ICEPOLE v2

Paweł Morawiecki[1,2], Kris Gaj[5], Ekawat Homsirikamol[5], Krystian Matusiewicz[8], Josef Pieprzyk[1,3,4], Marcin Rogawski[7], Marian Srebrny[1,2], and Marcin Wójcik[6]

[1] Institute of Computer Science, Polish Academy of Sciences, Poland

[2] Section of Informatics, University of Commerce, Kielce, Poland

[3] Department of Computing, Macquarie University, Australia

[4] Electrical Engineering and Computer Science School, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia

[5] Cryptographic Engineering Research Group, George Mason University, USA

[6] Cryptography and Information Security Group, University of Bristol, United Kingdom

[7] Cadence Design Systems, San Jose, USA

[8] Intel, Gdańsk, Poland

Contact email: pawel.morawiecki@gmail.com

August 24, 2015

This project was financed by National Science Centre, Poland, registration number DEC-2014/15/B/ST6/05130

Table of Contents

1 Specification

1.1 Parameters

ICEPOLE is a family of authenticated ciphers with three parameters: key length, secret message number length, nonce length. The key length is either 128 bits or 256 bits. The secret message number length is between 0 and 128 bits. The nonce length is between 0 and 128 bits.

1.2 Recommended Parameter Set

Our primary recommended parameter set is: 128-bit key, 128-bit secret message number, 128-bit nonce. The ICEPOLE variant with these recommended parameter values is called ICEPOLE-128. We also define two ICEPOLE variants serving as drop-in replacements for AES-128-GCM and AES-256-GCM. These variants are ICEPOLE-128a (128-bit key, 0-bit secret message number, 96-bit nonce) and ICEPOLE-256a (256-bit key, 0-bit secret message number, 96-bit nonce).

The following specification refers to the primary recommendation ICEPOLE-128. A specification of ICEPOLE-128a and ICEPOLE-256a is nearly the same and the differences are given at the end of this section.

1.3 State Organization and Notations

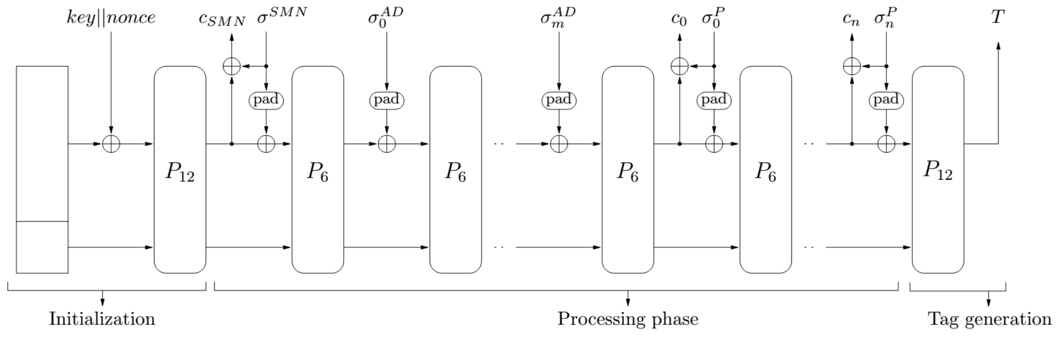
The algorithm works on the 1280-bit state S. The state S is organized as the two-dimensional array $S[4][5]$ where each element of the array is a 64-bit word. The little-endian convention is used throughout the document. When we refer to the particular bit, we introduce the third index: $S[x][y][z]$. The mapping between the bits of vector v and those of $S[x][y][z]$ is $v[64(x+4y)+z] = S[x][y][z]$. If the bits of the state share the same z coordinates, they form a *slice*. As z ranges from 0 to 63, there are 64 slices in the state. If the bits of the state share the same x and z coordinates, they form a *row*. It is also convenient to introduce a notation which allows referring to the first n bits of the state. Let $S_{\lfloor n \rfloor}$ denotes the first n bits of the state, namely those bits $S[x][y][z]$ for which $64(x + 4y) + z < n$.

We use the following notation: $\oplus$ (bitwise XOR), $\cdot$ (bitwise AND), $\neg$ (negation).

1.4 Scheme Overview

ICEPOLE-128 encrypts and authenticates a message with a 128-bit key and a 128-bit nonce. There are 3 phases of the algorithm as shown in Figure 1.

Fig. 1. General scheme of ICEPOLE encryption and authentication.



At the heart of ICEPOLE there is the 1280-bit permutation denoted by P. Let us first describe this permutation.

1.5 Permutation P

P is an iterated permutation and a number of rounds is a parameter of the permutation. In the presented algorithm the 6- and 12-round variants (denoted by P_6 and P_{12}) are used. Each round R consists of five steps labelled by the Greek letters: μ (mu), ρ (rho), π (pi), ψ (psi), κ (kappa).

$$R = \kappa \circ \psi \circ \pi \circ \rho \circ \mu$$

Each step updates the state as follows.

μ:

In the μ 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.

$$\begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 1 & 18 & 2 \\ 1 & 2 & 1 & 18 \\ 1 & 18 & 2 & 1 \end{pmatrix} \begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix} = \begin{pmatrix} 2Z_0 + Z_1 + Z_2 + Z_3 \\ Z_0 + Z_1 + 18Z_2 + 2Z_3 \\ Z_0 + 2Z_1 + Z_2 + 18Z_3 \\ Z_0 + 18Z_1 + 2Z_2 + Z_3 \end{pmatrix}$$

The operations are done in $GF(2^5)$. 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 μ step can be efficiently implemented with simple bitwise equations (see Appendix B).

ρ:

The ρ 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 $(z + r_{value})$ modulo 64. For each word r_{value} is different.

$$S[x][y] := S[x][y] \lll \text{offsets}[x][y] \qquad \text{for all } (0 \le x \le 3), (0 \le y \le 4)$$

The rotation offsets are as follows.

offsets[0][0] := 0	offsets[0][1] := 36	offsets[0][2] := 3	offsets[0][3] := 41
offsets[0][4] := 18	offsets[1][0] := 1	offsets[1][1] := 44	offsets[1][2] := 10
offsets[1][3] := 45	offsets[1][4] := 2	offsets[2][0] := 62	offsets[2][1] := 6
offsets[2][2] := 43	offsets[2][3] := 15	offsets[2][4] := 61	offsets[3][0] := 28
offsets[3][1] := 55	offsets[3][2] := 25	offsets[3][3] := 21	offsets[3][4] := 56

π:

π reorders the words in the state. Words are moved from $S[x][y]$ to $S[x'][y']$ and the new coordinates (x', y') are calculated from the following simple formula.

$$x' := (x + y) \; mod \; 4$$
$$y' := (((x + y) \; mod \; 4) + y + 1) \; mod \; 5$$

ψ:

In the ψ step the ICEPOLE S-box is applied to each of 256 rows of the state. The S-box maps a 5-bit input vector $(M_0, M_1, ..., M_4)$ to a 5-bit output vector $(Z_0, Z_1, ..., Z_4)$. The S-box functionality can be easily described by the following bitwise equation. Operations on the index k are done modulo 5. The bitwise AND operator $\cdot$ is omitted for clarity.

$$\begin{array}{l} \text{for all } (0 \leq k \leq 4) \\ Z_k = M_k \oplus (\neg M_{k+1} M_{k+2}) \oplus (M_0 M_1 M_2 M_3 M_4) \oplus (\neg M_0 \neg M_1 \neg M_2 \neg M_3 \neg M_4) \end{array}$$

κ:

In κ the 64-bit constant is xored with $S[0][0]$.

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

The constant values are taken as the output of a simple 64-bit maximum-cycle Linear Feedback Shift Register (LFSR). The polynomial representation of LFSR is $x^{64} + x^{63} + x^{61} + x^{60} + 1$. The LFSR state is initialized with the 64-bit vector '0123456789ABCDEF' (hexadecimal format) and then each cycle generates a subsequent constant. For implementors convenience the constant values are given in Appendix A.

1.6 Initialization Phase

First, the state is initialized with the 1280-bit pseudorandom constant. The constant was obtained by applying the Keccak-f[1600] permutation (an underlying permutation of the SHA-3 standard) to the all-zero vector and truncating the result to 1280 bits. The state is initialized as follows. The values are given in hexadecimal using the little-endian format.

$$\begin{array}{ll} S[0][0] := FF97A42D7F8E6FD4 & S[0][1] := 90FEE5A0A44647C4 \\ S[0][2] := 8C5BDA0CD6192E76 & S[0][3] := AD30A6F71B19059C \\ S[0][4] := 30935AB7D08FFC64 & S[1][0] := EB5AA93F2317D635 \\ S[1][1] := A9A6E6260D712103 & S[1][2] := 81A57C16DBCF555F \\ S[1][3] := 43B831CD0347C826 & S[1][4] := 01F22F1A11A5569F \\ S[2][0] := 05E5635A21D9AE61 & S[2][1] := 64BEFEF28CC970F2 \\ S[2][2] := 613670957BC46611 & S[2][3] := B87C5A554FD00ECB \\ S[2][4] := 8C3EE88A1CCF32C8 & S[3][0] := 940C7922AE3A2614 \\ S[3][1] := 1841F924A2C509E4 & S[3][2] := 16F53526E70465C2 \\ S[3][3] := 75F644E97F30A13B & S[3][4] := EAF1FF7B5CECA249 \end{array}$$

Once the state is filled with the constant, the 128-bit key K and the 128-bit *nonce* are introduced into the state. K_0 and K_1 denote two 64-bit words of the key, $nonce_0$ and $nonce_1$ denote two 64-bit words of the nonce.

$$\begin{array}{l} S[0][0] := S[0][0] \oplus K_0 \\ S[1][0] := S[1][0] \oplus K_1 \\ S[2][0] := S[2][0] \oplus nonce_0 \\ S[3][0] := S[3][0] \oplus nonce_1 \end{array}$$

Then, the P_{12} permutation is run on the state S.

$$S := P_{12}(S)$$

1.7 Processing Phase

The input data is processed in blocks. First, a single secret message number block σ^{SMN} is processed, then associated data blocks σ_i^{AD} and next plaintext blocks σ_i^P. σ^{SMN} and σ_i^P are authenticated and encrypted whereas the associated data blocks σ_i^{AD} are only authenticated.

The σ^{SMN} block length is 128 bits. σ_i^{AD} and σ_i^P block length has to be between 0 (the empty block) and 1024 bits. Each block is padded to be 1026 bits long and the padding rules are as follows. First, every block is appended with the *frame bit.* The frame bit is set to 1 for the last σ^{AD} block and all σ_i^P except the last one. For all other blocks the frame bit is set to 0. Once the frame bit is appended, a given block is padded according to the following simple rule: append 1 concatenated by enough 0's so that the block length is 1026 bits. Thus the padded block has at least two padding bits (the frame bit and 1) and maximally 1026 padding bits (in case of the empty block).

In the processing phase the ciphertext blocks c_i are produced and the state is updated.

```
$c_{SMN} = S_{\lfloor 128 \rfloor} \oplus \sigma^{SMN}$
$\sigma^{SMN} := pad(\sigma^{SMN})$
$S_{\lfloor 1026 \rfloor} := S_{\lfloor 1026 \rfloor} \oplus \sigma^{SMN}$

for all blocks $\sigma_i^{AD}$ {
    $S := P_6(S)$
    $\sigma_i^{AD} := pad(\sigma_i^{AD})$
    $S_{\lfloor 1026 \rfloor} := S_{\lfloor 1026 \rfloor} \oplus \sigma_i^{AD}$
}

for all blocks $\sigma_i^P$ {
    $S := P_6(S)$
    $c_i = S_{\lfloor l \rfloor} \oplus \sigma_i^P$ ($l$ is a length of $\sigma_i^P$)
    $\sigma_i^P := pad(\sigma_i^P)$
    $S_{\lfloor 1026 \rfloor} := S_{\lfloor 1026 \rfloor} \oplus \sigma_i^P$
}
```

1.8 Tag Generation

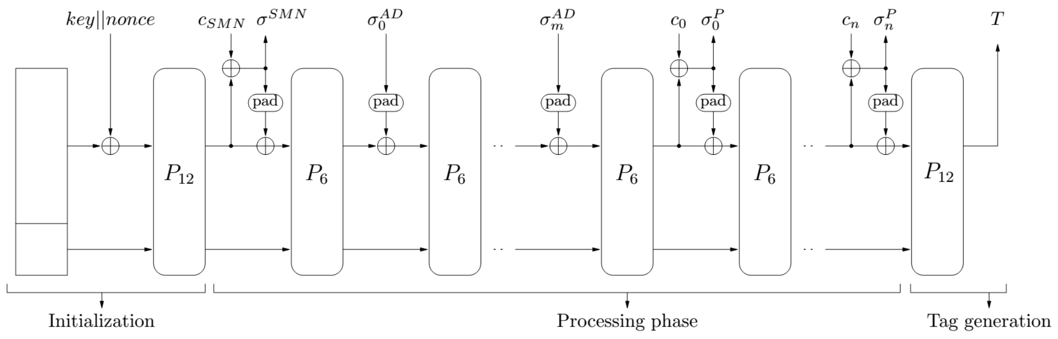
When the blocks processing is finished, the P_{12} permutation is run on the state and the 128-bit authentication tag T is derived. (T_0 and T_1 denote two 64-bit words of T.)

```
$S := P_{12}(S)$
$T_0 := S[0][0]$
$T_1 := S[1][0]$
```

1.9 Decryption and Verification

Decryption and verification are done basically with the same scheme as for encryption. The only difference is that now the input data are the ciphertext blocks and the associated data blocks. Figure 2 shows the scheme. We stress that the same permutation P (and not its inverse) is used for decryption and verification. Once the processing phase is finished, the tag T is generated and compared to the tag received from the sender. If the tags match, the data is authenticated.

Fig. 2. ICEPOLE decryption



1.10 ICEPOLE-128a

We specify ICEPOLE-128a to have a drop-in replacement for AES-128-GCM run with most common parameters, namely a 96-bit nonce and a 128-bit tag.

The only differences between ICEPOLE-128 (specified above) and ICEPOLE-128a is that in ICEPOLE-128a there is not a secret message number (the σ^{SMN} block is empty) and a nonce is 96 bits long. The nonce is padded with 32 zeros and introduced into the state in the same way as for ICEPOLE-128.

1.11 ICEPOLE-256a

We specify ICEPOLE-256a to have a drop-in replacement for AES-256-GCM run with most common parameters, namely a 96-bit nonce and a 128-bit tag.

ICEPOLE-256a encrypts data with a 256-bit key, a 96-bit nonce and the data is authenticated with a 128-bit tag. There is not a secret message number and a 96-bit nonce is padded with 32 zeros. A 256-bit key consists of four 64-bit words $K_0 \dots K_3$ and the padded nonce consists of two 64-bit words. The key and the nonce are introduced into the state as follows.

$$S[0][0] := S[0][0] \oplus K_0$$
$$S[1][0] := S[1][0] \oplus K_1$$
$$S[2][0] := S[2][0] \oplus K_2$$
$$S[3][0] := S[3][0] \oplus K_3$$
$$S[0][1] := S[0][1] \oplus nonce_0$$
$$S[1][1] := S[1][1] \oplus nonce_1$$

The associated data blocks σ_i^{AD} and the plaintext blocks σ_i^P have the length between 0 and 960 bits. The padded blocks are 962 bits long. For ICEPOLE-256a the number of blocks encrypted under a single key should be less than 2^{62}. All other parameters and steps of the specification are the same as for ICEPOLE-128.

2 Security Goals

In Table 1 we quantify the intended number of bits of security (i.e., the logarithm base 2 of the attack cost) in each of the given categories.

A public message number is a nonce. For ICEPOLE-128a and ICEPOLE-256a there is no secret message number. For 128-bit key ICEPOLE variants the number of blocks encrypted under a single key should less than 2^{126}. For 256-bit key variants the limit is 2^{62} blocks.

Table 1. The intended number of bits of security

Goal	ICEPOLE-128	ICEPOLE-128a	ICEPOLE-256a
confidentiality for the plaintext	128	128	256
confidentiality for the secret message number	128	128	256
integrity for the plaintext	128	128	128
integrity for the associated data	128	128	128
integrity for the secret message number	128	-	-
integrity for the public message number	128	128	128

A user of ICEPOLE is required to use a nonce. However, in the case of nonce reuse ICEPOLE provides some intermediate level of robustness. When the secret message number is present and respected, namely each message has the corresponding, unique secret message number, the key-recovery attack should not be possible.

3 Security Analysis

In 2010, Bertoni et al. introduced the duplex construction [6], which provides the framework for an authenticated encryption scheme. ICEPOLE is based on this construction and thus ICEPOLE general security claims are inherited from it. The monkey duplex construction can be seen as a particular way of using the sponge construction [5]. Similarly, there are two parameters, namely r (bitrate) and c (capacity). The sum of these two parameters makes the state size. Different values for bitrate and capacity give trade-offs between speed and security; a higher bitrate gives a faster construction at the expense of a lower security. For ICEPOLE-128, r is 1026 bits and c is 254 bits. In case of ICEPOLE variants with a 256-bit key, r is 962 bits and c is 318 bits.

In [2], it was proved that the sponge construction is secure against generic attacks with complexity below $2^{c/2}$. However, when a sponge or duplex object is used in conjunction with a secret key, one can prove more refined bounds taking into account the data complexity. In [3] Bertoni et al. proved that if the data complexity is limited to 2^a r-bit blocks, the keyed mode withstands generic attacks with time complexity up to 2^{c-a} calls of the underlying permutation. If $a < c/2$, this results in an increase of the security level from $c/2$ to $c-a$. This comes in handy particularly for 256-key ICEPOLE variants, where we would like to keep 256 bits of security without expanding the state or introducing any serious changes in the algorithm's specification. By limiting the number of blocks (encrypted under the same key) to 2^{62}, ICEPOLE-256 stands up to any attack up to $2^{318-62} = 2^{256}$ (unless easier generically). For ICEPOLE-128 the limit (rather purely theoretical) is 2^{126} blocks and hence the security level is $254 - 126 = 128$ bits.

We claim that the security level can be proven under the assumption that the underlying permutation P has not any exploitable properties (there are no structural distinguishers of the permutation). Therefore the security analysis of ICEPOLE comes down to analysis of the permutation P. Below we give our cryptanalysis indicating that P is indeed a secure permutation.

3.1 Differential Cryptanalysis

Differential cryptanalysis, introduced by Biham and Shamir [7], has become very powerful technique of modern cryptanalysis. One of the most convincing way of showing the resistance against differential attacks is to provide a lower bound on the weight of any differential characteristics (also called differential trails or paths) over a number of rounds. For example, in the AES the structure of the cipher and its diffusion properties allow to provide such bounds analytically [9]. However for 'bit-oriented constructions (e.g., Keccak or MD6 hash function) it is not possible to derive differential characteristics bounds in a very straightforward and convenient manner.

In such cases computer-aided proofs are provided and for ICEPOLE we take this approach.

Computer Aided Proof

A brute-force strategy to check all possible characteristics (even for a very small number of rounds) fails. The 1280-bit state is too big, even when exploiting all possible symmetries. Instead of the plain brute-force we used a SAT-solver. A SAT solver is an algorithm, which decides whether a given propositional (Boolean) formula (typically described in the Conjunctive Normal Form) has a satisfying valuation. Generally, to solve a problem instance: (1) the instance is translated to SAT (in such a way that a satisfying valuation represents a solution to the problem); (2) we run a favourite SAT solver to find a solution.

First, we focused on the problem of finding the 3-round characteristic with the minimum number of active S-boxes. The problem was encoded as a SAT formula in the Conjunctive Normal Form with the aid of the CryptLogVer toolkit [15]. CryptoMiniSat2 [23] is able to solve it in a few hours on a desktop PC. The solution, that is the minimum number of active S-boxes for 3 rounds, is 9. Then, we tried to repeat the experiments for 4 rounds, but the problem was too hard for the SAT-solver. However, if we slightly change the problem and ask the solver about a particular number (up to 13) of active S-boxes on the 4-round differential path, the answer is provided by the solver. For 4 rounds there are no paths with 13 or fewer S-boxes. Again, checking a higher number of S-boxes turned out to be infeasible.

If a number of active S-boxes is at least 14 (for 4 rounds) and the highest probability of a difference transition through the S-box is 2^{-2} (deduced from the difference distribution table of the S-box, given in Appendix C), then the lowest weight for 4 rounds is $2^{-2*14} = 2^{-28}$. So for 12 rounds a weight equals $2^{-28*3} = 2^{-84}$ and hence data complexity for the attack is 2^{84} plaintexts. Please note that this is already a much bigger number than the limitation (2^{62}) for ICEPOLE-256 on a number of blocks of plaintexts encrypted under the same key.

We believe that the complexity of differential attack should be much higher than the lower bound of 2^{84} plaintexts. The first reason for that is the difference transitions through the S-box with the weight 2^{-2} happen very rarely. Out of 337 possible difference transitions only 10 (3%) has the weight 2^{-2}. Most of the transitions (216) has the weight 2^{-4} and the average weight is $2^{3.4}$. The second argument strongly indicating that ICEPOLE is resistant to a differential attack is our experimental results. These gives some more insight into the difference propagation in ICEPOLE.

Differential Path Search

We were inspired by the work of Duc et al. [13] where the algorithm for differential path search was given for the Keccak permutation. They managed to provide the best differential paths for the round-reduced variants of the permutation. Our permutation shares some key features with the Keccak permutation (in particular how the state is organized and 'bit-oriented' propagation) and hence we think that a similar algorithm may be fruitful also for our analysis.

The goal of the algorithm is to derive differential paths by maintaining Hamming weight of bit differences as low as possible. We note that μ, ρ, π are all linear mappings (denoted altogether by λ, while ψ acts as the non-linear S-box. κ (adding round constants) does not affect differential analysis in any way. Furthermore, ρ and π do not change the number of active bits in a differential path, but change only bit positions. Hence, μ and ψ are critical when analysing differential paths. Since ψ is followed by μ in the next round (ignoring κ), we consider these two mappings together by treating a slice of the state as a unit, and try to find the potential best mapping of the slice through ψ with the following rule.

- Given an input difference of the slice, find all possible output differences by looking into the S-box differential profile. Then, among all combinations of possible output differences, choose a combination which would give the state the minimum Hamming weight after an application of μ.

It is not possible to check all possible states as the starting points for a differential path because the state is too big, even if we take advantage of symmetries. We limit the space of starting points to the states with a single active bit. For the 20-bit slice there are 20 such cases. For our permutation (as in the case of the Keccak permutation) a differential path is invariant through position rotation along the z axis so choosing a particular slice does not matter.

We start our search from b_1 point, i.e., the state after the linear mappings (denoted by λ) in the second round, and compute backwards for one round, and a few rounds forwards, as shown below.

$$\begin{tikzcd} a_0 & b_0 \arrow[l, "\lambda^{-1}"'] & a_1 \arrow[l, "\psi^{-1}"'] & \mathbf{b_1} \arrow[l, "\lambda^{-1}"'] \arrow[r, "\psi"] & a_2 \arrow[r, "\lambda"] & b_2 \arrow[r, "\psi"] & a_3 \arrow[r, "\lambda"] & b_3 \ldots \end{tikzcd}$$

The forward part is longer than the backward part because the diffusion of μ^{-1} is better than for μ, so it will be easier to control the bit differences Hamming weight for several rounds forwards (instead of backwards). Table 2 shows probabilities of the best paths we found with the aid of the algorithm.

Table 2. Best differential paths results. The third column shows the weights of rounds for a given path.

rounds	total probability	products
1	2^{-2}	2^{-2}
2	2^{-10}	$2^{-8} \cdot 2^{-2}$
3	$2^{-18.4}$	$2^{-8.4} \cdot 2^{-2} \cdot 2^{-8}$
4	$2^{-52.8}$	$2^{-8.8} \cdot 2^{-2} \cdot 2^{-8} \cdot 2^{-34}$
5	$2^{-186.2}$	$2^{-10.4} \cdot 2^{-2} \cdot 2^{-8} \cdot 2^{-36} \cdot 2^{-129.8}$
6	$2^{-555.3}$	$2^{-10.4} \cdot 2^{-2} \cdot 2^{-8} \cdot 2^{-36} \cdot 2^{-129.8} \cdot 2^{-369}$

The weight of the 3-round path matches the bound we provided (2^{-18}) very closely. We investigated up to 6 rounds as the complexity of the attack exploiting the 5-round path is already intractable.

Internal differentials The best collision attack against Keccak was obtained through the technique called internal differentials [10]. While in standard differential attacks we consider two different plaintexts, in internal differential attacks only one plaintext is considered, and the statistical evolution of the differences between its parts is followed. In the attack against Keccak two properties were exploited, which led to the successful attack against the round-reduced Keccak. These properties are: very low Hamming weight constants (which helps to keep the state in the desired symmetry) and the fact that the state is initialized with the all-zero vector (which allows to construct the initial difference). For ICEPOLE it is not the case as the state is initialized with the pseudorandom constant and the round constants have much higher Hamming weight. Therefore we conclude it is not possible (or heavily limited) to successfully apply the internal differential technique to our scheme.

3.2 Linear Cryptanalysis

Linear cryptanalysis, formally introduced by Matsui [19], has become another powerful tool against modern cryptographic primitives. The main idea is to construct the linear approximation of the algorithm. In many ways this technique resembles differential cryptanalysis. Tracing the evolution of differences has the counterpart in tracing linear masks. Usually the complexity of the attack is also determined by the number of active S-boxes in the trail. One excellent example of exploiting the duality between these two techniques is the analysis of AES provided by its designers.

Although the structure of ICEPOLE does not allow for a straightforward and completely parallel analysis with respect to the two types of attacks, we think that ICEPOLE (and its permutation P) should offer very similar security margin against linear and differential crypt-analysis. First indication of it is the examination of the linear profile of the S-box. (The complete profile is given in Appendix D.) The highest bias of the linear approximation of the S-box is 2^{-2} and on average the bias is lower as the value of 2^{-2} happens rarely. (Note that the highest probability of difference transitions in the S-box is also 2^{-2}). The complexity of the linear attack is not only determined by the S-box properties but also by a number of active S-boxes on the trail. The μ step brings diffusion to the algorithm and hence it is the main factor for increasing a number of active S-boxes. The μ step affects linear and differential trails in the same way. Therefore we conclude that the complexity of the linear attack against ICEPOLE should be comparable with differential analysis and after 5-6 rounds the complexity becomes completely intractable.

3.3 SAT-based (Logic) Cryptanalysis

We encoded the following problem into SAT. An adversary knows a part of the input state, a part of the output state and the goal is to retrieve the unknown part of the input state. For ICEPOLE this problem models two types of attacks. The first type is the key recovery where an unknown part of the state is a secret key. The second type of the attack is the state recovery (in the processing phase), where the adversary tries to recover the unknown capacity part of the state.

To encode the problem into a SAT instance we used the toolkit presented in [15]. We obtained a SAT instance describing a single round of P with roughly 6400 variables and 35300 clauses. In attacks we used CryptoMiniSAT2, a gold medallist from recent SAT competitions [23]. We tried to solve three variants of the problem where 64-, 80-, and 128-bit part of the input state remains unknown. For 2 rounds CryptoMiniSAT2 was able to find the solution in a few seconds on a desktop PC. For 3 rounds only 64-bit variant of the problem was solved (also in a matter of seconds) and for 4 rounds, with 48-hour time limit, CryptoMiniSAT2 was unable to provide any solution. It looks as the hardness of the problem grows super exponentially in a number of rounds and this effect has been also observed in SAT-based attacks on other cryptographic primitives [15, 22]. Thus we conclude that ICEPOLE with the 12-round initialization and the 6-round processing phase is secure against the SAT-based attack.

3.4 Rotational Cryptanalysis

The technique was formally introduced in [18]. Unlike differential analysis, where the attacker follows the propagation of the xor differences of two plaintexts through the cryptographic system, in rotational analysis, the adversary investigates the propagation of the rotational relations between plaintexts. In [20] rotational cryptanalysis was applied to Keccak and since there are some similarities between ICEPOLE and Keccak, we take a closer look whether that technique could be used against ICEPOLE. In the attack on Keccak two properties were exploited, namely very low Hamming weight constants (which helps to keep the states in the desired rotational relation) and the fact that the state is initialized with the all-zero vector (which allows to construct the initial rotational relation). For ICEPOLE it is not the case as the state is initialized with the pseudorandom constant and the round constants have much higher Hamming weight. Therefore we conclude it is not possible (or heavily limited) to apply rotational cryptanalysis to our scheme.

3.5 Techniques Exploiting Low Algebraic Degree

There are several cryptanalytic techniques, which exploit a low algebraic degree. These are, for example, the cube attack [11] or the zero-sum distinguisher [1]. An algebraic degree of a single round of P (or its inverse) is 4. Then, after four rounds the algebraic degree is 256, which stops the mentioned attacks from reaching the attack complexity lower than the claimed security level 2^{128}. Thus ICEPOLE with its 12-round initialization is completely secure against techniques exploiting a low algebraic degree.

4 Features

ICEPOLE is designed for high-throughput network nodes or any environment where specialized hardware (such as FPGAs or ASICs) can be utilized to provide desired high data processing rates. As the duplex-based construction it inherits useful features:

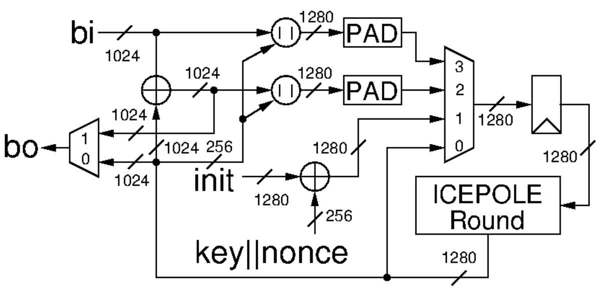
- it supports associated data processing
- it is an online cipher (ciphertext block c_i can be generated without knowing the plaintext block σ_{i+1})
- it can provide intermediate tags
- encryption is not expanding

Our new permutation P lets ICEPOLE work at the very high speed on the modern FPGA devices. Also our basic, non-optimized software implementation results are promising. Finally, we show how to process in parallel with ICEPOLE.

4.1 Hardware Performance

A proof-of-concept basic iterative architecture of ICEPOLE-128 was implemented. Figure 3 shows an overview of a datapath design. The presented cryptographic core is capable of performing encryption and decryption, and contains a full padding unit.

Fig. 3. A proof-of-concept single iterative round design for the hardware implementation of ICEPOLE



AES-GCM is used as a basis of our comparison as it is one of the most widely accepted standards for authenticated encryption [21]. The same basic iterative architecture is implemented for a direct comparison. Both implementations use also the same interface and communication protocol in order to reduce any discrepancies between the two designs. Similar to ICEPOLE, AES-GCM contains the full padding unit and supports both encryption and decryption within a single core.

Both cryptographic cores were described using VHDL language and verified against software generated test vectors using ModelSim. The results were generated using ATHENa [14] using two high-performance FPGA families from two major FPGA vendors, Xilinx and Altera. These FPGA families are Xilinx Virtex 6 and Altera Stratix IV, respectively. No dedicated resources, such as Block RAMs or DSP units, were used in either implementation. The comparison between ICEPOLE-128 and AES-128-GCM using a basic iterative architecture is shown in Table 3. The throughput shown in the table is based on the throughput of long messages.

Table 3. The comparison between ICEPOLE-128 and AES-128-GCM using an iterative architecture

	Xilinx Virtex 6			Altera Stratix IV		
	ICEPOLE-128	AES-128-GCM	ratio	ICEPOLE-128	AES-128-GCM	ratio
throughput (Gbit/s)	41.364	3.539	11.7	38.779	3.612	10.7
area (Slices/ALUT)	1501	940	1.6	4564	4025	1.13
throughput-to-area	27.56	3.76	7.3	8.5	0.9	9.4

With the exception of resource utilization, ICEPOLE-128 consistently outperforms AES-128-GCM in terms of the throughput and the throughput-to-area ratio. For Xilinx Virtex 6, with only 60% increases in area, ICEPOLE-128 achieves almost 12 times the speed of AES-128-GCM, and seven times higher the throughput-to-area ratio. For Altera Stratix IV, due to the unique behaviour of Altera Adaptive Look Up Tables (ALUTs), the resource utilization is similar for both algorithms, with ICEPOLE-128 consuming only 13% more area. At the same time, ICEPOLE-128 outperforms AES-128-GCM by a factor of 11 in terms of throughput and a factor of 9 in terms of the throughput-to-area ratio.

4.2 Software Performance

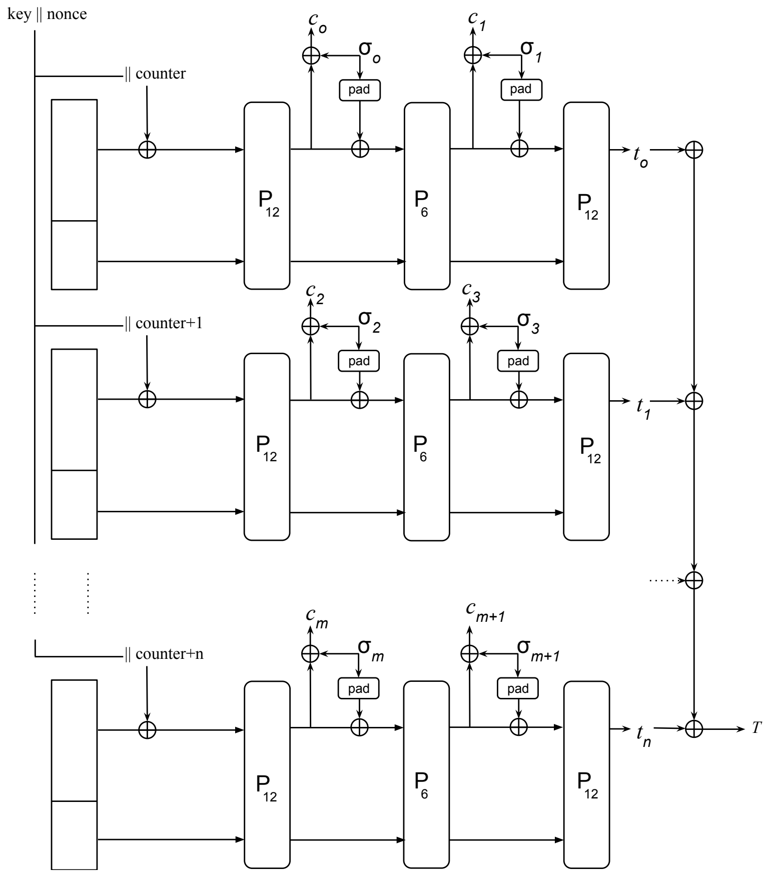
While the primary focus of the ICEPOLE design is hardware performance, the cipher is also amenable to efficient software implementations. The three steps that require non-trivial implementations are μ, ρ and ψ. They all can be easily implemented on platforms supporting 64-bit XORs, logical ANDs and rotations. We measured that a rather straightforward C implementation compiled for speed (with no beyond-C optimization efforts like code vectorization using AVX or intrinsics use) runs for very long messages at about 9 cycles per byte on Intel Ivy Bridge i5-3320M processor. The same implementation runs at about 8 cpb on a Haswell (Intel Xeon E3 1275) machine.

We believe there is still room for possible improvements. A better code optimization (e.g., making sure that the compiler uses the `andn a, b` instruction on Haswell for $\neg A \cdot B$ extensively used in the step ψ) could lead to a better performance than the reported 8 cycles per byte. Additionally, one could think about an AVX-based implementation where the whole state is kept in five YMM registers. Compared to the pure C code this could save time on memory loads and stores but at the expense of the more complex μ step. For 32-bit platforms, only rotation performance will scale worse than linearly (compared to the straightforward 64-bit version). It is because in such a case the rotations need to be combined from more than two instructions.

4.3 Parallel Processing

To be competitive with some block cipher based modes like OCB (Offset CodeBook) or GCM (Galois Counter Mode), the AEAD scheme should allow parallel processing.

Fig. 4. ICEPOLE used in the parallel mode. For clarity of the figure only plaintext blocks are shown, but the parallel mode supports associated data as well.



However, the scheme based on the duplex construction cannot be paralellized at the algorithmic level. In [6], the authors just briefly state that tree-like processing (described in detail for hashing, see Section 3.3.2 in [5]) could be used also for the duplex-based AEAD providing parallel streams with some overhead. Below we describe our simpler scheme where ICEPOLE is used to provide parallel processing.

As shown in Figure 4, blocks are processed in separate *streams*. Each stream has its own initialization phase that is the P_{12} call with a key, a nonce and a counter as an input. Then, each stream of blocks is processed in parallel. Each stream is ended with generating the 128-bit tag t_i, which are eventually combined with the XOR operation to produce the final tag T.

To simplify security analysis we let an attacker know intermediate tags $t_0, t_1, \ldots, t_n$. This way we can analyze every stream separately. And we already discussed the security analysis of a single stream (which is ICEPOLE in normal, sequential mode) and concluded it is secure. Then the only question remains whether the final XOR operation (producing the tag T) can be somehow exploited by the attacker. As $t_0, t_1, \ldots, t_n$ are already known, the key recovery attack (recovery of the state) starting from T does not make sense, it would only make the task harder. A tag forgery would mean that the attacker can control the xor differences between $t_0, t_1, \ldots, t_n$. But this would contradict the security of a single stream as we showed that after the initialization phase an attacker has no control of differences. (See Section 3.1.) Thus we conclude that the parallel processing we propose for ICEPOLE is secure.

5 Design Rationale

We have aimed at a high-speed, hardware-oriented authenticated encryption scheme, suitable for high-throughput network nodes or any environment where specialized hardware (such as FPGAs or ASICs) can be utilized to provide desired high data processing rates. Our main inspiration comes from the duplex construction with the round-reduced Keccak-f permutation [4]. We have decided to keep the general framework (namely the duplex construction) and design an underlying permutation from scratch.

In the Keccak-f permutation, the linear step θ brings most of diffusion to the algorithm and is roughly two times slower (when considering the FPGA design) than the non-linear part (a layer of S-boxes). Our general approach to the design of the permutation P has been to make steps more balanced. We have wanted to make the linear part simpler (faster) and improve the properties (diffusion, algebraic degree) of the non-linear part (in comparison to Keccak-f). The challenge has been that a more complex S-box layer should not nullify the gain from introducing a lighter linear part. Our second starting idea was to take into consideration cryptanalysis of Keccak-f and its round-reduced variants. Those findings have determined some of our decisions for ICEPOLE and its P permutation.

5.1 Permutation P Steps

Let us first explain design rationale behind the P permutation steps.

$\boldsymbol{\mu}$:

We have aimed at a possibly simple and implementation-friendly linear step. The step does not have to have an efficient inverse as in the duplex construction the permutation is calculated only in one way (both for encryption and decryption). Additionally, we have required that the linear step has very good diffusion properties. In [17] Junod and Vaudenay presented their research on building MDS matrix (known for an excellent diffusion property) under the criteria, which perfectly suit our needs, that is an efficient implementation and neglecting an inverse of the matrix. We have decided to use one of the 'optimal' matrices presented in [17].

The μ step also helped determine the size and organization of the state. First, we tried to keep the same state organization as in Keccak-f[1600] that is $5 \times 5 \times 64$. However the 'optimal' 5×5 MDS matrix did not give us a clear advantage (in terms of hardware implementation efficiency) over the linear step presented in Keccak. Hence we have decided to use the smaller 4×4 matrix, which would operate on four 5-bit vectors. The linear operation based on the chosen matrix can be implemented in just a single layer of LUTs in the modern FPGA devices. Consequently, to let our new linear step be applied naturally, the state has been organized as the two dimensional array 4×5 of 64-bit words, giving the 1280-bit state.

$\boldsymbol{\rho}$:

The ρ step is essential to bring diffusion along z axis in the state. Otherwise a given bit would only affect bits from its slice (the bits sharing the same z coordinate). The 20 offsets are calculated from a simple formula $i(i+1)/2$ modulo the word length (64 bits in the case of ICEPOLE). This formula is the same as in the Keccak permutation. A nice feature of this formula is that in the case of shorter word lengths, each word (or nearly each word) has a distinct offset value. This might come in handy when one tries to build ICEPOLE variant with a smaller state, better suited for constrained environments.

π:

The π step reorders the words in the state. We have introduced this step to bring extra diffusion between the words (which is already provided by μ and ψ). In hardware, π and ρ are 'cheap', their computational cost corresponds to wiring. The π formula has been chosen for its simplicity.

ψ:

We have aimed at the non-linear step (an S-box), which would have the following properties: good differential and linear profiles, an algebraic degree higher than 3, compact boolean circuit (low implementation cost). We have concluded that the Keccak S-box would not be the best choice mainly for its slow diffusion. Every bit affects only 3 others whereas we need better diffusion to complement μ. Secondly, the Keccak S-box algebraic degree is only 2 and for a small number of rounds techniques exploiting a low algebraic degree might be a threat. Therefore, ideally, we would like to keep good differential and linear profiles of the Keccak S-box and increase its diffusion and the algebraic degree. Our idea to achieve this goal was as follows. If we change the truth table of the Keccak S-box very little, differential and linear profiles should stay much the same (Though it needs to be verified.) Hopefully, a small change would improve diffusion and increase the algebraic degree. In the Keccak S-box the input vector '00000' is mapped onto '00000' and '11111' onto '11111'. If we switch them ('00000' $\Rightarrow$ '11111' and '11111' $\Rightarrow$ '00000'), this seemingly tiny change gives us all we want. Now every output bit depends on all 5 input bits (better diffusion) and the algebraic degree now equals 4, also the inverse of the new S-box has degree 4. The boolean description is still very compact, the equations are given in Section 1. As expected the differential and linear profiles remain very the same, keeping their good properties. The profiles are given in Appendix.

κ:

κ adds the 64-bit round constant and for each round a constant is different. Without κ all rounds of the permutation P would be equal making it subject to attacks exploiting symmetry such as slide attacks [8]. The constants used in Keccak have very low Hamming weight and this feature was exploited in two cryptanalytic attacks [10, 20] against the round-reduced Keccak. These results motivated us to introduce constants with much higher Hamming weight. The constant values are taken as the output of a simple 64-bit maximum-cycle Linear Feedback Shift Register (LFSR). The polynomial representation of LFSR is $x^{64} + x^{63} + x^{61} + x^{60} + 1$. The LFSR state is initialized with the 64-bit vector '0123456789ABCDEF' (hexadecimal format) and then each cycle generates a subsequent constant. Thus κ can be implemented as a simple LFSR circuit or a precomputed look-up table.

Steps order within a round

μ is the step, which provides the best mixing between the unknown part of the state (a secret key K) and the remaining part of the state which would be known to the attacker. Hence we have placed μ as the first step in a round. The order of other steps is arbitrary.

5.2 ICEPOLE Parameters and Decisions

ICEPOLE works on the 1280-bit state and the reason for that is explained above in the subsection on the μ step. ICEPOLE-128 (our primary recommendation) uses the 1024-bit input data block to be more hardware friendly. This value is a power of 2 and allows a more natural I/O operation in hardware as opposed to the slightly bigger size of 1088 bits (which could be also a choice). With 6-round processing phase where a large amount of data must be transferred within a short period, a non-power-of-2 block size can introduce an inefficiency in data transmission when the I/O width is large. Furthermore, with the 1024-bit input data block, a hardware implementation can more efficiently uses its storage, which can be important where aiming for an extremely small design.

Before P is applied, the state is initialized with the pseudorandom 1280-bit constant. This decision has been motivated by cryptanalysis on the round-reduced Keccak where two different techniques [10, 20] have exploited the fact the state is initialized with the all-zero maintaining many symmetries.

A number of rounds in Initialization is 12. This value is based on our differential cryptanalysis shown in Section 3. After Initialization an adversary should not have any control over the differences (when mounting the differential attack). The experiments indicate that 6 rounds are sufficient and we have doubled this value to get a solid security margin. A number of rounds in Processing Phase is 6. This value is based on our SAT-based cryptanalysis given in Section 3. We were able to recover a small unknown part of the state for 3 rounds. To get a solid security margin we have doubled the number of rounds in Processing Phase to 6. A number of rounds in Tag Generation is 12, providing a solid security margin against the ciphertext forgery [12].

The frame bit (introduced as a part of the padding) is needed for security analysis of the duplex construction working in the authenticated encryption mode [5, Section 4.1.5]. The chosen padding rule is the simplest sponge-compliant padding [5, Definition 2].

The designers have not hidden any weaknesses in this cipher.

Intellectual Property

ICEPOLE is available worldwide on a royalty-free, non-exclusive basis. If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

Consent

The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgements of the committee members and are not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

6 Changes

- From ICEPOLEv1 to ICEPOLEv2
  - A number of rounds in Tag Generation Phase has been increased to 12. This change introduces a solid security margin against the ciphertext forgery. In [12], it was shown the forgery can be mounted for 4 rounds. 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.
  - Claims on resistance against nonce misuse has been revised and formulated more precisely. It was motivated by [16].

References

1. Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Tech. rep., NIST mailing list (2009)
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the Indifferentiability of the Sponge Construction. In: Advances in Cryptology – Eurocrypt 2008. pp. 181–197 (2008), `http://sponge.noekeon.org/`
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. Symmetric Key Encryption Workshop (SKEW) (February 2011)
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption (July 2012)
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponges, `http://sponge.noekeon.org/CSF-0.1.pdf`
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: single-pass authenticated encryption and other applications. Cryptology ePrint Archive, Report 2011/499 (2011), `http://eprint.iacr.org/`
7. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
8. Biryukov, A., Wagner, D.: Slide attacks. In: FSE. pp. 245–259 (1999)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
10. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of sha-3 using generalized internal differentials. Cryptology ePrint Archive, Report 2012/672 (2012), http://eprint.iacr.org/
11. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: EUROCRYPT. pp. 278–299 (2009)
12. Dobraunig, C., Eichlseder, M., Mendel, F.: Forgery Attacks on round-reduced ICEPOLE-128. Cryptology ePrint Archive, Report 2015/392 (2015), `http://eprint.iacr.org/`
13. Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned Rebound Attack - Application to Keccak. Cryptology ePrint Archive, Report 2011/420 (2011)
14. Gaj, K., Kaps, J.P., Amirineni, V., Rogawski, M., Homsirikamol, E., Brewster, B.Y.: ATHENa - Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs. In: FPL. pp. 414–421 (2010)
15. Homsirikamol, E., Morawiecki, P., Rogawski, M., Srebrny, M.: Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In: 11th Int. Conf. on Information Systems and Industrial Management. LNCS, vol. 7564. Springer Berlin Heidelberg (2012)
16. Huang, T., Tjuawinata, I., Wu, H.: Differential-Linear Cryptanalysis of ICEPOLE. Cryptology ePrint Archive, Report 2015/160 (2015), `http://eprint.iacr.org/`
17. Junod, P., Vaudenay, S.: Perfect diffusion primitives for block ciphers. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3357, pp. 84–99. Springer (2004)
18. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In: Proceedings of the 17th international conference on Fast software encryption. pp. 333–346. LNCS, Springer-Verlag (2010)
19. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of feal cipher. In: EUROCRYPT. pp. 81–91 (1992)
20. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced Keccak. In: Fast Software Encryption. LNCS, Springer (2013)
21. National Institute of Standards and Technology: Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST special publication 800-38D (November 2007)
22. Rivest, R., Agre, B., Bailey, D.V., Crutchfield, C., Dodis, Y., Fleming, K.E., Khan, A., Krishnamurthy, J., Lin, Y., Reyzin, L., Shen, E., Sukha, J., Sutherland, D., Tromer, E., Yin, Y.L.: The MD6 hash function, `http://groups.csail.mit.edu/cis/md6/`
23. Soos, M.: CryptoMiniSat 2.5.0. In: SAT Race competitive event booklet (July 2010), `http://www.msoos.org/cryptominisat2`

Appendix

A

The round constants used in the κ step are given below. The values are given in hexadecimal using the little-endian format.

constant[0] := 0091A2B3C4D5E6F7	constant[1] := 0048D159E26AF37B
constant[2] := 002468ACF13579BD	constant[3] := 00123456F89ABCDE
constant[4] := 00091A2BFC4D5E6F	constant[5] := 00048D15FE26AF37
constant[6] := 0002468AFF13579B	constant[7] := 000123457F89ABCD
constant[8] := 000091A2BFC4D5E6	constant[9] := 000048D1DFE26AF3
constant[10] := 00002468EFF13579	constant[11] := 00001234F7F89ABC

B

The μ step changes the S state according to the following equations.

```
**for** $(z := 0;\ z < 64;\ z := z + 1)$ {
$S'[0][4][z] := S[0][3][z] \oplus S[1][4][z] \oplus S[2][4][z] \oplus S[3][4][z]$
$S'[0][3][z] := S[0][2][z] \oplus S[1][3][z] \oplus S[2][3][z] \oplus S[3][3][z]$
$S'[0][2][z] := S[0][4][z] \oplus S[0][1][z] \oplus S[1][2][z] \oplus S[2][2][z] \oplus S[3][2][z]$
$S'[0][1][z] := S[0][0][z] \oplus S[1][1][z] \oplus S[2][1][z] \oplus S[3][1][z]$
$S'[0][0][z] := S[0][4][z] \oplus S[1][0][z] \oplus S[2][0][z] \oplus S[3][0][z]$

$S'[1][4][z] := S[0][4][z] \oplus S[1][4][z] \oplus S[2][0][z] \oplus S[3][3][z]$
$S'[1][3][z] := S[0][3][z] \oplus S[1][3][z] \oplus S[2][4][z] \oplus S[3][2][z]$
$S'[1][2][z] := S[0][2][z] \oplus S[1][2][z] \oplus S[2][3][z] \oplus S[3][4][z] \oplus S[3][1][z]$
$S'[1][1][z] := S[0][1][z] \oplus S[1][1][z] \oplus S[2][2][z] \oplus S[2][0][z] \oplus S[3][0][z]$
$S'[1][0][z] := S[0][0][z] \oplus S[1][0][z] \oplus S[2][1][z] \oplus S[3][4][z]$

$S'[2][4][z] := S[0][4][z] \oplus S[1][3][z] \oplus S[2][4][z] \oplus S[3][0][z]$
$S'[2][3][z] := S[0][3][z] \oplus S[1][2][z] \oplus S[2][3][z] \oplus S[3][4][z]$
$S'[2][2][z] := S[0][2][z] \oplus S[1][4][z] \oplus S[1][1][z] \oplus S[2][2][z] \oplus S[3][3][z]$
$S'[2][1][z] := S[0][1][z] \oplus S[1][0][z] \oplus S[2][1][z] \oplus S[3][2][z] \oplus S[3][0][z]$
$S'[2][0][z] := S[0][0][z] \oplus S[1][4][z] \oplus S[2][0][z] \oplus S[3][1][z]$

$S'[3][4][z] := S[0][4][z] \oplus S[1][0][z] \oplus S[2][3][z] \oplus S[3][4][z]$
$S'[3][3][z] := S[0][3][z] \oplus S[1][4][z] \oplus S[2][2][z] \oplus S[3][3][z]$
$S'[3][2][z] := S[0][2][z] \oplus S[1][3][z] \oplus S[2][4][z] \oplus S[2][1][z] \oplus S[3][2][z]$
$S'[3][1][z] := S[0][1][z] \oplus S[1][2][z] \oplus S[1][0][z] \oplus S[2][0][z] \oplus S[3][1][z]$
$S'[3][0][z] := S[0][0][z] \oplus S[1][1][z] \oplus S[2][4][z] \oplus S[3][0][z]$
}
$S := S'$
```

C

Table 4. Difference distribution table of the S-box. Input and output differences are given in the hexadecimal format. Each element of the table represents the number of occurrences of the corresponding output difference ΔOUT given the input difference ΔIN. For clarity '-' denotes 0.

ΔOUT / ΔIN	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
00	32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
01	-	8	-	-	-	-	-	-	-	6	-	-	-	-	2	-	-	6	-	-	-	-	2	-	-	8	-	-	-	-	-	-
02	-	-	8	6	-	-	-	-	-	-	-	-	-	2	-	-	-	-	6	8	-	-	-	-	-	-	-	-	2	-	-	-
03	-	-	2	4	-	-	-	-	-	-	4	2	-	-	-	-	-	-	4	4	2	-	-	-	-	-	4	4	-	2	-	-
04	-	-	-	-	8	6	6	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	-	-	-	-	-
05	-	-	-	-	4	-	4	-	2	-	-	-	2	-	4	-	-	-	-	2	-	4	-	2	-	-	-	-	-	4	-	4
06	-	-	-	-	2	4	4	4	-	2	-	-	-	-	-	-	-	-	-	-	4	4	2	4	-	-	-	2	-	-	-	-
07	-	-	-	-	2	-	2	2	-	-	-	-	2	2	2	-	2	-	-	-	2	2	2	2	-	-	2	-	2	2	2	2
08	-	-	-	-	-	-	-	-	8	-	6	-	6	-	8	-	-	-	-	2	-	2	-	-	-	-	-	-	-	-	-	-
09	-	4	2	2	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	2	2	-	4
0a	-	-	-	-	-	-	-	2	4	-	-	4	4	-	-	2	2	-	-	-	-	-	-	-	2	-	-	4	4	-	-	4
0b	-	2	4	-	-	-	-	-	-	-	-	-	-	4	2	-	-	6	4	-	-	-	-	-	-	-	-	-	-	4	6	-
0c	-	-	-	-	-	-	-	-	2	4	4	4	4	2	4	4	-	-	2	-	-	-	-	2	-	-	-	-	-	-	-	-
0d	-	-	-	-	2	-	6	-	4	-	4	-	-	-	-	-	-	-	-	-	-	4	-	4	-	2	-	6	-	-	-	-
0e	-	2	-	-	-	-	-	-	2	2	-	2	2	2	2	2	-	-	-	-	-	2	-	-	2	2	2	2	2	2	-	2
0f	-	-	-	-	2	2	2	-	2	2	2	-	-	-	-	-	-	-	-	-	4	2	2	2	4	2	2	2	-	-	-	-
10	-	-	-	-	-	-	-	2	-	-	-	2	-	-	-	-	8	-	-	-	6	-	-	-	6	-	-	-	8	-	-	-
11	-	2	-	-	-	4	-	-	-	4	2	-	-	4	-	-	-	4	-	-	-	2	-	-	-	4	-	-	-	4	2	-
12	-	-	4	4	2	-	2	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	4	2	-	-	4	4
13	-	-	2	2	-	-	2	2	2	-	2	2	-	2	2	2	-	-	-	2	-	-	2	-	-	-	2	2	-	-	2	2
14	-	2	-	-	-	-	-	-	-	-	-	-	-	-	2	-	4	2	-	-	-	-	4	4	4	4	-	-	-	-	2	4
15	-	4	-	-	-	-	-	2	-	4	-	-	-	-	-	6	2	-	-	-	-	-	4	-	6	-	-	-	-	-	4	-
16	-	-	2	6	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	2	6	-	-
17	-	-	2	2	2	2	-	-	-	-	4	2	4	2	-	-	-	-	2	-	2	-	-	-	-	-	2	2	2	2	-	-
18	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	2	2	-	4	-	4	-	4	-	4	-	2	-	4	-	4	-
19	-	2	-	2	2	2	-	2	-	-	-	2	-	2	-	2	-	2	-	2	-	2	2	2	-	2	-	-	-	2	-	2
1a	-	-	-	-	-	-	-	-	2	-	-	4	6	-	-	4	4	-	-	2	4	-	-	6	-	-	-	-	-	-	-	-
1b	-	2	2	-	-	4	4	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	-	-	-	-	2	2	-
1c	-	-	2	-	-	-	-	-	-	-	-	2	-	-	-	-	2	2	2	2	-	2	2	2	2	2	2	2	2	-	2	2
1d	-	2	-	4	-	2	-	2	-	2	-	2	-	-	-	2	2	-	4	-	2	-	2	-	2	-	2	-	-	-	2	-
1e	-	-	-	-	-	-	-	-	2	4	2	2	2	2	-	2	2	4	2	2	2	2	-	2	-	-	-	-	-	-	-	-
1f	-	2	2	-	2	-	-	2	2	-	-	2	-	2	2	-	2	-	-	2	-	2	2	-	-	2	2	-	2	-	-	2

D

Table 5. Linear profile of the S-box. Input and output masks are given in the hexadecimal format. Each element in the table represents the number of mismatches between the linear equation represented by the input mask IN and the linear equation represented by the output mask OUT. Dividing an element value by 16 gives the probability that the corresponding equations are not equal.

OUT / IN	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
00	0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
01	16	10	18	24	18	12	16	14	18	16	16	18	16	22	18	20	18	16	16	18	16	14	18	12	16	18	18	16	18	12	16	14
02	16	18	10	16	18	16	24	18	18	16	12	14	16	18	14	12	18	16	16	18	16	18	18	16	16	18	22	12	18	16	20	14
03	16	24	16	16	16	12	16	20	16	16	12	12	16	20	12	16	16	16	16	16	16	12	16	12	16	16	12	20	16	12	12	16
04	16	18	18	16	10	16	16	18	18	16	16	18	24	18	18	16	18	16	16	18	12	22	14	12	16	18	18	16	14	20	12	14
05	16	8	16	8	16	12	16	20	16	16	16	16	16	20	16	12	16	16	16	16	20	16	12	16	16	16	16	16	20	16	12	16
06	16	16	24	16	16	16	16	16	16	16	12	12	16	16	20	12	16	16	16	16	12	12	12	20	16	16	20	12	12	12	16	16
07	16	10	18	16	18	20	16	22	18	16	12	14	16	14	14	16	18	16	16	18	12	18	22	16	16	18	14	20	14	16	16	22
08	16	18	18	16	18	16	16	18	10	12	16	22	16	14	18	12	18	16	16	18	16	18	18	16	24	14	18	20	18	12	16	14
09	16	16	16	16	16	20	16	20	8	12	16	20	16	16	16	16	16	16	16	16	16	12	16	12	8	20	16	12	16	16	16	16
0a	16	16	8	16	16	16	8	16	16	20	12	16	16	12	20	16	16	16	16	16	16	16	16	16	16	20	20	16	16	12	12	16
0b	16	18	18	8	18	20	16	14	18	20	12	18	16	18	22	20	18	16	16	18	16	14	18	12	16	14	14	16	18	16	20	14
0c	16	16	16	16	24	16	16	16	16	12	16	12	16	12	16	20	16	16	16	16	12	20	12	12	16	12	16	12	20	16	12	16
0d	16	18	18	16	18	20	16	14	18	12	16	14	8	18	18	16	18	16	16	18	20	18	14	16	16	22	18	20	14	20	12	14
0e	16	18	10	16	18	16	16	18	18	12	20	18	16	22	22	16	18	16	16	18	12	14	14	20	16	14	14	16	14	16	16	22
0f	16	16	16	8	16	12	16	12	16	12	20	16	16	16	12	20	16	16	16	16	12	16	20	16	16	20	20	16	12	12	16	16
10	16	18	18	16	18	16	16	18	18	16	16	18	16	18	18	16	10	24	12	14	16	18	22	20	16	18	14	12	18	16	12	14
11	16	16	16	16	16	12	16	12	16	16	16	16	16	12	16	12	24	16	12	20	16	12	20	16	16	16	12	12	16	20	12	16
12	16	16	16	16	16	16	16	16	16	16	20	12	16	16	20	12	8	8	12	20	16	16	20	12	16	16	16	16	16	16	16	16
13	16	18	18	16	18	12	16	14	18	16	12	22	16	14	14	16	10	16	20	22	16	14	14	16	16	18	18	16	18	20	16	22
14	16	16	16	16	8	16	16	16	16	16	16	16	8	16	16	16	16	16	20	20	12	20	16	16	16	16	12	12	20	12	16	16
15	16	18	18	16	18	12	16	22	18	16	16	18	16	14	18	20	18	8	20	14	20	18	18	20	16	18	14	12	14	16	16	14
16	16	18	18	16	18	16	8	18	18	16	20	14	16	18	14	12	18	16	20	14	12	14	18	16	16	18	18	16	22	20	20	14
17	16	16	16	16	16	20	16	12	16	16	12	20	16	20	12	16	16	8	12	12	12	16	16	20	16	16	16	16	20	16	12	16
18	16	16	16	16	16	16	16	16	24	12	16	20	16	12	16	12	16	16	12	12	16	16	12	12	16	20	12	16	16	12	20	16
19	16	18	18	16	18	12	16	14	10	20	16	14	16	18	18	16	18	16	12	14	16	22	14	16	16	22	14	16	18	16	20	22
1a	16	18	18	16	18	16	16	18	18	20	20	18	16	14	14	16	18	16	12	22	16	18	14	20	8	14	18	20	18	12	16	14
1b	16	16	16	16	16	12	16	12	16	12	12	16	16	16	20	12	16	16	20	12	16	20	20	16	8	12	16	20	16	16	16	16
1c	16	18	18	16	10	16	16	18	18	12	16	14	16	14	18	20	18	16	12	14	20	14	18	16	16	14	22	16	22	16	16	22
1d	16	16	16	16	16	12	16	20	16	20	16	20	8	16	16	16	16	16	12	12	12	16	16	12	16	12	20	16	12	20	16	16
1e	16	16	16	16	16	16	8	16	16	12	12	16	16	20	12	16	16	16	12	20	20	20	16	16	16	12	16	12	12	16	20	16
1f	16	18	18	16	18	20	16	14	18	20	20	18	16	18	14	12	18	16	20	14	20	18	18	12	16	14	18	12	14	12	12	22