CSE 484: Computer Security and Privacy

Cryptography 5

Spring 2023

David Kohlbrenner

dkohlbre@cs

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

Logistics

- Lab 1b coming up next week
- Homework 2 will go out today, due in 2 weeksish

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

- Why is hashing better than encryption here?

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
- Why is hashing better than encryption here?

- System does not store actual passwords!
- Don't need to worry about where to store the key!
- Cannot go from hash to password!

Application: Password Hashing

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?

Application: Password Hashing + Salting

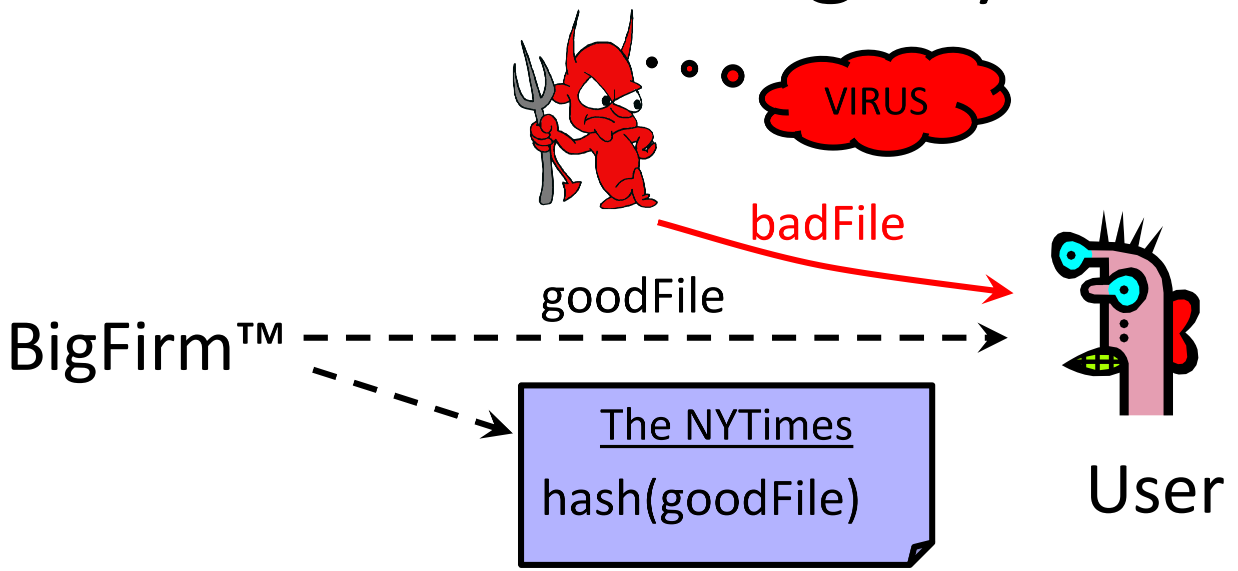
- Salting
  - We ‘salt’ hashes for password by adding a randomized suffix to the password
    - E.g. Hash(“coolpassword”+”35B67C2A”)
  - We then store the salt with the hashed password!
  - Server generates the salt

- The goal is to prevent *precomputation attacks*
  - If the adversary doesn’t know the salt, they can’t *precompute* common passwords

Hash Functions Review

- Map large domain to small range (e.g., range of all 160- or 256-bit values)
- Properties:
  - Collision Resistance: Hard to find two distinct inputs that map to same output
  - One-wayness: Given a point in the range (that was computed as the hash of a random domain element), hard to find a preimage
  - Weak Collision Resistance: Given a point in the domain and its hash in the range, hard to find a new domain element that maps to the same range element

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)

Application: Software Integrity

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?

Which Property Do We Need?

One-wayness, Collision Resistance, Weak CR?

- 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

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
- Commitments (e.g. auctions)
  - Alice wants to bid B, sends H(B), later reveals B
  - **One-wayness:** rival bidders should not recover B (this may mean that they need to hash some randomness with B too)
  - **Collision resistance:** Alice should not be able to change their mind to bid B’ such that H(B)=H(B’)

Commitments

Common Hash Functions

- **SHA-2: SHA-256, SHA-512, SHA-224, SHA-384**
- **SHA-3: standard released by NIST in August 2015**
- MD5 – Don't Use!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD
  - 160-bit version is OK
  - 128-bit version is *not* good
- SHA-1 (Secure Hash Algorithm) – Don't Use!
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005; practical attack 2017!

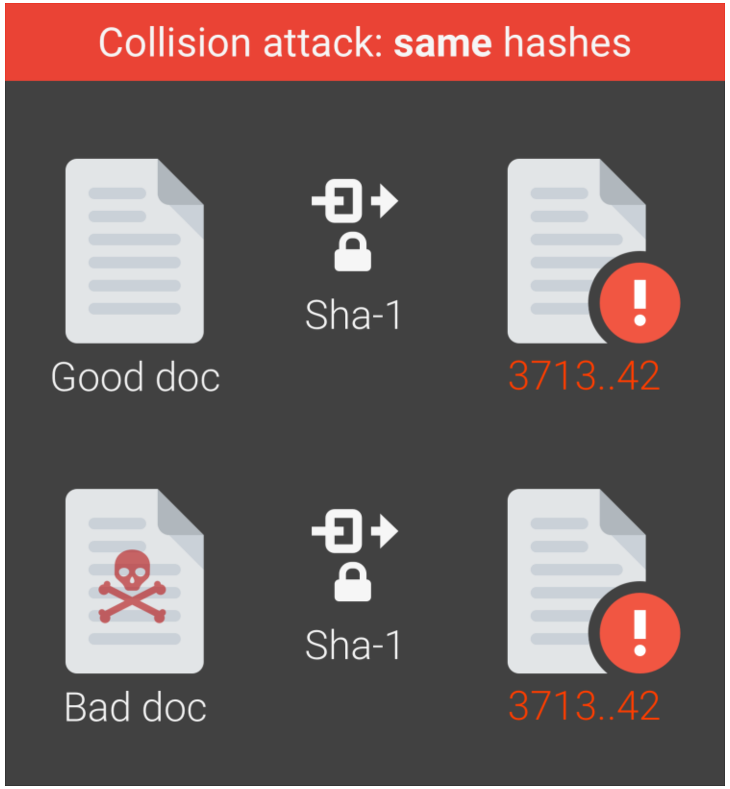
SHA-1 Broken in Practice (2017)

Google just cracked one of the building blocks of web encryption (but don't worry)

It's all over for SHA-1

by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST

https://shattered.io

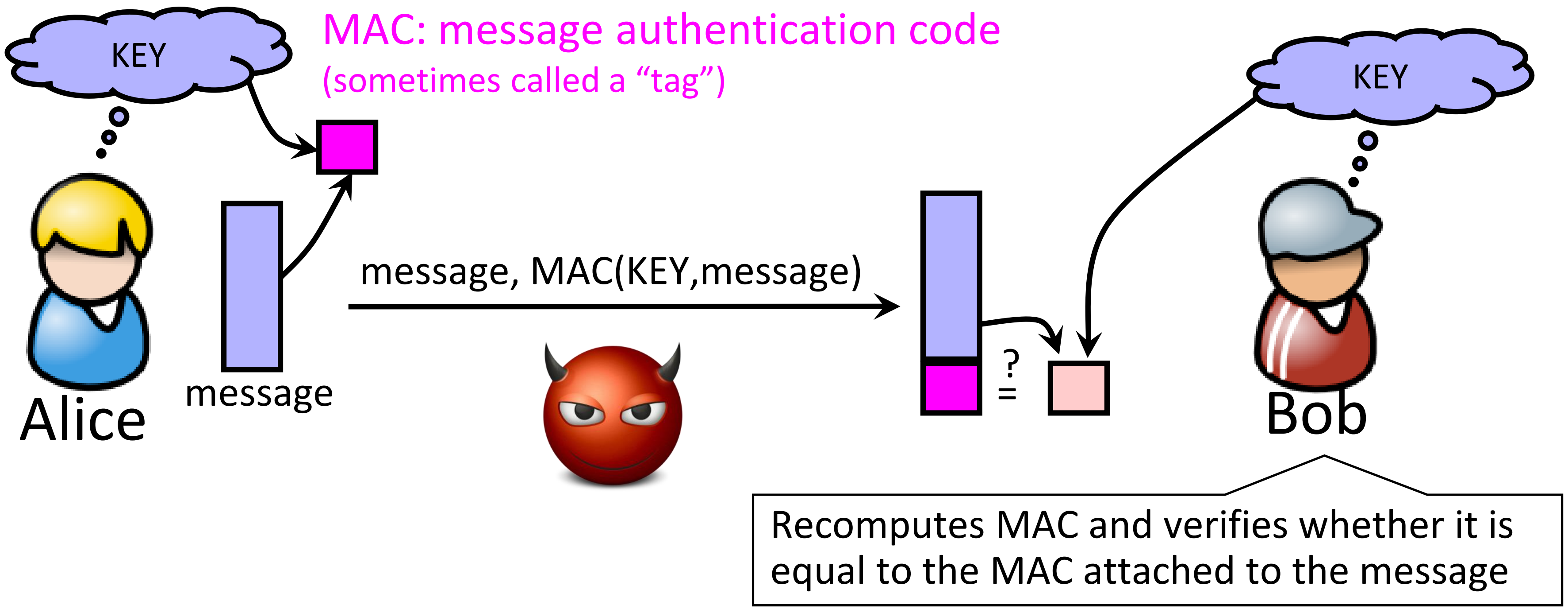


Aside: How we evaluate hash functions

- Speed
  - Is it amenable to hardware implementations?
- Diffusion
  - Does changing 1 bit in the input affect all output bits?
- Resistance to attack approaches
  - Collisions?
  - Length extensions?
  - etc

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? (Historical reasons)
  - 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

MAC with SHA3

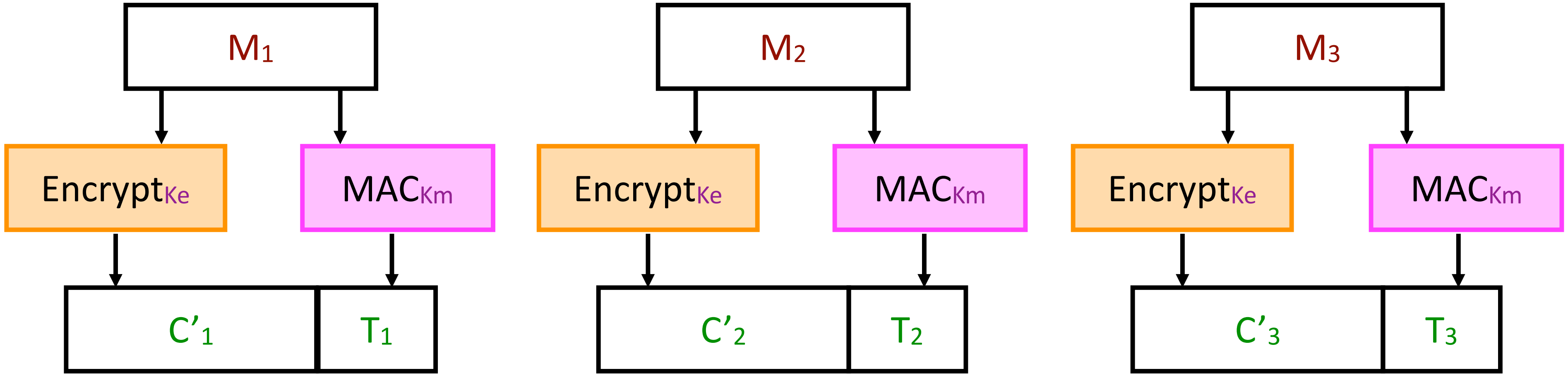
- SHA3(Key || Message)

- SHA3 is designed to get the same safety properties as HMAC constructions

Authenticated Encryption

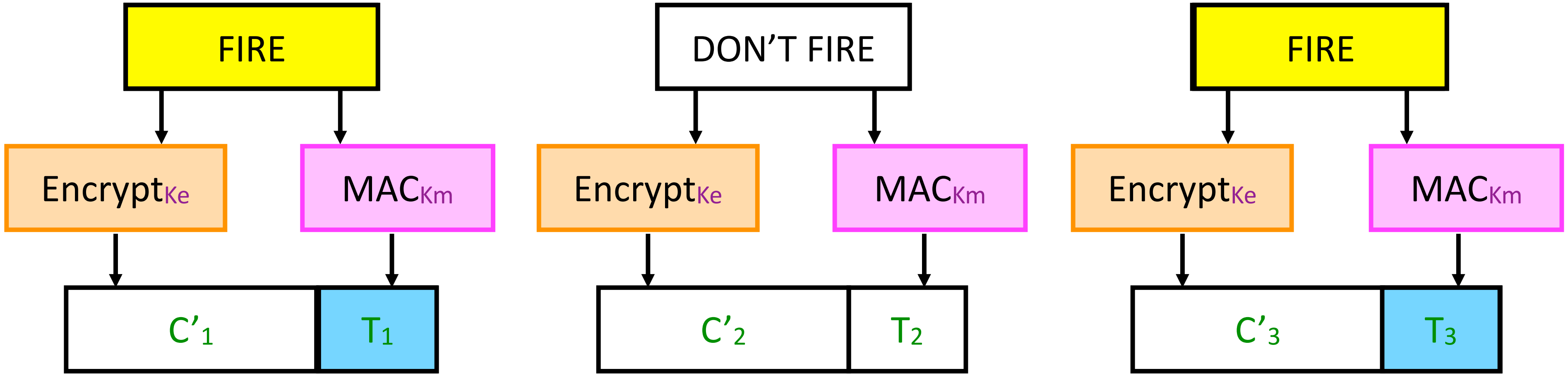
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.

- Is this fine? (Canvas!)



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

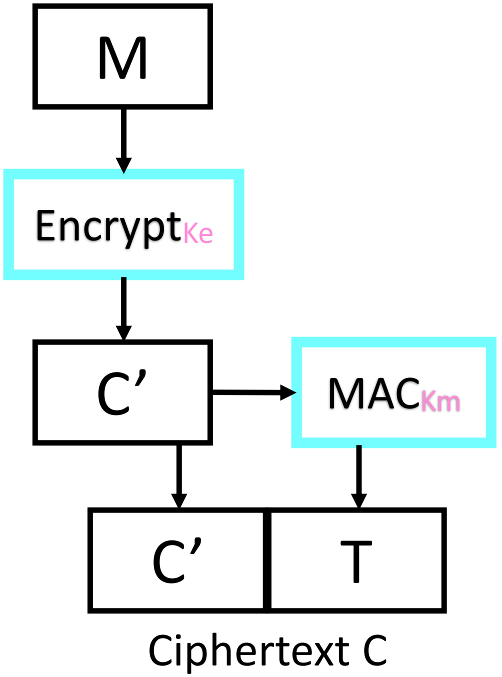


Authenticated Encryption

- Instead:

  Encrypt *then* MAC.

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



Encrypt-then-MAC

Back to cryptography land

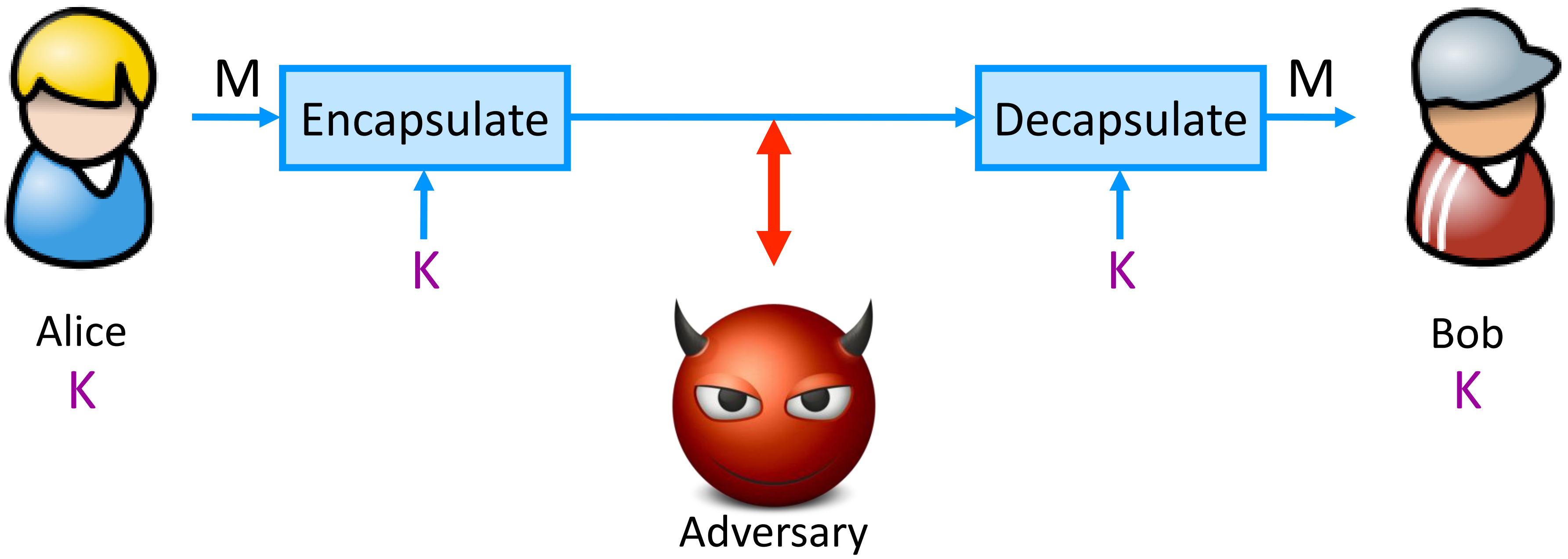
Stepping Back: Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.

- Asymmetric cryptography
  - Each party creates a public key pk and a secret key sk.

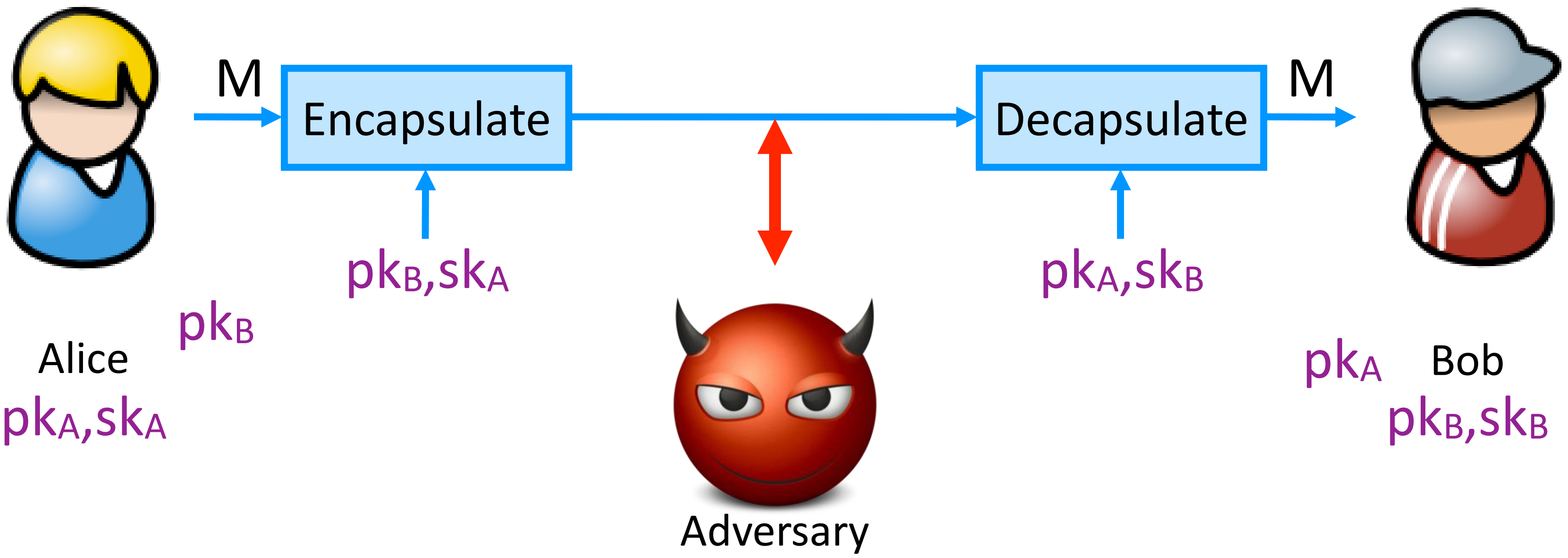
Symmetric Setting

Both communicating parties have access to a shared random string K, called the key.

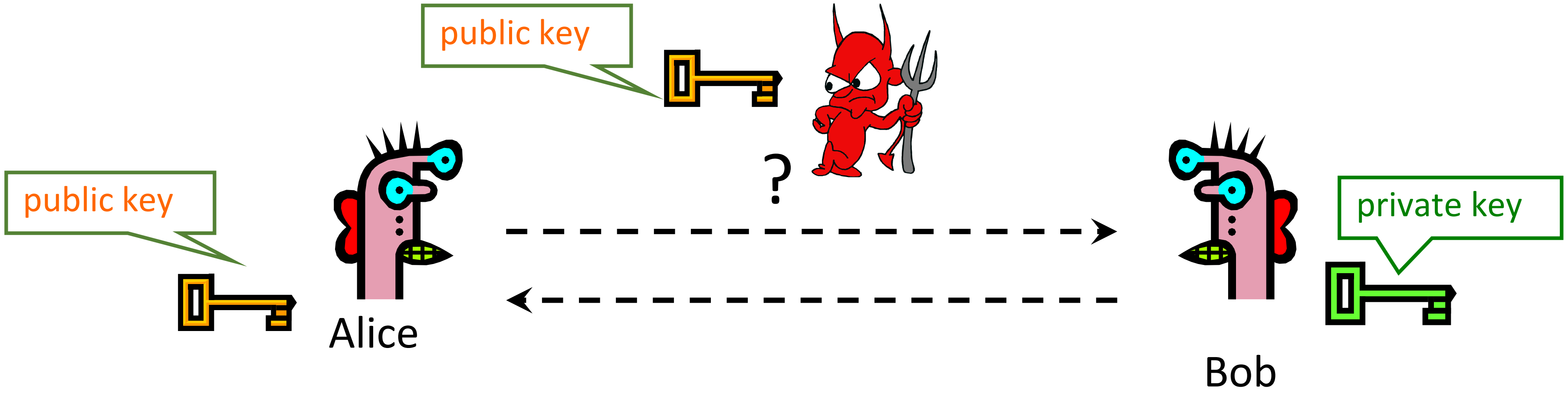


Asymmetric Setting

Each party creates a public key pk and a secret key sk.



Public Key Crypto: Basic Problem



Ignore for now: How do we know it's REALLY Bob's??

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 themself

Applications of Public Key Crypto

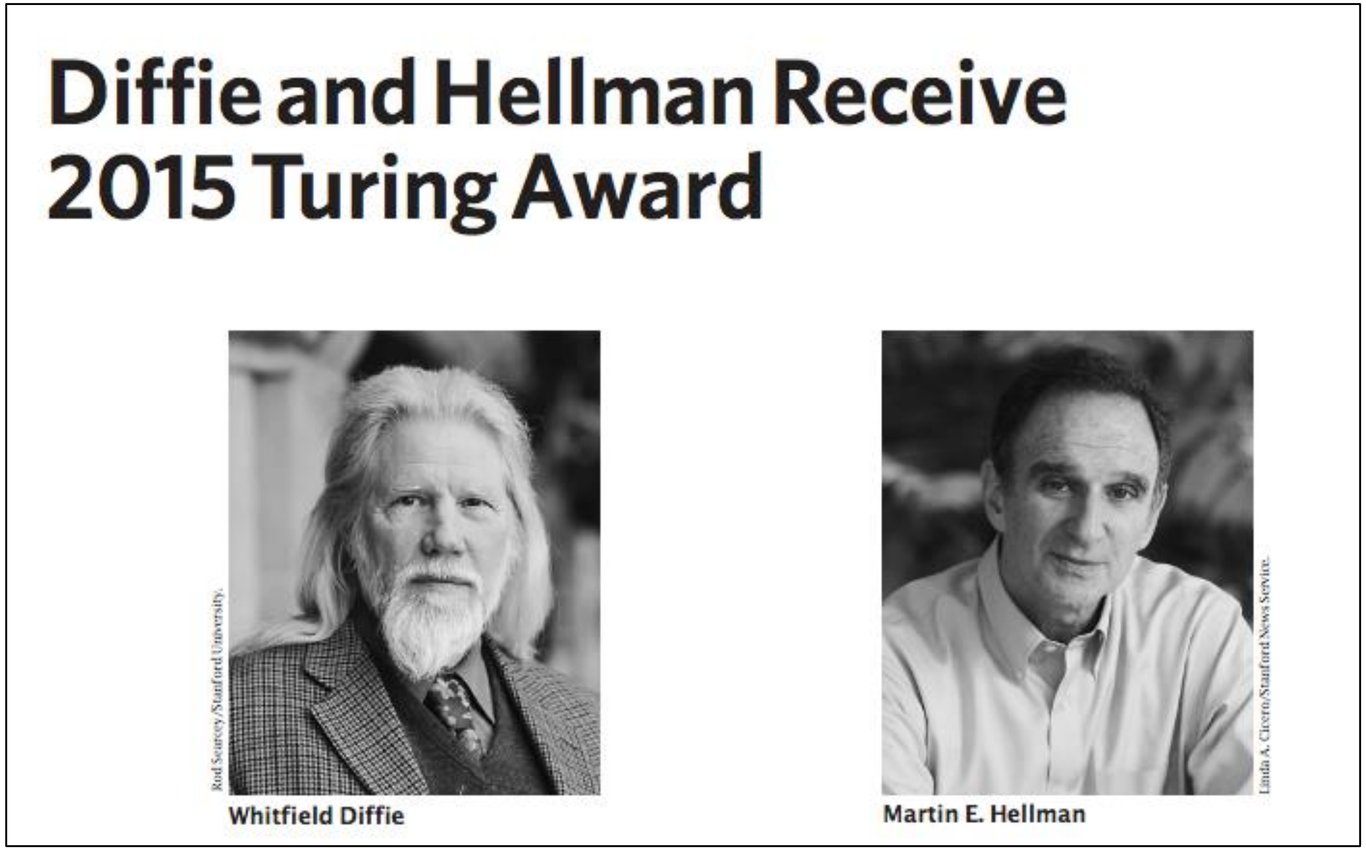
- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (or at least different)
    - Secret is stored only at one site: good for open environments
- 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?)

Session Key Establishment

Modular Arithmetic

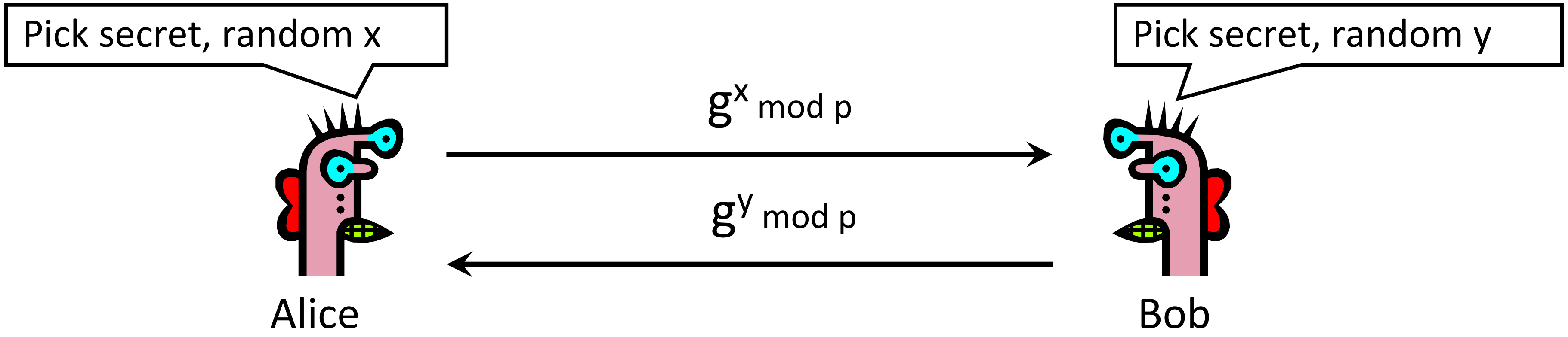
- Given g and prime p, compute: g^1 mod p, g^2 mod p, … g^{100} mod p
  - For p=11, g=10
    - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a "generator" of Z_{11}^*

Diffie-Hellman Protocol (1976)



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 Z_p^* i such that $a=g^i$ mod p
    - <u>Modular arithmetic</u>: numbers “wrap around” after they reach p



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

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

Example Diffie Hellman Computation

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*

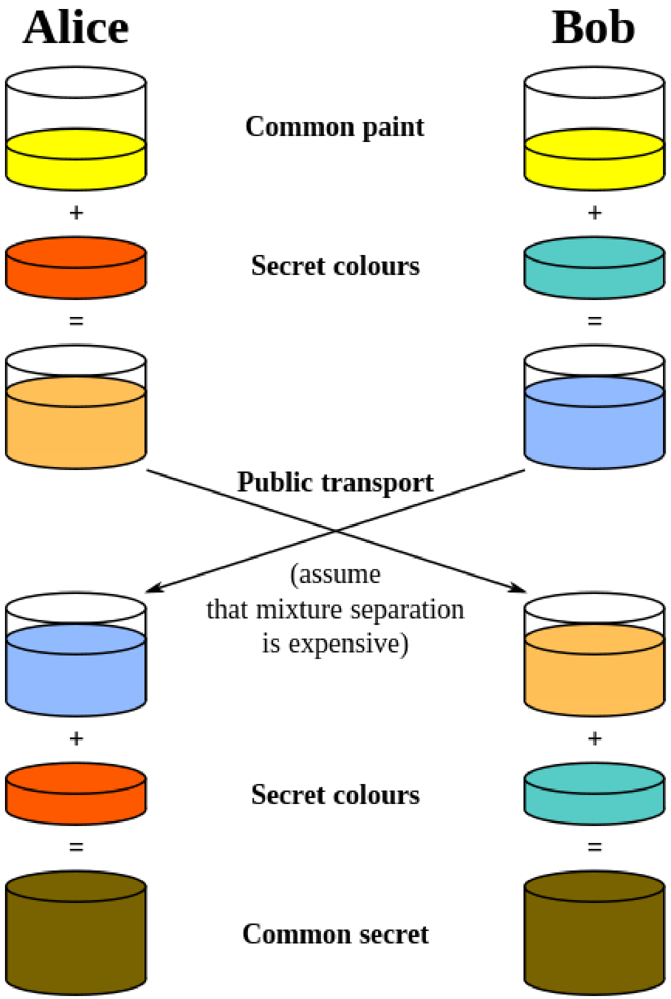
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:

  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

More on Diffie-Hellman Key Exchange

- 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

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:
$g^x \bmod p$
$g^y \bmod p$

Common secret: $g^{xy} \bmod p$

[from Wikipedia]

Diffie-Hellman Caveats

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive 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} *mod p*, and use the hash as the key
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Example from Earlier

- Given g and prime p, compute: g^1 mod p, g^2 mod p, … g^{100} mod p
  - For p=11, g=10
    - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a "generator" of $Z_{11}{}^*$
  - For p=11, g=3
    - 3^1 mod 11 = 3, 3^2 mod 11 = 9, 3^3 mod 11 = 5, …
    - Produces cyclic group {3,9,5,4,1} (order = 5) (5 is a prime)
    - g=3 generates a group of prime order

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