Kriptográfia és Információbiztonság
9. előadás

MÁRTON Gyöngyvér

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

2025

Miről volt szó?

- publikus kulcsú rendszerek: alapfogalmak
- publikus kulcsú rendszerek: matematika modellek,
- az RSA publikus kulcsú rendszer: RSA-textbook, RSA-OAEP, RSA-PSS

Miről lesz szó?

- a Diffie-Hellman kulcscsere
- digitális aláírások:
  - a Schnorr digitális aláírás
  - az ElGamal digitális aláírás
  - a DSA (Digital Signature Algorithm) vagy DSS (Digital Signature Standard)

Diffie-Hellman kulcscsere

- 1976-ban publikálták a szerzők, két távoli egység (számítógép, mobileszköz, stb.) kulcscsere mechanizmusára ad megoldást,
- a rendszer biztonsága a választott publikus paraméterek nagyságrendjétől függ, illetve a diszkrét logaritmus feltételezésen alapszik,
- a publikus paraméterek: egy k bites $p = 2 \cdot q + 1$ alakú biztonságos prímszám (safe prime), a q prímszám és p egy g generátor eleme, ahol k legalább 2048 bit kell legyen,
- áttérés figyelhető az elliptikus görbéken alapuló kriptográfiára

A diszkrét logaritmus feltételezés

A diszkrét logaritmus feltételezés, azaz feltételezzük, hogy a diszkrét logaritmus probléma (DL probléma) nehéz:

1. értelmezés

Az egész számok $\mathbb{Z}_p^*$ multiplikatív csoportja esetében, ahol p prímszám a DL probléma a következő: az A, g-alapú diszkrét logaritmusa $\pmod p$ szerint azt jelenti, hogy megkeressük azt az a pozitív egész számot, melyre fennáll:

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

ahol g generátor elem (primitív gyök), és $g, A \in \mathbb{Z}_p^*$.

- a g szám generátor elem $\pmod p$ szerint, ha g hatványai 1-től, $\phi(p)$-ig, azaz g, g^2, g^3, $\ldots$, $g^{\phi(p)}$, különböző maradékot adnak $\pmod p$ szerint
- a diszkrét logaritmus feltételezés azt jelenti, hogy megfelelő paraméterválasztás esetén (pl. a p prímszám legalább 2048 bites) nincs hatékony, polinomiális futási idejű algoritmus a DL problémára

Hatványok és generátor elemek

Határozzuk meg $x^n \pmod{11}$, értékeit $x = 2, 3, \ldots 10$-re illetve $n = 1, 2, \ldots 10$-re:

$2^1 = 2$	$3^1 = 3$	$4^1 = 4$	$5^1 = 5$	$6^1 = 6$	$7^1 = 7$	$8^1 = 8$	$9^1 = 9$	$10^1 = 10$
$2^2 = 4$	$3^2 = 9$	$4^2 = 5$	$5^2 = 3$	$6^2 = 3$	$7^2 = 5$	$8^2 = 9$	$9^2 = 4$	$10^2 = 1$
$2^3 = 8$	$3^3 = 5$	$4^3 = 9$	$5^3 = 4$	$6^3 = 7$	$7^3 = 2$	$8^3 = 6$	$9^3 = 3$	$10^3 = 10$
$2^4 = 5$	$3^4 = 4$	$4^4 = 3$	$5^4 = 9$	$6^4 = 9$	$7^4 = 3$	$8^4 = 4$	$9^4 = 5$	$10^4 = 1$
$2^5 = 10$	$3^5 = 1$	$4^5 = 1$	$5^5 = 1$	$6^5 = 10$	$7^5 = 10$	$8^5 = 10$	$9^5 = 1$	$10^5 = 10$
$2^6 = 9$	$3^6 = 3$	$4^6 = 4$	$5^6 = 5$	$6^6 = 5$	$7^6 = 4$	$8^6 = 3$	$9^6 = 9$	$10^6 = 1$
$2^7 = 7$	$3^7 = 9$	$4^7 = 5$	$5^7 = 3$	$6^7 = 8$	$7^7 = 6$	$8^7 = 2$	$9^7 = 4$	$10^7 = 10$
$2^8 = 3$	$3^8 = 5$	$4^8 = 9$	$5^8 = 4$	$6^8 = 4$	$7^8 = 9$	$8^8 = 5$	$9^8 = 3$	$10^8 = 1$
$2^9 = 6$	$3^9 = 4$	$4^9 = 3$	$5^9 = 9$	$6^9 = 2$	$7^9 = 8$	$8^9 = 7$	$9^9 = 5$	$10^9 = 10$
$2^{10} = 1$	$3^{10} = 1$	$4^{10} = 1$	$5^{10} = 1$	$6^{10} = 1$	$7^{10} = 1$	$8^{10} = 1$	$9^{10} = 1$	$10^{10} = 1$

$\pmod{11}$ szerint

- 2, 6, 7, 8 generátor elemek
- 1, 3, 4, 5, 9, 10 nem generátor elemek
- $p = 11$ prímszám esetében a generátor elemek száma: $\phi(p-1) = \phi(10) = 4$

Hatványok és generátor elemek

- kis Fermat tétel, kimondja, hogy ha p prím és $(x, p) = 1$, akkor $x^{p-1} = 1 \pmod{p}$,
- a táblázat alapján kijelenthetjük, hogy lesznek olyan x számok, ahol x-nek kisebb hatványértékei is kongruensek lesznek 1-el, ezek nem generátor elemek
- ha p egy prímszám, akkor mindig létezik $g \in \{1, 2, \ldots, p-1\}$, amelynek hatványértékei $\pmod{p}$ szerint előállítják az $\{1, 2, \ldots, p-1\}$ halmaz elemeit egy tetszőleges sorrendbe, ezeket a g elemeket generátor elemeknek hívjuk,
- fontos feladat, hogy egy adott prímszám esetében meghatározunk egy generátor elemet, erre általános esetben nincs hatékony algoritmus,
- biztonságos prímeknek (safe prime) hívjuk azokat a p prímszámokat, amelyek felírhatóak a $2 \cdot q + 1$ alakba, ahol q is prímszám
- ha a $p = 2 \cdot q + 1$ biztonságos prím, akkor egy generátor elem meghatározása nem számít nehéz feladatnak,
- egy biztonságos prím meghatározása azonban jóval időigényesebb, mint egy "közönséges" prím kiválasztása

Diffie-Hellman kulcscsere

Feltételezve, hogy a kommunikációban résztvevő két eszköz **A** és **B**, akkor a protokoll a következő:

1. *Gen*, a publikus paramétereket generáló algoritmus: meghatározzák a p, k-bites prímszámot, és a g generátor elemét,
2. *Send* algoritmus, az **A** eszköz számításai:
   - $a \xleftarrow{R} \{2, \ldots, p-2\}$ ez lesz az **A** privát kulcsa,
   - $A = g^a \pmod p$, ez lesz az **A** publikus kulcsa, amelyet elküld **B**-nek.
3. *Send* algoritmus, a **B** eszköz számításai:
   - $b \xleftarrow{R} \{2, \ldots, p-2\}$ véletlen szám, ez lesz a **B** privát kulcsa,
   - $B = g^b \pmod p$, ez lesz az **B** publikus kulcsa, amelyet elküld **A**-nak.
4. *Calc* algoritmus, **A** a közös K kulcsot a kövekezőképpen határozza meg, :
$$K = B^a \pmod p.$$
5. *Calc* algoritmus, **B** a közös K kulcsot a kövekezőképpen határozza meg:
$$K = A^b \pmod p.$$

- Helyesség:
$$K = A^b = B^a = g^{ab} \pmod p.$$

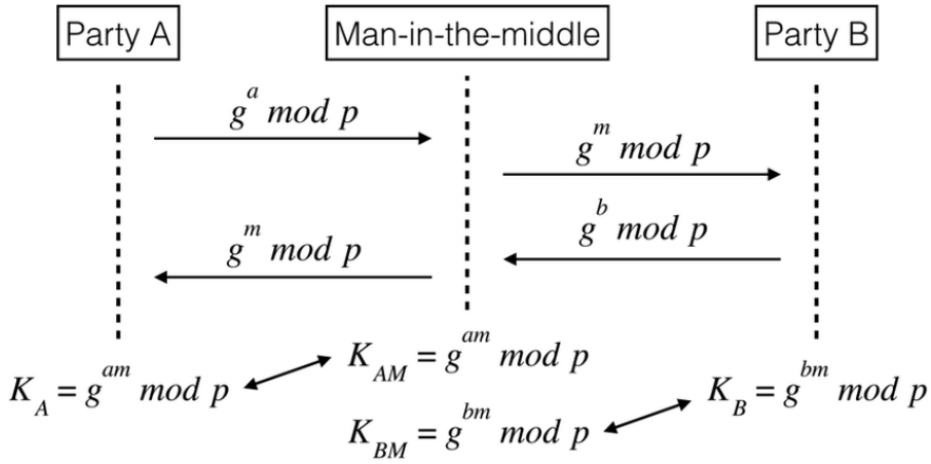
Diffie-Hellman kulcscsere, példa

- Legyen $p = 47, g = 13$ publikus paraméterek,
  - az **A** eszköz:
    - választja $a = 12$-t $\rightarrow A = g^a \pmod p = 9 \pmod{47}$,
    - elküldi Bobnak az A = 9 értéket.
  - a **B** eszköz:
    - választja $b = 34$-t $\rightarrow B = g^b \pmod p = 21 \pmod{47}$,
    - elküldi Alicenak a B = 21 értéket.
  - **A**: $K = B^a \pmod p = 21^{12} = 16 \pmod{47}$,
  - **B**: $K = A^b \pmod p = 9^{34} = 16 \pmod{47}$.
- a közös kulcs: $K = 16$.

A Diffie-Hellman kulcscsere biztonsága

- ha a kulcscsere protokollban a résztvevő felek nem tudnak meggyőződni egymás kiléte felöl, azaz a publikucs kulcsok nincsenek hitelesítve, akkor a rendszer nem lesz biztonságos, kivitelezhető a man in the meedle támadás:
  - egy **M** támadó kiadja magát **A**-nak, mint **B**, illetve kiadja magát **B**-nek, mint **A**,
  - **M** választ két random számot: m_1, m_2,
  - **A** irányába:
    - elküldi **A**-nak $M_2 = g^{m_2}$-t,
    - a közös kulcs **A**-val: $K_A = A^{m_2} = M_2^a = g^{a \cdot m_2}$,
  - **B** irányába:
    - elküldi **B**-nek $M_1 = g^{m_1}$-t,
    - a közös kulcs **B**-vel: $K_B = B^{m_1} = M_1^b = g^{m_1 \cdot b}$,
- minden kulcscsere protokoll támadható Man-in-the-Middle módszerrel, ha a protokoll résztvevői nem hitelesítik egymás publikus kulcsait, azaz nem győződnek meg egymás kiléte felől

Man-in-the-Middle támadás



- a támadó, a támadást azzal indítja, hogy meghatározza a mindkét fél irányába szükséges privát és publikus értékeket (kulcsokat)
- a publikus kulcsok hitelesítése:
  - digitális aláírással történik,
  - a protokolltól függ, hogy ezt hogyan valósítják meg a kommunikáló felek,
  - a TLS/SSL-ben egy központi hatóság (central authority) igénybevételével oldják meg, akinek tanúsítványok (certificate) kibocsájtására van joga

A Diffie-Hellman kulcscsere

A `SzerverDH.py` file tartalma:

```
import socket
from Crypto.Util.Padding import unpad
from Crypto.Cipher import AES
from Crypto.Random.random import randint

BS = AES.block_size
BF_SIZE = 2048; host = 'localhost'; port = 12365
def keySzerver(c, p, gen, plaintextFile):
    mess = c.recv(BF_SIZE)
    print ('a klienstől kapott üzenet: ', mess.decode())
    a = randint(2, p - 2); A = pow(gen, a, p)
    aByte = A.to_bytes((A.bit_length() + 7) // 8)
    c.send(aByte)
    bByte = c.recv(BF_SIZE)
    B = int.from_bytes(bByte);
    K = pow(B, a, p); print ('K:', K)
    key = K.to_bytes(2048 // 8)[:32]
    c.send('meghatároztam a közös kulcsot!'.encode())
    data = c.recv(BF_SIZE)
    print ('a klienstől kapott üzenet: ', data.decode())
    ciphertext = b''
    while True:
         data = c.recv(BF_SIZE)
         if not data: break
         ciphertext += data
    fileDecryp(ciphertext, decryptedFile, key)
    l = len(ciphertext); print(l)
    return c
```

A Diffie-Hellman kulcscsere

- A publikus paramétereket tartalmazó file tartalma: `DHKey.pem`
- A `SzerverDH.py` file tartalma:

```
def mainS(p, gen, plaintextFile):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print('várakozás (indítsd el a klienst)...')
    s.bind((host, port))
    s.listen(5)
    c, addr = s.accept()
    print('a kapcsolat a következő címen jött létre: ', addr)
    c = keySzerver(c, p, gen, plaintextFile)
    c.close()

def keyRead():
    with open('DHKey.pem', 'rt') as f:
        p = int.from_bytes(bytes.fromhex(f.readline()))
        q = int.from_bytes(bytes.fromhex(f.readline()))
        g = int.from_bytes(bytes.fromhex(f.readline()))
        return p, q, g
```

A Diffie-Hellman kulcscsere

A `SzerverDH.py` file tartalma:

```
def fileDecryp(ciphertext, decryptedFile, key):
    iv = ciphertext[:16]
    ciphertext = ciphertext[16: len(ciphertext)]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decryptedtext = cipher.decrypt(ciphertext)
    decryptedtext = unpad(decryptedtext, BS)
    with open(decryptedFile, 'wb') as outF:
        outF.write(decryptedtext)

p, q, gen = keyRead()
decryptedFile = ...
mainS(p, gen, decryptedFile)
```

A Diffie-Hellman kulcscsere

A `KliensDH.py` file tartalma:

```
import socket
from Crypto.Util.Padding import pad
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Random.random import randint

BS = AES.block_size
BF_SIZE = 2048; host = 'localhost'; port = 12365
def keyKliens(s, p, gen, decryptedFile):
    s.send(b'hello!')
    aByte = s.recv(BF_SIZE)
    A = int.from_bytes(aByte)
    b = randint(2, p - 2); B = pow(gen, b, p)
    bByte = B.to_bytes((B.bit_length() + 7) // 8)
    s.send(bByte)
    K = pow(A, b, p); print ('K: ', K)
    mess = s.recv(BF_SIZE)
    print ('a szervertől kapott üzenet: ', mess.decode())
    s.send('meghatároztam a közöstitkot'.encode())
    key = K.to_bytes(2048 // 8)[:32]
    iv = get_random_bytes(16)
    ciphertext = fileCrypt(plaintextFile, key, iv)
    s.send(ciphertext)
    l = len(ciphertext); print(l)
    return s
```

A Diffie-Hellman kulcscsere

A `KliensDH.py` file tartalma:

```
def mainK(p, gen, decryptedFile):
    s = None
    while True:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((host, port))
            print('letrejött a kapcsolat!')
            s = keyKliens(s, p, gen, decryptedFile)
            s.close()
            return
        except:
            s = None
            print('előbb indítsd el a szervert!')
            continue

def keyRead():
    with open('DHKey.pem', 'rt') as f:
        p = int.from_bytes(bytes.fromhex(f.readline()))
        q = int.from_bytes(bytes.fromhex(f.readline()))
        g = int.from_bytes(bytes.fromhex(f.readline()))
        return p, q, g
```

A Diffie-Hellman kulcscsere

A `KliensDH.py` file tartalma:

```
def fileCrypt(inputFile, key, iv):
    with open(inputFile, 'rb') as inF:
        text = inF.read()
        plaintext = pad(text, BS)
        cipher = AES.new(key, AES.MODE_CBC, iv)
        ciphertext = cipher.encrypt(plaintext)
        return iv + ciphertext


p, q, gen = keyRead()
plaintextFile = ...
mainK(p, gen, plaintextFile)
```

A digitális aláírások matematikai modellje

Jelölése legyen DS, ahol a (K, M, C) halmaz-hármas felett 3 algoritmust értelmezünk:

- Gen, a kulcs-generáló algoritmus, polinom idejű, véletlenszerű: $(pk, sk) \xleftarrow{R} Gen(1^k)$, ahol $(pk, sk) \in K \times K$, $k \in \mathbb{Z}_{\geq 0}$ a rendszer biztonsági paramétere,
  - publikus kulcs: pk,
  - privát kulcs: sk,
- Aut a hitelesítő algoritmus, polinom idejű, véletlenszerű, amelyet az egyik egység végez: $(m, c) \xleftarrow{R} Aut(sk, m)$, ahol $m \in P$,
- Ver az ellenőrző algoritmus, polinom idejű, determinisztikus, amelyet a másik egység végez: ha $Ver(pk, c) = m$, akkor az aláírás hiteles, ellenkező esetben nem.

A digitális aláírások, megjegyzések

- üzenethitelesítő kód (MAC):
  - biztosítja az átküldött üzenet integritását, hitelességét
  - a MAC nem biztosít eredet szolgáltatást, azaz a küldő bármikor letagadhatja, hogy üzenetet küldött a vevőnek
- a digitális aláírások:
  - biztosítja az üzenetek integritását, hitelességét,
  - biztosítja az üzenetek letagadhatatlanságát: a küldő utólag nem tagadhatja le az általa előállított áláírást,
  - az aláírást csak a küldő állíthatja elő,
  - az aláírást a rendszer bármelyik résztvevője ellenőrizheti,
  - nem titkosítja az üzenetet
  - nem az üzenetre, hanem annak egy hash értékére határozom meg: pl. SHA2, SHA3
  - más kulcsot kell használni a titkosításhoz/visszafejtéshez, más kulcspárt kell használni a kulcscseréhez, mást a digitális aláírásokhoz

A Schnorr digitális aláírás

- biztonsága a diszkrét logaritmus feltételezésen alapszik
- 1990-ben Claus Schnorr szerzője levédte, amely 2010-ben lejárt
- nagyon egyszerű, gyors, az aláírás mérete kicsi, de a gyakorlatban ritkán használták, a levédés miatt
- az EdDSA ezen az aláírási sémán alapszik
- egy nem interaktív zero-knowledge típusú azonosító protokollon (`rfc8235`) alapszik: a titok feltárása nélkül bizonyítja egy bizonyító hogy rendelkezik a titokkal

A Schnorr digitális aláírás

Feltételezzük, hogy a kommunikációban résztvevő két egység **A** és **B**:

- *Gen*, a kulcs-generáló algoritmus: $(p, g, q, a, A) \leftarrow Gen(1^k)$, ahol
  - p-prímszám, q, $p-1$-nek egy nagy prímosztója, $g : g^q = 1 mod p$,
  - p, q, g és H biztonságos hash függvény publikus paraméterek,
  - $a \xleftarrow{R} \{1, \ldots q-1\}, A = g^a \pmod p$,
  - A: publikus kulcs, a: privát kulcs
- $Aut_{(a)}(m) \rightarrow ((B, c), m)$ a hitelesítő algoritmus:
  - $b \xleftarrow{R} \{1, \ldots, q-1\}$, $B = g^b \pmod p$,
  - $h = H(B||m)$
  - $c = (b - h \cdot a) \pmod q$,
- $Ver_{(p,g,A)}((B, c), m)$ az ellenőrző algoritmus:
  - $h = H(B||m)$,
  - $v = A^h \cdot g^c \pmod p$,
  - ha $v == B$, akkor az aláírás hiteles, ellenkező esetben nem.

Helyesség:

$$v = A^h \cdot g^c = (g^a)^h \cdot g^c = g^{a \cdot h + c} = g^b \pmod p,$$
$$c = (b - h \cdot a) \pmod q \Rightarrow b = a \cdot h + c \pmod q.$$

Az ElGamal digitális aláírás

- biztonsága a diszkrét logaritmus feltételezésen alapszik
- 1985-ben publikálta Taher ElGamal
- a Diffie-Hellman kulcscsere módosított változta
- a gyakorlatban ritkán használják, több variánsa is ismert: a DSA standardként van elfogadva
- az aláírás randomizált

A DSA (Digital Signatures Algorithm)

- biztonsága a diszkrét logaritmus feltételezésen alapszik
- a Schnorr, illetve ElGamal alírási sémák egy változata
- a NIST 1991-ben standardként javasolta
- megjelenése óta számos kritika érte: nem volt publikus a NIST kiírása, az elején korlátozták a p értékét 512-re
- előszámításokkal, az aláírás-generálás algoritmus időigénye nagyon felgyorsítható: a b inverze és a $g^b \pmod p$ számítható ki előre, ebben az esetben meg kell oldani ezek biztonságos tárolását.

A Rabin digitális aláírás

- 1979-ben publikálta Michael O. Rabin, az első probabilisztikus digitális aláírási séma,
- a titkosító változat az oktatásban jelenik meg, de csak később, ezzel Rabin nem is foglalkozott,
- a titkosító változat esetében a visszafejtés nem egyértelmű,
- 2001-ben jelenik meg az SAEP,
- a gyakorlatban se a digitális aláírást, se a titkosító változatot nem használják,
- az alkalmazott függvény a moduláris négyzetreemelés,
- bizonyítható, hogy biztonsága ekvivalens a faktorizációs feltételezéssel: a rendszer feltörhetősége (aláírás hamisítás) ugyanolyan nehéz, mint megoldani a faktorizációs problémát,
- egy ütközésmentes (collision resistant) hash függvény is alkalmazásra kerül.