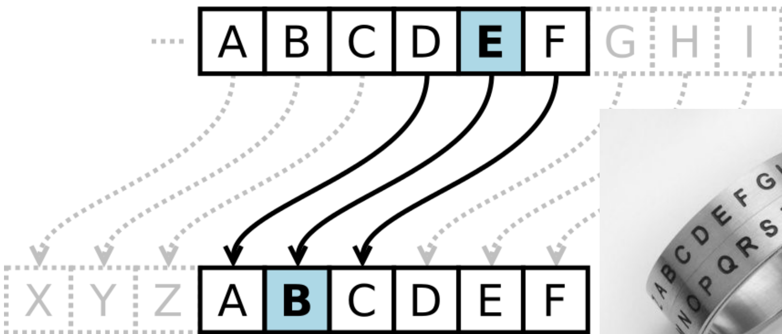
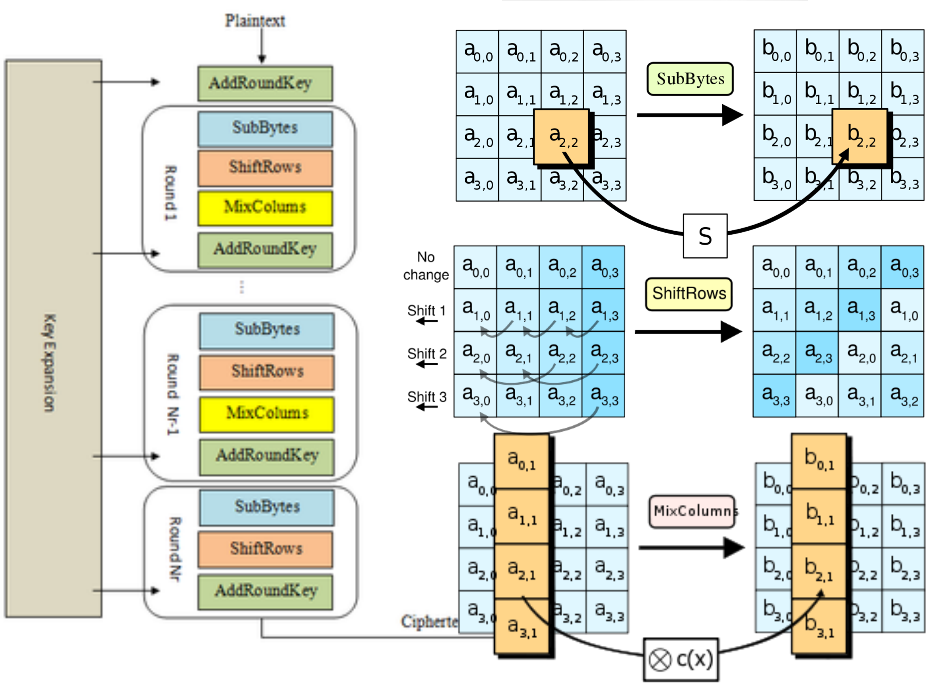
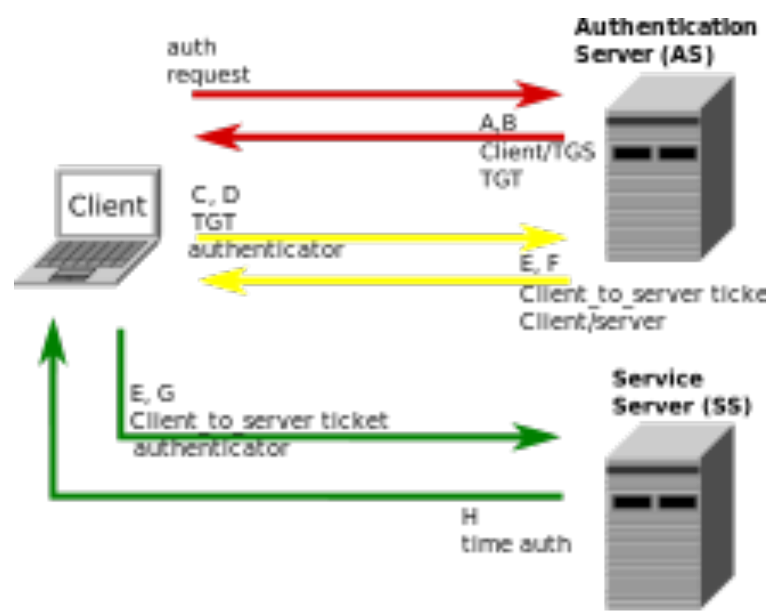
Lecture 9: Public-Key Cryptography

CS 5430 3/05/2018

Crypto Thus Far…







Key pairs

- Instead of sharing a key between pairs of principals...
- ...every principal has a pair of keys
  - **public key:** published for the world to see
  - **private key:** kept secret and never shared


Protocol to exchange encrypted message

```
1. A:  c = Enc(m; K_B)
2. A -> B:  c
3. B:  m = Dec(c; k_B)
```

key pair: `(K_B, k_B)`

- public key written with uppercase letter
- private key written with lowercase letter

Public keys

```
0. B:  (K_B, k_B) = Gen(len)
1.   ...
```

- All public keys published in "phonebook"
- So A can lookup B's key to send message
- Length of phonebook is O(n)
- So quadratic problem reduced to linear!
- Eliminates key distribution problem!

RSA

[Rivest, Shamir, Adleman 1977]

Shared Turing Award in 2002: *ingenious contribution to making public-key crypto*

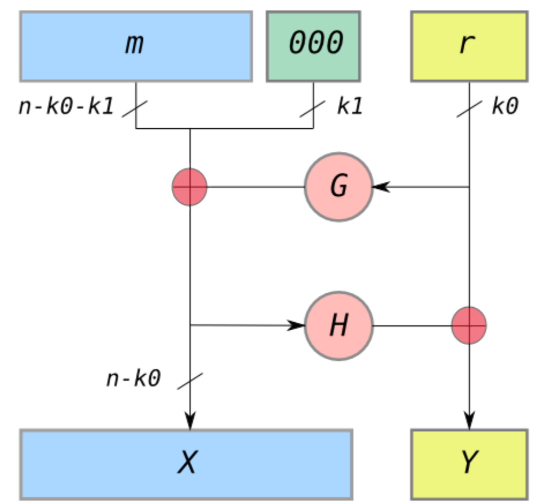
- Pick primes p, q
- Choose e, d such that $ed = 1 \bmod (p-1)(q-1)$
- $PK = (n, e)$
- $SK = (p, q, d)$

$$c = m^e \bmod n$$

$$m = c^d \bmod n$$

Textbook RSA is insecure

- *Deterministic*: given same plaintext and key, always produces the same ciphertext
- Several other attacks, too
- **Solution:** incorporate a nonce in the message before encrypting
  - Called *padding* but *encoding* might be a better term
  - Don't implement yourself; use OAEP implementation in your crypto library (Optimal Asymmetric Encryption Padding)



Problems of length

- Asymmetric encryption uses big integers, not byte arrays
  - all messages must be encoded as integers
  - modulus dictates maximum integer that can be encrypted
  - big integer operations are slow
    - say, **1 to 3 orders of magnitude slower** than block ciphers
- So the problems we had before crop up again...
  - what if message length is too short?
    - actually that's okay: a small integer is still an integer
  - what if message length is too long?
    - in theory could use block modes like with symmetric encryption
    - in practice, that's too inefficient...

HYBRID ENCRYPTION

Hybrid encryption

- Assume:
  - Symmetric encryption scheme (Gen_S, Enc_S, Dec_S)
  - Asymmetric encryption scheme (Gen_A, Enc_A, Dec_A)
- Use asymmetric encryption to establish a shared session key
  - Avoids quadratic problem, assuming existence of phonebook
  - Session key will be short, so avoids inefficiency
- Use symmetric encryption to exchange long plaintext encrypted under session key
  - Gain efficiency of block cipher and mode

Protocol to exchange encrypted message

```
0.  B: (K_B, k_B) = Gen_A(len_A)
1.  A: k_s = Gen_S(len_S)
       c1 = Enc_A(k_s; K_B)
       c2 = Enc_S(m; k_s) //mode
2.  A -> B: c1, c2
3.  B: k_s = Dec_A(c1; k_B)
       m = Dec_S(c2; k_s)
```

Session keys

- If key compromised, only those messages encrypted under it are disclosed
- Used for a brief period then discarded
  - cryptoperiod*:* length of time for which key is valid
  - in this case, for a single (long) message
  - not intended for reuse in future messages
  - only intended for unidirectional usage:
    - A->B, not B->A
    - why? A chose the key, not B

Encryption

- We can now protect confidentiality of messages against Dolev-Yao attacker
  - efficiently, thanks to hybrid of symmetric and asymmetric encryption
- But what about integrity...?

DIGITAL SIGNATURES

Recall: Key pairs

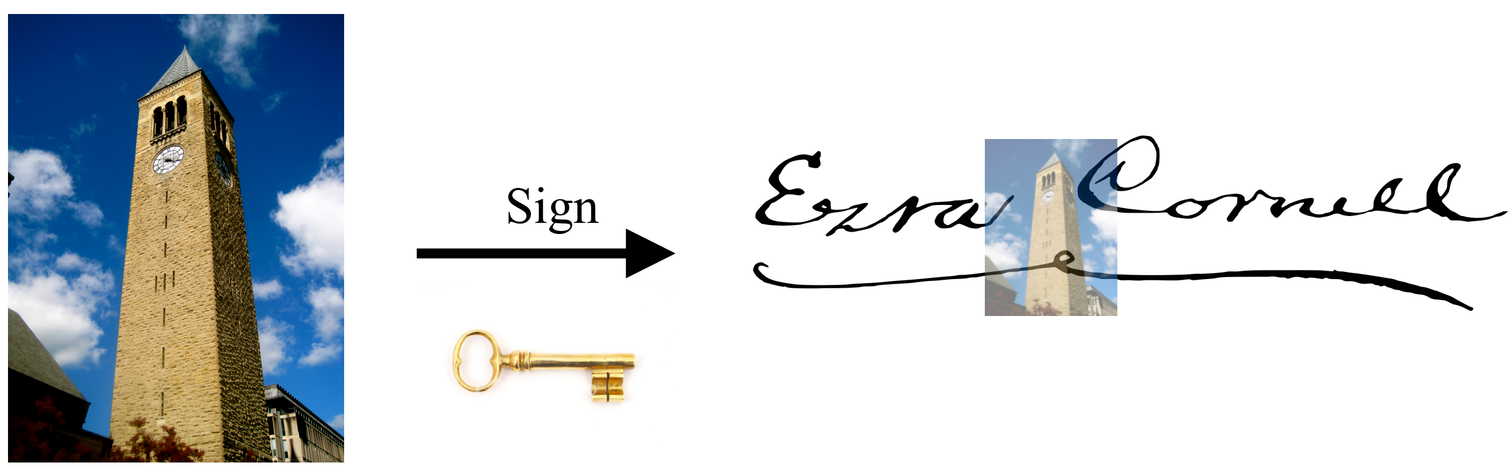
- Instead of sharing a key between pairs of principals...
- ...every principal has a pair of keys
  - **public key:** published for the world to see
  - **private key:** kept secret and never shared

Key pair terminology

	Encryption	Digital signatures
Public key	Encryption key	Verification key
Private key	Decryption key	Signing key

Digital signature scheme

- Sign(m; k): sign message m with key k, producing signature s as output
- Ver(m; s; K): verify signature s on message m with key K
- Gen(len): generate a key pair (K,k) of length len



Protocol to exchange signed message

```
0. A: (K_A,k_A) = Gen(len)
1. A: s = Sign(m; k_A)
2. A -> B: m, s
3. B: accept if Ver(m; s; K_A)
```

- Message is sent in plaintext: no protection of confidentiality
- Goal is to detect modification **not** prevent

Security of digital signatures

- Must be hard to forge signature for a message without knowledge of key

    ...like handwritten signatures

- Even if in possession of multiple (message, signature) pairs for that key

    ...unlike handwritten signatures

RSA

- Core ideas are the same as RSA encryption
- Common mistake: "RSA sign = encrypt with private key"
- Truth (in real world, outside of textbooks):
  - there's a core RSA function R that works with either K or k
  - RSA encrypt = do some prep work on m then call R with K
  - RSA sign = do **different** prep work on m then call R with k
  - Prep work: recall "textbook RSA is insecure"
    - (For encryption: OAEP)
    - For signatures: PSS (probabilistic signature scheme)
    - Also need to handle long messages…

Signatures with hashing

```
1. A: s = Sign(H(m); k_A)
2. A -> B: m, s
3. B: accept if Ver(H(m); s; K_A)
```

So common a practice that I won't bother to write the hashing from now on

DSA

DSA: Digital Signature Algorithm [Kravitz 1991]

- Standardized by NIST and made available royalty-free in 1991/1993
- Used for decades without any serious attacks
- Closely related to Elgamal encryption

Blind signatures

[Chaum 1983]

- Purpose: signer doesn't know what they are signing
- Two additional algorithms: Blind and Unblind
- Unblind(Sign(Blind(m); k)) = Sign(m; k)
- Uses: e-cash, e-voting

Group signatures

[Chaum and van Heyst 1991]

- Purpose: one member of group signs anonymously on behalf of group
- Introduces a *group manager* who controls membership
- Two new protocols: Join and Revoke, to manage membership
- One new algorithm: Open, which manager can run to reveal who signed a message