KRY-projekt 1

Sergey Panov xpanov00

13. dubna 2018

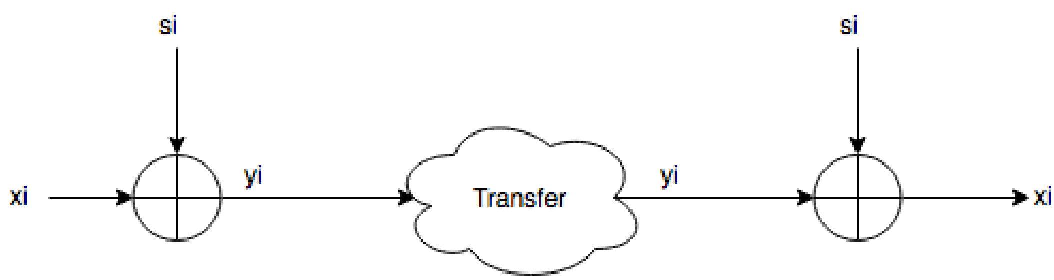
1 Zadání

Každý student dostal několik souborů, zašifrovaných neznámou synchronní proudovou šifrou. Cílem projektů bylo zjistit tajemství ve tvaru `KRY{24-znaků ASCII textu}`.

2 Analýza

Obdržený archiv obsahuje soubory `bis.txt`, `bis.txt.enc`, `hint.gif.enc`, `super_cipher.py.enc`. Soubory s příponou `.enc` jsou zašifrované stejnou proudovou šifrou.

Proudová šifra šifruje každý bit vstupního textu zvlášť. Pro šifrování se vezme bit vstupního textu, patřičný bit klíče a provede se logický `xor`. Dešifrování se provede stejným způsobem, až na to že se místo bitu otevřeného textu vezme bit zašifrovaného textu. Proces šifrování a dešifrování je znázorněn na obrázku 1.



Pro $x_i, y_i, s_i \in \{0, 1\}$
Šifrování: $y_i \equiv x_i \oplus s_i$
Dešifrování: $x_i \equiv y_i \oplus s_i$

Obrázek 1: Princip šifrování a dešifrování proudové šifry

3 Řešení

3.1 Generování klíče

Nejprve se provedlo odhalení klíče, použitého pro šifrováni souboru `bis.txt`. Pro odhalení klíče se provedla operace `xor` šifrovaného textu z `bis.txt.enc` a otevřeného z `bis.txt`. Výsledkem logického `xor`, mezi těmito dvěma soubory, je klíč jelikož platí:

$$y_i = x_i \oplus s_i$$

$$x_i = y_i \oplus s_i$$

$$x_i \oplus y_i = x_i \oplus x_i \oplus s_i = s_i$$

Po zjištění klíče se provedlo dešifrování části souboru `super_cipher.py.enc`. Z dešifrovaného souboru `super_cipher.py` se zjistil způsob generování počátečního klíče a generování následujícího klíče. Generování klíče zajistí funkce `step(x)`. Funkce je uvedená ve výpisu 1

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

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
```

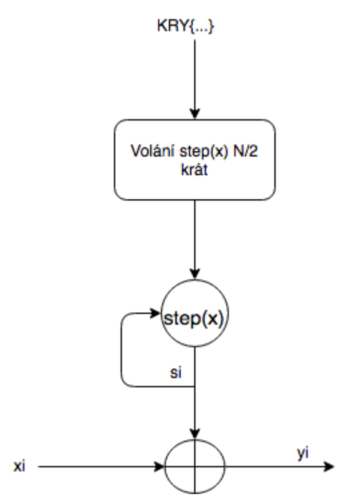
Listing 1: Generování dalšího klíče

tedy funkce `step(x)` funguje následujícím způsobem:

1. vezme vstup x, provede se počáteční modifikace vstupu:
   (a) vezme se nejpravější bit vstupu a posune se o $N+1$ bitů doleva
   (b) hodnota vstupu se posune o 1 bit doleva
   (c) hodnota vstupu se posune o $N-1$ bitů doprava
   (d) provede se logický `or` třech předchozích kroků
2. další posloupnost kroků se provede N krát:
   (a) vektor x se posune o i bitů doprava
   (b) vezmou se tři nejpravější hodnoty posunutého vektoru x
   (c) vezme se hodnota vektoru SUB, pozici které určují bity z předchozího kroku
   (d) provede se posun získané hodnoty o i bitů doleva
   (e) provede se logický `or` získané hodnoty s hodnotou z předchozího kroku $i-1$

   počáteční hodnota i je 0, s každou iteraci se hodnota i inkrementuje, tedy postupně nabývá hodnot $[0 \ldots N-1]$.

Generování počátečního klíče provede tak že se na neznáme tajemství aplikuje funkce `step(x)` $N/2$ krát. Generování dalšího klíče se provádí aplikací funkce `step(x)` na předchozí hodnotu klíče. Princip šifrování je uveden na obrázku 2



Obrázek 2: Princip šifrováním použity v rámce projektu

3.2 Odhalení tajemství

3.2.1 Ruční řešení

Pro odhalení tajemství bylo zapotřebí napsat inverzní funkci k funkce `step(x)`. Máme-li inverzní funkcí k funkce `step(x)` a máme-li počáteční klíč $S = s_{n-1} \dots s_0$, aplikujeme inverzní funkcí `N/2` krát na klíč a získáme tajemství.

Princip inverzní funkce si vysvětlíme na zjednodušeném příkladě. Uvažme funkci `step(x)`, která ale neprovádí počáteční modifikaci (případ s modifikaci si vysvětlíme později), tedy chybí krok
$x = (x \& 1) << N + 1 | x << 1 | x >> N - 1$, dále předpokládejme že $N = 8$, $X = x_7x_6x_5x_4x_3x_2x_1x_0$ a $Y = y_7y_6y_5y_4y_3y_2y_1y_0$. Zvolme binární hodnotu $X = 11001010$. Aplikujeme zjednodušenou verzi funkce `step(x)`, tedy bez počáteční modifikace vstupu, na X:

0. $X = 11001010$; $Y = 00000000 \,|\, 1 << 0 = 00000001 = 1$
1. $X = 11001010$; $Y = 00000001 \,|\, 0 << 1 = 00000001 = 1$
2. $X = 11001010$; $Y = 00000001 \,|\, 1 << 2 = 00000101 = 5$
3. $X = 11001010$; $Y = 00000101 \,|\, 1 << 3 = 00001101 = 13$
4. $X = 11001010$; $Y = 00001101 \,|\, 1 << 4 = 00011101 = 29$
5. $X = 11001010$; $Y = 00011101 \,|\, 1 << 5 = 00111101 = 61$
6. $X = 11001010$; $Y = 00111101 \,|\, 0 << 6 = 00111101 = 61$
7. $X = 11001010$; $Y = 00111101 \,|\, 1 << 7 = 10111101 = 189$

index kroku určuje hodnotu i z výpisu 1. Obarvené bity určují index prvku vektoru SUB (vektor se indexuje zleva doprava). Po provedení funkce `step(x)`, hodnota Y udává klíč, který se následně použije pro šifrování textu.

Pokud máme k dispozici Y, je možné vypočítat původní hodnotu X a to použitím následujícího algoritmu:

```
def calculateX(y, x):
  x = x << N
  for i in range(N-1, -1, -1):
    sub_value = (y & 1 << i) >> i
    if sub_value != SUB[(7 << i & x) >> i]:
        x |= 1 << i
  return x
```

Listing 2: Inverzní funkce

v tomto algoritmu `y` je klíč, získaný po aplikaci funkce `step(x)`, význam hodnoty `x` si vysvětlíme dále, prozatím považujme ji za nulovou.

Nechť $Y = 10111101 = 189$, budeme chtít vypočítat hodnotu X, ze které byla získaná hodnota Y, použijeme postup z výpisu 2:

7. $Y = 10111101$; $X = 00x_7x_6x_5x_4x_3x_2x_1x_0$; hodnota x_7 je buď 0 anebo 1, víme že hodnota bitu $y_7 = 1$, tedy $SUB[00x_7] = 1 = SUB[001]$, tedy $x_7 = 1$
6. $Y = 10111101$; $X = 001x_6x_5x_4x_3x_2x_1x_0$; hodnota x_6 je buď 0 anebo 1, víme že hodnota bitu $y_6 = 0$, tedy $SUB[01x_6] = 0 = SUB[011]$, tedy $x_6 = 1$
5. $Y = 10111101$; $X = 0011x_5x_4x_3x_2x_1x_0$; hodnota x_5 je buď 0 anebo 1, víme že hodnota bitu $y_5 = 1$, tedy $SUB[11x_5] = 1 = SUB[110]$, tedy $x_5 = 0$
4. $Y = 10111101$; $X = 00110x_4x_3x_2x_1x_0$; hodnota x_4 je buď 0 anebo 1, víme že hodnota bitu $y_4 = 1$, tedy $SUB[10x_4] = 1 = SUB[100]$, tedy $x_4 = 0$
3. $Y = 10111101$; $X = 001110x_3x_2x_1x_0$; hodnota x_3 je buď 0 anebo 1, víme že hodnota bitu $y_3 = 1$, tedy $SUB[10x_3] = 1 = SUB[100]$, tedy $x_3 = 0$

2. $Y = 10111101$; $X = 0011101x_2x_1x_0$; hodnota x_2 je buď 0 anebo 1, víme že hodnota bitu $y_2 = 1$, tedy $SUB[01x_2] = 1 = SUB[010]$, tedy $x_2 = 0$

1. $Y = 10111101$; $X = 00111010x_1x_0$; hodnota x_1 je buď 0 anebo 1, víme že hodnota bitu $y_1 = 0$, tedy $SUB[10x_1] = 0 = SUB[101]$, tedy $x_1 = 1$

0. $Y = 10111101$; $X = 001110101x_0$; hodnota x_0 je buď 0 anebo 1, víme že hodnota bitu $y_0 = 1$, tedy $SUB[01x_0] = 1 = SUB[010]$, tedy $x_0 = 0$; tedy $X = 0011001010$, zanedbáme 2 nejlevější bity a máme $X = 11001010$.

Jako i v předchozím případě index kroku reprezentuje hodnotu indexu `i` z výpisu 2. V 7 kroku se první dva bity jsou neznáme a musejí se "předpovědět", tedy funkce `calculateX` se musí zavolat 4 krát po každé s jinou hodnotou x, x může nabývat hodnot v rozmezí $[0 \ldots 3]$. "Předpověď"hodnoty x se musí provést kvůli ztratě dvou prvních bitů.

Teď uvažme případ s počáteční modifikaci vstupu. Posledním krokem před vlastním návratem hodnoty x, odvozené z y, je inverzní modifikace x. Inverzní modifikace se musí provést kvůli počáteční modifikaci ve funkci `step(x)` a provede se aplikaci $(x\&1) << N-1|(x\&max\_N\_bit\_val$[1]$) >> 1$ na x. Příklad modifikaci vstupu funkce `step(x)` a inverzní modifikaci:

Modifikujeme $X = 11001010$ aplikaci $(x\&1) << N+1|x << 1|x >> N-1$:

1. $(x\&1) << N+1 = 000000000$
2. $x << 1 = 110010100$
3. $x >> N-1 = 000000001$

$000000000|110010100|000000001 = 110010101$

Provedeme inverzní modifikací $X = 110010101$ použitím $(x\&1) << N-1|(x\&max\_8\_bit\_val) >> 1$

1. $(x\&1) << N-1 = 010000000$
2. $(x\&11111111) >> 1 = 001001010$

$010000000|001001010 = 011001010$, zanedbáme nejlevější bit a získáme původní $X = 11001010$

Shrnutí metody: Funkce `step(x)` provádí počáteční modifikaci vstupu X a následně provádí N kroků získávaní nového klíče Y. Inverzní funkce na vstup vezme získaný klíč Y, provede N kroků získaní původní hodnoty klíče X a před vrácením klíče se provede inverzní modifikace X. Vzhledem ke ztratě dvou bitů, funkce `calculateX(y, x)` se musí zavolat 4-krát pro 4 různé hodnoty x a minimálně jeden ze získaných výsledků bude hledanou hodnotou původního X. Celý ten cyklus se musí provést $N/2$ krát. Výsledkem je původní tajemství. Ve případě autora tajemstvím bylo `KRY{xpanov00-4b11d11e32df6ac}`

3.2.2 Řešení SAT-solverem

Jinou možnosti získaní klíče je použití SAT-solveru. SAT-problém lze formulovat takto: Je dána množina proměnných $V = \{v_1, v_2, \ldots v_n\}$ a množina klauzulí nad V. Je tato množina klauzulí splnitelná? SAT je NP-úplný problém, pro řešení SAT problému lze použit algoritmus DPLL.

Vysvětlíme si princip odvození klíče použitím SAT solveru na příkladu. Zase uvažme zjednodušenou verzi funkce `step(x)`, kde se neprovádí modifikace vstupu.

Předpokládejme že $N = 4$, $Y = 1101$, víme že nulové hodnoty vektoru SUB jsou na pozicích $ZERO = \{0, 3, 5, 7\}$, jedničkové na $ONE = \{1, 2, 4, 6\}$ a budeme chtít najít $X = x_3x_2x_1x_0$. Postupujeme následovně:

3. $Y = 1101$; $X = x_5x_4x_32_2x_1x_0$, víme že $SUB[x_5x_4x_3] = y_3 = 1$, pak index určený $x_5x_4x_3$ musí být ve množině ONE; sestavíme formuli $(!x_5 \wedge !x_4 \wedge x_3) \vee (!x_5 \wedge x_4 \wedge !x_3) \vee (x_5 \wedge !x_4 \wedge !x_3) \vee (x_5 \wedge x_4 \wedge !x_3)$, po aplikace dvojí-negace a De Morganoveho zákonu $!\big((x_5 \vee x_4 \vee !x_3) \wedge (x_5 \vee !x_4 \vee x_3) \wedge (!x_5 \vee x_4 \vee x_3) \wedge (!x_5 \vee !x_4 \vee x_3)\big) = \phi_3$

2. $Y = 1101$; $X = x_5x_4x_3x_2x_1x_0$, víme že $SUB[x_4x_3x_2] = y_2 = 1$, pak index určený $x_4x_3x_2$ musí být ve množině ONE; sestavíme formuli $(!x_4 \wedge !x_3 \wedge x_2) \vee (!x_4 \wedge x_3 \wedge !x_2) \vee (x_4 \wedge !x_3 \wedge !x_2) \vee (x_4 \wedge x_3 \wedge !x_2)$, po aplikace dvojí-negace a De Morganoveho zákonu $!\big((x_4 \vee x_3 \vee !x_2) \wedge (x_4 \vee !x_3 \vee x_2) \wedge (!x_4 \vee x_3 \vee x_2) \wedge (!x_4 \vee !x_3 \vee x_2)\big) = \phi_2$

1. $Y = 1101$; $X = x_5x_4x_3x_2x_1x_0$, víme že $SUB[x_3x_2x_1] = y_1 = 0$, pak index určený $x_3x_2x_1$ musí být ve množině $ZERO$; sestavíme formuli $(!x_3 \wedge !x_2 \wedge !x_1) \vee (!x_3 \wedge x_2 \wedge x_1) \vee (x_3 \wedge !x_2 \wedge x_1) \vee (x_3 \wedge x_2 \wedge x_1)$, po aplikace dvojí-negace a De Morganoveho zákonu $!\big((x_3 \vee x_2 \vee x_1) \wedge (x_3 \vee !x_2 \vee !x_1) \wedge (!x_3 \vee x_2 \vee !x_1) \wedge (!x_3 \vee !x_2 \vee !x_1)\big) = \phi_1$

[1] $max\_N\_bit\_val$ je maximální číslo, které lze zobrazit na N bitech, tedy $max\_8\_bit\_val = 1111\,1111$

0. $Y = 1101$; $X = x_5x_4x_3x_2x_1x_0$, víme že $SUB[x_2x_1x_0] = y_0 = 1$, pak index určený $x_2x_1x_0$ musí být ve množině ONE; sestavíme formuli $(!x_2 \wedge !x_1 \wedge x_0) \vee (!x_2 \wedge x_1 \wedge !x_0) \vee (x_2 \wedge !x_1 \wedge !x_0) \vee (x_2 \wedge x_1 \wedge !x_0)$, po aplikace dvojí-negace a De Morganoveho zákonu $!\big((x_2 \vee x_1 \vee !x_0) \wedge (x_2 \vee !x_1 \vee x_0) \wedge (!x_2 \vee x_1 \vee x_0) \wedge (!x_2 \vee !x_1 \vee x_0)\big) = \phi_0$

Řešením logické formule $F = \phi_0 \wedge \phi_1 \wedge \phi_2 \wedge \phi_3$ je vektor $X = x_5x_4x_3x_2x_1x_0$, zanedbáním bitů x_5x_4 získáme předcházející klíč. Jako i při řešení základním způsobem se hodnoty bitů x_5x_4 musejí "předpovědět"a po získaní X se musí provést inverzní modifikace.

Každý term se uchovává ve tvaru:

```
{"xi": {"is_negated": Bool, "value": Bool|None}}
```

Listing 3: Tvar termu

kde x_i je "název"termu, *is_negated* je booleovská hodnota určující jestli je term znegován, *value* určuje hodnotu termu (1, 0 nebo None). Každá klauzule je seznamem termů, spojených logickým `or`, každá pod-formule je seznamem klauzulí spojených logickým `and`. Výsledná formule je seznamem pod-formulí, získaných v jednotlivých krocích, spojených logickým `and`.

Zase po aplikaci popsaného řešení $N/2$ krát se získá klíč `KRY{xpanov00-4b11d11e32df6ac}`.

4 Závěr

Závěrem projektu je odhalení tajemství dvěma různými způsoby. Pro odhalení tajemství SAT-solverem byla implementovaní vlastní zjednodušená verze algoritmu DPLL pracující s formulemi ve tvaru $!\big(((!)p \vee (!)q \vee (!)r) \wedge ((!)p \vee (!)q \vee (!)r) \wedge ((!)p \vee (!)q \vee (!)r) \wedge ((!)p \vee (!)q \vee (!)r)\big)$. Odhalení tajemství použitím vlastní implementaci SAT-solveru je pomalejší než odhalení základním způsobem, vzhledem k složitosti struktury pro uchování formuli a složitosti algoritmu DPLL. Nicméně každý přístup vedl k úspěšnému nalezení tajemství, které, v případě autora, bylo `KRY{xpanov00-4b11d11e32df6ac}`.