Improved Cryptanalysis of Rijndael

Niels Ferguson[1], John Kelsey[1], Stefan Lucks[⋆2], Bruce Schneier[1], Mike Stay[3], David Wagner[4], and Doug Whiting[5]

[1] Counterpane Internet Security, Inc., 3031 Tisch Way Suite 100PE, San Jose, CA 95128
[2] University of Mannheim, 68131 Mannheim, Germany
[3] AccessData Corp. 2500 N. University Ave. Ste. 200, Provo, UT 84606
[4] University of California Berkeley, Soda Hall, Berkeley, CA 94720
[5] Hi/fn, Inc., 5973 Avenida Encinas Suite 110, Carlsbad, CA 92008

Abstract. We improve the best attack on Rijndael reduced to 6 rounds from complexity 2^{72} to 2^{44}. We also present the first known attacks on 7- and 8-round Rijndael. The attacks on 8-round Rijndael work for 192-bit and 256-bit keys. Finally, we discuss the key schedule of Rijndael and describe a related-key attack that can break 9-round Rijndael with 256-bit keys.

1 Introduction

Rijndael is one of the five AES candidate ciphers that made it to the second round [DR98]. Rijndael has 10, 12, or 14 rounds, depending on the key size. Previously it was known how to break up to 6 rounds of Rijndael [DR98]. Independently from our work, Gilbert and Minier [GM00] presented an attack on 7 rounds of Rijndael.

In section 2, we describe a new *partial sum* technique that can dramatically reduce the complexity of the 6-round attacks. We also show how to use these ideas to attack 7 and 8 rounds of Rijndael, in some cases using additional known texts (where available) to reduce the workfactor. The attacks against 7-round Rijndael with 128-bit keys and 8-round Rijndael with 192-bit and 256-bit keys require nearly the entire Rijndael codebook ($2^{128} - 2^{119}$ chosen plaintexts); they are therefore not very practical even for an adversary with sufficient computing power. All of these attacks use extensions of the dedicated Square attack, as described in [DKR97,DR98,DBRP99].

In section 3, we turn our attention to the key schedule. We show several unexpected properties of the key schedule that seem to violate the published design criteria. Although we do not know of any attacks that critically depend on these properties, we consider them unsettling. Finally, in section 4, we exploit the slow diffusion of the Rijndael key schedule to develop a related-key attack that can be mounted on 9 rounds of Rijndael with a 256-bit key.

A summary of these attacks, including time and data complexities, is described in table 1. We also refer the reader to appendix A for a detailed listing of notation used to refer to intermediate values in the cipher.

⋆ Supported by DFG grant KR 1521/3-2.

Cipher	Key size	Complexity [Data]	[Time]	Comments
Rijndael-6	(all)	2^{32} CP	2^{72}	[DR98] (previously known)
Rijndael-6	(all)	$6 \cdot 2^{32}$ CP	2^{44}	partial sums (new)
Rijndael-7	(192)	$19 \cdot 2^{32}$ CP	2^{155}	partial sums (new)
Rijndael-7	(256)	$21 \cdot 2^{32}$ CP	2^{172}	partial sums (new)
Rijndael-7	(all)	$2^{128} - 2^{119}$ CP	2^{120}	partial sums (new)
Rijndael-8	(192)	$2^{128} - 2^{119}$ CP	2^{188}	partial sums (new)
Rijndael-8	(256)	$2^{128} - 2^{119}$ CP	2^{204}	partial sums (new)
Rijndael-9	(256)	2^{85} RK-CP	2^{224}	related-key attack (new)

CP – chosen plaintext, RK-CP – related-key chosen plaintext.

Table 1. Summary of Attacks on Rijndael.

2 The Square Attack

2.1 The Original 6-round Attack

We start by describing the 6-round attack in the original proposal [DR98], which uses a technique first introduced to attack the block cipher Square [DKR97]. This is an attack that works against all block sizes and key sizes.

We use $m^{(r)}$, $b^{(r)}$, and $t^{(r)}$ to refer to intermediate text values used in round r after the MixColumn, key addition, and ShiftRow operations, respectively. We write $k^{(r)}$ for the subkey in round r, and $k^{(r)'}$ for an equivalent subkey value that may be XORed into the state before instead of after the MixColumn operation in round r. Please refer to appendix A for a more detailed explanation of our notation.

The attack starts by obtaining 256 encryptions that only differ in a single byte of $m^{(1)}$, and that take on all values for that particular byte. One byte of $m^{(1)}$ depends on four bytes of the plaintext and four bytes of $k^{(0)}$. We first choose 2^{32} plaintexts by taking a fixed starting point and varying those four bytes over all 2^{32} possible values. We then guess the four key bytes that are involved. For each possible value of the key bytes, we can find 2^{24} groups of 256 plaintexts such that within each group the encryptions differ in a specific byte of $m^{(1)}$; as the plaintexts are different, this one byte of $m^{(1)}$ must take on all 256 possible values.

Tracking these changes through the cipher, we find that each of the bytes of $t^{(4)}$ takes on all possible values. For each of these bytes, if we sum the values it takes on in the 256 encryptions, we get zero. This property is preserved by a linear function, so each of the bytes of $m^{(4)}$, and of $b^{(4)}$, also sums to zero over our 256 encryptions.

We now look at a particular byte of $b^{(4)}$, and how that relates to the ciphertext. For our analysis we rewrite the cipher slightly, and put the AddRoundKey before the MixColumn in round 5. Instead of applying MixColumn and then adding in $k^{(5)}$, we first add in $k^{(5)'}$ and then apply MixColumn. In this configuration it is easy to see that any byte of $b^{(4)}$ depends on the ciphertext, four bytes from $k^{(6)}$, and one byte from $k^{(5)'}$. We guess these five key bytes, compute the value of our $b^{(4)}$ byte for our 256 encryptions and check whether the sum is zero.

For each group of 256 plaintexts, this filter rejects 255/256 of all wrong key guesses. As we guess a total of nine key bytes, we will need 10 or so groups of 256 encryptions to find the key. (Note that these groups depend on the first four key bytes that we guessed, but not on the last five.)

Overall, this attack requires 2^{32} chosen plaintexts, 2^{32} memory to store those plaintext/ciphertext pairs, and 2^{72} steps in guessing the nine key bytes. Each step involves a partial decryption of 256 ciphertexts, but a proper ordering of these computations can make that very fast. This seems to be comparable to doing a single encryption, and agrees with the complexity estimate given in [DR98]. The overall complexity is thus comparable to 2^{72} encryptions.

2.2 A 7-round Extension

This attack can be extended to 7 rounds for 192- and 256-bit keys. One simply guesses the 16 bytes of the last round key. When used naively, this adds 128 bits to the key guessing, for a total workload of 2^{200}; the plaintext and memory requirements are not changed, although we do need to use more groups to verify the potential keys.[1]

This can be further improved. The key schedule ensures that there are dependencies between the expanded key bytes, and we can exploit them in this attack. For a 192-bit key, guessing the last round key $k^{(7)}$ gives us two of the four bytes from $k^{(6)'}$ that we would otherwise have to guess plus the byte from $k^{(5)'}$ that we would guess. This saves us 24 bits of key guessing, and results in an overall complexity of 2^{176}. For 256-bit keys, the bytes in the key schedule are aligned differently. Guessing all of $k^{(7)}$ provides no information about $k^{(6)'}$ but does give us the one byte from $k^{(5)'}$ that we need. For this key length, the complexity of the attack thus becomes 2^{192}. All the details can be found in [Luc00].[2]

2.3 An Improvement

The attack of section 2.1 on 6 rounds of Rijndael can be improved. Instead of guessing four bytes of $k^{(0)}$ we simply use all 2^{32} plaintexts. For any value of the first round key, these encryptions consist of 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(1)}$. All we have to do is to guess the five key bytes

[1] For this attack we use the alternate round representation for both rounds 5 and 6, and thus add $k^{(6)'}$ before the MixColumn in round 6.

[2] Note that the attack complexities in [Luc00] are given in S-box lookups, whereas we roughly approximate the complexity of a single encryption by 2^8 S-box lookups and use encryptions as our unit. The result is that all our complexity numbers are a factor of 2^8 lower.

at the end of the cipher, do a partial decrypt to a single byte of $b^{(4)}$, sum this value over all the 2^{32} encryptions, and check for a zero result. Compared to the original version, we guess only 40 bits of key instead of 72. On the other hand, we have to do 2^{24} times as much work for each guess. All in all, this improvement reduces the workload by a factor of 2^8, although it needs about $6 \cdot 2^{32}$ plaintexts to provide enough sets of 2^{32} plaintexts to uniquely identify the proper value for the five key bytes.

We will now look at this attack in more detail. We have 2^{32} ciphertexts. We guess five key bytes, do a partial decryption from each of the ciphertexts to a single byte in $b^{(4)}$, and sum this byte over all ciphertexts. Consider this partial decryption. From any ciphertext, we use four ciphertext bytes. Each of these is XORed with a key byte. We then apply the inverse S-box to each byte, and multiply each with an appropriate factor from the inverse MDS matrix. The four bytes are then XORed together, a fifth key byte is XORed into the result, the inverse S-box is applied, and the resulting value is summed over all ciphertexts.

Let $c_{i,j}$ be the jth byte of the ith ciphertext. (We leave out the i subscript if we are not talking about any particular ciphertext.) For simplicity we will number the four bytes of each ciphertext that we use from 0 to 3. Let $k_0, \ldots, k_4$ denote the five key bytes that we are guessing. We want to compute

$$\sum_i S^{-1}[S_0[c_{i,0} \oplus k_0] \oplus S_1[c_{i,1} \oplus k_1] \oplus S_2[c_{i,2} \oplus k_2] \oplus S_3[c_{i,3} \oplus k_3] \oplus k_4] \qquad (1)$$

where $S_0, \ldots, S_3$ are bijective S-boxes, each of which consists of an inverse Rijndael S-box followed by a multiplication by a field element from the inverse MDS matrix. Given 2^{32} ciphertexts and 2^{40} possible key guesses, we have to sum 2^{72} different values, which corresponds roughly in amount of work to doing about 2^{64} trial encryptions.

We can organize this more efficiently in the following manner. For each k, we associate a "partial sum" x_k to each ciphertext c, defined as follows:

$$x_k := \sum_{j=0}^{k} S_j[c_j \oplus k_j]$$

This gives us a map $(c_0, c_1, c_2, c_3) \mapsto (x_k, c_{k+1}, \ldots, c_3)$ that we can apply to each ciphertext if we know $k_0, \ldots, k_k$.

We start out with a list of 2^{32} ciphertexts. We guess k_0 and k_1 and compute how often each triple (x_1, c_2, c_3) occurs in the list. That is, for each i, we compute the three-byte value $(S_0[c_{i,0} \oplus k_0] \oplus S_1[c_{i,1} \oplus k_1], c_{i,2}, c_{i,3})$ as a function of the ith ciphertext and the guessed key material, and we count how many times each three-byte value appears during this computation. As there are only 2^{24} possible values for three bytes, we do not have to list all (x_1, c_2, c_3) values; rather, we count how often each triple occurs. We then guess k_2 and compute how often each tuple (x_2, c_3) occurs; and guess k_3 and compute how often each value of x_3 occurs. Finally, we guess k_4 and compute the desired sum.

Because all sums are taken using the XOR operation, and because $z \oplus z = 0$ for all z, it suffices to only count modulo two. Thus, a single bit suffices for each count, and so the space requirement for the 2^{24} counters is just 2^{24} bits.

How much work has this been? In the first phase we guessed 16 bits and processed 2^{32} ciphertexts, so this phase costs 2^{48} overall. In the next phase, we guessed a total of 24 bits but we only had to process 2^{24} triples, so this costs 2^{48} as well. This holds similarly for each of the phases. In total, the entire computation requires the equivalent of about 2^{48} evaluations of equation 1, or about 2^{50} S-box applications.

This is the amount of work required for a single structure of 2^{32} ciphertexts. The first structure already weeds out the overwhelming majority of the wrong key guesses, but we still have to do the first steps of our partial sum computation for each of the six structures that we use. The total number of S-box lookups is thus about 2^{52}.

Using our earlier rough equivalence of 2^8 S-box applications to a trial encryption with a new key, the 2^{52} S-box applications are comparable to 2^{44} trial encryptions. This is a significant improvement over the earlier 2^{72} workfactor.

2.4 Extension to 7 Rounds

We can apply this our improvement to the 7-round attack of section 2.2. To express a single byte of $b^{(4)}$ in the key and the ciphertext, we get a formula similar to equation 1 but with three levels, 16 ciphertext bytes, and 21 key bytes. The partial sum technique is only helpful during the last part of the computation as it only saves work if there are more ciphertexts than possible values for the intermediate result. With 2^{32} plaintext/ciphertext pairs in a structure, these techniques will not help until the very last part of the computation.

For 192-bit keys we first guess the 128 bits of the last round key. These guesses also define two of the four key bytes in round 6 that we are interested in, and the one key byte in round 5 that we need. Thus, after guessing the last round key we can reduce each structure to 2^{24} counters with our partial sum technique. Using some precomputed tables, we can do this for each of the 2^{128} key guesses in about 2^{32} memory lookups. The next phase guesses one byte more and requires 2^{24} steps to reduce the partial sum to 2^{16} counters, and the last phase guesses the last remaining byte and produces the final result. Each of these phases has a cost of 2^{160} lookups. We have three phases, each of which costs 2^{160}, and we need to process three structures before we start eliminating guesses for the last round key, so the overall cost of this attack is on the order of 2^{163} S-box lookups or about 2^{155} trial encryptions.

For 256-bit keys the alignment in the key schedule is different. Guessing the last round key does not give us any information about the round key of round 6, but it provides most of the round key for round 5. Working in a similar fashion as before, we guess 128 bits of the last round key and compute the four bytes we are interested in after round 6 for each of the 2^{32} texts for a total cost of 2^{160} lookups. The next phase guesses 16 more key bits and results in 2^{24} one-bit counters for a total cost of 2^{176} lookups. The remaining phases have a similar cost. The cost per structure is thus about 2^{178} lookups or about 2^{170} trial encryptions. We need five structures before we start cutting into the guesses of the last round key, so the overall complexity of this attack is about 2^{172}.

2.5 A Second Improvement

It is possible to push these attacks even further, if we are willing to trade texts for time and increase the data complexity to save on the workfactor.

We first show that 7 rounds of Rijndael may be broken with 2^{128} known texts (the entire codebook!) and workfactor equivalent to approximately 2^{120} trial encryptions. These encryptions consist of 2^{96} packs of 2^{32} encryptions that vary only in four bytes of $m^{(1)}$. Those four bytes are in a proper position to apply the attack of section 2.3: specifically, each pack of 2^{32} encryptions consists of 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(2)}$. Equivalently, we may view the entire set of 2^{128} encryptions as consisting of 2^{120} groups of 2^8 encryptions that vary only in one byte of $m^{(2)}$. This ensures that summing a single byte in $b^{(5)}$ over the 2^8 encryptions in a group yields zero, and thus summing over all 2^{128} encryptions also yields zero in this byte. This simple property is the basis for several attacks, as described below.

A naive way that one might try to exploit this property is to guess five key bytes at the end of the cipher, partially decrypt each ciphertext to a single byte of $b^{(5)}$, sum over all 2^{128} ciphertexts, and check for zero. However, the naive approach does not actually work. Even the wrong keys will yield zero when summing the byte in $b^{(5)}$ over all 2^{128} encryptions, because for any bijective 128-bit block cipher, $b^{(5)}$ (or any other intermediate value) will take on all possible 128-bit values as you cycle through all 2^{128} encryptions. Consequently, we will need to modify the attack slightly.

Instead, we use the following technique. Focus our attention on a fifth byte in $m^{(1)}$ (different from the four bytes selected earlier), say, $m^{(1)}_{a,b}$. Fixing a value x for this byte gives us a set of 2^{120} encryptions where $m^{(1)}_{a,b} = x$; this gives us a list of 2^{88} packs, where each pack contains 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(2)}$. We call this structure of 2^{120} encryptions a *herd*. Now we obtain 2^{128} known texts (2^8 herds), guess four key bytes at the beginning of the cipher, calculate $m^{(1)}_{a,b}$ for each encryption using our guessed key material, and separate the texts into herds. Examining a single such herd, we find that summing a byte in $b^{(5)}$ over all the encryptions in the herd yields zero, and moreover this property is unlikely to hold if our guesses at the key were incorrect. This yields a working attack against 7 rounds of Rijndael, but the complexity is very high ($2^{128} \times 2^{72}$ steps of computation or so).

One can do much better. Note that the byte in $b^{(5)}$ depends only on four bytes of the ciphertext (for simplicity, call them $c_0, \dots, c_3$) and the byte $m^{(1)}_{a,b}$ depends on only four bytes of the plaintext ($p_4, \dots, p_7$, say). We use a three-phase attack; the first phase uses 2^{64} counters (the m_y's), the second phase uses 2^{32} counters (the n_z's), and the third phase provides the filtering information for key guesses. As usual, all counters may be taken modulo 2, so we need just one bit for each counter.

The attack goes as follows. In the first phase, we increment the counter m_y corresponding to the 64-bit quantity $y = (c_0, \ldots, c_3, p_4, \ldots, p_7)$ as we see each known text (p, c). The second phase guesses four key bytes from the first round, separates the counters into herds (by computing $m^{(1)}_{a,b}$ for each counter position using $(p_4, \ldots, p_7)$ and the guessed key material), selects a single herd, and updates the counter n_z by adding m_y to it for each y that is in the correct herd and that agrees with $z = (c_0, \ldots, c_3)$. Afterwards, in the third phase, we guess five key bytes at the end of the cipher, partially decrypt each z to a single byte in $b^{(5)}$, sum this byte over all 2^{32} values of z (with multiplicities as given by the n_z), and check for zero. The third phase must be repeated for each guess of the four key bytes in the first round.

What is the complexity of this attack? The first phase requires us to update a counter for each ciphertext, so using our rough equivalence of 2^8 memory lookups to a trial encryption, the counting should take time comparable to 2^{120} trial encryptions. Compared to the first phase, the rest of the attack has negligible workfactor (equivalent to 2^{96} encryptions); there is no need to compute partial sums, an exhaustive key search will suffice.

This shows that one may break 7 rounds of Rijndael using 2^{128} known texts, 2^{120} work, and 2^{64} bits of memory. This 7-round attack trades texts for time: it uses a huge number of known texts, but it has better workfactor and overall complexity than the 7-round attack of section 2.3; and moreover, it applies to all key sizes (including the 128-bit keys).

There are a few more small improvements. We used a single byte of $m^{(1)}$ to define our herds, but the four plaintext bytes that we use in our attack and the four key bytes of the first round key that we guess define four bytes of $m^{(1)}$. We can create more (but smaller) herds by fixing three bytes of $m^{(1)}$ for each herd. This gives us 2^{24} herds of 2^{104} texts each.[3] We can even choose which of the four bytes will take on every value, and thus create 2^{26} herds of 2^{104} texts each, in which case each text is used in four different herds. Furthermore, we do not need all the plaintext/ciphertext pairs. If the four plaintext bytes take on $2^{32} - 2^{23}$ of the 2^{32} possible values (and for each of these values the other 12 bytes take on all possible values), then about half of our herds will have missing plaintext/ciphertext pairs while the other half are complete and undamaged. We can use the undamaged herds in our attack. This reduces the plaintext requirements to $2^{128} - 2^{119}$ texts. These changes do not change the complexity of the attack, but give us a slight reduction in the text requirements.

2.6 Extension to 8 Rounds

We can further extend the idea to break 8 rounds of Rijndael, though apparently not for 128-bit keys. As before, we obtain $2^{128} - 2^{119}$ texts (about 2^{23} undamaged

[3] Note that we cannot use all four bytes, as at least one of the four bytes has to vary within the pack.

herds), focus attention on a single herd, and use the fact that a single byte in $b^{(5)}$ will yield zero when summed over all 2^{104} encryptions in the herd. However, the byte in $b^{(5)}$ now depends on the entire ciphertext and on 21 subkey bytes at the end of the cipher, so now we must apply the partial sum techniques of section 2.4. Guessing the four key bytes of the first round first to define our herds and computing the partial sums x_k one at a time allows one to calculate the desired sum with 2^{104} bits of storage and work equivalent to about 2^{202} trial encryptions. We need to do this for about four herds before we start to cut our search tree. (We will need about 26 herds in total to get a unique solution, but the workload is dominated by the first four herds.) The overall attack complexity comes out at 2^{204} encryptions, and thus faster than exhaustive key search for 256-bit keys.

As this attack needs the equivalent of more than 2^{192} encryptions, it seems to be useless for 192-bit keys. But the 192-bit key schedule allows a 2^{16}-fold speed-up for the 8-round attack, which thus requires the equivalent of about $2^{204-16} = 2^{188}$ encryptions. We stress that the time complexity of 2^{188} encryptions only holds for 192-bit keys, not for 256-bit keys.

Each byte of $b^{(5)}$ depends on 21 subkey bytes, namely: all the 16 bytes from $k^{(8)}$, 4 bytes from $k^{(7)'}$ and one byte from $k^{(6)'}$. Similar to Section 2.2, fixing the last round key $k^{(8)}$ determines two of the four bytes from $k^{(7)'}$ and, depending on which byte of $b^{(5)}$ we target, possibly also the relevant subkey byte from $k^{(6)'}$. More precisely, by choosing three columns (12 bytes) of $k^{(8)}$ one can learn two columns of $k^{(7)'}$, and the fourth column of $k^{(8)}$ also determines one column of $k^{(6)'}$. In each column of $k^{(7)'}$ we find one subkey byte we need for the attack. In other words, fixing $k^{(8)'}$ (or even only three columns of $k^{(8)'}$) gives us two useful key bytes of $k^{(7)'}$. (This holds for the 192-bit key schedule. See [Luc00], where the Rijndael key schedule and this weakness are explained in more detail.)

To describe the attack, we look at the partial sums technique from a slightly different point of view. To attack 8-round Rijndael, we check the sum of the values taken on by one byte from $b^{(5)}$ in the 2^{104} encryptions of a herd. For this, we evaluate equation 1 five times: four times on the "bottom level", and, using these four results, a last time taking these four values instead of the ciphertexts $c_{i,0}, \ldots, c_{i,3}$. Each evaluation of equation 1 starts with counters for $(c_0, c_1, c_2, c_3, \langle\text{other}\rangle)$, where the bytes c_i and the corresponding keys are aligned in the same column. It can be described by the following substeps:

1. Guess two key bytes, w.l.o.g. k_0 and k_1, and compute the counters (mod 2) for $(x_{0,1}, c_2, c_3, \langle\text{other}\rangle)$.
2. Guess one key byte, w.l.o.g. k_2, and count $(x_{0,1,2}, c_3, \langle\text{other}\rangle)$.
3. Guess one key byte (k_3), and count $(x_{0,\ldots,3}, \langle\text{other}\rangle)$.

(Note that we just introduced a slightly different notation for the $x_{\langle\text{some}\rangle}$. The reason will become obvious below.)

To attack 8-round Rijndael with 192-bit keys, we obtain $2^{128} - 2^{119}$ texts, guess four first-round subkey bytes to obtain our herds, concentrate on a single herd, and target a single byte in $b^{(5)}$. We continue with guessing three columns of

k^8 and evaluating equation 1 for each column. This gives us gives us 2^{56} counters (mod 2) for

$$(x_{0,\dots,3}, x_{4,\dots,7}, x_{8,\dots,11}, c_{12}, c_{13}, c_{14}, c_{15}).$$

Note that the values $x_{0,\dots,3}$, $x_{4,\dots,7}$, and $x_{8,\dots,11}$ correspond to the bytes c_0, c_1, and c_2 we use within equation 1 to get the final count for the byte of $b^{(5)}$. Now we evaluate (not guess!)[4] two key bytes of $k^{(7)}$ and execute the first substep of the partial sums technique. Essentially for free, we reduce the number of counters from 2^{56} to 2^{48}, and we get counters (mod 2) for

$$(x_{0,\dots,7}, x_{8,\dots,11}, c_{12}, c_{13}, c_{14}, c_{15}).$$

By guessing the last column of $k^{(8)}$ and get 2^{24} counters for

$$(x_{0,\dots,7}, x_{8,\dots,11}, x_{12,\dots,15}).$$

Guessing another two bytes of $k^{(7)}$ we get 2^8 counters for x_{15}. We can evaluate one byte of $k^{(6)}$, which allows us to check the balancedness of one byte of $b^{(5)}$.

Note that the order in which things are done is crucial for the time complexity. If we first guessed all 16 bytes from $k^{(8)}$ and only then evaluated the two key bytes from $k^{(7)'}$ which we get for free, the speed-up would only be 2^8 compared to the running time for attacking Rijndael with 256-bit keys. In this case, the time complexity would be 2^{196} for 192-bit keys, i.e. *slower* than exhaustive search.

2.7 Summary

The Square attack can be improved so that it requires 2^{44} work to attack 6 rounds of Rijndael. The extension to 7 rounds has complexity 2^{155} for 192-bit keys and complexity 2^{172} for 256-bit keys. There is also an alternative extension to 7 rounds that can break all key sizes with lower overall complexity (2^{120} work) but which requires virtually the entire codebook of texts ($2^{128} - 2^{119}$ texts). Another result of our analysis is that, for the 256-bit and 192-bit key sizes, one may break 8 rounds of Rijndael faster than by exhaustive search, again with $2^{128} - 2^{119}$ texts. The 256-bit key size requires 2^{204} work, the 192-bit key size 2^{188}.

3 The Key Schedule

Compared to the cipher itself, the Rijndael key schedule appears to be more of an ad hoc design. It has a much slower diffusion structure than the cipher, and contains relatively few non-linear elements.

---

[4] This is where the 2^{16}-fold speed-up for 192-bit keys comes from.

3.1 Partial Key Guessing

The Rijndael submission document states that the key schedule was designed with the requirement that "Knowledge of a part of the Cipher Key or Round Key bits shall not allow to calculate many other Round Key bits" [DR98, section 7.5]. The key schedule does not seem to achieve that goal.

Let us look at the case of a 128-bit block size and 256-bit key in more detail. The key schedule consists of 8 cycles, which produces a total of 15 round keys (the last half of the last cycle is never used). The key schedule can be seen as four separate rows that only have a limited interaction. We will concentrate on a particular row; say, number i. We guess the values $K_{i,7}^{(s)}$ for $s = 0, \ldots, 6$. (There is no point in guessing it for $s = 7$, as that byte is not used in an expanded key.) Using the recurrent computation rule of the key schedule, we can now compute $K_{i,6}^{(s)}$ for $s = 1, \ldots, 6$, $K_{i,5}^{(s)}$ for $s = 2, \ldots, 6$, etc. (Please refer to appendix A for definitions of our notation, if it is not clear.) All in all, we learn 28 bytes of the expanded key for the cost of having guessed only seven bytes.

The bytes that we guessed in row i are exactly those bytes that affect row $i - 1 \bmod 4$. Thus, if we now guess the first eight bytes of row $i - 1$, then we can compute the rest of that row for a total of 60 bytes, and if we guess a total of 15 bytes, we learn 88 bytes of the expanded key.

We can extend this with further rows, and get 148 bytes of the expanded key by guessing 23 bytes, and 208 bytes by guessing 31 bytes. There are of course many other ways in which guessing some bytes results in knowledge of many more. On a smaller scale, several of our attacks in section 2 used dependencies between round key bytes to reduce the complexity of the attack.

3.2 Key Splitting

Another interesting property is that the key can be "split" into two halves. The two topmost rows interact with the two bottommost rows through only 14 bytes (in the case of 128-bit block size and 256-bit key). If we guess (or know) those 14 bytes, then the rest of the key has been split into two independent halves, each of which controls half of the expanded key bytes. There are many ways to split the key. By rows is the easiest way, but it is also possible to split it by column (at least for a few cycles).

This immediately suggests some kind of meet-in-the-middle attack to a cryptanalyst. However, as the expanded key bytes of the two halves are mixed very thoroughly in the non-linear cipher, we have not found a way to exploit this property. Note that the DES key schedule allows the key bits to be split into 56 independent parts, but no attack is known that uses this property.

3.3 Summary

The fact that these properties are present in spite of the stated design goal is unsettling. Some of our attacks make use of the relations between expanded key bytes and would have a higher complexity if these relations did not exist. The attack on 8-round Rijndael with 192-bit keys would be slower than exhaustive key search without these relations. Our attack in the next section also makes extensive use of the properties of the key schedule.

4 A 9-Round Related-Key Attack

Related-key attacks were first introduced by Biham in [Bih93] and later extended in [KSW96,KSW97]. We assume that the reader is familiar with the basics of related-key cryptanalysis.

The submission states that "The key schedule of Rijndael, with its high diffusion and non-linearity, makes it very improbable that [related-key attacks] can be successful for Rijndael" [DR98, section 8.7], and also lists resistance to related-key attacks as one of the requirements for the Rijndael key schedule [DR98, section 7.5]. We do not feel that the Rijndael key schedule has a very high level of diffusion. It can take many cycles before a low-weight difference starts to affect a significant number of other bytes. This can best be seen if we run the key schedule backwards; each byte affects two other bytes that are (almost) a full cycle further back.

We show how a related-key attack can be mounted on 9 rounds of Rijndael with a 256-bit key. This is basically a variant of the Square attack; we use 256 related keys that differ in a single byte in the fourth round key. We use plaintext differences to cancel out the earlier round key differences, and get three bytes at the end of round 6 that sum to zero when taken over the 256 encryptions. We guess key bytes of the last three rounds to compute backwards from the ciphertext and detect this property.

4.1 The Key Difference Pattern

Starting with an unknown base key L, we derive a set of 256 related keys $L_0, \ldots, L_{255}$. The difference $L_a \oplus L$ takes on the value a in bytes 21 and 25, and is zero elsewhere. The diffusion in the key schedule is slow enough that we can track all the differences in the round keys. Figure 1 shows the difference pattern. The key schedule for the 9-round cipher needs to generate 10 round keys. With a 128-bit block size and a 256-bit key, this requires five cycles of the key schedule, which are shown in the figure. Each of the cycles provides two round keys.

The dark gray bytes are the bytes of L that we guess. The light gray bytes are bytes that we can deduce from the guesses that we have made using the recurrence relationship between the expanded key bytes. We guess a total of 27 bytes of the key, and this allows us to compute a total of 66 bytes of the expanded key. We will use all of our guesses in the attack, but for the moment we concentrate on tracking the differences through the key schedule.

In the first cycle we have a difference a in $K^{(0)}_{1,5}$ and $K^{(0)}_{1,6}$. In the next cycle we get a difference a in $K^{(1)}_{1,5}$. In the third cycle we have difference a in $K^{(2)}_{1,5}$,

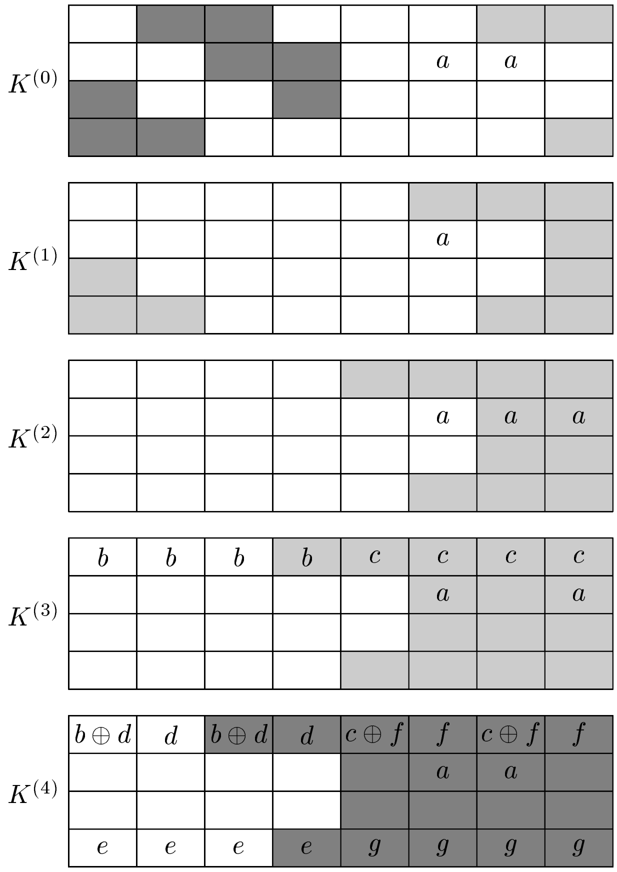

Fig. 1. Difference and guessing pattern in the key of the 9-round attack.

$K_{1,6}^{(2)}$, and $K_{1,7}^{(2)}$. At this point the difference is first confronted with a non-linear S-box. To track the difference, we need to know $K_{1,7}^{(2)}$ of key L; this allows us to compute the output difference b of the S-box given the input difference a. As the shading shows, this key byte can be deduced from the guesses that we have made. In the fourth cycle we get the difference b in $K_{0,i}^{(3)}$ for $i = 0, \ldots, 3$. Again we encounter an S-box, and therefore we need to know $K_{0,3}^{(3)}$ of L. This gives us the output difference c of that S-box given input difference b. We thus get a difference c in $K_{0,i}^{(3)}$ for $i = 4, \ldots, 7$. The differences from the previous cycle also come through as a difference a in $K_{1,5}^{(3)}$ and $K_{1,7}^{(3)}$. We can track the rest of the difference propagation in a similar way as is shown in the figure. All in all, it turns out that we have guessed more than enough bytes of L to be able to track all of the changes. That is, for each value of a, we know the exact value of b, c,

d, e, f, and g. Although we are guessing a very large number of bytes that we will use in our attack, we would only need to guess six bytes in order to track the difference pattern through the key schedule.

4.2 The Encryptions

Having guessed the dark-gray bytes shown in figure 1, we encrypt one plaintext under each key. These plaintexts are chosen such that all encryptions end up in the same state after the first round (i.e., after adding the second round key). We know the differences in the second round key $k^{(1)}$ and the key bytes that we guessed allow us to introduce appropriate differences in the plaintexts to ensure the same state after round 1. We now get a single byte difference introduced at the end of round 3; if we look at all our 256 encryptions, this one byte takes on each value exactly once. This propagates to ensure that each byte of $m^{(5)}$ runs over all possible values when taken over the 256 encryptions.

The next few steps are shown in figure 2. The round keys for round 5 and 6 are on the left, with their differences marked. On the right are some of the state differences. The bytes marked O are bytes that take on every possible value exactly once. Bytes marked X can behave in any manner. Bytes marked σ have the property that if you sum them over all 256 encryptions, the sum is zero. The important item to note is that we have three σ bytes in $b^{(6)}$.

We are going to compute $b^{(6)}_{1,3}$ from the ciphertext, our known key bytes, and some additional guessed key bytes. This is shown in the figure with the gray color. Note that we are using an equivalent representation for round 8, where we have swapped the order of the MixColumn and AddRoundKey, and add $k^{(8)'}$ instead of $k^{(8)}$. We know the ciphertext and the last round key, so we can compute backwards up to the AddRoundKey of round 8. We now guess the four marked bytes in $k^{(8)'}$. (We know several bytes of $k^{(8)}$, but that provides no information about these bytes of $k^{(8)'}$. However, as each column of $k^{(8)'}$ is the result of an inverse MixColumn operation on $k^{(8)}$, we can propagate our knowledge of the differences from $k^{(8)}$ to $k^{(8)'}$.) We can now compute the marked bytes in $t^{(8)}$, $s^{(8)}$, and $t^{(7)}$, and finally we can compute $b^{(6)}_{1,3}$. We check whether this value sums to zero when taken over all 256 encryptions. If this is not the case, we have made a wrong guess somewhere. As before, we can generate enough sets of plaintexts to uniquely identify the correct key guesses that we have made.

All in all, we have guessed 31 bytes of key material, and for each guess we perform an amount of work comparable to a single encryption. This puts the overall complexity of the attack at 2^{248}.

Looking at the plaintext requirements, we do not have to perform 256 encryptions for each of the key byte guesses that we have made. The eight bytes of plaintext that we use to cancel the differences can take on only 2^{64} values, so we can encrypt 2^{64} plaintexts with each of the 256 related keys for a total chosen plaintext requirement of 2^{72}.

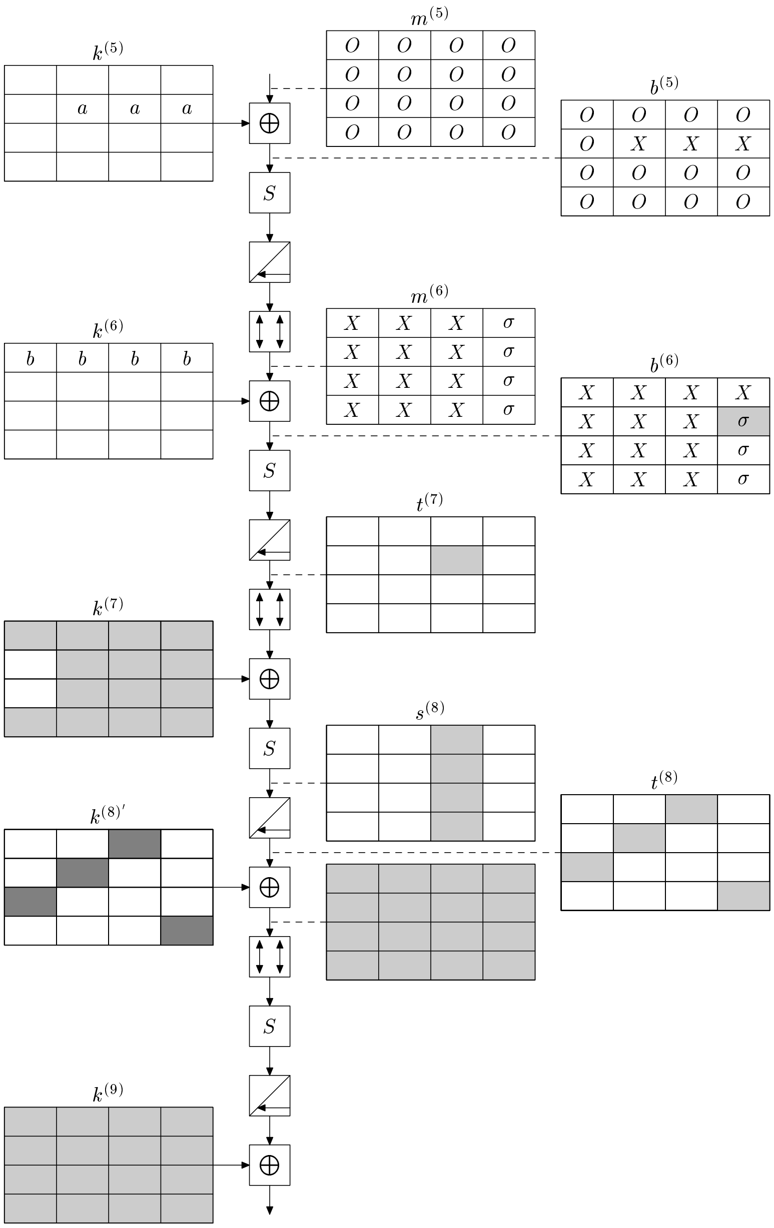

Fig. 2. Rounds 6–9 of the attack.

4.3 An Improvement

We can now use the techniques of section 2.3 to improve this attack. Instead of guessing the eight bytes of the first round key, we use all 2^{64} plaintexts for each of the keys, and sum over all 2^{72} encryptions. We will need about 32 structures of 2^{72} texts overall to uniquely identify the correct key guess, so our plaintext requirement grows to 2^{77}. We now guess the 19 dark gray bytes (152 bits) of $K^{(4)}$ in figure 1. (This also provides the required information to track the differences between the 256 keys.) We decrypt each of the 2^{72} ciphertexts of one structure for one round, and count how often each of the possible values for the four remaining interesting bytes occurs. We are now left with something very similar to equation 1, which requires 2^{48} elementary steps. It is clear that this process is dominated by the work of decrypting the last round. This reduces the attack complexity to $5 \cdot 2^{224}$. (The factor 5 comes from the fact that we need five of these structures before we start cutting into the guesses that dominate the workload.)

4.4 Further Work

There are many ways in which variations on this attack can be made, such as using a different key difference pattern, or possibly applying the partial-sum technique further to reduce the workload. We have not investigated these in any detail. This remains an area for further study.

4.5 Summary

There is a related-key attack on 9 rounds of Rijndael with 256-bit keys that uses 2^{77} plaintexts under 256 related keys, and requires 2^{224} steps to complete.

5 Conclusions

We examined the security of the AES candidate Rijndael, and described several new attacks and unexpected properties of the cipher. Up to now we have only looked at the Rijndael versions with a 128-bit block size. Although similar in structure, Rijndael with larger block sizes is different enough—the byte alignments that are so crucial to some of our attacks are different—that it will have to be analyzed separately.

We introduced the "partial sum" technique, which substantially reduces the workfactor of the dedicated Square attack. We also showed how one may trade texts for time, to penetrate through more rounds of Rijndael when many known texts are available. These techniques allowed us to find attacks that break as many as 7 (of 10) rounds for 128-bit keys, 8 (of 12) rounds for 192-bit keys, and 8 (of 14) rounds for 256-bit keys. Many of these attacks require virtually the entire codebook of texts and hence are not very practical.

The key schedule does not achieve its stated design goals, especially for 192-bit and 256-bit keys. Although we have not found a large-scale exploit of the key schedule properties described in section 3, we find them worrisome.

The 9-round related-key attack has a complexity of 2^{224}, which is of course completely impractical; but it is faster than an exhaustive key search, which is the standard measure to compare against. Our results have no practical significance for anyone using the full Rijndael.

6 Acknowledgments

The "extended Twofish team" met for two week-long cryptanalysis retreats during fall 1999, once in San Jose and again in San Diego. This paper is a result of those collaborations. Our analysis of Rijndael has very much been a team effort, with everybody commenting on all aspects. Tadayoshi Kohno contributed to our discussions. Ian Goldberg very kindly performed some calculations for us. We would like to thank them both for their contributions and for the great time we had together.

References

[Bih93] Eli Biham. New types of cryptanalytic attacks using related keys. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer-Verlag, 1993.

[DBRP99] Carl D'Halluin, Gert Bijnens, Vincent Rijmen, and Bart Preneel. Attack on six rounds of Crypton. In Lars Knudsen, editor, *Fast Software Encryption '99*, volume 1636 of *Lecture Notes in Computer Science*, pages 46–59. Springer-Verlag, 1999.

[DKR97] J. Daemen, L. Knudsen, and V. Rijmen. The block cipher Square. In *Fast Software Encryption '97*, pages 149–165. Springer-Verlag, 1997.

[DR98] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. In *AES Round 1 Technical Evaluation CD-1: Documentation*. NIST, August 1998. See `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/` or `http://www.nist.gov/aes`.

[GM00] Henri Gilbert, Marine Minier. A collision attack on 7 rounds of Rijndael. In *The third Advanced Encryption Standard Candidate Conference*, pages 230–241. NIST, April 2000. See `http://www.nist.gov/aes`.

[KSW96] John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer-Verlag, 1996.

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In *Information and Communications Security, First International Conference Proceedings*, pages 203–207. Springer-Verlag, 1997.

[Luc00] Stefan Lucks. Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In *The third Advanced Encryption Standard Candidate Conference*, pages 215–229. NIST, April 2000. See `http://www.nist.gov/aes`.

A Notation

The original description of Rijndael uses pictures to define the cipher, and re-uses symbols several times. This makes it difficult to refer in an unambiguous manner to intermediate values in an encryption. To help resolve this problem, we define some extra terminology and symbols for various values in the cipher. We have tried to retain as many of the symbols of [DR98] as possible. For completeness, we have named every intermediate value that seemed of use to us. Many of these definitions are not used in this paper but are included for completeness. All our explanations of the symbols refer to the description in [DR98].

3a	The byte $3a_{16}$ (and similarly for all other byte values). This can either be a direct byte value, or it can be interpreted as an element of $\mathrm{GF}(2^8)$.
$a_{i,j}^{(r)}$	The byte at position (i, j) at the beginning of round r.
$b_{i,j}^{(r)}$	The byte at position (i, j) at the output of round r (just after the key addition).
$c(x)$	The polynomial $\mathtt{03}x^3 + \mathtt{01}x^2 + \mathtt{01}x + \mathtt{02}$ that is used to define the MDS matrix.
c_i	The bytes of the ciphertext, where $i \in \{0, \ldots, 4N_b - 1\}$.
C_i	The number of positions that row i is shifted left in the ShiftRow function.
$k_{i,j}^{(r)}$	The expanded key byte in round r at position (i, j) where $r \in \{0, \ldots, N_r\}$, $i \in \{0, \ldots, 3\}$ and $j \in \{0, \ldots, N_b - 1\}$. For $r = 0$, it is the key that is XORed into the state before the first round. The entire round key is referred to as $k^{(r)}$.
$k_{i,j}^{(r)'}$	This is a simple linear function of the round key $k^{(r)}$. XORing $k^{(r)'}$ into the state before the MixColumn operation is equivalent to XORing $k^{(r)}$ into the state after the MixColumn operation (when looking at encryption).
K_i	The bytes of the expanded key in their canonical order, where $i \in \{0, \ldots, 4N_b(N_r + 1) - 1\}$. Note that the bytes $K_0, \ldots, K_{4N_k-1}$ form the key of the cipher itself.
$K_{i,j}^{(s)}$	The expanded key bytes in cycle s at position (i, j), where $s \in \{0, \ldots, N_s - 1\}$, $i \in \{0, \ldots, 3\}$, and $j \in \{0, \ldots, N_k - 1\}$.
$m_{i,j}^{(r)}$	The byte at position (i, j) at the output of the MixColumn operation in round r.
M	The MDS matrix.
N_b	The block size (in bits) divided by 32.
N_k	The number of key bits divided by 32.
N_r	The number of rounds.
N_s	The number of cycles in the key expansion; $N_s = \lceil (N_r + 1)N_b/N_k \rceil$.
p_i	The bytes of the plaintext, where $i \in \{0, \ldots, 4N_b - 1\}$.
r	The round number. The rounds are numbered $1, \ldots, N_r$, and the value 0 is sometimes used to refer to the initial AddRoundKey operation.
$R_i^{(s)}$	The round constant used at position $(i, 0)$ in cycle s.
s	The cycle number in the key expansion. Each cycle produces $4N_k$ expanded key bytes. The cycles are numbered from 0 to $N_s - 1$.

$s_{i,j}^{(r)}$ The byte at position (i,j) at the output of the S-boxes in round r.
S The S-box. Entry x is written as $S[x]$. The inverse S-box is written as S^{-1}.
$t_{i,j}^{(r)}$ The byte at position (i,j) at the output of the ShiftRow operation in round r.

Note that the key schedule operates in what we call "cycles." This is to distinguish it from the rounds of the cipher itself. If the block size and key size are the same, then a cycle corresponds to a round, but this is not the case in general.

We can now give the various formulae through which these values are tied together. This provides a complete specification of the cipher, although not one that is easy to understand. Note that N_b and N_k are the cipher parameters that can each take on the values 4, 6, and 8. The multiplication of two bytes is defined as multiplication in $\mathrm{GF}(2)[x]/(x^8+x^4+x^3+x+1)$ and the byte value $\sum_{i=0}^{7} a_i 2^i$ with $a_i \in \mathrm{GF}(2)$ is identified with the field element $\sum_{i=0}^{7} a_i x^i$.

$$
\begin{aligned}
a_{i,j}^{(1)} &= p_{4j+i} \oplus k_{i,j}^{(0)} && \text{Initial key addition}\\
s_{i,j}^{(r)} &= S[a_{i,j}^{(r)}] && \text{ByteSub}\\
t_{i,j}^{(r)} &= s_{i,(j+C_i) \bmod N_b}^{(r)} && \text{ShiftRow}\\
[m_{0,j}^{(r)},\dots,m_{3,j}^{(r)}]^T &= M[t_{0,j}^{(r)},\dots,t_{3,j}^{(r)}]^T && \text{MixColumn}\\
b_{i,j}^{(r)} &= m_{i,j}^{(r)} \oplus k_{i,j}^{(r)} && \text{AddRoundKey}\\
a_{i,j}^{(r)} &= b_{i,j}^{(r-1)} && \text{for } r = 2,\dots,N_r\\
c_{4j+i} &= t_{i,j}^{(N_r)} \oplus k_{i,j}^{(N_r)} && \text{Final round}\\
k_{i,j}^{(r)} &= K_{4rN_b+4j+i} && \text{Round keys}\\
K_{4sN_k+4j+i} &= K_{i,j}^{(s)} &&\\
K_{i,0}^{(s)} &= K_{i,0}^{(s-1)} \oplus R_i^{(s)} \oplus &&\\
&\qquad S[K_{(i+1) \bmod 4, N_k-1}^{(s-1)}] && \text{for } s = 1,\dots,N_s-1\\
K_{i,4}^{(s)} &= S[K_{i,3}^{(s)}] \oplus K_{i,j}^{(s-1)} && \text{if } N_k = 8\\
K_{i,j}^{(s)} &= K_{i,j-1}^{(s)} \oplus K_{i,j}^{(s-1)} && \text{if } j > 0 \text{ and } (j \neq 4 \vee N_k \neq 8)\\
C_i &= i + \lfloor i/2 \rfloor \cdot \lfloor N_b/8 \rfloor && \text{for } i = 0,\dots,3\\
N_r &= \max(N_b, N_k) + 6 && \text{Number of rounds}\\
R_i^{(s)} &= 0 && \text{for } i > 0\\
R_0^{(s)} &= \text{field element } x^{s-1} &&\\
M &= \begin{pmatrix} \mathsf{02\ 03\ 01\ 01} \\ \mathsf{01\ 02\ 03\ 01} \\ \mathsf{01\ 01\ 02\ 03} \\ \mathsf{03\ 01\ 01\ 02} \end{pmatrix} &&
\end{aligned}
$$