CSE 484 / CSE M 584: Asymmetric Cryptography

Fall 2024

Franziska (Franzi) Roesner
franzi@cs

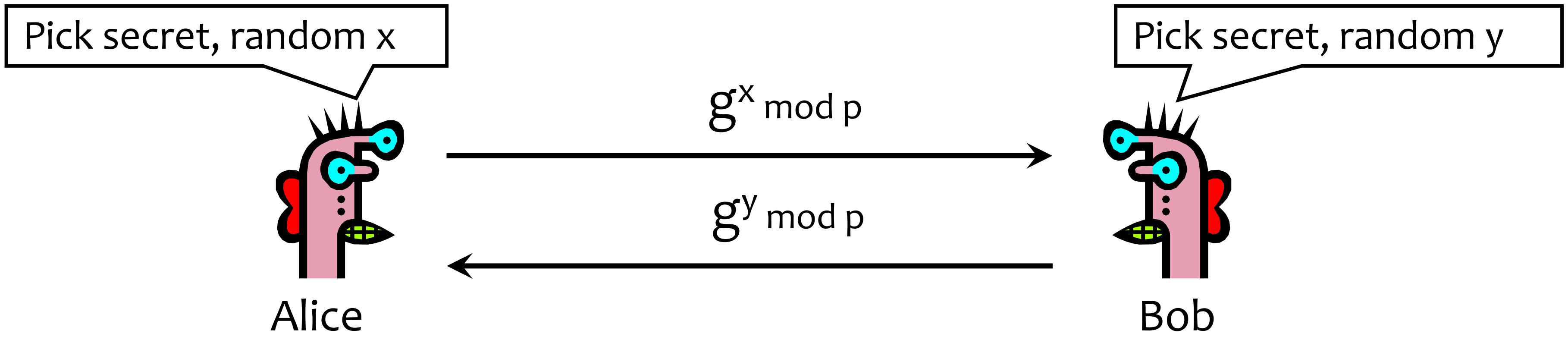
UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

- Things due
  - Lab 1: today!
  - Homework 2: Next Friday
  - CSE 584M: Don’t forget about weekly research readings
- Roadmap
  - Today: Finish asymmetric crypto
  - Friday: Crypto in practice (on the web)
  - Next week: Web security

Reminder: Diffie-Hellman Protocol (1976)

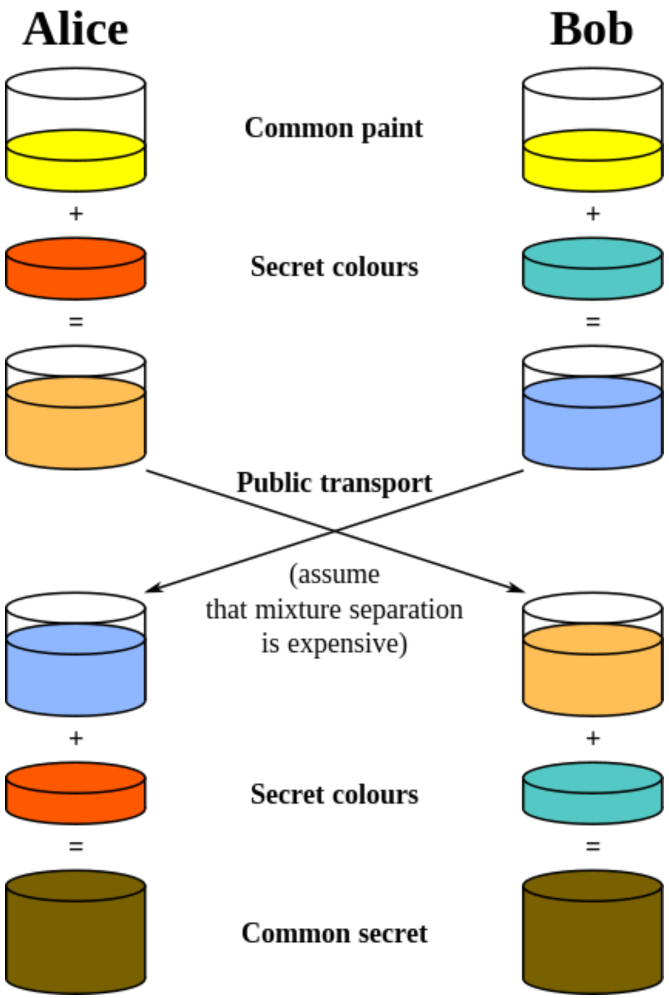
- Alice and Bob never met and share no secrets
- <u>Public</u> info: p and g
  - p is a large prime, g is a **generator** of Z_p^*
    - $Z_p^*=\{1, 2 \ldots p-1\}$; a is in Z_p^* if there is an i such that $a=g^i \bmod p$



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

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

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:
g^x mod p
g^y mod p

Common secret: g^{xy} mod p

[from Wikipedia]

Why is Diffie-Hellman Secure?

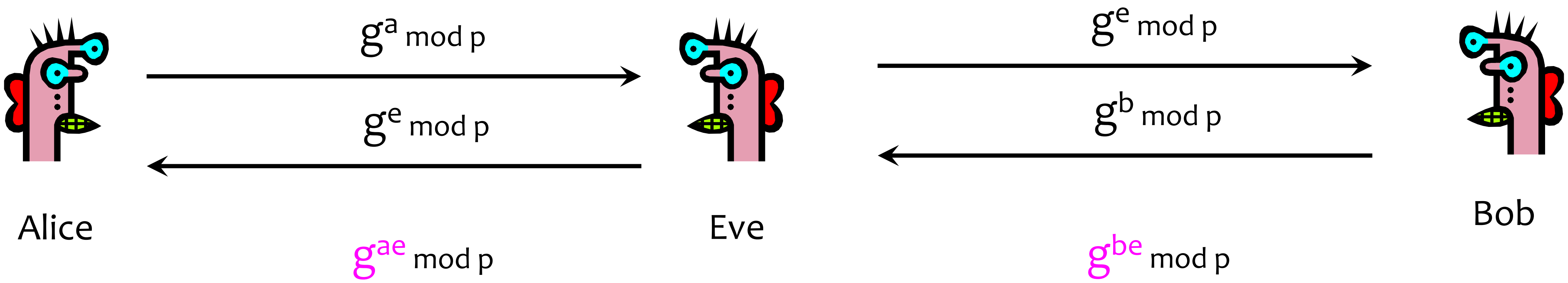
- Discrete Logarithm (DL) problem:
  given g^x *mod p*, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:
  given g^x and g^y, it's hard to compute g^{xy} *mod p*
  - … 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

Diffie-Hellman Caveats (1)

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> attackers
  - Common recommendation:
    - Choose p=2q+1, where q is also a large prime
    - Choose g that generates a subgroup of order q in Z_p*
    - DDH is hard in this group
  - Eavesdropper can't tell the difference between the established key and a random value
  - In practice, often hash $g^{xy} \bmod p$, and use the hash as the key
  - Can use the new key for symmetric cryptography

Diffie-Hellman Caveats (2)

- Diffie-Hellman protocol (by itself) does not provide authentication (against <u>active</u> attackers)
  – Person in the middle attack (aka "man in the middle attack")



Diffie-Hellman Key Exchange Today

- Important Note:
  - We have discussed discrete logs modulo integers
  - Significant advantages in using **elliptic curve groups**
    - Groups with some similar mathematical properties (i.e., are "groups") but have better security and performance (size) properties
    - Today's de-facto standard

Stepping Back: Asymmetric Crypto

- We've just seen session key establishment
  - Can then use shared key for symmetric crypto
- Next: public key encryption
  - For confidentiality
- Then: digital signatures
  - For authenticity

Requirements for 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 φ(n) (n≥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
    - Example: **34** (factors: 1, 2, 17, 34) and **35** (1, 5, 7, 35)
  - Easy to compute for primes: φ(p) = p-1
  - Note that φ(ab) = φ(a) φ(b) if a & b are relatively prime

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 2048 bits each (need primality testing, too)
  - Compute **n**=pq and φ(**n**)=(p-1)(q-1)
  - Choose small **e**, relatively prime to φ(n)
    - Typically, **e=3** or **e=2^{16}+1=65537**
  - Compute unique **d** such that ed ≡ 1 mod φ(n)
    - Modular inverse: d ≡ e^{-1} mod φ(n)
  - Public key = (e,n); private key = (d,n)
- Encryption of m: c = m^e mod n
- Decryption of c: c^d mod n = $(m^e)^d$ mod n = m

A 512-bit prime:
```
9238392041438079003450838646
6608111904604104720064333111
8274222861101608716534554412
4307595017420038487576191853
6796640686377031035140080035
82827766514729
```

How to compute?

- Extended Euclidian algorithm
- Wolfram Alpha ☺
- Brute force for small values

Why is RSA Secure?

- RSA problem: given c, n=pq, and e such that gcd(e, φ(n))=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 *without* knowing p and q
- 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
(knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n – but if it is, we don't know how

RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow.
- Plain RSA also does <u>not</u> provide integrity
  – Can tamper with encrypted messages

In practice, OAEP is used: instead of encrypting M, encrypt

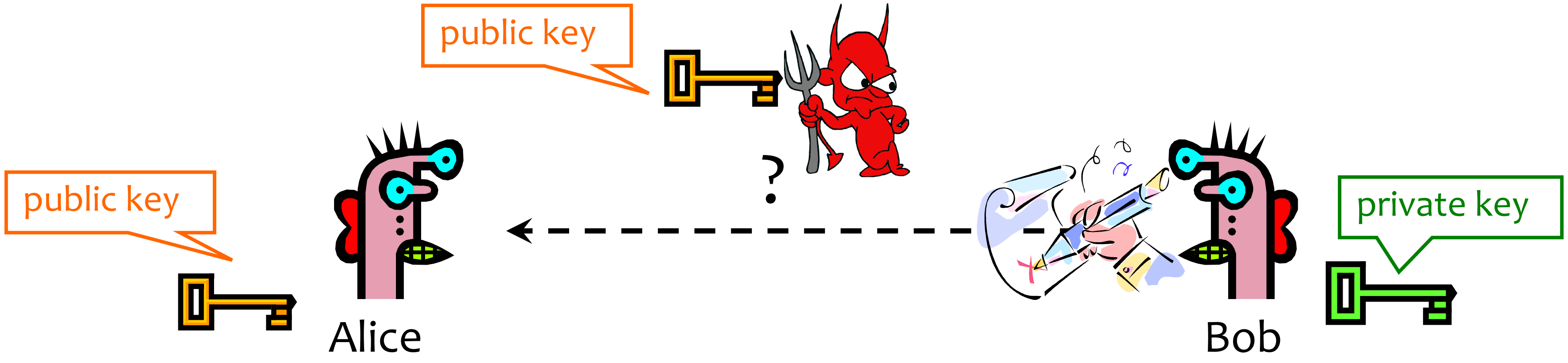
$M \oplus G(r) \,||\, r \oplus H(M \oplus G(r))$

– r is random and fresh, G and H are hash functions

Stepping Back: Asymmetric Crypto

- Last time we saw session key establishment (Diffie-Hellman)
  - Can then use shared key for symmetric crypto
- We just saw: public key encryption
  - For confidentiality
- Finally, now: digital signatures
  - For authenticity

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 **(n,d)**
- To sign message m: $s = m^d \bmod n$
  - Signing & decryption are 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 (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Without padding and hashing: Consider multiplying two signatures together
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- 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 mod p (public key)
- Again: We've discussed discrete logs modulo integers; significant advantages to using elliptic curve groups instead.

Post-Quantum Cryptography

- If quantum computers become a reality
  - It becomes much more efficient to break conventional asymmetric encryption schemes (e.g., factoring becomes “easy”)
- There exists efforts to make quantum-resilient asymmetric encryption schemes

Cryptography Summary

- Goal: Privacy
  - Symmetric keys:
    - One-time pad, Stream ciphers
    - Block ciphers (e.g., DES, AES) → modes: ECB, CBC, CTR
  - Public key crypto (e.g., Diffie-Hellman, RSA)
- Goal: Integrity
  - MACs, often using hash functions (e.g, SHA-256)
- Goal: Privacy and Integrity ("authenticated encryption")
  - Encrypt-then-MAC
- Goal: Authenticity (and Integrity)
  - Digital signatures (e.g., RSA, DSS)