Kryptografie - 1. projekt

Jan Pawlus
Brno University of Technology

April 3, 2019

1 Úvod a společná část

Tato dokumentace se zabývá popisem prvního projektu do předmětu *Kryprografie*. Popisuje postup získání klíče pro šifrování dat pro zadanou proudovou šifru jak ručně, tak pomocí *SAT solveru*.

Zadány jsou dva soubory, jejichž název nápadně napovídá, že jeden obsahuje *plaintext*, a druhý ciphertext prvního souboru. Dále jsou k dispozici zašifrované soubory `super_cipher.py.enc` a `hint.gif.enc`, kde první obsahuje pravděpodobně šifrovací algoritmus, kterým byly soubory zašifrovány. Jedinou možností je tedy zkusit `XOR` *plaintextu* a *ciphertextu*, z čehož získáme určitou sekvenci bitů (*keystream*), kterou se můžeme pokusit aplikovat na zašifrovaný skript s šifrou. Výsledkem je částečně dešifrovaný skript s šifrovacím algoritmem (o délce získaného *keystreamu*) obsahující tento kód:

```
#!/usr/bin/env python3

import argparse
import sys

parser = argparse.ArgumentParser()
parser.add_argument("key")
args = parser.parse_args()

SUB = [0, 1, 1, 0, 0, 1, 0, 1]
N_B = 32
N = 8 * N_B

# Next keystream
def step(x):
  x = (x & 1) << N+1 | x << 1 | x >> N-1
  y = 0
```

```
18    for i in range(N):
19      y |= SUB[(x >> i) & 7] << i
20    return y
21
22  # Keystream init
23  keystr = int.from_bytes(args.key.encode(),'little')
24  for i in range(N//2):
25    keystr = step(keystr)
26
27  # Encrypt/decrypt stdin2stdout
28  plaintext = sys.stdiZ|
```

Z tohoto kousku kódu lze vidět, jak byl námi získaný *keystream* vytvořen - cílem je tedy získání parametru `key` tohoto skriptu, ze kterého byl vygenerován *keystream*.

2 Ruční část

První částí projektu je získat klíč ručně, tedy sestrojením reverzní funkce k funkci `step`. K tomu je potřeba pořádně zanalyzovat funkci `step` a pokusit se v ní nalézt slabinu. Při bližším ohledáním si lze všimnout, že na řádku 16 je vstupní parametr x rozšířen o dva bity - zleva o původní `LSB` a zprava o původní `MSB`. Tento fakt zaručuje to, že se dva `MSB` a dva `LSB` x po tomto rozšíření budou rovnat, což bude dále stěžejní.

Dále si lze všimnout, že výstup y je udáván hodnotami pole `SUB` (řádek 10). Přístup do tohoto pole je určen posledními třemi bity x posunutými o i pozic v cyklu do N. Znamená to tedy, že dva `MSB` jednoho indexu musí odpovídat dvěma `LSB` dalšího indexu. Tento fakt je hledanou slabinou tohoto algoritmu a je díky němu možné pozpátku zrekonstruovat původní x na základě známého *keystreamu*, což ukazuje obrázek 1.

Na tomto obrázku lze vidět celý princip reverzní funkce. Pokud známe posloupnost bitů *keystreamu*, jdeme od jeho `LSB` po `MSB` a analyzujeme návaznost indexů do pole `SUB`. Kupříkladu pokud je `LSB` *keystreamu* 1, prozatímní potenciální výsledek je pole indexů do `SUB` takových, které ukazují na hodnotu 1. Pokud je následující bit *keystreamu* 0, ukládají se do pole `possibilities` indexy do `SUB` ukazující na hodnotu 0. Poté se porovnávají dva `MSB` prvků z `results` s dvěma `LSB` prvků z `possibilities`. Pokud se rovnají (červená kolečka), je vytvořeno nové pole `results` tak, že původní prvek z `result` (modrá kolečka) je rozšířen zleva o `MSB` odpovídajícího prvku z `possibilities` (zelená kolečka).

Po N iteracích obsahuje pole `results` všechny možnosti, kterými mohl být *keystream* vytvořen. Je třeba však vybrat jedinou, která je správná. K tomu lze využít již zmíněného faktu, že x je na začátku zleva rozšířeno o svůj `LSB` a zprava o svůj `MSB`. Stačí tedy ověřit, u kterého prvku z `results` platí, že se rovnají dva jeho `MSB` a `LSB`. Tímto je nalezen správný parametr x funkce `step`, tedy sestrojena reverzní funkce.

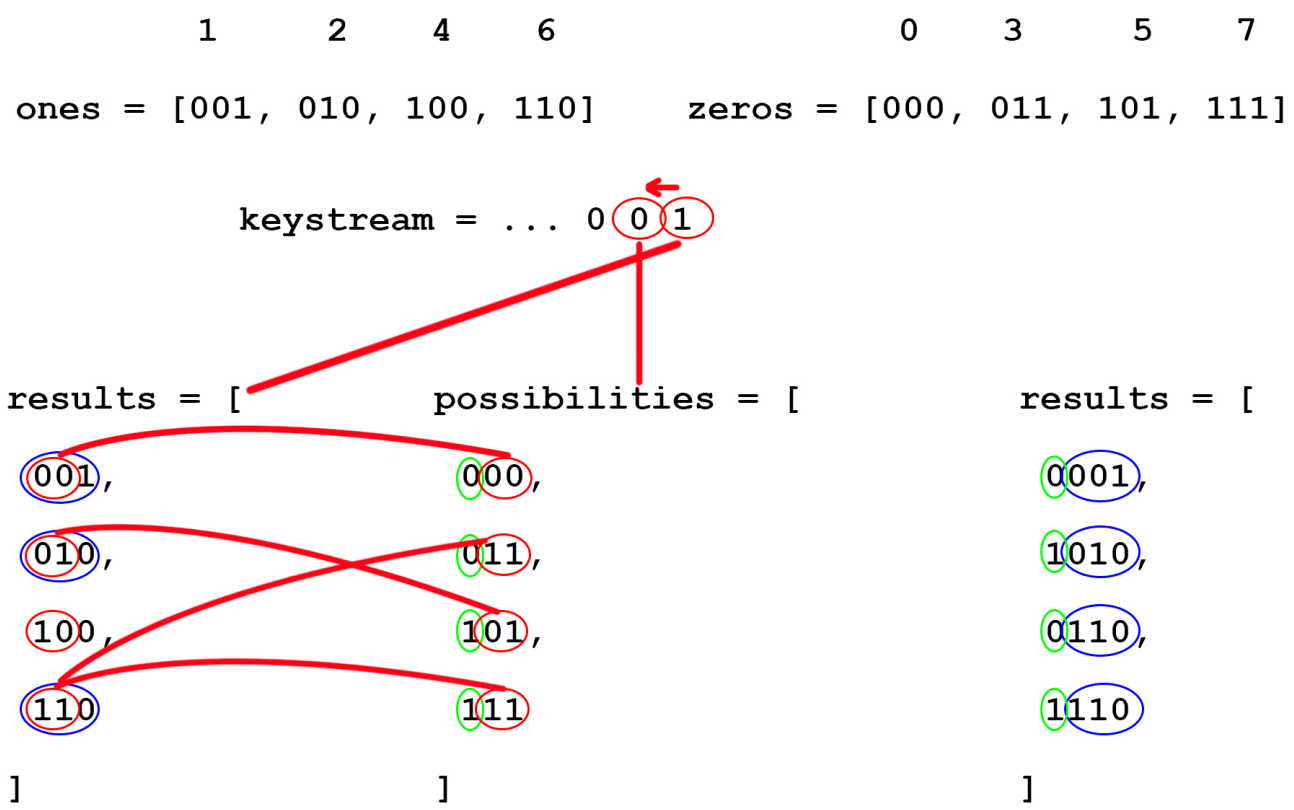

Figure 1: Princip řetězení indexů

Řádek 24 ze skriptu také informuje o tom, že *keystream* je inicializován provedení funkce `step` $N//2$*krát*, pro získání tajemství je tedy třeba funkci zavolat stejnou četností. Tímto je získáno cílové tajemství. K tomu byla sestrojena funkce `decrypt_file`, která umí dešifrovat jakékoliv soubory zašifrované touto šifrou.

Jediným háčkem byl fakt, že takto sestrojená funkce `reverse_step` mi nefungovala na zadaný *keystream*, avšak správnost jsem si ověřil vlastními daty, které jsem zašifroval funkcí `step` a následně dokázal dešifrovat pomocí `reverse_step`. Z toho vyplynulo, že zadaná data byla zašifrována trochu jiným algoritmem, než tím, jenž je obsahem skriptu `super_cipher.py.enc`. Následně vyšlo najevo, že byla v zadání chyba, a po změně pole `SUB` již skript fungoval správně.

3 SAT solver

Tato část projektu nechává analýzu a výpočet původního klíče z *keystreamu* na vyhodnocování logických klauzolí. Je zde třeba vytvořit funkci `solve`, která definuje bitový vektor (implementovaný v knihovně `z3`) x o velikosti klíče (plus dva bity, jelikož je klíč ve funkci `step` vždy rozšiřován). Pro vyhodnocování klauzolí je třeba definovat podmínky:

- hledáme ve finále takové x, které se bude rovnat po transformaci funkcí `step` známému *keystreamu*,
- hledané x musí být reprezentovatelné pomocí 256 bitů - solver totiž hledá $258 bitové$ hodnoty, avšak víme, že klíč je reálně reprezentovaný na 256 bitech,
- při analýze každé nalezené dílčí klauzole můžeme solveru dynamicky říct, že nemá nadále hledat stejné klauzole, které již dříve našel - zabrání se tak zbytečnému procházení možností, které již byly procházeny.

Jelikož je třeba funkci `step` volat $N//2 krát$, nabízí se rekurzivní řešení, kdy je funkce `solve` volána rekurzivně sama sebou až do hloubky zanoření $N//2$. V každém zanoření jsou vytvořeny výsledky, které jsou využity jako parametr k dalšímu zanoření. Průběh řešení tedy vypadá takto:

- ve funkci `solve` jsou v dané úrovni zanoření procházeny cyklem `while` všechny klauzole obsahující transformovaný bitový vektor funkcí `step`,
- pokud není úroveň zanoření rovna $N//2$, zavolá funkce `solve` rekurzivně sama sebe s výsledkem procházené klauzole jakožto parametrem,
- pokud v dané úrovni zanoření není pro aktuální klauzoli nalezen žádný výsledek, díky rekurzi se provede *backtracking* a pokračuje se jinou klauzolí o jednu úroveň zanoření výše,
- pokud je úroveň zanoření rovna $N//2$, dostáváme výsledek.

Jediným problémem je, že bitovým vektorem nelze indexovat pole `SUB`. Je tedy třeba patřičně upravit funkci `step`, a to tak, že jsou testovány poslední tři bity x (indexy do pole `SUB`) vůči sobě takovým způsobem, aby odpovídaly správné hodnotě pole `SUB`. Tento princip vysvětluje obrázek 2.

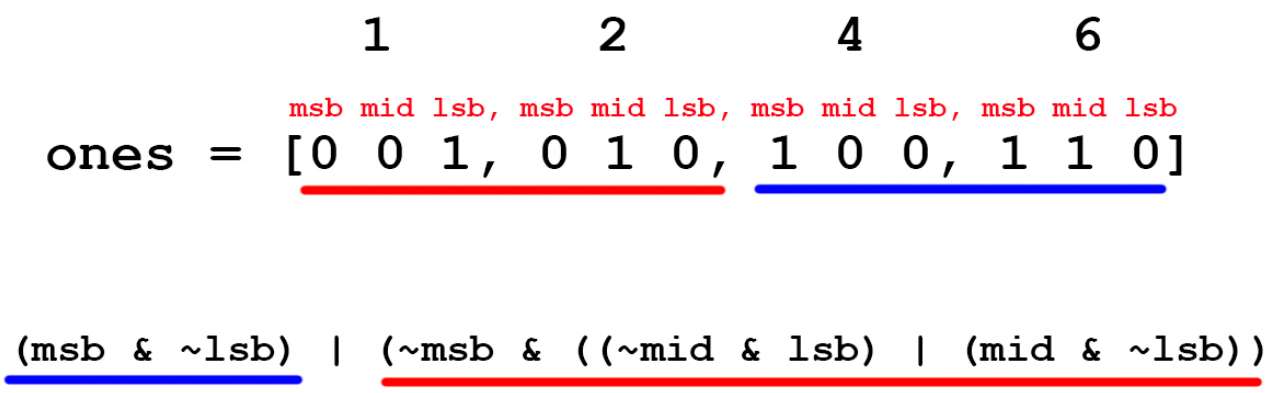

Figure 2: Princip úpravy přístupu do SUB

Tímto postupem je nalezen stejný klíč (tajemství), jako v ručním řešení, avšak s tím rozdílem, že *SAT* řešení trvá logicky výrazně déle, jelikož je procházeno mnohem více možností.

4 Závěr

Výstupem projektu jsou skripty `solution.py` (ruční řešení), `solution_sat.py` (*SAT* řešení) a `install.sh`, který pro jistotu instaluje *z3 solver*. Oba skripty potřebují jeden parametr, a to cestu k vstupním souborům. Řešení bylo také testováno na *Merlinovi* (avšak pouze ruční řešení, jelikož na *Merlinovi* není nainstalována knihovna *z3-solver*). Výstupem je tajemství

`KRY{xpawlu00-fad83f2e9be57ab}`

Figure 3: Nothing To See Here... Just An Eagle Flying Upside Down