Cryptanalysis of C2

Julia Borghoff*, Lars R. Knudsen, Gregor Leander, and Krystian Matusiewicz*

{J.Borghoff, Lars.R.Knudsen, G.Leander, K.Matusiewicz}@mat.dtu.dk
DTU Mathematics,
Technical University of Denmark

Abstract. We present several attacks on the block cipher C2, which is used for encrypting DVD Audio discs and Secure Digital cards. C2 has a 56 bit key and a secret 8 to 8 bit S-box. We show that if the attacker is allowed to choose the key, the S-box can be recovered in 2^{24} C2 encryptions. Attacking the 56 bit key for a known S-box can be done in complexity 2^{48}. Finally, a C2 implementation with a 8 to 8 bit secret S-box (equivalent to 2048 secret bits) and a 56 bit secret key can be attacked in $2^{53.5}$ C2 encryptions on average.

Keywords. block cipher, S-box recovery, key recovery, boomerang attack, C2, Cryptomeria

1 Introduction

C2 is the short name for Cryptomeria, a proprietary block cipher defined and licensed by the 4C Entity (a consortium consisting of IBM, Intel, Matsushita and Toshiba) [3]. According to Wikipedia, "It (...) was designed for the CPRM/CPPM Digital Rights Management scheme which is used by DRM-restricted Secure Digital cards and DVD-Audio discs." [4]. 4C Entity has published a specification of C2 in [2].

C2 is a 10-round Feistel cipher with 64-bit blocks and a 56-bit key. The S-box is secret and available under license from the 4C Entity. Therefore, one might consider the S-box as part of the secret key.

A CPRM compliant device is given a set of secret device keys when manufactured. These keys are used to decrypt certain data of the media to be protected, in order to derive the media keys which have been used in the encryption of the main media data. The device keys can be revoked.

The specification of the system gives rise to several attack scenarios for C2.

1. The 56-bit key can be chosen by the attacker, who will attempt to determine the values in the secret S-box.
2. The S-box is known to the attacker, who will attempt to determine the value of a secret 56-bit key.

* The author is supported by a grant from the Danish Research Council for Technology and Production Sciences grant number 274-07-0246.

3. The 56-bit key and the S-box are unknown to the attacker, who will attempt to determine the values of both.

In this paper we attack C2 in all three scenarios. The first attack requires 2^{24} chosen plaintext queries with negligible amount of other computations in the on-line phase. The complexity of the second attack is around 2^{48} of adaptive chosen ciphertext queries and a similar amount of computations. The third attack requires $2^{53.5}$ adaptive chosen ciphertext queries.

The first attack depends on the details of the key schedule. We show that by carefully selecting the value of the 56-bit key, we can ensure that only a limited number of three S-box entries are used in the first seven rounds of encryption of a chosen plaintext. By a trail-and-error approach these three entries can be determined. Subsequently other entries of the S-box can be determined in a similar approach. The attack has been successfully implemented and recovers the whole (secret) S-box in less than 30 seconds on a standard PC.

The second and third attacks make use of so-called boomerangs [12]. A study of the differential properties for C2 shows that there exist differential characteristics with good probabilities for up to 5 rounds of the total 10 rounds. These characteristics can be extended to more rounds but with a dramatic decrease in probability. It turns out that the differential characteristics can also be specified for 5 rounds the decryption operation of C2 with similar good probabilities. The average probability of the best such 5-round differential characteristics is 2^{-11}. The differential characteristics can be used to construct a boomerang, which has an average probability of 2^{-44}. One remarkable feature of this boomerang (and others) is that it exists regardless of what S-box is used.

We successfully generated plaintext pairs following the boomerang for various keys to verify the heuristic running times and to demonstrate the practical relevance of our attack.

It should be noted that, even though it has a better overall complexity, the second attack might still be slower in practice than a simple brute force attack which can of course be nicely distributed. Actually such a brute force attack on C2 has been carried out [1] (unsuccessfully as the S-box guess turned out to be wrong).

For the third attack scenario, brute force is clearly not an option, as not only the key but the entire S-box would have to be guessed, all together 2104 bits (or 1740 if we assume S-box is a permutation).

The only cryptanalytical result on C2 we are aware of is the S-box recovery attack (scenario 1) on 8 rounds of the cipher by Weinmann [13].

The rest of this paper is organized as follows. We start with a brief description of the cipher in Section 2. We present S-box recovery attack in Section 3. Then we discuss finding differential characteristics in Section 4. We present a key recovery attack for a known S-box in Section 5 followed by a key and S-box recovery attack in Section 6. Finally, we close with some conclusions.

2 Description of C2

In this section we fix our notation and present a description of the cipher.

2.1 Notation

Throughout the paper we will use the following notation.

- L_i, R_i – left and right word after $i = 1, 2, \ldots, 10$-th round of encryption (L_0, R_0 is the plaintext)
- $\mathsf{rotl}_m(b, n)$ – cyclic rotation of m bit sequence b by n positions left
- $X_{i,j}$ – j-th bit of word X_i
- $X_{i,p..q}$ – sequence of consecutive bits $X_{i,p}, X_{i,p+1}, \ldots, X_{i,q}$, e.g. $X_{i,0..7}$ is the least significant byte of X_i.
- $X \oplus Y$, $X \boxplus Y$ – respectively, bitwise XOR, addition modulo 2^{32} of words X and Y,

2.2 The block cipher C2

C2 [2] is a block cipher with 64-bit blocks and 56-bit keys. It consists of 10 Feistel rounds, each one using a 32-bit round key rk_i. The round function can be described as

$$\begin{aligned}
L_{i+1} &= R_i \\
X &= (R_i \boxplus rk_i) \oplus \texttt{0x2765ca00} \\
Z_{i,0..7} &= S[X_{i,0..7}] \\
Z_{i,8..15} &= X_{i,8..15} \oplus \mathsf{rotl}_8(Z_{i,0..7}, 1) \\
Z_{i,16..23} &= X_{i,16..23} \oplus \mathsf{rotl}_8(Z_{i,0..7}, 5) \\
Z_{i,24..31} &= X_{i,24..31} \oplus \mathsf{rotl}_8(Z_{i,0..7}, 2) \\
R_{i+1} &= L_i \boxplus (Z_i \oplus \mathsf{rotl}_{32}(Z_i, 9) \oplus \mathsf{rotl}_{32}(Z_i, 22)), \quad i = 0, \ldots, 9
\end{aligned}$$

and is illustrated in Fig. 1. We denote by Ψ the $GF(2)$-linear function that maps bits of Y_i to the bits of U_i, this part is framed in the dotted box in the figure and the explicit equations are given in Section A.1 in the Appendix.

Note that the original reference code [2] describes it slightly differently using three byte constants, but we present here a simpler, equivalent form using only one constant $C = \texttt{0x2765ca00}$.

The key schedule produces 10 round keys $rk_0, \ldots, rk_9$ out of 56-bit master key K in the following way.

$$\begin{aligned}
K'_i &= \mathsf{rotl}_{56}(K, 17 \cdot i) \ , \\
rk_i &= K'_{i,0..31} \boxplus (S[K'_{i,32..39} \oplus i] \ll 4), \quad i = 0, \ldots, 9 \ .
\end{aligned}$$

The exact numbers of bits of the master key used in each round are also given in Table 2 in the Appendix for reference.

Both the round transformation and the key scheduling use an 8-bit secret S-box S. An example S-box provided by 4C for the purpose of validating the implementations is available online [5].

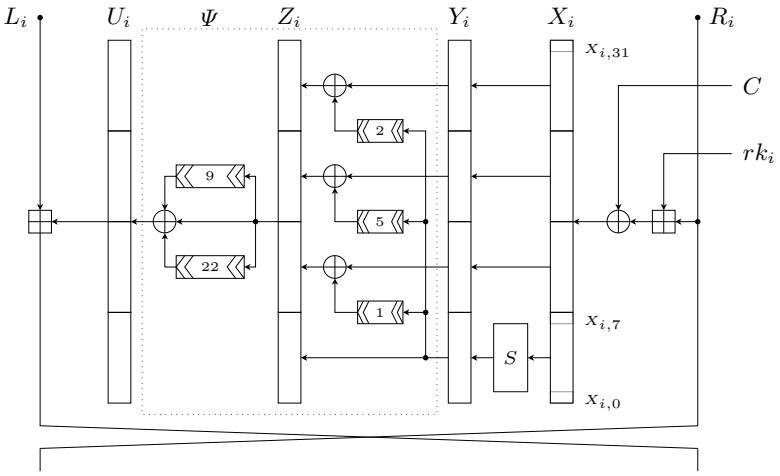

Fig. 1. Equivalent description of the round transformation of C2

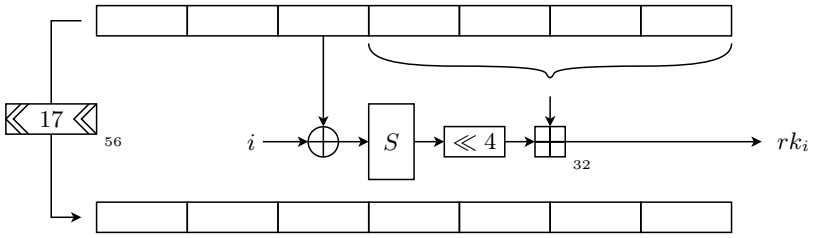

Fig. 2. One step of the key scheduling algorithm generates 32-bit round key rk_i.

3 Recovering secret S-box with chosen key attack

Our attack to recover the S-box when we are allowed to choose a encryption key is based on the observation that some keys generate only very few different inputs to the secret S-box in the key scheduling. It is easy to verify using a computer search that the smallest number of inputs generated in the key scheduling is three. An example of such a master key is

$$\texttt{0x40}, \texttt{0x84}, \texttt{0x88}, \texttt{0x40}, \texttt{0x02}, \texttt{0x80}, \texttt{0x09}$$

and the inputs generate to the S-box in rounds 1 to 10 are the following

$$\texttt{0x88}, \texttt{0x4}, \texttt{0x27}, \texttt{0x27}, \texttt{0x4}, \texttt{0x4}, \texttt{0x27}, \texttt{0x27}, \texttt{0x88}, \texttt{0x88}$$

For the attack we first fix the above key and guess the possible outputs of the S-box for the inputs `0x04`, `0x27` and `0x88`. For each possible guess we generate one plaintext that, under the assumption that our guess is correct, does not trigger

any additional entries in the secret S-box for 7 rounds. For such a plaintext, again under the assumption that our guess is correct, we know the output of the encryption process after 7 rounds, i.e. (L_7, R_7). As explained below, generating a plaintext for one fixed key guess requires approximately $2^{19.25}$ C2-encryptions and as there are 2^{24} possible values for the three entries in the secret S-box the complexity for this step is approximately $2^{43.25}$ C2-encryptions. However, this computation is independent of the actual S-box being attacked and therefore has to be done only once and is trivially parallelizable. We computed a table containing one plaintext for each guess. The actual running time was 96 hours and the size of the table is less than 400 MByte. The details are in Section 3.1.

When attacking an actual device or implementation using a secret S-box we proceed as follows. We encrypt each plaintext in the table –corresponding to one possible guess of the three S-box entries– using the device and observe the ciphertext. If our guess is correct we know the output after round 7. As explained in Section 3.2 it is possible to check if the observed ciphertext fits to our guess of the 7th round output. This test will never fail for the right guess and has a (heuristic) probability of accepting a wrong guess with a probability of 2^{-29}. Thus, on average, only the right guess will survive. Using the outlined approach we can recover three S-box entries with 2^{24} encryptions using the actual device and marginal overhead for the test.

After the first three entries have been recovered we continue in a very similar way. First, it is now easy to recover (up to) three additional entries corresponding to the inputs triggered in the last three rounds without querying the device. For all other entries we now generate plaintexts that do not trigger any unknown inputs in the first six rounds. Using the three round test explained in Section 3.2 on any possible output of the S-box in round 7 we can recover the output of the S-box in the 7th round and later recover the output of the S-box in the last three rounds again. Assuming that the inputs to the S-box in rounds $7, 8, 9$ and 10 behave randomly an estimate for the complexity (in terms of C2 encryptions) of successfully recovering the whole S-box is derived from the well known coupon collector's problem [8, Section II.7] and given by

$$C\frac{(256 \cdot H_{256})}{4} \approx 2^{19.4},$$

where H_n is the nth harmonic number and C is the complexity to generate a plaintext that fit for 6 rounds. As explained in Section 3.1, C can be upper bounded by

$$C \leq \left(\frac{256}{6}\right)^2.$$

However, it turns out that those inputs do not behave purely random and experimentally we measured a slightly higher complexity of $2^{20.2}$ as an average of 10000 tries (100 tests for 100 randomly generated S-boxes). Summarizing, when we are allowed to choose an encryption key, the S-box can be recovered with less than 2^{24} queries to the device on average. Of course, the actual running time highly depends on the encryption speed of the device, but for an implementation on a standard PC the whole S-box can be recovered in less than 30 seconds.

3.1 Generating plaintexts that fit for seven rounds

We next describe a procedure to generate a plaintext such that for known (or guessed) round keys and a set of known (or guessed) input/output pairs S the inputs to the Sbox in the first seven rounds are within this set S. First note that a naive method would be to randomly generate plaintexts and verify if the plaintext fulfills the conditions in all seven rounds. Under the assumption that those inputs behave randomly, the effort to generate such a plaintext is $(\frac{256}{|S|})^7$. For the first part of the attack, where $|S| = 3$, this is approximately $2^{44.9}$. As we have to generate not only one, but 2^{24} such plaintext the complexity of this naive approach is too high. However, it is easy to generate plaintexts that fulfil the conditions for four out of the seven rounds by construction. Then, again assuming things behave randomly, the effort is reduced to $(\frac{256}{|S|})^3$ which for $|S| = 3$ and 2^{24} plaintexts to be generated gives an overall complexity of approximately $2^{43.25}$.

Note that in the following the names of variables refers to Figure 1. To get the inputs in round 2 up to round 5 correct we first choose those inputs, i.e. we fix $X_{1,0..7}, X_{2,0..7}, X_{3,0..7}$ and $X_{4,0..7}$ to arbitrary inputs in the set S. Furthermore we choose $X_{2,8..31}$ and $X_{3,8..31}$ randomly. With this we can compute

$$\begin{aligned}
R_{1,0..7} &= (X_{1,0..7} \oplus C_{0..7}) - rk_{1,0..7} \pmod{2^8} \\
R_{2,0..7} &= (X_{2,0..7} \oplus C_{0..7}) - rk_{2,0..7} \pmod{2^8} \\
R_{3,0..7} &= (X_{3,0..7} \oplus C_{0..7}) - rk_{3,0..7} \pmod{2^8} \\
R_{4,0..7} &= (X_{4,0..7} \oplus C_{0..7}) - rk_{4,0..7} \pmod{2^8}.
\end{aligned}$$

Next, observe that for any 8 bit vector x it holds that

$$F(X \oplus (x << 23))_{0..7} = F(X)_{0..7} \oplus x$$

where F denotes the function mapping X_i to U_i. In particular we can choose bytes x and y such that

$$F(X_2 \oplus (x << 23))_{0..7} - R_{3,0..7} = R_{1,0..7} \pmod{2^8}$$

and

$$F(X_3 \oplus (y << 23))_{0..7} + R_{2,0..7} = R_{4,0..7} \pmod{2^8}.$$

Thus, if we choose

$$L'_3 = (X_2 \oplus (x << 23) \oplus C) - rk_2$$

and

$$R'_3 = (X_3 \oplus (y << 23) \oplus C) - rk_3$$

and decrypt this for three rounds to get a plaintext (L'_0, R'_0) we ensured that for this plaintext from the second until the fifth round all inputs to the Sbox are as previously fixed – and thus in the set S. We experimentally verified that the complexity of generating plaintext that also fit in the first, sixth and seventh round for $|S| = 3$ is approximately $(\frac{256}{3})^3 \approx 2^{19.25}$ as predicted by the heuristic. The overall running time to generate all 2^{24} plaintexts for each guess was distributed to 100 CPUs and took less than one hour.

3.2 A three round test

To make sure we guessed the S-box entries right we need to check if the output of the 7th round based on the guess and the ciphertext match, i.e. encrypting (L_7, R_7) with three rounds gives us the right ciphertext (L_{10}, R_{10}). This test would be trivial if we knew the S-box. However, we still can do it efficiently and with very good probability even without knowing the S-box.

Since we know the values of R_7 and R_{10}, we can compute $U_8 = R_{10} - R_7$ and going backwards through the F-function, we can determine Y_8. Since we do not know the S-box, we know only 24 msb bits of X_8. We have guessed the round key rk_8 and this means only if we knew whether a carry in the modular addition occurred or not, we would know 24 msb bits of R_8 and L_9. Now, using this knowledge and the values of L_7 and L_{10} we can determine 24 msb bits of $U_7 = R_8 - L_7$ and $U_9 = L_{10} - L_9$. Again, we do not know the carry bit so we have to test two possibilities for each of the words, either assuming a carry occurred or not. Provided that the carries are as predicted, we know exactly 24 msb bits of U_7 and U_9. Let us focus on the 7th round first. To test whether the input and the ciphertext match, we want to compare the values of U_7 obtained by the above procedure with $U'_7 = \Psi(Y_7)$, where Ψ is a $GF(2)$-linear map (marked with dotted box in Fig. 1). We cannot compare U_7 with U'_7 directly because we do not know bits $U_{7,0..7}$ and the unknown output of the S-box masks bits of U'_7. However, we can compare linear combinations of bits of U_7 and $\Psi(Y_7)$ that do not depend on any of the unknown bits $U_{7,0..7}$ and $Y_{7,0..7}$. There are 16 linear equations $\xi_j(U_7) = \xi_j(\Psi(Y_7))$ involving bits of U_7 and Y_7 that do not use any unknown bits. If the pair $(L_7, R_7), (L_{10}, R_{10})$ matches and we guessed all the carries correctly, all these equations will be satisfied. For an unrelated pair of inputs and outputs, this happens with probability 2^{-16}. The same happens for the test in round 10. We combine those two tests with a simple guessing of all the carries we need to know to obtain our testing procedure. For each of the two possible values of the carry in round 9, we test independently two possible carries in round 7 and round 10. If for any combination of these all the 32 pairs of check equations agree, we conclude the pair matches. Otherwise, we reject the pair.

This procedure always accepts right pairs $(L_7, R_7), (L_{10}, R_{10})$ as they will always produce a match in one of the tested carry combinations. To accept a wrong pair which is not coming from the encryption, all the 32 pairs of check equations would need to agree for one of the 2^3 combinations of carries. This happens with probability 2^{-29} if values are uniformly distributed. We experimentally verified that the probability is indeed around 2^{-29}. This is sufficient for us since we need to test only 2^{24} possibilities.

4 Search for S-box independent characteristics

The only components of C2 that are not linear over GF(2) are the S-box and the two modular additions. As the S-box is secret and therefore its differential behavior is unknown, we focus on characteristics not involving the S-box. Note

that if the input to the round function has zero difference in the least significant byte $R_{i,0..7}$, this zero difference cannot be destroyed by carries in the modular key addition. Thus, we can search for characteristics independent of the S-box by focusing on characteristics with $R_{i,0..7} \oplus R'_{i,0..7} = 0$.

To search for these characteristics we consider a linear model of the round function, that is, we replace the modular addition by XORs and assume that the S-box is the identity (or any linear mapping as the characteristic will be independent of this choice anyway). This linear model of the round function

$$(L_i,\ R_i) = (R_{i-1},\ L_{i-1} \oplus F(R_{i-1}, K_i))$$

can be written as $(L_i, R_i) = (L_{i-1}, R_{i-1}) \cdot M$ where M is a 64×64 matrix over $GF(2)$. Furthermore, the condition that the input difference to the S-box, i.e., the least significant byte of the output difference, shall be zero can be described as $((L, R)M)Q = 0$ where Q corresponds to the projection on the least significant 8 bits. Thus, for the linearized version of the cipher, the problem of finding a characteristic which has a zero input differences to the S-box is reduced to the problem of calculating the kernel of the linear mapping $x \rightarrow x \cdot M \cdot Q$. The kernel of the matrix $K = [Q|M \cdot Q| \cdots |M^i \cdot Q]$ contains all differences which have a zero input difference to the S-box over $i+1$ rounds. This kernel is non trivial for $i \leq 8$ implying that for version of C2 where the modular additions are replaced by XORs a characteristic over 9 rounds with probability 1 exists *independently of the S-box*.

As modular additions are not linear over $GF(2)$ we need to estimate the probability that the modular addition behaves like an XOR. Here we are interested in the two following cases.

1. The probability that the key addition behaves like an XOR
$$\Pr[(C \boxplus K) \oplus ((C \oplus \alpha) \boxplus K) = \alpha]$$
where C and K are random bit strings and α is the known difference.
2. The probability that the addition of the left half and the output of function F behaves like an XOR
$$\Pr[(L \boxplus F) \oplus ((L \oplus \alpha) \boxplus (F \oplus \beta)) = \alpha \oplus \beta]$$
where L and F random bit strings and α and β fixed known differences.

These probabilities have been studied for example in [10] where it was shown that

$$\Pr[(C \boxplus K) \oplus ((C \oplus \alpha) \boxplus K) = \alpha] = 2^{-(\mathrm{hw}(\alpha) - \mathrm{msb}(\alpha))}$$

and

$$\Pr[(L \boxplus F) \oplus ((L \oplus \alpha) \boxplus (F \oplus \beta)) = \alpha \oplus \beta] = 2^{-(\mathrm{hw}(\alpha \vee \beta) - \mathrm{msb}(\alpha \vee \beta))}$$

where $\mathrm{hw}(\alpha)$ denotes the Hamming weight of α and $\mathrm{msb}(\alpha)$ the most significant bit.

Since the probability of an XOR-characteristic depends mainly on the Hamming weight of all the intermediate input differences, we searched for characteristics minimizing it. This problem is equivalent to searching for low weight code words in the linear code generated by the matrix $B \cdot [I|M|\cdots|M^i]$ where B is the basis matrix of the kernel of K. Such an approach has been used before for finding differential characteristics in dedicated hash functions, cf. [11].

The best five round characteristic we found is

$$\Delta = (00020800\,80200100)_x \rightarrow (80200100\,00020800)_x$$

which does not require non-zero differences to the S-box in any round, and it has Hamming weight 15 over all intermediate input differences. Using the above formulas from [10] one gets a probability of 2^{-12} for independent round inputs and keys. Experimentally, the probability for randomly chosen master keys and S-boxes was even better, namely approximately $2^{-11.17}$, which is due to a differential effect which takes place inside the 5-round characteristic.

The differential characteristic can be specified also for the last five rounds of C2 and the average probability was estimated to be similar to the one for the first five rounds.

5 Key recovery attack for a known S-box

The five round characteristics described in Section 4 can be used to mount a boomerang attack on the whole cipher [12, 9, 6]. A boomerang attack is chosen plaintext and chosen ciphertext attack that involves four encryptions. We experimentally estimated (by testing 1000 random keys and multiplying probabilities of passing the first five and the last five rounds) that boomerangs exist with an average probability of $2^{-44.5}$. We observed that for all such boomerangs the pairs of texts followed the characteristic in the first round every time, but not always in later rounds. The reason one can obtain a boomerang anyway is the differential effect which is utilized also in the so-called rectangles [6]. We further observed a large variability in the probabilities over the keys and some keys were found for which the probability of the boomerang is as high as 2^{-32} but also there are keys for which no boomerangs were found. We present some of actual boomerangs we found in Table 1.

The possibility of finding boomerangs enables us to test if the differences in the first round propagate according to the characteristics. If not, we do not expect to get any boomerangs. We will use this observation to recover many bits of the first round key by a careful analysis of the carries appearing in the addition $R_0 \boxplus rk_0$. This method resembles the approach used by Contini and Yin to partially recover HMAC keys using a pseudo-collision differential for MD5 [7].

5.1 Recovering bits of the first round key

Here we are going to describe how to recover up to 22 bits of the first round key by applying the boomerang attack outlined above.

Table 1. Examples of boomerang plaintext pairs for different keys and S-boxes

S-box used	key (hex)	plaintext
AES	00 00 00 00 00 00 00 00	5707aec0 48a9c942
	00 30 20 08 00 20 28	0f42cd03 b7b5f077
	'c' 'r' 'y' 'p' 't' '0' '9'	b4b32db5 589913dc
C2 facsimile [5]	00 00 00 00 00 00 00	3af32bac 960693e1
	ee 9b 7f 2b 7c 26 cd	69676fdc 339879d4
	'c' 'r' 'y' 'p' 't' '0' '9'	d6b44956 36771c9d

Given a plaintext pair (L_0, R_1) and $(L_0 \oplus \alpha, R_0 \oplus \beta)$ from a boomerang, we know (with overwhelming probability) the difference after the first round and also know that the difference of the right halves after the modular key addition is still β. Therefore the first round key has to fulfill the equation

$$(R_0 \boxplus rk_0) \oplus ((R_0 \oplus \beta) \boxplus rk_0) = \beta. \tag{1}$$

We denote the vector of carry bits of the modular addition of R_0 and rk_0 by $c(R_0, rk_0)$, i.e.

$$R_0 \boxplus rk_0 = R_0 \oplus rk_0 \oplus c(R_0, rk_0)$$

where then

$$c_{-1} = 0 \text{ and } c_i = R_{0,i} rk_{0,i} \oplus c_{i-1} R_{0,i} \oplus c_{i-1} rk_{0,i}.$$

Using this, (1) can be rewritten as

$$R_0 \oplus rk_0 \oplus c(R_0, rk_0) \oplus R_0 \oplus \beta \oplus rk_0 \oplus c(R_0 \oplus \beta, rk_0) = \beta$$

which is equivalent to

$$c(R_0, rk_0) = c(R_0 \oplus \beta, rk_0) \tag{2}$$

and furthermore implies

$$\beta_i \; rk_{0,i} = \beta_i \; c_{i-1}.$$

Thus, whenever $\beta_i = 1$ the previous carry bit –which potentially depends on all previous key bits – equals the key bit. On the downside, Equation 2 implies that we cannot extract any key bits beyond the most significant non-zero bit of β. Using the 5 round characteristic from Section 4 we can therefore at most recover 22 bits of the first round key using (2).

In the following we describe how bits of the round key can be found one at a time. Instead of using randomly chosen plaintexts (L_0, R_0) we start by fixing the 8 least significant bits of R_0 to zero. This ensures that $c_7 = 0$. Equation (2) implies that boomerangs with this additional constraint exist iff $rk_{0,8} = 0$.

Thus, if after sufficiently many tries, we do not find any boomerang, we can conclude that $rk_{0,8} = 1$. Let us estimate the probability of making a mistake

there and wrongly assuming that $rk_{0,8} = 1$ while in reality it holds that $rk_{0,8} = 0$. If 2^{-b} is the probability of a boomerang and we make our decision after $t2^b$ tries, then the error probability can be approximated by

$$(1 - 2^{-b})^{t2^b} = \left((1 - 2^{-b})^{2^b}\right)^t \approx \left(\frac{1}{e}\right)^t .$$

After recovering $rk_{0,8}$ we modify our choice of plaintexts adaptively depending on the recovered bit $rk_{0,8}$.

First, consider the case $rk_{0,8} = 0$. Here we generate plaintext pairs where the least significant 8 bits of R_0 equal 01000000. In this case $c_7 = 0$ if and only if $rk_{0,7} = 0$ or, equivalently, boomerangs exist only when $rk_{0,7} = 0$. Thus, after sufficiently many tries, we can with a good probability recover $rk_{0,7}$.

Next, consider the case where $rk_{0,8} = 1$. Here we again fix the least significant 8 bits of R_0 to 01000000 and again $c_7 = 0$ if and only if $rk_{0,7} = 0$. However, in this case boomerangs exist only when $rk_{0,7} = 1$.

This procedure can now be applied recursively to finally recover all the key bits $rk_{0,0\ldots7}$. After those bits have been successfully recovered a very similar argument allows to recover the key bits $rk_{0,21\ldots8}$.

Assuming the average complexity for finding the boomerang is 2^{44}, the overall complexity of this procedure to recover B bits for a random key can be estimated to

$$B \cdot \left(\frac{t2^{44} + 2^{44}}{2}\right) \tag{3}$$

and the error probability is approximately

$$1 - \left(1 - \left(\frac{1}{e}\right)^t\right)^B . \tag{4}$$

If we want to recover 8 bits with a success probability of more than 0.5 we have to choose $t = 2.48$ and the effort will be $2^{47.8}$. The remaining 48 bits of the master key can then be recovered with a brute force search.

If we want to recover all 22 bits with a success probability of more than 0.99 we have to choose $t = 7.7$ and the effort will be $2^{50.59}$.

Note that for a given key it is unclear at first what the probability for the boomerang actually is. However, there are several ways to deal with this problem. On possibility is to first get an estimate of the probability by running the boomerang search for randomly selected plaintexts. Another possibility is to double the time until we decide on a key bit when no boomerang has been found step by step until the right key has been found.

6 Key and S-box recovery with chosen ciphertext attack

The attack recovering the key and the S-box is again based on the boomerang attack outlined above. As explained in Section 5.1 we can recover the least

significant 22 bits of the first round key with an average complexity of $2^{50.59}$ and an error probability less then 0.01. But turning the boomerang upside down, we can similarly recover 22 bits of the last round key with the same complexity. As explained in Section 6.1 it is possible to recover the remaining bits of these round keys and one entry of the secret S-box with an average complexity of 2^{52}. Thus with an effort of approximately 2^{53} we can recover the first and the last round key. This knowledge allows us to recover the second round key (see Section 6.2) with an average effort of $2^{45.32}$. The first two and the last round key together determine the entire master key uniquely (cf. Table 2 in the appendix). We are now in the position where we can recover additional entries of the secret S-box with an effort of 2^{44} by again applying the approach of Section 6.2. After recovering four more entries (with an effort of 2^{44+2}) of the S-box corresponding to what is triggered in the key scheduling in rounds $3, 4, 5$ and 6 we can use an attack very similar to the attack described in Section 3 to recover the remaining entries of the S-box. Namely, we guess the remaining three S-box entries triggered in the key scheduling in rounds $7, 8$ and 9. For each possible guess we generate a plaintext that does not trigger any unknown (or un-guessed) S-box entries in the first seven rounds. As we know or guessed 10 entries already the effort of generating such a plaintext is $(256/10)^3 \approx 2^{14}$. We encrypt each of those plaintexts and use the check of Section 3.2 to verify our guess. This way we will recover all 10 S-box entries used in the key scheduling and afterwards the remaining entries are recovered just as in Section 3 with a complexity less than 2^{20}. The complexity of recovering the S-box is therefore $2^{24+14} = 2^{38}$ and the overall complexity of the attack is

$$2 \cdot 2^{50.59} + 2 \cdot 2^{52} + 2^{45.3} + 2^{44+2} + 2^{38} + 2^{20} \approx 2^{53.5}$$

on average.

6.1 Recovering remaining unknown round key bits

Once we know bits $rk_{0,0..21}$ of the first round key we can recover the remaining most significant bits of the round key and the output of the S-box using the carry behaviour of the left addition $L_0 \boxplus U_0$.

If we have a boomerang plaintext, we know that the following equation is true

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha) \boxplus (U_0 \oplus \Psi(\beta))] = \texttt{0x80000000} \ ,$$

where $\alpha = \texttt{0x00020800}$, Ψ is a linear function mapping bits of Y_0 to U_0 and we have $\Psi(\beta) = \texttt{0x80020800}$. Since the difference in the most significant bit always propagates linearly as it does not induce any carries, we can focus on a simplified version of the above equation

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha) \boxplus (U_0 \oplus \alpha)] = 0 \ .$$

Using the same method as in Section 5.1 we get

$$c(L_0, U_0) = c(L_0 \oplus \alpha, U_0 \oplus \alpha)$$

and it simplifies to the condition

$$\alpha_i(L_{0,i} \oplus U_{0,i} \oplus 1) = 0 \ . \tag{5}$$

Since α has bits 11 and 17 set, (5) allows us to determine bits $U_{0,11}$, $U_{0,17}$ by trying to find boomerang plaintexts for all of the four possible combinations of $L_{0,11}$, $L_{0,17}$ in parallel. One of the choices will yield a boomerang and it contains the right combination of values of $L_{0,11}$, $L_{0,17}$ that determine the values of bits of U_0.

We have $U_{0,11} = Y_{0,0} \oplus Y_{0,11} \oplus Y_{0,21}$ and we can compute the values of $Y_{0,11}, Y_{0,21}$ because we know R_0 and the round key bits $rk_{0,0..21}$. Thus, we learn one bit of the output of the S-box ($Y_{0,0}$). Furthermore, we get another equation $U_{0,17} = Y_{0,1} \oplus Y_{0,4} \oplus Y_{0,7} \oplus Y_{0,8} \oplus Y_{0,17} \oplus Y_{0,27}$. The complete system of equations describing bits of U_0 can be found in Section A.1 of the appendix.

The same principle can be used to recover more bits. In order to do this, we need differences to appear at other bit positions in the addition $L_0 \boxplus U_0$. We can achieve this by inducing carry chains in the first addition $R_0 \boxplus rk_0$ by appropriately setting some bits of the plaintext so that the difference $\beta =$ `0x80200100` will trigger more bit flips in $R_0 \boxplus rk_0$. More precisely, we find plaintexts R_0 such that

$$(R_0 \boxplus rk_0) \oplus [(R_0 \oplus \beta) \boxplus rk_0] = \beta \oplus \gamma \ .$$

for some carry-induced difference γ. Remember that Ψ mapping Y_0 to U_0 is linear and so this induces an extra difference $\Psi(\gamma)$, so $U_0 \oplus U_0' = \Psi(\beta) \oplus \Psi(\gamma)$.

Later, we try to compensate for this extra difference in U_0 by the additional difference $\Psi(\gamma)$ in L_0. This situation can be described as

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha \oplus \Psi(\gamma)) \boxplus (U_0 \oplus \Psi(\beta) \oplus \Psi(\gamma))] = \texttt{0x80000000}$$

If this equation holds (and we know this when we find a boomerang) we have the following conditions

$$(\alpha_i \oplus \Psi_i(\gamma))(L_{0,i} \oplus U_{0,i} \oplus 1) = 0$$

which allow us to determine bits of U_0 at positions i where $\alpha_i \oplus \Psi_i(\gamma) = 1$.

Because of the effect of Ψ, each bit in γ usually requires 3 additional compensating bits of the difference in L_0 and this means we need to search for 8 boomerangs in parallel to determine the right values of L_0. After we find one, we obtain three more equations as explained before.

The complexity of this procedure depends on the configuration of carry chains we are able to induce and this depends on the round key. Assuming we can extend the difference in $\beta =$ `0x80200100` at position 8 to chains at positions 8-9, 8-10, 8-11, 8-12, 8-13, 8-14, 8-15 (so γ is `00000200`, `00000600`, `00000c00`, etc.) we get enough equations to uniquely determine the unknown bits of Y_0. We need to test 2^3 combinations of values of bits in L_0 and the total complexity is $2^3 \cdot 2^3 \cdot 2^{44} = 2^{52}$ where 2^{44} is the cost of finding the boomerang plaintext. For other configurations of secret key bits we may not be able to extend γ by one bit at a time and we will need to test more bits in L_0 each time. In that situation

we usually need to test less cases though because we learn more bits of U_0 at the same time. The exact increase in complexity very much depends on a particular case.

Note that we can always perform a search for the 13 missing bits by randomly choosing plaintext pairs (L_1, R_1) and (L'_1, R'_1) with a difference corresponding to the second round difference of our 5 round characteristic, decrypting them using all possible guesses for the missing 13 bits and searching (in parallel) a boomerang for all 2^{13} pairs. This upper bounds the complexity of recovering the remaining bits in the first round by $2^{13} \cdot 2^{44} = 2^{57}$.

A possible speed-up is to use both ends of the boomerang – if we find a boomerang plaintext we have actually two plaintexts that follow the characteristics in the first round of encryption. This can reduce the necessary number of boomerangs we need to find to completely recover the round key and the output of the S-box.

6.2 Attacking the second round

Knowing the entire first round key and one entry of the secret S-box we can (provided we fix $R_{0,0..7} = L_{1,0..7}$ to keep the input to the S-box the same) start the boomerang in the second round. For this we choose pairs (L_1, R_1) and (L'_1, R'_1) with the input difference of the best five round characteristic and compute backwards the corresponding values for (L_0, R_0) and (L'_0, R'_0). For the lower part of the boomerang we can now use our 5 round characteristic truncated to the first 4 rounds. This shortened boomerang will give pairs (L''_0, R''_0) and (L'''_0, R'''_0) with an average probability of $2^{-(2 \cdot 11 + 2 \cdot 8)} = 2^{-38}$. We cannot directly compare the corresponding pairs (L''_1, R''_1) and (L'''_1, R'''_1) as with high probability we do not know the S-box entry to decrypt in the first round. However, we can still check that the right half difference $R''_0 \oplus R'''_0$ is $\beta = \texttt{0x80200100}$ as desired. Furthermore, by exhaustively trying all possible output values for the S-box for pairs with the correct right half difference, we get an additional $32 - 8$ bit check for the left half difference. Thus, with high probability we detect correctly pairs following the boomerang characteristic.

Now, repeating the procedures outlined in Section 5.1 and 6.1 we first recover the 22 least significant bits of the second round key ($rk_{1,0..21}$) and afterwards the remaining 7 bits of the round key ($rk_{1,22..29}$) as well as one additional entry of the S-box. Note that the bits $rk_{1,29..31}$ are known from the last round key. The complexity of this is now

$$22 \cdot \left(\frac{7.7 \cdot 2^{38} + 2^{38}}{2} \right) \approx 2^{44.58}$$

for the first step and $2^3 2^3 2^{38} = 2^{44}$ for the second step.

Using this shortened boomerang described in the last section, we can moreover recover arbitrary S-box entries by fixing $R_{1,0..7}$ appropriately. The complexity for this is again 2^{44} on average.

7 Conclusions

We have shown three kinds of attacks on the block cipher C2.

When we are allowed to set the encryption key once and then encrypt plaintexts chosen by us, we can recover the secret S-box with only 2^{24} queries to the device and a reasonable precomputation phase that we have already done. The attack implemented on a PC recovers the whole S-box in less than 30 sec. Due to a low query complexity, we believe that this attack could be applied in practice to recover S-box from an actual device.

When the S-box is known, we present a boomerang attack that recovers the key with complexity equivalent to 2^{48} C2 encryptions and works for all possible S-boxes.

For the most difficult case, when both the key and the S-box are unknown and we are faced with an equivalent of at least 1740-bit long key, we present an attack that recovers both of them with complexity of around $2^{53.5}$ queries to the encryption device.

Furthermore, we show that the main strength of the cipher lies in the modular additions rather than the S-box. With modular additions replaced by XORs, one can find 9 round differentials with probability 1 and boomerangs for all 10 rounds with probability 1, both regardless of the S-box is used.

All our attacks do not assume anything about the S-box, not even its bijectiveness. Moreover, the first attack does not depend the choice of the linear mixing map Ψ used in the round function.

It is surprising that the addition of the secret S-box does not substantially improve the overall security of the design. It shows that to achieve the desired effect, the algorithm using a secret S-box must be designed very carefully. Probably a better option would be to use a longer secret key instead.

References

1. Distributed C2 brute force attack : Status page. web page, `http://www.marumo.ne.jp/c2/bf/status.html`. accessed on 12/02/2009.
2. C2 Block Cipher Specification, Revision 1.0. `http://www.4Centity.com`, 2003. used to be available online from 4C Entity, can be downloaded e.g. from: `http://edipermadi.files.wordpress.com/2008/08/cryptomeria-c2-spec.pdf`.
3. 4C Entity. Wikipedia article, `http://en.wikipedia.org/wiki/4C_Entity`, accessed on 11/02/2009.
4. Cryptomeria cipher. Wikipedia article, `http://en.wikipedia.org/wiki/Cryptomeria_cipher`, accessed on 11/02/2009.
5. 4C Entity. C2 facsimile s-box. `http://www.4centity.com/docs/C2_Facsimile_S-Box.txt`.
6. E. Biham, O. Dunkelman, and N. Keller. The rectangle attack - rectangling the serpent. In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 340–357. Springer, 2001.
7. S. Contini and Y. L. Yin. Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 37–53. Springer, 2006.

8. W. Feller. *An introduction to probability theory and its applications. Vol I.* Wiley, 3rd edition, 1968.
9. J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In *Fast Software Encryption – FSE 2000*, volume 1978 of *LNCS*, pages 75–93. Springer, 2000.
10. H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In *Fast Software Encryption – FSE 2001*, volume 2355 of *LNCS*, pages 35–45. Springer, 2002.
11. F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of step-reduced SHA-256. In *Fast Software Encryption – FSE 2006*, volume 4047, pages 126–143. Springer, 2006.
12. D. Wagner. The boomerang attack. In *Fast Software Encryption – FSE 1999*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.
13. R.-P. Weinmann. Algebraic S-Box recovery: the case of Cryptomeria. Presentation at Echternach Seminar on Symmetric Cryptography, 11/01/2008, Echternach, Luxembourg, available from `wiki.uni.lu/esc/docs/rpw_friday_algebraic_sbox_recovery.pdf`.

A Appendix

A.1 Equations describing $\Psi : Y \rightarrow U$

$$
\begin{aligned}
u_0 &= y_0 + y_1 + y_2 + y_{10} + y_{23} & u_{16} &= y_0 + y_3 + y_7 + y_{16} + y_{26} \\
u_1 &= y_1 + y_2 + y_6 + y_{11} + y_{24} & u_{17} &= y_1 + y_4 + y_7 + y_8 + y_{17} + y_{27} \\
u_2 &= y_2 + y_3 + y_7 + y_{12} + y_{25} & u_{18} &= y_0 + y_2 + y_5 + y_9 + y_{18} + y_{28} \\
u_3 &= y_0 + y_3 + y_4 + y_{13} + y_{26} & u_{19} &= y_1 + y_3 + y_6 + y_{10} + y_{19} + y_{29} \\
u_4 &= y_1 + y_4 + y_5 + y_{14} + y_{27} & u_{20} &= y_2 + y_4 + y_7 + y_{11} + y_{20} + y_{30} \\
u_5 &= y_2 + y_5 + y_6 + y_{15} + y_{28} & u_{21} &= y_0 + y_3 + y_5 + y_{12} + y_{21} + y_{31} \\
u_6 &= y_6 + y_{16} + y_{29} & u_{22} &= y_0 + y_1 + y_4 + y_{13} + y_{22} \\
u_7 &= y_7 + y_{17} + y_{30} & u_{23} &= y_1 + y_2 + y_5 + y_{14} + y_{23} \\
u_8 &= y_7 + y_8 + y_{18} + y_{31} & u_{24} &= y_2 + y_{15} + y_{24} \\
u_9 &= y_6 + y_9 + y_{19} & u_{25} &= y_7 + y_{16} + y_{25} \\
u_{10} &= y_7 + y_{10} + y_{20} & u_{26} &= y_0 + y_{17} + y_{26} \\
u_{11} &= y_0 + y_{11} + y_{21} & u_{27} &= y_1 + y_{18} + y_{27} \\
u_{12} &= y_1 + y_{12} + y_{22} & u_{28} &= y_2 + y_{19} + y_{28} \\
u_{13} &= y_2 + y_{13} + y_{23} & u_{29} &= y_3 + y_{20} + y_{29} \\
u_{14} &= y_6 + y_{14} + y_{24} & u_{30} &= y_0 + y_4 + y_7 + y_8 + y_{21} + y_{30} \\
u_{15} &= y_7 + y_{15} + y_{25} & u_{31} &= y_0 + y_1 + y_5 + y_9 + y_{22} + y_{31}
\end{aligned}
$$

A.2 Masterkey bits vs. round keys

Table 2. A list of master key bits used to generate the round keys in rounds 1 up to 10

round	master key bits used for the addition	bits input to the S-box
1	$\{0, \ldots, 31\}$	32 33 34 35 36 37 38 39
2	$\{39, \ldots, 55\} \cup \{0, \ldots, 14\}$	15 16 17 18 19 20 21 22
3	$\{22, \ldots, 53\}$	54 55 0 1 2 3 4 5
4	$\{5, \ldots, 36\}$	37 38 39 40 41 42 43 44
5	$\{44, \ldots, 55\} \cup \{0, \ldots, 19\}$	20 21 22 23 24 25 26 27
6	$\{27, \ldots, 55\} \cup \{0, 1, 2\}$	3 4 5 6 7 8 9 10
7	$\{10, \ldots, 41\}$	42 43 44 45 46 47 48 49
8	$\{49, \ldots, 55\} \cup \{0, \ldots, 24\}$	25 26 27 28 29 30 31 32
9	$\{32, \ldots, 55\} \cup \{0, \ldots, 7\}$	8 9 10 11 12 13 14 15
10	$\{15, \ldots, 46\}$	47 48 49 50 51 52 53 54