Exploring the Resilience of Some Lightweight Ciphers Against Profiled Single Trace Attacks

Valentina Banciu, Elisabeth Oswald, and Carolyn Whitnall

University of Bristol, Department of Computer Science
Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK
{valentina.banciu, elisabeth.oswald, carolyn.whitnall}@bristol.ac.uk

Abstract. This paper compares attack outcomes w.r.t. profiled single trace attacks of four different lightweight ciphers in order to investigate which of their properties, if any, contribute to attack success. We show that mainly the diffusion properties of both the round function and the key schedule play a role. In particular, the more (reasonably statistically independent) intermediate values are produced in a target implementation, the better attacks succeed. A crucial aspect for lightweight ciphers is hence the key schedule which is often designed to be particularly light. This design choice implies that information from all round keys can be easily combined which results in attacks that succeed with ease.

1 Introduction

In our increasingly digitally interconnected world developing secure cryptographic software is a key challenge in practice. The stakes at hand are considerable: with the advent of 'smart devices' (e.g. smart meters, smart appliances) cryptography becomes deeply embedded in consumers' everyday lives. Such devices' primary functionalities are to provide commodities to consumers. Consequently, despite the importance of cryptography, the resources remaining to implement it may be very limited. This requirement for lightweight cryptography has inspired many new designs, such as PRESENT [**?**] (recently standardised as ISO/IEC 29192-2:2012), KLEIN [**?**] and LED [**?**] among many others. Some were designed with suitability for hardware implementations in mind (such as PRESENT), but by and large all of them are also considered for implementations in software on small microprocessors as found in typical smart devices.

Deeply embedded systems are by nature very exposed to side-channel adversaries: their market is such that millions of these devices can be found 'in the wild' as part of applications, but many of the processors

This article is the final version submitted by the authors to Springer-Verlag. The final published version is available at 10.1007/978-3-319-21476-4_4.

that are used in smart devices can be picked up off the shelf for only a small cost. This makes them ideal targets for the most sophisticated side-channel attacks, which are based on profiling. Profiling can be used both in simple and differential style attacks. Differential style attacks exploit leaking information associated with different plaintext data but a fixed key and it is well understood that the choice of e.g. the substitution function has an important impact on the vulnerability of a cipher [**?**].

In this paper we aim to investigate the security of specific implementations of lightweight ciphers against profiled single trace attacks. In particular, we are interested to discover which properties, if any, contribute to the general vulnerability of a lightweight cipher against this particular type of adversary.

Our Contribution. Using a pragmatic approach to assess the resilience of ciphers against profiled single trace attacks, we conduct several experiments for four ciphers: AES (to provide a well-known baseline for comparison), PRESENT (to see how ciphers which are attractive for hardware implementations fare when implemented in software), KLEIN and LED (sharing some features with AES and PRESENT while aiming at resource minimisation). All experiments assume the same underlying architecture (8-bit) and are based on publicly available implementations of the investigated ciphers [**?**]. Apart from providing a common denominator, we regard this suite as relevant to a high proportion of real-life cases since algorithm implementations are done in a 'natural' way. Note that we work with simulated traces, which however does not imply perfect measurements (see Sect. 3 for details on how inherent noise is taken into account).

Our method, which we describe in Sect. 4.1, can be regarded as a basic tool for evaluating the resilience of specific cipher implementations against single trace attacks. As such, we draw inspiration from [**?**,**?**] which independently suggested using a pragmatic key enumeration approach. However, instead of enumerating single subkeys we test multiple candidates at the same time, thus achieving 100% success rate and running time of under 5 minutes for all test cases, which represent significant improvements.

We can show that the impact of choice of substitution boxes is negligible w.r.t. profiled single trace attacks and equally the impact of the global diffusion characteristics over a single round have shown no impact in the case of the four studied ciphers. We find however that simply the number of statistically independent intermediate steps (i.e. the 'attack

surface') that are required on the architecture is a good predictor for the vulnerability of a cipher against profiled single trace attacks. This shows that ciphers that can be elegantly described on a variety of architectures will be most resilient to such attacks as they provide a smaller attack surface.

Related Work. As mentioned above, this work is strongly linked to [**?**,**?**], but introduces significant improvements in terms of running time and success rate. As such, and because the output of our attack is a reduced key space, our method represents an uncostly means of evaluating the worst-case scenario of single-trace attacks such as solver-aided algebraic side-channel attacks [**?**,**?**,**?**]. The cited papers focus on AES and PRESENT encryption, under the Hamming weight leakage model. Further leakage models are considered in [**?**,**?**]. We are not aware of single trace attacks on KLEIN and LED. The security of the (unprotected) AES key schedule algorithm has been studied in [**?**,**?**]. More recently, [**?**] described attacks on masked implementations in an ideal scenario where noise is practically negligible. We are not aware of single trace attacks on the KLEIN or PRESENT key schedule.

2 Overview of Ciphers, Notation and Implementation Characteristics

The ciphers in our suite are instances of substitution-permutation networks (SPN), and therefore have similar components in their encryption functions. In particular, three common types of operations are utilised by all four ciphers:

- the key addition (bitwise `xor` between the round key and current state), denoted by `AddRoundKey`;
- the n-bit substitution (a highly non-linear transformation which acts upon groups of n bits substituting them via a lookup table called the S-box, with $n \in \{4, 8\}$ fixed), represented by `SubBytes`, `SubNibbles`, `sBoxLayer` and `SubCells`;
- finally, the byte mixing (a linear transformation acting on sets of bits), namely `MixColumns`, `MixNibbles`, `pLayer` and `MixColumnsSerial`.

Except PRESENT, the chosen ciphers utilise explicit byte (or 4-bit) renumbering functions, i.e. `ShiftRows` and `ShiftNibbles`. Additionally, the `AddRoundConstant` of LED `xor`s the state with a round constant (can be done at the same time as the key addition).

AES. The Advanced Encryption Standard (AES) is a symmetric block cipher with a fixed block size of 128 bits and a variable key size of 128, 192, or 256 bits respectively corresponding to 10, 12 and 14 encryption rounds. Throughout this document we refer to the 128-bit key variant simply as AES. At the beginning of the encryption process, the plaintext is `xor`-ed with the secret key. Subsequently, each encryption round bar the last one consists of the successive application of `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey`; the last encryption round skips `MixColumns`. The key expansion is elegant. It reuses components from the round function and operates on so-called 'words'.

KLEIN. KLEIN is an AES-like lightweight block cipher, supporting a fixed 64-bit state and 64, 80, or 96-bit keys for 12, 16, respectively 20 rounds. Throughout this document we refer to the 64-bit key variant simply as KLEIN. Each encryption round consists of the successive application of `AddRoundKey`, `SubNibbles`, `RotNibbles` and `MixNibbles`. A final key addition is performed after the encryption rounds. Although the order of operations inside an encryption round is different, their succession starting from the beginning of the encryption process is the same as with AES; moreover, the `MixNibbles` of KLEIN is identical to the `MixColumns` of AES. The key expansion process is fairly simple, each new key byte depending on exactly two bytes from a previous key.

PRESENT. PRESENT consists of 31 rounds, has a 64-bit block size and 80 or 128-bit keys. Throughout this document we refer to the 80-bit key variant simply as PRESENT. An encryption round consists of the key addition `AddRoundKey`, followed by the substitution and permutation layers, `sBoxLayer` and `pLayer`. The permutation layer is designed to match the effects of the combination between `ShiftRows` and `MixColumns` of AES. A final key addition is performed after the encryption rounds. The key schedule is fairly minimal: each new round key is derived from the previous key via a bit rotation, a single application of the S-box and a single `xor` with a round constant.

LED. LED accepts a 64-bit block and 64-bit or 128-bit keys, and consists of 32, respectively 48 rounds. Throughout this document we refer to the 64-bit key variant simply as LED. The structure of an encryption round is the succession of `AddRoundConstant`, `SubCells`, `ShiftRows` and `MixColumnsSerial`. Then, four rounds make a step, and the encryption process consists of adding the round key and performing a step for a total

of 8 times, followed by a final key addition. LED has no key expansion algorithm, and the secret key is used as a round key in each round.

As can be observed, the structure of the encryption algorithms is highly similar: first, because homologous subroutines are used, and second, because the order of the subroutines is virtually the same. We note that the first `AddRoundConstant` of LED can be performed before the key addition, i.e. directly on the plaintext.

2.1 Implementation Characteristics

SPA attacks are usually studied in the context of software implementations on serial microprocessors. Typical power models that are found in practice are the Hamming weight (HW) and the Hamming distance (HD). Leakages of this kind are observed mainly because of intermediate values being written to or read from memory.

For AES, KLEIN and PRESENT, we target publicly available 8-bit implementations, available at `http://perso.uclouvain.be/fstandae/lightweight_ciphers/`. In the case of LED, the publicly available implementations can be found at `http://led.crypto.sg/software`. In the remainder of this section, we further discuss implementation details of the substitution and byte mixing layers; we consider that implementing the key or round constant addition is fairly straightforward and does not require clarification.

The byte substitution layer. As described in Section 2, KLEIN, PRESENT and LED use 4-bit lookup tables. In order to optimally fit on 8-bit architectures two consecutive nibbles (2×4 bits) are considered a unit and new lookup tables are built [**?**].

The byte mixing layer. For all ciphers this is the most demanding component w.r.t. efficient implementations on an 8-bit platform. We briefly explain the approaches taken by the different ciphers in turn.

`MixColumns` (used in AES and KLEIN). The implementation that we are targeting [**?**] follows the specification of the original AES proposal [**?**], and is given in Alg. 1, where the index i wraps around $1 \ldots 4$, i.e. $i + 1 = 1$ if $i = 4$. As mentioned before, KLEIN uses the same component but calls it `MixNibbles`.

`pLayer` (used in PRESENT). A naïve implementation of `pLayer` would consist of storing a table that describes the bit-level permutation. However, this takes up a considerable amount of memory and is unnecessary because the `pLayer` permutation is highly structured. The targeted implementation [**?**] uses this property and does not require any table-lookup. Algorithm 2 shows how the implementation that we attack applies the permutation to half of the state.

`MixColumnsSerial` (used in LED). In the specification paper of LED [**?**] the authors suggest an 8-bit implementation of `MixColumnsSerial` using lookup tables, see Alg. 4. Each element is regarded as part of $\mathrm{GF}(2^4)$ with the underlying polynomial for field multiplication given by $x^4 + x + 1$.

3 Assessing the Vulnerability to Profiled Single Trace Attacks

Profiling can be used in both differential and simple side-channel analysis, and it is well understood that in general it improves attack efficiency. However, for some methods it is considered essential: so-called single trace attacks such as e.g. SPA against the AES key schedule [**?**] or algebraic side-channel attacks (ASCA, [**?**]) require the extraction of leakage values from traces and their assignment to specific intermediate values in a cipher's implementation. This is a demanding requirement which by and large can *only* be satisfied by profiling an implementation, i.e. by extracting leakage models for all exploitable intermediate values, and then using them during attacks. Such profiles hence contain information about 'when' an intermediate leaks (i.e. they have information about the timing of instructions) as well as 'how' (i.e. the leakage model itself).

Whilst it remains an open problem to deal with errors related to when leakages occur in profiling attacks, there has been some progress w.r.t. dealing with errors that result from matching the profiling information to new traces. For the two approaches to single trace attacks (which we call pragmatic SPA and ASCA) previous work introduced the notion of a 'set size' [**?**,**?**,**?**] to capture the impact of noise on attacks using profiling information. The larger the set size, the less certain we are about the assignment of a leakage value to an intermediate, e.g. a set size of three implies that for a certain intermediate we have three possible leakage values as a result of using the profiling information. To assess then the vulnerability to profiled single trace attacks we are hence interested to experiment with different set sizes. For a study on the practical re-

Algorithm 1 `MixColumns` 8-bit implementation algorithm

Input: in_1, in_2, in_3, in_4
Output: $out_1, out_2, out_3, out_4$
1: $Tmp \leftarrow in_1 \oplus in_2 \oplus in_3 \oplus in_4$;
2: **for** $i = 1 \rightarrow 4$ **do**
3: $\quad Tm \leftarrow in_i \oplus in_{i+1}$;
4: $\quad Tm \leftarrow \texttt{xtime}(Tm)$;
5: $\quad out_i \leftarrow in_i \oplus Tm \oplus Tmp$;
6: **end for**

Algorithm 2 `pLayer` (permuting 32 bits)

Input: in_1, in_2, in_3, in_4
Output: $out_1, out_2, out_3, out_4$
1: $carry_1 \leftarrow 0; carry_2 \leftarrow 0$;
2: $out_1 \leftarrow 0; out_2 \leftarrow 0; out_3 \leftarrow 0; out_4 \leftarrow 0$;
3: **for** $i = 4 \rightarrow 1$ **do**
4: $\quad iTmp \leftarrow in_i$;
5: $\quad$ pLayerByte$(iTmp, out_1, out_2, out_3, out_4, carry_2)$;
6: $\quad carry_2 \leftarrow carry_1$;
7: **end for**

Algorithm 3 pLayerByte$(iTmp, out_1, out_2, out_3, out_4, carry_2)$

Input: $iTmp, out_1, out_2, out_3, out_4, carry_2$
Output: $out_1, out_2, out_3, out_4, carry_1$
1: $carry_1 = \text{mod}(iTmp, 2)$;
2: $iTmp = \text{floor}(iTmp/2) + carry_2 \times 128$;
3: **for** $repeat = 1 \rightarrow 2$ **do**
4: $\quad$ **for** $i = 1 \rightarrow 4$ **do**
5: $\quad\quad carry_2 = \text{mod}(out_i, 2)$;
6: $\quad\quad out_i = \text{floor}(out_i/2) + carry_1 * 128$;
7: $\quad\quad carry_1 = \text{mod}(iTmp, 2)$;
8: $\quad\quad iTmp = \text{floor}(iTmp/2) + carry_2 * 128$;
9: $\quad$ **end for**
10: **end for**

Algorithm 4 `MixColumnsSerial` 8-bit implementation algorithm

Input: in_1, in_2, in_3, in_4
Output: $out_1, out_2, out_3, out_4$
1: $out_1 \leftarrow \texttt{4} \times in_1 \oplus \texttt{1} \times in_2 \oplus \texttt{2} \times in_3 \oplus \texttt{2} \times in_4$;
2: $out_2 \leftarrow \texttt{8} \times in_1 \oplus \texttt{6} \times in_2 \oplus \texttt{5} \times in_3 \oplus \texttt{6} \times in_4$;
3: $out_3 \leftarrow \texttt{B} \times in_1 \oplus \texttt{E} \times in_2 \oplus \texttt{A} \times in_3 \oplus \texttt{9} \times in_4$;
4: $out_4 \leftarrow \texttt{2} \times in_1 \oplus \texttt{2} \times in_2 \oplus \texttt{F} \times in_3 \oplus \texttt{B} \times in_4$;

quirements of extracting side-channel information to this end, we refer the reader to [**?**].

As previously mentioned there are broadly two types of single trace attacks in the literature at present. Pragmatic SPA-style attacks were described early on [**?**] and essentially consist of enumerating key candidates by exploiting the leakage information across a single trace. This enumeration is by and large manually implemented and the result of such attacks is hence information about the size of the key space left to search through to find the secret key. In contrast, ASCA was developed later [**?**] and essentially feeds side-channel information in addition to plain and ciphertext to a solver which will then return the secret key unless it halts. ASCA was hoped to be more efficient as solvers are sophisticated software tools. It should be evident however that to assess the vulnerability of ciphers they are less suitable than pragmatic attacks: they either return the key or produce no information. In contrast pragmatic attacks allow to assess the size of the remaining key space (i.e. remaining after all side-channel information has been used to prune the overall key space) and so give us some information about how much the side-channel information has helped. This becomes particularly useful when considering larger set sizes: recent work [**?**] shows how pragmatic attacks can produce useful information for set sizes up to 5, whereas ASCA is unable to cope with such large set sizes. Consequently it seems most appropriate to use pragmatic attacks as an evaluation tool with increasing set sizes (we report results for set sizes up to 5).

In the following sections we investigate three important characteristics in turn. Firstly, is there any high level difference between the round functions of ciphers w.r.t. profiled single trace attacks? This means we look at attacks that only use some selected intermediates corresponding to the key components of any substitution-permutation network. Next, we investigate how the inclusion of additional intermediate leakages changes attack outcomes. Thirdly, we study how key schedule characteristics impact on the vulnerability.

For these attacks, we generated a set of 100 random 16-byte plaintext and ciphertext pairs, which are the fixed inputs for the cipher suite; when a smaller block is required, the pairs are truncated (i.e., for e.g. KLEIN the secret keys will consist of the first 8 bytes of each 16-byte key). Note that our attack actually utilises a single trace, thus the reported results are in fact averaged over 100 experiments.

4 Attacking Selected Intermediates from a Single Encryption Round

There are four steps across all ciphers in which side-channel relevant computations occur:

- loading the secret key from memory (we assume the plaintext is always known);
- performing the key addition (and the `xor`-ing with the round constant if the case);
- performing the substitution;
- computing the output of the byte mixing layer (i.e., `MixColumns`, `MixNibbles`, `pLayer`, `MixColumnsSerial`).

In the remainder of this section we describe our attack methodology and show how the side-channel information reduces the key space when considering these four steps for a single encryption round. Because all ciphers effectively act on the state in some block-wise manner we first explain what our basic 'block' is. In AES all but `MixColumns` act on individual bytes of the state. `MixColumns` operates on columns which implies that a suitable block would be a column (i.e. 4 bytes). Studying how a pragmatic SPA reduces the key space with regards to this block allows us to conclude on the result of the entire key because the blocks are independent. It is easy to see that such a 4-byte block is also an appropriate unit for the other ciphers when implemented on an 8-bit platform. Consequently we settled for this choice and the tables in this section show the reduction of the subkey space for a block (i.e. from 2^{32}). Note that this definition overrides the one in Sect. 2 without contradicting or hindering any of the inherent cipher properties.

4.1 Attack Strategy

As mentioned in Sect. 1, our attack is derived from [**?**] and therefore similarly consists of two phases: first, extracting four independent sets of key (byte) values based on the side-channel information up to the byte mixing layer, and second, linking the extracted key bytes into 4-byte keys based on the information from the byte mixing layer. Indeed, we also build and use 8-bit tables that enable us to directly extract possible key values based on the known plaintext and side-channel information from the S-box for all ciphers. However, for the second part, instead of then enumerating the (4-byte) keys and testing each one sequentially as in [**?**], we generate all possible keys (corresponding to a block) as the Cartesian

product of the previously derived sets and simulate their action on the inputs. With this we are able to reject several key candidates at the same time based on the side-channel information, which allows us to report a short running time for our attacks (under 5 minutes, but under one second for set sizes up to 2) and a success rate of 100% (previous attacks were liable to run out of memory, or to fail to complete within a fixed time interval, e.g. 48 h). This is a significant improvement to previous work.

All our experiments ran on a regular PC equipped with a Intel Core i7-2600s processor at 2.80 GHz and 4 GB of RAM.

4.2 Exploiting the 'Basic' Attack Surface

We first consider that the sole available side-channel information is related to the input and output values of the four steps outlined at the beginning of this section. Then, Tab. 2a summarizes the reduced subkey space for a block of the ciphers. It appears that the size of the reduced subkey space strongly depends on the set size, and less so on the specific cipher particularities (i.e. the quality of the S-box or byte mixing function are by and large irrelevant). Of course the overall key space of the ciphers is different and hence there is an additional penalty for AES as it requires to replicate the attack for more subkeys than the other ciphers.

4.3 Exploring the Impact of Increased Numbers of Intermediates

A natural question to ask is whether more leaking intermediate values will make an implementation more vulnerable, and if so, whether there is any clear relationship between the number of leaking intermediates and the increase in vulnerability. We hence took all of the intermediate values that occur in our implementations into account (i.e., the 'maximum' attack surface). Previous work (e.g., [**?**]) studied the impact of using more intermediate values by targeting more encryption rounds. Note that we are still focusing on a single round. We now explain for each of the ciphers in turn what and how many intermediates our implementations offer.

The implementation of `MixColumns` given in Alg. 1 leads to a set of 17 intermediate values, as follows: 4 corresponding to computing $in_i \oplus in_{i+1}$, 4 corresponding to computing $\texttt{xtime}(Tm)$, 8 corresponding to computing $iv = in_i \oplus Tm$ and $iv \oplus Tmp$ (where iv is an auxiliary intermediate value), and finally one corresponding to computing Tmp (n.b.: $in_i \oplus in_{i+1}$

Table 1: Size of the attack surface (i.e., number of leaked intermediate values) corresponding to the diffusion layer

Attack surface \ Cipher	AES	KLEIN	PRESENT	LED
'Basic'	4	4	4	4
'Maximum'	21	21	12	32

have already been computed, therefore a single new value is leaked when computing Tmp).

The implementation of `pLayer` given in Alg. 2 leaks as follows: the pLayerByte procedure leaks $2 + 2 \times 4 \times 4$ byte values, and is repeated a total of 4 times, therefore leading to 116 intermediate values. Note, however, that this set consists of values that differ in a single bit. This implies that although many intermediate values are produced, they are highly correlated. Consequently, given the 8-bit architecture that we work on, we effectively observe multiple copies of only 8 intermediates.

The implementation of `MixColumns` given in Alg. 4 leaks 7 intermediate values (4 for the table lookup, and 3 for computing the binary `xor` operations) for each output byte, thus leaking a total of 28 leakage points.

Table 1 gives an overview of the number of leaking intermediates per cipher. We listed only 8 intermediates for PRESENT because of the evident high correlation between the intermediates. We note that also for `MixColumns` there will be some correlated intermediates due to the fact that e.g. the final sum is computed by `xor`-ing. From this table, we would expect to see that LED should suffer most from including these additional intermediates, followed by AES and KLEIN.

Table 2b shows the results when incorporating the additional intermediates into the attack. All ciphers show that the simple intuition that more statistically independent intermediates provide more efficient attacks is true. The results for PRESENT also provide a clear example for the importance of having statistically independent intermediate values: although the total number of used intermediates is almost 7 times as large as with AES and KLEIN, the sizes of the respective reduced key spaces remain comparable.

It is thus evident that the relationship between the number of intermediates and the attack efficiency does not follow a simple linear rule. This is most likely because different intermediates are not entirely independent from each other and so they do not equally contribute additional information.

Table 2: Reduced key space when targeting the encryption function

(a) Targeting the 'basic' attack surface

Cipher \ Setsize	HW model					HD model				
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
AES	3	2^{10}	2^{20}	2^{23}	2^{25}	30	2^{15}	2^{22}	2^{25}	2^{26}
KLEIN	3	2^{9}	2^{12}	2^{18}	2^{23}	90	2^{15}	2^{22}	2^{24}	2^{26}
PRESENT	23	2^{11}	2^{19}	2^{23}	2^{25}	60	2^{15}	2^{22}	2^{24}	2^{25}
LED	2	2^{10}	2^{18}	2^{21}	2^{24}	35	2^{16}	2^{21}	2^{23}	2^{25}

(b) Targeting the 'maximum' attack surface

Cipher \ Setsize	HW model					HD model				
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
AES	1	1	2^{10}	2^{18}	2^{24}	1	1	2^{12}	2^{19}	2^{24}
KLEIN	1	1	2^{7}	2^{12}	2^{20}	1	1	2^{9}	2^{14}	2^{21}
PRESENT	1	1	2^{8}	2^{12}	2^{20}	1	1	2^{10}	2^{13}	2^{22}
LED	1	1	2^{5}	2^{11}	2^{19}	1	1	2^{7}	2^{13}	2^{20}

5 Attacking the Key Expansion

The key expansion algorithms are substantially different w.r.t. their diffusion properties. We hence briefly run through them in turn to explain what attack strategies are possible. Let the shorthand RK stand for round key. We use $\mathrm{RK}_i(j)$ for the j-th byte of the i-th round key.

Our principal contribution in this section is describing single trace attacks on the key schedule of KLEIN and PRESENT. We remind the reader that the first attack on the key schedule of AES (which we reproduce here as well, considering larger set sizes) has been described in [**?**]. As mentioned in Sect. 2, LED uses the same secret key for all encryption rounds.

5.1 Attack Strategies

AES. The particularities of AES make it possible to target 5-byte subkeys and a set of 5 consecutive round keys, as first described in [**?**]. Thus, the results that we report are on one round key (as in [**?**]) and on 5 rounds (as in [**?**]) considering sets of up to 5 values. Because of the properties

of the AES key expansion, attacks utilising all 10 round keys become computationally demanding for larger set sizes [**?**].

KLEIN. The KLEIN key schedule is relatively simple to attack. One can target 2-byte subkeys and use as many round keys as available (see Fig. 1). Thus, we list results for the attack utilising one round key, all round keys and half of the round keys.

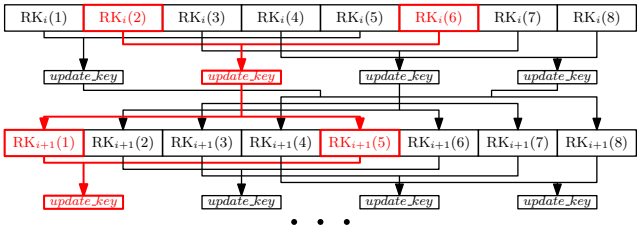

Fig. 1: Targeted KLEIN subkey

PRESENT. The key expansion of PRESENT is almost non-existent: each round key is derived from the previous via a cyclic shift of bits, a single application of the 4-bit S-box and an `xor`-ing with a round constant. Thus, reporting an attack on a single round key makes no sense, and we give the results on the full key schedule.

5.2 Attack Outcomes

Table 3 contains the outcomes of all attacks following the previously outlined attack strategies. We can observe that the diffusion properties, which impact on how much information from the key schedule we can incorporate given our computational abilities, play a significant role in attack outcomes. Consider AES for example: for larger set sizes we can only utilise the leakages from the first five round keys. Consequently the remaining key space is considerable, albeit much reduced. KLEIN in contrast is very vulnerable as we can effectively utilise all leakages across the key schedule and so we can tolerate high set sizes. PRESENT not only has a weak diffusion but also highly correlated intermediates in its key schedule and hence suffers much less from the lack of diffusion: it remains more resilient to attacks utilising leaks from the key expansion.

Table 3: Reduced key space when targeting the key expansion

(a) AES

# RK \ Set size	HW model					HD model				
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
1[**?**]	2^{58}	2^{74}	2^{95}	2^{106}	2^{115}	2^{60}	2^{75}	2^{99}	2^{107}	2^{118}
5	10	2^{15}	2^{35}	2^{58}	n.a.	30	2^{17}	2^{37}	2^{55}	n.a.
11[**?**]	1	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

(b) KLEIN

# RK \ Set size	HW model					HD model				
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
1	2^{35}	2^{45}	2^{50}	2^{57}	2^{60}	2^{40}	2^{48}	2^{55}	2^{57}	2^{61}
6	2^{8}	2^{15}	2^{35}	2^{45}	2^{55}	2^{12}	2^{21}	2^{37}	2^{49}	2^{57}
12	1	2^{4}	2^{20}	2^{32}	2^{45}	1	2^{4}	2^{22}	2^{37}	2^{50}

(c) PRESENT

# RK \ Set size	HW model					HD model				
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
31	2^{10}	2^{16}	2^{45}	2^{60}	2^{73}	2^{14}	2^{16}	2^{45}	2^{60}	2^{73}

6 Conclusion

In this paper we investigated, using pragmatic SPA attacks as an evaluation tool, how different lightweight ciphers compare with regards to their vulnerability against profiled single trace attacks. The aim was to tease out which of their properties, if any, have an influence on the efficiency of such attacks.

We found that for both the encryption round function and the key schedule the diffusion properties were decisive for attack success: the more reasonably statistically independent intermediate values occur in a concrete implementation, the better a profiled single trace attack could fare. This means that such attacks not only reduce the key space further for a subsequent brute force search, but also cope better with erroneous side-channel information i.e. they can tolerate larger set sizes. The fact that most lightweight ciphers feature a particularly lightweight key schedule

with little diffusion means that attacks can easily exploit the information from all round keys; this implies stronger attacks.

Acknowledgements V. Banciu has been supported by EPSRC via grant EP/H049606/1. E. Oswald and C. Whitnall have been supported in part by EPSRC via grant EP/I005226/1. The authors would like to thank the anonymous reviewers for the useful comments and suggestions.