Symetrické proudové šifry

Milan Radojčić

SSPŠaG – KBB

20. ledna 2026

Hlavní body

Úvod

One-time pad

RC4

ChaCha20

Hlavní body

Klasifikace symetrických šifer

- **Symetrické šifry** používají **stejný** klíč pro šifrování i dešifrování.
- **Blokové šifry** dělí data na bloky a každý blok šifrují **stejným** klíčem.
- **Proudové šifry** dělí data na bloky a každý blok šifrují **jiným** klíčem.

Úvod

- Bylo:
  - Diffieho-Hellmanova výměna klíčů – 1976
  - RSA – 1977
- Dnes bude:
  - One-time pad – 1882/1917
  - RC4 – 1987
  - ChaCha20 – 2008

Hlavní body

One-time pad (OTP)

- **Proudová** šifra.
- Matematicky absolutně bezpečná.[1]
- V praxi nepoužitelná: nutnost náhodného klíče stejně velkého jako plaintext.
- Jeden klíč také nemůžeme použít dvakrát.

[1] Claude Shannon – Communication Theory of Secrecy Systems (1949), Kapitola 10: Perfect Secrecy

One-time pad (OTP)

Šifrování

klíč	0	1	1	0	0	1	0	0	0	0
plaintext	1	1	1	0	0	0	0	0	1	1
ciphertext	1	0	0	0	0	1	0	0	1	1

Dešifrování

klíč	0	1	1	0	0	1	0	0	0	0
ciphertext	1	0	0	0	0	1	0	0	1	1
plaintext	1	1	1	0	0	0	0	0	1	1

XOR

- je **komutativní** a **asociativní**
- $A \oplus A = 0$ a $A \oplus 0 = A$

Proč nemůžeme zašifrovat dvě zprávy stejným klíčem?

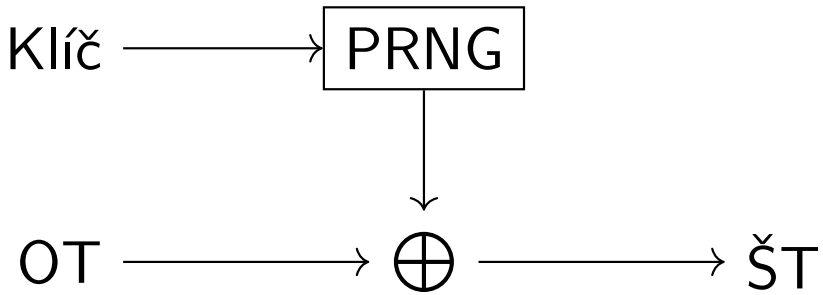
Vezměme $ct_1 = pt_1 \oplus key$ a $ct_2 = pt_2 \oplus key$.Potom:

$$ct_1 \oplus ct_2 = (pt_1 \oplus key) \oplus (pt_2 \oplus key) = (pt_1 \oplus pt_2) \oplus (key \oplus key) = (pt_1 \oplus pt_2) \oplus 0 = pt_1 \oplus pt_2$$

One-time pad prakticky

- Potřebujeme nějak slevit z nereálných požadavků na klíč:
  1. Můžeme vzít menší **klíč** a ten **opakovat**.
  2. Můžeme menším klíčem inicializovat **generátor náhodných čísel**, pomocí kterého budeme generovat posloupnost bitů.

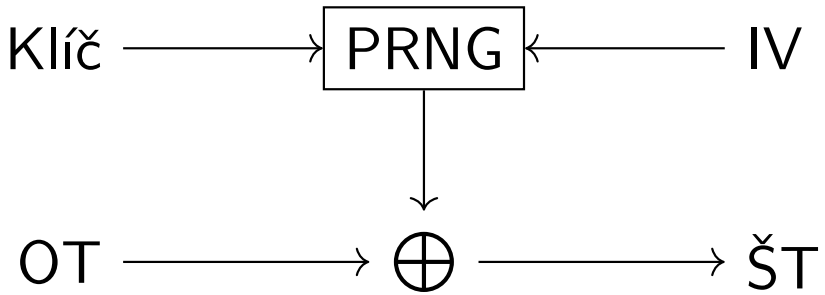
One-time pad prakticky



One-time pad prakticky

- Existují ale pořád problémy: stejný klíč vede ke stejné sekvenci PRNG.
- Jako vstup do PRNG přidáme **inicializační vektor** (IV).
  - Někdy se také používá název **nonce** (number used once).
- Daný inicializační vektor použijeme vždy **jen jednou**.

One-time pad prakticky



Hlavní body

RC4

- **Proudová** šifra, navržená v roce 1987.
- V roce 1994 byl zprvu neveřejný popis šifry zveřejněn na internetu.
- Byla použita v mnoha protokolech:
  - WEP/WPA
  - SSL
  - TLS (do roku 2015)
- Bylo populární díky jednoduchosti.

RC4

- RC4 funguje na principu one-time padu s generátorem náhodných čísel.
- Stačí nám tedy popsat, jak se náhodný keystream generuje.
- RC4 operuje nad polem 256 bytů, v pseudokódu ho budeme značit S.
- Klíč může mít délku 1 až 256 bytů.
- Prvním krokem bude na základě hodnoty klíče inicializovat S.
  - Tento krok se obecně nazývá key scheduling.

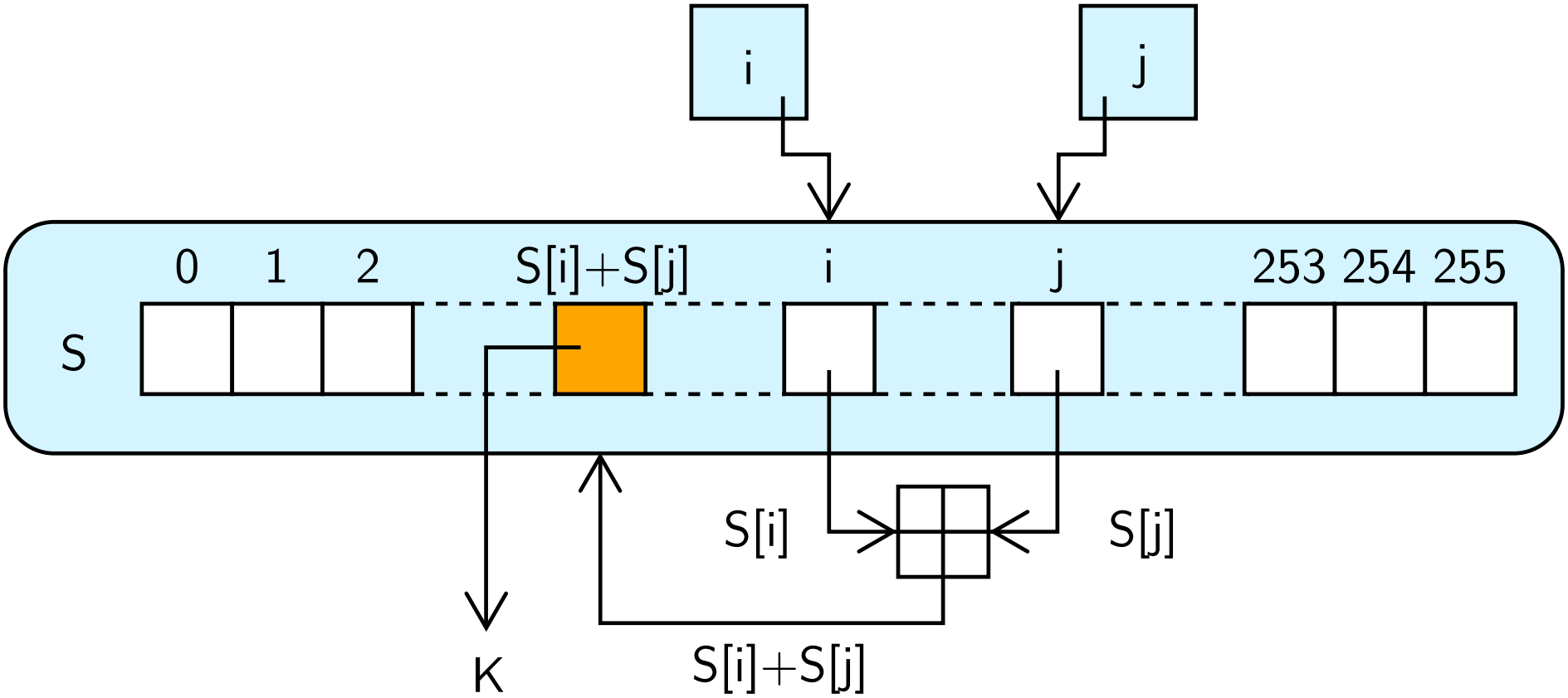
RC4 – key scheduling

1: **for** $i \leftarrow 0$ až 255 **do**
2: $\quad S[i] \leftarrow i$
3: **end for**
4: $j \leftarrow 0$
5: **for** $i \leftarrow 0$ až 255 **do**
6: $\quad j \leftarrow (j + S[i] + key[i \mod keylength]) \mod 256$
7: $\quad$ prohoď $S[i]$ a $S[j]$
8: **end for**

RC4 – generování keystreamu

1: $i \leftarrow 0$
2: $j \leftarrow 0$
3: **while** GeneratingOutput **do**
4: $\quad i \leftarrow (i + 1) \mod 256$ ▷ Posuneme ukazatele i a j.
5: $\quad j \leftarrow (j + S[i]) \mod 256$
6: $\quad$ prohoď $S[i]$ a $S[j]$
7: $\quad t \leftarrow (S[i] + S[j]) \mod 256$ ▷ Podle i a j vrátíme byte.
8: $\quad K \leftarrow S[t]$
9: $\quad$ vrať K
10: **end while**

RC4 – generování keystreamu



Vizualizace řádků 7 až 9

`https://en.wikipedia.org/wiki/File:RC4.svg`

Bezpečnost RC4

- Na RC4 bylo objeveno několik útoků a **není považována za bezpečnou**.
- Sama o sobě neobsahuje IV – protokoly, které ji používají, používají IV jako část klíče.

Hlavní body

ChaCha20

- Publikována v roce 2008.
- Jediná ze dvou symetrických šifer používána v TLS 1.3.
- Postavena na operacích add, rotate, XOR.[2]
- Založená na předchozí šifře Salsa20 (oproti Salsa20 je zvýšena difúze).
- Stejně jako RC4 generuje keystream, který je XORován s OT.

[2] Někdy zkracováno jako ARX.

ChaCha20: vnitřní stav

- ChaCha20 také operuje nad vnitřním stavem a podle něho odvozuje keystream.
- Stav popisujeme jako 4×4 tabulku s 32-bitovými hodnotami (slovy).

v_0	v_1	v_2	v_3
v_4	v_5	v_6	v_7
v_8	v_9	v_{10}	v_{11}
v_{12}	v_{13}	v_{14}	v_{15}

ChaCha20: inicializace vnitřního stavu

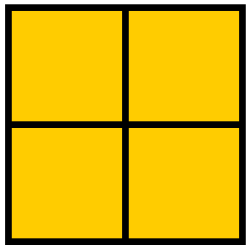
- Na začátku je stav inicializován 256-bitovým klíčem, 64-bitovým IV, 64-bitovým čítačem bloku a 128-bitovou konstantou.

”expa”	”nd 3”	”2-by”	”te k”
klíč	klíč	klíč	klíč
klíč	klíč	klíč	klíč
čítač	čítač	IV	IV

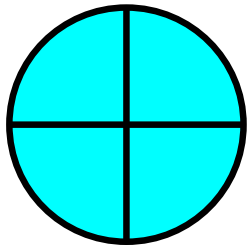
ChaCha20: generování keystreamu

- ChaCha20 funguje trochu jako hašovací funkce.
- Vnitřní stav inicializujeme podle předchozího slidu, z toho pak odvodíme 512-bitový klíč pro tento 512-bitový blok.

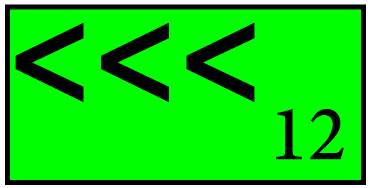
Poznámka ke značení



Sčítání modulo

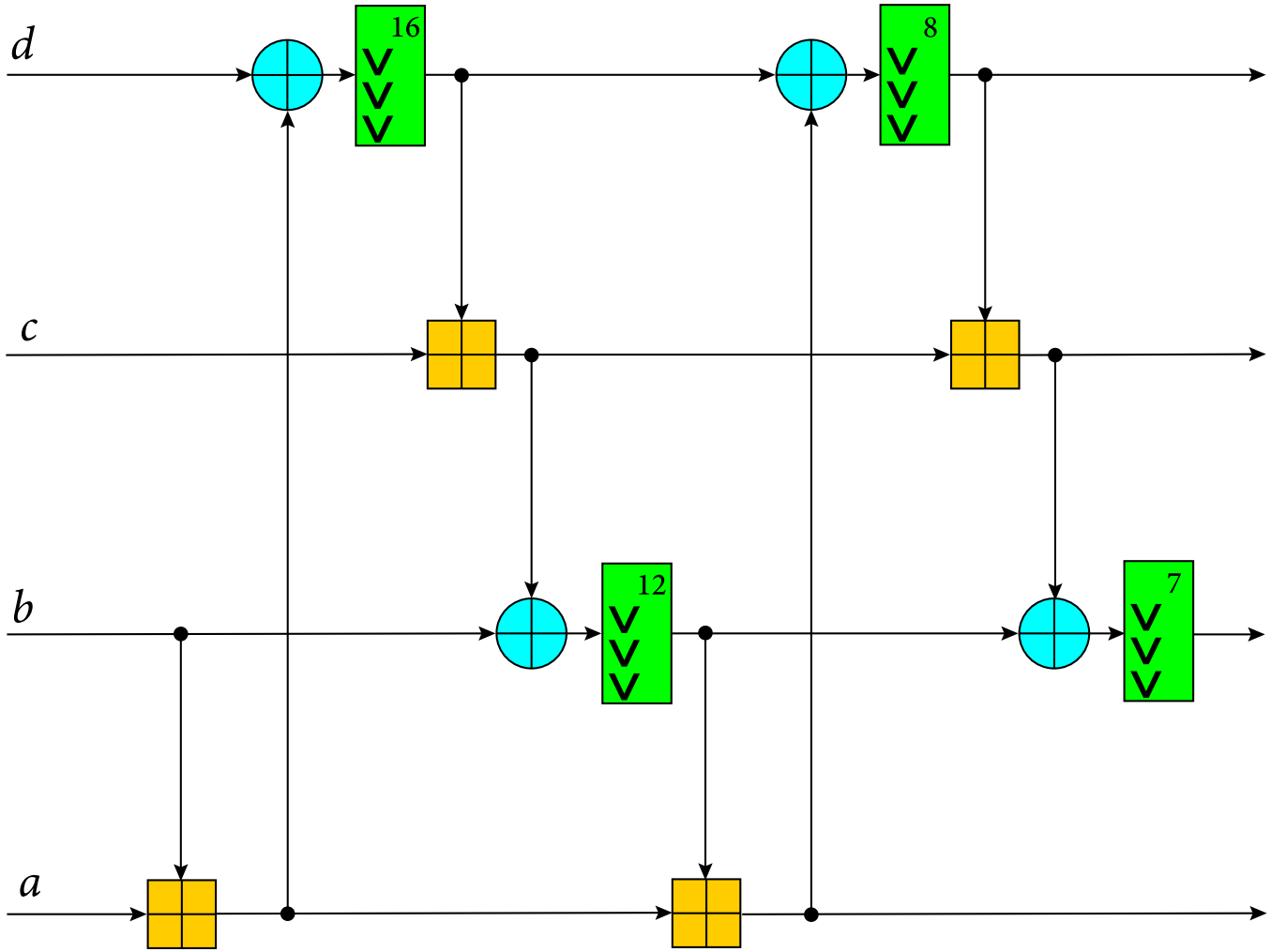


XOR



Rotace

ChaCha20: čtvrt-rundová funkce



https://en.wikipedia.org/wiki/File:ChaCha_Cipher_Quarter_Round_Function.svg

ChaCha20: čtvrt-rundová funkce

```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<=  8;
c += d; b ^= c; b <<<=  7;
```

ChaCha20: dvojitá runda

```
// Odd round
QR(0, 4,  8, 12) // column 1
QR(1, 5,  9, 13) // column 2
QR(2, 6, 10, 14) // column 3
QR(3, 7, 11, 15) // column 4
// Even round
QR(0, 5, 10, 15) // diagonal 1 (main diagonal)
QR(1, 6, 11, 12) // diagonal 2
QR(2, 7,  8, 13) // diagonal 3
QR(3, 4,  9, 14) // diagonal 4
```

ChaCha20

- Dvojitých rund provedeme 10.
- Potom vezmeme **originální** stav a **nový** stav a sečteme je po jednotlivých slovech.
- Výsledkem je 512 bitů keystreamu, které budou XORovány s OT.

Poznámky

- ChaCha20 je poměrně rychlá a jednoduchá šifra.
- Tím, jak je navržená, přirozeně vede na **constant-time implementaci**.