

# Overview of Public-Key Cryptography

---

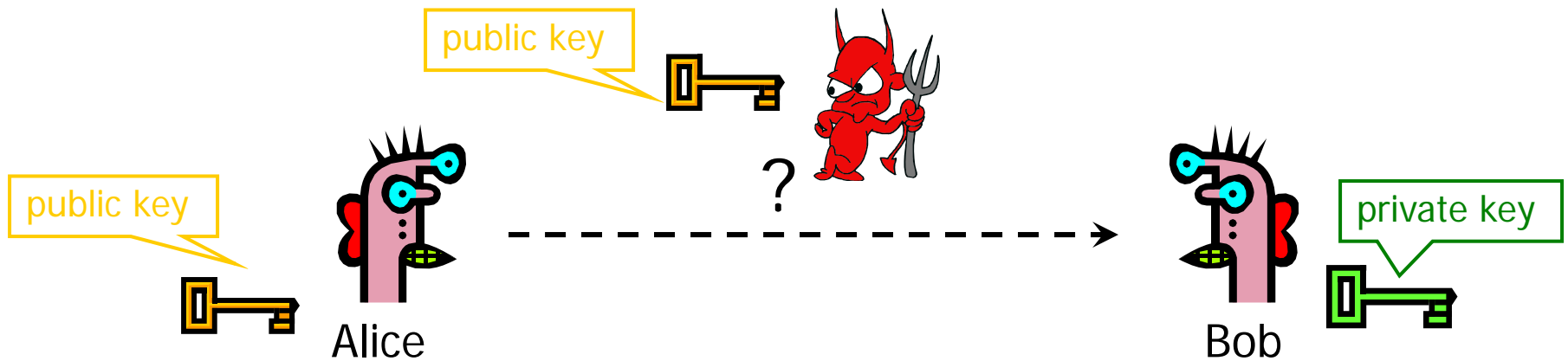
Vitaly Shmatikov

# Reading Assignment

---

◆ Kaufman 6.1-6

# Basic Problem



Given: Everybody knows Bob's **public key**

- How is this achieved in practice?

Only Bob knows the corresponding **private key**

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

# 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 (maybe)
  - Secret is stored only at one site: good for open environments

## ◆ Digital signatures for authentication

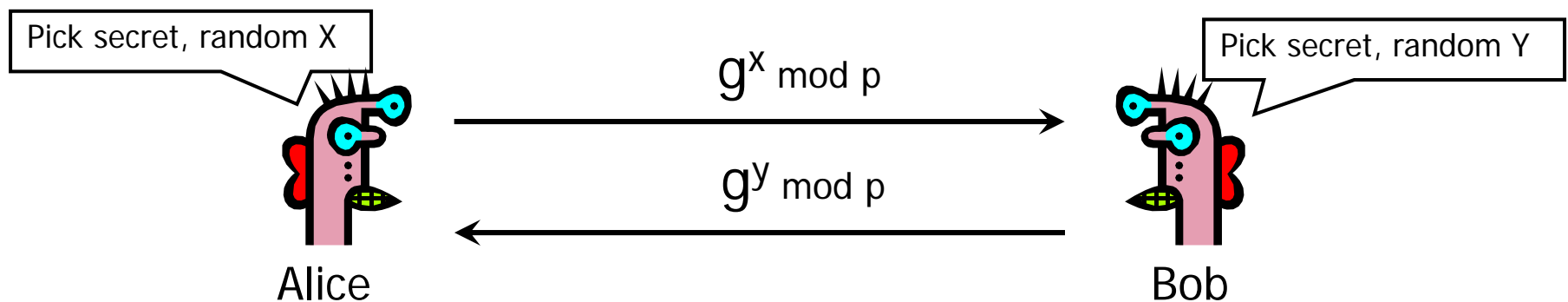
- Can “sign” a message with your private key

## ◆ 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  $Z_p^*$ 
    - $Z_p^* = \{1, 2 \dots p-1\}$ ;  $\forall a \in Z_p^* \exists i$  such that  $a = g^i \pmod p$
    - Modular arithmetic: numbers “wrap around” after they reach  $p$



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

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

# Why Is Diffie-Hellman Secure?

---

- ◆ **Discrete Logarithm (DL)** problem:  
given  $g^x \pmod p$ , 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  $g^{xy} \pmod p$ 
  - ... 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  $g^{xy} \pmod p$  and  $g^r \pmod p$  where  $r$  is random

# Properties of Diffie-Hellman

---

- ◆ Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- ◆ Diffie-Hellman protocol does not provide authentication
  - When we talk about IPsec, we'll see how to combine Diffie-Hellman with signatures, anti-DoS cookies, etc.

# 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
  - Trapdoor function:  $\text{Decrypt}(SK, \text{Encrypt}(PK, M)) = M$

# 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  $a \in \mathbb{Z}_n^*$ , then  $a^{\varphi(n)} = 1 \pmod n$
- ◆ Special case: Fermat's Little Theorem  
if  $p$  is prime and  $\gcd(a, p) = 1$ , then  $a^{p-1} = 1 \pmod p$

# 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,  $e=3$  (may be vulnerable) or  $e=2^{16}+1=65537$  (why?)
- Compute unique  $d$  such that  $ed = 1 \pmod{\varphi(n)}$
- Public key =  $(e,n)$ ; private key =  $d$

## ◆ Encryption of $m$ : $c = m^e \pmod n$

- Modular exponentiation by repeated squaring

## ◆ Decryption of $c$ : $c^d \pmod n = (m^e)^d \pmod n = m$

# Why RSA Decryption Works

---

- ◆  $e \cdot d = 1 \pmod{\varphi(n)}$
- ◆ Thus  $e \cdot d = 1 + k \cdot \varphi(n) = 1 + k(p-1)(q-1)$  for some  $k$
- ◆ If  $\gcd(m, p) = 1$ , then  $m^{ed} = m \pmod{p}$ 
  - By Fermat's Little Theorem,  $m^{p-1} = 1 \pmod{p}$
  - Raise both sides to the power  $k(q-1)$  and multiply by  $m$
  - $m^{1+k(p-1)(q-1)} = m \pmod{p}$ , thus  $m^{ed} = m \pmod{p}$
  - By the same argument,  $m^{ed} = m \pmod{q}$
- ◆ Since  $p$  and  $q$  are distinct primes and  $p \cdot q = n$ ,  
 $m^{ed} = m \pmod{n}$

# 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  $m^e=c \pmod n$ 
  - i.e., recover  $m$  from ciphertext  $c$  and public key  $(n,e)$  by taking  $e^{\text{th}}$  root of  $c$
  - There is no known efficient algorithm for doing this
- ◆ **Factoring** problem: given positive integer  $n$ , find primes  $p_1, \dots, p_k$  such that  $n=p_1^{e_1}p_2^{e_2}\dots p_k^{e_k}$
- ◆ If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring  $n$

# Integrity in RSA Encryption

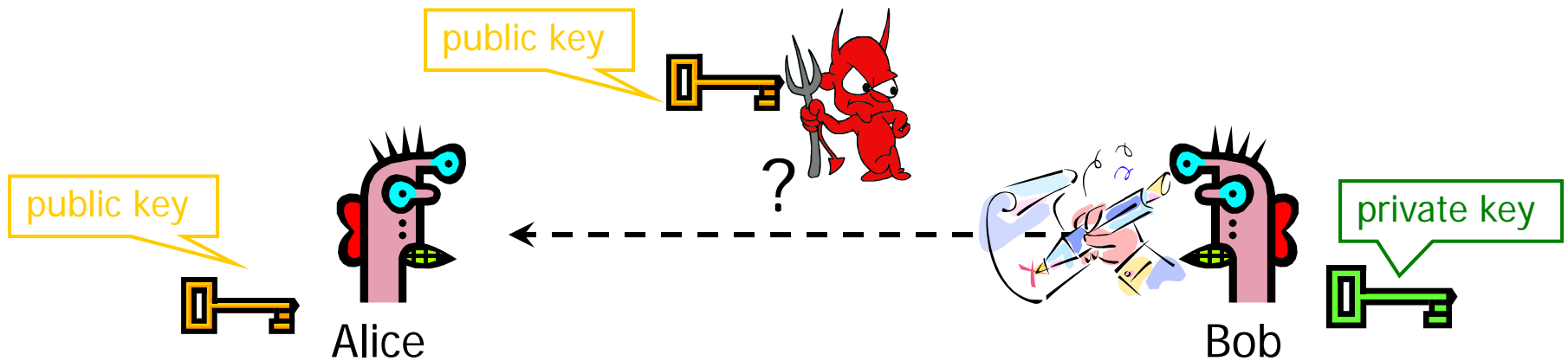
## ◆ Plain RSA does not provide integrity

- 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
  - $(m^e)^k \bmod n = (m^k)^e \bmod n$

## ◆ 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

# 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

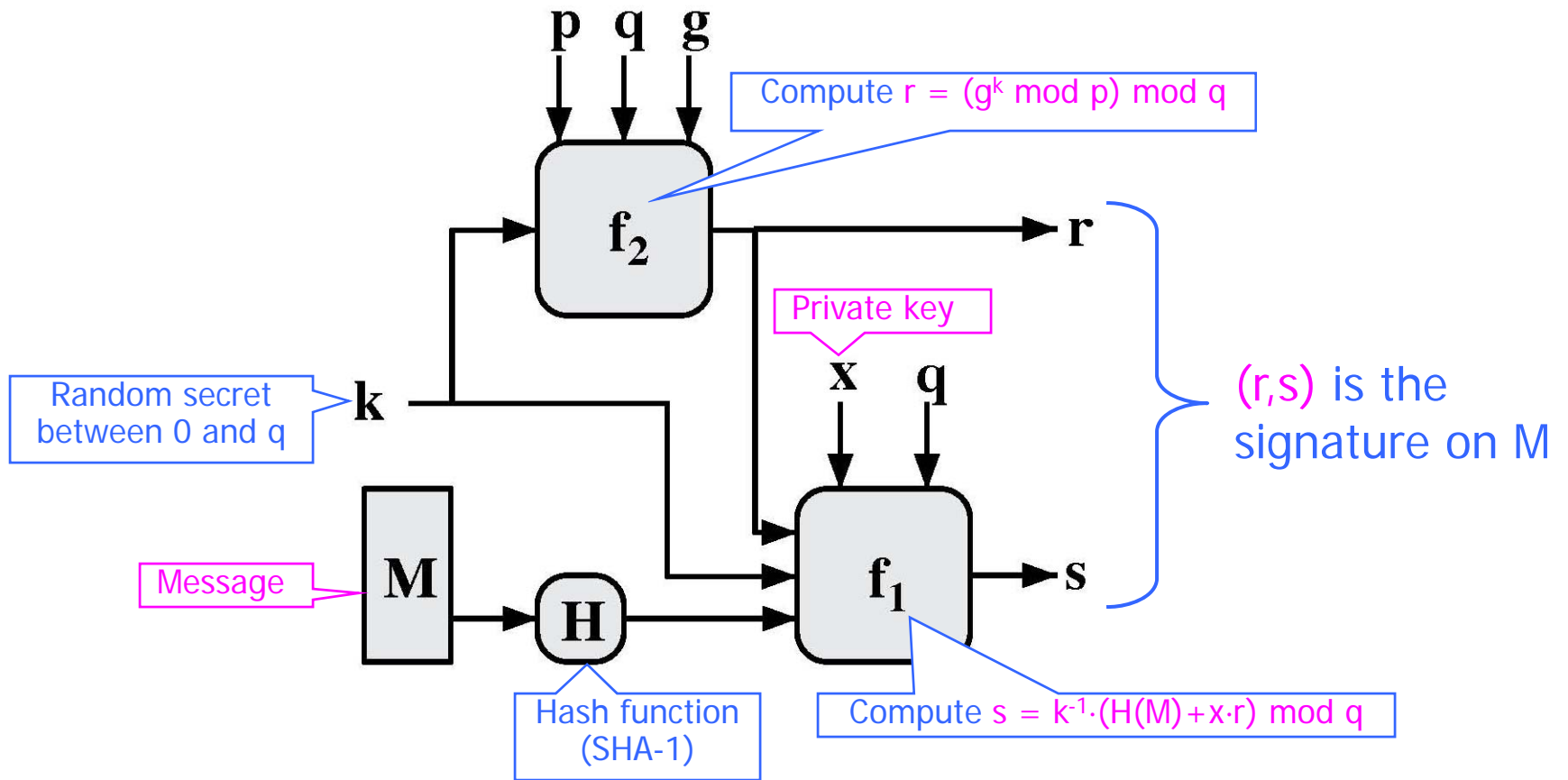
---

- ◆ Public key is  $(n, e)$ , private key is  $d$
- ◆ To **sign** message  $m$ :  $s = m^d \bmod n$ 
  - Signing and decryption are the same 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 (why?)

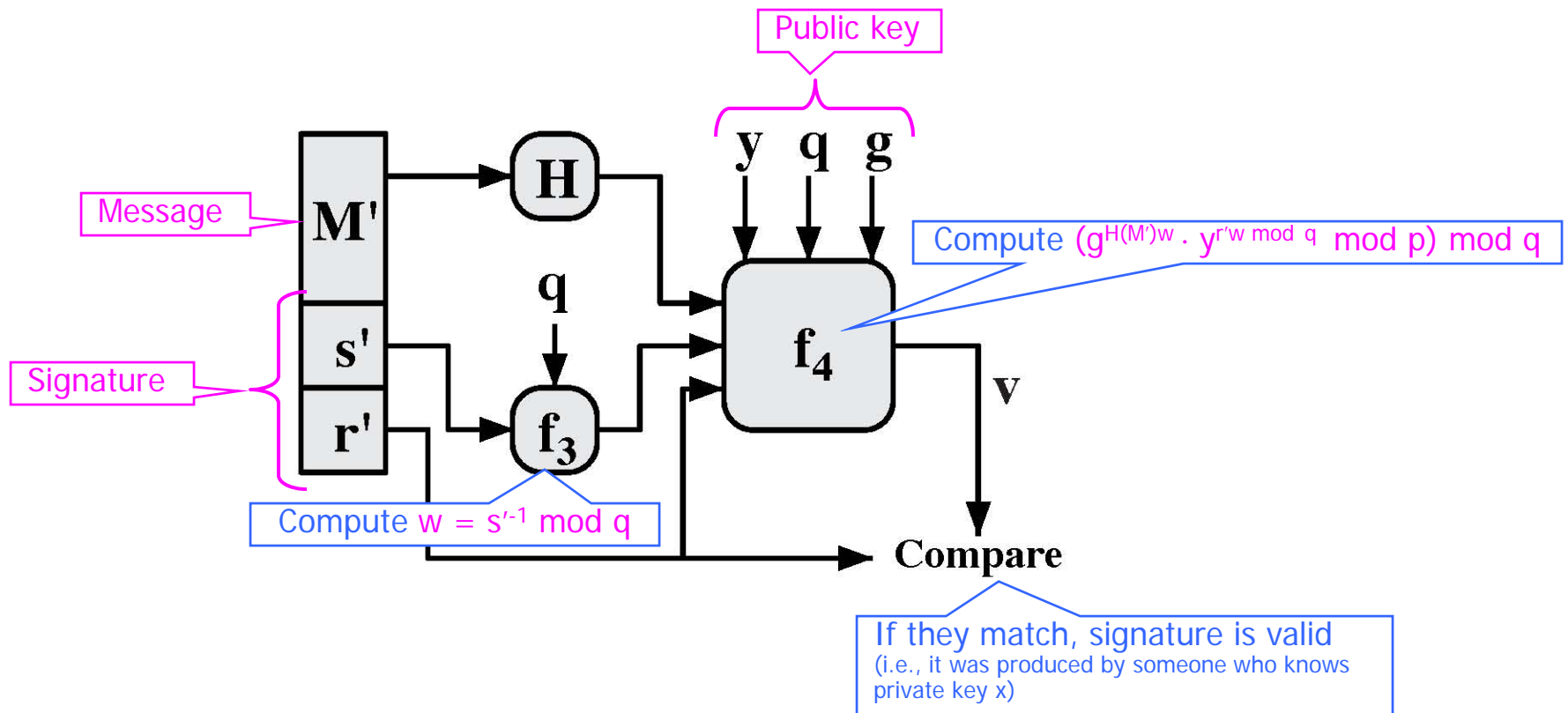
# Digital Signature Standard (DSS)

- ◆ 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  $h \in \mathbb{Z}_p^*$  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



# DSS: Verifying a Signature



# Why DSS Verification Works

---

- ◆ If  $(r,s)$  is a legitimate signature, then
$$r = (g^k \bmod p) \bmod q ; \quad s = k^{-1} \cdot (H(M) + x \cdot r) \bmod q$$
- ◆ Thus  $H(M) = -x \cdot r + k \cdot s \bmod q$ 
  - Multiply both sides by  $w = s^{-1} \bmod q$
- ◆  $H(M) \cdot w + x \cdot r \cdot w = k \bmod q$ 
  - Exponentiate  $g$  to both sides
- ◆  $(g^{H(M) \cdot w + x \cdot r \cdot w} = g^k) \bmod p \bmod q$ 
  - In a valid signature,  $g^k \bmod p \bmod q = r$ ,  $g^x \bmod p = y$
- ◆ Verify  $g^{H(M) \cdot w} \cdot y^{r \cdot w} = r \bmod p \bmod q$

# Security of DSS

---

- ◆ Can't create a valid signature without private key
- ◆ Given a signature, hard to recover private key
- ◆ Can't change or tamper with signed message
- ◆ If the same message is signed twice, signatures are different
  - Each signature is based in part on random secret  $k$
- ◆ Secret  $k$  must be different for each signature!
  - If  $k$  is leaked or if two messages re-use the same  $k$ , attacker can recover secret key  $x$  and forge any signature from then on

# 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
- ◆ 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 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
    - We'll see this in IPsec and SSL
- ◆ 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