Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Integrálna kryptoanalýza a jej aplikácie

Diplomová práca

2020
bc. Roman Ševaňák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Integrálna kryptoanalýza a jej aplikácie

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Martin Stanek, PhD.

Bratislava, 2020
bc. Roman Števaňák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Roman Števaňák
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Integrálna kryptoanalýza a jej aplikácie
Integral cryptanalysis and its applications

Anotácia: Metódy integrálnej kryptoanalýzy sú aktívne využívané pri skúmaní vlastností a odolnosti najmä blokových šifier a kryptografických hašovacích funkcií. Cieľom diplomovej práce je oboznámiť sa s týmito metódami a ich následná aplikácia a prípadné prispôsobenie na vybrané kryptografické konštrukcie. Súčasťou práce je experimentálne overenie správnosti výsledných kryptoanalytických útokov a odhad ich zložitosti.

Vedúci: doc. RNDr. Martin Stanek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 08.11.2018

Dátum schválenia: 15.12.2018 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Integrálna kryptoanalýza patrí spolu s lineárnou a diferenčnou kryptoanalýzou k základným druhom útokov, voči ktorým sú blokové šifry testované. V práci predstavíme zlepšenie integrálnej kryptoanalýzy a na jednoduchej šifre demonštrujeme, že pomocou neho je možné zaútočiť na väčší počet kôl šifry. Následne toto zlepšenie implementujeme do automatizovaného nástroja pre integrálnu kryptoanalýzu Solvatore. Nakoniec odhadneme časovú a priestorovú zložitosť takéhoto útoku.

Kľúčové slová: kryptografia, integrálna kryptoanalýza, Solvatore, blokové šifry

Abstract

Integral cryptanalysis, along with linear and differential cryptanalysis, is one of the basic types of cryptographic attacks, against which block ciphers are tested. In this thesis we propose improvement on integral cryptanalysis and demonstrate on a simple cipher that it can be used to attack more encryption rounds. We implement this enhancement into Solvatore – automatized tool for integral cryptanalysis. Finally, we estimate time and space complexity of such attack.

Keywords: cryptography, integral cryptanalysis, Solvatore, block ciphers

Obsah

Úvod

Integrálna kryptoanalýza patrí spolu s diferenčnou a lineárnou kryptoanalýzou medzi základné typy kryptografických útokov, proti ktorým sú blokové šifry testované. Táto oblasť je stále aktívna vďaka aplikáciám útoku na ďalšie šifry, rôznym rozšíreniam alebo jej automatizáciou, napríklad použitím SAT solverov [7] alebo lineárneho programovania [11].

Pri integrálnej kryptoanalýze zameranej na bity sa používa množina otvorených textov tvorená špecifickým spôsobom. Vstupné bity sa rozdelia na aktívne a konštantné. Následne je vytvorená taká množina textov, že na miestach aktívnych bitov sú v rámci množiny všetky možné hodnoty, a na miestach konštantných bitov sú ľubovoľné, ale pre všetky texty rovnaké hodnoty.

Myšlienka zlepšenia integrálnej kryptoanalýzy je v tom, že ak budú za konštantné bity zvolené konkrétne hodnoty, pri niektorých šifrách to spôsobí, že integrálna kryptoanalýza bude môcť byť použitá pre viac kôl šifrovania.

Cieľom práce je preskúmať tento typ integrálnej kryptoanalýzy, ktorý používa konštantné bity s konkrétnymi hodnotami. Najskôr demonštrujeme jej použitie na jednoduchej šifre, kde je možné použiť aj metódy, ktoré by boli pre šifry používané v praxi výpočtovo nerealizovateľné. Ďalším cieľom je zlepšiť tieto metódy natoľko, aby boli použiteľné aj na praktické šifry.

Prínosom našej práce sú dôkazy niektorých implicitných tvrdení v práci opisujúcej Solvatore [7] – automatizovaný nástroj na integrálnu kryptoanalýzu šifier. Ukážeme, že použitie konštantných bitov s konkrétnymi hodnotami umožní zaútočiť na viaceré kolá jednoduchej šifry. Navrhneme nový spôsob hľadania integrálnych rozlišovačov, ktoré používajú konštantné bity s konkrétnymi hodnotami tzv. modrých rozlišovačov. Implementujeme dvoma spôsobmi hľadanie modrých rozlišovačov v Solvatore [7]. Nakoniec analyzujeme časovú a priestorovú zložitosť jednoduchej integrálnej kryptoanalýzy.

V prvej kapitole vysvetlíme integrálny útok, integrálne rozlišovače a ich použitie. Uvedieme rámec Solvatore a dokážeme niektoré časti, ktoré sú v pôvodnej práci iba implicitné. V druhej kapitole skúmame platnosť myšlienky modrých rozlišovačov pre jednoduchú šifru RES. Používame pritom aj časovo náročné metódy, ktoré nemusia byť výpočtovo uskutočniteľné pri v praxi používaných šifrách. Uvádzame aj nový spôsob hľadania integrálnych rozlišovačov s použitím modrých bitov. Tretia kapitola obsahuje

spôsoby implementácie hľadania integrálnych rozlišovačov s použitím modrých bitov do rámca Solvatore a ich testovanie na zjednodušenej šifre, ako aj na praktickej šifre Speck. Štvrtá kapitola obsahuje jednoduchý všeobecný model integrálneho útoku použitím nájdených integrálnych resp. modrých rozlišovačov, spolu s odhadmi časových a priestorových požiadaviek takéhoto útoku.

Kapitola 1

Integrálna kryptoanalýza a Solvatore

Táto kapitola obsahuje úvod do integrálnej kryptoanalýzy a jej použitia na extrakciu kľúča. Ukážeme, že na integrálny rozlišovač sa dá pozrieť ako na deriváciu šifrovacích funkcií, uvedieme division property zameranú na bity a rámec Solvatore, ktorý ju požíva na automatizované hľadanie integrálnych rozlišovačov. Dokážeme, že stratégia, ktorá je použitá v Solvatore na hľadanie integrálnych rozlišovačov s čo najviac konštantnými bitmi vždy nájde taký rozlišovač, čo je v pôvodnej práci len implikované.

1.1 Integrálny útok

Prvé zmienky o integrálnom útoku sú v práci Daemena a i. [3], kde bol použitý ako útok zameraný špecificky na blokovú šifru Square, a teda bol známy ako *Square útok*. Pod týmto menom bol aplikovaný na viaceré šifry, medzi inými na Camellia [19], Rijandel [8], Hierocrypt [1] alebo Crypton [5]. Pod názvom *saturačný útok* bol použitý na šifru Twofish [12]. Neskôr bol sformalizovaný ako *integrálny útok* v práci autorov Knudsena a Wagnera [10].

1.1.1 Základy integrálneho útoku na blokovú šifru

Integrálny útok funguje v scenári chosen-plaintext attack (CPA) – útok s voľbou otvoreného textu, čo znamená, že útočník môže ľubovoľne veľa krát použiť šifrovacie orákulum s pevne zvoleným neznámym kľúčom na zašifrovanie ľubovoľného textu. Cieľom integrálnej kryptoanalýzy je obvykle odhalenie použitého kľúča.

Nech $E_k : \{0,1\}^n \to \{0,1\}^n$ je bijektívna šifrovacia funkcia prebiehajúca v kolách. Má n vstupných bitov, n výstupných bitov a $k \in \{0,1\}^l$ je kľúč šifrovacej funkcie dĺžky l. Funkcia $E_k^{-1} : \{0,1\}^n \to \{0,1\}^n$ je dešifrovacia funkcia a šifra je korektná – $\forall x \in \{0,1\}^n : x = E_k^{-1}(E_k(x))$. *Aktívne bity* sú podmnožinou vstupných bitov s indexmi v množine $A \subset \{1,\ldots,n\}$ a *konštantné bity* budú tie vstupné bity, ktoré nie sú aktívne. Nech S je označenie pre množinu binárnych vektorov $v \in \{0,1\}^n$ reprezen-

tujúcu množinu otvorených textov, textov v procese šifrovania alebo šifrových textov. Na prvky množiny S sa dá pozerať aj ako na prvky GF(2^n), napríklad pri sčítaní. Označenie $v[i]$ znamená i-ty bit binárneho vektoru v. *Rez* i množinou S, označený $S[i] = \{v[i] \mid v \in S\}$ je sada i-tych bitov z každého prvku množiny S. V reze sa môžu hodnoty opakovať.

Pri integrálnom útoku použijeme množinu otvorených textov S_0 pre aktívne bity s indexmi v A. Táto množina je vytváraná tak, že na miestach aktívnych bitov sa musia vyskytovať všetky variácie hodnôt $\{0, 1\}$ – nech je a aktívnych bitov s indexmi $A = \{i_1, \ldots, i_a\}$, potom $\{(v[i_1], \ldots, v[i_a]) \mid v \in S_0\} = \{0, 1\}^a$. Na miestach konštantných bitoch sú ľubovoľné, ale medzi textami vždy rovnaké hodnoty – $\forall c \in \{1, \ldots, n\} - A, \forall u, v \in S_0 : u[c] = v[c]$. Množina S_0 má mohutnosť $2^{|A|}$ a pre množinu indexov aktívnych bitov A existuje $2^{n-|A|}$ rôznych množín S_0.

Príklad 1.1. Pri šifre so 4 vstupnými bitmi a aktívnymi bitmi s indexmi $A = \{1, 2\}$, kde za konštantné bity na posledných dvoch pozíciách zvolíme hodnoty 1 a 0, je množina otvorených textov $S_0 = \{0010, 0110, 1010, 1110\}$.

Definícia 1.2. *Integrálny rozlišovač* je dvojica $I = (A, B)$ pre konkrétnu šifru $E_k : \{0, 1\}^n \to \{0, 1\}^n$ s l bitmi kľúča, kde $\emptyset \neq A \subset \{1, \ldots, n\}$ je množina indexov aktívnych bitov a $\emptyset \neq B \subseteq \{1, \ldots, n\}$ je množina indexov *balansovaných bitov*. Po vytvorení množiny S_0 otvorených textov s ľubovoľnými hodnotami na miestach konštantných bitov a ich následnom zašifrovaní ľubovoľným kľúčom k bude platiť, že pre každý balansovaný bit $b \in B$ sčítanie všetkých prvkov rezu b množinou zašifrovaných textov bude 0 :

$$\forall S_0, \forall k \in \{0, 1\}^l, \forall b \in B : \sum_{x \in \{E_k(v) | v \in S_0\}} x[b] = 0.$$

V práci uvádzame symbolický zápis $\exists I = (A, B)$ keď existuje integrálny rozlišovač pre konkrétnu šifru, kde je z kontextu jasné o akú šifru ide.

V práci sú použité integrálne rozlišovače zamerané na bity, ale je možné túto definíciu rozšíriť aj tak, aby zahŕňala aj prácu nad slovami z inej grupy ako GF(2), napríklad GF(2^8), a teda by sa vyberali aktívne a balansované bajty namiesto bitov [10].

1.1.2 Zistenie kľúča/podkľúča použitím integrálneho rozlišovača

Ilustrujeme použitie integrálneho rozlišovača na integrálnu kryptoanalýzu šifry Rijndael, známejšej ako AES [13]. Šifra má 128 vstupných a výstupných bitov. AES vytvorí z kľúča podkľúče pre každé kolo, ktoré sú použité pri šifrovaní a sú cieľom kryptoanalýzy. Ak existuje integrálny rozlišovač pre r kôl šifry, môže byť použitý pri útoku na $r + 1$ kôl.

Pre AES existuje integrálny rozlišovač $I = (A, B)$ pre 3 kolá, kde aktívne bity sú $A = \{1, 2, \ldots, 8\}$, a všetky výstupné bity sú balansované, $B = \{1, 2, \ldots, 128\}$ [10].

Tento môže byť použitý pri útoku na šifru 4-kolový AES. Najskôr si útočník vytvorí množinu otvorených textov S_0 s danými aktívnymi bitmi A a pomocou šifrovacieho orákula každý prvok množiny S_0 zašifruje. Nech S_4 je množina šifrových textov po použití 4 kôl šifrovania, ktorú útočník pozná a S_3 množina šifrových textov po použití 3 kôl šifrovania, ktorú útočník nepozná, iba vie, že pre ňu platí podmienka pre balansované bity integrálneho rozlišovača.

Útočník môže vďaka štruktúre AES hádať každý bajt podkľúča samostatne. Najskôr tipne jeden bajt podkľúča, čo použije na čiastočné dešifrovanie jedného kola šifrovania jedného balansovaného bajtu pre každý vektor z S_4. Všetky takto dešifrované bajty sčíta a overí, či ich súčet je 0. Ak nie je, tipnutý bajt podkľúča je nesprávny, pretože pre množinu S_3 takýto súčet platí, keďže každý bit je balansovaný. Ak áno, daná hodnota tipnutého bajtu podkľúča sa môže zaradiť medzi potenciálne hodnoty.

Ak je potenciálnych hodnôt pre bajt podkľúča viac, môžu sa overiť pomocou iného integrálneho rozlišovača, alebo rovnakého integrálneho rozlišovača s množinou S_0 v ktorej budú iné konštantné bity, a teda aj množina S_4 vyrobené z nej bude rozdielna.

Útok sa dá rozšíriť na 5-kolový AES, kde sa pre piate kolo uhádne 1 bajt posledného podkľúča, vo štvrtom kole 4 bajty predposledného podkľúča, čiastočne sa dešifrujú posledné dve kolá pre jeden balansovaný bajt a overí sa jeho súčet ako predtým. Existuje aj možnosť rozšírenia na 6 kôl tak, že integrálny rozlišovač bude fungovať od druhého kola šifry, nie od začiatku. Je možné pripraviť 2^{32} otvorených textov tak, že pre sa budú hádať 4 bajty pre podkľúč prvého kola, a pre každý možný tip sa bude dať nájsť taká množina 256 textov, ktorá spĺňa podmienky aby bola braná ako S_0 pre integrálny rozlišovač. Následne sa použije už skôr spomínaný integrálny rozlišovač pre 5 kôl, teda bude sa hádať vždy 9 bajtov. Podrobnosti základného útoku na AES, ako aj týchto modifikácií je možné nájsť v práci Knudsena a Wagnera [10].

1.1.3 ANF a integrálny rozlišovač ako derivácia

Každá šifra $E_k : \{0,1\}^n \to \{0,1\}^n$ sa dá zapísať ako n boolovských funkcií $f_{k,i} : \{0,1\}^n \to \{0,1\}$ pre $i \in \{1, \ldots, n\}$, kde výsledok každej $f_{k,i}$ predstavuje i-ty výstupný bit. Každá z týchto funkcií $f_{k,i}$ môže byť zapísaná ako:

$$f_{k,i}(x_1, x_2, \ldots, x_n) = \sum_{I \subseteq \{1,2,\ldots,n\}} a_I \prod_{j \in I} x_j$$

kde $a_I \in \{0,1\}$ sú konštanty a každý súčin $\prod_{j \in I} x_j$, pre ktorý $a_I = 1$ nazývame *monóm*. Takáto reprezentácia sa nazýva algebraická normálna forma (ANF).

Nasledujúce definície sú prebraté od autorov Martín del Rey a i. [4]. Nech $e_i \in \{0,1\}^n$ je binárny vektor obsahujúci jedinú 1 na i-tom mieste. *Parciálna deri-*

vácia n-árnej boolovskej funkcie f vzhľadom na i-tu premennú je n-árna boolovská funkcia $D_i f : \{0,1\}^n \to \{0,1\}$ definovaná nasledovne (+ označujeme sčítanie binárnych vektorov ako prvkov nad GF(2^n) aj sčítanie funkčných hodnôt ako prvkov nad GF(2)):

$$\begin{aligned} D_i f(x) &= f(x) + f(x + e_i) \\ &= f(x_1, \ldots, x_i, \ldots, x_n) + f(x_1, \ldots, x_i + 1, \ldots, x_n). \end{aligned}$$

Skladanie parciálnych derivácií n-árnej boolovskej funkcie f vzhľadom na jej i-tu a j-tu premennú je definované nasledovne:

$$\begin{aligned} (D_i \circ D_j) f(x) = D_i(D_j f)(x) &= D_j f(x) + D_j f(x + e_i) \\ &= f(x) + f(x + e_j) + f(x + e_i) + f(x + e_j + e_i). \end{aligned}$$

Navyše pre vzájomne rôzne $i_1, i_2, \ldots, i_k \in \{1, \ldots, n\}$ platí

$$(D_{i_1} \circ D_{i_2} \circ \ldots \circ D_{i_k}) f(x) = D_{i_1}(D_{i_2}(\ldots(D_{i_k}(f))))(x).$$

Vo všeobecnom prípade, pri zloženej derivácii vzhľadom na bity s indexmi $A = \{i_1, \ldots, i_k\}, k \in \mathbb{N}, k \leq n$ bude platiť

$$D_A f(x) = (D_{i_1} \circ D_{i_2} \circ \ldots \circ D_{i_k}) f(x) = \sum_{I \subseteq \{i_1, \ldots, i_k\}} f\left(x + \sum_{i \in I} e_i\right).$$

Príklad 1.3. Funkcia f je $f(x_1, x_2, x_3, x_4) = x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_4$ a $A = \{1, 2\}$, potom derivácia vzhľadom na x_1 a x_2 bude $D_A f(x) = x_3 + 1$.

Nech je a aktívnych bitov s indexmi v množine $A = \{i_1, i_2, \ldots, i_a\}$ a c konštantných bitov s indexmi $C = (j_1, j_2, \ldots, j_c)$. Pri tvorbe S_0 boli použité konštantné bity $cv = (c_1, c_2, \ldots, c_c) \in \{0,1\}^c$ tak, že $cv[i]$ je hodnota $C[i]$-teho vstupného bitu – $\forall i \in \{1, \ldots, c\}, x_{C[i]} = cv[i]$.

Sčítanie množiny šifrových textov $S = \{E_k(v) \mid v \in S_0\}$ je ekvivalentné dosadeniu konštantných bitov do n boolovských funkcií zderivovaných vzhľadom na všetky aktívne bity.

Nech x je binárny vektor taký, že pre všetky $i \in \{1, \ldots, c\}$ je na $C[i]$-tom mieste hodnota $cv[i]$ a ostatné bity sú ľubovoľné – $x \in \{0,1\}^n, \forall i \in \{1, \ldots, c\} : x[C[i]] = cv[i]$. Potom platí:

$$\sum_{v \in S_0} E_k(v) = \left(D_A f_{k,1}(x), D_A f_{k,2}(x), \ldots, D_A f_{k,n}(x)\right).$$

Príklad 1.4. Nech $A = \{1, 2\}$ a za konštantné bity v S_0 sú zvolené hodnoty $c_3, \ldots, c_n \in \{0,1\}$. Bit na j-tom mieste v súčte šifrových textov sa dá zapísať ako

$$f_{k,j}(0,0,c_3,\ldots,c_n) + f_{k,j}(0,1,c_3,\ldots,c_n) + f_{k,j}(1,0,c_3,\ldots,c_n) + f_{k,j}(1,1,c_3,\ldots,c_n).$$

Keď do zderivovanej funkcie $D_A f_{k,j}$ dosadíme za konštantné bity $c_3, \ldots, c_n$ a za aktívne bity ľubovoľné hodnoty x_1 a x_2, výsledok bude

$$D_A f_{k,j}(x_1, x_2, c_3, \ldots, c_n) = f_{k,j}(x_1, x_2, c_3, \ldots, c_n) + f_{k,j}(x_1, x_2 + 1, c_3, \ldots, c_n) + f_{k,j}(x_1 + 1, x_2, c_3, \ldots, c_n) + f_{k,j}(x_1 + 1, x_2 + 1, c_3, \ldots, c_n).$$

čo je ekvivalentné predchádzajúcemu zápisu pre súčet šifrových textov, keďže x_1 a x_2 môže nadobúdať iba hodnoty 0 alebo 1, teda v prvých dvoch vstupných hodnotách sa prestriedajú všetky variácie 0 a 1.

1.2 Division property

Integrálna kryptoanalýza bola výrazne rozšírená v práci autora Todo [17] definíciou metódy *division property* a jej použitím pri hľadaní integrálnych rozlišovačov.

Todo a Morii [18] definovali novú metódu *bit-based division property (using three subsets)*, ktorá bola použitá na šifru Simon32 z rodiny Simon [2]. Pre šifru bolo možné pomocou division property nájsť integrálny rozlišovač pre 10 kôl šifrovania, ale bol známy experimentálne nájdený integrálny rozlišovač pre 15 kôl, pre ktorý nebolo isté či funguje pre všetky kľúče.

Pomocou bit-based division property bolo možné nájsť integrálny rozlišovač pre 15 kôl a tým potvrdiť, že existuje integrálny rozlišovač, ktorý funguje pre všetky kľúče. Táto metóda má ale výrazne vyššiu časovú aj priestorovú zložitosť, v najhoršom prípade exponenciálnu od dĺžky bloku šifry, čo znemožnilo jej aplikáciu na šifry s väčšou dĺžkou bloku z rodiny Simon.

1.3 Solvatore

Eskandari a i. [7] navrhli rámec Solvatore na automatizovanú analýzu blokových a prúdových šifier použitím *division property zameranej na bity (conventional bit-based division property)*, čo je rozdielne od bit-based division property. Metóda division property dovoľuje zvoliť si dĺžku slova, ktoré sa označuje ako aktívne alebo konštantné. Pri division property zameranej na bity sa za dĺžku slova zvolí 1 bit, teda o každom bite samostatne hovorí, či je aktívny alebo nie.

Solvatore umožňuje nájsť integrálny rozlišovač, alebo rýchlo zistiť, pri akom počte kôl šifry by integrálny rozlišovač nebolo možné jeho pomocou nájsť. Rámec je zameraný na jednoduchosť používania a jednotlivé kryptografické primitíva sa dajú popísať použitím predpripravených funkcií napríklad pre lineárne operácie, definíciu a aplikáciu S-boxov alebo modulárne pripočítavanie.

Pri hľadaní integrálneho rozlišovača je cieľom nájsť čo najmenej aktívnych bitov A takých, že pre niektorú $f_{k,i} : \{0,1\}^n \to \{0,1\}$ po derivácii vzhľadom na všetky aktívne bity A platí $D_A f_{k,i}(x) = 0$, teda aspoň 1 bit je balansovaný . To znamená, že funkcia $f_{k,i}(x)$ v ANF obsahuje iba monómy, ktoré v sebe nezahŕňajú všetky aktívne bity naraz.

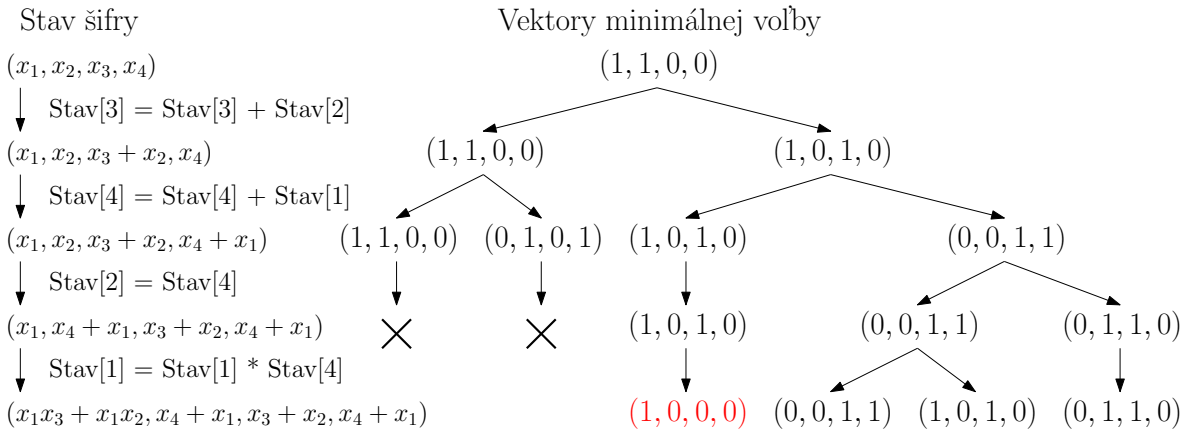
Ak by bola k dispozícii ANF danej funkcie, dalo by sa priamočiaro zistiť, ktoré vstupné bity sa nevyskytujú spolu v žiadnom monóme a teda jednoducho nájsť integrálny rozlišovač, kde by tieto bity boli aktívne a bit, ktorého funkcia sa skúma balansovaný. Počítanie ANF je však výpočtovo ťažko realizovateľné, nakoľko šifrovacia funkcia v ANF môže obsahovať exponenciálne veľa monómov v závislosti od počtu vstupných bitov. Solvatore preto používa zjednodušenie výpočtu – počas výpočtu sa nesleduje celá ANF, iba sa uchováva informácia, ktoré funkcie zodpovedajúce bitom stavu by bolo potrebné vynásobiť, aby vo výsledku v ANF vznikol monóm obsahujúci všetky aktívne bity.

Táto informácia je opísaná vo *vektoroch minimálnej voľby (minimal choice vectors)* $MCV = \{v \mid v \in \{0,1\}^n\}$, kde n je počet vstupných a výstupných bitov šifry. Každý *vektor minimálnej voľby* v reprezentuje jednu možnosť súčinu funkcií šifrovania, po ktorom by vznikol monóm obsahujúci všetky aktívne bity. Ak v takej možnosti súčinu figuruje boolovská funkcia pre i-ty bit v aktuálnom stave šifry, v minimálnom vektore voľby je na i-tom mieste 1, inak je na ňom 0.

Nech m_i je minimálny vektor voľby s Hammingovou váhou 1, s jedinou 1 na i-tom mieste. Nech M je množina všetkých takýchto minimálnych vektorov voľby – $M = \{m_i \mid i \in \{1,..,n\}\}$. Ak je v množine MCV nejaký vektor m_i, znamená to, že vo funkcii pre i-ty bit v aktuálnom stave šifry sa môže nachádzať monóm obsahujúci všetky aktívne bity, teda nevieme, čo bude výsledok funkcie po derivácii vzhľadom na aktívne bity A. Naopak, ak by v množine MCV nebol niektorý m_i, vieme, že funkcia pre i-ty bit v aktuálnom stave šifry bude po derivácii vzhľadom na aktívne bity A určite 0, lebo neobsahuje monóm v ktorom sú všetky aktívne bity.

Príklad 1.5. Uvažujme šifrovaciu funkciu s 4 vstupnými bitmi x_1, x_2, x_3, x_4 a aktívnymi bitmi $A = \{1, 2\}$. Stav šifry na začiatku je (x_1, x_2, x_3, x_4). Iniciálna množina MCV obsahuje iba $(1, 1, 0, 0)$, nakoľko jediný spôsob ako dostať monóm obsahujúci x_1x_2 je znásobiť prvé 2 bity aktuálneho stavu. Iniciálna množina MCV vo všeobecnosti bude vždy obsahovať iba jediný vektor, kde budú 1 na miestach aktívnych bitov a 0 na ostatných.

Nech sa v rámci šifrovania ako prvá vykoná operácia pripočítania druhého bitu k tretiemu bitu, teda nový stav šifry bude $(x_1, x_2, x_3 + x_2, x_4)$. Množina MCV teraz obsahuje 2 vektory – $(1, 1, 0, 0)$ a $(1, 0, 1, 0)$, keďže monóm obsahujúci x_1x_2 môžeme získať vynásobením prvých dvoch bitov ako predtým, ale v novom stave aj vynásobením prvého a tretieho bitu. Rozšírený príklad sa nachádza na obrázku 1.1.



Obrázok 1.1: Ilustrácia transformácií vektorov minimálnej voľby

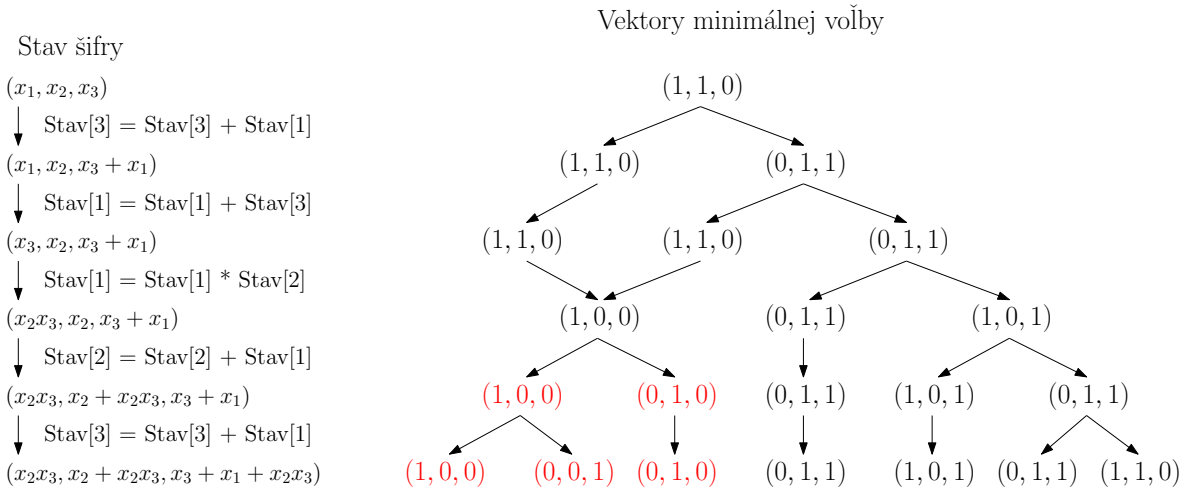
Pre hľadanie integrálnych rozlišovačov stačí sledovať vektory minimálnej voľby, nakoľko presne udržiavať stav a jeho ANF je pre väčšinu šifier používaných v praxi výpočtovo ťažko realizovateľná úloha.

Každá operácia v šifre, napríklad XOR, lineárna operácia alebo použitie S-boxu transformuje množinu MCV predpovedateľným spôsobom. Podobne ako na obrázku 1.1, tieto transformácie sa dajú zapísať ako strom, kde v koreni je iniciálny vektor minimálnej voľby a listy v najhlbšej úrovni tvoria poslednú množinu MCV – po vykonaní všetkých operácií šifrovania. Hľadanie integrálneho rozlišovača prebieha nasledovne – ak v poslednej množine MCV nie je niektorý $m_i \in M$, existuje integrálny rozlišovač, kde i-ty bit je balansovaný.

Takéto zjednodušenie výpočtu však nemusí nájsť každý rozlišovač. Na obrázku 1.2 je ukážka takých operácií. Môžeme vidieť, že v poslednej množine MCV sa nachádzajú všetky vektory s Hammingovou váhou 1, teda podľa Solvatore v takomto prípade po operáciách neexistuje integrálny rozlišovač. Môžeme však v ANF funkcií pre jednotlivé bity vidieť, že ani jeden monóm neobsahuje všetky aktívne bity, teda všetky výstupné bity sú balansované. Tento príklad nie je realistický, avšak demonštruje kde nastáva problém – zjednodušenie výpočtu neudržiava informáciu o tom, aké sú funkcie pre daný bit. Keď sa teda sčítajú funkcie (napríklad druhá operácia na obrázku 1.2) pri zjednodušenom výpočte nie je možné zistiť, že aktívny bit je vykrátený (x_1 na obrázku 1.2) a treba sa držať konzervatívneho odhadu, že sa vo funkcii stále nachádza.

Naopak, ak je pomocou zjednodušeného výpočtu nájdený integrálny rozlišovač, určite existuje, nakoľko sa Solvatore vždy drží konzervatívneho odhadu.

Ukladať celú množinu MCV však môže byť výpočtovo nerealizovateľné, nakoľko počet prvkov môže byť exponenciálny v závislosti od počtu bitov. V Solvatore je tento problém riešený použitím SAT solvera na zodpovedanie otázky „S danými aktívnymi bitmi a postupnými transformáciami množiny MCV zodpovedajúcimi šifrovaniu, je



Obrázok 1.2: Ilustrácia transformácií vektorov minimálnej voľby, kde zjednodušenie zlyhá

konkrétny vektor minimálnej voľby m_i dosiahnuteľný?“. Túto otázku riešia pre každý vektor $m_i \in M$ a ak je odpoveď SAT solvera záporná, teda m_i nie je dosiahnuteľný, môže byť zodpovedajúci i-ty výstupný bit označený ako balansovaný. Ak je balansovaných bitov nenulový počet, bol nájdený integrálny rozlišovač.

1.3.1 Transformácie vektorov minimálnej voľby

Aby bolo možné postupne aplikovať všetky operácie na iniciálny vektor minimálnej voľby, treba vedieť akým spôsobom daná operácia transformuje vektor. Pri operáciách na dvoch bitoch vektora sú aj dopady transformácie iba na dané dva bity vektorov minimálnej voľby. Stačí teda uvádzať dopady na vektory $(0, 0)$, $(0, 1)$, $(1, 0)$ a $(1, 1)$.

Transformácie pre niektoré operácie uvádzame v tabuľke 1.1. Viac informácií je možné nájsť v práci autorov Eskandari a i. [7]. V riadku pre operáciu bit-copy vektorov $(0, 1)$ a $(1, 1)$ sa nenachádzajú žiadne vektory minimálnej voľby. Keď sa použije operácia bit-copy spôsobom, že sa prvý bit skopíruje na miesto druhého, vektory minimálnej voľby $(0, 1)$ a $(1, 1)$ zaniknú, pretože sa prepíše funkcia v druhom bite, ktorá je potrebná na vytvorenie monómu obsahujúceho všetky aktívne bity. Nezáleží na tom, či je nahradená funkciou, ktorá je k vytvoreniu takého monómu tiež potrebná alebo nie, nakoľko sú to vektory minimálnej voľby, nie je možné takto prepísať ani jednu funkciu.

Dopad použitia S-boxu na stav šifry nemusí byť nutne lokálny na dvojici bitov, teda nemusí sa dať určiť na vektoroch dĺžky 2. Jeho celkový dopad je však možné spočítať z ANF funkcií, S-boxu. Jeden spôsob je pre vstupný vektor minimálnej voľby krok po kroku odsimulovať všetky transformácie dané ANF podľa známych pravidiel pre XOR a AND.

	(0,0)	(1,0)	(0,1)	(1,1)
Bit-copy $x_2 := x_1$	(0,0)	(1,0), (0,1)	N/A	N/A
XOR $x_2 := x_1 \oplus x_2$	(0,0)	(1,0), (0,1)	(0,1)	(1,1)
AND $x_2 := x_1 \wedge x_2$	(0,0)	(1,0), (0,1)	(0,1)	(0,1)

Tabuľka 1.1: Dopady niektorých operácií na vektory minimálnej voľby dĺžky 2

Spôsob použitý v Solvatore je funguje tak, že pre každý vstupný minimálny vektor voľby $v_i \in \{0,1\}^n$ a každý výstupný minimálny vektor voľby $v_o \in \{0,1\}^n$ sa skontroluje, či môže takáto transformácia MCV nastať podľa danej ANF a na základe toho sú vytvorené potrebné logické klauzuly. Celkový dopad S-boxu môže byť takto spočítaný iba raz a zapamätaný pre ďalšie použitia.

1.3.2 Preklad do matematickej logiky

V rámci Solvatore sa používa program riešiaci problém splniteľnosti boolovskej funkcie (SAT solver) na simuláciu transformácií vektorov minimálnej voľby a zistenie, či po kompletnom zašifrovaní je nejaký vektor dosiahnuteľný. Na toto je potrebný preklad problému do konjuktívnej normálnej formy (KNF).

Nech $S^i = (s_{(i,1)}, \ldots, s_{(i,n)})$ je vektor premenných reprezentujúcich vektor minimálnej voľby po i operáciách, kde $s_{(i,j)}$ predstavuje j-ty bit vektora minimálnej voľby. Pre S^0 majú premenné $s_{(0,i)}$ hodnotu 1 práve vtedy, keď i-ty vstupný bit šifry je aktívny, hodnotu 0 ak je konštantný. Následne sa každá operácia vytvorí ako vzťah medzi premennými v S^k a S^{k+1}. Pre operácie ako XOR, AND alebo kopírovanie bitu sa kódujú transformácie podľa tabuľky 1.1. Pri použití S-boxu sa aplikujú preň predpočítané klauzuly, ktorých tvorbu sme naznačili v časti 1.3.1.

Všetky operácie sú kódované do klauzúl konjuktívnej normálnej formy a sú pridávané do výslednej logickej formuly. Viac informácií o konkrétnych transformáciách je v pôvodnej práci [7].

1.3.3 Hľadanie a optimalizácia integrálnych rozlišovačov

Nech pre r kôl šifry s n vstupnými a výstupnými bitmi je potrebných h operácií a $\mathrm{hw}(x)$ označuje Hammingovu váhu vektora x. Testujeme, či vektor premenných S^h môže nadobudnúť všetky hodnoty, kde $\mathrm{hw}(S^h) = 1$.

Platí, že ak pre šifrovaciu funkciu existuje integrálny rozlišovač $I = (A, B)$, bude existovať aj $I' = (A', B')$ pre ľubovoľnú nadmnožinu aktívnych bitov $A' \supseteq A$, $A' \subset \{1, \ldots, n\}$ a nejakú nadmnožinu balansovaných bitov $B' \supseteq B$:

$$\exists I = (A, B) \implies \forall A', A \subseteq A' \subset \{1, \ldots, n\}, \exists B', B \subseteq B' : \exists I' = (A', B').$$

Ak pre všetky balansované bity $i \in B$ platí, že $D_A f_{k,i}(x) = 0$, potom aj pre $A' \supset A$ bude platiť $D_{A'} f_{k,i}(x) = 0$, pretože to znamená zderivovanie funkcie vzhľadom na ďalšie vstupné bity. Ak bola derivácia funkcie 0 pred dodatočnými deriváciami, bude aj po nich. Ak sa pre funkciu šifrovania pre niektorý bit $j \notin B$ stane, že po dodatočnej derivácii bude $D_{A'} f_{k,j}(x) = 0$, potom $j \in B'$.

Vďaka tomuto výroku sa dá jednoducho ukázať, že integrálny rozlišovač neexistuje pre daný počet kôl – stačí ukázať, že pre všetky S^0 s $\mathrm{hw}(S^0) = n - 1$ sú všetky S^h s $\mathrm{hw}(S^h) = 1$ dosiahnuteľné. Nie je potrebné skúšať S^0 s $hw(S^0) = n$, pretože to neprináša žiadnu novú informáciu, čo je ukážeme v leme 1.6.

Lema 1.6. Nech $E_k : \{0,1\}^n \to \{0,1\}^n$ je bijektívna šifrovacia funkcia s dĺžkou bloku $n > 1$. Nech sú všetky vstupné bity aktívne – $A = \{1, \ldots, n\}$. Potom $S_0 = \{0,1\}^n$ a súčet každého rezu i množinou zašifrovaných textov S je rovný 0:

$$\forall k \in \{0,1\}^l, \forall i \in \{1, \ldots, n\} : \sum_{x \in \{E_k(v) | v \in S_0\}} x[i] = 0.$$

Dôkaz. Keďže šifrovacia funkcia je bijektívna a $S_0 = \{0,1\}^n$, pre množinu zašifrovaných textov $S = \{E_k(x) \mid x \in S_0\}$ platí $S = \{0,1\}^n$. Ak $n > 1$, súčet každého rezu i takejto množiny je 0 – $\forall i \in \{1, \ldots, n\}, \sum_{x \in S[i]} x = 0$. □

Integrálne rozlišovače s menším počtom aktívnych bitov sú výhodnejšie z viacerých dôvodov. Množina S_0 má exponenciálnu mohutnosť od počtu aktívnych bitov – $|S_0| = 2^{|A|}$, teda každý aktívny bit navyše zdvojnásobí počet otvorených aj zašifrovaných textov, s ktorými musí útočník pracovať v scenári CPA. Ďalší dôvod je, že pri extrakcii kľúča použitím integrálnej kryptoanalýzy, ako je uvedené v časti 1.1.2, sa môže stať, že pre danú množinu S_0 bolo nájdených viac potenciálnych hodnôt podkľúča. Tieto môžu byť znovu overené rovnakým integrálnym rozlišovačom s inými konštantnými bitmi v množine S_0. Čím viac konštantných bitov má integrálny rozlišovač, tým viac rôznych množín S_0 sa dá vytvoriť.

Hľadanie integrálnych rozlišovačov s najmenším počtom aktívnych bitov je možné optimalizovať oproti priamočiarej stratégii skúšania všetkých možných kombinácií indexov aktívnych a konštantných bitov. Jedna stratégia začne so všetkými hodnotami S^0 s $\mathrm{hw}(S^0) = n - 1$, teda všetkými množinami $n - 1$ aktívnych bitov. Ak sa pre nejakú množinu indexov aktívnych bitov A nájde integrálny rozlišovač, vyskúšajú sa pre všetky podmnožiny A s $n-2$ aktívnymi bitmi nájsť ďalšie integrálne rozlišovače, a takto sa iteratívne postupuje až pokiaľ neostane jediný aktívny bit, alebo sa pre žiadnu podmnožinu nenájdu žiadne integrálne rozlišovače.

Stratégia, ktorá je implementovaná v Solvatore (ďalej budeme nazývať stratégia OS) spočíva v tom, že sa stačí pozerať iba na prieniky aktívnych bitov pre integrálne rozlišovače, ktoré majú aspoň jeden spoločný balansovaný bit.

Označenie DP(A) znamená najväčšiu množinu balansovaných bitov takú, že pre konkrétnu šifru s určeným počtom kôl existuje integrálny rozlišovač $I = (A, \mathrm{DP}(A))$, prípadne prázdnu množinu ak žiadny integrálny rozlišovač pre A neexistuje, formálne:

$$ALL\_B(A) = \{B \mid \exists I = (A, B)\} \cup \{\emptyset\}$$
$$\mathrm{DP}(A) = B, B \in ALL\_B(A), \forall B' \in ALL\_B(A) : B' \subseteq B$$

Keďže pre integrálne rozlišovače platí, že ak pre nejakú šifru existuje $I_1 = (A, B_1)$ a $I_2 = (A, B_2)$, potom existuje aj $I = (A, B_1 \cup B_2)$, takéto označenie je jednoznačne určené.

Nech G_1 je množina tých množín A s $n-1$ aktívnymi bitmi, pre ktoré existuje nejaký integrálny rozlišovač:

$$G_1 = \{A \mid |A| = n - 1, \mathrm{DP}(A) \neq \emptyset\}.$$

Nech G_i je množina prienikov množín indexov aktívnych bitov z G_{i-1}, ktorých integrálne rozlišovače majú spoločný niektorý balansovaný bit a existuje integrálny rozlišovač, ktorého aktívne bity sú tento prienik.

$$G_i = \{A_j \cap A_k \mid A_j, A_k \in G_{i-1}, A_j \neq A_k, \mathrm{DP}(A_j) \cap \mathrm{DP}(A_k) \neq \emptyset, \mathrm{DP}(A_j \cap A_k) \neq \emptyset\}.$$

Výpočet prebieha tak, že sa generujú množiny $G_i, i \in \{1, \ldots, n-1\}$, až pokým nie je niektorá prázdna. Posledná neprázdna množina obsahuje aktívne bity pre integrálny rozlišovač s najväčším počtom konštantných bitov, čo dokážeme v leme 1.7.

Lema 1.7. Stratégia OS vždy nájde pre danú šifru niektorý integrálny rozlišovač s najmenším počtom aktívnych bitov $I_{min} = (A, B)$, zároveň s najväčším počtom konštantných bitov $c = n - |A|$. Množina indexov aktívnych bitov A sa bude nachádzať v G_c a $G_{c+1} = \emptyset$.

Dôkaz. V G_1 sú všetky všetky množiny indexov aktívnych bitov, ktoré sú nadmnožinou A : $\{A' \mid A' \supseteq A, |A'| = n - 1\} \subseteq G_1$, pretože ak existuje $I_{min} = (A, B)$, tak $\forall A', A \subseteq A' \subset \{1, \ldots, n\}, \exists B', B \subseteq B'$ existuje $I' = (A', B')$. Vo všeobecnosti platí pre $j \leq k$

$$\{A' \mid A' \supseteq A, |A'| = n - j\} \subseteq G_j$$

pretože ak G_{j-1} obsahovalo všetky nadmnožiny A veľkosti $n - j + 1$, medzi ich vzájomnými prienikmi budú aj všetky nadmnožiny A veľkosti $n - j$.

Pre každú dvojicu nadmnožín $A_1, A_2 \in G_{n-j+1}, A \subseteq A_1, A \subseteq A_2$ platí $\mathrm{DP}(A_1) \cap \mathrm{DP}(A_2) \neq \emptyset$, pretože $\mathrm{DP}(A) \subseteq \mathrm{DP}(A_1), \mathrm{DP}(A) \subseteq \mathrm{DP}(A_2)$ a $DP(A) \neq \emptyset$.

V množine G_c bude určite aj množina indexov aktívnych bitov A, pretože $|A| = n - c$ a $A \supseteq A$.

Ukážeme, že pri hľadaní bola posledná neprázdna množina G_c a nasledujúca množina G_{c+1} bola prázdna. Postupujeme sporom, nech existuje integrálny rozlišovač s indexmi aktívnych bitov $A' \in G_{c+1}$. Keďže sa pri tvorbe každého G_i robí prienik dvoch neidentických množín aktívnych bitov, vždy prienikom aspoň jeden aktívny bit vypadne, teda v každej množine G_i budú množiny indexov aktívnych bitov s najviac $n-i$ prvkami. Ak by teda G_{c+1} bolo neprázdne, znamenalo by to, že existuje integrálny rozlišovač s $c+1$ konštantnými bitmi, čo je spor s výberom I_{min}. □

Kapitola 2

Modré bity v integrálnej kryptoanalýze

V tejto kapitole definujeme modré bity a integrálne rozlišovače s ich použitím. Pre účely testovania vytvoríme jednoduchú šifru RES, na ktorej ukážeme, že použitie modrých bitov môže zvýšiť maximálny počet kôl, pre ktorý existuje integrálny rozlišovač. Uvádzame spôsob hľadania modrých rozlišovačov, ktorý maximalizuje množstvo konštantných bitov.

2.1 Šifra RES

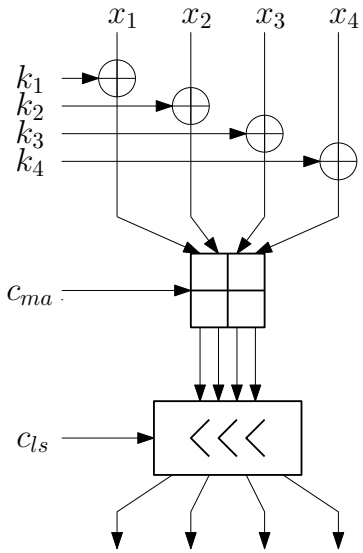
Šifra RES nám slúži na testovanie a demonštráciu možného zlepšenia integrálnych rozlišovačov. Ide o blokovú šifru s variabilnou dĺžkou bloku n. Dĺžka kľúča je rovnaká ako dĺžka bloku. Šifrovanie prebieha v r kolách, ktorých môže byť ľubovoľný počet.

Jedno kolo šifry RES je zložené z 3 operácií:

- Pripočítanie kľúča operáciou XOR
- Modulárne pripočítanie konštanty k stavu šifry
- Cyklický posun vľavo o určený počet bitov

Na obrázku 2.1 je znázornené jedno kolo šifry s dĺžkou bloku $n = 4$. Kľúč je v každom kole identický a po poslednom kole sa nevykonáva opätovné pripočítanie kľúča („key whitening“) kvôli čo najjednoduchšiemu návrhu šifry.

Pripočítanie kľúča na konci šifrovania z pohľadu existencie integrálneho rozlišovača nehrá žiadnu rolu. Je to ekvivalentné pripočítaniu jednočlenného monómu premennej kľúča ku každej funkcii $f_{k,i}(x)$, a teda ak má rozlišovač aspoň jeden aktívny bit, tento monóm nebude súčasťou zderivovanej funkcie. Rozdiel nastáva až pri realizácii integrálnej kryptoanalýzy, kedy je potrebné hádať podkľúč použitý na „key whitening“, spolu s podkľúčom pre posledné kolo.



Obrázok 2.1: Jedno kolo šifrovania RES(c_{ls}, c_{ma}, r)

Konkrétna konfigurácia šifry RES(c_{ls}, c_{ma}, r) je daná konštantou $c_{ls}, 0 \leq c_{ls} < n$, ktorá určuje o koľko bitov sa vykonáva cyklický posun, konštantou $c_{ma}, 0 \leq c_{ma} < 2^n$, ktorá je modulárne pripočítaná k stavu a počtom kôl $r, r > 0$. V práci pracujeme s dĺžkou bloku 4 bity, ak nie je uvedené inak.

2.1.1 Integrálne rozlišovače šifry RES podľa Solvatore

Použitím Solvatore boli zistené najväčšie počty kôl, pre ktoré existuje integrálny rozlišovač. Výsledky sú v tabuľke 2.1. V niektorých konfiguráciách sa ukázalo, že šifra nie je vôbec odolná voči integrálnej kryptoanalýze – existuje integrálny rozlišovač pre ľubovoľný počet kôl, čo dokážeme pomocou lemy 2.1 a lemy 2.2.

Lema 2.1. Pre konfiguráciu šifry RES s konštantou $c_{ls} = 0$ existuje integrálny rozlišovač pre ľubovoľný počet kôl.

Dôkaz. Nech $f_{k,i}(x_1, x_2, x_3, x_4)$ pre $i \in \{1, \ldots, 4\}$ sú funkcie šifrovania pre i-ty výstupný bit RES(0, c_{ma}, r). Vykonanie kola má na stav šifry nasledovný dopad – pripočítanie kľúča spôsobí to, že ku každej funkcii sa pripočíta jednočlenný monóm s premennou kľúča. Pri modulárnom sčítaní s konštantou, nakoľko jeho modulom je mocnina dvojky, viac významné bity neovplyvnia menej významné, teda do funkcií $f_{k,i}(x_1, x_2, x_3, x_4)$ pre $i \in \{2, \ldots, 4\}$ sa nedostane premenná pre najvýznamnejší bit. Funkcie pre menej významné bity teda nebudú obsahovať premennú x_1 a ich derivácia vzhľadom na ňu bude 0. Vďaka tomuto pre ľubovoľný počet kôl existuje integrálny rozlišovač s $A = \{1\}$ a $B = \{2, 3, 4\}$. □

Lema 2.2. Pre konfiguráciu šifry RES s dĺžkou bloku 4 bity a konštantou c_{ma}, ktorá je deliteľná 4 existuje integrálny rozlišovač pre ľubovoľný počet kôl.

c_{ls} \ c_{ma}	0	1	2	3	4	5	6	7
0	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	4	7	4	∞	4	7	4
2	∞	3	5	3	∞	3	5	3
3	∞	4	7	4	∞	4	7	4

c_{ls} \ c_{ma}	8	9	10	11	12	13	14	15
0	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	4	7	4	∞	4	7	4
2	∞	3	5	3	∞	3	5	3
3	∞	4	7	4	∞	4	7	4

Tabuľka 2.1: Maximálny počet kôl pre nájdené integrálne rozlišovače nástrojom Solvatore. Symbolom ∞ označujeme, že existuje integrálny rozlišovač pre ľubovoľný počet kôl.

Dôkaz. Nech $c_{ma}[i]$ predstavuje i-ty najvýznamnejší bit c_{ma}. Ukážeme, že pre konfigurácie RES, kde $c_{ma}[3] = c_{ma}[4] = 0$ bude platiť, že funkcie šifrovania $f_{k,i}(x_1, x_2, x_3, x_4)$ pre $i \in \{1, \ldots, 4\}$ budú mať algebraický stupeň 1, teda každý monóm v ANF funkcie bude mať jednu premennú.

Pred aplikáciou prvého kola šifrovania je každá funkcia $f_{k,i}(x_1, x_2, x_3, x_4)$ rovná x_i, teda má algebraický stupeň 1. Ukážeme, že jedno kolo šifrovania túto vlastnosť zachová.

Žiadna z troch operácií použitých v RES nespôsobí vo funkcii v ANF $f_{k,i}(x_1, x_2, x_3, x_4)$ pre takúto konfiguráciu vznik monómu s viac ako jednou premennou. Pripočítanie kľúča vždy iba pridá alebo odoberie monóm jednej premennej, cyklická rotácia nepridáva žiadne monómy, dá sa na ňu pozrieť ako premenovanie funkcií $f_{k,i}$ na $f_{k,j}$, kde $j = i + c_{ls} \mod 4$.

Modulárne sčítanie má nasledovný efekt – funkcie pre 2 najmenej významné bity $f_{k,3}$ a $f_{k,4}$ sa nemenia vôbec, keďže $c_{ma}[3] = c_{ma}[4] = 0$. K funkcii $f_{k,2}$ sa pripočíta $c_{ma}[2]$, teda konštantný monóm hodnoty 1 alebo 0 a ak doteraz funkcia mala algebraický stupeň 1, ostane aj po tejto operácii nezmenený. K funkcii $f_{k,1}$ sa pripočíta člen $f_{k,2} \wedge c_{ma}[2]$, čo reprezentuje prenos (carry) zo sčítania pre druhý najvýznamnejší bit. Tento člen je konštanta 0, ak $c_{ma}[2] = 0$ a funkcia s algebraickým stupňom 1 ak $c_{ma}[2] = 1$. Keďže sčítanie dvoch funkcií s algebraickým stupňom 1 bude tiež funkcia s algebraickým stupňom 1, aj pre $f_{k,1}$ ostane jej stupeň zachovaný.

Ak majú všetky funkcie $f_{k,i}(x_1, x_2, x_3, x_4)$ vždy algebraický stupeň 1 a ak sa za aktívne bity A zoberú ľubovoľné 2 vstupné bity, derivácie $D_A f_{k,i}$ budú konštanty 0 pre $i \in \{1, \ldots, 4\}$, teda bude existovať integrálny rozlišovač $I = (A, \{1, 2, 3, 4\})$. □

2.2 Modré bity

V predchádzajúcich prácach [7, 10, 3] boli konštantné bity ľubovoľné, museli byť iba vzájomne rovnaké vo všetkých otvorených textoch v S_0. Konštantné bity sú pod kontrolou útočníka a môže byť za ne dosadená konkrétna hodnota. Pri týchto útokoch sa nevyužíva situácia, keď zvolenie konkrétnej hodnoty za konštantný bit spôsobí, že funkcia pre niektorý výstupný bit po derivácii vzhľadom na nejakú množinu indexov aktívnych bitov A bude 0, aj keď bez dosadenia hodnôt by nebola. Konštantné bity, za ktoré sa dosádza explicitná hodnota budú nazvané *modré bity*.

Príklad 2.3. Nech $f_{k,1}(x_1, x_2, x_3, x_4)$ je boolovská funkcia v ANF vykonávajúca šifrovanie pre najvýznamnejší bit RES(3, 14, 9). Pri použití Solvatore je možné nájsť integrálny rozlišovač maximálne pre 7 kôl. Nech $A = \{1, 2\}$, po zderivovaní vzhľadom na x_1 a x_2 má funkcia nasledovný predpis, kde $k_1, \ldots, k_4$ sú bity kľúča, ktoré útočník nepozná:

$$D_A f_{k,1}(x_1, x_2, x_3, x_4) = x_4 + x_3k_1 + x_3k_2 + x_3k_3 + x_3k_4 + x_4k_3.$$

Z funkcie vidno, že nech sa zvolí ktorýkoľvek ďalší vstupný bit ako aktívny a funkcia sa vzhľadom naň zderivuje, nevznikne z funkcie konštantná, teda nemôže byť prvý bit balansovaný. Ak by sa za aktívne zvolili oba, všetky bity by boli aktívne, čo neposkytuje žiadnu informáciu (pozri lemu 1.6).

Ak by však bola za x_3 a x_4, čo sú konštantné a teda ľubovoľné bity, dosadená hodnota 0, $D_A f_{k,1}(x_1, x_2, x_3, x_4) = 0$, teda prvý bit šifrovacej funkcie by bol balansovaný.

Príklad 2.4. Iný príklad, pri ktorom je po dosadení 1 za vstupné bity možné nájsť integrálny rozlišovač je funkcia pre druhý výstupný bit šifry RES(2, 7, 5). Solvatore nenašiel žiadny integrálny rozlišovač pre 4 kolá takejto šifry. Nech $A = \{3, 4\}$ a boolovská funkcia pre druhý bit šifry – $f_{k,2}(x_1, x_2, x_3, x_4)$ má po zderivovaní vzhľadom na x_3 a x_4 nasledovný predpis:

$$\begin{aligned} D_A f_{k,2}(x_1, x_2, x_3, x_4) =& k_1 + x_1k_1 + k_3 + x_1k_3 + k_1k_2 + x_2k_1k_2 + \\ & k_2k_3 + x_2k_2k_3 + x_1k_2k_4 + x_2k_2k_4. \end{aligned}$$

V ANF $D_A f_{k,2}$ možno vidieť, že všetky monómy sa dajú rozdeliť do dvojíc, v ktorých oba obsahujú rovnaké bity kľúča a líšia sa iba tým, že jeden obsahuje vstupný bit a druhý nie, respektíve obsahujú rôzny vstupný bit. Keď za oba vstupné bity bude dosadená hodnota 1, budú oba monómy rovnaké, každá dvojica sa sčíta na 0 a funkcia $D_A f_{k,2} = 0$

Modré a aktívne bity majú rozdielny efekt na funkciu v ANF a nie je triviálne určiť, ktorý vstupný bit je vhodnejší vybrať ako aktívny alebo modrý. Keď sa na nejakú

šifrovaciu funkciu $f_{k,i}$ pozrieme v ANF, určenie nejakej množiny bitov A ako aktívnych znamená, že pre $D_A f_{k.i}$ sa odstránia všetky monómy, v ktorých nie sú všetky aktívne bity. Určenie nejakej množiny ako modrých bitov, za ktoré sa dosadí 0 znamená, že sa odstránia všetky monómy obsahujúce aspoň jeden z týchto bitov. Určenie nejakého vstupného bitu ako modrého, za ktorý sa dosadí 1 môže spôsobiť to, že ak vo funkcii existuje iný monóm, ktorý sa líši iba v tom, že obsahuje o daný modrý vstupný bit navyše, sčítaním sa z ANF oba odstránia.

Príklad 2.5. Pre šifru RES(1, 3, 6) platí, že jej druhý výstupný bit je balansovaný pre množinu aktívnych bitov $A = \{2, 4\}$ a ak sa za modrý bit s hodnotou 0 zoberie x_1, a modrý bit s hodnotou 1 vstupný bit x_3 – $D_A f_{k,2}(0, x_2, 1, x_4) = 0$. Zároveň pre žiadnu inú kombináciu označenia vstupných bitov ako aktívnych a modrých toto neplatí.

Po derivácii vzhľadom na aktívne bity má funkcia $f_{k,2}$ nasledovný predpis:

$$D_A f_{k,2}(x_1, x_2, x_3, x_4) = k_1 + k_2 + k_3 + k_4 + x_1k_2 + x_3k_1 + x_3k_2 + \dots$$

Ak by sme zobrali x_1 ako aktívny bit navyše, vo funkcii $D_{A\cup\{1\}} f_{k,2}$ budú iba monómy obsahujúce bity kľúča, lebo kvôli leme 1.6 nemôže ANF obsahovať monóm so všetkými vstupnými bitmi. Podľa predpisu môžeme vidieť, že vo funkcii $D_{A\cup\{1\}} f_{k,2}$ bude aspoň monóm k_2. To však znamená, že z takejto funkcie sa nedá spraviť 0, nech sa za x_3 dosadí čokoľvek. Symetricky to platí aj ak by sme namiesto x_1 zobrali ako aktívny bit x_3 .

Z popisu môžeme vidieť aj to, že ak by sme do funkcie $D_{A\cup\{1\}} f_{k,2}$ dosadili za x_1 a x_3 nuly, vo funkcii by stále ostali monómy obsahujúce bity kľúča.

Ak by sme za obe x_1 a x_3 dosadili hodnoty 1, $D_A f_{k,2} = k_2 + k_1k_2 + k_2k_4$ a ak by sme za x_1 dosadili hodnotu 1 a x_3 hodnotu 0, $D_A f_{k,2} = k_1 + k_3 + k_4 + \dots$, teda pre žiadnu inú kombináciu hodnôt x_1 a x_2 nebude druhý bit balansovaný.

Ako je čiastočne ukázané v príklade 2.5, pre RES(1, 3, 6) existuje iba jedno rozdelenie vstupných bitov na aktívne a modré, pre ktoré je druhý bit balansovaný. Ak chceme nájsť všetky integrálne rozlišovače s použitím modrých bitov, nemôžeme použiť spôsob opísaný v časti 1.3.3 a musíme prejsť všetky možnosti označenia vstupných bitov ako aktívnych a modrých. Viac informácií o tomto hľadaní je v časti na strane 24.

Pri použití modrých bitov sa bude množina otvorených textov S_0 tvoriť pre množinu indexov aktívnych bitov A a množiny modrých bitov s hodnotami 0 a 1 – C_0 a C_1. Množina S_0 je tvorená rovnako ako bez použitia modrých bitov, ale musí pre ňu navyše platiť, že na miestach modrých bitov sú im priradené hodnoty – $\forall c_0 \in C_0, \forall u \in S_0 : u[c_0] = 0$ a $\forall c_1 \in C_1, \forall u \in S_0 : u[c_1] = 1$. Modré bity neoznačujeme ako konštantné. Rôznych množín S_0 spĺňajúcich takéto požiadavky je $2^{n-|A|-|C_0|-|C_1|}$.

Zavedieme definíciu integrálneho rozlišovača, ktorá zohľadňuje aj modré bity.

Definícia 2.6. *Integrálny rozlišovač s použitím modrých bitov* je štvorica $I^b = (A, C_0, C_1, B)$ pre konkrétnu šifru $E_k : \{0,1\}^n \to \{0,1\}^n$, kde $\emptyset \neq A \subset \{1, \ldots, n\}$ je množina indexov aktívnych bitov, $C_0, C_1 \subset \{1, \ldots, n\}$ sú množiny modrých bitov s hodnotami 0 resp. 1, a môžu byť prázdne. Množiny A, C_0 a C_1 sú po dvojiciach disjunktné. $\emptyset \neq B \subseteq \{1, \ldots, n\}$ je množina indexov balansovaných bitov. Po vytvorení ľubovoľnej množiny S_0 pre A, C_0 a C_1, jej zašifrovaní ľubovoľným kľúčom k bude pre každý balansovaný bit $b \in B$ platiť, že súčet všetkých prvkov rezu b množinou zašifrovaných textov je rovný 0:

$$\forall S_0, \forall k \in \{0,1\}^l, \forall b \in B : \sum_{x \in \{E_k(v) | v \in S_0\}} x[b] = 0.$$

Konštantné bity sú také vstupné bity, ktorých indexy nie sú ani v jednej z množín A, C_0 a C_1. Integrálne rozlišovače s použitím modrých bitov budeme nazývať *modré rozlišovače* a integrálne rozlišovače bez použitia modrých bitov *integrálne rozlišovače*. Z definície platí, že integrálne rozlišovače bez použitia modrých bitov sú špeciálnym prípadom modrých rozlišovačov, pre $C_0 = C_1 = \emptyset$.

Použitie modrých rozlišovačov môže byť z hľadiska kryptoanalýzy výhodné v dvoch smeroch:

1. V niektorých prípadoch existuje modrý rozlišovač pre väčší počet kôl šifry ako ktorýkoľvek integrálny rozlišovač. To nám umožňuje previesť integrálny útok na väčší počet kôl šifry.

2. Útok pomocou modrého rozlišovača, ktorý má oproti integrálnemu rozlišovaču menej aktívnych bitov znamená menšie množiny S_0 a a teda v niektorých prípadoch nižšiu zložitosť integrálneho útoku.

2.2.1 Modré rozlišovače pre šifru RES

V tejto časti sa zaoberáme hľadaním modrých rozlišovačov. Sústredíme sa na prípady, kde existuje modrý rozlišovač pre väčší počet kôl ako integrálny rozlišovač bez ich použitia.

Nakoľko pre šifru RES môžeme aj pre väčší počet kôl vygenerovať kompletné funkcie v ANF, dajú sa integrálne a modré rozlišovače hľadať priamočiaro. Pre každú funkciu $f_{k,i}(x)$ a každú kombináciu aktívnych a modrých bitov môžeme overiť, či sa v ANF po dosadení za modré bity vyskytuje monóm, ktorý by obsahoval všetky aktívne bity. Takéto priamočiare vyhľadávanie má výhodu v tom, že pre danú šifru s daným počtom kôl a množinami aktívnych a modrých bitov je možné s istotou povedať, či existuje integrálny alebo modrý rozlišovač. Nakoľko Solvatore pracuje so zjednodušením výpočtu, je možné, že existujúce integrálne alebo modré rozlišovače nebudú nájdené.

Najskôr sme pomocou takéhoto priamočiareho hľadania overili, či existujú integrálne rozlišovače pre viac kôl ako boli nájdené pomocou Solvatore. Test prebiehal tak, že sme vyskúšali nájsť integrálne rozlišovače najskôr pre rovnaký počet kôl ako Solvatore a následne pre jedno kolo naviac. Výsledky však iba potvrdili už zistené maximálne počty kôl uvedené v tabuľke 2.1.

Algoritmus 2.1 Algoritmus pre hľadanie modrých rozlišovačov pre šifru RES pre viac ako r' kôl. Viac informácií o tomto hľadaní je v časti na strane 24.

Vstup: c_{ls}, c_{ma}, r'

Výstup: Maximálny počet kôl, pre ktoré je nájdený modrý rozlišovač; všetky nájdené modré rozlišovače

```
1: r ← r' + 1
2: distinguishers ← [ ]
3: while True do
4:     found ← False
5:     f_{k,i} ← ANF pre i-ty bit RES(c_ls, c_ma, r) pre i ∈ {1, ..., n}
6:     for all A, C_0, C_1 tvoriace rozklad {1, ..., n}, ∅ ≠ A ⊂ {1, ..., n} do
7:         B ← [ ]
8:         for i ∈ {1, ..., n} do
9:             subs ← f_{k,i}, kde sú za bity s indexmi v C_0 resp. C_1 dosadené 0 resp. 1
10:            if subs neobsahuje monóm, so všetkými bitmi s indexmi v A then
11:                found ← True
12:                B.append(i)
13:        if B ≠ [ ] then
14:            distinguishers.append((r, A, C_0, C_1, B))
15:    if found = True then
16:        r ← r + 1
17:    else
18:        r ← r − 1
19:        break
return r, distinguishers
```

Následne sme pomocou algoritmu 2.1 hľadali modré rozlišovače pre každú konfiguráciu c_{ls} a c_{ma}, kde sme do algoritmu ako r' zadali maximálny počet kôl, pre ktoré existuje integrálny rozlišovač pre konfiguráciu s konštantami c_{ls} a c_{ma} nájdený pomocou Solvatore. Nehľadali sme modré rozlišovače pre konfigurácie šifry, pre ktoré existuje integrálny rozlišovač pre ľubovoľný počet kôl. Najskôr sme skúsili použiť iba modré bity s hodnotou 0, teda $C_1 = \emptyset$ pre celý výpočet, potom iba s hodnotou 1, teda $C_0 = \emptyset$ a nakoniec s ľubovoľnou kombináciou 0 a 1, aby sme vedeli posúdiť, aký majú jednotlivé typy modrých bitov vplyv.

Pri testovaní s modrými bitmi s hodnotou výlučne 0, v 8 konfiguráciách z 36 testovaných boli nájdené modré rozlišovače pre väčší počet kôl ako pomocou Solvatore, vo väčšine prípadov až o 2 kolá. Tieto konfigurácie aj s porovnaním s maximálnym počtom kôl pre integrálne rozlišovače nájdené pomocou Solvatore sú v tabuľke 2.2.

Konfigurácia šifry		Počet kôl pre najlepší rozlišovač	
c_{ls}	c_{ma}	Solvatore	Použitie modrých bitov
1	2	7	8
1	10	7	8
2	1	3	5
2	2	5	7
2	9	3	5
2	10	5	7
3	6	7	9
3	14	7	9

Tabuľka 2.2: Konfigurácie RES, kde nastalo zlepšenie použitím výlučne modrých bitov s hodnotou 0

Podobne aj pri testovaní s modrými bitmi s hodnotou výlučne 1, v 8 z 36 konfigurácií sme našli modrý rozlišovač pre väčší počet kôl šifry ako bez použitia modrých bitov, väčšinou väčší o 2 kolá. Všetkých 8 konfigurácií je rozdielnych od tých, ktoré boli nájdené pomocou modrých bitov s hodnotou výlučne 0. Výsledky aj s porovnaním so Solvatore sú v tabuľke 2.3.

Konfigurácia šifry		Počet kôl pre najlepší rozlišovač	
c_{ls}	c_{ma}	Solvatore	Použitie modrých bitov
1	6	7	8
1	14	7	8
2	6	5	7
2	7	3	5
2	14	5	7
2	15	3	5
3	2	7	9
3	10	7	9

Tabuľka 2.3: Konfigurácie RES, kde nastalo zlepšenie použitím výlučne modrých bitov s hodnotou 1

Nakoniec pri testovaní s modrými bitmi s hodnotami 0 aj 1 sme našli modrý rozlišovač pre väčší počet kôl ako bez použitia modrých bitov pre väčšinu konfigurácií šifier – pre 28 z 36 testovaných konfigurácií. Pre konfigurácie šifier s konštantami $c_{ls} = 2$ a

$c_{ma} \in \{1, 7, 9, 15\}$ boli dokonca nájdené modré rozlišovače pre 13 kôl, kým pomocou Solvatore boli nájdené iba pre 3 kolá. Zároveň sme však zistili, že existujú aj integrálne rozlišovače pre 9,10 a 11 kôl šifier s týmito konfiguráciami, aj keď neexistujú pre 4 až 8 kôl. Aj pri týchto sa však ukázalo, že sa dajú nájsť modré rozlišovače pre viac kôl ako integrálne rozlišovače.

c_{ls} \ c_{ma}	0	1	2	3	4	5	6	7
0	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	4	7+1	4+2	∞	4+2	7+1	4
2	∞	3+10	5+2	3+3	∞	3+3	5+2	3+10
3	∞	4	7+4	4+1	∞	4+1	7+4	4
c_{ls} \ c_{ma}	8	9	10	11	12	13	14	15
0	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	4	7+1	4+2	∞	4+2	7+1	4
2	∞	3+10	5+2	3+3	∞	3+3	5+2	3+10
3	∞	4	7+4	4+1	∞	4+1	7+4	4

Tabuľka 2.4: Maximálne počty kôl pre šifru RES nájdené pomocou Solvatore a ich zlepšenie o daný počet kôl použitím modrých bitov s hodnotami 0 a 1. V každej bunke zodpovedajúcej c_{ls} a c_{ma} je prvé číslo pre koľko najviac kôl r šifry RES(c_{ls}, c_{ma}, r) je možné nájsť integrálny rozlišovač pomocou Solvatore a druhé číslo pre aký počet kôl navyše existuje modrý rozlišovač. Symbolom ∞ označujeme, že existuje integrálny rozlišovač pre ľubovoľný počet kôl.

Tvorba funkcií v ANF

Pre vytvorenie funkcií v ANF, ktoré sa následne budú dať použiť na hľadanie integrálnych rozlišovačov je potrebný nástroj, v ktorom sa bude s nimi dať pracovať. Je potrebné, aby sa s nimi dalo pracovať v jazyku Python, v ktorom je implementovaný aj Solvatore, kvôli jednoduchšej implementácii hľadania modrých rozlišovačov do tohto rámca.

Sympy [16] je knižnica v jazyku Python na prácu so symbolickou matematikou spĺňajúca predchádzajúce požiadavky. Pomocou tejto knižnice sa dajú jednoducho vygenerovať boolovské funkcie $f_{k,i}(x)$ opisujúce danú šifru, napríklad tak, že sa udržuje pole boolovských výrazov, ktoré predstavujú aktuálny stav funkcií $f_{k,i}(x)$ pre stav šifry a postupne sa na ne aplikujú všetky operácie šifry. Modulárne pripočítanie sme implementovali ako konkrétny S-box z ktorého je následne vypočítaná ANF a po jeho aplikácii na stav šifry sa vykoná prevedenie všetkých funkcií do ANF postupným

„roznásobením“. Cyklický posun sme implementovali ako cyklický posun poľa, v ktorom je uložený stav šifry.

Taktiež bolo potrebné implementovať funkciu AND, ktorá zachováva tvar ANF – pre 2 argumenty funkcií v ANF roznásobí každý monóm s každým. Pre zachovanie ANF je potrebné vyhnúť sa pripočítavaniu konštantnej hodnoty 1 k funkcii, nakoľko to pre výraz v Sympy spôsobí, že celá funkcia bude zjednodušená tak, že bude negovaná – použili sme novú premennú $const1$, do ktorej je na konci výpočtu dosadená hodnota 1.

Iná možnosť by bola použiť súbor nástrojov Sage [14], ktorý má vstavaných viac funkcií, napríklad konverziu funkcie do ANF. Narozdiel od Sympy, jeho použitie ako knižnice zahŕňa väčšie zmeny v projekte, čomu sme sa chceli vyhnúť pri integrácii do Solvatore.

Na koniec overujeme, že výsledná funkcia je ekvivalentná samotnému šifrovaniu. Vďaka tomu, že RES má len 16 možných kľúčov a 16 otvorených textov, môžeme vyskúšať všetky možnosti otvoreného textu a kľúča zašifrovať dosadením do vygenerovaných funkcií aj pomocou referenčnej implementácie a overiť, že sa šifrové texty vytvorené oboma spôsobmi zhodujú.

Hľadanie modrých rozlišovačov

Ako možno vidieť v algoritme 2.1, hľadáme iba modré rozlišovače bez konštantných bitov, pretože testujeme iba modré rozlišovače pre množiny A, C_0 a C_1 také, že tvoria rozklad množiny vstupných bitov a $\emptyset \neq A \subset \{1, \ldots, n\}$. Podmienky pre A vyplývajú z definície modrého rozlišovača.

Dôvod, prečo sa testujú iba množiny A, C_0 a C_1 bez konštantných bitov je taký, že ak chceme zistiť iba to, či existuje pre šifru s daným počtom kôl modrý rozlišovač, stačí otestovať všetky kombinácie označení vstupných bitov za aktívne a modré, pretože podľa lemy 2.7, ak existuje modrý rozlišovač s konštantnými bitmi, existuje aj taký, kde tieto konštantné bity sú označené ľubovoľne ako modré alebo aktívne (ak sa nestane, že budú aktívne všetky).

Trojicu (A, C_0, C_1) pre nejaké po dvojiciach disjunktné množiny $\emptyset \neq A \subset \{1, \ldots, n\}$ a $C_0, C_1 \subset \{1, \ldots, n\}$ označíme T, prípadne T_1, T_2 alebo T'. Mohutnosť T je súčet mohutností A, C_0 a C_1 – $|T| = |A \cup C_0 \cup C_1|$. Nech $T_1 = (A, C_0, C_1)$ a $T_2 = (A', C_0', C_1')$. Označenie $T_1 \geq T_2$ znamená, že $A \supseteq A'$, $C_0 \supseteq C_0'$ a $C_1 \supseteq C_1'$. Prienik T_1 a T_2 je definovaný ako $T_1 \cap T_2 = (A \cap A', C_0 \cap C_0', C_1 \cap C_1')$.

Lema 2.7. Ak pre ľubovoľnú šifru $E_k : \{0,1\}^n \to \{0,1\}^n$ existuje $I^b = (A, C_0, C_1, B)$, kde $T_1 = (A, C_0, C_1)$, potom pre každú trojicu $T_2 = (A', C_0', C_1'), T_2 \geq T_1$ existuje B' také, že $B' \supseteq B$ a $I^{b'} = (A', C_0', C_1', B')$.

Dôkaz. Nech šifrovacej funkcii E_k pre jednotlivé bity zodpovedajú boolovské funkcie $f_{k,i} : \{0,1\}^n \to \{0,1\}$ pre $i \in \{1, \ldots, n\}$.

Ak existuje $I^b = (A, C_0, C_1, B)$, tak pre všetky $A', A \subseteq A' \subset \{1, \ldots, n\}$, $A' \cap C_0 = \emptyset, A' \cap C_1 = \emptyset$ a nejaké $B_1 \supseteq B$ existuje aj $I_1^b = (A', C_0, C_1, B_1)$, pretože ak pre všetky funkcie $f_{k,j}$, kde $j \in B$ platí, že $D_A f_{k,j} = 0$, platí aj $D_{A'} f_{k,j} = 0$, pretože to znamená zderivovanie funkcie vzhľadom na ďalšie vstupné bity. Ak bola derivácia funkcie rovná 0 pred dodatočnými deriváciami, bude aj po nich. Zároveň pre niektorú funkciu $f_{k,j}$, kde $j \notin B$ môže byť $D_{A'} f_{k,j} = 0$, aj keď $D_A f_{k,j} \neq 0$ a teda by platilo $j \in B_1$.

Ak existuje $I_1^b = (A', C_0, C_1, B_1)$, tak pre ľubovoľné $C_0' \supseteq C_0$, $C_1' \supseteq C_1$, $C_0' \cap C_1' = \emptyset$ existuje $B_2 \supseteq B_1$ také, že $I_2^b = (A', C_0', C_1', B_2)$. Množiny A', C_0, C_1 a množina indexov konštantných bitov tvoria rozklad $\{1, \ldots, n\}$, teda $C_0' - C_0$ a $C_1' - C_0$ sú podmnožiny množiny konštantných bitov pre I_1^b. To znamená, že ak sú neprázdne, namiesto niektorého konštantného bitu pre I_1^b je v I_2^b modrý bit. Pre všetky funkcie $f_{k,j}$, kde $j \in B$ platí, že $D_{A'} f_{k,j} = 0$, nezávisle od toho, akú hodnotu majú konštantné bity, takže rovnosť platí aj ak sú za ne dosadené konkrétne hodnoty zodpovedajúce modrým bitom. Tiež ak pre niektorú funkciu $f_{k,j}$, kde $j \notin B$ po dosadení konkrétnych hodnôt za modré bity platí $D_{A'} f_{k,j} = 0$, potom $j \in B_2$.

Ak platí, že existuje $I^b = (A, C_0, C_1, B)$, potom pre ľubovoľnú A', $A \subseteq A' \subset \{1, \ldots, n\}$ existuje $B_1 \supseteq B$ taký, že $I_1^b = (A', C_0, C_1, B_1)$. Potom, pre ľubovoľné množiny $C_0' \supseteq C_0$ a $C_1' \supseteq C_1$, kde C_0', C_1' a A' sú po dvojiciach nezávislé, existuje množina $B_2 \supset B_1$ taká, že $I^{b'} = (A', C_0', C_1', B_2)$, teda pre ľubovoľnú trojicu $T_2 = (A', C_0' C_1'), T_2 \geq T_1$ platí, že existuje $B' = B_2$, také, že $I^{b'} = (A', C_0', C_1', B')$ □

2.2.2 Hľadanie modrých rozlišovačov s konštantnými bitmi

Podobne ako v Solvatore sa dajú nájdené integrálne rozlišovače s jedným konštantným bitom použiť na jednoduchšie hľadanie integrálnych rozlišovačov s viacerými konštantnými bitmi (pozri časť 1.3.3) sa budú dať aj modré rozlišovače nájdené spôsobom popísaným v časti 2.2.1 použiť na hľadanie ďalších s viacerými konštantnými bitmi.

Nech máme dva modré rozlišovače I_1^b a I_2^b, ktoré sú zhodné, až na to, že niektoré aktívne bity v I_1^b sú v I_2^b konštantné. Potom I_2^b je výhodnejší z hľadiska kryptoanalýzy v tom, že pri integrálnom útoku s jeho použitím je vytvorená množina S_0 menšia, teda útok bude časovo menej náročný.

Ak by boli I_1^b a I_2^b zhodné, až na to, že niektoré modré bity v I_1^b by boli v I_2^b konštantné, potom by bol I_2^b výhodnejší z hľadiska kryptoanalýzy ako je uvedená v kapitole 4 v tom, že rôznych množín otvorených textov S_0 môže byť vytvorených viac. Pre modré rozlišovače, kde množiny A, C_0 a C_1 tvoria rozklad množiny vstupných bitov môže byť vytvorená iba jedna množina S_0, čo nie je dostačujúce.

Nová stratégia hľadania modrých rozlišovačov (ďalej ju označujeme stratégia OS2) je rozšírením stratégie OS v časti 1.3.3. Pomocou nej sa dá nájsť niektorý modrý roz-

lišovač s najväčším množstvom konštantných bitov c – I^b_{max}. Označenie $DP(T)$, kde $T = (A, C_0, C_1)$, znamená najväčšiu množinu balansovaných bitov takú, že pre konkrétnu šifru s určeným počtom kôl existuje modrý rozlišovač $I^b = (A, C_0, C_1, \mathrm{DP}(T))$, prípadne ak žiadny taký neexistuje tak je to prázdna množina. Formálne:

$$ALL\_B(T) = \{B \mid T = (A, C_0, C_1), \exists I^b = (A, C_0, C_1, B)\} \cup \{\emptyset\}$$
$$\mathrm{DP}(T) = B, B \in ALL\_B(T), \forall B' \in ALL\_B(T) : B' \subseteq B$$

Vďaka leme 2.7 môžeme najskôr skontrolovať existenciu modrých rozlišovačov iba pre trojice $T = (A, C_0, C_1)$, kde A, C_0 a C_1, ktoré tvoria rozklad $\{1, \ldots, n\}$, pretože ak existuje modrý rozlišovač s nenulovým počtom konštantných bitov, existuje aj modrý rozlišovač, kde sú tieto konštantné bity priradené ľubovoľne k aktívnym alebo modrým bitom, s výnimkou keď sú všetky bity aktívne (pozri lema 1.6). Všetky také trojice, pre ktoré existuje modrý rozlišovač budú v množine G_0.

$$G_0 = \{T \mid T = (A, C_0, C_1), A \cup C_0 \cup C_1 = \{1, \ldots, n\}, \mathrm{DP}(T) \neq \emptyset\}.$$

Pre každú dvojicu rôznych $T_1, T_2 \in G_0$ takú, že $DP(T_1) \cap DP(T_2) \neq \emptyset$ následne overíme, či existuje modrý rozlišovač pre trojicu $T = T_1 \cap T_2$, a všetky také, pre ktoré existuje rozlišovač zaradíme do G_2. Takto iteratívne postupujeme pre $G_i, i \in \{1, \ldots, n-1\}$, prípadne kým nie je niektoré G_i prázdne. Modrý rozlišovač s najväčším počtom konštantných bitov potom bude v poslednej neprázdnej množine G_i.

$$G_i = \{T_1 \cap T_2 \mid T_1, T_2 \in G_{i-1}, T_1 \neq T_2, DP(T_1) \cap DP(T_2) \neq \emptyset, DP(T_1 \cap T_2) \neq \emptyset\}$$

Pomocou takéhoto výpočtu bude určite nájdený modrý rozlišovač I^b_{max}, čo dokážeme v leme 2.8. Modrý rozlišovač s najväčším počtom bitov však nemusí byť ten, s najmenším počtom aktívnych bitov.

Lema 2.8. Stratégia OS2 vždy nájde pre danú šifru niektorý modrý rozlišovač $I^b_{max} = (A, C_0, C_1, B)$ s najväčším počtom konštantných bitov c. Trojica $T_{max} = (A, C_0, C_1)$ bude v množine G_c a nasledujúca množina $G_{c+1} = \emptyset$.

Dôkaz. Množina G_0 obsahuje všetky trojice $T = (A', C'_0, C'_1)$ také, že A', C'_0 a C'_1 tvoria rozklad $\{1, \ldots, n\}$ a $T \geq T_{max}$, kvôli platnosti lemy 2.7. Všetky modré rozlišovače pre takéto trojice nebudú obsahovať žiadne konštantné bity.

Pre každé G_i, kde $i \leq c$ platí, že obsahuje všetky trojice T také, že $T \geq T_{max}$ a $|T| = n - i$.

$$\{T \mid T \geq T_{max}, |T| = n - i\} \subseteq G_i$$

Pre G_0 výrok zjavne platí. Ukážeme, že ak výrok platí pre G_{i-1}, potom platí aj pre G_i. Pre všetky $T \geq T_{max}$ platí, že $\mathrm{DP}(T) \supseteq B$, keďže podľa lemy 2.7, pre T existuje modrý rozlišovač s $B' \supseteq B$. Tým pádom, pre dve ľubovoľné $T_1 \geq T_{max}$ a $T_2 \geq T_{max}$, $\mathrm{DP}(T_1) \cap \mathrm{DP}(T_2) \neq \emptyset$. Keďže v G_{i-1} sú všetky $T, T \geq T_{max}$ také, že $|T| = n - i + 1$, medzi ich prienikmi sú aj všetky $T', T' \geq T_{max}$ také, že $|T| = n - i$.

Platí $T_{max} \in G_c$, pretože $|T_{max}| = n - c$ a $T_{max} \geq T_{max}$.

Ukážeme pomocou dôkazu sporom, že množina $G_{c+1} = \emptyset$. Pri tvorení množín G_i vždy platí, že sa robí prienik dvoch rôznych trojíc $T_1, T_2 \in G_{i-1}$, teda v ich prieniku $T_1 \cap T_2$ vznikne aspoň jeden konštantný bit – $|T_1 \cap T_2| < |T_1| \vee |T_1 \cap T_2| < |T_2|$. Potom $\forall T, T \in G_1 : |T| \leq n - 1$, a všeobecnejšie $\forall T, T \in G_i : |T| \leq n - i$. Ak by potom množina G_{c+1} bola neprázdna, znamenalo by to, že existuje $T \in G_{c+1}$ s viac ako $c + 1$ konštantnými bitmi, čo je spor s výberom T_{max}.

□

Kapitola 3

Modré rozlišovače v Solvatore

V tejto kapitole opisujeme jednotlivé časti Solvatore a navrhneme dva možné spôsoby implementácie hľadania modrých rozlišovačov do tohto rámca. Tieto spôsoby otestujeme na RES a šifre Speck [2]. Navrhneme nové spôsoby extrakcie MCV v Solvatore, aj pre vytvorené funkcie šifrovania v ANF.

V sekcii 2.2 sme ukázali, že v niektorých prípadoch existujú modré rozlišovače pre väčšie množstvo kôl šifry ako integrálne rozlišovače bez modrých bitov. Ich hľadanie však zahŕňalo počítanie ANF pre všetky kolá šifrovacích funkcií, čo je u väčších šifier výpočtovo neuskutočniteľné. Preto by bolo praktické, ak by sa takéto hľadanie dalo robiť automatizovane pomocou Solvatore.

3.1 Vnútorná štruktúra Solvatore

V tejto časti popíšeme jednotlivé časti Solvatore [15] a ako spolupracujú, pretože ich neskôr budeme upravovať.

Trieda `CipherDescription` definuje objekty, ktoré vytvárajú a obsahujú popis šifry. Objekt tejto triedy je inicializovaný pomocou argumentu, ktorý opisuje veľkosť stavu šifry (pri blokových šifrách je to dĺžka bloku). Trieda obsahuje metódy, ktorými sa k opisu pridávajú operácie použité v šifre, napríklad `apply_xor`, ktorou sa dá pridať operácia XOR medzi zvolenými dvoma bitmi stavu, pričom výsledok je zapísaný ďalšieho zvoleného bitu stavu. V triede sú implementované aj zložitejšie operácie, napríklad aplikácia operácie MixColumns pri šifre AES.

Premenná `transition` v tejto triede je pole operácií v poradí, v akom majú byť pri šifrovaní aplikované. Tieto operácie môžu byť 5 typov – operácia AND, operácia XOR, permutácia, priradenie hodnoty bitu inému a aplikácia S-boxu. Pri každej operácii sú uvedené bity stavu alebo dočasné bity, na ktorých má byť prevedená (zdrojové bity) a kam má byť uložený výsledok (cieľový bit). Bity stavu majú meno začínajúce na `s` a sú vytvárané pri inicializácii. Dočasné bity majú meno začínajúce na `t` a sú vyrobené po

vykonaní operácie, kde sú prvý krát uvedené ako cieľový bit. Implementácia kontroluje, či ako zdrojový bit nie je uvedený doteraz nepoužitý dočasný bit.

V priečinku `ciphers` sa nachádzajú moduly pre rôzne šifry, ktoré obsahujú kód na vytvorenie objektu triedy `CipherDescription` pre danú šifru.

Trieda `Solvatore` v module `solvatore.py` obsahuje funkcie potrebné pre načítanie objektu triedy `CipherDescription`, prekladu daných operácií do transformácií MCV a logického výrazu v KNF ako je opísané v častiach 1.3.1 a 1.3.2, a jeho následné vyriešenie pomocou SAT solvera. Logický výraz je vytváraný nasledovne – najskôr sa všetkým vstupným bitom priradia premenné. Ak je na niektoré vstupné alebo dočasné bity aplikovaná operácia, napríklad XOR, vytvoria sa pre ne nové premenné a pomocou klauzúl KNF sa medzi nimi a pôvodnými premennými vytvoria vzťahy, ako je opísané v časti 1.3.2. Ak je cieľový bit dočasný, ktorý zatiaľ nebol použitý, vytvorí sa preň nová premenná. V každom bode tvorby KNF sa udržiava pre každý vstupný a dočasný bit ktorá premenná mu bola priradená naposledy.

Moduly, ktorých názov začína `analysis_` sú spustiteľné a vykonajú analýzu šifry. V každom module sa najskôr vykoná vytvorenie objektu `CipherDescription` pre šifru s potrebnými parametrami. Ten je načítaný do objektu triedy `Solvatore` a postupným použitím funkcie `is_bit_balanced` sú hľadané rozlišovače. Modul `analysis_optimal` vykonáva hľadanie integrálnych rozlišovačov popísané v časti 1.3.3.

3.1.1 Odlišnosti modelovania šifier v Solvatore

Pri opise konkrétnych šifier v module `ciphers` sú vynechané niektoré operácie, ako pripočítania kľúča alebo podkľúča a iné sú opísané iba čiastočne – modulárne pripočítanie konštanty neberie do úvahy aká konštanta to je. Napríklad, ak bola pri popise šifry RES použitá táto operácia, všetky rozlišovače záviseli iba na konštante cyklického posunu a boli nájdené vo viacerých prípadoch iba integrálne rozlišovače pre výrazne nižšie počty kôl, ako keď sme tú istú operáciu modelovali pomocou S-boxu.

Bity kľúča alebo podkľúča sa nepripočítavajú, nakoľko pri transformácii vektorov minimálnej voľby, operácia XOR alebo AND s konštantou nemá žiadny vplyv [7]. Iný pohľad môže byť cez boolovské šifrovacie funkcie. Rozdeľme jedno kolo šifrovacej funkcie do 3 častí – operácie vykonávané pred pripočítaním kľúča – $f_{k,i,1}$ kde i je index bitu stavu šifry, pripočítanie kľúča – $f_{k,i,2}$, operácie vykonávané po pripočítaní kľúča – $f_{k,i,3}$ a nech sú všetky v ANF.

Lema 3.1. Nech $\forall i \in \{1, \ldots, n\}$, $f_{k,i,1}$ nie sú konštantné funkcie s hodnotou 0 alebo 1. Potom pri šifrovacích funkciách to, či sa vykoná pripočítavanie kľúča operáciou XOR alebo nie nemá žiadny vplyv na MCV funkcií.

Dôkaz. Pre funkciu $f_{k,i,3}$ po dosadení pripočítania kľúča pre prvý bit operáciou XOR $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$ bude platiť, že každý monóm, v ktorom pôvodne

bolo x_1 teraz obsahuje $(x_1 + k_1)$. Napríklad z $x_1x_2x_3$ sa stane $(x_1 + k_1)x_2x_3$. Po roznásobení bude v zloženej funkcii okrem pôvodného $x_1x_2x_3$ aj $k_1x_2x_3$. Všeobecnejšie, vo funkcii $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$ v ANF pribudne oproti funkcii $f_{k,i,3}(x_1, x_2, \ldots, x_n)$ pre každý monóm tvaru $x_1x_{i_1}x_{i_2} \ldots x_{i_m}$ kde $i_1, \ldots, i_m \subseteq \{2, \ldots, n\}$ nový monóm $k_1x_{i_1}x_{i_2} \ldots x_{i_m}$.

Pre takýto prípad však vždy platí, že ak funkcia $f_{k,i,3}(x_1, \ldots x_n)$ figurovala v znásobení pre nejaký vektor minimálnej voľby, figuruje v ňom aj $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$, pretože ak sú obe v ANF, monómy $f_{k,i,3}(x_1, \ldots x_n)$ sú podmnožinou monómov $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$.

Ak v nejakom znásobení pre minimálny vektor voľby figuruje $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$, bude v ňom figurovať aj $f_{k,i,3}(x_1, \ldots x_n)$, pretože pre každý monóm, ktorý vznikol po pripočítaní kľúča platí, že existuje rovnaký monóm, kde je namiesto bitu kľúča vstupný bit. Ak by teda po znásobení $f_{k,i,3}(f_{k,1,2}(x_1, \ldots, x_n), x_2, \ldots, x_n)$ s inou funkciou bol vo výsledku v ANF monóm obsahujúci všetky aktívne bity, bol by aj v znásobení $f_{k,i,3}(x_1, \ldots, x_n)$ s danou funkciou.

Ak pre každé pripočítanie kľúča platí, že nemenia vektory minimálnej voľby, tak ani pripočítanie všetkých bitov kľúča ich nemení. Ak by boli nejaké operácie pred pripočítaním kľúča, teda po dosadení všetkých funkcií $f_{k,i,2}$ za x_i by boli dosadené $f_{k,i,1}$ za všetky x_i, a žiadna $f_{k,i,1}$ nie je konštantná funkcia 0 alebo 1, stále bude platiť, že pre každý monóm, ktorý vznikol pripočítaním kľúča existuje monóm s nadmnožinou vstupných bitov, ktorý by sa vo funkcii nachádzal aj keby nebol kľúč pripočítaný. □

Dôsledok 3.2. Pri modelovaní šifry v Solvatore, kde $\forall i \in \{1, \ldots, n\}, f_{k,i,1}$ nie sú konštantné funkcie nie je potrebné modelovať pripočítavanie kľúčov.

Príklad 3.3. Nech je pripočítavanie kľúča operáciou XOR prvá operácia kola a pre niektorý bit $f_{k,i,3}$ je predpis nasledovný:

$$f_{k,i,3}(x_1, x_2, x_3, x_4) = x_1x_2x_3 + x_1x_2 + x_3x_4 + x_2x_3x_4.$$

Po pripočítaní kľúča pre druhý bit je funkcia nasledovná:

$$\begin{aligned} f_{k,i,3}(x_1, f_{k,2,2}(x_1, x_2, x_3, x_4), x_2, x_3, x_4) =& x_1(x_2 + k_2)x_3 + x_1(x_2 + k_2) + \\ & x_3x_4 + (x_2 + k_2)x_3x_4. \end{aligned}$$

Po roznásobení:

$$\begin{aligned} f_{k,i,3}(x_1, f_{k,2,2}(x_1, x_2, x_3, x_4), x_2, x_3, x_4) =& x_1x_2x_3 + x_1k_2x_3 + x_1x_2 + x_1k_2 + x_3x_4 + \\ & x_2x_3x_4 + k_2x_3x_4 \end{aligned}$$

Môžeme vidieť, že každý monóm ktorý vznikol pripočítaním kľúča (zvýraznené červenou) má vedľa seba monóm so vstupnými bitmi, ktoré sú nadmnožinou jeho vstupných bitov.

Pri dosádzaní modrých bitov musíme vo funkciách počítať s bitmi kľúča. Po dosadení konštanty 0 alebo 1 za vstupný bit už neplatí, že musí pre každý monóm, ktorý vznikol vďaka pripočítaniu kľúča existovať taký, ktorý má namiesto daného bitu kľúča vstupný bit.

Príklad 3.4. Použijeme šifrovaciu funkciu z príkladu 3.3:

$$f_{k,i,3}(x_1, f_{k,2,2}(x_1, x_2, x_3, x_4), x_2, x_3, x_4) = x_1x_2x_3 + x_1k_2x_3 + x_1x_2 + x_1k_2 + x_3x_4 + x_2x_3x_4 + k_2x_3x_4$$

Ak by sme neuvažovali s pripočítaním kľúča a za modrý bit rovný 0 by sme zvolili x_2, funkcia by bola nasledovná:

$$f_{k,i}(x_1, x_3, x_4) = x_3x_4.$$

Keď nezanedbáme pripočítanie kľúča, po zvolení rovnakého modrého bytu vyzerá funkcia nasledovne:

$$f_{k,i}(x_1, x_3, x_4) = x_1k_2x_3 + x_1k_2 + x_3x_4 + k_2x_3x_4.$$

Môžeme vidieť, že funkcia so zanedbaným pripočítaním kľúča po derivácii vzhľadom na $A = \{1\}$ s množinou $C_0 = \{2\}$ bude rovná 0, teda i-ty výstupný bit by bol balansovaný. Pri funkcii s pripočítavaním kľúča to neplatí.

Nakoľko je potrebné, aby sa pri šifrovaní pripočítavali aj bity kľúča, ku ktorým nie sú vstupné bity, potrebujeme pridať počas opisu šifry dočasný bit reprezentujúci bit kľúča. V rámci MCV má každý bit kľúča vždy hodnotu 0, pretože nikdy neobsahuje aktívny vstupný bit.

Metóda `apply_xor` za podmienky, že oba jej vstupné bity sú rovnaké, pre výstupný bit nastaví hodnotu 0, keďže súčet XOR premennej samej so sebou bude mať za výsledok 0, a tá neobsahuje žiadne aktívne bity. Ak by cieľom tejto operácie bol nový dočasný bit, bol by zároveň vytvorený a bola by mu priradená hodnota 0, čo je to, čo potrebujeme pre bit kľúča.

Pre zjednodušenie a sprehľadnenie zápisu šifry sme pridali metódu `apply_add_temp` s takouto funkcionalitou do triedy `CipherDescription`, aj na ňu naväzujúce funkcie v triedach `SolvatoreBlue1` a `SolvatoreBlue2` spomenutých v častiach 3.2 a 3.3.

Implementovali sme moduly `RES_with_key` a `speck_with_key`, v ktorých sa tvoria `CipherDescription` pre šifry RES a Speck [2], pri ktorých sa nezanedbáva pripočítanie kľúča. Vznikli malými úpravami modulov `RES` a `speck`.

3.2 Integrácia pomocou nových stavov bitu v MCV

Jeden spôsob, ako integrovať do rámca hľadanie modrých rozlišovačov je implementovaný v novej triede `SolvatoreBlue1`, ktorá upravuje triedu `Solvatore`. Myšlienka je v

tom, že namiesto pôvodnej tvorby logického výrazu opisujúceho transformácie MCV, kde premenná môže byť iba v stave 0 alebo 1, podľa toho akú má daný vstupný alebo dočasný bit hodnotu v nejakom vektore minimálnej voľby, zavedieme ďalšie 2 stavy – b_0 a b_1, ktoré nastanú vtedy, keď daný bit je modrý bit 0 alebo modrý bit 1.

Transformácie vektorov minimálnej voľby dĺžky 2 pre operácie XOR, AND a priradenie hodnoty je možné nájsť v tabuľke 3.1. Neimplementovali sme operáciu použitia S-boxu.

AND	0	1	b_0	b_1
0	(0,0)	(0,1)	(0,b_0)	(0,0)
1	(1,0), (0,1)	(0,1)	(1,b_0)	(1,0), (0,1)
b_0	(b_0,b_0)	N/A	(b_0,b_0)	(b_0,b_0)
b_1	(b_1,0)	(b_1,1)	(b_1,b_0)	(b_1,b_1)

XOR	0	1	b_0	b_1
0	(0,0)	(0,1)	(0,0)	(0,0)
1	(1,0), (0,1)	(1,1)	(1,0), (0,1)	(1,0), (0,1)
b_0	(b_0,0)	(b_0,1)	(b_0,b_0)	(b_0,b_1)
b_1	(b_1,0)	(b_1,1)	(b_1,b_1)	(b_1,b_0)

Bit-copy	0	1	b_0	b_1
0	(0,0)	N/A	(0,0)	(0,0)
1	(1,0), (0,1)	N/A	(1,0), (0,1)	(1,0), (0,1)
b_0	(b_0,b_0)	N/A	(b_0,b_0)	(b_0,b_0)
b_1	(b_1,b_1)	N/A	(b_1,b_1)	(b_1,b_1)

Tabuľka 3.1: Transformácie vektorov minimálnej voľby dĺžky 2 pomocou operácií XOR a AND a Bit-copy. Vektor (x,y) je reprezentovaný bunkou v x-tom riadku a y-tom stĺpci. Pre každú operáciu platí, že výsledok je zapísaný do bitu y.

Na základe tabuľky 3.1 môžeme vidieť, že pri hľadaní integrálnych rozlišovačov bez použitia modrých bitov pripočítanie podkľúča pomocou operácie XOR skutočne nezmenilo hodnotu MCV. Bit kľúča má vždy hodnotu 0, nakoľko nemôže obsahovať žiadne aktívne bity a vždy je pripočítaný bit kľúča k bitu stavu, nie naopak. Keď sa pozrieme na transformácie vektorov (0,0) a (0,1), môžeme vidieť že operácia XOR ich nijako nemení.

Naopak ak možné stavy zahŕňajú aj b_0 a b_1, pripočítanie kľúča musíme simulovať, pretože napríklad pri situácii, keď sa operáciou XOR pripočíta k bitu so stavom b_0 bit kľúča, (podľa tabuľky 3.1 operácia XOR na vektore minimálnej voľby (0, b_0)) tento sa pretransformuje na vektor $(0, 0)$. Ak by táto operácia bola vynechaná, množina MCV po jednom kole by vyzerala rozdielne.

Ak šifra pripočítava operáciou XOR ku každému bitu otvoreného textu kľúč alebo podkľúč, všetky vstupné bity, ktoré boli na začiatku v stave b_0 alebo b_1 budú po tejto operácii v stave 0. Takýto výpočet je potom identický s takým, ktorý mal všetky modré bity označené ako konštantné. Ak by šifra používala „key whitening“, táto metóda hľadania modrých rozlišovačov nebude mať žiadny efekt oproti hľadaniu integrálnych rozlišovačov.

Implementácia nových stavov vektorov minimálnej voľby do rámca Solvatore môže prebiehať viacerými spôsobmi. Jedným by bolo kódovanie stavu jedného bitu dvomi premennými v logickom výraze simulujúcom transformácie MCV, nakoľko už môže byť jeden bit v 4 stavoch.

Iná možnosť, ktorú sme implementovali v triede `SolvatoreBlue1`, využíva fakt, že vstupný alebo dočasný bit v nejakom vektore minimálnej voľby sa môže dostať do stavov b_0 a b_1 iba explicitným priradením, na začiatku výpočtu, ak je daný vstupný bit v množinách C_0 alebo C_1, prípadne operáciou, kde zdrojový bit je v stave b_0 alebo b_1. Ak je bit v stave b_0 alebo b_1, pre všetky vektory minimálnej voľby v MCV platí, že na prislúchajúcom mieste je b_0 resp. b_1. Vďaka týmto vlastnostiam pri konštruovaní logického výrazu simulujúceho transformácie MCV nepotrebujeme pre bity v stavoch b_0 a b_1 logickú premennú, stačí uchovávať uchovávať informáciu, či je daný bit v stave b_0 alebo v stave b_1 explicitne.

Pri tvorbe logického výrazu simulujúceho transformácie MCV v objekte triedy `Solvatore` sú jednotlivé premenné reprezentované číslom. Klauzuly do celkového logického výrazu sú pridávané ako pole čísel, ktoré sú záporné alebo kladné v závislosti od toho, či má byť daná premenná negovaná. Pre každý vstupný alebo dočasný bit sú im zodpovedajúce premenné vždy uložené kladné hodnoty.

Preto sme si vybrali konštanty -2 resp. -3, ktoré sú uložené ako premenné zodpovedajúce vstupnému alebo dočasnému bitu ak je v stave b_0 resp. b_1. Pri tvorení logického výrazu opisujúceho transformácie MCV hneď po inicializácii prepíšeme pre všetky vstupné modré bity ich pôvodné premenné hodnotami -2 alebo -3.

Následne pri vykonávaní operácií šifrovania sledujeme, či je niektorý vstupný alebo dočasný bit v stave b_0 alebo b_1 tým, že skontrolujeme, či bitu priradená premenná je záporná . Ak nie je ani jeden zdrojový ani cieľový bit operácie v stave b_0 alebo b_1, tak sa vykoná transformácia rovnako ako pre `Solvatore`. Ak je niektorý zdrojový alebo cieľový bit v stave b_0 alebo b_1, vykoná sa transformácia podľa tabuľky 3.1.

Implementovali sme nový modul `analysis_blue1`, ktorý vykonáva hľadanie modrých rozlišovačov pomocou nových stavov bitu v MCV. Pri hľadaní skúmame iba kombinácie vstupných bitov kde množiny A a C_0 tvoria rozklad množiny vstupných bitov. Keď sa pozrieme do tabuľky 3.1, bit v stave b_1 oproti bitu v stave 0 neponúka žiadnu výhodu, okrem prípadu keď je operácia XOR aplikovaná na dva bity v stave b_1 a b_1, kedy sa cieľ operácie zmení na b_0. Ak by teda namiesto b_1 boli všetky bity v stave b_0

bolo by to výhodnejšie.

3.2.1 Aplikácia na šifru Speck

Pre analýzu prezentovanú v časti 3.2 sú vhodné predovšetkým šifry, ktoré nepoužívajú „key whitening“ na začiatku, nakoľko takáto operácia zmení všetky modré bity tak, ako keby boli iba konštantné hneď na začiatku. Analyzovali sme šifru Speck [2], ktorá nepoužíva „key whitening“ kvôli jednoduchosti implementácie.

Implementovali sme pre šifru nový modul `speck_with_key`, ktorý je založený na module `speck`, ale keďže je potrebné pri tomto type hľadania modrých rozlišovačov dbať aj na pripočítanie kľúča, je v novom module implementované.

Testovali sme najskôr verziu Speck so skrátenou dĺžkou bloku na 16 bitov, nakoľko kompletný výpočet pre oficiálnu verziu s najkratším blokom dlhým 32 bitov nie je výpočtovo realizovateľný na nám dostupnom hardvéri v realistickom čase. Konštanty pre ľavý a pravý posun v šifre sme zvolili rovnaké ako pre 32 bitovú verziu – 7 pre pravý posun a 2 pre ľavý.

Pomocou modulu `analysis_optimal` bol pre takúto šifru nájdený integrálny rozlišovač pre 6 kôl šifrovania $I = (\{1, \ldots, 10, 12, \ldots, 16\}, \{8, 16\})$. Použitím nového spôsobu hľadania modrých rozlišovačov pomocou modulu `analysis_blue1` bol pre 6 kôl nájdený iba modrý rozlišovač $I^b = (\{1, \ldots, 10, 12, \ldots, 16\}, \{11\}, \{\}, \{8, 16\})$. Modrý rozlišovač I^b líši od I iba tým, že bit s indexom 6 bol konštantný, a v I^b je modrý s hodnotou 0. Pre 7 kôl šifrovania nebol nájdený integrálny resp. modrý rozlišovač ani jedným modulom.

Pre verziu Speck s dĺžkou bloku 32 analýza použitím `analysis_optimal` odhalila pre 6 kôl šifrovania jeden integrálny rozlišovač – $I = (\{1, \ldots, 26, 28, \ldots, 32\}, \{16\})$ a pre 7 kôl nenašla žiadny. Analýzu pomocou `analysis_blue1` sme kvôli výpočtovým obmedzeniam realizovali len pre kombinácie aktívnych a modrých bitov, kde bolo najviac 5 alebo aspoň 27 modrých bitov. Pre 6 kôl bol nájdený modrý rozlišovač $I^b = (\{1, \ldots, 26, 28, \ldots, 32\}, \{27\}, \{\}, \{16\})$, ktorý znovu zodpovedá už nájdenému integrálnemu rozlišovaču. Pre 7 kôl nebol nájdený modrý rozlišovač.

Implementácia hľadania modrých rozlišovačov pomocou nových stavov bitu v MCV na Speck neviedla k nájdeniu modrých rozlišovačov, ktoré by existovali pre viac kôl šifry, alebo mali pre rovnaký počet kôl menej aktívnych bitov. Dosiahnuté výsledky sú na rovnakej úrovni ako pôvodné hľadanie pomocou `analysis_optimal`. Navyše, výpočet novou metódou je výrazne časovo náročnejší.

3.3 Integrácia pomocou zjednodušenia prvého kola

Tento spôsob implementácie spočíva v tom, že pre hľadanie integrálneho rozlišovača pomocou Solvatore zameníme prvé kolo skúmanej šifry za také, aké by vzniklo, ak by do šifrovacej funkcie pre prvé kolo boli za vstupné bity, ktorých indexy sú v C_0 a C_1 hodnoty 0 a 1.

Na vytváranie boolovskej funkcie pre šifru sa môžeme pozrieť aj tak, že pridanie jedného kola na začiatku navyše je ekvivalentné dosadeniu boolovských funkcií šifrovania jedného kola za premenné zodpovedajúce vstupným bitom. Nech máme boolovské funkcie $f_{k,i}$ pre $i \in \{1, \ldots, n\}$ zodpovedajúce $r-1$ kolám šifrovania. Nech $f'_{k,i}$ sú boolovské funkcie šifrovania jedného kola pre i-ty bit, kde za modré bity boli dosadené im zodpovedajúce hodnoty a $f''_{k,i}$ sú boolovské funkcie šifrovania jedného kola pre i-ty bit, kde modré bity nie sú dosadené. Ak sa pri konštruovaní funkcií pre r kôl dosadia do $f_{k,i}$ za počiatočné kolo funkcie $f'_{k,j}$ pre $j \in \{1, \ldots, n\}$, výsledná funkcia bude ekvivalentná tomu, ak by do $f_{k,i}$ pre boli dosadené funkcie $f''_{k,j}$ pre $j \in \{1, \ldots, n\}$, a následne za modré bity dosadené im priradené hodnoty.

Vďaka tejto ekvivalencii by malo stačiť pri implementácii v Solvatore upraviť iba prvé kolo šifry dosadením hodnôt za modré bity a ostatné kolá nechať bez zmeny. Avšak kvôli tomu, že Solvatore pracuje iba s MCV, nie reálnymi funkciami, zjednodušený výpočet nemusí nájsť modrý rozlišovač, čo je detailnejšie popísané v časti 3.4.

Implementácia skladá z 3 krokov:

1. Vytvorenie boolovských funkcií pre prvé kolo šifrovania, kde budú za modré bity explicitne dosadené im priradené hodnoty.

2. Modelovanie týchto boolovských funkcií pomocou Solvatore.

3. Úprava Solvatore, aby pri hľadaní integrálnych rozlišovačov bola namiesto prvého kola skúmanej šifry zamenená funkcia z kroku 2.

3.3.1 Vytvorenie boolovských funkcií

V tomto kroku je potrebné vytvoriť boolovské funkcie pre každý bit pre jedno kolo šifry. To sme implementovali dvoma spôsobmi – jeden zahŕňa manuálne vytvorenie boolovských funkcií napríklad podľa špecifikácie šifry. Druhý spôsob spočíva v tom, že sa pre už existujúci opis šifry v module `ciphers` pridajú operácie ako pripočítanie kľúča alebo konštanty pre operáciu modulárneho pripočítania konštanty, a boolovské funkcie sa vytvoria na základe daného objektu triedy `CipherDescription`.

Manuálne vytvorenie boolovských funkcií

Môže byť použitý podobný spôsob, ako je uvedený pre šifru RES v časti 2.2.1. Manuálna tvorba boolovských funkcií je časovo náročnejšia na implementáciu, ale správnosť vygenerovaných funkcií sa však dá overiť dosadením hodnôt za vstupné bity a kontrolou, či výsledok zašifrovania je zhodný so zašifrovaním nejakou referenčnou implementáciou šifry.

Takýmto spôsobom sme vytvorili `RES_blue_manual` v module `ciphers`. Pri inicializácii objektu triedy `CipherDescription` je potrebné zadať, aké modré bity majú byť použité.

Vytvorenie boolovských funkcií z objektu triedy CipherDescription

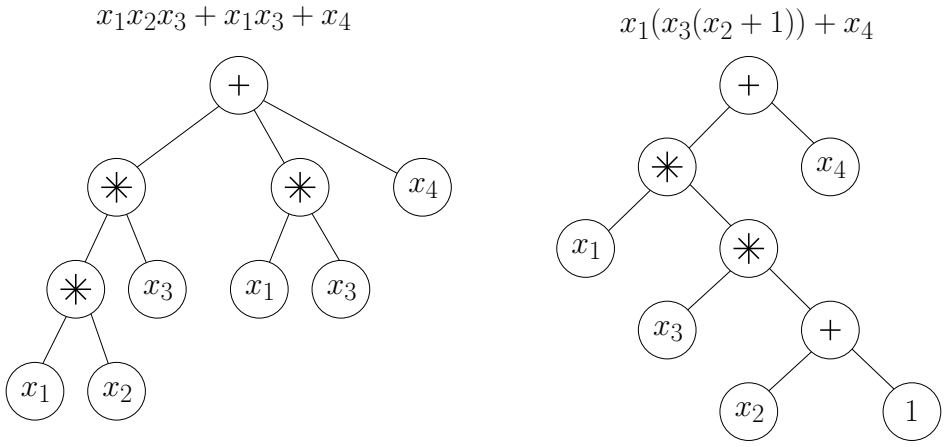
Pre šifru, ktorá je už modelovaná pomocou Solvatore sa dá vytvoriť objekt triedy `CipherDescription`, ktorý obsahuje informácie o boolovských funkciách šifry. Je však potrebné do opisu šifry obsiahnutého v `ciphers` doplniť pripočítanie kľúča, prípadne doplniť niektoré operácie tak, aby sa z nich dala odvodiť presná boolovská funkcia – napríklad nahradiť modulárne pripočítanie konštanty S-boxom.

Vytvorili sme novú triedu `CipherDescriptionBlue2`, ktorej objekt pri inicializácii dostane ako argument objekt triedy `CipherDescription` pre vzorovú šifru, ktorá má byť zjednodušená. Na základe tejto vzorovej šifry sú na vstupných bitoch postupne vykonané všetky operácie v poli `transition` v objekte vzorovej šifry a tak sú vybudované boolovské funkcie v ANF pre jednotlivé bity stavu. Tieto boolovské funkcie sú uložené v premennej v objekte a keď sa funkciou `set_blue_bits` určí, ktoré bity majú patriť do množín C_0 a C_1, sú do týchto funkcií dosadené príslušné hodnoty a na základe týchto funkcií je vytvorené nové pole `transition`, ako je opísané v časti 3.3.2.

3.3.2 Modelovanie boolovských funkcií v Solvatore

S boolovskými funkciami pre šifrovanie budeme pracovať ako so syntaktickými stromami, ako je vyobrazené na obrázku 3.1. Pre každú funkciu budeme pomocou prehľadávania do hĺbky postupne aplikovať všetky operácie, smerom od listov vyššie, kde výsledok bude vždy uložený v novej dočasnej premennej. Listy stromu, teda vstupné premenné boolovskej funkcie pred každým použitím operácie `apply_mov` skopírujeme do novej dočasnej premennej. Pri spracovávaní je pripočítavanie konštantného bitu 1 ignorované. Pre bity kľúča je možné vytvoriť nový dočasný bit pomocou metódy `apply_add_temp` a pracovať s ním ako s inými bitmi.

Po spracovaní operácií pre všetky funkcie výstupných bitov šifrovania sú dočasné premenné prislúchajúce koreňom ich syntaktických stromov presunuté do vstupných bitov, ktorých názvy začínajú písmenom *s*.



Obrázok 3.1: Ukážka syntaktického stromu pre funkciu $x_1(x_3(x_2 + 1)) + x_4$ v ANF a v neroznásobenom tvare

Vytvorený popis šifry je možné otestovať tak, že sa za modré bity zvolí prázdna množina a odskúša sa, či pomocou Solvatore budú nájdené rovnaké integrálne rozlišovače ako pre pôvodný popis funkcie.

3.3.3 Úprava modulu Solvatore pre výmenu prvého kola

Vytvorili sme novú triedu `SolvatoreBlue2`, ktorá je rozšírením triedy `Solvatore`. V nej sme pridali možnosť do objektu triedy `SolvatoreBlue2` načítať dva objekty triedy `CipherDescription` – jeden pre zjednodušené prvé kolo šifry, druhý pre pôvodný objekt pre skúmanú šifru.

Upravili sme funkciu `create_conditions`, v ktorom sa vytvára logický výraz opisujúci transformácie MCV tak, aby sa pre prvé kolo šifrovania vytváralo podľa objektu triedy `CipherDescription` pre zjednodušené kolo, a ostatné podľa objektu pre nezjednodušenú verziu.

Pridali sme spustiteľný modul `analysis_blue_optimal`, ktorý vykonáva postup v časti 2.2.2 pre nájdenie modrých rozlišovačov s najväčším počtom konštantných bitov a používa triedu `SolvatoreBlue2`.

3.3.4 Modelovanie viacerých počiatočných kôl šifry

Ako je naznačené v časti 3.4.3, zámena jedného kola nemusí pri hľadaní modrých rozlišovačov spôsobiť žiadny rozdiel oproti výpočtu pôvodným spôsobom. To je spôsobené tým, že pre niektoré šifry po jednom kole šifrovania sú rovnaké MCV či sú použité modré bity alebo nie. Ak existuje modrý rozlišovač pre ako integrálny, po po nejakom počet kôl sa musia MCV líšiť.

Myšlienkou tejto metódy je vytvorenie boolovských funkcií pre viac začiatočných kôl ako jedno, jeho zjednodušenie a následná simulácia v Solvatore pre zvyšné kolá.

Implementácia takejto zmeny je jednoduchá – v triede `CipherDescriptionBlue2` pri tvorbe boolovských funkcií z poľa `transition` v objekte triedy `CipherDescription` pre pôvodnú šifru, kde sa postupne aplikujú všetky operácie sa týmto poľom preje toľko krát, pre koľko kôl šifry chceme vytvoriť boolovské funkcie. Ak má šifra rôzne kľúče pre rôzne kolá, je potrebné zabezpečiť aby sa pre každý bit šifry vytvorila nová premenná.

Po vytvorení boolovských funkcií prebiehajú ostatné kroky rovnako, s výnimkou toho, že pre Solvatore sa týchto niekoľko kôl šifry javí ako jedno kolo, takže treba upraviť pre aký počet kôl hľadá Solvatore modré rozlišovače.

3.3.5 Aplikácia na RES

Použitím metódy zjednodušenia prvého kola sme otestovali hľadanie modrých rozlišovačov pre každú šifru RES(c_{ls}, c_{ma}, r), pre ktorú existuje modrý rozlišovač, ale integrálny nie – tam, kde podľa tabuľky 2.4 nastalo zlepšenie. Ak nastalo zlepšenie o viac kôl, napríklad pri konfigurácii s $c_{ls} = 2$ a $c_{ma} = 11$, kde boli nájdené integrálne rozlišovače pre najviac 3 kolá, ale modré rozlišovače pre 6 kôl, testovali sme všetky kolá – hľadali sme modrý rozlišovač pre 4, 5 aj 6 kôl. V žiadnom prípade však touto metódou nebol nájdený modrý rozlišovač.

Použitím modelovania viacerých kôl sme rovnako otestovali všetky takéto konfigurácie. Pre každú trojicu c_{ls}, c_{ma} a r sme postupne od najmenšieho nahrádzali r' počiatočných kôl pre $r' \in \{1, \ldots, r-1\}$, až dokým nebol nájdený modrý rozlišovač. K nahradeniu všetkých kôl r je potrebné vytvorenie ANF pre všetky kolá šifrovania, teda oproti vyhľadávaniu modrých rozlišovačov pomocou ANF v časti 2.2.1 by to neprinášalo žiadnu výhodu.

c_{ls}	c_{ma}	r	r'	c_{ls}	c_{ma}	r	r'
1	2	8	6	2	1	11	10
2	1	4	3	2	1	12	11
2	1	5	4	2	6	7	6
2	1	9	8	3	14	10	9
2	1	10	9	3	14	11	9

Tabuľka 3.2: Počet počiatočných kôl r', ktoré bolo potrebné simulovať aby bol nájdený modrý rozlišovač pre šifru RES(c_{ls}, c_{ma}, r).

Z 92 testovaných trojíc c_{ls}, c_{ma}, r bolo možné takýmto spôsobom nájsť modrý rozlišovač v 56 prípadoch. Pre niektoré z nich uvádzame v tabuľke 3.2 počet kôl r', ktoré bolo treba nahradiť, aby bol nájdený nejaký modrý rozlišovač. Nie vždy platilo, že pre danú šifru RES(c_{ls}, c_{ma}, r) boli takýmto spôsobom nájdené všetky modré rozlišovače.

Vo väčšine prípadov pri použití tejto metódy bolo potrebné zameniť všetky kolá okrem posledného, a v 12 prípadoch všetky okrem posledných dvoch kôl. To znamená,

že vo všetkých prípadoch bolo potrebné vypočítať boolovské funkcie v tvare ANF pre takmer všetky kolá šifry, teda výpočet bol porovnateľne časovo náročný ako analýza funkcií v ANF v časti 2.2.1.

3.4 Extrakcia MCV z funkcií

Implementovali sme pomocné metódy, ktoré umožňujú extrahovať minimálne vektory voľby po ľubovoľnom počte kôl šifrovania. To nám pomohlo pri odstraňovaní chýb a poskytlo lepší náhľad do toho, ako transformácie MCV fungujú. Výpočet vždy prebieha pre konkrétne aktívne bity A a modré bity C_0, C_1. Extrakcia MCV je rozšírením hľadania integrálnych alebo modrých rozlišovačov, pri ktorých sa rieši iba otázka, či sú v poslednej množine MCV všetky vektory s Hammingovou váhou 1, zatiaľ čo pri extrakcii sa snažíme zistiť všetky vektory MCV.

Časová zložitosť takéhoto výpočtu je exponenciálna v závislosti od počtu výstupných bitov pre výpočet MCV pre konkrétne modré a aktívne bity a nepridáva informáciu potrebnú k určeniu, či daný integrálny resp. modrý rozlišovač existuje alebo nie.

MCV je definovaná ako $\{v \mid v \in \{0,1\}^n\}$, kde n je počet vstupných a výstupných bitov šifry. Iná reprezentácia vektora minimálnej voľby v je množina indexov, kde v je rovný 1.

Hľadáme vždy len minimálne vektory voľby, teda v množine MCV nebude vektor u, ktorý predstavuje znásobenie iného vektora v s ďalšou funkciou – $\forall u, v \in MCV' : u \not\subset v \wedge v \not\subset u$

3.4.1 Solvatore

Hľadanie integrálnych rozlišovačov v triede `Solvatore` je implementované pomocou funkcie `is_bit_balanced`, ktorý dostane ako vstup dostane množinu aktívnych bitov A a index b výstupného bitu, o ktorom sa zisťuje, či je balansovaný. Najskôr sa vytvorí logický výraz opisujúceho transformácie MCV, ako je opísané v časti 1.3.2. Následne sa do neho na základe aktívnych bitov A a indexu b vložia klauzuly, ktoré opisujú ako má vyzerať prvá a posledná množina MCV – prvá má hodnoty 1 na indexoch aktívnych bitov, a posledná má 1 na indexe b a 0 na ostatných. Takýto celý výraz sa overí pomocou SAT solvera, ak nie je splniteľný, výstupný bit b je balansovaný.

Pridali sme funkciu `extract_mcv`, ktorá funguje podobne ako `is_bit_balanced`, ale otestuje dosiahnuteľnosť pre všetky vektory minimálnej voľby $v \subseteq \{1, \ldots, n\}$. Ak budú dosiahnuteľné, teda logický výraz bude splniteľný, daný vektor minimálnej voľby sa uloží. Po vyskúšaní všetkých sa vrátia všetky uložené vektory minimálnej voľby.

Nie je potrebné skúšať vektory minimálnej voľby, ktoré sú nadmnožinou iného vektoru minimálnej voľby, ktorý je dosiahnuteľný. Vďaka tomu, že pri skúšaní dosiahnuteľnosti vektorov minimálnej voľby sa postupuje od vektorov s najmenším počtom členov k najväčším, stačí skontrolovať, či práve skúšaný vektor minimálnej voľby nie je nadmnožina iného, ktorý už bol nájdený.

3.4.2 ANF funkcie

Pre vektor minimálnej voľby $v \subseteq \{1, \ldots, n\}$ platí, že znásobením všetkých funkcií $f_{k,i}$ pre $i \in v$ vznikne funkcia, ktorá v ANF bude obsahovať monóm, ktorého prvkami budú všetky aktívne bity. Môžeme teda testovať kandidáta v na vektor minimálnej voľby tak, že pre potenciálny vektor minimálnej voľby $v = \{v_1, v_2, \ldots, v_p\}$ funkcie $f_{k,v_1}, f_{k,v_2}, \ldots, f_{k,v_p}$ znásobíme, transformujeme do ANF a skontrolujeme existenciu takého monómu.

Pri extrakcii MCV sa postupne prechádzajú všetky možné v zoradené podľa mohutnosti, začínajúc od $|v| = 1$. Rovnako ako pri extrakcii MCV v Solvatore, neskúšame vektory u, ktoré sú už nadmnožinou nejakého iného už nájdeného vektora minimálnej voľby $v \in MCV$.

3.4.3 Rozdiely transformácií MCV na konkrétnom príklade

Porovnáme transformácie MCV z funkcií šifrovania v ANF a z výpočtu pomocou Solvatore. Pre obe extrahujeme MCV aj pre varianty s použitím modrých bitov, pri Solvatore použijeme metódu nahradenia prvého kola. Extrahujeme MCV po každom kole šifrovania RES(2, 1, 5), pretože je to jedna z konfigurácii, kde existuje modrý rozlišovač ale neexistuje integrálny rozlišovač. Všetky MCV sú uvedené pre aktívne bity $A = \{3, 4\}$, a bity s indexmi $\{1, 2\}$ sú konštantné alebo modré s hodnotou 0. Výsledky je možné vidieť v tabuľke 3.3.

	Analýza Solvatore		Analýza ANF	
Kolo	Bez modrých bitov	S Modrými bitmi	Bez modrých bitov	S Modrými bitmi
1	(1, 2), (3), (4)	(1, 2), (3), (4)	(1, 2), (3), (4)	(1, 2), (3), (4)
2	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)
3	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)
4	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3)
5	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (2), (3), (4)	(1), (3), (4)

Tabuľka 3.3: Množiny MCV vypočítané pomocou Solvatore a analýzou ANF po každom kole pre konfiguráciu šifry RES(2, 1, 5).

Ako môžeme vidieť v tabuľke 3.3, po troch kolách šifrovania neexistuje modrý ani integrálny rozlišovač s danými aktívnymi bitmi A, keďže v MCV sú všetky vektory s Hammingovou váhou 1.

Pre štyri kolá však existuje $I^b = (A, C_0, \{\}, \{4\})$. Pozrieme na funkciu $f_{k,4}$ pre štvrtý bit šifry RES(2, 1, 4). Nech $f'_{k,i}$ pre $i \in \{1, \ldots, 4\}$ sú šifrovacie funkcie pre RES(2, 1, 3). Pre funkciu $f_{k,4}$ platí $f_{k,4} = f'_{k,2} + k_2 + (f'_{k,3} + k_3)(f'_{k,4} + k_4)$. Vo funkcii $f'_{k,2}$ sa nachádzajú monómy obsahujúce všetky aktívne bity $x_3x_4k_1$, $x_3x_4k_3$ a $x_3x_4k_2k_4$. Rovnaké sa nachádzajú aj v $(f'_{k,3} + k_3)(f'_{k,4} + k_4)$, všetky sa sčítajú na 0 a vo funkcii $f_{k,4}$ sa už nevyskytujú. Takýto typ skrátenia však zjednodušený výpočet v Solvatore nezachytáva, ako je už spomenuté v časti 1.3.

Pri tejto konfigurácii platí, že v prvom kole sú všetky MCV rovnaké, a začnú sa líšiť až po štvrtom kole. Pri Solvatore platí, že hodnoty MCV po niektorom kole závisia iba od MCV pre ním. Ak teda pri hľadaní modrých rozlišovačov v Solvatore zameníme iba prvé kolo, ktoré má identické MCV, potom takáto zámena nebude mať žiadny vplyv na nájdené rozlišovače.

Kapitola 4

Realizácia integrálneho útoku

Táto kapitola opisuje realizáciu jednoduchého integrálneho útoku pomocou nájdených modrých resp. integrálnych rozlišovačov a analýzu jeho časovej a priestorovej zložitosti. Uvádzame aj možné zlepšenia tejto jednoduchej kryptoanalýzy.

4.1 Jednoduchý integrálny útok

Nech $E_{k,r} : \{0,1\}^n \to \{0,1\}^n$ je bijektívna funkcia šifrovania ako definovaná v sekcii 1.1.1, ktorá prebieha v r kolách a $f_{k,i,r} : \{0,1\}^n \to \{0,1\}$ pre $i \in \{1,\ldots,n\}$ sú funkcie šifrovania pre jednotlivé bity. Nech sa kľúč k v šifre $E_{k,r}$ použije na odvodenie r podkľúčov $k_j \in \{0,1\}^n$ pre $j \in \{1,\ldots,r\}$, kde k_j je použitý v j-tom kole. Nech sa pre jednoduchosť konštrukcie nepoužíva „key whitening“. Naším cieľom je získať všetky bity jednotlivých podkľúčov.

Nech $\mathbf{I^b} = \{I_1^b, I_2^b, \ldots, I_x^b\}$ je množina nájdených modrých rozlišovačov pre šifrovaciu funkciu $E_{k,r}$. Pomocou nich je možné vykonať útok na šifrovaciu funkciu $E_{k,r+1}$ a získanie posledného podkľúča k_{r+1}.

Zoberme si útok pomocou modrého rozlišovača $I_p^b = (A, C_0, C_1, B)$ pre nejaké $p \in \{1,\ldots,x\}$. Najskôr je potrebné identifikovať množinu K – ktoré bity podkľúča k_{r+1} je potrebné uhádnuť, aby bolo možné čiastočne dešifrovať jedno kolo šifry $E_{k,r+1}$. Funkcia $E_{k',1}^{-1} : \{0,1\}^n \to \{0,1\}^n$ je funkcia dešifrovania posledného kola $E_{k,r+1}$ pomocou podkľúča k' a funkcia $d_{k',i,1} : \{0,1\}^n \to \{0,1\}$ pre $i \in 1,\ldots,n$ predstavuje jej i-ty výstupný bit. Pre každý balansovaný bit $y \in B$ zaradíme všetky indexy bitov podkľúča vystupujúce vo funkcii $d_{k',y,1}$ do množiny K. Ak už je hodnota bitu podkľúča známa vďaka útoku pomocou iného rozlišovača, takýto bit nie je zaradený do množiny K.

Hodnoty bitov podkľúča k_{r+1} s indexmi v množine K sú následne extrahované pomocou algoritmu 4.1.

Algoritmus 4.1 Zistenie časti podkľúča k_{r+1} pre šifru $E_{k,r+1}$. Extrahované sú bity podkľúča s indexmi množine K použitím modrého rozlišovača I^b

Vstup: K, $I^b = (A, C_0, C_1, B), E_{k,r+1}, d_{k',i,1}$ pre $i \in B$

Výstup: Hodnoty pre bity podkľúča k_{r+1} s indexmi v K, prípadne prázdna množina neboli hodnoty s istotou zistené

1: $pk \leftarrow \{v \mid v \in \{0,1\}^{|K|}\}$ ▷ Množina kandidátskych kľúčov
2: **for all** S_0 pre A, C_0, C_1 **do**
3: $S_{r+1} = \{E_{k,r+1}(v) \mid v \in S_0\}$ ▷ Zašifrovanie pomocou orákula
4: **for all** $v_k \in pk$ **do**
5: $k' \leftarrow$ ľubovoľný kľúč, kde na miestach už známych bitov sú ich hodnoty a $\forall i \in \{1, \ldots, |K|\}, v_k[i]$ je na indexe i-teho najmenšieho prvku K
6: **for all** $b \in B$ **do**
7: $sum \leftarrow \sum_{t \in S_{r+1}} d_{k',b,1}(t)$
8: **if** sum $= 1$ **then** ▷ Otestovanie, či je bit b balansovaný
9: $pk \leftarrow pk - \{v_k\}$ ▷ Odobratie v_k z pk
10: **break**
11: **if** $|pk| = 1$ **then return** $pk[0]$
return $\emptyset$

Takýmto spôsobom je možné prejsť všetkými integrálnymi rozlišovačmi $I_p^b \in \mathbf{I^b}$. Ak niektoré bity podkľúča stále nie sú nájdené, je možné ich tipnúť a ak ďalší výpočet neprebehne úspešne tento tip zmeniť a počítať znovu.

Keď sú známe všetky bity daného podkľúča, je možné pre všetky šifrové texty dešifrovať jedno kolo a teda je možné rovnakým spôsobom útočiť na r kôl, tentokrát s použitím integrálnych rozlišovačov pre $r - 1$ kôl šifry.

Oproti integrálnym rozlišovačom sa útok použitím modrých rozlišovačov nemení. Jediný rozdiel je v tom, že pri modrom rozlišovači za modré bity nemôžeme dosadiť iné hodnoty. Ak by sme mali integrálny a modrý rozlišovač I a I^b, ktoré sa líšia iba tým, že namiesto niektorých konštantných bitov v I sú v I^b modré bity, množín S_0, ktoré by sme mohli pri útoku vytvoriť by bolo menej a v niektorých prípadoch by to znamenalo, že by pri útoku neprebehlo dostatočne iterácii cyklu začínajúceho v riadku 2 v algoritme 4.1. Keď však existuje modrý rozlišovač pre takú šifru, pre ktorú integrálny rozlišovač neexistuje, je takýto rozlišovač prospešný, pretože pomocou integrálneho rozlišovača by sa na šifru nedalo zaútočiť.

Pri nasledujúcom dôkaze používame zjednodušujúce predpoklady:

(A1) Pre nesprávne uhádnutý podkľúč k' bude pre jeden balansovaný bit $b \in B$ podmienka $\sum_{t \in S_{r+1}} d_{k',b,1}(t) = 0$ splnená s pravdepodobnosťou $1/2$. Pre každú dvojicu bitov $b_1, b_2 \in B, i \neq j$ bude táto pravdepodobnosť nezávislá.

(A2) Pravdepodobnosti vyradenia po i-tom kole pre nesprávne uhádnuté kľúče sú po dvojiciach nezávislé.

Lema 4.1. Očakávaný počet iterácií cyklu v algoritme 4.1 začínajúcom na riadku 3 pre $I^b = (A, C_0, C_1, B)$ s počtom bitov podkľúča, ktoré treba uhádnuť $|K|$ je ohraničený zospodu $\frac{1}{|B|*\log_e(2)} \sum_{j=1}^{2^{|K|}-1} \frac{1}{j}$ a zvrchu $1 + \frac{1}{|B|*\log_e(2)} \sum_{j=1}^{2^{|K|}-1} \frac{1}{j}$.

Dôkaz. Nech náhodná premenná X určuje pre konkrétny kandidátsky podkľúč v'_k rôzny od správneho v_k, po koľkých kolách bude z množiny kandidátskych podkľúčov odstránený. Podľa algoritmu 4.1 môžeme vidieť, že aby aj po i iteráciách s rôznymi S_0 a teda rôznymi S_{r+1} bol v'_k v množine kandidátskych podkľúčov pk, musela byť preň i krát splnená po dešifrovaní jedného kola podmienka $\forall b \in B \sum_{t \in S_{r+1}} d_{k',b,1}(t) = 0$.

Podľa predpokladu (A1) je pravdepodobnosť 1/2, že táto podmienka bude pre jeden balansovaný bit splnená, a pravdepodobnosti pre jednotlivé balansované bity sú po dvojiciach nezávislé. Potom pravdepodobnosť, že v'_k splní podmienku pre $|B|$ bitov je $2^{-|B|}$. Náhodná premenná X má geometrické rozdelenie, teda $P[X = i] = p^{i-1}(1-p)$, kde $p = 2^{-|B|}$.

Nech náhodná premenná $M^*_{2^{|K|}-1}$ je maximum identických nezávislých (kvôli predpokladu (A2)) premenných X pre každý kľúč. Táto náhodná premenná reprezentuje počet kôl, po ktorom je každý nesprávny kľúč v'_k odstránený z množiny kandidátskych podkľúčov pk – $M^*_{2^{|K|}-1} = \max(X_1, \ldots, X_{2^{|K|}-1})$. Stredná hodnota $M^*_{2^{|K|}-1}$ je ohraničená zospodu a zvrchu [6]

$$\frac{1}{|B| * \log_e(2)} \sum_{j=1}^{2^{|K|}-1} \frac{1}{j} < E(M^*_{2^{|K|}-1}) < 1 + \frac{1}{|B| * \log_e(2)} \sum_{j=1}^{2^{|K|}-1} \frac{1}{j}.$$

Označíme H_i i-ty čiastočný súčet harmonického radu. Potom:

$$\frac{H_{2^{|K|}-1}}{|B| \cdot \log_e(2)} < E(M^*_{2^{|K|}-1}) < 1 + \frac{H_{2^{|K|}-1}}{|B| \cdot \log_e(2)}.$$

□

4.1.1 Časová a priestorová zložitosť útoku

Zložitosť útoku je závislá od konkrétnych integrálnych rozlišovačov.

Priestorová zložitosť. Pre $I^b_p = (A_p, C_{0,p}, C_{1,p}, B_p)$ a množinu K_p vypočítanú z B_p je potrebné uložiť $2^{|A_p|}$ šifrových textov, prinajhoršom $2^{|K_p|}$ kandidátskych podkľúčov a konštantné bity pre tvorbu S_0 môžu byť zadávané postupne, teda na uloženie stavu stačí počítadlo s n bitmi, čo je zanedbateľné. Očakávaná priestorová zložitosť útoku je preto

$$O(\max_{I^b_p \in \mathbf{I^b}} \left(2^{|A_p|} + 2^{|K_p|}\right)).$$

Časová zložitosť. Pre jeden modrý rozlišovač $I_p^b = (A_p, C_{0,p}, C_{1,p}, B_p)$ a množinu K_p vypočítanú z B_p na základe šifry sa dá očakávať, že pre algoritmus 4.1 prebehne cyklus začínajúci v riadku 2 $\left(\frac{H_{2^{|K_p|}-1}}{|B_p|\cdot\log_e(2)}+1\right)$ krát. V každej iterácii cyklu je potrebné vytvoriť $2^{|A_p|}$ otvorených textov, pre každý prvok urobiť dopyt orákulu a pre každý z 2^{K_p} kandidátskych podkľúčov vykonať $2^{|A_p|}$ dešifrovaní jedného kola a $2^{|A_p|}$ sčítaní. Očakávaná časová zložitosť je potom

$$O\left(\sum_{I_p^b\in\mathbf{I^b}}\left(\left(2^{|A_p|}+2^{|K_p|+|A_p|}\right)\cdot\frac{H_{2^{|K_p|}-1}}{|B_p|\cdot\log_e(2)}\right)\right).$$

4.2 Možné zlepšenia jednoduchej kryptoanalýzy

V časti 4.1 uvádzame jednoduchý útok kvôli jednoduchosti výkladu. V tejto časti ukážeme niektoré všeobecné spôsoby, ktorými sa dá vylepšiť takýto útok.

Útok na $E_{k,r+2}$ s počtom kôl $r+2$. Realizácia kryptoanalýzy môže prebiehať rovnakým spôsobom ako na $E_{k,r+1}$ s $r+1$ kolami, ale namiesto $E_{k',1}^{-1}$ sa použije funkcia $E_{k',2}^{-1}$ dešifrovania posledných dvoch kôl $E_{k,r+2}$, a k nej zodpovedajúce $d_{k',i,2}$. Množina K bude obsahovať nie len bity podkľúča k_{r+1}, ale aj podkľúča k_{r+2}. Takýto útok má však vyššiu časovú a pamäťovú zložitosť zapríčinenú väčšou množinou K.

Hádanie kľúča pre podmnožinu balansovaných bitov samostatne. Ak existuje pre danú šifru $I^b = (A, C_0, C_1, B)$, existuje aj $I^{b'} = (A, C_0, C_1, B')$, kde $B' \subset B$. Myšlienka je v tom, že pre nejakú podmnožinu balansovaných bitov B' môže byť aj $K' \subset K$ pre I^b. Ak by platilo $K' = K$, oplatí sa použiť I^b. Takto by bolo možné hádať bity kľúča po častiach s nižšou časovou zložitosťou.

Použitie neúplnej informácie o podkľúči z nepodareného použitia integrálneho rozlišovača. Pri aktuálnom postupe ak po vyskúšaní všetkých možných S_0 neostane jeden kandidátsky podkľúč, celá informácia sa zahodí. Vhodným pospájaním informácie by ju však bolo možné využiť, napríklad ich zakódovaním do výrazu pre SAT solver, ktorý by po spočítaní všetkých rozlišovačov hľadal riešenie.

Záver

V práci sme skúmali vplyv voľby konkrétnych konštantných bitov na integrálne rozlišovače a možnosti ich použitia pri integrálnej kryptoanalýze.

Najskôr sme predstavili integrálnu kryptoanalýzu a rámec Solvatore, pri ktorom sme formálne dokázali, že stratégia hľadania integrálnych rozlišovačov použitá v tomto rámci vždy nájde integrálny rozlišovač s najväčším počtom konštantných bitov.

Dokázali sme, že pri modelovaní šifier kvôli hľadaniu integrálnych rozlišovačov nebolo nutné pripočítavanie bitov kľúča, a že pri práci s modrými bitmi to bude potrebné. Ukázali sme aj to, že pre hľadanie modrých rozlišovačov nebude stačiť pôvodná stratégia implementovaná v Solvatore.

Vytvorili sme jednoduchú šifru RES, na ktorej sme demonštrovali pomocou analýzy šifrovacích funkcií v ANF, že existujú modré rozlišovače pre väčší počet kôl ako integrálne rozlišovače, a teda že myšlienka použitia modrých bitov má prínos pre integrálnu kryptoanalýzu. Rozšírili sme pôvodnú stratégiu pre hľadanie integrálnych rozlišovačov s najväčším počtom konštantných bitov pre modré rozlišovače a dokázali, takýmto spôsobom bude vždy taký modrý rozlišovač nájdený.

Implementovali sme hľadanie modrých rozlišovačov do rámca Solvatore dvoma spôsobmi. Jeden spôsob zahŕňal pridanie nových stavov pre vektory minimálnej voľby, pričom sme využili niektoré vlastnosti takýchto nových stavov tak, že implementácia nezahŕňala väčšie zásahy do projektu Solvatore. Hľadanie použitím nových stavov však má značné obmedzenia, kvôli ktorým nemá pri časti šifier výhodu oproti hľadaniu integrálnych rozlišovačov. Otestovali sme pomocou neho skrátenú verziu šifry Speck s dĺžkou bloku 16 bitov a čiastočne sme otestovali aj verziu s dĺžkou bloku 32 bitov. Našli sa iba také modré rozlišovače, ktoré sa líšili od integrálnych rozlišovačov nájdených Solvatore iba tým, že mali modré bity na miestach konštantných bitov. Navyše, nový spôsob je časovo náročnejší na výpočet.

Druhý spôsob je založený na vytvorení prvého kola šifry ako boolovskej funkcie, jeho zjednodušení použitím modrých bitov a zámene prvého kola v Solvatore za jeho zjednodušenú verziu. Použitím tohto spôsobu na šifru RES bolo možné nájsť iba modré rozlišovače, ktoré existovali pre rovnaký počet kôl ako už nájdené integrálne rozlišovače. Zlepšením tohto spôsobu je vytvorenie boolovskej funkcie pre viaceré začiatočné kolá a po zjednodušení ich zámena za kolá v Solvatore. Týmto spôsobom sa nám podarilo

pomocou Solvatore nájsť modré rozlišovače pre väčší počet kôl ako integrálne rozlišovače. V porovnaní s analýzou pomocou funkcií šifrovania v ANF neboli touto novou metódou nájdené integrálne rozlišovače pre všetky konfigurácie šifry, ani pre konfigurácie kde boli nájdené, neboli vždy nájdené všetky. Vo väčšine prípadov bolo potrebné vytvárať boolovské funkcie v ANF pre takmer všetky kolá, čo znamená výpočtovú zložitosť porovnateľnú s priamočiarou analýzou ANF.

Navrhli a implementovali sme techniku pre extrakciu vektorov minimálnej voľby pre Solvatore aj pre vytvorené funkcie v ANF. Táto môže byť užitočná pre lepší náhľad na Solvatore, či už pre odstraňovanie chýb v implementácii, alebo pochopenie transformácii vektorov minimálnej voľby.

V poslednej kapitole sme opísali jednoduchý integrálny útok na šifru s použitím modrých alebo integrálnych rozlišovačov a odhadli sme jeho časovú a priestorovú zložitosť. Navrhli sme spôsoby, ktorými sa jednoduchý útok dá zlepšiť.

Problematika modrých bitov nie je touto prácou vyčerpaná. V budúcnosti by bolo zaujímavé skúmať existenciu integrálnych rozlišovačov s modrými bitmi v kľúči. Scenár by mohol byť taký, že útočník pozná časť kľúča, prípadne si ju tipne a vytvorí si sadu modrých rozlišovačov pre rôzne hodnoty kľúča.

Príloha A: obsah elektronickej prílohy

Elektronická príloha priložená k práci obsahuje experimentálne prostredie implementované v nástroji Jupyter-notebook [9], ktorým prebiehali výpočty v kapitole 2 a kópiu projektu Solvatore [15] v ktorom sú implementované zmeny opísané v kapitole 3. Prikladáme aj nájdené modré a integrálne rozlišovače z časti 2.2.1 a modré rozlišovače nájdené pomocou zjednodušenia niekoľkých začiatočných kôl 3.3.5, spolu aj s počtami kôl, ktoré bolo potrebné modelovať.

Experimentálne prostredie sa nachádza v priečinku `expr`, projekt Solvatore v priečinku `solvatore` a nájdené rozlišovače v súbore `distinguishers.txt`. Pripojený je aj súbor `requirements.txt`, ktorý slúži na opis balíčkov pre programovací jazyk Python potrebných k spusteniu jednotlivých častí. Tieto balíčky je možné nainštalovať pomocou príkazu `pip install -r requirements.txt`. Solvatore aj experimentálne prostredie používajú jazyk Python vo verzii 2.7.18 .

Literatúra

[1] Paulo S. L. M. Barreto, Vincent Rijmen, Jorge Nakahara, Bart Preneel, Joos Vandewalle, and Hae Y. Kim. Improved Square Attacks against Reduced-Round Hierocrypt. In Mitsuru Matsui, editor, *Fast Software Encryption*, pages 165–173, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[2] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/585, 2015. URL: `https://eprint.iacr.org/2015/585`.

[3] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption*, pages 149–165, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[4] Angel Martín del Rey, Gerardo Rodríguez Sánchez, and A de la Villa Cuenca. On the boolean partial derivatives and their composition. *Applied Mathematics Letters*, 25(4):739–744, 2012.

[5] Carl D'Halluin, Gert Bijnens, Vincent Rijmen, and Bart Preneel. Attack on Six Rounds of CRYPTON. In Lars Knudsen, editor, *Fast Software Encryption*, pages 46–59, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[6] Bennett Eisenberg. On the expectation of the maximum of IID geometric random variables. *Statistics & Probability Letters*, 78(2):135–143, 2008.

[7] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding Integral Distinguishers with Ease. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 115–138, Cham, 2019. Springer International Publishing.

[8] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, pages 213–230, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[9] Jupyter notebook. [online, navštívené 29.10.2019]. URL: https://jupyter.org/.

[10] Lars Knudsen and David Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 112–127, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[11] Ru Liu Meiqin Wang Ling Sun, Wei Wang. MILP – Aided Bit-Based Division Property for ARX-based block cipher. Cryptology ePrint Archive, Report 2016/1101, 2016. https://eprint.iacr.org/2016/1101.

[12] Stefan Lucks. The Saturation Attack – a Bait for Twofish. In Mitsuru Matsui, editor, *Fast Software Encryption*, pages 1–15, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[13] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. [online, navštívené 15.5.2020]. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[14] SageMath – Sage Mathematics Software System. [online, navštívené 10.4.2020]. URL: https://www.sagemath.org.

[15] Solvatore. [online, navštívené 1.7.2019]. URL: https://github.com/kste/solvatore.

[16] Sympy – python library for symbolic mathematics. [online, navštívené 29.10.2019]. URL: https://www.sympy.org/.

[17] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 287–314, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[18] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon family. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 357–377, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[19] Yongjin Yeom, Sangwoo Park, and Iljun Kim. On the Security of CAMELLIA against the Square Attack. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 89–99, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.