A Revised Version of CRYPTON - CRYPTON V1.0 -

Chae Hoon Lim

Information & Communications Research Center, Future Systems, Inc.
372-2, Yang Jae-Dong, Seo Cho-Gu, Seoul, 137-130, Korea: chlim@future.co.kr

Abstract. The block cipher CRYPTON has been proposed as a candidate algorithm for the Advanced Encryption Standard (AES). To fix some minor weakness in the key schedule and to remove some undesirable properties in S-boxes, we made some changes to the AES proposal, i.e., in the S-box construction and key scheduling. This paper presents the revised version of CRYPTON and its preliminary analysis.

1 Motivations and Changes

The block cipher CRYPTON has been proposed as a candidate algorithm for the AES [22]. Unfortunately, however, 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 this revision and analyzes its security and efficiency.

CRYPTON v1.0 is different from the AES proposal (v0.5) only in the S-box construction and key scheduling. As we mentioned at the 1st AES candidate conference, we already had a plan to revise the CRYPTON key schedule. The previous key schedule was in fact 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 (posted at NIST's AES forum) and by Johan Borst [4]). We thus made some enhancements to the original key schedule, while trying to keep changes minimal. The new key schedule now makes use of bit and word rotations, as well as byte rotations. We also used distinct round constants for each round key. This way we tried to make each byte of expanded keys 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.

In v0.5 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 V0.5 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 stronger for long-term security by allowing a large safety margin. We thus decided to strengthen the S-box in this opportunity.

Experiments show that most Feistel-type constructions seem to generate S-boxes with too many high-probability characteristics, so we 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 a SP network described in Sect.3.2. The resulting S-boxes are much stronger against differential and linear cryptanalysis when combined with linear transformations used. 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).

Finally, we would like to stress that the above modifications do not require any substantial change in existing analysis on the security and efficiency. The security evaluation of the new version can be done only by replacing old figures replated to S-box characteristics with new ones and there is no change in the overall structure of key scheduling. The performance figures in software implementations remain almost the same. The hardware complexity is a little bit increased due to the increased complexity for logic implementation of S-boxes.

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

- A 4×4 byte array A is represented by

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

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

2 Algorithm Specifications

CRYPTON processes a data block of 16 bytes by representing it into a 4×4 byte array as in SQUARE [6]. 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. This section presents a detailed description of CRYPTON v1.0.

2.1 Basic Building Blocks

Nonlinear Substitution γ The nonlinear transformation γ consists of byte-wise substitutions on a 4×4 byte array by using four 8×8 S-boxes, S_i $(0 \le i \le 3)$, such that $S_2 = S_0^{-1}$ and $S_3 = S_1^{-1}$ (see Sect.3.2 for details). Two different S-box

arrangements are used in successive rounds alternately; γ_o in odd rounds and γ_e in even rounds. They are defined as

$$B = \gamma_o(A) \;\Leftrightarrow\; b_{ij} = S_{i+j \bmod 4}(a_{ij}),$$
$$B = \gamma_e(A) \;\Leftrightarrow\; b_{ij} = S_{i+j+2 \bmod 4}(a_{ij}).$$

Observe that the four S-boxes are arranged so that $\gamma_o^{-1} = \gamma_e$ and $\gamma_e^{-1} = \gamma_o$. This property will be used to derive identical processes for encryption and decryption.

Bit Permutation π The bit permutation π bit-wise mixes each byte column of 4×4 byte array using four masking bytes m_i's given by

$$m_0 = \texttt{0xfc},\ m_1 = \texttt{0xf3},\ m_2 = \texttt{0xcf},\ m_3 = \texttt{0x3f}.$$

We first define four column permutations π_i's ($0 \le i \le 3$) as

$$[b_3, b_2, b_1, b_0]^t = \pi_i([a_3, a_2, a_1, a_0]^t) \;\Leftrightarrow\; b_j = \oplus_{k=0}^{3}(m_{i+j+k \bmod 4} \wedge a_k),$$

The b_j can be expressed alternatively using bit extraction and xoring as

$$b_j = \oplus_{k=0}^{3}(\overline{m}_{i+j+k \bmod 4} \wedge a_k) \oplus a,$$

where $\overline{m}_k$ denotes bit-wise complement of m_k and $a = \oplus_{k=0}^{3} a_k$.

As in γ, we use two slightly different versions of bit permutation to make encryption and decryption processes identical: π_o in odd rounds and π_e in even rounds. Let A^i be the i-th byte column of a 4×4 byte array A, i.e., $A^i = (a_{3i}, a_{2i}, a_{1i}, a_{0i})^t$. Then the bit permutations π_o and π_e are defined as

$$\pi_o(A) = (\pi_3(A^3), \pi_2(A^2), \pi_1(A^1), \pi_0(A^0)),$$
$$\pi_e(A) = (\pi_1(A^3), \pi_0(A^2), \pi_3(A^1), \pi_2(A^0)).$$

Note that $\pi_o^{-1} = \pi_o$ and $\pi_e^{-1} = \pi_e$ and that if $\pi_0([d, c, b, a]^t) = [h, g, f, e]^t$, then

$$\pi_1([d, c, b, a]^t) = [e, h, g, f]^t,\ \pi_2([d, c, b, a]^t) = [f, e, h, g]^t,\ \pi_3([d, c, b, a]^t) = [g, f, e, h]^t.$$

This property will be used to derive an efficient decryption key schedule from the encryption key schedule (see Sect.2.3).

Byte Transposition τ It simply moves the byte at the (i, j)-th position to the (j, i)-th position, i.e., $B = \tau(A) \Leftrightarrow b_{ij} = a_{ji}$. Note that $\tau^{-1} = \tau$.

Key Xoring σ 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 $0 \le i \le 3$. Obviously, $\sigma_K^{-1} = \sigma_K$.

Round Transformation ρ One round of CRYPTON consists of applying γ, π, τ and σ in sequence to the 4×4 data array. More specifically, the odd and even round functions are defined (for round key K) by

$$\rho_{oK} = \sigma_K \circ \tau \circ \pi_o \circ \gamma_o \quad \text{for odd rounds,}$$
$$\rho_{eK} = \sigma_K \circ \tau \circ \pi_e \circ \gamma_e \quad \text{for even rounds.}$$

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. The encryption transformation E_K of 12-round CRYPTON under key K consists of an initial key addition and 6 times repetitions of ρ_o and ρ_e and then a final output transformation. More specifically, E_K can be described as

$$E_K = \phi_e \circ \rho_{eK_e^{12}} \circ \rho_{oK_e^{11}} \circ \cdots \circ \rho_{eK_e^2} \circ \rho_{oK_e^1} \circ \sigma_{K_e^0}, \tag{1}$$

where ϕ_e is an output transformation to make 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:

$$D_K = \phi_e \circ \rho_{eK_d^{12}} \circ \rho_{oK_d^{11}} \circ \cdots \circ \rho_{eK_d^2} \circ \rho_{oK_d^1} \circ \sigma_{K_d^0}, \tag{2}$$

where the decryption round keys are defined by

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

This shows that decryption can be performed by the same function as encryption with a different key schedule.

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 $\phi_e \circ \rho_{eK_e^{12}} = \sigma_{K_d^0} \circ \tau \circ \gamma_e$.

2.3 Key Scheduling

CRYPTON requires total $4 \times 13 = 52$ 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.

Generating Expanded Keys Let $K = k_{31} \cdots k_1 k_0$ be a 256-bit user key. We first split K into U and V such that $U[i] = k_{8i+6} k_{8i+4} k_{8i+2} k_{8i}$ and $V[i] = k_{8i+7} k_{8i+5} k_{8i+3} k_{8i+1}$ for $i = 0, 1, 2, 3$. Then we compute the 8 expanded keys $E_e[i]$ ($0 \le i \le 7$) using round transformations with all-zero key as

$$\begin{gathered} U' = \rho_o(U), \qquad V' = \rho_e(V), \\ E_e[i] = U'[i] \oplus T_1, \; E_e[i+4] = V'[i] \oplus T_0, \end{gathered}$$

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

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

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

In addition, we also use the following 4 masking constants 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

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

2. for rounds $r = 2, 3, \cdots, 12$, repeat the following two steps alternately:
   2-1. even rounds:

$$\begin{aligned} \{E_e[3], E_e[2], E_e[1], E_e[0]\} &\leftarrow \{E_e[0]^{\lll_b 6}, E_e[3]^{\lll_b 6}, E_e[2]^{\lll 16}, E_e[1]^{\lll 24}\}, \\ K_e[4r+i] &\leftarrow E_e[i] \oplus C_e[r] \oplus MC_i \quad \text{for } 0 \le i \le 3. \end{aligned}$$

   2-2. odd rounds:

$$\begin{aligned} \{E_e[7], E_e[6], E_e[5], E_e[4]\} &\leftarrow \{E_e[6]^{\lll 16}, E_e[5]^{\lll 8}, E_e[4]^{\lll_b 2}, E_e[7]^{\lll_b 2}\}, \\ K_e[4r+i] &\leftarrow E_e[i+4] \oplus C_e[r] \oplus MC_i \quad \text{for } 0 \le i \le 3. \end{aligned}$$

Generating decryption round keys For efficient decryption key schedule, first observe that $\phi_o = \tau \circ \pi_o \circ \tau$ and $\phi_e = \tau \circ \pi_e \circ \tau$ can be rewritten as

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

Here ϕ_i is actually the same as π_i except that 4 input bytes are now arranged in a row vector (see Sect.2.1.2). Also note the shift and linear properties of ϕ_i

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

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 (Decryption round constants $C_d[i]$'s are given by $C_d[i] = \phi_2(C_e[12-i])$ for even i's and $C_d[i] = \phi_0(C_e[12-i])$ for odd i's.):

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

$$\begin{aligned} \{E_d[3], E_d[2], E_d[1], E_d[0]\} &\leftarrow \{\phi_0(E_e[1])^{\lll_b 2}, \phi_1(E_e[0]), \phi_1(E_e[3])^{\lll_b 2}, \phi_2(E_e[2])^{\lll_b 4}\}, \\ \{E_d[7], E_d[6], E_d[5], E_d[4]\} &\leftarrow \{\phi_2(E_e[6])^{\lll_b 4}, \phi_0(E_e[5])^{\lll_b 4}, \phi_1(E_e[4])^{\lll_b 6}, \phi_0(E_e[7])^{\lll_b 6}\}, \end{aligned}$$

2. compute the first 8 round keys as

$$K_d[i] \leftarrow E_d[i] \oplus C_d[0]^{\ll 32-8i} \oplus MC_i,$$
$$K_d[i+4] \leftarrow E_d[i+4] \oplus C_d[1]^{\ll 32-8i} \oplus MC_i \quad \text{for } 0 \le i \le 3.$$

3. for rounds $r = 2, 3, \cdots, 12$, repeat the following two steps alternately:
   3-1. even rounds:

$$\{E_d[3], E_d[2], E_d[1], E_d[0]\} \leftarrow \{E_d[2]^{\ll_b 2}, E_d[1]^{\ll 8}, E_d[0]^{\ll 16}, E_d[3]^{\ll_b 2}\},$$
$$K_d[4r+i] \leftarrow E_d[i] \oplus C_d[r]^{\ll 32-8i} \oplus MC_i \quad \text{for } 0 \le i \le 3.$$

   3-2. odd rounds:

$$\{E_d[7], E_d[6], E_d[5], E_d[4]\} \leftarrow \{E_d[4]^{\ll_b 6}, E_d[7]^{\ll 24}, E_d[6]^{\ll 16}, E_d[5]^{\ll_b 6}\},$$
$$K_d[4r+i] \leftarrow E_d[i+4] \oplus C_d[r]^{\ll 32-8i} \oplus MC_i \quad \text{for } 0 \le i \le 3.$$

3 Security Analysis

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 effective diffusion of S-box outputs by linear transformations plays an important role for resistance against various attacks such as differential and linear cryptanalysis (DC and LC for short) [3,23]

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. It is also easy to see that any column vector with n ($n < 4$) nonzero bytes is transformed by π_i into a column vector with at least $4-n$ nonzero bytes (we call this number 4 the diffusion order of π_i). 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 $\tau \circ \pi$ in successive rounds.

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

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

Let I_j be a set of input vectors with j nonzero bytes which are transformed by π_i into output vectors with $4-j$ nonzero bytes. Then all possible 32-bit values with minimum diffusion can be obtained as: for each x in Ω_x and y in Ω_y,

$$I_1 = \{(0,0,0,x)^t, (0,0,x,0)^t, (0,x,0,0)^t, (x,0,0,0)^t\},$$
$$I_2 = \{(0,0,x,x)^t, (0,x,x,0)^t, (x,x,0,0)^t, (x,0,0,x)^t, (0,y,0,y)^t, (y,0,y,0)^t\},$$
$$I_3 = \{(0,x,x,x)^t, (x,0,x,x)^t, (x,x,0,x)^t, (x,x,x,0)^t\}.$$

Therefore, we can see that there are only 204 vectors with minimum diffusion: 48 from $\pi_i(I_1) = I_3$, 108 from $\pi_i(I_2) = I_2$ and 48 from $\pi_i(I_3) = I_1$. 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 for inputs with two separated nonzero bytes (the last two cases in I_2).

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 S-box transformation without details of the S-box characteristics. Since it suffices to consider worst-case propagations, we only examine 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 active bytes (without considering S-box char.)

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 DC and LC.

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

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

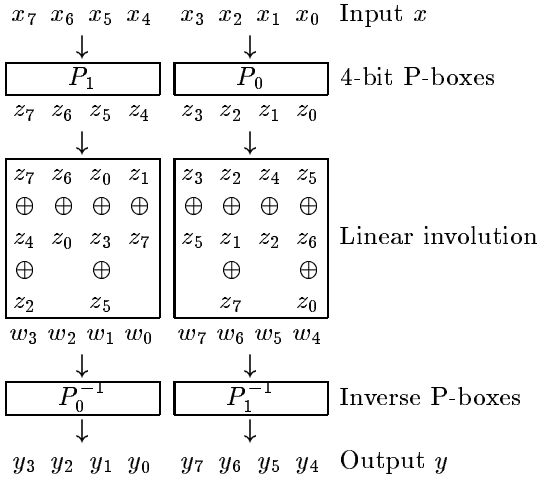
$$\lambda_S \stackrel{\text{def}}{=} \max_{\Gamma x, \Gamma y \neq 0} \left(\frac{|\#\{x \in X | x \bullet \Gamma x = S(x) \bullet \Gamma y\} - 2^{n-1}|}{2^{n-1}} \right)^2, \tag{5}$$

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 Fig.1. Then the actual four S-boxes were derived from S as follows: for each $x \in [0, 256)$,

$$S_0(x) = S(x)^{\ll 1}, \quad S_1(x) = S(x)^{\ll 3}, \quad S_2(x) = S(x^{\ll 7}), \quad S_3(x) = S(x^{\ll 5}).$$

It is easy to see that these S-boxes satisfy inverse relationships such that $S_0^{-1} = S_2$ and $S_1^{-1} = S_3$. 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 required in some limited computing environments (e.g., low-cost smart cards).



	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

Fig.1 The selected 8×8 involution S-box S

The involution S-box S was searched for, over some limited space of good 4-bit P-boxes and linear involutions, in such a way that it has best possible differential and linear characteristics. Moreover, among such a set of candidate involution S-boxes, we selected the final S-box S considering the following two additional requirements:

1. The high-probability I/O difference pairs (selection patterns, resp.) in S should have as high Hamming weights as possible.

2. 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.

These requirements are to ensure that high-probability differences/selection patterns should be more rapidly diffused by linear transformations and that it should be more difficult to form a chain of high-probability S-box characteristics/linear approximations through consecutive rounds.

Table 3 shows their statistics on the distribution of input-output difference/linear approximation pairs, where the entry values are computed by the numerator of equations (4) and (5).

D	entry value	0	2	4	6	8	10			
C	no of entries	39584	20158	4976	749	62	7			
L	entry value	0	4	8	12	16	20	24	28	32
C	no of entries	13927	22058	15948	8460	3731	1094	276	36	6

Table 3. Distribution of difference/linear approx. pairs for S_i

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

$$p_d \stackrel{\text{def}}{=} \delta_{S_i} = \frac{10}{256} = 2^{-4.68}, \quad p_l \stackrel{\text{def}}{=} \lambda_{S_i} = (\frac{32}{128})^2 = 2^{-4},$$

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 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.

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

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

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 1 such selection pattern in each S-box, as shown in Table 4. Note that even the pairs in the table 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

$$p'_d \stackrel{\text{def}}{=} \delta_{S_i}^{\Omega_y} = \frac{6}{256} = 2^{-5.42}, \quad p'_l \stackrel{\text{def}}{=} \lambda_{S_i}^{\Omega_y} = (\frac{24}{128})^2 = 2^{-4.83}.$$

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 to differential cryptanalysis.

First note that the probability of any characteristic in CRYPTON can be completely determined by the number of active S-boxes and their char. probabilities (e.g., see [10]). 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 char. 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 char. 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 char. probabilities should give better overall probability than a characteristic involving a larger number of active S-boxes with larger S-box char. probabilities. Therefore, we can obtain a tighter bound for the 8-round char. probability as $p_{C_8} < (p'_d)^{32} = 2^{-173.3}$. The actual probability will be much lower than this bound, but we do not proceed any more since this bound is lower enough to show the strong resistance of CRYPTON against DC 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. It is not easy to estimate the number of such characteristics that can reside in a differential. However, the estimated 8-round char. probability, together with a rough analysis solely based on the diffusion property of linear transformations, shows that no 8-round differential can have probability significantly larger than 2^{-128}. Therefore, we believe that CRYPTON with 9 or more rounds is far secure against the basic differential attack.

3.4 Linear Cryptanalysis

An r-round linear approximation involves 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 [17,18]. 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 [17]. 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 [18]. 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 Security against Other Possible Attacks

There are some variants to the basic differential attack discussed above. Knudsen introduced the idea of a *truncated differential* [15], i.e., a differential that predicts only part of the difference (not the entire value of difference), and demonstrated that this variant may be more effective against some ciphers than the basic differential attack and may be independent of the S-boxes used [16]. Due to the fairly uniform diffusion by bit-wise permutations, we believe that truncated differentials will not be much useful in CRYPTON compared to ordinary differentials.

The *higher order differential* attack was first considered by Lai [20] and further investigated in [15,12]. Let d be the poly. degree of $(r-1)$-round output bits expressed as polynomials of plaintext bits. Then the higher order DC can find some key bits of the last round for an r-round cipher using about 2^{d+1} chosen plaintexts [12]. 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 poly. degree of output bits after 3 rounds increases to $6^3 \gg 128$. Therefore, the higher order DC on CRYPTON will be completely infeasible after 4 rounds.

There also exist some algebraic attacks using polynomial relations between ciphertexts and plaintexts. The *interpolation attack* [12] proposed by Jakobsen and Knudsen is applicable if the number of coefficients in the polynomial expression of the ciphertext is less than the size of ciphertext space. Its probabilistic variant allows to use some probabilistic non-linear relations with increased complexity [11]. The S-boxes used in CRYPTON do not allow any simple algebraic description and the bit permutation π in each round further complicates algebraic relations between S-box outputs. We thus believe that this kind of algebraic attacks cannot be applied to CRYPTON.

Another notable attack is the differential attack based on *impossible differentials* recently introduced by Biham et al. [1,2]. It seems not easy to systematically find some impossible events in block ciphers based on the SP network. Thus the applicability of this attack to CRYPTON should be further investigated in the future. Other variants of differential attacks, such as the differential-linear attack [9] and the *boomerang attack* (differential-differential style attack) [24], don't appear to better work on CRYPTON than the basic differential attack.

There are also several variants or generalizations of linear cryptanalysis. These include linear cryptanalysis using non-linear approximations [19], *generalized linear cryptanalysis* using I/O sums [7], and *partitioning cryptanalysis* [8], 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.

Finally, we note that there exists a specialized attack to SQUARE-like ciphers, the so-call *dedicated SQUARE attack* [6], which can also be applied to a reduced variant of CRYPTON (see [5]). However, this attack only uses the balancedness of XOR sum of intermediate round outputs for a set of different inputs, so its applicability is limited to at most a 6-round version of CRYPTON.

3.6 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, equivalent keys, related keys and simple relations such as the complemetation property existing in DES (for details, see e.g. [13,14]). 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 a block cipher is used as a building block for hash functions).

The key schedule of CRYPTON is designed with the above known weaknesses in mind. First remember the two step generation of round keys in CRYPTON: First, a user key of 256 bits or less is transformed into 8 expanded keys via invertible nonlinear transformations. Then, the first 4 expanded keys are used to generate round keys in even rounds and the remaining 4 in odd rounds. In each round, the expanded keys are updated by a word rotation and bit or byte rotations and then xored with distinct constants to produce round keys.

The first step of the key schedule shows that no different user keys can produce the same expanded keys and that there is little possibility of simple relations between different user keys being preserved in expanded keys. Thus we believe that there are no equivalent keys or simple relations in the CRYPTON key schedule. It is also very unlikely that there exist related keys that can be used to mount related-key differential attacks or related-key slide attacks, since a nonlinearly transformed user key is applied 6 or 7 times throughout encryption, each time being updated by rotations and constant additions.

Weak keys or semi-weak keys, if any, are usually due to the symmetry in encryption and key scheduling processes. This symmetry can be destroyed most easily by using distinct round constants in the key schedule. In CRYPTON we used different rotation amounts and round constants for each round key. So, we also believe that no such keys exist in CRYPTON.

4 Implementation and Efficiency

The overall structure of CRYPTON allows a very high degree of parallelisms. This will result in high efficiency and flexibility in both software and hardware implementations.

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:

$$SS_i[j] = \oplus_{k=0}^{3}(S_i[j] \wedge m_{i+k \bmod 4})^{\lll 8k} \text{ for } 0 \le i \le 3 \text{ and } 0 \le j \le 255.$$

We can then implement the odd round function $B = \rho_{oK}(A)$ by

$$B[j] = \oplus_{i=0}^{3} SS_{i+j \bmod 4}[a_{ij}] \oplus K[j] \text{ for } 0 \le j \le 3.$$

Similarly, the even round function can be implemented by $B = \rho_{eK}(A) = \rho_{oK}((A[1], A[0], A[3], A[2])^t)$.

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. Our optimized C code runs quite fast, giving an encryption rate of about 6.7 Mbytes/sec on Pentium Pro and about 4.4 Mbytes/sec on UltraSparc. The partial assembly code on Pentium Pro can encrypt/decrypt about 8.0 Mbytes per second, running about 20 % faster than the optimized C code. We expect that a fully optimized assembly implementation will run a little bit faster.

Language\Clocks	Key setup (enc/dec)	Encryption
In-line Asm (PP)	N/A	381
MSVC 5.0 (PP)	327 / 397	452
GNU C (US)	496 / 564	575

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

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 as a building block for hash functions 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.

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). So we can expect that CRYPTON will run quite fast on low-cost

smart cards, since all computations can be efficiently implemented only using byte operations. Also, CRYPTON will be ideal to be implemented on DSPs which have multiple execution units due to its high parallelism.

optimized in	delay (nsec)	cycles	Mbits/sec	total (cell) area
Area	18.97	7	900	51527 (18323)
Time	10.24	7	1660	74021 (28180)

Table 6. Estimated speed of CRYPTON in gate array impl.(from Synopsys)

Hardware efficiency is one of design objectives of CRYPTON. To estimate the speed in hardware, we carried out some simulations with Synonsys using a commercial 0.35 micron gate array library. The result is shown in Table 6. This table shows that we can easily achieve a Giga bits/sec in hardware only using a small amount of chip area.

5 Conclusion

We described CRYPTON version 1.0, an enhanced version of our AES proposal, and analyzed its security and efficiency. CRYPTON was designed by considering efficiency in various implementation environments. Its symmetry in encryption and decryption greatly reduces the hardware complexity. The S-boxes and linear transformations are designed by considering efficient implementations in hardware logic as well. The key scheduling algorithm runs very fast and allows efficient implementations in hardware and under limited environments.

Our preliminary analysis shows that 12-round CRYPTON is far secure against most 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 those people who helped him during the development of CRYPTON. Hyo Sun Hwang and Myung Hee Kang helped implementations in assembly and Java, and the hardware simulation was done by Eun Jong Hong. He would also like to thank the anonymous refrees for their constructive comments.

References

1. E.Biham, A.Biryukov and A.Shamir, Cryptanalysis of Skipjack reduced to 31 rounds, In *Advances in Cryptology-EUROCRYPT'99*, Springer-Verlag, 1999.

2. E.Biham, A.Biryukov and A.Shamir, Miss in the middle attacks on IDEA, Khufu, and Khafre, in *this proceedings*.
3. E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, *Journal of Cryptology*, v. 4, 1991, pp. 3-72.
4. J.Borst, Weak keys of CRYPTON, public comment submited to the NIST, 1998.
5. C.D'Halluin, G.Bijnens, V.Rijmen and B.Preenel, Attack on six rounds of CRYPTON, in *this proceedings*.
6. J.Daemen, L.Knudsen and V.Rijmen, The block cipher Square, In *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp.149-171.
7. 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.
8. C.Harpes and J.Massey, Partitioning cryptanalysis, In *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp.13-27.
9. M.Hellman and S.Langford, Differential-linear cryptanalysis, In *Advances in Cryptology-CRYPTO'94*, LNCS 839, Springer-Verlag, 1994, pp.26-39.
10. H.M.Heys and S.E.Tavares, Substitution-permutation networks resistant to differential and linear cryptanalysis, *J. Cryptology*, 9(1), 1996, pp.1-19.
11. T.Jakobsen, Cryptanalysis of block ciphers with probabilistic non-linear relations of low-degree, In *Advances in Cryptology-CRYPTO'98*, LNCS 1462, Springer-Verlag, 1998, pp.212-222.
12. T.Jakobsen and L.R.Knudsen, The interpolation attack on block ciphers, In *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp.28-40.
13. 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.
14. J.Kelsey, B.Schneier and D.Wagner, Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA, In *Information and Communications Security*, LNCS 1334, Springer-Verlag, 1997, pp.233-246.
15. L.R.Knudsen, Truncated and higher order differentials, In *Fast Software Encryption*, LNCS 1008, Springer-Verlag, 1995, pp.196-211.
16. L.R.Knudsen and T.A.Berson, Truncated differentials of SAFER, In *Fast Software Encryption*, LNCS 1039, Springer-Verlag, 1996, pp.15-26.
17. 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.
18. 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.
19. 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.
20. X.Lai, *On the design and security of block ciphers*, PhD thesis, ETH, Zurich, 1992.
21. X.Lai and J.L.Massey, Markov ciphers and differential cryptanalysis, In *Advances in Cryptology-EUROCRYPT'91*, LNCS 547, Springer-Verlag, 1991, pp.17-38.
22. C.H.Lim, CRYPTON: A new 128-bit block cipher, NIST AES Proposal, June 1998.
23. M.Matsui, Linear cryptanalysis method for DES cipher, In *Advances in Cryptology-EUROCRYPT'93*, LNCS 765, Springer-Verlag, 1994, pp.386-397.
24. D.Wagner, The boomerang attack, in *this proceedings*.