Key Schedule Weaknesses in SAFER+

John Kelsey [*] Bruce Schneier[†] David Wagner[‡]

Abstract

We analyze the key schedule of the SAFER+ block cipher, focusing on the poor diffusion of key material through the cipher when using SAFER+ with 256-bit keys. We develop a meet-in-the-middle attack on 256-bit SAFER+ requiring 12×2^{24} bytes of memory, three known plaintext/ciphertext pairs, and work approximately equivalent to 2^{240} SAFER+ encryptions. We also develop a related-key attack on 256-bit SAFER+ requiring 3×2^{32} chosen plaintexts under two keys with a chosen XOR relationship, and work approximately equivalent to 2^{200} SAFER+ encryptions. We consider a number of other key-schedule properties, such as equivalent keys, DES-style weak and semiweak keys, and key-dependent linear and differential characteristics. We fail to find any such properties, and offer some arguments why some of these are unlikely to exist. Finally, we propose an improvement to the SAFER+ key schedule which defends against our attacks, while causing no apparent weakening of the cipher to other attacks.

1 Introduction

In this paper, we discuss the key schedule of SAFER+ [MKK98], one of the fifteen AES candidates. SAFER+, like the other AES candidates, is actually three different ciphers; one for 128-bit keys, one for 192-bit keys, and one for 256-bit keys. For simplicity of discussion, in the remainder of this document we will call these ciphers SAFER+/128, SAFER+/192, and SAFER+/256, respectively. Some AES candidates, such as CAST-256 [Ada98] and E2 [NTT98], use essentially the same key schedule and cipher structure for all three versions, simply padding shorter keys to 256 (or more) bits before applying the big key schedule. Other AES candidates, such as SAFER+, Rijndael [DR98], and Twofish [SKW+98], make changes to the actual cipher for the different key sizes. (SAFER+ and Rijndael change the encryption function, making the cipher slower for longer keys. Twofish changes the key schedule, so that the cipher encrypts and decrypts at the same speed regardless of key length[1].) In SAFER+, the 128-bit keys are used with eight rounds, the 192-bit keys are used with twelve rounds, and the 256-bit keys are used with sixteen rounds.

We have the following findings:

- We found that in SAFER+/192 and SAFER+/256, the key schedules do a poor job of getting the whole key involved quickly in the encryption process. SAFER+/192 takes five (of twelve) rounds to get the whole key involved in the encryption process; SAFER+/256 takes nine (of sixteen) rounds to do so.
- Due to this slow key diffusion, we were able to find a meet-in-the-middle attack on SAFER+/256. This attack requires work equivalent to about 2^{240} SAFER+/256 encryptions and about 12×2^{24} bytes of memory.
- Also due to this slow key diffusion, we were able to find a related-key attack on SAFER+/256. This attack requires very little memory, 3×2^{32} chosen plaintexts encrypted under two different keys with a chosen XOR relationship, and work approximately equivalent to 2^{200} SAFER+/256 encryptions.
- We were unable to find any weak or semi-weak keys, and we strongly suspect that there are none.
- We were unable to find any equivalent keys or any way for pairs of keys to be equivalent. We would be quite surprised to see a pair of equivalent keys for SAFER+.

[*]Counterpane Systems; 101 E Minnehaha Parkway, Minneapolis, MN 55419, USA; kelsey@counterpane.com.
[†]Counterpane Systems; schneier@counterpane.com.
[‡]University of California Berkeley, Soda Hall, Berkeley, CA 94720, USA; daw@cs.berkeley.edu.

[1]Actually, this is true only of the precomputing key schedule used for maximum encryption throughput with low key agility. Some Twofish implementations will have slower encryption *and* slightly slower key scheduling.

- We were unable to find any quasi-weak keys. However, we have no strong intuition about whether or not such keys might exist.
- We were unable to find any way for slide attacks [Wag95, BW99] or related-key slide attacks [Bih94, KSW96, KSW97] to work, due to the key biases.

1.1 Practical Implications of Our Findings

Both the meet-in-the-middle attack and the related-key attack demonstrate substantial violations of the SAFER+/256 security claims. A block cipher with a 256-bit key ought to provide 256 bits of security; that is, an attacker ought to have to try about 2^{255} trial encryptions under different keys, on average, to recover the key.

Neither of our attacks is practical at present, due to their enormous computational requirements. However, there is little reason to use a 256-bit key if it doesn't provide substantial improvements in security over a 192-bit key. This is especially true in the case of SAFER+, since SAFER+/192 is about 1.3 times faster than SAFER+/256.

Failing to find various properties in the key schedule implies good things for the cipher. Weak, semi-weak, equivalent, quasi-weak keys, or any complementation properties would cause security problems in some applications. For example, if SAFER+/256 had pairs of equivalent keys, it would be unsuitable for use as a hash function.

1.2 Guide to the Rest of the Paper

The rest of the paper is organized as follows. We first give a brief description of the SAFER+ cipher and key schedule. Next, we describe our meet-in-the-middle attack, and then our related-key attack. We finally propose a fix to the SAFER+/192 and SAFER+/256 key schedules, and end with some open questions. In an appendix we describe some additional properties of the SAFER+ key schedule.

2 SAFER+ and its Key Schedule

SAFER+ is defined in [MKK98]. The cipher, which is related to the SAFER family of ciphers defined over the last few years [Mas94], consists of a sequence of identical rounds (with only different round keys) and an output transformation. Each round consists of an S-box layer and a mixing layer.

2.1 The Cipher

The S-box layer processes each of the sixteen bytes of input independently: one subkey byte is combined into the input byte, then an 8-bit S-box is applied, then another key byte is combined into the byte. This is done in parallel to all sixteen bytes in the block being encrypted. SAFER+ uses two different byte transformations:

$$y_i = S[x_i \oplus sk_{2i}] + sk_{2i+1}$$

$$y_i = S^{-1}[x_i + sk_{2i}] \oplus sk_{2i+1}$$

This is more detail than we need for most of our analysis. For a more precise description, see [MKK98].

The mixing layer involves adding the bytes together in a complicated structure based on the PHT structure:

$$a = a + b$$

$$b = 2a + b$$

The ultimate effect of the mixing layer is described by a matrix M shown in [MKK98]. Note that the matrix M is not self-inverse, which creates an asymmetry between encryption and decryption.

The output transformation is simply a layer of adding and XORing of key material. Note that the output transformation isn't a full nonlinear layer of the cipher, which also contributes to asymmetry in the cipher. This has a small impact on our attacks, described below.

2.2 The Key Schedule

The SAFER+ key schedule expands a 16-, 24-, or 32-byte key to 272, 400, or 528 bytes of subkey, respectively. Each round uses 32 bytes of subkey; the output transformation uses 16 bytes of subkey. The key is first extended by one byte, thus to 17, 25, or 33 bytes of key; the extended byte is the XOR of all other key bytes. We will refer to that extended key as $e_k[0..n]$. The key schedule also makes use of "key biases," a set of constant bytes we will refer to as $k_b[0..m-1]$.

Let sk_{2i}, sk_{2i+1} be the subkeys used for processing the ith byte of the block. Then, we derive the subkeys as:

$$sk_{2i} = k_b[2i] + (e_k[i] \lll 3i)$$

$$sk_{2i+1} = k_b[2i+1] + (e_k[i+1] \lll 3i)$$

Note the effect of this: Each round has 32 subkey bytes, but they make use of only seventeen bytes of the extended key material. This has an enormous negative impact on the cipher's security, as we discuss below.

2.3 Key Diffusion

One useful thing to look at with any cipher is how quickly the key material affects the internal state of the cipher. Efficient mixing of the key bits into the encryption process minimizes the advantage an attacker can get by guessing some bits of the key, and then mounting an attack based of his guess.

In SAFER+, an examination of key diffusion exposes an apparent weakness in the cipher design. In SAFER+/128, every bit of key is used in the first round and in every round thereafter. This is an excellent property for a cipher to have;[2] an attacker cannot peel even a single round off without guessing the whole key. One might expect to see the same property for SAFER+/192 and SAFER+/256, since 256 bits of key material are actually used in each round. Instead, it takes five rounds (out of twelve) to get all of the key bytes involved in the encryption in SAFER+/192, and it takes nine rounds (out of a total of sixteen) to get the whole key involved in the encryption in SAFER+/256.

To understand the impact of this, consider Table 1, which shows, for a 256-bit key, how many bytes of key material must be guessed to get to complete knowledge of each round's output (guessing from the top) or input (guessing from the bottom).

A less impressive, but still interesting chart exists for 192-bit keys; see Table 2.

Both of these tables demonstrate the failure of SAFER+'s key schedule to handle keys longer than 128 bits. The failure in the 256-bit case is especially harmful.

Early in our analysis of SAFER+, the obviously poor key diffusion led us to expect related-key attacks to be possible. In Section 4, we will describe a related-key attack on SAFER+/256. While considering the key diffusion, we drew Table 1. This led us to our meet-in-the-middle attack on SAFER+/256. Table 1 will appear in slightly changed form twice below, discussing specific attacks on SAFER+/256.

3 A Low-Memory Meet-in-the-Middle Attack on SAFER+/256

A meet-in-the-middle attack on a cipher can be carried out when we can make a guess about part of the key from both the plaintext side and from the ciphertext side, and compute the same internal value from both sides. In this case, we try each possible partial key value from the plaintext side, computing the intermediate value and placing it into a sorted list. We then do the same thing from the ciphertext side, merging the results into the same sorted list. We then look for matches. Matching intermediate values indicate a possible correct guess; if the intermediate values are large enough, then when they match, the probability is very high that the guess from the plaintext side and from the ciphertext side that match are both correct guesses.

To see how the meet-in-the-middle attack on SAFER+ is carried out, we must first consider a variant of Table 1. Table 3 describes the attack.

In the attack, we consider a plaintext/ciphertext pair from the key we're trying to recover. From the plaintext side, we compute all $2^{29\times 8} = 2^{232}$ possible values of extended key bytes used in the first seven rounds. This tells us the value of all but two bytes of the input into the PHT layer in round 7. A brief look at the transition matrix that describes the PHT layer of SAFER+ will reveal an interesting property: Although every byte of output is unknown when a single byte of input is unknown, it is possible to compute expressions in the output bytes from the PHT that are unaffected by the two unknown input bytes. At this point in the attack, we use these expressions as our intermediate cipher state, and put them into our sorted list.

From the ciphertext side, we compute all $2^{30\times 8} = 2^{240}$ values of extended key bytes used in the output transformation and the last seven rounds. This gives us all but two bytes of output from the PHT layer of round 7. We can compute the same expressions as above from these bytes, and put them into our sorted list, as well.

The result is a meet-in-the-middle attack requiring on the order of 2^{240} encryptions and memory. This is an enormous amount of memory. However, we can improve this by noticing that we have guessed mostly the same extended key bytes from both sides. Whenever an extended key byte appears in guesses from both sides, we guess it first, and then try the meet-in-the-middle attack. Thus, we do the following:

1. Guess extended key bytes $0..14, 18..28$, for a total guess of 26 bytes, or 208 bits.

2. Using the above guess, try all possible values for extended key bytes 15..17. Use these bytes to encrypt forward through the output of round 6, and thus to compute a set of bytes derived from the output of round 7 that don't depend on extended key bytes 29..30, and which

[2]Note the key schedule we designed for Twofish [SKW+98].

Round	Top Guess	Bottom Guess	Bits
0	17	–	136
1	19	–	152
2	21	–	168
3	23	–	184
4	25	–	200
5	27	–	216
6	29	–	232
7	31	–	248
8	–	–	—
9	–	30	240
10	–	28	224
11	–	26	208
12	–	24	192
13	–	22	176
14	–	20	160
15	–	18	144
OX	–	16	128

Table 1: Partial Guessing Attack on SAFER+/256.

Round	Top Guess	Bottom Guess	Bits
0	17	–	136
1	19	–	152
2	21	–	168
3	23	–	184
4	–	–	—
5	–	–	—
6	–	–	—
7	–	–	—
8	–	24	192
9	–	22	176
10	–	20	160
11	–	18	144
OX	–	16	128

Table 2: Partial Guessing Attack on SAFER+/192.

Round	Top Guess	Bottom Guess	Comments
0	17	–	
1	19	–	
2	21	–	
3	23	–	
4	25	–	
5	27	–	
6	29	–	
7	–	–	Know all but two bytes into PHT layer.
8	–	–	Know all but two bytes output from above PHT layer.
9	–	30	
10	–	28	
11	–	26	
12	–	24	
13	–	22	
14	–	20	
15	–	18	
OX	–	16	

Table 3: Meeting in the Middle of SAFER+/256.

will also be computable from the bottom. Generate a list of all 2^{24} of these twelve-byte entries, in sorted order.

3. Using the guess in step 1, try all possible values for extended key bytes 29..32. Compute the same set of bytes as will be computed in step 2. For each set of intermediate bytes computed, look it up in the sorted list from the previous step.

4. If one or more matches occur, we try the matched key guesses from the plaintext and ciphertext sides on the two other plaintext/ciphertext pairs. If we get matching values for all three, we are overwhelmingly likely to have the right values for the key.

3.1 Computing the Intermediate Values

The ability to compute the intermediate values is what makes this attack work. Here we explain more clearly how these values are computed.

Consider the transition matrix describing the PHT layer of each round. Let $x_{1..16}$ be the bytes of input to the PHT layer, and $y_{1..16}$ be the bytes of output from the PHT layer. We know $x_{1..14}$ and $y_{3..16}$. What intermediate values can we compute from only these bytes?

Consider the expressions for y_3 and y_4:

$$y_3 = z_3 + x_{15} + x_{16}$$

$$y_4 = z_4 + x_{15} + x_{16}$$

where z_3 and z_4 are values that depend only on bytes $x_{1..14}$, and so are known. We can't yet compute these expressions for y_3 or y_4 from just the known x values, since x_{15}, x_{16} are unknown. However, the expression

$$y_3 - y_4 = z_3 - z_4 + x_{15} - x_{15} + x_{16} - x_{16} = z_3 - z_4$$

is known, since it depends only on $x_{1..14}$. Knowing only $y_{3..16}$, we can still compute this internal value. There are twelve independent expressions of this kind which we can use, all of the form $a_0 y_i - a_1 y_j$. This gives a 96-bit filtering condition, so incorrect guesses are expected to cause false alarms with probability only 2^{-96}.

3.2 Detecting Correct Key Guesses

As we described above, for each 208-bit guess, we compute 2^{32} twelve-byte intermediate values from the ciphertext, and 2^{24} such values from the plaintext. There are thus 2^{56} pairs of values, each with a 2^{-96} probability of matching. We expect that about 2^{-40} of our 208-bit guesses will result in a matching value, and so we expect about 2^{168} guesses will require more work.

To handle matching values, we use three plaintext/ciphertext pairs. For any 208-bit guess, if we find a matching pair of entries from one plaintext/ciphertext pair, we try the next. If that one gives us the same matching pair, we try the third. If all three match, then we with overwhelmingly high probability, we have found the right key values.

In practice, this has virtually no noticeable effect on the time taken for the attack. By comparison with 2^{240} work, the additional $2^{168} \times 2^{32}$ work is negligible.

3.3 Total Work Involved

To compute the total work involved, we simply compute the average amount of work done per 208-bit guess. It should be obvious that the rare occurrence of false positives (matches in our twelve intermediate bytes that don't indicate a correct key guess) has an insignificant effect on this total. For each 208-bit guess, we do the following:

1. Encrypt forward 2^{24} different values, each for essentially one half of the SAFER+/256 encryption function, and build the intermediate results into a sorted list, requiring about $24 \times 2^{24} \approx 2^{29}$ swaps in memory. (This is less than 2^{24} SAFER+/256 half encryptions, and could be further optimized with the use of a hash table instead of a sorted list.)

2. Decrypt backward 2^{32} different values, each for essentially one half of the SAFER+/256 encryption function, and check each resulting intermediate value in the table generated above, requiring about 24 compare operations. (This is less than 2^{32} SAFER+/256 half encryptions, and could be further optimized with the use of a hash table instead of a sorted list.)

Thus, the total requirements for this attack are as follows:

- Work equivalent to about $2^{208} \times 2^{32} = 2^{240}$ SAFER+/256 encryptions.
- Memory of about 12×2^{24} bytes of memory.
- Three known plaintexts and their corresponding ciphertexts.

4 A Related-Key Attack on SAFER+/256

Related-key attacks have been extensively discussed in [WH87, Bih94, KSW96, KSW97]. A related-key attack is an attack in which encryptions are requested under two or more keys with a relationship chosen by the attacker. By observing the resulting ciphertexts, the attacker is able to learn something about the actual keys used.

When we first saw the SAFER+/256 key schedule, we suspected a related-key attack might be possible, because of the poor key diffusion. The best related-key attack we have found so far is fairly straightforward:

1. Change some extended key bytes that are used in the early rounds, and then aren't used again until much later in the encryption process. Use these changes to determine $K' = K^* \oplus K$, where K is the original key and K^* is the altered key.

2. Choose difference $X' = X \oplus X^*$ to offset the difference in K with some reasonably high probability.

3. When we get a right pair, we will get the same values after several rounds of encrypting X under key K, and of encrypting X^* under key K^*. We guess the last few rounds' extended key bytes to test this.

Note Table 4, which shows the key bytes used in each round's nonlinear layer in SAFER+/256.

From this, we can see two facts that will be used below to build a related-key attack on SAFER+/256:

1. Extended key bytes 2 and 3, after being used in rounds 0 and 1, are not used again until round 10.

2. We can learn the output from round 11 by guessing 24 bytes (192 bits) of extended key material.

4.1 Overview of The Attack

In the attack, we do not start out knowing K, but we get to choose an XOR relationship between K and K^*, a related key. We then are able to request a sequence of chosen plaintexts under both keys. We use the relationships between the resulting ciphertexts to learn the original key, K. Our goal is to make a change in the key, and find some way to make an offsetting change in the plaintext encrypted under the changed key. We will try many plaintext pairs (3×2^{32}) under these two different keys; most of these pairs will diverge after a round or two of encryption, but in a small number of pairs, called *right pairs*, the changes will come out of round 0 in a certain form, will cancel out with key differences in the nonlinear

Round	Bottom Guess	Ext. Key Bytes
0	–	0-16
1	–	2-18
2	–	4-20
3	–	6-22
4	–	8-24
5	–	10-26
6	–	12-28
7	–	14-30
8	–	16-32
9	30	18-32,0-1
10	28	20-32,0-3
11	26	22-32,0-5
12	24	24-32,0-7
13	22	26-32,0-9
14	20	28-32,0-11
15	18	30-32,0-13
OX	16	32,0-14

Table 4: How Key Bytes are Used (and Guessed) in SAFER+/256.

layer of round 1, and will not reappear until the nonlinear layer of round 10. This will allow us to guess enough key material to see the output of round 11. ¿From this, we are able to distinguish right pairs from wrong pairs, when we have guessed the right value for that part of the key. Detecting right pairs, then, is how we determine whether our partial key guess is correct.

We will change key bytes 2 and 3 (numbering from zero). In the first round, we choose pairs to be encrypted under each key that will, in some pairs, lead to an output difference of $(0, 0, 128, 0, 128, 128, 0, ..., 0)$. Such pairs will lead to an input difference into the second round of $(128, 128, 0, 0, ..., 0)$. The changed key bytes will then have a probability of approximately 2^{-8} of canceling out this change, so that the output of the second round has an all-zero difference between the encryptions with two different keys. When all this happens, we say that we have a right pair. For such a right pair, we can guess the last 192 bits of key material used, and decrypt up through the output of round 11. This allows us to compute the same twelve bytes of intermediate result as we used in the meet-in-the-middle attack, above. If these bytes are identical for the pair of texts being considered, it means that this is very likely to be a right pair (probability of 2^{-96} for wrong pairs, and 1 for right pairs). This works because the changed key bytes, once used, don't get reused until round 10 (numbering from 0).

The work involved in the attack is the partial guess (192 bits) for each of 256 candidate plaintext pairs encrypted under the two related keys, plus a small additional amount of work as needed to weed out false alarms. Note that we do *not* try our partial guess on all plaintext pairs. Instead, we use parts of our guess to determine which pairs could be right pairs if this guess is correct. We thus do about 2^{200} SAFER+/256 partial trial decryptions for the attack. That's approximately the same amount of work as 2^{199} full SAFER+/256 encryptions. We trial decrypt pairs of blocks, so our total work is thus approximately equivalent to 2^{200} SAFER+ encryptions.

We thus do work approximately equivalent to 2^{200} SAFER+/256 encryptions, and request $3 * 2^{32}$ chosen plaintexts under each of the two related keys, and thus learn 192 bits of the extended key. The remaining key material can be learned after this with a 64-bit search.

4.2 Choosing the Key Difference

The key difference is $(0, 0, c, 128, 0, 0, ..., 0)$, where the differences are specified in terms of bytes, and c is some 1-byte difference other than 0 or 128. (About half of all possible c values ought to work for this attack.) We must choose c so that for at least one pair of byte u, we have the property that

$$exp[u] - exp[u \oplus 128 \oplus c] = 128,$$

where $exp[x] = 45^x \bmod 256$. The nonlinear layer of round 0 will have the following bytewise differences in its subkey bytes:

$$(0, 0, c, 128, 0, 0, ..., 0)(0, c, 128, 0, 0, ..., 0)$$

The nonlinear layer of round 1 will have the following bytewise differences in its subkey bytes:

$$(c, 128, 0, 0, ..., 0)(128, 0, 0, ..., 0)$$

4.3 Choosing the Plaintext Pairs

By analyzing the matrix that describes the mixing layer, we were able to find the following differential:

$$(0, 0, 128, 0, 128, 128, 0, 0, ..., 0) \longrightarrow (128, 128, 0, 0, ..., 0)$$

We were also able to see that with a plaintext pair with the difference $(128, 128, 0, 0, ..., 0)$ going into round 1, our chosen-key difference would have some possibility of canceling out against the plaintext difference, meaning that the next eight rounds would be identical in both texts.

This left the question of getting plaintext pairs that would get the desired output difference from round 0. This turns out to be fairly easy to do. We request plaintext pairs P, P' as

$$P = (t_0, u(xk_1), t_2, t_3, w(xk_4), y(xk_5), t_6, i, j, t_9, t_{10}, ..., t_{15})$$

$$P' = (t_0, v(xk_1), t_2 + c, t_3 \oplus 128, x(xk_4), z(xk_5), t_6, i, j, t_9, t_{10}, ..., t_{15})$$

where (u, v) satisfies $log[u + xk_1] \oplus log[v + xk_1] = c$, (w, x) satisfies $exp[w \oplus xk_4] - exp[x \oplus xk_4] = 128$, and (y, z) satisfies $log[y + xk_5] \oplus log[z + xk_5] = 128$ for each possible value of xk_1, xk_4, and xk_5, for indices $0 \leq i < 256$, $0 \leq j < 3$. We thus expect to need 3×2^{32} plaintext pairs.

Note that by having i take on all possible values, in the output of round 0, byte 7 takes on all possible values. After the mixing layer, this forces byte 0 of the input to take on all possible values. Since we know that for at least one value u we will satisfy

$$exp[u] - exp[u \oplus 128 \oplus c] = 128$$

(where $exp[x] = 45^x \bmod 256$), we know that at least one of these values, if it gets the input difference $(128, 128, 0, 0, ..., 0)$ into round 1, will be a right pair.

4.4 Tracing the Differentials through the Cipher

4.5 Extracting the Key

We now have 3×2^{32} pairs encrypted under the two related keys. To extract the key, we must guess enough key material to learn the output from round 11, 192 bits. We must then do a partial decryption on any pairs of texts that might be right pairs, to look for the right pairs.

Instead of doing the trial partial decryptions on all 3×2^{32} text pairs, we use some information about the key we have already guessed. Extended key bytes 1, 4, and 5 must be guessed to get the output from round 11. Each of these bytes leaves only 3×256 pairs of texts as possible right pairs, because of the way we chose the texts.

For each 192-bit guess of the last few rounds' key material, we do the following:

1. Select one set of 256 pairs that (because of the way we chose the pairs) is guaranteed to have at least one right pair in it. (This is one set of pairs with the (u, v), (w, x), and (y, z) pairs selected by the initial key guess, with $j = 0$, and with i taking on all values between 0 and 255 inclusive.)

2. Partially decrypt all 256 pairs, and compute the same 12-byte expressions used in the meet-in-the-middle attack, above. (This amounts to a 12-byte value that will be identical if this is a right pair.)

3. If we get a match between the 12-byte values for both texts in any pair, try the set of 256 pairs with $j = 1$ and with $j = 2$. If these all match, then with overwhelmingly high probability, we have a right pair, *and* our guess of 192 bits of key is correct.

5 Fixing the SAFER+/256 and SAFER+/192 Key Schedules

Having demonstrated weaknesses in the SAFER+/192 and SAFER+/256 key schedules, we wanted to try to fix those weaknesses. The problem with the SAFER+ key schedule for long keys is that it takes too many rounds for the whole key to get involved in the encryption. Both of our attacks exploit this weakness.

All three SAFER+ ciphers use 32 bytes of key material per round. The largest keys these ciphers

Part of Cipher	Truncated Differential in Output	Comments
Input	$(0, ?, c, 128, ?, ?, 0, 0, ..., 0)$	
First Key Addition	$(0, ?, 0, 0, ?, ?, 0, 0, ..., 0)$	Cancel out in bytes 2,3 with prob. = 1.
S-box	$(0, -c, 0, 0, 128, 128, 0, 0, ..., 0)$	Critical events in bytes 1,4,5.
Second Key Addition	$(0, 0, 128, 0, 128, 128, 0, 0, ..., 0)$	Cancel out in byte 1.
Mix	$(128, 128, 0, 0, ..., 0)$	Probability 1.
First Key Addition	$(?, 0, 0, ..., 0)$	Critical event in byte 0, cancel with prob. = 1 in byte 1.
S-box	$(128, 0, 0, ..., 0)$	
Second Key Addition	$(0, 0, ..., 0)$	Cancel in byte 1.

Table 5: Truncated Differential Used in Related Key Attack

need ever process are 32 bytes long. Intuitively, then, getting good key diffusion ought not to be a terribly difficult task. Below, we propose a change to the SAFER+/256 and SAFER+/192 key schedules. This brings these two key schedules into line with the SAFER+/128 key schedule in the sense that with our changes, all three ciphers will have the property that every single round is affected by every bit of key.

Let $sk_{i,j}$ be the jth subkey byte used in round i, xk_i be the i-th byte of extended key (the key with a parity byte appended to the end), and $f(x, i, j)$ represent the bit rotation and addition with a bias word to be applied to the extended key byte upon which $sk_{i,j}$ is to be based. Note that $f(x, i, j)$ is unchanged from the original key schedule.

We then have the formulae:

$$sk_{i,j} = f(xk_{2i+j \bmod 33}, i, j)$$

for 256 bit keys, and

$$sk_{i,j} = f(xk_{2i+j \bmod 25}, i, j)$$

for 192 bit keys.

Table 6 and Table 7 show the extended key bytes to be used for each byte position's two subkey bytes in each round. The rotation of the bytes and the use of key biases is unchanged—we simply select the extended key bytes to be used in a different order than in the original SAFER+ key schedules. In the tables, the first column of numbers represents the round, numbering from 0 to 15 for SAFER+/256, and from 0 to 11 for SAFER+/192. The remaining columns show, for each byte position, which extended key byte is to be used.

For both these new key schedules, we have the following properties:

1. To guess any round's subkey, an attacker must effectively guess the whole key. This eliminates the whole class of attacks that requires a guess of several rounds' key material.

2. Neither of our attacks works on these key schedules.

3. Any change to the key changes at least one byte of the key material used in every round.

4. As discussed in the Appendix, below, SAFER+ doesn't appear to have any weak, semi-weak, or equivalent keys. This appears to be equally true of our proposed key schedule.

5. Every extended key byte is combined with two others at different points in the key schedule.

6. With 192-bit keys, the proposed key schedule uses every byte of the extended key at least once in each round. With 256-bit keys, the proposed key schedule uses every extended key byte in at least fifteen rounds.

7. With 256-bit keys, no extended key byte is ever used twice in the same round.

8. Extended key bytes are never used with themselves in the nonlinear byte substitution layer.

00	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	00
02	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
	20	21	22	23	24	25	26	27	28	29	30	31	32	00	01	02
03	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30	31	32	00	01	02	03	04
04	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	24	25	26	27	28	29	30	31	32	00	01	02	03	04	05	06
05	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	26	27	28	29	30	31	32	00	01	02	03	04	05	06	07	08
06	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	28	29	30	31	32	00	01	02	03	04	05	06	07	08	09	10
07	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	30	31	32	00	01	02	03	04	05	06	07	08	09	10	11	12
08	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	32	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
09	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	00
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
10	20	21	22	23	24	25	26	27	28	29	30	31	32	00	01	02
	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
11	22	23	24	25	26	27	28	29	30	31	32	00	01	02	03	04
	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
12	24	25	26	27	28	29	30	31	32	00	01	02	03	04	05	06
	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22
13	26	27	28	29	30	31	32	00	01	02	03	04	05	06	07	08
	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
14	28	29	30	31	32	00	01	02	03	04	05	06	07	08	09	10
	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
15	30	31	32	00	01	02	03	04	05	06	07	08	09	10	11	12
	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
OX	32	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14

Table 6: Use of Extended Key Bytes in Proposed New SAFER+/256-SK Key Schedule.

9. Both of these proposed key schedules have some bytes that overlap between nonlinear layers of successive rounds, so that some bytes may get two values derived from the same extended key byte added into it in succession. (The original SAFER+ key schedule also has this property.) This does not appear to cause any weaknesses in the cipher.

10. Pairs of extended key bytes are used together in the nonlinear layer of the SAFER+ round function in many successive rounds, both in the original key schedule and in our proposed one. However, in ours, in the 256-bit key case, each extended key byte is used with only one other extended key byte in any given round, and it is used with the same key byte in many successive rounds. Thus, in our proposed 256-bit key schedule, extended key byte 15 is used with byte 31 in rounds 0–7, and with extended key byte 32 in rounds 9–15. This also does not appear to cause any weakness in the whole cipher.

We note that these altered key schedules are very similar in spirit to the original key schedules, requiring only a small programming change. Further, they simply apply the same principle as does the SAFER+/128 key schedule; i.e., the whole key affects every round. There is some reason to believe that such a change might be allowed to SAFER+'s key schedule, if it should advance to the second round of AES. In keeping with the naming convention of Lars Knudsen's recommended change to SAFER's key schedule [Knu95a], we suggest calling the changed ciphers SAFER+/192-SK and SAFER+/256-SK.

00	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	00	01	02	03	04	05	06
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17
	18	19	20	21	22	23	24	00	01	02	03	04	05	06	07	08
02	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
	20	21	22	23	24	00	01	02	03	04	05	06	07	08	09	10
03	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	00	01	02	03	04	05	06	07	08	09	10	11	12
04	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	24	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
05	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	00
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
06	12	13	14	15	16	17	18	19	20	21	22	23	24	00	01	02
	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
07	14	15	16	17	18	19	20	21	22	23	24	00	01	02	03	04
	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
08	16	17	18	19	20	21	22	23	24	00	01	02	03	04	05	06
	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22
09	18	19	20	21	22	23	24	00	01	02	03	04	05	06	07	08
	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
10	20	21	22	23	24	00	01	02	03	04	05	06	07	08	09	10
	11	12	13	14	15	16	17	18	19	20	21	22	23	24	00	01
11	22	23	24	00	01	02	03	04	05	06	07	08	09	10	11	12
	13	14	15	16	17	18	19	20	21	22	23	24	00	01	02	03
OX	24	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14

Table 7: Use of Extended Key Bytes in Proposed New SAFER+/192-SK Key Schedule.

6 Open Questions

These results clearly raise a number of questions about the design of the SAFER+ key schedule:

1. Are our meet-in-the-middle and related-key attacks the best of their kind that can be mounted? Can similar attacks be mounted somehow on SAFER+/192?

2. Can a variant of our related-key attack be used to find collisions more easily than should be possible when SAFER+/192 or SAFER+/256 is used as a hash function?

3. Are other attacks available which exploit the poor key diffusion of SAFER+/192 and SAFER+/256?

4. Are there attacks available on our proposed SAFER+/192-SK and SAFER+/256-SK key schedules?

5. Are there other interesting key schedule properties of SAFER+ yet to be discovered?

We hope that SAFER+ will receive more study to address these issues.

7 Acknowledgements

The authors wish to thank Jim Massey for useful feedback on this paper.

References

[Ada98] C. Adams, "The CAST-256 Encryption Algorithm," NIST AES Proposal, Jun 98.

[Bih94] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Journal of Cryptology*, v. 7, n. 4, 1994, pp. 229–246.

[BPS90] L. Brown, J. Pieprzyk, and J. Seberry, "LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications," *Advances in Cryptology — AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 229–236.

[BS93] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[BW99] A. Biryukov and D. Wagner, "Slide Attacks," *Fast Software Encryption, 6th International Workshop Proceedings*, Springer-Verlag, to appear.

[DR98] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, Jun 98.

[HKM95] C. Harpes, G. Kramer, and J. Massey, "A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma," *Advances in Cryptology — EUROCRYPT '95 Proceedings*, Springer-Verlag, 1995, pp. 24–38.

[HM97] C. Harpes and J. Massey, "Partitioning Cryptanalysis," *Fast Software Encryption, 4th International Workshop Proceedings*, Springer-Verlag, 1997, pp. 13–27.

[Knu93] L.R. Knudsen, "Cryptanalysis of LOKI," *Advances in Cryptology — ASIACRYPT '91*, Springer-Verlag, 1993, pp. 22–35.

[Knu95a] L.R. Knudsen, "A Key-Schedule Weakness in SAFER K-64," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, pp. 274–286.

[Knu95b] L.R. Knudsen, "Truncated and Higher Order Differentials," *Fast Software Encryption, 2nd International Workshop Proceedings*, Springer-Verlag, 1995, pp. 196–211.

[Knu95c] L.R. Knudsen, "New Potentially 'Weak' Keys for DES and LOKI," *Advances in Cryptology — EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 419–424.

[KSW96] J. Kelsey, B. Schneier, and D. Wagner, "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 237–251.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner, "Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA," *Information and Communications Security, First International Conference Proceedings*, Springer-Verlag, 1997, pp. 203–207.

[Mas94] J.L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 1–17.

[Mat94] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology — EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 386–397.

[MKK98] J. Massey, G. Khachatrian, and M. Kuregian, "Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES)," NIST AES Proposal, 1998.

[NBS77] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[NTT98] Nippon Telephone and Telegraph, "Specification of E2 — a 128-bit Block Cipher," NIST AES Proposal, Jun 98.

[SKW+98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher," NIST AES Proposal, Jun 98.

[Wag95] D. Wagner, "Cryptanalysis of S-1," sci.crypt Usenet posting, 27 Aug 1995.

[WH87] R. Winternitz and M. Hellman, "Chosen-key Attacks on a Block Cipher," *Cryptologia*, v. 11, n. 1, Jan 1987, pp. 16–20.

A Additional Properties of the SAFER+ Key Schedule

A.1 Round Subkeys and "Slide" Attacks

In some ciphers, it is possible for a sequence of subkeys to repeat somewhere during the cipher. For example, we might get the same subkeys reused every two rounds throughout a cipher. This allows an attack in which we choose plaintexts to get a pair such that text X_1 is the result of the first two rounds of the encryption of text X_0. If this is the case, then $Y_1 = E(X_1)$ is the result of two more rounds of encryption than $Y_0 = E(Y_0)$. This allows a straightforward attack to recover the first and last two rounds' subkeys. We call this the "slide" attack [Wag95, BW99].

In SAFER+, the subkey biases appear to prevent this from happening. There is no apparent way to get the subkeys to repeat, either for a single key, or in a Biham-style related-key attack [Bih94].

A.2 DES Type Weak and Semi-Weak Keys, Equivalent Keys, and Symmetry Properties

In DES, there exist a few keys that are self-inverse; encrypting any text twice under such a weak key gives back the original input text. We are unable to prove their nonexistence, but there are strong intuitive reasons to believe that there are no such weak keys for SAFER+.

The reason is that the encryption and decryption processes, though quite similar, are not identical. That means that even getting the same subkeys forwards as backwards will not lead to a self-inverse key. The transition matrix which describes the mixing layer is very different from its inverse; there is no way to change the key bytes used in a way that consistently undoes the effect of this transition matrix. Thus, if there are self-inverse keys, they can't be constructed simply by choosing keys to make round 0 the inverse of round 15, round 1 the inverse of round 14, etc. We would find it extremely surprising if there were weak or semi-weak key pairs in SAFER+.

In DES [NBS77], there are also semi-weak key pairs; these are key pairs such that encryption under one of the keys in the pair can be decrypted by encryption under the other key in the pair. For the reasons given above, we do not believe SAFER+ has any semi-weak key pairs.

DES also has quasi-weak keys; pairs of keys for which many more plaintexts than would otherwise be expected have identical ciphertexts for both keys. In some sense, a pair of quasi-weak keys represents a selection of two "nearby" permutations. We have not found any pairs of quasi-weak keys for SAFER+, but do not have a strong reason to doubt that they could exist. However, there does not appear to be any opportunity to construct quasi-weak keys in SAFER+ as they are constructed in [Knu95c], as any change to the key in SAFER+ causes some nonzero difference to result every time a changed part of the key is used.

In some ciphers, there are equivalent keys: pairs of keys that result in an identical encryption. For example, LOKI-89 [BPS90] is known to have classes of equivalent keys [Knu93]. We were unable to find any way to construct equivalent keys for SAFER+. In fact, it looks very difficult to do this for SAFER+. Any change in the key always gets used in many successive rounds, making it quite challenging to find a pair of keys whose changes will cancel one another out for more than one or two rounds. We would be very surprised to find equivalent keys in SAFER+.

A.3 Key-Dependent S-boxes and Characteristics

SAFER+ is typically described as a cipher with fixed S-boxes and key material combined in before and after the S-boxes. However, it is possible to describe SAFER+ in an alternative way, with sixteen different key-dependent S-boxes each round. In this case, key material is combined in directly only during the output transformation.

A key-dependent S-box can be generated by choosing two key bytes, k_0, k_1, and a fixed S-box, $exp[\,]$ or $log[\,]$. We then define $s(x, k_0, k_1) = k_0 + exp[k_1 \oplus x]$. The reason for considering these as key-dependent S-boxes is that there may be properties of the combined S-box that aren't present in the fixed $exp[\,]$ or $log[\,]$ S-boxes. As yet, we have no analysis based on this observation. We would be unsurprised to see some choices of k_0, k_1 for which the resulting S-box would be unusually weak with respect to differential [BS93], linear [Mat94], generalized linear [HKM95], or partitioning [HM97] attacks. Since a given succession of three key bytes generally defines about sixteen different key-dependent S-boxes, though, we would be very surprised to see keys that gave very bad linear, differential, or other properties throughout the whole cipher, due to the way the mixing layer spreads differential or linear characteristics out quickly. However, we might see some keys that gave us unusually bad linear, differential, or other properties through a small number of consecutive rounds of the cipher.

Another possible concern would be if there were some quadruple of extended key bytes possible, (w, x, y, z), such that one of the byte permutations defined by w, x was identical to one formed by y, z in the same positions. If this happened, then by swapping y, z for w, x, we would get no effect in some rounds in our new proposed key schedule. (Assuming the quadruple worked only for one of the byte permutation definitions, either the one starting with an XOR or the one starting with an addition, and considering SAFER+/256-SK, the change have no effect in four of the first eight rounds of the cipher, at the cost of a radical change in the last eight rounds.) This doesn't appear to pose a practical threat to the cipher.