Efficient Authenticated Encryption

Jonas Bruun Jacobsen

DTU Compute

18/12-2015

Outline

Jonas Bruun Jacobsen

Background:

- Bachelor: BSc Software Development from the IT University of Copenhagen
- Work: Software developer/consultant (employer: Immeo).

Interests:

- Privacy: Secure handling of private information.
- Cryptology: Understanding why and when something is secure (and when it is not!).
- Programming: Efficient designs and implementations.

Thesis Topic

Efficient Authenticated Ciphers

Results sneak peek

Managed to increase PRIMATEs by a factor 13-31x

Why AE?

Problems with current authentication:

- Authentication is not an integral part of the most currently used ciphers.
- Is added to them as a mode of operation.
- Combining the cipher and the mode can be an error-prone process and difficult. These issues has lead to actual attacks, e.g. on SSL/TLS.
- Incompatibility between some modes and ciphers (e.g. CCM and GCM are only defined for 128-bit block ciphers).

Undergoing Research

CAESAR competition:
Competition for Authenticated Encryption: Security, Applicability, and Robustness

PRIMATEs

PRIMATEs:

- CAESAR second-round candidate.
- Designed by: Elena Andreeva, Begl Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang and Kan Yasuda.
- A suite of ciphers:
  - APE
  - HANUMAN
  - GIBBON
- All 3 use the same 80- or 120-bit permutation.
- Designed with lightweight cryptology in mind.

	APE-s	HANUMAN-s	GIBBON-s
k (key size)	2s	s	s
t (tag size)	2s	s	s
n (nonce size)	s	s	s
PRIMATE	p_1	p_1, p_4	p_1, p_2, p_3

PRIMATEs

Permutation:

- 4 transformations:
  - Constant Addition (CA)
  - Mix Columns (MC)
  - Shift Rows (SR)
  - Sub Elements (SE)
- Based on security-level, these are applied to a state with:
  - 8×5 PRIMATEs elements.
  - 8×7 PRIMATEs elements.
  - (NB. 1 PRIMATEs element $=$ 5 bits).
- 6 rounds (GIBBON) or 12 rounds (HANUMAN, APE).

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$\boxed{a_{1,1}}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$

PRIMATEs 80-bit state with the CA element highlighted. Figure is from PRIMATEs' CAESAR submission paper.

Sub elements:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S(x)	1	0	25	26	17	29	21	27	20	5	4	23	14	18	2	28

x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S(x)	15	8	6	3	13	7	24	16	30	9	31	10	22	12	11	19

Shift rows:

- 80-bit: Rows are shifted 0,1,2,4,7 to the left
- 120-bit Rows are shifted 0,1,2,3,4,5,7 to the left.

Constant Addition:
Different constant for each round and for each cipher. Easily generated in hardware with a LFBR (Linear feedback-register) and initial value.
Mix columns:
A simple matrix is multiplied to the state 5 (80-bit) or 7 (120-bit) times.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 18 & 2 & 2 & 18 \end{bmatrix}$$

My task: Implement these ciphers efficiently for speed. (Why?)

How to design/implement it efficiently?

Bit slicing!

- Fastest AES (without AES-NI) and DES implementations.
- Simultanously also shows that PRIMATEs can be implemented to resist timing attacks.
- SIMD are a popular way of making efficient implementations.
- Other contestants in CAESAR also uses bit slicing (Ascon) or in other ways utilize CPU registers heavily (NORX).

What is bit slicing?

P_0	5	4	3	2	1	0
P_1:	5	4	3	2	1	0
R_0	0	0				
R_1	1	1				
R_2	2	2				
R_3	3	3				
R_4	4	4				

Finding a bit sliced state

Finding an optimized bit sliced state.

First idea for a bit sliced state: 8 elements per row, thus each row can be condensed into a byte:

State *n*																					Byte
Row 0			Row 1			Row 2			Row 3			Row 4			Row 5			Row 6			Free
Col 0	...	Col 7	Col 0	...	Col 7	Col 0	...	Col 7	Col 0	...	Col 7	Col 0	...	Col 7	Col 0	...	Col 7	Col 0	...	Col 7	Free

Bitsliced SE

Similar work already done by Ko Stoffelen.[1] Unfortunately, he never found a bit sliced S-box for PRIMATEs.

Went the simple way:

- Constructed ANF formulas from the S-Box.
- Manually nudged the circuits for effectiveness.

[1] https://ko.stoffelen.nl/papers/fse2016-sboxoptimization.pdf

$$y0 = 1 + x0 + x0x2 + x3 + x1x4$$
$$y1 = x0x1 + x2x3 + x4 + x0x4 + x2x4$$
$$y2 = x0x2 + x1x2 + x3 + x4 + x0x4 + x3x4$$
$$y3 = x1 + x0x2 + x1x2 + x1x3 + x2x3 + x4$$
$$y4 = x1 + x2 + x1x2 + x3 + x0x3 + x1x4 + x2x4$$

Figure: The ANF formulas used to create the PRIMATEs forward S-box for bit slicing

Bitsliced CA & SR

Constant addition:
Simple. Just find the matching register bits and XOR with the round constant bits.

Shift rows:
Intuitive and efficient. Unpack each register byte into 16 bits (there is an intrinsic for this). Multiply with shifting constants. Result is in upper 8 bits.

Shift rows example

Example:

R_0	7	6	5	4	3	2	1	0
R_1	1	1	2	2	3	3	4	4

Reg0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	1	1	2	2	3	3	4	4	1	1	2	2	3	3	4	4

_mm256_mullo_epi16(Reg0, 4, 32)

Reg0	5	4	3	2	1	0	7	6	5	4	3	2	1	0	0	0
	3	4	4	1	1	2	2	3	3	4	4	0	0	0	0	0

Bitsliced MC

Never found a “satisfying” one - why?

New State

New state:

- Avoids the MC problem with the old state
- Shift rows is trivial now (can be done in 1 cycle).
- Inspired by the bit sliced state Emilia Käsper and Peter Schwabe used for AES.[2]

Row 0										...	Row 4									
Col 0			Col 1			...	Col 7			...	Col 0			Col 1			...	Col 7		
State 0	...	State 7	State 0	...	State 7	...	State 0	...	State 7	...	State 0	...	State 7	State 0	...	State 7	...	State 0	...	State 7

[2] Paper: Faster and Timing-Attack Resistant AES-GCM

SE, SR & CA with new state

Sub elements:
The same. (Why?)

Constant Addition:
The only change is where in the register, the element is located.

Shift Rows:
Each byte contains a single element (from all states). Shift rows can be done with byte-shuffling now.

MC with new state

Mix Columns:
Normal matrix multiplication can easily be done now.

Since we apply the matrix 5 or 7 times, we technically perform the following:

$$new\_state = state \times A1 \times A1 \times A1 \times A1 \times A1 \times A1 \times A1$$

We found a way to speed this up.

We can multiply the matrices beforehand, e.g.

$$A3 = A1 \times A1 \times A1$$

and instead calculate (as an example):

$$new\_state = state \times A3 \times A3 \times A1$$

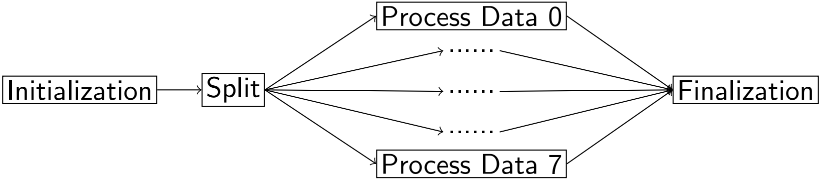
New vs old bit sliced state

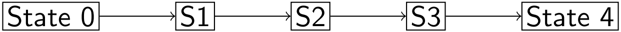
Pros/cons regarding the new state:

- **Good**: More efficient for the 120-bit security level. Few empty bits (32 bits out of 256 bits).
- **Good**: Few changes needs to be made between the 80-bit and 120-bit transformations.
- Good: Simpler transformations.
- **Bad**: Cannot be contained in 5x 256-bit registers (The AVX2 register length).
- **Bad**: Higher amount of bits not used for the 80-bit security level (96 bits out of 256 bits. Old state only had 16 empty).
- **Bad**: Can only increment the amount of parallel states with sets of 8.

New schemes

Problem when using the bit sliced permutation:
Bit slicing attains speed from parallelism, the current schemes are by nature sequential.





New schemes

New schemes:

- Names:
  - GIBBON-BS
  - HANUMAN-BS
  - APE-BS
- Supports parallel processing of data from 8 states.
- Loads 40 bytes of data at a time and spreads it out into the 8 parallel states.
- After processing the data, the states are combined (XORed) and a tag is created from a single state, identical to how the non-bit sliced schemes do it.

Tests - Setup & Environment

Environment:

- Model: Lenovo 20BV001BMD (Thinkpad T450)
- CPU: Intel Core i7-5600U @ 2.60 GHz (4 CPUs)
- RAM: 8GB
- OS: Windows 10 Pro 64-bit (10.0, Build 10586)

Setup:

- Intel Turboboost turned off (so consistent CPU frequency).
- All tests were ran on a single core.
- All code compiled with ICC (with /O3 and /QxHost optimization flags).
- **No** changes made to the reference code.
- Input data consists of only zeroes.
- Speed is calculated by reading the TSC register before and after encryption/decryption.

Tests - Results

Scheme	Level	CPB (reference)	CPB (bit sliced)	Improvement
APE	80	1263/1602	93/103	13.5x - 15.5x
	120	**4739**/2749	151/162	17x-31.5x
HANUMAN	80	1246/1247	93/89	13.5x-14x
	120	**4763**/**4749**	152/152	31x-31.5x
GIBBON	80	637/636	46/46	14x
	120	**2416**/**2418**	76/76	2416
Permutation	80	617/796	~44/50	14x-16x
	120	**2375**/1370	74/80	17x-32x

Table: CPB: Cycles per byte. The values listed are for encryption/decryption and are rounded. Test-data is 4MB. The permutation results are over 6 rounds, and is the average of all permutations (p_1 to p_4) and all their inverses. Bold results could likely easily be improved.

Comparison to other ciphers

Ascon:

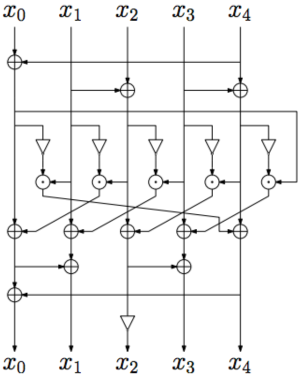
- Best results: 10.5 CPB.

Norx:

- Best results: 1.99 CPB.

PRIMATEs (GIBBON80-BS):

- Best results: 46 CPB. (43?)


x_0 x_1 x_2 x_3 x_4
x_0 x_1 x_2 x_3 x_4

Achievements:

- Managed to bit slice the PRIMATEs permutation.
- Managed to design a bit sliced variant of all existing schemes.
- Managed to show that PRIMATEs can be implemented resistant to timing attacks and be efficient.
- Managed to achieve competitive speeds for PRIMATEs on high-end hardware (46 CPB for GIBBON80-BS).