Report Project 2

Carl-Anton Grandelius, cagra@kth.se & Suhaib Abdi Muhummed, muhummed@kth.se

Course code: IX1500_HT2024
Date: 2024-10-02

1 Sammanfattning med Resultat

I detta projekt har vi dekrypterat det meddelande som Alice skickade till Bob enligt RSA med hjälp av enbart en offentlig nyckel. RSA-krypteringsalgoritmen bygger på den matematiska svårigheten att faktorisera stora tal i deras primtalskomponenter. I denna lösning kan man dock inte använda "trial division" för att faktorisera modulen $nBob$, vilket gör det möjligt att härleda den hemliga nyckeln och använda den för dekryptering. Detta eftersom p och q i vårt specifika fall är för stora primtal för datorn att beräkna på en rimlig tid. Därför har vi använt oss av sympy-bibliotekets mycket effektiva faktoriseringsfunktion $factorint$.
Siffrorna i listan (chiffertexten) var av samma storlek eftersom meddelandet har delats upp i block av samma storlek inom RSA-kryptering. Detta gör att varje block ligger inom $0 < m < n$.

1.1 Resultat

Resultatet efter dekryptering av den krypterade texten blev "RSA Cryptography is the world's most widely used public-key cryptography method for securing communication on the Internet. Instrumental to the growth of e-commerce, RSA is used in almost all Internet-based transactions to safeguard sensitive data such as credit card numbers. Introduced in 1977 by MIT colleagues Ron Rivest, Adi Shamir, and Leonard Adleman, RSA -its name derived from the initials of their surnames- is a specific type of public-key cryptography, or PKC, innovated in 1976 by Whitfield Diffie, Martin Hellman, and Ralph Merkle. Intrigued by their research, Rivest, with Shamir and Adleman, developed cryptographic algorithms and techniques to practically enable secure encoding and decoding of messages between communicating parties".

1.2 Sammanfattning av kodimplementation

Den huvudsakliga dekrypteringsprocessen utförs i funktionen `dekryptera_rsa`. För att dekryptera meddelandet måste vi känna till primtalsfaktorerna p och q för n, som används för att beräkna den hemliga nyckeln.

- Först faktoreras modulen $nBob$ för att få fram primtalen p och q. Om modulen inte är produkten av exakt två primtal, genererar funktionen ett fel. När de två primtalen har hittats returneras dessa.

$$p, q = \texttt{faktorisera\_modul}(nBob)$$

- Därefter beräknas $\varphi(n)$, som är Eulers totientfunktion, med formeln $(p-1) \times (q-1)$. Den hemliga nyckeln d beräknas sedan genom att ta den modulära inversen av den offentliga exponenten e modulo $\varphi(n)$. Funktionen `modul_inv` använder den utökade Euklides algoritmen för att beräkna denna invers:

$$\texttt{hemlig\_nyckel} = \texttt{modul\_inv}(eBob, \varphi(n))$$

- Den utökade Euklides algoritmen beräknar den största gemensamma delaren (gcd) mellan två heltal och hittar koefficienterna x och y som uppfyller ekvationen:

$$a \times x + b \times y = \gcd(a, b)$$

  Denna algoritm är viktig eftersom den används för att beräkna den modulära inversen.

- Chiffertexten består av flera block, och dessa bearbetas i omvänd ordning. Detta görs eftersom blocken kan ha sparats baklänges under krypteringen. Varje block dekrypteras genom att upphöja blocket med den hemliga nyckeln modulo $nBob$, vilket utförs med funktionen `pow()`:

$$\texttt{meddelande} = \texttt{pow}(chiffer, hemlig\_nyckel, nBob)$$

  Här sker den faktiska RSA-dekrypteringen, där $chiffer$ är varje block av chiffertexten och $meddelande$ är det dekrypterade resultatet.

- När ett heltal har dekrypterats måste det omvandlas till en byte-sekvens för att kunna avkodas som text. Funktionen `heltal_till_bytes` beräknar hur många bytes som behövs för att representera det dekrypterade heltalet och returnerar motsvarande byte-sekvens. Denna sekvens kan sedan omvandlas till en sträng för att få det ursprungliga meddelandet:

$$\texttt{dekrypterad\_text} = \texttt{''.join}(\texttt{heltal\_till\_bytes}(m).decode('utf-8', errors='ignore')$$
$$\text{for } m \text{ in } \texttt{dekrypterade\_meddelanden})$$

- Efter att alla chifferblock har dekrypterats och konverterats till text omvänds hela den dekrypterade texten med hjälp av `[::-1]`. Detta steg är avgörande eftersom texten kan ha kodats baklänges under krypteringen, och omvändningen ger tillbaka texten i rätt ordning:

$$\texttt{dekrypterad\_text} = \texttt{dekrypterad\_text[::-1]}$$

2 Formler och ekvationer

RSA-algoritmen möjliggör både kryptering och dekryptering. Även om uppgiften specifikt anger att vi ska dekryptera en viss sträng, blir det enklare att förstå dekrypteringsprocessen om vi först tittar på hur krypteringen och skapandet av nycklarna fungerar. I uppgiften har vi enbart den offentliga nyckeln (n, e), ej den hemliga nyckeln d för att dekryptera chiffertexten.

Av den anledningen har detta avsnitt delats upp i fyra underrubriker: Nyckelgenerering, kryptering, dekryptering och dekryptering med offentlig nyckel.

2.1 Nyckelgenerering

1. Välj (slumpmässigt) två stora primtal p och q.
2. Beräkna modulus n:
$$n = p \times q$$
3. Beräkna Eulers totientfunktion $\varphi(n)$. Ifall p och q är två stora primtal är det mycket svårt att få fram dem utifrån enbart n.
$$\varphi(n) = (p-1)(q-1)$$
4. Välj en offentlig heltalsexponent e inom intervallet $1 < e < \varphi(n)$ och att $\gcd(e, \varphi(n)) = 1$, vilket matematiskt kallas att de är "relativt prima".
5. Beräkna den hemliga exponenten d: - Beräkna d som den modulära inversen av e modulo $\varphi(n)$:
$$d \equiv e^{-1} \mod \varphi(n)$$
$$e \times d \equiv 1 \mod \varphi(n)$$
6. Generera nyckelpar: Offentlig nyckel: (n, e). Hemlig nyckel: (n, d).

2.2 Kryptering

1. Konvertera klartext till heltal m där $0 < m < n$.
2. Kryptera meddelandet med offentlig nyckel (n, e) för att beräkna chiffertext c:
$$c \equiv m^e \mod n$$

2.3 Dekryptering

1. Dekryptera chiffertexten c med hemlig nyckel (n, d) för att få m:
$$m \equiv c^d \mod n$$
2. Konvertera heltal m till klartext.

2.4 Dekryptering med offentlig nyckel

Dekryptering av chiffertexten med offentlig nyckel (n, e) genom att rekonstruera den hemliga nyckeln. Detta görs genom att faktorisera n för att hitta p och q, och sedan beräkna d.

1. **Faktorisera modulus n för att hitta p och q:** $n = p \times q$
2. **Beräkna Eulers totientfunktion $\varphi(n)$:** $\varphi(n) = (p-1)(q-1)$
3. **Beräkna den hemliga exponenten d:** $e \times d \equiv 1 \mod \varphi(n)$
4. **Dekryptera chiffertexten c med d:** $m \equiv c^d \mod n$
5. **Konvertera heltalet m till klartext.**

3 Diskussion

Den centrala tanken med RSA-kryptering är dess beroende av den matematiska svårigheten att faktorisera stora tal. I denna lösning faktoriseras modulen n i dess primtalsfaktorer p och q, vilket gör det möjligt att beräkna den hemliga nyckeln och använda den för dekryptering.

3.1 Angående storleken på siffrorna i listan

Siffrorna i chiffertextlistan är alla av liknande storlek på grund av hur RSA-kryptering fungerar. RSA arbetar med block av klartext, och varje block omvandlas till ett stort tal med hjälp av den offentliga nyckeln och modulen. Modulen n bestämmer den övre gränsen för storleken på dessa tal. Eftersom klartexten delas upp i block av lika storlek, tenderar även de resulterande chiffertexttalen att vara av liknande storlek.

Detta beteende är förväntat eftersom RSA säkerställer att varje chiffertextblock är mindre än modulen n. Därför kommer chiffertexttalen att vara relativt lika i storlek, begränsade av värdet av n.

4 Kod

Här nedan följer den fullständiga koden (2 sidor). Den finns också tillgänglig på Github: https://github.com/F7pvqc/Diskret-matte-398476.git

```
def faktorisera_modul(nBob):
    for kandidat in range(2, int(nBob**0.5) + 1):
        if nBob % kandidat == 0:
            return kandidat, nBob // kandidat
    raise ValueError("Inga faktorer hittades")

def utokad_euklides(a, b):
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = utokad_euklides(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def modul_inv(e, phi):
    gcd, x, y = utokad_euklides(e, phi)
    if gcd != 1:
        raise ValueError("Modulär invers existerar inte")
    else:
        return x % phi

def heltal_till_bytes(meddelande):
    byte_langd = (meddelande.bit_length() + 7) // 8
    return meddelande.to_bytes(byte_langd, byteorder='big')

def dekryptera_rsa(chiffertext, nBob, eBob):
    p, q = faktorisera_modul(nBob)
    phi_n = (p - 1) * (q - 1)
    hemlig_nyckel = modul_inv(eBob, phi_n)

    dekrypterade_meddelanden = []
    for chiffer in reversed(chiffertext):
        meddelande = pow(chiffer, hemlig_nyckel, nBob)
        dekrypterade_meddelanden.append(meddelande)

    dekrypterad_text = ''.join(heltal_till_bytes(m).decode('utf-8', errors='ignore')
    for m in dekrypterade_meddelanden)

    dekrypterad_text = dekrypterad_text[::-1]
```

```
    return dekrypterad_text

nBob = 126456119090476383371855906671054993650778797793018127
eBob = 7937

chiffertext = [
    718132566939409242968940779342145611728108797124744 11,
    944882228782809064699486485073739693819382920747629 1,
    886689704353892886973774393969253267419482376824652 70,
    865068771268828494060166866380471028386092481705766 18,
    167098979997847371369576854754375492417010905067822 83,
    112082150953644879808862406205324790087623126644040 573,
    101300870021945928543132671557050279918096489651239 300,
    329377348186985964985545678924627178573516354517528 37,
    103250795649561696933993996191026658588156558009063 626,
    994468839974155247761586401057903618424578341188305 7,
    119023583366882743798043931890543936842945498718476 068,
    805921576014742874989904430677787052561008033956778 17,
    102380508653117028882619903124827386257126674349361 274,
    298119664465635291234712752260079013121187734460427 93,
    927623306494482303991103754637132106161174618068619 15,
    527855801082199310445183087581101002696072325240316 05,
    966307684524306611699000359055643531660884430357009 46,
    104675165348205433706999623683285417639543643952502 324,
    409516327278786875489120073438393722585227830627452 55,
    343964857884196033193147758625493625292680706118412 8,
    926272963564792251805848685941655896141342615621665 37,
    177020269151079849311979753268521304813402328633884 90,
    350462023767324850193339991696871105823051377511486 12,
    772946806923819541057308034354725978013589638323331 13,
    584838889219872414643181096040795870345214047205546 34,
    362766384364001524149641240350093915203907481232436 84,
    516395234668907769094416789131101301147978203091316 76,
    887288722391489727598840188207090806180869995070117 67,
    456761472522561013405286473729879477833152450827017 01,
    237206501172966886536878238699492311404103669744064 35,
    116873909796842028543216809278057888647421675552833624,
    483669286050181729699208399688813823328200632468625 64,
    354254915944117384049165867856166966554119480018879 47,
    404505051187695495064121914793483411856116029353285 69,
    107418270783831708663699380219027152916779513788697702,
    101200673359310801145084267798164209444861857835311695,
    656164892966272513595006085404830191648607553725125 18,
    118474134505765241993518954727967248622755847770105 78,
    273121791554007137166144743648460687727020077792346 4,
    105994187840423492265438067269946241232232359468608 21
```

```
]

dekrypterat_meddelande = dekryptera_rsa(chiffertext, nBob, eBob)
print(dekrypterat_meddelande)
```