Specification and Analysis of CRYPTON Version 1.0

Chae Hoon Lim

Information and Communications Research Center
Future Systems, Inc.
372-2, Yang Jae-Dong, Seo Cho-Gu, Seoul, 137-130, Korea
Tel: +82-2-578-0581, Fax: +82-2-578-0584
chlim@future.co.kr

December 22, 1998

Abstract

The block cipher CRYPTON had been proposed as one of candidate algorithms for the Advanced Encryption Standard (AES). Though there is no serious weakness found, we decided to strengthen CRYPTON with as small changes as possible. As a result of such refinement effort, we now release CRYPTON Version 1.0, which only differs from the previous version in S-box construction and key scheduling. This paper describes this revised version and present our new analysis results on its security and efficiency.

1 Introduction

The block cipher CRYPTON is designed using a kind of substitution-permutation (SP) networks. In fact, its overall structure is based on the structure of SQUARE [2]. In CRYPTON each data block of 128 bits is processed by representing it into a 4×4 byte array as in SQUARE. The round transformation of CRYPTON consists of four parallelizable steps: byte-wise substitutions, column-wise bit permutation, column-to-row transposition, and then key addition. The encryption process involves 12 repetitions of (essentially) the same round transformation. The decryption process can be made identical to the encryption process with a different key schedule. Figure 1 shows the high level structure of CRYPTON.

The block cipher CRYPTON has the following features:

- 12-round self-reciprocal cipher (with identical processes for encryption and decryption) with block length 128 bits and key length up to 256 bits.
- Simplicity and ease of analysis: The simplicity of CRYPTON makes it easier to perform various security evaluations.
- Strong security against existing attacks: e.g, differential and linear cryptanalysis require much more ciphertexts than are available.
- Efficiency in both SW and HW: Thanks to the high degree of parallelisms and the use of only very simple operations (just logical ANDs/XORs, rotates and table lookups), CRYPTON runs very fast on most platforms in software as well as in hardware.
- Wide tradeoffs between speed and memory: It can be implemented using the storage of 32, 256, 1K, 4K, 16K or 512K bytes, etc. Its speed on Pentium Pro 200 MHz ranges from 51 Mbps (in C) to 63 Mbps (in in-line ASM).
- Fast key scheduling: The encryption key schedule runs much faster than one-block encryption, so it is very efficient for use in the application requiring frequent key changes (e.g., in hashing mode).

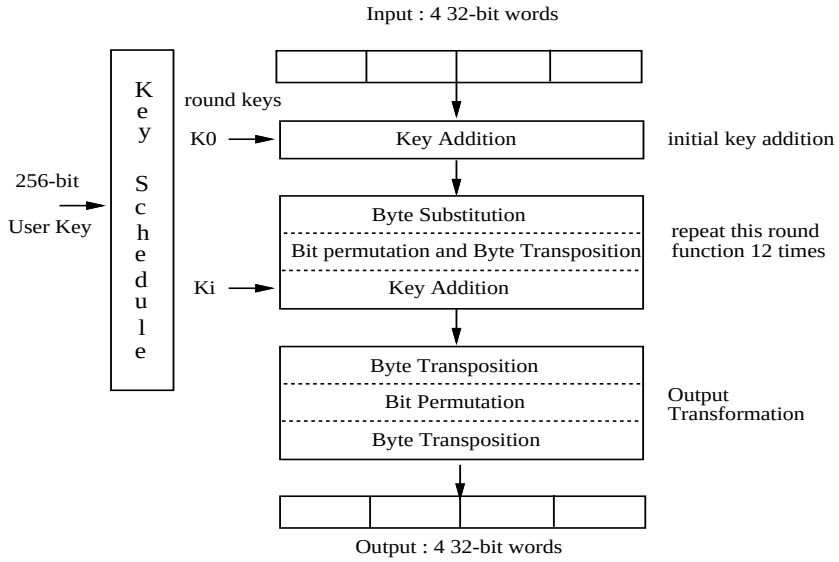


Figure 1: The structure of CRYPTON

We had submitted CRYPTON (Version 0.5) as a candidate algorithm for the AES. Unfortunately, however, as we noted at the CRYPTON home page, we couldn't have enough time to refine our algorithm at the time of submission. So, we later revised part of the AES proposal. This paper describes the revised version (Version 1.0) and analyzes its security and efficiency.

1.1 Motivations and Changes

CRYPTON Version 1.0 is different from the AES proposal (Version 0.5) only in the S-box construction and key scheduling. In Version 0.5 we used two 8×8 S-boxes constructed from 4-bit permutations using a 3-round Feistel cipher. Such S-boxes, however, turned out to have too many low-weight, high-probability characteristics that may cause weak diffusion by the linear transformation following the S-box transformation. For example, the S-boxes used in Version 0.5 have about 300 characteristics with probability 2^{-5} and 160 linear approximations with probability 2^{-4}. Furthermore, some of such I/O pairs turned out to have minimal diffusion under linear transformations. Though we could achieve reasonably high security bounds even with such S-boxes, we wanted to make CRYPTON more robust and stronger by allowing a large safety margin. We thus decided to redesign the S-boxes.

Experiments show that most Feistel-type constructions seem to generate S-boxes with too many high-probability characteristics, not much desirable as CRYPTON S-boxes. We thus decided to use other construction methods. We first started with a Feistel structure involving three or four 4-bit permutations and repeated modifications and testings of the structure to get better 8-bit S-boxes. In the end we arrived at the structure of SP networks shown in Figure 7. The resulting S-boxes are much stronger against differential and linear cryptanalysis when combined with linear transformations (though the hardware complexity for logic implementation of the S-box is almost doubled compared to the previous one). We decided to use four variants of one S-box, instead of independent four S-boxes, to allow greater flexibility in memory requirements (e.g., for cost-effective implementations on smart cards).

As we mentioned at the 1st AES candidate conference, we already had a plan to revise the CRYPTON key schedule. Unlike Feistel-type ciphers, CRYPTON needs a distinct decryption key schedule completely determined by, but requiring one more transformations for each round key than, an encryption key schedule. This complicates key scheduling for CRYPTON, since we want both key schedules almost equally efficient.

The previous key schedule was expected from the begining to have some minor weaknesses due to its too simple round key computations (actually a slight weakness was found by Serge Vaudenay etc. at ENS, as posted at the electronic forum for the AES). We thus made some enhancements to the key

schedule, while trying to keep changes minimal (otherwise we could have a better key schedule !). The new key schedule now makes use of byte-wise rotations, word-swappings as well as 32-bit rotations. We also used distinct round constants for each round key. This way we tried to make each byte of expanded keys be used in different locations (and different bit positions within a byte) of 4×4 byte array in different rounds. The new key schedule still runs much faster than one-block encryption.

Finally, we would like to stress that the above modifications will not require substantial changes in any existing analysis on the security and efficiency. We just enhanced security by replacing S-boxes, the most primitive element in most block ciphers. So, for security evaluations of the new version it suffices to replace old figures replated to S-box characteristics with new ones. There is no change in the overall structure of key scheduling. The performance figures in software implementations on large microprocessors remain the same. On a smart card, the performance will be a little degraded. The hardware complexity for VLSI implementations will be somewhat increased because of the increased complexity for logic implementation of S-boxes. All that should be changed is related to S-boxes (and key scheduling for some).

1.2 Symbols and Notation

Throughout this paper we will use the following symbols and notation:

- We write $A = (A[3], A[2], A[1], A[0])^t$ when the data variable A represent a 4×4 byte array, where $A[i]$ $(0 \le i \le 3)$ is a 4-byte word represented by $A[i] = a_{i3} \parallel a_{i2} \parallel a_{i1} \parallel a_{i0}$. Here $\parallel$ denote concatenation of two bit strings and the superscript t in a vector or array denotes transposition (hinged on the right top position). That is,

$$A = (A[3], A[2], A[1], A[0])^t = \begin{pmatrix} A[0] \\ A[1] \\ A[2] \\ A[3] \end{pmatrix} = \begin{pmatrix} a_{03} & a_{02} & a_{01} & a_{00} \\ a_{13} & a_{12} & a_{11} & a_{10} \\ a_{23} & a_{22} & a_{21} & a_{20} \\ a_{33} & a_{32} & a_{31} & a_{30} \end{pmatrix}$$

- $X^{\ll n}$ denotes left rotation of X by n-bit positions and $X^{\ll_b n}$ denotes byte-wise n-bit left rotation of a 32-bit number X (i.e., each byte in X is rotated to the left by n-bit positions).
- $f \circ g$ denotes composition of functions f and g, i.e., $(f \circ g)(x) = f(g(x))$.
- $\wedge, \vee, \oplus$: bit-wise logical operations for AND, OR and XOR (exclusive-or), respectively. $\overline{x}$ denotes the bit-wise complement of x.

2 Algorithm Specifications

2.1 Basic Building Blocks

2.1.1 Nonlinear Substitution γ

CRYPTON uses four 8×8 S-boxes, S_i $(0 \le i \le 3)$, for a nonlinear transformation. These S-boxes are in fact derived from one 8×8 involution S-box S (i.e., $S = S^{-1}$) and satisfy the inverse relationships $S_2 = S_0^{-1}$ and $S_3 = S_1^{-1}$ (see Sect.3.2 for details).

The S-box transformation γ consists of byte-wise substitutions on a 4×4 byte array. Two different transformations are used alternately in successive rounds: γ_o in odd rounds and γ_e in even rounds. They are depicted in Figure 2. Observe that the four S-boxes are arranged so that the following holds for any 4×4 byte array A:

$$\gamma_o(\gamma_e(A)) = \gamma_o(\gamma_e(A)) = A.$$

This property will be used to derive the identical process for encryption and decryption.

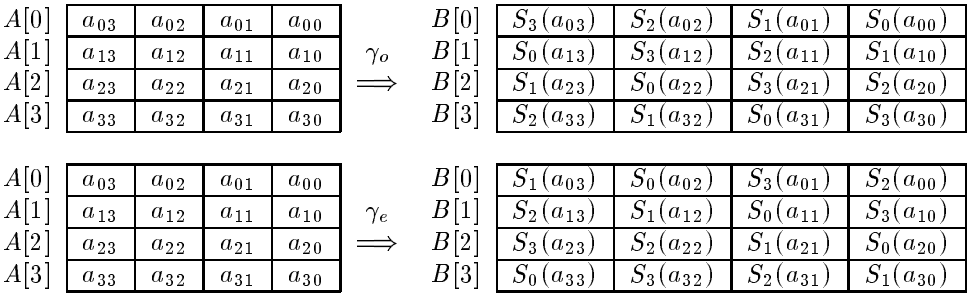


Figure 2: The S-box transformation γ

2.1.2 Linear Transformations π and τ

As linear transformations, CRYPTON uses a sequence of bit permutation and byte transposition. The bit permutation π mixes each byte column in 4×4 byte array and the byte transposition τ transposes the resulting byte columns into byte rows. Detailed diffusion property of $\tau \circ \pi$ will be presented in Sect.3.1.

Let us first define four masking vectors (M_3, M_2, M_1, M_0) for bit extraction used in π as

$$\begin{aligned} M_0 &= m_3 \| m_2 \| m_1 \| m_0 = \texttt{0x3fcff3fc}, \\ M_1 &= m_0 \| m_3 \| m_2 \| m_1 = \texttt{0xfc3fcff3}, \\ M_2 &= m_1 \| m_0 \| m_3 \| m_2 = \texttt{0xf3fc3fcf}, \\ M_3 &= m_2 \| m_1 \| m_0 \| m_3 = \texttt{0xcff3fc3f}, \end{aligned}$$

where $m_0 = \texttt{0xfc}, m_1 = \texttt{0xf3}, m_2 = \texttt{0xcf}, m_3 = \texttt{0x3f}$. We will use two slightly different versions of a bit permutation to make the encryption and decryption processes identical: π_o in odd rounds and π_e in even rounds. They are defined as follows:

- Bit permutation π_o for odd rounds: $B = \pi_o(A)$ defined by

$$\begin{aligned} B[0] &\leftarrow (A[3] \wedge M_3) \oplus (A[2] \wedge M_2) \oplus (A[1] \wedge M_1) \oplus (A[0] \wedge M_0), \\ B[1] &\leftarrow (A[3] \wedge M_0) \oplus (A[2] \wedge M_3) \oplus (A[1] \wedge M_2) \oplus (A[0] \wedge M_1), \\ B[2] &\leftarrow (A[3] \wedge M_1) \oplus (A[2] \wedge M_0) \oplus (A[1] \wedge M_3) \oplus (A[0] \wedge M_2), \\ B[3] &\leftarrow (A[3] \wedge M_2) \oplus (A[2] \wedge M_1) \oplus (A[1] \wedge M_0) \oplus (A[0] \wedge M_3). \end{aligned}$$

- Bit permutation π_e for even rounds: $B = \pi_e(A)$ defined by

$$\begin{aligned} B[0] &\leftarrow (A[3] \wedge M_1) \oplus (A[2] \wedge M_0) \oplus (A[1] \wedge M_3) \oplus (A[0] \wedge M_2), \\ B[1] &\leftarrow (A[3] \wedge M_2) \oplus (A[2] \wedge M_1) \oplus (A[1] \wedge M_0) \oplus (A[0] \wedge M_3), \\ B[2] &\leftarrow (A[3] \wedge M_3) \oplus (A[2] \wedge M_2) \oplus (A[1] \wedge M_1) \oplus (A[0] \wedge M_0), \\ B[3] &\leftarrow (A[3] \wedge M_0) \oplus (A[2] \wedge M_3) \oplus (A[1] \wedge M_2) \oplus (A[0] \wedge M_1). \end{aligned}$$

Note that $\pi_o^{-1} = \pi_o$ and $\pi_e^{-1} = \pi_e$ and that π_o and π_e can be implemented by the same function as follows:

$$\pi_e((A[3], A[2], A[1], A[0])^t) = \pi_o((A[1], A[0], A[3], A[2])^t).$$

.

The bit permutation π can be viewed as consisting of two operations: Each byte column is rearranged in such a way that each byte in the same byte column contributes two bits to each new byte and then the mod-2 sum of the four input bytes is exclusive-ored with each new byte to form the final output byte. For example, the expression for π_o can be rewritten as follows (see also Figure 3 for graphical view of π_o.):

$$\begin{aligned} T &= A[3] \oplus A[2] \oplus A[1] \oplus A[0], \\ B[0] &\leftarrow (A[3] \wedge MI_3) \oplus (A[2] \wedge MI_2) \oplus (A[1] \wedge MI_1) \oplus (A[0] \wedge MI_0) \oplus T, \end{aligned}$$

$$
\begin{aligned}
B[1] &\leftarrow (A[3] \wedge MI_0) \oplus (A[2] \wedge MI_3) \oplus (A[1] \wedge MI_2) \oplus (A[0] \wedge MI_1) \oplus T, \\
B[2] &\leftarrow (A[3] \wedge MI_1) \oplus (A[2] \wedge MI_0) \oplus (A[1] \wedge MI_3) \oplus (A[0] \wedge MI_2) \oplus T, \\
B[3] &\leftarrow (A[3] \wedge MI_2) \oplus (A[2] \wedge MI_1) \oplus (A[1] \wedge MI_0) \oplus (A[0] \wedge MI_3) \oplus T,
\end{aligned}
$$

where MI_i is the bit-wise complement of M_i.

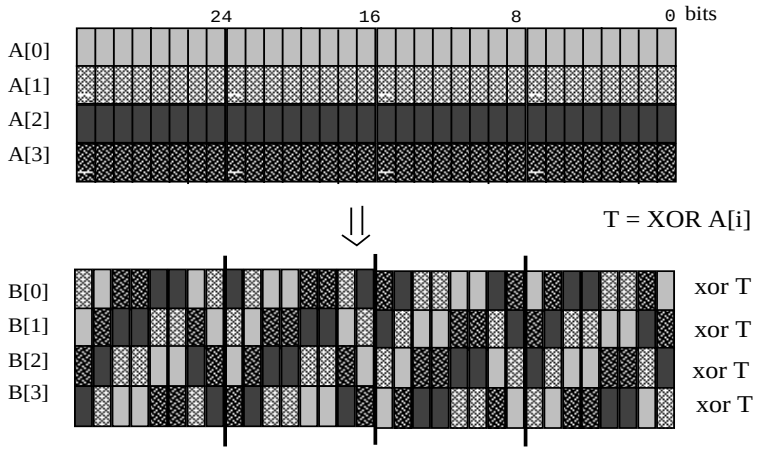


Figure 3: Graphical view of the bit permutation π_o

As shown above, the bit permutation π_o (π_e, resp.) in fact consists of four column-wise permutations. Let A^i be the i-th byte column of A, i.e., $A^i = (a_{3i}, a_{2i}, a_{1i}, a_{0i})^t$. And Let π_i be the bit permutation of the i-th column induced by π. Then we can rewrite π_o and π_e as

$$
\begin{aligned}
\pi_o(A) &= (\pi_3(A^3), \pi_2(A^2), \pi_1(A^1), \pi_0(A^0))^t, \\
\pi_e(A) &= (\pi_1(A^3), \pi_0(A^2), \pi_3(A^1), \pi_2(A^0))^t.
\end{aligned}
$$

The component column-permutation π_i can be easily derived from the description of π. For example, $B^0 = \pi_0(A^0)$ is given by

$$
\begin{aligned}
b_{00} &\leftarrow (a_{30} \wedge m_3) \oplus (a_{20} \wedge m_2) \oplus (a_{10} \wedge m_1) \oplus (a_{00} \wedge m_0), \\
b_{10} &\leftarrow (a_{30} \wedge m_0) \oplus (a_{20} \wedge m_3) \oplus (a_{10} \wedge m_2) \oplus (a_{00} \wedge m_1), \\
b_{20} &\leftarrow (a_{30} \wedge m_1) \oplus (a_{20} \wedge m_0) \oplus (a_{10} \wedge m_3) \oplus (a_{00} \wedge m_2), \\
b_{30} &\leftarrow (a_{30} \wedge m_2) \oplus (a_{20} \wedge m_1) \oplus (a_{10} \wedge m_0) \oplus (a_{00} \wedge m_3).
\end{aligned}
$$

It is also easy to see that if $\pi_0([d, c, b, a]^t) = [h, g, f, e]^t$, then we have

$$
\begin{aligned}
\pi_1([d, c, b, a]^t) &= [e, h, g, f]^t, \\
\pi_2([d, c, b, a]^t) &= [f, e, h, g]^t, \\
\pi_3([d, c, b, a]^t) &= [g, f, e, h]^t.
\end{aligned}
$$

The byte transposition τ simply rearranges a 4×4 byte array by moving the byte at the (i, j)-th position to the (j, i)-th position (see Figure 4 for $B = \tau(A)$). It is also involution, i.e., $\tau^{-1} = \tau$.

					τ					
$A[0]$	a_{03}	a_{02}	a_{01}	a_{00}		$B[0]$	a_{30}	a_{20}	a_{10}	a_{00}
$A[1]$	a_{13}	a_{12}	a_{11}	a_{10}		$B[1]$	a_{31}	a_{21}	a_{11}	a_{01}
$A[2]$	a_{23}	a_{22}	a_{21}	a_{20}	$\Longrightarrow$	$B[2]$	a_{32}	a_{22}	a_{12}	a_{02}
$A[3]$	a_{33}	a_{32}	a_{31}	a_{30}		$B[3]$	a_{33}	a_{23}	a_{13}	a_{03}

Figure 4: The byte transposition τ

2.1.3 Key Addition σ

Key mixing is performed by simply exclusive-oring round keys with data words. That is, for a round key $K = (K[3], K[2], K[1], K[0])^t$, $B = \sigma_K(A)$ is defined by $B[i] = A[i] \oplus K[i]$ for $i = 0, 1, 2, 3$. Obviously, $\sigma_K^{-1} = \sigma_K$.

2.1.4 Round Transformation ρ

One round of CRYPTON consists of applying in sequence S-box transformation, bit permutation, byte transposition and key addition. That is, the encryption round functions used for odd and even rounds are defined (for round key K) by

$$\begin{aligned} \rho_{oK} &= \sigma_K \circ \tau \circ \pi_o \circ \gamma_o \quad \text{for } r = 1, 3, \cdots \text{ etc.}, \\ \rho_{eK} &= \sigma_K \circ \tau \circ \pi_e \circ \gamma_e \quad \text{for } r = 2, 4, \cdots \text{ etc.} \end{aligned}$$

Note the inverse relationships of component functions. All component functions except for γ_o and γ_e are involutions. That is, $\gamma_o^{-1} = \gamma_e$ ($\gamma_e^{-1} = \gamma_o$), $\pi_o^{-1} = \pi_o$ ($\pi_e^{-1} = \pi_e$), $\tau^{-1} = \tau$ and $\sigma_K^{-1} = \sigma_K$. Therefore, the decryption round transformation can be obtained by

$$\begin{aligned} \rho_{oK}^{-1} &= \gamma_e \circ \pi_o \circ \tau \circ \sigma_K, \\ \rho_{eK}^{-1} &= \gamma_o \circ \pi_e \circ \tau \circ \sigma_K. \end{aligned}$$

With this decryption round we can decrypt the ciphertext by applying encryption round keys in reverse order. However, it would be better to be able to decrypt ciphertexts by using the same encryption process. This is possible in CRYPTON, as can be seen below. Of course, for this we have to generate decryption round keys using a different key schedule.

2.2 Encryption and Decryption

Let K_e^i be the i-th encryption round key consisting of 4 words, derived from a user-supplied key K using the encryption key schedule described later. The encryption transformation E_K of r-round CRYPTON under key K consists of an initial key addition and $r/2$ times repetitions of ρ_o and ρ_e and then a final output transformation (we assume r is even). More specifically, E_K can be described as

$$E_K = \phi_e \circ \rho_{eK_e^r} \circ \rho_{oK_e^{r-1}} \circ \cdots \circ \rho_{eK_e^2} \circ \rho_{oK_e^1} \circ \sigma_{K_e^0}, \tag{1}$$

where ϕ_e is the output transformation to make the encryption and decryption processes identical and is given by

$$\phi_e = \tau \circ \pi_e \circ \tau.$$

Similarly we define $\phi_o = \tau \circ \pi_o \circ \tau$. The corresponding decryption transformation D_K can be shown to have the same form as E_K, except for using suitably transformed round keys:

$$D_K = \phi_e \circ \rho_{eK_d^r} \circ \rho_{oK_d^{r-1}} \circ \cdots \circ \rho_{eK_d^2} \circ \rho_{oK_d^1} \circ \sigma_{K_d^0}, \tag{2}$$

where the decryption round keys are defined by

$$K_d^{r-i} = \begin{cases} \phi_e(K_e^i) & \text{for } i = 0, 2, 4, \cdots, \\ \phi_o(K_e^i) & \text{for } i = 1, 3, 5, \cdots. \end{cases} \tag{3}$$

Notice that

$$\begin{aligned} \phi_e \circ \sigma_{K_e^i} &= \sigma_{\phi_e(K_e^i)} \circ \phi_e = \sigma_{K_d^{r-i}} \circ \phi_e \quad \text{for } i = 0, 2, 4, \cdots, \\ \phi_o \circ \sigma_{K_e^i} &= \sigma_{\phi_o(K_e^i)} \circ \phi_0 = \sigma_{K_d^{r-i}} \circ \phi_o \quad \text{for } i = 1, 3, 5, \cdots. \end{aligned}$$

Using this property, we can incorporate the output transformation ϕ_e into the final round as follows:

$$\begin{aligned} \phi_e \circ \rho_{eK_e^r} &= \phi_e \circ \sigma_{K_e^r} \circ \tau \circ \pi_e \circ \gamma_e \\ &= \sigma_{K_d^0} \circ \phi_e \circ \tau \circ \pi_e \circ \gamma_e \\ &= \sigma_{K_d^0} \circ \tau \circ \gamma_e. \end{aligned}$$

Also note that $\tau \circ \gamma = \gamma \circ \tau$.

We next explain how the decryption process is derived. For simplicity, we consider a 2-round version of CRYPTON ($r = 2$). The encryption process of this two round version can be rewritten as:

$$\begin{aligned}
E_K &= \phi_e \circ \rho_{eK_e^2} \circ \rho_{oK_e^1} \circ \sigma_{K_e^0} \\
&= \phi_e \circ (\phi_e \circ \sigma_{K_d^0} \circ \phi_e \circ \tau \circ \pi_e \circ \gamma_e) \circ (\phi_o \circ \sigma_{K_d^1} \circ \phi_o \circ \tau \circ \pi_o \circ \gamma_o) \circ \sigma_{K_e^0} \\
&= (\sigma_{K_d^0} \circ \tau \circ \gamma_e) \circ (\phi_o \circ \sigma_{K_d^1} \circ \tau \circ \gamma_o) \circ \sigma_{K_e^0} \\
&= \sigma_{K_d^0} \circ (\gamma_e \circ \pi_o \circ \tau \circ \sigma_{K_d^1}) \circ (\gamma_o \circ \tau \circ \sigma_{K_e^0}) \\
&= \sigma_{K_d^0} \circ (\sigma_{K_d^1} \circ \tau \circ \pi_o \circ \gamma_o)^{-1} \circ (\sigma_{K_e^0} \circ \tau \circ \gamma_e)^{-1} \\
&= (\sigma_{K_d^0})^{-1} \circ (\rho_{oK_d^1})^{-1} \circ (\phi_e \circ \rho_{eK_d^2})^{-1}.
\end{aligned}$$

Therefore, the decryption process of the reduced version is given by

$$D_K = E_K^{-1} = \phi_e \circ \rho_{eK_d^2} \circ \rho_{oK_d^1} \circ \sigma_{K_d^0}.$$

This shows that encryption and decryption can be performed by the same code (logic) if different round keys are applied.

2.3 Key Scheduling

R-round CRYPTON requires total $4(r+1)$ round keys each of which is 32 bits long. These round keys are generated from a user key of $8k$ ($k = 0, 1, \cdots, 32$) bits in two steps: first nonlinear-transform the user key into 8 expanded keys and then generate the required number of round keys from these expanded keys using simple operations. This two-step generation of round keys is to allow efficient on-the-fly round key computation in the case where storage requirements do not allow to store the whole round keys (e.g., implementation in a portable device with restricted resources). It also facilitates hardware implementations.

2.3.1 Generating Expanded Keys

Let $K = k_{u-1} \cdots k_1 k_0$ be a user key of u bytes ($u = 0, 1, \cdots, 32$). We assume that K is 256 bits long (by prepending as many zeros as required).

1. split the user key into U and V as: for $i = 0, 1, 2, 3$,

$$U[i] = k_{8i+6} k_{8i+4} k_{8i+2} k_{8i}, \quad V[i] = k_{8i+7} k_{8i+5} k_{8i+3} k_{8i+1}.$$

2. nonlinear-transform U and V using round transformations ρ_o and ρ_e, respectively, with all-zero round keys:

$$U' = \rho_o(U), \quad V' = \rho_e(V).$$

3. compute 8 expanded keys $E_e[i]$ for encryption as: for $i = 0, 1, 2, 3$,

$$E_e[i] = U'[i] \oplus T_1, \quad E_e[i+4] = V'[i] \oplus T_0,$$

where $T_0 = \bigoplus_{i=0}^{3} U'[i]$ and $T_1 = \bigoplus_{i=0}^{3} V'[i]$.

2.3.2 Generating encryption round keys

The following 13 round-constants will be used for encryption key schedule:

$$C_e[0] = \texttt{0xa54ff53a}, \quad C_e[i] = C_e[i-1] + \texttt{0x3c6ef372} \bmod 2^{32} \quad \text{for } i = 1, 2, \cdots, 12,$$

where the two constants, `0xa54ff53a` and `0x3c6ef372`, were obtained from the fractional parts of $\sqrt{7}$ and $\sqrt{5}$, respectively. In addition, the following 4 masking constants will be used to generate distinct constants for each round key from a given round constant:

$$MC_0 = \texttt{0xacacacac}, \quad MC_i = MC_{i-1}^{\lll_b 1} \text{ for } i = 1, 2, 3.$$

1. compute the round keys for the first 2 rounds as

$$\begin{aligned} K_e[i] &\leftarrow E_e[i] \oplus C_e[0] \oplus MC_i, \\ K_e[i+4] &\leftarrow E_e[i+4] \oplus C_e[1] \oplus MC_i \ \text{ for } 0 \le i \le 3. \end{aligned}$$

2. for rounds $r = 2, 3, \cdots$, repeat the following two steps alternately:

   2-1. for even rounds, update the first 4 expanded keys as

$$\{E_e[3], E_e[2], E_e[1], E_e[0]\} \leftarrow \{E_e[0]^{\lll_b 6}, E_e[3]^{\lll_b 6}, E_e[2]^{\lll 16}, E_e[1]^{\lll 24}\}$$

   and compute the round keys for round r as

$$K_e[4r+i] \leftarrow E_e[i] \oplus C_e[r] \oplus MC_i \ \text{ for } 0 \le i \le 3.$$

   2-2. for odd rounds, update the second 4 expanded keys as

$$\{E_e[7], E_e[6], E_e[5], E_e[4]\} \leftarrow \{E_e[6]^{\lll 16}, E_e[5]^{\lll 8}, E_e[4]^{\lll_b 2}, E_e[7]^{\lll_b 2}\}$$

   and compute the round keys for round r as

$$K_e[4r+i] \leftarrow E_e[i+4] \oplus C_e[r] \oplus MC_i \ \text{ for } 0 \le i \le 3.$$

2.3.3 Generating decryption round keys

For efficient decryption key schedule, we first observe that the transformations $\phi_o = \tau \circ \pi_o \circ \tau$ and $\phi_e = \tau \circ \pi_e \circ \tau$ are actually row-wise bit permutations and can be rewritten as

$$\begin{aligned} \phi_o(A) &= (\phi_3(A[3]), \phi_2(A[2]), \phi_1(A[1]), \phi_0(A[0]))^t, \\ \phi_e(A) &= (\phi_1(A[3]), \phi_0(A[2]), \phi_3(A[1]), \phi_2(A[0]))^t, \end{aligned}$$

where the component transformation ϕ_i is actually the same as π_i, except that 4 input bytes are now arranged in row vector (see Sect.2.1.2). Also note the shift property of ϕ_i

$$\begin{aligned} \phi_i(X^{\lll 8k}) &= \phi_i(X)^{\lll 32-8k} \ \text{ for } k = 1, 2, 3, \\ \phi_i(X) &= \phi_j(X)^{\lll 8} \ \text{ for } j = i+1 \bmod 4, \\ \phi_i(X^{\lll_b 2k}) &= (\phi_i(X)^{\lll_b 2k})^{\lll 8k} \ \text{ for } k = 1, 2, 3, \end{aligned}$$

and the linear property under exclusive-oring

$$\phi_i(A[j] \oplus C) = \phi_i(A[j]) \oplus \phi_i(C).$$

In particular, $\phi_i(C) = C$ if C consists of 4 identical bytes. Using these properties, we can design a decryption key schedule similar to and almost as efficient as the encryption key schedule as follows:

1. compute the expanded keys and round constants for decryption as follows:

$$\begin{aligned} \{E_d[3], E_d[2], E_d[1], E_d[0]\} &\leftarrow \{\phi_0(E_e[1])^{\lll_b 2}, \phi_1(E_e[0]), \phi_1(E_e[3])^{\lll_b 2}, \phi_2(E_e[2])^{\lll_b 4}\}, \\ \{E_d[7], E_d[6], E_d[5], E_d[4]\} &\leftarrow \{\phi_2(E_e[6])^{\lll_b 4}, \phi_0(E_e[5])^{\lll_b 4}, \phi_1(E_e[4])^{\lll_b 6}, \phi_0(E_e[7])^{\lll_b 6}\}, \\ C_d[i] &\leftarrow \phi_2(C_e[12-i]) \ \text{ for } i = 0, 2, \cdots, 12 \ (\text{even } i\text{'s}), \\ C_d[i] &\leftarrow \phi_0(C_e[12-i]) \ \text{ for } i = 1, 3, \cdots, 11 \ (\text{odd } i\text{'s}). \end{aligned}$$

2. compute the first 8 round keys as

$$\begin{aligned} K_d[i] &\leftarrow E_d[i] \oplus C_d[0]^{\lll 32-8i} \oplus MC_i, \\ K_d[i+4] &\leftarrow E_d[i+4] \oplus C_d[1]^{\lll 32-8i} \oplus MC_i \ \text{ for } 0 \le i \le 3, \end{aligned}$$

   where rotation amounts should be interpreted as integers mod 32.

3. for rounds $r = 2, 3, \cdots$, repeat the following two steps alternately:

   3-1. for even rounds, update the first 4 expanded keys as

$$\{E_d[3], E_d[2], E_d[1], E_d[0]\} \leftarrow \{E_d[2]^{\lll_b 2}, E_d[1]^{\lll 8}, E_d[0]^{\lll 16}, E_d[3]^{\lll_b 2}\}$$

   and compute the round keys for round r as

$$K_d[4r+i] \leftarrow E_d[i] \oplus C_d[r]^{\lll 32-8i} \oplus MC_i \quad \text{for } 0 \le i \le 3.$$

   3-2. for odd rounds, update the second 4 expanded keys as

$$\{E_d[7], E_d[6], E_d[5], E_d[4]\} \leftarrow \{E_d[4]^{\lll_b 6}, E_d[7]^{\lll 24}, E_d[6]^{\lll 16}, E_d[5]^{\lll_b 6}\}$$

   and compute the round keys for round r as

$$K_d[4r+i] \leftarrow E_d[i+4] \oplus C_d[r]^{\lll 32-8i} \oplus MC_i \quad \text{for } 0 \le i \le 3.$$

Table 6 in Appendix shows encryption and decryption round keys expressed in terms of expanded keys. This table may be useful for checking various key schedule weaknesses.

3 Security Analysis

We first investigate the diffusion property of linear transformations and the differential and linear characteristics of S-boxes together with their construction method. We then analyze the security of CRYPTON against various possible attacks. This analysis shows that the complexity of differential and linear attacks on 9-round CRYPTON would require much more ciphertexts than are available. We thus propose to use 12 rounds for CRYPTON, with a safety margin of three more rounds.

3.1 Diffusion Property of Linear Transformations

Due to memory requirements, small size S-boxes are commonly used in most block cipher designs and thus the diffusion of S-box outputs by linear transformations plays an important role for resistance against various attacks such as the differential and linear attacks.

From Sect.2.1.2, we can see that it suffices to consider any one component transformation π_i of π to examine the diffusion property of π, since π acts on each byte column independently. Consider π_0, for example. It is easy to see that any column vector with n ($n < 4$) nonzero bytes is transformed by π_0 into a column vector with at least $4 - n$ nonzero bytes (we call this number 4 the diffusion order of π_0). This is due to the operation of exclusive-or sum in π. More important is that the number of such input vectors giving minimal diffusion is very limited. This is due to the masked bit permutation. Table 1 shows the distribution of diffusion orders by π_i over all 32-bit numbers. We can see that there are only 204 values achieving the minimum diffusion order 4 and about 99.96 % of 32-bit numbers have diffusion order 7 or 8. This shows the effectiveness of diffusion by our combined linear transformation $\tau \circ \pi$.

diffusion order	4	5	6	7	8
no. elements	204	13464	1793364	130589784	4162570479
ratio	4.75×10^{-8}	3.13×10^{-6}	4.18×10^{-4}	3.04×10^{-2}	96.92×10^{-2}

Table 1: Distribution of diffusion orders under π_i

Let us examine in more detail the set of 32-bit numbers giving minimal diffusion. For this, we define two sets of byte values, Ω_x and Ω_y, as

$$\begin{aligned}
\Omega_x &= \{0x01, 0x02, 0x03, 0x04, 0x08, 0x0c, 0x10, 0x20, 0x30, 0x40, 0x80, 0xc0\}, \\
\Omega_y &= \{0x11, 0x12, 0x13, 0x21, 0x22, 0x23, 0x31, 0x32, 0x33, 0x44, 0x48, 0x4c, \\
&\quad 0x84, 0x88, 0x8c, 0xc4, 0xc8, 0xcc\} \cup \Omega_x.
\end{aligned}$$

Let I_{jk} be a set of input vectors with j nonzero bytes which are transformed by π_i into output vectors with $4-j$ nonzero bytes (k differentiates the position of nonzero byte(s)). Then all possible 32-bit values with minimum diffusion order 4 can be obtained as follows: for each x in Ω_x and y in Ω_y,

$$I_{10} = \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad I_{11} = \begin{bmatrix} 0 \\ x \\ 0 \\ 0 \end{bmatrix}, \quad I_{12} = \begin{bmatrix} 0 \\ 0 \\ x \\ 0 \end{bmatrix}, \quad I_{13} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x \end{bmatrix},$$

$$I_{20} = \begin{bmatrix} x \\ x \\ 0 \\ 0 \end{bmatrix}, \quad I_{21} = \begin{bmatrix} 0 \\ x \\ x \\ 0 \end{bmatrix}, \quad I_{22} = \begin{bmatrix} 0 \\ 0 \\ x \\ x \end{bmatrix}, \quad I_{23} = \begin{bmatrix} x \\ 0 \\ 0 \\ x \end{bmatrix}, \quad I_{24} = \begin{bmatrix} y \\ 0 \\ y \\ 0 \end{bmatrix}, \quad I_{25} = \begin{bmatrix} 0 \\ y \\ 0 \\ y \end{bmatrix},$$

$$I_{30} = \begin{bmatrix} x \\ x \\ x \\ 0 \end{bmatrix}, \quad I_{31} = \begin{bmatrix} x \\ x \\ 0 \\ x \end{bmatrix}, \quad I_{32} = \begin{bmatrix} x \\ 0 \\ x \\ x \end{bmatrix}, \quad I_{33} = \begin{bmatrix} 0 \\ x \\ x \\ x \end{bmatrix}.$$

Therefore, we can see that there are only 204 vectors with minimum diffusion:

$$\begin{aligned} \pi_i(I_{1j}) &= I_{3k} \quad \text{for } 0 \le j,k \le 3 : \ 4 \times 12 = 48, \\ \pi_i(I_{2j}) &= I_{2k} \quad \text{for } 0 \le j,k \le 3 : \ 4 \times 12 = 48, \\ \pi_i(I_{2j}) &= I_{2k} \quad \text{for } 4 \le j,k \le 5 : \ 2 \times 30 = 60, \\ \pi_i(I_{3j}) &= I_{1k} \quad \text{for } 0 \le j,k \le 3 : \ 4 \times 12 = 48. \end{aligned}$$

Observe that the nonzero bytes in each input vector should have the same value to achieve minimum diffusion. Also note that the 18 values in $\Omega_y - \Omega_x$ can only occur in the two sets I_{24} and I_{25}.

Now let us examine the diffusion effect of $\tau \circ \pi$ through consecutive rounds. This analysis can be done by assuming that in each round the S-box output can take any desired value, irrespective of the input value. This assumption is to maximally take into account the probabilistic nature of the S-box transformation without details of the S-box characteristics. Since it suffices to consider worst-case propagations, we only examine the inputs with 1, 2, or 3 nonzero bytes in any one column vector of a 4×4 byte array, say the first byte column. The result is shown in Table 2, where we only showed the nonzero column vector in the starting 4×4 byte array. The sum of the number of nonzero bytes throughout the evolution is of great importance to ensure resistance against differential and linear cryptanalysis. Table 2 shows that the number of nonzero bytes per round is repeated with period 4 and their sum up to round 8 is at least 32.

starting nonzero vector \ round	1	2	3	4	5	6	7	8
I_{1j} $(0 \le j \le 3)$	1	3	9	3	1	3	9	3
I_{2j} $(0 \le j \le 5)$	2	2	6	6	2	2	6	6
I_{3j} $(0 \le j \le 3)$	3	1	3	9	3	1	3	9

Table 2: Minimum possible no. of nonzero bytes in S-box inputs in successive rounds

3.2 S-boxes Construction and their Property

The S-box for a block cipher should be chosen to have two important requirements: differential uniformity and nonlinearity. Combined with the diffusion effect of linear transformations used, they directly determine the security level of the block cipher against differential and linear cryptanalysis (DC and LC, for short) [1, 17].

The maximum differential and linear approximation probabilities for an $n \times n$ S-box S (δ_S and λ_S for short) can be defined as follows. Let X and Y be a set of possible 2^n inputs/outputs of S, respectively.

Then δ_S and λ_S are defined by

$$\delta_S \stackrel{\text{def}}{=} \max_{\Delta x \neq 0, \Delta y} \frac{\#\{x \in X | S(x) \oplus S(x \oplus \Delta x) = \Delta y\}}{2^n}, \tag{4}$$

$$\lambda_S \stackrel{\text{def}}{=} \max_{\Gamma x, \Gamma y \neq 0} \left(\frac{|\#\{x \in X | x \bullet \Gamma x = S(x) \bullet \Gamma y\} - 2^{n-1}|}{2^{n-1}} \right)^2, \tag{5}$$

where $a \bullet b$ denotes the parity of bit-wise product of a and b.

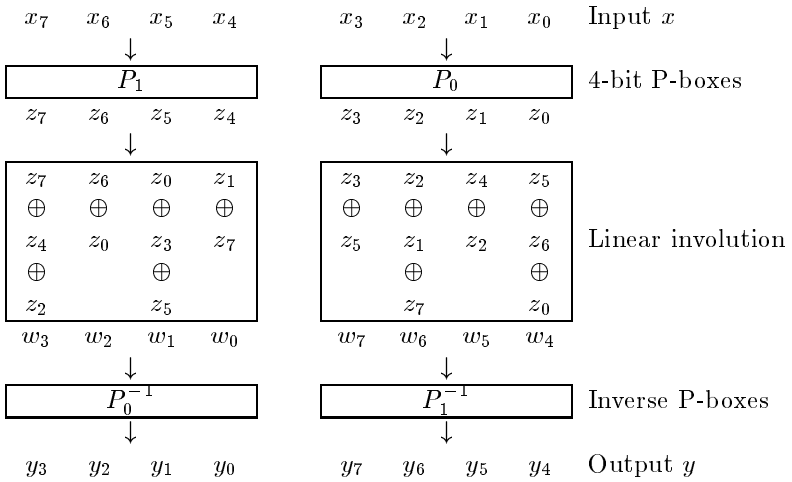
The nonlinear transformation adopted in CRYPTON is byte-wise substitutions using four 8×8 S-boxes, S_i $(i = 0, 1, 2, 3)$. We first constructed an 8×8 involution S-box S from two 4-bit permutations (P-boxes, for short), P_0 and P_1, using a SP network, as shown in Figure 5. Then the actual four S-boxes were derived from S as follows (see Figure 6): for each $x \in [0, 256)$,

$$\begin{aligned} S_0(x) &= S(x)^{\ll 1}, \\ S_1(x) &= S(x)^{\ll 3}, \\ S_2(x) &= S(x^{\gg 1}), \\ S_3(x) &= S(x^{\gg 3}). \end{aligned}$$

It is easy to see that these S-boxes satisfy inverse relationships such that for any 8-bit number x,

$$\begin{aligned} S_0(S_2(x)) &= S_2(S_0(x)) = x, \\ S_1(S_3(x)) &= S_3(S_1(x)) = x. \end{aligned}$$

We decided to use four variants of S, rather than just one involution S-box or four independent S-boxes, because this will make iterative characteristics harder to occur while reducing the storage requirement for limited environments such as smart card implementations.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P_0	15	14	10	1	11	5	8	13	9	3	2	7	0	6	4	12
P_1	11	10	13	7	8	14	0	5	15	6	3	4	1	9	2	12
P_0^{-1}	12	3	10	9	14	5	13	11	6	8	2	4	15	7	1	0
P_1^{-1}	6	12	14	10	11	7	9	3	4	13	1	0	15	2	5	8

Figure 5: The selected 8×8 involution S-box S

To find a good 8-bit involution S-box, we tried with various invertible structures of generating 8-bit S-boxes from three or four 4-bit P-boxes. Starting with a Feistel structure, we partially restructured it in step-by-step and tested the resulting 8-bit S-boxes for the following requirements:

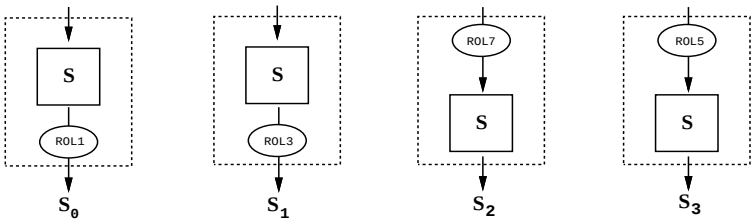


Figure 6: Construction of four 8×8 S-boxes S_i $(i = 0, 1, 2, 3)$ from S

1. The 8-bit involution S-box should have best possible differential and linear characteristics.
2. The high-probability I/O difference pairs (selection patterns, resp.) in S should have as high Hamming weights as possible.
3. The number of high-probability difference pairs (selection patterns, resp.) in the resulting 8×8 S-boxes S_i's should be as small as possible when the input is restricted to the minimal diffusion set Ω_y.

The last two requirements are to ensure that high-probability difference values should be more rapidly diffused by the linear transformation and that it should be more difficult to form a chain of high-probability S-box characteristics/linear approximations through consecutive rounds.

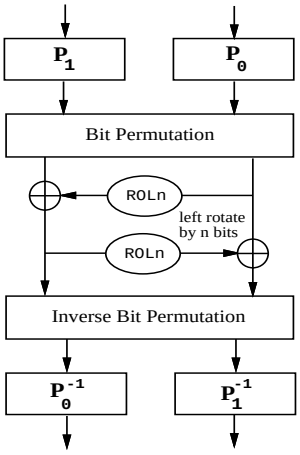


Figure 7: The selected search model for 8×8 involution S-box S

Figure 7 shows the S-box search model we arrived at in the end. Based on this structure and the above requirements, the final involution S-box S was searched for over some limited space of good 4-bit P-boxes and linear involutions. The four 8×8 S-boxes constructed as above are presented in Appendix (see Table 7). Table 3 shows their statistics on the distribution of input-output difference/linear approximation pairs, where the entry value is the number computed by the numerator of equation (4) (equation (5) in the case of linear approximations).

From the table, we can see that for each i,

$$p_d \stackrel{\text{def}}{=} \delta_{S_i} = \frac{10}{256} = 2^{-4.68}, \tag{6}$$

$$p_l \stackrel{\text{def}}{=} \lambda_{S_i} = (\frac{32}{128})^2 = 2^{-4}, \tag{7}$$

and that there are only 7 difference pairs achieving the best characteristic probability p_d (6 selection patterns achieving the best linear approximation probability p_l). As we aimed at the S-box selection

Difference distribution									
entry value	0	2	4	6	8	10			
no of entries	39584	20158	4976	749	62	7			
Linear approx. distribution									
entry value	0	4	8	12	16	20	24	28	32
no of entries	13927	22058	15948	8460	3731	1094	276	36	6

Table 3: Distribution of nonzero entries in the difference/linear approx. tables of S_i

process, these high-probability characteristics have fairly heavy Hamming weights. E.g., for the characteristics with top two high probabilities (69 pairs for DC and 42 pairs for LC), the sum of input and output Hamming weights are at least 4 and larger than 8 on average (see Table 8 in Appendix)

More importantly, if the input is restricted to the minimal diffusion set Ω_y, the maximum entry values are at most 6 and 24 for differential and linear characteristics, respectively. There are only 4 such difference pairs and 2 such selection patterns in each S-boxes, as shown in Table 4. Note that even the pairs in Table 4 belong to the more restricted set $\Omega_y - \Omega_x$. Since they are more important for worst-case analysis of DC and LC, we define these probabilities as

$$p'_d \stackrel{\text{def}}{=} \delta_{S_i}^{\Omega_y} = \frac{6}{256} = 2^{-5.42}, \tag{8}$$

$$p'_l \stackrel{\text{def}}{=} \lambda_{S_i}^{\Omega_y} = (\frac{24}{128})^2 = 2^{-4.83}. \tag{9}$$

S_0	DC (6)	(11, c0) (22, 8c) (32, cc) (88, 11)
	LC (24)	(88, 11)
S_1	DC (6)	(11, 3) (22, 32) (32, 33) (88, 44)
	LC (24)	(88, 44)
S_2	DC (6)	(c0, 11) (11, 88) (8c, 22) (cc, 32)
	LC (24)	(11, 88)
S_3	DC (6)	(3, 11) (32, 22) (33, 32) (44, 88)
	LC (24)	(44, 88)

Table 4: The most probable characteristics over the restricted set Ω_y

3.3 Differential Cryptanalysis

Let us first evaluate the best r-round characteristic probability for CRYPTON. In the following we only consider characteristics up to 8 rounds since that will be sufficient to show the resistance of CRYPTON against DC.

First note that the probability of any characteristic in CRYPTON can be completely determined by the number of active S-boxes and their characteristic probabilities (e.g., see [5]). Since the number of active S-boxes involved in any 8-round characteristic is at least 32, we can obtain the most rough upper bound for the best 8-round characteristic probability as $p_{C_8} = p_d^{32} = 2^{-149.7}$ under the assumption of independent and uniform distribution for plaintexts and round keys, where we assumed that all the S-boxes involved have the best characteristic probability p_d.

However, as can be seen from Table 4, the minimum number of active S-boxes shown in Table 2 can not be achieved even with S-box characteristics with probability p'_d. Moreover, if we allow intermediate S-box output differences with larger diffusion orders, then the number of active S-boxes up to round 8 will grow much larger than the bound 32. Considering the rapid diffusion by linear transformations, we can reasonably assume that a characteristic involving a smaller number of active S-boxes with smaller S-box characteristic probabilities should give better overall probability than a characteristic involving a

larger number of active S-boxes with larger S-box characteristic probabilities. Therefore, we can obtain a tighter bound for the 8-round characteristic probability as

$$p_{C_8} < (p'_d)^{32} = 2^{-173.3}.$$

Of course, the actual probability will be much lower than this bound, but we do not proceed any more since this bound is enough to show the strong resistance of CRYPTON against differential cryptanalysis based on the best characteristic.

Given a pair of input and output differences, there may be a relatively large number of characteristics starting with the input difference and ending with the output difference. With a little close examination of the diffusion property of linear transformations, we can obtain a very loose bound on the number of possible characteristics that can reside in any differential achieving the smallest number of active S-boxes. The most rough bound can be obtained by assuming that each S-box can take any desired output difference for a given input difference. The bound obtained under this assumption is $15^8 \times 4^2 \times 3^6 \times 2^3 \approx 2^{47.8}$. In fact, the actual number will be only a very small fraction of this bound. Even if we assume the existence of such number of best characteristics, the best 8-round differential probability is sufficiently small, i.e., $p_{D_8} < 2^{-125.5}$.

The above rough analysis shows that we can get strong enough resistance against DC only with 9 rounds. Though it is assumed in this analysis that the plaintexts and round keys are independent and uniformly random, which does not hold in practice, such an analysis has been found to provide a reasonable estimation on the security against DC.

3.4 Linear Cryptanalysis

R-round linear approximations involve a number of S-box linear approximations and, as in differential cryptanalysis, the number of such S-boxes (i.e., active S-boxes) determines the complexity of linear cryptanalysis. Much the same way as DC, we can obtain a rough bound for the best 8-round linear approximation probability p_{L_r} as

$$p_{L_r} < (p'_l)^{32} = 2^{-154.6}.$$

Again this value is a very loose upper bound. Actually there will be no linear approximation achieving this probability, considering the linear characteristic of S-boxes and the linear transformation involved.

As in the differential attack, we may use multiple linear approximations to improve the basic linear attack [13, 14]. Suppose that one can derive N linear approximations involving the same key bits with the same probability. Then the complexity of a linear attack can be reduced by a factor of N, compared to a linear attack based on a single linear approximation [13]. However, a large number of linear approximations involving the same key bits are unlikely to be found in most ciphers, in particular in CRYPTON. Multiple linear approximations involving different key bits may be used to derive the different key bits in the different linear approximations simultaneously with almost the same complexity [14]. However, this will be of little help to improve the basic linear attack, since we already have a linear approximation probability far beyond any practical attack. Therefore, we believe that there will be no linear attack on CRYPTON with 9 or more rounds with a complexity lower than 2^{128}.

3.5 Variants/Extensions of DC and LC

There are some variants to the basic differential attack discussed above. Knudsen introduced the idea of *truncated differentials* [10]. A truncated (or partial) differential is a differential that predicts only part of the difference (not the entire value of difference). The existence of good truncated differentials and their usefulness depend on specific cipher algorithms. Knudsen demonstrated that this variant of DC may be more effective against some kind of ciphers than the basic differential attack and may be independent of the S-boxes used [12]. The effective and fairly uniform diffusion by linear permutations in CRYPTON shows that truncated differentials will not be much useful for differential cryptanalysis of CRYPTON compared to ordinary differentials.

Another variant of DC is the *higher order differential* attack considered by Lai [16]. This variant is also quite effective for some ciphers [10, 6]. Let d be the polynomial degree of $(r-1)$-round output bits expressed as polynomials of plaintext bits. Then the higher order differential attack allows us to find some key bits of the last round key for an r-round cipher using about 2^{d+1} chosen plaintexts [6].

Obviously the success of this attack depends on the nonlinear order of S-box outputs. Since CRYPTON uses S-boxes with nonlinear order 6, the polynomial degree of output bits after 3 rounds increases to $6^3 \gg 128$. Therefore, the higher order differential attack on CRYPTON will be completely infeasible after 4 rounds.

There are also several variants or generalizations of linear cryptanalysis. These include linear cryptanalysis using non-linear approximations [15], generalized linear cryptanalysis using I/O sums [3], and partitioning cryptanalysis [4], etc. We have not checked in detail the effectiveness of these attacks against CRYPTON. However, our observation on the diffusion property of π shows that any kind of I/O relations involving more than two bits in S-boxes should rapidly increase the number of active S-boxes involved in the overall I/O relations. So, we believe that there will be little chance of these attacks substantially improving the basic linear attack.

3.6 Security against Other Possible Attacks

Another notable attack is the *interpolation attack* [6] proposed by Jacobsen and Knudsen. It is a kind of algebraic attacks on block ciphers. This attack exploits the fact that the ciphertext can be expressed as a polynomial of the plaintext with a fixed number (say n) of unknown coefficients and thus the encryption polynomial can be reconstructed given n pairs of plaintexts/ciphertexts encrypted under a fixed key K. This polynomial should then be equivalent to an encryption algorithm under the key K. Clearly the complexity of this attack depends upon the number of S-boxes applied throughout encryption and/or the polynomial degree of S-box outputs. The S-boxes used in CRYPTON do not allow any simple algebraic description. Furthermore, the bit permutation π in each round destroys any potential algebraic structure through the bit-wise mixing of the S-box outputs and encryption involves a large enough number of S-box applications. Therefore, we believe that CRYPTON also provides strong resistance against algebraic cryptanalysis, such as the interpolation attack.

There are some kinds of non-cryptographic, implementation-dependent attacks on cryptosystems. These include the timing attack [11] and other side-channel cryptanalysis [9], such as differential fault analysis and differential power analysis, etc. The timing attack is hard to apply to CRYPTON, since each processing steps in CRYPTON involves the same kind of operations up to byte levels. Due to the same reason, we believe that CRYPTON is more reliable against side-channel cryptanalysis than most other ciphers.

3.7 Key Schedule Cryptanalysis

Key schedule cryptanalysis is another important category of attacks on block ciphers. Typical weaknesses exploited in key schedule cryptanalysis include weak keys or semi-weak keys, key collisions (equivalent keys), linear factors, simple relations such as the complemetation property existing in DES, etc. (for details, see e.g. [7, 8]). These weaknesses can be exploited to speed up an exhaustive key search or to mount related key attacks. Though most of these attacks on key schedules are not practical in normal use, they may be a serious flaw in certain circumstances (e.g., when used as a basic building block for hash functions).

The key schedule for CRYPTON is designed with the above known weaknesses in mind. We can easily check that no weak keys exist in the CRYPTON key schedule (mainly due to the use of different round constants in each round). Semi-weak keys are a pair of keys (K, K^*) such that $E_{K^*}(E_K(X)) = X$. That is, semi-weak keys exist if round keys produced by the forward key schedule is identical in sequence to those produced by the reverse key schedule. There cannot exist such semi-weak keys in the CRYPTON key schedule due to the complete asymmetry of the forward and backward directions in rotation amounts and round constants.

The CRYPTON key schedule ensures that equivalent keys exist only if different user keys produce the same expanded keys. However, expanded keys are derived from a user key via invertible transformations and thus no different user keys can produce the same expanded keys. This guarantees that there are no equivalent keys in CRYPTON

There is no complementation property either, since both key expansion and encryption processes involve parallel nonlinear substitutions. The same reason ensures that there will be no other simple relations between different user keys.

We also believe that there are no related keys that can be used to mount related-key differential attacks or related-key slide attacks. First, a user-supplied key is transformed into expanded keys in such a way that any controlled change in the user key should result in at least one-byte change in the expanded keys. Second, the same 8 expanded keys are used 6 or 7 times throughout encryption, each time the expanded keys being updated by rotations and swappings. Third, distinct round constants are used to generate each round key. Finally, parallel nonlinear substitutions in each round make it difficult for an attacker to exploit any controlled change in expanded keys.

4 Implementation and Efficiency

The overall structure of CRYPTON allows a very high degree of parallelisms. This enables very high efficiency and flexibility in both software and hardware implementations. Below we only consider implementation and efficiency aspects in typical computing environments, but CRYPTON is equally well-suited and efficient for use on most other platforms, such as 16-bit and 64-bit microprocessors and digital signal processors.

4.1 Implementation on 32-bit Microprocessors

The round transformation of CRYPTON can be efficiently implemented on a 32-bit microprocessor using table lookups, if we use 4 KBytes of storage in addition. The idea is to precompute and store 4 tables of 256 words as follows: for $0 \le j \le 255$,

$$\begin{aligned}
SS_0[j] &= S_0[j] \wedge m_3 \parallel S_0[j] \wedge m_2 \parallel S_0[j] \wedge m_1 \parallel S_0[j] \wedge m_0, \\
SS_1[j] &= S_1[j] \wedge m_0 \parallel S_1[j] \wedge m_3 \parallel S_1[j] \wedge m_2 \parallel S_1[j] \wedge m_1, \\
SS_2[j] &= S_2[j] \wedge m_1 \parallel S_2[j] \wedge m_0 \parallel S_2[j] \wedge m_3 \parallel S_2[j] \wedge m_2, \\
SS_3[j] &= S_3[j] \wedge m_2 \parallel S_3[j] \wedge m_1 \parallel S_3[j] \wedge m_0 \parallel S_3[j] \wedge m_3.
\end{aligned}$$

Then it is easy to see that the odd round function $B = \rho_{oK}(A)$ can be implemented by

$$\begin{aligned}
B_0 &= SS_3[a_{30}] \oplus SS_2[a_{20}] \oplus SS_1[a_{10}] \oplus SS_0[a_{00}] \oplus K[0], \\
B_1 &= SS_0[a_{31}] \oplus SS_3[a_{21}] \oplus SS_2[a_{11}] \oplus SS_1[a_{01}] \oplus K[1], \\
B_2 &= SS_1[a_{32}] \oplus SS_0[a_{22}] \oplus SS_3[a_{12}] \oplus SS_2[a_{02}] \oplus K[2], \\
B_3 &= SS_2[a_{33}] \oplus SS_1[a_{23}] \oplus SS_0[a_{13}] \oplus SS_3[a_{03}] \oplus K[3].
\end{aligned}$$

The even round function can be implemented by $B = \rho_{eK}(A) = \rho_{oK}((A[1], A[0], A[3], A[2])^t)$. Therefore, one round of CRYPTON can be performed using 20 table lookups (16 to SS-boxes, 4 to round keys), 16 XORs, 12 shifts and 16 ANDs (for byte extraction).

We have implemented CRYPTON in C (with in-line assembly in the case of Pentium Pro) and measured its speed on 200 MHz Pentium Pro running Windows 95 (with 32 Mbytes of RAM) and on 167 MHz UltraSparc running Solaris 2.5. The result is shown in Table 5. (These timings were obtained by iterating our core routines several million times and may have a certain amount of deviation from running to running.) Our optimized C code runs quite fast, giving an encryption rate of about 6.4 Mbytes/sec (51.4 Mbps) on Pentium Pro. The partial assembly code can encrypt/decrypt about 7.9 Mbytes per second (63.4 Mbps), running about 20 % faster than the optimized C code. (Only the encryption routine is implemented in in-line assembly.) We expect that a full assembly language implementation will run a little faster. On UltraSparc, CRYPTON runs somewhat slower, achieving an encryption rate of about 4.1 Mbytes/sec with the same code (the speed could be increased a little bit by further optimization on UltraSparc).

The key setup time of CRYPTON is different for encryption and decryption. Decryption key setup requires a little more computation due to the need of transformation of expanded keys. Our encryption key schedule is very fast, taking much less time than one-block encryption (though the code for key scheduling was not fully optimized). As a result, CRYPTON will be very efficient for use in hash function construction or in the case of encrypting/decrypting only a few blocks of data (e.g., MACs for entity authentication). Note that all the timings remain almost the same for different sizes of user keys.

Language\Clocks	Key setup (enc/dec)	Enc/Dec
In-line Asm (PC)	N/A	385 / 385 (63.4 Mbps)
MSVC 5.0 (PC)	355 / 430	475 / 475 (51.4 Mbps)
GNU C (UltraSparc)	505 / 570	615 / 615 (39.7 Mbps)

Table 5: Speed of CRYPTON on Pentium Pro and UltraSparc (for 128-bit keys)

4.2 Implementation on 8-bit Microprocessors

CRYPTON can be efficiently implemented on other platforms as well. For smart card implementations, we can only store 256 bytes of the involution S-box S and compute each entry of S_i's using just one rotation. The RAM requirement is also very small, just 52 bytes in total (20 bytes for data variables and 32 bytes for a user key). CRYPTON is expected to run quite fast on smart cards, since all computational steps can be efficiently implemented only using byte operations. We can estimate that encryption and decryption, including in-line key scheduling, can be performed using about 4500 and 4890 instructions, respectively, under the very restricted computing model such that data must be first loaded into a register, processed there using other data in a register and then stored back into memory. Here an instruction may be one of register-to-memory/memory-to-register MOV, register-to-register XOR, immediate-to-register AND and ROTate in a register.

Furthermore, considering typical smart card applications, we can reasonably assume that keys are not changed frequently and thus 32 bytes of expanded keys can be computed at the first time of use and then stored for later use. In this case, the cost for key expanding can be saved and thus encryption and decryption can be performed using about 3800 and 4190 instructions, respectively. If desired, we can even achieve the same speed for decryption, at the cost of the program size, by implementing direct inverse of the encryption process with the same round keys applied in reverse order. In the worst case, each S-box table entry can be directly computed from 4-bit P-boxes in about 35 instructions and only using 32 bytes of storage.

4.3 VLSI Implementation

In hardware the most complex part is of course the logic implementation of S-boxes. (Of course, we may implement the S-boxes in ROM or EEPROM, which is very simple but will be slower due to 16 parallel accesses to the S-boxes.) A simple counting based on boolean expressions shows that each S-box requires more than 200 gates, where a gate may be one of (n)and, (n)or and inverter. Since other operations needed are only XORs, we can estimate that a full parallel implementation of CRYPTON would require about 65000 gates and could easily achieve a speed of Giga bits/second (Compared to the previous version, the overall hardware complexity is increased by about 30% due to the increased complexity for S-boxes). Obviously, there are various trade-offs between speeds and costs. For example, we may implement just two rounds with much less complexity. Note, however, that this kind of gate counts may be of little use for actual VLSI implementations, since most hardware design tools utilize their own cell library (a cell may consist of several gates) and optimize the circuit using this library.

5 Conclusion

We described CRYPTON Version 1.0, a revised version of our AES proposal, and analyzed its security. CRYPTON Version 1.0 is only different from the previous version in S-box construction and key scheduling. We replaced S-boxes with stronger ones and a little strengthened the key schedule. These changes do not substantially change the analysis results of the previous version: The encryption/decryption speed in software remains the same and the key setup time is slightly increased. Only the S-box characteristic probabilities can be replaced for security analysis.

Our analysis shows that 12-round CRYPTON is far secure against any known attacks. At present the best attack on CRYPTON appears to be exhaustive key search. However, as usual, more extensive analysis should be done before practical applications of a newly introduced cipher, so we strongly

encourage the reader to further investigate our new version of CRYPTON. We would greatly appreciate any reports on its analysis.

Acknowledgement

The author is very grateful to Hyo Sun Hwang and Myung Hee Kang in Future Systems for their help in programming CRYPTON, in particular, in assembly and Java, respectively.

References

[1] E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, *Journal of Cryptology*, v. 4, 1991, pp. 3-72.

[2] J.Daemen, L.Knudsen and V.Rijmen, The block cipher Square, In *Fast Software Encryption*, Lecture Notes in Computer Science (LNCS) 1267, Springer-Verlag, 1997, pp.149-171.

[3] C.Harpes, G.Kramer and J.Massey, A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma, In *Advances in Cryptology-EUROCRYPT'95*, LNCS 921, Springer-Verlag, 1995, pp.24-38.

[4] C.Harpes and J.Massey, Partitioning cryptanalysis, In *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp.13-27.

[5] H.M.Heys and S.E.Tavares, Substitution-permutation networks resistant to differential and linear cryptanalysis, *J. Cryptology*, 9(1), 1996, pp.1-19.

[6] T.Jakobsen and L.R.Knudsen, The interpolation attack on block ciphers, In *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp.28-40.

[7] J.Kelsey, B.Schneier and D.Wagner, Key-schedule cryptanalysis of IDEA, DES, GOST, SAFER, and triple-DES, In *Advances in Cryptology-CRYPTO'96*, LNCS 1109, Springer-Verlag, 1996, pp.237-252.

[8] J.Kelsey, B.Schneier and D.Wagner, Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA, In *Proc. of ICICS'97*, Beijing, 1997.

[9] J.Kelsey, B.Schneier, D.Wagner and C.Hall, Side channel cryptanalysis of product ciphers, In *Computer Security-ESORICS'98*, LNCS 1485, Springer-Verlag, 1998.

[10] L.R.Knudsen, Truncated and higher order differentials, In *Fast Software Encryption*, LNCS 1008, Springer-Verlag, 1995, pp.196-211.

[11] P.Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, In *Advances in Cryptology-Crypto'96*, LNCS 1109, Springer-Verlag, 1996, pp.104-113.

[12] L.R.Knudsen and T.A.Berson, Truncated differentials of SAFER, In *Fast Software Encryption*, LNCS 1039, Springer-Verlag, 1996, pp.15-26.

[13] B.S.Kaliski Jr. and M.J.B.Robshaw, Linear cryptanalysis using multiple linear approximations, In *Advances in Cryptology-CRYPTO'94*, LNCS 839, Springer-Verlag, 1994, pp.26-39.

[14] B.S.Kaliski Jr. and M.J.B.Robshaw, Linear cryptanalysis Using multiple linear approximations and FEAL, In *Fast Software Encryption*, LNCS 1008, Springer-Verlag, 1995, pp.249-264.

[15] L.Knudsen and M.J.B.Robshaw, Non-linear approximations in linear cryptanalysis, In *Advances in Cryptology-EUROCRYPT'96*, LNCS 1070, Springer-Verlag, 1996, pp.252-267.

[16] X.Lai, *On the design and security of block ciphers*, PhD thesis, ETH, Zurich, 1992.

[17] M.Matsui, Linear cryptanalysis method for DES cipher, In *Advances in Cryptology-EUROCRYPT'93*, LNCS 765, Springer-Verlag, 1994, pp.386-397.

R	Encryption round keys					Decryption round keys				
0	$K_e[00]$	=	$E_e[0]$	$\oplus$	0x09e35996	$K_d[00]$	=	$E_d[0]$	$\oplus$	0x0bd72bc2
	$K_e[01]$	=	$E_e[1]$	$\oplus$	0xfc16ac63	$K_d[01]$	=	$E_d[1]$	$\oplus$	0x37fe22de
	$K_e[02]$	=	$E_e[2]$	$\oplus$	0x17fd4788	$K_d[02]$	=	$E_d[2]$	$\oplus$	0x35dc15c9
	$K_e[03]$	=	$E_e[3]$	$\oplus$	0xc02a905f	$K_d[03]$	=	$E_d[3]$	$\oplus$	0x1ee20bc2
1	$K_e[04]$	=	$E_e[4]$	$\oplus$	0x4d124400	$K_d[04]$	=	$E_d[4]$	$\oplus$	0xfaa1efd5
	$K_e[05]$	=	$E_e[5]$	$\oplus$	0xb8e7b1f5	$K_d[05]$	=	$E_d[5]$	$\oplus$	0x200f541a
	$K_e[06]$	=	$E_e[6]$	$\oplus$	0x530c5a1e	$K_d[06]$	=	$E_d[6]$	$\oplus$	0xf1cbe4bf
	$K_e[07]$	=	$E_e[7]$	$\oplus$	0x84db8dc9	$K_d[07]$	=	$E_d[7]$	$\oplus$	0x68261c33
2	$K_e[08]$	=	$E_e[1]^{\ll 24}$	$\oplus$	0xb28170b2	$K_d[08]$	=	$E_d[3]^{\ll_b 2}$	$\oplus$	0x6cf756b0
	$K_e[09]$	=	$E_e[2]^{\ll 16}$	$\oplus$	0x47748547	$K_d[09]$	=	$E_d[0]^{\ll 16}$	$\oplus$	0x459902a3
	$K_e[10]$	=	$E_e[3]^{\ll_b 6}$	$\oplus$	0xac9f6eac	$K_d[10]$	=	$E_d[1]^{\ll 8}$	$\oplus$	0x48ae72e9
	$K_e[11]$	=	$E_e[0]^{\ll_b 6}$	$\oplus$	0x7b48b97b	$K_d[11]$	=	$E_d[2]^{\ll_b 2}$	$\oplus$	0x3e9f79a5
3	$K_e[12]$	=	$E_e[7]^{\ll_b 2}$	$\oplus$	0xf630633c	$K_d[12]$	=	$E_d[5]^{\ll_b 6}$	$\oplus$	0xe351ea13
	$K_e[13]$	=	$E_e[4]^{\ll_b 2}$	$\oplus$	0x03c596c9	$K_d[13]$	=	$E_d[6]^{\ll 16}$	$\oplus$	0xe616a41f
	$K_e[14]$	=	$E_e[5]^{\ll 8}$	$\oplus$	0xe82e7d22	$K_d[14]$	=	$E_d[7]^{\ll 24}$	$\oplus$	0xf40dfd4f
	$K_e[15]$	=	$E_e[6]^{\ll 16}$	$\oplus$	0x3ff9aaf5	$K_d[15]$	=	$E_d[4]^{\ll_b 6}$	$\oplus$	0x9823da2a
4	$K_e[16]$	=	$E_e[2]^{\ll 8}$	$\oplus$	0x3ba76fae	$K_d[16]$	=	$E_d[2]^{\ll_b 4}$	$\oplus$	0x613b7d32
	$K_e[17]$	=	$(E_e[3]^{\ll_b 6})^{\ll 16}$	$\oplus$	0xce529a5b	$K_d[17]$	=	$(E_d[3]^{\ll_b 2})^{\ll 16}$	$\oplus$	0xc794ce88
	$K_e[18]$	=	$E_e[0]^{\ll_b 4}$	$\oplus$	0x25b971b0	$K_d[18]$	=	$E_e[0]^{\ll 24}$	$\oplus$	0x632c7f25
	$K_e[19]$	=	$(E_e[1]^{\ll_b 6})^{\ll 24}$	$\oplus$	0xf26ea667	$K_d[19]$	=	$(E_d[1]^{\ll_b 2})^{\ll 8}$	$\oplus$	0xf2b4fba8
5	$K_e[20]$	=	$(E_e[6]^{\ll_b 2})^{\ll 16}$	$\oplus$	0x7fd61ad8	$K_d[20]$	=	$(E_d[6]^{\ll_b 6})^{\ll 16}$	$\oplus$	0x34e12521
	$K_e[21]$	=	$E_e[7]^{\ll_b 4}$	$\oplus$	0x8a23ef2d	$K_d[21]$	=	$E_d[7]^{\ll 8}$	$\oplus$	0xd4c114d0
	$K_e[22]$	=	$(E_e[4]^{\ll_b 2})^{\ll 8}$	$\oplus$	0x61c804c6	$K_d[22]$	=	$(E_d[4]^{\ll_b 6})^{\ll 24}$	$\oplus$	0x3b3f2aff
	$K_e[23]$	=	$E_e[5]^{\ll 24}$	$\oplus$	0xb61fd311	$K_d[23]$	=	$E_d[5]^{\ll_b 4}$	$\oplus$	0x28ece8fd
6	$K_e[24]$	=	$(E_e[3]^{\ll_b 6})^{\ll 8}$	$\oplus$	0xa345054a	$K_d[24]$	=	$(E_d[1]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xe22fcca8
	$K_e[25]$	=	$(E_e[0]^{\ll_b 4})^{\ll 16}$	$\oplus$	0x56b0f0bf	$K_d[25]$	=	$(E_d[2]^{\ll_b 4})^{\ll 16}$	$\oplus$	0x5d17da39
	$K_e[26]$	=	$(E_e[1]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xbd5b1b54	$K_d[26]$	=	$(E_d[3]^{\ll_b 2})^{\ll 24}$	$\oplus$	0xd2b6fc31
	$K_e[27]$	=	$(E_e[2]^{\ll_b 6})^{\ll 8}$	$\oplus$	0x6a8ccc83	$K_d[27]$	=	$(E_d[0]^{\ll_b 2})^{\ll 24}$	$\oplus$	0xe605612b
7	$K_e[28]$	=	$(E_e[5]^{\ll_b 2})^{\ll 24}$	$\oplus$	0xe0f431f4	$K_d[28]$	=	$(E_d[7]^{\ll_b 6})^{\ll 8}$	$\oplus$	0x9d75b033
	$K_e[29]$	=	$(E_e[6]^{\ll_b 4})^{\ll 16}$	$\oplus$	0x1501c401	$K_d[29]$	=	$(E_d[4]^{\ll_b 6})^{\ll 8}$	$\oplus$	0xc6688045
	$K_e[30]$	=	$(E_e[7]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xfeea2fea	$K_d[30]$	=	$(E_d[5]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xae2d836b
	$K_e[31]$	=	$(E_e[4]^{\ll_b 2})^{\ll 24}$	$\oplus$	0x293df83d	$K_d[31]$	=	$(E_d[6]^{\ll_b 4})^{\ll 16}$	$\oplus$	0xbc79fa54
8	$K_e[32]$	=	$(E_e[0]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x246b3c66	$K_d[32]$	=	$(E_d[0]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xf273ea36
	$K_e[33]$	=	$(E_e[1]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xd19ec993	$K_d[33]$	=	$(E_d[1]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xc307861f
	$K_e[34]$	=	$(E_e[2]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x3a752278	$K_d[34]$	=	$(E_d[2]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xf428ec6d
	$K_e[35]$	=	$(E_e[3]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xeda2f5af	$K_d[35]$	=	$(E_d[3]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xba23ff3b
9	$K_e[36]$	=	$(E_e[4]^{\ll_b 4})^{\ll 24}$	$\oplus$	0x699a2890	$K_d[36]$	=	$(E_d[4]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xaaedb769
	$K_e[37]$	=	$(E_e[5]^{\ll_b 4})^{\ll 24}$	$\oplus$	0x9c6fdd65	$K_d[37]$	=	$(E_d[5]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x9c5f1842
	$K_e[38]$	=	$(E_e[6]^{\ll_b 4})^{\ll 24}$	$\oplus$	0x7784368e	$K_d[38]$	=	$(E_d[6]^{\ll_b 4})^{\ll 8}$	$\oplus$	0xa977b4f3
	$K_e[39]$	=	$(E_e[7]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xa053e159	$K_d[39]$	=	$(E_d[7]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x247ea063
10	$K_e[40]$	=	$E_e[1]^{\ll_b 4}$	$\oplus$	0xad09db02	$K_d[40]$	=	$(E_d[3]^{\ll_b 6})^{\ll 24}$	$\oplus$	0x43734180
	$K_e[41]$	=	$(E_e[2]^{\ll_b 4})^{\ll 24}$	$\oplus$	0x58fc2ef7	$K_d[41]$	=	$(E_d[0]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x75b686b4
	$K_e[42]$	=	$(E_e[3]^{\ll_b 2})^{\ll 8}$	$\oplus$	0xb317c51c	$K_d[42]$	=	$E_d[1]^{\ll_b 4}$	$\oplus$	0x5f9e5d6d
	$K_e[43]$	=	$(E_e[0]^{\ll_b 2})^{\ll 8}$	$\oplus$	0x64c012cb	$K_d[43]$	=	$(E_d[2]^{\ll_b 6})^{\ll 24}$	$\oplus$	0xba88498a
11	$K_e[44]$	=	$(E_e[7]^{\ll_b 6})^{\ll 24}$	$\oplus$	0x92b8c78c	$K_d[44]$	=	$(E_d[5]^{\ll_b 2})^{\ll 8}$	$\oplus$	0x1b551a4f
	$K_e[45]$	=	$(E_e[4]^{\ll_b 6})^{\ll 24}$	$\oplus$	0x674d3279	$K_d[45]$	=	$(E_d[6]^{\ll_b 4})^{\ll 24}$	$\oplus$	0xbaeea0ef
	$K_e[46]$	=	$E_e[5]^{\ll_b 4}$	$\oplus$	0x8ca6d992	$K_d[46]$	=	$E_d[7]^{\ll_b 4}$	$\oplus$	0x0451054b
	$K_e[47]$	=	$(E_e[6]^{\ll_b 4})^{\ll 8}$	$\oplus$	0x5b710e45	$K_d[47]$	=	$(E_d[4]^{\ll_b 2})^{\ll 8}$	$\oplus$	0x9cd386d2
12	$K_e[48]$	=	$(E_e[2]^{\ll_b 4})^{\ll 16}$	$\oplus$	0xd62ff23e	$K_d[48]$	=	$E_d[2]^{\ll 24}$	$\oplus$	0xf00fa47e
	$K_e[49]$	=	$(E_e[3]^{\ll_b 2})^{\ll 24}$	$\oplus$	0x23da07cb	$K_d[49]$	=	$(E_d[3]^{\ll_b 6})^{\ll 8}$	$\oplus$	0x8b05fa51
	$K_e[50]$	=	$E_e[0]^{\ll 8}$	$\oplus$	0xc831ec20	$K_d[50]$	=	$(E_d[0]^{\ll_b 4})^{\ll 16}$	$\oplus$	0xba60ee11
	$K_e[51]$	=	$E_e[1]^{\ll_b 2}$	$\oplus$	0x1fe63bf7	$K_d[51]$	=	$E_d[1]^{\ll_b 6}$	$\oplus$	0xc66db739

Table 6: Encryption and decryption round keys in terms of expanded keys

		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S_0	0	63	ec	59	aa	db	8e	66	c0	37	3c	14	ff	13	44	a9	91
	1	3b	78	8d	ef	c2	2a	f0	d7	61	9e	a5	bc	48	15	12	47
	2	ed	42	1a	33	38	c8	17	90	a6	d5	5d	65	6a	fe	8f	a1
	3	93	ca	2f	0c	68	58	df	f4	45	11	a0	a7	22	96	fb	7d
	4	1d	b4	84	e0	bf	57	e9	0a	4e	83	cc	7a	71	39	c7	32
	5	74	3d	de	50	85	06	6f	53	e8	ad	82	19	e1	ba	36	cb
	6	0e	28	f3	9b	4a	62	94	1f	bd	f6	67	41	d8	d1	2d	a4
	7	86	b7	01	c5	b0	75	02	f9	2c	29	6e	d2	5f	8b	fc	5a
	8	e4	7f	dd	07	55	b1	2b	89	72	18	3a	4c	b6	e3	80	ce
	9	49	cf	6b	b9	f2	0d	dc	64	95	46	f7	10	9a	20	a2	3f
	a	d6	87	70	3e	21	fd	4d	7b	c3	ae	09	8a	04	b3	54	f8
	b	30	00	56	d4	e7	25	bb	ac	98	73	ea	c9	9d	4f	7e	03
	c	ab	92	a8	43	0f	fa	24	5c	1e	60	31	97	cd	c6	79	f5
	d	5e	e5	34	76	1c	81	b2	af	0b	5b	d9	e2	27	6d	d0	88
	e	c1	51	e6	9c	77	be	99	23	da	eb	52	2e	b5	08	05	6c
	f	b8	1b	a3	69	8c	d3	40	26	f1	c4	9f	35	ee	7c	4b	16
S_1	0	8d	b3	65	aa	6f	3a	99	03	dc	f0	50	ff	4c	11	a6	46
	1	ec	e1	36	bf	0b	a8	c3	5f	85	7a	96	f2	21	54	48	1d
	2	b7	09	68	cc	e0	23	5c	42	9a	57	75	95	a9	fb	3e	86
	3	4e	2b	bc	30	a1	61	7f	d3	15	44	82	9e	88	5a	ef	f5
	4	74	d2	12	83	fe	5d	a7	28	39	0e	33	e9	c5	e4	1f	c8
	5	d1	f4	7b	41	16	18	bd	4d	a3	b6	0a	64	87	ea	d8	2f
	6	38	a0	cf	6e	29	89	52	7c	f6	db	9d	05	63	47	b4	92
	7	1a	de	04	17	c2	d5	08	e7	b0	a4	b9	4b	7d	2e	f3	69
	8	93	fd	77	1c	55	c6	ac	26	c9	60	e8	31	da	8f	02	3b
	9	25	3f	ad	e6	cb	34	73	91	56	19	df	40	6a	80	8a	fc
	a	5b	1e	c1	f8	84	f7	35	ed	0f	ba	24	2a	10	ce	51	e3
	b	c0	00	59	53	9f	94	ee	b2	62	cd	ab	27	76	3d	f9	0c
	c	ae	4a	a2	0d	3c	eb	90	71	78	81	c4	5e	37	1b	e5	d7
	d	79	97	d0	d9	70	06	ca	be	2c	6d	67	8b	9c	b5	43	22
	e	07	45	9b	72	dd	fa	66	8c	6b	af	49	b8	d6	20	14	b1
	f	e2	6c	8e	a5	32	4f	01	98	c7	13	7e	d4	bb	f1	2d	58
S_2	0	b1	72	76	bf	ac	ee	55	83	ed	aa	47	d8	33	95	60	c4
	1	9b	39	1e	0c	0a	1d	ff	26	89	5b	22	f1	d4	40	c8	67
	2	9d	a4	3c	e7	c6	b5	f7	dc	61	79	15	86	78	6e	eb	32
	3	b0	ca	4f	23	d2	fb	5e	08	24	4d	8a	10	09	51	a3	9f
	4	f6	6b	21	c3	0d	38	99	1f	1c	90	64	fe	8b	a6	48	bd
	5	53	e1	ea	57	ae	84	b2	45	35	02	7f	d9	c7	2a	d0	7c
	6	c9	18	65	00	97	2b	06	6a	34	f3	2c	92	ef	dd	7a	56
	7	a2	4c	88	b9	50	75	d3	e4	11	ce	4b	a7	fd	3f	be	81
	8	8e	d5	5a	49	42	54	70	a1	df	87	ab	7d	f4	12	05	2e
	9	27	0f	c1	30	66	98	3d	cb	b8	e6	9c	63	e3	bc	19	fa
	a	3a	2f	9e	f2	6f	1a	28	3b	c2	0e	03	c0	b7	59	a9	d7
	b	74	85	d6	ad	41	ec	8c	71	f0	93	5d	b6	1b	68	e5	44
	c	07	e0	14	a8	f9	73	cd	4e	25	bb	31	5f	4a	cc	8f	91
	d	de	6d	7b	f5	b3	29	a0	17	6c	da	e8	04	96	82	52	36
	e	43	5c	db	8d	80	d1	e2	b4	58	46	ba	e9	01	20	fc	13
	f	16	f8	94	62	37	cf	69	9a	af	77	c5	3e	7e	a5	2d	0b
S_3	0	b1	f6	8e	07	72	6b	d5	e0	76	21	5a	14	bf	c3	49	a8
	1	ac	0d	42	f9	ee	38	54	73	55	99	70	cd	83	1f	a1	4e
	2	ed	1c	df	25	aa	90	87	bb	47	64	ab	31	d8	fe	7d	5f
	3	33	8b	f4	4a	95	a6	12	cc	60	48	05	8f	c4	bd	2e	91
	4	9b	53	27	de	39	e1	0f	6d	1e	ea	c1	7b	0c	57	30	f5
	5	0a	ae	66	b3	1d	84	98	29	ff	b2	3d	a0	26	45	cb	17
	6	89	35	b8	6c	5b	02	e6	da	22	7f	9c	e8	f1	d9	63	04
	7	d4	c7	e3	96	40	2a	bc	82	c8	d0	19	52	67	7c	fa	36
	8	9d	c9	3a	43	a4	18	2f	5c	3c	65	9e	db	e7	00	f2	8d
	9	c6	97	6f	80	b5	2b	1a	d1	f7	06	28	e2	dc	6a	3b	b4
	a	61	34	c2	58	79	f3	0e	46	15	2c	03	ba	86	92	c0	e9
	b	78	ef	b7	01	6e	dd	59	20	eb	7a	a9	fc	32	56	d7	13
	c	b0	a2	74	16	ca	4c	85	f8	4f	88	d6	94	23	b9	ad	62
	d	d2	50	41	37	fb	75	ec	cf	5e	d3	8c	69	08	e4	71	9a
	e	24	11	f0	af	4d	ce	93	77	8a	4b	5d	c5	10	a7	b6	3e
	f	09	fd	1b	7e	51	3f	68	a5	a3	be	e5	2d	9f	81	44	0b

Table 7: The CRYPTON S-boxes

S_0	D	(d,72; 8) (12,24; 8) (1a,72; 8) (1b,49; 8) (1d,a5; 8) (29,4f; 8) (36,eb; 8) (39,1a; 8) (39,34; 8) (3e,7c; 8) (45,53; 8) (48,90; 8) (51,95; 8) (51,c4; 8) (5a, 7; 8) (5c,b8; 8) (61,77; 8) (61,c2;10) (62,a2; 8) (62,c7; 8) (67,3b; 8) (68,49; 8) (6f,f3; 8) (71,e7; 8) (79,29; 8) (7c,f8; 8) (7f,4f; 8) (83,b4; 8) (85, b; 8) (8c,79; 8) (91,79; 8) (94,f2; 8) (9c,9b; 8) (9d,ce; 8) (a4,36; 8) (a4,d0; 8) (a7,52; 8) (a7,fe; 8) (a9,8a; 8) (ab,a1; 8) (ac,59;10) (af,9b; 8)
	C	(b3,67; 8) (bb,c2; 8) (bc,19; 8) (bc,23; 8) (c1,d5;10) (c3,87; 8) (c6,a3; 8) (ca,a2; 8) (cd,39; 8) (cd,5f; 8) (cf,9f; 8) (d0,57; 8) (d1,8d; 8) (d2,3a; 8) (d4,a9; 8) (dd,f9;10) (e3,c4; 8) (e4,c9;10) (e6,cf; 8) (e7,cd; 8) (e7,eb; 8) (ea,83;10) (f3,e2; 8) (f5,6c; 8) (f5,cf; 8) (f9,de; 8) (fc,bb;10)
	L	(3,bf;28) (7,51;28) (9,23;28) (1a,e5;28) (1c,38;32) (1c,ca;28) (1c,ff;28) (21,a6;28) (22,fd;28) (2a,a4;28) (2c,e2;28) (34,c2;28) (44, f;28) (44,3d;32) (52,54;28) (53,42;28) (58,bd;28) (5b,cf;28) (61,68;28) (65,38;28) (71,58;28) (75,a9;28) (87,88;28) (8e,55;28) (8e,eb;28) (91,12;28) (9e,88;32) (a8, e;28)
	C	(aa,1d;28) (af,65;28) (af,ed;32) (b2,5f;28) (cb,97;32) (d4,ea;28) (de,b0;28) (df, 6;28) (e7,b6;28) (f2,34;28) (f5,1d;28) (f6,5f;32) (fe,44;28) (ff,38;28)
S_1	D	(d,c9; 8) (12,90; 8) (1a,c9; 8) (1b,25; 8) (1d,96; 8) (29,3d; 8) (36,af; 8) (39,68; 8) (39,d0; 8) (3e,f1; 8) (45,4d; 8) (48,42; 8) (51,13; 8) (51,56; 8) (5a,1c; 8) (5c,e2; 8) (61, b;10) (61,dd; 8) (62,1f; 8) (62,8a; 8) (67,ec; 8) (68,25; 8) (6f,cf; 8) (71,9f; 8) (79,a4; 8) (7c,e3; 8) (7f,3d; 8) (83,d2; 8) (85,2c; 8) (8c,e5; 8) (91,e5; 8) (94,cb; 8) (9c,6e; 8) (9d,3b; 8) (a4,43; 8)
	C	(a4,d8; 8) (a7,49; 8) (a7,fb; 8) (a9,2a; 8) (ab,86; 8) (ac,65;10) (af,6e; 8) (b3,9d; 8) (bb, b; 8) (bc,64; 8) (bc,8c; 8) (c1,57;10) (c3,1e; 8) (c6,8e; 8) (ca,8a; 8) (cd,7d; 8) (cd,e4; 8) (cf,7e; 8) (d0,5d; 8) (d1,36; 8) (d2,e8; 8) (d4,a6; 8) (dd,e7;10) (e3,13; 8) (e4,27;10) (e6,3f; 8) (e7,37; 8) (e7,af; 8) (ea, e;10) (f3,8b; 8) (f5,3f; 8) (f5,b1; 8) (f9,7b; 8) (fc,ee;10)
	L	(3,fe;28) (7,45;28) (9,8c;28) (1a,97;28) (1c,2b;28) (1c,e0;32) (1c,ff;28) (21,9a;28) (22,f7;28) (2a,92;28) (2c,8b;28) (34, b;28) (44,3c;28) (44,f4;32) (52,51;28) (53, 9;28) (58,f6;28) (5b,3f;28) (61,a1;28) (65,e0;28) (71,61;28)
	C	(75,a6;28) (87,22;28) (8e,55;28) (8e,af;28) (91,48;28) (9e,22;32) (a8,38;28) (aa,74;28) (af,95;28) (af,b7;32) (b2,7d;28) (cb,5e;32) (d4,ab;28) (de,c2;28) (df,18;28) (e7,da;28) (f2,d0;28) (f5,74;28) (f6,7d;32) (fe,11;28) (ff,e0;28)
S_2	D	(7,5a; 8) (b,85; 8) (19,bc; 8) (1a,39; 8) (23,bc; 8) (24,12; 8) (29,79; 8) (34,39; 8) (36,a4; 8) (39,cd; 8) (3a,d2; 8) (3b,67; 8) (49,1b; 8) (49,68; 8) (4f,29; 8) (4f,7f; 8) (52,a7; 8) (53,45; 8) (57,d0; 8) (59,ac;10) (5f,cd; 8) (67,b3; 8) (6c,f5; 8) (72, d; 8) (72,1a; 8) (77,61; 8) (79,8c; 8) (79,91; 8) (7c,3e; 8) (83,ea;10) (87,c3; 8) (8a,a9; 8) (8d,d1; 8) (90,48; 8) (95,51; 8)
	C	(9b,9c; 8) (9b,af; 8) (9f,cf; 8) (a1,ab; 8) (a2,62; 8) (a2,ca; 8) (a3,c6; 8) (a5,1d; 8) (a9,d4; 8) (b4,83; 8) (b8,5c; 8) (bb,fc;10) (c2,61;10) (c2,bb; 8) (c4,51; 8) (c4,e3; 8) (c7,62; 8) (c9,e4;10) (cd,e7; 8) (ce,9d; 8) (cf,e6; 8) (cf,f5; 8) (d0,a4; 8) (d5,c1;10) (de,f9; 8) (e2,f3; 8) (e7,71; 8) (eb,36; 8) (eb,e7; 8) (f2,94; 8) (f3,6f; 8) (f8,7c; 8) (f9,dd;10) (fe,a7; 8)
	L	(6,df;28) (e,a8;28) (f,44;28) (12,91;28) (1d,aa;28) (1d,f5;28) (23, 9;28) (34,f2;28) (38,1c;32) (38,65;28) (38,ff;28) (3d,44;32) (42,53;28) (44,fe;28) (51, 7;28) (54,52;28) (55,8e;28) (58,71;28) (5f,b2;28) (5f,f6;32) (65,af;28)
	C	(68,61;28) (88,87;28) (88,9e;32) (97,cb;32) (a4,2a;28) (a6,21;28) (a9,75;28) (b0,de;28) (b6,e7;28) (bd,58;28) (bf, 3;28) (c2,34;28) (ca,1c;28) (cf,5b;28) (e2,2c;28) (e5,1a;28) (ea,d4;28) (eb,8e;28) (ed,af;32) (fd,22;28) (ff,1c;28)
S_3	D	(b,61;10) (b,bb; 8) (e,ea;10) (13,51; 8) (13,e3; 8) (1c,5a; 8) (1e,c3; 8) (1f,62; 8) (25,1b; 8) (25,68; 8) (27,e4;10) (2a,a9; 8) (2c,85; 8) (36,d1; 8) (37,e7; 8) (3b,9d; 8) (3d,29; 8) (3d,7f; 8) (3f,e6; 8) (3f,f5; 8) (42,48; 8) (43,a4; 8) (49,a7; 8) (4d,45; 8) (56,51; 8) (57,c1;10) (5d,d0; 8) (64,bc; 8) (65,ac;10) (68,39; 8) (6e,9c; 8) (6e,af; 8) (7b,f9; 8) (7d,cd; 8) (7e,cf; 8)
	C	(86,ab; 8) (8a,62; 8) (8a,ca; 8) (8b,f3; 8) (8c,bc; 8) (8e,c6; 8) (90,12; 8) (96,1d; 8) (9d,b3; 8) (9f,71; 8) (a4,79; 8) (a6,d4; 8) (af,36; 8) (af,e7; 8) (b1,f5; 8) (c9, d; 8) (c9,1a; 8) (cb,94; 8) (cf,6f; 8) (d0,39; 8) (d2,83; 8) (d8,a4; 8) (dd,61; 8) (e2,5c; 8) (e3,7c; 8) (e4,cd; 8) (e5,8c; 8) (e5,91; 8) (e7,dd;10) (e8,d2; 8) (ec,67; 8) (ee,fc;10) (f1,3e; 8) (fb,a7; 8)
	L	(9,53;28) (b,34;28) (11,fe;28) (18,df;28) (22,87;28) (22,9e;32) (2b,1c;28) (38,a8;28) (3c,44;28) (3f,5b;28) (45, 7;28) (48,91;28) (51,52;28) (55,8e;28) (5e,cb;32) (61,71;28) (74,aa;28) (74,f5;28) (7d,b2;28) (7d,f6;32) (8b,2c;28)
	C	(8c, 9;28) (92,2a;28) (95,af;28) (97,1a;28) (9a,21;28) (a1,61;28) (a6,75;28) (ab,d4;28) (af,8e;28) (b7,af;32) (c2,de;28) (d0,f2;28) (da,e7;28) (e0,1c;32) (e0,65;28) (e0,ff;28) (f4,44;32) (f6,58;28) (f7,22;28) (fe, 3;28) (ff,1c;28)

Table 8: Partial Differential/Linear characteristics for CRYPTON S-boxes