Kriptográfia és Információbiztonság
5. előadás

MÁRTON Gyöngyvér

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

2023

Miről volt szó az elmúlt előadáson?

- blokk-titkosító rendszerek
- blokk-titkosító módok
- a Crypto++ könyvtárcsomag
- 3DES ECB, CBC módban, C++ kód
- a DES (Data Encryption Standard): specifikáció, biztonság, tervezési szempontok
- TEA, Blowfish

Miről lesz szó?

- az AES (Advanced Encryption Standard): specifikáció, biztonság, tervezési szempontok
- hash függvények (hash function)
- publikus kulcsú titkosítók: RSA, RSA-OAEP
- digitális aláírások: RSA, RSA-RSS

Modern kriptográfia - területek, felosztása

Titkos kulcsú kriptográfia: a tulajdonképpeni adatfolyam titkosítása

- folyamtitkosítók
- blokktitkosítók

Publikus kulcsú kriptográfia: az adatfolyam titkosításánál alkalmazott kulcs megosztása, a kulcsmegosztáshoz használt publikus kulcs/publikus információ hitelesítése

- publikus kulcsú titkosítók
- kulcscserék
- digitális aláírások

Egyéb kriptográfia primitívek:

- pszedudo-random és random számgenerátorok
- hash függvények
- üzenethitelesítő kódok (MAC - Message Authentication Code)

AES (Advanced Encryption Standard)

- Daemen és Rijmen, belga kriptográfusok tervezték 1997-ben
- az 1990-ben meghírdetett nyilvános pályázat győztese
- valószínűleg nincs benne "backdoor", mert a győztes pályázat elbírálása teljesen nyilvánosan történt
- 2001-ben fogadta el a NIST standard blokk titkosítónak
- kulcs-mérete: 128, 192, 256 bit, blokk-mérete: 128 bit,
- hatékonysága: 109 Mb/sec
- szakvélemények szerint a 256 bites kulcsméret biztonsága "örök" időkre szól
- nem használja a Feistel-sémát, viszont hasonlóan a DES-hez, iterációs szerkezetű: körkulcsokat generál:
  - 128 bites kulcs esetén a körök száma 10
  - 192 bites kulcs esetén a körök száma 12
  - 256 bites kulcs esetén a körök száma 14
- nem találtak sikeres támadást ellen, a mai napig az implementációkból adódó gyengeségek okozzák a biztonsági problémákat

AES (Advanced Encryption Standard)

- helyettesítést és permutációt alkalmaz
- véges testek felett alkalmaz $\oplus$ (aritmetikai és) műveletet
- motiváció: kriptográfiában egész számokkal dolgozunk, 0 és $2^n - 1$ közötti értékekkel, és az összes n-bit hosszúságú számra szükség van
- olyan halmazt kell választani, ahol az összeadás, kivonás, szorzás, osztás után is halmazbeli elemet kapunk $\rightarrow$ $GF(2^n)$ feletti véges testek:
  - az AES esetében bináris együtthatójú, és $n = 8$ fokszámnál kisebb polinomokkal dolgozunk:
    $$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0,$$
  - egy $GF(2^n)$ feletti polinom egyedi módon, az együtthatók segítségével leírható, n biten: $(a_{n-1}a_{n-2}\ldots a_0)$
  - a műveleteket a szabályos polinomok feletti műveleti szabályok szerint kell végezni, ahol az együtthatók esetében $\pmod 2$ kell számolni
  - ha egy művelet elvégzése után nagyobb kitevőt kapunk mint x^{n-1}, akkor meg kell határozni az eredmény $m(x)$ szerinti osztási maradékát, ahol $m(x)$ egy n-ed fokú irreducibilis polinom
  - n-ed fokú irreducibilis polinom: olyan polinom, amely nem bontható fel két n-nél kisebb fokú polinom szorzatára

AES (Advanced Encryption Standard)

- az AES minden műveletet 8 biten végez,
- az összeadás, kivonás, szorzás, osztás a $GF(2^8)$ feletti véges testben történik,
- összesen 30 irreducibilis polinom van, ahol az AES a következővel dolgozik:

$$x^8 + x^4 + x^3 + x + 1$$

- minden bájtot a $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$ test elemeként kezel, ahol a $b_7 b_6 \dots b_1 b_0$ bájt megfelel a

$$b_7 \cdot x^7 + \dots + b_1 \cdot x + b_0 \pmod{x^8 + x^4 + x^3 + x + 1}$$

  maradékosztálynak
- Példa

$x^6 + x^3 + x$ -nek megfelelő bináris alak: 0100 1010.

- három algoritmust kell definiálni: kulcsgenerálás, titkosítás, visszafejtés.
- a bemeneti $128 = 16 \cdot 8$ bites $S_1, S_2, \dots S_{16}$ szekvenciát (*state*-et) egy 4×4-es mátrix formában kezeli:

$$\begin{matrix} S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \\ S_4 & S_8 & s_{12} & S_{16} \end{matrix}$$

Hash függvények

- a H hash függvény a bemeneti tetszőleges hosszúságú M adatszekvenciára egy fix méretű $h = H(M)$ hash értéket állít elő
- a "jó" hash függvény egyenletes eloszlású és látszólag véletlenszerű kimenetet eredményez
- a biztonsági alkalmazásokhoz szükséges hash függvényt kriptográfiai hash függvénynek nevezzük
- a kriptográfiai hash függvény több biztonsági követelménynek is eleget kell tegyen
- egyike a legsokoldalúbb kriptográfiai primitívnek (algoritmusnak): számos biztonsági alkalmazásban, internetes protokollban használják
- elsődleges alkalmazási területük az adatintegritás, de használják:
  - jelszavak tárolásakor
  - titkosító rendszerekben
  - kulcscsere protokollokban
  - digitális aláírásokhoz
  - blokkláncok létrehozásakor

Hash függvények

Jelszavak tárolása esetében

- biztonsági okokból nem közvetlenül a jelszavaknak megfelelő bájtszekvenciát tárolják, hanem egy hashfüggvény segítségével egy másik bájt szekvenciát képeznek és ezt tárolják el
- a hashfüggvény alkalmazása előtt a jelszón egy "normalizálást" is elvégeznek: azaz egy kulcsképző függvényt (key derivation function) és egy "toldalék"-bájtszekvenciát (salt-ot) felhasználva, egy újabb bájtszekvenciát állítanak elő
- a hash függvény kimeneti értéke mellett, eltárolásra kerül a salt értéke is

Blokkláncok esetében

- a blokklánc összekapcsolt blokkok sorozata, ahol minden blokk tartalmazza az előző blokk hash értékét
- Pl. egy blokkláncban hatékonyan lehet pénzügyi tranzakciókat tárolni, mert a blokkban lévő tranzakciók nem módosíthatók, csak ha a tranzakciót megelőző összes hash-értéket megváltoztatjuk

Hash függvények

- MD2, MD4, MD5, MD6
  - az MD5-t Ron Rivest publikálta, 1991-ban, az MD4-nek egy továbbfejlesztett változata
  - 2004-ben komoly biztonsági problémák merültek fel az MD5-el kapcsolatosan
- SHA-1: a NIST publikálta először 1993-ban, változatai standardok
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512): hasonló szerkesztésű, mint az SHA-1, de a belső állapot mérete 256 bit
- SHA-3
  - 2007-ben a a NIST pályázatot írt ki, 2012-ben a *Keccak* nyert: biztonság, flexibilitás, egyszerűség a jellemzői
  - nem Merkle-Damgård szerkezetű
- RIPEMD-128-160-256-320
  - először Belgiumban publikálták, 1996-ban
  - szabad felhasználási lehetőség, nincs levédve
- WHIRLPOOL
  - 2000-ben publikálták,
  - egyik szerkesztője Rijmen,
  - az ISO által elfogadott nemzetközi standard.

Hash függvények, általános szerkesztés

A legismertebb, legelterjetteb szerkesztési mód a Merkle-Damgård szerkesztés:

- $h : X \to Y$, hash függvény: tetszőleges hosszúságú bitsorozatból fix hosszúságú bitsorozatot állít elő,
- ha az x bitsorozat hash értékét akarjuk megszerkeszteni:
  - felosztjuk x-et k-bit hosszúságú bit-sorozatokra: $x_1, x_2, \ldots x_l$,
  - szükség esetén kiegészítjük x-et további bitekkel
  - definiálunk egy F tömörítő függvényt, amelyet rekurzívan alkalmazunk
  - meghatározunk egy kezdeti IV értéket, amely betöltheti a kulcs szerepét is, általában azonban előredefiniált konstans értéket jelent
  - az általános szerkesztési lépéssorozat:

$$\begin{aligned} H_0 &= IV \\ H_i &= F(H_{i-1}, x_i), i = 1, 2, ..., l \\ h(x) &= H_l \end{aligned}$$

  - az F függvény és a IV határozzák meg a hash függvény biztonságát

Hash függvények, tulajdonságok

Az alábbi problémák ne legyenek megoldhatóak polinomiális időben:

- 1. probléma, egyirányú (preimage) tulajdonság:
  Bemenet: h, és $y \in Y$
  Kimenet: határozzuk meg x-t,úgy hogy: $h(x) = y$
- 2. probléma, gyengén ütközésmentes (second preimage) tulajdonság:
  Bemenet: h, és $x \in X$
  Kimenet: határozzuk meg x_1-t,úgy hogy: $x_1 \neq x$ és $h(x) = h(x_1)$
- 3. probléma, ütközésmentes (collision) tulajdonság:
  Bemenet: h
  Kimenet: határozzuk meg x, x_1-t,úgy hogy: $x_1 \neq x$ és $h(x) = h(x_1)$

további tulajdonságok:

- hatékonyan lehessen kiszámítani a hash értékét
- a hash kimenete legalább annyi bit legyen, hogy ne lehessen brute-force támadást alkalmazni rá
- lavina effektus: a bemenet egyetlen bitjének a megváltoztatása vonja maga után a kimenet teljes megváltozását

Hash függvények, születésnap paradoxon

- egy $h : X \to Y$ hash függvény esetében a X jóval nagyobb elemszámú, mint az Y, ez azt is jelenti, hogy ütközések mindig fennállnak
- a születésnapi paradoxon megmutatja, hogy az ütközések meglepően gyakran előállhatnak:
  - születésnap paradoxon: 23 véletlenszerűen kiválasztott ember esetében, legalább 50%-os az esélye annak, hogy legalább ketten lesznek olyanok, akiknek ugyanazon a napon van a születésnapjuk
  - két ugyanazzal a születésnappal rendelkező ember kiválasztása ugyanazt jelenti, mint ütközést találni egy hash függvény esetében
  - bebizonyítható, hogy ha M az elemszáma, a lehetséges hash értékeknek, akkor elég $1.17 \cdot \sqrt{M}$ elemet kiválasztani ahhoz, hogy 50% felett legyen annak a valószínűsége, hogy ütközést találjunk,
  - a születésnap paradoxon esetében $M = 365$, akkor tehát $1.17 \cdot \sqrt{M} = 1.17 \cdot \sqrt{365} = 1.17 \cdot 19.10 = 22.35$ elemet elég ha kiválasztanunk
  - a fentiek alapján pl. egy 40 bites hash nem nyújt biztonságot, mert $2^{20} \simeq 10^6$ hash érték meghatározása 50%-nál nagyobb valószínűséggel eredményez ütközést ($1.17 \cdot 2^{20} = 1226833.92$)
  - a jelenlegi ajánlások minimum 160 bites kimenetű hash függvények használatát ajánlják

A publikus-kulcsú rendszerek, alapfogalmak

- 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 bitműveletek, a helyettesítés és permutáció, hanem egész számokkal vagy számpárokkal végzett algebrai műveletek
- a publikus kulcsú titkosítók, a kulcscserék a szimmetrikus titkosítók kulcsainak a kommunikáló felek közötti megoszátását végzik
- a digitális aláírások a kulcscserénél használt publikus kulcsok/publikus információk hitelesítését végzik
- a rendszerekben megkülönböztetnek publikus kulcsot/információt és privát kulcsot/információt, ahol a privát kulcsok/információk szigorúan titkosak

A publikus kulcsú titkosítók matematikai modellje

Publikus kulcsú titkosítók, egyéb megnevezések: nyilvános kulcsú titkosítók, aszimmetrikus titkosítók (public-key encryption, asymmetric cryptography). Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- *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 bit-hossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$
- $Enc(pk, \cdot)$ a rejtjelező algoritmus, polinom idejű, véletlenszerű, 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

Digitális aláírások, egyéb megnevezések: publikus kulcsú tanúsítványok (digital signature, public key certificates). Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- 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 kulcscsere mechanizmusok matematikai modellje

Kulcscsere mechanizmusok (key exchange mechanism): ez egy interaktív protokoll, amikor a két kommunikáló fél egyidőben kell online legyen. Feltételezzük, hogy a két kommunikáló fél **A** és **B**. Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- *Gen*, a publikus információt/kulcsot generáló algoritmus, polinom idejű, véletlenszerű, meghatározza a pk - publikus információt:

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

  $pk \in K$, ahol k a rendszer biztonsági paramétere, a generált kulcs bithossza
- a $PGen_{\mathbf{X}}(pk, \cdot)$, $\mathbf{X} \in \{\mathbf{A}, \mathbf{B}\}$ algoritmus polinom idejű, véletlenszerű:
  - ezt mindkét fél végrehajtja, a kapott A, B értékeket megosztják egymással, az a, b értékek azonban szigorúan titkosak maradnak:

$$A \xleftarrow{R} PGen_{\mathbf{A}}(pk, a), \quad a \xleftarrow{R} M$$
$$B \xleftarrow{R} PGen_{\mathbf{B}}(pk, b), \quad b \xleftarrow{R} M$$

- a $KGen_{\mathbf{X}}(x, \cdot)$, $\mathbf{X} \in \{\mathbf{A}, \mathbf{B}\}$, $x \in \{a, b\}$ algoritmus polinom idejű, determinisztikus, és amelyet mindkét fél végre kell hajtson:

$$key \leftarrow KGen_{\mathbf{A}}(a, B), \quad key \leftarrow KGen_{\mathbf{B}}(b, A)$$

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

$$KGen_{\mathbf{A}}(a, B) = KGen_{\mathbf{A}}(a, PGen_{\mathbf{B}}(pk, b)) = KGen_{\mathbf{B}}(b, PGen_{\mathbf{A}}(pk, a)) = KGen_{\mathbf{B}}(b, A).$$

A publikus-kulcsú rendszerek, követelmények

- a publikus kulcsú titkosítók esetében publikus és privát kulcsokat különböztetünk meg
- a kulcscsere mechanizmusok esetében publikus és privát információt különböztetünk meg
- hatékony algoritmussal lehessen meghatározni a rendszerben használt kulcspárt/információt: a publikus és privát kulcsot, vagy a publikus és privát információt
- a publikus kulcs/információ ismeretében ne lehessen hatékonyan meghatározni a privát kulcsot/információt
- a publikus kulcsú kriptorendszerekben egész számokkal vagy számpárokkal végzünk aritmetikai műveleteket, az adatküldés miatt aztán ezeket a számokat bájtszekvenciákká alakítják, a fogadó fél oldalán pedig a kézhez kapott a bájtszekvenciákat visszaalakítják egész számokká
- 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

Publikus kulcsú titkosítók esetében:

- a publikus kulcs ismeretében hatékony algoritmussal lehessen meghatározni az üzenet rejtjelezett értékét
- a privát kulcs ismeretében hatékony algoritmussal lehessen visszafejteni a rejtjelezett üzenetet
- a publikus kulcs és rejtjelezett szöveg ismeretében ne lehessen hatékonyan meghatározni az üzenetet, ne lehessen következtetni annak tartalmára

Kulcscsere mechanizmusok esetében:

- a küldő publikus és a fogadó privát információjának ismeretében hatékony algoritmussal lehessen meghatározni a megosztásra kerülő kulcsot
- a küldő privát és a fogadó publikus információjának ismeretében hatékony algoritmussal lehessen meghatározni a megosztásra kerülő kulcsot
- a publikus információk alapján ne lehessen hatékonyan meghatározni a megosztásra kerülő kulcsot, ne lehessen következtetni annak tartalmára

A legismertebb publikus-kulcsú rendszerek

- 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
- Diffie-Hellman kulcscsere, biztonsága a diszkrét logaritmus probléma nehézségén alapszik
- 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 faktorizációs probléma

1. értelmezés

Az egész számok $\{1, 2, \ldots, n\}$ véges halmaza esetében, ahol n két prímszám szorzata a faktorizációs probléma azt jelenti, hogy megkeressük azt a két p és q prímszámot, amelyekre fennáll: $n = p \cdot q$.

- napjainkban ahhoz, hogy a faktorizációs algoritmusok ne vezessenek sikerre, hogy a rendszer megfelelő biztonságú legyen minimum 1024 bites prímszámok használatát írja elő a standard

A diszkrét logaritmus probléma

- a DL problémának több verziója is megadható: az egész számok $\pmod p$ multiplikatív csoportjára, illetve az elliptikus görbékre

Az egész számok $\pmod p$ multiplikatív csoportjára a következőképpen értelmezett a DL probléma:

2. értelmezés

Az egész számok $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ véges halmaza esetében, ahol p prímszám és $g \in \mathbb{Z}_p^*$ generátor elem a diszkrét logaritmus probléma azt jelenti, hogy megkeressük azt az $\boldsymbol{a} \in \mathbb{Z}_p^*$ pozitív egész számot, amelyre fennáll:

$$g^a \equiv A \pmod p.$$

- az a számot A g alapú diszkrét logaritmusának hívjuk
- nagy számok esetében a DL problémára nem ismert elfogadható futásidejű algoritmus
- jelenleg minimum 1024 bites prímszám használatát írja elős a standard

Az RSA rendszer

- 1977-ban publikálták a szerzők: Rivest, Shamir és Adleman, biztonsága többek között a faktorizációs problémán alapszik
- két távoli egység nyilvános csatornán történő kis méretű titkosított információ cseréjére alkalmas
- RSA titkosító: a gyakorlatban a titkosított információ a szimmetrikus rendszerekben használt titkos kulcs
- RSA digitális aláírás: a gyakorlatban a hitelesített információ a publikus kulcs
- a gyakorlatban az RSA-OAEP (RSA Optimal Asymmetric Encryption Padding) rendszert használják, amely az RSA titkosító egy módosított, biztonságos és standardizált változata
- az RSA-OAEP-t Bellare és Rogaway dolgozták ki, és 1995-ben publikálták
- a gyakorlatban az RSA-PSS (RSA Probabilistic Signature Scheme) rendszert is használják, amely az RSA digitális aláírás változta
- az RSA-PSS-t Bellare és Rogaway publikálták 1998-ban

Az RSA titkosító

- a rendszerben meg kell adni három algoritmust: a kulcsgeneráló, a titkosító, és a visszafejtő algoritmusokat
- a kulcsgeneráló algoritmus egy-egy értékpárt, pontosabban egy-egy kulcspárt határoz meg, ahol az egyik értékpárt publikus kulcsnak, a másik értékpárt privát kulcsnak hívjuk
- feltételezve, hogy a kommunikációban résztvevő két egység **A** és **B**, akkor a protokoll első lépéseként **A**, vagy egy harmadik megbízható egység végrehajtja a kulcsgeneráló algoritmust, majd a publikus kulcsot egy nyilvános csatornán megosztja a **B** egységgel
- ha egy harmadik egység végzi a kulcsgenerálást, akkor a privát kulcsot egy titkos csatornán megosztja **A**-val
- **B** véletlenszerűen generál egy $1 < K < n$ egész számot, és a titkosító algoritmussal, illetve a publikus kulccsal meghatároz egy cK értéket, a cK-t elküldi egy nyilvános csatornán **A**-nak
- **A** a visszafejtő algoritmussal, illetve a privát kulccsal meghatározza a **K** értéket
- a megosztott titkos információ tehát a K lesz
- a rendszer helyessége az Euler-tétellel bizonyítható

A baby-RSA rendszer

a kulcsgeneráló algoritmus:

- bemenete egy k biztonsági paraméter, amely a generált kulcsméretet jelenti,
- véletlenszerűen generál két k bites prímszámot, legyenek ezek p és q, és meghatározza az $n = p \cdot q$-t,
- kiválasztja azt a legkisebb $1 < e < n$ számot, amelyre fennáll $(e, \phi(n)) = 1$,
- meghatározza e moduláris inverzét $\pmod{\phi(n)}$ szerint, legyen ez d, fennáll tehát $e \cdot d = 1 \pmod{\phi(n)}$,
- a publikus kulcs: (e, n), a privát kulcs: (d, n)

a titkosító algoritmus:

- bemeneti paramétere a publikus kulcs, és a K nyílt-szöveg (K egy egész szám, ahol $1 < K < n$),
- meghatározza a $cK = K^e \pmod{n}$ egész számot, a titkosított szöveget,

a visszafejtő algoritmus:

- bemeneti paramétere a privát kulcs és a titkosított szöveg,
- meghatározza $K = cK^d \pmod{n}$ nyílt-szöveget,

A baby-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:
  - $K = 1526^{823} \equiv 2014 \pmod{5917}$.

Az RSA-OAEP rendszer

- standardként elfogadott módszer
- a gyakorlatban szinte soha nem használják titkosító algoritmusként, leggyakrabban kulcscsere protokollokban, valamilyen a szimmetrikus titkosító kulcsának a megosztására használják,
- 1995-ben Bellare és Rogaway publikálja, az RSA CCA-biztonságú változatát
- helyes paraméterezés esetében az RSA-OAEP a jelenlegi standardnak megfelelő biztonságot garantálja
- standardizált leírása megtalálható a PKCS#1 v2.0.
- egy véletlen bit-generátort (G) és egy hash függvényt (H) használ
- az RSA-függvény elveszíti multiplikatív tulajdonságát és probabilisztikus lesz
- az $e = 3$ publikus kulcs esetében is megfelelően biztonságos lesz
- egyszerűsített változata:
  - az SAEP rendszer, David Boneh publikálta 2001-ben,
  - az RSA-függvény helyett a Rabin-függvényt használjuk:

$$R_n : \{\mathbb{Z}_n^* \to \mathbb{Z}_n^*, x \to x^2 \pmod{n}\},$$

Az RSA-OAEP rendszer

- két függvény kerül alkalmazásra egy álvéletlen bit-generátor és egy hash függvény, jelöljük ezeket

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

- a standardban a H függvénynek az SHA újabb verzióját használják (min 160 bites),
- a G függvény szerkesztéséhez szintén az SHA újabb verzióját használják:

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

  - r véletlen bitsorozat és az $< i >$ jelölés az i kettes számrendszerbeli értékét jelenti 4 bájton ábrázolva,
  - az SHA^j jelölés azt jelenti, az SHA függvény első j, legnagyobb helyértékű bájtjait használjuk fel.

- a *Gen* kulcsgeneráló algoritmus ugyanaz, mint a baby RSA-nál

Az RSA-OAEP rendszer

Titkosítás, az $Enc_{(e,n)}$ a rejtjelező algoritmus bemenete az m nyílt szöveg, probabilisztikusan, polinomiális időben meghatározza a c rejtjelezett értéket:

- a nyílt-szöveg $P = (\mathbb{Z}_2)^{l_P}$, a titkos szöveg $C = (\mathbb{Z}_2)^{l_H + l_G}$ halmazon van értelmezve,
- legyen $r \in \{0,1\}^k$ véletlenszerű bitsorozat, (ha a modulus 1024 bit, akkor k = 16 bájt)
- $x = m \mid\mid 0^{l_G - l_P}$, ($m$-et kiegészítjük nullásokkal),
- meghatározzuk $y = (x \oplus G(r)) \mid\mid (r \oplus H(x \oplus G(r)))$,
- a titkosított érték: $y^e \pmod n \rightarrow c$

Megjegyzések:

- tipikus esetben, ha a modulus 1024 bites (128 bájt), akkor $l_H = 20$ bájt (160 bit), $l_G = 108$ bájt (864 bit), $j = 10$ bájt (80 bit), $k = 20$ bájt
- ha $l_P = 32$ bájt (256 bit), akkor x-et 76 bájtnyi nullással egészítjük ki

Az RSA-OAEP rendszer

Visszafejtés: az $Dec_{(d,n)}$ polinomiális idejű visszafejtő algoritmus bemenete a c rejtjelezett szöveg:

- 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 $l_G - l_P$ bitje nulla-e:
  - ha nem, akkor REJECT kimeneti értékkel leállunk,
  - ellenkező esetben vissza térítjük x első l_P darab bitjét

A RSA - megjegyzések

- ha $n = p \cdot q$, akkor az Euler függvény: $\phi(n) = (p-1) \cdot (q-1)$,
- a generált p, q prímszámokat és a $\phi(n)$ értékét titokban kell tartani; a titkosító és visszafejtő algoritmusok nem használják őket
- az e értéket választhatjuk véletlenszerűen, a standard a 65537 konstans értékkel dolgozik,
- a standard előírja, hogy a p és a q minimum 512 bites prímszámok legyenek, ekkor az n 1024 bites lesz,
- ha d is megközelítőleg 1024 bites, akkor p és a q ismerete nélkül, egyelőre nincs algoritmus, amely meghatározná a d-t,
- a d értéke a p és a q ismeretében, a kiterjesztett eukleidészi algoritmussal azonban meghatározható,
- ha a d értéke kicsi, akkor Wiener-algoritmusa megtudja határozni a d értékét, anélkül, hogy ismerné a p, q értékeket.

Az RSA digitális aláírás

3 algoritmussal értelmezhető, matematikai biztonsága a faktorizáció feltételezésen alapszik:

- Gen, a kulcs-generáló algoritmus: $(n = p \cdot q, e, d) \xleftarrow{R} Gen(1^k)$, ahol
  - p, q-prímszámok, $n = p \cdot q$,
  - $e \xleftarrow{R} \{1, \ldots, n-1\}$, úgy hogy $lnko(e, (p-1) \cdot (q-1)) = 1$,
  - d, az e inverze: $e \cdot d = 1 \pmod{(p-1) \cdot (q-1)}$ szerint,
- $Aut_{(d)}(m) \rightarrow (c, m)$ a hitelesítő/aláírást előállító algoritmus:
  - $c \leftarrow h^d \pmod{n}$, ahol $h \leftarrow hash(m)$,
- $Ver_{(e)}(c, m)$ 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 digitális aláírás, megjegyzések

- a kulcsgenerálás az RSA titkosító rendszer kulcsgeneráló algoritmusa szerint történik,
- az aláírást előállító algoritmus bemenete az m üzenet és a d, n titkos kulcs; kimenete a c hitelesített érték,
- az m üzenetnek meghatározzuk a hash értékét, a *hash* algoritmussal és utána alkalmazzuk a moduláris hatványozást a d titkos kulccsal,
- az ellenőrző algoritmus bemenete a c aláírt érték, az m üzenet, az e, n nyilvános kulcs; kimenete 1 ha hiteles az aláírás és 0, ha nem,
- a gyakorlatban a véletlenszerű (random) RSA digitális aláírási rendszert, az RSA-PSS rendszert használják

Az RSA-PSS digitális aláírás

- RSA-PSS: RSA-probabilistic signature scheme
- legelső formáját Bellare és Rogaway publikálták 1998-ban
- 2009-ben az RSA Laboratories több digitális aláírást tett közzé, ezek a paddingolási technikában különböznek
- az RSA Laboratories szerint az RSA-PSS a legbiztonságosabb
- egy H hash függvényt és egy G álvéletlen bit-generátort használ, amely hasonló, mint amit az RSA-OAEP-nél is alkalmaztak:

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

- a *Gen* kulcs-generáló algoritmus ugyanaz, mint a textbook RSA-OAEP-nél

Az RSA-PSS digitális aláírás

Aláírás előállítás: az $Aut_{(d,n)}$ aláírást előállító algoritmus bemenete az m üzenet, probabilisztikusan, polinomiális időben meghatározza a (c, m) értéket:

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

Ha a *salt* megfelelő hosszúságú és véletlenszerűen kerül kiválasztásra, akkor

- 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 ellenőrzés: az $Ver_{(e,n)}$ aláírást ellenőrző algoritmus bemenete az (c, m) determinisztikusan, polinomiális időben ellenőrzi az aláírást:

- meghatározza az $y = c^e \pmod{n}$ értéket
- ellenőrzi az y bithosszát, illetve hogy a vége egyenlő-e a 0xbc bájttal, ezt a bájtot levágja
- felosztja a levágott bitszekvenciát $y_1 \mid\mid y_2$-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 az r alapján, ez az alsó helyértékű l_{salt} darab bit kell legyen
- létrehozza $x = 0^{64} \mid\mid H(m) \mid\mid salt$
- ellenőrzi, hogy fennáll-e $y_2 = H(x)$