ICEPOLE v2: High-speed, Hardware-oriented Authenticated Encryption Scheme

Paweł Morawiecki¹  Kris Gaj³  Ekawat Homsirikamol³
Krystian Matusiewicz⁶  Josef Pieprzyk¹,²  Marcin Rogawski⁵
Marian Srebrny¹  Marcin Wójcik⁴

Institute of Computer Science, Polish Academy of Sciences, Poland ¹
Queensland University of Technology, Brisbane, Australia ²
Cryptographic Engineering Research Group, George Mason University, USA ³
Cryptography and Information Security Group, University of Bristol, United Kingdom ⁴
Cadence Design Systems, San Jose, USA ⁵
Intel, Gdańsk, Poland ⁶

DIAC 2015, Singapore
1. Brief description of ICEPOLE
2. Tweak for 2nd round
3. Third-party cryptanalysis
4. Hardware performance
ICEPOLE General Overview

- based on the variant of duplex framework introduced by Bertoni et al. "Duplexing the sponge: (...)” Cryptology ePrint archive 2011/499
- high-speed hardware-oriented ICEPOLE permutation is the heart of our design
- family of authenticated encryption schemes with three parameters: key, nonce and secret message number
- primary recommendation: ICEPOLE-128: 128-bit key and 128-bit nonce
Encryption and Tag Generation - Overview
Decryption and Tag Verification

- The same permutations used for encryption and decryption
ICEPOLE Internal State Organization

- 1280-bit internal state $S$
- can be viewed as two-dimensional array $S[4][5]$, where each element of array is a 64-bit word
ICEPOLE Round and $P_6$, $P_{12}$ Permutations

\[ R = \kappa \circ \psi \circ \pi \circ \rho \circ \mu \]

**ICEPOLE Permutations**

- $P_6$: 6-round permutation, used in Processing Phase
- $P_{12}$: 12-round permutation, used in Initialization and Tag Generation
μ Step

GF(2^5) multiplication modulo \(x^5 + x^2 + 1\)
- easy to implement (just XOR operations)
- main source of diffusion in the algorithm

\[
\begin{pmatrix}
2 & 1 & 1 & 1 \\
1 & 1 & 18 & 2 \\
1 & 2 & 1 & 18 \\
1 & 18 & 2 & 1 \\
\end{pmatrix}
\begin{pmatrix}
Z_0 \\
Z_1 \\
Z_2 \\
Z_3 \\
\end{pmatrix}
= 
\begin{pmatrix}
2Z_0 + Z_1 + Z_2 + Z_3 \\
Z_0 + Z_1 + 18Z_2 + 2Z_3 \\
Z_0 + 2Z_1 + Z_2 + 18Z_3 \\
Z_0 + 18Z_1 + 2Z_2 + Z_3 \\
\end{pmatrix}
\]
$S[x][y] := S[x][y] \ll offsets[x][y]$ for all $(0 \leq x \leq 3), (0 \leq y \leq 4)$

- each word has a distinct offset value
- $\rho$ introduced to mix information between ‘slices’ of the state
\( \pi \) Step

\[ R = \kappa \pi \rho \mu \]

Each step updates the state as follows.

**\( \mu \) Step:**

In the \( \mu \) step bits are mixed through the MDS (Maximum Distance Separable) matrix. Every 20-bit slice is mixed through the matrix given below. Formally, a column vector \( (Z_0, Z_1, Z_2, Z_3) \) is multiplied by a constant matrix producing a vector of four 5-bit words.

\[
\begin{bmatrix}
0 & B & B \\
@ & 2 & 1 \\
1 & 1 & 8 \\
1 & 2 & 1 \\
1 & 1 & 8 \\
1 & 8 & 2 \\
1 & 1 & 8 \\
1 & 8 & 2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
Z_0 \\
Z_1 \\
Z_2 \\
Z_3 \\
\end{bmatrix} =
\begin{bmatrix}
0 & B & B \\
@ & 2 & 1 \\
1 & 1 & 8 \\
1 & 2 & 1 \\
1 & 1 & 8 \\
1 & 8 & 2 \\
1 & 1 & 8 \\
1 & 8 & 2 \\
\end{bmatrix}
\begin{bmatrix}
Z_0 + Z_1 + 18 Z_2 + 2 Z_3 \\
Z_0 + 2 Z_1 + 1 Z_2 + 18 Z_3 \\
Z_0 + 1 Z_1 + 8 Z_2 + 2 Z_3 \\
Z_0 + 8 Z_1 + 2 Z_2 + Z_3 \\
\end{bmatrix}
\]

The operations are done in \( \mathbb{GF}(2^5) \). Here the multiplication is defined as the multiplication of binary polynomials modulo the irreducible polynomial \( x^5 + x^2 + 1 \). There are only three distinct terms in the chosen matrix, namely 18, 2, 1 and they correspond to the polynomials \( x^4 + x \), \( x \), and 1, respectively. The \( \mu \) step can be efficiently implemented with simple bitwise equations (see Appendix H).

**\( \rho \) Step:**

The \( \rho \) step is the bitwise rotation applied to each of the twenty 64-bit words of the state. The bitwise rotation moves bit at position \( z \) into position \((z + r \text{ value}) \mod 64\). For each word \( r \text{ value} \) is different.

\[
S[x'][y'] \leftarrow \pi(S[x][y])
\]

**\( \pi \) reorders the words in the state \( S \)

introduced to provide more mixing between words

\[
x' := (x + y) \mod 4
\]
\[
y' := (((x + y) \mod 4) + y + 1) \mod 5
\]
Step

ICEPOLE S-box

- The S-box maps a 5-bit input vector \((M_0, ..., M_4)\) to a 5-bit output vector \((Z_0, ..., Z_4)\)
- inspired by the Keccak S-box
- the only non-linear step in ICEPOLE

for all \((0 \leq k \leq 4)\)

\[
Z_k = M_k \oplus (\neg M_{k+1}M_{k+2}) \oplus (M_0M_1M_2M_3M_4) \oplus (\neg M_0 \neg M_1 \neg M_2 \neg M_3 \neg M_4)
\]
In the 64-bit constant is xored with $S[0][0]$.

$S[0][0] := S[0][0] \oplus \text{constant[numberOfRound]}$

**Round Constants**
- each round with a distinct constant
- introduced to break similarities between rounds
- The constants are calculated as the output of a simple 64-bit maximum-cycle Linear Feedback Shift Register (LFSR).
In Tag Generation, now we use 12-round permutation rather than 6-round.

This change introduces a solid security margin against the ciphertext forgery. It was shown the forgery can be mounted for 4 rounds [Dobraunig, Eichlseder, Mendel; FSE 2015].

As ICEPOLE aims at high data processing rates, a few more rounds in the very last call of the permutation basically does not affect performance of the algorithm.
ICEPOLE requires a nonce

In case of nonce reuse, some level of intermediate robustness provided by secret message number and associated data (if distinct)

In case of violating all nonce-like mechanisms (nonce reused, secret message number reused, the same associated data), security claims do not hold [Huang, Wu, Tjuawinata, FSE 2015]
Huang, Tjuawinata, Wu: **Differential-Linear Cryptanalysis of ICEPOLE.** FSE 2015
(When nonce, secret message number and associated data are reused, ICEPOLE can be broken with differential-linear cryptanalysis. If nonce requirement is respected, ICEPOLE is secure)

Dobraunig, Eichlseder, Mendel: **Forgery Attacks on round-reduced ICEPOLE-128.** FSE 2015
(When Tag Generation is reduced to 4 rounds, ciphertext forgery can be mounted by means of differential cryptanalysis)
Huang, Tjuawinata, Wu: **Differential-Linear Cryptanalysis of ICEPOLE.** FSE 2015
(When nonce, secret message number and associated data are reused, ICEPOLE can be broken with differential-linear cryptanalysis. If nonce requirement is respected, ICEPOLE is secure)

Dobraunig, Eichlseder, Mendel: **Forgery Attacks on round-reduced ICEPOLE-128.** FSE 2015
(When Tag Generation is reduced to 4 rounds, ciphertext forgery can be mounted by means of differential cryptanalysis)
Dobraunig, Eichlseder, Mendel: **Heuristic Tool for Linear Cryptanalysis with Applications to CAESAR Candidates**
AsiaCrypt 2015
(Linear trails were provided for 5-round ICEPOLE-256a with bias $2^{-90}$, and for 4-round ICEPOLE-128 with bias $2^{-44}$)
FPGA Implementation Results

Xilinx Virtex-6
- Throughput: 37432 Mbps
- Area: 6052 LUTs
- Throughput/Area: 6.185 Mbps/LUT

Xilinx Virtex-7
- Throughput: 39665 Mbps
- Area: 5746 LUTs
- Throughput/Area: 6.90 Mbps/LUT

Xilinx Zynq-7000
- Throughput: 34020 Mbps
- Area: 5753 LUTs
- Throughput/Area: 5.91 Mbps/LUT

Source: George Mason University, CAESAR Benchmarking
https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view
FPGA Implementation - Throughput

Throughput comparison of various FPGA implementations:

- **AES-GCM**
  - Virtex-6: 2 Gb/s
  - Virtex-7: 5 Gb/s
  - Zynq-7000: ~1 Gb/s

- **Keyak**
  - Virtex-6: 25 Gb/s
  - Virtex-7: 35 Gb/s
  - Zynq-7000: ~20 Gb/s

- **ICEPOLE**
  - Virtex-6: 35 Gb/s
  - Virtex-7: 40 Gb/s
  - Zynq-7000: ~30 Gb/s
FPGA Implementation - Throughput/Area

- AES-GCM
- Keyak
- ICEPOLE

[Mbps/slice]

- Virtex-6
- Virtex-7
- Zynq-7000
Cyril Arnould: Towards Developing ASIC and FPGA Architectures of High-Throughput CAESAR Candidates, Master’s Thesis, ETHZ, Zurich
supervised by Michael Mühlberghuber and Frank K. Gürkaynak

ASIC implementations (tape-out included!) of a few CAESAR algorithms (AEZ, Prost, AES-GCM, ICEPOLE, Tiaoxin-346, Silver). The authors aimed at 100 Gbit/s architectures.
ICEPOLE is roughly 3 times as small and area efficient as comparable implementation of AES-GCM.

ICEPOLE is the only candidate (out of those 5) to achieve over 50 GBit/s when processing maximum sized Ethernet packets.

Author concluded that ICEPOLE is the best algorithm in terms of ”high throughput suitability”.
Conclusion

- monkeyDuplex construction + very efficient permutation = ICEPOLE
- highly efficient in modern FPGAs
- excellent choice for high performance platforms, backbone networks
- secure algorithm, already with a decent amount of cryptanalysis
Thank you!

Questions?

Questions?

Questions?

Questions?