

Single-Trace Template Attack on the DES Round Keys of a Recent Smart Card

Mathias Wagner, Stefan Heyse

`mathias.wagner@nxp.com`

Abstract. A new template attack on the DES key scheduling is demonstrated that allows recovery of a sufficiently large portion of the DES key of a recent and widely deployed smart card chip with a *single* EM (electromagnetic) trace during the Exploitation Phase. Depending on the use case, the remainder of the key may then be found with reasonable brute-force effort on a PC. Remaining rest entropies as low as ≈ 19 bits have been found for some single-trace attacks, meaning that effectively 37 bits were recovered in a single trace. The nature of single-trace attacks has it that conventional software countermeasures are rendered useless by this attack, and thus the only remaining remedy is a hardware redesign.

1 Introduction

Side-channel attacks have by now a very long history [1–3], and also the particular variant of template attacks [4–7] has been studied for quite some years. Most of the work has focussed on the data path in the de- or encryption engine, whilst only a few have performed Simple Power Analysis (SPA) on the key scheduling of the AES, Camelia, or Serpent[8–10]. To our knowledge, except for [11], no work has been published yet on template attacks targeting the DES key schedule.

Template attacks take place in two phases — a first Profiling Phase, where templates are generated of the target function when all parameters and data are known. These templates are stored in a data base and are used in the second phase — the Exploitation Phase — to be matched against measurements of the target system when not all data are known anymore. The target system may be a physically different one than the one the templates were created with. The attack is successful, when the patterns found in Exploitation Phase can be matched with the correct templates generated during Profiling Phase. It often does not matter how many measurements (traces) are used in the Profiling Phase, since there it is assumed that the attacker has full control over the device and can take as many measurements as are required. It is even conceivable that more than one device is used in the Profiling Phase and this may in fact improve the robustness of the attack when applied to many target devices, due to the averaging effect. On the other hand, usually, the number of traces available for attacking the target device in the Exploitation Phase may well be limited on account of the countermeasures built into the target device, or the eco system it is operating in.

Table 1. Position of round key bits 0...23 in the original key 0...55 (horizontally) versus rounds 1...16 (vertically). In bold is indicated ring R_0^A .

		Position of round key bits 0...23 in the original key 0...55 (C-Register)																						
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	8	44	29	52	42	14	28	49	1	7	16	36	2	30	22	21	38	50	51	0	31	23	15	35
2	1	37	22	45	35	7	21	42	51	0	9	29	52	23	15	14	31	43	44	50	49	16	8	28
3	44	23	8	31	21	50	7	28	37	43	52	15	38	9	1	0	42	29	30	36	35	2	51	14
4	30	9	51	42	7	36	50	14	23	29	38	1	49	52	44	43	28	15	16	22	21	45	37	0
5	16	52	37	28	50	22	36	0	9	15	49	44	35	38	30	29	14	1	2	8	7	31	23	43
6	2	38	23	14	36	8	22	43	52	1	35	30	21	49	16	15	0	44	45	51	50	42	9	29
7	45	49	9	0	22	51	8	29	38	44	21	16	7	35	2	1	43	30	31	37	36	28	52	15
8	31	35	52	43	8	37	51	15	49	30	7	2	50	21	45	44	29	16	42	23	22	14	38	1
9	49	28	45	36	1	30	44	8	42	23	0	52	43	14	38	37	22	9	35	16	15	7	31	51
10	35	14	31	22	44	16	30	51	28	9	43	38	29	0	49	23	8	52	21	2	1	50	42	37
11	21	0	42	8	30	2	16	37	14	52	29	49	15	43	35	9	51	38	7	45	44	36	28	23
12	7	43	28	51	16	45	2	23	0	38	15	35	1	29	21	52	37	49	50	31	30	22	14	9
13	50	29	14	37	2	31	45	9	43	49	1	21	44	15	7	38	23	35	36	42	16	8	0	52
14	36	15	0	23	45	42	31	52	29	35	44	7	30	1	50	49	9	21	22	28	2	51	43	38
15	22	1	43	9	31	28	42	38	15	21	30	50	16	44	36	35	52	7	8	14	45	37	29	49
16	15	51	36	2	49	21	35	31	8	14	23	43	9	37	29	28	45	0	1	7	38	30	22	42

Starting with a recent generalisation to template attacks put forward in [12], we address in this paper a couple of improvements over what is currently state of the art for template attacks.

In Sec. 2 a template attack on the round key of the DES algorithm is developed. The challenge here is that as the encryption / decryption key and hence the round keys do not change during a cryptographic operation, one cannot rely on the effect of averaging over many traces as much as would be the case for a normal template or DPA attack — some effects simply will not average out, even when taking “infinitely many” traces. We therefore try to be successful with the extreme case of a single trace in the Exploitation Phase — thus likely requiring a final brute-force step. To improve the attack, advantage is taken of a ring structure found in the round keys across all 16 rounds of the DES algorithm. Furthermore, the concept of highly overlapping templates is introduced to reduce the effect of outliers and improve the strength of the attack. This is supplemented with a simple key-search strategy tailored to these new concepts.

Finally, in Sec. 3 this technique is applied to a contemporary smart card with a JAVA OS, where at least the underlying hardware IC has been CC certified at EAL 5+, i.e., at a level with “High Attack Potential”. Although it is hard to say whether the OS manufacturer has obeyed the so-called User Guidance Manual normally issued with a CC-certified product, it can be argued that such a User Guidance Manual would not help, anyway, since for a single-trace attack there is very little that can be done in terms of software countermeasures.

Table 2. Position of round key bits 24...47 in the original key 0...55 (horizontally) versus rounds 1...16 (vertically). In bold is indicated ring R_3^A .

	Position of round key bits 24...47 in the original key 0...55 (D-Register)																																																
	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47																									
1	19	24	34	47	32	3	41	26	4	46	20	25	53	18	33	55	13	17	39	12	11	54	48	27																									
2	12	17	27	40	25	55	34	19	24	39	13	18	46	11	26	48	6	10	32	5	4	47	41	20																									
3	53	3	13	26	11	41	20	5	10	25	54	4	32	24	12	34	47	55	18	46	17	33	27	6																									
4	39	48	54	12	24	27	6	46	55	11	40	17	18	10	53	20	33	41	4	32	3	19	13	47																									
5	25	34	40	53	10	13	47	32	41	24	26	3	4	55	39	6	19	27	17	18	48	5	54	33																									
6	11	20	26	39	55	54	33	18	27	10	12	48	17	41	25	47	5	13	3	4	34	46	40	19																									
7	24	6	12	25	41	40	19	4	13	55	53	34	3	27	11	33	46	54	48	17	20	32	26	5																									
8	10	47	53	11	27	26	5	17	54	41	39	20	48	13	24	19	32	40	34	3	6	18	12	46																									
9	3	40	46	4	20	19	53	10	47	34	32	13	41	6	17	12	25	33	27	55	54	11	5	39																									
10	48	26	32	17	6	5	39	55	33	20	18	54	27	47	3	53	11	19	13	41	40	24	46	25																									
11	34	12	18	3	47	46	25	41	19	6	4	40	13	33	48	39	24	5	54	27	26	10	32	11																									
12	20	53	4	48	33	32	11	27	5	47	17	26	54	19	34	25	10	46	40	13	12	55	18	24																									
13	6	39	17	34	19	18	24	13	46	33	3	12	40	5	20	11	55	32	26	54	53	41	4	10																									
14	47	25	3	20	5	4	10	54	32	19	48	53	26	46	6	24	41	18	12	40	39	27	17	55																									
15	33	11	48	6	46	17	55	40	18	5	34	39	12	32	47	10	27	4	53	26	25	13	3	41																									
16	26	4	41	54	39	10	48	33	11	53	27	32	5	25	40	3	20	24	46	19	18	6	55	34																									

2 Template Attack on the DES Round Key

Clearly, when attacking the round key of the DES using a template attack, the challenge is that in Exploitation Phase the round key will be constant from one DES call to the next, yet with up to 48 bits simultaneously toggling in each of the 16 rounds of the DES. Thus, pattern-template matching will tend to yield poor results for typical template sizes of 6 to 15 bits, where a strong 48-bit cross talk and many patterns accidentally locking into “wrong” templates is to be expected. Consequently, there is little if any improvement expected due to averaging, when increasing the number of traces in the Exploitation Phase.

However, the mathematical properties of the DES key schedule do help to improve template attacks with reasonably small templates. Firstly, we notice that in any given round a subset of 48 bits of the 56 key bits will be used. More precisely, from one round to the next 8 bits get “swapped in”, and 8 bits get “swapped out”.

Secondly, rather than targeting the round key directly, it may be more interesting to target the \oplus or Hamming distance between two successive DES rounds, as this is the relevant information one would expect to leak when the 48-bit register containing the round key of a given DES round gets overwritten by the next round key without any protection built in. The drawback here is that because of the \oplus relation only 28 bits, so half of the key bits, appear to be recoverable at first sight. This can be largely remedied, though, when taking advantage of some properties of the DES key schedule and ordering the \oplus operation in rings, as will be described in detail in the following Subsection.

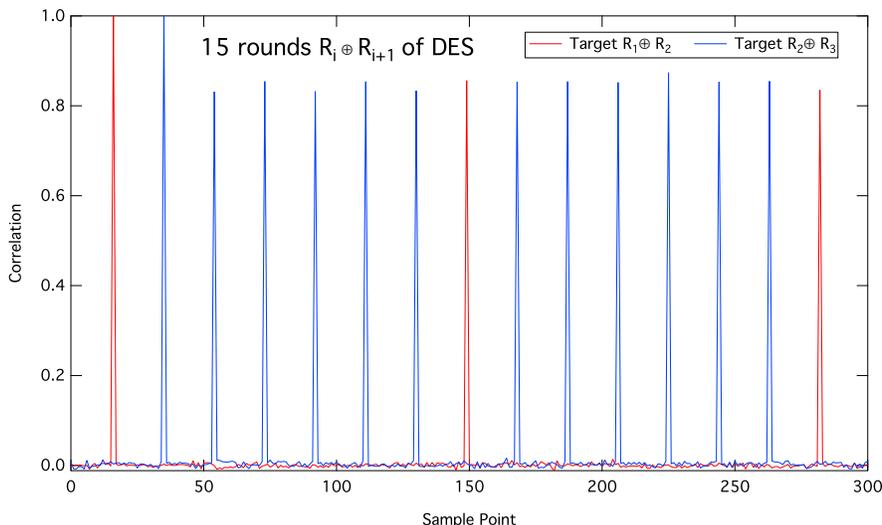


Fig. 1. Correlation functions of the \oplus between two successive rounds in the DES algorithm for an artificial perfectly leaky set of traces. Correlation on the first (red) and second (blue) round are representative of their respective classes. These target rounds achieve a correlation amplitude of 100% since the simulated traces leak perfectly. All satellite peaks are around 83%, which is in line with the fact that these peaks share only 40 bits with the target rounds.

2.1 Exploiting Ring Structures

It turns out that this \oplus select function has some interesting properties, where the 15 \oplus “rounds” (so the 15 \oplus of two successive real DES rounds) fall into one of two classes: The Hamming distances $R_1 \oplus R_2$, $R_8 \oplus R_9$, and $R_{15} \oplus R_{16}$ all have the same characteristic correlation pattern with strong peaks in these 3 “ \oplus rounds”, whilst the Hamming distances $R_2 \oplus R_3 \dots R_7 \oplus R_8$ and $R_9 \oplus R_{10} \dots R_{14} \oplus R_{15}$ form the second class, with 12 strong correlation peaks seen in those “ \oplus rounds”. See Fig. 1 as an illustration based on artificial, perfectly leaky traces generated simply with `printf` statements in a DES C code using an appropriate leakage model. Since these traces leak perfectly, the correlation function is expected to hit 100% at the target “ \oplus round”, whilst for the other satellite peaks one expects their amplitudes at roughly $40/48 = 83\%$, given that all these satellite peaks share only 40 \oplus pairs with the original target “ \oplus round”.

Let us start with the second class containing 12 peaks in the correlation function. Upon closer inspection of the \oplus between any two rounds (see Tables 1 and 2 for details on the round key bits¹), one finds that the key bits that are connected by \oplus from one round to the next form 4 so-called A rings, each ring

¹ In enumerating the DES keys we follow [13], except that we use base 0 and remove all parity bits.

being 14 bits long, thereby spanning all 56 bits as they should,

$$\begin{aligned}
R_0^A : & \\
& 0 \rightarrow 14 \rightarrow 28 \rightarrow 42 \rightarrow 31 \rightarrow 45 \rightarrow 2 \rightarrow 16 \rightarrow 30 \rightarrow 44 \rightarrow 1 \rightarrow 15 \rightarrow 29 \rightarrow 43 \rightarrow 0 \\
R_1^A : & \\
& 3 \rightarrow 17 \rightarrow 4 \rightarrow 18 \rightarrow 32 \rightarrow 46 \rightarrow 5 \rightarrow 19 \rightarrow 33 \rightarrow 47 \rightarrow 6 \rightarrow 20 \rightarrow 34 \rightarrow 48 \rightarrow 3 \\
R_2^A : & \\
& 7 \rightarrow 21 \rightarrow 35 \rightarrow 49 \rightarrow 38 \rightarrow 52 \rightarrow 9 \rightarrow 23 \rightarrow 37 \rightarrow 51 \rightarrow 8 \rightarrow 22 \rightarrow 36 \rightarrow 50 \rightarrow 7 \\
R_3^A : & \\
& 10 \rightarrow 24 \rightarrow 11 \rightarrow 25 \rightarrow 39 \rightarrow 53 \rightarrow 12 \rightarrow 26 \rightarrow 40 \rightarrow 54 \rightarrow 13 \rightarrow 27 \rightarrow 41 \rightarrow 55 \rightarrow 10 \quad (1)
\end{aligned}$$

where $k_i \rightarrow k_j$ stands for $k_i \oplus k_j$. Based on this, we construct now 8 sets of templates — two sets for each ring to avoid templates having too many bits. Although each ring is 14 bits long, we split every ring into $2 \times (7 + 1)$ bits — so 8 bits for each template — such that the templates overlap in each ring on either side of the template. To be more precise, for ring R_0^A the first set of templates would comprise of bits $\{0, 14, 28, 42, 31, 45, 2, 16\}$, whilst the second set of templates comprises of bits $\{16, 30, 44, 1, 15, 29, 43, 0\}$. Both halves of this ring then overlap in the bits 0 and 16. This is how templates are constructed when targeting the key bits themselves. When targeting the \oplus between two neighbouring key bits, the templates are constructed by taking all \oplus between the bits $\{0, 14, 28, 42, 31, 45, 2, 16, 30\}$ for the first template, and between the bits $\{16, 30, 44, 1, 15, 29, 43, 0, 14\}$ for the second. This way, also for the case of \oplus there are 8 bits per template and overlap between templates is again guaranteed.

The first advantage of doing so is that although there are 256 templates in each set, because of the overlapping 2 bits, we can eliminate wrong pattern-template matches that are not compatible with each other in the two sets belonging to the same ring, thereby effectively recovering up to roughly 2 bits in the search space again. See Sec. 2.2 for more details. It would have been of course also possible to go for 7-bit templates that do not overlap in the first place, but

Table 3. Ring A templates for the bits themselves, as well as the \oplus Hamming distance function between these bits.

R_0^A : bits	$\{0, 14, 28, 42, 31, 45, 2, 16\}$	$\{16, 30, 44, 1, 15, 29, 43, 0\}$
R_1^A : bits	$\{3, 17, 4, 18, 32, 46, 5, 19\}$	$\{19, 33, 47, 6, 20, 34, 48, 3\}$
R_2^A : bits	$\{7, 21, 35, 49, 38, 52, 9, 23\}$	$\{23, 37, 51, 8, 22, 36, 50, 7\}$
R_3^A : bits	$\{10, 24, 11, 25, 39, 53, 12, 26\}$	$\{26, 40, 54, 13, 27, 41, 55, 10\}$
R_0^A : \oplus	$\{0 \oplus 14, 14 \oplus 28, 28 \oplus 42, 42 \oplus 31, 31 \oplus 45, 45 \oplus 2, 2 \oplus 16, 16 \oplus 30\}$	
	$\{16 \oplus 30, 30 \oplus 44, 44 \oplus 1, 1 \oplus 15, 15 \oplus 29, 29 \oplus 43, 43 \oplus 0, 0 \oplus 14\}$	
R_1^A : \oplus	$\{3 \oplus 17, 17 \oplus 4, 4 \oplus 18, 18 \oplus 32, 32 \oplus 46, 46 \oplus 5, 5 \oplus 19, 19 \oplus 33\}$	
	$\{19 \oplus 33, 33 \oplus 47, 47 \oplus 6, 6 \oplus 20, 20 \oplus 34, 34 \oplus 48, 48 \oplus 3, 3 \oplus 17\}$	
R_2^A : \oplus	$\{7 \oplus 21, 21 \oplus 35, 35 \oplus 49, 49 \oplus 38, 38 \oplus 52, 52 \oplus 9, 9 \oplus 23, 23 \oplus 37\}$	
	$\{23 \oplus 37, 37 \oplus 51, 51 \oplus 8, 8 \oplus 22, 22 \oplus 36, 36 \oplus 50, 50 \oplus 7, 7 \oplus 21\}$	
R_3^A : \oplus	$\{10 \oplus 24, 24 \oplus 11, 11 \oplus 25, 25 \oplus 39, 39 \oplus 53, 53 \oplus 12, 12 \oplus 26, 26 \oplus 40\}$	
	$\{26 \oplus 40, 40 \oplus 54, 54 \oplus 13, 13 \oplus 27, 27 \oplus 41, 41 \oplus 55, 55 \oplus 10, 10 \oplus 24\}$	

Table 4. Ring B templates for the bits themselves, as well as the \oplus Hamming distance function between these bits.

R_0^B : bits	$\{0, 7, 14, 21, 28, 35, 42, 49\}$	$\{49, 31, 38, 45, 52, 2, 9, 16\}$
	$\{16, 23, 30, 37, 44, 51, 1, 8\}$	$\{8, 15, 22, 29, 36, 43, 50, 0\}$
R_1^B : bits	$\{3, 10, 17, 24, 4, 11, 18, 25\}$	$\{25, 32, 39, 46, 53, 5, 12, 19\}$
	$\{19, 26, 33, 40, 47, 54, 6, 13\}$	$\{13, 20, 27, 34, 41, 48, 55, 3\}$
$R_0^B : \oplus$	$\{0 \oplus 7, 7 \oplus 14, 14 \oplus 21, 21 \oplus 28, 28 \oplus 35, 35 \oplus 42, 42 \oplus 49, 49 \oplus 31\}$	
	$\{49 \oplus 31, 31 \oplus 38, 38 \oplus 45, 45 \oplus 52, 52 \oplus 2, 2 \oplus 9, 9 \oplus 16, 16 \oplus 23\}$	
	$\{16 \oplus 23, 23 \oplus 30, 30 \oplus 37, 37 \oplus 44, 44 \oplus 51, 51 \oplus 1, 1 \oplus 8, 8 \oplus 15\}$	
	$\{8 \oplus 15, 15 \oplus 22, 22 \oplus 29, 29 \oplus 36, 36 \oplus 43, 43 \oplus 50, 50 \oplus 0, 0 \oplus 7\}$	
$R_1^B : \oplus$	$\{3 \oplus 10, 10 \oplus 17, 17 \oplus 24, 24 \oplus 4, 4 \oplus 11, 11 \oplus 18, 18 \oplus 25, 25 \oplus 32\}$	
	$\{25 \oplus 32, 32 \oplus 39, 39 \oplus 46, 46 \oplus 53, 53 \oplus 5, 5 \oplus 12, 12 \oplus 19, 19 \oplus 26\}$	
	$\{19 \oplus 26, 26 \oplus 33, 33 \oplus 40, 40 \oplus 47, 47 \oplus 54, 54 \oplus 6, 6 \oplus 13, 13 \oplus 20\}$	
	$\{13 \oplus 20, 20 \oplus 27, 27 \oplus 34, 34 \oplus 41, 41 \oplus 48, 48 \oplus 55, 55 \oplus 3, 3 \oplus 10\}$	

in general it should be advantageous to use templates as large as possible to reduce the impact of toggling bits outside the template (the cross talk referred to elsewhere).

Another and likely more important advantage of forming overlapping templates along the rings is that in this way all \oplus and hence key bits are related to each other in a fixed manner. If we fix one key bit in the ring arbitrarily, all other key bits are uniquely determined along the ring. Hence, the only degree of freedom remaining is whether the first key bit was set to 0 or 1, and thus for each ring only one key bit cannot be recovered in the attack — and not as many as half of all bits, as was the case for uncorrelated \oplus relations. Consequently, for the A rings we find that up to $56 - 4 = 52$ bits can be recovered in the attack, which is substantially more than the original 28 bits.

In a similar manner, the first class containing 3 peaks in the correlation function of Fig. 1 leads to rings as well, namely to two so-called B rings of 28 bits each,

$$\begin{aligned}
 R_0^B : & \\
 & 0 \rightarrow 7 \rightarrow 14 \rightarrow 21 \rightarrow 28 \rightarrow 35 \rightarrow 42 \rightarrow 49 \rightarrow 31 \rightarrow 38 \rightarrow 45 \rightarrow 52 \rightarrow 2 \rightarrow 9 \rightarrow 16 \\
 & \rightarrow 23 \rightarrow 30 \rightarrow 37 \rightarrow 44 \rightarrow 51 \rightarrow 1 \rightarrow 8 \rightarrow 15 \rightarrow 22 \rightarrow 29 \rightarrow 36 \rightarrow 43 \rightarrow 50 \rightarrow 0 \\
 R_1^B : & \\
 & 3 \rightarrow 10 \rightarrow 17 \rightarrow 24 \rightarrow 4 \rightarrow 11 \rightarrow 18 \rightarrow 25 \rightarrow 32 \rightarrow 39 \rightarrow 46 \rightarrow 53 \rightarrow 5 \rightarrow 12 \rightarrow 19 \\
 & \rightarrow 26 \rightarrow 33 \rightarrow 40 \rightarrow 47 \rightarrow 54 \rightarrow 6 \rightarrow 13 \rightarrow 20 \rightarrow 27 \rightarrow 34 \rightarrow 41 \rightarrow 48 \rightarrow 55 \rightarrow 3
 \end{aligned} \tag{2}$$

Again, these rings are decomposed into $4 \times (7+1)$ bits, so 8 bits for each template, such that the templates overlap by one bit on either side.

Now, rings A and B leak in different rounds of the DES key schedule as visualised in Fig. 1, and spelled out in detail in Table 5. Hence, it must be possible to combine and stack their leakages and thereby improve the attack substantially. Indeed, when inspecting rings A and B of Eqs. (1) and (2) one

For these two C rings we decided to construct the templates maximally overlapping. So, if the first template is, e.g.,

$$\begin{array}{cccc}
 7 & \rightarrow & 21 & \rightarrow & 35 & \rightarrow & 49 \\
 \nearrow & \searrow & \nearrow & \searrow & \nearrow & \searrow & \\
 0 & \rightarrow & 14 & \rightarrow & 28 & \rightarrow & 42
 \end{array} \tag{4}$$

then the next template “to its right” is

$$\begin{array}{cccc}
 21 & \rightarrow & 35 & \rightarrow & 49 & \rightarrow & 38 \\
 \nearrow & \searrow & \nearrow & \searrow & \nearrow & \searrow & \\
 14 & \rightarrow & 28 & \rightarrow & 42 & \rightarrow & 31
 \end{array} \tag{5}$$

and so on. These templates contain $12 \oplus$ relations, but those are not all linearly independent. In fact, there are 5 closed \oplus loops found in each template, e.g., $0 \rightarrow 7 \rightarrow 14 \rightarrow 0$, meaning 5 \oplus equations that need to be always satisfied, and hence the number of linearly independent variables is only $f = 12 - 5 = 7$ in this case. The internal bit representation of these templates has therefore been chosen such that the f linearly independent bits are the lowest-valued ones when counting from right to left, like

$$\{7\oplus 14, 14\oplus 21, 21\oplus 28, 28\oplus 35, 35\oplus 42, 42\oplus 28, 28\oplus 14, 14\oplus 0, 0\oplus 7, 7\oplus 21, 21\oplus 35, 35\oplus 49\}$$

for the template of Eq. (4). Numerically, it is then easy to mask out all bits that are linearly dependent when needed. In fact, the linearly independent bits are simply the \oplus relations obtained when traversing the template of Eq. (4) on the outer “rim” anticlockwise, starting at bit 49, and ending at 42.

Because of the strong overlap between the 28 templates thus constructed, the original pattern–template rankings get modified as discussed in more detail in Sec. 2.2, but for now let it suffice to say that the templates of Eq. (4) and Eq. (5) overlap by 5 linearly independent bits.²

The advantage of using C–type templates like Eq. (4) is that they concentrate all relevant electrical variations in as few template positions as possible.³ For instance, when an “inner” key bit such as bit 21 changes its value in the first

² Another way of working out the effective depth of each pattern–template ranking list after accounting for overlapping is to observe that the overlap between Eqs. (4) and (5) is a set of eight \oplus links, but there are three closed \oplus loops included within those, and hence only five degrees of freedom left. Now, the total number of degrees of freedom for the \oplus in template Eq. (4) is seven, meaning that after accounting for overlap, only two degrees of freedom remain. This makes sense, since there are also only two key bits in this non–overlapping region.

³ Here and in the following we will refer to *template positions* or *template ranking lists* when we refer to a template, or a list of templates, at a particular position along the rings. In another context the notion *template* may also refer to the actual *template value*.

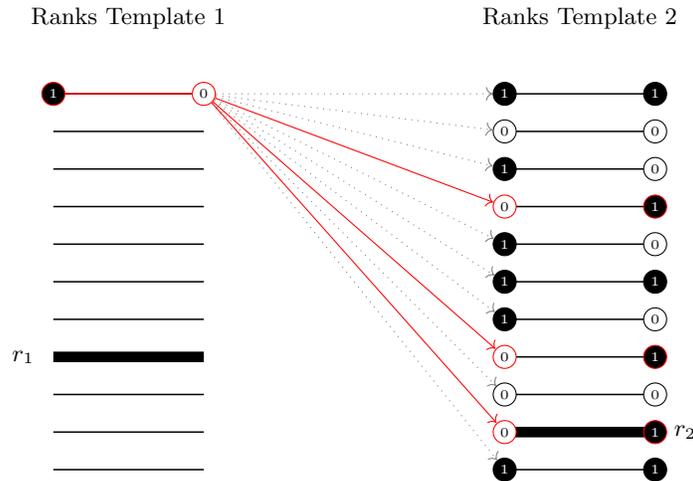


Fig. 2. Illustration of the effect of overlapping templates on the search strategy as expressed formally in Eq. (6). In this example the two ranking lists of templates overlap by 2 bits, such as is the case for Ring A, and thus roughly only every 4th rank on the right-hand side would have matching overlapping bits.

template, it will trigger the four \oplus links to its neighbours 7, 14, 28, and 35 to also flip. Naturally, this results in a much larger electrical change than when those four \oplus links had all been scattered over differently defined templates — like when creating 6-bit templates based on the round keys at the input of the eight S-boxes. Therefore, choosing C-ring-type templates results in different template values being much more easily distinguished from each other above the noise level, and consequently the pattern / template matching process is much more robust as a result. Clearly, the larger the template is for the C rings, the more “inner” key bits there are compared to the fixed number of key bits at the “surface” of the template, i.e. key bits 0, 7, 42, and 49 in Eq. (4). Based on these observations one expects results to be better for C-type templates compared to A- or B-type templates and, furthermore, to improve when increasing the size of the C-type templates. Only when the C-type templates are chosen so large in size — and hence can assume so many different values — that there are not enough traces during Profiling Phase to populate each template value properly anymore, should the C-type templates start to deteriorate again.

2.2 Key Search Strategies on Rings

Since we are dealing with multiple overlapping ranking lists of templates, key-search strategies need to be somewhat more elaborate than normal. It is not sufficient to search in each ranking list individually anymore.

So, let us assume we have a template with f linearly independent bits, resulting in $M = 2^f$ template values, having o bits overlap between two neighbouring

ranking lists of templates.⁴ Then, let us consider two such neighbouring ranking lists of templates, where the correct pattern / template match is found at ranks r_1 and r_2 , respectively.⁵ Then, for each entry in the left ranking list, starting from the top, one needs to search for those ranks in the right ranking list that match in the o common bits, and ignore the rest. An illustration of this process is given in Fig. 2. A pseudo code for this would look like

```

for ( i1 = 1; i1 <= M; i1++ )
{
    for ( i2 = 1; i2 <= M; i2++ )
        if ( Overlap_Matching(ranking_1(i1), ranking_2(i2) )
            Put_on_Search_List(i1,i2)
}

```

Without any statistical bias and further knowledge, this matching would only occur with a probability of roughly $O^{-1} = 2^{-o}$, and hence at first sight on average only every O -th ranking entry in the loop over i_2 will need to be tried in the search strategy, resulting in as little as M/O matches for each value of i_1 . However, this rough analysis is not quite accurate, since the correct pattern / value match is by definition found at position r_2 , and hence the matching probability for this particular ranking entry is definitely 1. Consequently, the average matching probability over all ranks from 1 to r_2 is slightly higher than O^{-1} . More precisely, the match at rank r_2 means that one match out of the M/O possible matches is already consumed, and thus not available anymore when searching in the previous $(r_2 - 1)$ ranks. Likewise, one ranking entry out of the M has also been consumed by the same reasoning. As a result, for those ranks up to but excluding r_2 the matching probability is not O^{-1} , but rather $\tilde{O}^{-1} = (M/O - 1)/(M - 1)$, leading to an average matching probability over all r_2 ranks of

$$P(r_2) = \frac{1 + (r_2 - 1)\frac{M-O}{(M-1)O}}{r_2} . \quad (6)$$

So, what this effectively means is that the original ranking r_2 for the right-side template gets statistically improved to $r_2 P(r_2)$. In the limit of $r_2 = 1$ we have $P(1) = 1$ as it should, whilst for the statistical average ranking value of $r_2 = (M + 1)/2$ we have $P((M + 1)/2) = (M + O)/(O(M + 1))$, leading to $r_2 P(r_2) = (M/O + 1)/2$, which is the statistical average ranking value in the new, reduced search space, again as it should. Finally, for $r_2 = M$ one

⁴ For rings A and B the overlap chosen in this work was $o = 2$, whilst for the C rings the overlap is much larger and depends on the template size used in the end. For templates of the form Eq. (4) with the next template being Eq. (5), the number of overlapping bits is $o = 5$ and hence $O = 2^5 = 32$. This can be worked out by counting the number of \oplus in the overlap region, of which there are 8, and then subtracting the number of closed \oplus loops, which come to 3. On the other hand, the number of linearly independent \oplus bits is $f = 7$ in this case.

⁵ Here and in the following all ranking lists will denote rank 1 as the top rank, so everything is base 1.

finds $P(M) = O^{-1}$, which again makes good sense as when searching the entire ranking list of M entries, M/O matches will be found in total.

It can be argued that Eq. (6) only applies to the case when the starting point on the left ranking list was such that its overlapping bits with the right ranking list match the bits of the *correct* key. If this is not the case, the correction in Eq. (6) due to the correct key being found at r_2 would not necessarily apply. In this case the reduction formula would simply read

$$P'(r_2) = \frac{1}{O} . \quad (7)$$

However, with $r_2(P(r_2) - P'(r_2)) = (M - r_2)(O - 1)/(O(M - 1)) \geq 0$ one finds that Eq. (6) is a conservative approximation underestimating the effect of overlapping, and hence it will be used in the remainder of this paper.

Perhaps the assumption holding the least when deriving Eq. (6) is the assumption that all ranks in the right-hand list that belong to the same “parent” rank in the left list — by way of having the same pattern in the o overlapping bits — will be statistically independent. They are mutually statistically dependent by the very fact that they do have the same “parent”, and this regardless of whether they represent the correct key or not. This will lead to some clustering.

Because of the very existence of overlapping bits between ranking lists and the correlations this induces between ranking lists, it is not necessarily the best possible strategy to search each template ranking list all the way to the bottom of it. Rather, it is preferable to search up until a certain threshold value r across all ranking lists and, if unsuccessful, increase this threshold r step by step. A pseudo code for this could look like

```

for ( r = 1; r <= M; r++ )
{
  /* The right list is at maximal value r */
  for ( i1 = 1; i1 <= r; i1++ )
  {
    if ( Overlap_Matching(ranking_1(i1), ranking_2(r) )
        Put_on_Search_List(i1,r)
  }
  /* The left list is at maximal value r */
  for ( i2 = 1; i2 < r; i2++ )
  {
    if ( Overlap_Matching(ranking_1(r), ranking_2(i2) )
        Put_on_Search_List(r,i2)
  }
}

```

If these are the only two ranking lists of templates to consider, as is the case for A rings, for instance, then this search strategy will have found the correct match once $r_{\max} = \max(r_1, r_2)$ has been reached. In doing so roughly $r_{\max}^2 P(r_{\max})$

calls to `Put_on_Search_List()` are required. For the A rings we have $P(r_{\max}) \approx O^{-1} = 1/4$.⁶

When more than two ranking lists of templates are connected in a ring, like in the C rings, the analysis becomes a bit more involved, but in essence stays the same — the $\max(\cdot)$ operation will then have to run over all ranking lists of templates along the ring.

Finally, it should be noted that although each template above may have f linearly independent bits, when closing a C ring, one more degree of freedom will be consumed by the boundary condition of the total sum of \oplus around the ring needing to be even, necessarily. With this it turns out that for each C ring a total of 27 linearly independent bits exist and need to be determined. The 28th bit is taking account of the fact that the \oplus defines the original key bits only up to an overall inversion of all bits.

Clearly, if there is statistical bias in the bits of the patterns in the ranks $1 \dots r_{\max}$, then the approximation of Eq. (6) will not be quite accurate anymore. Whilst this effect is likely not important when $r_{\max} \approx M$, it is expected that Eq. (6) will provide a too small estimate for the effective ranking when $r_{\max} \ll M$.

Also, it should be noted that with this strategy the correct pattern / template matching will not be found before rank $r_{\max} \equiv \max(r_i)$ has been reached, where i runs over all ranking lists of template positions along the ring. Consequently, this strategy will yield poor results if just one of those lists screws up in its ranking r_i . However, due to the overlapping bits between neighbouring template ranking lists, there is also strong correlation between their r_i , and hence outliers are much less likely than expected from statistically independent ranking lists.

Having such an effective maximal rank r_{\max} one can calculate the estimated remaining rest entropy by calculating the key space that needs to be ploughed through with brute force, based on the template ranking lists. For the A rings one finds

$$E_{r_{\max}}^A = 4 + 8 \log_2(r_{\max}) , \quad (8)$$

where the additional term 4 is to account for the fact that under the \oplus operation the 4 A rings are only determined up to an overall sign for each ring, and the factor 8 stems from the total number of overlapping template lists used for A rings. Each template list accounts for a factor r_{\max} when counting the number of possible combinations of template ranks across all 8 ranking lists — hence $8 \log_2(r_{\max})$. Likewise, one finds for the B rings, which are only two rings but still eight template ranking lists,

$$E_{r_{\max}}^B = 2 + 8 \log_2(r_{\max}) . \quad (9)$$

For the C rings, where the construction of templates has been for maximal overlap, resulting in 28 template lists split over two rings, one finds

$$E_{r_{\max}}^C = 2 + 28 \log_2(r_{\max}) , \quad (10)$$

⁶ The factor 1/4 is important as it ensures that even in the worst-case limit of $r_{\max} = M$ the number of calls to `Put_on_Search_List()` does not exceed the number of templates possible for a 14-bit vector, i.e., 2^{14} .

or

$$E_{r_{\max}}^{C'} = 1 + 14 \log_2(r_{\max}) \quad (11)$$

for just one of these rings. There may well be more optimal search strategies than those described above, which would lead to smaller estimates of the remaining entropy, and thus the above can be considered as approximations only.

As just discussed, because of the overlap and hence correlation between neighbouring templates, there is a tendency for all rankings r_i to be of roughly the same value and hence, in order to get a quick glance, it is useful to make a rather bold worst-case approximation and work with the arithmetic average⁷

$$r = \frac{1}{R} \sum_{i=1}^R r_i \quad (12)$$

instead of r_{\max} , where $R = 8$ for A and B rings, and $R = 28$ for C rings.⁸ This is a crucial assumption, which will generally somewhat overestimate the number of broken key bits. In the following we will refer to this assumption as the worst-case assumption. Assumptions of this kind are often used in Common Criteria evaluations to be on the safe side.⁹

Eqs. (6) and (10) provide a first estimate about the remaining rest entropy when doing a brute-force attack, but it is interesting to see how efficient a key-search strategy can in fact be implemented. First we notice that the problem decomposes into two parts, the search over the 28 key bits related to the C-Register, and the search over the 28 key bits stemming from the D-Register, since both registers effectively set up independent key schedules [13]. This makes the problem much more manageable.

A simple key search is done in a few characteristic steps. But before going there, let us briefly outline the data structure we implemented. For each of the 28 template ranking lists, and each entry in all these lists, we created lists containing the 4 pointers to the matching templates and their rankings on the

⁷ It is normally not allowed to reverse the order of steps taken when calculating the rest entropy and performing an averaging, as we do here by first averaging over r_i and then calculating the rest entropy. However, the overlap and hence correlation between neighbouring templates justify this as a worst-case scenario. A more detailed and accurate analysis of the resulting ranking distribution is performed later in this Section.

⁸ There is actually no reason for r to be the same in different rings, since there is no correlation between templates across different rings. It is straightforward, though, to generalise Eqs. (8–10) below to such a scenario. For instance, Eqs. (10) would then read as $E_{r_{\max}, r'_{\max}}^C = 2 + 14(\log_2(r_{\max}) + (\log_2(r'_{\max})))$.

⁹ When using Eqs. (8–10) with a statistically average value for $r = \bar{r}$, one finds $E_{\bar{r}=64.5}^A = 52.09$, $E_{\bar{r}=64.5}^B = 50.09$, and $E_{\bar{r}=2.5}^C = 39.01$. At first sight this appears to be in contradiction to the expectation that without any leakage, one would expect $E \approx 55$ for DES. However, even if there is no statistical leakage for an individual template position, if we assume all ranking lists across all template positions to be the same, then we *are* still assuming perfect correlation between all ranking lists, and hence *not* a statistically uncorrelated distribution.

“right-hand” side of each ranking list. The matching is here simply defined by the overlap properties of neighbouring template values. With these lists it is then easily possible to set up graphs across all 28 nodes that present valid combinations of nodes.¹⁰ Since we only need to tackle C-type rings / graphs containing 14 elements at the time, this is neither a challenge memory-wise (of the order of 15GB are required for the entire list of 2×2^{27} graphs) nor is it from a computational point of view.

In a first step we create the list of all valid graphs — there will be 2^{27} valid graphs in total for the C-Register, and another 2^{27} graphs for the D-Register. These graphs need to have the property of being closed along the C ring and, secondly, also need to have an even number of \oplus along the ring on the top line as well as the bottom line (i.e., the two A rings that are contained in each C ring).

The second step is to sort this list of graphs according to some sorting principle — this is the key enumeration step. We have implemented two different key enumeration metrics so far. Firstly, we take the average ranking across all 14 nodes of a graph and, secondly, we choose the maximal ranking r_{\max} found across all 14 nodes. In both cases we sort for increasing values. When performing the corresponding sorting of the original list it is important to realise that there is only a limited number of possible sorting values that can be had, and there is no point in sorting graphs that are “degenerate” by having the same average or maximal ranking. This feature helps in accelerating the sorting tremendously.

The final step is to actually perform the search through this sorted list, taking care to account for the variability introduced by the degeneracy mentioned above — leading to a best-case rest entropy, a worst-case rest entropy and, finally, an average rest entropy. Since it turns out that best-case and worst-case rest entropy are very close to each other, in what follows we will only report the average rest entropy.

The first two steps — so the creation of the unordered list of graphs and the subsequent key enumeration — execute in a time roughly independent of where the key is actually ranked in the list. They only depend on the template size. Only the last, third step is highly dependent on the key ranking, but it is a normal brute-force search just like for a normal DES, with almost no additional overhead. Of course, the actual brute-force attack will have to deal with the C- and D-Registers simultaneously, but this is only required for this last, third step.

A pseudo code for this key-search strategy could look like

```
/* Read in 14 input files containing the rankings of 14 template lists */
for ( p = 0; p < 14; p++ )
{
    for ( r = 1; r <= M; r++ )
    {
```

¹⁰ A node is defined by a template position along the C ring and the value that this template has.

```

        /* For each r there will be 4 matches returned */
        Work_Out_List_of_Matching_Overlaps_with_Right_Hand_Template(p)
    }
}

/* Create an unordered list of nodes connected as graphs that */
/* represent valid combinations of neighbouring templates */
Set_Time()
Pick_first_Template_Ranking_List(0)

for ( r = 1; r <= M; r++ )
{
    For_All_Paths:
    {
        Follow_Path_Via_Next_Neighbour_Matching_along_Ring()

        if ( Path_Closes_Along_Ring && Number_Xors_in_A_and_B_Ring_is_Even )
            Add_Path_2_List_of_Graphs()
    }
}
printf(Elapsed_Time T_1)

/* Key enumeration step: */
/* Order the list of graphs according to minimal average, */
/* or minimal max_rank_r. Note that this ordering is fast, */
/* since no ordering is needed between graphs having the */
/* same minimal average / minimal max_rank_r */
Set_Time()
Order_List_of_Graphs()
printf(Elapsed_Time T_2)

/* Perform the actual key search on the C or D Register */
Search_Ordered_List_for_Matching_Key()

```

The pseudo code as given here is valid for searching for the subkey belonging to either the C-Register or the D-Register, but not for both at the same time. However, this is trivially extended to in the final section of the pseudo code.

These are basic search algorithms and it is expected that one can improve on those. To start with we notice that according to Table 5 the \oplus between different key bits do not leak all with the same weight. Some leak only once, whilst others leak up to 12 times.¹¹ It is thus not unreasonable to assume that those \oplus relations that leak more will also yield more stable results in the template matching, and hence require a different treatment in the key search than those

¹¹ To be more precise, the links on the top and bottom lines of the C rings have large weights, i.e., the links stemming from the A rings, whilst all links related to the B rings have much less weight.

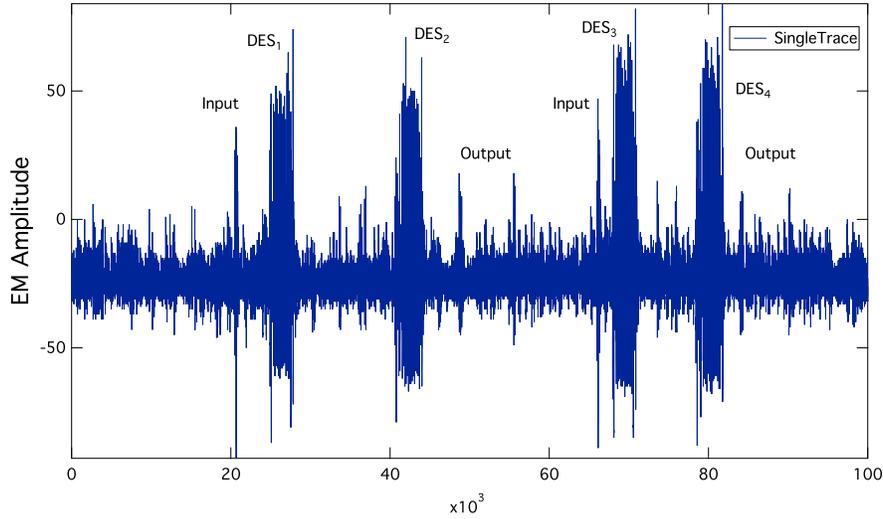


Fig. 3. A typical single EM (electromagnetic) trace of the TOE showing 4 calls to the DES HW engine. It was obtained by placing a Langer EM probe on top of what had previously been identified as a DES coprocessor hard macro in the chip layout. Sampling rate: 5 GS/s.

bits that leak less. Augmenting the key-search strategy along these lines should give substantial improvements.

Secondly, if we assume that the total Hamming weight of the correct key leaks perfectly, the key-search strategy could be modified in a way that for each 28 key bits tried for the C-Register, knowing the Hamming weight of the total key, we can work out the remaining Hamming weight of the 28 key bits stemming from the D-Register (or vice versa), and this will cut down on the number of possible key candidates in that second search. Actually, this approach yields more bits than one might first think, since the number of possible Hamming weights of the full key is not 57, but rather much more, 575, since each key bit contributes to the total effective Hamming weight with a weight according to its occurrence as given in Table 5. Trials have shown that the remaining rest entropy can be reduced by as much as 5 to 7 bits this way.¹²

In a similar fashion, if there are any other leakages present — for instance like those found in [12] for the same device — they could be factored into this key-search strategy as well.

¹² In order to attack the total Hamming weight of the key, for each trace one has to fold back all $15 \oplus$ rounds into a single round by summing them up. This is to make sure the leakage occurs always at the same sample point in the trace.

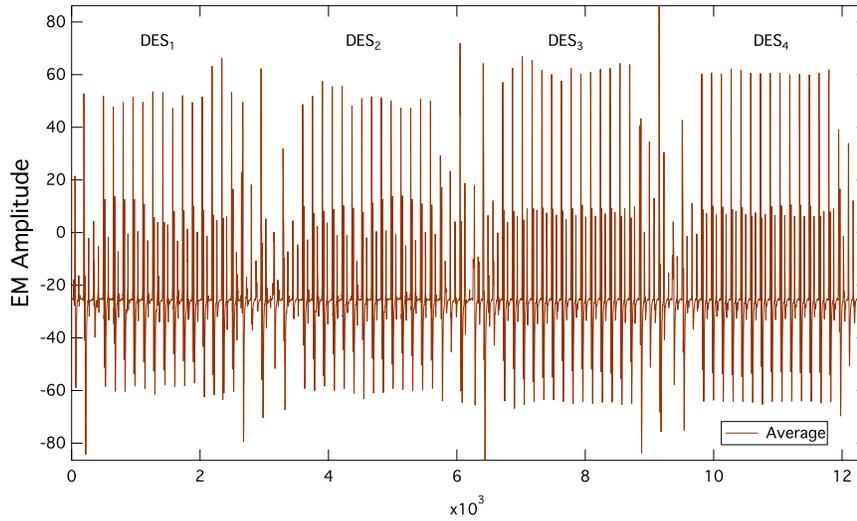


Fig. 4. Average trace with all four DES blocks aligned.

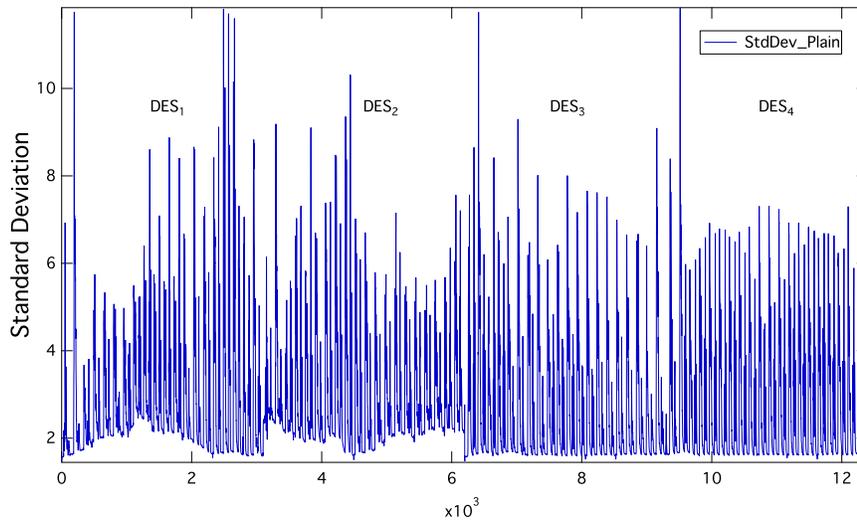


Fig. 5. Standard deviation with all four DES blocks aligned. The first DES block is the most difficult to align and far from perfect yet.

3 Results for a Contemporary Smart Card

Target of Evaluation (TOE) was a smart card from 2012 containing a JAVA OS where we could freely call the DES function — although likely via a wrapper in an underlying crypto library. A typical single EM trace is given in Fig. 3, showing essentially 4 calls to the DES hardware engine. First we remove the strong timing jitter between these blocks using a simple rigid-pattern filter that locks

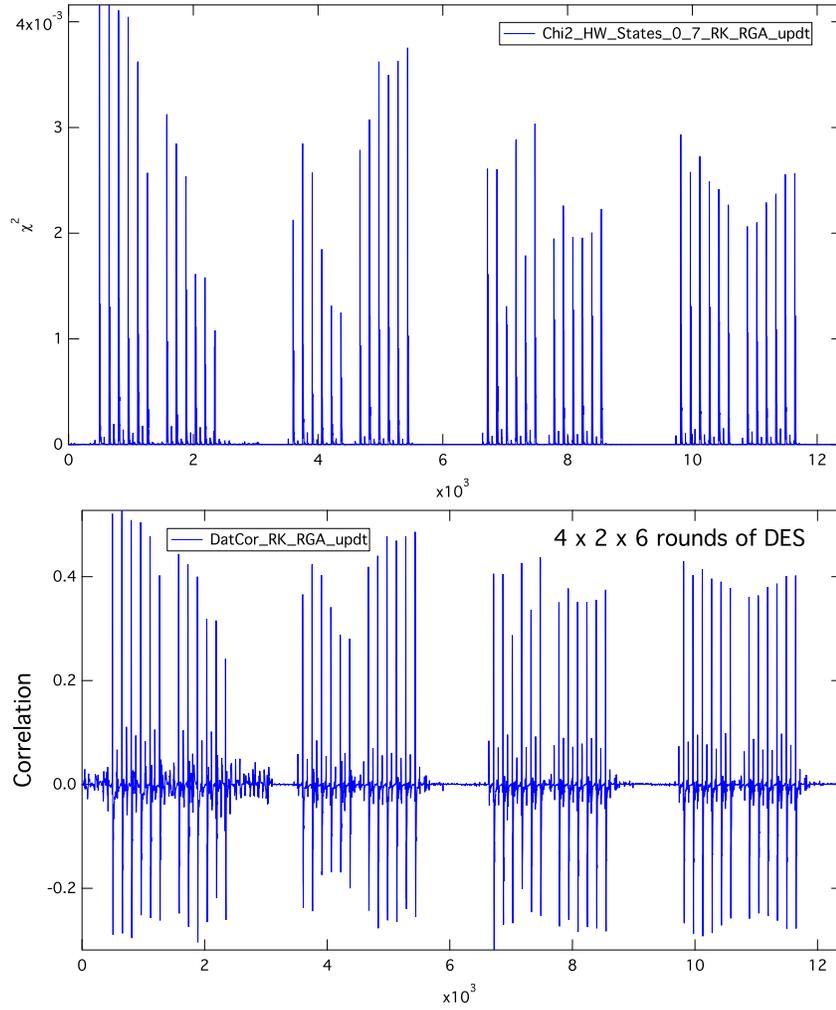


Fig. 6. χ^2 and correlation of the \oplus round key updating along the A rings: Eq. (1).

into these four DES blocks. In a next step we refine the alignment by applying an elastic alignment filter based on segment-wise parametrizing the internal clock / time as $T = a + bt + ct^2 + dt^3 + et^4 + ft^5$, where t corresponds to the external clock / time, and then finding the best set of coefficients (a, b, c, d, e, f) for each DES segment of each trace.¹³ Starting with a little over 7M raw traces with random plain text and random key, this procedure yielded just over 5M aligned

¹³ The raw traces show two different types of timing jitter: Firstly, there are random delays inserted between the four DES blocks. These are removed by taking one trace as a reference trace and then for each trace perform a least-square search for each of the 4 DES blocks separately to get a rough alignment of those 4 DES blocks. Secondly, the internal clock does not appear to be very stable. In the most simple

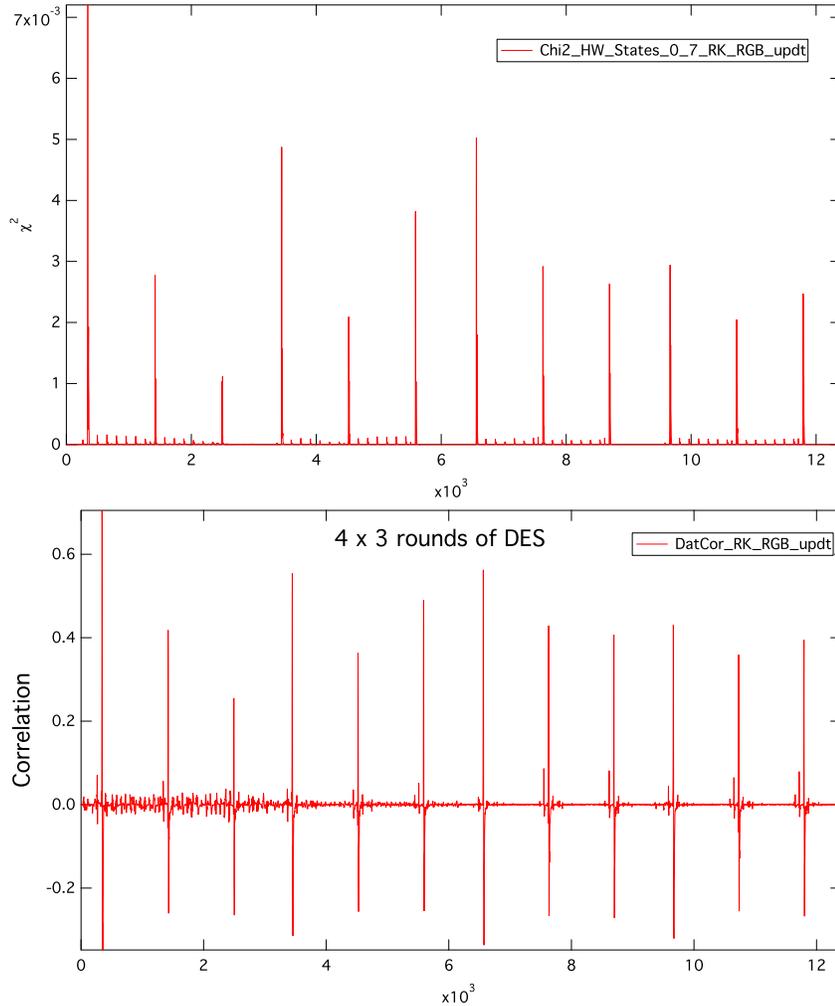


Fig. 7. χ^2 and correlation of the \oplus round key updating along the B rings: Eq. (2).

traces, with their average trace given in Fig. 4, and its standard deviation shown in Fig. 5. Likely, the multiple calls to the DES hardware engine are due to countermeasures against fault attacks. We do not need to know whether these DES calls are “forward” or “backward” calculations — it does not matter, anyway,

approximation we assume the internal clock frequency to vary smoothly over time and model this with a polynomial fit. Since polynomial fits only work well over rather short time intervals, it then becomes necessary to split each DES block into a couple of time segments and apply the polynomial fit to each of them separately, using continuity equations as needed. Again, for each trace and each time segment, the best set of coefficients is determined using a least-square algorithm.

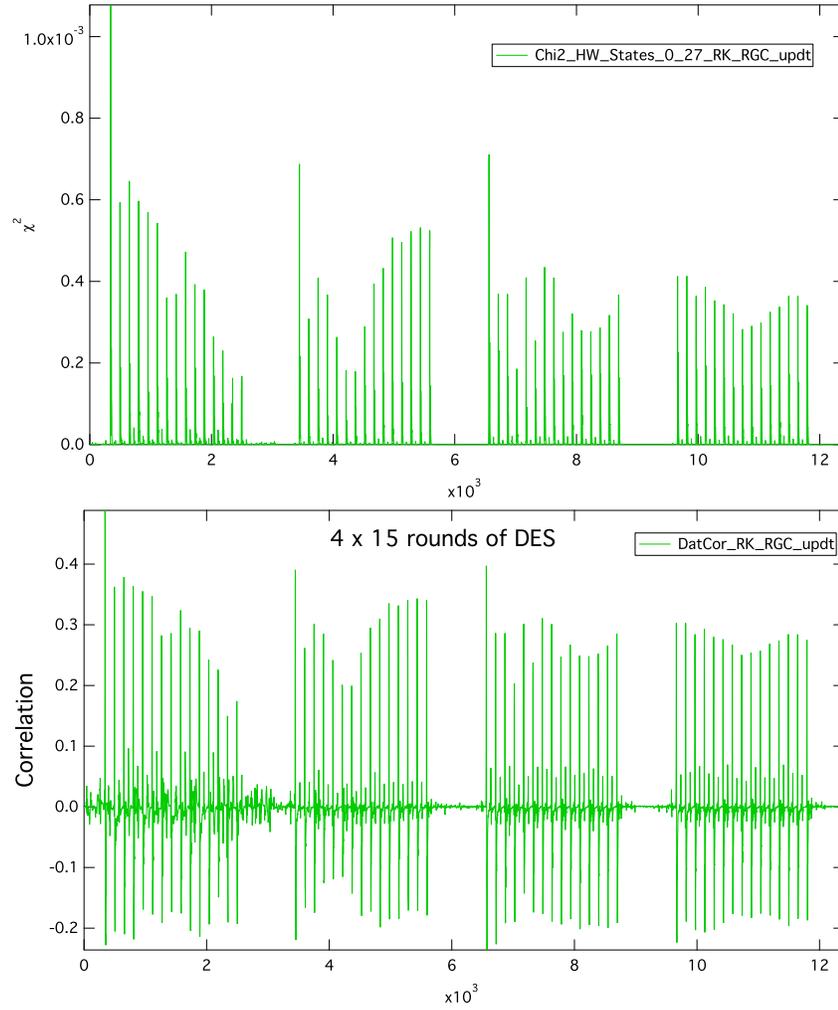


Fig. 8. χ^2 and correlation of the \oplus round key updating along the C rings: Eq. (3).

for launching a template attack, as long as they are static and do not change from one call to another. However, subsequent analysis reveals that these are forward – backward – forward – backward DES calls. Interestingly, they do not have precisely the same run-time behaviour, and also their standard deviations are slightly different as per Fig. 5. The last two DES blocks are much easier to align. Other than this alignment, no further preprocessing has been performed for calculating the templates later on.

3.1 Results based on Exploiting the Ring Structure

A number of standard DPA attacks were initially tried out on this TOE, such as targeting the input and output of the S-Boxes, after the \oplus at the output, as well as the Hamming distance model, and some leakage was indeed seen here and there, but none of those first-order approaches seemed to work very well with a million traces, and thus we conclude that the DES hardware engine has been reasonably hardened against these attacks. Also, targeting the round key itself did not yield any useful correlation signal, although some strong spikes are visible in a χ^2 analysis, indicating that Points of Interest for a template attack are at least available. However, the \oplus between two successive round keys seemed most promising, showing strong correlation signals.¹⁴ Figure 6 displays the χ^2 and correlation function for the select function based on Eq. (1), i.e., for A rings. Since we do not know the countermeasures implemented in this DES hardware engine, we can only speculate, but it seems that some blinding of the rounds has been implemented, which does its job when looking at each round in isolation, but which perhaps does not get updated from one round to the next and thus gets removed when performing an \oplus operation between two rounds and thereby attacking the Hamming distance.

Likewise, we have plotted the χ^2 and correlation functions for rings B and C in Figs. 7 and 8 respectively. Overall, we find good qualitative agreement with the simulated leakage shown in Fig. 1.

Clearly, this weakness should be exploitable and, given the strength of the leakage observed, in the remaining part of this paper we will aim to exploit it using a *single trace* in the Exploitation Phase only.

Statistical Analysis In a first step, the Profiling Phase, we create 28 sets of $(2n + 1)$ -bit templates for the C rings as described in Sec. 2 using 4.75 million traces with random keys as input.¹⁵ Then, in a second step, the Exploitation Phase, we test how good these templates are by applying them to a further set of a few ten thousand random-key traces (32K traces to be precise), treating each of them as a single-trace attack during Exploitation Phase, and extracting some statistical properties of this 32K ensemble of single-trace attacks.

Firstly, we analyse the auto-correlation of the rankings of neighbouring templates along the ring. Because of them overlapping, one can expect a strong auto-correlation that will decay as the distance between the templates along the ring increases. To be more precise, a C-type template of size $(2n + 1)$ can be shifted by n steps before it loses overlap with the template at its original position. This

¹⁴ In fact, we checked whether there are any other \oplus leakages visible in the correlations and χ^2 between *any* two DES key bits other than those predicted by the A, B, or C-rings and listed in Table 5, but none were found other than those also seen in [12], suggesting that the chosen leakage model is indeed very suitable for the analysis of this smart card.

¹⁵ For each substantial positive and negative peak in the correlation function ≈ 9 POIs were taken, leading to some 1035 POIs in total for the C rings.

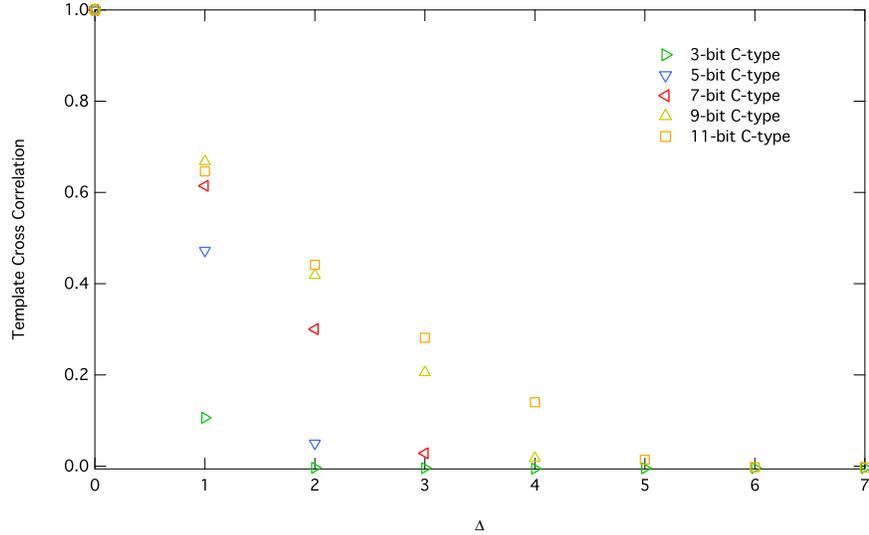


Fig. 9. Correlation $C(\Delta)$ of neighbouring templates as a function of their mutual distance Δ along the ring, as defined in Eq. (13). The correlation decays at the scale of the size of the C-type template. So, for $(2n + 1)$ -bit templates, which overlap with their nearest neighbour by $2n$ bits, the correlation spans up to $\Delta_{\max} \approx n$. (C-type templates of the type Eq. (4) shift in units of 2 because of the two rails that they have.)

hypothesis is confirmed in Fig. 9.¹⁶ Here we have plotted the auto-correlation between the rankings of neighbouring C-type templates of varying sizes as a function of their mutual distance Δ along the ring — and this averaged over all template positions i and $14 + i$ along the two rings as

$$C(\Delta) = \frac{1}{28} \sum_{i=0 \dots 13} (\text{Corr}(r_{(i+\Delta) \bmod 14}, r_i) + \text{Corr}(r_{14+(i+\Delta) \bmod 14}, r_{14+i})) . \quad (13)$$

The auto-correlation $C(\Delta)$ as shown in Fig. 9 decays almost linearly with the mutual distance Δ , like $C(\Delta) \approx 1 - \Delta/n$ for not too large template sizes, and it is slightly larger than what one would expect given the size $2n + 1$ of the template. This is attributed to higher-order effects. It is currently not clear why the 11-bit templates deteriorate for $\Delta = 1$ compared to the 9-bit templates. Perhaps, it has to do with an insufficient number of traces used for generating the templates.

In Figs. 10 to 13 we have plotted the distribution of the maximal rankings, $r_{\max} = \max(r_i)_{i=0 \dots 27}$, for this random ensemble of 32K traces with randomly chosen keys — based on 5-bit, 7-bit, 9-bit, and 11-bit templates, respectively. For reference, we have also plotted the expected scaled distribution of r_{\max} if all

¹⁶ Here and in the following we will always use — as defined in [12] — the `LnP_AvC` approximation to the probability density function of the multivariate normal distribution.

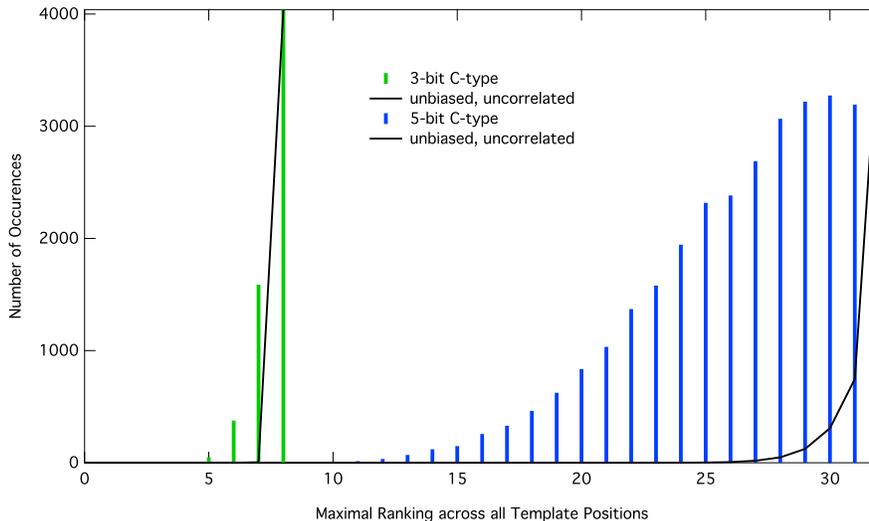


Fig. 10. Histogram of the maximal ranking $r_{\max} = \max(r_i)_{i=0..27}$ along the C rings for 3-bit and 5-bit templates. In a given random-key ensemble of 32K traces for the Exploitation Phase, the smallest maximal rankings found at the left-hand onset of the distribution are $r_{\max} = 4$ and $r_{\max} = 7$, respectively.

rankings were uncorrelated and unbiased. This distribution can be easily derived by working out how many ranking results $\{r_{i=0..27}\}$ are possible when each ranking r_i needs to be within the range $1..r_{\max}$, leading to $P_{\text{uncorrelated}} \propto r_{\max}^{27}$. As can be seen, the centre of the distribution moves to lower ranks (relative to the maximal rank possible) as the template size increases, which suggests that further improvements to this attack should be possible by collecting even more traces during the Profiling Phase and then creating even larger templates such as 13-bit and 15-bit templates — both of which are still within computational range.

Next we study the average ranking values for each of the 28 template positions along the C ring, when averaged over this ensemble of 32K random traces. Figure 15, top graph, shows the average rankings \bar{r}_i found for the 28 possible C-type 9-bit template positions — again for the case of LnP_AvC as defined in [12].¹⁷ Clearly, these rankings fall into two classes: Template positions 0 to 13 which correspond to the C-Register, and positions 14 to 27, which map to the D-Register. The fact that the D-Register leaks more can be used to improve the key-search strategy. This effect seems to be getting stronger when going for larger template sizes as is apparent when comparing Figs. 14, 15 and 16.

¹⁷ Note, the approximations LnP_AvC, LnP_nAvC, and LnP_wAvC defined in [12] work equally well here, with LnP_gAvC, LnP_lAvC, LnP_lgAvC, and LnP_lwAvC being a tad worse. The same observation had been made with an older data set already, and hence there seems to be some consistency in this regard.

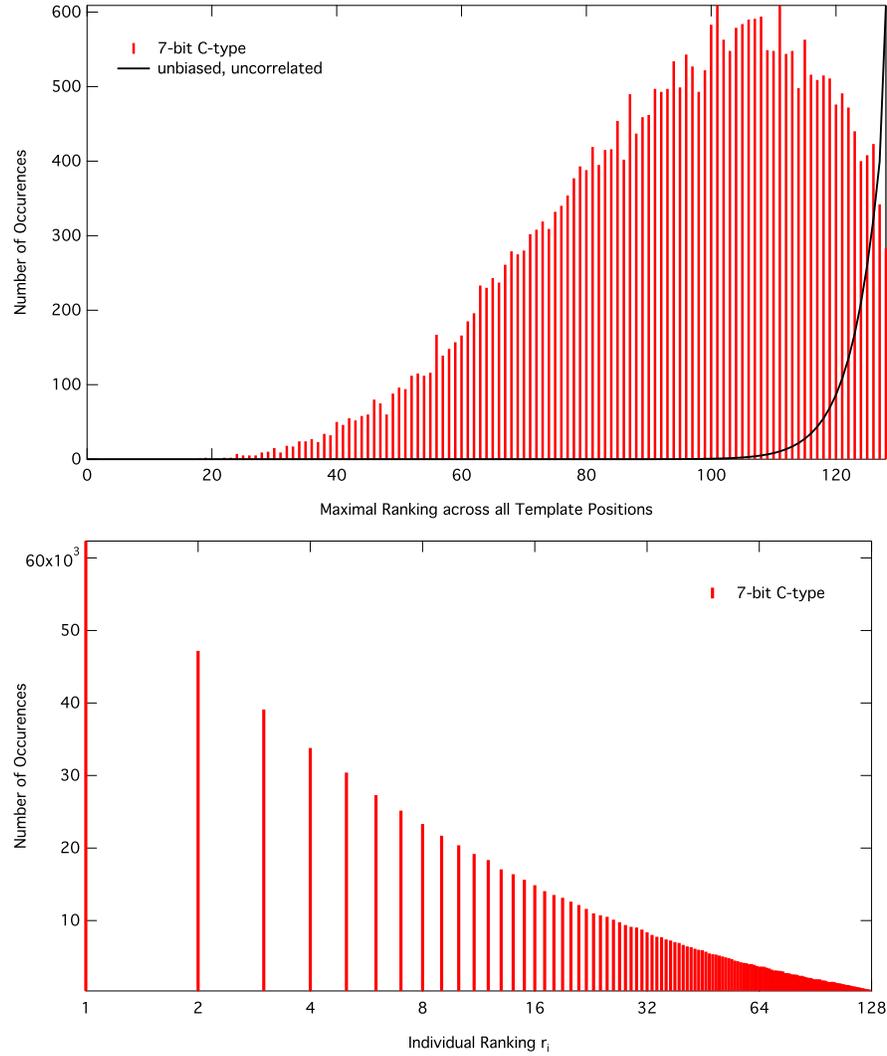


Fig. 11. Histogram of the individual ranking r_i (bottom) as well as the maximal ranking $r_{\max} = \max(r_i)_{i=0..27}$ (top) along the C rings for a random-key ensemble of 32K traces, based on 7-bit templates. The smallest maximal ranking at the left-hand onset of the distribution in the top graph is $r_{\max} = 18$. For reference, the expected (scaled) distribution based on uncorrelated, unbiased rankings at 28 template positions is also shown: $P_{\text{uncorrelated}} \propto r_{\max}^{27}$.

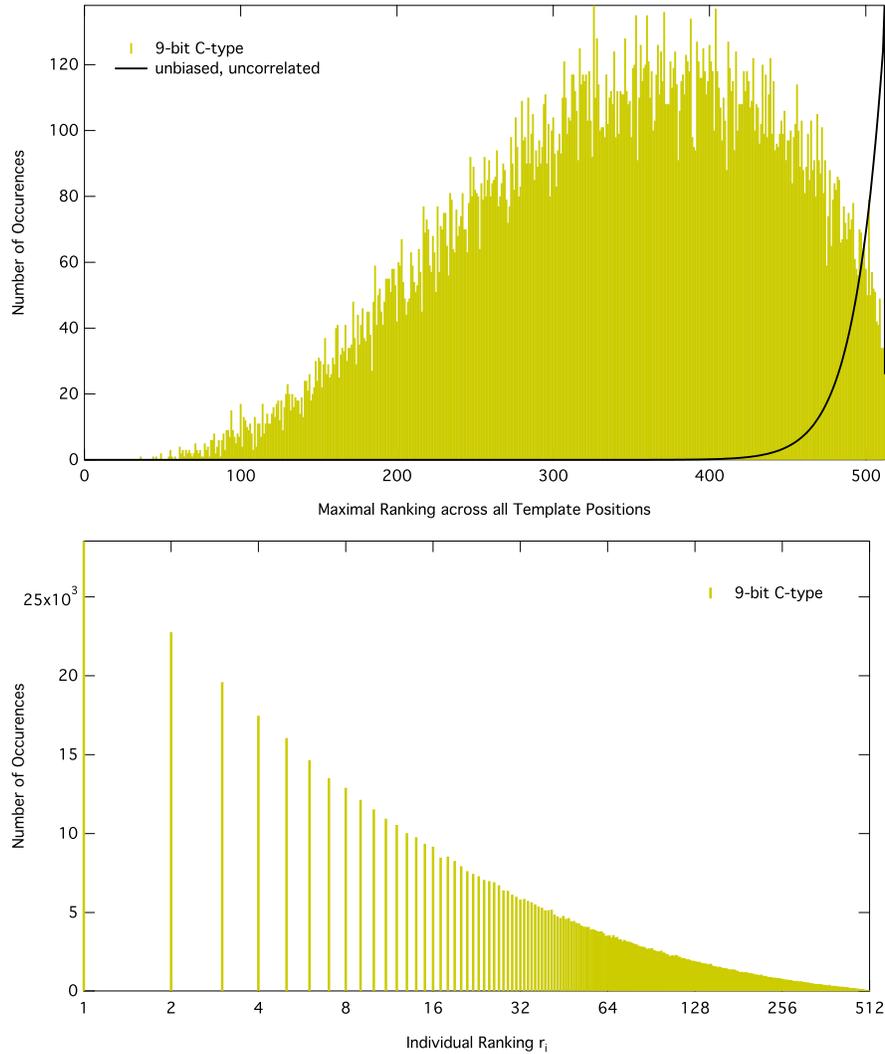


Fig. 12. Same as Fig. 11, but for 11-bit templates and a random-key ensemble of 32K traces. The smallest maximal ranking at the left-hand onset of the distribution in the top graph is $r_{\max} = 36$. The corresponding actual rankings r_i of the associated single trace are shown in Fig. 15, bottom part. For reference, the expected (scaled) distribution based on uncorrelated, unbiased rankings at 28 template positions is also shown: $P_{\text{uncorrelated}} \propto r_{\max}^{27}$.

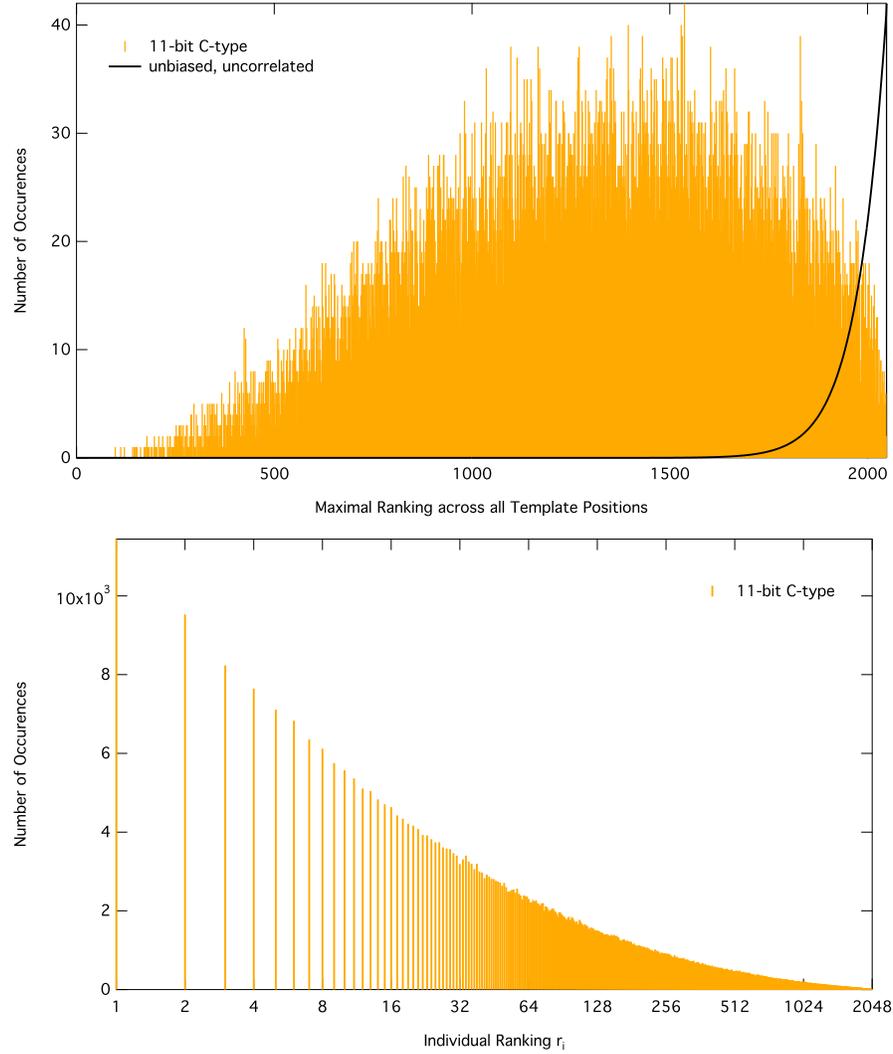


Fig. 13. Same as Fig. 11, but for 11-bit templates and a random-key ensemble of 32K traces. The smallest maximal ranking at the left-hand onset of the distribution in the top graph is $r_{\max} = 98$. The corresponding actual rankings r_i of the associated single trace are shown in Fig. 16, bottom part. For reference, the expected (scaled) distribution based on uncorrelated, unbiased rankings at 28 template positions is also shown: $P_{\text{uncorrelated}} \propto r_{\max}^{27}$.

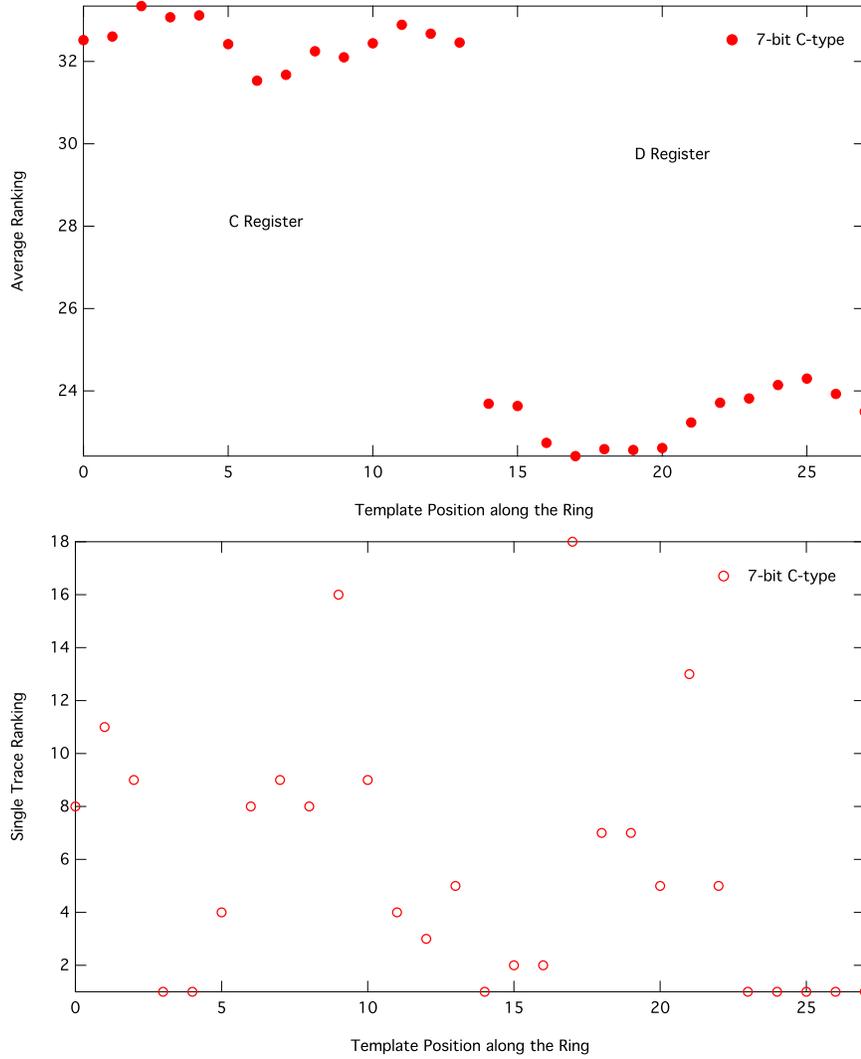


Fig. 14. Top: Average (original) rankings \bar{r}_i as a function of the template position i in the C rings, based on a random-key ensemble of 32K traces. “Original” ranking means that Eq. (6) has not yet been applied, and thus a 7-bit template results in 128 different possible ranks. The average rankings for templates associated with the so-called C register in the DES are found to be rather higher than those associated with the D register. In both cases the rankings \bar{r}_i vary rather smoothly as a function of the position index i , which is a tell-tale of the correlation between them. Bottom Figure: Same, but for a *single* trace. Here the maximal ranking along the C rings is $r_{\max} = 18$, found at positions $i = 17$, which with Eqs. (6) and (10) translates to an estimated remaining rest entropy of 15.64 bits.

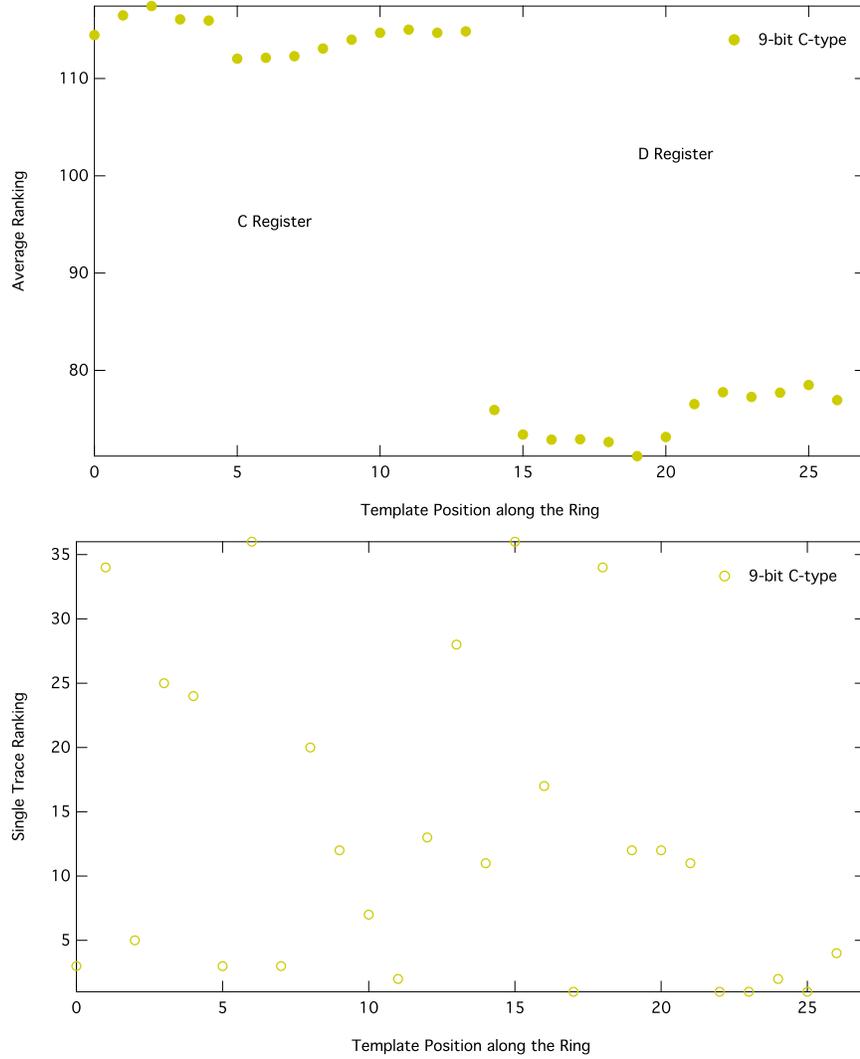


Fig. 15. Top: Average (original) rankings \bar{r}_i as a function of the template position i in the C rings, based on a random-key ensemble of 32K traces. “Original” ranking means that Eq. (6) has not yet been applied, and thus a 9-bit template results in 512 different possible ranks. The average rankings for templates associated with the so-called C register in the DES are found to be rather higher than those associated with the D register. In both cases the rankings \bar{r}_i vary rather smoothly as a function of the position index i , which is a tell-tale of the correlation between them. Bottom Figure: Same, but for a *single* trace. Here the maximal ranking along the C rings is $r_{\max} = 36$, found at positions $i = 6$ and $i = 15$, which with Eqs. (6) and (10) translates to an estimated remaining rest entropy of 9.55 bits.

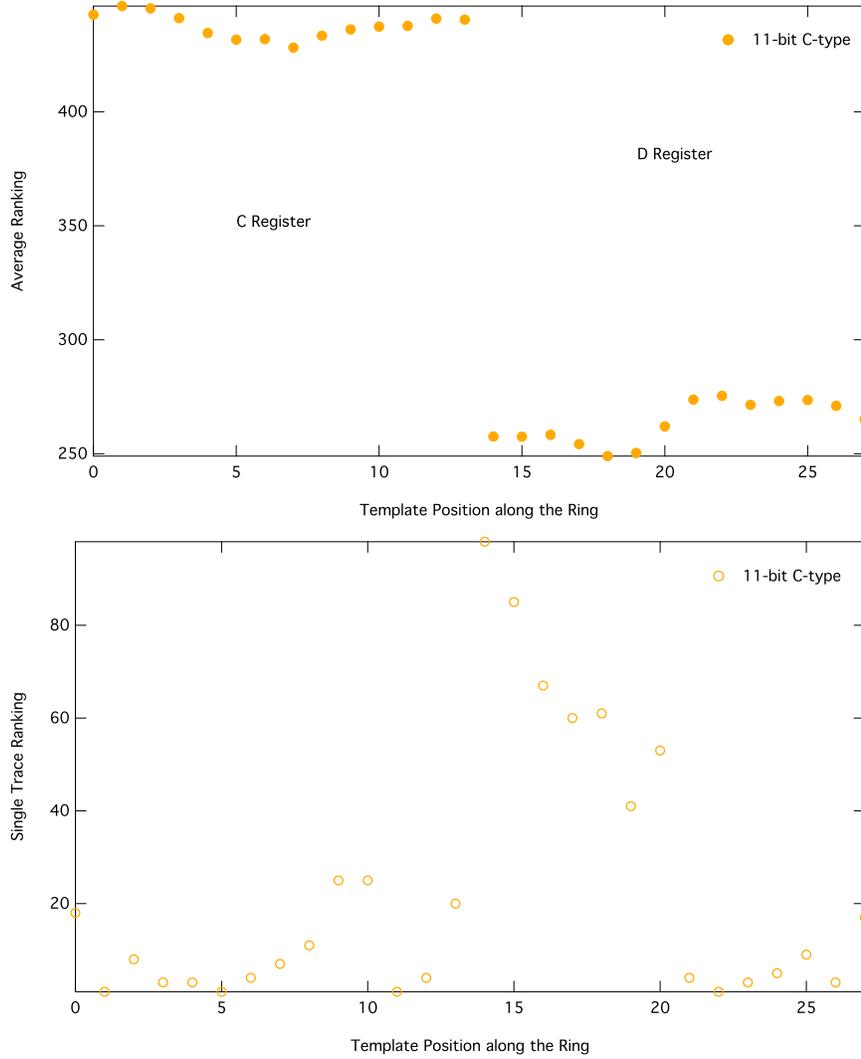


Fig. 16. Top: Average (original) rankings \bar{r}_i as a function of the template position i in the C rings, based on a random ensemble of 32K traces. “Original” ranking means that Eq. (6) has not yet been applied, and thus a 11-bit template results in 2048 different possible ranks. The average rankings for templates associated with the so-called C register in the DES are found to be rather higher than those associated with the D register. In both cases the rankings \bar{r}_i vary rather smoothly as a function of the position index i , which is a tell-tale of the correlation between them. Bottom Figure: Same, but for a *single* trace. Here the maximal ranking along the C rings is $r_{\max} = 98$, found at position $i = 14$, which with Eqs. (6) and (10) translates to an estimated remaining rest entropy of 7.37 bits.

Table 6. Characteristic statistical data of the r_{\max} ranking distributions shown in Figs. 10 – 13 based on the analysis of an ensemble of 32K random-key traces in Exploitation Phase: The onset ranking and estimated rest entropy found for a particularly leaky individual trace (a characteristics which clearly is highly dependent on the size of the ensemble of traces analysed), the ranking $r_{\max}(Q)$ reached at various first quantiles Q , and finally the estimated average rest entropy \bar{E}_Q calculated within those respective quantiles, based on Eqs. (6) and (10) and a simple search strategy based on $r_{\max} = \max(r_i)|_{i=0\dots27}$. For comparison, we also show the “worst case” results based on first averaging the template rankings over all 28 template positions along the C rings as well as over the ensemble of all traces with $\bar{r} = 1/28 \sum_i \bar{r}_i$, and then calculating the estimated rest entropy $E(\bar{r})$.

Templates	3-bit (32K)		5-bit (32K)		7-bit (32K)		9-bit (32K)		11-bit (32K)	
	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}
Onset	3	20.01	7	20.49	18	15.64	36	9.55	98	7.37
Permillie	3.05	29.56	10.06	27.19	25.70	19.59	69.00	13.94	218.00	11.38
Percentile	4.03	36.05	13.42	31.96	39.94	25.48	112.86	19.30	386.00	16.86
64th Quantile	4.12	38.31	14.36	33.33	43.41	26.92	125.94	20.79	433.00	18.37
32th Quantile	4.37	40.32	16.01	35.51	50.07	29.43	150.63	23.44	530.67	21.03
Hexacile	4.88	41.33	17.85	38.05	58.03	32.29	180.07	26.45	641.50	24.10
Octile	5.21	44.44	20.02	40.75	67.34	35.48	218.21	29.89	788.00	27.57
Quartile	5.67	46.35	22.53	43.72	79.59	39.04	271.16	33.96	999.03	31.74
Median	6.25	48.98	25.87	46.97	96.81	43.24	348.46	38.93	1324.30	36.99
Average	6.58	51.20	25.60	50.90	94.53	48.56	341.10	45.53	1300.05	44.14
	\bar{r}	$E(\bar{r})$	\bar{r}	$E(\bar{r})$	\bar{r}	$E(\bar{r})$	\bar{r}	$E(\bar{r})$	\bar{r}	$E(\bar{r})$
“Worst Case”	3.12	28.14	8.80	24.72	27.93	21.89	94.86	19.73	350.71	18.71

Next we look at the extreme cases to the left in the histograms of Figs. 10 to 13 — single traces that are particularly leaky. Figure 15, bottom graph, shows the rankings of all 28 template positions, with an r_{\max} as low as $r_{\max} = \max(r_i)|_{i=0\dots27} = 36$, occurring at positions $i = 6$ and $i = 15$. With Eqs. (6) and (10) this maximal ranking then translates to an estimated remaining rest entropy of 9.55 bits.

Likewise, in Fig. 16, bottom graph, we show the results for the most leaky trace found in the ensemble of 32K traces when analysing with 11-bit templates. Here $r_{\max} = \max(r_i)|_{i=0\dots27} = 98$ at $i = 14$, so in the D-Register this time, leading to an estimated remaining rest entropy of 7.37 bits.

Finally, in Table 6 we provide an overview over the most important characteristic statistical properties of the r_{\max} ranking distributions for $(2n + 1)$ -bit templates. For various quantiles Q the corresponding value of $r_{\max}(Q)$ at the quantile boundary is provided, as well as the estimated rest entropy $\bar{E}(Q)$ averaged *over* that quantile (as opposed to the estimated rest entropy $E(Q)$ at the quantile boundary Q , which can be simply derived by plugging $r_{\max}(Q)$ into Eqs. (6) and (10)). Generally, as expected, it is observed that results improve when going for larger template sizes $(2n + 1)$. For the largest template size analysed so far, 11 bits, it seems the improvement when going from 9-bit to 11-bit

Table 7. Characteristic statistical data of the $r_{\max} = \max(r_i)|_{i=0..55}$ ranking distributions and the corresponding estimated remaining rest entropy \bar{E} for a hypothetical 2-key 2DES implementation for various template sizes.

Templates	3-bit (16K)		5-bit (16K)		7-bit (16K)		9-bit (16K)		11-bit (16K)	
	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}	r_{\max}	\bar{E}
Permille	4.22	83.80	16.39	77.21	50.40	62.55	153.33	52.19	561.00	48.55
Percentile	5.05	91.40	19.21	83.88	63.60	73.08	202.86	61.88	744.67	58.27
64th Quantile	5.10	93.24	20.07	85.62	66.53	75.17	216.57	64.34	789.50	60.62
32th Quantile	5.24	94.88	21.20	88.64	72.49	78.74	242.67	68.61	886.80	64.72
Hexacile	5.52	95.70	22.52	91.76	79.24	82.72	270.50	73.53	997.27	69.45
Octile	6.01	96.87	24.08	94.99	87.27	87.23	305.11	78.76	1139.20	74.91
Quartile	6.15	101.6	25.93	98.66	96.95	92.28	347.79	84.66	1323.60	81.23
Median	6.43	104.34	28.04	102.72	108.48	97.85	405.58	91.55	1569.17	88.67
Average	6.88	105.60	27.94	107.41	106.47	104.56	396.17	100.22	1531.59	98.19

templates is not as large as when going from 7 to 9 bits. This may have to do with the rather small ensemble analysed, but it could also be an indication that the base of 4.75M traces used for generating the templates is not enough for obtaining sufficiently accurate 11-bit templates.

In any case, when comparing the results with the worst-case result based on \bar{r} discussed earlier in the context of Eq. (12), one finds that the worst-case assumption is indeed a very aggressive, but still realistic one. Moreover, as the template size increases, the worst-case results correspond to higher quantiles, meaning the worst case becomes more likely to occur. This is attributed to the effect of overlapping templates and the resulting correlation between the results r_i at the 28 template positions i along the rings. To conclude, the worst-case analysis is a useful tool to get a quick impression about the strength of the attack without having to work out a large statistics.

To be fair, though, one needs to add a few bits to the values shown in Table 6 to weight in the fact that the results only hold for a certain quantile of a given random set of traces, whilst for the rest of the traces the attack would not succeed within the given quantile. Thus, for instance, for the Quartile results shown here, 2 bits need to be added, for the Octile results 3 bits, and so on.

An example attack scenario could be a set of 800 smart cards that are simultaneously attacked on the basis of a given single trace in Exploitation Phase — different for each card — and out of these roughly 100 smart cards would be broken with an average brute force effort of 27.57 bits when using 11-bit templates. A further 100 smart cards would yield when spending another 4 bits of brute-force effort.

In order to get an idea how the brute-force effort would look like to break 2-key and 3-key 3DES implementations, we have summarized in Table 7 the most important parameters of the r_{\max} ranking distributions and their corresponding estimated remaining rest entropy \bar{E} for a hypothetical 2-key 2DES implementation. It is based on the same ensemble of traces as used for generating Table 6,

but now always pairing two consecutive traces to form one longer trace of a hypothetical 2-key 2DES operation, and finally determining $r_{\max} = \max(r_i)_{i=0\dots55}$ across all 56 template positions.

This can then be applied to a 2-key 3DES implementation of the form $\text{DES}(k_1)\text{DES}^{-1}(k_2)\text{DES}(k_1)$ by applying Table 7 on the effectively 112-bit search, with the first trace belonging to the two outer $\text{DES}(k_1)$ operations, and the second trace always belonging to the inner $\text{DES}^{-1}(k_2)$ operation. Such a simultaneous attack of k_1 and k_2 is not possible with a standard DPA. Also, it should be noted that in doing so we grossly underestimate the leakage of the two outer $\text{DES}(k_1)$ operations having the same key k_1 , since for these twice as many POIs are in fact available than had been used for calculating the templates and pattern-template matching results underlying Table 6. Hence, for the two outer $\text{DES}(k_1)$ operations results better than those shown in Table 6 are to be expected. Consequently, from this point of view Table 7 will be a too conservative estimate for the leakage of a 2-key 3DES implementation.

On the other hand, Table 7 should be fairly accurate for a 3-key 3DES in a meet-in-the-middle-attack scenario, if and when it is applicable.

From all these results and observations we conclude that the correlation of neighbouring templates due to overlap along the rings is an important aspect that needs to be heeded in any key-search strategy and it will generally improve results substantially. Secondly, we conclude that C-type templates as large as possible should be used.

To conclude this Section we investigate a few example traces and apply a full key search as outlined at the end of Sec. 2.2. To be precise, we only present results for key searches within the C-Register or the D-Register separately, but not both at the same time. However, it is fairly straightforward to generalise our results to the case of a complete key search across both registers. In doing so one should take advantage of the different average rankings in the C- and D-Registers, as visualised in Figs. 14 to 16, top graphs, which translate to different average search depths in these registers. For large template sizes, like 13 bits, the difference in average rankings between the two is as large as a factor two.

In Tables 8 & 9 we show for selected traces — identified simply by their Trace ID / position in the ensemble of traces — what the found remaining rest entropy is. Here $E_{r_{\max}}^{C'}$ is the remaining rest entropy as estimated by Eqs. (6) and (11), whilst $E_{r_{\max}}$ is the result of a real key search based on sorting the graphs introduced at the end of Sec. 2.2 according to increasing values of r_{\max} . Likewise, E_r is based on sorting the graphs for increasing average ranking values, $r = \frac{1}{14} \sum_{i=0}^{13} r_i$ for the C-Register, and $r = \frac{1}{14} \sum_{i=14}^{27} r_i$ for the D-Register.

Firstly, we notice that whilst the execution time required for setting up the list of nodes, T_1 , as well as the time required for sorting /enumerating this list, T_2 , do depend on the size of the template, of course, they do not really depend on the position of the correct key in this list — as expected. Note that to keep things simple T_1 is the time taken for setting up the lists for $E_{r_{\max}}$, E_r as well as a few other key enumeration schemes that we tried, all together in a single run.

Table 8. Comparison of the rest entropy $E_{r_{\max}}^{C'}$ as estimated by Eqs. (6) and (11) (applied to C-Register and D-Register separately) for selected traces, with actual key searches based on minimal average ranking r ($:E_r$) or minimal r_{\max} ranking ($:E_{r_{\max}}$). Also, rough run times on a PC are given for creating the lists of graphs, T_1 , and enumerating / sorting these lists for efficient subsequent brute-force search, T_2 .

Trace ID	T'pl. Size	C-Register						D-Register					
		[bits]	$E_{r_{\max}}^{C'}$	T_1 [s]	T_2 [s]	E_r	T_2 [s]	$E_{r_{\max}}$	$E_{r_{\max}}^{C'}$	T_1 [s]	T_2 [s]	E_r	T_2 [s]
4750272	5	27.48	58	21	25.38	20	26.17	28.51	59	20	26.65	20	27.18
4750032	5	25.25	58	22	20.99	20	24.41	25.25	58	21	20.09	20	24.25
4750068	5	24.04	58	20	18.62	20	23.47	22.07	59	21	13.9	20	21.37
4756552	5	18.3	64	23	16.16	23	18	19.89	66	21	13.86	20	19.22
4750232	5	17.45	58	21	16.1	20	17.37	15.64	57	21	15.62	19	15.35
4760532	5	15.64	59	21	14.45	20	15.41	17.45	59	21	14.81	20	17.36
4763629	5	19.11	58	21	14.81	19	18.88	11.45	67	24	9.59	23	11.3
4754072	5	12.58	57	20	11.7	19	13.73	18.3	60	20	14.07	20	18.59
4777975	5	13.65	58	20	13.61	20	13.44	8.97	58	20	8.95	19	8.46
4781560	5	12.58	58	21	12.47	20	12.3	10.25	58	19	10.01	20	9.78
4763788	5	13.65	59	21	13.84	20	13.87	11.45	58	20	9.86	20	11.63
4780499	5	12.58	63	22	7.36	21	11.77	14.67	58	21	10.27	20	14.36
4750798	5	10.25	58	19	7.32	19	10.24	19.89	60	21	16.32	20	19.43
4763782	5	10.25	62	23	6.49	22	9.93	12.58	59	20	12.35	20	12.09
4757225	5	10.25	59	20	10.53	20	9.44	10.25	62	20	10.79	19	9.51
4750272	7	27.9	136	54	25.14	51	26.91	28.02	136	54	26.22	52	27.06
4750032	7	17.61	140	59	19.9	56	20.16	23.24	144	56	20.76	51	23.53
4750068	7	21.07	136	55	17.8	51	22.49	17.18	138	55	14.28	52	19.7
4750232	7	15.86	137	54	16.86	51	18.51	8.49	136	55	14.96	56	12.34
4756552	7	14.19	137	54	14.93	52	17.01	11.82	136	55	14.19	51	15.42
4763629	7	13.43	137	54	14.15	51	16.51	6.77	137	54	8.48	51	9.70
4750798	7	9.45	136	55	8.94	51	13.53	13.43	136	55	16.08	52	16.99
4760532	7	8.82	136	55	12.79	52	12.18	12.91	137	55	15.11	52	16.63
4757225	7	10.96	136	55	11.15	51	14.21	7.48	137	55	11.29	52	11.25
4780499	7	10.67	150	56	8.92	57	13.29	8.16	150	71	10.32	74	12.12
4754072	7	7.48	136	54	11.54	51	12.8	10.96	136	55	12.89	51	14.54
4777975	7	10.07	137	55	11.97	51	13.23	6.77	137	54	9.60	51	10.79
4763788	7	8.16	153	56	13.59	57	12.78	6.41	152	57	10.47	52	9.60
4781560	7	7.13	148	61	11.38	55	10.08	7.82	151	61	8.58	59	11.43
4763782	7	4.89	137	55	7.06	52	7.73	8.81	137	55	12.03	52	12.37

Secondly, there is a fairly good agreement found between the estimated rest entropy $E_{r_{\max}}^{C'}$ and the entropies E_r and $E_{r_{\max}}$ actually obtained with one of the two key-enumeration strategies, as long as the trace does not leak too much, i.e., when its r_{\max} is in the bulk of the distributions shown in Figs. 10 to 13. For traces stemming from the left tail of these distributions, the estimated rest entropy $E_{r_{\max}}^{C'}$ is generally underestimating the rest entropy by up to 8 bits. Still, the results, e.g., for Trace ID 4763782 and 11-bit templates are such that

Table 9. Same as Table 8, but for larger template sizes.

Trace ID	T'pl. Size	C-Register						D-Register					
		[bits]	$E_{r_{\max}}^{C'}$	T_1 [s]	T_2 [s]	E_r	T_2 [s]	$E_{r_{\max}}$	$E_{r_{\max}}^{C'}$	T_1 [s]	T_2 [s]	E_r	T_2 [s]
4750272	9	27.01	414	176	25.15	176	26.53	25.72	414	176	25.84	175	25.71
4750032	9	21.77	466	200	21.5	194	23.85	20.73	433	198	20.54	185	22.96
4750068	9	9.80	414	175	17.33	176	16.22	11.91	415	175	15.27	176	17.74
4750232	9	11.49	424	176	16.11	175	17.16	8.86	414	176	15.98	176	15.56
4754072	9	8.12	415	175	11.43	175	15.51	9.10	415	175	13.54	175	16.05
4780499	9	11.56	415	176	11.05	176	16.65	4.97	415	176	10.42	176	10.91
4756552	9	7.95	416	176	12.84	176	14.26	6.38	417	176	14.12	176	13.15
4750798	9	5.83	419	180	9.60	169	12.1	8.45	452	210	16.11	192	15.08
4760532	9	5.92	418	179	12.15	168	12.22	7.27	418	179	14.73	169	14.09
4757225	9	7.87	415	176	12.39	176	14.16	5.64	414	176	11.26	176	12
4763782	9	3.97	415	176	7.90	177	9.79	8.86	415	176	12.19	175	15.12
4763629	9	7.44	414	176	12.08	175	14.08	3.97	414	176	8.99	176	9.49
4763788	9	5.55	462	200	13.31	184	12.36	4.77	452	178	10.82	188	10.92
4781560	9	6.56	414	175	12.58	176	12.39	4.38	414	176	7.38	176	10.26
4777975	9	4.77	417	180	11.09	168	9.83	4.77	418	179	10.63	169	11.41
4750272	11	26.57	1741	824	24.74	805	26.34	24.83	1649	811	25	814	25.36
4750032	11	21.95	1633	793	22.37	795	24.18	18.17	1640	805	21.48	806	22.24
4750232	11	9.34	1638	800	15.12	805	16.08	8.13	1636	795	16.66	799	16.21
4750068	11	6.98	1634	797	16.36	800	14.35	10.61	1639	801	17.38	804	18.07
4750798	11	5.11	1632	789	8.52	794	11.58	14.41	1633	788	17.63	778	20.35
4780499	11	9.30	1678	836	9.13	817	15.18	5.09	1628	790	12.24	797	12.74
4757225	11	7.09	1636	794	11.01	798	14.63	4.52	1629	786	11.09	790	12.34
4754072	11	5.13	1790	878	10.32	873	13.17	6.67	1657	796	13.79	800	15.77
4781560	11	7.54	1694	866	12.37	861	14.28	2.86	1824	842	3.81	793	8.34
4763788	11	5.98	1628	786	13.47	788	13.81	3.92	1630	792	10.66	794	11.45
4760532	11	3.86	1639	800	10.65	804	10.41	4.89	1638	804	13.63	810	13.15
4777975	11	5.28	1689	802	11.06	805	11.23	3.99	1630	798	10.37	804	11.62
4763629	11	5.56	1635	797	11.77	803	13.69	3.61	1632	793	10.33	793	11.15
4756552	11	3.29	1640	804	11.55	806	8.40	4.27	1636	799	13.48	803	12.1
4763782	11	1.70	1629	791	5.61	795	4.81	3.68	1631	789	11.58	792	11.01

— if taken alone — the C-Register key search terminates in as little as 4.81 bits when enumerating keys according to r_{\max} , and also the D-Register search is done after 11.01 bits. When combining this one has to take the maximum of the two, meaning that the total key will be found after a brute-force effort of ≈ 22.02 bits.¹⁸ Even better, for Trace ID 4757225 and 5-bit templates we find when enumerating keys according to r_{\max} that the search in the C-Register

¹⁸ Strictly speaking, it is not quite correct to simply take the maximum of the C- and D-Register results and simply double it, since both registers will have slightly different search / key enumeration lists and thus the search will terminate slightly differently in those two registers. But it should be good enough to give an indication.

Table 10. Comparison of the rest entropy $E_{r_{\max}}^{C'}$ as estimated by Eqs. (6) and (11) (applied to C-Register and D-Register separately) for selected traces, with actual key searches based on minimal average ranking r ($:E_r$) or minimal r_{\max} ranking ($:E_{r_{\max}}$), and using the additional assumption of knowing the Hamming weight of the correct key.

Trace ID	T'pl. Size [bits]	C-Register			D-Register		
		$E_{r_{\max}}^{C'}$	E_r	$E_{r_{\max}}$	$E_{r_{\max}}^{C'}$	E_r	$E_{r_{\max}}$
4750032	5	25.25	15.28	18.45	25.25	14.25	17.44
4750068	5	24.04	12.2	17.78	22.07	9.05	15.06
4763788	5	13.65	7.02	7.71	11.45	4.32	5.25
4763782	5	10.25	3.58	4.25	12.58	4.64	4.09
4777975	5	13.65	6.77	7.08	8.97	4.17	3.7
4757225	5	10.25	5.36	4.58	10.25	2.00	2.32
4750272	7	27.9	18.83	20.76	28.02	19.76	20.57
4750032	7	17.61	14.28	14.56	23.24	14.94	17.07
4750068	7	21.07	11.55	16.65	17.18	9.15	13.57
4763629	7	13.43	7.71	9.31	6.77	1	3.17
4757225	7	10.96	5.67	9.16	7.47	2	3.32
4763788	7	8.16	6.97	6.19	6.41	4.81	4.58
4763782	7	4.89	3.81	3.58	8.81	4	4.17
4750272	9	27.01	18.84	20.39	25.72	19.41	19.25
4750032	9	21.77	15.97	18.14	20.73	14.79	16.71
4750232	9	11.49	11.20	12.17	8.86	10.87	10.39
4757225	9	7.87	7.34	8.94	5.64	2	4.17
4760532	9	5.92	6.41	6.49	7.27	8.61	7.98
4750798	9	5.83	4.32	6.61	8.45	10.03	9.16
4763629	9	7.44	5.91	7.29	3.97	1	3.58
4763788	9	5.55	7.12	6.17	4.77	5.29	4.86
4777975	9	4.77	4.39	3.17	4.77	6.13	6.58
4750032	11	21.95	16.83	18.44	18.17	15.73	16.19
4750232	11	9.34	10.37	11.13	8.13	11.61	11.21
4763788	11	5.98	7.17	7.73	3.92	5.00	5.64
4756552	11	3.29	6.49	4.91	4.27	8.75	7.38
4763782	11	1.70	3.32	1.58	3.68	3.32	2.58

terminates at 9.44 bits, and in D-Register at 9.51 bits. So, in this case the total brute-force effort is about 19 bits only.

When comparing these results based on key-search strategies with the “worst case” results predicted by evaluating the rest entropy formula Eq. (10) for a simple average ranking r , as tabulated at the bottom of Table 6, one finds those “worst case” results actually to be in surprisingly good agreement and they do live up to their name.

Secondly, we want to point out that whilst Eqs. (6) and (11/10) are only accurate in the bulk of the r_{\max} distributions and are predicting too small rest entropies in the left wing of such a distribution, where $r_{\max} \ll M$, this effect is

much less important in the case of a triple DES, be it two key or three key. The reason is simply that in those cases $r_{\max} \ll M$ is much less often fulfilled than in the case of single-key DES. Consequently, Table 7 should be a fairly accurate assessment of the 2-key TDES case.

Preliminary results indicate that the key-enumeration / search algorithm might be slightly improved upon when first sorting for $r_{\min} = \min(r_i)|_{i=0..13}$ for the C-Register, respectively $r_{\min} = \min(r_i)|_{i=14..27}$ for the D-Register, and then, within each cluster of r_{\min} , perform a sorting according to either minimal average ranking or minimal r_{\max} . More generally, a better understanding of the statistical properties of the correct key rankings will help to improve the search strategies.

As indicated in Sec. 2.2, these results can be improved upon when the effective Hamming weight of the total key is known. Given the strong leakage seen, this is not an unreasonable assumption to make. When the effective Hamming weight of the total key is known, then for each candidate of the C-Register, one can work out the only possible remaining Hamming weight of the candidate of the D-Register that is worth trying in the brute-force search. Table 10 shows the results when assuming perfect Hamming-weight leakage. For instance, for Trace ID 4763782 and 11-bit templates, if we take the C-Register to be the primary search list, then for a key-enumeration metric based on minimal r_{\max} , the partial remaining rest entropy found in Table 9 is just 4.81 bits. From Table 10 we find for the same Trace ID, same template size, and same key-enumeration metric based on r_{\max} a partial remaining rest entropy for the D-Register of merely 2.58 bits when the Hamming weight of the D-Register is known. Of course, it is not quite correct to simply add things up here, since the search depth is likely not the same in both registers, but it does give an impression of where things are heading if the effective Hamming weight is also known. In the example given, the total brute-force effort is likely to be less than 10 bits to recover the entire DES key.

4 Conclusions

A new template attack has been performed on the DES hardware engine of a fairly recent smart card using a *single* measurement in Exploitation Phase. A vulnerability was found in the key scheduling, leading to leakages as large as 70% in the correlation function. The template analysis is exploiting a ring structure seen in the key scheduling of the DES, which is independent of the particular implementation at hand. Using the concept of highly overlapping templates and a key-enumeration strategy adapted to this scenario we analysed the statistical key ranking distribution of 32K single-trace attacks. The key-enumeration step introduces only a negligible overhead in the entire key search of the order of a few minutes to a few hours, for a standard PC, depending on the size of the templates chosen. A particularly leaky trace was found with a remaining rest entropy as low as ≈ 19 bits that only need recovering by brute force. More representative is our finding that the first Octile of the key-ranking distribution has an estimated

average remaining rest entropy of $\approx 28 + 3$ bits, and the Median of $\approx 37 + 1$ bits. Whether or not such a brute-force attack is indeed possible will depend on the application at hand. With this result, and provided a meet-in-the-middle attack scenario is applicable, even a triple-key triple DES seems below the limit of 60 bits of remaining entropy often imposed in Common Criteria, when using 13-bit templates or larger.

For this particular platform the findings become even more severe as it is using a DES hardware coprocessor built in what is called a hard-macro design methodology, meaning that it is a design block with fixed geometry all the way down to the gate level. This hard macro is thus optically visible in the layout, be it in 140nm, 130nm, or 65nm, and it is used in many other products in precisely the same shape over and over again. Hence, once a weakness such as this one has been found in one family member, it can be easily reproduced in all other family members as well. Worse still, one has to assume that templates generated for one family member will be applicable to all other family members as well, leading to potential collateral damages between different markets where this product is deployed. The security risks that could result from hard-macro designs and their reuse have been exposed before [14].

In fact, as this hard macro or slight variants of it is used in billions of devices today, ranging from banking cards to electronic passports, ID cards, health cards and driving licences to TPMs, IoT applications and mobile phones, it is important to address this vulnerability in the proper way. Following a responsible disclosure policy the authors made the relevant certification authorities aware of this vulnerability in April 2016 and provided all details requested since then.

The underlying hardware has a Common Criteria certificate at level EAL 5+, but the JAVA OS itself is not certified as far as we are aware. Typically, the hardware CC certification would result in some guidance for the software development on that platform as to how to use the hardware platform in such a way that the assurance level of EAL 5+ can actually be reached. Since this is a black-box analysis without any knowledge about the hardware design nor the OS, we do not know whether the OS developer has indeed heeded any such guidance possibly given on the usage of the DES coprocessor of this hardware platform.

However, in terms of possible and — more importantly — effective countermeasures against this template attack that could in principle be part of such a guidance, one needs to realise that when merely a single trace is needed for the attack, most if not all of the traditional countermeasures used in software will fail, since one way or another they all rely on averaging. Introducing a random jitter between the DES calls is not effective, as was already demonstrated in this work. Another countermeasure is based on calling the DES N times, where $N - 1$ times a wrong key was used and only once the correct key, and this at random positions of the N DES calls. This countermeasure will only have a limited value, as one can attack each of the N DES calls separately in the same single trace, and simply check by brute force whether it is using the correct key or not. Obviously, when N becomes very large, the effort for brute forcing each

and every DES call will increase substantially, but then also the performance impact on the application will be very massive and it may fail to meet targets for, e.g., transaction times. In any case, the brute-force effort only scales linearly with this countermeasure, which seems quite affordable for the attacker. Finally, running another coprocessor in parallel is also not effective when it is possible to localise the DES coprocessor in the layout of the chip and place the EM probe on top of it. This should be straightforward to do as this smart card — as just discussed — uses a hard macro for the DES hardware coprocessor which is easily located by optical inspection of the layout. Based on all these considerations we conclude that countermeasures in software — for instance in a crypto library — are simply not effective in mitigating this attack.

A number of improvements are clearly possible for this attack. Trivially, it should be noted that the measurements were done with a sampling rate that yielded only 9 sample points per peak in the correlation function and in χ^2 . Thus, increasing the sampling rate alone could possibly already improve the results shown here as it would increase the number of POIs. Using an oscilloscope with more than 8 bits resolution should also help. Likewise, it will pay off to spend more effort on aligning all traces even further. Careful alignment is particularly crucial for single-trace attacks. Figures 5 and, e.g., 8 indicate that the alignment of especially the first DES block is not good yet. But perhaps the most straightforward improvement is to collect more than 5M traces in the Profiling Phase, perhaps as much as 10M, and construct larger C-type templates, as Tables 6 & 7 seems to suggest that it is beneficial to use larger templates.¹⁹ 15-bit templates are certainly within computational range, and beyond that stochastic models may help [15].

In addition, one can also increase the overlap between neighbouring templates further by inserting between any two templates of the type Eqs. (4) and (5) another template of the shape

$$\begin{array}{ccccccc}
 7 & \rightarrow & 21 & \rightarrow & 35 & \rightarrow & 49 \\
 & \searrow & \nearrow & \searrow & \nearrow & \searrow & \nearrow \\
 & & 14 & \rightarrow & 28 & \rightarrow & 42 & \rightarrow & 31
 \end{array} \tag{14}$$

which makes for another 28 templates along the C rings. Now neighbouring templates only differ not by two, but just by one key bit. It should be noted that this will not only affect the search / key-enumeration strategies, but also the formula for the remaining rest entropy will be modified to

$$E_r^{C'} = 2 + 56 \log_2(r) . \tag{15}$$

Another improvement of the results will be possible when combining the results of the C rings with other leakages found, such as for instance the leakage

¹⁹ On the other hand, for the smaller template sizes of 5 or 7 bits, it is possible to use less traces than we have during the Profiling Phase — perhaps of the order of 1M or 2M traces.

exploited in [12], which was on the same device. Also there substantial leakage of the key was present in a single trace.

But perhaps the biggest improvements to this attack can be had by improving the key–enumeration strategy for the brute–force attack. One direction already identified is to make use of the fact that some bits leak up to 12 times during the entire DES key schedule, whilst others leak just once or twice, and this should have an effect on the “stickiness” of these bits.

More work is required to get a better understanding of the statistical properties of the correct key rankings as this will help to improve the search strategies.

References

1. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) *Advances in Cryptology - CRYPTO '99*, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397. Springer (1999), http://dx.doi.org/10.1007/3-540-48405-1_25
2. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2000*, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings. *Lecture Notes in Computer Science*, vol. 1965, pp. 238–251. Springer (2000), http://dx.doi.org/10.1007/3-540-44499-8_19
3. Gierlichs, B., Batina, L., Preneel, B., Verbauwhede, I.: Revisiting higher-order DPA attacks:. In: Pieprzyk, J. (ed.) *Topics in Cryptology - CT-RSA 2010*, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings. *Lecture Notes in Computer Science*, vol. 5985, pp. 221–234. Springer (2010), http://dx.doi.org/10.1007/978-3-642-11925-5_16
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2002*, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. *Lecture Notes in Computer Science*, vol. 2523, pp. 13–28. Springer (2002), http://dx.doi.org/10.1007/3-540-36400-5_3
5. Archambeau, C., Peeters, E., Standaert, F., Quisquater, J.: Template attacks in principal subspaces. In: Goubin, L., Matsui, M. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2006*, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. *Lecture Notes in Computer Science*, vol. 4249, pp. 1–14. Springer (2006), http://dx.doi.org/10.1007/11894063_1
6. Medwed, M., Oswald, E.: Template attacks on ECDSA. In: Chung, K., Sohn, K., Yung, M. (eds.) *Information Security Applications*, 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, Revised Selected Papers. *Lecture Notes in Computer Science*, vol. 5379, pp. 14–27. Springer (2008), http://dx.doi.org/10.1007/978-3-642-00306-6_2
7. Fouque, P., Leurent, G., Réal, D., Valette, F.: Practical electromagnetic template attack on HMAC. In: Clavier, C., Gaj, K. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2009*, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. *Lecture Notes in Computer Science*, vol. 5747, pp. 66–80. Springer (2009), http://dx.doi.org/10.1007/978-3-642-04138-9_6

8. Mangard, S.: A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: Lee, P.J., Lim, C.H. (eds.) Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2587, pp. 343–358. Springer (2002), http://dx.doi.org/10.1007/3-540-36552-4_24
9. Xiao, L., Heys, H.M.: A simple power analysis attack against the key schedule of the camellia block cipher. *Inf. Process. Lett.* 95(3), 409–412 (2005), <http://dx.doi.org/10.1016/j.ipl.2005.03.013>
10. Compton, K.J., Timm, B., VanLaven, J.: A simple power analysis attack on the serpent key schedule. IACR Cryptology ePrint Archive 2009, 473 (2009), <http://eprint.iacr.org/2009/473>
11. El Aabid, M.A., Guilley, S., Hoogvorst, Ph: Template Attacks with a Power Model — Illustration on the Side-Channel Cryptanalysis of an Unprotected DES Crypto-Processor. Cryptology ePrint Archive, Report 2007/443, 2007, <https://eprint.iacr.org/2007/443.pdf>
12. Wagner, M., Hu, Y., Zhang, C., Zheng, Y.: Comparative Study of Various Approximations to the Covariance Matrix in Template Attacks. Cryptology ePrint Archive, Report 2016/1155, 2016
13. Stinson, D.R.: Cryptography Theory and Practice. CRC Press, 1995, 1st edition (ISBN 0-8493-8521-0)
14. Tarnovsky, Ch.: Deconstructing a 'Secure' Processor: Hacking the Smartcard Chip. Blackhat 2010 DC, Arlington, VA, February 2, 2010, <https://www.blackhat.com/presentations/bh-dc-08/Tarnovsky/Presentation/bh-dc-08-tarnovsky.pdf>
15. Schindler W., Lemke K., Paar C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao J.R., Sunar B. (eds) Cryptographic Hardware and Embedded Systems. CHES 2005. Lecture Notes in Computer Science, vol 3659. Springer, Berlin, Heidelberg