

# Lehký úvod do dvou algoritmů postkvantové kryptografie

---

Mirek Kratochvíl

KSI MFF UK

# RSA IS DEAD



ALSO: DHE, DSA, ElGamal, ECRSA, ECDSA+BTC, ECDHE, ...

- předpokládat, že kvantové počítače nevzniknou je hloupost
- předpokládat, že současná šifrovaná data nebudou mít za 50–100 let cenu je naivní
- celosvětová kryptografická monokultura je nebezpečná
- post-quantové algoritmy jsou často jednodušší i zajímavější

## O čem bude přednáška

- McEliece kryptosystém
- Jak vyrobit digitální podpis z hashovací funkce?
- Co že to má ten kvantový počítač navíc?

## O čem bude přednáška

- McEliece kryptosystém
- Jak vyrobit digitální podpis z hashovací funkce?
- Co že to má ten kvantový počítač navíc?

**Spoiler:** Codecrypt the post-quantum cryptography tool

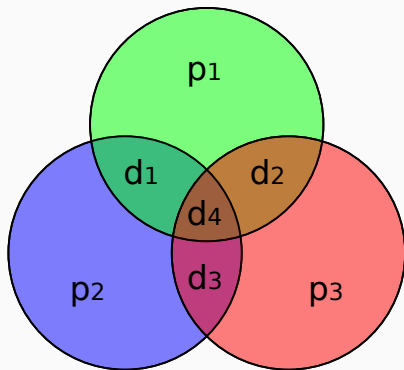
`https://github.com/exaexa/codecrypt`

`(apt-get install codecrypt / emerge codecrypt)`

McEliece

---

Hammingův obecně známý samoopravný kód



Hammingův (7, 4, 1)-kód formálně.

Generátor:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}_{4,7}$$

Kontrola parity:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}_{3,7}$$



Chvíli budeme naivně předpokládat, že nepřítel neumí myslet.  
(např. že nepřítel je jogurt)

Zakódujeme zprávu (0 1 0 1) Hammingovým kódem:

$$\mathbf{m} = (0\ 1\ 0\ 1) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = (0\ 1\ 0\ 1\ 0\ 1\ 0)$$

Jeden náhodný bit zprávy úmyslně rozbijeme:

$$\mathbf{c} = \mathbf{m} \oplus \mathbf{e} = (0\ 1\ 1\ 1\ 0\ 1\ 0)$$

Nepřátelský jogurt vidí:

$$(0\ 1\ 1\ 1\ 0\ 1\ 0)$$

Protože ale *neumí opravovat Hammingovy kódy*, netuší, která zpráva z následujících 4 byla původní:

$$(1\ 1\ 1\ 1), (0\ 0\ 1\ 1), (0\ 1\ 0\ 1), (0\ 1\ 1\ 0)$$

Nepřátelský jogurt vidí:

$$(0\ 1\ 1\ 1\ 0\ 1\ 0)$$

Protože ale *neumí opravovat Hammingovy kódy*, netuší, která zpráva z následujících 4 byla původní:

$$(1\ 1\ 1\ 1), (0\ 0\ 1\ 1), (0\ 1\ 0\ 1), (0\ 1\ 1\ 0)$$

Obecně: při  $t$  chybách v  $n$  písmenech vznikne  $\binom{n}{t}$  možností:

$$\binom{4}{1} = 4 \quad \binom{8}{2} = 28 \quad \binom{100}{10} \approx 1.73 \cdot 10^{13} \approx 2^{43}$$

$$\binom{1024}{38} \approx 10^{69} \approx 2^{230}$$

# McEliece kryptosystém

Z toho můžeme vyrobit asymetrickou šifru!

**Tajný klíč** Samoopravný kód s dostatečným  $n$  a  $t$ , který nepřítel neumí opravit.

**Veřejný klíč** Odpovídající matice  $\mathbf{G}$ , ze které se informace o opravování kódu nedá zjistit.

**Šifrování** Zprávu vynásobíme  $\mathbf{G}$ , znečitelníme ji přidáním chyb a odešleme.

**Rozšifrování** Pomocí znalosti o opravování kódu odstraníme chyby.

Z toho můžeme vyrobit asymetrickou šifru!

**Tajný klíč** Samoopravný kód s dostatečným  $n$  a  $t$ , který nepřítel neumí opravit.

**Veřejný klíč** Odpovídající matice  $G$ , ze které se informace o opravování kódu nedá zjistit.

**Šifrování** Zprávu vynásobíme  $G$ , znečitelníme ji přidáním chyb a odešleme.

**Rozšifrování** Pomocí znalosti o opravování kódu odstraníme chyby.

Problémy:

- Kde vzít takový samoopravný kód?
- Velikost  $G$
- Tohle bez#potíží pře2tu.

- Kde vzít takový samoopravný kód?

Гоппа, В. Д. *Новый класс линейных корректирующих кодов.*

Проблемы передачи информации, Том. VI., Вып. 3, 1970.

Goppovy algebraické kódy jsou poměrně složité, ale kryptosystém funguje!

[https://en.wikipedia.org/wiki/Binary\\_Goppa\\_code](https://en.wikipedia.org/wiki/Binary_Goppa_code)

- Tohle bez#potíží pře2tu.

Problém: Lidské jazyky obsahují docela dost redundance na to, aby se jednobitové chyby daly opravit.

Řešení je víc:

- Vztít náhodnou invertibilní matici a vynásobit s ní  $\mathbf{G}$ .
- Zašifrovat symetrický klíč a zprávu připojit zašifrovanou symetricky. (Fujisaki-Okamoto padding)

- Velikost **G**.

Veřejný klíč má  $\mathcal{O}(n^2)$  bitů, pro původní parametry asi 100kB!

(pro srovnání: ssh-ed25519 AAAAC3NzaC1lZDI1NTE5A-  
AAAIK4bCej3ZCkxHcXLZKLvnlmlwA2Ettb8BeUrkirnmVSb  
user@host)

Řešení: **G** s jednoduchou vnitřní strukturou!



Cyklické matice mají pěknou vnitřní strukturu, k uložení celé stačí opsat první řádek.

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & 1 & 1 \\ 1 & \cdot & 1 & \cdot & \cdot & 1 & 1 \\ 1 & 1 & \cdot & 1 & \cdot & \cdot & 1 \end{pmatrix}$$

Idea:

- Když vyrobíme cyklickou matici  $\mathbf{H}$ , můžeme z ní spočítat i cyklickou  $\mathbf{G}$ !

Cyklické matice mají pěknou vnitřní strukturu, k uložení celé stačí opsat první řádek.

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & 1 & 1 \\ 1 & \cdot & 1 & \cdot & \cdot & 1 & 1 \\ 1 & 1 & \cdot & 1 & \cdot & \cdot & 1 \end{pmatrix}$$

Idea:

- Když vyrobíme cyklickou matici  $\mathbf{H}$ , můžeme z ní spočítat i cyklickou  $\mathbf{G}$ !
- Shodou okolností existují vhodné samoopravné kódy s cyklickou  $\mathbf{H}$ .

Výsledný kryptosystém se jmenuje MDPC.

Matice  $\mathbf{H}$  se vyrobí cyklením nějakého *řídkeho* vektoru:

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\ & & \vdots & & & & & & \vdots & \end{pmatrix}$$

$\mathbf{G}$  se vyrobí trochou lineární algebry.

## Zjednodušená korekce chyb v MDPC

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 \end{pmatrix} \begin{matrix} \text{kódové slovo} \\ \left( \begin{matrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{matrix} \right) \end{matrix} = \overbrace{\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}}^{\text{syndrom}}$$

Který bit může za nejvíc jedniček v syndromu?

## Zjednodušená korekce chyb v MDPC

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = (0 \ 0 \ 0 \ 1)$$

## Zjednodušená korekce chyb v MDPC

$$\begin{pmatrix} 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = (0 \ 0 \ 0 \ 0)$$

Implementace MDPC je poměrně rychlá a klíče jsou dostatečně malé.

```
$ ccr -p -F @8dc |wc -c  
1346
```

```
$ ccr -er exa |ccr -d  
asd
```

```
incoming encrypted message details:
```

```
  algorithm: MCEQCMDPC128F0-CUBE256-CHACHA20
```

```
  recipient: @8dc45217e14d3fdc244f8578e788b3f91c75...
```

```
  recipient local name: 'exa'
```

```
asd
```

## Digitální podpis z hashů

---



## Ale hashovací funkce jsou symetrické?

Kryptografické hashovací funkce se běžně používají jako symetrická kryptografie.

Ověření konzistence přenesené informace:

$$H(\text{sent}) \stackrel{?}{=} H(\text{received})$$

Ověření autenticity zprávy:

$$H(\text{sent}||k) \stackrel{?}{=} H(\text{received}||k)$$

*Asymetrický* digitální podpis je trochu složitější.

Zkusíme aspoň jednorázový podpis jednoho bitu.

Vygenerujeme 2 náhodné řetězce:

- S0: bb2f841c80976e985bd161557d952309c44c3ddd
- S1: 141f87be1330a105a87923f4ee6383bd7de46541

Z obou spočítáme hash:

- H0: 8f431d33712287ea7cf0663553aa000de4595125
- H1: 67573f94f164e5bfdf8bce2dec6486100e716dd0

Hashe H0,H1 zveřejníme, původní řetězce S0,S1 uschováme.

Chceme podepsat jednobitovou zprávu '0':

- Jako podpis nuly zveřejníme  $s=S_0$ .
- Smažeme  $S_1$ .
- Pro ověření spočítáme  $H(s)$  a podpis akceptujeme, když je výsledek stejný jako  $H_0$ .

Kdyby nepřítel chtěl zprávu '1' prohlásit za validní, musel by někde sehnat  $S_1$  nebo invertovat hash  $H_1$ .

Víc podpisů ale udělat nejde.

Schéma jde rozšířit i na  $n$  bitů. Privátní klíč bude:

$S0-1, S0-2, S0-3, S0-4, \dots, S0-n$

$S1-1, S1-2, S1-3, S1-4, \dots, S1-n$

Veřejný:

$H0-1, H0-2, H0-3, H0-4, \dots, H0-n$

$H1-1, H1-2, H1-3, H1-4, \dots, H1-n$

Zprávu hashujeme na  $n$  bitů a každý bit podepíšeme odděleně.

Veřejný i soukromý klíč mají velikost  $2n^2$  bitů (jde to stáhnout na  $n(n + \log_2 n)$ ).

Jak vyrobit znovupoužitelný digitální podpis?

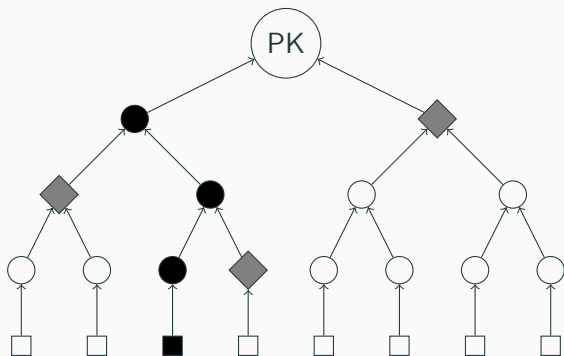
Jak vyrobit znovupoužitelný digitální podpis?

1. Vygenerujeme hromadu párů klíčů pro Lamportův podpis. :)
2. Zveřejníme všechny veřejné klíče.
3. Pro každý podpis použijeme postupně jednu dvojici klíčů!

Problém: 'hromada' je moc velký veřejný klíč.

# Merkle Trees

Řešení: Všechny veřejné klíče zahashujeme Merkleho stromem a zveřejníme jen vrchol stromu.



$i$ -tý opakovatelný podpis se skládá z  $i$ -tého jednorázového podpisu a sousedů cesty v Merkleho stromě.

Běžné technické potíže:

- není potřeba pamatovat si všechny  $S$   
— stačí PRNG



Běžné technické potíže:

- není potřeba pamatovat si všechny S
  - stačí PRNG
- není potřeba mít celý strom v paměti
  - postupné generování např. FMTSeq

Běžné technické potíže:

- není potřeba pamatovat si všechny  $S$   
— stačí PRNG
- není potřeba mít celý strom v paměti  
— postupné generování např. FMTSeq
- spodní patro stromu není potřeba generovat najednou  
— místo hashů se ve stromě použijí podpisy

# Hash-based signatures

Běžné technické potíže:

- není potřeba pamatovat si všechny  $S$   
— stačí PRNG
- není potřeba mít celý strom v paměti  
— postupné generování např. FMTSeq
- spodní patro stromu není potřeba generovat najednou  
— místo hashů se ve stromě použijí podpisy
- podpisů může být i nekonečno  
— few-time signatures ve SPHINCS256

# Hash-based signatures

Běžné technické potíže:

- není potřeba pamatovat si všechny  $S$ 
  - stačí PRNG
- není potřeba mít celý strom v paměti
  - postupné generování např. FMTSeq
- spodní patro stromu není potřeba generovat najednou
  - místo hashů se ve stromě použijí podpisy
- podpisů může být i nekonečno
  - few-time signatures ve SPHINCS256
- podpisy jsou poměrně velké, ale ne tak strašně

## Podpisy v Codecryptu

```
$ ccr -pF @9b3 |wc -c
```

```
109
```

```
$ ccr -s |ccr -v
```

```
Nazdar
```

```
fmtseq notice: 65131 signatures remaining
```

```
incoming signed message details:
```

```
  algorithm: FMTSEQ128C-CUBE256-CUBE128
```

```
  signed by: @9b399912a7f0561839b1048f8a9f81a369a....
```

```
  signed local name: 'exa'
```

```
  verification status: GOOD signature ;-)
```

```
Nazdar
```

# Náhled na složitost

---

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!



## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!

- Premisa: Spočítat NP v polynomiálním čase je podivné.

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!

- Premisa: Spočítat NP v polynomiálním čase je podivné.
- Kdyby šel SAT spočítat v  $P$ , můžu s ním v  $P$  simulovat nedeterministický Turingův stroj, takže spočítám cokoliv z  $NP$ . (Cook-Levin)

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!

- Premisa: Spočítat  $NP$  v polynomiálním čase je podivné.
- Kdyby šel  $SAT$  spočítat v  $P$ , můžu s ním v  $P$  simulovat nedeterministický Turingův stroj, takže spočítám cokoliv z  $NP$ . (Cook-Levin)
- Když někdo umí najít řešení problému s Hamiltonovskou kružnicí, umí spočítat i  $SAT$ , tj. i  $NP$ , tj. je to podivné.

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!

- Premisa: Spočítat NP v polynomiálním čase je podivné.
- Kdyby šel SAT spočítat v  $P$ , můžu s ním v  $P$  simulovat nedeterministický Turingův stroj, takže spočítám cokoliv z  $NP$ . (Cook-Levin)
- Když někdo umí najít řešení problému s Hamiltonovskou kružnicí, umí spočítat i SAT, tj. i  $NP$ , tj. je to podivné.
- ...
- Když někdo spočítá problém dekodování syndromu (SD), vyřeší (tranzitivně) i cokoliv z  $NP$ !

## Jak je složitý nějaký problém?

Pichlavá otázka: Co když jsme náhodou ještě nepřišli na existující jednoduchý algoritmus, který např. invertuje hashovací funkce?

Pichlavá odpověď: Stalo by se dost podivných věcí!

- Premisa: Spočítat NP v polynomiálním čase je podivné.
- Kdyby šel SAT spočítat v  $P$ , můžu s ním v  $P$  simulovat nedeterministický Turingův stroj, takže spočítám cokoliv z  $NP$ . (Cook-Levin)
- Když někdo umí najít řešení problému s Hamiltonovskou kružnicí, umí spočítat i SAT, tj. i  $NP$ , tj. je to podivné.
- ...
- Když někdo spočítá problém dekodování syndromu (SD), vyřeší (tranzitivně) i cokoliv z  $NP$ !

Takže zlomit McEliece by bylo minimálně dost podezřelé.

Obecně se věří, že:

$$RSA, DLog \notin NPC$$

Shor 1993:

$$RSA, DLog \in BQP$$

Obecně se věří, že:

$$RSA, DLog \notin NPC$$

Shor 1993:

$$RSA, DLog \in BQP$$

Co že ty kvantové počítače vlastně dělají?

K čemu jsou qubity? (velmi zjednodušený pohled)

- Jeden qubit je dvoustavová pravděpodobnostní paměť s neomezenou kapacitou rozložení pravděpodobnosti.
- Více qubitů jde spojovat (entanglovat), složitost rozložení roste exponenciálně.
- Rozložení jde rozumně ovlivňovat kvantovými obvody.
- Při měření stavu qubitu celé rozložení zkolabuje a vrátí nějakou relativně pravděpodobnou možnost.



Pro kryptografii jsou zásadní 2 algoritmy:

- Grover search — hledání v neseříděné posloupnosti v  $\mathcal{O}(n^{\frac{1}{2}})$ .
- Shor factorization — rozbíjení RSA modulu a diskretního logaritmu.

# Shorův algoritmus

Redukuje problém na hledání periody  $r$  funkce  $f(x) = a^x \pmod N$  pro náhodné  $a$ .

V kvantovém počítači můžeme:

1.  $f$  vhodně aplikovat najednou na všechny  $x$
2. Výsledek prohnat kvantovou Fourierovou transformací
3. Výsledek QFT změřit.

Měření které periodu  $f$  jasně ukazuje je velice pravděpodobné.

Dělitele  $N$  jsou:

- $\gcd(a^{\frac{r}{2}} - 1, N)$
- $\gcd(a^{\frac{r}{2}} + 1, N)$

## Drobný výhled do budoucnosti

---

- Existují další kryptosystémy!
  - NTRU, NewHope, Stern, MQ, .....
  - SIDH-KEX <https://github.com/elkablo/pqc>

- Existují další kryptosystémy!
  - NTRU, NewHope, Stern, MQ, .....
  - SIDH-KEX <https://github.com/elkablo/pqc>
- Integrace postkvantových prvků do běžných knihoven už probíhá (pomalu)
- Hodně příležitostí vyrobit nový zajímavý software (především pro studenty!)

Konec!

Q&A?