Attack on Six Rounds of Crypton

Carl D'Halluin, Gert Bijnens, Vincent Rijmen*, and Bart Preneel*

Katholieke Universiteit Leuven , ESAT-COSIC
K. Mercierlaan 94, B-3001 Heverlee, Belgium
carl.dhalluin@esat.kuleuven.ac.be
gert.bijnens@esat.kuleuven.ac.be
vincent.rijmen@esat.kuleuven.ac.be
bart.preneel@esat.kuleuven.ac.be

Abstract. In this paper we present an attack on a reduced round version of Crypton. The attack is based on the dedicated Square attack. We explain why the attack also works on Crypton and prove that the entire 256-bit user key for 6 rounds of Crypton can be recovered with a complexity of 2^{56} encryptions, whereas for Square 2^{72} encryptions are required to recover the 128-bit user key.

1 Introduction

The block cipher Crypton was recently proposed as a candidate algorithm for the AES [5]. In this paper we describe a chosen plaintext attack that works if the cipher is reduced to 6 rounds instead of the specified 12 rounds. Our attack is based on the dedicated Square attack presented in [2], but because of the differences between Square and Crypton, the attack has to be modified in several points.

Previous analysis of Crypton led to the discovery of a failure of the key scheduling, resulting in a number of weak keys [1,6]. Our attack works on a reduced version of Crypton for all keys. For a final optimisation of the attack, we exploit another feature of the key scheduling.

In Section 2 we give a short description of Crypton. We present the basic attack in Section 3. Section 4 discusses the recovery of the user key. Section 5 and Section 6 discuss the extension of the attack to five rounds, and Section 7 gives the six round attack. We conclude in Section 8.

2 Description of Crypton

The block cipher Crypton is based on Square [2]. The plaintext data is ordered in 16 bytes, which are put in a square scheme, called the *state*. If A is the state at a certain moment, the different bytes of A are called $(A)_{ij}$ with i and j varying from 0 to 3 (see Figure 1). Crypton uses 6 elementary transformations.

* F.W.O. Postdoctoral Researcher, sponsored by the Fund for Scientific Research - Flanders (Belgium)

L. Knudsen (Ed.): FSE'99, LNCS 1636, pp. 46–59, 1999.

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}

Fig. 1. Byte coordinates of the state

- $\sigma_{K_e^i}$ is a key addition (EXOR) with round key i. This operation is the same as the key addition in Square.
- π_e and π_o are linear transformations that act on columns of the state. The π-transformations operate on two bits at a time, calculating a new value by exoring the old values of two bits in corresponding positions in three different bytes of the column. These operations are the replacement of the MDS-based θ in Square. They can be implemented using four masks, denoted $M_0, \ldots M_3$.
- γ_e and γ_o are non-linear transformations that apply S-boxes to the different state bytes. They have the additional property that $\gamma_e = \gamma_o^{-1}$. These operations correspond to the single γ in Square.
- τ is a simple transposition (upper row becomes rightmost column, lower row becomes leftmost column, ...). This operation is the same as the π of Square. Note that $(D, C, B, A)^t$ means that A^t is the upper row and D^t is the lower row.

Throughout this text we use versions of Crypton with less rounds than the standard number of rounds which is 12. The standard version of Crypton is :

$$\text{Encrypt} = \phi_e \circ \rho_{eK_e^{12}} \circ \rho_{oK_e^{11}} \circ \cdots \circ \rho_{eK_e^2} \circ \rho_{oK_e^1} \circ \sigma_{K_e^0}$$

$$\text{with} \begin{cases} \phi_e = \tau \circ \pi_e \circ \tau \\ \rho_{eK} = \sigma_K \circ \tau \circ \pi_e \circ \gamma_e \\ \rho_{oK} = \sigma_K \circ \tau \circ \pi_o \circ \gamma_o \end{cases}$$

Since ϕ_e uses no key material, it is easily invertible by a cryptanalyst and therefore we do not consider it in the following text. For example a *five round version* of Crypton means in this text :

$$\text{Encrypt}_5 = \rho_{oK_e^5} \circ \rho_{eK_e^4} \circ \rho_{oK_e^3} \circ \rho_{eK_e^2} \circ \rho_{oK_e^1} \circ \sigma_{K_e^0}.$$

Unless stated otherwise the output state of round n is denoted R_n in this paper. R_0 represents the output after the initial key addition $\sigma_{K_e^0}$. PT represents the plaintext and CT represents the ciphertext. So $CT = \phi_e(R_{12})$.

3 Basic Attack: 4 Rounds of Crypton

In this section we will explain how the dedicated Square attack [2] can be modified to attack Crypton. Due to the differences between both algorithms, the attack works in a slightly different way. The final 6-round attack allows to recover the entire 256-bit user key immediately, using less computer time than the equivalent attack on Square, which only recovers a 128-bit key. This makes the attack much more significant on Crypton than on Square.

First we explain the attack on 4 rounds of Crypton and the reason why it works. The attack on 4 rounds uses approximately 2^9 chosen plaintexts and their corresponding ciphertexts. We show a way to recover a 128-bit user key without using additional plaintexts. In the next Sections a 5-round and a 6-round attack on Crypton are described, which require significantly more chosen plaintexts.

Let a Λ-set be a set of 256 states that are all different in some of the (16) state bytes (the *active*) and all equal in the other state bytes (the *passive*). The 256 elements of the Λ-set are denoted Λ_a with a varying from 0 to 255. Let λ be the set of indices of the active bytes (indices varying from 0 to 3). We have:

$$\forall a, b \in \{0 \dots 255\} : \qquad \begin{cases} (\Lambda_a)_{ij} \neq (\Lambda_b)_{ij} \text{ for } (i,j) \in \lambda \\ (\Lambda_a)_{ij} = (\Lambda_b)_{ij} \text{ for } (i,j) \notin \lambda. \end{cases}$$

We start with a Λ-set with a single active byte. From the definition it follows that this byte will take all 256 possible values {0x00, 0x01, ..., 0xff} over the different states in the Λ-set. As a consequence, the Λ-set is balanced, by which we mean:

$$\bigoplus_{a=0}^{255} (\Lambda_a)_{ij} = \texttt{0x00}, \qquad \forall i, j.$$

This is valid for the one active byte because $\bigoplus_{a=0}^{255} a = \texttt{0x00}$ and for the fifteen passive bytes because $\bigoplus_{a=0}^{255} b = \texttt{0x00}$ if b is a constant byte.

We now investigate the balancedness, the positions and the properties of the active bytes through the subsequent transformations. After each transformation we have a new set of active bytes, which is called the *state scheme*. If we say that the state scheme is balanced we mean that the transformation of the original Λ-set up to this point is still balanced. It is also important in the state scheme to know some properties of each byte e.g. this byte is a constant for each element of the Λ-set, or that byte takes every value of the set {0x00, 0x01, ..., 0xff} over the Λ-set (mathematically : $\bigcup_{a=0}^{255} (\Lambda_a)_{ij} =$ {0x00, 0x01, ..., 0xff}).

Round 0 (initial key addition) The state scheme evolution through round 0 is displayed in Figure 2. We have a single active byte which takes 256 different values over the Λ-set (full black square in the figure). The 15 other bytes are passive and therefore they have a constant value over the Λ-set.

The initial key addition $\sigma_{K_e^0}$ does not change the balancedness of the state scheme because the EXOR-sum of 256 times the same key byte cancels out.

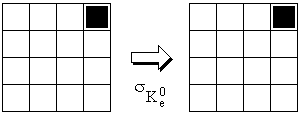

Fig. 2. State scheme through round 0

$\sigma_{K_e^0}$ cannot change the state scheme either because the EXOR-addition with a constant byte acts as a simple bijection in the set of all possible byte values {0x00, 0x01, ..., 0xff}. Hence, if we have 256 different byte values, the key addition will map them on 256 different byte values, and if we have 256 times the same byte value, the key addition will obviously map them on 256 times the same byte value.

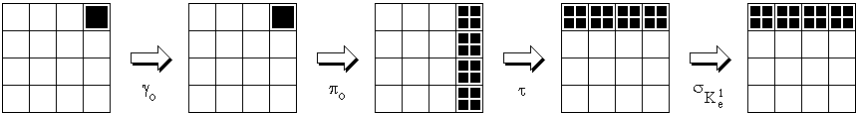

Fig. 3. State scheme through round 1

Round 1 γ_o does not change the state scheme nor the balancedness of the scheme since γ_o is bijective for each of the bytes. So 256 different byte values are mapped onto the same 256 different byte values. After π_o we have one active column. The other three columns are still passive. Now we investigate the properties of this active column. Therefore we have to take a closer look at the linear transformation π_o which acts separately on the four columns. Let A denote the input state of π_o and let B denote the output state. Now we can write π_o as

$$\begin{aligned} B &= \pi_o(A) \\ &\Updownarrow \\ (B)_{ij} &= \bigoplus_{k=0}^{3} \Big((A)_{kj} \wedge M_{(i+k) \bmod 4} \Big) , \end{aligned}$$

where $\wedge$ is the binary AND-operator and M_i with $i = 0, 1, \ldots, 3$ is a masking word, leaving 6 bits of every byte unmasked. For example $M_0 =$ 0x3fcff3fc. In state A we have one active byte which takes 256 different values over the Λ-set. Now π_o acts on this Λ-set generating one active column, in which each of the four bytes contains 6 bits of the original active byte of state A. Hence each byte of the active column of state B must take 64 different values, each occurring exactly 4 times.

Taking the EXOR of 4 times the same byte results in 0x00; state $B = \pi_o(A)$ is still balanced. In Figure 3 the new active bytes, which take 64 different values

each occurring 4 times, are displayed with a white + symbol in a black square. τ is a simple matrix transposition, which only changes the positions of the bytes, but does not change their value. Thus, τ does not change the balancedness or the properties of the state scheme. $\sigma_{K_e^1}$ does not change the balancedness or the properties either.

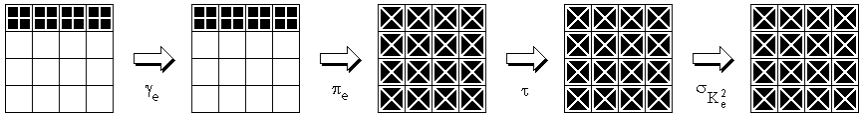

Fig. 4. State scheme through round 2

Round 2 Figure 4 shows the state scheme evolution through round 2. γ_e does not change the state scheme since γ_e will map a byte value occurring 4 times onto another byte value occurring 4 times. This does not change the EXOR-sum of the bytes, which remains `0x00`.

π_e generates 16 active bytes (all bytes of the state are now active). Due to the specific structure of π_e (similar to that of π_o), each byte of the output state of π_e takes $n \leq 64$ different values. On a fixed position in the output state of π_e, every possible byte value (from `0x00` to `0xff`) occurs either zero or a multiple of 4 times, due to π_e. On Figure 4 these bytes are shown as a white $\times$ symbol on a black square. This occurrence in multiples of 4 can be explained with the following example.

Suppose that we are working with nibbles [1]. We have a Λ-set consisting of 16 nibbles e.g. $\Lambda = \{$`1110`, `1110`, `1110`, `1110`, `1010`, `1010`, `1010`, `1010`, `1111`, `1111`, `1111`, `1111`, `0001`, `0001`, `0001`, `0001`$\}$. We now have a similar situation as the real input Λ-set of π_e. Now we look at the effect of a binary nibble mask e.g. $M =$ `1110` which leaves 3 of the 4 bits unmasked.

`1110` $\wedge$ `1110` $=$ `1110`	`1111` $\wedge$ `1110` $=$ `1110`
`1110` $\wedge$ `1110` $=$ `1110`	`1111` $\wedge$ `1110` $=$ `1110`
`1110` $\wedge$ `1110` $=$ `1110`	`1111` $\wedge$ `1110` $=$ `1110`
`1110` $\wedge$ `1110` $=$ `1110`	`1111` $\wedge$ `1110` $=$ `1110`
`1010` $\wedge$ `1110` $=$ `1010`	`0001` $\wedge$ `1110` $=$ `0000`
`1010` $\wedge$ `1110` $=$ `1010`	`0001` $\wedge$ `1110` $=$ `0000`
`1010` $\wedge$ `1110` $=$ `1010`	`0001` $\wedge$ `1110` $=$ `0000`
`1010` $\wedge$ `1110` $=$ `1010`	`0001` $\wedge$ `1110` $=$ `0000`

The table shows the effect of the bit masking. After the masking $\Lambda = \{$`1110`, `1110`, `1110`, `1110`, `1010`, `1010`, `1010`, `1010`, `1110`, `1110`, `1110`, `1110`, `0000`, `0000`, `0000`, `0000`$\}$. Now the value `1110` occurs 8 times, the values `1010` and `0000` both

[1] nibble = four-bit quantity

occur 4 times, and the other possible nibble values do not occur at all. We see that the values in the resulting Λ-set occur in multiples of 4.

If we look at the real π_e then the only thing we do is masking (leaving unmasked 6 bits of every byte) and EXOR-additions. We can generalize our conclusion from above and say that on every byte position every byte value in the output state of π_e occurs in multiples of 4 over the Λ-set. This leaves the state balanced. τ and $\sigma_{K_e^2}$ do not change the state scheme or the balancedness for reasons mentioned before.

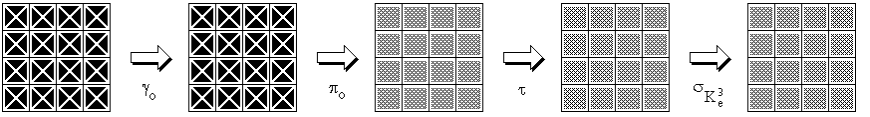

Fig. 5. State scheme through round 3

Round 3 γ_o does not change the state scheme properties or the balancedness since γ_o will map a byte value occurring a multiple of 4 times on another byte value occurring the same multiple of 4 times. This does not change the EXOR-sum of the bytes, which remains `0x00`.

After π_o all bytes are still active but the different values no longer occur in multiples of four. This destroys our structure and limits the power of the whole attack. These active bytes are displayed in Figure 5 as grey squares. Nevertheless the scheme is still balanced because of the linearity of π_o. This can be proven as follows. Let A_a denote the balanced input state of π_o for the a-th element of the Λ-set, and let B_a denote the corresponding output state. Then we have the following:

$$
\begin{aligned}
B_a &= \pi_o(A_a) \\
&\Downarrow \\
\bigoplus_{a=0}^{255}((B_a)_{ij}) &= \bigoplus_{a=0}^{255}\bigoplus_{k=0}^{3}\Big(((A_a)_{kj}) \wedge M_{(i+k) \bmod 4}\Big) \\
&= \bigoplus_{k=0}^{3}\left(\underbrace{\bigoplus_{a=0}^{255}((A_a)_{kj})}_{=\texttt{0x00}}\right) \wedge M_{(i+k) \bmod 4} \\
&= \texttt{0x00}.
\end{aligned}
$$

τ and $\sigma_{K_e^3}$ do not change the state scheme properties or the balancedness.

Round 4 The output of round 3 is balanced:

$$\bigoplus_{a=0}^{255}(R_3)_a = (0),$$

with (0) the all-zero state scheme ($(0)_{ij} = \texttt{0x00}, \forall i, j$). We drop the index a because the following formulae are valid for each element of the Λ-set. We have for the output of round 4:

$$\begin{aligned} R_4 &= \rho_{eK_e^4}(R_3) \\ &= \sigma_{K_e^4} \circ \tau \circ \pi_e \circ \gamma_e(R_3). \end{aligned}$$

Taking the inverse of this formula and using the linearity of π_e and τ leads us to

$$R_3 = \gamma_o\left(\pi_e(\tau(R_4)) \oplus \pi_e(\tau(K_e^4))\right). \tag{1}$$

Since R_4 is known, we can determine $\pi_e(\tau(R_4))$ completely. Now we guess one of the 16 bytes of $\pi_e(\tau(K_e^4))$. Using (1) we can calculate R_3 for this byte position for all 256 members of our Λ-set. If the EXOR of all these values is zero then we have found a possible value for this key byte. We can do this independently for all 16 key bytes. Experiments show that only one or two values are possible for each key byte. If we do the same again starting with a different Λ-set, we can find the 128 bits of round key K_e^4 with overwhelming probability.

4 Calculation of the User Key of 128 Bits

In this section we show how to calculate the user key of 128 bits when we know one round key of 128 bits. In the previous section we have extracted round key K_e^4. The user key can now be calculated by taking into account the key expansion:

$$\begin{aligned} (V_e[3], V_e[2], V_e[1], V_e[0])^t &= (\tau \circ \gamma_o \circ \sigma_P \circ \pi_o)((U[6], U[4], U[2], U[0])^t) \\ (V_e[7], V_e[6], V_e[5], V_e[4])^t &= (\tau \circ \gamma_e \circ \sigma_Q \circ \pi_e)((U[7], U[5], U[3], U[1])^t) \\ T_0 &= V_e[0] \oplus V_e[1] \oplus V_e[2] \oplus V_e[3] \\ T_1 &= V_e[4] \oplus V_e[5] \oplus V_e[6] \oplus V_e[7] \\ E_e[i] &= V_e[i] \oplus T_1 \text{ for } i = 0, 1, 2, 3 \\ E_e[i] &= V_e[i] \oplus T_0 \text{ for } i = 4, 5, 6, 7 \end{aligned}$$

with $U[i], i = 0, 1, \ldots, 7$ the user key and $E_e[i]$ the expanded keys. In [3] it is stated that if the user key is shorter than 256 bits it must be prepend by zero-words e.g. a 64-bit user key means $U[i] = \texttt{0x00000000}$ for $i > 1$.

We now try to calculate a 128-bit user key given K_e^4. Using appropriate shifts and constant additions we can easily find $E_e[0], E_e[1], E_e[2]$ and $E_e[3]$ given K_e^4

with the following formulae (see appendices in [3]):

$$\begin{aligned} K_e^4 &= (K_e[19], K_e[18], K_e[17], K_e[16])^t \\ K_e[16] &= E_e[0]^{\ll 8} \oplus RC_1 \\ K_e[17] &= E_e[1]^{\ll 24} \oplus RC_0 \\ K_e[18] &= E_e[2]^{\ll 16} \oplus RC_1 \\ K_e[19] &= E_e[3]^{\ll 8} \oplus RC_0. \end{aligned}$$

Here $A^{\ll i}$ denotes the left-wise bit rotation of a 32-bit word A over i positions. The problem to be solved can be stated as follows : *Given* $E_e[i]$ *for* $0 \leq i \leq 3$ *calculate* $U[i]$ *for* $0 \leq i \leq 3$ *knowing that* $U[j]$ *for* $4 \leq j \leq 7$ *are all zero.* We are able to solve this problem with a byte-wise reconstruction of the unknown values T_0 and T_1.

Rightmost byte of T_1 (byte 0) First of all we have to guess byte 0 of T_1. This enables us to calculate byte 0 of $V_e[i]$ for $i = 0, 1, \ldots, 3$. Since

$$(V_e[3], V_e[2], V_e[1], V_e[0])^t = (\tau \circ \gamma_o \circ \sigma_P \circ \pi_o)((U[6], U[4], U[2], U[0])^t),$$

we find that

$$\sigma_P \circ \gamma_e \circ \tau(V_e[3], V_e[2], V_e[1], V_e[0])^t = \pi_o((U[6], U[4], U[2], U[0])^t).$$

We know the upper row of the left side of this expression. From the right side we know something about the structure of $\pi_o((U[6], U[4], U[2], U[0])^t)$ since $U[i] =$ `0x0` for $i = 4, 5, \ldots, 7$. This structure is:

2 + + 0	0 2 + +	+ 0 2 +	+ + 0 2
+ + 0 2	2 + + 0	0 2 + +	+ 0 2 +
+ 0 2 +	+ + 0 2	2 + + 0	0 2 + +
0 2 + +	+ 0 2 +	+ + 0 2	2 + + 0

The symbols in this scheme denote 2-bit quantities in the total state. The four symbols in the same row of a sub-group form together one byte of the state. E.g., `2 + + 0` in the top left corner denotes the leftmost byte of $U[0]^t$. A `0` or a `2` respectively indicate to copy the corresponding two bits of $U[0]$ or $U[2]$. A `+` indicates to write the EXOR of the corresponding two bits of $U[0]$ and $U[2]$. The scheme can be derived by taking into account the different masks used in the linear transformation π_o (see [3]).

Byte 1 of T_1 This byte can by found by checking the second row of $\pi_o((U[6], U[4], U[2], U[0])^t)$. The four 2-bit-positions in the scheme where we have a `+` symbol in the upper and the second row must contain the same 2-bit values. This results in approximately one possible value for byte 1 of T_1.

Byte 2 of T_1 This byte can be found by checking the third row of $\pi_o((U[6], U[4], U[2], U[0])^t)$. We can calculate in advance 12 2-bit-positions of the third row of the scheme since $s_1 \oplus s_2 = s_3$ with $s_1 s_2 s_3$ a random permutation of the symbols `+ 0 2`. This also results in approximately one possible value for byte 2 of T_1.

Leftmost byte of T_1 (byte 3) Since we have the upper three rows of the scheme, we can calculate the lower row (using the same formula $s_1 \oplus s_2 = s_3$), and calculate back to the leftmost column of $(V_e[3], V_e[2], V_e[1], V_e[0])^t$. If we find four times the same value for the leftmost byte of T_1 by checking the $E_e[i]$ values, we have a possible user key. We do not expect that more than one valid user key can be found.

5 Addition of a Fifth Round at the End

In this section we add a fifth round in the end to the basic attack by guessing one column of round key K_e^5 at once. To recover K_e^4 we have to know only one of the 16 bytes of $\pi_e(\tau(R_4))$ at a time, so knowledge of one row of R_4 is sufficient. To add a fifth round to the attack we use the following formula:

$$\begin{aligned} R_5 &= \sigma_{K_e^5} \circ \tau \circ \pi_o \circ \gamma_o(R_4) \\ &\Downarrow \\ R_4 &= \gamma_e \left(\pi_o(\tau(R_5)) \oplus \pi_o(\tau(K_e^5)) \right), \end{aligned}$$

which is valid because of the linearity of π_o and τ.

If we guess a row of $\pi_o(\tau(K_e^5))$ we can calculate a single row of R_4 and a single column of $\pi_e(\tau(R_4))$. Since $R_3 = \gamma_o \left(\pi_e(\tau(R_4)) \oplus \pi_e(\tau(K_e^4)) \right)$ must be balanced, we can exclude approximately $\frac{255}{256}$ of our 2^{40} guessed key values (2^{32} for the row of $\pi_o(\tau(K_e^5))$ and 2^8 for the one byte of $\pi_e(\tau(K_e^4))$ gives 2^{40} guessed key values). This means that we have to repeat this procedure for at least 5 Λ-sets in order to find the round keys K_e^5 and K_e^4 from which we can calculate the entire 256-bit user key due to the simple key scheduling mechanism (see appendices in [3]).

6 Addition of a Fifth Round in the Beginning

In this section we add a fifth round in the beginning to the basic attack. We try to generate Λ-sets with only one active byte (taking 256 different values) at the output of round 1. We start with a pool of 2^{32} plaintexts that differ only in the byte values of the first column. We assume a value for the 4 bytes of the first column of the first roundkey. This enables us to compose a few sets.

Let A be the desired output state of π_o of the first round and let PT be the plaintext state.

$$A = (\pi_o \circ \gamma_o \circ \sigma_{K_e^0})(PT)$$

$$\begin{aligned} &\Updownarrow \\ PT &= (\sigma_{K_e^0} \circ \gamma_e \circ \pi_o)(A) \\ &= K_e^0 \oplus \gamma_e(\pi_o(A)) \end{aligned}$$

Since in $\gamma_e(\pi_o(A))$ only the first column is active, we can reuse the texts of our pool for every value of K_e^0. Given a Λ-set, we can recover the value of K_e^5 with our four round attack on rounds 2, 3, 4 and 5. We repeat the attack several times with different Λ-sets. If the values suggested for K_e^5 are inconsistent, we have made a wrong assumption for the column of K_e^0. With this method we can find K_e^0 and K_e^5, hence we can find the full 256 bits of the user key.

7 6-Round Version of Crypton

The six round attack is a combination of the two previous extensions of the basic 4-round attack. Due to the specific generation of round key K_e^6 we can make an improvement of 2^{16} on the dedicated Square attack, and recover the full 256 key bits.

We first guess 1 byte column of K_e^0 (2^{32} possibilities). For each guess we can generate some Λ-sets at the output of π_o of round 1 with the formula:

$$PT = \gamma_e(\pi_o(A)) \oplus K_e^0.$$

Addition of a round at the end requires the knowledge of a row of $\pi_e(\tau(K_e^6))$. If we know a column of K_e^0 then we also know 4 bytes of K_e^6:

$$\begin{aligned} K_e^0 &= (K_e[3], K_e[2], K_e[1], K_e[0])^t \\ K_e[0] &= E_e[0] \\ K_e[1] &= E_e[1] \\ K_e[2] &= E_e[2] \\ K_e[3] &= E_e[3] \\ K_e^6 &= (K_e[27], K_e[26], K_e[25], K_e[24])^t \\ K_e[24] &= E_e[0]^{\ll 24} \oplus RC_1 \\ K_e[25] &= E_e[1]^{\ll 24} \oplus RC_{02} \\ K_e[26] &= E_e[2]^{\ll 8} \oplus RC_1 \\ K_e[27] &= E_e[3]^{\ll 8} \oplus RC_{02} \end{aligned}$$

If we want to know a row of $\pi_e(\tau(K_e^6))$ we have to know a column of K_e^6 and we have to guess only 16 bits instead of the full 32 bits as in the Square attack if we choose the right columns of K_e^6.

This 6-round attack will recover K_e^0 and the equivalent K_e^6, but also K_e^5. From these values we can calculate $E_e[i]$ for $i = 0, 1, \ldots, 7$, hence we can calculate the entire 256-bit user key.

8 Conclusion

We have described attacks on several reduced round versions of the block cipher Crypton. Table 1 summarizes the requirements of the attacks. The 5-round (a) attack is described in section 5 and the 5-round (b) attack in section 6.

In its present form the described attack means no real threat to the full 12-round version of Crypton. However, after the discovery of weak keys [1,6] of Crypton, this is the second time that the key scheduling of Crypton is brought into discredit.

Table 1. Requirements for the described attacks on Crypton.

Attack	# Plaintexts	Time	Memory
4-round	2^9	2^9	small
5-round (a)	2^{11}	2^{40}	small
5-round (b)	2^{32}	2^{40}	2^{32}
6-round	2^{32}	2^{56}	2^{32}

References

1. J. Borst, "Weak keys of Crypton," technical comment submitted to NIST.
2. J. Daemen, L. Knudsen and V. Rijmen, "The block cipher Square," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.
3. Lim, "CRYPTON : A New 128-bit Block Cipher," available from [5].
4. Lim, "Specification and Analysis of Crypton Version 1.0," FSE '99, these proceedings.
5. NIST's AES home page, `http://www.nist.gov/aes.`
6. S. Vaudenay, "Weak keys in Crypton," announcement on NIST's electronic AES forum, cf. [5].

A Attack on Six Rounds of Crypton Version 1.0

In [4] a new version of Crypton is proposed, Crypton version 1.0. We explain briefly how to extend our results to version 1.0, which features two major changes.

1. The nonlinear transformations γ_o and γ_e use now two S-boxes instead of only one. This doesn't influence our attack
2. The key scheduling has been changed, both in the generation of the expanded keys and in the generation of the roundkeys from the expanded keys. This influences our attack, but we will see that the attack still applies.

A.1 Round Key Derivation in Version 1.0

The relation between roundkey 0 and roundkey 6 is very important for the complexity of our attack. In the new version this relation is more complex and uses a new operation $A^{\lll_b i}$, which is defined as a left-wise bit rotation of each of the four bytes of the 32-bit word A. The new calculation of roundkey 0 and roundkey 6 is:

$$\begin{array}{|ll|}\hline
K_e[00] = E_e[0] & \oplus \texttt{0x09e35996} \\
K_e[01] = E_e[1] & \oplus \texttt{0xfc16ac63} \\
K_e[02] = E_e[2] & \oplus \texttt{0x17fd4788} \\
K_e[03] = E_e[3] & \oplus \texttt{0xc02a905f} \\
K_e[24] = (E_e[3]^{\lll_b 6})^{\lll 8} & \oplus \texttt{0xa345054a} \\
K_e[25] = (E_e[0]^{\lll_b 4})^{\lll 16} & \oplus \texttt{0x56b0f0bf} \\
K_e[26] = (E_e[1]^{\lll_b 4})^{\lll 24} & \oplus \texttt{0xbd5b1b54} \\
K_e[27] = (E_e[2]^{\lll_b 6})^{\lll 8} & \oplus \texttt{0x6a8ccc83} \\\hline
\end{array}$$

Notice that if we know one column of $(K_e[03], K_e[02], K_e[01], K_e[00])^t$ then we know 16 bytes of a certain column of $(K_e[27], K_e[26], K_e[25], K_e[24])^t$ because of the double occurrence of the $\cdot^{\lll 8}$ operator in the previous table. This is the reason why our six-round attack still works on version 1.0 with the gain of 2^{16} time.

B Calculation of the User Key of 128 Bits

B.1 Generation of the Expanded Key

We show in this section how we can calculate the 128-bit user key when we know one roundkey of 128 bits. In the specifications of Crypton version 1.0 [4] the new generation of the expanded keys is as follows.

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

1. Split the user key into U and V as: for $i = 0, 1, 2, 3$,
$$U[i] = k_{8i+6} k_{8i+4} k_{8i+2} k_{8i}, \qquad V[i] = k_{8i+7} k_{8i+5} k_{8i+3} k_{8i+1}.$$
2. Transform U and V using round transformations ρ_o and ρ_e, respectively, with all-zero round keys :
$$U' = \rho_o(U), \qquad V' = \rho_e(V).$$
3. Compute 8 expanded keys $E_e[i]$ for encryption as: for $i = 0, 1, 2, 3$,
$$E_e[i] = U'[i] \oplus T_1, \qquad E_e[i+4] = V'[i] \oplus T_0,$$
where $T_0 = \bigoplus_{i=0}^{3} U'[i]$ and $T_1 = \bigoplus_{i=0}^{3} V'[i]$.

Since we know that $U[i]$ and $V[i]$ are all-zero for $i = 2, 3$ we know the lower 2 rows of U and V. Since $U' = \tau \circ \pi_o \circ \gamma_o(U)$ we can calculate T_1, T_0 and the user key by a byte-wise reconstruction of T_1.

B.2 Reconstruction of the 128-Bit User Key

We have $\pi_o \circ \gamma_o(U) = \tau(U')$ with $U = (\mathtt{0x0}, \mathtt{0x0}, U[1], U[0])^t$. Let $\gamma_o(U) = (\mathtt{b}, \mathtt{a}, \mathtt{1'}, \mathtt{0'})^t$ with $\mathtt{b} = \mathtt{0x8d63b1b1}$ and $\mathtt{a} = \mathtt{0xb18d63b1}$. The a and b values can be calculated from the definition of γ_o and from the S-boxes [4]. Now we try to find the unknown values 0' and 1'.

If we guess byte 0 of T_1 (rightmost byte of T_1) it is possible to calculate the upper row of $\pi_o \circ \gamma_o(U)$. The structure of this state is:

$\underline{0'}$	b	a	$\underline{1'}$	$\underline{1'}$	$\underline{0'}$	b	a	a	$\underline{1'}$	$\underline{0'}$	b	b	a	$\underline{1'}$	$\underline{0'}$
b	a	$\underline{1'}$	$\underline{0'}$	$\underline{0'}$	b	a	$\underline{1'}$	$\underline{1'}$	$\underline{0'}$	b	a	a	$\underline{1'}$	$\underline{0'}$	b
a	$\underline{1'}$	$\underline{0'}$	b	b	a	$\underline{1'}$	$\underline{0'}$	$\underline{0'}$	b	a	$\underline{1'}$	$\underline{1'}$	$\underline{0'}$	b	a
$\underline{1'}$	$\underline{0'}$	b	a	a	$\underline{1'}$	$\underline{0'}$	b	b	a	$\underline{1'}$	$\underline{0'}$	$\underline{0'}$	b	a	$\underline{1'}$

The rows in this scheme are counted from top to bottom starting with row 0. The symbols in the scheme denote to copy the corresponding 2-bit values of the following 32-bit values:

$$\begin{aligned}
\underline{\mathtt{0'}} &= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{0'},\\
\underline{\mathtt{1'}} &= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{1'},\\
\underline{\mathtt{a}} &= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{a},\\
\underline{\mathtt{b}} &= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{b}.
\end{aligned}$$

Since we know the upper row of the scheme (due to our initial guess of byte 0 of T_1) we can calculate byte 1 of T_1 because we can calculate 4 times 2 bits of T_1 on the positions of the second row of the scheme where we find a $\underline{\mathtt{a}}$ symbol (the symbols 1 in figure 6), because:

$$\begin{aligned}
\underline{\mathtt{a}} &= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{a}\\
&= (\mathtt{0'} \oplus \mathtt{1'} \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{b} \oplus \mathtt{b} \oplus \mathtt{a}\\
&= \underline{\mathtt{b}} \oplus (\mathtt{a} \oplus \mathtt{b}),
\end{aligned}$$

and we know $\mathtt{a} \oplus \mathtt{b}$. Now we can calculate row 1 of our scheme $\pi_o \circ \gamma_o(U)$ completely.

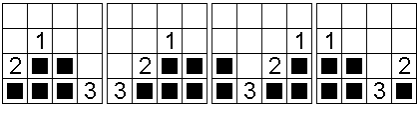

Fig. 6. 128-bit user key recovery

Next we calculate byte 2 of T_1 using the positions in row 2 where we find a $\underline{\mathtt{a}}$ symbol (the symbols 2 in figure 6). We can check the correctness of byte 0 by

checking 8 additional symbols in row 2 (the black squares in row 2 in figure 6) since we have the formulae:

$$\begin{aligned}
\underline{\mathtt{a}} \oplus \underline{\mathtt{1}'} \oplus \mathtt{b} &= (\mathtt{0}' \oplus \mathtt{1}' \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{a} \oplus (\mathtt{0}' \oplus \mathtt{1}' \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{1}' \oplus \mathtt{b} \\
&= (\mathtt{0}' \oplus \mathtt{1}' \oplus \mathtt{a} \oplus \mathtt{b}) \oplus \mathtt{0}' \\
&= \underline{\mathtt{0}'} \\
\underline{\mathtt{1}'} \oplus \underline{\mathtt{0}'} \oplus \mathtt{a} &= \ldots \\
&= \underline{\mathtt{b}}.
\end{aligned}$$

and a and b are known in advance.

Finally we calculate byte 3 of T_1 using the formula $s_0 \oplus s_1 \oplus s_2 = s_3$ with $s_0 s_1 s_2 s_3$ a permutation of the symbols $\{ \underline{\mathtt{0}'}, \underline{\mathtt{1}'}, \underline{\mathtt{a}}, \underline{\mathtt{b}} \}$. If we obtain four times the same value for byte 3 (in each of the four columns), we have found T_1. If we obtain several different values for byte 3 of T_1 the initial assumption of byte 0 of T_1 was wrong and we have to continue guessing it.

If we have found a correct value for T_1 then we have found the state U' and U completely. So we can calculate $T_0 = \bigoplus_{i=0}^{3} U[i]$ so we have state V' and finally state V. $V[2]$ and $V[3]$ should both be 0x00000000.