Merkleovy stromy

- listy hešují bloky vstupu (lze počítat paralelně)
- vnitřní vrcholy hešují výsledky svých synů
- pozor, kořen je potřeba odlišit! Jinak bych mohl vytrhnout podstrom, uložím hladinu → k vrcholu přihešuji flag
- → kódování Sakura (součást specifikace kolem SHA-3)
- výhody: - paralelizace
  - random-access ověřování i update

MESSAGE AUTHENTICATION CODES (MACs, symetrické podpisy)

- obecně: funkce na generování a ověřování podpisu
  - ↳ pokud je deterministická, ověření je memcmp
- můžeme si představit jako keyed hash
  - pro náhodný klíč se má chovat jako náhodná funkce
- příklad: $MAC(k,X) = hash(k \| X)$
  - ! rozbité pro hеše typu Merkle-Damgård → extension attacks!
    - ↳ ale pro ideální hash funguje a pro SHA-3 také (spec. kMAC)
- co třeba $hash(X \| k)$?
  - to není bezpečné proti obecným kolizním útokům na h (většina výpočtu nezávisí na klíči)
  - ← s klíčem i na začátku je to pro SHA-1, jakž takž OK i dnes
- → konstrukce $HMAC_h(k,X) := H(k \oplus C_{out} \| H(k \oplus C_{in} \| X))$
  - konstanty
  - HMAC nad SHA-1/2 je běžný v internetových protocolech
  - myšlenka: skládám 2 funkce:
    - vnitřní je odolná proti kolizím a bere $\{0,1\}^*$
    - vnější je bezpečná MAC, ale stačí fixní délky
  - jako třeba, ale
- Bezpečnost (CPA): - útočník dostane podepisovací orákulum
  - má vyprodukovat korektní podpis pro zprávu, na kterou se neptal orákula.

- kombinace šifra + MAC:
  1. nezávisle obě (auth & encrypt): MAC prozrazuje info o plaintextu (třeba rovnost)
  2. encrypt-then-MAC: bezpečné
  3. MAC-then-encrypt: často míval padding orákula

- **CBC-MAC** – šifruji pomocí CBC, MAC = poslední blok zašifr. textu
  - nutná konstantní IV (jinak děravé – možno měnit 1. blok zprávy a kompenzovat změnou IV)
  - nesmíme vyhodit jiné zašifrované bloky (jinak truncate/reassemble)
  - umíme dokázat bezpečnost, pokud: ① šifra je ideální, ② množina zpráv je bezprefixová [konst. délky / délka na začátku atd.]

! Nesmíme použít stejný klíč pro šifrování a MAC ze jinak reassemble kombinací 2 zpráv

- **Shannonovsky bezpečný MAC** – předpokládejme one-time klíč
  - pořídíme si rodinu 2-nezávislých heš. funkcí ← ne v kryptograf. smyslu!

$$\mathcal{H} := \{h_k \mid k \in \mathcal{K}\}, \quad \forall k\ h_k : X \to Y$$

$$\forall x, x' \in X,\ x \neq x',\ \forall y, y' \in Y \quad Pr_{h \in \mathcal{H}}[h(x) = y \ \&\ h(x') = y'] = \frac{1}{|Y|^2}$$

je to také integer $|K|$ $\Rightarrow |K| \geq |Y|^2$

Příklad: $X = Y = \mathbb{Z}_p$, $K = \mathbb{Z}_p^2$, $h_{a,b}(x) = ax + b$ — pro x, x', y, y' dostanu soustavu 2 lineárních rovnic pro a, b $\Rightarrow \exists! a,b \Rightarrow Pr = 1/p^2$

  - podepisuji pomocí h_k s náhodným klíčem $k \in \mathcal{K}$
  - jaká je Pr, že po pozorování dvojice $(x, \underbrace{h(x)}_{y})$ mi vyjde tip (x', y')?

$$Pr[\text{útok vyšel}] = Pr_h[h(x') = y' \mid h(x) = y] = \frac{Pr[h(x) = y \ \&\ h(x') = y']}{Pr[h(x) = y]} = \frac{1/|Y|^2}{1/|Y|} = \frac{1}{|Y|}$$

Pr[h(x)=y] ← postčítáním Pr z definice $\mathcal{H}$ přes všechna y'

→ nepřekonám náhodné tipnutí podpisu.

klíč musí být aspoň 2x delší než zpráva – cvič.

- **Praktická implementace**: pořídím si nonci, z té šifrováním taj. klíčem odvodím pseudonáhodný klíč pro $\mathcal{H}$
- **Aproximace**: $(2,c)$-nezávislost … $Pr[\ldots = d] \leq c/|Y|^2$
  … např. $((ax+b) \bmod p) \bmod m$ je $(2,4)$-nezávislé.

Jak se změní Pr úspěšného útoku?

$$\frac{Pr[h(x) = y \ \&\ h(x') = y']}{Pr[h(x) = y]} \leq \frac{c/|Y|^2}{1/|K|} = \frac{c \cdot |K|}{|Y|^2}$$

tato je moc slabé, více méně konstanta!

tohle je $\leq c/|Y|$, ale to nám je ve jmenovateli navíc … ovšem také je to $\geq 1/|K|$

- **Polynomiální MAC** ... opět nad tělesem $\mathbb{Z}_m$, klíče $\in \mathbb{Z}_m^2$

$h_{a,b}(x_1 \dots x_n) = x_1 \cdot a^n + x_2 \cdot a^{n-1} + \dots + x_n \cdot a^1 + b$ } snadno spočteme Hornerovým schématem

Falšování pro 2 zprávy téže délky.

$\Pr_{a,b}[h_{a,b}(x) = y \;\&\; h_{a,b}(x') = y']$

... odečtením rovnic: $(x_1 - x_1')a^n + (x_2 - x_2')a^{n-1} + \dots + (x_n - x_n')a^1 = y' - y$

čili a musí být kořenem nějakého polynomu stupně nejvýš n $\Rightarrow$ takových a je max. n

... pro každé takové a $\exists!\, b$ takové, že platí i druhá rovnice.

$\Rightarrow \Pr \le n/m^2$.

$\Pr_{a,b}[h_{a,b}(x) = y] = 1/m$ ... pro každé a $\exists!\, b$, pro které to platí.

$\Rightarrow$ Pr[útok uspěje] $\le n/m$. – tedy chci $m \ge n^2$, abych Pr sfláčil pod narozeninové útoky.

- **Mód blokových šifer GCM** [Galois/Counter Mode], populární s AES
  - pracuje v tělese $GF(2^{128})$: sčítání je XOR, násob. je CLMUL
  - autentikuji nezašifrovaná data $A_1 \dots A_m$ a $Y_1 \dots Y_n$ $\{Y_i = X_i \oplus E_k(IV+i)\}$, za to přilepím blok kódující m, n a blok $E_k(IV)$.
  - výsledné bloky dají koeficienty polynomu, ten vyhodnotím v bodě $E_k(0)$
  - $\Rightarrow$ je to polynomiální MAC s klíčem generovaným blokovou šifrou

- **Poly1305** [Bernstein 2005]
  - poly-MAC v tělese $GF(2^{\overbrace{130}^{\text{prvočíslo}} - 5})$ ... o něco větší rozsah než 128b bloky, výsledek nakonec modulime 2^{128}
  - každý blok se padduje (to se vždy vejde ☺)
  - potřebuji nonci a tajný klíč (k, r) [k je 128b, r má nějaké bity konst. → jen 106b]
  - polynom vyhodnotím v bodě r a přičtu $E_k(\text{nonce})$
    - r mod 2^{128}
    - tohle už mod 2^{128}
    - to zjednodušší implementaci aritmetiky

Jelikož přičítám "one-time pad", z žádné odchycené zprávy se nedozvím nic o r $\Rightarrow$ není třeba pokaždé jiné r

Pr[útočník uspěje] $\le$ délka zprávy $/ 2^{106}$, což je ok pro zprávy rozumných délek.

- původně specifikován s AES, dnes se často kombinuje s ChaCha 20.

NÁHODNÉ GENERÁTORY

<u>Požadavky:</u> Útočník ani se znalostí předchozího výstupu nedovede (efektivně) předpovídat budoucí výstup.
[to speciálně implikuje statistickou rovnoměrnost]

<u>Možná řešení:</u> → pseudonáhodný generátor (třeba šifra v CTR módu)
→ HW generátor náhodnosti
- šum na odporu / diodě apod.
- radioaktivní zářič
- průchod/odraz fotonu na polopropust. zrcátku
- lávové lampy
- rádiový šum
- kruhový oscilátor
- timing kláves / disku / sítě ...

} pozor, útočník může tyto jevy také měřit a případně ovlivňovat

→ kombinace obojího – /dev/random a spol. } máme-li reálnou náhodu, pomůže nám; pokud ne, stále je to PRNG
- RDRAND v procesoru (jak moc důvěryhodný?)
  └ metastabilní oscilátor, automatická kontrola anomálií, lepší PRNG založ. na AES

<u>Problémy:</u>
- Je-li vnitřní stav kompromitován, potřebujeme přidat hodně entropie najednou, jinak útočník probere všechny možnosti a určí nový stav → pooling, odhadování entropie (šarlatánství...)
- Inicializace po bootu → ukládání stavu, riziko rollbacku

<u>Fortuna</u> [Ferguson, Schneier 2003] ... elegantní RNG, který nepotřebuje odhadování entropie zdrojů

- <u>Generátor</u>
  - používá AES s 256b klíčem
  - šifruje 128b počítadlo (nikdy nepřeteče)
  - po vygenerování nejvýše 2^{16} bloků (nebo požadovaného množství dat) vygeneruje nový klíč (CTR neopakuje bloky, na to by se časem přišlo), ale neresetuje počítadlo (tím rozbije potenciální krátké cykly)
- <u>Akumulátor</u>
  - sbírá externí náhodnost do kyblíčků $P_0 - P_{31}$, každý zdroj náhodnosti přidává j-tý vzorek do $P_{j \bmod 32}$.

- jakmile P_0 naakumuluje dost vzorků (ale ne častěji
reseed { než jednou za 100 ms), přihešuji jeho obsah ke klíči generátoru a v i-tém kroku ještě všechny P_j pro $2^i \setminus j$. Použité kyblíčky vysypu.
  - ⇒ z kompromitovaného stavu se časem vzpamatuji (čas závisí na rychlosti přitékání entropie) .. přesněji:
    - nechť za 1 krok reseedování přiteče ϱ bitů entropie
    - 128 bitů určitě stačí k zotavení
    - pokud $\varrho \geq 128$, zotavíme se příštím reseedem (P_0 použijeme vždy)
    - jinak se zotavím po reseedu z P_i takového, že

$$128 \leq 2^i \cdot \varrho/32 < 256$$

(pod $2^i \cdot \varrho/32$: přítok do 1 kyblíčku; u < 256: jinak mohu zmenšit i)

    - čili chci $2^{12} \leq 2^i \cdot \varrho \leq 2^{13}$

$$\frac{2^{12}}{\varrho} \leq 2^i \leq \frac{2^{13}}{\varrho}$$

(u 2^i: # kroků na zotavení)

BEZPEČNÝ KANÁL (příklad z Practical Crypto)

- Alice a Bob mají unikátní tajný klíč (pro ∀ kanál jiný)
- chtějí obousměrně komunikovat
- jeden pošle m_1 — m_r, druhý přijme podposloupnost (a ví, kterou)
- zkombinujeme:
  - šifru (třeba AES/CTR)
  - MAC (after encrypt)
  - sekvenční číslo

  } pro každý směr zvlášť

    ↳ po 2^{32} zprávách chceme měnit klíče
  - odvozování klíčů (2x šifra, 2x MAC) hešovací funkcí z hlav. klíče

---

Pozn. k Linuxovému /dev/(u)random
- pooly míchá pomocí CRC (tady stačí inf.-teor. mixování)
- PRNG založen na ChaCha20
- entropy estimators