New Slide Attacks on Almost Self-Similar Ciphers

Orr Dunkelman[1], Nathan Keller[2], Noam Lasry[2], and Adi Shamir[3]

[1] Computer Science Department, University of Haifa, Israel
orrd@cs.haifa.ac.il
[2] Department of Mathematics, Bar-Ilan University, Israel
nkeller@math.biu.ac.il,noam.lasry@gmail.com
[3] Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Israel
adi.shamir@weizmann.ac.il

Abstract. The slide attack is a powerful cryptanalytic tool which has the unusual property that it can break iterated block ciphers with a complexity that does not depend on their number of rounds. However, it requires complete self similarity in the sense that all the rounds must be identical. While this can be the case in Feistel structures, this rarely happens in SP networks since the last round must end with an additional post-whitening subkey. In addition, in many SP networks the final round has additional asymmetries – for example, in AES the last round omits the MixColumns operation. Such asymmetry in the last round can make it difficult to utilize most of the advanced tools which were developed for slide attacks, such as deriving from one slid pair additional slid pairs by repeatedly re-encrypting their ciphertexts. Consequently, almost all the successful applications of slide attacks against real cryptosystems (e.g., FF3, GOST, SHACAL-1) had targeted Feistel structures rather than SP networks.

In this paper we overcome this "last round problem" by developing four new types of slide attacks. We demonstrate their power by applying them to many types of AES-like structures (with and without linear mixing in the last round, with known or secret S-boxes, with periodicity of 1,2 and 3 in their subkeys, etc). In most of these cases, the time complexity of our attack is close to $2^{n/2}$, which is the smallest possible complexity for slide attacks. Our new slide attacks have several unique properties: The first attack uses *slid sets* in which each plaintext from the first set forms a slid pair with some plaintext from the second set, but without knowing the exact correspondence. The second attack makes it possible to create from several slid pairs an exponential number of new slid pairs which form a hypercube spanned by the given pairs. The third attack has the unusual property that it is always successful, and the fourth attack can use known messages instead of chosen messages, with only slightly higher time complexity.

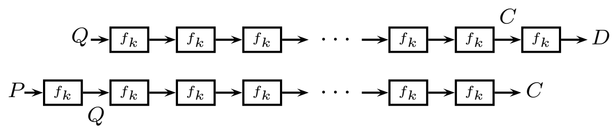

Fig. 1. A Slid Pair

1 Introduction

Most modern block ciphers are constructed as a cascade of r keyed components, called rounds. Each round by itself can be cryptographically weak, but as r increases, the scheme becomes resistant against almost all the standard cryptanalytic attacks (e.g., differential cryptanalysis [9], linear cryptanalysis [28], Square attack [13]). However, there is one type of attack called a *slide attack* (introduced in 1999 by Biryukov and Wagner [10][1]) which can handle an arbitrarily large number of rounds with the same complexity.

The original slide attack targets ciphers that are a cascade of r *identical* rounds, i.e.,

$$E_k = f_k^r = f_k \circ f_k \circ \cdots \circ f_k,$$

and tries to find a *slid pair* of plaintexts (P, Q) such that $Q = f_k(P)$, as demonstrated in Fi. 1. Due to the structure of E_k, the corresponding ciphertexts $C = E_k(P), D = E_k(Q)$ must satisfy $D = f_k(C)$. Hence, if a slid pair (P, Q) is given, the adversary can use the simplicity of f_k to solve the system of equations:

$$\begin{cases} Q = f_k(P), \\ D = f_k(C), \end{cases} \tag{1}$$

and thus to retrieve the secret key k.

The adversary can start from any collection of $O(2^{n/2})$ plaintexts along with their ciphertexts, and consider their $O(2^n)$ pairs. One of them is likely to be a slid pair, but the adversary does not know which one it is. By trying to solve the system of equations (1) for all the pairs, she gets a simple slide attack whose data complexity is $O(2^{n/2})$ known plaintexts, memory complexity is $O(2^{n/2})$ (which is used to store the data), and time complexity is $O(t \cdot 2^n)$ (where t is the time required for solving the system (1)).

1.1 Applicability of Slide Attacks to Modern Ciphers

The original slide attack can be used only when f_k is so simple that it can be broken efficiently using only two known input/output pairs. Subsequent papers

[1] We note that the slide attack is related to several previous techniques, including the attack of Grossman and Tucherman on Feistel constructions [25] and Biham's related-key attack [7].

(e.g., [5,11,15,17,19]) developed advanced variants of the slide attack that allow attacking self-similar constructions in which f_k is rather complex. A central observation used in many of these variants is that if (P, Q) is a slid pair, then $(E_k(P), E_k(Q))$ is also a slid pair, and thus the adversary can create from a single slid pair arbitrarily many additional *friend pairs* by repeatedly encrypting P and Q in an adaptively chosen message attack. These advanced variants made it possible to attack various generic forms of Feistel constructions with a periodic key schedule, such as constructions with 1-round [10], 2-round [10], 3-round [5] and 4-round [11] self similarity. Furthermore, they allowed obtaining practical attacks on several real life cryptosystems – most notably, breaking the block cipher Keeloq [2] and the 128-bit key variant of the block cipher GOST [5], and attacking several hash functions [21].

While the advanced slide attacks extended the applicability of the technique, the basic requirement that all the round functions must be exactly the same has remained. As a result, it seemed that slide attacks can be thwarted completely by inserting into the encryption process round constants that break the full symmetry between the rounds. This countermeasure has become standard and is applied in most modern block ciphers.

However, it turned out that round constants are not an ultimate solution, as in many cases improper choice of the constants or interrelation between the round constants and other components of the cipher, can be used to mount a slide attack despite the countermeasure.

A recent example is the well known format preserving encryption scheme FF3 [12]. This scheme was selected as a US standard by NIST in 2016, but had to be withdrawn in 2017 due to a devastating slide attack by Durak and Vaudenay which was announced at CRYPTO 2017 [18]. This happened even though the designers of the cryptosystem (Brier, Peyrin and Stern) are all highly experienced cryptographers who were fully aware of the danger posed by slide attacks and used the standard countermeasure of using different round constants to avoid them. An even more recent example is Lilliput-AE [1], a candidate of the NIST lightweight competition. While the cipher uses round constants, a problem in the tweakey schedule made it possible to find slid pairs that effectively reduce the number of rounds from 32 to 16. Yet another example is the block cipher SHACAL-1 that was broken by Biham et al. [8] using a slide attack, although it used round constants. It thus turns out that while adding round constants may be a useful countermeasure, it is far from being a universal countermeasure, and slide attacks remain highly relevant in practice.

1.2 Slide Attacks on SP Networks

Most of the previously known slide attacks, including the attacks on FF3, GOST, Lilliput-AE and SHACAL1 described above, apply to Feistel constructions. The other major type of a block cipher, the Substitution-Permutation (SP) Network, cannot be directly attacked by a slide attack since its last round is always different from the other rounds.

Consider, for example, an AES-like structure in which each round consists of XORing a subkey (which we denote by K), applying a parallel layer of S-boxes (denoted by S), and linearly mixing their outputs (denoted by A). Assume in addition that two subkeys are used in the cyclic order $(k_1, k_2, k_1, k_2, \ldots, k_2)$. Simply composing these round functions makes no sense since the last layers of S-boxes and linear mapping are known and can thus be stripped off. Any sensible design must thus add to the last round a final post-whitening subkey, such as k_1, before outputting the ciphertext. This makes the last round different from all the other rounds, and we cannot simply complete the construction into a self similar structure by applying to it $A \circ S \circ K \circ A \circ S$ since we do not know the subkey k_2 used in K. A similar situation arises when the S-boxes are secret or when the last round operation A (before the post-whitening key) is different than in previous rounds. Any such asymmetry suffices in order to destroy the crucial property that if two plaintexts (P, Q) are a slid pair then so are their ciphertexts $(E_k(P), E_k(Q))$, upon which many advanced slide attacks rely. In fact, the only advanced slide attack on SP networks published so far, by Bar-On et al. [5, Sect. 2.2, 2.3] on an AES-like cipher with a 2-round or 3-round self-similarity, applies only under a non-standard additional assumption on the structure of the cipher.[2]

1.3 Our Settings

In this paper we overcome the *last round problem* by developing new slide attacks that can be applied to SP-networks with an arbitrarily large number of rounds in which the last round is different from the previous rounds. To be concrete, we consider ciphers that can be viewed as a cascade

$$K \circ A \circ S \circ K \circ A \circ S \circ \ldots K \circ A \circ S \circ K,$$

where K denotes the XORing of a secret subkey, S is a non-linear operation (S-box) applied in parallel to sub-blocks of size s of the state, and A is an affine operation. We call this structure KSA, and say that it has an ℓ-round self-similarity if the subkeys have the periodic structure $k_1, k_2, \ldots, k_\ell, k_1, k_2, \ldots, k_\ell, \ldots$. We denote such a structure by ℓ*-KSA*.

We note that extremely simple key schedules, and in particular periodic key-schedules with a short period, are widely used in modern lightweight block ciphers, for the sake of saving place on the hardware taken by the key schedule mechanism. Examples include LED-64 [26], Zorro [20], PRINTcipher [27], CGEN [29] and MIDORI128 [4] (which have identical subkeys), LED-128 [26], CRAFT [6] and MIDORI64 [4] (which have period 2), and many others.

Of course, designers of most modern block ciphers protect the ciphers against slide attacks by adding round constants in order to destroy the self-similarity. However, as was mentioned above, this countermeasure is not always sufficient. In

[2] The additional assumption (that was not mentioned in [5]) is that either the final key whitening step is omitted, or the number of rounds is *odd* (for 2-round self similarity) or of the form $3\ell + 2$ (for 3-round self-similarity).

addition, some lightweight cryptosystems have a large number of simple rounds, and XOR'ing a different randomly generated constant to each round greatly increases the amount of memory required to implement the scheme, which is very undesirable in many IoT applications. Consequently, designers of such cryptosystems may be tempted to use other forms of asymmetry into their designs, such as using a different last round, but as we show in this paper, such a simple countermeasure can be defeated by new variants of slide attacks.

We study two types of KSA constructions: The first is *ℓ-KSAf*, composed of a sequence of rounds of the form $A \circ S \circ K$ with an ℓ-periodic key, augmented by a final key whitening – which is the structure of many modern SP networks. The second type is *ℓ-KSAt*, which differs from ℓ-KSAf by omission of the affine operation A in the last round. Such a change is performed in some block ciphers for implementation reasons — most notably, in the AES.

We usually assume that the operations S, A are not key-dependent (like in AES). However, interestingly, some of our new attacks apply with only a small complexity overhead when S is key-dependent (like in the AES variant with a secret S-box studied in [22,24,31,32]). We denote the block size by n and the S-box size by s, and state our results in terms of the parameters n, s.

1.4 Our Contributions

We present four entirely new types of slide attacks, which solve the last round problem in four different ways:

Slid sets. In this attack, we attach to each candidate slid pair (P, Q) a pair of *sets* $\mathcal{T}_P = \{P_1, P_2, \ldots, P_d\}$ and $\mathcal{T}_Q = \{Q_1, Q_2, \ldots, Q_d\}$ such that for each i there exists j for which (P_i, Q_j) is a slid pair. That is, the set $\mathcal{T}_P$ is transformed into the set $\mathcal{T}_Q$, while we do not know what is the counterpart of each specific value in $\mathcal{T}_P$. Of course, this technique requires entirely different ways to solve the equation system (1), and we provide such techniques as well.

Hypercube of slid pairs. This technique first uses differential properties of the cipher to attach to each candidate slid pair (P, Q) a pair of d-tuples $\mathcal{T}_P = (P_1, P_2, \ldots, P_d)$ and $\mathcal{T}_Q = (Q_1, Q_2, \ldots, Q_d)$ such that with some unexpectedly high probability, each (P_i, Q_i) is a slid pair. Then, it uses a 'mixing' construction reminiscent of the recently proposed *mixture* attack [23] to leverage the d-tuples into 2^d-tuples of slid pairs. Roughly speaking, if the slid pairs are placed at d vertices of a d-dimensional hypercube, the technique allows us to attach to them $2^d - d$ additional slid pairs which are placed at all other vertices of the cube.

Suggestive plaintext structures. This attack uses two plaintext structures $\mathcal{T}_P$ and $\mathcal{T}_Q$, designed in such a way that the mere knowledge that some $P \in \mathcal{T}_P$ has a slid counterpart $Q \in \mathcal{T}_Q$ reveals significant key information, which is used in the solution of the equation system (1). An interesting feature of this attack is that while its data complexity is $3 \cdot 2^{n/2}$, which is only slightly more than the $2^{n/2}$ complexity of standard slide attacks, it has 100% success probability. Note that the success probability of standard slide attacks is about 63%; it can be increased by using more data, but cannot get to 100% success unless the data complexity is made extremely large.

Substitution slide. This attack is aimed at truncated ℓ-KSA constructions, in which in the equation system (1), the second equation is much more complex than the first one. We use substitution into the (easier) first equation in order to remove the key dependence from the (harder) second equation and transform it into an even more complex equation which depends only on plaintexts and ciphertexts and not on the key. This attack type applies even in the more restrictive (and more realistic, of course) known plaintext model.

1.5 Our Results

Here are a few concrete results that can be obtained with our new slide attacks (the full summary can be found in Tab. 1):

1. Using the *suggestive plaintext structures* technique, we can break 1-KSAt (e.g. a variant of AES with identical round subkeys and with no MixColumns operation in the last round) with data and time complexity of $2^{n/2}$ (2^{64} in the special case of AES). In [5], Bar-On et al. presented an attack with the same complexity, but only on 1-KSAf, or equivalently, AES in which the MixColumns operation in the last round is not omitted.
2. Using *substitution slide*, we can break 1-KSAt with complexity of $2^{(n+4s)/2}$ known plaintexts and time (2^{80} in the special case of AES).
3. Using *slid sets*, we can break 2-KSAt (e.g., a variant of AES with 2-periodic round subkeys and with no MixColumns operation in the last round) with data and time complexity of $2^{(n+3s)/2}$ (2^{76} in the specific case of AES).

Organization of the Paper

In Section 2 we present the setting and notations used throughout the paper, as well as some preliminary steps that are routinely performed in all our attacks. In addition, we present the previous attack by Bar-On et al. [5] on 1-KSA. In Section 3 we present the *slid sets* technique and use it for attacking several constructions (e.g., 2-KSAf and 1-KSA with secret S-boxes). Section 4 presents the new *hypercube of slid pairs* technique and presents an attack on 1-KSA with secret S-boxes. The *suggestive plaintext structures* technique is presented in Section 5. We introduce the *substitution slide* in Section 6. Several of our attacks are presented in the Appendix. Finally, Section 7 concludes the paper.

2 Preliminaries

In this section we present the setting and notations that are used throughout the paper, and describe the slide attack of Bar-On et al. [5] on SPNs with a 1-round self similarity, which provides a simple example of the attack frameworks that we use in this paper.

Cipher	Technique	Complexity (general)		AES-like	
		Data/Memory	Time	Data/Memory	Time
Known S-Boxes					
1-KSAf	Slide [5]	$2^{n/2}$ (KP)	$2^{n/2}$	2^{64} (KP)	2^{64}
2-KSAf	Slide [5]*	$s \cdot 2^{s+n/2}$ (ACPC)	$s \cdot 2^{s+n/2}$	2^{69}	2^{69}
3-KSAfi †	Slide [5]**	$2^{(m+n)/2}$ (ACPC)	$2^{(m+n)/2}$	2^{81}	2^{81}
1-KSAt	Suggestive str. (Sect. 5)	$3 \cdot 2^{n/2}$ (CP)	$4 \cdot 2^{n/2}$	$2^{65.6}$ (CP)	2^{66}
1-KSAt	Sub. slide (Sect. 6)	$2^{n/2}$ (KP)	$2^{3n/4}$	2^{64} (KP)	2^{96}
2-KSAf	Slid sets (Sect. 3)	$2^{(n+s)/2+1}$ (CP)	$2^{(n+s)/2+1}$	2^{69} (CP)	2^{69}
2-KSAf	Slide + Key Guessing (App. B)	$(n/s)2^{n/2}$ (CP)	$2^{n/2+s}$	2^{68} (CP)	2^{72}
2-KSAf	Slide + Pt/Ct Coll. (App. C)*	See App. C for full details		$2^{82\ddagger}$ (KP)	2^{82}
2-KSAtpi †	Slid sets (App. A)	$2^{(n+m)/2+1}$ (CP)	$\max\{2^{(n+m)/2+1}, 2^{2m}\}$	2^{78} (CP)	2^{78}
3-KSAfi †	Slid sets (App. A)	$2^{(n+m)/2+1}$ (CP)	$\max\{2^{(n+m)/2+1}, 2^{2m}\}$	2^{81} (CP)	2^{81}
Secret S-Boxes					
1-KSAf	Slid sets (Sect. 3)	$1.17\sqrt{s}2^{(n+s)/2}$ (CP)	$1.17\sqrt{s}2^{(n+s)/2}$	$2^{70.3}$ (CP)	$2^{70.3}$
1-KSAf	Hypercube (Sect. 4)	$\sqrt{s}2^{n/2+s(s+3)/4+1}$ (CP)	$\sqrt{s}2^{n/2+s(s+3)/4+1}$	2^{88} (CP)	2^{88}

The exact definition of all variants is given in Section 2.1
KP – Known Plaintext; CP – Chosen Plaintext; ACPC – Adaptive Chosen Plaintext and Ciphertext
For AES-like $n = 128, s = 8$
† – this version has incomplete diffusion layer, m denotes the "word" size of the linear operation.
‡ – the memory complexity of this attack is 2^{47}.
* – this attack works for an odd number of rounds.
** – this attack works when the number of rounds is 1 mod 3.

Table 1. Summary of our new results

2.1 Setting and notations

While the attacks presented in the paper target many different constructions and use different techniques, they are all presented using a uniform setting and set of notations. All these notations are given and explained in this section.

The general structure of the ciphers we study. Throughout the paper, we consider a block cipher $E : \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^n$, which transforms an n-bit plaintext P into an n-bit ciphertext C, using a κ-bit key k. For the sake of simplicity, we assume that $\kappa = n$, but the results can be easily adapted for other values of κ. We assume that the cipher is iterative, that is, consists of a composition of r simpler functions, called rounds. All the attacks we present are applicable with the same complexity to an arbitrarily large number of rounds.[3]

We assume that the first $r-1$ rounds of the cipher have the standard general structure of an SPN, that is,

$$(A \circ S \circ K)^{r-1} = A \circ S \circ K \circ A \circ S \circ \ldots K \circ A \circ S \circ K,$$

[3] The attacks presented in Appendices B and C depend on the residue of r modulo the period of the subkey sequence, but not on the number of repetitions. All other attacks are independent of the number of rounds.

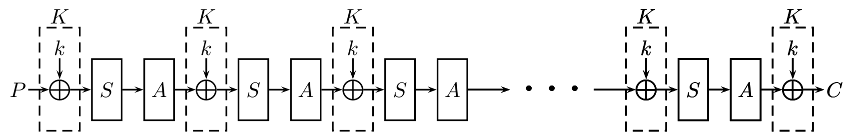

Fig. 2. The Structure of 1-KSAf

where K denotes key addition, S denotes a non-linear operation (S-box) applied in parallel to words of s bits into which the state is partitioned, and A denotes an affine operation. As the cipher essentially consists of repetitions of the sequence of operations $A \circ S \circ K$, we name it KSA.

The structure of the last round. Regarding the last round, we study two types of constructions:

– *Full last round* constructions, in which a single key addition operation is appended at the end of the last round. That is,

$$\text{Full } r\text{-round KSA } = (K \circ A \circ S \circ K) \circ (A \circ S \circ K)^{r-1} = K \circ (A \circ S \circ K)^r.$$

  This structure is exemplified in Fig. 2.
– *Truncated last round* constructions, in which a key addition is appended at the end of the last round, and in addition, the last round affine transformation A is omitted. That is,

$$\text{Truncated } r\text{-round KSA } = (K \circ S \circ K) \circ (A \circ S \circ K)^{r-1}.$$

The first type corresponds to a generic SPN construction, while the second type corresponds to an AES-like construction, as removal of the last round affine operation is adopted in the AES design.[4]

The structure of the operations K, S, A. In addition to the last round, the constructions we study differ in the assumptions on the operations K, S, A:

– *Key addition:* We shall always assume that the operation K in round i denotes XOR with an n-bit round subkey k_i, where the sequence of subkeys $k_1, k_2, k_3, \ldots$ is periodic. We study the variants 1-KSA, 2-KSA, and 3-KSA, in which the length of the period is 1,2, and 3, respectively. We assume that all subkeys are derived from the n-bit secret key K using some "sufficiently

[4] We note that in AES, only part of the last round affine layer is omitted. Namely, the MixColumns operation is omitted, while the ShiftRows operation is left unchanged. While maintaining the ShiftRows operation affects the complexity of some attacks that exploit the key schedule (just like the omission of MixColumns, see [16]), it has no effect on our attacks. Hence, for the sake of this paper, the design of the last round of AES is equivalent to removing the entire affine layer.

complex" function; hence, we never exploit relations between distinct subkeys, and at the same time, we aim for attacks of complexities lower than 2^n, as otherwise, the attack is slower than exhaustive key search. (We note that such an assumption on the key schedule algorithm is made in many papers analyzing the security of generic constructions; see, e.g., [3]).

– *The S-box layer S:* We shall always assume that the operation S consists of partition of the state into s-bit words and parallel application of the same function $S : \{0,1\}^s \rightarrow \{0,1\}^s$ to the blocks. We study two types of constructions: the standard type in which the S-box S is publicly known (like in AES), and the secret S-box type in which S is derived from the secret key using a complex function, and thus, is unknown to the adversary (like in Twofish [30] and in the variants of AES studied in [22,24,31,32]). In both types of constructions, we do not exploit the specific structure of the S-box.
– *The affine layer A:* We consider two variants of the operation A. In the *complete diffusion* variant, A applies a publicly known affine transformation to the entire state (i.e., the state is viewed as an n-bit vector v, and is transformed into $A'v + w$, where A' is an n-by-n binary matrix, $w \in \{0,1\}^n$, and the operations are performed over Z_2). In the *incomplete diffusion* variant, the state is partitioned into several parts (e.g., 4 parts in the case of AES), and the same affine transformation A is applied to each of them in parallel. In this variant, we introduce an additional parameter, m, to denote the size of each part (e.g., 32 bits in AES).

Summary of types of constructions. To summarize, the constructions we consider are defined by four parameters:

1. The length of the key period (1,2, or 3);
2. Type of the last round – *full* (only a key addition appended) or *truncated* (key addition appended and affine operation removed);
3. Type of the substitution layer S – *public* S-box or *secret* S-box derived from the secret key;
4. Type of the affine layer – *complete diffusion* (i.e., A acts on the entire state) or *incomplete diffusion* (i.e., A acts on several parts of the state in parallel).

Notation of types of constructions. The notation we use for the constructions reflects all four parameters: the number at the beginning is the length of the key period, then the letter 'f' or 't' says whether the last round is full or truncated, then the letter 'p' or 's' denotes whether the S-box is public or secret, and finally, the letter 'c' or 'i' denotes whether the diffusion is complete or incomplete. If some parameter is not included (e.g., neither 'p' nor 's' appear), this means that the attack applies to both types described by that parameter.

For example, 2-KSAfpi denotes KSA with a 2-round key period, full last round, public S layer and incomplete diffusion. Similarly, 1-KSAtc denotes KSA with a 1-round key period, truncated last round, and complete diffusion, where the omission of 'p' and 's' means that the corresponding attack works for both public and secret S-boxes.

Notation of data sets and slid pairs. In all the attacks proposed in this paper, the data consists of two sets of plaintexts/ciphertext pairs. The first set is called $\mathcal{T}_{\mathcal{P}}$ and its elements are denoted by $(P_i, C_i)_{i=1,2,\ldots,d}$. The second set is called $\mathcal{T}_{\mathcal{Q}}$ and its elements are denoted by $(Q_i, D_i)_{i=1,2,\ldots,d'}$. The corresponding sets of ciphertexts are denoted by $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{D}}$, respectively. The slid pairs are always of the form (P_i, Q_j), for some i, j.

If the considered variant is ℓ-KSA, then a pair (P_i, Q_j) of plaintexts is called a *slid pair* if $(A \circ S \circ K)^{\ell}(P_i) = Q_j$. If the cipher was completely self-similar like in standard slide attacks, this would guarantee that the corresponding pair of ciphertexts (C_i, D_j) satisfies $(A \circ S \circ K)^{\ell}(C_i) = D_j$. In our case, the relation depends on whether the considered ℓ-KSA construction is full or truncated. If (P_i, Q_j) is a slid pair, we call Q_j *the slid counterpart* of P_i.

In some of our attacks, in order to save data complexity we use the same plaintext set $\mathcal{T}$ both as $\mathcal{T}_{\mathcal{P}}$ and as $\mathcal{T}_{\mathcal{C}}$. In such cases, we use both notations $\mathcal{T}_{\mathcal{P}}$ and $\mathcal{T}_{\mathcal{Q}}$ for $\mathcal{T}$, and in each candidate slid pair, we denote the 'left' element by $P_i \in \mathcal{T}_{\mathcal{P}}$ and the 'right' element by $Q_j \in \mathcal{T}_{\mathcal{Q}}$. In this context, it is worth noting that for any X, Y, the pairs (X, Y) and (Y, X) are distinct candidates for a slid pair, since the equations $(A \circ S \circ K)^{\ell}(X) = Y$ and $(A \circ S \circ K)^{\ell}(Y) = X$ are not equivalent.

Modification of the plaintexts and the ciphertexts. In all our attacks, we consider a pair of plaintexts (P_i, Q_j) for which we want to decide whether it is a slid pair or not, and study the relation between P_i and Q_j, and the relation between the corresponding ciphertexts C_i and D_j. In order to simplify these relations, we would like to "remove" unkeyed operations that can be computed in advance for all plaintexts/ciphertexts in the data set. There are two types of operations we can remove: the first is operations that can be precomputed directly, and the second is operations that can be precomputed after interchanging the order of the operations K and A.

Let us exemplify this modification process on a concrete example. In 1p-KSA, for each slid pair (P_i, Q_j), we have

$$Q_j = A \circ S \circ K(P_i),$$

or equivalently, $S^{-1} \circ A^{-1}(Q_j) = K(P_i)$. The left hand side $S^{-1} \circ A^{-1}(Q_j)$ can be computed in advance for any plaintext Q_j. We thus replace each $Q_j \in \mathcal{T}_{\mathcal{Q}}$ by $Q'_j = S^{-1} \circ A^{-1}(Q_j)$, and work with the simplified equation

$$Q'_j = K(P_i).$$

Furthermore, the corresponding ciphertexts, (C_i, D_j), satisfy

$$D_j = K \circ A \circ S(C_i),$$

(or equivalently, $A^{-1} \circ K^{-1}(D_j) = S(C_i)$. The right hand side $S(C_i)$ can be computed in advance for any ciphertext C_i. As for the left hand side, note that by distributivity, for every binary matrix A' and binary vectors x, w, k the following holds: $(A'x + w) + k = A'(x + (A')^{-1}k) + w$. Hence, we can always interchange the order of the operations A, K, at the expense of replacing the subkey k in the operation K with $(A')^{-1}k$, where A' is the matrix used in the operation A. Thus, we have $A^{-1} \circ K^{-1}(D_j) = (K')^{-1} \circ A^{-1}(D_j)$, where K' denotes addition of the key $A'k$. The value $D'_j = A^{-1}(D_j)$ can be computed an advance for any ciphertext D_j. Thus, we replace each $C_i \in \mathcal{T}_\mathcal{C}$ with $C'_i = S(C_i)$ and each $D_j \in \mathcal{T}_\mathcal{D}$ with $D'_j = A^{-1}(D_j)$, and work with the simplified equation

$$D'_j = K'(C'_i).$$

Notations for modified plaintexts and ciphertexts. We perform such a change routinely, whenever there is an unkeyed operation that can be performed in advance (including cases where one has to interchange the order of the operations K, A). We use the notation $\bar{P}_i, \bar{C}_i$ to say that such a modification was performed to P_i, C_i (respectively), and the notation $\tilde{Q}_j, \tilde{D}_j$ to say that such a modification was performed to Q_j, D_j (respectively). Note that the exact modification differ between different variants of KSA.

We denote the sets of modified values that correspond to $\mathcal{T}_\mathcal{P}, \mathcal{T}_\mathcal{C}, \mathcal{T}_\mathcal{Q}$, and $\mathcal{T}_\mathcal{D}$ by $\bar{\mathcal{T}}_P, \bar{\mathcal{T}}_C, \tilde{\mathcal{T}}_Q$, and $\tilde{\mathcal{T}}_D$, respectively. We abuse notation and call the pair $(\bar{P}_i, \tilde{Q}_j)$ a *slid pair* whenever the corresponding pair (P_i, Q_j) is a real slid pair.

2.2 AES Notations

As the best-known prototype of the constructions we consider is AES, we shall present all our attacks in the special case of an AES-like construction with a periodic key schedule, and then we will briefly explain how do these attacks apply for generic ℓ-KSA constructions. Hence, for the sake of convenience, we briefly recall the structure of AES.

The structure of AES. The Advanced Encryption Standard (AES) [14] is an SPN that supports key sizes of 128, 192, and 256 bits. A 128-bit plaintext is treated as a byte matrix of size 4x4, where each byte represents a value in $GF(2^8)$. An AES round, depicted in Fig. 3, applies four operations to the state matrix:

- SubBytes (SB) — applying the same 8-bit to 8-bit invertible S-box 16 times in parallel on each byte of the state,
- ShiftRows (SR) — cyclically shifting the i'th row by i bytes to the left,
- MixColumns (MC) — multiplication of each column by a constant 4x4 matrix over the field $GF(2^8)$, and
- AddRoundKey (ARK) — XORing the state with a 128-bit subkey.

Before the first round, an additional AddRoundKey operation takes place. This operation allows us to "redefine" an AES round as starting with an AddRoundKey operation, with the last round AddRoundKey operation serving as a post-whitening key. In the last round of AES, the MixColumns operation is omitted. The number of rounds depends on the key size, and ranges between 10 and 14.

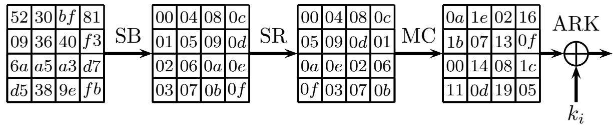

Fig. 3. An AES round

Notations for the variants of AES we study. Since we use AES-like constructions as a prototype of general KSA constructions, their types and notations are similar to the types of KSA constructions discussed above. Namely, in all variants we consider, the key schedule is replaced by a periodic key schedule, with a period of 1,2, or 3. Following [5], we denote by ℓK-AES a variant with period ℓ in the key schedule. We call the variant *truncated* if in its last round, the MixColumns operation is removed (like in original AES), and otherwise, we call the variant *full*.[5] We say that the S-box is *public* if it is publicly known (like in AES), and say that it is *secret* if it is key-dependent (like in the variants of AES studied in [22,24,31,32]). The diffusion of the affine layer in AES (namely, $MixColumns \circ ShiftRows$) is inherently incomplete, and so we use ℓK-AES as a prototype only for KSA constructions with *incomplete diffusion*; constructions with complete diffusion are treated separately.

Like for general KSA constructions, the notation we use for AES-like constructions reflects the three relevant parameters: the number at the beginning is the length of the key period, then the letter 'f' or 't' says whether the last round is full or truncated, and then the letter 'p' or 's' denotes whether the S-box is public or secret. If some parameter is not included (e.g., neither 'p' nor 's' appear), this means that the attack applies to both types described by that parameter. (Note that the letters 'c' or 'i' are irrelevant in the case of AES as explained above, and so are always omitted.) For example, 3K-AESts denotes a variant of AES with a 3-round key period, no MixColumns operation in the last round, and secret S-boxes.

Notations for intermediate values in AES. We denote the bytes of the state matrix of AES by $0, 1, 2, \ldots, 15$, in the order described in Fig. 3, and denote the value of the i'th byte of a state x by x_i. When several bytes $i_1, \ldots, i_\ell$ are considered simultaneously, they are denoted $x_{\{i_1,\ldots,i_\ell\}}$. The columns are numbered $0, 1, 2, 3$; the j'th column of the state x is denoted by $x_{\mathrm{Col}(j)}$, and if several columns are considered simultaneously, we denote them by $x_{\mathrm{Col}(j_1,\ldots,j_\ell)}$. Sometimes we are interested in 'shifted' columns, i.e., the result of the application of ShiftRows to a set of columns. This is denoted by $x_{SR(\mathrm{Col}(j_1,\ldots,j_\ell))}$. Similarly, a set of 'inverse shifted' columns (i.e., the result of the application of SR^{-1} to a set of columns) is denoted by $x_{SR^{-1}(\mathrm{Col}(j_1,\ldots,j_\ell))}$.

[5] We note that in [5], the notation ℓK-AES was used for a variant with a MixColumns operation in the last round (unlike AES), and the variant with no MixColumns in the last round was not considered.

Algorithm 1 A Slide Attack on 1-KSAf [5]

Initialize an empty hash table T.
Ask for the encryption of a set $\mathcal{T}$ of $2^{n/2}$ known plaintexts.
for each plaintext/ciphertext pair (P_i, C_i), where $P_i \in \mathcal{T}$ **do**
  Compute the value $\bar{C}_i = A \circ S(C_i)$,
  Compute the value $P_i \oplus \bar{C}_i$,
  Store in T the value $(P_i \oplus \bar{C}_i, P_i)$.
for each plaintext/ciphertext pair (Q_j, D_j), where $Q_j \in \mathcal{T}$ **do**
  Compute the value $\tilde{Q}_j = S^{-1} \circ A^{-1}(Q_j)$,
  Compute the value $\tilde{Q}_j \oplus D_j$,
  if $\tilde{Q}_j \oplus D_j$ is the first coordinate of an entry $(P_i \oplus \bar{C}_i, P_i) \in T$ **then**
    Test the key candidate $k = P_i \oplus \tilde{Q}_j$ by trial encryption.

2.3 The attack of [5] on 1-KSAf

Bar-On et al. [5] considered 1-KSAf, that is, $E = K \circ (A \circ S \circ K)^r$ where all operations K use the same key k. They showed that this variant can be broken with probability of about 63%, given $2^{n/2}$ known plaintexts, and roughly the same amount of time and memory.

The idea behind the attack is simple. Assume that (P_i, Q_j) is a slid pair, i.e., that $A \circ S \circ K(P_i) = Q_j$. Denoting $\tilde{Q}_j = S^{-1} \circ A^{-1}(Q_j)$, we have

$$P_i \oplus \tilde{Q}_j = k. \tag{2}$$

On the other hand, by the structure of E, the corresponding ciphertexts (C_i, D_j) must satisfy $D_j = K \circ A \circ S(C_i)$. Thus, denoting $\bar{C}_i = A \circ S(C_i)$, we have

$$D_j \oplus \bar{C}_i = k. \tag{3}$$

Combining Eq.(2) and (3), we get

$$P_i \oplus \bar{C}_i = \tilde{Q}_j \oplus D_j. \tag{4}$$

This relation allows one to mount the attack described in Algorithm 1. Note that the data used in the attack consists of a single set $\mathcal{T}$ of $2^{n/2}$ known plaintexts. As was described above, this single set is treated both as $\mathcal{T}_\mathcal{P}$ and as $\mathcal{T}_\mathcal{Q}$, and when we consider a candidate slid pair composed of two elements of $\mathcal{T}$, we denote it by (P_i, Q_j) and denote the corresponding ciphertexts by C_i, D_j.

As the data set contains $2^{n/2} \cdot (2^{n/2} - 1) \approx 2^n$ pairs, the probability that the data set contains a slid pair, i.e., a pair that satisfies $Q_j = A \circ S \circ K(P_i)$, is about $1 - (1 - 2^{-n})^{2^n} \approx 0.63$. Each slid pair leads to a collision in the table which suggests the right key candidate. On the other hand, for a random pair (P_i, Q_j), the probability that $P_i \oplus \bar{C}_i = \tilde{Q}_j \oplus D_j$ is 2^{-n}, and thus, only a few collisions in the table are expected. Thus, the right key can be found easily by going over all collisions in the table and checking the values of k they suggest. The data complexity of the attack is $2^{n/2}$ known plaintexts, its time and memory complexities are about $2^{n/2}$ operations, and its success probability is 63%.

In addition to the attack described above, Bar-On et al. presented a memoryless variant of the attack, based on classical cycle detection algorithms. The attack requires $2^{n/2}$ adaptively chosen plaintexts, $2^{n/2}$ time, and a negligible amount of memory.

3 The Slid Sets Attack

In this section we present a new cryptanalytic technique, the *slid sets attack*, and use it to attack 2-KSAfp with complexity $O(2^{(n+s)/2})$ and 1-KSAs with complexity $O(\sqrt{s} \cdot 2^{(n+s)/2})$. In particular, our attack allows us to break an AES-like cipher with secret S-boxes and the same round keys with complexity of $2^{70.3}$ – only slightly higher than 2^{64}, which is a natural lower bound for the complexity of a slide attack on a 128-bit cipher.

The key idea behind the slide sets technique is to consider pairs of plaintext sets $U = \{P_i\}_{i=1,\ldots,d}$ and $V = \{Q_j\}_{j=1,\ldots,d}$, such that if for some (i_0, j_0), (P_{i_0}, Q_{j_0}) is a slid pair, then the entire set V is the *slid counterpart* of the entire set U, in the sense that for any $P_i \in U$, there exists $1 \leq j \leq d$ such that the slid counterpart of P_i is Q_j. Interestingly, we will not be able to know (until the very end of the attack) which Q_j is the counterpart of a specific P_i. This attack paradigm stands in contrast with all previously known slide attacks which treated either single slid pairs or slid tuples $(P_1, \ldots, P_d), (Q_1, \ldots, Q_d)$ in which each Q_i is the slid counterpart of P_i.

We begin with presenting the attack in the special case of 2-KSAf, where its application is the simplest one. Then we show the more complex attack on 1-KSAs. Even more complex attacks on 2-KSAtpi and on 3-KSAfpi are given in Appendix A.

3.1 Slid sets attack on 2-KSAf

The setting. For the sake of helping readability, we present the attack in the special case of 2K-AESfp (i.e., an AES-like cipher with 2-round periodic subkeys, publicly known S-boxes, and with a MixColumns operation in the last round). We assume that the number of rounds is *even*; it will be apparent from the attack that it applies to the 'odd' case without change. First, we would like to simplify the problem.

Assume that (P_i, Q_j) is a slid pair. This means that

$$Q_j = MC \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i),$$

where ARK_ℓ denotes key addition with the subkey k_ℓ. As was described in Section 2.1, we can peel off unkeyed operations by denoting $\tilde{Q}_j = SR^{-1} \circ MC^{-1} \circ SB^{-1} \circ SR^{-1} \circ MC^{-1}(Q_j)$, and obtain the equation

$$\tilde{Q}_j = ARK'_2 \circ SB \circ ARK_1(P_i), \tag{5}$$

where ARK'_2 denotes the addition of the subkey $MC^{-1} \circ SR^{-1}(k_2)$. By the basic slide property, the relation between the corresponding ciphertexts (C_i, D_j)

is similar to the relation between the plaintexts (but of course, is not the same, due to the last round asymmetry). Namely, we have

$$D_j = ARK_1 \circ MC \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB(C_i).$$

Like with the plaintexts, this relation can be simplified to

$$\tilde{D}_j = ARK_1' \circ SB \circ ARK_2(\bar{C}_i), \tag{6}$$

where $\bar{C}_i = MC \circ SR \circ SB(C_i)$, $\tilde{D}_j = MC^{-1} \circ SR^{-1}(D_j)$, and ARK_1' denotes the addition of the subkey $MC^{-1} \circ SR^{-1}(k_1)$ The important gain from obtaining the simplified equations is that now the transformation from P_i to $\tilde{Q}_j$ consists of application of 16 independent functions on the bytes of the state, and the same goes for the transition from $\bar{C}_i$ to $\tilde{D}_j$. This plays a significant role in the attack.

Construction of candidate slid sets. The idea behind this step is as follows. Let $(P_i, Q_j), (P_{i'}, Q_{j'})$ be slid pairs, and let $\tilde{Q}_j, \tilde{Q}_{j'}$ be computed from $Q_j, Q_{j'}$, as defined above. The fact that the transformation from P_i to $\tilde{Q}_j$ consists of application of 16 independent functions on the bytes of the state, implies that if $P_{i'}$ differs from P_i only in a single byte, then $\tilde{Q}_{j'}$ differs from $\tilde{Q}_j$ only in a single byte as well.

We observe that this property can be generalized from pairs to sets, as follows. Consider two sets $U = \{P_i\}$, $\tilde{V} = \{\tilde{Q}_j\}$ which form Λ-sets (see [13]) with respect to byte 0 of the state, i.e., each of them is a set of 256 values that are equal in all S-boxes but S-box 0, and attains all possible values in S-box 0. (Of course, the same can be performed with another byte instead of byte 0.) Let $V = \{Q_j\}$ be the plaintext set obtained from $\tilde{V}$ by setting $Q_j = MC \circ SR \circ SB \circ MC \circ SR(\tilde{Q}_j)$ for each $\tilde{Q}_j \in \tilde{V}_j$. By the above property, if the slid counterpart of some $P_i \in U$ is $Q_j \in V$, then any $P_{i'} \in U$ has a slid counterpart $Q_{j'}$ in V. We call two sets of plaintexts U, V that satisfy this property (namely, that each element of U has a slid counterpart in V and vice versa) *slid sets*.

The same process can be performed in the converse direction: Each candidate slid pair (P_i, Q_j) suggests a pair of *slid sets* (U, V), by defining U to be a Λ-set that contains P_i, defining $\tilde{V}$ to be a Λ-set that contains $\tilde{Q}_j$, and computing V from $\tilde{V}$ as described above. (Of course, we have to make sure that the permuted byte in the Λ-set is the same byte.) Importantly, we do not know which element in V is the slid counterpart of a given element of U; we only know that this counterpart exists in V, if indeed the original pair (P_i, Q_j) is a slid pair.

The attack is based on collecting sufficiently many pairs of sets (U, V), such that with a high probability the data contains a pair of slid sets. Then, the question is how to find the slid sets among them.

Identifying the slid sets. Let (U, V) be a candidate pair of slid sets. Let $W = \{C_i\}$ be the set of ciphertexts corresponding to the plaintexts of U, and let $X = \{D_j\}$ be the set of ciphertexts corresponding to the plaintexts of V. Define the sets $\bar{W}$ and $\tilde{X}$ by setting $\bar{C}_i = MC \circ SR \circ SB(C_i)$ for any $C_i \in W$ and

$\tilde{D}_j = MC^{-1} \circ SR^{-1}(D_j)$ for any $D_j \in X$. If (U, V) are slid sets, then for each $\bar{C}_i \in \bar{W}$, there exists $\tilde{D}_j \in \tilde{W}$ such that Eq. (6) holds for the pair $(\bar{C}_i, \tilde{D}_j)$. However, we have to check many combinations of U and V, and even if we know that (U, V) are slid sets, we do not know which Q_j corresponds to which P_i.

Luckily, the relation (6) consists of applying 16 independent functions on the bytes of the state. This implies that in each byte separately, for each pair $C_{i_1}, C_{i_2} \in W$, the equality $\bar{C}_{i_1} = \bar{C}_{i_2}$ holds if and only if the equality $\tilde{D}_{j_1} = \tilde{D}_{j_2}$ holds for some $D_{j_1}, D_{j_2} \in X'$ (though, we still do not know for which values!). Consequently, the statistic: "how many values are attained q times in byte ℓ" is preserved between the sets $\bar{W}$ and $\tilde{X}$, for any byte ℓ and any multiplicity!

This can be used for obtaining a significant amount of filtering, in the following way. We pick sufficiently many Λ-sets U^l (all with the same permuted byte), and for each corresponding $\bar{W}^l$, for each byte ℓ, we compute the sequence of multiplicities (i.e., the sequence which records: how many values are not obtained, how many are obtained once, etc.), defined formally by

$$a_q^\ell = \left| \left\{ v \in \{0, 1, \ldots, 255\} : \left| \left\{ \bar{C}_i \in \bar{W}^l : (\bar{C}_i)_\ell = v \right\} \right| = q \right\} \right|,$$

and store the sequence-of-sequences $(a_q^\ell)_{\ell=0,1,\ldots,15, q=0,1,\ldots}$ in a hash table. Then, we pick sufficiently many Λ-sets $\tilde{V}^l$, and for each corresponding V^l, we look at the ciphertext structure X^l corresponding to V^l. For each corresponding $\tilde{X}^l$, we compute the sequence $\{b_q^\ell\}_{\ell=0,1,\ldots,15, q=0,1,\ldots}$ defined by

$$b_q^\ell, = \left| \left\{ v \in \{0, 1, \ldots, 255\} : \left| \left\{ \tilde{D}_j \in \tilde{X}^l : (\tilde{D}_j)_\ell = v \right\} \right| = q \right\} \right|,$$

and check for a match in the table. If (U^i, V^j) are slid sets, a match must occur.

We now analyze the probability that two unrelated sets match. Namely, we are interested in the probability that two sets which are not slid sets match their multiplicity vectors. We note that when the two sets are not slid sets, then we can treat them as two sets of 256 random values selected from $\{0, 1, \ldots, 255\}$. To estimate the probability of a collision, we study the distribution of the multiplicy vector for such a random set. The actual distribution of the multiplicaty vector is a multinomial one. At the same time, the number of values that do not appear (i.e., have multiplicity 0) is about $256/e$ (disregarding dependecies between the different values, the number of times a specific value appears follows a Poisson distribution with a mean value of 1). The actual number of values that do not appear can be approximated by a binomial distribution of 256 experiments, each with success probability of $1/e$. This suggests that the amount of information conveyed by the number of entries that do not appear has $\frac{1}{2}\log_2(2\pi \cdot e \cdot 256 \cdot \frac{1}{e} \cdot (1 - \frac{1}{e})) \approx 4.99$ bits of information. The same is true also w.r.t. the number of entries which appear once. Thus, each byte of the state carries at least 9.98 bits of information. This means that the entire sequence contains more than 159 bits of information, which are sufficient to detect all correct pairs of slid sets (U^i, V^j) with an overwhelming probability. We verified experimentally that this statistic contains at least 8 bits of information in each byte (and thus, at least

128 bits of information in total), assuming random and uniform distribution of the ciphertexts.

Retrieving the key from a pair of slid sets. Given a pair of slid sets (U^i, V^j), and the corresponding sets of values $(\bar{W}^i, \tilde{X}^j)$ we can easily and efficiently find the round keys k_2 and $k_1' = MC^{-1}(k_1)$. The attack is based on Eq. (6), which consists of 16 independent byte equations of the form $(\tilde{D}_j)_\ell = (ARK_1' \circ SB \circ ARK_2(\bar{C}_i))_\ell$, as was mentioned above. In each byte ℓ, we know from W the multiplicity of each value entering this byte (e.g., input value 0 appears once in $\bar{W}^i$ in this byte position). Note that the statistic we use here is more refined than the statistic we used above: we do not only ask how many values are obtained q times, but rather which are the values that are obtained q times.

We now guess the value of byte ℓ of k_2 and of k_1', and so, we can compute the value $(ARK_1' \circ SB \circ ARK_2(\bar{C}_i))_\ell$ for each $C_i \in W$. We compute this value for every $\bar{C}_i \in \bar{W}$, and check whether the multiplicities of the obtained values conform to their multiplicities in $\tilde{X}^j$. If there is no match, we discard the guess of $(k_2)_\ell, (k_1')_\ell$.

It is easy to see that this procedure offers a very strong filtering, and so with overwhelming probability, in each byte only a single candidate for k_2 and k_1' remains.

We note that this attack algorithm does not rely on the actual order of keys used in the last two rounds. Thus, even though we presented the attack for the case of even number of rounds, it can be applied in exactly the same way to an odd number of rounds (where Eq. (6) is replaced by $\tilde{D}_j = ARK_2' \circ SB \circ ARK_1(\bar{C}_i)$ and we obtain a single candidate for k_1 and k_2').

The attack algorithm. As is shown in detail in Algorithm 2, we consider two structures $\mathcal{T}_P, \mathcal{T}_Q$ of 2^{68} chosen plaintexts each. The structure $\mathcal{T}_P$ consists of 2^{60} Λ-sets, all with the first byte permuted and the rest fixed. Similarly, $\mathcal{T}_Q$ is chosen such that $\tilde{\mathcal{T}_Q}$ contains 2^{60} Λ-sets, all with the first byte permuted and the rest fixed. We then compute for each Λ-set in $\mathcal{T}_P$ its a_q^ℓ statistics and for each Λ-set in $\mathcal{T}_Q$ its b_q^ℓ statistics, and look for collisions between the statistics. Once such a collision is found (i.e., a pair of slid sets is identified), we apply the key recovery algorithm.

The data complexity of the attack is 2^{69} chosen plaintexts, the memory complexity is 2^{69} and the time complexity is 2^{69} as well. The success probability is the probability that the data contains a pair of slid sets. As the probability of each set pair of sets $U^i \in \mathcal{T}_P$ and $V^j \in \mathcal{T}_Q$ to be slid is 2^{-120} (since a match in 15 bytes is needed), the probability of containing a pair of slid sets is $1 - (1 - 2^{-120})^{2^{120}} \approx 0.63$. Therefore, the success probability of the attack is 63%.

Attacking 2-KSAfp. The same attack applies to any variant of 2-KSAfp, either with complete or incomplete diffusion. The data, memory and time complexities are $2^{(n+s)/2+1} = O(2^{(n+s)/2})$.

Algorithm 2 A slide attack on 2K-AESfp using slid sets

Ask for the encryption of two structures $\mathcal{T}_P, \mathcal{T}_Q$, each of size 2^{68}, defined as explained above.

Initialize an empty hash table T.

for each Λ-set $U^i \in \mathcal{T}_P$ **do**

Let the ciphertexts corresponding to the plaintexts U^i be W^i, and consider the corresponding set $\bar{W}^i$,

Compute the sequence-of-sequences $(a_q^\ell)_{\ell=0,1,\ldots,15,q=0,1,\ldots}$, and store it in T, along with the index i.

for Each $V^j \in \mathcal{T}_Q$ **do**

Let the ciphertexts of corresponding to the plaintexts of V^j be X^j.

Compute from X^j the corresponding $\tilde{X}^j$.

Compute the sequence-of-sequences $(b_q^\ell)_{\ell=0,1,\ldots,15,q=0,1,\ldots}$, and check for a matching sequence in T.

if a match exists **then**

Assume that (U^i, V^j) are slid sets, and consider the corresponding sets $(\bar{W}^i, \tilde{X}^j)$.

for each byte $\ell \in \{0,\ldots,15\}$ **do**

for each guess of byte $k_{2,\ell}$ and $k'_{1,\ell}$ **do**

Partially encrypt all $(\bar{C}_i)_\ell \in W^i$ and obtain a set of values $\{t_1, t_2, \ldots t_{256}\}$.

if the set $\{t_1, t_2, \ldots t_{256}\}$ matches the set of values $\{\tilde{D}_{j,\ell} : \tilde{D}_j \in \tilde{X}\}$ **then**

Output "the subkey values in byte ℓ are $k_{2,\ell}$ and $k'_{1,\ell}$.

3.2 Slid sets attack on 1-KSAs

In this section we show that a modification of the above attack can be used to break 1-KSA in which the operation S is key-dependent – i.e., consists of a parallel application of n/s key-dependent permutations on s-bit words. The complexity of the attack is only slightly higher than the complexity of the attack described above – namely, data, memory, and time complexity of $2\sqrt{s \log 2}2^{(n+s)/2} = O(\sqrt{s}2^{(n+s)/2})$ (i.e., an increase by a factor of $\sqrt{s \log 2}$ with respect to the attack of Section 3.1).

The setting. For the sake of helping readability, we first present the attack in the special case of 1K-AESf with a key-dependent S-box. A related variant (AES with key-dependent S-box) was studied in a number of papers, e.g., [22,24,31,32]. First, we would like to simplify the problem.

Assume that (P_i, Q_j) is a slid pair. This means that

$$Q_j = MC \circ SR \circ SB \circ ARK(P_i),$$

where ARK denotes key addition with the subkey k. We can peel off the unkeyed operations MC, SR by denoting $\tilde{Q}_j = SR^{-1} \circ MC^{-1}(Q_j)$, and obtain the equation

$$\tilde{Q}_j = SB \circ ARK(P_i). \tag{7}$$

By the slide property, the relation between the corresponding ciphertexts (C_i, D_j) is

$$D_j = ARK \circ MC \circ SR \circ SB(C_i).$$

We can simplify this relation by interchanging the operations ARK and MC, at the expense of replacing the subkey k with $SR^{-1} \cdot M^{-1}k$, and then peeling off MC and SR as well. We obtain the equation

$$\tilde{D}_j = ARK' \circ SB(C_i). \tag{8}$$

Detection of slid sets. Eq. (7) and (8) show that the transformation from P_i to $\tilde{Q}_j$ consists of application of 16 independent functions on the bytes of the state, and the same goes for the transition from C_i to $\tilde{D}_j$. Hence, we can use the same algorithm for detecting slid sets in as the previous attack (i.e., using the sequences a_q^ℓ and b_q^ℓ that count multiplicities of values).

Deducing slid pairs from slid sets. The remaining goal is to retrieve the subkey k and the key-dependent S-box S given a few pairs of slid sets (U^i, V^j). (As we shall see, a single pair of slid sets does not contain enough information for determining the S-box uniquely). The simple algorithm for this step described above cannot be applied here since the S-box S is unknown. Instead, we make use of a refined statistic that allows us deducing the slid counterpart $Q_j \in V$ of each $P_i \in U$. Namely, while in Section 3.1 we used the multiplicities of values in each byte separately, here we use the sequence of multiplicities of a value in all bytes simultaneously.

As in Section 3.1, we denote by W, X the sets of ciphertexts that correspond to the plaintext sets U, V, respectively. Furthermore, we denote by $\tilde{X}$ the set obtained from X by setting $\tilde{D}_j = SR^{-1} \circ MC^{-1}(D_j)$, for any $D_j \in X$.

For each $C_i \in W$, and for each byte $0 \leq \ell \leq 15$, we count the number of other elements $C_{i'} \in W$ such that $(C_i)_\ell = (C_{i'})_\ell$. That is, we construct the 16-element sequence $\{c_\ell^i\}_{\ell=0,1,\ldots,15}$, where

$$c_\ell^i = |\{C_{i'} \in W : (i' \neq i) \wedge ((C_i)_\ell = (C_{i'})_\ell)\}|.$$

Similarly, for each $\tilde{D}_j \in \tilde{X}$, and for each byte $0 \leq \ell \leq 15$, we construct the 16-element sequence $\{d_\ell^j\}_{\ell=0,1,\ldots,15}$, where

$$d_\ell^j = |\{\tilde{D}_{j'} \in \tilde{X} : (j' \neq j) \wedge ((\tilde{D}_{j'})_\ell = (\tilde{D}_j)_\ell)\}|.$$

We observe that the statistic represented by the sequences $\{c_\ell^i\}$ and $\{d_\ell^j\}$ is preserved by slid pairs. That is, if Q_j is the slid counterpart of P_i, then the corresponding sequences $\{c_\ell^i\}, \{d_\ell^j\}$ must be equal! Indeed, if for some i' we have $(C_i)_\ell = (C_{i'})_\ell$, then the equality $(\tilde{D}_j)_\ell = (\tilde{D}_{j'})_\ell$ must hold for $\tilde{D}_j$, where $Q_{j'}$ is the slid counterpart of $P_{i'}$.

Therefore, we can retrieve the right slid pairs (P_i, Q_j) by the simple procedure described in Algorithm 3.

Algorithm 3 Retrieving slid pairs from slid sets, for 1K-AESfs

Initialize a list L of candidate slid pairs.
for Each $C_i \in W$ **do**
  Compute the sequence $(c^i_\ell)_{\ell=0,1,\ldots,15}$, and store in a hash table, along with P_i.
for Each $\tilde{D}_j \in \tilde{X}$ **do**
  Compute the sequence $(d^j_\ell)_{\ell=0,1,\ldots,15}$, and check for a match in the table,
  for Each match in the hash table **do**
    Add the corresponding pair (P_i, Q_j) to L.

We checked experimentally and found that the statistic $(c^i_\ell)_{\ell=0,1,\ldots,15}$ contains about 27 bits of information, assuming random and uniform distribution of the ciphertexts. This means that the probability of a random pair (P_i, Q_j) to yield a match in the table is 2^{-27}. As the plaintext sets (U, V) contain only 2^{16} pairs (P_i, Q_j), with a high probability only the right slid pairs yield matches in the table.

Hence, the above algorithm, whose complexity is about 2^{16} operations, finds the slid counterpart $Q_j \in V$ of each $P_i \in U$.

Retrieving the secret material, given several pairs of slid sets. By Eq. (8) (applied in each byte separately), each slid pair (P_i, Q_j) provides us with an input/output pair for the function $f_\ell(x) = k'_\ell \oplus SB(x)$, where k'_ℓ denotes the ℓ's byte of $k' = SR^{-1} \cdot M^{-1} k$. Hence, each pair of slid sets provides us with 256 input/output pairs for each function f_ℓ. However, these input/output pairs are not distinct. A reasonable assumption is that the values $(C_i)_\ell$ (where C_i ranges over elements of W) are distributed uniformly at random in $\{0, 1, \ldots, 255\}$. Hence, by the *coupon collector's* problem, we need $256 \cdot \log 256$ input/output pairs in order to recover f_ℓ completely with a high probability. Therefore, about $\log 256 \approx 6$ pairs of slid sets are sufficient for recovering all functions f_ℓ.

Once the function $ARK' \circ SB$ is recovered, the key k can be recovered instantly, by picking some (already queried) ciphertext C and partially decrypting it using the knowledge of the functions $ARK' \circ SB, SR, MC$. The entire decryption process can be simulated, except for the initial ARK operation. Hence, we obtain the value $P \oplus k$, where P is the plaintext that corresponds to C. As P is known, k can be retrieved.

The complexity of the attack. The attack presented above contains two steps, in addition to the steps of the attack described in Section 3.1. The first is a step that recovers slid pairs from pairs of slid sets. As described above, the complexity of this step is 2^{16}, which is negligible with respect to other steps of the attack. The second step is recovering the function $ARK' \circ SB$. Its complexity is also negligible, but it requires 6 pairs of slid sets, instead of a single pair in the attack of Section 3.1. This increases the data complexity of the attack by a factor of $\sqrt{6}$, and increases the data and time complexity of the attack accordingly.

Therefore, the data, memory and time complexity of the attack on 1K-AES with a secret S-box and a MixColumns operation in the last round, is about $2^{70.3}$, and its success probability is about 63%.

Attacking 1-KSAs. The same attack applies to any variant of 1-KSAfs. The only difference is that the number of required pairs of slid sets is $s \log 2 = \log(2^s)$ (instead of $\log 256$ in 1K-AES). Hence, the data, memory, and time complexity of the attack is $2\sqrt{s \log 2} \cdot 2^{(n+s)/2}$.

Furthermore, the attack applies with the same complexity also to any variant of 1-KSAts. Indeed, the difference between 1-KSAfs and 1-KSAts is in the relation between C_i and D_j, which becomes

$$D_j = ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(C_i).$$

By replacing ARK with linear operations, we can simplify this equation into

$$\tilde{D}_j = ARK' \circ SB \circ ARK''(\bar{C}_i), \tag{9}$$

where $\tilde{D}_j = SR^{-1}(D_j)$, $\bar{C}_i = MC(C_i)$, ARK' denotes addition with $SR^{-1}(k)$ and ARK'' denotes addition with $MC(k) \oplus k$. Eq. (9) has exactly the same structure as Eq. (8), and hence, the attack described above applies, with the same complexity, to 1-KSAts.

4 Slide Attack using a Hypercube of Slid Pairs

In this section we present a new technique which we call a *hypercube of slid pairs*, and use it to attack 1-KSAts (*with a secret S-box*) with data, memory, and time complexity of $\sqrt{s}2^{(n+s(s/2+1)+s/2)/2+1}$ (in the special case of 1K-AESt: 2^{88}). For sake of concreteness, we demonstrate the attack on 1K-AES.

The idea behind the attack. The attack consists of two steps. First we detect a slid pair, and then we use it to recover the key used in the ARK operation and in the secret S-box. In order to detect a slid pair, we want to attach to each candidate slid pair many "friend pairs", such that if the candidate is indeed a slid pair, then all the friend pairs are slid pairs as well.

To be specific, we consider 1K-AES with a secret S-box. Consider a slid pair (P_i, Q_j). As was shown in Section 3.2, the relation between P_i and Q_j can be simplified into the equation $\tilde{Q}_j = SB \circ ARK(P_i)$, where $\tilde{Q}_j = SR^{-1} \circ MC^{-1}(Q_j)$. Furthermore, it was shown that if $(P_i, Q_j), (P_{i'}, Q_{j'})$ are slid pairs, $\tilde{Q}_j, \tilde{Q}_{j'}$ are computed from $Q_j, Q_{j'}$, and if $P_{i'}$ differs from P_i only in a single byte, then $\tilde{Q}_{j'}$ differs from $\tilde{Q}_j$ only in a single byte as well.

It follows that if we take a, a' be two vectors that are non-zero only in byte 0 (where they assume arbitrary values), then with probability 2^{-8}, $(P_i \oplus a, \tilde{Q}_j \oplus a')$ also corresponds to a slid pair.

In the same way, we take values b, c, d, e which are non-zero only in byte $1, 2, 3, 4$, respectively. Then we define b', c', d', e' similarly to the definition of a', and obtain the pairs $(P_i \oplus b, \tilde{Q}_j \oplus b'), \ldots, (P_i \oplus e, \tilde{Q}_j \oplus e')$, such that each of them is a slid pair with probability 2^{-8}. Thus, we may attach to the pair (P_i, Q_j) five friend pairs, such that if (P_i, Q_j) is a slid pair, then each of its friend pairs is a slid pair with probability 2^{-8}.

Constructing a hypercube of slid pairs. We are ready to present the construction of the *hypercube of slid pairs*. Assume that all five pairs $(P_i \oplus a, \tilde{Q}_j \oplus a'), \ldots, (P_i \oplus e, \tilde{Q}_j \oplus e')$ correspond to slid pairs. We observe that this implies that for any quintet $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \in \{0,1\}^5$, the pair

$$(P_i \oplus \alpha_1 a \oplus \alpha_2 b \oplus \alpha_3 c \oplus \alpha_4 d \oplus \alpha_5 e, \tilde{Q}_j \oplus \alpha_1 a' \oplus \alpha_2 b' \oplus \alpha_3 c' \oplus \alpha_4 d' \oplus \alpha_5 e')$$

is a slid pair as well. Indeed, in each of the 16 functions applied in parallel, the two values of the new slid pair are equal either to the values of $(P_i, \tilde{Q}_j)$ or to the values of one of its 5 "friends" which we assumed to be slid pairs as well. (For example, in byte 0 the values are equal either to those of $(P_i, \tilde{Q}_j)$ or to those of $(P_i \oplus a, \tilde{Q}_j \oplus a')$.) We denote the new pair by $(P_{i,\alpha}, \tilde{Q}_{j,\alpha})$.

This allows us to leverage 5 friend pairs into $2^5 - 1$ friend pairs (or more generally, t friend pairs into $2^t - 1$ friend pairs). As the friend pairs we construct correspond to the vertices of the hypercube $\{0,1\}^t$, we call this method of constructing a *hypercube of slid pairs*. We note that this construction idea is motivated by the *mixture differential* attack presented by Grassi [23]. Hence, so far we have attached to the pair (P_i, Q_j) 31 friend pairs, such that if (P_i, Q_j) is a slid pair, then with probability 2^{-40}, all the friend pairs are slid pairs as well.

Using the hypercube of slid pairs in the attack. Consider the ciphertexts (C_i, D_j) that correspond to a slid pair (P_i, Q_j). As was shown in Section 3.2, the relation between C_i and D_j can be simplified into the equation

$$\tilde{D}_j = ARK' \circ SB(C_i).$$

As both the transformation from P_i to $\tilde{Q}_j$ and the transformation from C_i to $\tilde{D}_j$ consist of application of 16 independent functions on the bytes of the state, it follows that if for some $\alpha, \alpha' \in \{0,1\}^5$ and for some byte $\ell \in \{0, 1, \ldots, 15\}$, we have $(C_{i,\alpha})_\ell = (C_{i,\alpha'})_\ell$, then we must have $(D_{j,\alpha})_\ell = (D_{j,\alpha'})_\ell$ as well. Note that the same property was exploited in the attack of Section 3.2. In our attack, the size of the structure is smaller, which restricts the amount of information that can be collected. On the other hand, we know that the slid counterpart of each $P_{i,\alpha}$ is $Q_{i,\alpha}$, and this turns out to be sufficient for detecting the slid pairs.

Indeed, the expected number of such collisions is $2^{-8} \cdot \binom{32}{2} \cdot 16 \approx 31$. We denote each such collision by the triple (α, α', ℓ), and store the list of all collisions in a lexicographic order. The exhaustive list of all locations of collisions contains more than 256 bits of information, and thus, the probability that two lists of triples that do not originate from a slid pair are equal, is negligible. Hence, equality of two lists implies a slid pair (with overwhelming probability).

Algorithm 4 A Slide Attack on 1K-AES with a Secret S-box using Hypercube of Slid Pairs

Ask for the encryption of two structures $\mathcal{T}_P, \mathcal{T}_Q$, each of 2^{87} chosen plaintexts, constructed as defined above..
Initialize an empty list L (intended to store the detected slid pairs).
for each plaintext/ciphertext pair $(P_i, C_i) \in \mathcal{T}_P$ **do**
  Compute the 31 friend pairs $(P_{i,\alpha}, C_{i,\alpha})$ and the corresponding values $\tilde{D}_{i,\alpha}$,
  Find all collisions of the form $(\bar{C}_{i,\alpha})_l = (\tilde{C}_{i,\alpha'})_l$,
  Store in a hash table the sequence of triples (α, α', l) that represent all collisions, arranged in lexicographic order, along with the value P_i used to create them.
for Each plaintext/ciphertext pair (Q_j, D_j) **do**
  Compute the 31 'friend values' $\bar{Q}_{j,\alpha}$ and the corresponding pairs $(Q_{j,\alpha}, D_{j,\alpha})$,
  Find all collisions of the form $(D_{j,\alpha})_l = (D_{j,\alpha'})_l$,
  Compute the sequence of triples (α, α', l) that represent all collisions and check for a match in the hash table.
for Each collision in the table **do**
  Add the corresponding pair (P_i, Q_j) and its 31 friends to L.
for Each slid pair $(P_i, Q_j) \in L$ **do**
  Use the relation between P_i and $\bar{Q}_j$ to detect an input/output pair of $SB \circ ARK$ for each byte, until the entire function is detected.
Once $SB \circ ARK$ in all bytes is detected, find the final key whitening operation ARK using a single trial encryption.

Recovering the secret S-box. Once a slid pair (P_i, Q_j), along with 31 friend pairs, are detected, they provide us with 32 input/output values to the function $ARK \circ SB$. As was shown in Section 3.2, about $256 \log 256 \approx 1420$ input/output values are needed in order to recover the S-box, and thus, we have to take a sufficiently large data set so that it will contain at least 45 slid pairs. Namely, we take two structures $\mathcal{T}_P, \mathcal{T}_Q$ of 2^{87} plaintexts each. The structures contain 2^{174} pairs. As the probability that a pair and all its friend pairs are slid pairs is $2^{-128} \cdot 2^{-40} = 2^{-168}$, the expected number of slid hypercubes is 64, and so, with a high probability the number of slid pairs is sufficient for recovering $ARK \circ SB$. Once this operation is recovered, all the operations in the cipher except for the final ARK operation are known, and thus, the key k can be immediately retrieved. The resulting attack algorithm is given in Algorithm 4.

We note that the plaintext structures can be chosen in such a way that constructing the friend pairs does not require increasing the data complexity. Indeed, we can choose each of the structures $\mathcal{T}_P, \tilde{\mathcal{T}}_Q$ as a union of 2^{47} sub-structures of size 2^{40}, where in each sub-structure, all plaintexts attain some equal value in bytes $5, 6, \ldots, 15$ and all possible values in bytes $0, 1, \ldots, 4$. This guarantees that for any a, b, c, d, e, α and for any $P_i \in \mathcal{T}_P$, the value $P_i \oplus \alpha_1 a \oplus \alpha_2 b \oplus \alpha_3 c \oplus \alpha_4 d \oplus \alpha_5 e$ also belongs to $\mathcal{T}_P$, and the same for $\tilde{\mathcal{T}}_Q$.

As was explained above, the algorithm requires 2^{88} chosen plaintexts, memory and time, and succeeds with a high probability. The same attack applies to any variant of 1-KSAts (possibly with a complete diffusion). First, in the detection of a hypercube of slid pairs of dimension t (given s-bit S-boxes in n-bit cipher) we get from each candidate hypercube $2^{-s} \cdot \binom{2^t}{2} \cdot n/s$ values in the list. As each such value suggests about s bits of entropy (i.e., a total of $2^{-s} \cdot \binom{2^t}{2} \cdot n$ bits), and as we have at most 2^{2n} sets of slid pairs, we require that $2^{-s} \cdot \binom{2^t}{2} \cdot n \approx 2n$. In other words, one needs to set $2^{2t-s-1} \cdot n = 2n$, i.e., $t = \lceil s/2 \rceil$. Now, if $\mathcal{T}_P$ and $\mathcal{T}_Q$ have D plaintexts each, we expect $D^2 \cdot 2^{-n} \cdot 2^{-ts}$ hypercubes of slid pairs, each suggesting 2^t slid pairs. As we need about $\log 2^s \cdot 2^s \approx 0.7 \cdot s \cdot 2^s$ slid pairs, we need $D = \sqrt{s} \cdot 2^{(n+s(s/2+1)+s/2)/2}$, or a total of data, memory, and time complexities of $\sqrt{s} 2^{(n+s(s/2+1)+s/2)/2+1}$.

We note that the complexity of the 'hypercube of slides' attack on 1-KSAts is inferior to the complexity of the 'slid sets' attack presented in Section 3.2. However, this attack may be advantageous in specific instances of 1-KSAts, e.g., when the operation S admits differential characteristics with a non-negligible probability.

5 Slide Attack using Suggestive Plaintext Structures

In this section we present a new technique which we call *suggestive plaintext structures*, and use it to attack 1-KSAt (and in particular, 1K-AES) with data, memory of $3 \cdot 2^{n/2}$ and time complexity of $4 \cdot 2^{n/2}$. Interestingly, unlike most other slide attacks, this attack is *deterministic*, in the sense that it has 100% success probability.

The idea behind the attack is using two tailor-made plaintext structures $\mathcal{T}_P = \{P_i\}_{i=1,\dots,2^{n/2}}$ and $\mathcal{T}_Q = \{Q_j\}_{j=1,\dots,2^{n/2}}$, such that the mere knowledge that some P_i has a slid counterpart in the structure $\{Q_j\}$ (even without the knowledge of which Q_j exactly is the counterpart) yields some key information that can be used in the attack.

To be specific, we consider 1K-AES. Let $\mathcal{T}_P = \{P_i\}$ be a structure of 2^{64} plaintexts that assume the constant value 0 in Columns 2,3, and assume all 2^{64} possible values in Columns 0,1. We let $\mathcal{T}_Q = \{Q_j\}$ be a structure of 2^{64} plaintexts such that the plaintexts of the corresponding structure $\tilde{\mathcal{T}_Q} = \{\tilde{Q}_j\}$ (where for each j, $\tilde{Q}_j = SB^{-1} \circ SR^{-1} \circ MC^{-1}(Q_j)$) assume the constant value 0 in Columns 0,1, and assume all 2^{64} possible values in Columns 2,3.

The main observations behind the attack. Observe that (P_i, Q_j) is a slid pair if and only if the corresponding pair $(P_i, \tilde{Q}_j)$ satisfies $P_i \oplus \tilde{Q}_j = k$. We use two conclusions of this observation:

1. *Friend pairs for free.* If $(P_i, \tilde{Q}_j)$ is a slid pair, then for any a, $(P_i \oplus a, \tilde{Q}_j \oplus a)$ is a slid pair as well.
   In principle, this allows attaching to each candidate slid pair a friend pair, which allows us to enhance the filtering condition on the ciphertext side. However, in our case, we have $\tilde{Q}_j \oplus a \in \tilde{\mathcal{T}_Q}$ only if $a_{\mathrm{Col}(0,1)} = 0$. In such a case, $P_i \oplus a \notin \mathcal{T}_P$, unless $a = 0$ (which means that the new pair is identical to the initial one).

To overcome this problem, we add to the data set another structure $\mathcal{T}_R = \{R_i\}$ of 2^{64} plaintexts that assume the constant value 0 in Column 2 and the constant value 1 in Column 3, and assume all 2^{64} possible values in Columns 0,1. Then, we can attach to each $P_i \in \mathcal{T}_P$ a friend $R_i = P_i \oplus (0,0,0,1) \in \mathcal{T}_R$, such that for each $Q_j \in \mathcal{T}_Q$, the pair $(P_i, \tilde{Q}_j)$ is a slid pair if and only if $(R_i, \tilde{Q}_j \oplus (0,0,0,1))$ is a slid pair as well. We denote the ciphertext that corresponds to the plaintext R_i by F_i. Furthermore, we denote the element of $\mathcal{T}_Q$ that corresponds to $\tilde{Q}_j \oplus (0,0,0,1) \in \tilde{\mathcal{T}}_Q$ by Q'_j, and denote the corresponding ciphertext by D'_j.

2. *Key information for free.* Since all $\tilde{Q}_j \in \mathcal{T}_Q$ satisfies $(\tilde{Q}_j)_{\mathrm{Col}(0,1)} = 0$, it follows that for any $P_i \in \mathcal{T}_P$, we may have $P_i \oplus \tilde{Q}_j = k$ only if $(P_i)_{\mathrm{Col}(0,1)} = k_{\mathrm{Col}(0,1)}$. Therefore, when we consider some $P_i \in \mathcal{T}_P$ as a candidate for being part of a slid pair (with counterpart from $\mathcal{T}_Q$), we immediately obtain a candidate value for the two initial columns of the key k!
   Of course, the adversary does not know whether some $P_i \in \mathcal{T}_P$ has a slid counterpart in $\mathcal{T}_Q$, and so does not obtain the key information directly. However, this key information can be used indirectly to check the validity of many slid pair candidates simultaneously, as shown below.

We note that the latter observation also explains why the attack succeeds deterministically. By the choice of the structure $\mathcal{T}_P$, its elements assume all possible values in Columns 0,1. In particular, for the right secret key k, there exists $P_i \in \mathcal{T}_P$ such that $(P_i)_{\mathrm{Col}(0,1)} = k_{\mathrm{Col}(0,1)}$. For that plaintext P_i, we have $(P_i \oplus k)_{\mathrm{Col}(0,1)} = 0$. However, the structure $\tilde{\mathcal{T}}_Q$ contains all 2^{64} values whose first two columns are equal to 0. Hence, $\tilde{Q}_j := P_i \oplus k \in \tilde{\mathcal{T}}_Q$, and so, $(P_i, \tilde{Q}_j)$ is a slid pair. Hence, the data set is *guaranteed* to contain a slid pair.

Exploiting the key information. Assume that (P_i, Q_j) is a slid pair. Then, due to the omission of MixColumns from the last round of AES, the corresponding ciphertexts satisfy the relation

$$D_j = ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(C_i). \qquad (10)$$

Similarly, since $(R_i, \tilde{Q}_j \oplus (0,0,0,1))$ is a slid pair (by property (1) above), the corresponding ciphertexts F_i, D'_j, satisfy

$$D'_j = ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(F_i). \qquad (11)$$

Now, assume that some specific $P_i \in \mathcal{T}_P$ has a slid counterpart in $\mathcal{T}_Q$. By property (2) above, this implies $k_{\mathrm{Col}(0,1)} = (P_i)_{\mathrm{Col}(0,1)}$. This allows us to compute Columns 0,1 of $ARK \circ MC \circ ARK(C_i)$ (since we know Columns 0,1 of k), and consequently, also shifted columns $SR(\mathrm{Col}(0,1))$ of the state $SR \circ SB \circ ARK \circ MC \circ ARK(C_i)$. In a similar way, we can compute the value of shifted columns $SR(\mathrm{Col}(0,1))$ of the state $SR \circ SB \circ ARK \circ MC \circ ARK(F_i)$. Hence, we can

Algorithm 5 A Slide Attack on 1K-AES

Ask for the encryption of three structures $\mathcal{T}_P, \mathcal{T}_Q, \mathcal{T}_R$, each of 2^{64} plaintexts, as described in the text.
Initialize an empty hash table T.
for each plaintext/ciphertext pair $(Q_j, D_j) \in \mathcal{T}_Q$ **do**
  Compute the value $\tilde{Q}_j = SB^{-1} \circ SR^{-1} \circ MC^{-1}(Q_j)$,
  Compute the value $Q'_j = MC \circ SR \circ SB(\tilde{Q}_j \oplus (0,0,0,1))$,
  Denote the corresponding ciphertext by D'_j.
  Store in T the pairs $((D_j \oplus D'_j)_{SR(\mathrm{Col}(0,1))}, Q_j)$.
for each plaintext/ciphertext pair $(P_i, C_i) \in \mathcal{T}_P$ **do**
  Set $k_{\mathrm{Col}(0,1)} = (P_i)_{\mathrm{Col}(0,1)}$,
  Compute shifted columns $SR(\mathrm{Col}(0,1))$ of the value $SR \circ SB \circ ARK \circ MC \circ ARK(C_i) \oplus SR \circ SB \circ ARK \circ MC \circ ARK(R_i)$,
  if the computed value is the first coordinate of an entry $(((D_j \oplus D'_j)_{SR(\mathrm{Col}(0,1))}, Q_j)$ **then**
    Test the key candidate $k = P_i \oplus \tilde{Q}_j$ by trial encryption.

compute the value of shifted columns $SR(\mathrm{Col}(0,1))$ of

$$\begin{aligned} SR \circ SB \circ ARK \circ MC \circ ARK(C_i) \oplus SR \circ SB \circ ARK \circ MC \circ ARK(F_i) \\ = ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(C_i) \oplus \\ ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(F_i). \end{aligned}$$

By Eq. (10),(11), this value is equal to $(D_j \oplus D'_j)_{SR(\mathrm{Col}(0,1))}$. This gives us a 64-bit filtering condition that can be checked for all j's simultaneously, by searching for a collision in a precomputed hash table. This results in the attack algorithm given in Algorithm 5.

Since the match checked in the hash table is a 64-bit filtering condition, in expectation a single value of j is suggested for each value of i. As each match yields a suggestion for the entire key, any random match is almost surely discarded using a single additional encryption operation. (The probability that some wrong guess survives is as low as 2^{-64}, and so, can be neglected.) On the other hand, as explained above, the data set must contain a slid pair (P_i, Q_j), and this slid pair suggests the correct value of the secret key.

Therefore, the attack requires data complexity of $3 \cdot 2^{64}$ chosen plaintexts, memory complexity of $3 \cdot 2^{64}$, time complexity of $4 \cdot 2^{64}$ encryptions, and succeeds with probability 100%.

The attack applies, with exactly the same complexity, to any variant of 1-KSAt with *incomplete diffusion*. Indeed, the only place where the exact structure of AES was used in the attack is the ability to compute 64 bits of the value $ARK \circ MC \circ ARK(C_i)$, given $k_{\mathrm{Col}(0,1)}$. The adversary has this ability (or equivalent ability with some other part of the state) as long as the operation A is applied to blocks of size at most half of the state. This is indeed the case in any variant of 1-KSAt with incomplete diffusion. Therefore, we obtain an attack with data complexity of $3 \cdot 2^{n/2}$ chosen plaintexts, memory complexity of $3 \cdot 2^{n/2}$, time complexity of $4 \cdot 2^{n/2}$ encryptions, and success probability of 100%.

For 1-KSAt with complete diffusion, the above attack does not apply, and we are not aware of any attack with complexity close to $2^{n/2}$ on this variant.

6 Substitution Slide Attack

We now present a new technique which we call *substitution slide*, and use it to attack 1-KSAt (and in particular, 1K-AES) using only $2^{n/2}$ *known plaintexts*, $2^{n/2}$ memory and about $2^{3n/4}$ time. Unlike the attack presented in Section 5, this attack applies also for 1-KSAt with complete diffusion.

The idea behind the attack. As before, we present the attack on 1K-AES for sake of simplicity. Consider a structure $\mathcal{T}_P$ of 2^{64} *known* plaintexts, and let $\tilde{T}$ be the structure obtained by[6] setting $\tilde{P}_i = SB^{-1} \circ SR^{-1} \circ MC^{-1}(P_i)$ for any $P_i \in \mathcal{T}_P$. As was explained in Section 5, if (P_i, P_j) is a slid pair, then we have:

$$\begin{cases} P_i \oplus \tilde{P}_j = k, \\ C_j = ARK \circ SR \circ SB \circ ARK \circ MC \circ ARK(C_i). \end{cases}$$

The basic observation we use in this attack is that the (simpler) first equation can be substituted into the (complex) second equation, in order to get rid of key dependence.

Specifically, the second equation can be rewritten as

$$SB^{-1} \circ SR^{-1} \circ ARK(C_j) = ARK \circ MC \circ ARK(C_i). \qquad (12)$$

The right hand side of this equation can be written as

$$ARK \circ MC \circ ARK(C_i) = k \oplus MC(C_i \oplus k) = MC(k) \oplus k \oplus MC(C_i),$$

Now, we can get rid of the key dependence by *substituting* the value of k from the first equation above. We have

$$MC(k) \oplus k \oplus M \cdot C_i = MC(P_i \oplus \tilde{P}_j) \oplus P_i \oplus \tilde{P}_j \oplus MC(C_i).$$

Hence, Eq. (12) can be rewritten as

$$SB^{-1} \circ SR^{-1} \circ ARK(C_j) \oplus MC(\tilde{P}_j) \oplus \tilde{P}_j = MC(P_i) \oplus P_i \oplus MC(C_i). \qquad (13)$$

Eq. (13) is almost what we need. The right hand side depends only on (P_i, C_i) and thus can be computed in advance for all values of i and stored in a hash table. The left hand side depends on (P_j, C_j); however, it depends also on the

[6] We alert the reader that in this section we use (P_i, P_j) to denote a slid pair (rather than (P_i, Q_j). This was done to emphasize that P_i and P_j, both, are part of a set of known plaintexts.

Algorithm 6 A Known Plaintext Slide Attack on 1K-AES

Ask for 2^{64} known plaintexts/ciphertext pairs (P_i, C_i).
Initialize an empty hash table T.
for each plaintext/ciphertext pair (P_i, C_i) **do**
  Compute the value $\mathcal{P}_i = MC(P_i) \oplus P_i \oplus MC(C_i)$,
  Store in T the triples $((\mathcal{P}_i)_{\mathrm{Col}(0)}, (P_i)_{SR(\mathrm{Col}(0))}, (P_i)_{SR(\mathrm{Col}(1,2,3))})$, indexed by the first two coordinates.
for each guess of $k_{SR(\mathrm{Col}(0))}$ **do**
  for each plaintext/ciphertext pair (P_j, C_j) **do**
    Compute Column 0 of the value $\mathcal{Q} = SB^{-1} \circ SR^{-1} \circ ARK(C_j) \oplus MC(P_j) \oplus P_j$,
    Check for entries in the hash table whose first two coordinates match the pair $((\mathcal{Q}_j)_{\mathrm{Col}(0)}, (\tilde{P}_j \oplus k)_{SR(\mathrm{Col}(0))})$.
  for Each match found in the table **do**
    Test the key candidate $k = P_i \oplus \tilde{P}_j$.

secret key, and thus, we cannot just evaluate it for all j and check for a match in the table.

In order to evaluate ℓ bytes of the left hand side, we have to guess ℓ bytes of the key k. However, this does not really provide filtering, as the amount of filtering we obtain is equal to the amount of key material we have to guess. Instead, we appeal again to the first equation, and note that it also provides ℓ bytes of filtering, once ℓ bytes of k are guessed. Therefore, we obtain 2ℓ bytes of filtering, at the expense of guessing ℓ key bytes.

The attack algorithm. Choosing $\ell = 4$, this allows mounting the attack described in Algorithm 6.

Since the match checked in the hash table is a 64-bit filtering condition, on expectation a single value of i is suggested for each value of j. As each match yields a suggestion for the entire key, any random match is almost surely discarded using a single additional encryption operation. (The probability that at least one wrong candidate pair is not discarded is as low as 2^{-32}, and thus, can be neglected.) On the other hand, the data set contains a slid pair with probability $1 - (1 - 2^{-128})^{2^{128}} \approx 0.63$, and for the correct guess of $k_{SR(\mathrm{Col}(0))}$, each slid pair suggests the correct value of the secret key.

Therefore, the attack requires data complexity of 2^{64} known plaintexts, memory complexity of 2^{64}, and time complexity of 2^{96} encryptions, and succeeds with probability 63%.

The attack applies to any variant of 1-KSAt in which the transformations S,A are publicly known, including variants with *complete diffusion*. Indeed, the exact structure of AES (or more generally, the incomplete diffusion of the MixColumns transformation) are not used in the attack at all. Therefore, we obtain an attack with data complexity of $2^{n/2}$ known plaintexts, memory complexity of $2^{n/2}$, time complexity of $2^{3n/4}$ encryptions, and succeeds probability of 63%.

We note that the time complexity can be somewhat reduced by choosing another value of ℓ and using two plaintext structures of different sizes. For example, in the case of AES, the time complexity can be reduced to 2^{88}, by guessing 5 key bytes (instead of 4), taking two different structures of plaintexts – $\mathcal{T}_P$ of size 2^{84} and $\mathcal{T}_Q$ of size 2^{44}, and searching for slid pairs of the form (P_i, Q_j) where $P_i \in \mathcal{T}_P$ and $Q_j \in \mathcal{T}_Q$. However, this leads to a significant increase in the data and memory complexities (in the case of AES we described – to 2^{84}), and thus, this tradeoff does not seem profitable.

7 Summary and Conclusions

In this paper we studied slide attacks on *almost self similar* constructions, in which the symmetry is broken by the last round. As a study case, we concentrated on SP networks, in which such a symmetry break is inherent due to the final key whitening step, and especially, on AES-type constructions. We devised four new techniques: slid sets, hypercube of slid pairs, suggestive plaintext structures and substitution slides. Using these new techniques, we offered attacks against various general SPN schemes of different key periods, with different structures of the last round, with known or secret S-boxes, and with full or an incomplete diffusion.

Open problems left for further work include:

- Use the techniques proposed in the paper to attack other general SPN constructions.
- Find other types of slide attacks on almost self similar constructions.
- Find (lightweight) block ciphers, with periodic key schedule, susceptible to these attacks.

Acknowledgements

The research was partially supported by European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. Orr Dunkelman was supported in part by the Israel Ministry of Science and Technology, the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by the Israeli Science Foundation through grant No. 880/18.

References

1. Adomnicai, A., Berger, T.P., Clavier, C., Francq, J., Huynh, P., Lallemand, V., Gouguec, K.L., Minier, M., Reynaud, L., Thomas, G.: Lilliput-AE: a New Lightweight Tweakable Block Cipher for Authenticated Encryption with Associated Data. Submission to the NIST's lightweight cryptographic standardization process (2019)

2. Aerts, W., Biham, E., Moitie, D.D., Mulder, E.D., Dunkelman, O., Indesteege, S., Keller, N., Preneel, B., Vandenbosch, G.A.E., Verbauwhede, I.: A Practical Attack on KeeLoq. J. Cryptology 25(1), 136–157 (2012)
3. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the Indifferentiability of Key-Alternating Ciphers. In: Advances in Cryptology - CRYPTO 2013. LNCS, vol. 8042, pp. 531–550. Springer (2013)
4. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A Block Cipher for Low Energy. In: Advances in Cryptology - ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer (2015)
5. Bar-On, A., Biham, E., Dunkelman, O., Keller, N.: Efficient Slide Attacks. J. Cryptology 31(3), 641–670 (2018)
6. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. IACR Trans. Symmetric Cryptol. 2019(1), 5–45 (2019)
7. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology 7(4), 229–246 (1994)
8. Biham, E., Dunkelman, O., Keller, N.: A Simple Related-Key Attack on the Full SHACAL-1. In: Topics in Cryptology - CT-RSA 2007. LNCS, vol. 4377, pp. 20–30. Springer (2007)
9. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer (1993)
10. Biryukov, A., Wagner, D.: Slide Attacks. In: Fast Software Encryption, FSE '99. LNCS, vol. 1636, pp. 245–259. Springer (1999)
11. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Advances in Cryptology - EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer (2000)
12. Brier, E., Peyrin, T., Stern, J.: BPS: a Format-Preserving Encryption Proposal. Available online at `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf` (2010)
13. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Fast Software Encryption, FSE '97. LNCS, vol. 1267, pp. 149–165. Springer (1997)
14. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
15. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Reflections on slide with a twist attacks. Des. Codes Cryptography 77(2-3), 633–651 (2015)
16. Dunkelman, O., Keller, N.: The effects of the omission of last round's MixColumns on AES. Inf. Process. Lett. 110(8-9), 304–308 (2010)
17. Dunkelman, O., Keller, N., Shamir, A.: Slidex Attacks on the Even-Mansour Encryption Scheme. J. Cryptology 28(1), 1–28 (2015)
18. Durak, F.B., Vaudenay, S.: Breaking the FF3 Format-Preserving Encryption Standard over Small Domains. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017. LNCS, vol. 10402, pp. 679–707. Springer (2017)
19. Furuya, S.: Slide Attacks with a Known-Plaintext Cryptanalysis. In: Information Security and Cryptology - ICISC 2001. LNCS, vol. 2288, pp. 214–225. Springer (2001)
20. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.X.: Block Ciphers That Are Easier to Mask: How Far Can We Go? In: Cryptographic Hardware and Embedded Systems, CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer (2013)
21. Gorski, M., Lucks, S., Peyrin, T.: Slide Attacks on a Class of Hash Functions. In: Advances in Cryptology - ASIACRYPT 2008. LNCS, vol. 5350, pp. 143–160. Springer (2008)

22. Grassi, L.: MixColumns Properties and Attacks on (Round-Reduced) AES with a Single Secret S-Box. In: Topics in Cryptology - CT-RSA 2018. LNCS, vol. 10808, pp. 243–263. Springer (2018)
23. Grassi, L.: Mixture Differential Cryptanalysis: a New Approach to Distinguishers and Attacks on round-reduced AES. IACR Trans. Symmetric Cryptol. 2018(2), 133–160 (2018)
24. Grassi, L., Rechberger, C., Rønjom, S.: Subspace Trail Cryptanalysis and its Applications to AES. IACR Trans. Symmetric Cryptol. 2016(2), 192–225 (2016)
25. Grossman, E.K., Tucherman, B.: Analysis of a Weakened Feistel-like Cipher. In: Proceedings of International Conference on Communications 1978. pp. 46.3.1–46.3.5 (1978)
26. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Cryptographic Hardware and Embedded Systems, CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer (2011)
27. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: Printcipher: A block cipher for ic-printing. In: Cryptographic Hardware and Embedded Systems, CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer (2010)
28. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Advances in Cryptology - EUROCRYPT '93. LNCS, vol. 765, pp. 386–397. Springer (1993)
29. Robshaw, M.J.B.: Searching for Compact Algorithms: cgen. In: Progressin Cryptology - VIETCRYPT 2006. LNCS, vol. 4341, pp. 37–49. Springer (2006)
30. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128-bit Block Cipher (1998), `https://www.schneier.com/twofish.html`, AES Candidate and Finalist
31. Sun, B., Liu, M., Guo, J., Qu, L., Rijmen, V.: New Insights on AES-Like SPN Ciphers. In: Advances in Cryptology - CRYPTO 2016. LNCS, vol. 9814, pp. 605–624. Springer (2016)
32. Tiessen, T., Knudsen, L.R., Kölbl, S., Lauridsen, M.M.: Security of the AES with a Secret S-Box. In: Leander, G. (ed.) Fast Software Encryption, FSE 2015. LNCS, vol. 9054, pp. 175–189. Springer (2015)

A Additional Attacks Using Slid Sets

We now show that the slid sets technique can be used to attack 3-KSAfpi with complexity $\max\{2 \cdot 2^{(n+m)/2}, 2^{2m}\}$ (and in the case of 3K-AES in time 2^{81}) and attack 2-KSAtpi with complexity $\max\{2 \cdot 2^{(n+m)/2}, 2^{2m}\}$. These attacks complement the slid set attacks presented in Section 3.

A.1 Slid sets attack on 3-KSAfpi

In this section we show that a modification of the attack presented in Section 3.2 can be used to break 3-KSAf with incomplete diffusion. The modification in the structure of the attack is very minor. However, the size of the Λ-sets is increased very significantly, and so is the complexity of the attack.

The setting. For the sake of helping readability, we present the attack in the special case of 3K-AESf. For concreteness, we assume that the number of rounds is 2 mod (3). However, it will be clear from the attack that the same argument applies for other numbers of rounds as well.

Assume that (P_i, Q_j) is a slid pair. This means that

$$Q_j = MC \circ SR \circ SB \circ ARK_3 \circ MC \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i),$$

where ARK_ℓ denotes key addition with the subkey k_ℓ. As in the previous sections, we can peel off unkeyed operations by applying $SB^{-1} \circ SR^{-1} \circ MC^{-1}(Q_j)$, interchanging the operations ARK_3 and MC, at the expense of replacing the subkey k_3 with $MC^{-1}k_2$. This allows us to peel off MC and SR as well, and obtain the equation

$$\tilde{Q}_j = ARK'_3 \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i). \tag{14}$$

Similarly, the relation between the corresponding ciphertexts can be simplified to

$$\tilde{D}_j = ARK'_3 \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(\bar{C}_i).$$

Finding pairs of slid sets using a variant of the attack of Section 3.2. Note that the transformation from P_i to $\tilde{Q}_j$ consists of application of four independent functions on (shifted) columns of the state (super S-boxes), and the same goes for the transition from $\bar{C}_i$ to $\tilde{D}_j$. Hence, a variant of the attack of Section 3.2 can be applied to our version as well – the only difference is that we have to consider Λ-sets of size 2^{32} (instead of 2^8). This allows us to find pairs of slid sets (U, V) of size 2^{32}, with data, memory and time complexity of 2^{80}.

Retrieving the key from a pair of slid sets. This step is similar to the corresponding step in the attack on 2K-AES presented in Section 3.1, but requires two additional observations.

We consider each of the four parallel 32-bit functions separately. For each of them, we have

$$\tilde{D}_j = ARK'_3 \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(\bar{C}_i). \tag{15}$$

If we guess the 32 subkey bits used in each of the operations ARK_1, ARK_2, and ARK'_3, then we can compute the right hand side of (15) for all $\bar{C}_i \in \bar{W}$ and check the key guess by checking whether the multiplicities of the obtained values in $\tilde{X}^j$ conform to the respective multiplicities of the $\bar{C}_i$'s in $\bar{W}$.

However, guessing all three subkeys and going over all values in W leads to time complexity of $(2^{32})^3 \cdot 2^{32} = 2^{128}$. Hence, we need a more refined procedure. We use two observations.

First, we observe that there is no need to check the multiplicities of all elements in $\bar{W}$. It is sufficient to take *a single element* $\bar{C}_i$ (or a few elements) whose multiplicity is the maximal among the elements of $\bar{W}$ and check whether the corresponding value $ARK'_3 \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(\bar{C}_i)$ has maximal multiplicity among the elements of $\tilde{X}$. Note that while this check does require going over all elements of $\bar{W}$ and $\tilde{X}$ in order to count their multiplicities, this count can be performed before the key guessing step. After the key guessing, we have to check only a few elements of $\bar{W}$, and so, we save a factor of almost 2^{32}. It it easy to verify that performing this check for a few elements provides a sufficient amount of filtering, while having complexity of only a few operations (for each key guess).

Second, we observe that there is no need to guess the subkey used in the operation ARK_3', as we can bypass it by XORing pairs of values. Note that the value $\tilde{D}_j \oplus \tilde{D}_{j'}$ does not depend on the key k_3, since the operation ARK_3' cancels out. Hence, we can consider a set of a few elements whose multiplicity is almost maximal among the elements of $\bar{W}$ (here, it is important to have at least two elements), guess the subkeys k_1, k_2, and compute the XORs of values of the form $SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(\bar{C}_i)$ with one of them. These values can be found without guessing k_3, yet they carry a sufficient amount of information for checking the key guess.

Hence, the key guess of k_1, k_2 can be checked in slightly more than 2^{64} operations, which are dominated by the complexity of encrypting the plaintexts at the beginning of the attack.

The complexity of the attack. In total, we obtain an attack on 3K-AESf, with data, memory and time complexity of 2^{81} and success probability of 63%.

The same attack can be applied to any variant of 3-KSAfp with incomplete diffusion. Denoting the size of the block on which A operates by m, the data and memory complexity of the attack is $2 \cdot 2^{(n+m)/2}$, and the time complexity is $\max(2 \cdot 2^{(n+m)/2}, 2^{2m})$. Here, 2^{2m} is the complexity of recovering the secret keys from slid sets.

A.2 Slid sets attack on 2-KSAtp

In this section we show that the slid sets attack can be applied also to 2-KSAt with incomplete diffusion.

The setting. For the sake of helping readability, we present the attack in the special case of 2K-AES with an even number of rounds. However, this assumption is not essential in the attack.

First, we would like to simplify the problem. Assume that (P_i, Q_j) is a slid pair. This means that

$$Q_j = MC \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i),$$

where ARK_ℓ denotes key addition with the subkey k_ℓ. As described in Section 3.1, this equation can be simplified to

$$\tilde{Q}_j = ARK_2' \circ SB \circ ARK_1(P_i). \tag{16}$$

Due to the omission of MixColumns in the last round, the relation between the corresponding ciphertexts is much more complex. We have

$$D_j = ARK_1 \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1 \circ MC \circ ARK_1(C_i).$$

We can interchange the order of the operations ARK_1, MC, in exchange for replacing the key addition of k_1 with key addition of $MC(k_1)$. Then we can peel off the unkeyed operation MC and unify the two key additions into addition of the subkey $(M+I)k_1$, which we denote by ARK_1'. From the other side, we can interchange the operations ARK_1 and SR and peel off SR. We obtain:

$$\tilde{D}_j = ARK_1 \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1'(\bar{C}_i), \tag{17}$$

where $\bar{C}_i = MC(C_i)$ and $\tilde{D}_j = SR^{-1}(D_j)$.

Here, there is a significant difference between Eq. (16) and (17). While the transformation from P_i to $\tilde{Q}_j$ (represented by Eq. (16)) consists of application of 16 independent functions on the bytes of the state, the transition from $\bar{C}_i$ to $\tilde{D}_j$ consists of application of four independent functions on (inverse shifted) columns of size 32 bits.

Applying the attack of Appendix A.1. One possible way to proceed at this stage is to apply the attack on 3K-AES presented in Appendix A.1 without change. Indeed, as the relation between ciphertexts in our case is equivalent to the relation in the attack on 3K-AES and the relation between plaintexts is even simpler, the attack on 3K-AES can be applied without change in this case, resulting in an attack with data, memory, and time complexity of 2^{81}.

Improved attack using smaller Λ-sets. The simpler relation between P_i and $\tilde{Q}_j$ (represented by Eq. (16)) allows us to reduce the data complexity of the attack, using smaller Λ-sets. Note that Eq. (16) is the same as was the relation between P_i and $\tilde{Q}_j$ in the attack on 2K-AESfp presented in Section 3.1. Hence, as was shown in that section, we can use Λ-sets of size 2^8 and have the property that if $\tilde{Q}_j$ corresponds to the slid counterpart of P_i then the Λ-set $\tilde{V}$ that contains $\tilde{Q}_j$ corresponds to 'the slid set counterpart' of the Λ-set U that contains P_i.

Using the statistic of sequences-of-sequences $\{a_q^\ell\}_{q=0,1,\ldots}$ and $\{b_q^\ell\}$ defined in Section 3.1, applied to a structure of size 2^{29} composed of 2^{21} Λ-sets of size 2^8 each, we can obtain a sufficient amount of filtering, while reducing the complexity by a factor of 8. It seems that the complexity can be reduced even further, using other statistics.

Once a pair of slid sets is detected, the secret key can be found with complexity of about 2^{64} (which is negligible with respect to other steps of the attack), using the attack of Appendix A.1.

Therefore, we obtain an improved attack on 2K-AESt, with data, memory, and time complexity of 2^{78}.

Attacking 2-KSAtpi. The same attack applies to any variant of 2-KSAtp with an incomplete diffusion. Like in the case of 3-KSAfpi, we can obtain an attack with data and memory complexity of $2 \cdot 2^{(n+m)/2}$, and time complexity of $\max(2 \cdot 2^{(n+m)/2}, 2^{2m})$. The data and memory complexities can be reduced using smaller Λ-sets, as described in the case of 2K-AES. However, as the exact reduction depends on the relation between n, m, s, we omit it here.

B Slide Attack using Key Guessing

In this section we present a slide attack on 2-KSAf, based on specifically chosen data structures and key guessing. The complexity of the attack slightly depends on the parity of the number of rounds. For an odd number of rounds, the complexity is $(n/s)2^{n/2}$ chosen plaintexts, the memory complexity is $(n/s)2^{n/2}$ and the time complexity is $2^{(n/2)+s}$ (in the case of 2K-AESf: $2^{68}, 2^{68}$, and 2^{72}, respectively). For an even number of rounds, the complexity is a bit higher, and there are possible tradeoffs between the data, memory and time complexities.

The setting. For sake of simplicity, we consider 2K-AES with a MixColumns operation in the last round. We assume that the number of rounds is *odd*; the even case is considered below. First, we would like to simplify the problem.

Assume that (P_i, Q_j) is a slid pair. This means that

$$Q_j = MC \circ SR \circ SB \circ ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i),$$

where ARK_ℓ denotes key addition with the subkey k_ℓ. As usual, we can peel off unkeyed operations by denoting $Q'_j = SB^{-1} \circ SR^{-1} \circ MC^{-1}(P'_j)$, and obtain the equation $Q'_j = ARK_2 \circ MC \circ SR \circ SB \circ ARK_1(P_i)$. Furthermore, we can interchange the operations ARK_2 and MC, at the expense of replacing the subkey k_2 with $M^{-1}k_2$, where the matrix M denotes parallel application of MC in four columns. This allows to peel off MC as well, and obtain the equation

$$\tilde{Q}_j = ARK'_2 \circ SR \circ SB \circ ARK_1(P_i). \tag{18}$$

In a similar way, the relation between the corresponding ciphertexts can be simplified to

$$\tilde{D}_j = ARK'_2 \circ SR \circ SB \circ ARK_1(\bar{C}_i). \tag{19}$$

The important gain from obtaining the simplified equations is that now the transformation from P_i to $\tilde{Q}_j$ consists of application of 16 independent functions on the bytes of the state. The same goes for the transition from $\bar{C}_i$ to $\tilde{D}_j$. This will play a significant role in the attack.

Attaching friend pairs. As in some of our previous attacks, our main challenge is to attach to each candidate slid pair several friend pairs, such that if the initial pair is indeed a slid pair then so are the friend pairs as well. Since the transition between P_i and $\tilde{Q}_j$ involves a non-linear operation, we cannot find friend pairs 'for free', as we did in Sections 5 and 6. However, as the relation consists of 16 byte functions applied in parallel, a guess of a single key byte is sufficient for finding friend pairs.

Indeed, let $(P_i, \tilde{Q}_j)$ for some $P_i \in \mathcal{T}_P$ and $\tilde{Q}_j \in \tilde{\mathcal{T}_Q}$ be a slid pair, and let $P_{i,a} = P_i \oplus a$, where a has a non-zero value only in byte 0. Then the slid counterpart of $P_{i,a}$ in $\tilde{\mathcal{T}_Q}$ must be of the form $\tilde{Q}_j \oplus a'$, where a' has a non-zero value only in byte 0. Moreover, if the byte $(k_1)_0$ is known to the adversary, she can compute a' and thus determine the slid counterpart of $P_{i,a}$. Indeed, Eq. (18) implies that if $\tilde{Q}_{i,a}$ denotes the slid counterpart of $P_{i,a}$, then

$$\begin{aligned}\tilde{Q}_{i,a} \oplus \tilde{Q}_i &= ARK_2' \circ SR \circ SB \circ ARK_1(P_{i,a}) \oplus ARK_2' \circ SR \circ SB \circ ARK_1(P_i)\\ &= SR(SB \circ ARK_1(P_{i,a}) \oplus SB \circ ARK_1(P_i).\end{aligned}$$

Therefore, as the knowledge of $(k_1)_0$ allows computing byte 0 of $SB \circ ARK_1(P_{i,a})$ and of $SB \circ ARK_1(P_i)$, it allows computing byte 0 of $\tilde{Q}_{i,a} \oplus \tilde{Q}_i$, and thus, of $\tilde{Q}_{i,a}$ as well. All other bytes of $\tilde{Q}_{i,a}$ are simply equal to those of $\tilde{Q}_i$.

In a similar way, for any element $\tilde{Q}_j \oplus a' \in \tilde{\mathcal{T}_Q}$, where a' has a non-zero value only in byte 0, an adversary that knows $(k_1)_0$ can find a' such that $P_{i,a'}$ is its slid counterpart.

The attack algorithm. In our attack, we consider two structures of plaintexts. The first structure $\mathcal{T}_P$ consists of 2^{64} plaintexts in which the value of inverse shifted columns $SR^{-1}(\mathrm{Col}(2,3))$ is fixed to 0 and the value in $SR^{-1}(\mathrm{Col}(0,1))$ assumes all 2^{64} possible values. The second structure $\mathcal{T}_Q$ is chosen such that the corresponding structure $\tilde{\mathcal{T}_Q}$ consists of 2^{68} elements in which bytes $1,2,3,4,5,6,7$ are fixed to 0, byte 0 attains the values $0_x, 1_x, \ldots, F_x$, and Columns 2,3 assume all 2^{64} possible values.

For each $Q_j \in \mathcal{T}_Q$, we attach to Q_j 15 friends $Q_{j,l}$, where for each $l = 1_x, 2_x, \ldots, F_x$, $Q_{j,l}$ is the element of $\mathcal{T}_Q$ that corresponds to $\tilde{Q}_j \oplus l \in \tilde{\mathcal{T}_Q}$. We compute the value of byte 0 in the corresponding ciphertexts $D_{j,l}$ and store the 15-byte values $((D_j \oplus D_{j,l})_0)_{l=1,\ldots,15})$ in a hash table.

In the other direction, for each guess of byte 0 of k_1 and for each P_i, we find 15 friends $P_{i,l}$ such that if the slid counterpart of P_i in $\tilde{\mathcal{T}_Q}$ is Q_j then the slid counterpart of $P_{i,l}$ in $\tilde{\mathcal{T}_Q}$ is $Q_{j,l} = Q_j \oplus l$. We consider the corresponding ciphertexts $\{C_{i,l}\}_{l=1,\ldots,15}$ and use the knowledge of $(k_1)_0$, to compute byte 0 of

$$SR \circ SB \circ ARK_1(\bar{C}_i) \oplus SR \circ SB \circ ARK_1(\bar{C}_{i,l}),$$

for each $l = 1, \ldots, 10$. As by Eq. (19), this value is equal to $(D_j \oplus D_{j,l})_0$, we can use it to check for a match in the precomputed hash table. This yields a 120-bit filtering that allows finding the correct slid pair (P_i, Q_j) and discarding the wrong candidates. Once a slid pair (P_i, Q_j) is detected, the keys k_1, k_2 can be extracted easily from the corresponding ciphertext pair (C_i, D_j) and a few of its friend pairs, by analyzing Eq. (19) in each byte separately.

The attack algorithm is the following.

Algorithm 7 A Slide Attack on 2K-AES with MixColumns in the last round, using key guessing

Ask for the encryption two structures $\mathcal{T}_P, \mathcal{T}_Q$ of chosen plaintexts, defined as described above.
for Each plaintext $Q_j \in \mathcal{T}_Q$ **do**
  Compute the 15 friend values $Q_{j,l}$ as described above and the corresponding values $\tilde{D}_{j,l}$,
  Store in a hash table the 15-byte sequence of values $((D_j \oplus D_{j,l})_0)_{l=1,\ldots,15}$, along with the value P_i used to create them.
for Each guess of $(k_1)_0$ **do**
  for Each plaintext/ciphertext pair (P_i, C_i) **do**
    Compute the 15 'friend values' $P_{i,l}$ as described above and the corresponding ciphertexts $C_{i,l}$,
    Compute the 15-byte values $((SR \circ SB \circ ARK_1(\bar{C}_i) \oplus SR \circ SB \circ ARK_1(\bar{C}_{i,l}))_0)_{l=1,\ldots,15}$, and check for a match in the hash table.
  for Each collision in the table **do**
    Deduce that each $(P_{i,l}, Q_{j,l})$ is a slid pair and use Eq. (19) to retrieve k_1, k_2.

In total, the attack examines 2^{128} candidate slid pairs (P_i, Q_j). As the probability of a wrong pair to lead to a collision in the table is 2^{-120}, with overwhelming probability all wrong pairs are discarded instantly. On the other hand, by the same argument as for the attack in Section 5, the data set must contain a slid pair, and each slid pair leads to a collision in the hash table, and thus, is detected. Hence, the attack has 100% success probability. The data complexity is 2^{68} chosen plaintexts, the memory complexity is 2^{68}, and the time complexity is about 2^{72}. We note that the data complexity can be slightly reduced to about 2^{67} by rebalancing the sizes of the data structures T, T', at the expense of increasing the time complexity to 2^{74}.

The same attack applies to any variant of 2-KSAf with publicly known S, A and an odd number of rounds, including variants with a full diffusion. The data complexity is about $(n/s)2^{n/2}$ chosen plaintexts (since each 'friend' provides an s-bit filtering, and we need an n-bit filtering), the memory complexity is $(n/s)2^{n/2}$ and the time complexity is $2^{(n/2)+s}$.

Attack on 2-KSAf with an even number of rounds. The only difference between the 'odd' and the 'even' cases is that in the 'even' case, in Eq. (19), the roles of k_1, k_2 are interchanged, and so, the knowledge of $(k_1)_0$ cannot be used.

One possible solution is guessing also the byte (k_{2_0}). The rest of the attack can be performed without any modification, and as a result, the time complexity is increased by a factor of 2^s and the other parameters are unchanged.

An alternative solution is to guess (k_{1_0}), but reduce the time complexity by re-balancing the sizes of the structures $\mathcal{T}_P, \mathcal{T}_Q$. By taking $\mathcal{T}_P$ of size $2^{(n/2)-s+\log(n/s)/2}$ and $\mathcal{T}_Q$ of size $2^{(n/2)+s+\log(n/s)/2}$, the data, memory, and time complexities of the attack become all equal to $2^{(n/2)+s+\log(n/s)/2+1}$.

A third solution is to perform the key guessing in Eq. (19) on the side of D, namely, to guess the byte $k'_1(0)$, and then for each j and for each guess of $k'_1(0)$, to compute and store in the hash table the values

$$(SB^{-1} \circ SR^{-1} \circ ARK'_1(\tilde{D}_j) \oplus SB^{-1} \circ SR^{-1} \circ ARK'_1(\tilde{D}_{j,l}))_0,$$

which, by the modified Eq. (19), are equal to the corresponding values $(\bar{C}_i \oplus \bar{C}_{i,l})_0$. As a result, the data complexity is unchanged, the time complexity is slightly increased, and the memory complexity is increased by a factor of 2^s. The complexity can be reduced a bit by re-balancing between the sizes of the structures $\mathcal{T}_P$ and $\mathcal{T}_Q$, and so we get data complexity of $2\sqrt{n/s}2^{n/2}$ chosen plaintexts, and memory complexity and time complexity of $\sqrt{n/s}2^{(n/2)+s}$.

In the case of 2K-AES, the first version yields data and memory complexity of 2^{68} and time complexity of 2^{80}, the second version yields data, memory and time complexity of 2^{75}, and the third version yields data complexity of 2^{67} and memory and time complexities of 2^{74}.

C Slide Attack using Plaintext/Ciphertext Collision

In this section we present a *known plaintext* slide attack on 2-KSAf with an odd number of rounds, based on a partial collision between a plaintext and the corresponding ciphertext.

The key observation behind the attack. Like in Appendix B, we present the attack in the special case of full 2K-AES. As was shown in Appendix B, if (P_i, Q_j) is a slid pair and C_i, D_j are the corresponding ciphertexts, then we have

$$\tilde{Q}_j = ARK'_2 \circ SR \circ SB \circ ARK_1(P_i) \qquad \tilde{D}_j = ARK'_2 \circ SR \circ SB \circ ARK_1(\bar{C}_i). \quad (20)$$

The key observation behind the attack is that the relation between $\tilde{Q}_j$ and P_i is *exactly the same* as the relation between $\tilde{D}_j$ and $\bar{C}_i = MC \circ SR \circ SB(C)$. That is, if we denote $f = ARK'_2 \circ SR \circ SB \circ ARK_1$, then we simply have

$$\tilde{Q}_j = f(P_i) \qquad \tilde{D}_j = f(\bar{C}_i). \quad (21)$$

Moreover, the function f consists of partitioning the state into bytes and applying (distinct) permutations to the bytes in parallel.

This implies that for any byte ℓ, we have

$$(P_i)_\ell = (\bar{C}_i)_\ell \Leftrightarrow (\tilde{Q}_j)_\ell = (\tilde{D}_j)_\ell.$$

Thus, if we restrict ourselves to 'special' slid pairs (P_i, Q_j), for which the equality $(P_i)_\ell = (\bar{C}_i)_\ell$ holds for several bytes, we can use this equality to filter wrong candidates of Q_j and detect the right slid pair.

The attack algorithm. In the attack, we consider a set $\mathcal{T}_P$ (which we also call $\mathcal{T}_Q$) of 2^{82} known plaintexts.

At the first stage, we consider the corresponding ciphertexts, compute $\bar{C}_i$ for each $P_i \in \mathcal{T}_P$, and discard all plaintexts P_i except those for which the equality $(P_i)_\ell = (\bar{C}_i)_\ell$ holds in exactly 6 bytes. We divide the remaining plaintexts into $M = \binom{16}{6} \approx 2^{13}$ classes $\{P_\ell\}_{\ell=1,\dots,M}$, according to the location of the bytes in which equality is satisfied. Each class contains approximately $2^{82} \cdot 2^{-48} = 2^{34}$ elements.

At the second stage, we use the set $\mathcal{T}_P$ "in the role of" $\mathcal{T}_Q$. That is, for each $P_j \in \mathcal{T}_P$, we denote it by Q_j, and compute the values $\tilde{Q}_j$ and $\tilde{D}_j$. Then, we discard all plaintexts Q_j except those for which the equality $(\tilde{Q}_j)_\ell = (\tilde{D}_j)_\ell$ holds in exactly 6 bytes. We divide the remaining plaintexts into M classes $\{Q_\ell\}_{\ell=1,\dots,M}$, according to the location of the bytes in which equality holds.

Now, we observe that for any remaining P_i, some Q_j can be its slid counterpart only if it was not discarded. Moreover, if for P_i equality holds in bytes $\ell_1, \ell_2, \dots, \ell_6$, then for Q_j equality must hold in bytes $SR(\ell_1, \dots, \ell_6)$. Hence, we can detect all slid pairs in the data by going over all remaining P_i's, and for each of them, trying all remaining Q_j in the corresponding class.[7] For each such candidate pair (P_i, Q_j), by XORing the two equations of (20) we obtain the input and output differences of the SubBytes operation. Using a single lookup into a precomputed difference distribution table (DDT) of SubBytes, this yields the actual values, and hence, the value of the subkey k_1, and thus also the value of k_2. The suggested values can be then checked by trial encryption.

The attack algorithm is given in Algorithm 8.

The data complexity of the attack is 2^{82} known plaintexts. After the initial discarding stage, we are left with about $2^{13} \cdot 2^{34} = 2^{47}$ plaintexts P_i. For each of them, the probability that it has a slid counterpart in the data set is about $1-(1-2^{-128})^{2^{82}} \approx 2^{-46}$. Hence, on expectation the non-discarded data contains $2^{47} \cdot 2^{-46} = 2$ slid pairs, and so, using a Poisson approximation, the data set contains at least one slid pair with probability $1 - e^{-2} \approx 0.86$. As any right slid pair suggests the correct key, the success probability of the attack is 86%.

In the attack, we check about $2^{34} \cdot 2^{13} \cdot 2^{34} = 2^{81}$ candidate pairs (P_i, Q_j), and for each of them, we get a suggestion of k_1, k_2 in time complexity of a few operations and check the suggestion with a single trial encryption. Hence, the total time complexity of the attack is 2^{82} (dominated by encrypting the data). The memory complexity is as low as 2^{47}, since we can discard instantly all plaintexts for which the partial plaintext/ciphertext collision we search for is not satisfied.

The same attack works for any variant of 2-KSAf with public S, A, including the 'complete diffusion' variant. If the seek equality in q S-boxes, then the amount of data D needed for containing a slid pair with high probability satis-

[7] We note that checking each Q_j separately may seem inefficient. However, there can be no shortcut here, as for any Q_j in the corresponding class, there exists (on expectation) one value of k_1, k_2 for which Q_j is the slid counterpart of P_i; hence, we have to check each of them separately.

Algorithm 8 A known plaintext slide attack on full 2K-AES using plaintext/ciphertext collision

Ask for the encryption of 2^{82} known plaintexts.
Initialize $\binom{16}{6}$ empty lists $L(\{\ell_1, \ldots, \ell_6\})$ for all 6-element sets $\{\ell_1, \ldots, \ell_6\} \subset \{1, \ldots, 16\}$.
for each plaintext/ciphertext pair (P_i, C_i) **do**
  Compute $\bar{C}_i$ as described above,
  if $(P_i)_{\ell_i} = (\bar{C}_i)_{\ell_i}$ holds for exactly 6 byte positions ℓ_i **then**
    Add P_i to $L(\{\ell_1, \ldots, \ell_6\})$.
for each plaintext/ciphertext pair (Q_j, D_j) **do** $\triangleright$ note that we go again over the same plaintexts
  Compute $\tilde{Q}_j, \tilde{D}_j$ as described above,
  if $(\tilde{Q}_j)_{\ell_i} = (\tilde{D}_j)_{\ell_i}$ holds in exactly 6 bytes positions ℓ_i **then**
    for each $P_i \in L(SR^{-1}(\{\ell_1, \ldots, \ell_6\}))$ **do**
      Assume that (P_i, Q_j) is a slid pair and use Eq. (20) to retrieve (k_1, k_2) as described above,
      Check the suggestion of (k_1, k_2) by trial encryption.

fies $\binom{n/s}{q} 2^{-qs} D^2 \approx 2^n$. For a given amount D of data, the memory complexity is $2\binom{n/s}{q} 2^{-qs} D$ and the time complexity is $\max(D, \binom{n/s}{q}(2^{-qs}D)^2)$. The exact complexities are determined by fine tuning, using the concrete values of n, s.