CSE 484 / CSE M 584 (Winter 2013)

(Continue) Cryptography

Tadayoshi Kohno

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

Goals for Today

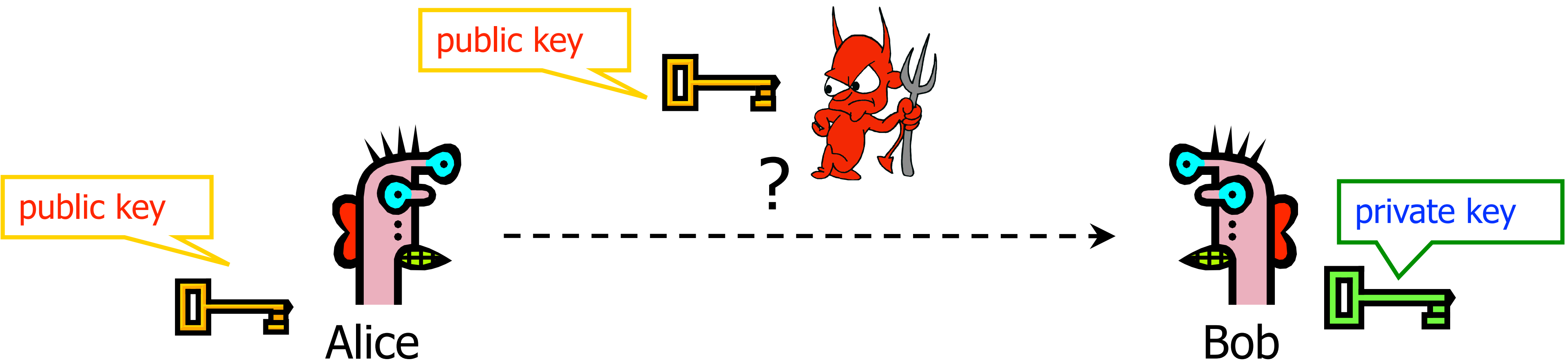
- Cryptography: Now on to asymmetric cryptography
- HW2 out soon (on cryptography)

(Reminder:) Symmetric Cryptography

- **1 secret key (or 2 or 3 or 4)**, shared between sender/receiver
- Repeat fast and simple operations lots of times (rounds) to mix up key and ciphertext
- **Why do we think it is secure?** (simplistic)
  - Lots of heuristic arguments
    - If we do lots and lots and lots of mixing, no simple formula (and reversible) describing the whole process (cryptographic weakness).
    - Mix in ways we think it's hard to short-circuit all the rounds. Especially non-linear mixing, e.g., S-boxes.
  - Some math gives us confidence in these assumptions

Public Key Cryptography

Basic Problem



Given: Everybody knows Bob's public key

Only Bob knows the corresponding private key

Goals: 1. Alice wants to send a secret message to Bob

2. Bob wants to authenticate himself

Public-Key Cryptography

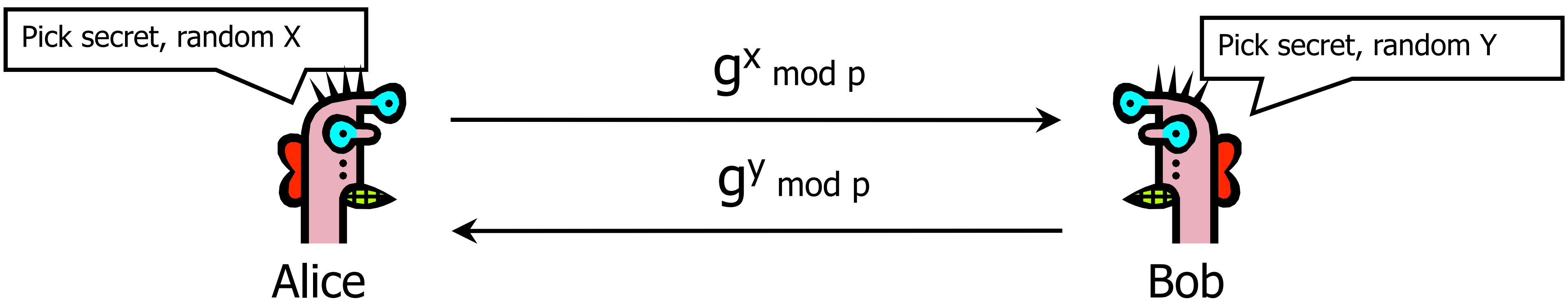
- Everyone has **1 private key and 1 public key**
  - **Or 2 private and 2 public, when considering both encryption and authentication**
- Mathematical relationship between private and public keys
- **Why do we think it is secure?** (simplistic)
  - Relies entirely on **problems we believe are "hard"**

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (or at least different)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can "sign" a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime number, g is a generator of Z_p^*
    - $Z_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in Z_p^* \; \exists i$ such that $a = g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach p



Compute $k=(g^y)^x=g^{xy} \bmod p$

Compute $k=(g^x)^y=g^{xy} \bmod p$

http://www.wolframalpha.com/ and http://www.google.com

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:

  given $g^x \bmod p$, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:

  given g^x and g^y, it's hard to compute $g^{xy} \bmod p$
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:

  given g^x and g^y, it's hard to tell the difference between $g^{xy} \bmod p$ and $g^r \bmod p$ where r is random

Properties of Diffie-Hellman

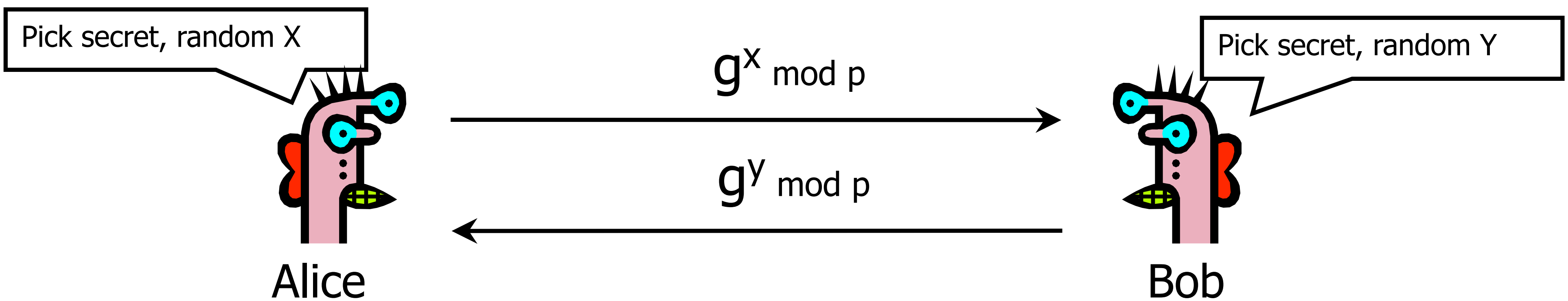
- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Many times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication

Properties of Diffie-Hellman

- DDH: not true for integers mod p, but true for other groups
- DL problem in p can be broken down into DL problems for subgroups, if factorization of p-1 is known.
- Common recommendation:
  - Choose p = 2q+1 where q is also a large prime.
  - Pick a g that generates a subgroup of order q in Z_p*
    - DDH is hard for this group
    - (OK to not know all the details of why for this course.)
  - Hash output of DH key exchange to get the key

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p, q are large prime numbers, p=2q+1, g a generator for the subgroup of order q
    - Modular arithmetic: numbers "wrap around" after they reach p



Compute $k=H((g^y)^x)=H(g^{xy} \bmod p)$ Compute $k=H((g^x)^y)=H(g^{xy} \bmod p)$

Requirements for Public-Key Encryption

- **Key generation:** computationally easy to generate a pair (public key PK, private key SK)
  - Computationally infeasible to determine private key SK given only public key PK
- **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 compute M from C without SK
  - Even infeasible to learn partial information about M
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ 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
- Euler's theorem:

  if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$

  Z_n^*: multiplicative group of integers mod n (integers relatively prime to n)
- Special case: <u>Fermat's Little Theorem</u>

  if p is prime and $gcd(a,p)=1$, then $a^{p-1} = 1 \bmod p$

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 $\varphi(n)=(p-1)(q-1)$
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, $e=3$ or $e=2^{16}+1=65537$ (why?)
  - Compute unique d such that $ed = 1 \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
  - Modular exponentiation by repeated squaring
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why RSA Decryption Works (Simplified)

- $e \cdot d = 1 \bmod \varphi(n)$, thus $e \cdot d = 1 + k \cdot \varphi(n)$ for some k
  Can rewrite: $e \cdot d = 1 + k(p-1)(q-1)$

- Let m be any integer in Z_n^* (not all of Z_n)
- $c^d \bmod n = (m^e)^d \bmod n$
- $= m^{1+k(p-1)(q-1)} \bmod n$
- $= (m \bmod n) * (m^{k(p-1)(q-1)} \bmod n)$
- Recall: Euler's theorem:
  if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$
- $c^d \bmod n = (m \bmod n) * (1 \bmod n)$
- $= m \bmod n$
- But: True for all m in Z_n, not just m in Z_n^*

Why RSA Decryption Works (skip)

- $e \cdot d = 1 \bmod \varphi(n)$, thus $e \cdot d = 1 + k \cdot \varphi(n)$ for some k
  Can rewrite: $e \cdot d = 1 + k(p-1)(q-1)$

- Let m be any integer in Z_n
- If $\gcd(m,p)=1$, then $m^{ed} = m \bmod p$
  - By Fermat's Little Theorem, $m^{p-1} = 1 \bmod p$
  - Raise both sides to the power $k(q-1)$ and multiply by m
  - $m^{1+k(p-1)(q-1)} = m \bmod p$, thus $m^{ed} = m \bmod p$
  - By the same argument, $m^{ed} = m \bmod q$
- Since p and q are distinct primes and $p \cdot q = n$,
  $m^{ed} = m \bmod n$ (using the Chinese Remainder Theorem)
- True for all m in Z_n, not just m in Z_n^*

Why Is RSA Secure?

- RSA problem: given $n=pq$, e such that $gcd(e, \varphi(n))=1$ and c, find m such that $m^e=c \bmod n$
  - i.e., recover m from ciphertext c and public key (n,e) by taking e^{th} root of c
  - 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 (because knowing factors means you can compute d -- inverse of e mod (p-1)(q-1)), but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

On RSA encryption

- Encrypted message needs to be in interpreted as an integer less than n
  - Reason: Otherwise can't decrypt.
  - Message is very often a symmetric encryption key.
- But still not quite that simple

Caveats

- e =3 is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of c to recover m (i.e., no operations taken module n)
    - Even problems if "pad" m in some ways [Hastad]
  - Let $c_i = m^3 \bmod n_i$ - same message is encrypted to three people
    - Adversary can compute $m^3 \bmod n_1 n_2 n_3$ (using CRT)
    - Then take ordinary cube root to recover m
- Don't use RSA **directly** for privacy! Need to pre-process input in some way.

Sample Encryption

- 26 2 15 13 7 14 13 13 1 28 14 15 13 14 20 9 6 31 25 26 14 16 23 15 26 2 6 13 1

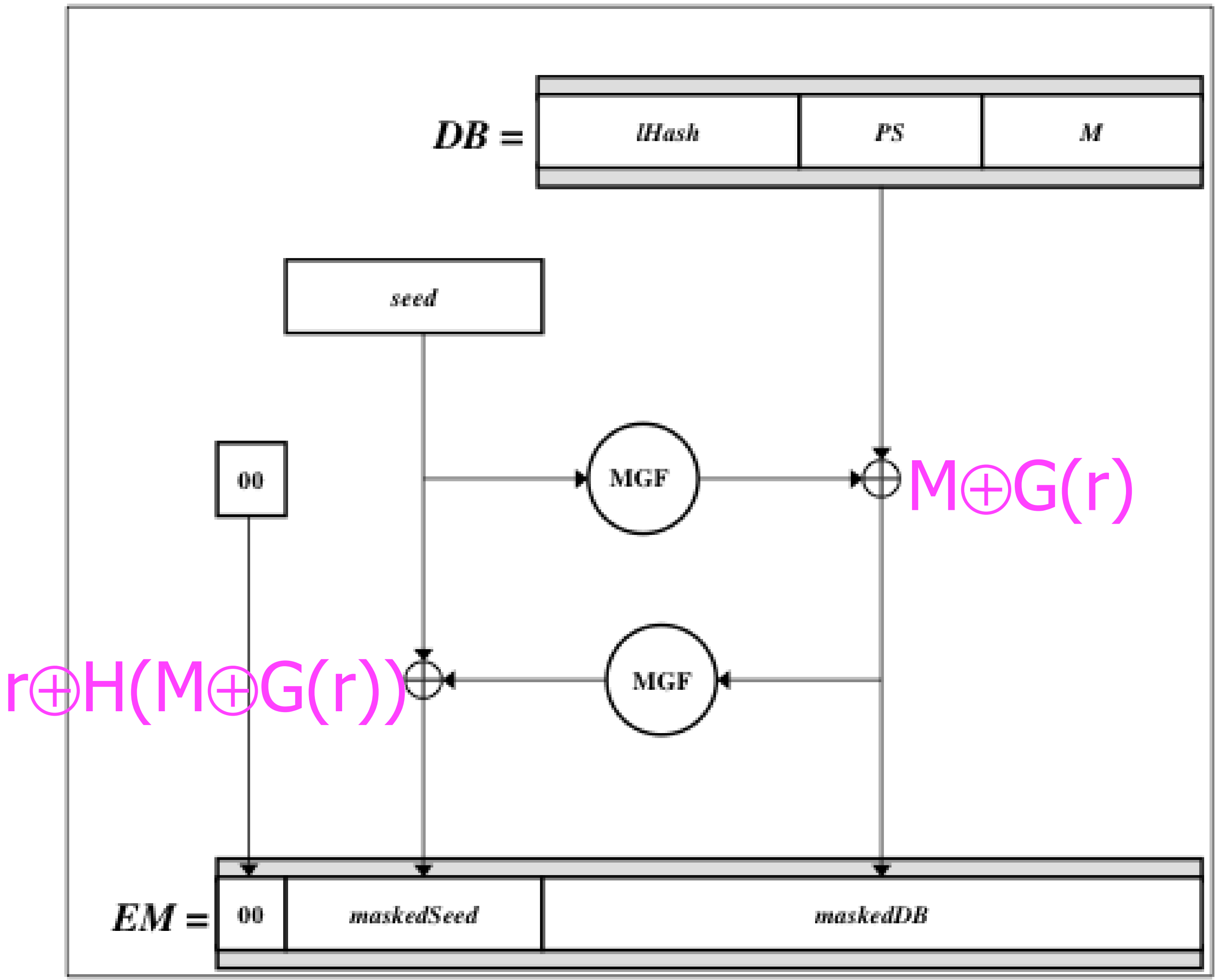
- P=3, Q=11, N=33, E=7, D=3
- ‘A’ converted to 1 before encryption; ‘B’ Converted to 2 before encryption; ...

- A-1 B-2 C-3 D-4 E-5 F-6 G-7 H-8 I-9 J-10 K-11 L-12 M-13 N-14 O-15 P-16 Q-17 R-18 S-19 T-20 U-21 V-22 W-23 X-24 Y-25 Z-26
- http://www.wolframalpha.com/ or http://www.google.com

Integrity in RSA Encryption

- Plain RSA does <u>not</u> provide integrity
  - Given encryptions of m_1 and m_2, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \bmod n = (m_1 \cdot m_2)^e \bmod n$
  - Attacker can convert m into m^k without decrypting
    - $(m_1^e)^k \bmod n = (m^k)^e \bmod n$
- In practice, OAEP is used: instead of encrypting M, encrypt $M \oplus G(r) \,;\, r \oplus H(M \oplus G(r))$
  - r is random and fresh, G and H are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - … if hash functions are “good” and RSA problem is hard

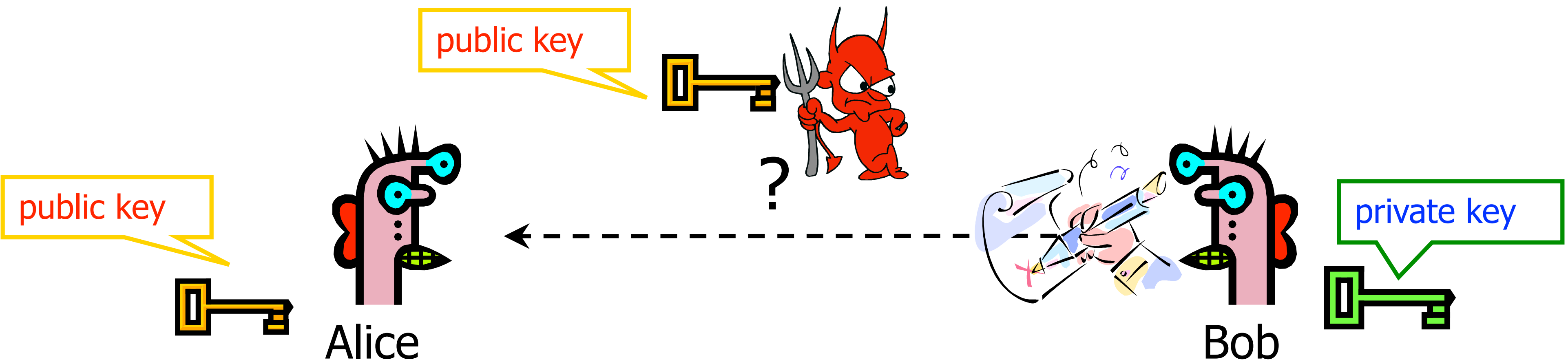
OAEP (image from PKCS #1 v2.1)



Summary of RSA

- Defined RSA primitives
  - Encryption and Decryption
  - Underlying number theory
  - Practical concerns, some mis-uses
  - OAEP

Digital Signatures: Basic Idea



Given: Everybody knows Bob's public key

Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key

RSA Signatures

- Public key is (n,e), private key is d
- To sign message m: $s = m^d \bmod n$
  - Signing and decryption are the 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
  - Anyone who knows n and e (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures

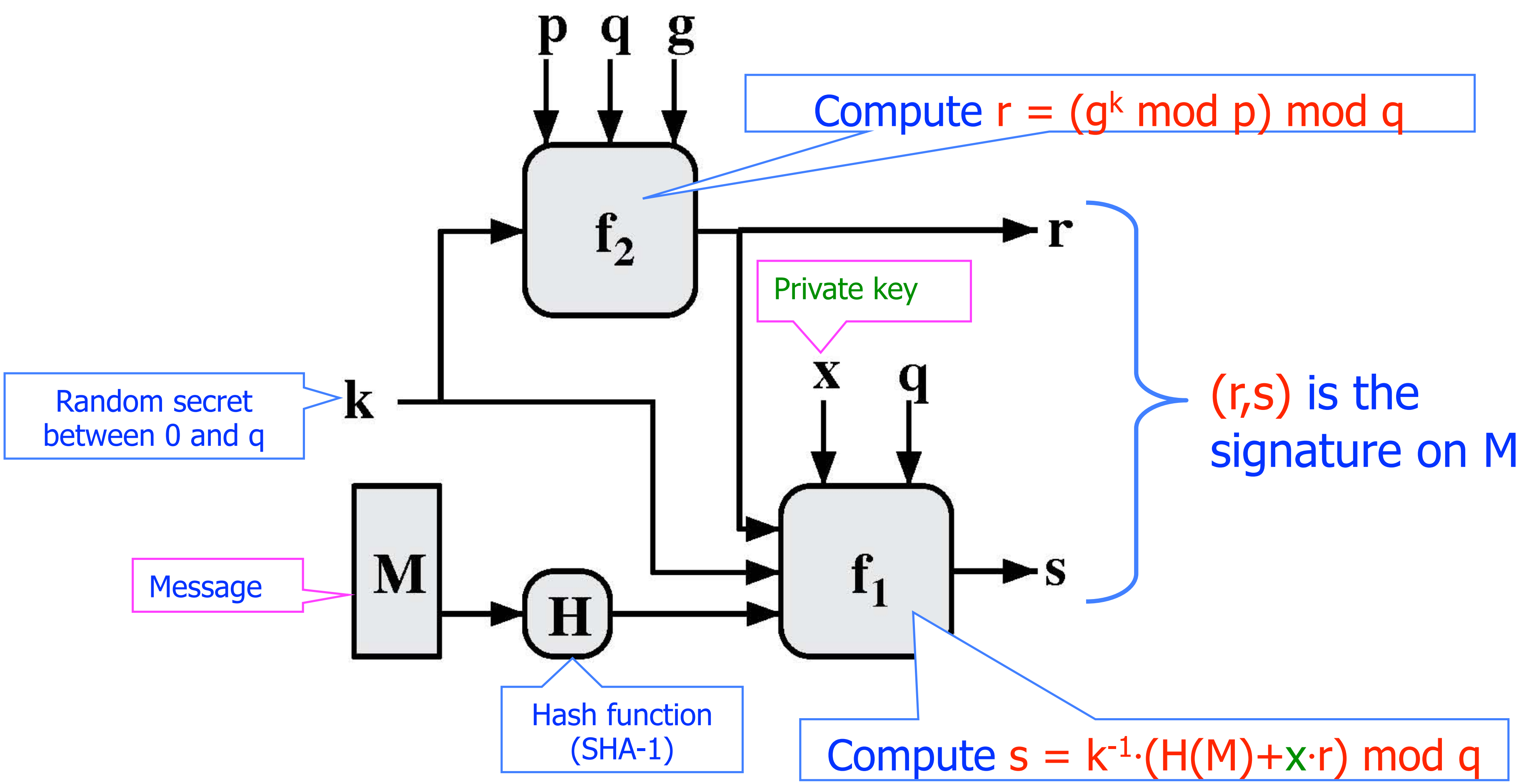
Encryption and Signatures

- Often people think: Encryption and decryption are inverses.
- That's a common view
  - True for the RSA **primitive (underlying component)**
- But not one we'll take
  - To really use RSA, we need padding
  - And there are many other decryption methods
  - And there are many other signing methods

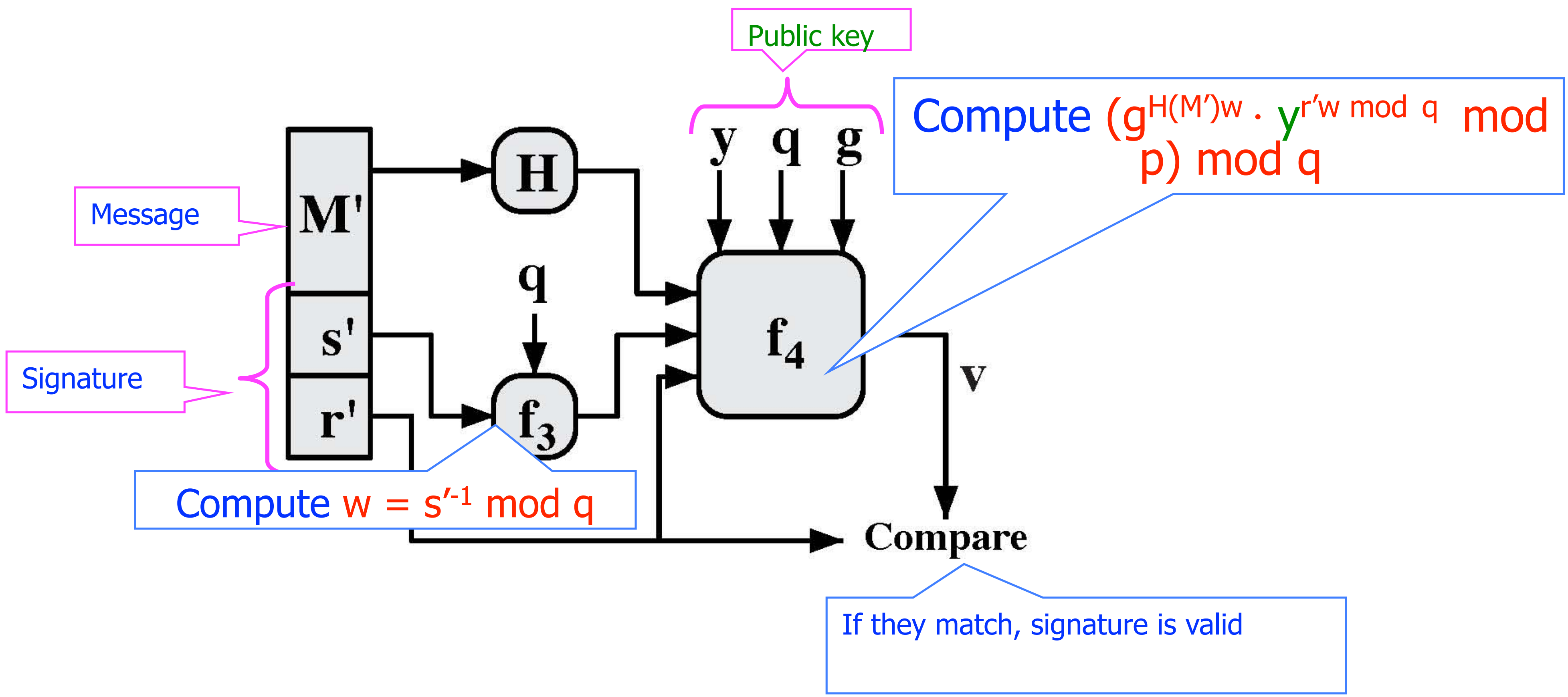
Digital Signature Standard (DSS) (Skim Details)

- U.S. government standard (1991-94)
  - Modification of the ElGamal signature scheme (1985)
- Key generation:
  - Generate large primes p, q such that q divides p-1
    - $2^{159} < q < 2^{160}$, $2^{511+64t} < p < 2^{512+64t}$ where $0 \leq t \leq 8$
  - Select $h \in Z_p^*$ and compute $g = h^{(p-1)/q} \bmod p$
  - Select random x such $1 \leq x \leq q-1$, compute $y = g^x \bmod p$
- Public key: $(p, q, g, y = g^x \bmod p)$, private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from $g^x \bmod p$ (public key)

DSS: Signing a Message (Skim)



DSS: Verifying a Signature (Skim)



Advantages of Public-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - **Fewer** "chicken-and-egg" key establishment problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
    - (With caveats)
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
- Reduce protection of information to protection of authenticity of public keys and secrecy of individual private keys
  - No need to keep public keys secret, but must be sure that Alice's public key is really her true public key

Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - E.g., IPsec, SSL, SSH, ...
- Keys are longer
  - 1024+ bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - What if factoring is easy?
    - Factoring is believed to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions)