- **ElGamalův podpis** (založený na dlogu)
  - Parametry: p, g (jako u DH)
  - Tajný klíč: $k \in_R \{2 \dots p-3\}$
  - Veřejný klíč: $a = g^k \bmod p$
  - Podpis(x): $t \in_R \{2 \dots p-3\}$ tž. $t \perp p-1$
    $r \equiv g^t \pmod p$
    $s \equiv (x - kr) \cdot t^{-1} \pmod{p-1}$
    $\rightarrow$ podpis (r, s)
  - Verify(x, r, s): ① $0 < r < p$, $0 < s < p-1$
    ② $g^x \equiv a^r \cdot r^s \pmod p$
  - proč funguje: $a^r \cdot r^s \equiv g^{kr} \cdot g^{t(x-kr)t^{-1}} \equiv g^x$
  - opět nepříjemné algebraické vlastnosti $\Rightarrow$ hašujeme
  - lze provést v jakékoliv grupě, kde je dlog těžký
    $\Rightarrow$ pak je φ $p-1$ velikost grupy q
    $\Rightarrow$ r pak navíc modulárně q, pokud vyjde 0, přegenerujeme t
  - navíc má zabudované hašování zprávy
- **Digital Signature Algorithm** [USA 1994]
  - Parametry: $p = c \cdot q + 1$ (p je ~4Kbit, q cca 256b),
    g je generátor podgrupy $\mathbb{Z}_p^*$ velikosti q (tedy c-tá mocnina generátoru $\mathbb{Z}_p^*$)
  - Tajný klíč: $k \in_R \{1 \dots q-1\}$
  - Veřejný klíč: $a = g^k \bmod p$
  - Sign(x): $t \in_R \{1 \dots q-1\}$
    $r \equiv (g^t \bmod p) \bmod q$, pokud $r \neq q$, ↺ zkusit znovu
    $s \equiv t^{-1} \cdot (\text{hash}(x) + kr) \bmod q$, pokud $s \neq q$ ↺ restart
    $\rightarrow$ podpis (r, s) (příjemně krátký)
  - Verify(x, r, s): $s^{-1} \bmod q$
    $u_1 \equiv \text{Hash}(x) \cdot s^{-1} \pmod q$
    $u_2 \equiv r \cdot s^{-1} \pmod q$
    check $(g^{u_1} \cdot a^{u_2} \bmod p) \equiv r \pmod q$

Nelze přímo vyrobit z ElGamalovy šifry, neboť ta nemá šifrovat tajným klíčem a dešifrovat veřejným

- Proč to funguje: z $s \equiv_q t^{-1}(h(x)+kr)$

dostaneme $t \equiv_q s^{-1}(h(x)+kr) \equiv \underbrace{s^{-1}h(x)}_{u_1} + \underbrace{s^{-1}kr}_{u_2 \cdot k}$

Proto $g^t \equiv_p g^{u_1} \cdot \underbrace{g^{u_2 k}}_{a^{u_2}}$

↓ tohle dá r po modulo q

- Podobně ECDSA s eliptickou křivkou: místo $g^{\text{něco}}$ operace v grupě křivky.

! Ve všech variantách DSA / ElGamala nesmíme opakovat t

Pro obyč. ElGamala:

① Pokud se prozradí t, pak: víme $s \equiv (x-kr)\cdot t^{-1} \pmod{p-1}$ (vše kromě k známe)

$ts \equiv x-kr$

$kr \equiv x-ts$

$k \equiv (x-ts)\cdot r^{-1}$ a máme taj. klíč

② Pokud zopakujeme t, zopakuje se i r ⇒ pro zprávy x_1, x_2 máme podpisy (r,s_1), (r,s_2).

z definice: $g^{x_1} \equiv_p \underbrace{a^r}_{g^{kr}} \cdot \underbrace{r^{s_1}}_{g^{ts_1}} \Rightarrow x_1 \equiv_{p-1} kr + ts_1$

$\Rightarrow x_2 \equiv_{p-1} kr + ts_2$

$x_1 - x_2 \equiv_{p-1} t(s_1 - s_2)$

⇒ kdykoliv $s_1 - s_2 \perp p-1$, umíme zjistit t ⇒ ①

Oprava (dnes již běžná): t generuji PRNG současným zprávou x (třeba pomocí hašovací funkce)

Typické protokoly

typicky pak symetrická šifra a MAC

① vyjednáme si nonce (proti replay)
① vygeneruji náh. klíč (master secret)
② pošlu zašifrovaný veř. klíčem protistrany
③ oba podepíšeme dosavadní průběh protokolu
④ odvodíme klíče odvozené z hlav. klíče

nebo DH výměna klíčů ⇒ pak získám dopředná bezpečnost

(ani uvařením taj. klíčů nemůžeme dešifrovat staré zprávy)

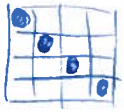
Implementační záležitosti

- neumíme navrhovat bezpečný SW ... největším nepřítelem je komplikovanost
- neumíme ho ani implementovat
  - testování bezpeč. chyby neodhalí (fuzzing trochu pomůže)
    ... ale při útoku na to obvykle spoléháme
  - píšeme v Céčku ⇒ buffer overruns
  - nepíšeme v Céčku ⇒ side channels je nemožné kontrolovat
- příliš mnoho závislostí: SW i HW
  ... navíc většina z nich není optimalizovaná na bezpečnost
- reálný útočník má k dispozici víc informací a možností zásahu než útočník teoretický ☹ (různé postranní kanály)
  - útočník po síti: - časovací útoky (memcmp, padding oráckla, ...)
    - generování nekorektních dat / nejednoznačných
      → útoky na parser
      → v jakém formátu jsou data?
      (JPEG ∩ Java, UTF-8 malformed ...)
      zero-terminated / counted strings
    - zahlcení příjemce, zahazování stavu

    (Chceme, aby primitiva trvala vždy stejně dlouho)
  - v téže místnosti: - měření spotřeby (modular exponentiation v RSA)
    - elmag. záření
    - zvuk (klávesy, pískání měničů) → lze snímat chvění okem
    - tepelné stopy (třeba po heslo)
    - aktivování výpočtu elmag. pulsy, napájením atd.
  - fyzický přístup k počítači:
    - "cold boot attack" (vč. tekutého dusíku)
    - přidání špehujícího HW
    - otevřený ve vypnuté paměti
  - jiný program na témže stroji: (třeba Javascript)
    - HW side-efekty (třeba keše) ← zejména s HyperThreadingem
    - CPU bugs

Příklad: Útok na AES (128b klíče) ... synchronní verze (na totožném stroji během testu před/po AES), known plaintext

- Typ. implementace rundy: ① xor rundového klíče (v první rundě navíc whitening) ② tabulky 256 × 32b



$\rightarrow (T_0(a_0) \oplus T_1(a_1) \oplus T_2(a_2) \oplus T_3(a_3))$

1. řádek výsledku

a podobně pro další 3 řádky: posunuté diagonály, týž tabulky

↓

10 rund, poslední atypická (jiné 4 tabulky)

- Keš má 64B bloky ⇒ 16 položek tabulky na blok ⇒ z čísla bloku poznáme 4 bity stavu ⇒ pro známý plaintext 4 bity klíče
- ovšem v dalších rundách se do keše otisknou další přístupy do tabulek ...

Nechť j je blok v T_i, do kterého se neotiskne 1. přístup a vyčerpáme náhodně

... a provedeme ještě $9 \cdot 4 - 1 = 35$ přístupů do téže tabulky

$\Rightarrow Pr[\text{žádný z nich se netrefí do } j] = (1 - 1/16)^{35} \doteq 0.1$

$\Rightarrow$ při malém počtu náhodných pokusů izolujeme jediný řádek, ke kterému se přistoupí vždy

- z první rundy máme 64 bitů klíče, trochu sofistikovanější útok na 2. rundu pak dává zbylých 64 ☺
- Nyní nejlepší: řádově stovky pokusů bez znalosti plaintextu (!)
- Obrana: bit-slicing implementace S-boxů (toto je problém každé šifry s velkými S-boxy)
  HW implementace AES (v CPU)

Ukládání tajemství

- klíče v paměti: mohou skončit na disku (swap, core dump, ...) nebo na síti (po uvolnění paměti někdo alokuje paket a inicializuje jen částečně)
- klíče v registrech: mohou skončit v paměti (typ. zásobník)
- best practices: - mlock & vypnutí core dumpů
  - po použití smazat
  - případně dělení tajemství na 2 části

- tajná data na disku: nutno přemazat (jak dlouho?)
  - pozor na: FS (fragmenty putují...)
    realokaci vadných sektorů
    posouvání stop na řádu let
- SSD: mazat prostě nemůžeme
- existují disky se "safe erase" – interně šifrují AES, pak smažou klíč z EEPROM
  – některé přístroje při švindlování

Dedikovaný HW

- často v "tamper-proof" provedení – štíty na vyhřívání napájení, Faradayův klec
  (smart-karty, USB tokeny, TPM apod.)
  - jednoduchý deterministický procesor
  - self-destruct při otevření
  - randomizovaná struktura čipů

⇒ luštění je poměrně náročné, ale umí se.

Důležité: Důvěra ve firmware – většinou chyby. (autor má svou vlastní agendu...)

Udržování stavu

- nonce, sériová čísla, stavy RNG apod. se nesmí zapomenout!
- problémy: reboot, power-cycle, obnovení ze zálohy

Poučení

- nic není dokonalé, vše jde překonat...
- inspirujme se fyzickou bezpečností:
  - cena útoku > cena chráněných dat
  - útočníka chceme zdržet, aby byl chycen (⇒ motivace neútočit)
  - logy, monitorování, ..., aby bylo chycení snazší
  - "požární cvičení"
  - penetration testing