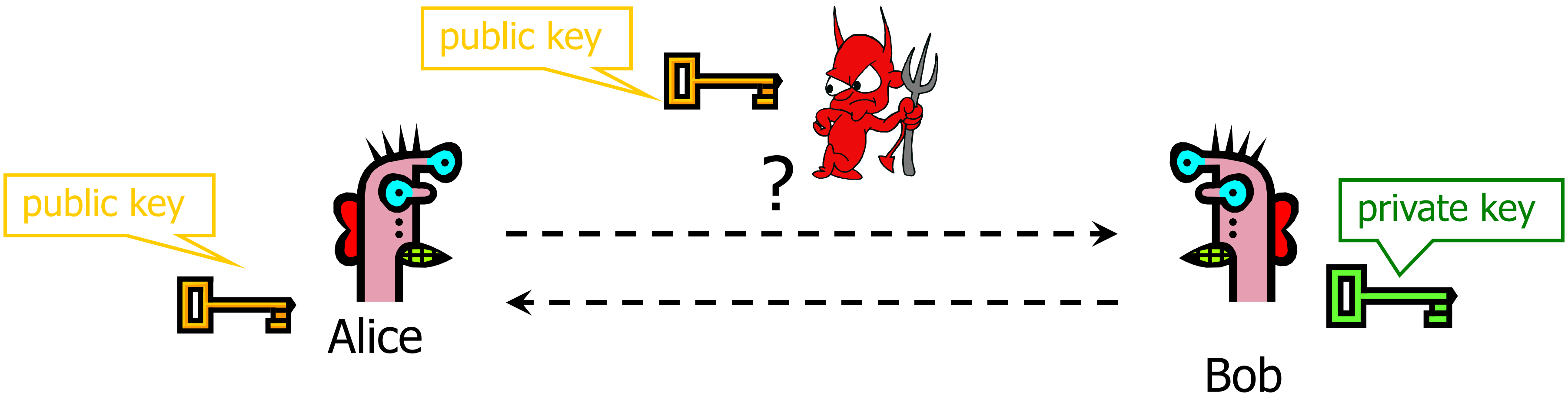
Overview of Public-Key Cryptography

Vitaly Shmatikov

Reading Assignment

- Kaufman 6.1-6

Public-Key Cryptography



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 message that only Bob can read

2. Bob wants to send a message that only Bob could have written

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know the secret key to encrypt
  - Only someone who knows the private key can decrypt
  - Secret keys are only stored in one place
- Digital signatures for authentication
  - Only someone who knows the private key can sign
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

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

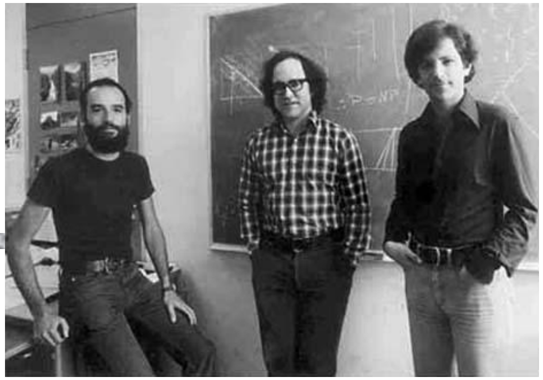
Some Number Theory Facts

- Euler totient function $\varphi(n)$ 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 $a^{\varphi(n)} \equiv 1 \bmod n$
- Special case: <u>Fermat's Little Theorem</u>

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

RSA Cryptosystem



[Rivest, Shamir, Adleman 1977]

- **Key generation:**
  - Generate large primes p, q
    - At least 2048 bits each… need primality testing!
  - Compute n=pq
    - Note that $\varphi(n)=(p-1)(q-1)$
  - 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 $ed \equiv 1 \bmod \varphi(n)$
  - Public key = (e,n); private key = d
- **Encryption** of m: $c = m^e \bmod n$
- **Decryption** of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why RSA Decryption Works

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

  $m^{ed} \equiv m \bmod n$

Why Is RSA Secure?

- RSA problem: given c, $n=pq$, and e such that $\gcd(e,(p-1)(q-1))=1$, 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, but may be possible to break RSA without factoring n

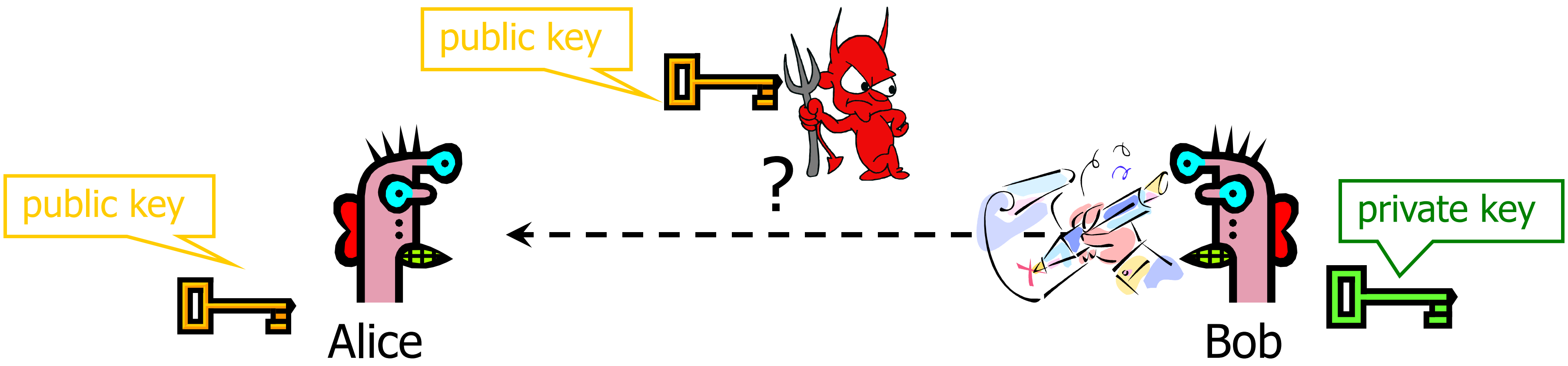
“Textbook” RSA Is Bad Encryption

- Deterministic
  - Attacker can guess plaintext, compute ciphertext, and compare for equality
  - If messages are from a small set (for example, yes/no), can build a table of corresponding ciphertexts
- Can tamper with encrypted messages
  - Take an encrypted auction bid c and submit $c(101/100)^e \bmod n$ instead
- Does not provide semantic security (security against chosen-plaintext attacks)

Integrity in RSA Encryption

- "Textbook" RSA does not provide integrity
  - Given encryptions of m_1 and m_2, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \bmod n \equiv (m_1 \cdot m_2)^e \bmod n$
  - Attacker can convert m into m^k without decrypting
    - $(m^e)^k \bmod n \equiv (m^k)^e \bmod n$
- 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, only the public key is needed

RSA Signatures

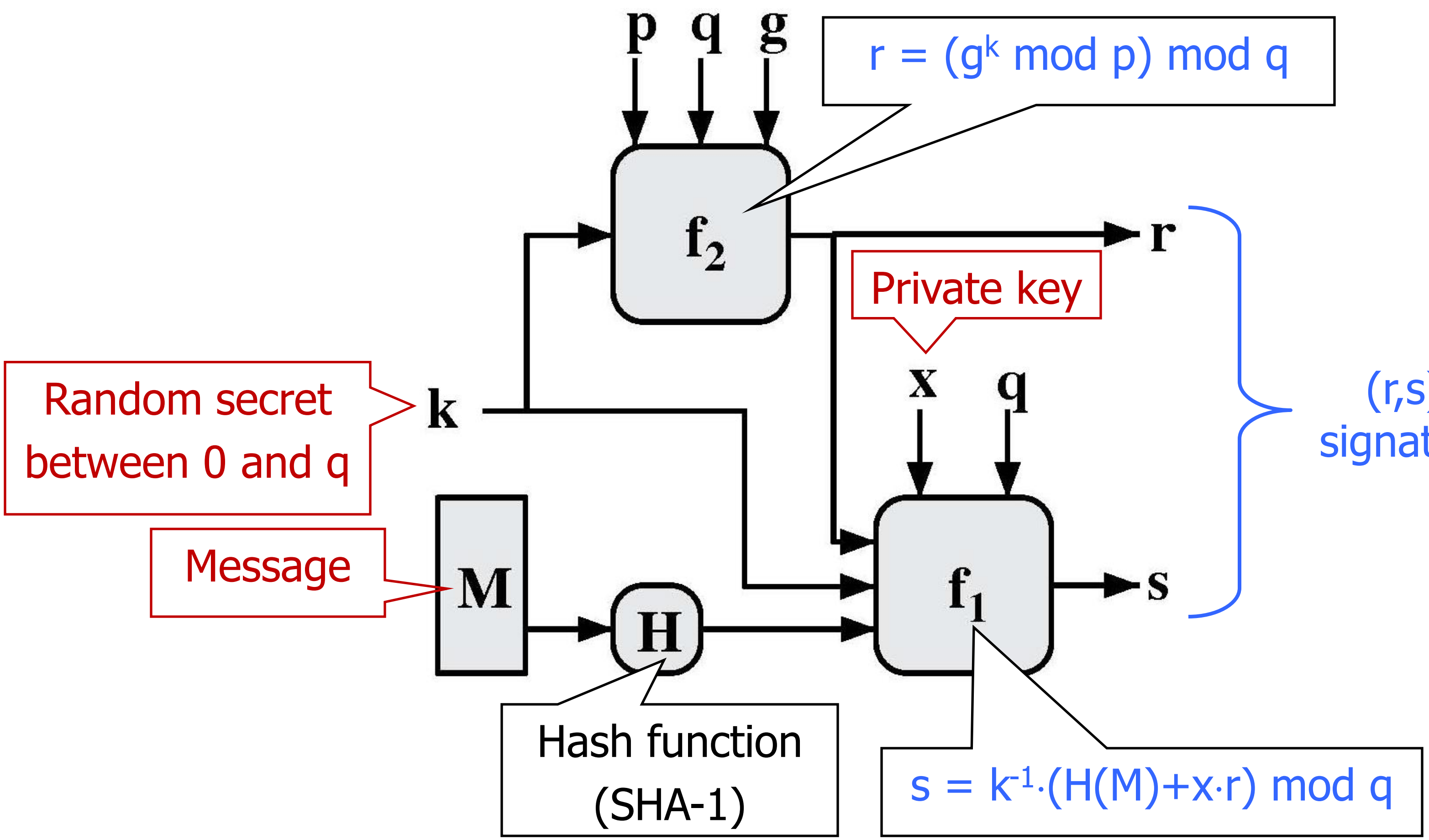
- Public key is (n,e), private key is d
- To sign message m: $s = hash(m)^d \bmod n$
  - Signing and decryption are the same mathematical operation in RSA
- To verify signature s on message m:

  $s^e \bmod n = (hash(m)^d)^e \bmod n = hash(m)$
  - Verification and encryption are the same mathematical operation in RSA
- Message must be hashed and padded (why?)

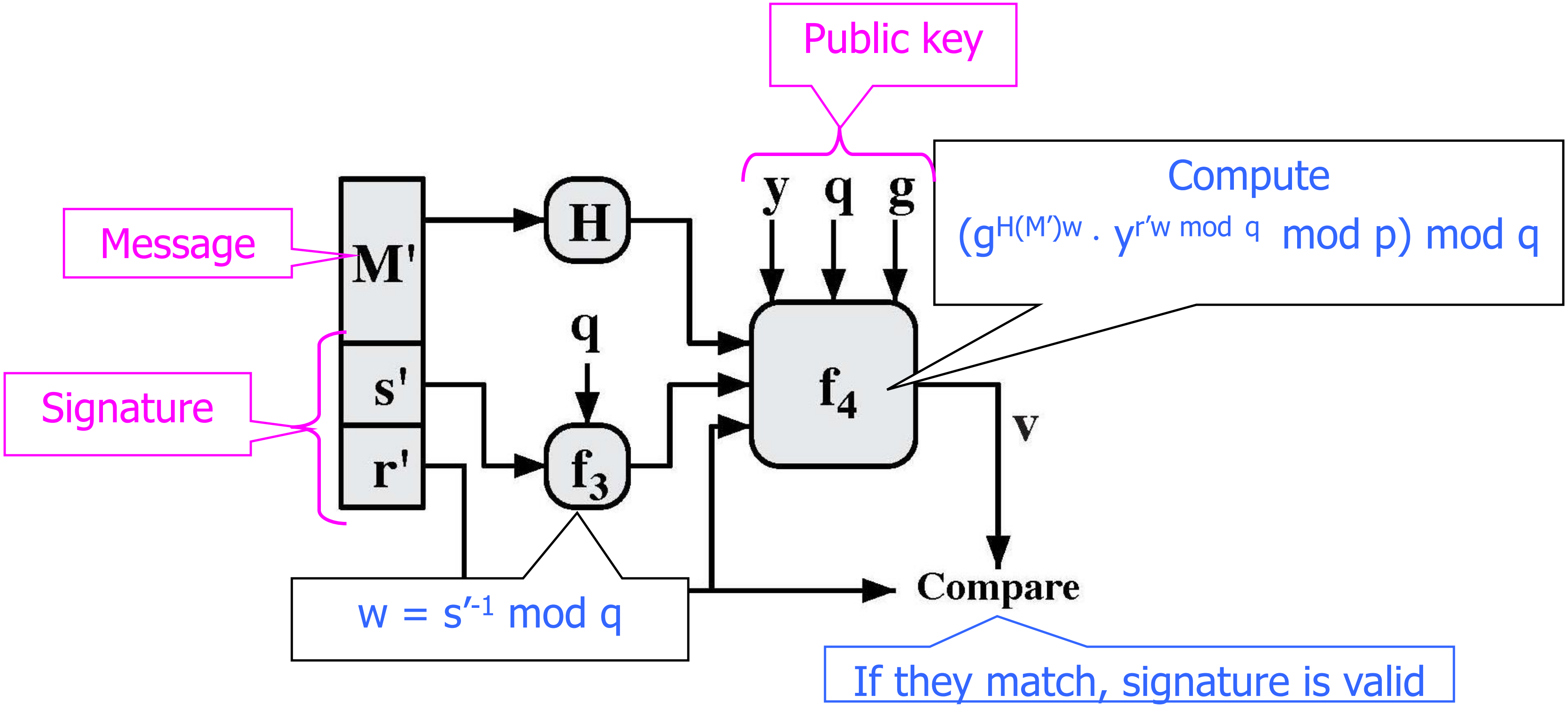
Digital Signature Algorithm (DSA)

- 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: $(p, q, g, g^x \bmod p)$, private key: x
- Security of DSA requires hardness of discrete log
  - If one can take discrete logarithms, then can extract x (private key) from $g^x \bmod p$ (public key)

DSA: Signing a Message



DSA: Verifying a Signature



Why DSA Verification Works

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

Security of DSA

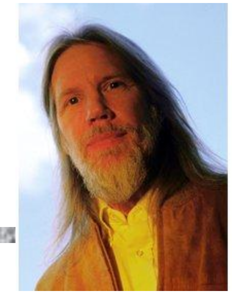
- Can't create a valid signature without private key
- Can't change or tamper with signed message
- If the same message is signed twice, signatures are different
  - Each signature is based in part on random secret k
- Secret k must be different for each signature!
  - If k is leaked or if two messages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

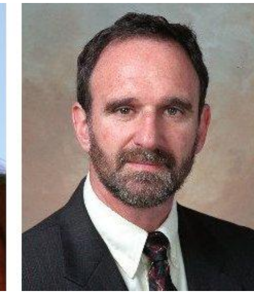
PS3 Epic Fail

- Sony uses ECDSA algorithm to sign authorized software for Playstation 3
  - Basically, DSA based on elliptic curves
    … with the same random value in every signature
- Trivial to extract master signing key and sign any homebrew software – perfect "jailbreak" for PS3
- Announced by George "Geohot" Hotz and Fail0verflow team in Dec 2010

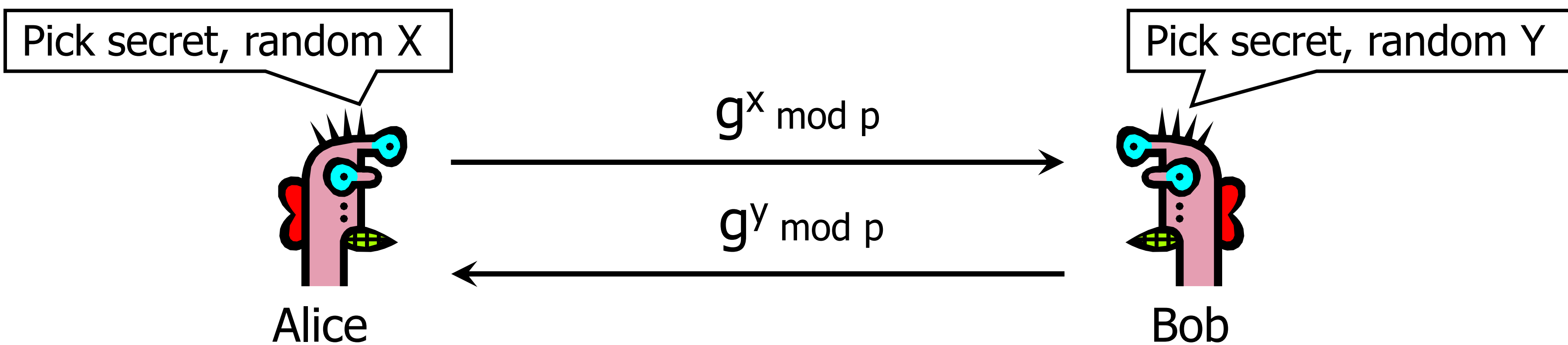
Q: Why didn't Sony just revoke the key?

Diffie-Hellman Protocol





- 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 $a = g^i \bmod p$



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

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

Why Is Diffie-Hellman Secure?

- **Discrete Logarithm (DL)** problem: given g^x mod 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} mod 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} mod p and g^r mod 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 the established key and a random value
  - Can use the new key for symmetric cryptography
- Basic Diffie-Hellman protocol does not provide authentication
  - IPsec combines Diffie-Hellman with signatures, anti-DoS cookies, etc.

Advantages of Public-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - Can use this for key establishment, avoiding the "chicken-or-egg" problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
- Authentication without shared secrets
- Encryption keys are public, but must be sure that Alice's public key is really <u>her</u> public key
  - This is a hard problem… Often solved using public-key certificates

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
    - SSL, IPsec, most other systems based on public crypto
- Keys are longer
  - 2048 bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - Factoring, RSA problem, discrete logarithm problem, decisional Diffie-Hellman problem…