Crypto/Math Paper

SAND2020-7996R

Side Channel Considerations for AES Intermediate Rounds

W.R. Cordwell

Sandia National Laboratories

SAND xxxxxxx

June 2020

You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.

–John von Neumann, discussing with Claude Shannon what to call Shannon's invention

Summary

We illustrate a theoretical side-channel analysis on the intermediate rounds of AES, using only the Hamming weights of the bytes registered after the S-box operation. Input and output state values are unknown.

Simulations and a blind test were used to show the feasibility of the analysis under ideal conditions.

General applicability of the idea and possible extensions are discussed, as well as limiting assumptions.

Some implementation approaches are described in Appendix A, in the case of constrained computing capabilities (desktop or laptop).

Introduction

Most side-channel attacks on the Advanced Encryption Standard (AES)[1], in particular, differential power analysis attacks, focus on the first rounds. Typically, the input values are known, or, at least, largely constant (as in the counter-mode case with unknown IV). Instead, we shall focus on the situation where the state vector of the block that is being encrypted (or decrypted) is unknown. We assume that the key is constant: i.e., it is being used repeatedly for many encryptions of varying inputs. We also assume that we can measure the byte Hamming weights of the outputs of the SBox operation perfectly. This is a strong assumption, but, in some cases, can be approximated if the analyst can repeat the same operation(s) many times, averaging their signals to obtain "clean" traces.

Entropy

Entropy can be described in various ways. One interpretation is that it is the amount of uncertainty in the outcome of a measurement. It is usually defined in terms of a probability distribution. In a discrete probability distribution, the entropy of an event can be quantified as $-lg(p_i)$, with units of "bits", where $lg(.)$ is the base two logarithm, and p_i is the probability that the event i occurred. For example, if there is a secret 8-bit byte value chosen at random from all 256 possible values, each value has a probability of $2^{-8} = \frac{1}{256}$ of occurring, and the amount of uncertainty in selecting or measuring any particular byte value is 8 bits.

Entropy can also be considered as the amount of information in a quantity or in a measurement. Information can be hidden or it can be revealed. If there is a secret byte value involved, there are eight hidden bits of information that we should like to discover. If the byte value is directly revealed to the analyst, he obtains $-lg(\frac{1}{256}) = 8$ bits of information. Instead of learning everything, some partial information about the secret byte might be revealed to the analyst: if the analyst is told that the first bit is a 1, he knows one bit of information. Correspondingly, the uncertainty in the number of possible values has gone down by a factor of $2^{-1} = \frac{1}{2}$ to 128 possibilities (vs. 256), as the probability of such a byte being chosen is $\frac{128}{256} = \frac{1}{2}$. Alternately, if the analyst discovers that the byte value is a multiple of 8, then he knows that the last three bits of the secret byte are all zeros, and he has gained 3 bits of information. This corresponds to a reduction in the search space of $2^{-3} = \frac{1}{8}$, or the number of possibilities being reduced from $2^8 = 256$ to $2^{(8-3)} = 2^5 = 32$ possibilities.

Hamming Weights

The Hamming weight (HW) of a byte is the number of 1s that it contains. It is easiest to visualize in binary format. A byte value of 0 looks like 0000 0000 (in binary), and it has $HW(0) = 0$. A byte value of 255 looks like 1111 1111, and $HW(255) = 8$, as all eight bits are 1. The number of bytes of a particular HW corresponds to the probability that a byte has a particular Hamming weight. For example, there are $\binom{8}{1} = 8$ different bytes that have HW = 1, namely 0000 00001, 0000 0010, 0000 0100, ..., 1000 0000. The most likely HW is 4, with four 1s and four 0s in the byte, and there are $\binom{8}{4} = 70$ such bytes. This means that, if the analyst discovers that a (secret) byte has a HW = 4, that there are 70 possibilities, and each one occurs with a probability of $\frac{1}{256}$. The information that is given away by narrowing the possibilities from 256 to 70 is $-lg(\frac{70}{256}) \approx 1.87$ bits. If the analyst discovers that a secret byte has HW = 0, he knows that it is the all-zeros byte, and he has gained 8 bits of information. Table 1 gives a listing of Hamming weights and the information revealed by their measured values.

HW	**Probability**	**Revealed Information**	**Num Bytes**
0	$^1/_{256}$	8.0	1
1	$^8/_{256}$	5.0	8
2	$^{28}/_{256}$	3.19	28
3	$^{56}/_{256}$	2.19	56
4	$^{70}/_{256}$	1.87	70
5	$^{56}/_{256}$	2.19	56
6	$^{28}/_{256}$	3.19	28
7	$^8/_{256}$	5.0	8
8	$^1/_{256}$	8.0	1

Table 1

Note that at least 1.87 bits of information will always be revealed by measuring the Hamming weight of a byte.

Shannon Entropy

The Shannon entropy of the distribution is the weighted sum of the individual entropies, or $H = -\sum_i p_i \cdot lg(p_i)$. It is the expected value (or average) of the individual entropies. For the Hamming weight probability distribution, the Shannon entropy is approximately 2.54 bits. Thus, *on average*, if a HW value is provided to the analyst, it will give away about 2.5 bits of information about the (secret) byte value.

Theoretical Analysis

AES Round Structure

AES should be familiar to most of the readers. For our purposes, an intermediate round starts by taking the Substitution Box (SBox) of each byte of the previous internal 128-bit state. The sixteen output bytes are grouped into four groups of four bytes. Each group of four bytes is arranged in a column vector, and a MixColumn operation is applied to each column, yielding an output column of four new output bytes for each original column. The MixColumn is an invertible linear operation. Following this, each group of four mixed, updated bytes is XORed with four bytes of secret round key–thus, the 16-byte state can be grouped into four groups of four bytes each, and the round key can be separated into four associated sub-round keys. This ends the first and all intermediate rounds. The next operation, at the beginning of the following round, is to take the SBox of each current byte. The SBox lookup table is listed in Table 2, where the input and output values are listed in hexadecimal. If the input byte equals 7b, for example, you go down to row 70, then over to column b. The listed value is 21 (hex), so SBox(7b) = 21. The SBox is one-to-one, so it is invertible.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2: AES SBox

The goal of the analyst is to recover the secret round key. We shall focus just on a group of four bytes that come together in a MixColumn operation, so we are concerned with recovering the corresponding four bytes of the secret round key, the sub-round key. If the analyst can recover all four sub-round keys, then he has recovered the (entire) round key.

Connection with Hamming Weights and Entropy–The Analysis Idea

As stated in the introduction, we assume that the analyst can measure exactly the Hamming weights of each SBox output byte, but that he otherwise has no knowledge of the exact values being processed.

In terms of entropy, for an initial group of four bytes at the beginning of an intermediate round, there are 32 bits of unknown. The analyst knows the AES algorithm details, so that he can compute, for each possible set of four bytes, the values of the resulting four bytes of state after the MixColumn operation. At that point, four unknown bytes of round key are XORed in, for an additional 32 bits of unknown (entropy). That gives 64 bits of entropy. Equivalently, the analyst needs to know 64 bits to know everything about what is going on in that round, for that set of four bytes.

The analyst is allowed to know, however, the SBox Hamming weights of the initial four bytes of the round (near the beginning of the round, just after the SBox operation). If the HW of a byte is 4, for example, it must have one of 70 possible values: 0000 1111, 0001 0111, 0001 1101, 0001 1011, 0001, 0111, ..., 1111 0000 (in binary). This is better than having 256 possible values, and it corresponds to knowing $\sim$ 1.87 bits of information. He gains more information from the other three initial byte Hamming weights measured just after the initial SBox operation.

Towards the end of the round, we have four bytes from the MixColumn that came from the earlier four bytes. Each of those MixColumn output bytes is XORed with a byte of round key. This ends that round. Those values are then fed through an SBox at the start of the next round. Again, we assume that the analyst can measure each of the HWs of those SBox outputs. Suppose that the HW of the first such SBox output byte is 5. Looking at the (green-colored) values of 5 in Table 3, below, we see that the 56 possible 5s each came from a distinct possible input, which we can read at the left and upper edges of the chart: 01, 04, 05, ..., ec (in hex). So the first byte at the end of the (previous) round, which is a byte of secret round key XORed with a byte of the MixColumn output, *must* be one of those values. Reducing to 56 out of 256 values gives the analyst an additional ~ 2.19 bits of information about the system in the previous round (see Table 1). Again, he gets more bits of information from measuring the other three HWs just after the SBox in that following round.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
00	4	5	6	6	5	5	6	4	2	1	5	4	7	6	5	5
10	4	2	4	6	6	4	4	4	5	4	3	6	4	3	4	2
20	6	7	4	3	4	6	7	4	3	4	5	5	4	4	3	3
30	1	5	3	4	2	4	2	4	3	2	1	4	6	4	4	5
40	2	3	3	3	4	5	4	2	3	5	5	5	3	5	5	2
50	4	4	0	6	1	6	4	5	4	5	6	4	3	3	3	6
60	3	7	4	7	3	4	4	3	3	6	1	7	2	4	6	3
70	3	4	1	5	3	5	3	6	5	5	5	2	1	8	6	4
80	5	2	3	5	6	5	2	4	3	5	6	5	3	5	3	5
90	2	2	5	5	2	3	2	2	3	6	4	2	6	5	3	6
a0	3	3	4	2	3	2	2	4	3	5	4	3	3	4	4	5
b0	6	3	5	5	4	5	4	4	4	4	5	5	4	5	5	1
c0	5	4	3	4	3	4	4	4	4	6	4	5	4	6	4	3
d0	3	5	5	4	2	2	6	3	3	4	5	5	3	3	4	5
e0	4	5	3	2	4	5	4	3	5	4	4	5	5	4	2	7
f0	3	3	3	3	7	5	2	3	2	4	4	4	3	3	6	3

Table 3: SBox Output Hamming Weights

It might appear that the analyst can simply accumulate more and more bits of information by making measurements at the same points in the algorithm in consecutive encryptions (or decryptions). However, each new encryption gives a new set of four bytes of unknown input into the round in question, adding another 32 bits of entropy (the secret sub-round key remains the same, so it does not add any additional entropy beyond its 32 original bits).

The Shannon entropy of the SBox distribution is about 2.5 per byte HW measurement. There are four initial measurements and four post-round measurements involved, so about 20 bits of entropy, on average, might be revealed to the analyst. To determine the four bytes of round key, he needs to recover 64 total bits of entropy (32 bits for the four state bytes, and 32 for the four key bytes) from one encryption. Or, he can look at extra encryptions, where he needs an additional 32 bits of information for each extra set of state bytes, while gaining only 20 bits per encryption. A losing proposition, indeed.

However, assuming that the key is kept constant over many encryption operations (a valid assumption for many systems), the analyst can take advantage of the following: while the Shannon entropy only gives away about 2.5 bits of information per HW measurement, that is an average value. It could happen that several of the HW measurements were in the very low or very high part of the range of Hamming weights. A HW = 0 or 8 gives 8 bits of information to the analyst as he knows that the byte being measured must be 0s or all 1s. A HW = 1 or 7 give eight possibilities (each), or 5 bits of information. What the analyst needs are cases where the initial and final eight measurements together provide more than the 32 bits of entropy that come from the initial (unknown) input. If he could measure HWs that gave, say, 36 bits of information (total, for that encryption), then, potentially, he can gain 4 (= 36 - 32) bits of information about the round key. This is illustrated by example in Figure 1 below, where we assume that the SBox Hamming weights for two consecutive rounds on the relevant sets of four bytes that are processed by the MixColumn operation reveal 36.44

total bits of information. Since the round key is constant for all of the encryptions, every time that he can do this could (potentially) provide another few bits of information about the round key. He would like to accumulate enough information to match the 32 unknown bits corresponding to the 4 unknown bytes of the round key involved with that particular MixColumn portion of the algorithm.

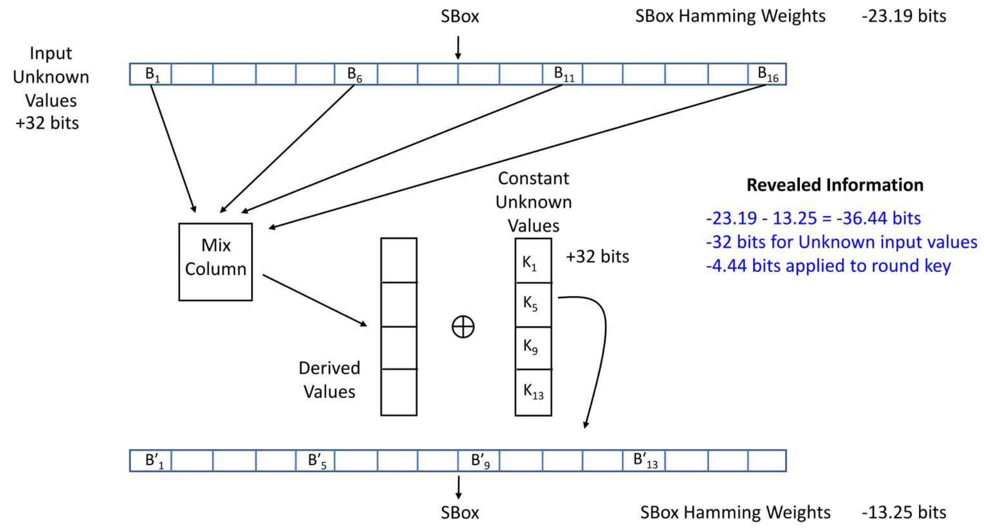

Figure 1

Estimating the Amount of Data Required

By comparing all possible combinations of eight HWs and their probabilities, we can calculate the probability that a set of eight HWs contains more than 32 bits of information. That probability is 0.000824185 . The expected information produced in such a set is 33.4644 bits, or about 1.5 bits extra information, beyond the 32 bits of the unknown inputs, per “lucky” set of Hamming weights (see Figure 2). That would mean that about 22 such measurements are required to recover the secret four-byte round key, which would, in turn, require a data set of about 26,500 total measurements.

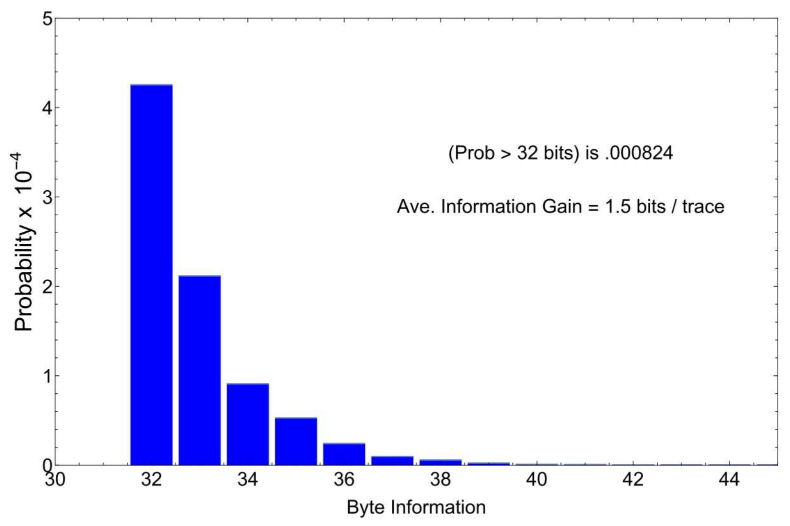

Figure 2

As a check, we ran a simulation to double-check how often the analyst might “get lucky” in the HW measurements. We repeatedly picked eight bytes at random (using Mathematica), four of them representing the initial four bytes coming out of the initial SBox, and four of them representing the four bytes coming out of the SBox of the first part of the following round. For each iteration, we kept track of the cases where the eight measurements exceeded 32 bits of information. (The probability density for the information contained in eight bytes is shown in Figure 2 below.) When we had accumulated enough information to match the 32 bits of the round key, we ended the run and recorded the number of traces/encryptions required for that run. Doing this repeatedly 1000 times, on average it took about 27,500 traces (encryptions) for each run to accumulate the extra 32 bits of information needed. (The results for 100 runs was essentially the same). Of course, the number of AES encryptions varied from run to run, from a minimum of fewer than 10,000 traces to as many as 60,000 traces in the 1000 runs performed. Note that designers might need to assume the worst case.

The same analysis can be performed on the three other groups of four bytes in the intermediate round (almost certainly, a different set of traces, but out of the same data set, would be needed for each different MixColumn group of four bytes), thus recovering the entire round key. For AES-128, it is straightforward to show that knowledge of any round key gives complete knowledge of all of the round keys, in particular, the original 128-bit secret key. For AES-256, recovery of any two consecutive round keys is sufficient to recover the original 256-bit secret key[3] .

We do not necessarily need to recover the entire key. If we can reduce the number of possibilities to 2^{60} or so, we can do a search on the remaining possibilities. With this in mind, we also kept track of the conditions for a group of four bytes to be recovered up to 5 bits (or 4 sets of sub-round key bytes $\times$ 5 bits = 20 remaining bits of entropy for the entire round key), 10 bits (or 40 bits of remaining entropy for the entire round key), and 15 bits (necessitating a 60-bit search). The results are show in Figure 2. The minimum number of traces to reduce the search to 60 bits, in 1000 simulated trials, was 2734.

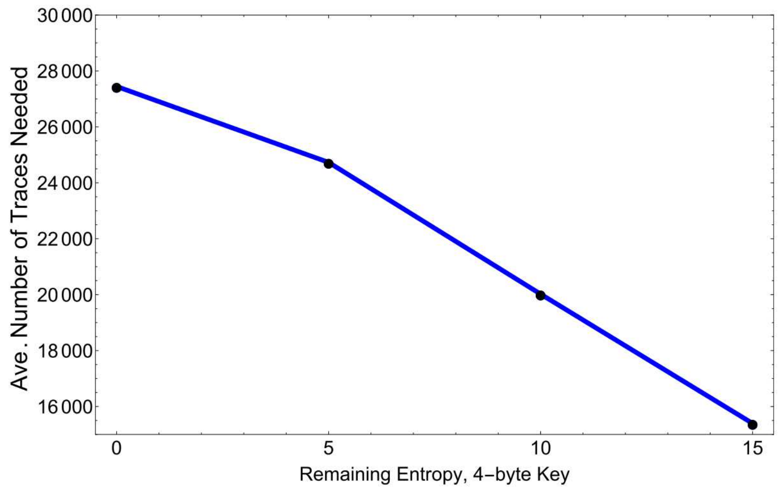

Figure 3

The simulations above assumed that the output of the MixColumn operation XORed with a secret byte was essentially random, although the output is completely determined by the four input bytes. It could be that there is some correlation or dependence between the HW measurements of the input and output values. To test this, we ran ten sets of 100 trials with four random starting values, then computed the output of the MixColumn operation, XORing with a randomly selected four-byte sub-round key that was fixed for a given set of

100 trials. The average number of traces required to gain an extra 32 bits of information was $\sim 30,000$. (There was one outlier in the process, otherwise the average for nine sets of 100 was about 26,000 traces.) The minimum number of traces required among all of the sets and all of the trials was about 8400 for complete recovery, and about 2300 for a 60-bit search. We conclude that it is probably fine to use all random values for projecting possible outcomes, at least to first order.

Some Test Results

We generated a secret four-byte sub-round key, and generated random values for the early post-SBox state values, using MixColumn to derive the next state value of the round, XOR-ing in the round key, and running the output through the next round's SBox. We saved only the Hamming weights of the input state value and the SBox output values, as well as the secret key. We then recovered the secret (four-byte) sub-round key.

For a blind test, we then took provided HW values for a few inner rounds, recovered a secret round key (16 bytes), and reversed the key schedule algorithm to obtain the original secret key (successfully).
All tests were run on a desktop or laptop using Mathematica. Mathematica is expected to be much slower than compiled code, but it was convenient to the Author.

More details are provided in Appendix A on aspects of implementations.

Comments

Having perfect Hamming weight measurements appears to be a strong assumption, although perhaps not unreasonable, as mentioned earlier. If the HW measurements are not perfect, say, a HW might be 0 or 1, the same process might still apply, with an expanded set of state values, implying a larger possible key set. The information content would drop, meaning that more traces would be needed to obtain a sufficient number with total information exceeding 32 bits for each sub-round key (four bytes).

Alternately, the Hamming weight values from registering the MixColumn operation might be available in addition to the SBox HWs. In this case, many (most?) traces might be usable, and the problem approaches the case of known input values. In that case, the number of traces required to recover a 128-bit key is typically about 3 to 5 for perfect byte HW measurements, as shown in the Advanced Crypto Class demo[2].

If one is working with a limited set of traces (thus, limited HW information), one might be able to recover, say, Column 3 of Round Key 5. Using this information and the traces involved, it is likely that the corresponding four bytes of state for those traces would also be recovered. These four bytes could be pushed forward into the next round, or backwards into the previous round, giving 8 bits of information for the particular byte involved (four of them). This would increase the likelihood (but only for those particular traces) that the information content would exceed the necessary 32 bits for several of the (new) sub-round keys involved, again for those particular traces. In other words, information from one round might be usable for recovering keys from other rounds.

Although AES has some nice properties that we were able to exploit, the idea of using side channel information to recover round keys is broadly applicable. It works best when the key schedule is invertible (as with AES), although it might be possible to recover enough round keys and other information to make an exhaustive search feasible in some other cases.

As mentioned above, this analysis, using only SBox HW values, requires a fairly large number of traces/encryptions/decryptions that all use the same secret key (an average of ∼26,500 traces, assuming perfect HW measurements).

Conclusion

Using a straightforward application of entropy/information theory, we have shown that it is theoretically possible to recover the secret AES key using only HW measurements of the SBox outputs in intermediate rounds, viz., without knowing the input or output states of the encryption. This requires about 26,500 encryptions/traces, all using the same secret key.

Appendix A: Possible Implementations

A Basic Approach

The analyst should start by selecting the usable sets of eight HWs, which correspond to four unknown initial bytes of state and four unknown corresponding round key bytes with each set providing a total of more than 32 bits of information.

For the first pass, he picks a usable set that has one of the highest amounts of total information among all of the usable sets. A straightforward approach would then be to take that usable set of HWs, create a list of possible state vectors (from the SBox outputs that begin the round), run each four-byte state through a MixColumn operation, XOR with each 2^{32} possible four-byte round key values, and see if resulting bytes give the proper SBox Hamming weights corresponding to the HW measurement after the SBox operation at the beginning of the following round. If a four-byte putative key passes, he checks it as "OK"; if not, he deletes it from the list. This is, essentially, a search of 32 bits (for the possible round key bytes) + the lg of the number of possible initial states (near the beginning of the round). This is likely to be a high 30's to mid 40's bit search, which is certainly feasible (although difficult on a desktop or laptop). The resulting list of possible (four-byte) keys is probably in the mid-20's of bits[†]. He would then repeat the process with another usable set, but he can restrict the choices of putative keys to his reduced list. This subsequent search is, perhaps, in the mid 20's of bits, which is much easier. Each subsequent operation should reduce the list of possible four-byte keys.

More Efficient Search

We outline a more efficient method by way of an example. Suppose that the four *output* HWs are 1, 8, 7, and 1, respectively. These are the Hamming weights after applying the

[†]List sizes may be given in bits or in number of entries. A list of size 25 bits means that there are 2^{25} entries

SBox, so we look at all of the SBox input possibilities that could result in those HW outputs. These correspond to possible SBox input values (the state XOR the round key), respectively, of

{ 09, 30, 3a, 54, 6a, 72, 7c, bf } ,

{ 7d } ,

{ 0c, 21, 26, 61, 63, 6b, ef, f4 }, and

{ 09, 30, 3a, 54, 6a, 72, 7c, bf } .

If we expanded this set, by combining every possible first value (of eight) with every possible second value (only one, here), with every possible third value and every possible fourth value, we would have a list that is $7 \times 1 \times 7 \times 7 = 343$ possible expanded values of four bytes each, representing the possible values for the state XOR the (unknown) round key. However, it is probably more efficient to defer this.

Instead, we put those values aside, and we consider the possible state values given by the *initial* SBox Hamming weights. Those HWs give, directly, the possible values of the four state bytes near the beginning of the round. For example, if the input HWs were 0, 8, 1, 8, then the possible state values at that point would be, by byte, {00}, {ff}, {01, 02, 04, 08, 10, 20, 40, 80}, {ff}. Because the MixColumn operation, well, mixes these bytes, we need to convert them into a list of possible individual state column values, thus

00, ff, 01, ff

00, ff, 02, ff

00, ff, 04, ff

00, ff, 08, ff

00, ff, 10, ff

00, ff, 20, ff

00, ff, 40, ff

00, ff, 80, ff

We end up with a list of size(# first byte possibilities) × size(# second byte possibilities) × size(# third byte possibilities) × size(# fourth byte possibilities).

We shall call this the state input list. Next, we run MixColumn on each of these possible state values, and obtain a MixColumn list of the same size. This is the list of all possible state values prior to XORing with the secret round key. For example, MC(00, ff, 01, ff) = e4, 19, e7, 1b would be the first entry in the MixColumn list.

Now, we take each four-byte element of the MixColumn list (possible state values) and the four-set element of the (put aside) state XOR round key. That is, we take e4, 19, e7, 1b and XOR the first element of the MixColumn list, e4, with the set { 09, 30, 3a, 54, 6a, 72, 7c, bf }, giving a new set of eight elements. We then XOR 19 with { 7d }, giving one element (a possible value for the second byte of round key), etc. We end up with a new list, with as many entries as the MixColumn list, but each entry comprises four sets of possible secret key byte values for that part of the round key.

This new list implicitly contains all possible round key values (for that four-byte portion of the round key), but in a much more compact form. If, say, the input HW entropy is 20 bits of information, and the output HW entropy is 15 bits of information, then the creation of the initial state is a 2^{20} long list, and the final list, with set entries, is also 2^{20} long, with a total information reduction of only 3 bits from the 32 bits of (that portion of) the round key. Still, a 2^{20} list is much easier to store and manipulate than a 2^{29} long list. Because the number of final list entries is determined by the number of entries for the initial state, it might be better to pick values with a high input HW information content, giving fewer initial state entries.

We do this process for several different sets of input and output HW, giving several lists of the above form,

$\{keybyte1\}_1, \{keybyte2\}_1, \{keybyte3\}_1, \{keybyte4\}_1$

$\{keybyte1\}_2, \{keybyte2\}_2, \{keybyte3\}_2, \{keybyte4\}_2$

$\{keybyte1\}_3, \{keybyte2\}_3, \{keybyte3\}_3, \{keybyte4\}_3$

$$\vdots$$

The question then becomes, what is the most efficient way to combine the lists to produce a single key (or just a few possible keys)?

At this point, one has various choices. For our experiments, we intersected two lists. Intersecting means intersecting each line of the first list (set-wise) with each line of the second list, sorting, and eliminating duplicates. This can result in a fairly long list, so it is probably best to use two of the highest HW values initially, so as to reduce the number of entries of the two lists.

We then, typically, expanded the resulting list of sets into a list of single possible keys (longer, but only one possible key per line). We intersected two more lists of sets, expanded into a list of single key possibilities, intersected that with the previous single key list, deleted duplicates, and so worked our way down to a single key.

In practice, it became clear that it was often faster to reuse a HW data value that had high total information content in conjunction with a new data value, because the shorter resulting lists were easier and faster to use, even though reuse resulted in extra steps.

Later, rather than expanding to distinct single possible keys after merging two lists, we tried just using the lists of sets. Once the initial few merges of set lists were accomplished, the resulting candidate set list was short enough that intersecting it with a new set list went quickly.

It should be apparent that the analyst should use the best sets of Hamming weights available. He should look at the overall information content for each set of eight HWs, as well as the information content for the input and output sets separately when considering the order in which to exploit them.

Simulation Results

For convenience in a simulation, we chose a four-byte secret key at random, and we ran 100,000[†] trials of four random inputs (each) to the first SBox, put the results through a MixColumn, XORed the secret key bytes, and put it through the SBox at the beginning of the following round. The (exact) Hamming weights for the 8 SBox outputs were retained. The two best total HWs corresponded to

i = 36   Hinputs = 26.19   Htotal = 40.25

i = 60   Hinputs = 27.19   Htotal = 39.76

Processing the first set gave a list of keys that was 23.74 bits long (-8.25 bits of the 32 possible bit length), while processing the second gave a list that was 24.23 bits long (-7.75 bits reduction). The first list had 56 entries, and the second had 28 entries. In this case, the smaller number of entries meant that each entry contained more possible keys.

After merging (intersecting) the two lists and deleting any entries where an intersection was the empty set, the merged key list had 221 entries, representing $2^{16.125}$ possible keys (with possible duplications), an entropy reduction of ~ 16 (= 8.25 + 7.75) bits from an initial 32 bits.

In this case, intersecting the sets of possible keys and then expanding to one key per line was much faster than trying to expand the two initial key lists and then intersecting. There were a few duplications, and the final number of possible keys after intersecting, expanding, and deleting duplicates was 16.12 bits

The next two best sets were

i = 23   Hinputs = 23.19   Htotal = 38.76

i = 57   Hinputs = 16.74   Htotal = 39.74

The first gave a list of keys 25.23 bits long (-6.76 bits reduction); the second 24.26 bits long

[†]The relatively large number of trials was to ensure large outlying entropy values, so as to reduce the computation required.

(-7.74 bits reduction). Merging, expanding, and eliminating duplicates gave a key size of 17.54 bits, about a 14.5 bit reduction from 32 bits, as expected.

The fifth and sixth best lists were

i = 30   Hinputs = 22.87   Htotal = 37.44

i = 65   Hinputs = 18.38   Htotal = 36.76

The fifth set gave a list of keys 26.55 bits long (-5.45 bit reduction); the sixth set 27.23 bits long (-4.76 bits reduction). Merging, expanding, and eliminating duplicates resulted in a key size list of 21.66 bits, about what would be expected. We note that processing this last pair of lists took substantially longer on our desktop device.

In the event, it turned out that the first two pairs of sets were sufficient, yielding (only) the correct key. We would have expected a list of about 1.4 bits long (heh). For comparison, the intersection of the first and third pairings was a list of 56 entries, and the second and third pairings intersected to give a list of 140 keys. Intersecting the lists in Mathematica was extremely fast–one would expect that a check of a small ordered list of size n_1 against a longer ordered list of n_2 would take less than $\mathbb{O}(n_1 \cdot lg(n_2))$ operations.

Although we focussed on using high HW input values, to shorten the number of entries in a list of sets, one could work in reverse, if most of the high HW values were on the output side. One would just run the list building in reverse, creating a list of single four-byte values of state XOR possible round keys, then using the Inverse MixColumn operation. The resulting list of single values would then be combined with a single set of values from the early SBox HW set. Because MixColumn and Inverse MixColumn are linear, invertible operations, one ends up with a list of Inverse MixColumn possible secret (four-byte) sub-round keys. When the final list of possibilities is small enough, the final set list can be expanded, MixColumn applied to each value, and a set of possible sub-round keys obtained.

Blind Test

With (only) SBox HW data provided for a few inner rounds by an independent party, we successfully recovered a secret round key (Round 5), then backed it up to recover the original secret key.

Acknowledgements

Thanks to Jason Meeks of Sandia National Laboratories for providing the data for the Blind Test.

Thanks to Josh Etzkin of Sandia National Laboratories for reviewing the paper.

References

[1] Federal Information Processing Standards Publication, FIPS-197, Specification for the ADVANCED ENCRYPTION STANDARD (AES). NIST. Nov. 26, 2001.

[2] Advanced Crypto-Math Class, offered through the AT Program Office. *Sandia National Laboratories.*

[3] Cordwell, W., AES Key Recovery. *Sandia National Laboratories.* 03 Nov 2008.

[4] C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal.* 27:379-423,623-646. 1948.

[5] R.M. Gray. Entropy and Information Theory. Springer-Verlag, New York, NY. 1990.

WRC