CLOC, SILC and OTR

Kazuhiko Minematsu (NEC Corporation)
Outline

• Describe AE schemes, CLOC, SILC and OTR
  – Merged as “CLOC and SILC” for CAESAR
  – Both are CAESAR third-round candidates
  – Both are blockcipher modes with provable security proofs

• Topics:
  – Motivation
  – Design rationale
  – Idea of security proof
  – Implementations etc.
CLOC and SILC
CLOC and SILC

• CLOC (Compact Low-overhead CFB)
  • presented at FSE 2014 [IMGM14]
    – Designers:
      – Tetsu Iwata (Nagoya University),
      – Jian Guo (Nanyang Technological University),
      – Sumio Morioka (Interstellar technologies), and myself

• SILC (SImple Lightweight CFB)
  • presented at DIAC 2014 [IMGMK14]
    – Designers: CLOC designers + Eita Kobayashi (NEC)

The story of CLOC

• In 2011, ANSI defined a new AE scheme called EAX’ (EAX-prime)
  – for their standard ANSI-C12.22 defined for Smartgrid
• Based on EAX [BRW04], ANSI tried to optimize it in terms of precomputation and memory
  – Suitable for constrained devices
• ANSI pushed EAX-prime to NIST, and NIST requested public comments for inclusion it into NIST SP-800 series

The story of CLOC

• While EAX comes with provably security results (reduction to blockcipher security), EAX-prime did not
• In fact, EAX-prime was seriously broken [MLMI13]
  – Single-query forgery etc.
• Still the original motivation of EAX-prime seems valuable anyway
  – Constrained devices, blockcipher-based, design simplicity, small footprint
• Let’s do it in a right way!

Predecessors: CCM, EAX, and EAX-Prime

- **CCM** (NIST SP 800-38C)
  - not online
- **EAX** (ISO/IEC 19772)
  - Simple design, reusing CMAC
  - precomputation cost \((L = E_K(0), E_K(1), \text{ and } E_K(2))\) may be a problem for highly constrained devices
    - Time and memory
- **EAX-prime** (ANSI C12.22)
  - reduced precomputation \((L = E_K(0))\) from EAX
  - efficiently handles short input data with small memory
  - practical attacks
CLOC’s design goal

• Provably secure AEAD based on a blockcipher
  – Standard security notions for privacy and authenticity
• Primary focus:
  – design simplicity
  – the precomputation complexity
  – the memory requirement

• Efficient for short input data, say up to 64 bytes
• Suitable for small microprocessors
  – Small word size and number of registers
  – High-cost for RAM access
Short Input Data

- Performance for short input data matters:
  - Low-power sensor networks
    - Zigbee: at most 127 bytes
    - Bluetooth Low Energy: at most 47 bytes
    - Electronic Product Code (EPC): typically 96 bits
- For long input data, the efficiency of CLOC is the same as CCM, EAX, and EAX-prime
  - 2 blockcipher calls per 1 plaintext block
CLOC Properties

• Nonce-based AEAD
• uses only the encryption of the blockcipher both for encryption and decryption
• When $|A| \geq 1$, it makes $|N|_n + |A|_n + 2|M|_n$ blockcipher calls for a nonce $N$, associated data $A$, and a plaintext $M$
  – where $|X|$ is the length of $X$ in bits and $|X|_n$ is the length in $n$-bit blocks
  – $1 \leq |N| \leq n-1$, so $|N|_n = 1$
  – No precomputation beyond the blockcipher key schedule
  – When $|A| = 0$, it needs $|N|_n + 1 + 2|M|_n$ calls
• It works with two state blocks (i.e. $2n$ bits)
• Sequential
CLOC Properties

• For short input data
  – 1-block nonce, 1-block associated data, and 1-block plaintext
  – CCM: 5 or 6 calls
  – EAX: 7 calls (where 3 out of 7 can be precomputed)
  – EAX-prime: 5 calls (where 1 out of 5 can be precomputed)
  – CLOC: 4 calls
Comparison with other modes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Calls</td>
<td>$a + 2m + 2$</td>
<td>$m + 1$</td>
<td>$a + 2m + 1$</td>
<td>$a + 2m + 1$</td>
<td>$a + m + 1$</td>
<td>$a + 2m + 1$</td>
</tr>
<tr>
<td>Setup</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>On-line</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static AD</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Primitive</td>
<td>E, E, GHASH</td>
<td>E</td>
<td>E</td>
<td>E, D</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

PRIVATE/AUTH: $O(2^{n/2})[25]$ | $O(2^{n/2})[24]$ | $O(2^{n/2})[13]$ | $O(1)[30]$ | $O(2^{n/2})[26]$ | $O(2^{n/2})$

NON-PRIVATE/AUTH: $2^{n/2}[20,19]$ | $O(1)[19]$ | $O(1)[19]$ | $O(1)[30]$ | $O(1)[19]$ | $O(2^{n/2})$

(from [IMM14])

Overview of the Scheme

- Encrypt-then-PRF paradigm
- uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part
Tools

• The one-zero padding function: ozp
  – For $0 \leq |X| \leq n$
  – $ozp(X) = X$ if $|X|=n$, and $ozp(X) = X||10\ldots0$ otherwise

• The tweak functions: $f_1$, $f_2$, $g_1$, $g_2$, and $h$
  – use them to directly update the state
  – Word-based linear functions

• The bit fixing functions: $fix0$ and $fix1$
  – $fix0(X)$: fix $msb_1(X)$ to 0
  – $fix1(X)$: fix $msb_1(X)$ to 1
    • $fix1(0000) = 1000$, $fix1(1100) = 1100$
\[ V \leftarrow \text{HASH}_K(A,N) \]

- A variant of CBC MAC
- \(1 \leq |N| \leq n-1\)
V <- HASH\_K(A,N)

- A variant of CBC MAC
- 1 ≤ |N| ≤ n–1
V <- HASH_{K}(A,N)

- A variant of CBC MAC
- $1 \leq |N| \leq n-1$
V <- HASH\textsubscript{K}(A,N)

- A variant of CBC MAC
- \(1 \leq |N| \leq n-1\)

\[
\text{if } |A[a]| = n, \text{ then } f_1, \text{ else } f_2
\]

\[
\text{if } \text{msb}_1(A[1]) = 1, \text{ then } h, \text{ else } i
\]
C <- ENC_K(V,M)

- A variant of CFB mode
$T \leftarrow PRF_K(V, C)$

- A variant of CBC MAC
\( T \leftarrow \text{PRF}_K(V,C) \)

- A variant of CBC MAC

- \( g_1 \) is used when \(|C|=0\)
Rationale

• The bit fixing functions
  – used to logically separate CBC MAC and CFB mode
  – otherwise, attacks are possible
Rationale

• The tweak functions
  – There are 55 differential probability constraints
    • $K \oplus f_1(K), f_1(K) \oplus g_1(f_1(h(K))), \ldots$
  – Each term should be close to uniform when $K$ is uniform
  – optimality result: any lack of single constraint would lead to attack [KMI15]

Rationale

• Constant multiplications over GF(2^n) can work
  – $2X = X$ multiplied by the generator of the field, called doubling [R04]
  – $3X = 2X + X$ and so on
  – $2X$ needs 1-bit shift and conditional XOR of constant
• But we want to avoid bit-level functions (for embedded processors )

[R04] Rogaway: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. ASIACRYPT 2004
Rationale

• Instead, we define a matrix $M$ as

\[
M = (K[2], K[3], K[4], K[1] \oplus K[2])
\]

- $K \cdot M = (K[1], K[2], K[3], K[4]) \cdot M$
  
  \[
  = (K[2], K[3], K[4], K[1] \oplus K[2])
  \]

• We specify tweak functions as

  \[
  f_1: M^{i_1}, f_2: M^{i_2}, g_1: M^{i_3}, g_2: M^{i_4}, h: M^{i_5}
  \]
  
  With $(i_1, i_2, i_3, i_4, i_5) = (8, 1, 2, 1, 4)$

  - Computer-aided search for secure and efficient ones
Works with Two State Blocks
Security

• Privacy: standard Nonce-based AE (NAE) privacy notion
  – Indistinguishability of ciphertexts from random bits against nonce-respecting adversaries in a chosen plaintext attack setting

\[
\text{Adv}_{\text{CLOC}[E, \ell_N, \tau]}^{\text{priv}}(A) \overset{\text{def}}{=} \Pr \left[ A^{\text{CLOC-}\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1 \right] - \Pr \left[ A^{\mathcal{S}(\cdot,\cdot,\cdot)} \Rightarrow 1 \right]
\]

\[
\text{Adv}_{\text{CLOC}[\text{Perm}(n), \ell_N, \tau]}^{\text{priv}}(A) \leq \frac{5\sigma_{\text{priv}}^2}{2^n}, \text{ where } \sigma_{\text{priv}} = q + \sigma_A + 2\sigma_M
\]
Security

• Authenticity:
  – Unforgeability against nonce-reusing adversaries in a chosen ciphertext attack setting
  – A stronger adversary than standard one for NAE

\[ \text{Adv}_{\text{CLOC}[E,\ell_N,\tau]}^{\text{auth}}(\mathcal{A}) \overset{\text{def}}{=} \Pr \left[ \mathcal{A}^{\text{CLOC}-\mathcal{E}_K(\cdot,\cdot,\cdot),\text{CLOC}-\mathcal{D}_K(\cdot,\cdot,\cdot)} \text{ forges} \right] \]

\[ \text{Adv}_{\text{CLOC}[\text{Perm}(n),\ell_N,\tau]}^{\text{auth}}(\mathcal{A}) \leq \frac{5\sigma_{\text{auth}}^2}{2^n} + \frac{q}{2^\tau}, \]

where \( \sigma_{\text{auth}} = q + \sigma_A + 2\sigma_M + q' + \sigma_A' + \sigma_C' \)
Software Implementation

• Embedded software
• Atmel AVR ATmega128
  – 8-bit microprocessor
  – AES from [AVR-Crypto-Lib] written in assembler
    • 156.7 cpb for encryption, 196.8 cpb for decryption
  – CLOC, EAX, and OCB3
    • modes are written in C
    • OCB3 code from official cite [OCB] w/ small modification
      – doubling operations are on-line, large precomputation may not be suitable to handle short input data for microprocessors
  – compiled with Atmel Studio 6

Software Implementation

<table>
<thead>
<tr>
<th></th>
<th>ROM (bytes)</th>
<th>RAM (bytes)</th>
<th>Init (cycles)</th>
<th>Speed (cycles/byte)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data 16</td>
<td>32</td>
<td>64</td>
<td>96</td>
<td>128</td>
</tr>
<tr>
<td>CLOC</td>
<td>2980</td>
<td>362</td>
<td>1999</td>
<td></td>
<td>750.1</td>
<td>549.0</td>
<td>448.4</td>
<td>414.9</td>
<td>398.2</td>
</tr>
<tr>
<td>EAX</td>
<td>2772</td>
<td>402</td>
<td>12996</td>
<td></td>
<td>913.6</td>
<td>632.5</td>
<td>490.8</td>
<td>443.6</td>
<td>419.9</td>
</tr>
<tr>
<td>OCB-E</td>
<td>5010</td>
<td>971</td>
<td>4956</td>
<td></td>
<td>1217.5</td>
<td>736.1</td>
<td>495.5</td>
<td>412.2</td>
<td>375.1</td>
</tr>
<tr>
<td>OCB-D</td>
<td>5010</td>
<td>971</td>
<td>4956</td>
<td></td>
<td>1252.2</td>
<td>773.4</td>
<td>534.0</td>
<td>451.2</td>
<td>414.3</td>
</tr>
</tbody>
</table>

• 1-block AD, no static AD computation
• cycle counting is obtained by the simulation of Atmel Studio 6
• RAM is measured with a public tool [EZSTACK]
• In CLOC, the RAM usage is low and Init is fast, and it is fast for short input data, up to around 128 bytes
Software Implementation

• Performance on Intel processor, Core i5-3427U 1.80GHz (Ivy Bridge family)
• AES-128, using AES-NI
• CLOC: about 4.9 cpb for long input data (more than $2^{20}$ blocks)
• AES calls in CFB mode and CBC MAC (in tag generation) can be done in parallel
Software Implementation

- General purpose CPU
- Intel processor, Core i5-3427U 1.80GHz (Ivy Bridge family)
- AES-128, AES-NI
- CLOC: about 4.9 cpb for long input data (more than $2^{20}$ blocks)
  - AES runs in 4.3 cpb
Software Implementation

• AES calls in CFB mode and CBC MAC (in tag generation) can be done in parallel
Software Implementation

- AES calls in CFB mode and CBC MAC (in tag generation) can be done in parallel
Software Implementation

- AES calls in CFB mode and CBC MAC (in tag generation) can be done in parallel

Latest performance at public benchmark (SUPERCOP by D. Bernstein)

Intel Core i5-6600 (Skylake):
- 2.82 C/B for long message, 7.81 C/B for 64-byte message
CAESAR version (v3)

- When n=128, AES, 96-bit nonce
- When n=64, TWINE [SMMK12], 48-bit nonce
  - 64-bit blockcipher, thus small security margin ($2^{32}$ blocks ~ 32 Gbytes per key)
  - Not for generic-purpose (e.g. Internet), but
  - may be acceptable for low-powered, small-bandwidth, limited-lifetime communications
- Parameter encoding into nonce block (as param||N)
- Please refer to the latest CAESAR document

---

SILC

- A variant of CLOC with more hardware focus
  - CLOC is focusing on embedded software
- Mostly the same security goal and performance features as CLOC

- For CAESAR: AES and two lightweight 64-bit blockciphers
  - Present [BKL+07] (ISO 29192) for speed
  - LED [GPPR11] for high-security margin

Design Strategy

- CLOC uses five tweaking functions which requires non-negligible number of logic gates for Hw
- SILC reduces the use of tweaking functions
  - even simpler than CLOC,
  - at the cost of the constant number of increase of blockcipher calls
  - Other properties of CLOC hold as well
SILC Properties

- Nonce-based AEAD
- uses only the encryption of the blockcipher both for encryption and decryption
- It makes $|N|_n + |A|_n + 2|M|_n + 2$ blockcipher calls for a nonce $N$, associated data $A$, and a plaintext $M$
  - where $|X|_n$ is the length of $X$ in $n$-bit blocks
  - $1 \leq |N| \leq n - 1$, so $|N|_n = 1$
  - blockcipher key scheduling can be precomputed
  - No precomputation beyond that (blockcipher calls, generation of key dependent tables, . . . ) is needed
- Works with two state memories
SILC (v3)
Differences between CLOC and SILC

• Reduce the required number of tweaking functions
  – only one function ("g") instead of five
  – Fix1 still needed
• Zero prepending/appending instead of 10* padding
• Length encoding into inputs
  – Simple at the cost of one additional BC call (for both A and M)
OTR
What is OTR?

- OTR: a blockcipher mode of operation for Nonce-based AE (NAE) [M14]
  - Based on OCB, removing the need for blockcipher decryption function needed by OCB ("inverse-free")
- AES-OTR for CAESAR

Features:
- Rate-1 (needs one AES call for one block)
- Parallelizable for encryption/decryption
- Inverse-free

- Unique AES-based NAE CAESAR candidate achieving all of them

Basics: How to build AE?

- Generic composition
- Nonce-based Encryption + MAC (message authentication code) basically works
- If we focus on blockcipher (BC)-based schemes, an example is CTR encryption + CMAC, using two keys
- Security analyzed [BN00][K00][NRS14]
- Limitation: rate is 2 (two rate-1 functions)
  - rate = # of BC calls per input block
  - (or if we define “rate=1/# of BC calls per input block”, in this case rate = 1/2)

[K00] H. Krawczyk: The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). CRYPTO 2001
Can we go further?

- Rate-1 AE by integration of Enc and MAC
- Many early attempts broken (~’90)
- Provably-secure modes appeared since 2001
  - IACBC, IAPM [J01], X CBC [GD01]
  - OCB [RBB03] [R04][KR11]

[Ro04] Rogaway : Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. ASIACRYPT 2004
Structure of OCB (w/o AD)

- **Enc** = ECB mode with tweakable BC (TBC) [LRW02]
  - TBC = BC taking tweaks, \((N,1), (N,2), \ldots\)
  - Realized by BC w/ I/O masks (called XE mode [R04])
  - Mask \(g(*)\) : a function of Nonce, block index, and key
- **MAC** = Plaintext checksum (XOR) encryption

\[
\begin{align*}
\text{M[1]} & \quad g(N,1) & \quad \text{E}_K & \quad \text{C[1]} \\
\text{M[2]} & \quad g(N,2) & \quad \text{E}_K & \quad \text{C[2]} \\
& \quad \vdots & \quad \vdots & \quad \vdots \\
\text{M[m]} & \quad g(N,m) & \quad \text{E}_K & \quad \text{C[m]} \\
\text{Checksum} & \quad g(N,l') & \quad \text{E}_K & \quad \text{msb} \\
& \quad \text{Tag} & \quad \tau \text{ bits} \\
\end{align*}
\]

Checksum = \(M[1] \oplus M[2] \ldots \oplus M[m]\)

OCB

• Many good properties
  – Rate-1
    • mask generation can be done with few BC calls (usually one)
  – Parallelizable (for E & D)
  – On-line
    • operation can start w/o knowing the input length
  – Provably secure if BC is a strong pseudorandom permutation (SPRP)*

• So, can’t we go further?

*[AY13] showed a relaxation from SPRP

Existence of Blockcipher Inverse

- One potential disadvantage of OCB: the existence of BC inverse (decryption function)
  - Popular rate-2 modes use only the forward (encryption) function of BC, i.e. inverse-free
- Undesirable in some cases
  - Increased size (Sw, Hw)
  - BC inverse may be slower than forward (or vice versa)
    - E.g. Byte-wise Sw AES on microcontrollers
      - Stronger security assumption (SPRP rather than PRP/PRF)

- **Can we remove BC inverse?**
Using Feistel rounds

- Substituting n-bit TBC with 2n-bit balanced Feistel permutation
  - Round function = n-bit TBC built from n-bit BC
    - forward function, with input mask
    - Tweak consists of Nonce, block index, and round index
- How many rounds are needed?
Using Feistel rounds (Contd.)

- 4 rounds are sufficient, as it is 2n-bit SPRP (Luby-Rackoff), but rate-2, no gain
- To keep rate-1, we have to use 2 rounds

2-R is not even PRP, so we cannot directly follow the proof of OCB
2-round AE construction

- OTR uses 2n-bit 2-R Feistel permutation instead of OCB’s n-bit TBC
- n-bit checksum needs to be defined (later)
- Inverse-free, rate-1
2-round AE skeleton

- We can safely assume internal TBCs are independent random functions indexed by tweak
  - if masks are properly chosen (differentially uniform [LRW02])
- The scheme is called 2-R AE skeleton
- We analyze PRIV and AUTH of 2-R AE skeleton
Privacy of 2-round AE skeleton

- Each $C[i]$ contains an output of RF invoked only once (as Nonce is unique)
- Ciphertext and tag are uniformly random
- **PRIV bound is zero**

![Diagram of 2-round AE skeleton](image)
Authenticity of 2-round AE skeleton

- Now checksum is defined as a sum of **even** plaintext blocks
- Consider simple attack using one encryption query and one decryption query
- Forgery is successful iff $T^*$ (true tag for dec query) = $T'$ (fake tag)
- Suppose $(C[1], C[2])$ was changed to $(C'[1], C'[2])$ and $N$ was not changed

```
Encryption Query (N,M)→(C,T)
F(N,1,1)  F(N,1,1')
```

```
Decryption Query (N,C',T')→M' or ⊥
F(N,1,1)  F(N,1,1')
C'[1] C'[2]
```

If $T^* = T'$ the forgery is successful
Authenticity of 2-R AE skeleton (Contd.)

- Case $C'[1] \neq C[1]$
- Then the first round input ($Z'$) is random -> $M'[2]$ is random, **unless the collision between $Z$ and $Z'$**
- If $M'[2]$ is random, then checksum is random -> $T^*$ is random, **unless the checksum collision**
- Two collision events of prob. $1/2^n$
- If $T^*$ is random, the chance of guessing $T^*$ is $1/2^\tau$, for $\tau$-bit $T^*$
- -> **AUTH bound is $2/2^n + 1/2^\tau$**

![Encryption Query](image)

**Encryption Query**
$(N,M)\rightarrow(C,T)$

![Decryption Query](image)

**Decryption Query**
$(N,C',T')\rightarrow M'$ or ⊥
Authenticity of 2-R AE skeleton (Contd.)

- AUTH is bounded by $2/2^n + 1/2^\tau$, for single dec query.
  - The bound for multiple dec queries is derived using [BGM04].
- 2-R Feistel actually works.

Full figure of OTR

- Doubling-based masking
  - XE mode [R04], turning BC into TBC
  - An issue in the original spec shown by [BS16], fixed now

**Encryption part**

Encryption

when $m$ is even

- [R04] Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. Asiacrypt 2004
- [BS16] Bost and Sanders. Trick or Tweak: On the (In)security of OTR’s Tweaks. To appear at Asiacrypt 2016
Full figure of OTR

- Tag computation: TE and TA
- TE = encryption of check sum
- TA = Output of PMAC-based PRF taking AD (shown here)
  - Additionally, CMAC-based PRF for specific cases

---

[R04] Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. Asiacrypt 2004
[BS16] Bost and Sanders. Trick or Tweak: On the (In)security of OTR’s Tweaks. To appear at Asiacrypt 2016
Concrete security bounds

• Privacy and Authenticity bounds based on perfect permutation
• Computational security based on the PRP assumption

\[
\text{Adv}^\text{priv}_{\text{OTR}[P,\tau]}(A) \leq \frac{6\sigma^2_{\text{priv}}}{2^n}
\]

\[
\text{Adv}^\text{auth}_{\text{OTR}[P,\tau]}(A) \leq \frac{6\sigma^2_{\text{auth}}}{2^n} + \frac{q_v}{2^\tau}
\]

\(\tau\): tag bit length
\(q\): # of encryption queries
\(q_v\): # of decryption queries
\(\sigma_{\text{priv}}\): # of the total queried blocks in encryption queries \((q+\sigma_A + \sigma_M)\)
\(\sigma_{\text{auth}}\): # of the total queried blocks in encryption and decryption queries \((q+q_v + \sigma_A + \sigma_M + \sigma_{A'} + \sigma_{C'})\)
Security limitations

• As in standard NAEs:
  • Nonce must be unique for encryptions
  • No protection against nonce-misuse and decryption-misuse
    – If needed use outer protection such as [FJMV03]

Performance of OTR : AESNI

- Intel/AMD CPUs with AES instructions (AESNI)
- Using intrinsic
- Software pipeline (i.e. way to efficiently compute AES in parallel) as optimized OCB implementation
- Batch GF-doubling optimization [A13][MSK15]

- With all efforts...
- Result on Skylake processor (Intel Core i5-6600 ) with AES-128
  - 0.68 cycles/bytes for long message at SUPERCOP
    - Pretty small gap from OCB (0.64 cycles/byte)

Performance of OTR : ARMv7

• Platform: Beaglebone Black (Cortex-A8 1GHz), with gcc 4.7.3
• Combining single-block T-table AES with Bitslice AES available from SUPERCOP (originally proposed by Kasper-Schwabe [KS09])
  – Single-block for Nonce/tag encryption
  – Bitslice processes 8 blocks in parallel
    • Only AES-Encryption code available, which is sufficient for OTR
• Use NEON SIMD engine
• Use intrinsic

Results

- Peak speed: ~23.5 c/b (+7% of AESBS)
- For reference, in Gouveay-Lopez's GCM runs 32.8 c/b, using BS-AES, on Cortex A9 [GL15]
- A fast single-block AES on ARMv7 would contribute short-input performance
  - E.g. vector-permutation [H09]

AES-OTR (AES-128, no AD)

<table>
<thead>
<tr>
<th>Msglen (byte)</th>
<th>Enc (median c/b)</th>
<th>AESBS ratio</th>
<th>Dec (median c/b)</th>
<th>AESBS ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1056</td>
<td>25.42</td>
<td>1.14</td>
<td>25.42</td>
<td>1.14</td>
</tr>
<tr>
<td>2080</td>
<td>24.19</td>
<td>1.07</td>
<td>24.2</td>
<td>1.07</td>
</tr>
<tr>
<td>16416</td>
<td>23.5</td>
<td>1.07</td>
<td>23.51</td>
<td>1.07</td>
</tr>
</tbody>
</table>

AES Bit-slice

<table>
<thead>
<tr>
<th>Msglen (byte)</th>
<th>Enc (med c/b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1056</td>
<td>22.24</td>
</tr>
<tr>
<td>2080</td>
<td>22.58</td>
</tr>
<tr>
<td>16416</td>
<td>21.87</td>
</tr>
</tbody>
</table>

AES T-table

<table>
<thead>
<tr>
<th>Msglen (byte)</th>
<th>Enc (med c/b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1056</td>
<td>41.31</td>
</tr>
<tr>
<td>2080</td>
<td>43.56</td>
</tr>
<tr>
<td>16416</td>
<td>43.54</td>
</tr>
</tbody>
</table>

Summary

• CLOC, SILC and OTR explained
  – CLOC and SILC: lightweight, suitable for constrained devices
  – OTR: high-performance for various platforms

• Blockcipher-based AE much advanced since the announcement of CAESAR
Summary

• CLOC, SILC and OTR explained
  – CLOC and SILC: lightweight, suitable for constrained devices
  – OTR: high-performance for various platforms

• Blockcipher-based AE much advanced since the announcement of CAESAR

Thank you!