On the Key Schedule of Lightweight Block Ciphers

Jialin Huang[1,2], Serge Vaudenay[1], and Xuejia Lai[2⋆]

[1] EPFL, Switzerland
[2] Shanghai Jiao Tong University, China
{serge.vaudenay}@epfl.ch, {xuejia.lai, jlhuang.cn}@gmail.com

Abstract. Key schedules in lightweight block ciphers are often highly simplified, which causes weakness that can be exploited in many attacks. Today it remains an open problem on how to use limited operations to guarantee enough diffusion of key bits in lightweight key schedules. Also, there are few tools special for detecting weakness in the key schedule.
In 2013 Huang et al. pointed out that insufficient actual key information (AKI) in computation chains is responsible for many attacks especially the meet-in-the-middle (MITM) attacks. Motivated by this fact, in this paper we develop an efficient (with polynomial time complexity) and effective tool to search the computation chains which involve insufficient AKI for iterated key schedules of lightweight ciphers. The effectiveness of this tool is shown by an application on TWINE-80.
Then, we formulate the cause of key bits leakage phenomenon, where the knowledge of subkey bits is leaked or overlapped by other subkey bits in the same computation chain. Based on the interaction of diffusion performed by the key schedule and by the round function, a necessary condition is thus given on how to avoid key bits leakage.
Therefore, our work sheds light on the design of lightweight key schedules by guiding how to quickly rule out unreasonable key schedules and maximize the security under limited diffusion.

Keywords: automatic tool, meet-in-the-middle, PRESENT, TWINE, actual key information

1 Introduction

Today the demands of secure communication on source constrained environments such as RFID tags and sensor networks motivate the development of many lightweight block ciphers. In these lightweight ciphers, the security margin that conventional block ciphers are equipped with is reduced as much as possible in order to optimize the software and hardware efficiency. One obvious sacrifice is that the key schedules are highly simplified for saving memory. Some key schedules have round-by-round

⋆ This work was supported by the National Natural Science Foundation of China (61073149 and 61272440 and 61472251).

iteration with low diffusion [20, 3, 21]. Some key schedules do simple permutation or linear operations on master keys [17, 10]. Some have no key schedule, and just use master keys directly in each round [9, 14]. These key schedules are succinct but responsible for many attacks, especially related-key attacks [18, 13], MITM attacks and their variants [19, 12, 5], and special attacks such as the invariant subspace attack on PRINTcipher [15].

Although causing so many risks, the key schedules in lightweight block ciphers are still designed in a heuristic and ad-hoc way, especially when a tradeoff between security and memory constraints should be made. Because of the lack of systematic guidelines, research on design principles of lightweight key schedules is pressing.

1.1 Related work

Avoiding MITM attacks and key bits leakage are two of the goals in the design of key schedules [16, 11]. Designers tend to exploit relatively fast diffusion or avalanche to achieve these goals, which is infeasible for lightweight key schedules. The resistance to MITM attacks is usually claimed by ensuring that all master key bits are used within several rounds. Huang et al. show that considering the diffusion of key bits in the "round" level is not enough by investigating the interaction of diffusion between the key schedule and the round function [11]. They also propose a measure called actual key information (AKI) to evaluate the effective speed of diffusing key bits and claim that a computation path should have as high AKI as possible.

From the other aspect, automatic tools have been developed to search cryptanalytic properties relevant to key schedules. Most of these tools focus on searching related-key differential characteristics [8, 1, 2], while the tools aiming at other weaknesses of key schedules are much fewer. Bouillaguet et al. propose automatic algorithms for searching guess-and-determine and MITM attacks on round-reduced AES, using the linear relations in the key schedule [4]. Derbez et al. give a way to automatically search the kind of Demirci-Selçuk MITM attacks against AES [7]. Huang et al. present a method to calculate the AKI for 2-round iterated key schedules whose key size is double of the block size [11].

1.2 Our contribution

While the concept of AKI that considers the minimal information of involved key bits in a computation path is meaningful, the algorithm described in [11] targets a too particular type of key schedules. Firstly, the master key size is double of the block size. Secondly, the number of searched rounds is fixed to two. Additionally, the time complexity of this 2-round search is exponential to the key size. If this algorithm is extended

to R rounds, the time complexity is exponential to not only the key size but also R, making practical applications infeasible.

Instead of computing AKI for conventional block ciphers, in this paper we consider computing AKI for lightweight block ciphers. We target iterated key schedules with low diffusion, as they exist in a certain amount of lightweight block ciphers. Computing AKI for the static or permutation-based key schedules is much easier, and thus not our concern here. We generalize the problem of computing AKI to any number of rounds in lightweight block ciphers. Then, we develop an efficient and effective algorithm to solve it (Sect. 3). Based on the observation of characters of lightweight iterated key schedules, we use a greedy strategy, resulting in an algorithm with polynomial time complexity. The tools in [7] focus on discovering a special kind of MITM attacks for AES (the Demirci-Selçuk attack that depends on algebraic relations in AES), while our tool aims to present a simple, efficient and generic approach to evaluate the design of iterated key schedules in lightweight ciphers. We show the feasibility of this algorithm by applying it to TWINE-80.

Although the key bits leakage has been mentioned in [11, 16], it does not achieve deserving attention in cipher design as other properties. This phenomenon is revisited in Sect. 4, and found in almost all the computation paths that contain insufficient information of key bits. We analyze the major cause of key bits leakage by explicitly relating the incidence matrix of the diffusion in key schedules to that of round functions. Furthermore, a formulated necessary criterion for key schedule design is proposed to guide to avoid key bits leakage within a given number of rounds. This necessary criterion can be a guidance of quickly ruling out unreasonable key schedules at first step, and then our tool can be used to do further examination.

2 Preliminary

2.1 Notations

The lightweight block ciphers studied here have a block size of n bits and a master key size of k bits. Each round function includes a key extraction and incorporation layer (AK), and a cipher permutation layer F that updates the n-bit internal state with confusion and diffusion after adding key bits. The output internal state in round i is denoted as s_i. The iterated key schedule updates each k-bit key state round-by-round from the previous one. That is, $k_i = \mathcal{KS}(k_{i-1})$, where k_i is the output *subkey* bits in the ith round of the key schedule, and k_0 is the master key. Normally $\mathcal{KS}$ is invertible, thus the size of all k_i is k. An rk-bit *round key* rk_i is extracted from k_i in the key extraction phase and added to the internal state. The most common case is to apply parts of the subkey k_i directly to the round key rk_i. Other notations include:

- $k_i[j]$ (resp. $s_i[j]$) denotes the jth bit of bit-string k_i (resp. s_i).
- M_s is the incidence matrix representing the diffusion of F. In the encryption direction, $M_s[j', j] = 1$ means that the $s_i[j']$ depends on $s_{i-1}[j]$, where $M_s[j', j]$ denotes the (j', j) entry in matrix M_s. In the decryption direction, $M_s[j', j] = 1$ means that $s_{i-1}[j']$ depends on $s_i[j]$.
- M_k is the incidence matrix for the diffusion of $\mathcal{KS}$, defined in a similar way with M_s.
- $\boldsymbol{e_i}$ is a vector with the ith bit equal to 1 and other bits equal to 0.
- $|\cdot|$ denotes the size of a set, or the Hamming weight of a bit-string, depending on the context.

2.2 Definitions

Definition 1. *Given an input set $\Delta_{input} \subset \{x_1,x_2,...,x_n\}$, and the set Δ_{sk} of all subkey bits that are generated from the master key, a computation chain*[3] *in a block cipher is a sequence o^1, o^2,...,o^h, in which each o^j is either an element of Δ_{input}, or the output of $f_j(o^{i1}, o^{i2}, ..., o^{ip}, \Delta_{jk})$, where $\Delta_{jk} \subset \Delta_{sk}$, for $1 \leq i1, i2, ..., ip < j$, and f_j depends on the cipher structure and round function. Consider $\Delta_{output} \subset \{o^j | j = 1, \ldots, h, o^j \notin \Delta_{input}\}$ as the output set of this chain.*

Definition 2. *([11, Definition 3]) According to a key schedule, the minimal amount of subkey bits from which one can derive all subkey bits in a computation chain is called actual key information (AKI).*

The n-bit $\{x_1,x_2,...,x_n\}$ could be n-bit plaintexts ($\mathcal{P}$) or ciphertexts ($\mathcal{C}$).

A forward computation chain has a calculation direction consistent with the encryption process, while the backward chain is consistent with the decryption process. The length of a chain is denoted as r if it covers r rounds of the block cipher. AKI($\mathcal{K}$) is the AKI of a set $\mathcal{K}$ of subkey bits. Normally, AKI($\mathcal{K}$) is the key length, but it could be smaller. We give a simple example to explain the above definitions.

In Fig.1[4], $n = k = 6$, $\Delta_{input} = \{s_0[0], s_0[1]\}$, and $\Delta_{output} = \{o^h\}$, where $o^h = s_4[1] = f_h(s_0[0], s_0[1], s_1[4], s_1[5], s_2[0], s_2[1], s_3[4], \Delta_{hk})$. Obviously, f_h is specifically defined by M_s. Also, $\Delta_{hk} = \{k_0[0],k_0[1],k_1[4],k_1[5],k_2[0], k_2[1],k_3[4]\}$. According to the key schedule (defined by M_k), with the knowledge of $\{k_0[0], k_0[1], k_0[2], k_0[3], k_0[5]\}$, we can deduce all 7 subkey bits in Δ_{hk}.

Obviously AKI is always not larger than k since the knowledge of all master key bits is enough to produce all subkey bits. For a static key schedule or a permutation-based key schedule, since each subkey is

[3] For convenient explanation in our context, we use this terminology similar but not the same as [11].

[4] In this example, we don't need to give the detailed description of F, and just need to concern its input, output and diffusion pattern M_s.

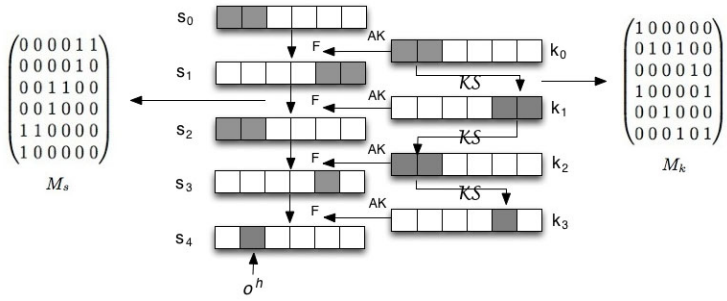


Fig. 1: The 4-round forward computation chain determined by M_s is marked by grey boxes, both in the round function and the key schedule. Here F represents a round transformation.

directly derived from some master key bits, it is relatively simple to find the AKI.

2.3 A brief description of meet-in-the-middle (MITM) attack

Among the risks caused by the insufficient AKI in a computation chain, the most serious and relevant ones are MITM attacks and their variants. A basic MITM attack uses the fact that the cipher can be decomposed into two consecutive parts and the computation of each part only involves partial master key (or its mapping). Then, a check point is calculated by each part independently, and used to filter a number of wrong key pairs.

Thus, for mounting an $(r_f + r_b)$-round MITM attack, we need at least: an r_f-round forward computation chain with $o^h = E_f(\Delta_f, \mathcal{K}_f)$ where $\Delta_f \subset \mathcal{P}$, and $\mathcal{K}_f$ is all involved subkey bits in this forward chain; an r_b-round backward computation chain with $o^h = E_b(\Delta_b, \mathcal{K}_b)$ where $\Delta_b \subset \mathcal{C}$, and $\mathcal{K}_b$ is all subkey bits in this backward chain. The total time complexity C_{comp} using N plaintext/ciphertext pairs is:

$$C_{comp} = \max(2^{\mathrm{AKI}(\mathcal{K}_f)}, 2^{\mathrm{AKI}(\mathcal{K}_b)}) \cdot N + 2^{\mathrm{AKI}(\mathcal{K}_f)+\mathrm{AKI}(\mathcal{K}_b)-N\cdot|o^h|-|\mathcal{K}_{common}|}$$

where $\mathcal{K}_{common}$ is the common key bits shared by $\mathcal{K}_f$ and $\mathcal{K}_b$. The MITM attack works faster than the brute force attack when C_{comp} is less than 2^k, which is the required computations for the brute force attack in the worst case.

3 Automatic Search Tool for Detecting Weakness on Iterated Key Schedules

3.1 AKI computation problem

The core of our algorithm is to calculate the AKI of one r-round computation chain. With this algorithm, all r-round computation chains are

examined (given a fixed r), so that the one with the least AKI can be found. And by repeating this algorithm with different r, the longest chain with AKI less than k can be determined.

Let V be the set of all subkey bits, and they are linked according to the diffusion of key schedules. Record all subkey bits involved in an r-round computation chain as a set $AV_r \subset V$. Our aim is **to find a set $BV_r \subset V$ such that the subkey bits in BV_r can derive all subkey bits in AV_r**, and **find the BV_r with the smallest size**. More precisely, BV_r should satisfy the following property: for each bit $b_i \in AV_r$, either $b_i \in BV_r$, or b_i can be reached by the bits in BV_r, where we define that a subkey bit can be reached iff all its parent subkey bits can be reached. $\mathrm{AKI}(AV_r)$ is equal to the size of the smallest BV_r.

AV_r is determined by the cipher structure, the round function and the key extraction phase. BV_r is decided based on AV_r and the key schedule. Our tool contains two phases: the precomputation phase and the online phase, which are described in the next two sections. All explanation is for the forward direction, and for the backward direction we only need to exchange the terms of i and $i-1$ in our analysis.

3.2 Precomputation

The precomputation phase determines AV_r of r-round computation chains. In a forward r-round chain, AV_r is the set of involved subkey bits (see $\mathcal{K}_f$ in Sect. 2.3), and decided by the diffusion of round functions (i.e., M_s) and the key extraction phase. Based on our representation of M_s, the direction of deducing AV_r is opposite to the direction of its corresponding computation chain. That is, a forward computation chain uses Δ_f (plaintext bits) and $\mathcal{K}_f$ as inputs to compute the output o^h, while in the process of computing AV_r, we start from the output o^h and date back round-by-round to find all related inputs of o^h so that $\mathcal{K}_f$ can be figured out. An r-round computation chain is obtained by multiplying M_s r times with an initial state determined by o^h. Note that in this way, what we obtain is the involved round key bits in a chain, and they should be traced back to the corresponding subkey bits in the key schedule according to the key extraction phase. For byte or nibble-oriented ciphers, one byte or one nibble can be represented with one bit.

For a fixed length r, there are 2^n possible initial states, hence corresponding to 2^n r-round computation chains. Normally, all these 2^n chains require to be checked to find the one with minimal AKI. However, in fact we only need to enumerate n computation chains based on the following observation: the minimum AKI will be found among the chains whose output are only one bit. Since the AV_r starting from one bit in round r are always subsets of the remaining, the initial states with only one active bit (i.e., o^h is one bit) always lead to computation chains with less AKI than the chains with more than one active bit in the initial states. In

practical attacks, exploitable chains are often among these n ones. The pseudocode that computes AV_r starting from the ith bit of the rth round internal state is as Algorithm 1, which will be called n times for different i, $i = 1, \ldots, n$. Hereafter we use $AV_r[j]$ ($AV_r[j_1..j_2]$) to represent the set of related subkey bits in the jth (from j_1th to j_2th) round of the r-round chain.

Algorithm 1 Compute AV_r

```
1: procedure ComputeAV_r(r, M_s, i)
2:     BlockState[r] = e_i;          ▷ Start from the initial state with the ith bit active.
3:     for j = r − 1 → 1 do          ▷ Deduce all related input bits round-by-round.
4:         BlockState[j] = BlockState[j + 1] × M_s; ▷ Derive the state bits in round j.
5:         AV_r[j] = KeyExtraction(BlockState[j]);   ▷ Relate round key to subkey.
6:     return AV_r;
```

3.3 Online phase

Given AV_r and M_k, the online phase looks for BV_r for AV_r. The simplest approach is to brute-force search all predecessors of all subkey bits in AV_r as well as themselves, and among these bits find the smallest subset that can derive AV_r. Using such method, the time complexity is exponential to the number of all bits, about $2^{r \cdot k}$. For a lightweight block cipher, if the diffusion in the round function is not very fast, the size of AV_r is limited by r to a great extent. Also, the low diffusion in the key schedule results in a sparse M_k. Thus, the exhaustive search can be pruned efficiently. However, the time complexity is still too high.

In practical situations, most of the time a discovery of weakness is already able to demonstrate insecurity of the design. Hence, instead of exhaustively searching to guarantee finding the optimal solution, we can relax our goal to find sub-optimal results which still can satisfy our requirements. This inspires us to exploit a greedy algorithm that makes the best choice at each step with the hope of approximating the optimum finally. It is easier to find a BV_r satisfying $|BV_r| \leq |AV_r|$ at each step. We expect that with our greedy strategy, the size of final BV_r is small enough, and always not larger than the size of AV_r and k. Thanks to the characters of iterated lightweight key schedules, our greedy method generates sub-optimal results, while consumes much less time compared with the brute-force searching. If the sub-optimal results we find are already enough to suggest weaknesses in the design, then the real situation of the design may be worse. Since the optimal AKI may indicate fewer key bits involved in a chain, which is even more undesirable for a secure design.

To obtain the final solution, we divide the problem into a series of subproblems, make a choice that is best at the moment, and then keep on applying the greedy strategy to the current result. The choice made each time depends on the choices made so far but not on the future choices. Our

greedy choice is based on the following assumption and characteristics of lightweight key schedules. For a forward computation chain:

- **Property 1.** The orientation of key information is along the encryption direction, i.e., from k_{i-1} to k_i. Thus, we assume that the knowledge of a bits of subkey k_i can bring about the knowledge of b bits of k_j iff $i < j$, and $a \geq b$[5].
- **Property 2.** $\mathcal{KS}$ is a bijective and invertible mapping, and each round of subkey has k bits information. A subkey bit cannot be derived from other subkey bits in the same round.
- **Property 3.** A majority of rows in M_k have Hamming weight 1, and the number of these rows is denoted as N_{one}. In most lightweight key schedules, N_{one} is close to k. The remaining tiny part of the rows in M_k have weight larger than 1, while still very small. Denote the maximal Hamming weight among all rows as MAX_{weight}.

Note that Property 1 is made for conveniently explaining our greedy strategy, instead of placing restrictions on the key schedules. Without this assumption, the results may be better but more time complexity is required, i.e., a smaller BV_r for the same AV_r may be found, since more relations in the key schedule can be utilized among the subkey bits. What we obtain with Algorithm 2 is in fact an upper bound of real AKI. If the upper bound of real AKI is insufficient, then real AKI is more insufficient.

We explain how our approach works (see also Algorithm 2) according to the general components required in a greedy algorithm. We start from the given AV_r, and each time we try to reduce the size of current AV_r while making sure that we still can derive all subkey bits in the original AV_r from it according to M_k. The subkey bits in the top round[6] of AV_r require no process due to Property 1. Thus, we only need to deduce the subkey bits within 2 to r rounds, denoting the number of these bits as n_a. At each step we trace the currently remaining n_a subkey bits as back as possible. That is, for the bits in round i of AV_r (i.e., $AV_r[i]$), we try to deduce them with the least amount of bits in round j, where $j < i$ and j should be as small as possible. If any subset of AV_r can be represented with fewer (then the size of AV_r is reduced) or equal subkey bits in former rounds[7], then we remove the original subset from AV_r and add these new bits of former rounds to AV_r. According to Property 1 and Property 2, deducing the same amount of subkey information to former rounds will increase the chance of leaking other subkey bits, while the total number of subkey bits in AV_r doesn't increase. The above process continues until no bits in the current state of AV_r can be deduced to former rounds without increasing $|AV_r|$. The final state of AV_r is the BV_r we want to find. In

[5] This is because of the diffusion direction we consider here.

[6] In an r-round forward chain, w.l.o.g., the top round is the round 1, and the bottom round is the round r.

[7] In a forward chain, round j is a former round of round i iff $j < i$.

Algorithm 2, the function GenNextCombination(num) is to generate a combination of num subkey bits from AV_r, and each time a subset of num subkey bits selected from the current n_a bits is considered.

Algorithm 2 Compute AKI

procedure ProcessAV$_r$(num) ▷ *Deduce num key bits to former rounds.*
  S = GenNextCombination(num); ▷ *S is the set of currently chosen num bits.*
  while S != NULL **do**
    MIN_{round} = ChooseMinRound(S);
    for $i = 1 \rightarrow MIN_{round}$ **do**
      $S[i]$ = DeduceS(i); ▷ *Deduce the bits in S to round i according to M_k.*
      $tmpAV_r[i] = AV_r[i] \cup S[i]$;
      if ($|tmpAV_r[i]|$ - $|AV_r[i]|$) $\leq num$ **then**
        $AV_r[i] = tmpAV_r[i]$; ▷ *Include the new bits in round i to AV_r.*
        $AV_r = AV_r$ - S; ▷ *Remove the original num bits from AV_r.*
        break;
    S = GenNextCombination(num);
  $n_a = |AV_r[2..r]|$; ▷ *Update the number of remaining bits to be processed.*
  return AV_r;

procedure ComputeAKI(MAX_{weight}, N_{one}, AV_r)
  $num = 1$; ▷ *Consider all subsets with size one at first.*
  while (AV_r is reducible) **do** ▷ *Process until AV_r is unchangable.*
    ProcessAV$_r$(num); ▷ *Select num bits from current $AV_r[2..r]$ to process.*
    Choose num according to n_a, MAX_{weight}, N_{one};
  $BV_r = AV_r$; ▷ *The final state of AV_r is what we want.*
  AKI = $|BV_r|$;
  return AKI;

Our method is suitable for both forward and backward directions by using corresponding matrices. We compute the AKI of all computation chains with the same length to find the minimal AKI.

3.4 The complexity

The memory complexity of matrix M_k is k^2. The time complexity to build n AV_r is $O(nr)$. The memory cost to store AV_r corresponds to the diffusion of the round function M_s. The total time complexity is dominant by the online phase. The main trick in our algorithm is that we process the subkey bits which can be derived by former one bit (i.e., the in-degree is one) at first. Since most bits in AV_r have in-degree one (Property 3), these subkey bits can be deduced to the top rounds directly. Hence, the number of remaining subkey bits that require processing (i.e., n_a) reduces greatly. Because of this sharp decrease of n_a, as well as limited MAX_{weight} in

lightweight key schedules, GenNextCombination(num) only enumerates the combinations with a time complexity polynomial to n_a. When deducing any combinations of current AV_r cannot help to reduce the size of AV_r, AV_r is non-reducible. The total time complexity to obtain the final AKI for one AV_r is polynomial to $n_a r^2$. Examples to explain the time complexity are given in Appendix B. Our results listed in the following sections are obtained within seconds on a PC.

3.5 Application to TWINE-80

To show the strength of our tool, we apply it to TWINE-80 [20] and mount a MITM attack based on the found results. The number of rounds we can reach is 14 rounds, while the designers confirmed that the subkeys for the first 3 rounds contain all 80 master key bits. This result is far from reaching the number of rounds of the best known attack in a single key model, which is a 23-round impossible differential attack. However, our attack is the first one with very low data complexity on TWINE-80. Indeed, as pointed out in [6, 4], it is important to determine the number of rounds that can be attacked with only a few available data, for a better security understanding. Since the attack is not our main concern in this paper and just an example to show the application of the tool, we refer to more details about TWINE-80 and our attack in Appendix A.

4 Key Bits Leakage

4.1 The least AKI in TWINE and PRESENT

As we mentioned before, low data complexity attack is an important research direction to make us understand the security of block ciphers in a "real-life" scenario. If a forward and a backward computation chain with insufficient AKI are discovered, then an adversary is likely to mount an MITM attack even the available data is restricted to only a few pairs of plaintext/ciphertext. Therefore, the least AKI among all forward (backward) chains provides a probable security bound for the low data complexity attack, particularly, the MITM attack. For example, if we know the smallest AKI is 58 bits in all 6-round forward chains, and the smallest AKI is 60 bits in all 6-round backward chains, then basically, we cannot mount any 12-round MITM attacks with time complexity less than about 2^{60}. Also, if the least AKI has reached to k bits after r rounds for both forward and backward directions, then the cipher is resistant against $2r$-round basic MITM attacks.

In this section, we concern the approximately smallest AKI given the number of rounds, for TWINE and PRESENT. Both of these two ciphers have 64 bits of block size, and 80/128-bit master key. For PRESENT, a 64-bit round key is extracted from the 80/128-bit subkey in each round,

Table 1: The least AKI of the forward computation chains in TWINE and PRESENT, and corresponding theoretical AKI.

	TWINE-128(nibbles)		PRESENT-128(bits)		TWINE-80(nibbles)		PRESENT-80(bits)	
round	theoretical	AKI	theoretical	AKI	theoretical	AKI	theoretical	AKI
1	0	0	4	4	0	0	4	4
2	1	1	20	20	1	1	20	20
3	2	2	84	77	2	2	80	64
4	4	4	128	125	4	4	80	80
5	7	7	128	128	7	6	80	80
6	12	11	128	128	12	10	-	-
7	19	17	-	-	19	15	-	-
8	27	23	-	-	20	18	-	-
9	32	27	-	-	20	19	-	-
10	32	30	-	-	20	20	-	-
11	32	31	-	-	-	-	-	-
12	32	32	-	-	-	-	-	-

while for TWINE a 32-bit (8 nibbles) round key is extracted from the 80/128-bit subkey in each round. Again, we use one bit to represent one nibble for TWINE.

Our search covers different rounds until the AKI is larger than k. The results are in Table 1. Details of some chains with the least AKI in Table 1 are given in Appendix B. The theoretical value of AKI can be computed by the diffusion of round function. Indeed, it is the total number of round key bits we expect to have in a computation chain, i.e., $|AV_r|$, and is not larger than k. Take Fig.1 as an example again, the number of involved round key bits (the grey boxes) in the chain is 7. However, all 6 bits of the master key should be enough to derive these 7 bits, which means that the theoretical AKI of this chain is 6 bits.

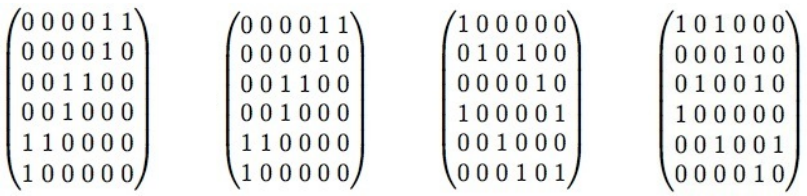
Remark. As we mentioned before, the real AKI of a chain could be less than what we find, thus our results in fact indicate a necessary but not sufficient security bound.

The authors in [11] mention three causes responsible for poor key schedules: the size of round keys is too small, no avalanche effect, and key bits leakage. However, no details about these causes are discussed carefully. In the next, we will focus on the key bits leakage.

4.2 Key bits leakage and three examples

We find that, almost all chains whose practical AKI are less than the theoretical one have a particular phenomenon, where the knowledge of subkey bits are leaked by some other subkey bits in the same chain. This phenomenon is denoted as *key bits leakage*.

Besides the structure of the cipher, for iterated key schedules the interplay of diffusion patterns between key schedules and round functions plays a crucial role on this phenomenon. In this section, we use three examples to demonstrate different levels of key bits leakage caused by different interaction of M_s and M_k. Recall that M_s and M_k are the incidence matrices of diffusion in round functions and key schedules, respectively.



(a) M_s (b) Case 1: M_k (c) Case 2: M_k (d) Case 3: M_k

Fig. 2: The M_s and different M_k of these three cases.

Assume that we already have the round transformation for a block cipher, and now we need to compare the security of three candidate key schedules that have the same level of diffusion (i.e., to some extent they have the same efficiency). Here, we use the same cipher structure as Fig.1. The block size is equal to the key size and every internal state bit is XORed by one bit of the round key. All these three cases share the same M_s, as in (a) of Fig.2, and the three key schedules correspond to different M_k as in (b), (c) and (d) of Fig.2. They have the same level of diffusion: three subkey bits are respectively derived from two subkey bits of the previous round, and the remaining three subkey bits only depend on one bit in the previous round. Therefore, a common property in three M_k is that they all have three rows with weight 2 and three rows with weight 1.

The first case (replace the M_k in Fig.1 with (b) of Fig.2) is a typical and extreme example for key bits leakage, where M_s and M_k are identical. Recall that according to the diffusion of round function F (decided by M_s), a 4-round computation chain for calculating $s_4[1]$ (i.e., we can use $\boldsymbol{e_1}$ to represent the involved and non-involved bits of s_4 in this chain) is marked by the grey boxes. That is, $\boldsymbol{k_0} = \boldsymbol{e_1} \cdot M_s^4 = (1, 1, 0, 0, 0, 0)$, $\boldsymbol{k_1} = \boldsymbol{e_1} \cdot M_s^3 = (0, 0, 0, 0, 1, 1)$, $\boldsymbol{k_2} = \boldsymbol{e_1} \cdot M_s^2 = (1, 1, 0, 0, 0, 0)$, $\boldsymbol{k_3} = \boldsymbol{e_1} \cdot M_s = (0, 0, 0, 0, 1, 0)$. The vector $\boldsymbol{k_i}$ represents the involved and non-involved subkey bits with 1 and 0, respectively. M_s^4 is M_s to the power of 4. The involved subkey bits are: $k_0[0]$, $k_0[1]$, $k_1[4]$, $k_1[5]$, $k_2[0]$, $k_2[1]$ and $k_3[4]$, 7 bits in total. Thus, what we expect is that 6 bits of the master key are all exploited in this 4-round chain. However, $\boldsymbol{k_3} \cdot M_k^3 = \boldsymbol{k_0}$, $\boldsymbol{k_2} \cdot M_k^2 = \boldsymbol{k_0}$, $\boldsymbol{k_1} \cdot M_k = \boldsymbol{k_0}$. This reveals the situation far from our anticipation: once

$k_0[0]$ and $k_0[1]$ are known, the remaining five key bits are not required to guess. Therefore, this computation chain actually contains only two key bits, and the AKI is 2. The ratio between AKI and the theoretical value (6 bits) now is $\frac{1}{3}$, meaning that only 33.3% of the key bits in this chain are effective.

Above is the worst case of key bits leakage in a block cipher. Apparently, this is a bad example for cipher design when M_s and M_k have incomplete diffusion. More generally, we should avoid constructing homologous M_s and M_k in lightweight block ciphers, since the low diffusion will make the consequence of key bits leakage worse.

For the second case, similarly we compute AKI according to M_k in (c) of Fig.2 and find that all 4-round computation chains have 5 bits of AKI. Again for the third case, the AKI of all 4-round chains are 6 bits, meaning that the whole key space is covered now. The security of these two cases is much better than the first case. Thus, we can conclude that not only the amount of diffused bits but also the positions of diffused bits can affect the number of actual key information in a computation chain. With the same amount of diffusion operations, some diffusing patterns will cause severe key bits leakage, while others will not.

5 How to Avoid Key Bits Leakage

In practical design, we hope that key bits leakage does not occur in any round and the AKI can be guaranteed as large as possible in every chain. Hereafter, in our multiplication between matrices and vectors, bit-addition is OR and bit-multiplication is AND. The sum of binary matrices or vectors, denoted as $\sum$, is a bitwise OR of these vectors or matrices.

5.1 The cause of key bits leakage

After examining the computation chains with least AKI in Table 1, as well as other chains having much lower AKI compared to the theoretical value, we find that almost all of them have the following property: for the computation chain starting from $\boldsymbol{e_j}$, we can always derive subkeys in this chain from fewer key bits in the top round[8]. That is,

$$|\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}| < \sum_{i=1}^{r} |\boldsymbol{e_j} \cdot M_s^i|. \tag{1}$$

Recall that $|\cdot|$ is the Hamming weight of the binary vector, M_s^i is the matrix of M_s to the power of i, the sum of weight ($\sum$ in the right side) is a sum over integers. The same notations are used hereafter.

[8] In a forward computation chain, the top round is the round with the smallest index. In a backward computation chain, the top round is the round with the biggest index.

Also, the number of finally involved key bits in the top round that can derive all subkey bits in the chain is less than the master key size. i.e.,

$$|\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}| < k. \tag{2}$$

For example, in Table 1, Eq.(1) and Eq.(2) hold for the 6 to 11 rounds of TWINE-128. The theoretical value is $\min(\sum_{i=1}^{r} |\boldsymbol{e_j} \cdot M_s^i|, k)$. Thus, in these rounds,

$$\text{AKI} \leq |\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}| < \min(\sum_{i=1}^{r} |\boldsymbol{e_j} \cdot M_s^i|, k), \tag{3}$$

where the ratio of AKI and the theoretical value is smaller than 1.

Note that $\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}$ is just the upper bound of AKI, since we may be able to find a set of fewer subkey bits from different rounds (instead of only from the top round) that also can be used to compute all key bits in the chain. However, the first equality sign in Eq.(3) can be obtained for most of the cases we checked for PRESENT and TWINE. This means that the smallest set of subkey bits that can derive all key bits in the chain is selected from the top round. There are two probable reasons responsible for this. Firstly, a large number of rows have weight 1 in M_k so that most subkey bits can directly date back to previous rounds without diffusing. Also, even for the remaining small quantity of rows having weight larger than 1, these weights are low and sometimes neighboring (e.g. the key schedule of PRESENT).

5.2 Necessary condition for avoiding key bits leakage

Based on the above cause, we explicitly formulate a necessary condition for avoiding key bits leakage, i.e., gaining the maximal AKI in each computation chain. The tool in Sect. 3 can compute the AKI for each chain and check the ratio. However, this only can be done after the key schedule has been designed. Here we give a design principle only in terms of M_s and M_k, so that we can use it as a guideline when designing the key schedule for a known M_s. For convenience and w.l.o.g., we take the case that M_s and M_k have a equal size k again. In the case where they have different sizes, M_s just needs to be merged or converted according to the key extraction phase.

Corollary 3. *For a block cipher, if there is no key bits leakage within R rounds, then, $\forall r \in [2 \dots R]$ and $\forall j \in [0 \dots k-1]$*

$$\begin{cases} |\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j \geq \sum_{i=1}^{r} |M_s^i|_j, \text{ if } \sum_{i=1}^{r} |M_s^i|_j < k \\ |\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j = k, \qquad \text{if } \sum_{i=1}^{r} |M_s^i|_j \geq k \end{cases} \tag{4}$$

where $|\cdot|_j$ represents the Hamming weight of the jth row of the matrix.

Proof. In order to prove that Eq.(4) is a necessary condition for avoiding key bits leakage, we need to prove that once Eq.(4) doesn't hold, there is key bits leakage within R rounds.

We discuss the case that $\sum_{i=1}^{r} |M_s^i|_j < k$ firstly. When Eq.(4) doesn't hold, then $\exists r$, $\exists j$ so that $|\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j < \sum_{i=1}^{r} |M_s^i|_j$. Since $\boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}$ is equal to the jth row of $M_s^i \cdot M_k^{r-i}$, and $\boldsymbol{e_j} \cdot M_s^i$ is equal to the jth row of M_s^i, hence this r and j satisfies that $|\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}| < \sum_{i=1}^{r} |\boldsymbol{e_j} \cdot M_s^i|$, which is the same as Eq.(1). This means that, considering the rounds from $R-r+1$ (resp. $R+r-1$) to R in a forward (resp. backward) chain which computes the jth bit of round R, the involved subkey bits between these rounds can leak each other through some subkey bits in round $R-r+1$ (resp. $R+r-1$).

The second case is that $\sum_{i=1}^{r} |M_s^i|_j \geq k$. Note that since M_k and M_s are $k \times k$ matrices, $|\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j$ is always not larger than k, for all j. Hence, when Eq.(4) doesn't hold, then $\exists r$, $\exists j$ so that $|\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j < k$. Similarly, this r and j satisfies that $|\sum_{i=1}^{r} \boldsymbol{e_j} \cdot M_s^i \cdot M_k^{r-i}| < k$. In this case, we can find that the r-round chain, which computes the jth bit in the beginning round, contains fewer than k bits of AKI. However, this chain should have contained k bits of AKI. □

Note that in a computation chain that calculates the jth bit of the internal state, indeed AKI can be smaller than $|\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j$, i.e., $|\sum_{i=1}^{r} M_s^i \cdot M_k^{r-i}|_j$ only provides an upper bound for AKI. This is why Corollary 3 is not a sufficient condition for key bits leakage.

Based on Corollary 3, we can design diffusion of a lightweight key schedule in a more targeting way when the round function has been decided, so that a computation chain is able to diffuse as many key bits as possible even though the number of diffusion operations is limited.

6 Conclusion

In this paper, we develop a effective and efficient tool to automatically search AKI in iterated key schedules of lightweight block ciphers. Also, we formulate the cause of key bits leakage phenomenon from the point of the relation of incidence matrices that represent diffusion of round functions and key scheduling. Based on this cause, a necessary condition on how to avoid key bits leakage in design of lightweight key schedules is given, which can be a guidance of quickly ruling out unreasonable key schedules and maximizing security under low diffusion. In further research we will consider using our algorithm and the necessary condition to examine more designs of lightweight key schedules.

References

1. Alex Biryukov and Ivica Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In *EUROCRYPT 2010*, volume 6110, pages 322–344. 2010.
2. Alex Biryukov and Ivica Nikolić. Search for Related-Key Differential Characteristics in DES-Like Ciphers. In *Fast Software Encryption*, volume 6733, pages 18–34. 2011.
3. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, pages 450–466, 2007.
4. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In *CRYPTO 2011*, volume 6841, pages 169–187. 2011.
5. Özkan Boztaş, Ferhat Karakoç, and Mustafa Çoban. Multidimensional Meet-in-the-Middle Attacks on Reduced-Round TWINE-128. In *Lightweight Cryptography for Security and Privacy*, volume 8162, pages 55–67. 2013.
6. Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-Middle: Improved MITM Attacks. In *CRYPTO*, pages 222–240, 2013.
7. Patrick Derbez and Pierre-Alain Fouque. Exhausting Demirci-Selcuk Meet-in-the-Middle Attacks against Reduced-Round AES. In *FSE 2013*. 2013.
8. Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128. In *CRYPTO 2013*, volume 8042, pages 183–203. 2013.
9. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917, pages 326–341. 2011.
10. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In *CHES*, volume 4249, pages 46–59. Springer, 2006.
11. Jialin Huang and Xuejia Lai. Revisiting Key Schedule's Diffusion in Relation with Round Function's Diffusion. *Designs, Codes and Cryptography*, pages 1–19, 2013.
12. Takanori Isobe and Kyoji Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. volume 7372, pages 71–86. 2012.
13. Jérémy Jean, Ivica Nikolić, Thomas Peyrin, Lei Wang, and Shuang Wu. Security Analysis of PRINCE. In *FSE 2013*. 2013.
14. Lars Knudsen, Gregor Leander, Axel Poschmann, and MatthewJ.B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *CHES 2010*, volume 6225, pages 16–32. 2010.
15. Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In *CRYPTO*, pages 206–221, 2011.
16. Lauren May, Matt Henricksen, William Millan, Gary Carter, and Ed Dawson. Strengthening the Key Schedule of the AES. volume 2384, pages 226–240. 2002.
17. Roger M. Needham and David J. Wheeler. TEA Extensions. Report, Cambridge University, Cambridge, UK, October 1997.
18. Onur Özen, Kerem Varici, Cihangir Tezcan, and Çelebi Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. volume 5594, pages 90–107. 2009.

19. Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In *CT-RSA 2011*, volume 6558, pages 250–267. 2011.
20. Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In *Selected Areas in Cryptography*, volume 7707, pages 339–354. 2013.
21. Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In *Applied Cryptography and Network Security*, volume 6715, pages 327–344. 2011.

Appendix

A A MITM attack on 14 rounds of TWINE-80

TWINE is a lightweight block cipher using a variant of Type-2 generalized Feistel scheme. TWINE-80 iterates 36 rounds to encrypt 64-bit block message with 80-bit master key. We refer to [20] for more details about this cipher.

Here we choose similar notations as in [20], $X^i = X^i_0||X^i_1||\dots||X^i_{14}||X^i_{15}$ and $X^{i+1} = X^{i+1}_0||X^{i+1}_1||\dots||X^{i+1}_{14}||X^{i+1}_{15}$ as the input and output of round i, and $P = X^1$. RK^i is the round key used in round i, extracted from WK^i, where WK^i is the key state after running i rounds of the key schedule. Note that $WK^0_0||\dots||WK^0_{19}$ is the master key, and $RK^i = WK^{i-1}_1||WK^{i-1}_3||WK^{i-1}_4||WK^{i-1}_6||WK^{i-1}_{13}||WK^{i-1}_{14}||WK^{i-1}_{15}||WK^{i-1}_{16}$.

Here we use a forward computation chain which computes X^8_5 from the plaintext (X^1), and a backward chain computing X^8_5 from the ciphertext (X^{15}). The AKI of this forward chain is 17 nibbles, i.e., $\mathcal{K}_f$ can be decided by all nibbles in WK^0 except WK^0_{10}, WK^0_{11} and WK^0_{12}. The AKI of the backward chain is 18 nibbles, that is, $\mathcal{K}_b$ can be known from WK^{12}_4, and all nibbles in WK^{13} except WK^{13}_0, WK^{13}_7 and WK^{13}_{12}. Note that according to the key schedule, after guessing all nibbles in WK^{13} except WK^{13}_0, WK^{13}_7 and WK^{13}_{12}, 10 nibbles in WK^0 can be known (WK^0_0, WK^0_1, WK^0_3, WK^0_6, WK^0_8, WK^0_{10}, WK^0_{11}, WK^0_{13}, WK^0_{15}, WK^0_{19}). Thus, these two computation chains share 8 nibbles of key information, which means that we can filter 32 bits of information from the key. The attack process is as Algorithm 3. We use 9 pairs of plaintext/ciphertext, providing a 36-bit filter from the internal state.

The time complexity of the meeting phase is $9 \cdot 2^{18 \cdot 4} \cdot 0.5 = 2^{74.17}$ encryptions. The number of remaining candidate keys is $2^{(17+18)\cdot 4-32-36} = 2^{72}$. Thus, 2^{72} trivial encryptions are needed in the search phase. The total time complexity is about $2^{74.46}$ encryptions of 14-round TWINE-80. For simplicity, we omit the cost of memory access for finding a match in T_1, assuming that the time complexity of one table look-up is negligible compared with that of one encryption. This assumption is quite natural and reasonable in most cases. However, strictly speaking, those costs should be considered. The memory complexity is $2^{17 \cdot 4} = 2^{68}$ blocks.

Algorithm 3 The 14-round MITM attack on TWINE-80

1: **for** each possible value of 17 nibbles for $\mathcal{K}_f$ **do**
2: Encrypt $P_{(i)}$ to get $X^8_{5(i)}$, $i = 1 \dots 9$.
3: Store corresponding key values for $\mathcal{K}_f$ in T_1 indexed by $X^8_{5(1)}|| \dots ||X^8_{5(9)}$.
4: **for** each possible values of 18 nibbles for $\mathcal{K}_b$ **do**
5: Decrypt $C_{(i)}$ to get $X^8_{5(i)}$, $i = 1 \dots 9$.
6: **if** this $X^8_{5(1)}|| \dots ||X^8_{5(9)}$ exists in T_1 **then**
7: Take the corresponding key values for $\mathcal{K}_f$ as well as current key value for $\mathcal{K}_b$ as candidate keys.
8: Exhaustively search each remaining candidate key.

B Details of computation chains in Table 1

For TWINE-128, o^h is the output nibble of the chain, $s_i[j]$ here is the jth nibble of the internal state in round i, and $k_i[j]$ ($k_i[j_1 - j_2]$) is the jth (j_1 to j_2) output subkey nibble(s) in the ith round of key schedule. For PRESENT-128, the results are bit-oriented. The subkey bits (nibbles) in BV_r are enough to compute the output o^h of the r-round chain.

Table 2: The BV_r corresponding to TWINE128 for $r = 7, 8, 9$ in Table 1.

round	the output o^h	BV_r	AKI
7	$s_7[10]$	$k_5[23]$,$k_6[2-3]$,$k_6[8-15]$,$k_6[17]$,$k_6[22]$, $k_6[24-25]$,$k_6[28]$,$k_6[28]$,$k_6[31]$	17
8	$s_8[14]$	$k_6[23]$,$k_7[0]$,$k_7[2-4]$,$k_7[7-15]$,$k_7[17-18]$, $k_7[22]$,$k_7[24-25]$,$k_7[27-28]$,$k_7[30-31]$	23
9	$s_9[4]$	$k_8[0]$,$k_8[2-3]$,$k_8[5-15]$,$k_8[17-18]$,$k_8[20]$,$k_8[22-31]$	27

Table 3: The BV_r corresponding to PRESENT128 for $r = 3, 4$ in Table 1.

round	the output o^h	BV_r	AKI
3	$s_3[3]$	$k_2[51-127]$	77
4	$s_4[0]$	$k_2[3-127]$	125

For PRESENT-128, 120 rows of M_k have weight 1, and 8 rows have weight 4. We consider the 4-round chain with the least AKI, which is 125 bits. The size of original AV_4 is 148 bits, and n_a is 84. After we deduce all one in-degree cases, n_a decreases to 4. Then we can quickly deduce these 4 bits 2-by-2, 3-by-3, and 4-by-4.