CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography

Autumn 2018

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Ada Lerner, John Manferdelli, John Mitchell, Franziska Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- HW2: Due Nov 7, 4:30pm

- Looking ahead, rough plan:
- Lab 2 out ~Nov 5, due ~Nov 19 (Quiz Section on Nov 8)
- HW 3 out ~Nov 19, due ~Nov 30
- Lab 3 out ~Nov 26, due Dec 7 (Quiz Section on Nov 29)

- HW1s were **awesome**

Public Key Encryption

Requirements for Public Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to learn anything about M from C without SK
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ ($n \geq 1$) is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that $\varphi(ab) = \varphi(a)\,\varphi(b)$

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 bits each (need primality testing, too)
  - Compute **n**=pq and φ(**n**)=(p-1)(q-1)
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, e=3 or $e=2^{16}+1=65537$
  - Compute unique d such that ed $\equiv$ 1 mod $\varphi(n)$
    - Modular inverse: $d \equiv e^{-1} \bmod \varphi(n)$ ← How to compute?
  - Public key = (e,n); private key = (d,n)
- Encryption of m (m a number between 0 and n-1): $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e \bmod n)^d \bmod n = m$

Why Decryption Works (FYI)

- Decryption of c: c^d mod n = $(m^e \bmod n)^d$ mod n = $(m^e)^d$ mod n = m
- Recall **n**=pq and φ(**n**)=(p-1)(q-1) and ed ≡ 1 mod φ(n)

- Chinese Remaind Theorem: To show m^{ed} mod n ≡ m mod n, sufficient to show:
  - m^{ed} mod p ≡ m mod p
  - m^{ed} mod q ≡ m mod q

- If m ≡ 0 mod p → m^{ed} ≡ 0 mod p

- Else $m^{ed} = m^{ed-1}m = m^{k(q-1)(p-1)}m = m^{h(p-1)}m$ for some k, and h=k(q-1). Why? Recall how d was chosen and the definition of mod.
- Fermat Little Theorem: $m^{(p-1)h}m \equiv 1^h m \bmod p \equiv m \bmod p$

Why is RSA Secure?

- RSA problem: given c, $n=pq$, and e such that $\gcd(e, \varphi(n))=1$, find m such that $m^e=c \bmod n$
  - In other words, recover m from ciphertext c and public key (n,e) by taking e^{th} root of c modulo n
  - There is no *known* efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$

- If factoring is easy, then RSA problem is easy (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- Plain RSA also does <u>not</u> provide integrity
  - Can tamper with encrypted messages

In practice, OAEP is used: instead of encrypting M, encrypt M xor G(r) ; r xor H(M xor G(r))

- r is random and fresh, G and H are hash functions

More on RSA + OAEP

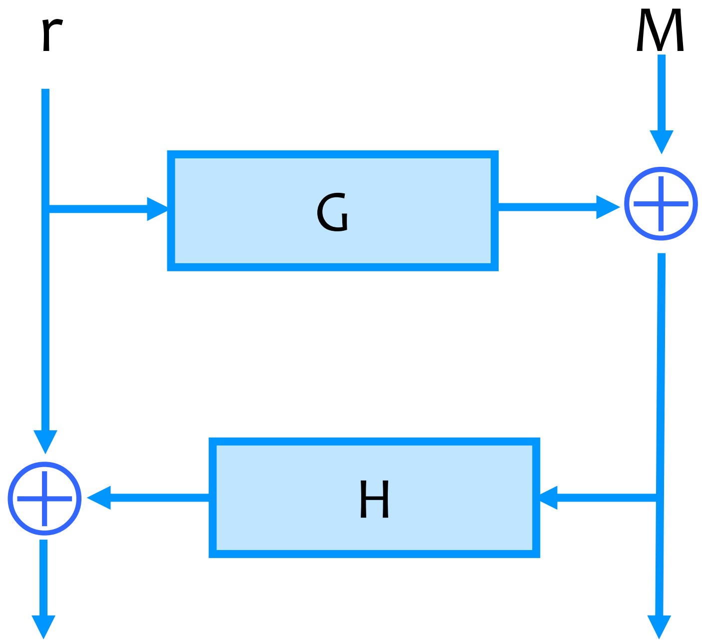
In practice, OAEP is used: instead of encrypting M, encrypt M xor G(r) ; r xor H(M xor G(r))

– r is random and fresh, G and H are hash functions

Question: How do you decrypt a message encrypted with RSA + OAEP?

OAEP as a Figure

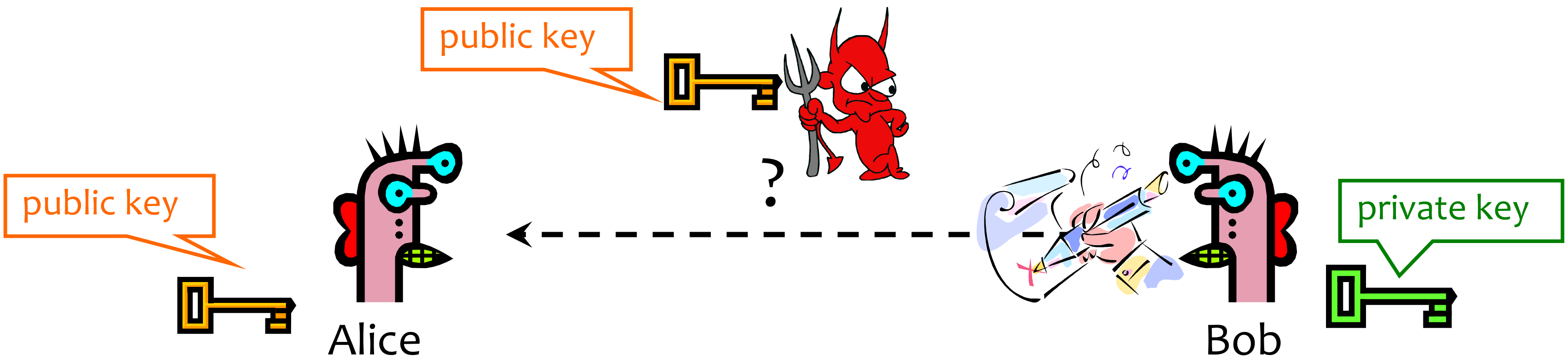
- M xor G(r) ; r xor H(M xor G(r))



- Do you see how to invert? (Side note, similar to DES internals)

Digital Signatures

Digital Signatures: Basic Idea



<u>Given</u>: Everybody knows Bob's public key
Only Bob knows the corresponding private key

<u>Goal</u>: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

RSA Signatures

- Public key is **(n,e)**, private key is **(n,d)**
- To sign message m: $s = m^d \bmod n$
  - Signing & decryption are same **underlying** operation in RSA
  - It's infeasible to compute **s** on **m** if you don't know **d**
- To verify signature s on message m:
  verify that $s^e \bmod n = (m^d)^e \bmod n = m$
  - "Just like encryption" (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- "Just like encryption" in quotes!
  - In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- Public key: (p, q, g, $y=g^x$ mod p), private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from g^x mod p (public key)

- **Important Note: We have discussed discrete logs modulo integers.**
- **Significant advantages in using elliptic curve groups – groups with some similar mathematical properties (i.e., are "groups") but have better security and performance (size) properties**

Stepping Back

Cryptography Summary

- Goal: Privacy
  - Symmetric keys:
    - One-time pad, Stream ciphers
    - Block ciphers (e.g., DES, AES) → modes: EBC, CBC, CTR
  - Public key crypto (e.g., Diffie-Hellman, RSA)
- Goal: Integrity
  - MACs, often using hash functions (e.g, MD5, SHA-256)
- Goal: Privacy and Integrity
  - Encrypt-then-MAC
- Goal: Authenticity (and Integrity)
  - Digital signatures (e.g., RSA, DSS)