Enhanced Cryptanalysis of Substitution Cipher Chaining mode (SCC-128)

Mohamed Abo El-Fotouh and Klaus Diepold

Institute for Data Processing (LDV)
Technische Universität München (TUM)
80333 Munich Germany
mohamed@tum.de,kldi@tum.de

Abstract. In this paper, we present an enhanced cryptanalysis of the Substitution Cipher Chaining mode (SCC) [1]. In [2], SCC-128 (SCC which uses AES with 128-bit key) was broken using 5 attacks, where the authors used an active attack model (where the attacker can force the disk encryption application to re-encrypt a sector for her), the complexity of these attacks are at most 2^{40} cipher executions. In this paper, we enhance the main attack on SCC-128, this enhancement decrease the complexity of SCC-128 attacks to be at most 2^{14} cipher executions. We also cryptanalze SCC-128 in a less restrictive attack model, our attacks are upper bounded with 2^{40} cipher executions.

1 Introduction

In [1], three new disk encryption modes of operations have been introduced, the Substitution Cipher Chaining mode (SCC) which is a narrow block mode of operation that provides error propagation, and two new variants of ELEPHAT (ELEPHANT^{+} and ELEPHANT$^{\times}$) that uses SCC, where ELEPHANT is a synonym for Windows Vista's disk encryption algorithm [3]. All these modes of operations uses the Advanced Encryption Standard (AES) [4] as there block cipher.

In [2], SCC-128 (SCC which uses AES with 128-bit key) was attacked using 5 different attacks. These attacks were able to recover all the unknown keys/masks used by SCC-128. In this paper, we enhance the main attack, our enhanced attack requires less chosen plaintext and less execution time. We also present, how to break SCC-128 in a less restrictive attack model, where the attacker can only modify the ciphertext and read some decrypted plaintext.

This paper proceeds as follows: In Section 2, we briefly describe the AES. In Section 3, we present the basic idea of our enhancement to the previous attack, where we pre-processed the chosen plaintext to skip an AES round. In Section 4, we briefly review the square attack [5] and present our proposed variant of square attack. In Section 5, we describe briefly the SCC-128. In Section 6, we briefly describe the previous attacks used to break SCC-128. In Section 7, we present our optimization for the previous main attack on SCC-128. In Section 8, we modify the main attack on SCC-128, to work in a less restrictive attack model. We conclude in section 9.

2 The Advanced Encryption Standard (AES)

The 128-bit plaintext is arranged as an array of bytes. This array is called the state, the bytes of the state are treated as elements of the finite field GF(2^8). Figure 1 represents the state S as a matrix. AES has N rounds, where N equals to 10 for AES-128, 12 for AES-192, and 14 for AES-256. Table 1 present a high-level description of AES encryption.

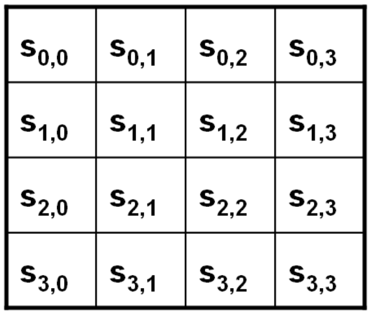

$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$

Fig. 1. AES State.

Table 1. AES Encryption.

```
Encrypt-AES(State,ExpandedKey)
    AK(State,ExpandedKey)
    for i=1 to N-1
        SB(State)
        SR(State)
        MC(State)
        AK(State,ExpandedKey + 4 × i)
    end for
    SB(State)
    SR(State)
    AK(State,ExpandedKey + 4 × N)
return State
```

An AES round is composed of the following four operations:

SubBytes (SB): a bytewise transformation that is applied on each byte of the current state, using an 8-bit to 8-bit nonlinear S-box. This transformation is the only non-linear transformation in AES and the S-box is known to have excellent differential and linear properties [6].

ShiftRows (SR): a transposition step where each row of the state is rotated to the left a certain number of steps.

MixColumns (MC): is an MDS matrix multiplication which confuses the four entries of each column of the state.

AddRoundKey (AK): each byte of the state is xored with a round key.

The MixColumns operation is omitted in the last round and an initial key addition is performed before the first round for whitening. Table 2 present a high-level description of the AES decryption, where SR^{-1}, SB^{-1} and MC^{-1} are the inverse of SR, SB and MC respectively. For full description of AES, refer to [4].

Table 2. AES Decryption.

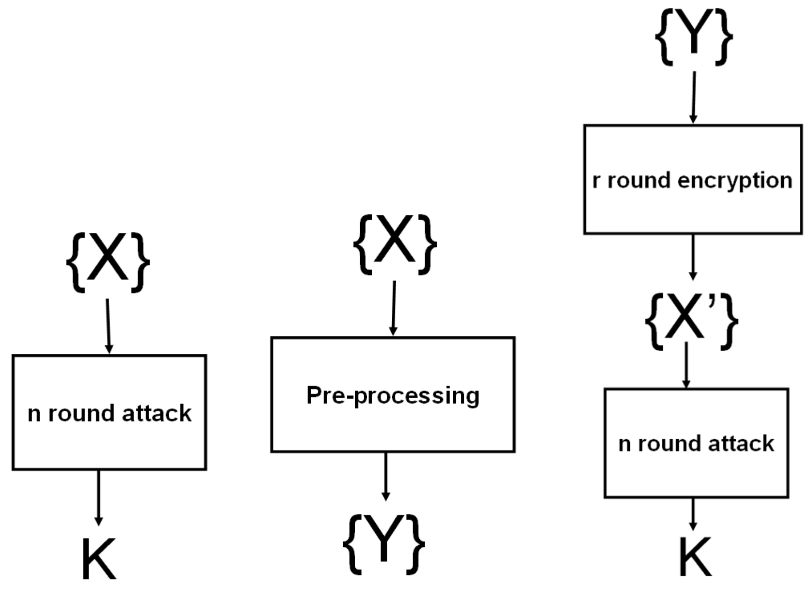
```
Decrypt-AES(State,ExpandedKey)
    AK(State,ExpandedKey + 4 $\times$ N)
    SR$^{-1}$(State)
    SB$^{-1}$(State)
    for i=N-1 to 1
        AK(State,ExpandedKey + 4 $\times$ i)
        MC$^{-1}$(State)
        SR$^{-1}$(State)
        SB$^{-1}$(State)
    end for
    AK(State,ExpandedKey)
return State
```

3 Pushdown attacks

The Pushdown attack hosts a chosen plaintext attack, where it prepares the chosen plaintext for that attack [7]. Figure 2 gives an overview on how it works, where:

1. In Fig. 2(a), an n round chosen plaintext attack is presented, that accepts the set **{X}** of chosen plaintexts to calculate the Key **K**.
2. Fig. 2(b) presents the pre-processing step of the Pushdown attack, where **{X}** is transformed to **{Y}**.
3. Fig. 2(c) presents the Pushdown attack, where after encrypting **{Y}** with r rounds, we get **{X'}**, that should be equivilant to **{X}**, in the sense that applying the n round chosen plaintext attack on **{X'}** will recover **K**. Thus, we are able to increase the number of rounds attacked by the chosen plaintext attack to **(n+r)** rounds.

In the next sub section, we will present how to perform the pre-processing step to apply the Pushdown attack on the AES (when the pre-whitening step is omitted).



(a) n round attack (b) Pre-processing (c) Pushdown attack

Fig. 2. Overview on the Pushdown attack.

3.1 Preparing a Λ^1-Set

Let a Λ-set be a set of 256 AES states that are all different in some of the state bytes (the active) and all equal in the other state bytes (the passive), recall that the state of Rijndael is a 4 × 4 byte matrix. In other words for two distinct states A and B in a Λ-set we have:
$A_{i,j} \neq B_{i,j}$ if the byte at position (i,j) is active, and
$A_{i,j} = B_{i,j}$ else, i.e the byte at position (i,j) is passive
A Λ-set with exactly k active bytes is a "Λ^k-set".

Proposition 1. *Let Γ^1 be defined by:*

$$\Gamma^1 = R^{-1}(\Lambda^1, 0) \tag{1}$$

*where $R^{-1}(X, K)$ applies an AES decryption round on the state X and Round key **K**. We claim that Γ^1-Set is a Λ^4-Set.*

Proof. The proof is straight forward. Let us follow the difference propagation, after applying AK the difference will not be changed and the result is still Λ^1-Set, but after applying MC^{-1} the difference will propagate to one column, thus the result is a Λ^4-Set. Applying SR^{-1} will not change the difference but will change the positions of the active bytes and applying SB^{-1} will not change the difference, thus the result is still Λ^4.

Note, here we use a key with all zeros as a kind of optimization and simplification of the analysis, in the sense that it has no effect on the input and can be removed from the pre-processing process.

Proposition 2. *Let Δ^1 be defined by:*

$$\Delta^1 = R(\Gamma^1, K) \tag{2}$$

*where $R(X, K)$ applies an AES encryption round on the state X and Round key **K**. We claim that Δ^1-Set is a Λ^1-Set.*

Proof. The proof is straight forward. SB, SR and MC will remove the effect of SB^{-1}, SR^{-1} and MC^{-1} respectively, thus the result will be the original Λ^1-Set. AK will not change the difference, thus the result is still a Λ^1-Set.

Figure 3 illustrates the difference propagation of a Λ^1-Set after applying an AES decryption round followed by an AES encryption round (here $S_{0,0}$ is assumed to be the active byte).

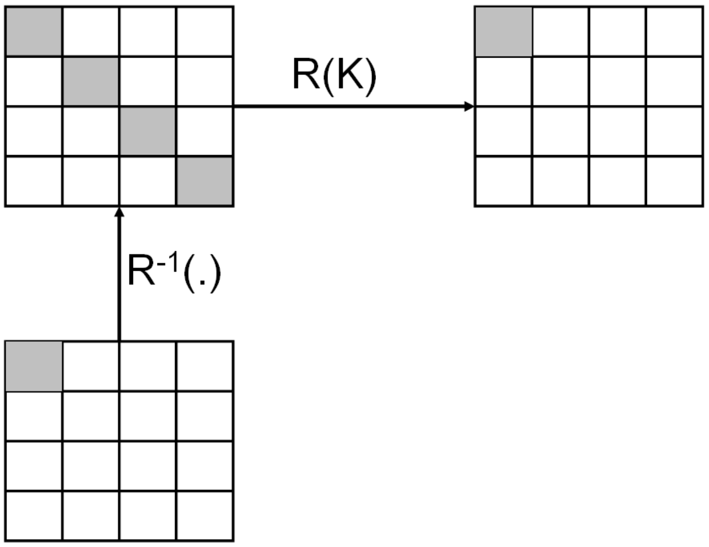


Fig. 3. An example of applying the pre-processing step on a Λ^1-Set, when $S_{0,0}$ is the active byte.

4 The Square Attack

The Square attack is a dedicated attack that exploits the byte-oriented structure of Square cipher and was published in the paper presenting the Square cipher itself [5]. This attack is also valid for Rijndael (AES) [4], as Rijndael inherits many properties from Square. The attack is a chosen plaintext attack and is independent of the specific choices of SB, the multiplication polynomial of MC and the key schedule. It is faster than an exhaustive key search for Rijndael versions of up to 6 rounds. In the following sub sections, we will present the square attack on 4 and 5 rounds AES.

4.1 Square-4

Square-4 is the basic attack, that attacks 4 rounds of Rijndael. The attacker chooses one Λ^1-set P_0 of plaintexts (where by P_i we denote the set of 256 states which are the output of the ith round). From the round properties of Rijndael we have P_1 is a Λ^4-set, P_2 is a Λ^{16}-set, and P_3 is unlikely to be a Λ-set,

As explained in [4] all the bytes of P_3 are balanced, i.e. the following property holds:

$$\textit{For all } (i,j) \in \{0,1,2,3\}^2 : \bigoplus_{A \in P_3} A_{i,j} = 0. \tag{3}$$

Hence, all bytes at the input of the 4^{th} round are balanced. This balance is in general destroyed by the subsequent application of SB. We assume the 4^{th} round to be a final round, i.e., it does not include a MC operation. Every output byte of the 4^{th} round depends on only one input byte of the 4^{th} round. Let a be the output of the 4^{th} round, b its input and k the Round Key of the 4^{th} round. We have:

$$a_{i,j} = Sbox(b_{i',j'}) \oplus k_{i,j} \tag{4}$$

By assuming a value for $k_{i,j}$, the value of $b_{i',j'}$ for all elements of the Λ-set can be calculated from the ciphertexts. If the values of this byte are not balanced over Λ, the assumed value for the key byte was wrong. This is expected to eliminate all but approximately 1 key value. This can be repeated for the other bytes of k. Since by checking a single Λ^1-set leaves only 2^{-8} of the wrong key assumptions as possible candidates, the Cipher Key can be found with overwhelming probability with only 2 Λ-sets. The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions.

4.2 Square-5

If an additional round is added at the end, we have to calculate the above value of $b_{i',j'}$ from the output of the 5^{th} round instead of the 4^{th} round. This can be done by additionally assuming a value for a set of 4 bytes of the 5^{th} Round Key. As in the case of the 4-round attack, wrong key assumptions are eliminated by verifying that $b_{i',j'}$ is not balanced. In this 5-round attack 2^{40} key values must be checked, and this must be repeated 4 times. Since by checking a single Λ-set leaves only 2^{-8} of the wrong key assumptions as possible candidates, the Cipher Key can be found with overwhelming probability with only 5 Λ-sets. The cost of this attack is 5 $\times 2^8$ chosen plaintext and requires about 2^{40} cipher executions.

4.3 Pushdown-Square-5*

In this subsection, we present an attack on 5 rounds AES (where the pre-whitening process is omitted). We use the pre-processing of the chosen plaintext in Sect. 3.1, together with Square-4 attack to deploy our Pushdown attack on AES.

Pushdown-Square-5* uses an Γ^1-Set instead of a Λ^1-set. Note that by applying an AES round on a Γ^1-Set, the result will always be a Λ^1-set for any used Round key (refer to Sect. 3.1).

Pushdown-Square-5* attack is based on Square-4 attack, where we input a Γ^1 to a 5 rounds AES, and after applying the first AES round, the input to the second round is now a Λ^1-set, from here we can apply Square-4 attack. Note that whenever a Λ^1-set is needed to be calculated the corresponding Γ^1-set is calculated instead. The complexity of Pushdown-Square-5* is the same as that of Square-4 (2^9 chosen plaintext and 2^9 cipher executions).

5 Substitution Cipher Chaining mode

SCC [1] is a dedicated mode of operation for disk encryption, that offers error propagation.

5.1 SCC Keys

The secret key in SCC is divided into three different keys (each of which can be either 128- or 256-bit):

1. **EK**: which is used to generate the expanded key, used in encrypting the blocks.
2. **TK**: which is used to encrypt the sector ID to produce the tweak.
3. **BK**: which is used to generate the **BT** (mask layers), where **BT** is an array of sixty four 128-bit words:
   – **BT** is constructed once at the initialization of SCC mode.
   – **BT** is constructed using the AES in the counter mode, where the counter is initialized with zero and **BK** is the encryption key for the counter mode.

5.2 Terminologies

The following terminologies are used to describe SCC.

IN: The input plaintext of size 4096-bit.
SID: The sector ID encoded as 64-bit unsigned integer.
GetTweak(TK,SID): Encrypts (using AES) **SID** after padding with zeros with **TK** and returns the result.
T: The tweak.
ExKey: The expanded AES key.

Expand-Key(EK): Expands the EK with the AES key setup routine and returns the result.
X_i: The ith block of text X, where a block is 128-bit.
$\bigoplus$: Bitwise xor operation.
OUT=Encrypt-AES(IN,ExKey): Encrypts IN, using the AES encryption routine with ExKey as the expanded key, and returns OUT.
Substitute(T,ExKey,i): Replaces the ith round subkeys in ExKey with T (note that: the first round of the AES is round zero and it is the pre-whitening process).
len(X): Returns the length of the string X in bits.

5.3 Design

The listing of SCC is in table 3 and it works as follows:

Table 3. SCC listing for disk encryption.

```
Encrypt-SCC(IN,EK,Keylen,TK,SID)
    T=GetTweak(TK,SID)
    ExKey=Expand-Key(EK)
    KL=len(EK)
    if(KL==128)
        x=4    y=5    z=6
    else
        x=5    y=7    z=10
    end if
    Substitute(T,ExKey,y)
    Substitute($\alpha_0$,ExKey,x)
    Substitute($\beta_0$,ExKey,z)
    OUT$_i$=Encrypt-AES(IN$_i$,ExKey)
    for i=1 to 31
        Substitute($\alpha_i$,ExKey,x)
        Substitute($\beta_i$,ExKey,z)
        $\tau$=OUT$_{i-1}$ $\oplus$ T
        Substitute($\tau$,ExKey,y)
    OUT$_i$=Encrypt-AES(IN$_i$,ExKey)
    end for
return OUT
where $\alpha_i = BT_{2\times i}$ and $\beta_i = BT_{(2\times i)+1}$
```

- The tweak **T** is calculated by encrypting the sector ID with the tweak key **TK**.
- The expanded key **ExKey** is calculated.
- the values of x, y and z are determined by the encryption key size.

- For the first block, the secret tweak **T** replaces the subkeys of the yth round, the secret 128-bit α_0 replace the subkeys of the xth round, and the secret 128-bit β_0 replace the subkeys of the zth round. Finally, the first block is encrypted by the new expanded key.
- A loop that runs 31 times (where i takes the values from 1 to 31), the secret 128-bit α_i replace the subkeys of the xth round, the secret 128-bit β_i replace the subkeys of the zth round, and τ replaces the subkeys of the yth round (where τ is calculated by xoring ciphertext of the previous block with T and it acts as the *active tweak*). Finally, the ith block is encrypted by the new expanded key.

6 Previous Attacks

6.1 Active attack model

In [2], it was assume that the adversary has access to the encrypted hard disk and can read the ciphertext stored on the hard disk, modify the ciphertext stored on the hard disk, can ask the disk encryption application to encrypt some sectors for her, and force the hard disk encryption application to re-encrypt a sector for her, this can be done if the adversary can modify the plaintext of the sector in memory and ask the disk encryption application to save it.

Following this attack model, five attacks have been mounted on SCC-128 and were able to recover all the unknown keys/masks used by SCC-128, in the next subsections we will present briefly these attacks.

6.2 The Unknowns

For SCC-128, we have the following set of unknown keys/masks: $\Pi = \{\kappa_i, \alpha_j, \beta_j, \tau_k : 0 \leq i \leq 10, i \neq x, y, z \text{ and } 0 \leq j \leq 31\}$,where κ_i is the round key of the i^{th} round and k is the sector ID.

6.3 Attack1

Let B_i^x be the ith block in the sector number x. The active tweak of the block B$_{i+1}^x$ is defined by:

$$\tau_{i+1}^x = T_i^x \oplus B_i^x \tag{5}$$

So by controlling the values of B$_i^x$, we can control the values of τ_{i+1}^x, we use our ability to modify τ_{i+1}^x to mount the square attack [5] on the last 5 rounds of the AES in SCC-128. The following procedure is used to generate a Λ^1-set:

1. Generate a set A of 256 plaintexts, so as A is a Λ^1-set.
2. For each plaintext A$_i \in A$
   (a) Set B$_i^x$= A$_i$
   (b) Read the sector x and ask the disk encryption application to re-encrypt it.

(c) Store C_{i+1}^x, where C_{i+1}^x is the ciphertext of B_{i+1}^x.

This will create a Λ^1-set as the input to the 6^{th}, which implies that we can apply the square attack on the rest 5 rounds, and obtaining the round key of the last round, from which we can calculate the rest of the round keys. The cost of this attack is 5×2^8 chosen plaintext and requires about 2^{40} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau_k : 0 \leq j \leq 31\}$, where k is the sector ID.

6.4 The other Attacks

Attack2: recovers the β mask, using a variant of the square attack presented in Sect. 10.1, this attack depends on the fact that the encryption subkeys are successfully recovered from **Attack1**. The cost of this attack is 2^{14} chosen plaintext and requires about 2^{14} cipher executions or 2^{13} chosen plaintext and requires about 2^{21} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \tau_k : 0 \leq j \leq 31\}$, where k is the sector ID.

Attack3: recovers τ_x for the sector x, this attack assumes that **Attack1** and **Attack2** were successful. The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions or 2^8 chosen plaintext and requires about 2^{16} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \tau_k : 0 \leq j \leq 31\}$, where k is the sector ID and k$\neq$ x.

Attack4: recovers the α mask, this attack assumes that **Attack1**, **Attack2** and **Attack3** were successful. The cost of this attack is a known sector plaintext and ciphertext and 64 cipher executions, this will reduce the unknown set to: $\Pi = \{\tau_k\}$, where k is the sector ID and k$\neq$ x.

Attack5: recovers the encrypt sector ID for a sector (τ_x for any sector), this attack assumes that **Attack1**, **Attack2** and **Attack4** were successful. This attack requires 1 known plaintext/ciphertext block and 2 cipher executions.

For more details about these attacks, refer to the Appendix.

7 Proposed optimization to the previous attacks

7.1 Attack1′

Let B_i^x be the ith block in the sector number x. The active tweak of the block B_{i+1}^x is defined by:

$$\tau_{i+1}^x = T_i^x \oplus B_i^x \qquad (6)$$

So by controlling the values of B_i^x, we can control the values of τ_{i+1}^x, we use our ability to modify τ_{i+1}^x to mount the Pushdown-Square-5* attack (refer to Sect. 4.3) on the last 5 rounds of the AES in SCC-128. The following procedure is used to generate a Λ^1-set:

1. Generate a set A of 256 plaintexts, so as A is a Γ^1-set.
2. For each plaintext $A_i \in A$

(a) Set B_i^x= A_i
(b) Read the sector x and ask the disk encryption application to re-encrypt it.
(c) Store C_{i+1}^x, where C_{i+1}^x is the ciphertext of B_{i+1}^x.

This will create a Γ^1-set as the input to the 6^{th}, which implies that we can apply the Pushdown-Square-5* attack on the rest 5 rounds, and obtaining the round key of the last round, from which we can calculate the rest of the round keys. The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau_k : 0 \leq j \leq 31\}$,where k is the sector ID.

With this improvement, when replacing **Attack1** with **Attack1′**, we achieved to decrease the complexity of the attacks on SCC-128 to be at most 2^{14} cipher executions (the complexity of **Attack2**).

8 Proposed Attacks

8.1 A less Restrictive Attack model

We assume that the adversary has access to the encrypted hard disk and can read the ciphertext stored on the hard disk, modify the ciphertext stored on the hard disk, can read some of the decrypted plaintext and can ask the disk encryption application to encrypt some sectors for her. Note that our assumptions are less restrictive than those in [2], where the attacker does not force the hard disk encryption application to re-encrypt a sector for her.

8.2 Pushup Attacks

The idea is to modify the chosen ciphertext, in such a way that after applying r decryption rounds, we get the original/equivalent chosen chiphertexts. This idea can increase the strength of an (n) round attack to an (n + r) round attack, as the last r rounds are bypassed [7].

Proposition 3. *Let Θ^1 be defined by:*

$$\Theta^1 = R^{-1}(R(\Lambda^1, 0), K) \tag{7}$$

*where $R(X, K)$ applies an AES encryption round on the state X and Round key **K**. We claim that Θ^1 is a Λ^1-Set.*

Proof. The proof is straight forward. Let us follow the difference propagation, after applying SB the difference will not be changed and the result is still Λ^1-Set, after applying SR the position of the active byte may change but the result is still a Λ^1-Set, but after applying MC the difference will propagate to one column, thus the result is a Λ^4-Set, applying AK has no effect on the difference. MC^{-1}, SR^{-1} and SB^{-1} will remove the effect of MC, SR and SB respectively, thus the result will be the original Λ^1-Set. After applying AK the result is still a Λ^1-Set.

8.3 Attack1$''$

Let B_i^x be the ith block in the sector number x. The active tweak of the block B$_{i+1}^x$ is defined by:

$$\tau_{i+1}^x = T_i^x \oplus B_i^x \tag{8}$$

So by controlling the values of B$_i^x$, we can control the values of τ_{i+1}^x, we use our ability to modify τ_{i+1}^x to mount the square attack on the first 6 rounds of the AES in SCC-128. The following procedure is used to generate Λ^1-sets:

1. Generate a set A of 256 plaintexts, so as A is a Θ^1-set.
2. For each plaintext A$_i \in A$
   (a) Set B$_i^x$= A$_i$
   (b) Read the plaintext of sector x.
   (c) Store P$_{i+1}^x$, where P$_{i+1}^x$ is the plaintext of B$_{i+1}^x$.

This will create a Λ^1-set as the input to the 6th decryption round, which implies that we can apply the square attack on the first 5 decryption rounds, and obtaining the round key of the first round, from which we can calculate the rest of the round keys. The cost of this attack is 5×2^8 chosen plaintext and requires about 2^{40} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau_k : 0 \leq j \leq 31\}$,where k is the sector ID.

Note that the square property on which the square attack is build on, is independent of the specific choices of SB, MC and the key schedule [4], thus the property holds in the decryption direction.

Attack2, Attack3, Atack4 and Attack5 as described in [2] can be used to recover the rest of unknowns (Note that these attacks conform to our less restrictive attack model).

9 Conclusion

In this paper, we improve the complexity of the previous attacks on SCC-128 (which were done using an active attack model), were we reduced the number of cipher executions from 2^{40} to 2^{14}. We also introduce, how to break SCC-128 using a less restrictive attack model, our attacks require at most 2^{40} cipher executions.

References

[1] M. El-Fotouh and K. Diepold. The Substitution Cipher Chaining mode. In *SECRYPT 2008*, Porto, Portugal, July 2008.

[2] M. El-Fotouh and K. Diepold. Cryptanalysis of Substitution Cipher Chaining mode (SCC). In *to appear, in 2009 IEEE International Conference on Communications (ICC 2009)*, Dresden, Germany, June 2009.

[3] N. Ferguson. AES-CBC + Elephant diffuser : A Disk Encryption Algorithm for Windows Vista. `http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a0be4bbb36e/BitLockerCipher200608.pdf`, 2006.

[4] J. Daemen and V. Rijmen. *The Design of Rijndael.* 2002.

[5] J. Daemen, L. Knudsen, and V. Rijmen. The Block Cipher Square. In *FSE '97: Proceedings of the 4th International Workshop on Fast Software Encryption*, pages 149–165, 1997.

[6] K. Nyberg and L. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8:27–37, 1995.

[7] M. El-Fotouh and K. Diepold. The Pushdown attack on AES. In *SECUREWARE 2009*, Athens, Greece, June 2009.

[8] S. Lucks. Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In *Proceedings of AES 3, NIST*, 2000.

10 Appendix

The implementation of AES has a great degree of freedom to change the order of its elementary operations, without changing the behavior f the cipher. We use the L-representation defined in [8], where L^r is defined as:

$$L^r = MC^{-1}(SB^{-1}(K^r)) \tag{9}$$

Note that by knowing L^r is equivalent to knowing K^r, where K^r is the rth round key. In table 4, we present the original round of AES and its equivalent using the L-representation of the round subkeys.

Table 4. AES original and equivalent round functions.

Round function	Equivalent function
`SB(State)`	`SB(State)`
`SR(State)`	`AK(State,L`^{r}`)`
`MC(State)`	`SR(State)`
`AK(State,K`^{r}`)`	`MC(State)`
`return State`	`return State`

10.1 Square attack for L-representation

The square attack in [4] assumes that the last round is a final round (i.e. there is no MC function). After recovering the encryption key EK, what is left is to attack three rounds of AES with independent subkeys, and all these round are full rounds (as they are in the middle of the cipher). By using the L-representation as in [8], we can re-state the square attack to work on full rounds.

The adversary chooses one Λ^1-set P_0 of plaintexts (where by P_i we denote the set of 256 states which are the output of the ith round). As explained in [4] all the bytes of P_3 are balanced, i.e. the balance property holds (3). P$_4$ is the set of 256 ciphertexts the adversary learns. Let L^4 be the L-representation of K^4 (9). The adversary can calculate the set Q$_4$ in between P$_3$ and P$_4$ by applying MC^{-1} and

SR^{-1} on the elements of P_4, where for any $X_i \in \mathrm{P}_4$ the corresponding element is $Z_i \in \mathrm{Q}_4$. So we invert the fourth round step by step, by inverting the MC function , inverting the SR function, add (a possible choice for) the key $\mathrm{L}^4_{i,j}$ and invert the SR function. If our guess $a \in \{0,1\}^8$ for $\mathrm{L}^4_{i,j}$ is correct, the set of bytes $\mathrm{SB}^{-1}(\mathrm{Z}_{i,j} \oplus a)$ is balanced, it is estimated that this guess will eliminate all the wrong guesses but one. So, we can construct an expected number of about 2^{16} candidates of L^4.

Each candidate corresponds with a unique choice of the key of the last round. We can try another set of Λ^1-set to eliminate the wrong candidates , or just use exhaustive key search over all the key candidates using the same 256 known pairs of plaintext and ciphertext as before. With overwhelming probability, either approaches uniquely determined the last round key. The memory requirement of this attack is low and need either 2^9 chosen plaintext and 2^9 cipher executions or 2^8 chosen plaintext and 2^{16} cipher executions.

10.2 Attack2

Generate a set A of 256 plaintexts, so as A is a Λ^1-set. Apply two decryption AES round function using the recovered κ_2 and κ_1, then xor the result with κ_0. Name the resulting set D. Note that encrypting the set D will result in a Λ^1-set as the input for the third round.

The attack works as follows, for each block B^x_i in a sector x:

- Use D as its input and encrypt that block.
- Decrypt the ciphertext till the 7^{th} round with the recovered round keys, name this set E.
- Apply the attack in Sect. 10.1, where the set A is the plaintext and the set E is the ciphertext.
- The recovered round key is β_i.

The cost of this attack is 2^{14} chosen plaintext and requires about 2^{14} cipher executions or 2^{13} chosen plaintext and requires about 2^{21} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \tau_k : 0 \leq j \leq 31\}$, where k is the sector ID.

10.3 Attack3

Generate a set A of 256 plaintexts, so as A is a Λ^1-set. Apply one decryption AES round function using the recovered κ_1, then xor the result with κ_0. Name the resulting set C. Note that encrypting the set C will result in a Λ^1-set as the input for the second round.

The attack works as follows, for any block B^x_i in a sector x:

- Use C as its input and encrypt that block.
- Decrypt the ciphertext till the 8^{th} round with the known round keys, name this set D.

– Apply the attack in Sect. 10.1, where the set A is the plaintext and the set D is the ciphertext.
– The recovered round key is τ_x.

The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions or 2^8 chosen plaintext and requires about 2^{16} cipher executions and this will reduce the unknown set to: $\Pi = \{\alpha_j, \tau_k : 0 \leq j \leq 31\}$, where k is the sector ID and k$\neq$ x.

10.4 Attack4

For the sector x, the set of unknowns is: $\Pi = \{\alpha_j : 0 \leq j \leq 31\}$, which can be recovered directly, by the following procedure:

– For each block B_i^x:
  1. Encrypt the plaintext X_i using the first three rounds and the known round keys to get Y_i.
  2. Apply SB, SR and MC on Y_i to produce W_i.
  3. Encrypt X_i using SCC to get A_i.
  4. Decrypt A_i using four rounds using the known round keys to get B_i.
  5. Apply an inverse AES round on B_i using β_i as the round key to get C_i.
  6. Calculate $TT_i^x = B_{i-1}^x \oplus \tau_i \ : \ B_{-1}^x = 0$, use TT_i^x as the key of the 5^{th} round.
  7. Apply an inverse AES round on C_i using TT_i^x as the round key to get D_i.
  8. $\alpha_i = D_i \oplus W_i$

Now we have all the unknowns of the sector x, this attack requires only a known sector plaintext and ciphertext and 64 cipher executions, this will reduce the unknown set to: $\Pi = \{\tau_k\}$, where k is the sector ID and k$\neq$ x.

10.5 Attack5

To recover the encrypt sector ID for a sector, we use the following procedure:

1. Encrypt a known plaintext X_i to get A_i.
2. Encrypt X_i using four rounds and the known round keys and α_i to get Y_i.
3. Apply SB, SR and MC on Y_i to produce W_i.
4. Decrypt A_i using five rounds with the known round keys and β_i to get B_i.
5. Calculate $TT_i^x = B_i \oplus W_i$
6. Calculate $\tau_x = TT_i^x \oplus B_{i-1}^x : B_{-1}^x = 0$

This attack requires 1 known plaintext/ciphertext block and 2 cipher executions.