Kriptográfia és Információbiztonság
6. előadás

MÁRTON Gyöngyvér

Sapientia Egyetem, Matematika-Informatika Tanszék
Marosvásárhely, Románia
mgyongyi@ms.sapientia.ro

2024

Miről volt szó az elmúlt előadáson?

- hash függvények (hash function)
- üzenethitelesítő kódok (message authentication code)
- hitelesített titkosítás (authenticated encryption)

Miről lesz szó?

- publikus kulcsú rendszerek, alapfogalmak
- matematika modellek,
- publikus kulcsú titkosítók: RSA, RSA-OAEP
- digitális aláírások: matematikai modell
- digitális aláírások: RSA, RSA-RSS

A publikus-kulcsú rendszerek, alapfogalmak

- elméleti alapjait 1976-től kezdve dolgozták/dolgozzák ki:
  - 1976 *Diffie-Hellman cikk*
  - 1977, *Rivest-Shamir-Adleman cikk*
  - 1978, *Rabin cikk*
- nem helyettesíti a szimmetrikus kriptográfiát, de kis méretű (pár kilobájt) információ esetén alkalmas bizalmas információcserére is
- a rendszerek biztonsága matematikai problémákon, feltételezéseken alapszik
- az alapműveletek nem a helyettesítés és permutáció, hanem egész számokkal vagy számpárokkal végzett algebrai műveletek
- a rendszerekben megkülönböztetnek publikus kulcsot és privát kulcsot, ahol a privát kulcsok szigorúan titkosak, a publikus kulcsokat pedig mindenki ismeri

A publikus-kulcsú rendszerek, alapfogalmak

felosztása:

- kulcscserék (key exchange, KE): lehetővé teszik, hogy a kommunikáló felek megeggyezenek egy szimmetrikus titkosító kulcsának az értékében:
  - pl. a küldő és fogadó fél is generál egy-egy random értéket, amelyeket felhasználva megtudnak állapodni egy AES-256 kulcsban
- publikus kulcsú titkosítók (public key encryption, PKC): lehetővé teszi, hogy a kommunikáló felek megosszák egy szimmetrikus titkosító kulcsát
  - pl. a küldő fél generál egy AES-256 kulcsot, amelyet PKC-t alkalmazva elküld a fogadó félnek
- digitális aláírások (digital signature, DS):
  - pl. kulcscserénél, publikus kulcsú titkosítóknál: publikus kulcsok hitelesítése,
  - pl. kriptovaluták esetében a tranzakciók hitelesítése

A legismertebb publikus-kulcsú rendszerek

- Diffie-Hellman kulcscsere, biztonsága a diszkrét logaritmus probléma nehézségén alapszik
- RSA (Rivest-Shamir-Adleman), biztonsága azon alapszik, hogy nehéz meghatározni valamely összetett szám prímosztóit (faktorizációs probléma): alkalmas titkosításra és digitális aláírásra
- Rabin, SAEP mindkettő biztonsága a faktorizációs és kvadratikus maradék problémán alapszik, alkalmasak titkosításra, a Rabin digitális aláírásra is
- ElGamal, biztonsága a diszkrét logaritmus probléma nehézségén alapszik, a Diffie Hellman módosított változata, alkalmas titkosításra és digitális aláírásra
- ECC, elliptikus görbén alapuló kriptográfia, biztonsága az elliptikus görbe diszkrét logaritmus probléma nehézségén alapszik, alkalmas kulcscserére, titkosításra és digitális aláírásra
- Blum-Goldwasser kriptorendszer, biztonsága a faktorizációs és kvadratikus maradék problémán alapszik, titkosításra alkalmas

A publikus-kulcsú rendszerek, alapfogalmak

- matematikai modellek segítségével tárgyalhatóak az algoritmusok biztonsága
- általában három algoritmust szükséges értelmezni,
- a következő modellekben K a kulcsok, M az üzenetek, D a publikus adatok halmazát fogja jelenteni,
- a kulcscsere mechanizmusok esetében:
  - mindig egy interaktív protokollról beszélünk,
  - a két kommunikáló fél (eszköz) egyidőben kell online legyen,
  - feltételezzük, hogy a két kommunikáló fél A és B.

A kulcscsere mechanizmusok matematikai modellje

- Gen, a publikus adatokat generáló algoritmus, polinom idejű, véletlenszerű, meghatározza a d-t:

$$d \xleftarrow{R} Gen(1^k),$$

  ahol $d \in D$, és k a rendszer biztonsági paramétere, a generált adatok bithossza,
- a $PGen(d, \cdot)$ algoritmus polinom idejű, véletlenszerű, ezt mindkét fél végrehajtja:
  - a kapott A, B értékek a publikus kulcsok, ezeket megosztják egymással, az a, b értékek a privát kulcsok, ezek szigorúan titkosak:

$$\begin{aligned} \text{A} &: A \leftarrow PGen(d, a), \quad a \xleftarrow{R} M \\ \text{B} &: B \leftarrow PGen(d, b), \quad b \xleftarrow{R} M \end{aligned}$$

- a $KGen(sk, \cdot)$, $sk \in \{a, b\}$ algoritmus polinom idejű, determinisztikus, a $PGen$ után mindkét fél végrehajtja:

$$\begin{aligned} \text{A} &: key \leftarrow KGen(a, B), \\ \text{B} &: \hat{key} \leftarrow KGen(b, A) \end{aligned}$$

A helyesség fennáll, ha minden $a, b \in M$-re:

$$key = KGen(a, B) = KGen(a, PGen(d, b)) = KGen(b, PGen(d, a)) = KGen(b, A) = \hat{key}.$$

A publikus kulcsú titkosítók matematikai modellje

- Gen, a kulcs-generáló algoritmus, polinom idejű, véletlenszerű, meghatározza a pk - publikus kulcsot, és az sk - privát kulcsot:

$$(pk, sk) \xleftarrow{R} Gen(1^k),$$

  ahol $(pk, sk) \in K$ és k a rendszer biztonsági paramétere, legtöbb esetben a generált kulcs bithossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$

- $Enc(pk, \cdot)$ a rejtjelező algoritmus, polinom idejű, véletlenszerű, valamely $m \in M$-re meghatározza a c rejtjelzett üzenetet:

$$c \xleftarrow{R} Enc(pk, m),$$

- a $Dec(sk, \cdot)$ a visszafejtő algoritmus, polinom idejű, determinisztikus, visszafejti a c rejtjelezett üzenetet:

$$m \leftarrow Dec(sk, c).$$

A helyesség fennáll, ha minden $m \in M$ esetében:

$$Dec(sk, Enc(pk, m)) = m.$$

A digitális aláírás matematikai modellje

- Gen, a kulcs-generáló algoritmus, polinom idejű, lehet véletlenszerű is, meghatározza a pk - publikus kulcsot, és az sk - privát kulcsot:

$$(pk, sk) \xleftarrow{R} Gen(1^k),$$

  ahol $(pk, sk) \in K$ és k a rendszer biztonsági paramétere, legtöbb esetben a generált kulcs bit-hossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$

- $Aut(sk, \cdot)$ a hitelesítő algoritmus, polinom idejű, véletlenszerű aláírja az m üzenetet:

$$(m, c) \xleftarrow{R} Aut(sk, m),$$

- a $Ver(pk, \cdot)$ az ellenőrző algoritmus, polinom idejű, determinisztikus, ellenőrzi, hogy az m üzenet hiteles-e, kimenete `True` vagy `False` aszerint, hogy $m = \hat{m}_1$-el vagy sem:

$$\hat{m} \leftarrow Ver(pk, c).$$

A helyesség fennáll, ha minden $m \in M$ esetében:

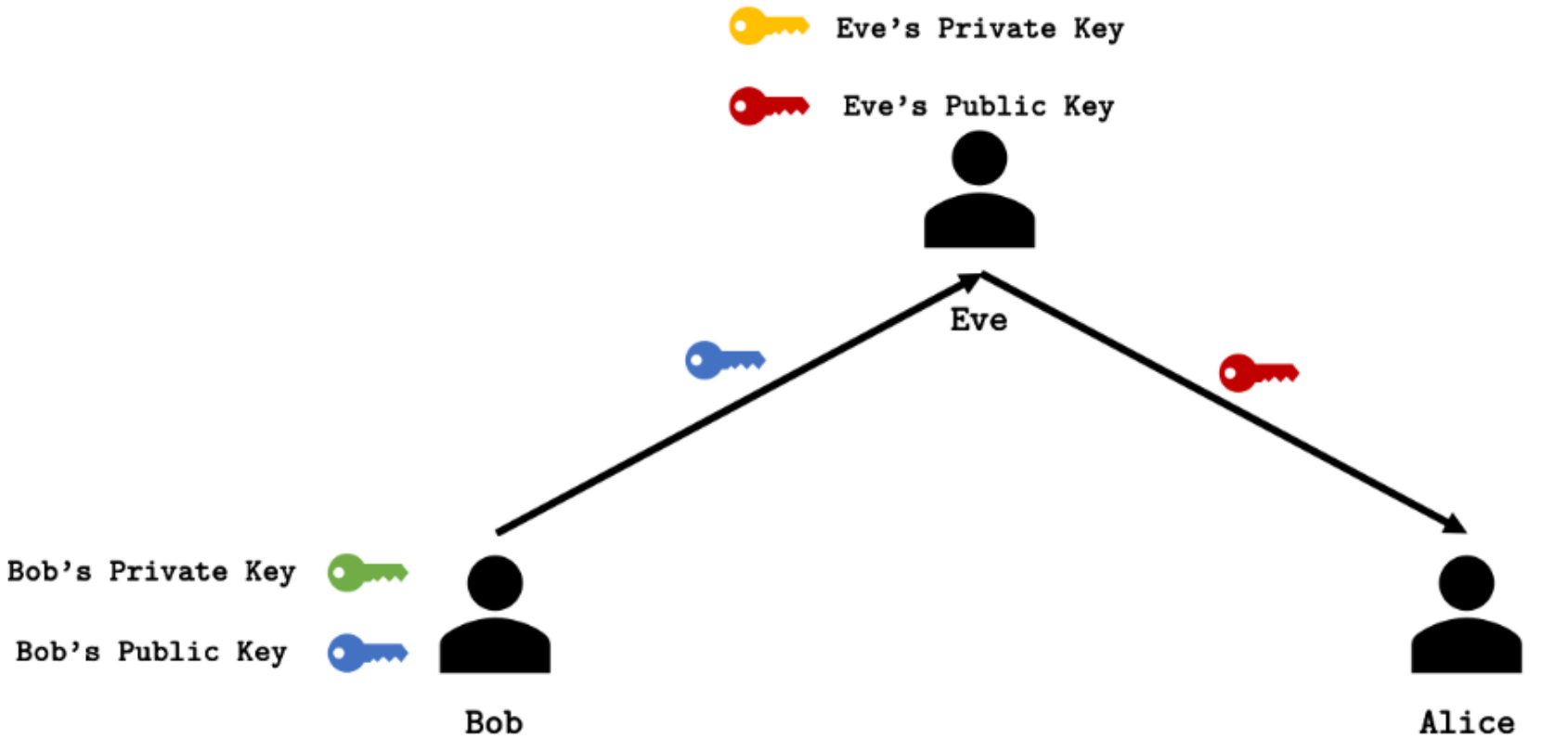
$$Ver(pk, Aut(sk, m)) = m.$$

A publikus-kulcsú rendszerek, követelmények

- az arra jogosult fél (eszköz) hatékony algoritmussal tudja kigenerálni a publikus és privát kulcsot,
- a publikus kulcs ismeretében ne lehessen hatékonyan meghatározni a privát kulcsot,
- az rendszerekben alkalmazott matematikai objektumok átlakíthatóak kell legyenek bájtszekvenciákká, pl.
  - egy szöveg 10-os számrendszerben: 63 169 75 210 184,
  - a szöveg 16-os számrendszerben: $0x3f\ 0xa9\ 0x4b\ 0xd2\ 0xb8$,
  - a szöveghez tartozó szám:
    $63 \cdot 256^0 + 169 \cdot 256^1 + 75 \cdot 256^2 + 210 \cdot 256^3 + 184 \cdot 256^4 = 793802156351.$
- kevés olyan rendszert sikerült kidolgozni, amely eleget tesz a fenti követelményeknek: például kudarcos próbálkozásnak bizonyult a hátizsák feladaton alapuló publikus kulcsú rendszer

A publikus-kulcsú rendszerek, követelmények

- a kulcscseréknél használt publikus kulcsokat hitelesíteni kell!! ezt digitális aláírás használatával lehet megoldani, nem elég a MAC-használat!!
- a mai napig is ez a legsérülékenyebb pontja az adatbiztonságnak



Az RSA rendszer

- a legismertebb, a leggyakrabban, a legtöbbet vizsgált, használt publikus kulcsú rendszer
- alkalmazható kulcscserére (az encryption változata) és hitelesítésre (a digital signature változata)
- a TLS 1.3 protokoll már nem használja az encryption változatot, csak a digital signature változatot
- manapság a 2048 bites kulcsmérettel ajánlott használni
- a gyorsaság miatt kis e-vel kell dolgozni, az $e = 2$ választást a Rabin, illetve SAEP specifikációik tartalmazzák, eltérő műveletvégzést jelent ez
- a biztonság miatt konstans e-vel dolgoznak, amely prímszám, ekkor 17 szorzást kell végezni a titkosítás során:

$$e = 65537 = 2^{16} + 1 = 10000000000000001$$

A textbook titkosító RSA rendszer

A textbook RSA titkosító rendszer: feltételezzük, hogy a kommunikációban résztvevő két egység A és B:

- a *Gen* kulcs-generáló algoritmus segítségével meghatározásra kerülnek a $pk = (n, e)$, illetve $sk = (n, d)$ értékek, ahol:
  - $n = p \cdot q$, p, q prímszámok, $\phi = (p-1) \cdot (q-1)$
  - $e \xleftarrow{R} \{3, \ldots, \phi\}$, úgy hogy $gcd(e, \phi) = 1$
  - a kiterjesztett eukleidészi algoritmussal meghatározzuk d-t, azaz e inverzét $\pmod{\phi}$ szerint: $e \cdot d = 1 \pmod{\phi}$
  - a privát kulcs értékét ugyanakkor $\pmod{[p-1, q-1]}$ szerint is meg lehet határozni, ahol $[p-1, q-1]$ a legkisebb közös többszöröst jelöli. Az RSA standard így jár el.
- az A egység az $Enc((e, n), m)$-vel meghatározza c-t, majd elküldi B-nek:

$$c = m^e \pmod{n}.$$

- a B egység a $Dec((d, n), c)$ algoritmussal meghatározza $\hat{m}$-t:

$$\hat{m} = c^d \pmod{n}.$$

- a megosztásra került titkos információ: $m = \hat{m}$
- helyesség:

$$\hat{m} = c^d = (m^e)^d = m^{ed} = m \pmod{n}.$$

A textbook titkosító RSA rendszer, példa

- Kulcsgenerálás
  - Legyen $p = 61$, $q = 97$ a két prímszám.
  - Meghatározzuk:
    - $n = 61 \cdot 97 = 5917$,
    - $\phi = (p-1) \cdot (q-1) = 60 \cdot 96 = 5760$.
  - Legyen $e = 7$, ahol $(7, \phi) = 1$.
  - Meghatározzuk e inverzét $\pmod{\phi}$ szerint, kapjuk: $d = 823$, mert $7 \cdot 823 = 1 \pmod{5760}$.
  - A publikus kulcs : $(7,\ 5917)$.
  - A privát kulcs : $(823,\ 5917)$.
- Titkosítás:
  - A $K = 2014$ mesterkulcs értéket szeretnék titkosítani/megosztani. Ekkor a titkosított érték: $cK = 2014^{7} \equiv 1526 \pmod{5917}$.
- Visszafejtés:
  - $K = 1526^{823} \equiv 2014 \pmod{5917}$.

Szervezetek, standardizáló intézmények

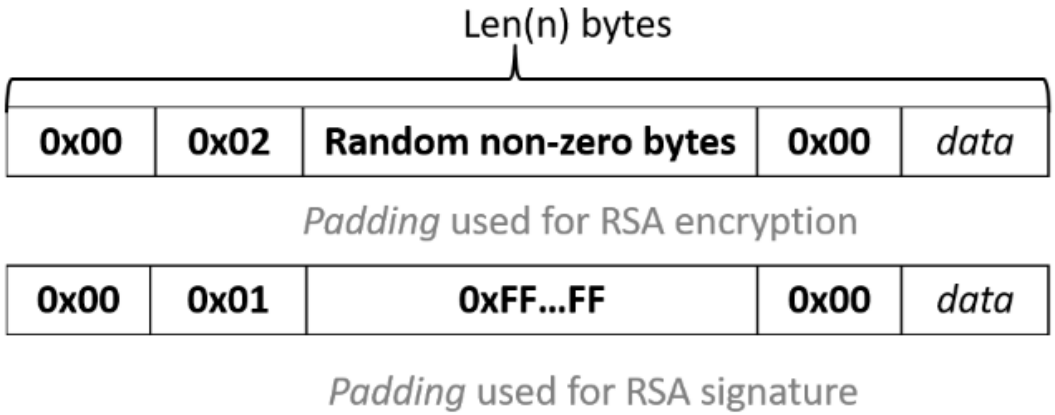
- RSA Security LLC:
  - az RSA tervezői által létrehozott szervezet,
  - általuk kibocsájtott dokumentumok PKCS (Public Key Cryptography Standrads) névvel jelennek meg, amelyek az különböző algoritmusok/protkollok működését, implementációját írják le
- IETF (Internet Engineering Task Force):
  - internetes szabványok kidolgozója
  - az általuk közreadott dokumentumok RFC (Request For Comments) névvel jelennek meg

RSA változatok

- RSA encryption
  - PKCS#1 v1.5
    - 1993, az RSA titkosító első standard változata,
    - az SSL/TLS-ben használták/használják, nem biztonságos!!!!!,
    - *PKCS#1 v1.5*
  - RSA PKCS#1 v2
    - az RSA-OAEP leírását tartalmazza, az RSA titkosító második standard változata, ezt kell használni!!!!!
    - *PKCS#1 v2*
- RSA digital signatures:
  - a PKCS#1 v1.5:
    - az RSA digitális aláírás első standard változata,
    - habár az algoritmus biztonságos, helytelen implementációk láttak napvilágot, amelyekben lehetőség van aláírást előállítani a privát kulcs ismerete nélkül is
  - PKCS#1 v2:
    - az RSA-PSS (probabilistic signatures scheme) leírása, biztonságos
    - hasonló padding technika kerül alkalmazásra, mint az RSA-OAEP-ben

Az RSA PKCS#1 v1.5

- 1998-ban jelent meg a *Bleichenbacher támadás*, ami megmutatta, hogy a titkosító feltörhető
- mégis nagyan gyakran használják: a fejlesztők nem látnak sok különbséget a v1.5 és a v2 között!!
- a *Gen* kulcs-generáló algoritmus ugyanaz, mint a textbook RSA-nál
- az üzenetet random bittekkel egészítik ki, úgy a titkosító, mint a digitális aláírás esetében:



Az RSA-OAEP rendszer

- 1995-ben Bellare és Rogaway publikálják
- egy véletlen G bit-generátort, és egy H hash függvényt használ:

$$G : \{0,1\}^{l_H} \rightarrow \{0,1\}^{l_G}$$
$$H : \{0,1\}^{l_G} \rightarrow \{0,1\}^{l_H}$$

- a H függvénynek min 160 bites hash függvényt kell használni,
- a G függvény szerkesztése:

$$G(r) = SHA^j(r \mid\mid < 0 >) \mid\mid SHA^j(r \mid\mid < 1 >) \mid\mid \ldots, \text{ahol}$$

  - r véletlen bitsorozat és az $< i >$ jelölés az i kettes számrendszerbeli értékét jelenti 4 bájton ábrázolva,
  - az SHA^j jelölés azt jelenti, hogy a hash első j, legnagyobb helyértékű bájtjait kell felhasználni
- a *Gen* kulcsgeneráló algoritmus ugyanaz, mint a textbook RSA-nál

Tipikus esetben, ha a modulus 1024 bites (128 bájt), akkor:

- $l_H = 160$ bit (20 bájt), $l_G = 864$ bit (108 bájt), $j = 80$ bit (10 bájt).

Az RSA-OAEP rendszer

Titkosítás, legyen m a nyílt szöveg, amelynek hosszát l_m-el jelöljük:

$$r \xleftarrow{R} \{0,1\}^k$$
$$x = m \;||\; 0^{l_G - l_m}$$
$$y = (x \oplus G(r)) \;||\; (r \oplus H(x \oplus G(r)))$$

- tipikus esetben $k = 160$ bit (20 bájt), és ha $l_m = 256$ bit (32 bájt), akkor x-et 608 nullás bittel (76 bájt) kell kiegészíteni,
- a titkosított érték: $y^e \pmod n \rightarrow c$

Visszafejtés, legyen c a titkosított érték:

- meghatározzuk az $y = c^d \pmod n$ értéket
- felosztjuk y-t $y_1 \;||\; y_2$-re úgy, hogy $|y_1| = l_G$ és $|y_2| = l_H$,
- meghatározzuk $r = H(y_1) \oplus y_2$,
- meghatározzuk $x = y_1 \oplus G(r)$,
- ellenőrizzük, hogy x utolsó $l_G - l_m$ bitje nulla-e:
  - ha nem, akkor REJECT kimeneti értékkel leállunk,
  - ellenkező esetben vissza térítjük x első l_m darab bitjét

Az RSA digitális aláírása, PKCS#1 v1.5

- a *Gen* kulcsgeneráló algoritmus ugyanaz, mint a textbook RSA-nál
- $Aut_{(d)}(m) \rightarrow (c, m)$ a hitelesítő/aláírást előállító algoritmus:
  - $c \leftarrow h^d \pmod{n}$, ahol $h \leftarrow H(m)$, ahol H minimum 160 bites hash függvény,
- $Ver_{(e)}(c, m)$ az ellenőrző algoritmus:
  - $h_1 \leftarrow c^e \pmod{n}$,
  - $h_2 \leftarrow hash(m) \pmod{n}$,
  - ha $h_1 = h_2$, akkor az aláírás hiteles, ellenkező esetben nem.

Az RSA-PSS digitális aláírás, PKCS#1 v2

- legelső formáját Bellare és Rogaway publikálták 1998-ban
- hasonlóan az RSA-OAEP-hez egy véletlen G bit-generátort, és egy H hash függvényt használ:

$$G : \{0,1\}^{l_H} \rightarrow \{0,1\}^{l_G}$$
$$H : \{0,1\}^{l_G} \rightarrow \{0,1\}^{l_H}$$

- a *Gen* kulcs-generáló algoritmus ugyanaz, mint a textbook RSA-nál
- egy véletlenszerűen generált *salt*-al dolgozik, ezért
  - az RSA-PSS aláírás véletlenszerű lesz: különböző aláírást kapunk ha ugyanazt az üzenetet kétszer, ugyanazzal a kulccsal írjuk alá
  - a támadó különböző aláírásokat vizsgálva/összehasonlítva nem fogja tudni megállapítani, hogy van-e olyan két aláírt üzenet amelyek egyformák.

Az RSA-PSS digitális aláírás

Aláírás előállítás: az $Aut_{(d,n)}$ aláírást előállító algoritmus bemenete az m üzenet:

$$
\begin{aligned}
&salt \xleftarrow{R} \{0,1\}^{l_{salt}} \\
&x = 0^{64} \;||\; H(m) \;||\; salt \\
&r = 0^{l_y - l_{salt} - l_H - 2} \;||\; salt \\
&y = (r \oplus G(H(x))) \;||\; H(x) \;||\; 0xbc \\
&c = y^d \pmod{n}
\end{aligned}
$$

Aláírás ellenőrzés: az $Ver_{(e,n)}$ aláírást ellenőrző algoritmus bemenete az (c, m) aláírás:

- meghatározza az $y = c^e \pmod{n}$ értéket
- ellenőrzi az y bithosszát, illetve hogy a vége egyenlő-e a $0xbc$ bájttal, ha igen akkor ezt a bájtot levágja
- felosztja a levágott bitszekvenciát $y_1 \;||\; y_2$-re úgy, hogy $|y_1| = l_G$ és $|y_2| = l_H$,
- meghatározza $r = y_1 \oplus G(y_2)$,
- ellenőrzi az r hosszát, tartalmát: a felső helyértékű bitek nullások kell legyenek
- meghatározza a $salt$ értékét: ez az r l_{salt} darab alsó helyértékű bitje lesz
- létrehozza $x = 0^{64} \;||\; H(m) \;||\; salt$
- ellenőrzi, hogy fennáll-e $y_2 = H(x)$

Az RSA gyorsítása

- a visszafejtés időigényének javítása érdekében alkalmazható a kínai maradéktétel: négyszer gyorsabb lesz az algoritmus
- alkalmazásával a rendszer nem veszít biztonságából,
- ahelyett hogy az n nagyságrendjével megegyező d hatványkitevővel számolnánk, elvégzünk két kisebb (a p nagyságrendjével megegyező) hatványkitevővel való hatványozást.
- meghatározzuk dp, dq, Mq, Mp értékeket a következő módon:

$$\begin{aligned} dp &= d \pmod{p-1} \\ dq &= d \pmod{q-1} \\ q \cdot Mq &= 1 \pmod{p} \\ p \cdot Mp &= 1 \pmod{q}. \end{aligned}$$

- a $c^d \pmod{n}$ értéket megadja az x értéke, ahol

$$\begin{aligned} x &= (Mq \cdot q \cdot xp + Mp \cdot p \cdot xq) \pmod{n} \\ xp &= c^{dp} \pmod{p} \\ xq &= c^{dq} \pmod{q}. \end{aligned}$$

RSA, biztonsági problémák

- az n faktorizálásához kapcsolódó problémák:
  - a k nagyságrendje: min 1024 bit
  - Fermat féle faktorizáció: a p és a q ne legyenek túl közel egymáshoz
  - Pollard ρ féle faktorizáció: a p, vagy a q kicsi,
  - Pollard $p-1$ féle faktorizáció: $p-1$, illetve $p+1$-nek legyen legalább egy "nagy" prímosztója,
  - ...
- az RSA kulcsok generálása során adódó hibák:
  - a p és a q értékek egyforma nagyságrendűek, egymástól függetlenek, véletlenszerűen generáltak kell legyenek, szorzatuk pedig legalább 2048 bites szám legyen
  - ugyanazt a p, vagy q értéket nem lehet egy másik modulushoz felhasználni
  - ugyanazt a modulust nem lehet különböző e értékhez felhasználni
  - e kicsi kell legyen, d azonban az n nagyságrendjével kell egyenlő legyen
  - minden egyes titkosításhoz/aláíráshoz más r random értéket kell generálni
- legjobb módszer: ha véletlenszerűen választjuk meg a prímeket!!

RSA, biztonsági problémák

Implementációhoz kapcsolódó problémák:

- timing attack, Koecher és tsai 1997: le lehet mérni, hogy mennyi időbe telik $c^d \pmod n$ meghatározása
- power attack, Koecher és tsai 1999: le lehet mérni, hogy mekkora energiafogyasztás szükséges $c^d \pmod n$ meghatározásához

```
def myPow(a, x, n):
    res = 1
    while x != 0:
        if x & 1 == 1:
            res = (res * a) % n
        a = (a * a) % n
        x = x >> 1
    return res
```

```
def myPow1(a, x, n):
   res = (a * a) % n
   binX = bin(x)[2:]
   for b in binX[1:] :
       if b == '0':
           res = (res * a) % n;
           a = (a * a) % n
       else:
           a = (res * a) % n;
           res = (res * res) % n
   return a
```

RSA, biztonsági problémák

Implementációhoz kapcsolódó problémák:

- faults attack, Boneh és tsai 1997: a $c^d \pmod n$ meghatározásakor fellépő számítási hibák vizsgálata:
  - ha a $xq = c^{dq} \pmod q$ meghatározásánál hiba adódik, míg az $xp = c^{dp} \pmod p$-nél nem, akkor legnagyobb közös osztót számolva lehetségessé válik az egyik prim meghatározása:

$$x \equiv (Mq \cdot q \cdot xp + Mp \cdot p \cdot xq) \pmod n$$
$$x^e \equiv c \pmod p, x^e \not\equiv c \pmod q.$$

  - tehát $x^e - c$ osztható p-vel, de nem osztható q-vel: $(x^e - c, N) = p$.

RSA, biztonsági problémák

faults attack, Boneh és tsai 1997, példa:

- legyen

$$p = 13, q = 17,$$
$$n = p \cdot q = 221, phi = (p-1) \cdot (q-1) = 192,$$
$$e = 5, d = 77, e \cdot d = 1 \pmod{phi}$$
$$m = 207, c = 207^5 \equiv 90 \pmod{221}$$

- visszafejtéshez a következő számítások szükségesek:

$$dp \equiv d \pmod{p-1} \equiv 5$$
$$dq \equiv d \pmod{q-1} \equiv 13$$
$$Mq = 10 \cdot 17 \cdot 10 \equiv 1 \pmod{13}$$
$$Mp = 4 \cdot 13 \cdot 4 \equiv 1 \pmod{17}$$
$$xp \equiv c^{dp} \pmod{p} = 90^5 \equiv 12 \pmod{13}$$
$$xq \equiv c^{dq} \pmod{q} = 90^{13} \equiv 3 \pmod{17}$$

- tegyük fel, hogy hiba adódott xq meghatározásakor, legyen a hibás érték 7, ezért visszafejtéskor a következő x értéket kapjuk:

$$x \equiv 10 \cdot 17 \cdot 12 + 4 \cdot 13 \cdot 7 \equiv 129$$

- legnagyobb közös osztót számolva meg lehet határozni a p-t:

$$129^5 \pmod{221} - 90 = 52$$
$$(52, 221) = 13$$

RSA - biztonsági problémák

A textbook RSA esetében az $e = 3$-as eset lehetővé teszi a rendszer feltörését, meghatározható az eredeti K érték, a privát kulcs ismerete nélkül.

- a kínai maradék tétel és egy köbgyököt meghatározó eljárást kell alkalmazni
- tegyük fel, hogy a K értéket háromszor titkosítottuk a $(3, n_1)$, $(3, n_2)$, $(3, n_3)$ publikus kulcsokkal, és rendre a cK_1, cK_2, cK_3 értékeket kaptuk
- a következő lineáris kongruencia rendszer adható meg:

$$\begin{aligned} x \equiv K^3 \equiv cK_1 \equiv \pmod{n_1} \\ x \equiv K^3 \equiv cK_2 \equiv \pmod{n_2} \\ x \equiv K^3 \equiv cK_3 \equiv \pmod{n_3}, \end{aligned}$$

- ezt a kínai maradéktételt alkalmazva könnyedén meg tudjuk oldani
- a kongruencia rendszer eredményeként kapott x értékből köbgyököt kell vonni:

```
>>> c = 677394484934938164599918310783958190345576648
>>> decimal.getcontext().prec = 100
>>> K = math.ceil(pow(c, 1/decimal.Decimal(3)))
```

RSA - biztonsági problémák

A textbook RSA esetében az egyforma modulusok problémája azt fogja jelenteni, hogy ha a K értéket kétszer rejtjelezzük egyszer az (e, n), másodszor az (f, n) publikus kulcsokkal, és ha fennáll, hogy $(e, f) = 1$, akkor meghatározható a K értéke a privát kulcs ismerete nélkül:

- Legyen cK_1, cK_2 a két rejtjelzett értéke a K-nak.
- A kiterjesztett euklideszi algoritmussal meghatározzuk azokat az x, y egész számokat, amelyekre fennáll: $e \cdot x + f \cdot y = 1$. Ez a két egész szám egyértelműen meghatározható, mert $(e, f) = 1$.
- A $cK_1^x \cdot cK_2^y \pmod{n}$ érték a keresett K számot fogja eredményezni, fennáll ugyanis:

$$cK_1^x \cdot cK_2^y = (K^e)^x \cdot (K^f)^y = K^{e \cdot x + f \cdot y} \equiv K \pmod{n}.$$

- A kiterjesztett euklideszi algoritmus negatív értéket is számolhat az x, vagy y-ban. Ha $x < 0$, akkor szükséges meghatározni a cK_1 inverzét $\pmod{n}$ szerint, és a $cK_1^x \pmod{n}$ hatványérték helyett az $inv^{-x} \pmod{n}$ hatványértékkel kell dolgozni, ahol inv-vel a cK_1 inverzét jelöltük. Hasonlóan kell eljárni, ha $y < 0$.

RSA - biztonsági problémák

A textbook RSA esetében az egyforma modulusok problémája, példa:

- legyen $p = 37, q = 127, e = 11, f = 17$. Ekkor $n = 4699$ és rejtjelezzük a $K = 446$ értéket. Kapjuk:

$$cK_1 = 446^{11} \pmod{4699} = 3158$$
$$cK_2 = 446^{17} \pmod{4699} = 1757$$

- a kiterjesztett euklideszi algoritmus meghatározza a $x = -3, y = 2$ értékeket
- mivel $x < 0$ szükséges meghatároni 3158 inverzét, ez 2906 lesz
- $K : 2906^3 \cdot 1757^2 \pmod{4699} = 446$.
- Python: `pow` függvény automatikusan az alap inverzével fogja a hatványértéket számolni:

```
>>> p, q, e, f = 37, 127, 11, 17
>>> n = p * q
>>> cK1, cK2 = 3158, 1757
>>> x, y = -3, 2
>>> (pow(cK1, x, n) * pow(cK2, y, n)) % n
    446
```

Le is ellenőrizhetjük:

```
>>> (pow(2906, -x, n) * pow(cK2, y, n)) % n
    446
```