Projekt 1 - prúdová šifra

Michal Ormoš
Kryptografia 2018/2019

21. marca 2019

Abstrakt

Táto správa popisuje riešenie prvého projektu z Kryptografie a to prelomenie neznámej prúdovej šifry. V úvode predstavuje zadanie a vysvetľuje princíp prúdovej šifry. V ďalšej časti popisuje ručné a SAT riešenie. V riešení sa odkazuje na zdrojové súbory v jazyku Python, ktorými bolo riešenie dosiahnuté a sú obsiahnuté v archíve. Popisovať budeme len zaujímavé a podstatne pasáže riešenia.

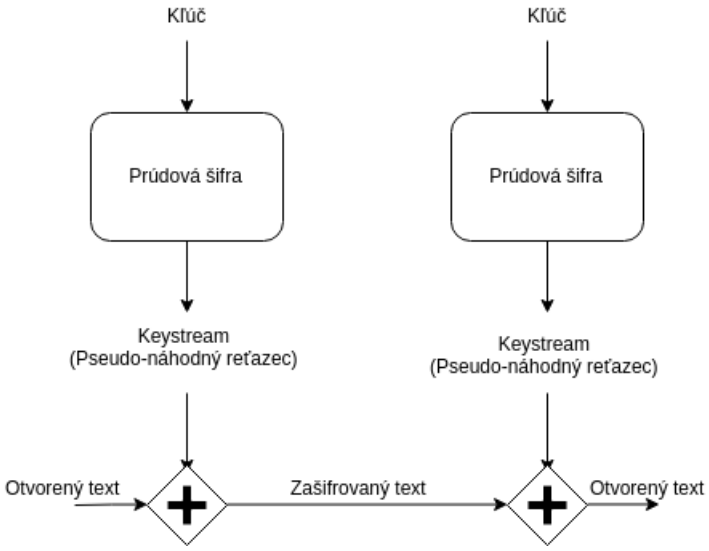
1 Úvod

Cieľom tohto projektu je prelomiť neznámu synchronnú prúdovú šifru, ktrorou boli zašifrované súbory. Bolo nám prezradené, že súbory boli zašifrované neznámou prúdovou šifrou s kľúčom o dĺžke 29 znakov, vo formáte `KRY{24 znakov ASCII textu}`.

2 Analýza

Predložený archív obsahoval štyri súbory, bez dodatočných informácií. Ich význam som odhadol nasledne:

- bis.txt - nešifrovaný plaintext [509B]
- bis.txt.enc - zašifrovaný plaintext textu bis.txt [512B]
- hint.gif.enc - zašifrovaný obrázok vo formáte gif [20970B]
- super_cipher.py.enc - zašifrovaný skript [714B]



Obr. 1: Fungovanie prúdovej šifry

Prúdová šifra je symetrický šifrovací algoritmus, ktorý používa kľúč pre zašifrovanie jednotlivých bitov textu, obrázok 1. Narozdiel od one-time-pad šifier, nevyužíva XOR len na zašifrovanie správy na náhodný string, ale XORuje správu na pseudonáhodnú sekvenciu bitov, ktorá sa nazýva keystream. Keystream je generovaný z malej sekvencie naozaj náhodných bitov. Dešifrovanie prebieha rovnakým spôsobom a to XORom zašifrovaného textu s keystreamom generovaným z rovnakej malej sekvencie náhodných bitov. [1]

3 Ručné riešenie

3.1 Keystream

Ako prvý krok som zvolil pokus o odhalenie keystreamu, použitého pre šifrovanie súboru `bis.txt`. Kedže som k súboru `bis.txt` mal odpovedajúci zašifrovaný text `bis.txt.enc`, použil som operáciu `xor` podľa vzorca. nasledovne:

$$\texttt{bis.txt} \oplus \texttt{bis.txt.enc} = \texttt{keystream}$$

Vo výsledku som získal onen požadovaný keystream, ktorý som následnu aplikoval na súbor `super_cypher.py.enc`:

$$\texttt{keystream} \oplus \texttt{super\_cypher.py.end} = \texttt{super\_cypher.py}$$

Tým som odhalil odšifrovaný plaint text súboru `super_cypher.py` Keystream bol kratší než obsah tohto zašifrovaného skriptu, a tak sa mi po darilo odkryť len časť tohto súboru. Naštastie táto časť obsahovala dôležité konštanty a algoritmus, ktorým bol tento keystream generovaný.

SUB = [0, 1, 1, 0, 1, 0, 1, 0]
N_B = 32

[1] https://www.globalspec.com/reference/81191/203279/2-6-stream-ciphers

```
N = 8 * N_B

# Next keystream
def step(x):
  x = (x & 1) << N+1 | x << 1 | x >> N-1
  y = 0
  for i in range(N):
    y |= SUB[(x >> i) & 7] << i
  return y

# Keystream init
keystr = int.from_bytes(args.key.encode(),'little')
print(keystr)
for i in range(N//2):
  keystr = step(keystr)
```

V dešifrovanom súbore som objavil dôležité konšanty `SUB`, `N_B` a algoritmus, ktorým bol keystream generovaný. Rovnakým spôsobom ako v predošlom kroku som tento keystream aplikoval aj na súbor `hint.gif.enc`. Ten mi odhalil druhú časť skriptu `super_cypher.py` a to riadky ktoré spracujú vstup. Postupu získania tohto časti tajmostva zodpovedá v zdrojových kódoch funkcia `xorKnownTexts` v súbore `solution.py`

3.2 Odhalenie celého obsahu super_cipher

Tento súbor mi neprišiel ako kompletný, preto som skúšal dalšie možnosti XOR-ovania medzi súbormi, ktoré mi boli poskytnuté. Pri XOR-e súboru `super_cipher.py` a `hint.gif.enc` som si všimol anomáliu a to další kód na konci tohto XOR, po jeho spojení stým čo som už získal som dosiahol plné znenie šifrovacieho skriptu:

```
plaintext = sys.stdin.buffer.read(N_B)
while plaintext:
  sys.stdout.buffer.write((
    int.from_bytes(plaintext,'little') ^ keystr
  ).to_bytes(N_B,'little'))
  keystr = step(keystr)
  plaintext = sys.stdin.buffer.read(N_B)
```

Súbor `hint.gif.enc` nepriniesol žiadne dalšie prínosné informácie, ktoré by som mohol použiť a ďalej som sa ním už nezaoberal. Postupu získania tohto časti tajmostva zodpovedá v zdrojových kódoch funkcia `xorKnownTexts` v súbore `solution.py`

Súbor `bis.txt` má zašifrovaný väčšiu veľkosť ako dešifrovaný, ale to vo výsledku nehraje rolu, kedže vidíme, podľa konštanty `N_B`, že keystream sa po 32B vždy opakuje

3.3 Reverzácia funkcie step

Po dôkladnom naštudovaní skriptu, kde som predpokladal, že žiadne dalšie riadky neobsahuje, mi bolo jasné, že chýbajúci parameter `key` môžem získať len reverzáciou funkcie `step`. Funkcia step sa skladá z dvoch zákadných častí a funguje nasledujúcim spôsobom:

Vykoná rotáciu:

- vezme vstup a vypočíta z neho výstup, vždy o veľkosti `32B`.
- vezme najpravejší bit vstupu a posunie sa o `N+1` bitov doľava.
- hodnota vstupu sa posunie o 1 bit doľava.
- hodnta vstupu sa posunie o `N-1` bitov doprava.
- prevedie sa logický `OR` troch predchadzajúcich krokov.

Vykoná substitúciu:

- vektor x sa posunie o i bitov doprava.
- vezmu sa tri najpravejšie hodnoty posunutého vektoru x.
- vezme sa hodnota vektoru `SUB`, pozícia ktorú určujú bity z predchádzajúceho kroku.
- prevedie sa posun získanej hodnoty o i bitov doľava.
- prevedie sa logický `OR` získanej hodnoty s hodnorou z predchádzajúceho kroku `i-1`.

Získaný keystream je teda výsledok v premennej y, my sa budeme snažiť získať reverzným algoritmom hodnotu x.

Najtažším krokom bolo reverzovať substitúciu, ktorá sa vykonáva pre každý bit. Vstupom substitúcie je aktuálny bit a dva predchádzajúce bity. Spolu teda tri bity, indexujúce čísla od 0 do 7. Tento index sa následne používa ako index prístupu do poľa `SUB`. Indexovaným prvkom je nahradený bit aktuálnej pozície. Najväčší problém spočíval v tom, že najvyššie dva bity nie su známe a je teda potrebné ich uhádnuť. Ako najlepšia možnosť sa javí skúšať všetky možnosti. Ako podmienka nám poslúži fakt, že spodné dva bity sa musia rovnať vrchným dvom bitom.

```
if (x & 3) == (x >> 256)
```

3.4 Záver ručného riešenia

Vo výlsedku sme získali jedinečný kľúč:

KRY{xormos00-3aa5d40d35f624b}

V tomto riešení som získal, pochopil a aplikoval reverzáciu funkcie step na získanie kľúča.

Výsledok som si overil tak, že som pomocou skritpu `super_cypher.py` a mnou získaným kľučom aplikoval na pôvodný súbor `bis.txt.enc` a získal rovnaké znenie suboru `bis.txt`.

Riešeniu zodpovedajú funckie `brakeCipherKey` a `stepReversed` v súbore `solution.py` Časová náročnosť bola testovaná na školskom servery merlin a v priemere prelomenie zabralo 6 sekúnd.

4 SAT riešenie

V tomto riešení som už využil poznatky z ručného riešenia a to presne, znenie algoritmu pomocou ktorého bolo šifrovanie vykonané a fakt, že pole SUB je vo výsledku iné ako pôvodne algoritmus preznetoval.

Riešenie s pomocou SAT solvera je podobné ručnému riešenie, okrem substitúcie, ktorú tu budeme riešiť pomocou SAT solvera. Ten pozostáva z nasledujúcich krokov:

1. Prevod aktuálneho kľúča na formulu v boolovej algebre. Každému jednotlivému bitu keystreamu je vytvorená jedna premenná. Z toho je následne vytvorená formula v konjuktívnej normálnej forme, kde jej každý člen je podfromula a popisuje jednotlivé bity nasledovne, obdobne ako pri ručnom riešení s opraveným poľom `SUB`.

   - bit je nulový, existujú štyri možnosti ako môžu vyzerať dva predchádzajúce bity
   - bit je jednotkový, existujú rovnako štyri možnosti ako môžu vyzerať dva predchádzajúce bity

   Jedná sa o štyri podformule spojené logickým `OR`.

   ```
   (-v & -v1 & -v2) | (v & (v1 | v2)) | (v & -v1 & -v2) | (-v & (v1 | v2))
   ```

2. Ohodnotenie premených, v spolupráci s knižnicou `satispy`. Vstupom je formula vo formáte boolovej algebry a výstupom je binárna hodnota toho či je formula riešiteľná (True), alebo nie je riešitelná (False)

3. Konverzia ohodnoteným premenných späť na kľúč. Prechádzame bit po bite.

   - bit je jednotkový ak je jeho i-tá premenná ohodnotená ako riešitelná
   - bit je nulový, ak je i-tá premenná ohodnotená inak

4.1 Záver SAT riešenia

Rovnako ako v predchádzajúcom riešení, po aplikácii SAT riešenia N/2 krát som získal výsledok a to kľúč akým boli súbory zašifrované. Tento kľuč sa zhoduje s kľučom z ručného riešenia, teda ho považuje za správny:

KRY{xormos00-3aa5d40d35f624b}

Riešenie pomocou SAT solveru som považoval, že zaujímave vďaka nutnosti využiť formuly boolovej algerby.

A Použitie programu

Skript `solution.py` nepotrebuje žiadne dalšie dodatočne inštalácie. Spúšta sa s jediným argumentom, ktorým je cesta k zdrojovým súborom popisaným v kapitole Úvod.
Príklad spustenia:

```
python3 solution.py in/
```

V prípade nezadania vstupného argumentu očakáva súbory v zložke `in/`.

Skript `solution_sat.py` potrebuje doinštalovať rozšírenia pomocou skriptu `install.sh`. Jeho spustenie je obdobné ako pri skripte `solution.py` a to pomocou jediného vstupného argumentu, cesta k vstupným súborom.
Príklad spustenia:

```
python3 solution_sat.py in/
```

V prípade nezadania vstupného argumentu, skript očakáva súbory v zložke in/ vo formáte v akom boli poskytnuté v zadaní.