CS228 - Cryptography

Nathan Sprague

January 31, 2014

Shift Ciphers

- Encrypt: $f(p) = (p + k) \bmod 26$
- Decrypt: $f^{-1}(p) = (p - k) \bmod 26$

Example: Caesar cipher shifts by 3

```
 Plain: ABCD EFGH IJKL MNOP QRST UVWX YZ
Cipher: DEFG HIJK LMNO PQRS TUVW XYZA BC
```

Cryptanalysis - Breaking Codes

- Shift ciphers are very easy to attack.
  - Brute force, or
  - Letter frequency

Block Ciphers

Block ciphers help prevent frequency attacks

Transposition:

$\sigma(1) = 3$

$\sigma(2) = 1$

$\sigma(3) = 4$

$\sigma(4) = 2$

Encrypt: PIRATE ATTACK (IAPR ETTA AKTC)

Decrypt: SWUE TRAE OEHS (USE WATER HOSE)

(Note: frequency analysis will reveal that this is probably a transposition cipher.)

RSA: Generating Keys

RSA Uses different Keys for encryption/decryption
(Public) Encryption key: (e, n)
(Private) Decryption key: (d, n)

RSA: Generating Keys

RSA Uses different Keys for encryption/decryption
(Public) Encryption key: (e, n)
(Private) Decryption key: (d, n)

- Select two large (200 digit-ish) prime numbers p and q.
  - This can be done efficiently.
- $n = pq$

RSA: Generating Keys

RSA Uses different Keys for encryption/decryption
(Public) Encryption key: (e, n)
(Private) Decryption key: (d, n)

- Select two large (200 digit-ish) prime numbers p and q.
  - This can be done efficiently.
- $n = pq$
- Pick an e that is relatively prime to $(p-1)(q-1)$
  - Euclidean Algorithm can be used to check if $gcd(e, (p-1)(q-1)) = 1$

RSA: Generating Keys

RSA Uses different Keys for encryption/decryption
(Public) Encryption key: (e, n)
(Private) Decryption key: (d, n)

- Select two large (200 digit-ish) prime numbers p and q.
  - This can be done efficiently.
- $n = pq$
- Pick an e that is relatively prime to $(p-1)(q-1)$
  - Euclidean Algorithm can be used to check if $gcd(e, (p-1)(q-1)) = 1$
- Solve for d such that $de \equiv 1 \pmod{(p-1)(q-1)}$
  - We know that there is such a d because of Theorem 1 in section 4.4
  - It can be found efficiently... Process is described in section 4.4.

RSA: Encryption

- Convert blocks of text into integers $m_1, m_2, \ldots m_k$, where each $m_i < n$.
- Encrypt the blocks:
  - $c_i = m_i^e \bmod n$
- Modular exponentiation can be done efficiently using algorithm from Section 4.2.

RSA: Decryption

- Decrypt the blocks:
  - $m_i = c_i^d \bmod n$
- Textbook presents an argument that this works using Fermat's little theoreom and the Chinese remainder theoerom from Section 4.4.
- Convert the blocks back into text.

RSA: Public and Private Keys

- Recall the encryption step:
  - $c_i = m_i^e \bmod n$
- Even if you **know** c_i, e and n, it is not possible to efficiently find m_i.
- You **can** decrypt if you know d:
  - $m_i = c_i^d \bmod n$
- Our method for finding d depends on knowing p and q.
  - No problem!(?) Just factor n...

Public Key Applications: Digital Signatures

- "Encrypt" your message with your **private** key.
- The recipient "decrypts" your message with your public key.
- Recipient knows the message came from you.