VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projekt č. 1

Projekt do předmětu Kryptografie (KRY)

Bc. Klára Nečasová (xnecas24)

7. dubna 2019

Úvod

K dispozici bylo několik souborů, které byly šifrovány neznámou synchronní proudovou šifrou [1]. Cílem projektu bylo zjistit tajemství, které bylo ve formátu `KRY{24 znaků ASCII textu}`.

Zjištění části keystreamu a dešifrování souborů

Mezi dostupnými soubory byly soubory `bis.txt` a `bis.txt.enc`. Z názvů bylo možné odvodit, že první soubor by mohl obsahovat plaintext a druhý soubor odpovídající ciphertext. Dále byl k dispozici soubor `super_cipher.py.enc`, jehož název napovídal, že by mohl obsahovat šifrovací algoritmus, který byl použit pro šifrování souborů. Pro zjištění keystreamu byl proveden XOR plaintextu a ciphertextu. Získaný keystream byl poté využit pro dešifrování souboru s šifrovacím algoritmem. Pomocí operace XOR se však podařilo dešifrovat pouze část keystreamu. Dešifrovaný úsek kódu zahrnoval funkci `step()`, která je uvedena níže, pomocí které byl keystream vytvořen.

```
# Next keystream
def step(x):
    SUB[0,1,1,0,1,0,1,0]
    ...
    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
```

S využitím funkce `step()` bylo možné postupně generovat další bloky keystreamu a dešifrovat tak libovolný text, zašifrovaný pomocí tohoto keystreamu. Proto bylo také možné dešifrovat obsah celého souboru `super_cipher.py.enc`. Zbývalo tedy zjistit parametr `key`, na základě něhož byl keystream vygenerován, jak naznačuje níže uvedený úsek kódu.

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

Popis funkce `step()`

Vstupním parametrem funkce `step()` je keystream (`x`). Po analyzování funkce bylo zjištěno, že se nejdříve prováděla rotace a poté substituce. V rámci **rotace** se rotoval keystream o délce N bitů o jeden bit doleva. Pomocí operace OR byl poté keystream rozšířen o další dva bity. Hodnota MSB rozšířeného keystreamu odpovídala hodnotě LSB původního keystreamu a hodnota LSB rozšířeného keystreamu odpovídala hodnotě MSB původního keystreamu. Díky rozšíření bylo zajištěno, že si musí dva nejlevější a dva nejpravější bity odpovídat, což bylo slabinou šifrovacího algoritmu. **Substituce** byla provedena pomocí pole `SUB`. Index pole byl určen třemi bity, které odpovídaly třem nejpravějším bitům rozšířeného vstupu, získaným po posunutí o i bitů doprava, kde $i = 0, \ldots, N-1$. Na daném indexu byl uložen bit 0 nebo 1, z čehož plyne, že se prováděla substituce tří bitů jedním bitem. Po postupném aplikování substituce byl získán nový keystream.

Popis reverzace funkce `step()`

Parametr `key` bylo možné zjistit poté, co byla na 32B část keystreamu 128 krát aplikována reverzovaná funkce `step()`. Reverzace mohla být provedena ručně, nebo s využitím SAT solveru.

Ruční řešení

Nejdříve bylo potřeba provést reverzaci substituce, kdy byl jeden bit nahrazen třemi bity. Nebylo však možné jednoznačně určit, jakou trojicí má být daný bit nahrazen. Bitu 0 odpovídají trojice bitů 000, 011, 101 a 111, bitu 1 trojice 001, 010, 100 a 110 (viz tabulka 1). Vstupní keystream byl zpracováván bit po bitu (zprava doleva), přičemž byla vždy uvažována dvojice po sobě jdoucích bitů. Vstupem mohl být například keystream `...0001`, uvažována byla nejdříve dvojice bitů `01`. Na základě hodnoty daného bitu byly uvažovány možné trojice bitů, kterými mohl být bit nahrazen (viz sloupce 0 a 1 v tabulce 2). Některé trojice bitů mohly být vyřazeny, protože nesplňovaly podmínku, která požadovala, aby dva nejlevější bity dané trojice bitů odpovídaly dvěma nejpravějším bitům následující trojice bitů. Pro názornost jsou

bity trojic bitů, které splňují uvedenou podmínku, označeny zelenou barvou. Trojice vyhovující podmínce byly rozšířeny o jeden bit zleva, který odpovídal nejlevějšímu bitu následující trojice bitů (viz sloupec key part (bits 0...3) v tabulce 2). Výstupem reverzované funkce byly čtyři klíče (viz tabulka 3)). Dále bylo možné ignorovat klíče, jejichž dva nejpravější a dva nejlevější bity se neshodovaly, což je v tabulce 3 znázorněno červenými obdélníky. Nakonec zbyl jen jeden klíč, jehož bity se shodovaly (viz zelené obdélníky v tabulce 3). Pro získání zcela korektního klíče (parametr `key`) bylo potřeba zanedbat nejlevější a nejpravější bit klíče, jelikož byl těmito bity rozšířen (viz přeškrtnuté bity klíče 1 v tabulce 3)).

dec	0	1	2	3	4	5	6	7
bin	000	001	010	011	100	101	110	111
val	0	1	1	0	1	0	1	0

Tabulka 1: Pole `SUB`

0	**1**	**key part (bits 0...3)**
000	001	0001
011	010	0110
101	100	1010
111	110	1110

Tabulka 2: Možné části klíče na základě dvou po sobě jdoucích bitů keystreamu

key	**key part (bits 0...3)**	**key part (bits 4...N-5)**	**key part (bits N-4...N-1)**
1	0001	...	1000
2	0110	...	0111
3	1010	...	1011
4	1110	...	0101

Tabulka 3: Možné výsledné klíče

Pro ověření, že bylo zjištěné správné tajemství, bylo možné využít pokyny ze zadání projektu, kde byl specifikován formát tajemství, případně bylo možné dešifrovat obsah souboru `hint.gif.enc` (viz obrázek 1).

```
for file in *.*
do ./super_cipher.py "KRY(                    )" < $file > $file.enc
done
```

Obrázek 1: Obsah dešifrovaného souboru `hint.gif.enc`

SAT solver

Pro reverzaci substituce lze kromě ručního řešení také využít SAT[1] solver [3], [2]. V projektu byla využita knihovna z3[2]. Nejdříve bylo nutné vytvořit N různých proměnných typu `boolean`. Dále byl zpracován vstupní keystream, a to po jednotlivých bitech. Dle hodnoty aktuálně zpracovávaného bitu bylo potřeba brát vždy v úvahu čtyři různé možnosti, jak

[1] SAT – Boolean satisfiability problem
[2] https://github.com/Z3Prover

mohl být daný bit substituován. SAT solver vyžadoval zápis zmíněných možností v konjunktivní normální formě (KNF). Pro bit 0 byly definovány formule (1), (2) a pro bit 1 formule (3), (4).

$$\neg var0 \wedge \neg var1 \wedge \neg var2 \tag{1}$$
$$var0 \wedge (var1 \vee var2) \tag{2}$$
$$var0 \wedge \neg var1 \wedge \neg var2 \tag{3}$$
$$\neg var0 \wedge (var1 \vee var2) \tag{4}$$

Vytvořené formule byly postupně vloženy do SAT solveru a nakonec vyhodnoceny. Pokud je výsledkem `sat` (satisfiable), řešení je nalezeno. Výsledek `unsat` (unsatisfiable) značí, že řešení neexistuje. Řešení je označováno jako model pro množinu uvažovaných formulí. Poté bylo nutné ohodnocené proměnné převést na keystream, což bylo realizováno tak, že pokud byla proměnné přiřazena hodnota `True`, tak byl bit odpovídající indexu proměnné nastaven na 1, v opačném případě se neprovedla žádná operace, bit tedy zůstal nastaven na 0. Nakonec byla provedena reverzace rotace, kdy bylo potřeba orotovat získaný klíč (parametr `key`) o 1 bit doprava.

Závěr

Cílem projektu bylo prolomit synchronní proudovou šifru. Byly vytvořeny soubory `solution.py` a `solution_sat.py`, jejichž parametrem je cesta k adresáři, kde jsou uloženy soubory `bis.txt`, `bis.txt.enc`, `hint.gif.enc`, `super_cipher.py.enc`. Programy vypíší hledané tajemství na standardní výstup. Pomocí ručního řešení i s využitím SAT solveru bylo získáno tajemství `KRY{xnecas24-105f261391fc98a}`. Časová náročnost ručního řešení je značně menší než časová náročnost řešení pomocí SAT solveru.

Literatura

[1] P. Hanáček. *Symetrická kryptografue*. cit. 5. dubna 2019. 2007.

[2] Frank Van Harmelen, Vladimir Lifschitz a Bruce Porter. *Handbook of knowledge representation*. Sv. 1. Elsevier, 2008.

[3] Dennis Yurichev. *SAT/SMT by example*. 2019. URL: <https://yurichev.com/writings/SAT_SMT_by_example.pdf>.