CSE 484 / CSE M 584 (Spring 2012)

Asymmetric 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

- Asymmetric Cryptography

- Lab 3 this week
- HW 2 also announced this week.
- There will also be an extra credit HW assignment (can only help your grade)

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

- $e \cdot d = 1 \bmod \varphi(n)$, 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 $\gcd(e,(p-1)(q-1))=1$ 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), 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.

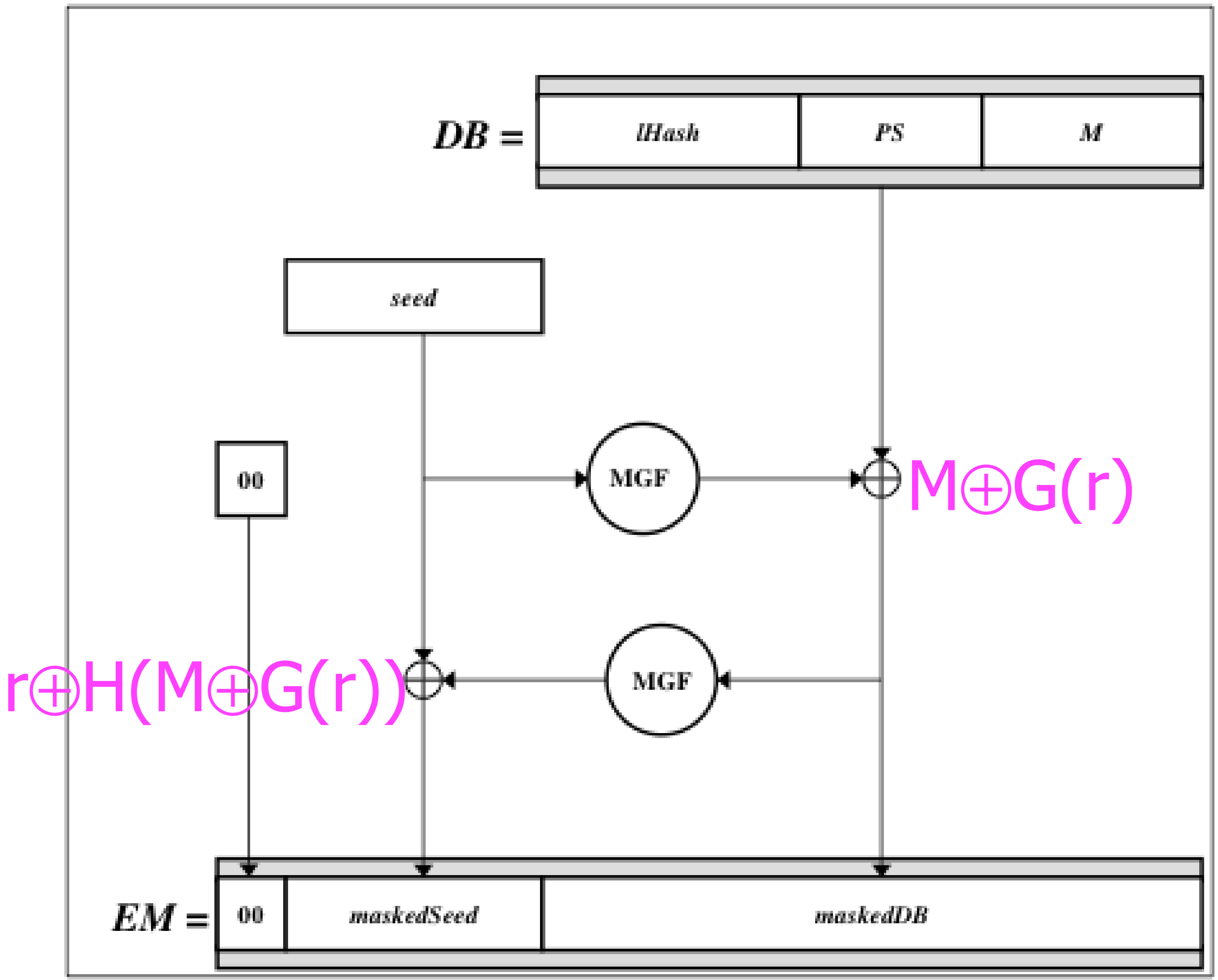
Sample Encryption

- 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/

Integrity in RSA Encryption

- Plain RSA does not 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

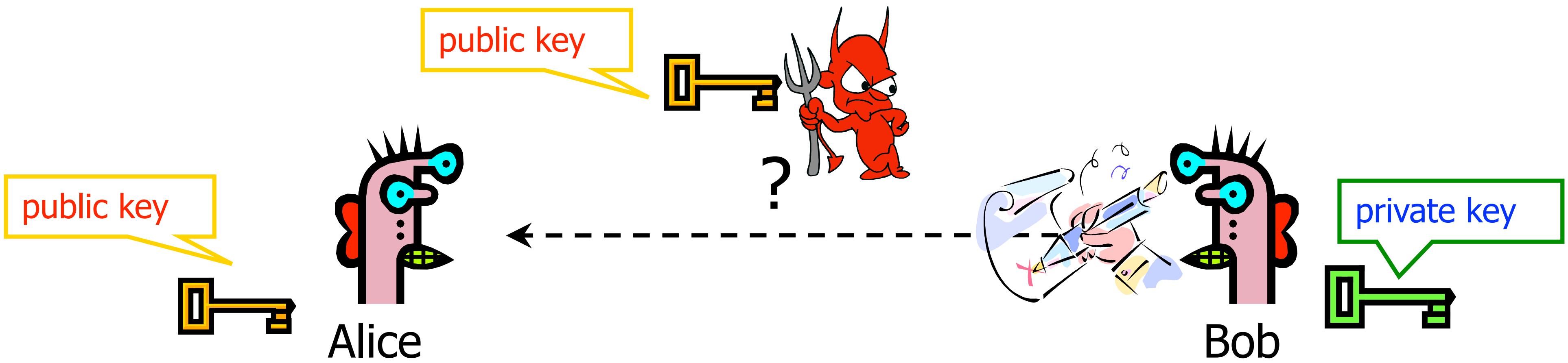
OAEP (image from PKCS #1 v2.1)



Summary of RSA

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

Digital Signatures: Basic Idea



<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, 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:

  $s^e \bmod n = (m^d)^e \bmod n = m$
  - 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

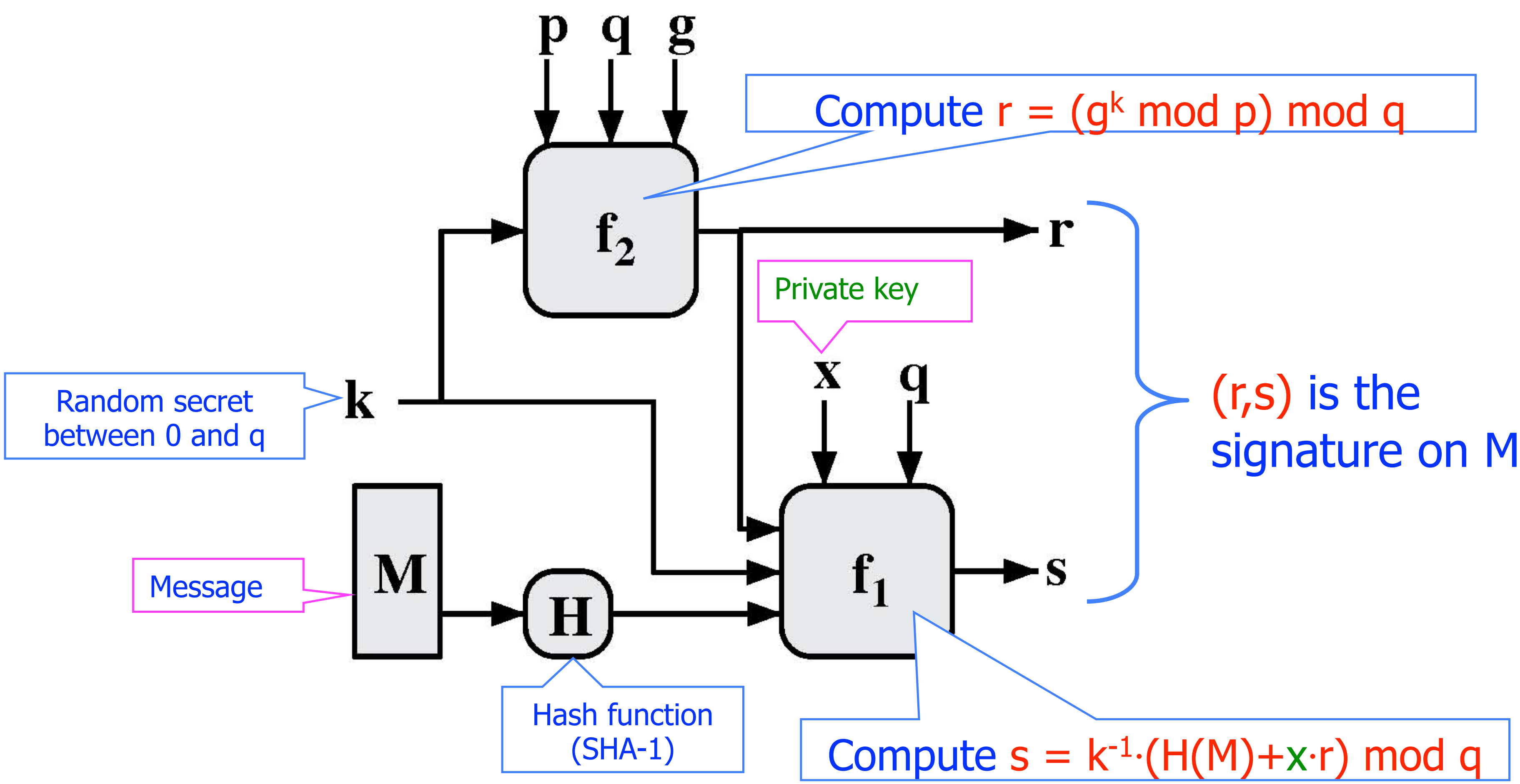
Encryption and Signatures

- 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

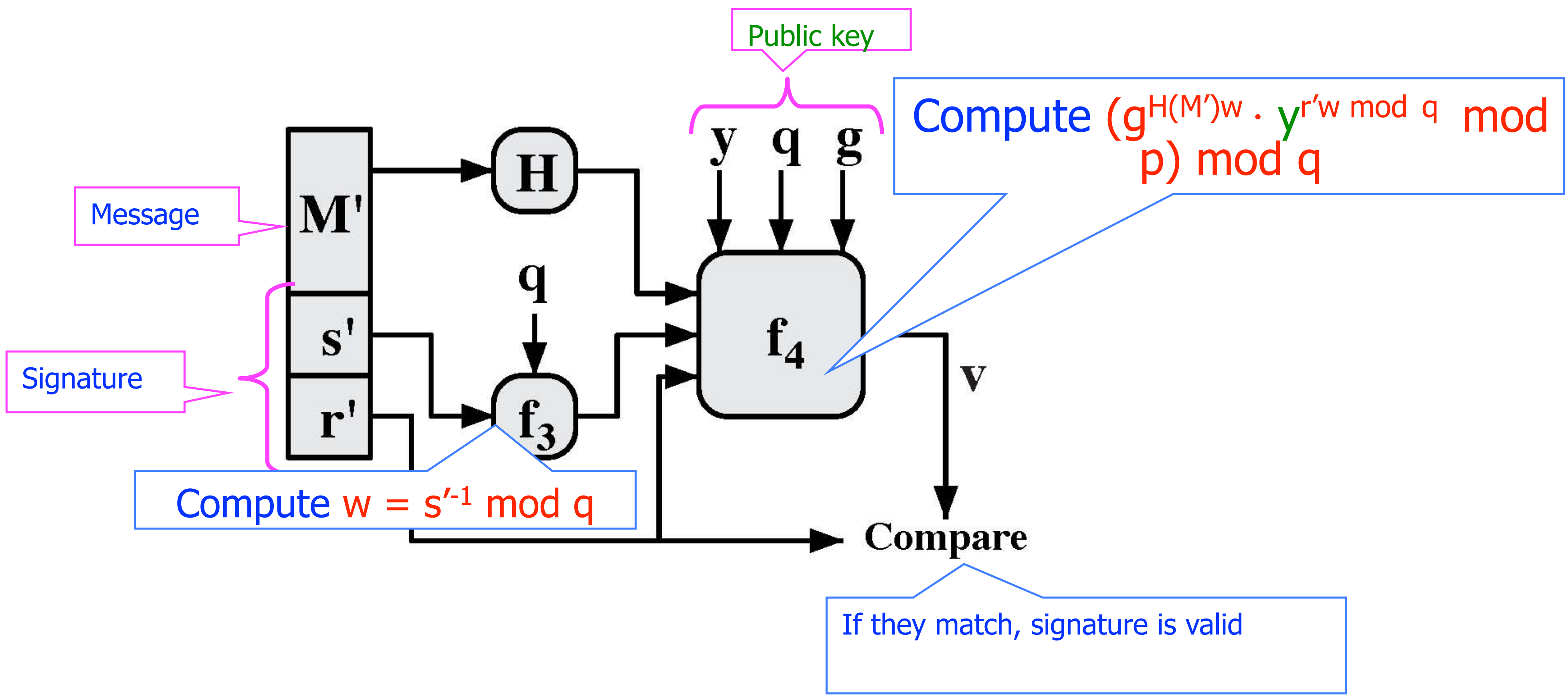
Digital Signature Standard (DSS)

- 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
  - No "chicken-and-egg" key establishment problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
    - Caveats to come
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
- Reduce protection of information to protection of authenticity of public keys
  - No need to keep public keys secret, but must be sure that Alice's public key is <u>really</u> 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)

Note: Optimizing Exponentiation

- How to compute M^x mod N? Say x=13
- Sums of power of 2, x = 8+4+1 = $2^3+2^2+2^0$
- Can also write x in binary, e.g., x = 1101
- Can solve by repeated squaring
  - y = 1;
  - $y = y^2 * M \bmod N$ // $y = M$
  - $y = y^2 * M \bmod N$ // $y = M^2 * M = M^{2+1} = M^3$
  - $y = y^2 \bmod N$ // $y = (M^{2+1})^2 = M^{4+2}$
  - $y = y^2 * M \bmod N$ // $y = (M^{4+2})^2 * M = M^{8+4+1}$
- Does anyone see a potential issue?

Timing attacks

Collect timings for exponentiation with a bunch of messages M1, M2, ... (e.g., RSA signing operations with a private exponent)

Assume (inductively) know b_3=1, b_2=1, guess b_1=1

i	b_i = 0	b_i = 1	Comp	Meas
3	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		
2	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		
1	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$	X1 secs	
0	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		Y1 secs

i	b_i = 0	b_i = 1	Comp	Meas
3	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		
2	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		
1	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$	X2 secs	
0	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		Y2 secs

Timing attacks

- If b_1 = 1, then set of { Yj - Xj | j in {1,2, ..} } has distribution with "small" variance (due to time for final step, i=0)
  - "Guess" was correct when we computed X1, X2, ...
- If b_1 = 0, then set of { Yj - Xj | j in {1,2, ..} } has distribution with "large" variance (due to time for final step, i=0, and incorrect guess for b_1)
  - "Guess" was incorrect when we computed X1, X2, ...
  - So time computation wrong (Xj computed as large, but really small, ...)
- Strategy: Force user to sign large number of messages M1, M2, ... Record timings for signing.
- Iteratively learn bits of key by using above property.