

# ICEPOLE: High-Speed, Hardware-Oriented Authenticated Encryption

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

<sup>1</sup> Institute of Computer Science, Polish Academy of Sciences, Poland

<sup>2</sup> Section of Informatics, University of Commerce, Kielce, Poland

<sup>3</sup> Department of Computing, Macquarie University, Australia

<sup>4</sup> Electrical Engineering and Computer Science School,  
Science and Engineering Faculty,

Queensland University of Technology, Brisbane, Australia

<sup>5</sup> Cryptographic Engineering Research Group, George Mason University, USA

<sup>6</sup> Cryptography and Information Security Group,  
University of Bristol, United Kingdom

<sup>7</sup> Cadence Design Systems, San Jose, USA

<sup>8</sup> Intel, Gdańsk, Poland

**Abstract.** This paper introduces our dedicated authenticated encryption scheme ICEPOLE. ICEPOLE is a high-speed hardware-oriented scheme, suitable for high-throughput network nodes or generally any environment where specialized hardware (such as FPGAs or ASICs) can be used to provide high data processing rates. ICEPOLE-128 (the primary ICEPOLE variant) is very fast. On the modern FPGA device Virtex 6, a basic iterative architecture of ICEPOLE reaches 41 Gbits/s, which is over 10 times faster than the equivalent implementation of AES-128-GCM. The throughput-to-area ratio is also substantially better when compared to AES-128-GCM. We have carefully examined the security of the algorithm through a range of cryptanalytic techniques and our findings indicate that ICEPOLE offers high security level.

**Keywords:** authenticated encryption scheme, authenticated cipher, ICEPOLE.

## 1 Introduction

Protocols such as SSL/TLS [12,16], the backbone of the Internet, are designed to provide data confidentiality and authenticity. Often the underlying algorithms of these protocols realize encryption and authentication separately (e.g., AES in CBC mode for encryption and HMAC-SHA1 for authentication). Although this approach leads to relatively easy security analysis, the performance, due to

two separate algorithms, does not meet demands of modern applications. Hence, recently, the symmetric crypto community focuses its attention on dedicated authentication encryption schemes, which aim to provide message encryption and authentication more efficiently. An interest in new efficient and secure solutions is manifested in the recently launched competition called CAESAR [1].

This paper presents our new design of the authenticated encryption scheme called ICEPOLE. It is a family of authenticated ciphers with two parameters: key length (128 or 256 bits) and nonce length (between 0 and 128 bits). Our primary recommendation is ICEPOLE-128 which uses 128-bit key and 128-bit nonce. The claimed security level for the primary variant is 128 bits and it is supported by our extensive cryptanalysis. ICEPOLE is based on the duplex framework introduced by Bertoni et al. in [8]. At the heart of the duplex framework is a permutation and the ICEPOLE permutation is our new design. In particular, inspired by the Keccak non-linear step [7], we introduce a new S-box with good security properties and low implementation cost.

ICEPOLE is a high-speed hardware-oriented scheme, suitable for high-throughput network nodes or more generally any environment where specialized hardware (such as FPGAs or ASICs) can be used to provide high data processing rates. ICEPOLE-128 is very fast. On the modern FPGA device Virtex 6, a basic iterative architecture of ICEPOLE reaches 41 Gbits/s, which is over 10 times faster than the equivalent implementation of AES-128-GCM [23] (one of the most common standards for authenticated encryption). The throughput-to-area ratio is also substantially better than AES-128-GCM results.

The paper is organized as follows. ICEPOLE specification is given in Section 2. Our security analysis is presented in Section 3. Next, Section 4 shows hardware performance on FPGA devices and the comparison with the AES-GCM hardware implementation. Then, software performance is given in Section 5. Finally, in Section 6, we describe all the key decisions with motivation and justification behind them.

## 2 Specification

Our primary recommended parameter set is: 128-bit key and 128-bit nonce. The ICEPOLE variant with these recommended parameters 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, 96-bit nonce) and ICEPOLE-256a (256-bit key, 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 described in Appendix D.

### 2.1 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.

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  $v[64(x + 4y) + z] = S[x][y][z]$ . (At some points in the algorithm’s description we xor the state with a vector so this mapping must be specified.) 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  $S_{[n]}$  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).

### 2.2 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 2.2.

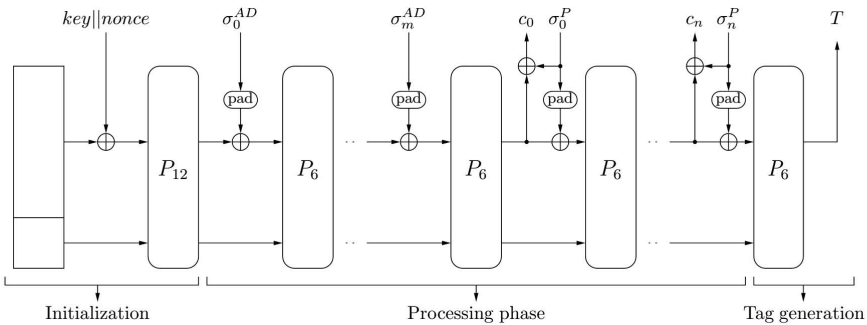


Fig. 1. General scheme of ICEPOLE encryption and authentication

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

### 2.3 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:  $\mu$  (mu),  $\rho$  (rho),  $\pi$  (pi),  $\psi$  (psi),  $\kappa$  (kappa).

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

Each step updates the state as follows.

**$\mu$ :**

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.

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

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  $\mu$  step can be efficiently implemented with simple bitwise equations (see Appendix G).

### $\rho$ :

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

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

The rotation offsets are given in Appendix A.

### $\pi$ :

$\pi$  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.

$$\begin{aligned} x' &:= (x + y) \bmod 4 \\ y' &:= (((x + y) \bmod 4) + y + 1) \bmod 5 \end{aligned}$$

### $\psi$ :

In the  $\psi$  step the ICEPOLE S-box is applied to each of 256 rows of the state. The S-box maps a 5-bit input vector  $(M_0, M_1, \dots, M_4)$  to a 5-bit output vector  $(Z_0, Z_1, \dots, Z_4)$ . 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.

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

**$\kappa$ :**

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

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

The constant value for each round is different. The values are given in Appendix B.

**2.4 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 constant is given in Appendix C.)

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,  $\text{nonce}_0$  and  $\text{nonce}_1$  denote two 64-bit words of the nonce.

$$\begin{aligned} S[0][0] &:= S[0][0] \oplus K_0 \\ S[1][0] &:= S[1][0] \oplus K_1 \\ S[2][0] &:= S[2][0] \oplus \text{nonce}_0 \\ S[3][0] &:= S[3][0] \oplus \text{nonce}_1 \end{aligned}$$

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

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

**2.5 Processing Phase**

The input data is processed in blocks. First, the associated data blocks  $\sigma_i^{AD}$  are processed and next the plaintext blocks  $\sigma_i^P$ . The plaintext blocks are authenticated and encrypted whereas the associated data blocks are only authenticated.

A 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  $\sigma_i^{AD}$  block and all  $\sigma_i^P$  except the last one. Otherwise the frame bit is set to 0. Once the frame bit is appended, a given block is padded with a simple rule: append 1 and such a number of 0's which gives 1026-bit block. 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.

$$\begin{aligned} \text{for all blocks } \sigma_i^{AD} \{ \\ \sigma_i^{AD} &:= \text{pad}(\sigma_i^{AD}) \\ S_{[1026]} &:= S_{[1026]} \oplus \sigma_i^{AD} \end{aligned}$$

```

    S := P6(S)
  }

  for all blocks σiP {
    ci = S[l] ⊕ σiP (l is a length of σiP)
    σiP := pad(σiP)
    S[1026] := S[1026] ⊕ σiP
    S := P6(S)
  }

```

### 2.6 Tag Generation

When the blocks processing is finished,  $P_6$  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 := P6(S)
    T0 := S[0][0]
    T1 := S[1][0]

```

### 2.7 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 2.7 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 verified with the tag received from the sender. If the tags match, the data is authenticated.

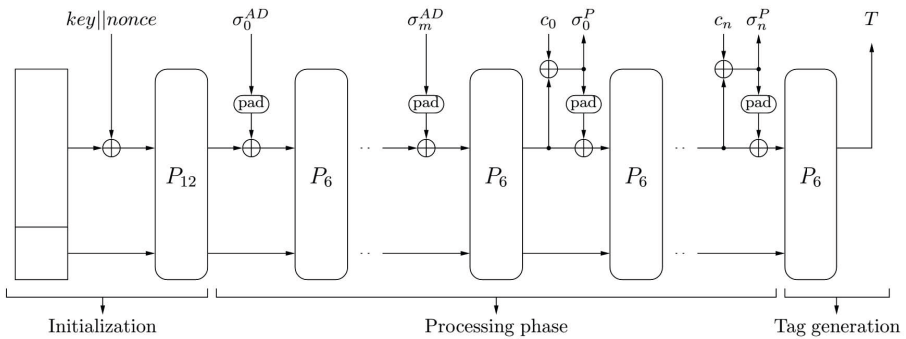


Fig. 2. ICEPOLE decryption

### 3 Security Analysis

In 2010, Bertoni et al. introduced the duplex construction [8] 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 duplex construction can be seen as a particular way of using the sponge construction [6]. 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 [3], 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 [4] 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 strength 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.

In [6] it was shown 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.

A user of ICEPOLE is required to use a nonce. However, in the case of nonce reuse ICEPOLE provides some intermediate level of robustness. The nonce reuse leads to leaking XOR of plaintexts (via XOR of ciphertexts) and this cannot be avoided in our case. The secret key used with the same nonce also leads to the situation where the adversary can control the XOR differences after the 12-round initialization. But our cryptanalysis (given below) strongly indicates that the 6-round permutation  $P$  (run at the processing phase) is secure, in particular against differential and linear cryptanalysis. Therefore we argue that in the case of nonce reuse the key-recovery attack is not possible.

#### 3.1 Differential Cryptanalysis

Differential cryptanalysis, introduced by Biham and Shamir [9], 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 [11]. 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: (1) translate the problem to SAT (in such a way that a satisfying valuation represents a solution to the problem); (2) 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 [18]. Crypto-MiniSat2 [25] 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 F), 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. Details are presented in Appendix D.

**Internal Differentials.** The best collision attack against Keccak was obtained through the technique called internal differentials [13]. While in standard differ-



ential 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 [21], 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 cryptanalysis. First indication of it is the examination of the linear profile of the S-box. (The complete profile is given in Appendix H.) 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  $\mu$  step brings diffusion to the algorithm and hence it is the main factor for increasing a number of active S-boxes. The  $\mu$  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 attacker tries to recover the unknown capacity part of the state.

To encode the problem into a SAT instance we used the toolkit presented in [18]. 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 [25]. 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 [18,24]. 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

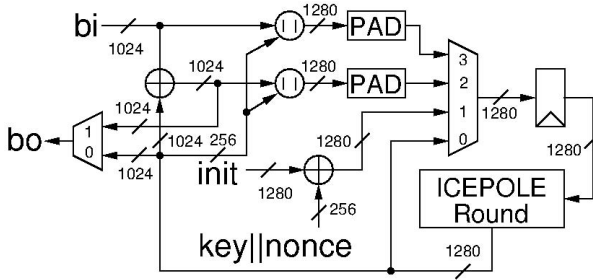
The technique was formally introduced in [20]. Unlike differential analysis, where the attacker follows the propagation of the xor differences of two plaintexts through the cryptographic system, in rotational analysis, the adversary investigates the propagation of the rotational relations between plaintexts. In [22] 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 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 [14] or the zero-sum distinguisher [2]. 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 Hardware Performance

A proof-of-concept basic iterative architecture of ICEPOLE-128 was implemented. Figure 4 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 [23]. 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 [17] 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 1. The throughput shown in the table is based on the throughput of long messages.

**Table 1.** 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.

We strive to provide the fairest possible comparison with AES-GCM. When we state that the design used is an ‘iterative architecture’, this means that it takes 10 clock cycles to calculate the AES. Certainly, we selected the multiplier unit that optimizes the unit to meet that latency. This means that we are performing a 128-bit  $\times$  128-bit multiplication using at least a 128-bit  $\times$  16-bit multiplier to satisfy the 10 clock cycles requirement. For a fair comparison, we selected a 128-bit  $\times$  16-bit multiplier because 128-bit  $\times$  32-bit or larger multiplier would increase the area unnecessarily.

As ICEPOLE is inspired by the Keccak permutation, it is natural to ask whether our new construction offers better performance over the Keccak permutation. The main reason ICEPOLE permutation is faster than Keccak is that the linear step  $\mu$  can be implemented in the single layer of LUTs whereas Keccak  $\theta$  needs 2 layers. Hence both area and throughput for this step would be to ICEPOLE advantage. Other steps in permutation are basically equivalent in terms of performance. Our experiments supports this analysis. The experiment was conducted by wrapping ICEPOLE and Keccak permutations with a shift register for I/O. Then the same optimization technique used in our paper was applied to both designs. The improvement in speed is consistent across all device families with the highest performance gain in Altera as much as 20%.

## 5 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 nontrivial implementations are  $\mu$ ,  $\rho$  and  $\psi$ . 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  $\psi$ ) 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  $\mu$  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.

## 6 Design Rationale

We have aimed at 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 [5]. 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  $\theta$  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.

### 6.1 Permutation P Steps

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

#### $\mu$ :

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 [19] 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 [19].

The  $\mu$  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’  $5 \times 5$  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  $4 \times 4$  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  $4 \times 5$  of 64-bit words, giving the 1280-bit state.

**$\rho$ :**

The  $\rho$  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.

 **$\pi$ :**

The  $\pi$  step reorders the words in the state. We have introduced this step to bring extra diffusion between the words (which is already provided by  $\mu$  and  $\psi$ ). In hardware,  $\pi$  and  $\rho$  are ‘cheap’, their computational cost corresponds to wiring. The  $\pi$  formula has been chosen for its simplicity.

 **$\psi$ :**

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  $\mu$ . 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 then) and the algebraic degree now equals 4, also the inverse of the new S-box has degree equals 4. The boolean description is still very compact, the equations are given in Section 2. As expected the differential and linear profiles remain very the same, keeping their good properties. The profiles are given in Appendix.

 **$\kappa$ :**

$\kappa$  adds the 64-bit round constant and for each round a constant is different. Without  $\kappa$  all rounds of the permutation  $P$  would be equal making it subject to attacks exploiting symmetry such as slide attacks [10]. The constants used in

Keccak have very low Hamming weight and this feature was exploited in two cryptanalytic attacks [13,22] 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  $\kappa$  can be implemented as a simple LFSR circuit or a precomputed look-up table.

### Steps order within a Round

$\mu$  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  $\mu$  as the first step in a round. The order of other steps is arbitrary.

## 6.2 ICEPOLE Parameters and Decisions

ICEPOLE works on the 1280-bit state and the reason for that is explained above in the subsection on the  $\mu$  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 use 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 [13,22] have exploited the fact the state is initialized with the all-zero maintaining many different symmetries.

The 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.

The 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.

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 [6, Section 4.1.5]. The chosen padding rule is the simplest sponge-compliant padding [6, Definition 2].

## 7 Conclusion

We have proposed the dedicated authenticated encryption scheme called ICEPOLE. It is very fast on the modern hardware platforms and its hardware performance is substantially better than the AES-GCM. Our software non-optimized implementation, running at 8 cycles per byte, is also a promising result. We performed a security analysis with aid of many cryptanalytic tools and our findings show that ICEPOLE offers solid security margin. Our new permutation  $P$  (combining with the sponge construction [6] or other permutation-based modes) could become a building block for a new, high-speed cryptographic primitive such as a hash function or a stream cipher.

**Acknowledgment.** Project was financed by Polish National Science Centre, project DEC-2013/09/D/ST6/03918. Josef Pieprzyk was supported by the ARC grant DP0987734. The work described in this paper has been supported in part by EPSRC grant EP/H001689/1.

## References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, <http://competitions.cr.yt.to/caesar.html>
2. 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)
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008) <http://sponge.noekeon.org/>
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. Symmetric Key Encryption Workshop (SKEW) (February 2011)
5. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption (July 2012)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponges, <http://sponge.noekeon.org/CSF-0.1.pdf>
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document, <http://keccak.noekeon.org/Keccak-main-2.1.pdf>
8. 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/>
9. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
10. Biryukov, A., Wagner, D.: Slide attacks. In: FSE, pp. 245–259 (1999)
11. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
12. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol. Tech. rep., Network Working Group (2008)
13. 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/>



14. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
15. Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned Rebound Attack - Application to Keccak. Cryptology ePrint Archive, Report 2011/420 (2011)
16. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol. Tech. rep., Internet Engineering Task Force, IETF (2011)
17. 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)
18. Homsirikamol, E., Morawiecki, P., Rogawski, M., Srebrny, M.: Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In: Cortesi, A., Chaki, N., Saeed, K., Wierzchoń, S. (eds.) CISIM 2012. LNCS, vol. 7564, pp. 56–67. Springer, Heidelberg (2012)
19. Junod, P., Vaudenay, S.: Perfect diffusion primitives for block ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 84–99. Springer, Heidelberg (2004)
20. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 333–346. Springer, Heidelberg (2010)
21. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81–91. Springer, Heidelberg (1993)
22. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced Keccak. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 241–262. Springer, Heidelberg (2014)
23. 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)
24. 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/>
25. Soos, M.: CryptoMiniSat 2.5.0. In: SAT Race competitive event booklet (July 2010), <http://www.msoos.org/cryptominisat2>

## Appendix

### A

The rotation offsets used in the  $\rho$  step are given below.

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

**B**

The round constants used in the  $\kappa$  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

**C**

At the start of Initialization Phase the 1280-bit state is initialized with the pseudorandom constant. The values are given in hexadecimal using the little-endian format.

$S[0][0]$ := <i>FF97A42D7F8E6FD4</i>	$S[0][1]$ := <i>90FEE5A0A44647C4</i>
$S[0][2]$ := <i>8C5BDA0CD6192E76</i>	$S[0][3]$ := <i>AD30A6F71B19059C</i>
$S[0][4]$ := <i>30935AB7D08FFC64</i>	$S[1][0]$ := <i>EB5AA93F2317D635</i>
$S[1][1]$ := <i>A9A6E6260D712103</i>	$S[1][2]$ := <i>81A57C16DBCF555F</i>
$S[1][3]$ := <i>43B831CD0347C826</i>	$S[1][4]$ := <i>01F22F1A11A5569F</i>
$S[2][0]$ := <i>05E5635A21D9AE61</i>	$S[2][1]$ := <i>64BEFEF28CC970F2</i>
$S[2][2]$ := <i>613670957BC46611</i>	$S[2][3]$ := <i>B87C5A554FD00ECB</i>
$S[2][4]$ := <i>8C3EE88A1CCF32C8</i>	$S[3][0]$ := <i>940C7922AE3A2614</i>
$S[3][1]$ := <i>1841F924A2C509E4</i>	$S[3][2]$ := <i>16F53526E70465C2</i>
$S[3][3]$ := <i>75F644E97F30A13B</i>	$S[3][4]$ := <i>EAF1FF7B5CECA249</i>

**D**

**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 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.

**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. The 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.

$$\begin{aligned}
S[0][0] &:= S[0][0] \oplus K_0 \\
S[1][0] &:= S[1][0] \oplus K_1 \\
S[2][0] &:= S[2][0] \oplus K_2 \\
S[3][0] &:= S[3][0] \oplus K_3 \\
S[0][1] &:= S[0][1] \oplus \textit{nonce}_0 \\
S[1][1] &:= S[1][1] \oplus \textit{nonce}_1
\end{aligned}$$

The data blocks have the length between 0 and 960 bits and 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.

## E

### Differential Path Search

We were inspired by the work of Duc et al. [15] 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 the bit difference Hamming weight as low as possible. We note that  $\mu$ ,  $\rho$ ,  $\pi$  are all linear mappings (denoted altogether by  $\lambda$ , while  $\psi$  acts as the non-linear S-box.  $\kappa$  (adding round constants) does not affect differential analysis in any way. Furthermore,  $\rho$  and  $\pi$  do not change the number of active bits in a differential path, but change only bit positions. Hence,  $\mu$  and  $\psi$  are critical when analysing differential paths. Since  $\psi$  is followed by  $\mu$  in the next round (ignoring  $\kappa$ ), 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  $\psi$  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  $\mu$ .

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  $\lambda$ ) in the second round, and compute backwards for one round, and a few rounds forwards, as shown below.

$$a_0 \xleftarrow{\lambda^{-1}} b_0 \xleftarrow{\psi^{-1}} a_1 \xleftarrow{\lambda^{-1}} \mathbf{b}_1 \xrightarrow{\psi} a_2 \xrightarrow{\lambda} b_2 \xrightarrow{\psi} a_3 \xrightarrow{\lambda} b_3 \dots$$

The forward part is longer than the backward part because the diffusion of  $\mu^{-1}$  is better than for  $\mu$ , 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 completely intractable.

**F**

**G**

The  $\mu$  step changes the  $S$  state according to the following equations.

```

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

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

```

**Table 3.** The 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  $\Delta OUT$  given the input difference  $\Delta IN$ . For clarity ‘-’ denotes 0.

$\frac{\Delta OUT}{\Delta IN}$	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	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	-	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	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	2	-	-	
18	-	-	-	-	-	2	-	-	-	-	-	-	-	-	2	2	-	4	-	4	-	4	-	4	-	4	-	-	4	-	-	-	
19	-	2	-	2	2	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	2	2	2	-	2	2	
1c	-	-	2	-	-	-	-	-	-	-	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	-	-	-	2	-
1e	-	-	-	-	-	-	-	2	4	2	2	2	2	-	2	2	4	2	2	2	2	2	2	-	2	-	-	-	-	-	-	-	-
1f	-	2	2	-	2	-	-	2	-	-	2	-	2	-	2	-	-	-	-	-	2	2	-	2	2	-	2	2	-	-	-	2	-

$$\begin{aligned}
 S'[2][4][z] &:= S[0][4][z] \oplus S[1][3][z] \oplus S[2][4][z] \oplus S[3][0][z] \\
 S'[2][3][z] &:= S[0][3][z] \oplus S[1][2][z] \oplus S[2][3][z] \oplus S[3][4][z] \\
 S'[2][2][z] &:= S[0][2][z] \oplus S[1][4][z] \oplus S[1][1][z] \oplus S[2][2][z] \oplus S[3][3][z] \\
 S'[2][1][z] &:= S[0][1][z] \oplus S[1][0][z] \oplus S[2][1][z] \oplus S[3][2][z] \oplus S[3][0][z] \\
 S'[2][0][z] &:= S[0][0][z] \oplus S[1][4][z] \oplus S[2][0][z] \oplus S[3][1][z]
 \end{aligned}$$

$$\begin{aligned}
 S'[3][4][z] &:= S[0][4][z] \oplus S[1][0][z] \oplus S[2][3][z] \oplus S[3][4][z] \\
 S'[3][3][z] &:= S[0][3][z] \oplus S[1][4][z] \oplus S[2][2][z] \oplus S[3][3][z] \\
 S'[3][2][z] &:= S[0][2][z] \oplus S[1][3][z] \oplus S[2][4][z] \oplus S[2][1][z] \oplus S[3][2][z] \\
 S'[3][1][z] &:= S[0][1][z] \oplus S[1][2][z] \oplus S[1][0][z] \oplus S[2][0][z] \oplus S[3][1][z] \\
 S'[3][0][z] &:= S[0][0][z] \oplus S[1][1][z] \oplus S[2][4][z] \oplus S[3][0][z] \\
 \}
 \end{aligned}$$

$$S := S'$$

## H

**Table 4.** The linear profile of the S-box. Input and output masks are given in the hexadecimal format. Each element in the table is 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.

$\frac{OUT}{IN}$	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		
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	12	16	12	16	16	16	12	20	16	12	16		
04	16	18	18	16	10	16	16	18	18	16	16	18	24	18	18	16	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	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		
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	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	16	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	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	16	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	16	20	16	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	