

The *Iak* Block Cipher

Kai Chen, Bill Sacks

1 Overview

This document presents a brief discussion of the block cipher *Iak*. We start with some mathematical preliminaries necessary to understand the cipher. Then we describe the design rationale, followed by the specification, its implementation, and possible extensions.

Disclaimer: The designer of the cipher does not hold responsibility for any damage or loss caused by any direct or indirect use of *Iak* or its implementations.

2 Preliminaries

A byte $B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ can be viewed as an element of the *external direct product*: Z_2^8 .

Define a *permutation* \mathcal{P} on Z_2^8 by: $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0) \rightarrow (b_7, b_6, b_4, b_5, b_3, b_2, b_0, b_1)$

Break a byte into 4 *2-bit blocks*. Each *block* is represented by the *direct product* $Z_2 \oplus Z_2$. Clearly the effect of \mathcal{P} on a *block* is either doing nothing or swapping the two bits. Call this operation \mathcal{P}' . Notice that $\mathcal{P}' = \mathcal{P}'^{-1}$.

Next, define $\phi : Z_2 \oplus Z_2 \rightarrow Z_2 \oplus Z_2$ by:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Since all entries are in Z_2 , we have: $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, it follows that $\phi = \phi^{-1}$.

3 Design Rationale

The goal here is to design a cipher at byte level to:

- **model major features of some real ciphers:** *Iak* is *block-structured*, *non-linear*, and can be easily extended to support *multi-round feistel features*(see Section 6).
- **reduce implementation complexity:** In particular, *Iak* is *symmetric*, so that the same piece of hard can be used to both encrypt and decrypt.
- **minimize design complexity:** No unnecessary feature is added.

4 Specification

In this section we describe how *Iak* and its inverse work.

4.1 Encryption

We take 2 bytes as input: one byte for *plaintext* and one byte for *key*. The output is one byte of *ciphertext*. We break the *plaintext* byte and the *key* byte into 4 2-bit blocks. On each *block*, we perform the following:

1. **Permutation:** Perform \mathcal{P} on the *plaintext* byte, or equivalently, perform \mathcal{P}' on each *block*.
2. **Matrix Multiplication:** Perform ϕ on each *block*.
3. **Key Addition:** Add each *key block* to a *plaintext block*. Addition here is in $Z_2 \oplus Z_2$, or *bitwise exclusive-or*.
4. **Permutation:** Perform \mathcal{P}' on each *block* again.

So if we denote a *plaintext block* as x , a *key block* as k , and a *ciphertext block* as y , we have: $Iak : y = \mathcal{P}'(\phi(\mathcal{P}'(x)) + k)$. Notice here $x, k, y \in Z_2 \oplus Z_2$

Next we look at the inverse of the cipher.

4.2 Decryption

Here we will make use of the fact that: $\mathcal{P}' = \mathcal{P}'^{-1}$ and $\phi^{-1} = \phi$. Also both addition and subtraction are *bitwise exclusive-or* in Z_2

$$\begin{aligned} & y = \mathcal{P}'(\phi(\mathcal{P}'(x)) + k) \\ \rightarrow & \mathcal{P}'(y) = \phi(\mathcal{P}'(x)) + k \\ \rightarrow & \mathcal{P}'(y) + k = \phi(\mathcal{P}'(x)) \\ \rightarrow & \phi(\mathcal{P}'(y) + k) = \mathcal{P}'(x) \\ \rightarrow & \phi(\mathcal{P}'(y)) + \phi(k) = \mathcal{P}'(x), \text{ since } \phi \text{ is a } \textit{homomorphism} \\ \rightarrow & x = \mathcal{P}'(\phi(\mathcal{P}'(y)) + \phi(k)) \end{aligned}$$

Notice that Iak^{-1} is almost exactly the same as Iak , except that we need to perform ϕ on k . This greatly reduces the implementation complexity.

5 Implementation

Permutation can be done by direct wiring. *Key* addition can be implemented as *bitwise exclusive-or*. So we only need to examine the function ϕ .

By enumerating the mappings of ϕ :

$$(0, 0) \rightarrow (0, 0), (0, 1) \rightarrow (0, 1), (1, 0) \rightarrow (1, 1), (1, 1) \rightarrow (1, 0).$$

We notice that this is exactly converting normal binary to *gray-code*, which can be efficiently implemented as:

$$(x_1, x_0) \rightarrow (x_1, x_1 + x_0)$$

So the cipher and its inverse can be represented by the following *boolean* equation:

$(x_1, x_0) \rightarrow ((x_0 \oplus x_1 \oplus k_0 \oplus k_1 * DEC), (x_0 \oplus x_1))$, where \oplus denotes *XOR* and $*$ denotes *AND*.

When the bit *DEC* is high, the device works as decryption, and when low, it works as encryption.

6 Extension

As was mentioned in Section 3, *Iak* can be easily extended to support the *multi-round* feature. We will need to design a scheme *Key-Expansion* to expand the key to n bytes, where n is the number of rounds to perform the cipher. Our hardware model is a *Systolic Array* with 8 *nodes*. At each *node*, we perform one round of the cipher, then it passes the partial result, called a *State* to the next *node* for the next round of encryption. At the same time it takes in a new byte of input from the previous *node*, and encrypt it with its *round key*. So at any moment, exactly 8 bytes can be encrypted in parallel.

7 References

Joan Daemen, Vincent Rijmen, “The Rijndael Block Cipher” AES Proposal.