01 Teorie kódování

tags: řsss-řk

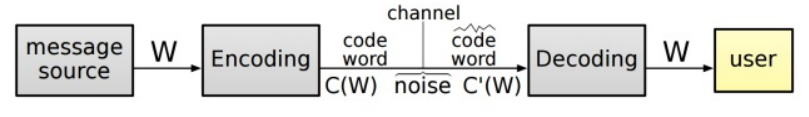
> Teorie kódování. Základy teorie kódování, Shannonova věta. Entropie. Generování skutečně- a pseudo-náhodných sekvencí. Kryptografické protokoly, metody ustavení klíčů, protokoly zero-knowledge. Kvantová kryptografie. (IV054, PV079)

Základy teorie kódování

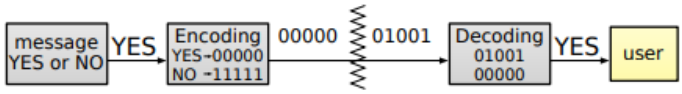
Cílem teorie kódování je vytvořit systémy a metody, které nám dovolí detekovat/opravovat chyby způsobené přenosem informací přes hlučné kanály (noisy channels). Kódování je používáno pro kompresi dat, v kryptografii a error-correction pro přenos informací.

Rozdělení: 1. Noiseless coding theory - Shannon entropie - Huffman coding 2. Noisy coding theory - Error correcting codes - Block codes

Error-correcting kódy jsou používány pro opravu chyb ve zprávě, když je chybně zaslaná přes hlučný kanál.



Příklad:



- Kód C abecedy $\sum$ je neprázdná množina $\sum^*(C \subseteq \sum^*)$.
- q-nární kód je kód abecedy o q-symbolech.
- binární kód je kód abecedy $\{0, 1\}$.

Channel (kanál) je fyzické médium ve kterém je informace uchována a přes které je informace posílána. - Např. telefonní linka, optické vlákna, etc.

Šum (noise) mohou způsobovat sluneční paprsky, záření, blesky, sprcha meteorů, náhodné radiové vlny, etc.

Typy kanálů

Hlavní typy kanálů jsou diskrétní (discrete) a kontinuální/nepřetržité (continuous) kanály.

Diskrétní Shannonův stochastický kanál je trojice $(\sum, \Omega, p)$, kde: - $\sum$ je vstupní abeceda - Ω je výstupní abeceda - Pr je pravděpodobnostní distribuce $\sum \times \Omega$ a pro každé $i \in \sum, o \in \Omega, Pr(i, o)$ je pravděpodobnost, že výstup z kanálu je o pokud je vstup i.

Hlavní model šumu v diskrétním kanálu je: - *Shannon stochastic (probabilistic) noise model*: Pravděpodobnost $Pr(i, o)$ je známá a pravděpodobnost hodně chyb je nízká. - *Hamming adversarial (worst-case) noise model*: Kanál se chová jako záškodník, který může samovolně poškozovat vstupní kódová slova (codewords) v rámci daného počtu chyb.

Další důležité kanály: - Binární symetrický kanál mapuje, s fixní pravděpodobností p_0, každý binární vstup na opačný výstup. Proto, $Pr(0, 1) = Pr(1, 0) = p$ a $Pr(0, 0) = Pr(1, 1) = 1 - p$. - $Pr(0, 1)$ se čte: pravděpodobnost, že výstup je 1, pokud byl vstup 0. - Binary erasure channel mapuje, s fixní pravděpodobností p_0, binární vstupy do $\{0, 1, e\}$, kde e je nazýváno **erasure symbol** a $Pr(0, 0) = Pr(1, 1) = p_0$, $Pr(0, e) = Pr(1, e) = 1 - p_0$

Dvě základní kódovací metody: 1. **BEC** (Backward Error Correction) příjemci dovoluje pouze detekovat chyby. Pokud je detekována chyba, požádá odesílatele o znovuzaslání

zprávy 2. **FEC** (Forward Error Correction) příjemci dovoluje opravit určité množství chyb. - vyžaduje delší zpracování na straně příjemce (větší latency), ideální na dlouhé vzdálenosti, kde přeposlání zprávy by bylo neefektivní, např. satelitová komunikace

Základní předpoklady o kánálech: 1. **Zachování délky slova**. Každý výstupní slovo by mělo mít stejnou délku jako jeho korespondující vstupní slovo 2. **Nezávislost chyb**. Pravděpodobnost, že jeden symbol bude ovlivněn chybou v průběhu přenosu, by měla být vždy stejná.

Základní strategie pro dekódování: - Pro dekódování použijeme *maximum-likelihood principle* nebo *nearest neighbor decoding strategy*: - Příjemce by měl dostat slovo w' jako kódové slovo w, které je nejblíže slovu w'.

Hammingova vzdálenost:

- Koncept jak spočítat podobnost dvou kódových slov. Pro kódové slova x a y, jejich Hammingova vzdálenost $h(x, y)$ je počet symbolů, ve kterých se liší.
- Vlastnosti Hammingovy vzdálenosti
  - $h(x, y) = 0 \rightarrow x = y$
  - Symetrie: $h(x, y) = h(y, x)$
  - Trojúhelníková nerovnost: $h(x, z) \leq h(x, y) + h(y, z)$
- **Minimální vzdálenost kódu je důležitá vlastnost kódu a značí nejmenší počet chyb, které změní jedno kódové slovo do druhého.**
- Počítá se jako: $h(C) = min\{h(x, y)|x, y \in C, x \neq y\}$

Základní error-correction theorem: 1. Kód C dokáže detekovat až s chyb, pokud $h(C) \geq s + 1$. 2. Kód C dokáže opravit až t chyb, pokud $h(C) \geq 2t + 1$.

(n, M, d)-kód C je takový kód, kde: - n - je délka kódového slova - M - je počet kódových slov - d - je minimální vzdálenost v kódu C.

Příklad: - $C1 = \{00, 01, 10, 11\}$ je (2,4,1)-kód. - $C2 = \{000, 011, 101, 110\}$ je (3,4,2)-kód.

Dobrý (n, M, d)-kód má malé n, velké M a velké d.

Information rate, nebo **code rate** q-nárního (n, M, d)-kódu C reprezentuje poměr počtu potřebných vstupních znaků k počtu přenášených znaků. Pokud binární kód má **code rate** R, říkáme, že přenese R bitů za *channel use* (použití kanálu?).

Entropie

:movie_camera: StatQuest - Dlouhé, ale entropie vysvětlena :movie_camera: Computerphile - Entropie

Shannonova entropie značí střední hodnotu množství dat v zasílané informaci, je měřena v bitech. Když datový zdroj vyprodukuje hodnotu, která má nízkou pravděpodobnost (tj. nastane událost s nízkou pravděpodobností), nese tato událost více "informace" (způsobí větší "překvapení"). Entropie značí take míru nejistoty – čím vyšší entropie, tím je způsobeno větší "překvapení". Entropie je míra neurčitosti náhodného pokusu. Pojem **entropie** lze chápat jako pojmenování pro *nahodilost*.

Entropie je definována jako: $S(X) = -\sum_x p(x) \log p(x)$

- Entropie systému je minimální (nulová), pokud dokážeme předpovědět jeho výstup v každém případě (s pravděpodobností 1)
  - Například budeme mít minci, která má na obou stránách pannu. Její entropie bude tedy 0, jelikož pokaždé padne panna.
- Entropie systému je maximální, pokud všechny hodnoty produkuje se stejnou pravděpodobností (rovnoměrné rozložení) - $S(X) = \log_2 n$
  - Tohoto se využívá i v kryptografii. Například 128bitový klíč, který je rovnoměrně a skutečně-náhodně generovaný, má entropii 128 bitů. Entropie však selhává, když klíče nejsou voleny rovnoměrně. Třeba předpokládejme, že z důvodu chyby je první bit vždy jako 1. Tudíž nám stačí cracknout zbylých 127.

Shannonův teorém

Shannonův noiseless teorém říká, že pro přenos n hodnot informace X, potřebujeme použít $nS(X)$ bitů. Nemůžeme použít méně bitů pro kódování, pokud nechceme ztratit nějakou část informace.

Příklad: Zdroj X produkuje hodnotu 1 s pravděpodobností $p=\frac{1}{4}$ a hodnotu 0 s pravděpodobností $1-p=\frac{3}{4}$. Předpokládejme, že chceme zakódovat 4-bitový blok. Za použití Shannonova teorému musíme použít v průměru 3.245 bitů.

Jinak řečeno, pokud budeme posílat n bitů s pravděpodobností chyby p, by mělo nastat přibližně $p \times n$ chyb.

Kódy s proměnnou délkou

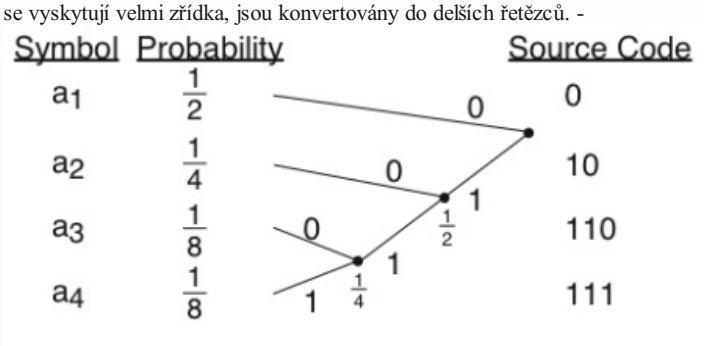
Kódy s proměnnou délkou, jako je např. ASCII, mají všechna kódová slova o stejné délce. V teorii kódování, kód s proměnnou délkou je takový kód, který mapuje zdrojové symboly na proměnnou délku bitů.

Třídy kódů s proměnnou délkou: - Non-singular codes - Každý symbol ze zdrojové abecedy S je mapován na unikátní neprázdnou sekvenci symbolů z cílové abecedy T. - Např. $\{a \mapsto 1, b \mapsto 011, c \mapsto 1100, d \mapsto 11001\}$ - Uniquely decodable codes - Jakákoliv sekvence symbolů validního kódového slova z T může být unikátně dekódovaná na symboly z S. - Např. mapping $M_3 = \{a \mapsto 0, b \mapsto 01, c \mapsto 011\}$ je unikátně dekódovatelná, jelikož každé slovo ukončíme, pokud uvidíme bit 0 - což znamená, že začíná nové slovo. - Preffix/Suffix codes - žádný jiné slovo nezačíná prefixem/suffixem jiného slova. - Např. $\{a \mapsto 0, b \mapsto 10, c \mapsto 110, d \mapsto 111\}$ - *aabacdab* → 00100110111010
→ |0|0|10|0|110|111|0|10| → *aabacdab*

Unární kódování - zakóduje přirozené číslo n pomocí n po sobě následujících jedniček a jednou nulou a naopak. - Např. $\{1 \mapsto 0, 2 \mapsto 10, 3 \mapsto 110, \ldots\}$

Binární kód - Kód není *unique decodable*, nejdříve musíme definovat oddělovač nebo délku slova. Když chceme definovat délku slova, musíme znát největší možnou hodnotu, kterou chceme zakódovat.

Huffmanovo kódování - Algoritmus pro bezeztrátovou kompresi dat. Konvertuje znaky vstupního souboru do bitových řetězců různé délky. Znaky, které se ve vstupním souboru vyskytují nejčastěji, jsou konvertovány do bitových řetězců s nejkratší délkou (nejfrekventovanější znak tak může být konvertován do jediného bitu), zatímco znaky, které se vyskytují velmi zřídka, jsou konvertovány do delších řetězců. -



-

Huffmanovo kódování je optimální, pokud pravděpodobnosti jednotlivých symbolů jsou násobky $\frac{1}{2}$. Symbol s pravděpodobností 0.99 nese pouze $\log_2(\frac{1}{0.99}) = 0.014$ bitů informace, ale Huffmanovo kódování zakóduje každý symbol separátně, takže minimální délka symbolu bude 1 bit, což není optimální. - Toto se dá vyřešit použitím **Extended Huffman coding**, kde kódujeme sekvence zdrojových symbolů a ne jednotlivé symboly. - Např. budeme mít sekvenci 0000111100001111, tak kódujeme sekvence 0000 ↦ 0 a 1111 ↦ 1.

Fixed-length (=block) codes

Blokové kódy zahrnují Hammingův, Golayův, Reed-Solomon, cyclic a polynomiální kód. :::danger Tady bychom se měli věnovat lineárním kódům. Nejsem si jistý, jestli to po nás někdo bude chtít u státnic. :::

Generování skutečně- a pseudo-náhodných sekvencí

Náhodná data jsou stěžejní pro kryptografické klíče, *padding values*, nebo *nonces*. Potřebujeme kvalitní (aby splňovala statistické vlastnosti) a nepředvídatelná data. Nicméně v deterministickém prostředí (= počítače), je občas složité získat opravdu náhodná data v rozumném čase.

True random number generators (TRNGs)

TRNGs bývají pomalé a jejich kvalita závisí na zdroji náhodnosti: - Ideální zdroj: nedeterministický fyzický jev (radioaktivní rozpad zdroje záření, atmosférický nebo elektronický šum, vesmírná radiace). - Excelentní: HW-based, buď zabudované čipy nebo přídavné karty. - Dobré: Jakýkoliv vstup (časování stisku kláves, pohyb myši, vstup z mikrofonu nebo videokamery), jelikož I/O HW je většinou levný a generuje hodně šumu. - Přijatelné: SW-based (procesy, síť, …) - Špatné (předvídatelné, nedostatečná entropie): systémové datum a čas, process ID, process runtime

Pseudo-random number generators (PRNGs)

Pseudo-random generátor je deterministický konečný automat, který: - Je v jakémkoliv čase v určitém stavu: - Stav by měl být secret (výstup generátoru musí být pro kryptografické účely nepředvídatelný) - Stav se opakovatelně mění (generátor musí produkovat jiný výstup)

Na základě krátkého vstupu (většinou hash skutečně náhodné hodnoty) zvané *seed*, produkuje deterministický algoritmus pseudo-náhodný výstup. Pro stejný *seed* se generuje stejná sekvence čísel. V ideálním případě je výstup nerozeznatelný od skutečně náhodných dat.

Linear feedback shift register (LFSR)

:movie_camera: Computerphile - LFSR

LSFR je shift register (posuvný registr), kde vstupní bit je lineární funkcí jeho předchozího stavu. Typicky se XORuje několik předchozích bitů. Výstup je tedy pseudo-náhodný bit.

LSFRs jsou velice rychlé a dobré pro HW implementaci. - Mají konečný počet možných stavů, pseudo-náhodná sekvence je generována pořád dokola. - Např. bude generovat pořád dokola čísla $1, 3, 5, 1, 3, 5, 1, 3, 5$ - proto chceme dlouho *periodu generátoru* - Jelikož použitá funkce je lineární, výstup je náchylný ke kryptoanalýze. - To můžeme zlepšit použitím nelineární kombinace několika LSFRs použitím vhodné nelineární funkce.

Cryptographically secure RSA and BBS PRNGs

RSA PRNG založen na systému RSA: - p, q jsou prvočísla; $n = pq$; $\varphi(n) = (p - 1)(q - 1)$;$gcd(e, \varphi(n)) = 1$ - φ = Euler's totient function - gcd() = greatest common divisor - Seed x_0 je vybrán z rozsahu $[2, n - 2]$ - Pro $i \in [1, m]$ udělej: - $x_i = (x_{i-1})^e \mod n$; - $z_i = lsb(x_i)$; i.e., z_i je least significat bit od x_i - Výstup je sekvence $z_1, z_2, \ldots, z_m$ o délce m BBS PRNG je podobné, ale používá $x_i = (x_{i-1})^2$. Obě dvě jsou ale celkem pomalé.

ANSI X9.17/X9.31 PRNG

- Založen na 64-bit 3DES-3 nebo 128-bit AES.
- Klíč K je použit pouze pro generátor a ne pro žádný jiný účel.
- Seed je 64/128-bitová hodnota s.
  - Podle toho, jestli použijeme DES nebo AES.
- DT je 64/128-bit reprezentace datumu/času.
  - Náhodně inicializovaný counter může být taky použit.
- Pokaždé, co potřebujeme náhodné číslo, je potřeba:
  - Získat DT;
  - Vypočítat dočasnou hodnotu $t = E_K(DT)$; (E_K znamená šifrování s klíčem K).
  - Vypočítat náhodnou hodnotu $r = E_K(s \oplus t)$;
  - Updatnout seed pro další iteraci: $s = E_K(r \oplus t)$. Tato PRNG se ale nikdy nezotaví z

kompromitovaného stavu (např. s uniklým klíčem K), maximálně se zotaví po dlouhé době. Sofistikovaným řešením je Yarrow a Fortuna PRNG.

Yarrow PRNG

Bežný design: - *Entropy accumulator*: sbírá vzorky entropie ve 2 poolech (fast a slow - poolu rozumějte jako zdroji entropie (např. to zmíněné záření, pohyb myši, etc.)) - *Reseed mechanism*: periodicky *reseeduje* klíče z poolu: *fast pool* pro častější *reseedování* a *slow pool* pro méně časté. - *Reseed control*: rozhoduje, kdy se má *reseed* provést (založeno na *entropy estimators* a predefinovaných thresholdech) - *Generation mechanism*, generuje výstupy založené na klíči/seedu: - Založen na blokové cifře v CTR(counter) módu. - Seed je klíč K: - $C_i = (C_{i-1} + 1) \mod 2^n$ - $\text{output}_i = E_K(C_i)$ - Yarrow-160 používá 3DES-3 a SHA-1 - SHA-1 má maximálně 160 bitů - Po 10 výstupech se klíč mění

Fortuna PRNG

- State-of-the-art
- Zotaví se rychle z interního stavu kompromitace, přidává často entropii
- Založen na blokových šifrách (AES, Serpent, Twofish) a SHA-256
- Jeho *entropy accumulator* cyklicky sbírá entropii z 32 poolů:
  - Po n-tém reseedování PRNG, pool k je použit pouze pokud 2^k dělí n.
    - P_0 je použit pro každé reseedování, P_1 každé druhé, P_2 každé čtvrté, …
  - Vždy bude nějaký pool s dostatečnou dávkou entropie

Cryptanalytic attacks

1. Direct cryptanalytic
   - Útočník je schopen rozeznat, zda-li je to výstup z PRNG nebo skutečně-náhodný výstup
   - Je potřeba, aby viděl výstup z PRNG.
2. Input-based attack
   - Útočník používá vstupy pro PRNG, aby rozeznal výstupy PRNG a náhodných hodnot.
3. State compromise attack
   - Útočník zneužívá předchozí kompromitované stavy, aby rozeznal mezi PRNG výstupy a náhodnými hodnotami

Statistické testování náhodnosti

Kvalita random number generátorů se měří pomocí statistických testů. Generované náhodné sekvence by měly vypadat "náhodně" - což znamená, že by měly splňovat statistické vlastnosti jako data pocházející z rovnoměrného rozložení.

1. NIST testy
   - 15 testů náhodnosti
     - Statistika bitů - počet jedniček, nejdelší jedničková sekvence
     - Statistika m-bit bloků - frekvence m-bit bloků
     - complexní statistiky m-bit bloků - lineárni komplexita, matrix rank
2. DIEHARD
   - DIEHARD taky testuje implementaci

Kryptografické protokoly

Protocol

- Algoritmus definovaný sekvencí kroků, které musí splnit každý (dva a více) účastník komunikace, abychom dosáhli cíle protokolu
  - Cíle protokolu: důvěryhodnost, autentizace(=ověření identity), integrita, ustanovení klíčů, non-repudiation(=odesílatel nemůže popřít svoje autorství)
- Protokoly jsou založené na *secret* (tajemství):
  - Důvěra je založena na vlastnictví *secretu*
  - Pro symetrickou kryptografii je důvěra založena na sdílení klíče
  - Pro asymetrickou kryptografii je důvěra založena na držení privátního klíče a *chain*

of trust mezi veřejným klíčem a dalšími daty (např. webové certifikáty - u nich věříte, že jsou podepsány certifikační autoritou)

Nonce

- *Number used once*
- Mohou být:
  - unikátní/náhodné - pocházejí z rovnoměrného rozložení, používají se např. pro challenge-response
  - sekvenční (inkrementální) čísla - např. z čitače
  - Timestamps - např. jako *acceptance window* (kolik času jsem ochoten čekat, abych přijal zprávu jako *fresh*); bezpečně synchronizovaná a distribuovaná informace o čase

Autentizace jedince

- Jednostranná / vzájemná. Jedna strana se autentizuje / dvě strany se autentizují navzájem.
- Autentizace u asymetrické kryptografie (Alice a Bob znají své public klíče):
  1. Alice → Bob: $E_B(n)$ - náhodný nonce zašifrovaný public klíčem Boba.
  2. Alice ← Bob: $E_A(n + 1)$ - Bob vrátí nonce inkrementovaný o jedna zašifrovaný public klíčem Alice - ta ví, že komunikuje s Bobem.
  3. Alice → Bob: $E_B(n + 1)$ - Bob ví, že komunikuje s Alice
- Challenge response protokoly (Alice a Bob sdílejí stejný secret):
  1. Alice → Bob: Challenge (zašli mi secret)
  2. Alice ← Bob: Response (secret)
  3. Alice → Bob: OK/NOK (verifikace secretu) #### Commitment scheme - e.g. vzdálené házení mincí
- Problem: Dvě strany jsou vzdáleně od sebe, Alice si tipne výsledek, Bob hodí mincí a oznámí výsledek. Pokud by Bob věděl, co Alice tipovala, mohl by oznámit opačný výsledek. Na druhou stranu, pokud by Alice věděla Bobův výsledek, mohla by podle toho zaslat svůj tip.
- Řešení:
  - Alice pošle svůj tip Bobovi ve formátu $H(g + n)$, kde H je hashovací funkce, g je její tip (panna/orel) a n je náhodný nonce.
  - Bob flipne mincí a zašle ho Alici
  - Alice oznámí svůj výsledek a nonce
  - Bob ověří, že se jedná o správný výsledek.

Protokoly pro ustanovení klíčů

- Ustanovené klíče jsou sdíleným *secretem*, který sdílí všichni účastníci komunikace
- *Key transport*: Bezpečná distribuce secretu - Evil Eve nedokáže odposlechnout tento secret
- *Key agreement*: sdílený secret je odvozen Alicí a Bobem na základě vyměněných dat a (ideálně), nikdo z nich nedokáže předpovědět výslednou hodnotu.
- Vlastnosti (koncepty jistoty):
  - *Implicit key authentication*: Jistota Boba, že nikdo jiný, kromě Alice, nemohl přečíst klíč.
  - *Key confirmation*: Alice ujistí Boba, že pouze ona zná klíč.
  - *Explicit key authentication*: Obě dvě vrchní vlastnosti platí.
  - *Entity authentication*: Ujištění jedné strany o identitě druhé strany. Alice ví, že komunikuje s Bobem a naopak.
- Session klíče:
  - Short-term klíče
  - Malý počet zpráv zašifrovaných s tímto klíčem → trápí nás méně, když klíč unikne
  - *Perfect forward secrecy* znamená, že kompromitace dlouhodobého klíče nijak nekompromituje zpětně session-klíče
    - Budoucí odhalení klíče nekompromituje zprávy z minulosti
  - *Perfect backward secrecy* kompromitace klíče z minulosti nijak neovlivní zprávy v budoucnosti.
  - Například: **Diffie-Hellman**: umožňuje přes nezabezpečený kanál vytvořit mezi komunikujícími stranami šifrované spojení

Zero-knowledge protokoly

- Dovoluje účastníkům komunikace dokázat, že drží určitý *secret*, aniž by odhalili jakékoliv informace, které by ho mohly kompromitovat (*secret*)
- Proof of knowledge s následujícími vlastnosti:
  - Completeness (Úplnost) - poctivé strany vždy dosáhnou úspěšného výsledku
  - Soundness (Korektnost) - pravděpodobnost, že nepoctivý útočník přesvědčí druhou stranu o něčem, co není pravda, je mizivá
- Obecný proces:
  - 2 strany: Prover (P) a Verifier (V)
  - P → V: witness, V → P: challenge, P → V: response
  - Výsledek: V je přesvědčeno, že P zná secret, aniž by se V o něm cokoliv dozvědělo
    - V ale nedokáže toto potvrdit nikomu jinému
- Příklad: Máme chodbu ve tvaru kruhu, kterou dělí dveře. P chce dokázat V, že od nich vlastní klíč. P se tedy vydá jednou stranou a přijde opačnou stranou. Tímto dokázalo V, že má klíč od dveří, aniž by to V viděl.
- Protipříklad: Chceme dokázat, že n = 670592745 není prvočíslo a odhalíme 670592745 = 12345 · 54321 - což není zero-knowledge.

Feige-Fiat-Shamir příklad nepřidávám, je uveden v https://www.fi.muni.cz/usr/gruska/crypto21/cr2108_2_2.pdf slide 6

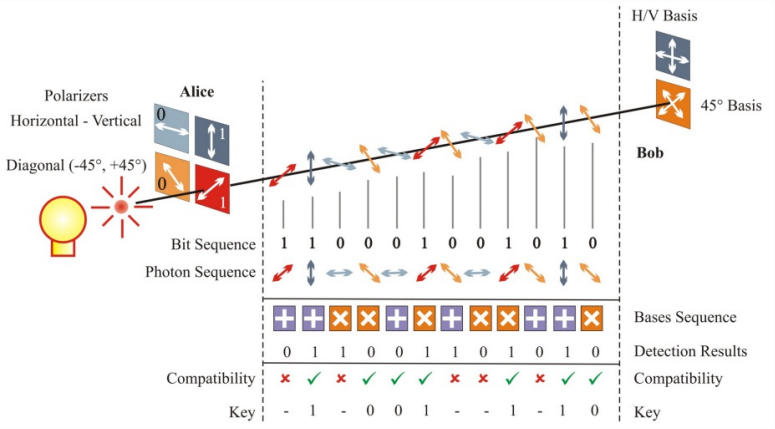
Kvantová kryptografie

Doporučuju se nejdřív podívat na následující videa: 1. :movie_camera: Quantum Cryptography in 6 Minutes 2. :movie_camera: BB84 Protocol of quantum key distribution

- Založena na kvantové fyzice (přírodní zákony), ne na předpokladu, že některé problémy je velice složité vypočítat.
- Kvantová fyzika pracuje se základními entitami fyziky - částice (*particles*):
  - Protony, elektrony, neutrony (z čeho je *matter* vytvořena)
  - Fotony (nosiče elektromagnetického záření)
  - Další "*elementary particles*" které zprostředkovávají další interakce ve fyzice
- Kvantová fyzika je plná mind-fucků, záhadných a paradoxních jevů.
  - Například *quantum particle* "se může nacházet na dvou místech najednou" a interagovat se sebou.
- Klasická vs. kvantová informace:
  - **Klasická**:
    - Jednoduchá na uložení, přenos i zpracování v čase a prostoru
    - Je jednoduché udělat nekonečné množství kopií klasické informace
    - Můžeme ji měřit aniž bychom ji narušili
  - **Kvantová**:
    - Složité ji uložit, přenést i zpracovat
    - *No-cloning theorem*: Není možné, abychom zkopírovali *perfectly unknown* kvantovou informaci
    - Změření kvantové informace ji zničí - útočníci jsou odhaleni
    - *Heisenbergův princip neurčitosti*: čím přesněji měříme polohu částice, tím méně přesněji měříme její momentum a naopak.
- V klasických počítačích je informace reprezentována na makroskopické (věci, které můžeme vidět pouhým okem) úrovni klasickými bity, v kvantových počítačích je informace reprezentována na mikroskopické úrovni (věci, které nemůžeme vidět pouhým okem) a je reprezentována qubity (*quantum bits*): mohou nabývat nepočítatelně mnoho hodnot:
  - $\alpha|0\rangle + \beta|1\rangle$, kde α, β jsou komplexní čísla taková, že $|\alpha^2| + |\beta^2| = 1$
- Běžný n-bitový registr dokáže v jednu chvíli uložit pouze jeden n-bitový string.
- Kvantový n-qubit registr dokáže v jednu chvíli udržet superpozici všech 2^n n-bitových stringů. Tím pádem kvantový počítač dokáže "vypočítat" v jednom kroku všech 2^n hodnot funkce definované na n-bitovém vstupu.
  - *Kvantová superpozice stavů je základním principem kvantové mechaniky, podle kterého lze každé dva (a více) kvantové stavy kombinovat (superponovat), čímž vznikne nový kvantový stav. To znamená, že každý kvantový stav lze popsat jako součet dvou (a více) jiných stavů.*

BB-84 [Bennet-Brassard, 1984] shared key establishment

1. Alice vygeneruje náhodnou sekvenci bitů a zakóduje je jako qubity
2. Alice zasílá Bobovi photony polarizované se základem buď jako 0° a 90° (*rectalinear scheme*), nebo jako 45° a 135°(*diagonal scheme*); základ je vybrán náhodně pro každý zasílaný qubit.
    - Jakmile Bob přijme všechny qubity, tak ví, že Eve nemůže mít žádnou kopii qubitů (no-cloning theorem)
    - Nicméně Eve mohla odposlouchávat, ale tím by narušila cca 50% jednotlivý qubitů
3. Bob náhodně měří jednotlivé qubity (použije pro každý náhodnou *scheme*) a následně sekvenci použitých schemes si vymění s Alici přes nezabezpečený kanál
    - Takhle oba dva vidí, kdy použili stejnou scheme a tím pádem které bity byly změřeny správně.
    - Pokud Eve odposlouchávala a změřila špatně qubit, tak Alice s Bobem zjistí hned při první vyměněné zprávě, jelikož budou mít rozdílný klíč - dobře je to vysvětlené zde: https://youtu.be/44G9UuB2RWI?t=476



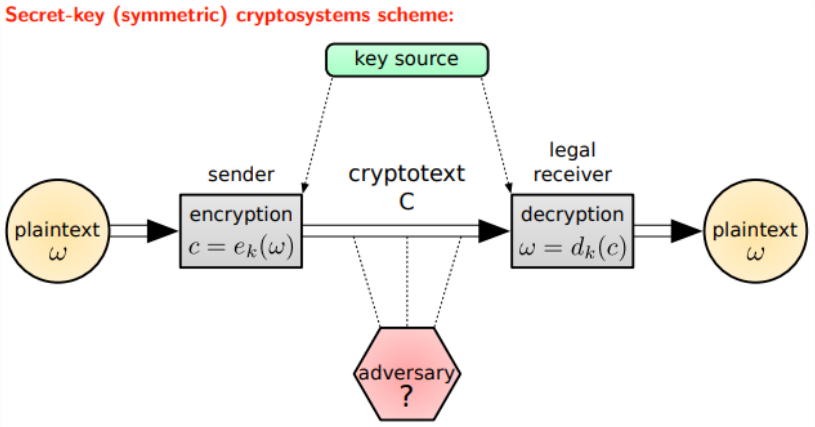
02 Symetrické a asymetrické šifry (2hodinky)

tags: `řsss-řk`

> Symetrické a asymetrické šifry. Principy symetrických blokových šifrovacích algoritmů (Feistelovy šifry, DES, AES) a asymetrické algoritmy (RSA, Diffie-Hellman, DSA / ElGamal). Faktorizace a testování prvočíselnosti. Principy konstrukce hašovacích funkcí. Kryptosystémy na bázi eliptických křivek. (PV079)

Symetrická kryptografie

Matematická definice: - *(Symmetric) cryptosystem* je pětice (P, C, K, E, D): - P: množina všech možných plaintextů (zpráv) nad danou abecedou - C: množina všech možných *ciphertextů* (šifer) - K: množina všech možných klíčů - E: šifrovací funkce - D: je dešifrovací funkce - Cíl je poskytnout rychlé šifrování a dešifrování zpráv mezi dvěma účastníky komunikace, kteří sdílí klíč k. - *(Symmetric) cryptosystem* by měl splnit následující vlastnosti: - Daným $p \in P$ a $k \in K$, musí být jednoduché vypočítat $c = E(p, k)$ - S pouze šifrovaným textem $c \in C$, musí být složité vypočítat $p = D(c, k)$ - Musí být jednoduché vygenerovat hodně různých klíčů $k \in K$ - Šifra je dvojice (E, D) z šifrovací a dešifrovací funkce: - $E : P \times K \rightarrow C$ - $D : C \times K \rightarrow P$ - $D(E(p, k)k) = p$ - E a D jsou *one-to-one* funkce



Odesílatel a příjemce sdílí stejný šifrovací klíč. Pomocí něho šifrují/dešifrují zaslaný text. Útočník může odposlechnout pouze šifru.

- Kerckhoffsův princip: Šifra musí být bezpečná, i pokud všechny informace o algoritmu jsou veřejné. Jinými slovy, bezpečnost kryptosystému musí záviset pouze na klíči a prozrazení principu šifrování ho nemá ohrozit. (Opak je *security by obscurity*).
  - Všechny historické šifry (Caeser(shift)-cipher / Enigma) jsou symetrické šifry, které byly primárně založeny na *security by obscurity*. Např. u Caesar šifry nám stačilo znát posunutí a mohli jsme daný text odšifrovat.

Blokové šifry

- Blokové šifry jsou symetrické šifry, které procesují data v blocích o fixní délce bitů.
  - Např. DES, AES

***Stream* (proudové) šifry**

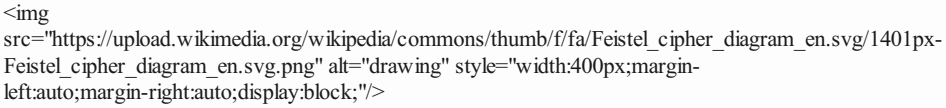
- *Stream* šifry jsou symetrické šifry, které procesují data jako *stream* bitů. Generuje *keystream* a XORuje s ním plaintext data bit po bitu.
  - Například One Time Pad
  - Blokové šifry mohou být změněny na *stream* šifry použitím různých módů operace (např. CTR nebo OFB)

Feistelovy šifry

:movie_camera: Feistel Cipher - Computerphile

Feistelova síť/šifra je běžný přístup k návrhu blokových a *stream* čifer (DES, TEA, Blowfish, Twofish, KASUMI) Proces: 1. Text je rozdělen na dvě části - levá a pravá, *L* a *R*. 2. V každém kroku: - *R* vstupuje do funkce *F* s klíčem *K* a xoruje se s *L* - Funkce F "*is a round function, a function which takes two inputs – a data block and a subkey – and returns one output of the same size as the data block. - WIKI*" - Prohodíme pravou a levou stranu - Klíče měníme v každém kroku

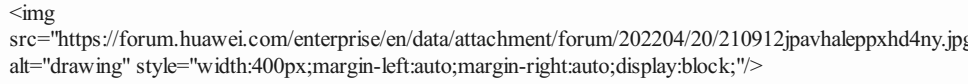
Šifrování můžeme provést několikrát. Pro dešifrování pouze prohodíme pořadí - viz. diagram.



DES - Data encryption standard

- Prolomen v roce 1999, v roce 2001 nahrazen AES
- Délka bloku: 64 bitů
- Délka klíče: 56 b + 8 paritních bitů (každý byte má paritní bit)
- Počet kol (*rounds*): 16 Algoritmus si předpočítá 16 *subklíčů* o délce 48 bitů z originálního klíče pro každé kolo. Šifrování je pak stejné jako ve Feistelově šifře -

prohazujeme levou a pravou stranu.



Triple DES

Nevýhoda **DES**u je taková, že používá relativně krátký klíč. Z toho důvodu se začal používat **Triple DES**, který provede tři operace na jeden blok dat. Nejběžnější způsob je použití $E_{K3}(D_{K2}(E_{K1}(plaintext)))$. Dešifrování D_{K2} nám umožní zpětnou kompatibilitu (pokud jsou všechny tři klíče stejné) a zlepší bezpečnost, pokud $K_3 = K_1$ Varianty: - Všechny tři klíče jsou nezávislé: 168 b klíč (56b*3), ale kvůli meet-in-the-middle útoku, je to pouze o trošku bezpečnější, než použití $K_1 = K_3$. - Meet-in-the-middle útok - :movie_camera: Meet-in-the-middle - easy explanation - Tl;dr Příklad s 2DES - Vezmeme $E_{K1}(p) = X \rightarrow E_{K2}(X) = c$, pro dešifrování potřebujeme vypočítat $D_{K2}(c) = X \rightarrow D_{K1}(X) = p$. Z toho vidíme, že útočník stačí cracknout 2 klíče o délce 2^{56}. - $K_1 = K_2$: 112 b klíč, ale jsou známe known chosen-plaintext útoky, které sníží bezpečnost na 80 bitů. Je to bezpečnější než $E(E(p))$, protože chrání před meet-in-the-middle útokem.

AES

:movie_camera: AES explained - Computerphile

- Použití: FIPS standard, HW implementace v Intel CPUs
- Délka bloku: 128 bitů
- Délka klíče: 3 varianty: 128, 192 nebo 256 bitů
- Počet kol (*rounds*): záleží na délce klíče: 10, 12 nebo 14 kol.

AES nepoužívá Feistelovu síť. Jeho stav je reprezentován 4 × 4 maticí bytů. *S*-box přidává *confusion* (=substituce, maskuje pravý obsah). Posouvání v rámci řádku umožňuje *diffusion* (=transpozice, roztáhnutí bitů zprávy). Jednotlivé fáze jednoho kola: 1. **SubBytes**: nelineární funkce (*S*-box) transformuje jednotlivé byty do matice 2. **ShiftRows**: jednotlivé řádky jsou posunuty doleva, první řádek o 0 bytů, druhý řádek o 1 byte, třetí řádek o dva byty, čtvrtý řádek o 3 byty. 3. **MixColumns**: Každý sloupec je vynásoben *constant* maticí. 4. **AddRoundKey**: matice je XORována pomocí *round* klíče.

Asymetrická kryptografie

Problém symetrické kryptografie je distribuce klíče - jak přes nezabezpečený kanál pošlu svůj symetrický klíč kamarádovi, který žije na druhé straně zeměkoule? Asymetrická kryptografie řeší tento problém použitím dvojice klíčů: veřejný klíč, který je *veřejný* a privátní klíč, který je znám pouze majiteli. Asymetrická kryptografie má 3 hlavní použití: - šifrování/dešifrování - digitální podpisy - inicializace sdíleného tajemství

Její nevýhodou ale je pomalejší výpočet a potřeba extensivního key managementu - *public key infrastructure*.

RSA (Rivest, Shamir, Adleman)

RSA je nejrozšířenější public-key cryptosystem. Je založen na výpočetní náročnosti faktorizace prvočísel. Poprvé publikován v roce 1977, v tuto chvíli je standardizován jako PKCS #1.

Proces: 1. Alice vygeneruje dvě velká prvočísla p a q a vypočítá $n = pq$. - Alespoň 2048-bit dlouhé N je doporučeno. 2. Spočítá hodnotu Eulerovy funkce $\varphi(n) = (p-1)(q-1)$ 3. Zvolí celé číslo e menší než $\varphi(n)$, které je s ním nesoudělné - tzn. obě dvě čísla nemají jiného společného dělitele, než je 1. - Malá hodnota e je preferovaná pro rychlé šifrování, nicméně $e = 3$ je moc malé a degraduje bezpečnost $\rightarrow 2^{16} + 1 = 65537$ je často používané. 4. Spočítá *modular multiplicative inverse* $d \equiv e^{-1} \pmod{\varphi(n)}$ - Pokud e nebylo nesoudělné k $\varphi(n)$, *inverse* nebude existovat 5. Veřejný klíč je dvojice (n, e). Privátní klíč je d. - Důvěra ve veřejný klíč může být vyřešena pomocí *chain of trust* nebo certifikační autoritou. 6. Šifrování: Bob šifruje jeho zprávu m jako $c \equiv m^e \pmod{n}$ 7. Dešifrování: Alice dešifruje

jeho zprávu jako $c^d \equiv (m^e)^d = m^{ed} = m \pmod{n}$ - Možné to je díky Eulerově teorému

DSA (Digital Signature Algorithm)

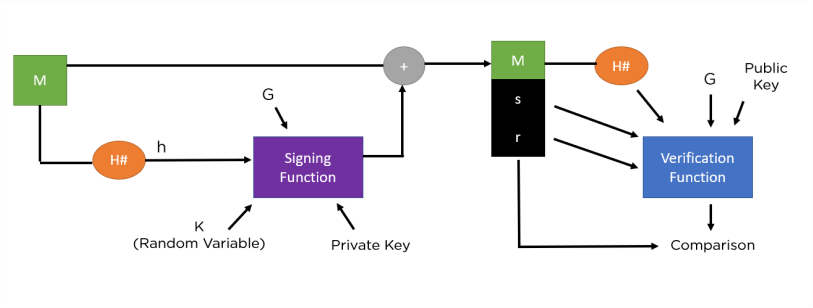
Kromě šifrování umožňuje asymetrická kryptografie taky podepisování. Ve většině případů ale podepisujeme pouze hash zprávy (z výpočetních důvodů - dosáhneme stejného cíle a nemusíme podepisovat obrovské množství dat). Požadavky na digitální podpis: - Musí záviset na podepisované zprávě - Musí používat data, které patří unikátně odesílateli (privátní klíč) - Aby nikdo nemohl podpis zfalšovat - Musí být relativně jednoduchý na vytvoření, rozpoznání a verifikaci - Musí být výpočetně neproveditelné ho zfalšovat

Digitální podpis poskytuje: - Autentizaci odesílatele - Víme, kdo ho odeslal - Integritu - Pokud podpis sedí, víme, že zprávu nikdo po cestě neupravoval - Non-repudiation (neodmítnutelnost) - Odesílatel nemůže popřít, že vytvořil podpis. Pokud mu tedy nikdo nezcizil soukromý klíč.

Základní schéma: - Odesílatel & příjemce. - Předpokládáme, že příjemce zná odesílatelův veřejný klíč. - Případně ho odesílatel může zaslat v zasílané zprávě - nicméně pozor na podvody - útočník se může vydávat za někoho jiného. - Digitální podpis je vytvořen buď z celé zprávy, nebo jejího hashe - důvody viz. výše. - Šifrovat můžeme veřejným klíčem příjemce - Poznámka: Pokud chceme šifrovat i podepisovat, musíme nejdřív zprávu podepsat a až pak zašifrovat - abychom měli jednoznačně spojenou zprávu+podpis.

DSA: - Pouze pro podepisování, 320bitový podpis - 512-3072 bitová bezpečnost - NIST 800-57 doporučuje používat délku klíče 2048 (nebo 3072) po roce 2010 (a 2030). - Menší a rychlejší, než RSA - Bezpečnost závisí na složitosti výpočtu diskrétních logaritmů

Schéma:



ElGamal Signature scheme

Neplést si ElGamal Signature scheme a ElGamal encryption.

Na ElGamal signature scheme byl založen DSA.

As for ElGamal signatures: that scheme is more expensive. ElGamal signatures work modulo a prime p and require one modular exponentiation (for generation) or two (for verification) with exponents as big as p; and the signature is two integers modulo p. This contrasts with DSA and Schnorr, which both work in a subgroup, traditionally a 160-bit subgroup for a 1024-bit modulus. ***DSA and Schnorr are 6 times faster than ElGamal, and produce signatures which are 6 times smaller. This difference in performances is enough to explain not choosing ElGamal****. Indeed, the short size of DSA and Schnorr signatures has long been the selling point for these algorithms when compared to RSA. - https://crypto.stackexchange.com/a/10389*

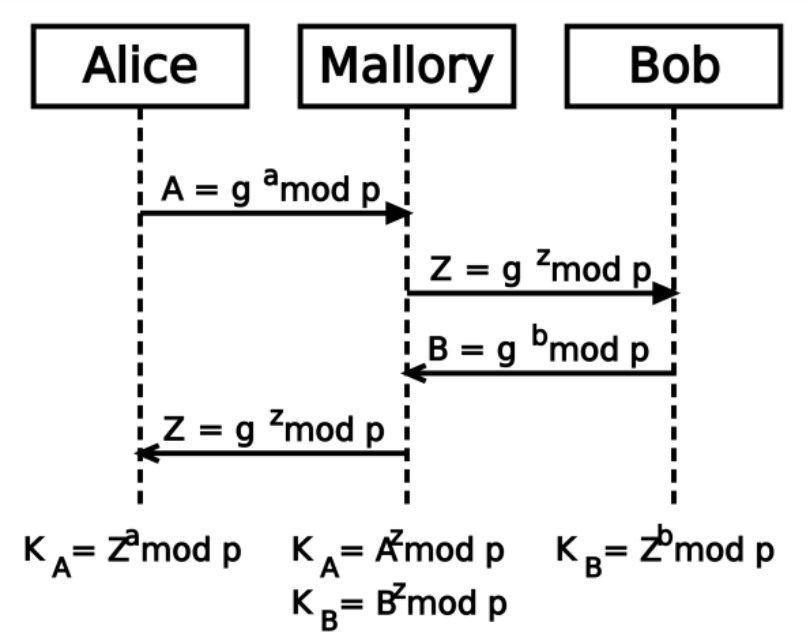
Diffie-Hellman (DH)

Skipuju matematické vzorce, protože důležitejší mi přijde umět to vysvětlit, než vypočítat. inb4 F

Diffie-Hellman key exchange je algoritmus, který umožňuje přes nezabezpečený kanál vytvořit mezi komunikujícími stranami šifrované spojení, bez předchozího dohodnutí šifrovacího klíče. Výsledkem tohoto protokolu je vytvoření symetrického šifrovacího klíče, který může být následně použit pro šifrovaní zbytku komunikace. *Poznámka: DH není*

asymetrický krypto algoritmus, DH je použit v asymetrickém krypto algoritmu ElGamal
Proces: 1. Alice se s Bobem dohodne na společné hodnotě (nonce) 2. Alice i Bob si zvolí privátní hodnotu 3. K společné hodnotě přidají svou privátní hodnotu a výsledky si vymění 4. K vyměněným hodnotám znova přidají svou privátní hodnotu a výsledek je *sdílený klíč*.

Výhodou je, že případný útočník odposlouchávající komunikaci tento klíč nezachytí. Klíč je zkonstruován všemi účastníky komunikace a nikdy není poslán v otevřené formě. Nevýhodou tohoto protokolu je bezbrannost proti útoku Man in the middle, protože neumožňuje autentizaci účastníků. Tento protokol bez kombinace s jinými metodami je tedy vhodný pouze tam, kde útočník nemůže aktivně zasahovat do komunikace.



Hybrid encryption

Hybridní šifrování kombinuje symetrickou a asymetrickou kryptografii. Asymetrická kryptografie může být použita pro ustanovení spojení (autentizace účastníků, výměna symetrických klíčů). Jelikož asymetrické algoritmy jsou pomalejší, zbytek komunikace je šifrováno symetrickým klíčem. Tento způsob je využit např. v TLS, PGP (přílohy emailu jsou zašifrovány AES, symetrický klíč je zašifrován veřejným PGP klíčem příjemce).

Faktorizace a testování prvočíselnosti

Prvočíslo je takové číslo, které je větší než 1 a má právě dva dělitele (jedničku a sebe samé). Prvočísla v kryptografii: - Vynásobit dvě prvočísla je jednoduché, ale extrémně pomalé udělat opak, tj. z *x* získat dvě prvočísla. - Mějme čísla 73 × 79 = 5767, abychom z 5767 zjistili, z jakých čísel se skládá, museli bychom vyzkoušet všechny možnosti, nebo použít určitý algoritmus. - Kryptografické algoritmy využívají velká prvočísla - stovky číslic, např. RSA-2048 má 617 číslic. - V roce 2009 se podařilo faktorizovat RSA-768, které mělo 232 číslic (768 bitů), trvalo jim to 2 roky za použití několika stovek počítačů po celém světě. Na běžném počítači by to trvalo 2000 let.

Jednoduché algoritmy: - Zkusmé dělení: - Brute-force, zkoušíme číslo dělit všemi možnými děliteli. Optimalizované varianty zkouší čísla dělit pouze dvojkou a pak lichými čísly, nebo pouze prvočísly menšími než odmocnina testovaného čísla.

Obecné pravděpodobnostní algoritmy: - **Fermatův test** - Fermatův test je založen na Malé Fermatově větě, která říka, že pokud p je prvočíslo, $a \in \mathbb{Z}$, tak $a^{p-1} \equiv 1 \pmod{p}$ - Např. $p = 5$, $a = 2$. Jelikož 5 je prvočíslo a 2 není násobek 5, má podle malé Fermatovy věty platit, že $2^5 - 2$ je dělitelné 5. Což platí: $2^5 - 2 = 32 - 2 = 30$ je dělitelné 5. - Nevýhodou tohoto postuju je to, že neodhalí tzv. *Carmichaelova* čísla. Proto pokud

Fermatův test platí, říkáme, že p je *asi* prvočíslo.

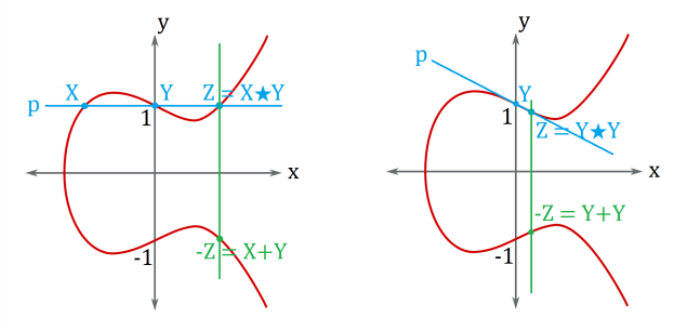
- *Carmichaelovo* číslo je takové $n \in \mathbb{Z}$, že platí $b^{n-1} \equiv 1 \pmod{n}$, pro všechna celá čísla b nesoudělná s n.
  - $561 = 3 \times 11 \times 17$
  - $1105 = 5 \times 13 \times 17$
- **Algoritmus Rabin-Miller**
  - Pravděpodobnostní test prvočíselnosti. Je podobný Fermatovu testu, jelikož je založen na existenci rovností, které jsou splněny pro prvočísla.

Kryptosystémy na bázi eliptických křivek.

:movie_camera: Elliptic Curve Diffie Hellman - dobře vysvětleno :movie_camera: Elliptic Curve - Computerphile

ECC (elliptic curve cryptography) způsob asymetrické kryptografie založený na vlastnostech eliptických křivek. Jeho hlavní výhodou je to, že mnohem menší klíče nám poskytnou stejnou ochranu jako klíče např. RSA. Další výhoda je taková, že hodně útoků založených na faktorizaci a diskrétních logaritmech nefungují na ECC.

Eliptická křivka je dvojice (E, O), kde: - E je set bodů v rovině daných Weierstrassovou funkcí - $y^2 = x^3 + ax + b$ - kde $a, b \in \mathbb{R}$ jsou taková, že $4a^3 + 27b^2 \neq 0$. - $O \in E$ je zvolen jako *"point at infinity"* na křivce.



elliptic curves

Bezpečnost ECC

:movie_camera: Elliptic curve discrete log problem Mějme eliptickou křivku E a body A, B, takové, že $B = kA = (A + \cdots + A)$ - k-krát. Bezpečnost ECC je založena na tom, že je velice složité vypočítat k.

I když máme startovní bod G a konečný bod nG, tak je velice složité zjistit, které číslo je n, tzn. počet kroků, kolikrát jsme museli sečíst bod G, abychom se dostali do bodu nG.

Principy konstrukce hashovacích funkcí

Kryptografická hash funkce je mapování $h : \{0,1\}^a \rightarrow \{0,1\}^b$, kde $a, b \in \mathbb{N}$, $a>>b$, a je libovolné a b má fixní délku pro svou funkci (např. 160 bitů pro SHA1). Jinými slovy, hash funkce h promítne hodnotu z množiny s hodně (nebo dokonce nekonečno) členy na hodnotu z množiny s pevným počtem (méně) členy. - Hash funkce je jednosměrná - Tzn. Ze vstupu dokážeme vypočítat výstup(hash), ale z hashe nedokážeme vypočítat vstup. - Výstup hashovací funkce se nazývá hash, *message digest*, *fingerprint* (otisk).

Vlastnosti dobré hash funkce

- ***Deterministická***: Jeden vstup vyprodukuje vždy stejný výstup
- ***One-way***: Z hashe nelze vypočítat vstup. Jedinou možností, jak získat vstup pro daný hash, je zkoušet všechny možné vstupy (brute-force)

- ***Weak collision resistant***: Pro x je složité najít $H(y) = H(x)$
  - Pro jeden konkrétní vstup nelze najít jiný vstup, který vyprodukuje stejný hash
- ***Strong collision resistance***: Je složité najít jakékoliv x, y, takové, že $H(y) = H(x)$
  - *Weak collision resistance* se vztahuje na nějaký konkrétní vstup, kdežto *strong collision resistance* se vztahuje na jakékoliv dva náhodné rozdílné vstupy
    - Např. Když chceme ke konkrétnímu heslu "*password123*" najít jiný string, který bude mít ve výsledku stejný hash, jedná se o *weak collision*. Příklad pro *strong collision* je třeba birthday paradox, který říká, že stačí pouze 23 náhodně vybraných lidí, abychom měli 50% šanci, že dva z nich budou mít narozeniny ve stejný den.
- ***Strict avalanche criterion***: Změna jednoho bitu na vstupu by měla v průměru změnit polovinu výstupních bitů. Jinak řečeno, malá změna na vstupu by měla způsobit velkou změnu na výstupu.
- ***Fast***: Rychlost výpočtu záleží na případu užití. Obecně ale výpočet hashe by měl být rychlý.

Možnosti použití hash funkcí

- **Integrity check**: Testování integrity zprávy (nebo souboru) tím, že porovnáme její skutečný hash s hashem, který jsme dostali. Dokážeme tím detekovat záměrné nebo nezáměrné chyby - např. útočník pozmění zprávu.
- **Key derivation**: Hesla by neměla být ukládána v plaintextu, ale měla by být uložena jako "hash and salt". *Salt* je náhodná hodnota, která je přidána k heslu, než se zahashuje. Díky tomuto můžeme předejít *rainbow attack*, kde útočník má předpočítané hashe k jednotlivým heslům a uniklé hashe pouze vyhledává, nemusí nic počítat.
  - Kdežto když budeme ke každému heslu připojovat náhodný řetezec, např. "password123&@$" - tomto případě to je **&@$**, útočník bude muset znova cracknout hesla.
  - Pro klasické hashe (MD5, SHA1) musíme salt přidávat ručně, nicméně **bcrypt** má *solení* hesel zabudován v sobě a automaticky ho přidává.
    - Bcrypt: *Salt is the first 22 characters after the third $ in the hash:*
    - *$2y$13$<this is the salt, 22 chars><this is the password hash>*
- **Commitment scheme**: Viz. příklad z otázky 1 - remote coinflip
  - Alice s Bobem si chtějí na dálku hodit mincí. Alice pošle Bobovi hash svého tipu + nějaký náhodný řetězec (protože by Bobovi stačilo pouze vyzkoušet 2 hodnoty), Bob flipne mincí, oznámí Alice svůj výsledek, Alice mu zašle plaintext+salt svého tipu, Bob to vypočítá a je zajištěno, že nikdo nemohl v průběhu změnit svůj tip.
- **IDs** (non-cryptographic): verzovací systémy používají hash funkce, aby identifikovali jednotlivé soubory a commity. Podobně to funguje i v hash tabulkách.

Merkle-Damgard constrution of hash function

:movie_camera: Merkle Damgard and Sponge Constructions

- Merkle-Damgard konstrukce je asi nejznámější konstrukce hashovacích funkcí
- Používá se např. v MD5, SHA1 a SHA2
- Message M je rozdělena do bloků o fixní délce $M = m_0||m_1||\ldots||m_t$ a je použit vhodný *padding*
  - Například budeme zprávu rozdělovat do bloků o délce 16 bitů a zpráva o délce 25 bitů bude rozdělena na $16b||9b$ + *padding*
- Jednotlivé bloky zprávy jsou *compressed* jeden po druhém funkcí f, aby vyprodukovali h_i
- h_i je tzv. *chaining variable* a je použita v kompresi dalšího bloku m_{i+1}
  - Compression funkce f využije m_i a h_{i-1}, aby vyprodukovala h_i
  - počáteční hodnota h_0 je fixní a je specifikována v rámci hash funkce.

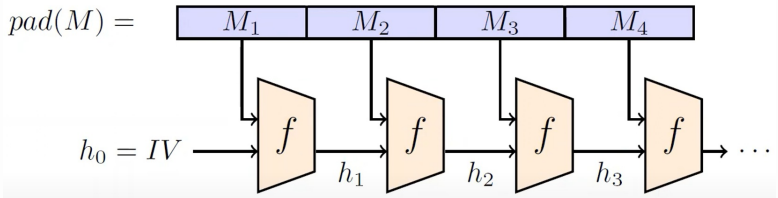
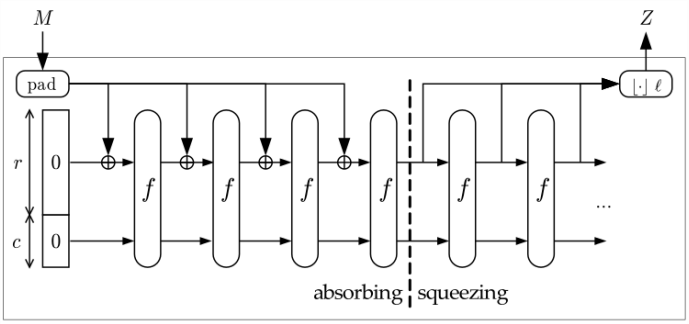


Figure: Merkle-Damgard Construction

Sponge construction of hash functions: SHA-3 Keccak

- SHA-3 (Secure Hash Algorithm) je standard, Keccak je název algoritmu.
- Délka hashe: 224 bitů do 512 bitů
- SHA-3 využívá Sponge a duplex konstrukce
- Je designovaný pro hashování, stream encryption, MAC
- SHA-3 pracuje s bloky o velikosti *w* (běžně 64 bitů; *padding* je použit, pokud je to potřeba). Uchovává si interní stav $5 \times 5 \times w$ jako pole, které se nazývá *sponge*.
- SHA-3 má dvě hlavní fáze: V první jsou data *absorbed* do *sponge* a ve druhé je výsledek *squeezed*.



sponge

keccak

Password hashing functions

brypt (1999) je hash funkce určena hashování hesel. Je založena na Blowfish block cipher. Má v sobě automaticky zahrnut salt, viz. výše. - $**2a**$**12**$R9h/cIPz0gi.URNNX3kh2OPST9/PgBkqquzi.Ss7KIUgO2t0jWMUW - **2a** - specifikuje verzi algoritmu - **12** - specifikuje *input cost*, tzn. počet iterací - čím víc, tím pomalejší je výpočet = zpomalení útočníka. **scrypt** (2009) je další hash funkce pro hashování hesel, navrhnuta tak, aby byla náročně vypočitatelná pomocí custom HW.

TABLE 1. Estimated cost of hardware to crack a password in 1 year.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1	$1.5T
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k	$1.5 × 10^15
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k	$2.2 × 10^17
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M	$48B
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M	$6 × 10^19
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M	$11 × 10^18
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M	$1.5T
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B	$2.3 × 10^23

hash comparison