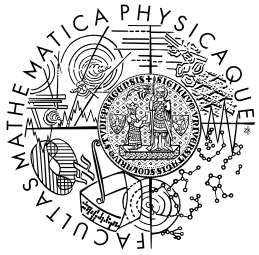


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Karolína Zenknerová

Slide Attacks

Department of Algebra

Supervisor of the bachelor thesis: Dr. rer. nat. Faruk Gologlu

Study programme: Mathematics for Information Technologies

Study branch: Mathematics for Information Technologies

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor for consultations and advices.

Title: Slide Attacks

Author: Karolína Zenknerová

Department: Department of Algebra

Supervisor: Dr. rer. nat. Faruk Gologlu, Department of Algebra

Abstract: A slide attack is an attack against block ciphers which have all rounds the same. The success and the complexity of the attack is independent on the number of rounds. The original slide attack was mainly used on a Feistel structure, but very rarely on SPN networks, because in general, SPN networks have the last round diferent. This property does not allow to use normal slide attack. In Dunkelman et al. [2020] are introduced new slide attacks (four of them) which focus on SPN networks and they overcome a problem of the last round.

In this thesis we explain main idea of the original slide attack and the main idea of two new slide attacks – a slid sets attack and a slide attack using a hypercube of slid pairs. In both these attacks we create and use special structures of plaintexts and ciphertexts to get more pairs of plaintexts which we call slid pairs. Moreover, we explain some selected parts of two new slide attacks and we compute the complexity.

The thesis is based on selected chapters in Dunkelman et al. [2020].

Keywords: symmetric cryptography, attacks on block ciphers

Contents

Introduction

Many block ciphers are composed of r same components — rounds, which usually use different round keys. One such round can be cryptographically weak and the cipher is still secure because of the large number of rounds. Topic of this thesis are slide attacks which attack ciphers whose encryption scheme is a sequence of the same rounds. Rounds even use the same round key or a sequence of the same keys is repeated. Slide attack has an interesting property: it can attack ciphers with arbitrary number of rounds without change of the complexity.

During the attack we need to find a special pair of plaintexts, which is called a slid pair (this will be described more precisely in the thesis). Once this pair is found, only one weak round is attacked.

The original slide attack [Biryukov and Wagner, 1999] was mainly used on Feistel structure, but very rarely on SPN networks, because in general SPN networks have the last round diferent or omitted. This property does not allow to use normal slide attack.

In Dunkelman et al. [2020] authors introduce four new slide attacks which attack SPN networks and overcome the problem of the last round. To attack SPN we need more slid pairs. How to find them and how to use them to deduce the key (or keys) is described in Dunkelman et al. [2020].

The aim of this thesis is to explain the main idea of the original slide attack [Biryukov and Wagner, 1999] as well as the main idea of two selected new slide attacks [Dunkelman et al., 2020]. The first chapter presents some preliminaries that the reader should be familiar with before reading the thesis and sets up terminology. In the second chapter we explain main idea of generic slide attacks [Biryukov and Wagner, 1999]. In the third chapter we will look more closely on two new slide attacks on SPN and we explain selected parts of the algorithm on AES without a key schedule. We focus on AES with one key and secret S-boxes, which will be explained in the thesis. We compute the complextity of attacks as well. We extend some computations or explanations from Dunkelman et al. [2020].

1. Preliminaries

Some preliminaries are required for better understanding the thesis and for mathematical completeness. In this chapter we will introduce knowledge you should be familiar with before reading the thesis.

Firstly, we will introduce some basic definitions. As a source we used lecture notes written by Kozlík [c] and Bashir and book Paar and Pelzl [2010].

This thesis is about attacks on block ciphers, thus our first three definitions will explain what it means. Our first definiton is a definition of a cipher how we understand it in cryptographic terms.

Definition 1. A cipher *is a tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$*, where* $\mathcal{P}$ *is a set of plaintexts,* $\mathcal{C}$ *is a set of ciphertexts,* $\mathcal{K}$ *is a key space,* E *is an encryption algorithm* E*:* $\mathcal{P} \times \mathcal{K} \to \mathcal{C}$ *and* D *is a decryption algorithm* D*:* $\mathcal{C} \times \mathcal{K} \to \mathcal{P}$*.*

Following two definitions explain us what we mean by symmetric and block cipher.

Definition 2. *(Bashir)* A symmetric cipher *is a cipher with an invertible encryption map* E *such that* $D_k = E_k^{-1}$ *for every key* $K \in \mathcal{K}$ *and every plaintext* $P \in \mathcal{P}$*. Briefly, it uses the same key for an encryption and a decryption.*

Definition 3. *(Delfs and Knebl [2015])* A block cipher *is a cipher with symmetric encryption algorithm* E *and decryption algorithm* D *such that*

$$E(P, K) : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n,$$

where plaintext has n *bits, ciphertext has* n *bits and the lenght of the key is* k *bits.*

A block cipher has a fixed length of input (key and plaintext) and output (ciphertext). In this thesis we suppose $k = n$.

Exmples of such ciphers we will see in following sections. The following definition describes what is the product of ciphers. The definition is formal, in informal way we could say that the product of cihers is when we use an encryption algorithm of the first cipher and then of the second cipher, which creates us the new encryption algorithm.

Definition 4. *(Bashir, Kozlík [c])Let us have two ciphers* $S_1 = (\mathcal{P}_1, \mathcal{C}_1, \mathcal{K}_1, E_1, D_1)$ *and* $S_2 = (\mathcal{C}_1, \mathcal{C}_2, \mathcal{K}_2, E_2, D_2)$*. We define* the product of ciphers S_1 *and* S_2 *to be a cipher* $S_1 \times S_2 = (\mathcal{P}_1, \mathcal{C}_2, \mathcal{K}_1 \times \mathcal{K}_2, E, D)$*, where*

$$E(x, (K_1, K_2)) = E_2(E_1(x, K_1), K_2)$$

and

$$D(y, (K_1, K_2)) = D_1(D_2(y, K_2), K_1)$$

for every $x \in \mathcal{P}_1$*, every* $y \in \mathcal{C}_2$ *and every* $(K_1, K_2) \in \mathcal{K}_1 \times \mathcal{K}_2$*.*

If we repeat the encryption algorithm of one cipher n-times, we get its product which is described in the following definition.

Definition 5. *(Bashir) Let* S *be the cipher. We will use the following notation for its product* $S^n = S \times S \times S \times \cdots \times S$*,* $n \in \mathbb{N}$*.*

The following definition describes formally an iterated cipher, a round function and a key schedule. This will be needed for describing enryption schemes in the next section. Informally, an encryption algorithm of an iterated cipher is a product of the same round functions. A round functions use keys which are created by a key schedule with one key as input.

Definition 6. *(Bashir) Let us have two ciphers* $S_1 = (\mathcal{P}, \mathcal{C}, \mathcal{K}_1, E_1, D_1)$ *and* $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ *with symmetric encryption algorithms* E_1 *and* E*, where* $\mathcal{P} = \mathcal{C}$*. Let* f *be the mapping* $f : S \to S_1^n$ *and* t *be the mapping* $t : \mathcal{K} \to \mathcal{K}_1^n$*,* $n \in \mathbb{N}$*.* S *is* an iterated cipher *and* S_1 *is* a round *of* S*.* S *has* n *rounds.* t *is called* a key schedule*, it creates* n *round keys.*

In the next definition we will describe an isomorfism of two ciphers which will be needed for the definition of a commutativity of two ciphers. Informally we could say that for isomorfic ciphers is possible to transform first encryption algorithm into second encryption algorithm.

Definition 7. *(Bashir, Kozlík [c]) Let us have two ciphers* $S_1 = (\mathcal{P}, \mathcal{C}, \mathcal{K}_1, E_1, D_1)$ *and* $S_2 = (\mathcal{P}, \mathcal{C}, \mathcal{K}_2, E_2, D_2)$*. We call* S_1 *and* S_2 isomorphic *if there exists the mapping* $f_1 : \mathcal{K}_1 \to \mathcal{K}_2$ *and the mapping* $f_2 : \mathcal{K}_2 \to \mathcal{K}_1$ *such that* $E_1(x, K_1) = E_2(x, f_1(K_1))$ *and* $E_2(z, K_2) = E_1(z, f_2(K_2))$ *for every* $x \in \mathcal{P}$*, every* $z \in \mathcal{P}$*, every* $K_1 \in \mathcal{K}_1$ *and every* $K_2 \in \mathcal{K}_2$*. We denote the isomorphism by* $S_1 \simeq S_2$*.*

The following definition explains the commutativity of two ciphers. In following section about encrypting schemes we will see how a commutativity of ciphers can be used to manipulate with a key. It also can be used to shorten an encryption algorithm.

Definition 8. *(Bashir) Ciphers* S_1 *and* S_2 commute *if* $S_1 \times S_2 \simeq S_2 \times S_1$*.*

We can classify attacks by knowledge and options the attacker has. In this thesis we will see two types of attacks. They are defined below.

Definition 9. Known-plaintext attack *is a situation when the attacker is able to get some plaintexts (and encrypt them to get corresponding ciphertexts), but he can not choose which ones.*

Definition 10. Chosen-plaintext attack *is a situation when the attacker is able to choose plaintexts and he is able to encrypt them to obtain corresponding ciphertexts.*

1.1 Encryption schemes and ciphers used in the thesis

In this section we will introduce ecryption schemes used in the thesis. Again sources of information were lecture notes by Bashir (SPN), the paper Dunkelman et al. [2020] (KSA), lecture notes by Kozlík [a](Feistel) and book Knudsen and Robshaw [2011] (AES).

Firstly, we will describe SPN network and we will introduce two specific cases used in the thesis – KSA structure, which is more general and AES cipher. Secondly, we will introduce Feistel structure.

1.1.1 SPN (Substitution-permutation network)

Definition 11. *Let us have three ciphers S_1, S_2, S_3, where $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^n$.* SPN (a substitution permutation network) *is an iterated cipher, with the round $S_1 \times S_2 \times S_3$. The last round is different, it has the form $S_1 \times S_2 \times S_1$. The SPN with n rounds has the following form*

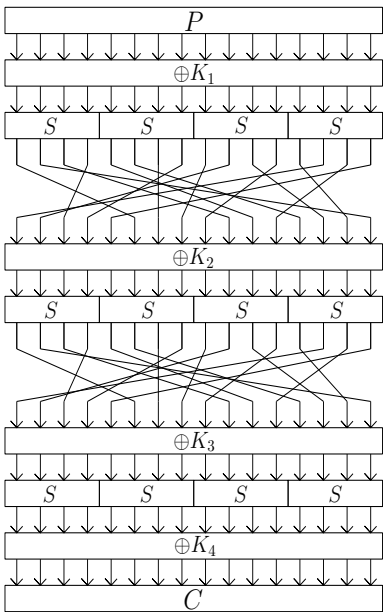
$$(S_1 \times S_2 \times S_3)^{n-1} \times (S_1 \times S_2 \times S_1).$$

Ciphers S_1, S_2, S_3 have the symmetric encryption algorithm and they can be described following way:

- $S_1 = (\mathcal{P}, \mathcal{C}, \mathcal{K}, E_1, D_1)$ – *a key addition, $E_1(x, K) = x \oplus K$, for every $x \in \mathcal{P}$ and for every $K \in \mathcal{K} = \mathbb{F}_2^n$, it is equivalent to a vector addition in $\mathbb{F}_2$*
- $S_2 = (\mathcal{P}, \mathcal{C}, \{s\}, E_2, D_2)$ – *a substitution, the key $s \in S_{\mathcal{P}}$ is a concatenation of S-boxes $s_1, s_2, \ldots, s_m$. These S-boxes are strongly non linear mappings $s_1, s_2, \ldots, s_m : \mathbb{F}_2^l \to \mathbb{F}_2^l$, $n = m \cdot l$. In other words, $s_1, s_2, \ldots s_m \in S_{\mathbb{F}_2^l}$ are substitutions on $\mathbb{F}_2^l$. E_2 is a substitution on the input such that $E_2(x, s) = s(x)$ for every $x \in \mathcal{P}$.*
- $S_3 = (\mathcal{P}, \mathcal{C}, \{\pi\}, E_3, D_3)$ – *a permutation, the key is a permutation $\pi \in S_n$ (it defines us one permutation), E_3 is a permutation on elements of the input. For the input $x \in \mathcal{P}$, $x = (x_1, x_2, \ldots, x_k)$, we have $E_3(x, \pi) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(k)})$.*

The last operation is S_1 - adding the round key. We call it a key whitening and it will be explained in the next definition. There is omitted the permutation layer S_3, in the last round, because S_1 and S_3 commute. It has no effect to have S_3 in the last round, because it can be removed out of it.

Example. As an example we have a three round SPN which we can see in the picture. The length of an input is $n = 16$ bits and the length of an input of one S-box is $l = 4$ bits. P is a plaintext, C is a ciphertext, S denotes S-box and $\oplus K_i$ is an addition of a round key K_i, $(i = 1, 2, 3, 4)$.



Definition 12. *The addition of the key after the last round and in the first round is called* a key whitening. *Without it, the substitution layer and the affine layer could be removed out of the last (and the first) round, because they are public. Removing them would shorten the encryption algorithm which is not desirable.*

KSA

A specific case of SPN is KSA construction, which we can see in the picture (P is a plaintext, C is a ciphertext):

$$P \to \boxed{K} \to \boxed{S} \to \boxed{A} \to \boxed{K} \to \boxed{S} \to \boxed{A} \to \cdots \to \boxed{K} \to \boxed{S} \to \boxed{A} \to \boxed{K} \to C$$

Let us have three layers in KSA:

- K denotes *the key addition layer* – it is same as in the SPN above,
- S denotes *the substitution layer* – S-boxes (they should be non linear, it is the general property the S-boxes should have), S-boxes are same as in the SPN above,
- A denotes *the affine layer* – this means some affine mapping $a : \mathbb{F}_2^n \to \mathbb{F}_2^n$.

The KSA repeats the sequence of operations ($K \times S \times A$). At the end of the encryption algorithm is the key whitening addition because, as in the SPN, the affine and substitution layer are public and the algorithm could be shorten.
We will distinguish between the full KSA and the KSA with the truncated last round.

Definition 13. *The KSA, which consists of* full k rounds – full KSA *has an encryption algorithm:*

$$K \times S \times A \times K \times \cdots \times K \times S \times A = (K \times S \times A)^k \times K.$$

Definition 14. *The KSA with the* truncated last round – truncated KSA *has an encryption algorithm:*

$$(K \times S \times A)^{k-1} \times (K \times S \times K).$$

We would like to emphasise that an affine layer and a key addition layer commute as in the SPN construction (where we denote them S_1 and S_3). Let K be key and P be some text. For us it means that we can switch order of operations in the following way:

- If we have somewhere in the encryption process part: $A(K \oplus P)$ (as in the definition of commutativity we have $A \times K$), we can remove K out - $A(K) \oplus A(P)$ (we get $A(K) \times A$).

  It is caused by the linearity of affine layer. We obtain some key addition again with $A(K)$ instead of K, which is not a problem, because we know A and A^{-1} (it is public information). Then we are able to compute $A(K)$ from K and vice versa.

- On the other hand, if we have: $K \oplus A(P)$ $(K \times A)$, we can put K inside brackets, using A^{-1} we obtain $A(A^{-1}(K) \oplus P)$ (which is $A \times A^{-1}(K)$). It is an affine layer applied on addition of key $A^{-1}(K)$. Again this $A^{-1}(K)$ can be computed from K and vice versa.

Thus, by the definition of commutativity, K and A commute. By the affine layer we mean multiplying by some matrix and adding a vector. We can show that more specifically:
Let M be used matrix and v be added vector. Then $M(K \oplus P) \oplus v$ can be rewritten as $M^{-1}(K) \oplus M(P) \oplus v = (M^{-1}(K)) \oplus (M(P) \oplus v)$.
Similarly, $K \oplus M(P) \oplus v$ can be rewritten as $M(M^{-1}(K) \oplus P) \oplus v$.

This property will be used several times in the text.

AES

We will use AES cipher for demonstration and better understanding of attacks in this thesis. It has similar structure to KSA structure - it has a key addition layer, a substitution layer and also an affine layer and (like in SPN) its last round is different. The affine layer in the last round is different as we will see later. We will not use any key schedule, because attacks in this thesis are against ciphers which use the same key through all rounds or repeat keys in some period. In every round we have the same key or they are used periodically.
AES with r rounds has an encryprion scheme (without a key schedule):

- AddRoundKey (K) (whitening key addition)
- r-1 rounds
  - SubBytes (SB)
  - ShiftRows (SR)
  - MixColumns (MC)
  - AddRoundKey (K)
- the last round
  - SubBytes (SB)
  - ShiftRows (SR)
  - AddRoundKey (K)

We shall briefly explain layers. For more details see Knudsen and Robshaw [2011] The input of AES cipher is 128 bit plaintexts which is divided by bytes and put into matrix (4×4).

Key K is also a matrix of 128 bits. The key addition proceeds byte by byte (more exactly bit by bit). It is XORing of two vectors. This operation can be seen as the addition of two vectors in $\mathbb{F}_2(\mathbb{Z}_2)$ bit by bit.In the text we will denote the key addition (AddRoundKey) by K.

The substitution layer consists of 16 8-bit S-boxes and they are applied on every byte of the state separately in parallel. For the slide attack and thesis it is not important how S-box exactly works inside.

There are more details in Knudsen and Robshaw [2011] and Paar and Pelzl [2010]. In this thesis substitution layer (SubBytes) is denoted by SB.

ShiftRows is the first part of an affine layer. The i-th row is shifted by i bytes to the left. We denote this layer by SR.

MixColumns is the second part of an affine layer. It takes each column of the state and mixes it into new column. To be more precise, it multiplies each comlumn of the state by constant matrix. We compute with columns as with 4×1 vectors in $GF(2^8)$ and matrix is 4×4 in $GF(2^8)$. We denote it by MC.

As in the KSA, we will define truncated and full AES. Normal AES is of course truncated, but for explaining how attack works we will sometimes use full AES.

Definition 15. *AES where MixColumns is omitted in the last round is called* truncated AES.

Definition 16. *Modified AES where all rounds are identical is called* full AES.

We have to distinguish if S-boxes are public or secrtet. In classical AES they would be public, but for our uses we will also consider AES with secret S-boxes.

Definition 17. *We call S-box* public *if it is publicly known and we are able to applicate it on some plaintext. Whereas if S-box is* secret*, it is key-dependent and we do not know inputs and outputs without the key.*

Notation

Truncated AES we denote by AESt and truncated KSA we denote by KSAt. Full AES we denote by AESf and full KSA we denote by KSAf. Secret S-boxes we denote by s and public S-boxes by p.

For example truncated AES with secret S-boxes we denote by AESts. We also note that we use notation for keys. In this thesis we will use encryption algorithms which use the same key in each round. We denote such encryption algorithm by 1K. For example truncated AES using one key with secret S-boxes in denoted by 1K-AESts.

1.1.2 Feistel structure

The original slide attack was mostly used on a Feistle structure. A Feistel structure is a symmetric encryption scheme widely used in block ciphers. Let us denote the left side of some general plaintext or ciphertext R by R^l. and the right side R^r. An encryption proceeds following way:
1) The plaintext P is divided into two halves - P^l and P^r
2) Then the encryption proceeds using the following scheme:
$P^l_{i+1} = P^r_i$
$P^r_{i+1} = P^l_i \oplus f(P^r_i, k_i)$.
where k_i is a round key and f is a round function. For abbreviation we denote the encryption in round i by F_i, $F_i(P^l_i, P^r_i) = (P^l_i \oplus f(P^r_i, k_i), l_i) = (P^l_{i+1}, P^r_{i+1})$.
3) The output is (P^r_k, P^l_k) for k rounds - the output has halves in the opposite order.

1.2 Other useful information

In this section we briefly introduce some basic terms used in the thesis.

1.2.1 Time and data complexity

The data, the memory and the time complexity is computed in the thesis several times.

The data complexity means how many plaintexts (in general some data) we need for the attack. Next important information is if these plaintexts are chosen or just known.

The time complexity in general means how many elementary operations is needed for the attack and it is dependent on the length of input (in bits). In general we are intrested in the worst case which can happen. It means we are interested in the asymptotic complexity, thus we consider oprations with the greatest demands on time - in our attack it always will be encryption operations (in general our encryptions can have an arbitrary number of rounds as we will see later). In the text the time comlexity means number of encryption operations, because other oprations are negligible if we compare them with the encryption.

The memory complexity means how many memory we need for the attack. Again we are interested in the most demanding memory storage. In this thesis it will aways be equal to the data complexity, because remaining used memory is negligible if we compare it with stored plaintexts.

1.3 Useful probabilistic problems

In this section we will introduce two probabilistic problems that are mentioned in the thesis and we use their results. The first will be the birthday paradox, which is called paradox, because many people finds the result of it surprising and next is the coupon collector's problem.

1.3.1 Birthday paradox

Main idea

We are interested in finding a collision of some data – usually plaintexts, ciphertexts or some sequences very often in this thesis. The crucial questin we should ask is how many data and time we need to find the first collision. To compute it we find useful the birthday paradox.

There is a lemma, proof and corollary as it was explained in lecture notes Kozlík [b]. I added my computation of the inequality $1 - x \leq e^{-x}$.

Lemma 1 (Birthday paradox - general version)**.** *Let us have a set of n elements. We choose an element from this set k-times, $k \leq n$. The distribution of probability of choosing one element is uniform and all choices are random and independent. Then the probability of choosing one element more than once (in other words, the probability that a collision occurs) is equal to*

$$1 - \frac{n!}{n^k(n-k)!}$$

Proof. We first compute the probability that there in no collision $p(n,k)$ of choosing k different elements from the set of n elements. It is clear that if we choose the first time we have n possible options, if we choose the second time we get n-1 possible options etc. Thus

$$p(n,k) = \frac{n \cdot (n-1) \cdot (n-2) \cdots (n-(k-1))}{n^k} = \frac{\prod_{i=0}^{k-1}(n-i)}{n^k} = \frac{1}{n^k} \cdot \frac{n!}{(n-k)!}.$$

The complementary rule says that for an event A it holds that $P(A^C) = 1 - P(A)$. Hence the probability of choosing one element more than once is equal to

$$1 - \frac{n!}{n^k(n-k)!}.$$

□

Now we will prove the following corollary.

Corollary. For finding the first collision with probability greater than or equal to $\frac{1}{2}$, if we have n data (which is the fixed number), we need $\sqrt{n}$ data.

We first show that $1 - x \leq e^{-x}$ for every $x \in \mathbb{R}$. This part we show differently than it was in Kozlík [b]. It is evident that $1 - x \leq 0$ and $e^{-x} > 0$ for every $x \in \langle 1, \infty)$ and consequently $1 - x \leq e^{-x}$ for every $x \in \langle 1, \infty)$.
We need to show that $1 - x \leq e^{-x}$ for every $x \in (-\infty, 1\rangle$. To show it we use Taylor series of e^{-x}.

$$\exp(-x) = 1 - x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} \cdots .$$

Thus

$$\begin{aligned} e^{-x} - (1-x) &= 1 - x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} \cdots - (1-x) \\ &= \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} \cdots = x^2\left(\frac{3-x}{3!}\right) + x^4\left(\frac{5-x}{5!}\right) + x^6\left(\frac{7-x}{7!}\right) \cdots . \end{aligned}$$

It is easily seen that

$$x^2\left(\frac{3-x}{3!}\right) + x^4\left(\frac{5-x}{5!}\right) + x^6\left(\frac{7-x}{7!}\right) \cdots \geq 0$$

for every $x \in (-\infty, 1\rangle$, because
$x^2, x^4, x^6 \ldots \geq 0$ for every $x \in \mathbb{R}$ and $\left(\frac{3-x}{3!}\right), \left(\frac{5-x}{5!}\right), \left(\frac{7-x}{7!}\right) \ldots \geq 0$ for every $x \in (-\infty, 3\rangle$. This gives $1 - x \leq e^{-x}$ for every $x \in (-\infty, 1\rangle$. It follows that $1 - x \leq e^{-x}$ for every $x \in (-\infty, 1\rangle \cup \langle 1, \infty) = \mathbb{R}$.

Let us rewrite the formula of the probability

$$1 - \frac{n!}{n^k(n-k)!} = 1 - \frac{\prod_{i=0}^{k-1}(n-i)}{n^k} = 1 - \prod_{i=0}^{k-1}\left(1 - \frac{i}{n}\right).$$

Replacing $1 - \frac{i}{n}, i = 0, 1, 2, ..., k-1$, by $\exp(-\frac{i}{n})$ and using the fact that $1 - x \leq \exp(-x)$ and $\sum_{i=1}^{k-1} -\frac{i}{n} = \frac{(1-k)(-\frac{1}{n} - \frac{k-1}{n})}{2} = \frac{-k(1-k)}{2n}$ (sum of the arithmetic

sequence) we obtain the inequality

$$1 - \prod_{i=0}^{k-1} \left(1 - \frac{i}{n}\right) \geq 1 - \prod_{i=0}^{k-1} e^{-\frac{i}{n}} = 1 - e^{\left(\sum_{i=0}^{k-1} -\frac{i}{n}\right)} = 1 - e^{\left(\frac{-k(1-k)}{2n}\right)}.$$

We would like to know when the probabilty p of the collision is greater than $\frac{1}{2}$. We have some fix n and we are interested in k which is the number of choices. Let us express the inequlity as follows

$$p(n,k) = 1 - \frac{n!}{n^k(n-k)!} \geq 1 - e^{\left(\frac{-k(k-1)}{2n}\right)} \geq \frac{1}{2}.$$

Let us approximate k. By the inequality above we have

$$\frac{1}{2} \geq e^{\left(\frac{-k(k-1)}{2n}\right)}$$

Using the logarithm we get

$$\ln\left(\frac{1}{2}\right) \geq \frac{-k^2 + k}{2n}.$$

Which gives us the inequality

$$k^2 - k - \ln 2 \cdot 2n \geq 0.$$

Solving the inequality yields

$$k \leq \frac{1 - \sqrt{1 + 4 \cdot \ln 2 \cdot 2n}}{2} \quad \text{or} \quad k \geq \frac{1 + \sqrt{1 + 4 \cdot \ln 2 \cdot 2n}}{2}.$$

We are interested in an estimation of k. We need k to be greater than 0 (it would not make the sence to have $k \leq 0$, we choose elements) we thus choose interval where $k \geq \frac{1+\sqrt{1+4\cdot\ln 2\cdot 2n}}{2}$. We also want estimate lower bound of k, hence the result is

$$k = \frac{1 + \sqrt{1 + 4 \cdot \ln 2 \cdot 2n}}{2} \approx 1.6774 \cdot \sqrt{n}.$$

Thus for finding the first collision, if we have n data, we need only $\sqrt{n}$ choosings. It means that $\sqrt{n}$ data is enough for finding a collision. In the thesis is number of data usually 2^n, we thus need $2^{n/2}$ data.

1.3.2 Coupon collector's problem

Main idea

Imagine the following problem. We collect coupons. There are n different kinds of coupons and we want to collect them all. Coupons are sold in closed boxes and we can not say which coupon is inside until we buy it and open it. The probability that one exact coupon is inside is always equal to $\frac{1}{n}$. How many boxes we should buy to collect all n kinds of coupons?

Definition 18. *Let X be a random variable, which can have values $x_1, x_2, \ldots, x_m$ each with corresponding probability $p_1, p_2, \ldots, p_m$. We define* the expected value of X *as $\sum_{i=1}^{m} x_i \cdot p_i$ and we denote it by $\mathbb{E}[X]$.*

Definition 19. Bernoulli trials *is the series of independent trials, where one trial can be successful with probability* p *or unsuccessful with probability* $1 - p$.

Definition 20. *Let us have a sequence of independent Bernoulli trials. We assume the probability* p *of success is greater than* 0. *Let* X *be the random variable which denotes the number of trials where the last one is successful and all previous trials are unssuccesful. Then* $P[X = k] = p(1-p)^{k-1}$, *for* $k = 0, 1, \ldots$ *This probability distribution is called* the geometric distribution.

We note that the expected value of the geometric distribution is equal to $\frac{1}{p}$. (p. 102 Gordon [1997])

Lemma 2. *Let* N *be the random variable which denotes the number of boxes we should buy to collect* n *kinds of coupons. Then* $\mathbb{E}[N] \approx n \ln(n)$.

This proof is taken from Ferrante and Saltalamacchia [2014], where the coupon colloctor's problem is described in more detail.

Proof. Let N_i be the random variable, it denotes the number of boxes we should buy if we have $i-1$ kinds of coupons and we want the i-th different coupon. The probability p_i (associated with N_i) of finding the coupon we have not collected yet, if we have $i-1$ different coupons, is equal to $\dfrac{n-(i-1)}{n}$ (the number of coupons we do not have yet divided by the number of all coupons) . The distribution of N_i is geometric (we have unsuccessful trials until we find some new coupon). This implies that $\mathbb{E}[N_i] = \dfrac{1}{p_i} = \dfrac{n}{n-(i-1)}$. We know that N is the random variable which denotes the total sum of boxes we need to buy, thus $N = \sum_{i=1}^{n} N_i$ We can now start the computation of the expected value.

$$\mathbb{E}[N] = \mathbb{E}[\sum_{i=1}^{n} N_i].$$

Using the linearity of the expected value we get

$$\mathbb{E}[\sum_{i=1}^{n} N_i] = \sum_{i=1}^{n} \mathbb{E}[N_i] = \sum_{i=1}^{n} \frac{n}{n-(i-1)} = \frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} \cdots \frac{n}{1} = n \sum_{i=1}^{n} \frac{1}{i}.$$

Authors of Ferrante and Saltalamacchia [2014] claim we can approximate the expected value as n approaches the infinity as follows:

$$\sum_{i=1}^{n} \frac{1}{i} = \ln(n) + \gamma + \frac{1}{2n} + O(\frac{1}{n^2})$$

, where γ is the Euler-Mascheroni constant. $\gamma \approx 0.5772156649$. Thus

$$\mathbb{E}[N] = n \sum_{i=1}^{n} \frac{1}{i} \approx n(\ln(n) + \gamma + \frac{1}{2n} + O(\frac{1}{n^2})) = n \ln(n) + n\gamma + \frac{1}{2} + O(\frac{1}{n}).$$

Hence

$$\mathbb{E}[N] \approx n(\ln(n)).$$

□

Coupon collector's problem can be used in general way. If we want to collect n values and we are not able to predict which values we will receive, then we should take about $n \ln n$ values. We assume the probability is uniformly distributed.

2. Slide attack

We will introduce some basic definitions [Biryukov and Wagner, 1999] which are necessary for understanding the attack. This chapter is based on sections 2 and 3 Biryukov and Wagner [1999].

Definition 21. A self-similar cipher *is a cipher whose encryption algorithm is a sequence of* r *identical rounds* F_k.

By definition every round function F_k uses the same key k. The definition can be also viewed as a sequence of identical rounds using periodical keys (repeating a sequence of the same keys).

$$P \longrightarrow \boxed{F_k} \longrightarrow \boxed{F_k} \longrightarrow \quad \cdots \quad \longrightarrow \boxed{F_k} \longrightarrow C$$

Example. Full KSA using the same key K is a good example of a self-similar cipher, its round function is $(K \times S \times A)$.

$$P \rightarrow \boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \quad \cdots \quad \rightarrow \boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \boxed{K} \rightarrow C$$

Definition 22. *Let* E *be an encryption algorithm of a self-similar cipher,* $E = F_k \times F_k \times \cdots F_k \times F_k$. *Let* P *and* Q *be plaintexts,* C *and* D *be ciphertexts of* P *and* Q*, respectively. We call* P *and* Q a slid pair *if* $Q = F_k(P)$. *It follows from the self-similarity of the encryption algorithm that* $D = F_k(C)$ *also holds.*

$$\begin{aligned} P \rightarrow \boxed{F_k} &\xrightarrow{Q} \boxed{F_k} \rightarrow \boxed{F_k} \rightarrow \cdots \rightarrow \boxed{F_k} \rightarrow \boxed{F_k} \rightarrow C \\ Q \rightarrow \boxed{F_k} &\rightarrow \boxed{F_k} \rightarrow \boxed{F_k} \rightarrow \cdots \rightarrow \boxed{F_k} \xrightarrow[C]{} \boxed{F_k} \rightarrow D \end{aligned}$$

Example. The example of a slid pair P, Q (C, D are corresponding ciphertexts) on KSA using the same key K.

$$\begin{aligned} P \rightarrow &\boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \cdots \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \boxed{K} \rightarrow C \\ Q \rightarrow &\boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \cdots \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \boxed{K} \rightarrow \boxed{S} \rightarrow \boxed{A} \rightarrow \boxed{K} \rightarrow D \end{aligned}$$

If we find a slid pair P, Q (with ciphertexts C, D), we obtain two equations $A(S(K \oplus P)) = Q$ and $K \oplus A(S(C)) = D$. This relation can be easily seen in the previous picture.

Definition 23. *A cipher when given two equations* $F_k(P_1, k) = C_1$ *and* $F_k(P_2, k) = C_2$ *we are able to deduce the key* k *"easily" is called* a cipher with a weak round function.

Example. An example for *"weak" round function* is one round of KSA, which uses the same key K in all rounds. Once we have a slid pair P,Q, we know that $A(S(K \oplus P)) = Q$, which means $K \oplus P = S^{-1}(A^{-1}(Q))$ and thus $K = P \oplus S^{-1}(A^{-1}(Q))$. If we know P and Q, we are able to compute right-hand side of the last equation and we immediately obtain a key.

Even one round of Feistel structure is weak as we will see in the following section.

2.1 Slide attack

The slide attack is an attack on self-similar ciphers with weak round functions using one key or periodical keys. Its advantage is that slide attack is independent on the number of rounds. It is a structural attack, because it uses a weakness of the structure of the encryption algorithm — it uses self-similarity. It was first introduced in Biryukov and Wagner [1999]. Slide attacks were then demonstrated on several real-life cryptosystems, for example on TREYFER, GOST etc. [Biryukov and Wagner, 1999], [Biryukov and Wagner, 2000] Feistel scheme is the most common for slide attacks on block ciphers. An example of slide attack on KSA scheme (on SPN) can be found in the section 2 Dunkelman et al. [2020].

2.1.1 The algorithm

In general the algorithm has two phases:

- Finding a slid pair
- Using a slid pair to recover the key

After finding a slid pair it is "easy" to recover the key, because the cipher we are attacking should have a weak round function. The attack is independent on the number of rounds, because once we have a slid pair, we do not use the encryption scheme, we use only round function to recover the key.

3. Slide attacks on almost self-similar ciphers

First we will introduce some useful definitions. As a source we used Dunkelman et al. [2020].

Definition 24. *A cipher whose encryption algorithm repeats the same round function as the self-similar cipher, but the last round is different, is called* an almost self-similar cipher.

In other words, the last round breaks the self-similarity of the encryption algorithm. Again it uses the same key in all rounds (even in the last one) or repeats a sequence of keys.

Example. An example of an almost self similar function is the SPN which uses the same keys in all rounds, where the permutation layer is omitted in the last round.

Definition of a slid pair in an almost self-similar cipher is similar to the one in the previous chapter, where we used self-similar ciphers.

Definition 25. *Let F_k be one of the same rounds of an almost self-similar cipher and let F_{last} be the last round of an almost self-similar cipher (if we use one key, in the case of two keys, F_k are two rounds we repeat in encryption and F_{last} are two last rounds). Let us have plaintexts P and Q and their ciphertexts C and D respectively. We call P and Q* a slid pair *if and only if $Q = F_k(P)$ and $D = F_{last}(C)$ hold.*

In AES and in general KSA or SPN structure if $Q = F_k(P)$, then it follows that $D = F_{last}(C)$ holds. It follows from the commutativity of affine layer and from the fact that the the key addition is the addition of vectors in $\mathbb{Z}_2$. An example we will see in the subsection Settings 3.1.3, where we derive the equation for 1K-AESts.

3.1 New slide attacks

A motivation for new slide attacks is the fact that the original slide attack can not be used on almost self-similar ciphers. Original slide attacks were mostly used on a Feistel structure due to its self-similarity. New slide attacks focus on SPN, where the self-similarity is broken by the last round. Some designers of ciphers could think that a cipher with a weak key schedule and the different last round should be resistent against slide attacks. Authors of Dunkelman et al. [2020] explain that it is not true and their paper can be a warning for designers of ciphers.

In this chapter we will explore two of the four new slide attacks introduced in Dunkelman et al. [2020] - slid sets attack and attack using hypercube of slid pairs. Both of these attacks are chosen-plaintext attacks. This chapter is based on sections 3 and 4 in Dunkelman et al. [2020]. We will describe essential parts of the algorithm of the slid sets attack on 1K-AESfs and 1K-AESts (1K-AESfs is

described as it was in Dunkelman et al. [2020], whereas 1K-AESts was not fully described there). We will also describe essential parts of the slide attack using hypercube of slid pairs on general SPN with 1K-AESfs structure. For selected parts we add explanations or computations for the general case.

3.1.1 Main idea

The idea is similar to the one in the original slide attack. First we collect some slid pairs. In these attacks we need more than just one pair. Then we use "weakness" of the cipher to solve the equations and to recover the key. The "weakness" is caused by the structure of the encryption algorithm.

3.1.2 General algorithm

Attacks presented in this chapter have two parts.

- Collecting sufficiently many slid pairs
  - Creating some structures of plaintexts where it is easy to find more slid pairs, ideally all together in one structure
  - Detecting these slid pairs
- Using these slid pairs to recover the key

3.1.3 Settings

We will use the definition of a slid pair to get equations and relations for a slid pair in 1K-AESfs (which is self-similar) and 1K-AESts (which is almost self-similar).

1K-AESfs

1K-AESfs and 1K-AESts both use just one round key K. Suppose P_i and Q_j form a slid pair. By the definition of slid pair, we can write the first round as

$$(MC(SR(SB(K \oplus P_i)))) = Q_j,$$

We can rewrite it as

$$SB(K \oplus P_i) = \tilde{Q}_j, \tag{3.1}$$

where $\tilde{Q}_j = SR^{-1}(MC^{-1}(Q_j))$.

Similarly we can get the equation for ciphertexts C_i, D_j of P_i, Q_j, respectively. We can write the last round as follows

$$K \oplus MC(SR(SB(C_i))) = D_j.$$

From this equation we obtain

$$K^* \oplus SB(C_i) = \tilde{D}_j, \tag{3.2}$$

where $\tilde{D}_j = SR^{-1}(MC^{-1}(D_j))$ and $K^* = SR^{-1}(MC^{-1}(K))$.

1-KSAts

Consider the same settings as for 1-KSAfs. We describe it in more detail than in Dunkelman et al. [2020]. The last round is different but it is the only difference. Let P_i be the plaintext and C_i its ciphertext. Then it follows that the following relationship exists between them

$$C_i = K \circ SR \circ SB \circ K \circ MC \circ SR \circ SB \circ K \circ \cdots \circ MC \circ SR \circ SB \circ K(P_i).$$

Let Q_j be the slid counterpart of P_i and D_j be its ciphertext. Then

$$Q_j = MC \circ SR \circ SB \circ K(P_i).$$

and

$$D_j = K \circ SR \circ SB \circ K \circ MC \circ SR \circ SB \circ K \circ \cdots \circ MC \circ SR \circ SB \circ K(Q_j).$$

The task is to find the relation between C_i and D_j. Using the relation between P_i and Q_j and due to the fact that adding the key twice is equivalent to adding zero (addition in $\mathbb{Z}_2$), we can rewrite the last two lines as follows

$$\begin{aligned} D_j = K \circ SR \circ SB \circ K \circ MC \circ \boldsymbol{K} \circ \boldsymbol{K} \circ SR \circ SB \circ \cdots \\ \cdots \circ MC \circ SR \circ SB \circ K \circ \underbrace{MC \circ SR \circ SB \circ K(P_i)}_{Q_j} \end{aligned}$$

We now can see that C_i appears in this relation

$$D_j = K \circ SR \circ SB \circ K \circ MC \circ \boldsymbol{K} \circ \overbrace{\boldsymbol{K} \circ \cdots \circ SB \circ K \circ \underbrace{MC \circ SR \circ SB \circ K(P_i)}_{Q_j}}^{\boldsymbol{C_i}}$$

It easily follows that

$$D_j = K \circ SR \circ SB \circ K \circ MC \circ K(C_i)$$

which can be rewritten as follows

$$K \oplus (SR(SB(K \oplus (MC(K \oplus C_i))))) = D_j.$$

Using the linearity of MC and SR and substituting $SR^{-1}(K)$ by K^*, $K \oplus MC(K)$ by $K^{\heartsuit}$, $MC(C_i)$ by $\tilde{C}_i$ and $SR^{-1}(D_j)$ by $\tilde{D}_j$ we obtain

$$K^* \oplus SB(K^{\heartsuit} \oplus \tilde{C}_i) = \tilde{D}_j \tag{3.3}$$

We note that all equations (3.1), (3.2), (3.3) have the following property.

Property. Operations SubBytes and KeyAddition are done byte by byte. In general case of s-bit S-box, they are done by s-bit blocks separately. Thus in all equations above can be seen as transformation of plaintexts or ciphertexts into another plaintexts or ciphertexts by bytes separately, using 16 different functions. One function stands for every byte. We call this property *Byte by byte* property.

3.2 Slid sets attack

Firstly, we will define λ-set.

Definition 26. *A $\lambda-$set of plaintexts (in this definition plaintexts of AES, $s = 8$) is a set of 2^{8^k} plaintexts where k bytes of the state are* active. *It means that plaintexts in the λ-set attain all possible values in all active bytes separately. The difference of plaintexts in all other bytes is equal to* 0.

Example. In the slid sets attack we will use λ-set with the first byte active. We can see how some plaintexts of one λ-set can look like in the picture below.

01	03	25	3c
2a	ff	57	b1
80	a9	12	a6
e5	03	03	61

02	03	25	3c
2a	ff	57	b1
80	a9	12	a6
e5	03	03	61

03	03	25	3c
2a	ff	57	b1
80	a9	12	a6
e5	03	03	61

. . .

fe	03	25	3c
2a	ff	57	b1
80	a9	12	a6
e5	03	03	61

ff	03	25	3c
2a	ff	57	b1
80	a9	12	a6
e5	03	03	61

Secondly, we will define slid sets, which is a structure used in the attack.

Definition 27. *Let us have sets of plaintexts A and B, $|A| = |B|$. We call them slid sets when if some arbitrary plaintext $A_i \in A$ has a slid counterpart $B_j \in B$, then every plaintext in A has a slid counterpart in B.*

Main idea and general information

The main idea of the slid sets attack is to use large slid sets (see definition 27). If we find large slid sets, we will immediately have enough slid pairs, even without the knowledge of exact pairs.

3.2.1 Construction of candidate sets

In this and following section we will describe the algorithm on 1K-AESfs. The same algorithm can be applied on 1K-AESts, the only difference are equations (which have the same Byte by byte property) and substitutions.

Consider two slid pairs P_1, Q_1 and P_2, Q_2. Let us compute $\tilde{Q_1}$ and $\tilde{Q_2}$ from Q_1 and Q_2 the same way we did it in (3.1). Because of Byte by byte property it follows that if P_1 and P_2 differ in one byte, then $\tilde{Q_1}$ and $\tilde{Q_2}$ differ in the same one byte. This can be used in the construction of sets, which should be candidates for slid sets.

Consider two λ - sets P and $\tilde{Q}$ where only the first byte of the state is active as in the example 3.2. Without loss of generality we can assume it is the first byte of the state.

By definition of the λ - set (26), the first byte attains all 256 ($= 2^8$) possible values and other bytes of state have the same values in all plaintexts in the set. Let Q be the set obtained from $\tilde{Q}$ such that every $Q_k \in Q$ is derived from $\tilde{Q}_k \in \tilde{Q}$ using the substitution we used before in (3.1). Thus for every $\tilde{Q}_k \in \tilde{Q}$ it holds that $Q_k = MC(SR(\tilde{Q}_k))$. It is obvious that all plaintexts in P differ only in the first byte. Hence if there is any set Q such that P and Q are slid sets, the corresponding set $\tilde{Q}$ is also a λ-set where the first byte is active.

In general, let us consider SPN construction with n-bit input and output and s-bit S-boxes. We use general λ-sets.

Definition 28. A general $\lambda-$set *of plaintexts is a set of* 2^{sk} *plaintexts where* k s*-bit blocks of the state are* active. *It means that plaintexts in the* λ*-set attain all possible values in all active* s*-bit blocks separately. The difference of plaintexts in all other* s*-bit blocks is equal to* 0.

The argumentation would be the same as above – we would derive equations, where the transformation of slid plaintexts would be done by s-bit blocks separately.

3.2.2 Detecting slid sets

The aim of this part of attack is to obtain the slid sets. Let us have many λ-sets, which are our candidates for slid sets. We can use sets of ciphertexts to identify the slid sets.

Let us have two sets of λ-sets – S_P and S_Q. Firstly, we encrypt every λ-set $P \in S_P$ and for P we obtain set of ciphertexts C. Secondly, we encrypt every λ-set $Q \in S_Q$ and for every Q we obtain set of ciphertexts D.

Let $\tilde{D}$ be obtained from D using the substitution we used before (3.2). Every $\tilde{D}_j \in \tilde{D}$ is obtained from $D_j \in D$ as follows: $\tilde{D}_j = MC(SR(D_j))$.

The main idea is to use the Eq. (3.2). The equation has Byte by byte property which means $C_i \in C$ is transformed into $\tilde{D}_j \in \tilde{D}$ using 16 functions one for each byte separately. Therefore, for every two ciphertexts C_i, $C_j \in C$ and arbitrary byte l holds the equality $C_i^l = C_j^l$ (l-th bytes of C_i and C_j are equal) if and only if there exist $\tilde{D}_n, \tilde{D}_m \in \tilde{D}$ such that $\tilde{D}_n^l = \tilde{D}_m^l$. Using this property we can construct a sequence of multiplicities for every byte for every set C created from $\lambda - set\ P \in S_P$. Then we try match such sequences of sequences using a hash table. If match occurs for some C and $\tilde{D}$, then corresponding sets of plaintexts P and Q are the slid sets.

This is described in the section 3 in Dunkelman et al. [2020] in more details. Authors of the paper claim that with a high probability there should not be any false match, in other words, a collision of non slid sets. We found a small typo they made in one sentence in the paragraph about a collision. Sentence "If the number of elements not appearing in both sets is different, then the sequences do collide," shoud be replaced by "If the number of elements not appearing in both sets is different, then the sequences do **not** collide".

3.2.3 Deducing slid pairs from slid sets

For attacking 1K-AESfs (or 1K-AESts) with a secret key dependent S-box, we will also need to deduce exact slid pairs to determine the S-box. Suppose we found some slid sets. For every pair of slid sets, we compute their ciphertexts and corresponding set such that their plaintexts occurs in Eq. (3.2) or Eq. (3.3) (it depends on the attack). We compute a sequence of sequences for every ciphertext in such pair of slid sets. Then we try to match the sequences. The exact algorithm is described in the section 3 in Dunkelman et al. [2020]. At the end of this process we have find all slid pairs in a given pair of slid sets.

3.2.4 1K-AESfs and 1K-AESts - Connecting slid pairs

The S-box is key dependent which means we can not guess the key and simply compute multiplicities. Just knowledge of slid sets is not enough to recover the secret material. We need to deduce exact counterparts from more given slid sets.

Assume P and Q is a given pair of slid sets and C, D are corresponding sets of ciphertexts. $\tilde{D}$ is derived from D (like in (3.2), for every plaintext in D).

Now for every ciphertext $C_i \in C$ and for each l-th byte of it we compute the number of ciphertexts $C_j \in C$, $i \neq j$, such that the l-th byte of C_j is equal to l-th byte of C_i. Let us make the same computation for D. Let us make the same computation for D. If $P_i \in P$, $Q_j \in Q$ form a slid pair, then corresponding ciphertexts $C_i \in C$ and $D_j \in D$ have the same sequences, which follows from (3.2) (the transformation of bytes separately). There are more details in the section 3 in Dunkelman et al. [2020]. Authors claim that if we find a match of sequences for ciphertexts, their plaintexts form a slid pair with the high probability.

3.2.5 1K-AESfs - Recovering secret material

We can use the fact that the Eq. (3.2) has the Byte by byte property and it determines 16 functions, one for each byte. In the previous step we connected slid pairs P_i and Q_j and we know their pairs of ciphertexts C_i, D_j, respectively. Every such pair of ciphertexts $C_i \in C$ and $D_j \in D$ satisfies the equation. Thus for every l-th byte there exists a function

$$F_l(x) = K_l^* \oplus SB(x),$$

where input x is the l-th byte of a ciphertext C_i, K_l^* is the l-th byte of a key $K^* = SR^{-1}(MC^{-1}(K))$ and $F(l, x)$ is equal to l-th byte of a corresponding ciphertext $\tilde{D}_j$.

We will compute later, how many of connected pairs we need to get all possible inputs and outputs of $F(x)$. Firstly, suppose we have enough of slid pairs, thus we have all possible input and output pairs of $F(x)$ for all bytes.

The AES encryption process is

$$C = K \circ MC \circ SR \circ SB \circ K \circ \cdots \circ K \circ MC \circ SR \circ SB \circ K(P)$$

which is equivalent to

$$\begin{aligned} C = MC \circ SR \circ (SR^{-1} \circ MC^{-1} \circ K) \circ SB \circ \cdots \\ \cdots \circ MC \circ SR \circ (SR^{-1} \circ MC^{-1} \circ K) \circ SB \circ K(P), \end{aligned}$$

which means

$$C = MC \circ SR \circ K^* \circ SB \circ MC \circ SR \circ K^* \circ SB \circ \cdots \circ MC \circ SR \circ K^* \circ SB \circ K(P),$$

where P is a plaintext and C is a ciphertext.

We are able to take ciphertext C and partially decrypt it because we know inverses of MC and SR operations and now we know the inputs and outputs of the function $F(x) = K^* \oplus SB(x) = SR^{-1}(MC^{-1}(K) \oplus SB(x))$ (which is equivalet to $F(x) = K^* \circ SB(x) = SR^{-1} \circ MC^{-1} \circ K \circ SB(x)$). Therefore we are able to go through the encryption process and only operation which remains is $K \oplus P$,

(which is equivalent to $K(P)$). Now we have everything we need — plaintext P and corresponding ciphertext C (we can take some we already have), which is partially decrypted (let denote it C^{part}). Using simply XOR operation we can solve the equation $C^{part} = P \oplus K$. Our key K is equal to $C^{part} \oplus P$.

Secondly, there is a question how many slid pairs we need to get all inputs and outputs of $F_l(x)$ which is the l-th byte of input and output of $F(x)$.

We have 256 pairs of input and output values for every byte. Obviously to retrieve all input and output values for every such function we will need more than just one pair of slid sets or one slid hypercube, because we can assume that one set or one hypercube of ciphertexts C does not contain all 256 needed values in every byte (as inputs). To determine how many values we need we can use the coupon collector problem 1.3.2. We can assume that values of l-th byte are uniformely random distributed. Then the probability of every value is $\frac{1}{256}$. Therefore using the expected value we should have $n \cdot \ln n = 256 \cdot \ln 256$ pairs to recover F_l with high probability.

In general, we have n/s s-bit blocks and we want to recover all 2^s input and output pairs of function $F(x)$ derived similarly like in the case of 1K-AESfs. Bytes are replaced by s-bit blocks. Then the number of slid pairs needed for the recovery of all input and output pairs of the function $F(x)$ is equal to $2^s \ln(2^s)$ by coupon collector's problem.

3.2.6 The data complexity

In the specific case of 1K-AESfs we can assume that one pair of slid sets gives us 256 pairs and $\ln 256 \approx 6$, thus we need 6 pairs of slid sets to recover all functions for all 16 bytes. This implies how many plaintext we will need for the attack. Firstly, we will compute how many data we need if we want to get one pair of slid sets. We use chosen structures of plaintexts. We need two of them for the attack algorithm. We need λ-sets with (without loss of generality) the first byte active. To create them we can take all possible plaintexts and for every plaintext permute its first byte. The number of all possible plaintexts is $2^n = 2^{128}$ and 2^8 are all possible values of first byte. Hence, we can get $2^8 \cdot 2^{128} = 2^{136}$ plaintexts in all λ-sets in total. Plaintexts are obtained more than once. By birthday paradox 1.3.1, $2^{(128+8)/2}$ of plaintexts is enough to detect a slid set. Hence, we need two structures of 2^{68} plaintexts. Let us denote them T_P and $T_{\tilde{Q}}$. Both of these structures contain 2^{60} $\lambda - sets$ because there are 2^8 plaintexts in every $\lambda - set$. Thus the data complexity of getting one slid pair is equal to $2^{68} \cdot 2 = 2^{69}$ chosen plaintexts.

For retrieving secret material in the attack on 1K-AESfs and 1K-AESts we need 6 pairs of slid sets, thus the data complexity is higher. If x denotes the number of slid sets we have, then $x \cdot x$ is the number of all possible pairs we can create and thus $x \cdot x = 6$. From which follows that for us $x = \sqrt{6}$. Consequently, our data complexity is $2^{69} \cdot \sqrt{6} = 2^{69+\log_2 \sqrt{6}} \approx 2^{69+1.3} = 2^{70.3}$. It means 2^{71} chosen plaintexts. The time complexity is 2^{71} encryptions and the same is the memory complexity.

In general, for KSA with s-bit S-box the complexity of getting one slid pair is (using the same steps as in not general case) $2 \cdot 2^{(n+s)/2} = 2^{(n+s)/2+1}$ chosen plaintexts. We add $s+n$ in the exponent because 2^n is a number of all plaintexts,

2^s all general $\lambda - sets$ for them and we use the birthday paradox, so exponent is divided by 2.

The attack on 1-KSAfs or 1-KSAts with s-bit S-boxes requires $\ln(2^s) = s \cdot \ln(2)$ slid pairs by the coupon collector's problem. Using the same steps as before in the not general case we have the data complexity $2 \cdot \sqrt{s \ln(2)} \cdot 2^{(n+s)/2} = \sqrt{s \ln(2)} \cdot 2^{(n+s)/2+1}$ chosen plaintexts. The same is the memory and the time complexity.

3.2.7 Slid sets 1K-AESts - Recovering secret material

The attack on 1-KSAts proceeds the same way as on 1-KSAfs, but instead of $F_l(x) = K_l^* \oplus SB(x)$, we use the function $F_l(x) = K_l^* \oplus SB(K_l^{\heartsuit} \oplus x)$, which is equivalent to $F_l(x) = K_l^* \circ SB \circ K^{\heartsuit}(x)$. However, the algorithm is the same. Futhermore, the retrieving of the key, if we have enough inputs and outputs, is done similarly. The only difference is the partial decryption of ciphertext C, because we use Eq. (3.3).

We describe the algorithm in more detail than in Dunkelman et al. [2020]. Again, the AES encryption process is

$$C = K \circ SR \circ SB \circ K \circ MC \circ SR \circ SB \cdots \circ K \circ MC \circ SR \circ SB \circ K(P)$$

where P is a plaintext and C is a ciphertext. Using the fact that adding K twice is equal to adding zero this can be rewritten as

$$C = K \circ SR \circ SB \circ K \circ MC \circ \boldsymbol{K} \circ \boldsymbol{K} \circ SR \circ SB \circ \cdots \circ K \circ MC \circ \boldsymbol{K} \circ \boldsymbol{K} \circ SR \circ SB \circ K(P).$$

Using the linearity of SR we have

$$C = SR \circ \boldsymbol{SR^{-1}} \circ \boldsymbol{K} \circ SB \circ K \circ MC \circ K \circ SR \circ \boldsymbol{SR^{-1}} \circ \boldsymbol{K} \circ SB \circ \cdots$$
$$\cdots K \circ MC \circ K \circ SR \cdot \boldsymbol{SR^{-1}} \circ \boldsymbol{K} \circ SB \circ K(P).$$

If we subtitue $SR^{-1} \circ K$ by K^* we obtain

$$C = SR \circ \boldsymbol{K^*} \circ SB \circ K \circ MC \circ K \circ SR \circ \boldsymbol{K^*} \circ SB \circ \cdots$$
$$\cdots K \circ MC \circ K \circ SR \cdot \boldsymbol{K^*} \circ SB \circ K(P).$$

Using the linearity of MC we obtain

$$C = SR \circ K^* \circ SB \circ K \circ \boldsymbol{(MC \circ K)} \circ \boldsymbol{MC} \circ SR \circ K^* \circ SB \circ \cdots$$
$$\cdots K \circ \boldsymbol{(MC \circ K)} \circ \boldsymbol{MC} \circ SR \cdot K^* \circ SB \circ K(P).$$

If we subtitue $K \circ (MC \circ K)$ by $K^{\heartsuit}$ we have

$$C = SR \circ K^* \circ SB \circ \boldsymbol{K^{\heartsuit}} \circ MC \circ SR \circ K^* \circ SB \circ \cdots$$
$$\cdots \boldsymbol{K^{\heartsuit}} \circ MC \circ SR \cdot K^* \circ SB \circ K(P).$$

Now we can partially decrypt ciphertext C because we know inputs and outputs of the function $F_l(x) = K_l^* \oplus SB(K^{\heartsuit} \oplus x)$ which is equivalent to $F_l(x) = K_l^* \circ SB \circ K^{\heartsuit}(x)$ and inverses of MC,SR. In the end of the partial decryption only $K \oplus P$ remains and the rest of the retrieving key algorithm is identical to the rest of key retrieving algorithm in the attack on 1K-AESfs 3.2.5. The slid sets attack on 1-KSAts has the same time, memory and data complexity because used equations are the only difference.

3.3 Slide attack using a hypercube of slid pairs

Equations used in this section follow from 1K-AESfs. However, we will compute with general SPN with s-bit S-boxes and n-bit input and output. All equations will remain the same for simplicity. Operations MC, SR can be viewed as some general affine operations, SB as some general substitution layer and K as an addition of n-bit key K. These changes do not affect the algorithm. We note that our general SPN network has secret S-boxes.

We will start with a formal definition of a hypercube. Firstly, we will define a graph.

Definition 29. *[Matoušek and Nešetřil, 2019]* A graph G *is an ordered tuple* (V, E)*, where* V *is an non empty set, its elements are called* vertices*, and* E *is a set of subsets of* V*, where every of these subsets contains exactly two elements. Elements of* E *are called* edges*.*

Now we can define a hypercube.

Definition 30. A k-dimensional hypercube *is a graph* (V, E)*, where* V *are all* k*-bit numbers,* $k \in \mathbb{N}$*.* E *is a set of all edges, each edge comprises of two vertices such that their difference is exactly one bit.*

Let us make the following observation. Let P, Q and F, G be slid pairs. Plaintexts $\tilde{G}$ and $\tilde{Q}$ are computed using (3.1). If P and F differ in arbitrary i-th s-bit block, $\tilde{G}$ and $\tilde{Q}$ must only differ in the i-th s-bit block as well. This property follows from Eq.(3.1) and Byte by byte property. It means this i-th s-bit block is the only place where P and F differ and $\tilde{G}$ and $\tilde{Q}$ differ.

Assume we have a slid pair P, Q. Then we create new slid pair F, G, where we change i-th s-bit block in F and $\tilde{G}$. Such a new pair definitely satisfies the equation of a slid pair in all s-bit block but the i-th. The i-th block satisfies th Eq.(3.1) with probability 2^{-s}. Then new pair is a slid pair with probability 2^{-s}.

This property gives us a friend pair defined below.

Definition 31. A friend pair F,G *is a pair obtained from a slid pair* P*,* Q *such that* F *and* P *differ in only* i*-th* s*-bit block and* $\tilde{G}$ *and* $\tilde{Q}$ *differ in only* i*-th* s*-bit block.*

$\tilde{G}$ and $\tilde{Q}$ are computed from G and Q using Eq.(3.1). The transformation in (3.1) is done by s-bit blocks separately, because the Eq.(3.1) has Byte by byte property. A friend pair is a slid pair with probability 2^{-s}, because of the observation above the definition.

Main idea and general information

The main idea of the attack is to use a structure called a hypercube of plaintexts to detect a hypercube of slid pairs. A hypercube of slid pairs is created using one pair of plaintexts and its friend pairs as we will see.

Hypercube of plaintexts

Definition 32. *Let us have $t \in \mathbb{N}, 1 \leq t \leq n/s$. Let $P_i \in T_P$ be plaintext, T_P is a set of plaintexts. Let $a_k, 1 \leq k \leq t$, be arbitrary fixed vectors whose only non-zero s-bit block is k-th.* A t-dimensional hypercube of plaintexts *is a hypercube whose every vertex is one $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_t) \in \mathbb{F}_2^t$ and corresponding plaintext*

$$P_{i,\alpha} = P_i \oplus \alpha_1 a_1 \oplus \alpha_2 a_2 \oplus \ldots \oplus \alpha_t a_t.$$

Therefore, a hypercube plaintexts contains 2^t plaintexts.

Hypercube of pairs of plaintexts

Definition 33. *Let P_i, Q_j be plaintexts. Let $a_k, b_k, 1 \leq k \leq t$, be arbitrary fixed vectors whose only non-zero s-bit block is k-th. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_t) \in \mathbb{F}_2^t$.* A t-dimensional hypercube of pairs of plaintexts *is a hypercube whose every vertex is α and corresponding pair $(P_{i,\alpha}, Q_{j,\alpha})$, where*

$$P_{i,\alpha} = P_i \oplus \alpha_1 a_1 \oplus \alpha_2 a_2 \oplus \ldots \oplus \alpha_t a_t$$

and

$$Q_{j,\alpha} = MC(SR(\tilde{Q}_j \oplus \alpha_1 b_1 \oplus \alpha_2 b_2 \oplus \ldots \oplus \alpha_t b_t)).$$

$\tilde{Q}_j$ is obtained from Q_j using Eq.(3.1).

Hypercube of slid pairs

The following lemma gives us an idea, what is meant by a t-dimensional hypercube of slid pairs.

Lemma 3. *Let $t \in \mathbb{N}, 1 \leq t \leq n/s$. Let $a_k, b_k, 1 \leq k \leq t$, be fixed randomly chosen vectors whose only non-zero s-bit block is k-th. Assume that P_i, Q_j form a slid pair and t friend pairs $P_i \oplus a_k$ and $MC(SR(\tilde{Q}_j \oplus a'_k))$ are also slid pairs. Then for any $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_t) \in \mathbb{F}_2^t$ plaintexts*

$$P_{i,\alpha} = P_i \oplus \alpha_1 a_1 \oplus \alpha_2 a_2 \oplus \ldots \oplus \alpha_t a_t$$

and

$$Q_{j,\alpha} = MC(SR(\tilde{Q}_j \oplus \alpha_1 a'_1 \oplus \alpha_2 a'_2 \oplus \ldots \oplus \alpha_t a'_t))$$

are also slid pairs.

Proof. The proof follows from the Byte by byte property of Eq. (3.1). The equation represents transformation by s-bit blocks. A slid pair P_i, Q_j and t friend pairs $P_i \oplus a_k$, $MC(SR(\tilde{Q}_j \oplus a'_k))$ which are by assumption slid pairs satisfy the Eq. (3.1). It mneans they satidfy the equation by s-bit blocks separately. Hence our new pairs must be slid pairs as well, because s-bit block by s-bit block they also satisfy the Eq. (3.1).

□

Definition 34. *Let P_i, Q_j be plaintexts. Let $a_k, b_k, 1 \leq k \leq t$, be arbitrary fixed vectors whose only non-zero s-bit block is k-th. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_t) \in \mathbb{F}_2^t$.* A hypercube of slid pair*s is a hypercube of pairs of plaintexts $(P_{i,\alpha}, Q_{j,\alpha})$*

$$P_{i,\alpha} = P_i \oplus \alpha_1 a_1 \oplus \alpha_2 a_2 \oplus \ldots \oplus \alpha_t a_t$$

and

$$Q_{j,\alpha} = MC(SR(\tilde{Q}_j \oplus \alpha_1 a'_1 \oplus \alpha_2 a'_2 \oplus \ldots \oplus \alpha_t a'_t))$$

such that $(P_{i,\alpha}, Q_{j,\alpha})$ are slid pairs.

We construct a hypercube of slid pairs by using all possible α. It contains all pairs $P_{i,\alpha}, Q_{j,\alpha}$. Each α is one vertex of a hypercube of slid pairs and every created pair of plaintexts correspond to one α. Thus we have one pair of plaintexts in every vertex. Therefore, a hypercube of slid pairs contains 2^t slid pairs.

We note that there is a little typo in the section 4 in Dunkelman et al. [2020]. Authors use wrong slid pairs many times. For example the pair $P_i \oplus a$ and $\tilde{Q}_j \oplus a'$ instead of $P_i \oplus a$ and $MC(SR(\tilde{Q}_j \oplus a'))$. However, these typos do not affect the result.

By the Lemma 3 for construction of hypercube of slid pairs we need a slid pair P_i, Q_j. A randomly chosen pair is a slid pair with probability 2^{-n}, because all n bits must satisfy the Eq.(3.1). Then we construct t friend pairs, each of them is a slid pair with probability 2^{-s}. We need all t vectors (non-zero bytes) to satisfy the Eq. (3.1) to create a hypercube of slid pairs by Lemma3. Thus all created pairs are slid pairs with probability 2^{-st}.

We will summarize what we know:

- The probability that a random pair is a slid pair is equal to 2^{-n}.
- The probability that a friend pair of a slid pair is also a slid pair is equal to 2^{-s}.
- The probability that all t friend pairs of a slid pair are also slid pairs is 2^{-ts}.
- The probability that a hypercube of plaintexts constructed using a random pair of plaintexts is a hypercube of slid pairs is $2^{-n} \cdot 2^{-ts}$.

Definition 35. *Let P_i be a plaintext. Let us have a corresponding hypercube of plaintexts $P_{i,\alpha}$. Let $C_{i,\alpha}$ be corresponding ciphertexts. Let us have a list of all collisions (α, β, l), $(C_{i,\alpha})_l = (C_{i,\beta})_l$ which denotes that the l-th s-bit block of $C_{i,\alpha}$ is equal to the l-th s-bit block of $C_{i,\beta}$. This list is lexicographically ordered and we denote it by $Coll_{P_i}$.*

Lemma 4. *Assume P_i, Q_j are a slid pair, C_i, D_j are corresponding ciphertexts, respectively. Assume we have a t-dimensional hypercube of slid pairs $(P_{i,\alpha}, Q_{j,\alpha})$. $C_{i,\alpha}, D_{j,\alpha}$ are ciphertexts of $P_{i,\alpha}, Q_{j,\alpha}$, respectively. The ciphertext $\tilde{D}_{j,\alpha}$ is computed using (3.2). If $(C_{i,\gamma})_l = (C_{i,\beta})_l$, for some $\gamma, \beta \in \mathbb{F}_2^t$, then $(\tilde{D}_{j,\gamma})_l = (\tilde{D}_{j,\beta})_l$.*

Proof. Let us have l, γ and β such that l-th s-bit of $C_{i,\gamma}$ is equal to the l-th s-bit block of $C_{i,\beta}$. $(P_{i,\alpha}, Q_{j,\alpha})$ are slid pairs for every α. It follows that for a pair of their ciphertexts hold the equations (using Eq.(3.2))

$$\tilde{D}_{j,\gamma} = K^* \oplus SB(C_{i,\gamma})$$

and

$$\tilde{D}_{j,\beta} = K^* \oplus SB(C_{i,\beta}).$$

We apply the same function on the same value of s-bit block. The function has Byte by byte property. It follows that the value of the funcion will be the same for the same s-bit blocks.

□

3.3.1 The algorithm

Let T_P, T_Q be two structures of plaintexts, where we want to find a slid pairs $P_i \in T_P, Q_j \in T_Q$. Let $|T_P| = |T_Q| = D$. If we would take all possible pairs P_i, Q_j and we would check if they form a slid pair the time complexity would be equal to D^2. We note because of secret S-box, the check can not be done easily.

Authors of the paper Dunkelman et al. [2020] have better solution and they introduce the algorithm with the time complexity equal to $2D$.

3.3.2 Construction of hypercubes

Let us have two sets of plaintexts T_P and $T_{\tilde{Q}}$. We start with plaintexts $P_i \in T_P$, we fix t s-bit blocks $a_k, 1 \leq k \leq t$ and we compute their t-dimensional hypercubes using $\alpha \in \mathbb{F}_2^t$ and a_k. We create hypercubes of plaintexts $P_{i,\alpha}$ described in 3.3. Then we compute ciphertexts C_i of all plaintexts in all created hypercubes. For every $P_{i,\alpha}$ in a hypercube we have corresponding $C_{i,\alpha}$. We take plaintexts $Q_j \in T_{\tilde{Q}}$ and compute Q_j using (3.1). We fix s-bit blocks $b_k, 1 \leq k \leq t$ and we compute t-dimensional hypercubes of $Q_{j,\alpha}$ using $\tilde{Q}_j$ and b_k. Again we use the construction of hypercube described in 3.3. Then we compute ciphertexts $D_{j,\alpha}$ corresponding to $Q_{j,\alpha}$ and $\tilde{D}_{j,\alpha}$ using (3.2). Finally, we have hypercubes of plaintexts $P_{i,\alpha}$ and corresponding ciphertexts $C_{i,\alpha}$. We also have hypercubes of plaintexts $Q_{j,\alpha}$ (based on $\tilde{Q}_{j,\alpha}$) and their corresponding ciphertexts $D_{j,\alpha}$ and $\tilde{D}_{j,\alpha}$.

3.3.3 Detection of hypercubes of slid pairs

We will try to match created hypercubes to obtain a hypercube of slid pairs.

The Lemma4 implies that the equality $Coll_{P_i} = Coll_{Q_j}$ holds.

We will now describe, how to use the Lemma 4 to match hypercubes.

We search for match of lists corresponding to some P_i and Q_j. For every Q_j separately we try to a match of lists in a hash table. Match occurs when $Coll_{P_i} = Coll_{Q_j}$. If we find a match, then we have a hypercube of slid pairs with high probability.

It means we have a slid pair P_i and Q_j and $2^t - 1$ pairs in the hypercube, where all of them are slid pairs with high probability. We store all found pairs in list L. However, the dimension t of hypercubes must be chosen correctly to have enough matches and high probability.

Input: arbitrary vectors a_k, b_k whose k-th s-bit block is the only non-zero and $1 \leq k \leq t.$, dimension t and structures of plaintexts $T_P, T_{\tilde{Q}}$ such that $|T_P| = |T_{\tilde{Q}}| = D$

Output: list L of pairs of plaintexts witch are with high probabiliy slid pairs

Compute T_Q from $T_{\tilde{Q}}.$;

Ask for the encryption of two structures $T_P, T_Q.$;

Initialize an empty list $L.$;

Construction of hypercubes:

for *each plaintext* $P_i \in T_P$ **do**

Compute the $2^t - 1$ plaintexts in the hypercube of plaintexts $P_{i,\alpha}$ using vectors a_k and find corresponding ciphertexts $C_{i,\alpha}.$;

Find all collisions of the form $(C_{i,\alpha})_l = (C_{i,\beta})_l,$;

Store them in a list $Coll_{P_i}$ of triples (α, β, l) arranged in lexicographic order, along with the value P_i used to create them.

end

for *each plaintext* $\tilde{Q}_j \in T_Q$ **do**

Compute Q_j, the $2^t - 1$ plaintexts $\tilde{Q}_{j,\alpha}$ and all corresponding $Q_{j,\alpha}$ in the hypercube of plaintexts using vectors b_k. Then find corresponding ciphertext $D_{j,\alpha}$), and compute values $\tilde{D}_{j,\alpha},$;

Find all collisions of the form $(\tilde{D}_{j,\alpha})_l = (\tilde{D}_{j,\beta})_l,$;

Store them in a list $Coll_{Q_j}$ of triples (α, β, l) and check for a match $Coll_{P_i} = Coll_{Q_j}$ in the hash table.

end

Detection of hypercubes of slid pairs:

for *each match in the table* **do**

Add the corresponding pair (P_i, Q_j) and corresponding $2^t - 1$ pairs in the hypercube of pairs of plaintexts to $L.$;

end

Recovering secret material:

for *each slid pair* $(P_i, Q_j) \in L$ **do**

Use the relation between P_i and $\tilde{Q}_j$ to detect an input/output pair of $SB \circ K$ for each byte, until the entire function is detected.

end

Key recovery:

Once $SB \circ K$ in all s-bit blocks is detected, find the final key whitening operation K using a single trial encryption.

Algorithm 1: A slide attack using hypercubes of slid pairs

3.3.4 Recovering secret material and key recovery

The algorithm of recovery is similar to the algorithm in the section 3.2.5 for slid sets attack.

3.3.5 The data complexity

Let us have to structures of plaintexts T_P, T_Q same as in the algorithm. Let $|T_P| = |T_Q| = D$. We consider all possible pairs $P_i \in T_P, Q_j \in T_Q$. For every pair we can construct a t-dimensional hypercube of pairs of plaintexts. Then the expected number of t-dimensional hypercubes of slid pairs which can be found in our data is equal to $D^2 \cdot 2^{-n} \cdot 2^{-ts}$. The value D^2 is the number of all possible hypercubes of plaintexts created using some fixed vectors a_k (see the definition of the hypercube 3.3). The value $2^{-ts} \cdot 2^{-n}$ is the probability that the hypercube of pairs of plaintexts is a hypercube of slid pairs. Every hypercube of slid pairs contains 2^t slid pairs. Then $D^2 \cdot 2^{-ts} \cdot 2^{-n} \cdot 2^t$ is the expected number of slid pairs which can be found in our data.

By coupon collector's problem we need at least $2^s \ln(2^s)$ slid pairs for the key recovery. It means our data must contain enough slid pairs and we will show that if we choose correct t, the algorithm will find them with the high probability. Firstly, we will determine D.

By the coupon collector's problem the equality

$$D^2 \cdot 2^{-n} \cdot 2^{-ts} \cdot 2^t = D^2 \cdot 2^{-n+t(1-s)} \geq 2^s \ln(2^s)$$

must hold. Then

$$D^2 \geq \ln(2) \cdot s \cdot 2^{n+s+t(s-1)}.$$

We have some fixed n and s. We want D be as small as possible, because of the data complexity. Thus

$$D = \sqrt{\ln(2) \cdot s} \cdot 2^{(n+s+t(s-1))/2}.$$

For simplicity we round $\sqrt{\ln(2)}$ up to 1. Then $D = \sqrt{s} \cdot 2^{(n+s+t(s-1))/2}$.

To reduce the data complexity, we want t to be as smallest as possible. Thus we have to find the smallest t for which the algorithm works.

Authors of Dunkelman et al. [2020] proved and heuristically verified that $t = s/2 + 1$ is the optimal choice of t. The probability of a random match of two plaintexts $P_i \in T_P$ and $Q_j \in T_Q$ is equal to 2^{-2n}. Authors explained that if $t > s/2 + 1$ then the probability of a match of $Coll_{P_i}$ and $Coll_{Q_j}$ is smaller than 2^{-2n}. Which means that the algorithm detects only hypercubes of slid pairs.

While we construct hypercubes from plaintexts in T_P and T_Q we need to create new plaintexts. However, we can construct T_P and $T_{\tilde{Q}}$ such that they contain all friend pairs already. It is useful, because we do not have to create new plaintexts and thus the amount of data will remain the same.

We take smaller structures, in general case we take general λ-set with s-bit blocks, where t of them are active. Let P and $\tilde{Q}$ be such structures. If we construct a hypercube and we use pair $P_i \in P$, $\tilde{Q}_j \in \tilde{Q}$, then created friend pairs (and other created pairs) are elements of P and $\tilde{Q}$. One such structure contains 2^{ts} plaintexts. Consequently, we need $\sqrt{s} \cdot 2^{(n+s-t(s+1))/2}$ such structures in S_P and

in $S_{\tilde{Q}}$. Hence S_P and $S_{\tilde{Q}}$ are unions of such λ-sets with t s-bit blocks active. We can get S_Q from $S_{\tilde{Q}}$. Here we have some general Q and $\tilde{Q}$, substitution depends on the cipher.

In conclusion, we will need $2 \cdot \sqrt{s} \cdot 2^{(n+s+t(s-1))/2} = \sqrt{s} \cdot 2^{(n+s+t(s-1))/2+1}$ chosen plaintexts. Thus the data complexity is $\sqrt{s} \cdot 2^{(n+s+t(s-1))/2+1}$, the time complexity is $\sqrt{s} \cdot 2^{(n+s+t(s-1))/2+1}$ encryptions and the same is the memory complexity, it is equal to $\sqrt{s} \cdot 2^{(n+s+t(s-1))/2+1}$.

Conclusion

Let us summarize this thesis. The aim of this thesis was to explain main idea of original slide attack [Biryukov and Wagner, 1999] as well as the main idea of two selected new slide attacks [Dunkelman et al., 2020].

The first chapter presents some preliminaries which are needed to understand the thesis, notation and terminology. The second chapter provides an explanation of the main idea of the original slide attack[Biryukov and Wagner, 1999]. The third chapter introduces two new slide attacks on SPN [Dunkelman et al., 2020] – a slid sets attack and a slide attack using a hypercube of slid pairs. A slid sets attack uses special structures to get more slid pairs which are called slid sets. In such slid sets are in general slid pairs which are not determined exactly. A slide attack using hypercube of slid pairs uses hypercubes to detect more slid pairs. It creates such hypercube using one slid pair and his friend pairs.

We describe two parts of the algorithm of attack on 1K-AESfs. The variant of the slid sets attack on 1K-AESts is also explained. In addition we provide some computations for general case of the attack.

Bibliography

Robert El Bashir. Matematická kryptografie a kryptoanalýza 1 - lecture notes. URL https://sent.cz/TK/symetricke_kryptosystemy.pdf.

Alex Biryukov and David A. Wagner. Slide attacks. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999. doi: 10.1007/3-540-48519-8_18. URL https://doi.org/10.1007/3-540-48519-8_18.

Alex Biryukov and David A. Wagner. Advanced slide attacks. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000. doi: 10.1007/3-540-45539-6_41. URL https://doi.org/10.1007/3-540-45539-6_41.

Hans Delfs and Helmut Knebl. *Introduction to Cryptography - Principles and Applications, Third Edition*. Information Security and Cryptography. Springer, 2015. ISBN 978-3-662-47973-5. doi: 10.1007/978-3-662-47974-2. URL https://doi.org/10.1007/978-3-662-47974-2.

Orr Dunkelman, Nathan Keller, Noam Lasry, and Adi Shamir. New slide attacks on almost self-similar ciphers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 250–279. Springer, 2020. doi: 10.1007/978-3-030-45721-1_10. URL https://doi.org/10.1007/978-3-030-45721-1_10.

Marco Ferrante and Monica Saltalamacchia. The coupon collector's problem. *MATerials MATemàtics*, 2014(2), 2014. URL https://mat.uab.cat/~matmat/PDFv2014/v2014n02.pdf.

Hugh Gordon. *Discrete probability*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1997. ISBN 0-387-98227-2. doi: 10.1007/978-1-4612-1966-8. URL https://doi-org.ezproxy.is.cuni.cz/10.1007/978-1-4612-1966-8.

Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011. ISBN 978-3-642-17341-7. doi: 10.1007/978-3-642-17342-4. URL https://doi.org/10.1007/978-3-642-17342-4.

Andrew Kozlík. Úvod do kryptografie - lecture notes. a. URL https://www2.karlin.mff.cuni.cz/~kozlik/udk_mat/des.pdf.

Andrew Kozlík. Úvod do kryptografie - lecture notes. b. URL https://www2.karlin.mff.cuni.cz/~kozlik/udk_mat/hash.pdf.

Andrew Kozlík. Úvod do kryptografie - lecture notes. c. URL https://www2.karlin.mff.cuni.cz/~kozlik/udk_mat/spn.pdf.

Jiří Matoušek and Jaroslv Nešetřil. *Kapitoly z diskrétní matematiky*. Čtvrté, upravené a doplněné vydání. Karolinum, Praha, 2019. ISBN 978-80-246-1740-4.

Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer, 2010. ISBN 978-3-642-04100-6. doi: 10.1007/978-3-642-04101-3. URL https://doi.org/10.1007/978-3-642-04101-3.