CS144: Public-Key Infrastructure (PKI)

Asymmetric-Key Cryptography

• Q: How can two parties send and receive encrypted messages without agreeing on a shared secret key?

• Basic idea
  – Two pairs of keys
    * e: encryption key
    * d: decryption key
  – c = F(m, e): encryption function m = F'(m, d): decryption function
    * Of course, F'(F(m, e), d) == m

• Q: How can we keep communication secret using this mechanism?

• Q: How do we use this to alleviate the key agreement problem?
  – Users share their “encryption” key: public key
    * Others use the public key to encrypt the message to the user
  – Users keep their “decryption” key secret: private key
    * Users use their private key to decrypt message
  – No need to send the secret key over insecure channel
    * Secret key NEVER leave the owner of the key

• Q: What properties should F, F’, e and d satisfy to make this work?
  – One should never guess m from c without d (~ perfect secrecy)
  – One should never guess d from e

• Idea first developed by Ellis, Cocks, and Williams (working for British NSA)
  – In early 70’s, but could not publish
  – First public-key cryptosystem by Diffie and Hellman in 1976

• RSA (Rivest, Shamir and Adleman)
Most widely used asymmetric key cryptography
  * Other example: ECC (elliptic curve cryptography)

Used by many security protocols
  * e.g., SSL, PGP, CDPD, . . .

Algorithm
  1. Pick two random prime numbers p and q.
  2. Pick e < (p-1)(q-1)
     * e does not have to be random
     * Popular choice: e = 65537 (=2^16 + 1), 3, 5, 35, . . .
  3. Find d < (p-1)(q-1) such that “de mod (p-1)(q-1) = 1”
     * Using extended-euclid algorithm

Two important theorems
  1. There exists such unique d if e is a coprime to (p-1)(q-1), i.e., e does not share any factor with (p-1)(q-1)
  2. If n = pq, then m = m^ed mod n

RSA
  * n, e: public key
  * n, d: private key
  * F(m, e): c = m^e mod n
  * F'(c, d): m = c^d mod n

Three things to verify to ensure its “security”
  1. F'( F(m, e), d) == m ?
  2. Can we derive m from c = m^e mod n?
  3. Can we derive d from de mod (p-1)(q-1) = 1 ?

Q: Is F'(F(m, e), d) == m?

Q: Can we compute m from c = m^e mod n?

* RSA problem
Q: Can we compute $d$ by solving $de \mod (p-1)(q-1) = 1$?

* Q: Isn’t it easy to get $p$ and $q$ from $n = pq$?
  * Large-number factorization problem

Note
* Security of RSA depends on the difficulty of factorization and RSA problems
* Asymmetric cryptography is typically 1000x slower than symmetric cryptography

Application of Asymmetric-Key Cryptography

Recap: authentication, authorization, confidentiality, message integrity

- Q: How can we keep message “confidential”?

  - Performance and complexity issue

- Q: How can we “authenticate” the other party?

  - Challenge: generate random value $r$ and send $c = F(r, e)$
  - Response: send back $F'(c, d) = r$

- Q: How can we check the message integrity?

  - Q: How can we make sure others did not temper with checksum?
    - Signature
      * Main idea: $F(F'(m, d), e) = m$
        * In RSA, for example, $m = (m^e)^d = (m^d)^e$
      * Secret key encrypted checksum of the text
* Others can ensure the authenticity of message by decrypting it using public key of the author

**Public-Key Infrastructure (PKI)**

- **Q: How do we know the public key for A really belongs to A?**

  - PKI (public key infrastructure)
  - CA (certificate authority)
    - Guarantees that the public key really belongs to the entity
    - Out of band identity check
    - Issues *certificate* to each entity
    - Certificate
      - “text” (XXXX is the public key of A) signed by CA’s secret key
      - Others can “trust” the public key if they trust CA

- **High-level description of SSL (HTTP)**
  1. When contacted by client, server presents its signed certificate
     “XXX is the public key of amazon.com. This certificate is valid until …”
  2. Client “authenticates” server through challenge/response using the public key
  3. Client/server agrees on a symmetric-key to use using though a secure channel established through asymmetric-key encryption
  4. Client/server communicate securely through symmetric-key encryption

  - Note: real protocol is much more complicated
    - Mutual authentication
    - Handshake of encryption algorithm
    - Make sure freshness of conversation
Multi-Factor Authentication

• **Q:** How should a user pick a secret key?
  
  – User selection vs random-number generator
  – Random-number generator + encryption by user password
  – Note:
    * Need for perfect random number generator
    * Need for “safe” key storage

• **Q:** What if a key/password is stolen?
  
  – *Multi-factor authentication*
    * To minimize possibility of compromised keys, systems authenticate users based on combinations of
      ▶ What you have (e.g., physical key, id card)
      ▶ What you know (e.g., password)
      ▶ Who you are (e.g., fingerprint)
    * 2-factor authentication

• Commonly-used second factor
  
  – Smartphone/Laptop
    * Send an SMS/push notification on a registered device
    * User provides the random number for log in
  – Smartcard
    * Temper-resistant card with a unique secret key
    * Provide smartcard to a smartcard reader for log in
      ▶ Some smartcards perform on-board RSA encryption/decryption to avoid revealing the key to the reader
  – OTP (one time password) key
    * A physical card flashing a new security code, say, every minute
      ▶ e.g. SecurID by RSA security
    * New security codes are generated from current time + “seed key”
      ▶ Server knows the security code generation algorithm
      ▶ Needs to synchronize time between the server and the key
    * User provides the security code to log in
– Biometric key
  * Fingerprint, iris, face, …