CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography:
Hash Functions, MACs (finish)
Asymmetric Cryptography (start)

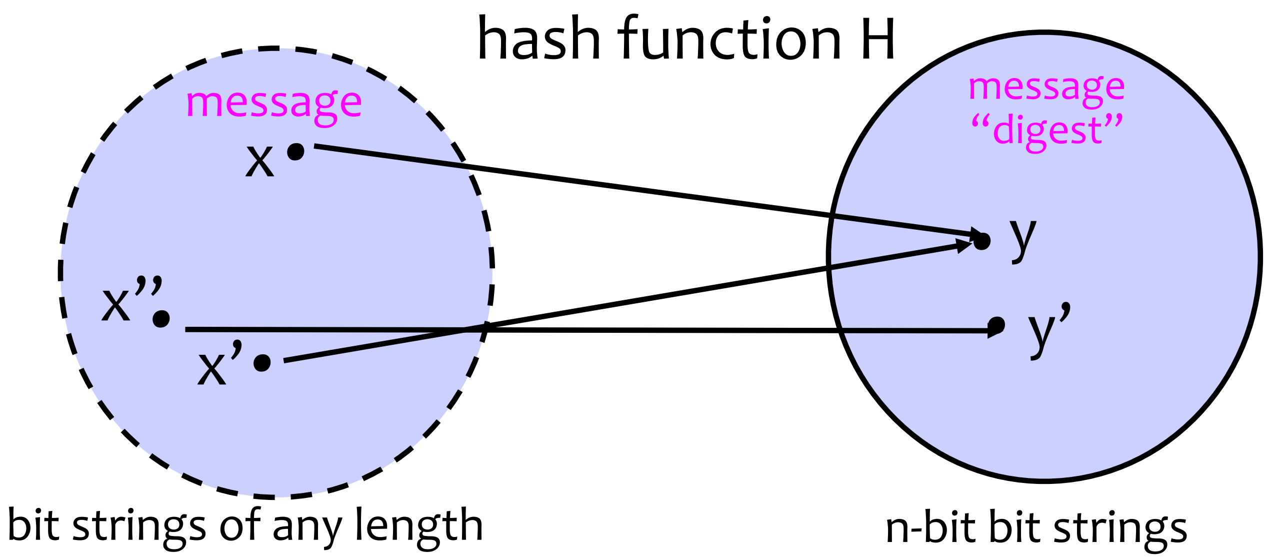
Spring 2016

Franziska (Franzi) Roesner
franzi@cs.washington.edu

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

Hash Functions

Hash Functions: Main Idea



- Hash function H is a lossy compression function
  - Collision: h(x)=h(x') for distinct inputs x, x'
- H(x) should look "random"
  - Every bit (almost) equally likely to be 0 or 1
- <u>Cryptographic</u> hash function needs a few properties…

Property 1: One-Way

- Intuition: hash should be hard to invert
  - "Preimage resistance"
  - Let $h(x') = y \in \{0,1\}^n$ for a random x'
  - Given y, it should be hard to find any x such that h(x)=y
- How hard?
  - Brute-force: try every possible x, see if h(x)=y
  - SHA-1 (common hash function) has 160-bit output
    - Expect to try 2^{159} inputs before finding one that hashes to y.

Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x)=h(x')$
- Birthday paradox means that brute-force collision search is only $O(2^{n/2})$, *not* $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$

One-Way vs. Collision Resistance

- One-wayness does <u>not</u> imply collision resistance
  - Suppose g is one-way
  - Define h(x) as g(x’) where x’ is x except the last bit
    - h is one-way (to invert h, must invert g)
    - Collisions for h are easy to find: for any x, h(x0)=h(x1)
- Collision resistance does <u>not</u> imply one-wayness
  - Suppose g is collision-resistant
  - Define y=h(x) to be 0x if x is n-bit long, 1g(x) otherwise
    - Collisions for h are hard to find: if y starts with 0, then there are no collisions, if y starts with 1, then must find collisions in g
    - h is not one way: half of all y’s (those whose first bit is 0) are easy to invert (how?); random y is invertible with probab. ½

Property 3: Weak Collision Resistance

- Given randomly chosen x, hard to find x' such that h(x)=h(x')
  - Attacker must find collision for a specific x. By contrast, to break collision resistance it is enough to find any collision.
  - Brute-force attack requires $O(2^n)$ time
- Weak collision resistance does not imply collision resistance.

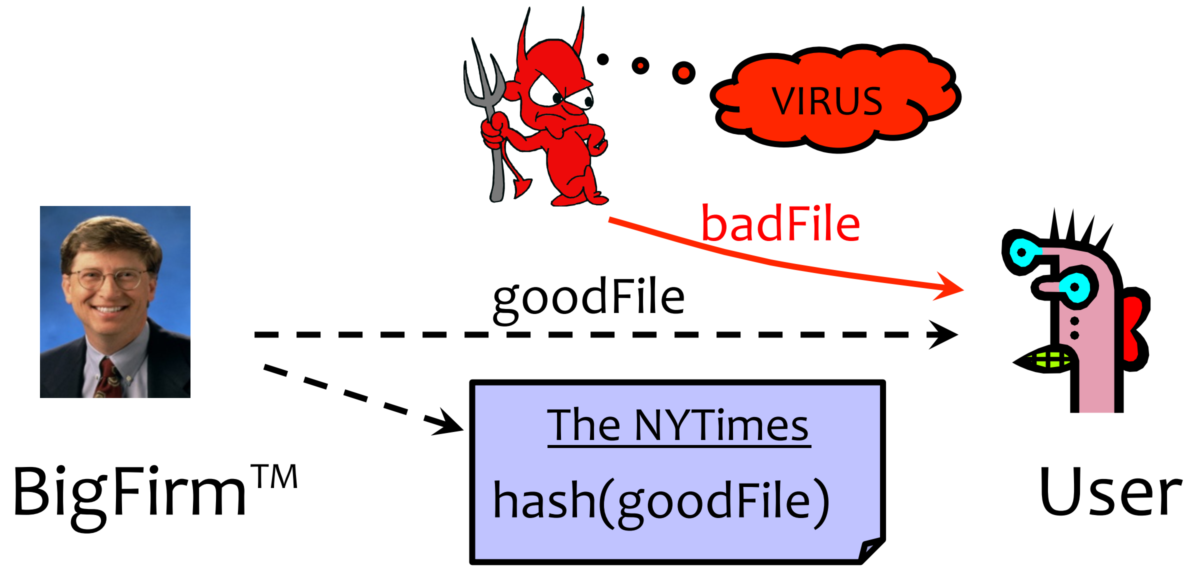
Hashing vs. Encryption

- Hashing is one-way. There is no "un-hashing"
  - A ciphertext can be decrypted with a decryption key… hashes have no equivalent of "decryption"
- Hash(x) looks "random" but can be compared for equality with Hash(x')
  - Hash the same input twice → same hash value
  - Encrypt the same input twice → different ciphertexts
- Crytographic hashes are also known as "cryptographic checksums" or "message digests"

Application: Password Hashing

- Instead of user password, store hash(password)
- When user enters a password, compute its hash and compare with the entry in the password file
  - System does not store actual passwords!
  - Cannot go from hash to password!
- Why is hashing better than encryption here?
- Does hashing protect weak, easily guessable passwords?

Application: Software Integrity



Goal: Software manufacturer wants to ensure file is received by users without modification.

Idea: given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)

Which Property Do We Need?

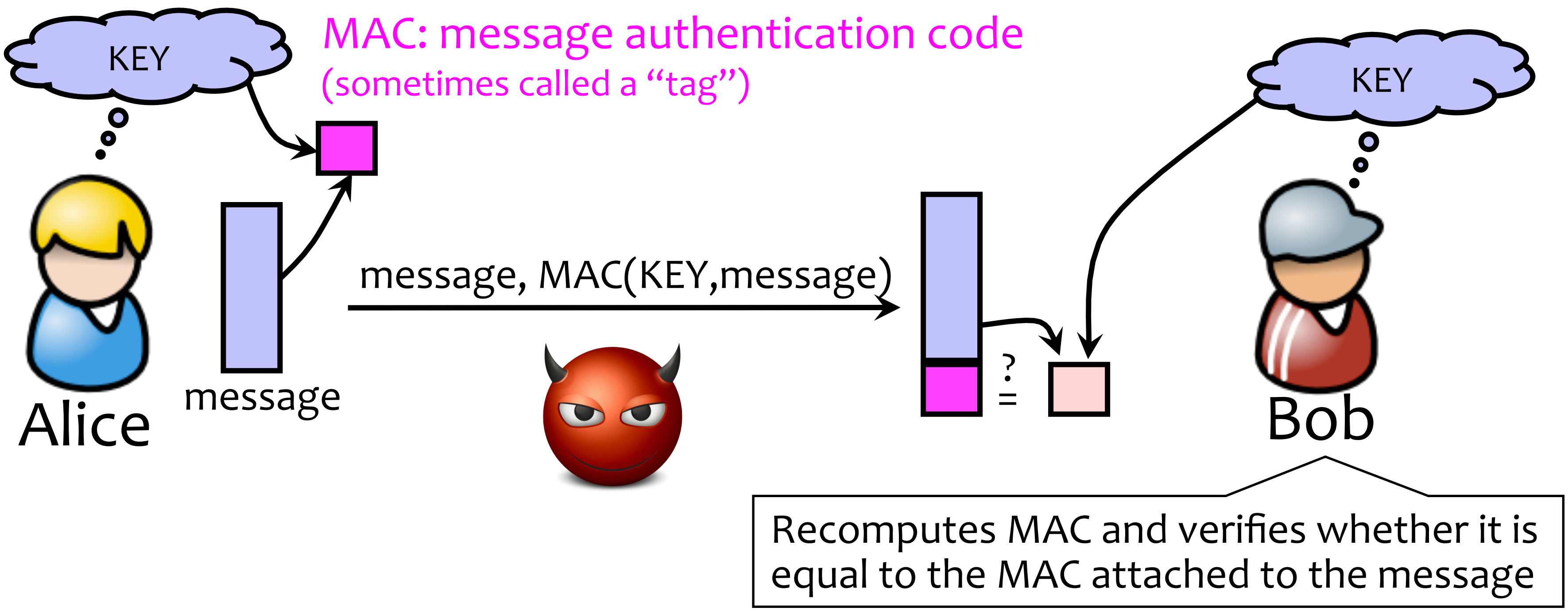
- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover the/a valid password
- Integrity of software distribution
  - Weak collision resistance
  - But software images are not really random… may need full collision resistance if considering malicious developers
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B (this may mean that she needs to hash some randomness with B too)
  - Collision resistance: Alice should not be able to change her mind to bid B' such that H(B)=H(B')

Common Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Also recently broken! (Theoretically -- not practical.)
- SHA-256, SHA-512, SHA-224, SHA-384
- SHA-3: standard released by NIST in August 2015

Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



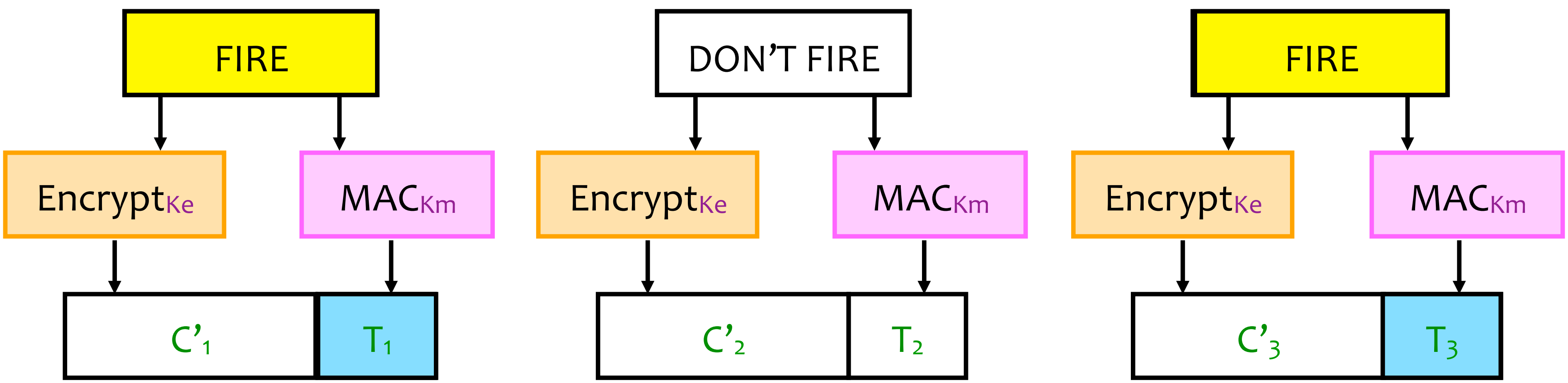
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

HMAC

- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec
- Why not encryption?
  - Hashing is faster than block ciphers in software
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

Authenticated Encryption

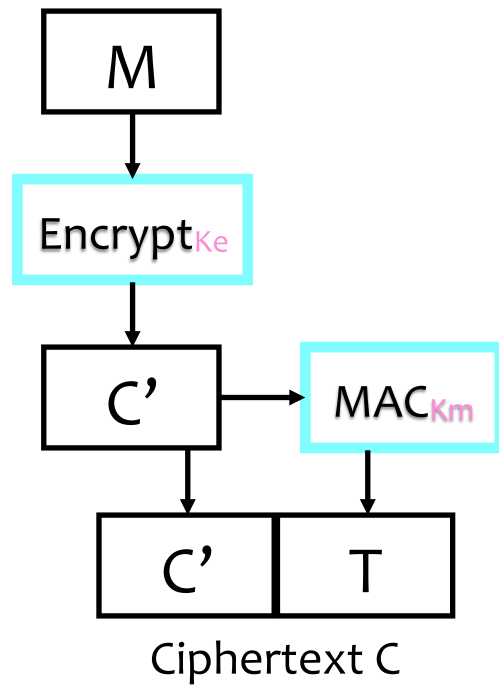
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.
- But be careful!
  - Obvious approach: Encrypt-and-MAC
  - Problem: MAC is deterministic! same plaintext → same MAC



Authenticated Encryption

- Instead:
  Encrypt *then* MAC.

- (Not as good:
  MAC-then-Encrypt)



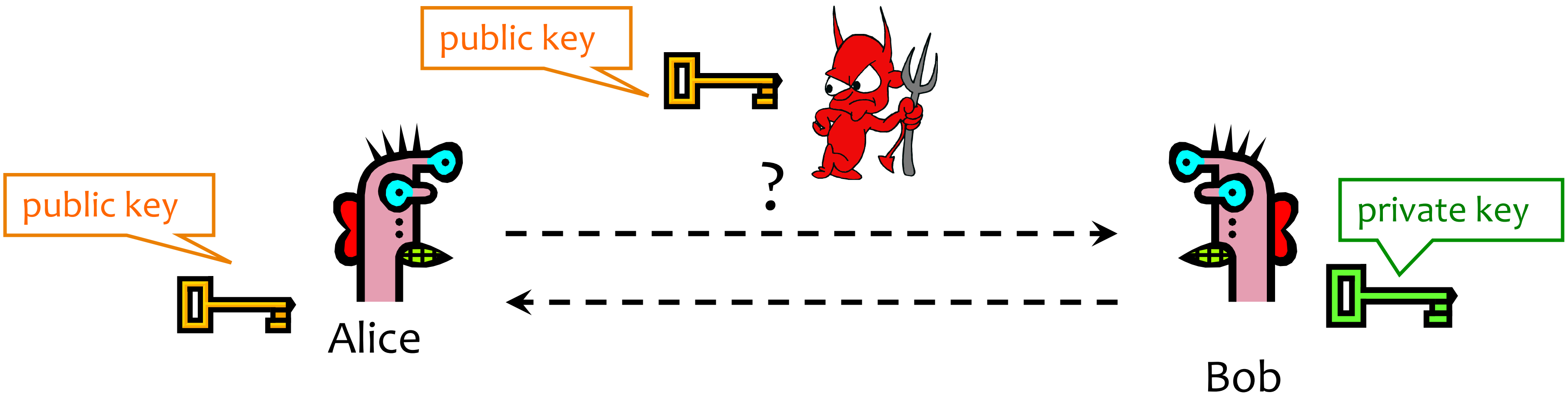
Encrypt-then-MAC

Asymmetric (Public Key) Cryptography

Reminder: Symmetric Cryptography

- **1 secret key (or 2 or …)**, shared between sender/receiver
- Repeat fast and simple operations lots of times (rounds) to mix up key and ciphertext
- **Why do we think it is secure?** (simplistic)
  - Lots of heuristic arguments
    - If we do lots and lots and lots of mixing, no simple formula (and reversible) describing the whole process (cryptographic weakness).
    - Mix in ways we think it's hard to short-circuit all the rounds. Especially non-linear mixing, e.g., S-boxes.
  - Some math gives us confidence in these assumptions

Public Key Crypto: Basic Problem



<u>Given</u>: Everybody knows Bob's public key
Only Bob knows the corresponding private key

<u>Goals</u>: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

Public Key Cryptography

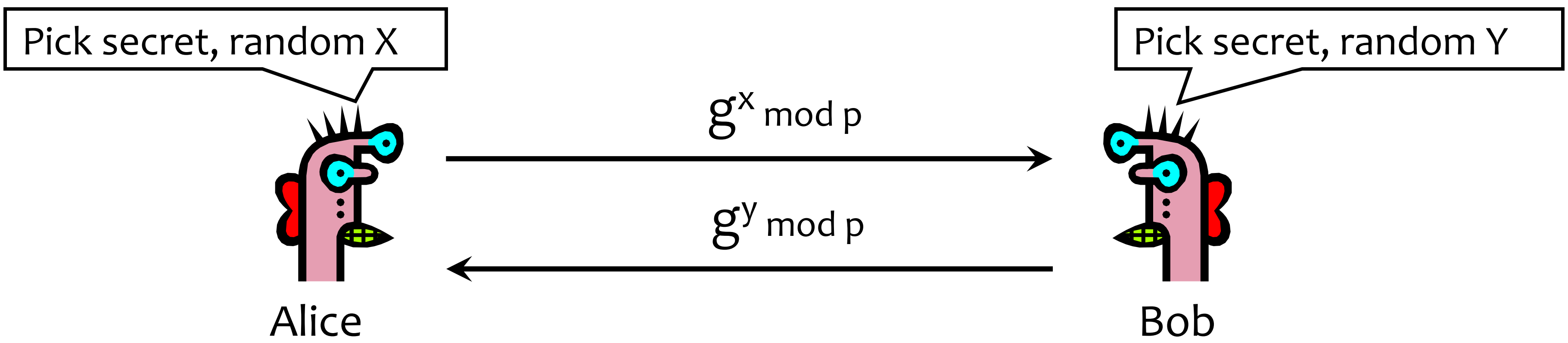
- Everyone has **1 private key and 1 public key**
  - Or 2 private and 2 public, when considering both encryption and authentication
- Mathematical relationship between private and public keys
- **Why do we think it is secure?** (simplistic)
  - Relies entirely on **problems we believe are "hard"**

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 (or at least different)
    - 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 <u>efficient</u> algorithm for doing this
  - This is <u>not</u> 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 (depends on choice of parameters!), 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
- Diffie-Hellman protocol (by itself) does not provide authentication

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