
Encountered Problems

Control Unit suboptimal

- Difficulty in inferring an overlap between completing the last round and reading the next input block
- One additional clock cycle used for initialization of the state at the beginning of each round
- The formulas for throughput:

  HLS: \( \text{Throughput} = \frac{\text{Block}_\text{size}}{(#\text{Rounds}+2) \times T_{\text{CLK}}} \)

  RTL: \( \text{Throughput} = \frac{\text{Block}_\text{size}}{(#\text{Rounds}+C) \times T_{\text{CLK}}} \)

  \( C=0, 1 \) depending on the algorithm
RTL vs. HLS Clock Frequency in Zynq 7000

The graph compares the clock frequency in MHz between RTL and HLS for different projects. The projects are represented by different markers and colors, including:
- PRIMATEs-HANUMAN (1,3)
- PRIMATEs-GIBBON (2,1)
- AES-GCM (3,2)
- Keyak (4,4)
- CLOC (5,5)
- ICEPOLE (6,7)
- POET (7,6)
- AES-COPA (8,8)
- SCREAM (9,9)
RTL vs. HLS Throughput in Zynq 7000

The diagram shows a comparison of throughput (Mbit/s) between RTL and HLS for various applications in Zynq 7000. The X-axis represents RTL and HLS, while the Y-axis represents throughput in Mbit/s. Different applications are represented by various markers and lines on the graph.

- Keyak (1,1)
- ICEPOLE (2,2)
- AES-GCM (3,3)
- CLOC (4,4)
- PRIMATEs-GIBBON (5,6)
- POET (6,5)
- AES-COPA (7,7)
- SCREAM (8,8)
- PRIMATEs-HANUMAN (9,9)
RTL vs. HLS Ratios in Zynq 7000

Clock Frequency

Throughput
RTL vs. HLS #LUTs in Zynq 7000
RTL vs. HLS Throughput/#LUTs in Zynq 7000
RTL vs. HLS Ratios in Zynq 7000

#LUTs

Throughput/#LUTs
Throughput vs. LUTs in Zynq 7000
RTL vs. HLS Throughput

![Graph showing throughput comparison between RTL and HLS for Virtex 6, Virtex 7, and Zynq X-V6, X-V7, Zynq with throughputs for various algorithms: ICEPOLE, Keyak, AES-GCM, CLOC, OCB, POET, AES-COPA, Scream, PRIMATEs GIBBON, PRIMATEs HANUMAN, and Joltik.]
RTL vs. HLS #LUTs

![Graph showing comparison of RTL vs. HLS #LUTs for Virtex 6, Virtex 7, and Zynq.]
RTL vs. HLS Throughput/#LUTs

![Graph showing throughput and area comparison between RTL and HLS for various algorithms on Virtex 6, Virtex 7, and Zynq.](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>X-V6</th>
<th>X-V7</th>
<th>Zynq</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEPOLE</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>Keyak</td>
<td>(2,2)</td>
<td>(2,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>(3,3)</td>
<td>(3,3)</td>
<td>(3,3)</td>
</tr>
<tr>
<td>CLOC</td>
<td>(4,4)</td>
<td>(5,5)</td>
<td>(6,5)</td>
</tr>
<tr>
<td>PRIMATES GIBBON</td>
<td>(5,5)</td>
<td>(4,4)</td>
<td>(4,4)</td>
</tr>
<tr>
<td>OCB</td>
<td>(6,7)</td>
<td>(6,7)</td>
<td>(5,7)</td>
</tr>
<tr>
<td>PRIMATES HANUMAN</td>
<td>(7,6)</td>
<td>(7,6)</td>
<td>(7,6)</td>
</tr>
<tr>
<td>SCREAM</td>
<td>(8,8)</td>
<td>(8,8)</td>
<td>(9,8)</td>
</tr>
<tr>
<td>Joltik</td>
<td>(9,10)</td>
<td>(9,9)</td>
<td>(8,9)</td>
</tr>
<tr>
<td>POET</td>
<td>(10,9)</td>
<td>(10,10)</td>
<td>(10,10)</td>
</tr>
<tr>
<td>AES-COPA</td>
<td>(11,11)</td>
<td>(11,11)</td>
<td>(11,11)</td>
</tr>
</tbody>
</table>
Throughput vs. LUTs in Virtex 6
Throughput vs. LUTs in Virtex 7

RTL

HLS
Throughput vs. LUTs in Zynq 7000

RTL

HLS
Implementation of CAESAR Round 1 Candidates

- 19 Round 2 CASER candidates to be implemented manually in VHDL as a part of ECE 545 in Fall 2015. One cipher per student.

- One PhD student, Ice, will implement the same 19 ciphers in parallel using HLS.

- Preliminary results in mid-December 2015.

Expected by the end of Fall 2015

19 RTL results
generated by 19 ECE 545 students

19 HLS results
generated by “Ice” alone
Questions?  Suggestions?

ATHENa:  http:/cryptography.gmu.edu/athena
CERG:  http://cryptography.gmu.edu