CSE 484 and CSE M 584 (Winter 2009)

Networks (missed material)
Public key cryptography

Tadayoshi Kohno

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

Intrusion Detection

Intrusion Detection Systems

- Advantage: can recognize new attacks and new versions of old attacks
- Disadvantages
  - High false positive rate
  - Must be trained on known good data
    - Training is hard because network traffic is very diverse
  - Protocols are finite-state machines, but current state of a connection is difficult to see from the network
  - Definition of “normal” constantly evolves
    - What’s the difference between a flash crowd and a denial of service attack?

Intrusion Detection Problems

- Lack of training data with real attacks
  - But lots of "normal" network traffic, system call data
- Data drift
  - Statistical methods detect changes in behavior
  - Attacker can attack gradually and incrementally
- Main characteristics not well understood
  - By many measures, attack may be within bounds of "normal" range of activities
- False identifications are very costly
  - Sysadm will spend many hours examining evidence

Intrusion Detection Errors

- False negatives: attack is not detected
  - Big problem in signature-based misuse detection
- False positives: harmless behavior is classified as an attack
  - Big problem in statistical anomaly detection
- Both types of IDS suffer from both error types
- Which is a bigger problem?
  - Attacks are fairly rare events

Conditional Probability

- Suppose two events A and B occur with probability Pr(A) and Pr(B), respectively
- Let Pr(AB) be probability that both A and B occur
- What is the conditional probability that A occurs assuming B has occurred?

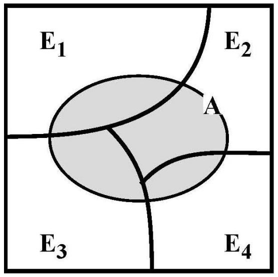
$$\Pr(A \mid B) = \frac{\Pr(AB)}{\Pr(B)}$$

Bayes' Theorem

- Suppose mutually exclusive events $E_1, \ldots, E_n$ together cover the entire set of possibilities
- Then probability of any event A occurring is

$$\Pr(A) = \Sigma_{1 \le i \le n} \Pr(A \mid E_i) \bullet \Pr(E_i)$$

  - Intuition: since $E_1, \ldots, E_n$ cover entire probability space, whenever A occurs, some event E_i must have occurred



- Can rewrite this formula as

$$\Pr(E_i \mid A) = \frac{\Pr(A \mid E_i) \bullet \Pr(E_i)}{\Pr(A)}$$

Base-Rate Fallacy

- 1% of traffic is SYN floods; IDS accuracy is 90%
  - IDS classifies a SYN flood as attack with prob. 90%, classifies a valid connection as attack with prob. 10%
- What is the probability that a connection flagged by IDS as a SYN flood is actually valid traffic?

$$\Pr(\text{valid} \mid \text{alarm}) = \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm})}$$

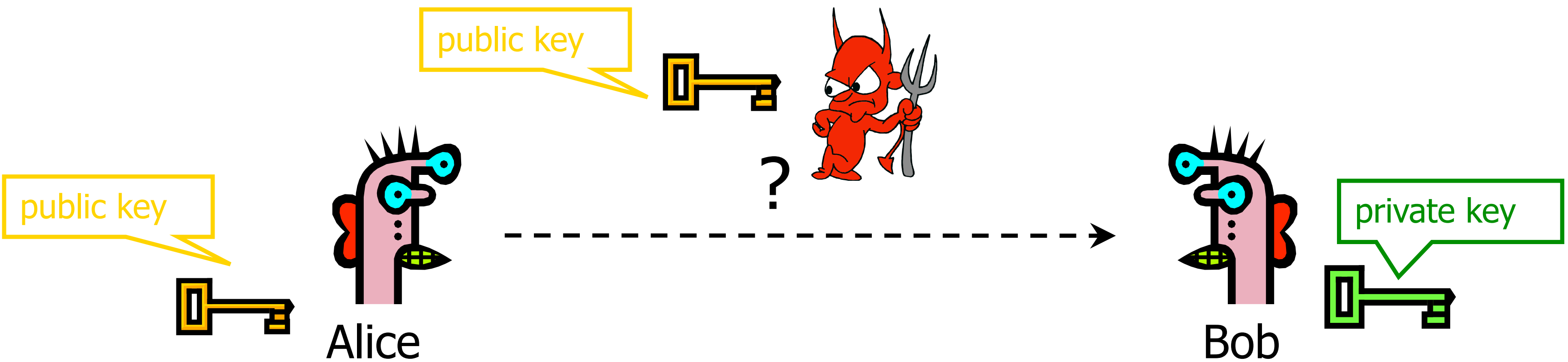
$$= \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid}) + \Pr(\text{alarm} \mid \text{SYN flood}) \cdot \Pr(\text{SYN flood})}$$

$$= \frac{0.10 \cdot 0.99}{0.10 \cdot 0.99 + 0.90 \cdot 0.01}$$

= 92% chance raised alarm is false!!!

Public Key Cryptography

Basic Problem



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

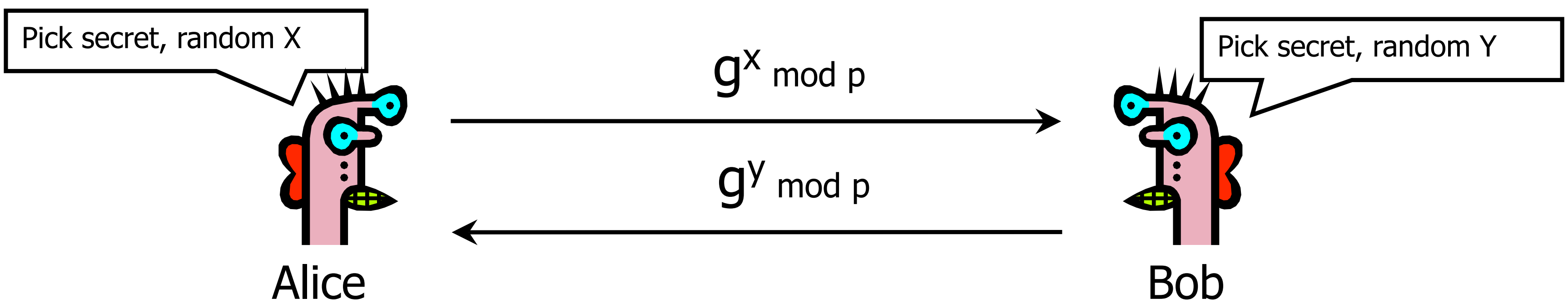
Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

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^*
    - $Z_p^*=\{1, 2 \ldots p-1\}$; $\forall a \in Z_p^*\ \exists i$ 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 $k=(g^x)^y=g^{xy} \bmod p$

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:**

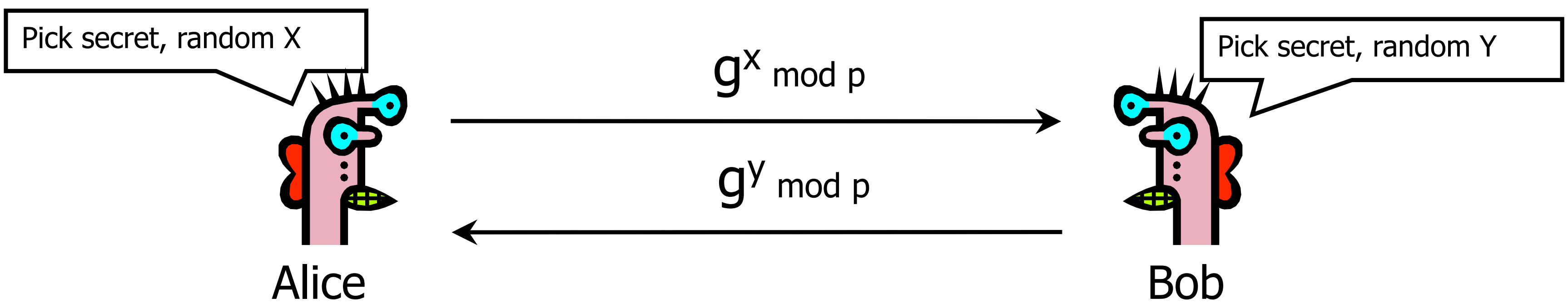
  given g^x and g^y, it's hard to tell the difference between g^{xy} mod p and g^r mod p where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> 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 (by itself) does not provide authentication
- DDH: not true for integers mod p, but true for other groups

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 $a=g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach p



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

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

Requirements for Public-Key Crypto

- **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 $a^{\varphi(n)} = 1 \bmod n$
- Special case: Fermat's Little Theorem

  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 $e \cdot d = 1 + k \cdot \varphi(n) = 1 + k(p-1)(q-1)$ for some k

- 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 $p \cdot q = n$,

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

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, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring n

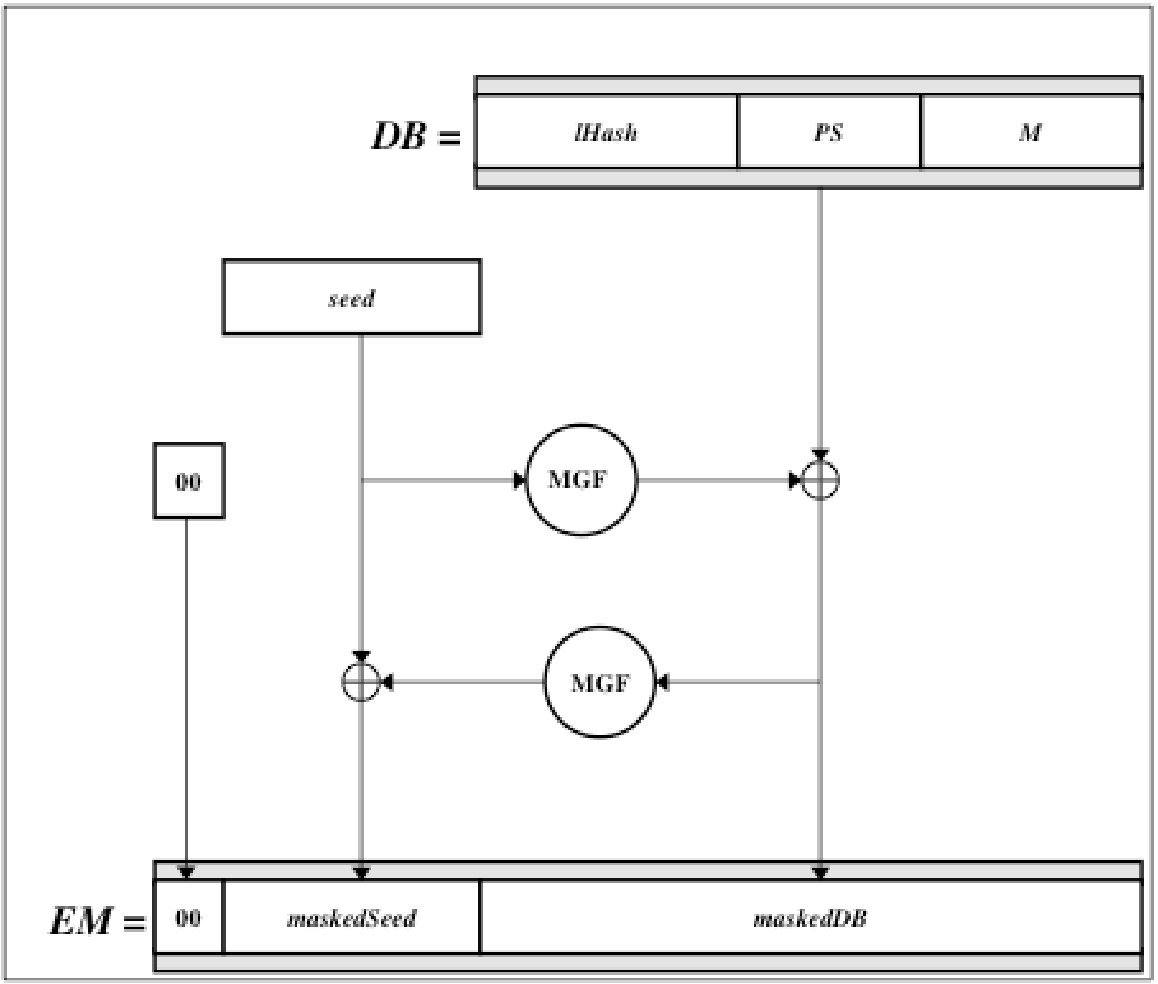
Caveats

- 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
    - 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!

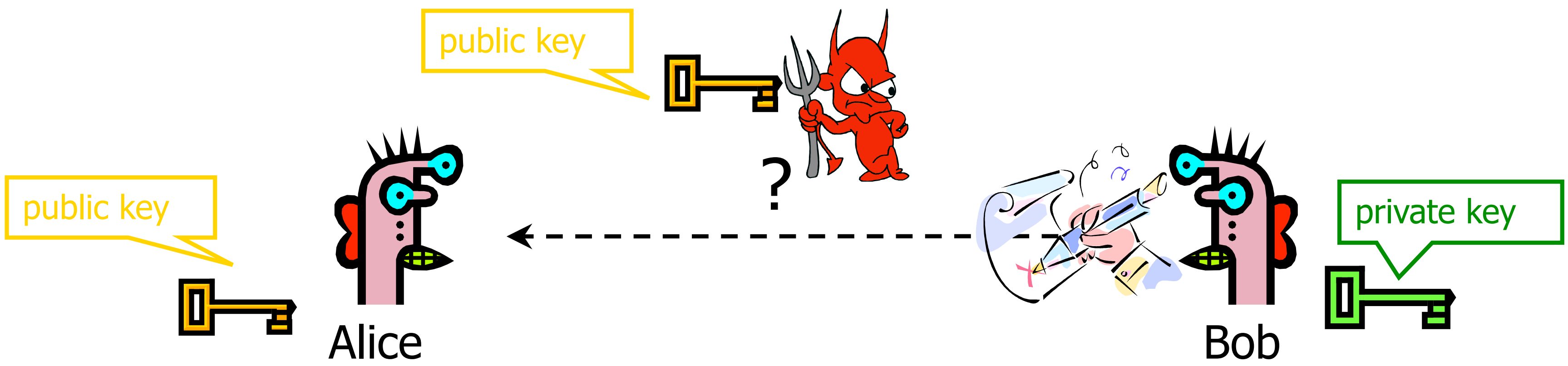
Integrity in RSA Encryption

- 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)



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

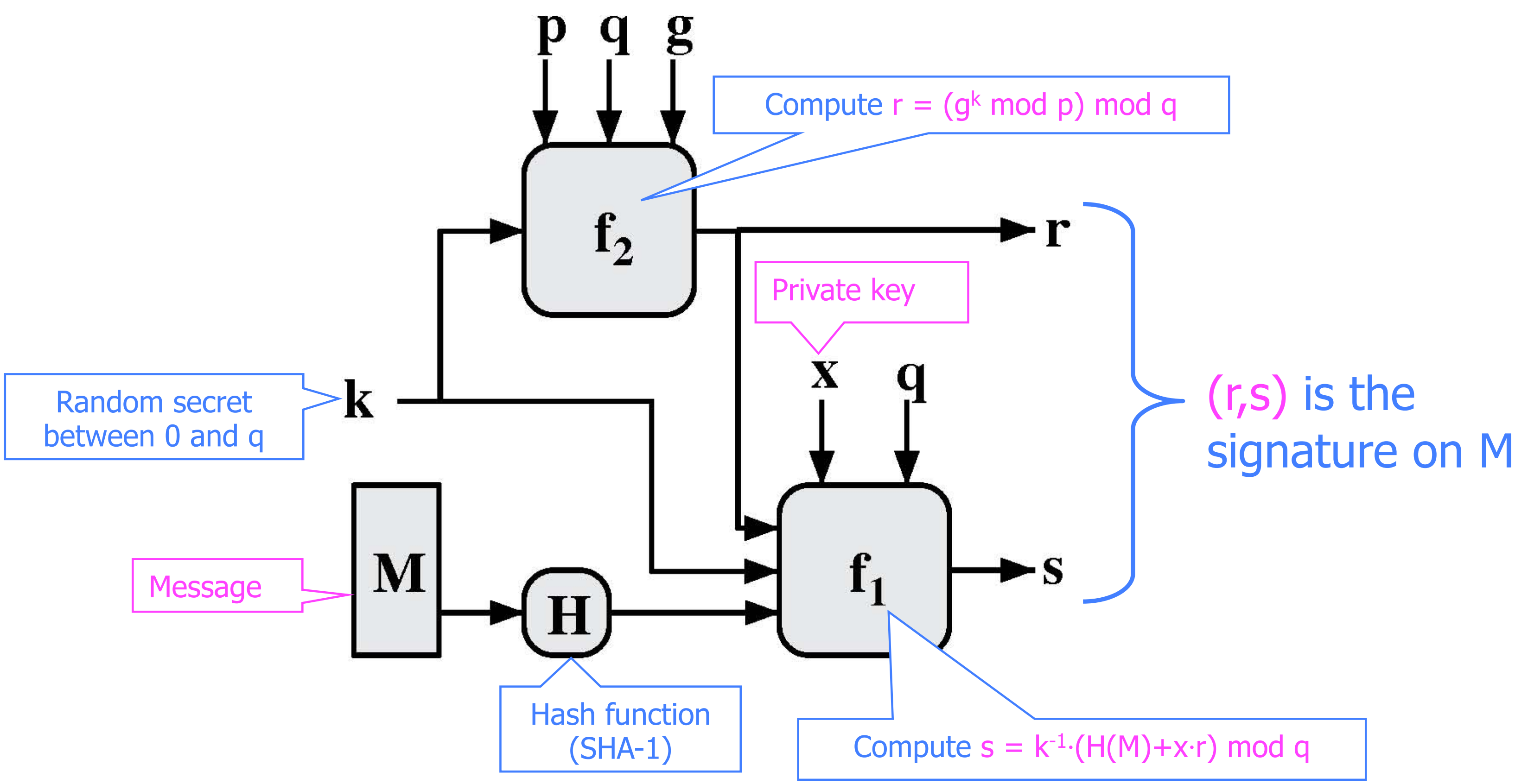
Encryption and Signatures

- 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

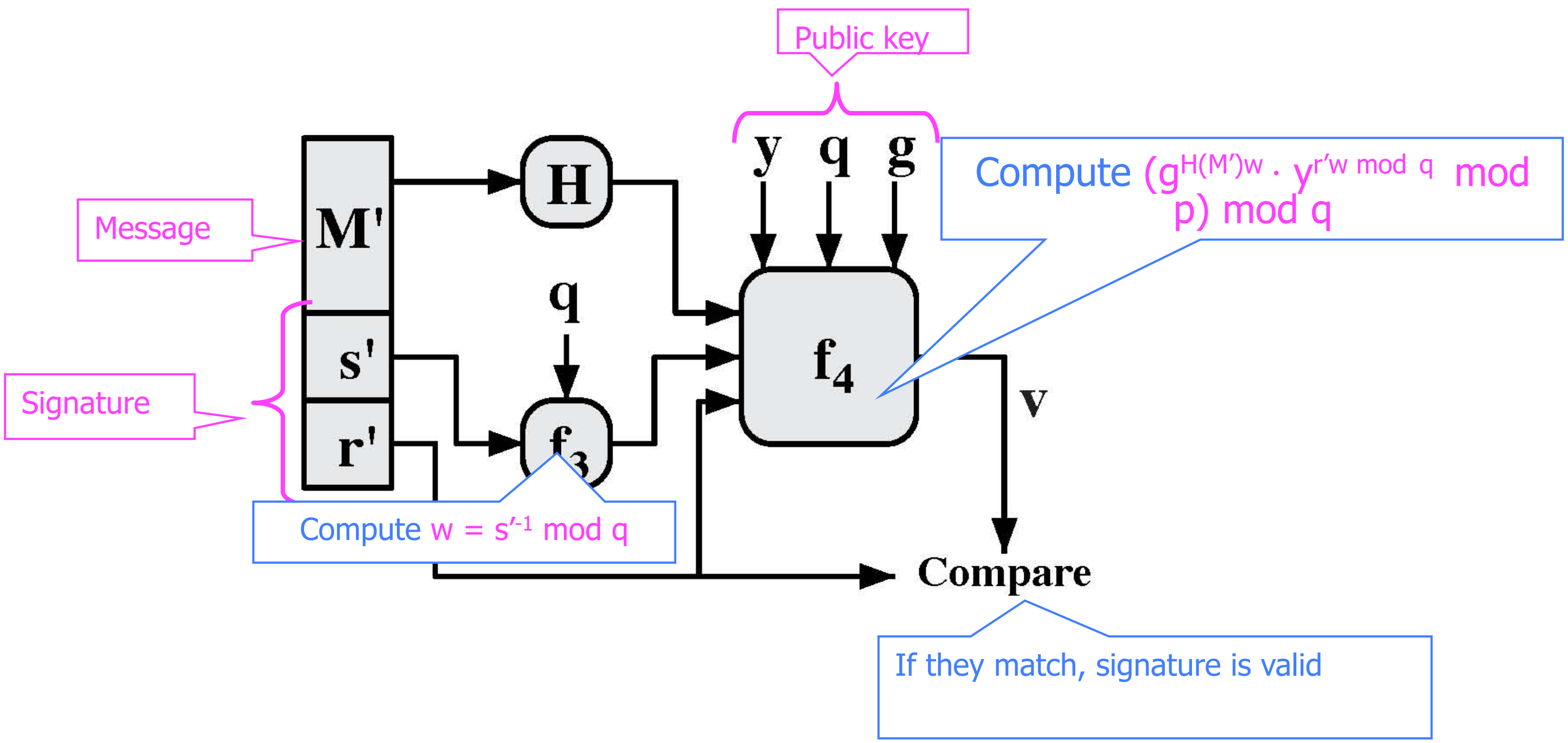
Digital Signature Standard (DSS)

- 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 $0 \le t \le 8$
  - Select $h \in Z_p^*$ and compute $g = h^{(p-1)/q} \bmod p$
  - Select random x such $1 \le x \le q-1$, 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
  $r = (g^k \bmod p) \bmod q$; $s = k^{-1} \cdot (H(M)+x \cdot r) \bmod q$
- 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
    - 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
    - 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 <u>believed</u> to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions)

Next Homework

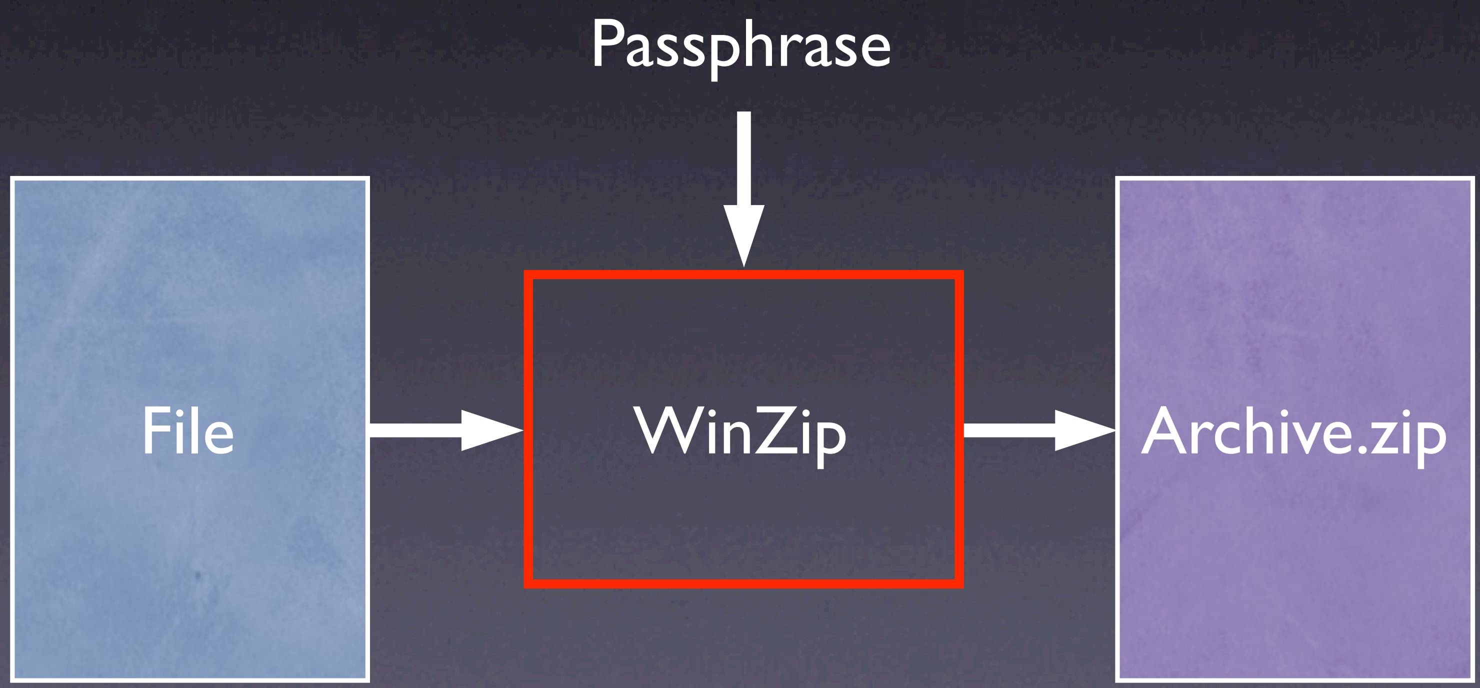
- You'll be looking at WinZip's new AE-2 encryption scheme
  - Based on "Encrypt-then-MAC" (recall a few classes ago --- this is a provably secure mode)
  - But things aren't always that simple
    - Many protocols seem secure but actually have problems
  - Your job: Analyze AE-2

What is WinZip?

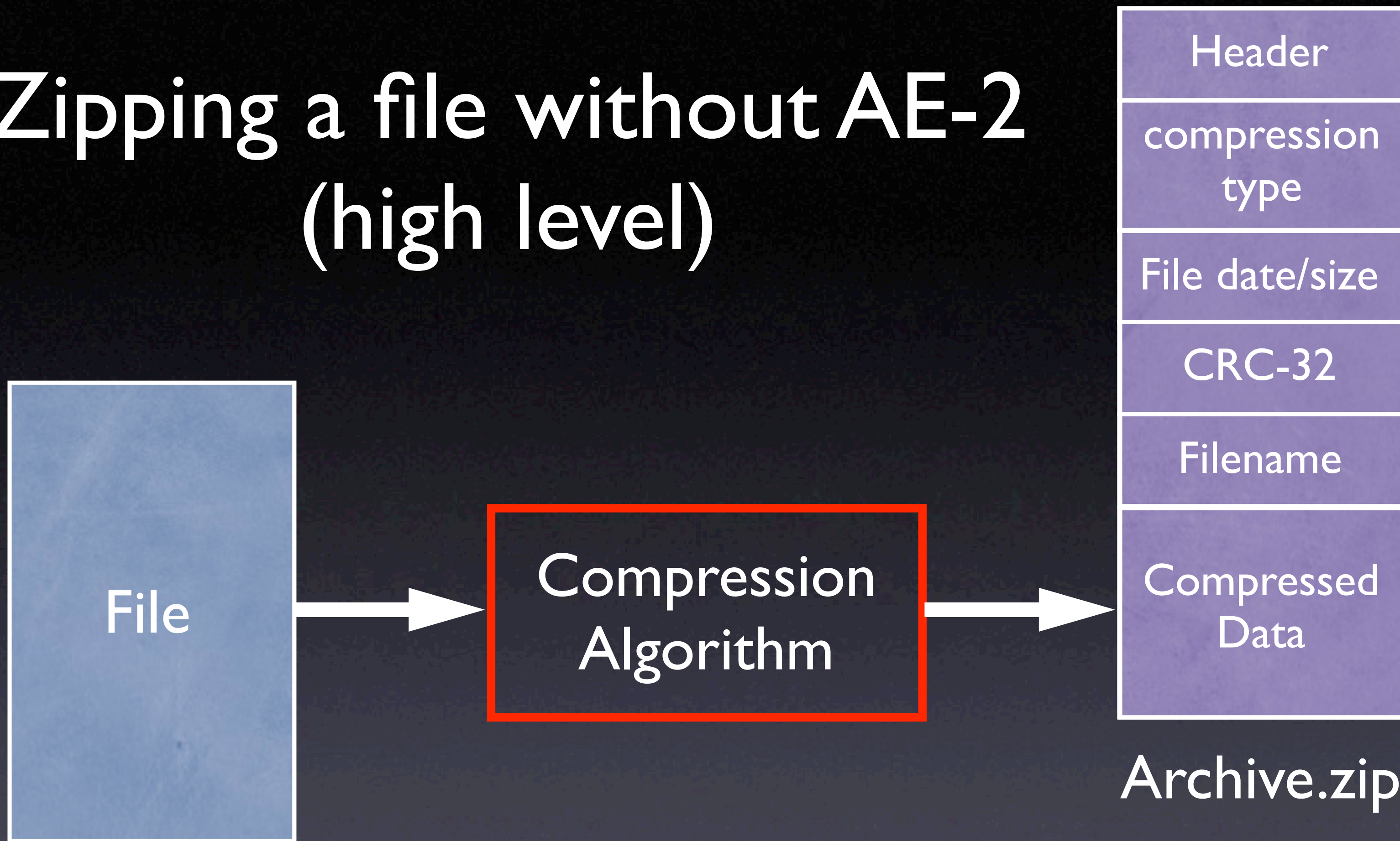
Very popular Windows compression utility. Also an Outlook email plugin. Over 160 million downloads from download.com alone [http://www.winzip.com/empopp.htm].

WinZip encryption

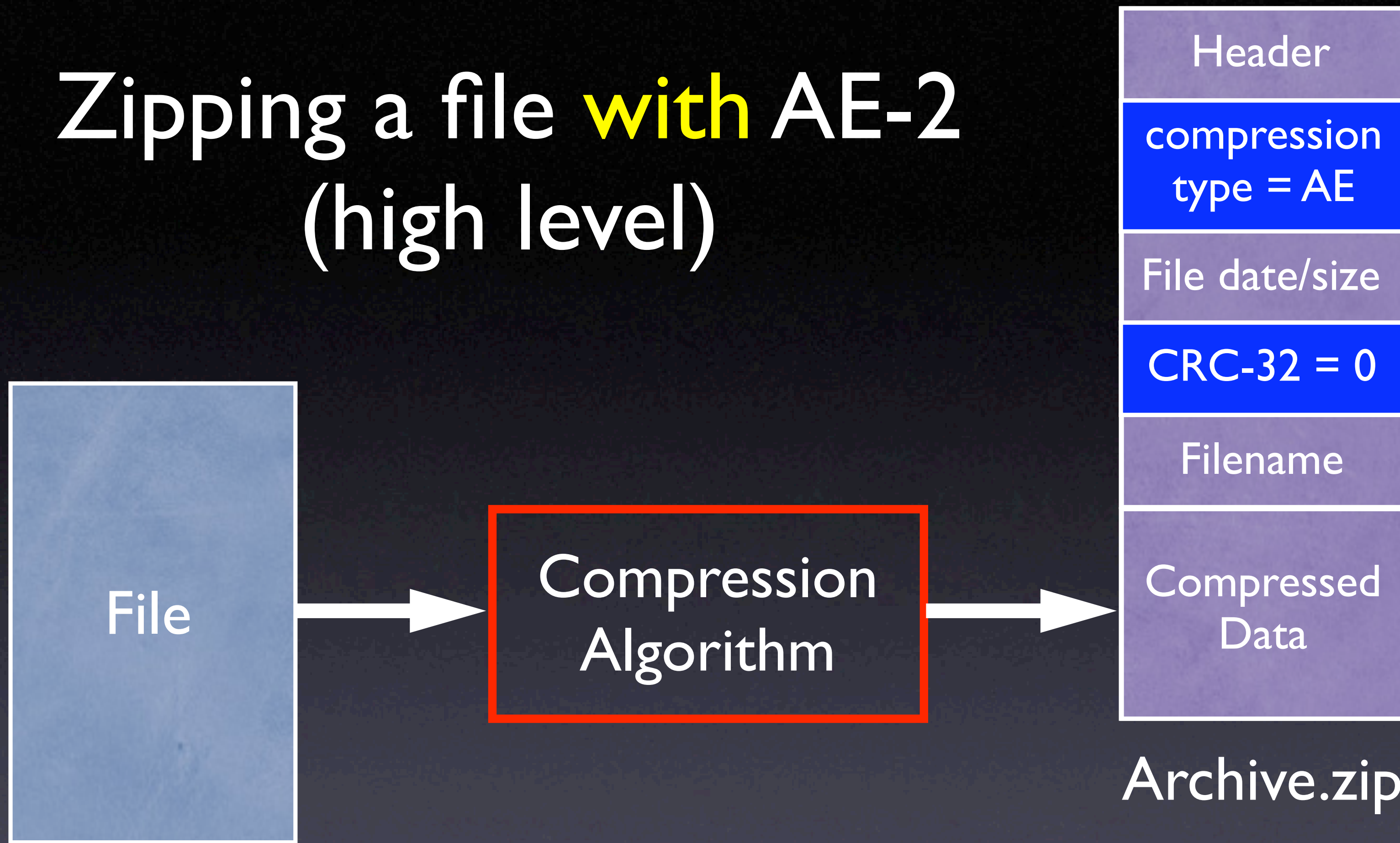
WinZip has the ability to encrypt files. Lots of history, but we'll look at the AE-2 method.



Zipping a file without AE-2 (high level)



Zipping a file with AE-2 (high level)



Zipping a file with AE-2 (high level)