Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys

Stefan Lucks*

Theoretische Informatik
University of Mannheim, 68131 Mannheim A5, Germany
e-mail: lucks@th.informatik.uni-mannheim.de

Abstract. The authors of Rijndael [3] describe the "Square attack" as the best known attack against the block cipher Rijndael. If the key size is 128 bit, the attack is faster than exhaustive search for up to six rounds. We extend the Square attack on Rijndael variants with larger keys of 192 bit and 256 bit. Our attacks exploit minor weaknesses of the Rijndael key schedule and are faster than exhaustive search for up to seven rounds of Rijndael.

1 Introduction

The block cipher Rijndael [3] has been proposed as an AES candidate and was selected for the secound round. It is a member of a fast-growing family of Square-like ciphers [2–6].

Rijndael allows both a variable block length of $M * 32$ bit with $M \in \{4, 6, 8\}$ and a variable key length of $N * 32$ bit, N an integer. In the context of this paper we concentrate on $M = 4$, i.e., on a block length of 128 bit, and on $N \in \{4, 6, 8\}$, i.e., on key sizes of 128, 192, and 256 bit. We abridge these variants by RD-128, RD-192 and RD-256. The number R of rounds is specified to be $R = 10$ for RD-128, $R = 12$ for RD-192, and $R = 14$ for RD-256. In the context of this paper, we consider reduced-round versions with $R \leq 7$.

The authors of Square [2] described the "Square attack", a dedicated attack exploiting the byte-oriented structure of Square. The attack works for Square reduced to six rounds and is applicable to Rijndael and other Square-like ciphers as well [3, 4, 1]. This paper deals with extensions of the Square-attack for RD-192 and RD-256.

In Section 2, we shortly describe Rijndael, leaving out many details and pointing out some properties relevant for our analysis. Section 3 deals with the Square attack for up to six rounds of Rijndael, originating from

* Supported by DFG grant Kr 1521/3-1.

[2, 3]. In Sections 4–6 we describe attacks for seven rounds of Rijndael. The attack in Section 4 and its analysis is valid for all versions of Rijndael, while the attacks in Section 5 and Section 6 are dedicatedly for Rijndael-256 and Rijndael-192, exploiting minor weaknesses of the Rijndael key schedule. We give final comments and conclude in Section 7.

2 A Description of Rijndael

Rijndael is a byte-oriented iterated block cipher. The plaintext (a 128-bit value) is used as initial state, the state undergoes a couple of key-dependent transformations, and the final state is taken as the ciphertext. A state $A \in \{0,1\}^{128}$ is regarded as a 4×4 matrix $(A_{i,j})$, $i, j \in \{0, 1, 2, 3\}$ of bytes (see Figure 1). The four columns of A are $A_i = (A_{0,j}, A_{1,j}, A_{2,j}, A_{3,j})$.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Fig. 1. The index positions (i, j) for a 4*4 matrix of bytes.

Given the initial state, R rounds of transformations are applied. Each round can be divided into several elementary transformations.

By the *key schedule*, the key, a $(N * 32)$-bit value with $N \in \{4, 6, 8\}$, is expanded into an array $W[\cdot]$ of $4(R+1)$ 32-bit words $W[0], \ldots, W[4(R+1)-1]$. Four such words $W[4r+j]$ with $j \in \{0, 1, 2, 3\}$ together are used as r-th "round key" K^r, with $r \in \{0, \ldots, R\}$. Like the state, we regard a round key K^r as a $4 * 4$ matrix of bytes $K^r_{i,j}$ with four columns $K^r_j = W[4r + j]$ for $j \in \{0, 1, 2, 3\}$.

2.1 The Elementary Transformations of Rijndael

Rijndael uses four elementary operations to transform a state $A = (A_{i,j})$ into a new state $B = (B_{i,j})$, see also Figure 2:

1. The *byte substitution* (BS): $B_{i,j} := S(A_{i,j})$ for $i, j \in \{0, 1, 2, 3\}$. Here, S denotes a permutation over $\{0,1\}^8$, i.e., S^{-1} is defined with $A_{i,j} = S^{-1}(B_{i,j})$.
2. The *shift row* operation (SR), a cyclic shift of bytes: $B_{i,j} := A_{i,(j+i) \bmod 4}$.
3. The *mix column* transformation (MC). Each column A_i of state A is transformed via a linear transformation μ over $\{0,1\}^{32}$, i.e. $B_i := \mu(A_i)$ for $i \in \{0, 1, 2, 3\}$. Also, μ is invertible.
   An input $X \in \{0,1\}^{32}$ for μ can be seen as a vector $X = (X_0, X_1, X_2, X_3)$ of four bytes. Consider $X' = (X'_0, X'_1, X'_2, X'_3)$ to be different from X in exactly k bytes ($1 \le k \le 4$), i.e.
   $$k \;=\; \left|\{\; i \in \{0,1,2,3\} \mid X_i \neq X'_i \;\}\right|.$$
   Then $Y = \mu(X)$ and $Y' = \mu(X')$ are different in at least $5 - k$ of their four bytes. The same property holds for the inverse μ^{-1} of μ.
4. The *key addition* (KA). The r-th round key $K^r = (K^r_{i,j})$ is added to the state A by bit-wise XOR: $B_{i,j} := A_{i,j} \oplus K^r_{i,j}$.

Note that all elementary transformations of Rijndael are invertible.

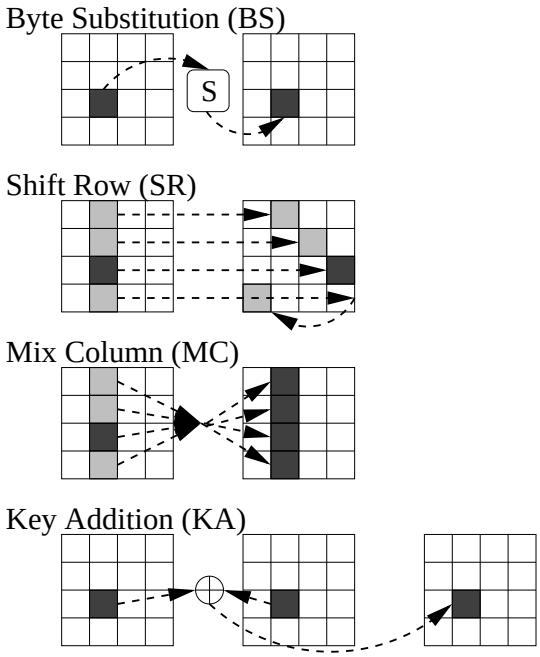

Fig. 2. The four elementary transformations of Rijndael.

2.2 The Rijndael Round Transformation

For $r \in \{0, \ldots, R\}$, the round key K^r consists of the expanded key words $W[4r], \ldots, W[4r+3]$. The structure of Rijndael is defined as follows[1]:

1. $S :=$ plaintext;
2. KA (S, K^0); ($*$ add round key 0 before the first round $*$)
3. for $r := 1$ to R do: ($*$ run through round 1, 2, ..., R $*$)
  4. $S :=$ BS(S); ($*$ byte substitution $*$)
  5. $S :=$ SR(S); ($*$ shift row $*$)
  6. $S :=$ MC(S); ($*$ mix column $*$)
  7. $S :=$ KA(S, K^r); ($*$ add round key r $*$)
8. ciphertext $:= S$.

Steps 4–7 are the "standard representation" of the Rijndael round structure. The implementor of Rijndael has a great degree of freedom to change the order the elementary operations are done – without changing the behavior of the cipher. (We refer the reader to the description of the "algebraic properties" and the "equivalent inverse cipher structure" for details [3, Section 5.3].) We describe one alternative representation of the round structure. As an "alias" for the r-th round key K^r we use the value

$$L^r = \mathrm{SR}^{-1}(\mathrm{MC}^{-1}(K^r)). \tag{1}$$

Accordingly, we distinguish between the "L-representation" L^r of a round key and its "K-representation". Knowing L^r is equivalent to knowing K^r, and knowing a column K^r_j of K^r is equivalent to knowing four bytes of L^r, see Table 1.

known column of K^r	known bytes $L^r_{i,j}$ of L^r
K^r_0	$(i,j) \in \{(0,0),(1,3),(2,2),(3,1)\}$
K^r_1	$(i,j) \in \{(0,1),(1,0),(2,3),(3,2)\}$
K^r_2	$(i,j) \in \{(0,2),(1,1),(2,0),(3,3)\}$
K^r_3	$(i,j) \in \{(0,3),(1,2),(2,1),(3,0)\}$

Table 1. Known columns of a key in K-representation and the corresponding known key bytes in L-representation.

[1] Actually, the authors of Rijndael [3] specify an exception: in the last round, the MC-operation is left out. As was stressed in [3], this modification does not strengthen or weaken the cipher. In the current paper, we assume for simplicity that the last round behaves exactly like the other rounds.

The following describes a functionally equivalent round structure for Rijndael, see also Figure 3.

4. $S := \mathrm{BS}(S)$; ($*$ byte substitution $*$)
5. $S := \mathrm{KA}(S, L^r)$; ($*$ add round key, given in L- representation $*$)
6. $S := \mathrm{SR}(S)$; ($*$ shift row $*$)
7. $S := \mathrm{MC}(S)$; ($*$ mix column $*$)

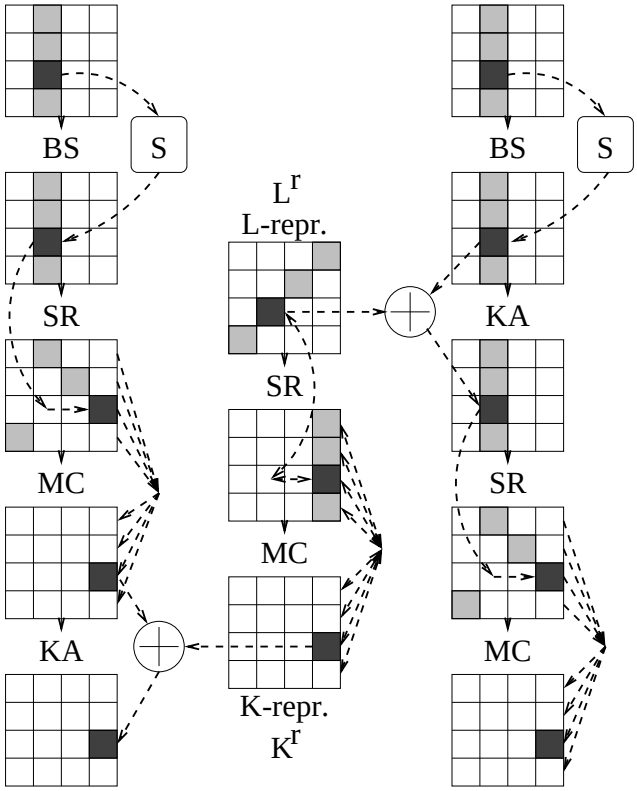

Fig. 3. The Structure of a Rijndael round.
Left: The standard representation of the Rijndael round transformation
Middle: The round key – changing between K-representation and L-representation
Right: The alternative representation of the Rijndael round transformation

2.3 The Rijndael Key Schedule

The key schedule is used to generate an expanded key from a short (128–256 bit) "cipher key". We describe the key-schedule using word-wise oper-

ations (where a word is a 32-bit quantity), instead of byte-wise ones. The cipher key consists of N 32-bit words, the expanded key of $4*(R+1)$ such words $W[\cdot]$. The first N words $W[0], \ldots, W[N-1]$ are directly initialised by the N words of the cipher key.

For $k \in \{1, 2, \ldots\}$, const(k) denotes fixed constants, and $f, g : \{0,1\}^{32} \to \{0,1\}^{32}$ are nonlinear permutations.[2] For $i \in \{N, \ldots, 4*(R+1)-1\}$ the words $W[i]$ are defined recursively:

$$\begin{array}{l} \text{If } (i \bmod N) = 0 \\ \quad \text{then } W[i] := W[i-N] \oplus f(W[i-1]) \oplus \text{const}(i \operatorname{div} N) \\ \quad \text{else if } ((N > 6) \text{ and } (i \bmod N) = 4) \\ \qquad \text{then } W[i] := W[i-N] \oplus g(W[i-1]) \\ \qquad \text{else } W[i] := W[i-N] \oplus W[i-1]. \end{array} \tag{2}$$

Note that two words $W[i-1]$ and $W[i-N]$ suffice to compute the word $W[i]$. Similarly, we can go backwards: Given two words $W[i]$ and $W[i-1]$, we can compute $W[i-N]$. (This will be useful for our attacks below.) Hence, any N consecutive words $W[k], \ldots, W[k+N-1]$ of the expanded key suffice to efficiently generate the compete expanded key and thus to completely break Rijndael.

3 The Square Attack for Rijndael

In this section we describe the dedicated Square-Attack for Rijndael. More details can be found in [2, 3]. We start with a simple attack on four rounds and extend the simple attack by an additional round at the beginning and another one at the end. This leads to the "Square-6" attack for six rounds of Rijndael. Analysing the performance of our attacks with respect to RD-192 and RD-256 is delayed until the end of this section.

3.1 Attacking Four Rounds – the Simple Attack

To describe the attack we need the notion of a "Λ-set", i.e., a set of 2^8 states that are all different in some of their $4*4$ bytes (the "active" bytes), and all equal in the other ("passive") bytes. In other words, for two distinct states A and B in a Λ-set we always have

$$\begin{array}{ll} A_{i,j} \neq A_{i,j} & \text{if the byte at position } (i,j) \text{ is active, and} \\ A_{i,j} = B_{i,j} & \text{else, i.e., if the byte at } (i,j) \text{ is passive.} \end{array}$$

[2] We omit the definition of f and g, but we point out that the four functions f, g, f^{-1} and g^{-1} are fixed in the definition of Rijndael and can be computed efficiently.

A Λ-set with exactly k active bytes is a "Λ^k-set".

The adversary chooses one Λ^1-set P_0 of states (plaintexts). By P_i we denote the sets of 2^8 states which are the output of round i. P_1 is a Λ^4-set, all four active bytes in the the same column. P_2 is a Λ^{16}-set. P_3 is unlikely to be a Λ-set. But, as explained in [2, 3], all the bytes of S_3 are "balanced", i.e., the following property holds:

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

Recall that we consider a four-round attack, i.e., P_4 is the set of 2^8 ciphertexts the adversary learns. It is unlikely that the bytes of P_4 are balanced, but the balancedness of the bytes of P_3 can be exploited to find the fourth round key K^4. Let L^4 be the L-representation of K^4, cf. Equation (1). The attack defines a set Q_4 "in between"[3] P_3 and P_4:

1. For $X \in P_4$:
   $Y := \mathrm{MC}^{-1}(X)$;
   $Z := \mathrm{SR}^{-1}(Y)$.
   Denote the set of 2^8 states Z by Q_4.
2. For all $(i,j) \in \{0,1,2,3\}^2$:
   for $a \in \{0,1\}^8$:
   $b(a) := \bigoplus_{Z \in Q_4} S^{-1}(Z_{i,j} \oplus a)$;
   if $b(a) \neq 0$ then conclude $L^4_{i,j} \neq a$.

In short, we invert round four step by step: invert the mix column operation, invert the shift row operation, add (a possible choice for) the key byte $L^4_{i,j}$ and invert the byte substitution. If the guess $a \in \{0,1\}^8$ for the key $L^4_{i,j}$ is correct, the set of 2^8 bytes $S^{-1}(Z_{i,j} \oplus a)$ is balanced, i.e., $b(a) = 0$. But if our guess a' for $L^4_{i,j}$ is wrong, we estimate $b(a') = 0$ to hold with only a probability of 2^{-8}. Thus, on the average two candidates for for each byte $L^4_{i,j}$ are left – the correct byte and a wrong one. We can easily reconstruct an expected number of less than 2^{16} candidates for L^4.

Each candidate corresponds with a unique choice for the 128-bit cipher key of RD-128. To find the cipher key, we may either choose a second Λ^1-set of plaintexts, or just use exhaustive search over all key candidates, using the same 2^8 known pairs of plaintext and ciphertext as before. With overwhelming probability, either approach uniquely determines the

[3] In general, we regard Q_5 to be a set of states "in between" P_{r-1} and P_r. Note that converting a state in P_r into its counterpart in P_4 does not depend on the key and can be just like converting a round key from its K-representation into its L-representation, similarly to Equation (1).

cipher key, using few memory and an amount of work determined by step 2, i.e., about 2^{20} byte-wise XOR-operations. Note that the first approach needs twice as many chosen plaintexts as the second one.

3.2 An Extension at the End

As suggested in [2, 3], the above basic attack can be extended by one additional round at the beginning and another round at the end. We start with extending the additional round at the end.

Let P_0 be chosen as above. By P_5, we denote the set of 2^8 outputs of round 5. Similar to Q_4 above, the adversary can find Q_5 by applying MC^{-1} and applying SR^{-1}. Given the set Q_5, we can compute P_3 by inverting $1\frac{1}{2}$ rounds of Rijndael.

If the set P_5 (or Q_5) is fixed, the bytes of P_3 at position, say, $(0,1)$ only depend on $L^5_{0,1}$, $L^5_{1,1}$, $L^5_{2,1}$, $L^5_{3,1}$, and on $L^4_{0,1}$, see Figure 4.

Similar to the four-round attack, we may guess such a five-tuple of key bytes and compute the corresponding bytes of P_3. If these aren't balanced, we reject the corresponding key bytes. We expect one out of 2^8 incorrect five-tuples to be *not* rejected. With five Λ-sets of plaintexts, i.e., $5 * 2^8$ chosen plaintexts, the the cipher key can easily be found via exhaustive search. (A more diligent treatment would allow us to reduce the number of chosen plaintexts for this attack, but without much effect on the required number of chosen plaintexts for the six-round attack below.)

To measure the running time of our attacks, we use the notion of a "basic operation". Given a column Y_j of bytes of a state Y in Q_r, the key column L^r_j and another key byte $L^{r-1}_{k,j}$ with the row index k as the input, we compute the byte $X_{k,j}$ of a state X in P_{r-2}, using $V = (V_0, V_1, V_2, V_3)$ and $W = (W_0, W_1, W_2, W_3)$ as intermediate values and define the basic operation $X_{k,j} = \mathrm{BO}(Y_j, k, L^r_j, L^{r-1}_{k,j})$ as follows:

1. For $i := 0$ to 3: $V_i := S^{-1}(Y_{i,j} \oplus L^r_{i,j})$.
2. $W := \mu^{-1}(V)$.
3. $X_{k,j} := S^{-1}(W_{k,j} \oplus L^{r-1}_{k,j})$.

In short: one basic operation requires 5 byte-wise XORs, 5 evaluations of S^{-1}, and one evaluation of μ^{-1}.

To check the correctness of a quintuple of bytes, we have to do 2^8 basic operations and to XOR the results for a balance-check by verifying Equation (3). We do this for every quintuple of bytes. Thus, the five-round attack takes the time of about 2^{48} basic operations.

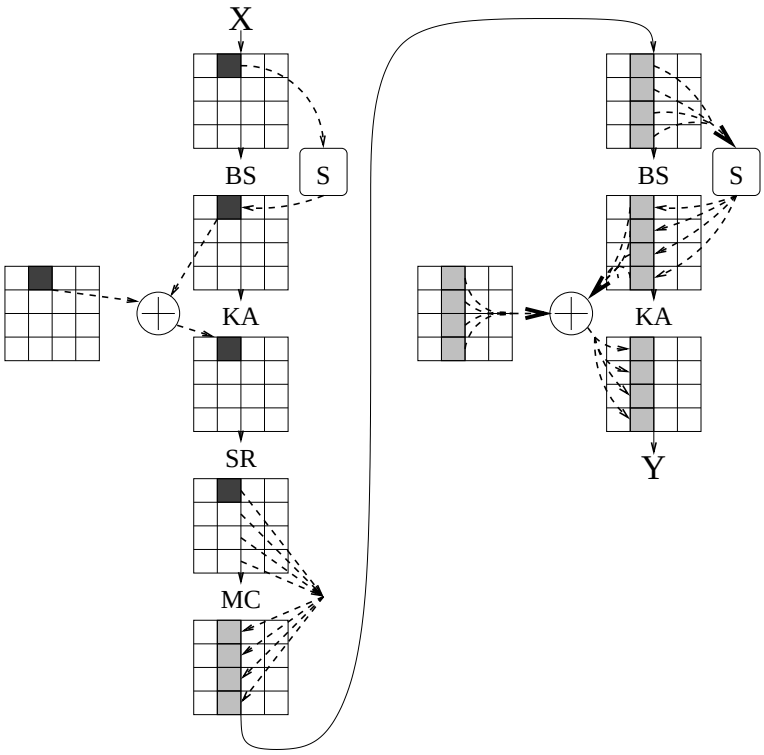

Fig. 4. $1\frac{1}{2}$ rounds of Rijndael: Given one column Y_j of the output state Y and five corresponding key bytes, one can find one byte $X_{k,j}$ of the input X by inverting these $1\frac{1}{2}$ rounds of Rijndael; we write $X_{k,j} = \mathrm{BO}(Y_j, k, \textit{key bytes})$ and consider this a "basic operation" for our attacks.

3.3 Attacking Six Rounds – the Square-6 attack

Now we extend the above attack by an additional "round 0". We denote this attack the "Square-6 attack".

Let P_0 be a Λ^1-set, as before. By doing one additional round of *de*cryption, we get a Λ^4-set P_{-1}. The active bytes of P_{-1} are at positions determined by the positions of the active P_0-byte. E.g., if the P_0-byte at position $(0,0)$ is the active one, the P_{-1}-bytes at positions $(0,0)$, $(1,3)$, $(2,2)$, and $(3,1)$ are active.

The idea is to arbitrarily fix the plaintext bytes at the passive positions and to choose 2^{32} plaintexts varying at the active positions. If the four corresponding bytes of the round key K^{-1} for round 0 are known (e.g. for the active P_0-byte at position $(0,0)$: if $K^{-1}_{0,0}$, $K^{-1}_{1,3}$, $K^{-1}_{2,2}$ and

$K_{3,1}^{-1}$ are known), the adversary can easily determine many 2^8-sets P_{-1} of plaintexts, such that the sets P_0 are Λ^1-sets.

The adversary accordingly chooses 2^{32} plaintexts and, for all 2^{32} relevant key bytes, runs the five-round attack described above. This is 2^{32} times slower than the five-round attack itself, i.e. takes about 2^{80} basic operations. The memory requirement for this attack is dominated by the need to store 2^{32} ciphertexts.

3.4 Considering RD-192 and RD-256

Note that the six-round Square-6 attack and the five-round attack allow us to finds two round keys K^4 and K^5 at the same time. (The attacker chooses a five-tuple of key bytes, one byte from K^4 and four from K^5, and probabilistically verify if that choice is correct.) Once the attacker knows two consecutive round keys, i.e. eight consecutive words from the expanded key, the attacker can easily run the key schedule backwards to find the cipher key. In other words, the performance of the Square-6 attack does not depend on which flavor of Rijndael we attack, RD-128, RD-192, or RD-256. We call such an attack a "generic" attack.

The simple four-round attack only provides the attacker with the round key K^4. But finding the round key K^3 is easy, since the set P_2 of states is a Λ^{16}-set.

4 A Generic Attack for Seven Rounds of Rijndael

It is easy to extend the Square-6 attack to seven rounds of Rijndael:

1. Choose 2^{32} input plaintexts for the Square-6 attack and ask for the corresponding ciphertexts.
2. For all $K^7 \in \{0,1\}^{128}$:
   3. Last-round-decrypt the 2^{32} ciphertexts under K^7.
   4. Run the Square-6 attack for the results, to get the round keys K^6 and K^5.
   5. Given the round keys K^5, K^6 and K^7, we have more than sufficient key material to recover the complete extended key and to check it for correctness.

The seven-round attack requires the same amount of chosen plaintexts and memory as the Square-6 attack. The running time increases by a factor of 2^{128}, i.e. to the equivalent of

$$2^{80} * 2^{128} = 2^{208} \text{ basic operations.}$$

Even though the attack is generic, it is pointless for attacking either RD-128 or RD-192 – exhaustive key search is much faster for these variants of Rijndael.

5 Attacking Seven Rounds of RD-256

For RD-256, the above generic seven-round attack improves on exhaustive search. But, as shown here, the RD-256 key schedule allows us to accelerate the attack by a factor of 2^8.

Note that if we know (or have chosen) K^7, we know the expanded key words $W[28]$, $W[29]$, $W[30]$, and $W[31]$. By Formula (2), we get

$$\begin{aligned} W[21] &= W[28] \oplus W[29], \\ W[22] &= W[29] \oplus W[30], \text{ and} \\ W[23] &= W[30] \oplus W[31]. \end{aligned}$$

Hence, we know know three columns of K^5, including e.g. K^5_1. As explained in Section 2.3, this implies knowing 12 bytes of L^5, including e.g. $L^5_{0,1}$. To test the bytes of 256-set P_4 at position $(0,1)$, we need the bytes in column 1 from Q_6, the corresponding key column L^6_1 from L^6 and the key byte $L^5_{0,1}$ from L^5 (cf. Figure 4 at page 9). We attack seven rounds of RD-256 by the following algorithm:

1. Choose 2^{32} distinct input plaintexts, varying at the byte positions $(0,0)$, $(1,2)$, $(2,2)$, and $(3,1)$ and constant at the other byte positions. Ask for the corresponding ciphertexts.
2. For all 2^{32} combinations of $K^0_{0,0}$, $K^0_{1,3}$, $K^0_{2,2}$, $K^0_{3,1}$:
  3. Fix 32 distinct sets $P_0[i]$ of plaintexts ($i \in \{0,\ldots,31\}$) with $|P_0[i]| = 2^8$, such that the corresponding $P_1[i]$ are Λ^1-sets.
  4. For all 2^{128} round keys K^7:
    5. Decipher the 32 sets of ciphertexts $P_7[i]$ to get $P_6[i]$ and $Q_6[i]$.
    6. Compute $L^5_{0,1}$.
    7. For all 2^{32} combinations of $L^6_1 = (L^6_{0,1}, L^6_{1,1}, L^6_{2,1}, L^6_{3,1})$:
      8. $i := 0$; reject := false;
      9. while $i \leq 31$ and reject=false:
        begin
          10. Compute

$$b[i] := \bigoplus_{A \in Q_6[i]} \mathrm{BO}(A_1, 1, L^6_1, L^5_{0,1}).$$

11. If $b[i] = 0$ then i := i+1
else reject := true.
end (* while *).
12. If reject=false then stop (* and accept key bytes *).

The above algorithm exhaustively searches a subspace of size 2^{192} of the full key space. When all 24 key bytes are correct, step 11 always executes then-clause and increments the counter i. After 32 such iterations, the algorithm stops in step 12.

If any of the 24 byte key bytes is wrong, we execute the then-clause only with a probability of 2^{-8}. Since the counter i runs from 0 to 31, the probability for a wrong 24-tuple of key bytes to be accepted is below $2^{-8*32} = 2^{-256}$. There are only 2^{192} such tuples of key bytes, thus the probability to accept any wrong 24-tuple is less than $2^{32}*2^{128}*2^{32}*2^{-256} \leq 2^{-64}$, i.e. negligible.

When stopping, the algorithm accepts K^7 and four bytes of K^6 (or L^6). By exhaustive search, it is easy to find the other 12 bytes of K^6. Having done that, the key schedule allows to find the full expanded key.

For the attack, 2^{32} chosen plaintexts suffice, and the required storage space is dominated by the need to store the corresponding 2^{32} ciphertexts.

What about the running time? The loop in step 2 is iterated 2^{32} times, step 4 takes 2^{128} iterations, and the loop in step 7 is iterated 2^{32} times. On the average, the while-loop is iterated $1 + 2^{-8} + 2^{-16} + \ldots$ times, i.e., about once. Step 12 needs 2^8 basic operations. This makes about

$$2^{32} * 2^{128} * 2^{32} * 1 * 2^8 = 2^{200} \text{ basic operations.}$$

6 Attacking Seven Rounds of RD-192

In the case of RD-192, accelerating the generic attack by a factor of 2^8, as in the case of RD-256, would still not suffice to outperform exhaustive search. Fortunately (for the cryptanalyst), the RD-192 key schedule allows an acceleration by a factor of 2^{24}, compared to the generic attack,

The columns of K^7 are the words $W[28]$, $W[29]$, $W[30]$, and $W[31]$ of the expanded key. These four words allow us to compute three more words – in the case of RD-256, these are $W[23]$, $W[24]$, and $W[25]$, cf. Section 2.3. Two of these words are columns of the round key K^6, while the third word is a column of K^5: $W[24] = K^6_0$, $W[25] = K^6_1$, and $W[23] = K^5_3$.

From $W[23]$, $W[24]$, and $W[25]$, we can compute three useful key bytes for the attack, for example $L^5_{0,3}$, $L^6_{1,3}$, and $L^6_{2,3}$, cf. Table 1 on page 4. The two remaining key bytes (in our example $L^6_{0,3}$ and $L^6_{3,3}$) still have to be found:

1. Choose 2^{32} distinct input plaintexts, varying at the byte positions $(0,0)$, $(1,2)$, $(2,2)$, and $(3,1)$ and constant at the other byte positions. Ask for the corresponding ciphertexts.
2. For all 2^{32} combinations of $K^0_{0,0}$, $K^0_{1,3}$, $K^0_{2,2}$, $K^0_{3,1}$:
  3. Fix 32 distinct sets $P_0[i]$ of plaintexts ($i \in \{0,\ldots,31\}$) with $|P_0[i]| = 2^8$, such that the corresponding $P_1[i]$ are Λ^1-sets.
  4. For all 2^{128} round keys K^7:
    5. Decipher the 32 sets of ciphertexts $P_7[i]$ to get $P_6[i]$ and $Q_6[i]$.
    6. Compute $L^5_{0,3}$, $L^6_{1,3}$, and $L^6_{2,3}$.
    7. For all 2^{16} combinatios of $L^6_{0,3}$ and $L^6_{3,3}$:
      8. $i := 0$; reject := false;
      9. while $i \leq 31$ and reject=false:
      begin
        10. Compute

$$b[i] := \bigoplus_{A \in Q_5[i]} \mathrm{BO}(A_3, 3, L^6_1, L^5_{0,1}).$$

        11. If $b[i] = 0$ then i := i+1
                else reject := true.
      end ($*$ while $*$).
      12. If reject=false then stop ($*$ and accept key bytes $*$).

The analysis of the attack is essentially the same as its counterpart for RD-256. The only difference is that the loop in step 7 is iterated 2^{16} times instead of 2^{32}. So the attack needs the time of about

$$2^{32} * 2^{128} * 2^{16} * 1 * 2^8 = 2^{184} \text{ basic operations.}$$

7 Final Comments, Summary, and Conclusion

In [3], the authors of Rijndael described the Square-6 attack for RD-128. Extensions of this attack for RD-192 and RD-256 where missing, though. The target of the current paper is to close this gap.

The attacks described in this paper are highly impractical. Considering even such certificational attacks as ours is good scientific practice. And the design of Rijndael was determined "by looking at the maximum number of rounds for which shortcut attacks have been found" [3, Chapter 7.6], allowing an additional margin of security. Any attack which is faster than exhaustive search counts as "shortcut attack".

Attack	target	# Rounds	# Chosen Plaintexts	Time [# basic operations]	Memory [# Ciphertexts]
simple Square	generic	4	2^8	small	small
ext. at the end	generic	5	$5 * 2^8$	2^{48}	small
Square-6	generic	6	2^{32}	2^{80}	2^{32}
7-round	generic	7	2^{32}	2^{208}	2^{32}
	RD-192	7	2^{32}	2^{184}	2^{32}
	RD-256	7	2^{32}	2^{200}	2^{32}

Table 2. Summary of Results.

Table 2 summarises how the different attacks perform. The results for 4–6 rounds of Rijndael originate from [3]. Note that [3] counted the number of "cipher executions" to measure the running time.

Our results exhibit a weakness in the Rijndael key schedule. If, e.g., the words $W[\cdot]$ of the expanded key where generated pseudorandomly using a cryptographically secure pseudorandom bit generator, dedicated attacks could not be more efficient than their generic counterparts.

This does not indicate the necessity to modify the Rijndael key schedule, though. The improvements on the generic case are quite small. If we concentrate on counting the number of rounds for which shortcut attacks exist, the cryptanalytic gain of this paper is one round for RD-192, not more. The authors of Rijndael seem to have anticipated such cryptanalytic results by specifying a high security margin for the number of rounds (two additional rounds for RD-192, compared to RD-128 with its ten rounds).

References

1. C. D'Halluin, G. Bijnens, V. Rijmen, B., Preneel: "Attack on six round of Crypton", Fast Software Encryption 1999, Springer LNCS 1636, pp. 46–59.
2. J. Daemen, L. Knudsen, V. Rijmen: "The block cipher Square", Fast Software Encryption 1997, Springer LNCS 1267, pp. 149–165.
3. J. Daemen, V. Rijmen: "AES proposal: Rijndael" (2nd version), AES submission.
4. J. Daemen, V. Rijmen: "The block cipher BKSQ", Cardis 1998, Springer LNCS, to appear.
5. C. H. Lim: "Crypton: a new 128-bit block cipher", AES submission.
6. C. H. Lim: "A revised version of Crypton – Crypton V 1.0 – ", Fast Software Encryption 1999, Springer LNCS 1636, pp. 31–45.