Cryptography
Lecture 8

Dr. Panagiotis Rizomiliotis

TOC

- Key derivation function
  - HKDF
- key agreement/transfer
  - Diffie Hellman
  - (non)-KEM
  - KEM
  - Quantum Key distribution
- Key size

Contemporary communication protocol

- **First Phase: Authentication (sometimes mutual)**
  - Public Key
  - Symmetric Key

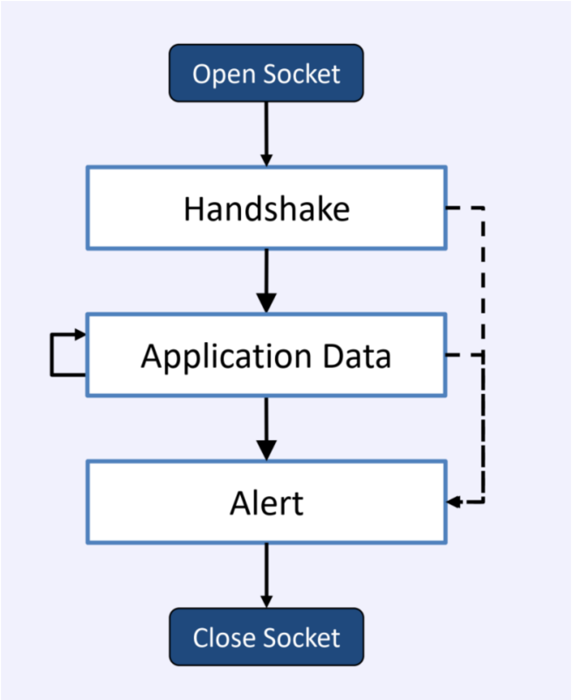
- **Second Phase: Key Establishment (master key)**
  - Key agreement
  - Key distribution

- **Third Phase: Data Encryption**
  - KDF (master key)
  - Symmetric key encryption

TLS 1.3 (example)

Handshake
- Agree a cipher suite.
- Agree a master secret.
- Authentication using certificate(s).

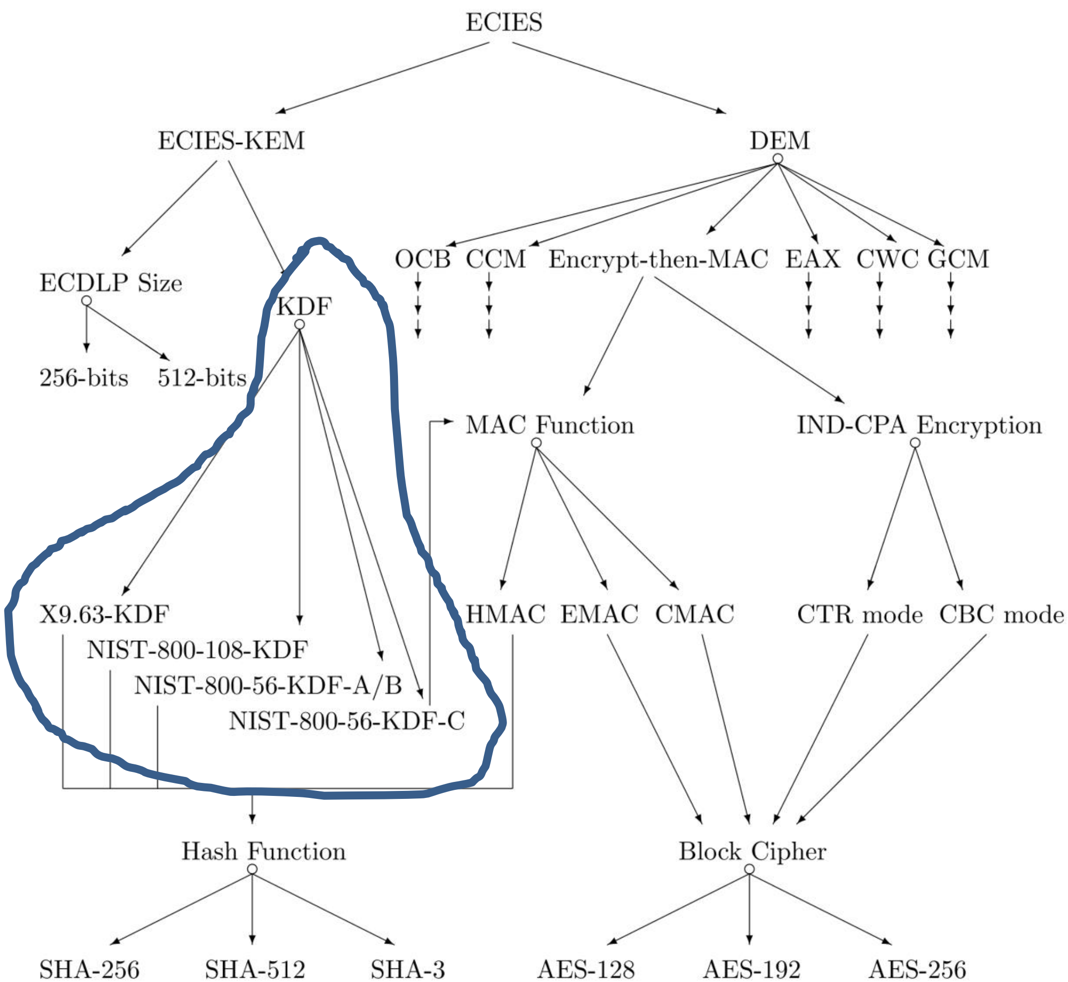
Application Data
- Use KDF to generate sessions keys
- Symmetric key encryption.
  - AEAD cipher modes.
- Typically HTTP



(OWASP presentation)

KEY DERIVATION

Overview



*** Algorithms, key size and parameters report. ENISA– 2014**

Key derivation function

- Key Derivation Functions (KDFs) are used to derive cryptographic keys

1. from a source of keying material shared random strings (in the case of key agreement protocols)
2. from an entropy source (in the case of key generation)

- KDFs act both as a randomness extractor as well as an expander

Deriving many keys from one

Typical scenario. a single source key (SK) is sampled from:

- Hardware random number generator
- A key exchange protocol (discussed later)

Need many keys to secure session:

- unidirectional keys; multiple keys for nonce-based CBC.

Goal: generate many keys from this one source key

$$\boxed{SK} \xrightarrow{\textbf{KDF}} \boxed{k_1,\ k_2,\ k_3,\ \ldots}$$

When source key is uniform

F: a PRF with key space K and outputs in $\{0,1\}^n$

Suppose source key SK is uniform in K

- Define Key Derivation Function (KDF) as:

KDF(SK, CTX, L) :=

$$F(SK, (\mathbf{CTX} \parallel \mathbf{0})) \parallel F(SK, (\mathbf{CTX} \parallel \mathbf{1})) \parallel \cdots \parallel F(SK, (\mathbf{CTX} \parallel \mathbf{L}))$$

CTX: a string that uniquely identifies the application

$\mathbf{KDF}(SK, CTX, L) :=$

$$F(SK, (\mathbf{CTX} \parallel \mathbf{0})) \parallel F(SK, (\mathbf{CTX} \parallel \mathbf{1})) \parallel \cdots \parallel F(SK, (\mathbf{CTX} \parallel \mathbf{L}))$$

What is the purpose of CTX?

Even if two apps sample same SK they get indep. keys

It's good practice to label strings with the app. name

It serves no purpose

What if source key is not uniform?

Recall: PRFs are pseudo random only when key is uniform in K

- SK not uniform $\Rightarrow$ PRF output may not look random

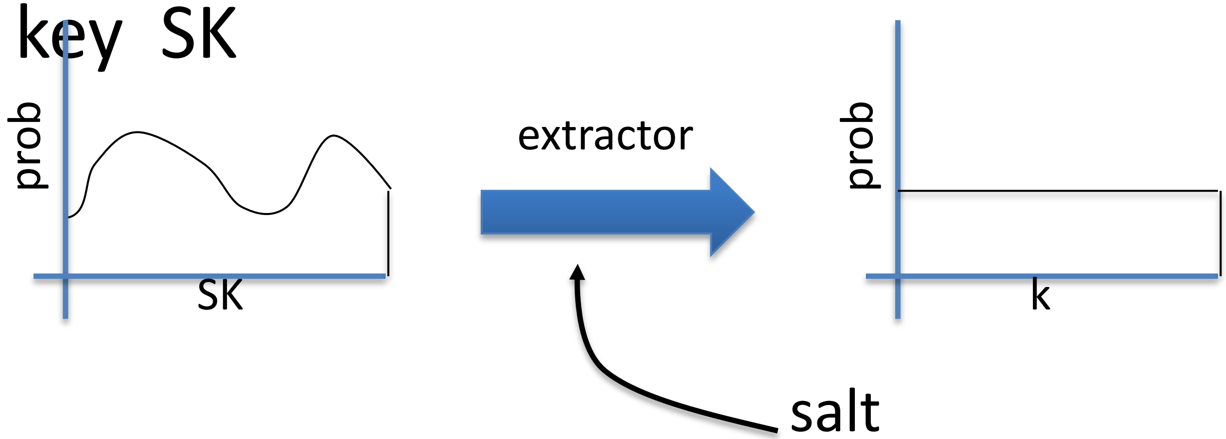
Source key often not uniformly random:

- Key exchange protocol: key uniform in some subset of K

- Hardware RNG: may produce biased output

Extract-then-Expand paradigm

Step 1: **extract** pseudo-random key k from source key SK



salt: a fixed non-secret string chosen at random

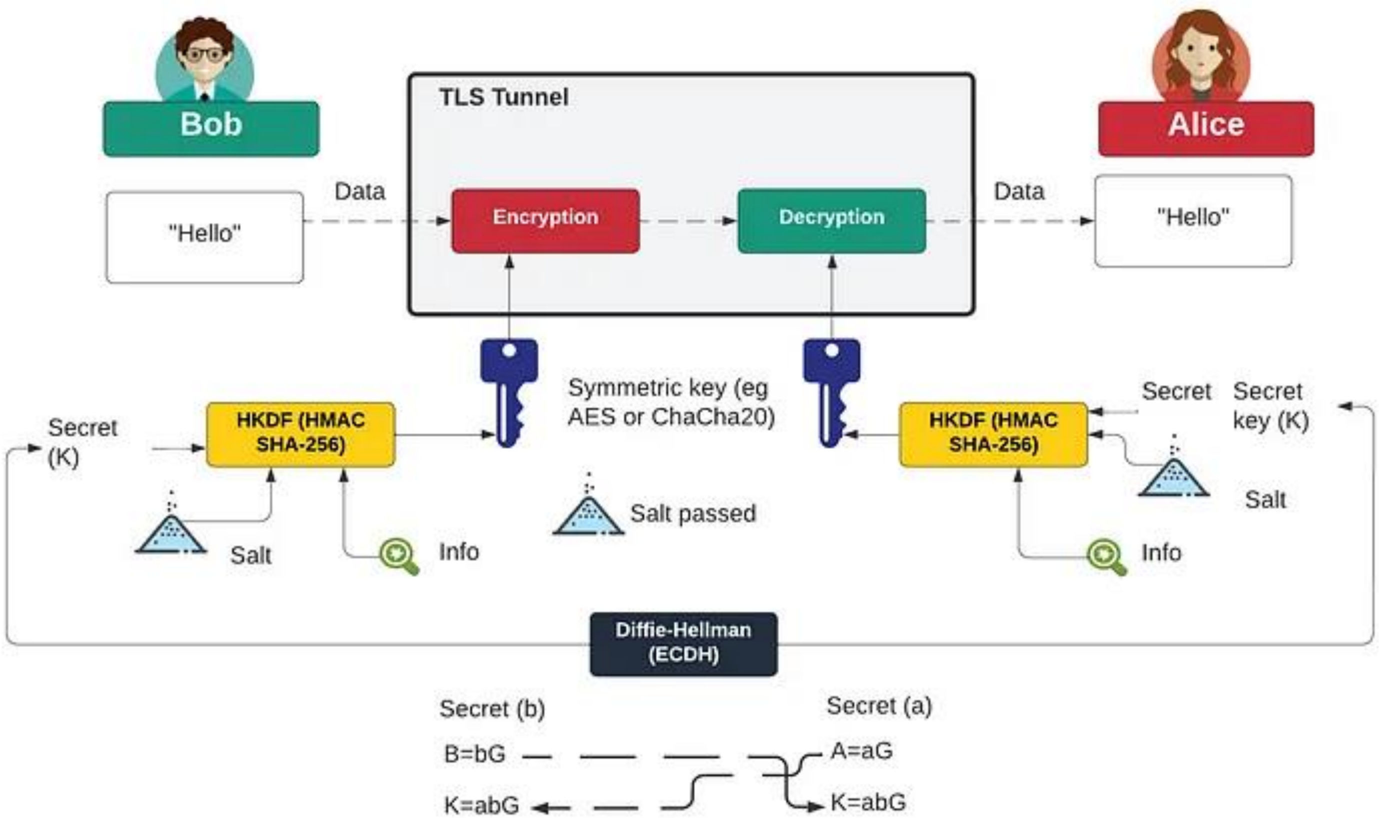
step 2: **expand** k by using it as a PRF key as before

HKDF: a KDF from HMAC

Implements the extract-then-expand paradigm:

- extract: use **$k \leftarrow$ HMAC(salt, SK)**
- Then expand using HMAC as a PRF with key **k**

HKDF in TLS



Bob
Alice
TLS Tunnel
"Hello"
Data
Encryption
Decryption
Data
"Hello"
Symmetric key (eg
AES or ChaCha20)
Secret
(K)
HKDF (HMAC
SHA-256)
Salt
Info
Salt passed
HKDF (HMAC
SHA-256)
Secret
Secret
key (K)
Salt
Info
Diffie-Hellman
(ECDH)
Secret (b)
B=bG
K=abG
Secret (a)
A=aG
K=abG

Key derivation function

Primitive	Classification: Legacy	Classification: Future	Building Block
NIST-800-108-KDF(all modes)	✓	✓	A PRF
X9.63-KDF	✓	✓	Any hash function
NIST-800-56-KDF-A/B	✓	✓	Any hash function
NIST-800-56-KDF-C	✓	✓	A MAC function
HKDF	✓	✓	HMAC based PRF
IKE-v2-KDF	✓	✓	HMAC based PRF
TLS-v1.2-KDF	✓	✓	HMAC (SHA-2) based PRF
IKE-v1-KDF	✓	✗	HMAC based PRF
TLS-v1.1-KDF	✓	✗	HMAC (MD-5 and SHA-1) based PRF

Password-Based KDF (PBKDF)

Deriving keys from passwords:

- Do not use HKDF: passwords have insufficient entropy
- Derived keys will be vulnerable to dictionary attacks

PBKDF defenses: **salt** and a **slow hash function**

Standard approach: **PKCS#5** (PBKDF1)

$\mathbf{H^{(c)}(pwd \parallel salt)}$: iterate hash function c times

Password based key derivation

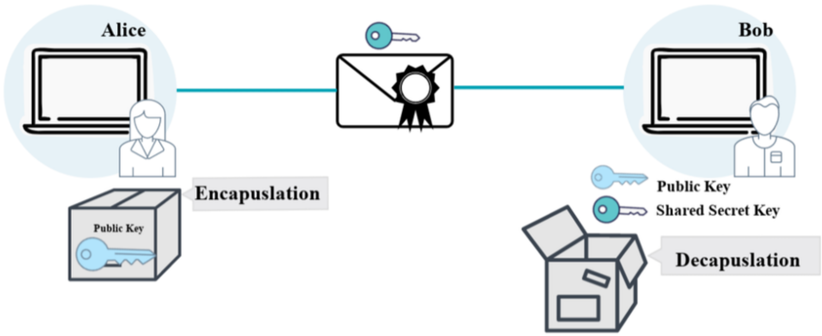
Goal: derive cryptographic keys from a secret random string (passwords)

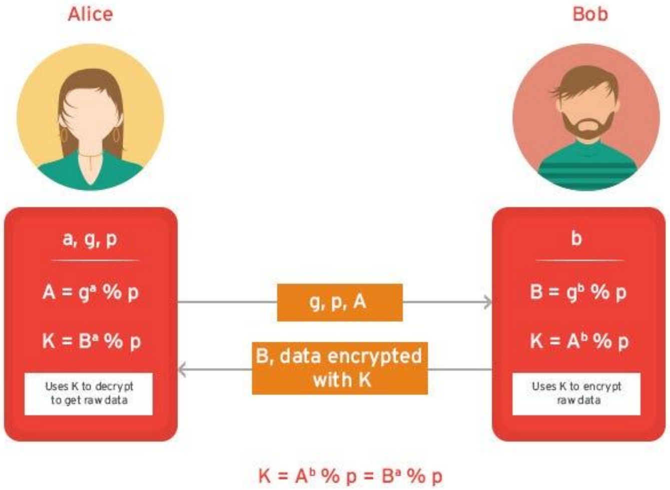
- PBKDF2
  - NIST SP 800-132
    Based on any secure PRF (for instance a hash function)
  - The PRF is iterated several times (at least 103, recommended 4*104)
    increase the workload of dictionary attacks
  - Input is the password, a salt and the desired key length
  - Possible to implement dictionary attacks on ASICs or GPUs
- Bcrypt
  - Based on block cipher (Blowfish)
- Scrypt
  - Since 2009. Looks more resistant so far.
- Argon2
  - From 2013 to 2015 the Password Hashing Competition (https://password-hashing.net/)
  - Main security goal is that these hash functions are 'memory hard', it is difficult to speed them up with dedicated hardware
  - Another similar proposal is Blocki

KEY AGREEMENT/TRANSFER

ToC

- Bob and Alice must agree on a common key.
- Then, they use a key derivation function to produce several symmetric keys





Protecting data confidentiality

- Public key encryption and decryption are expensive computations.
- Rarely used for plaintext confidentiality protection.

- Main schemes used in practice:
  - KEM: Key Encapsulation Mechanism

    Combine a public key encryption with key derivation functions (KDF)

  - Non-KEM

    Just traditional public key encryption (only two options in practice):

    1. RSA-PKCS# 1 v1.5
    2. RSA-OAEP

- Symmetric key based data protection.
  - DEM: Data Encryption Mechanism

Protecting data confidentiality

Scheme	Classification: Legacy	Classification: Future
RSA-OAEP	✓	✓
RSA-KEM	✓	✓
PSEC-KEM	✓	✓
ECIES-KEM	✓	✓
RSA-PKCS# 1 v1.5	✗	✗

Non-kem

- RSA-PKCS# 1 v1.5
  - No modern security proof
  - Used in SSL/TLS protocol extensively (until v1.2)
  - The weak form of padding
  - Attacks on various cryptographic devices

- RSA-OAEP
  - the preferred method of using the RSA primitive to encrypt a small message
  - Provably secure in the random oracle model
  - The hash functions used can be SHA-1 for legacy applications and SHA-2/SHA-3 for future applications

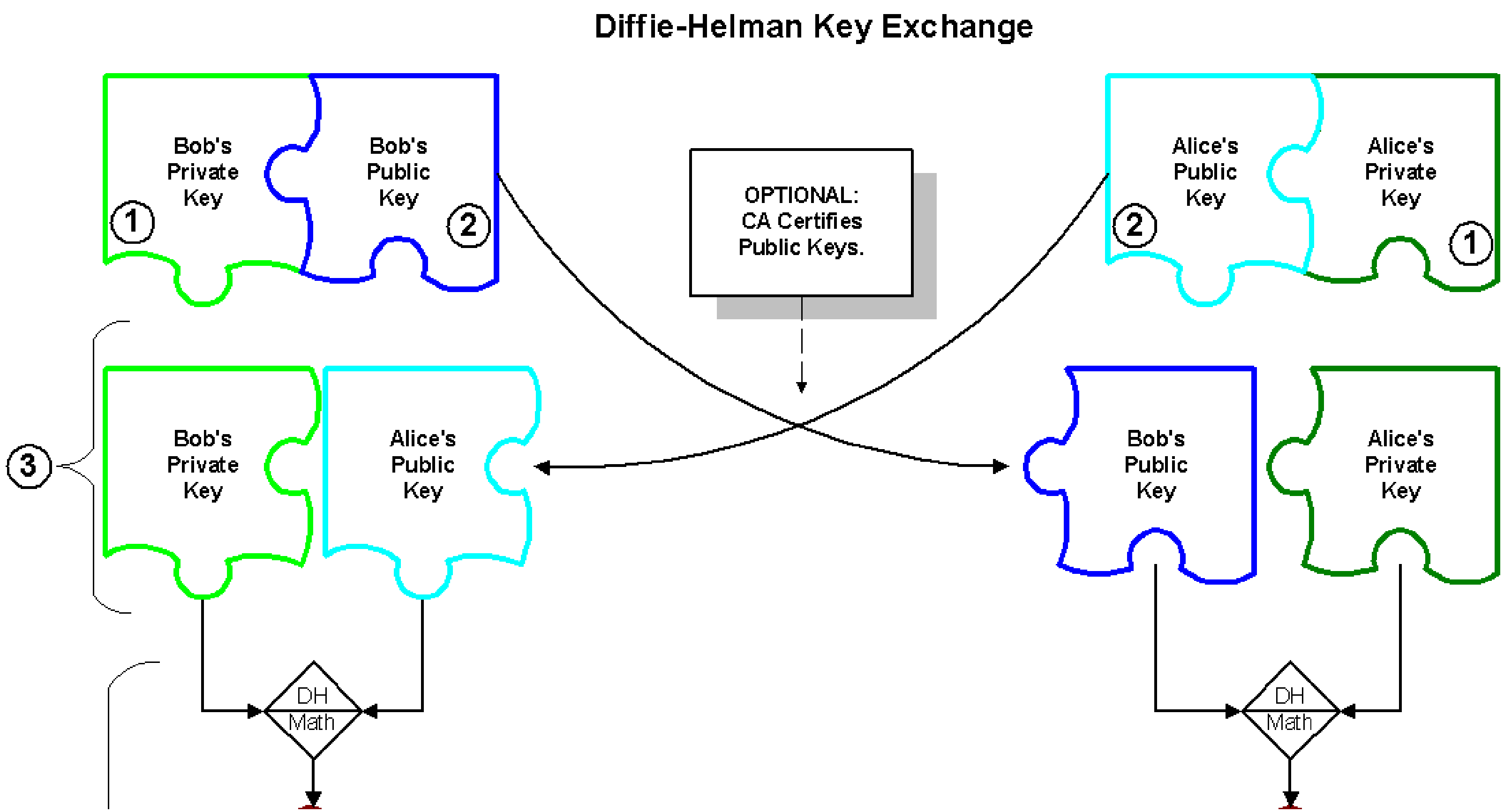
Key Encapsulation Mechanism (KEM)

- RSA-KEM
  - Takes a random element m and encrypts it using the RSA
  - The output key is computed by applying a KDF to m
  - Secure in the random oracle model
- PSEC-KEM
  - It is based on elliptic curves.
  - Provable secure
  - Based on the hardness of the (computational) DH problem
  - More secure than ECIES-KEM, less efficient
- ECIES-KEM
  - Discrete logarithm based encryption scheme
  - Very popular

Key agreement

- 1976: "New directions in Cryptography"
- Two entities agree upon a common secret over a public channel
  - No pre-shared keys.
- Based on the discrete logarithm problem

The main idea - DH



Implementation

- p and g are both publicly available numbers
- Users, Alice and Bob, pick private random values (when used once are called ephemeral):
  - Private Alice: a
  - Private Bob: b
- They compute public values
  - Public Alice: $x = g^a \bmod p$
  - Public Bob: $y = g^b \bmod p$
- Public values x and y are exchanged

(Ephemeral) DH

Alice

$x = g^a \bmod p$

$\longrightarrow$

$y = g^b \bmod p$

$\longleftarrow$

Bob

$K = k_a = y^a \bmod p$

$K = k_b = x^b \bmod p$

- Algebraically it can be shown that $k_a = k_b$
  - Users now have a symmetric secret key to encrypt
  - They use a KDF first…

Toy Example

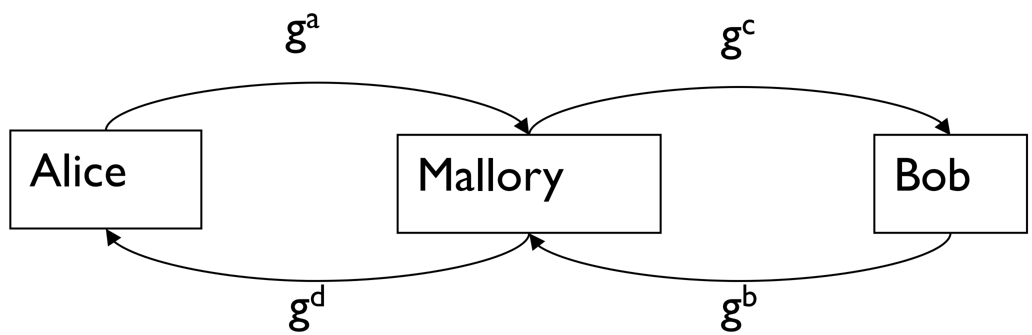
- Alice and Bob get public numbers
  - $p = 23, \; g = 9$

- Alice and Bob compute public values
  - $X = 9^4 \bmod 23 = 6561 \bmod 23 = 6$
  - $Y = 9^3 \bmod 23 = 729 \bmod 23 = 16$

- Alice and Bob exchange public numbers

- Alice and Bob compute symmetric keys
  - $k_a = y^a \bmod p = 16^4 \bmod 23 = 9$
  - $k_b = x^b \bmod p = 6^3 \bmod 23 = 9$
- Alice and Bob now can talk securely!

Person-in-the-middle attack



$Key_1 = g^{ad}$ $Key_1 = g^{cb}$

Mallory gets to listen to everything.

Solution

- AKE protocols (authentication and key establishment protocols)
  - Authenticate before key establishment
  - Literally hundreds of AKE protocols

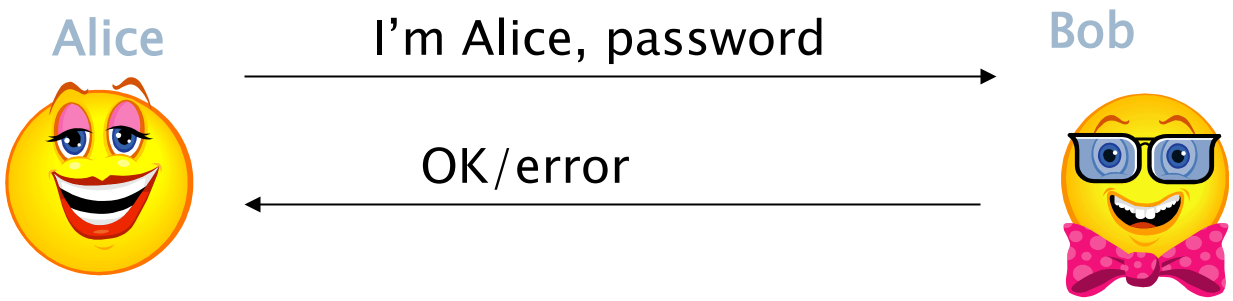
- Authentication:
  - Use public key encryption (and usually certificates)
  - Use pre-shared keys (like passwords)

- Two main types of key establishment:
  - Key agreement (DH)
  - Key distribution/transfer (key encryption/KEM)

Authentications

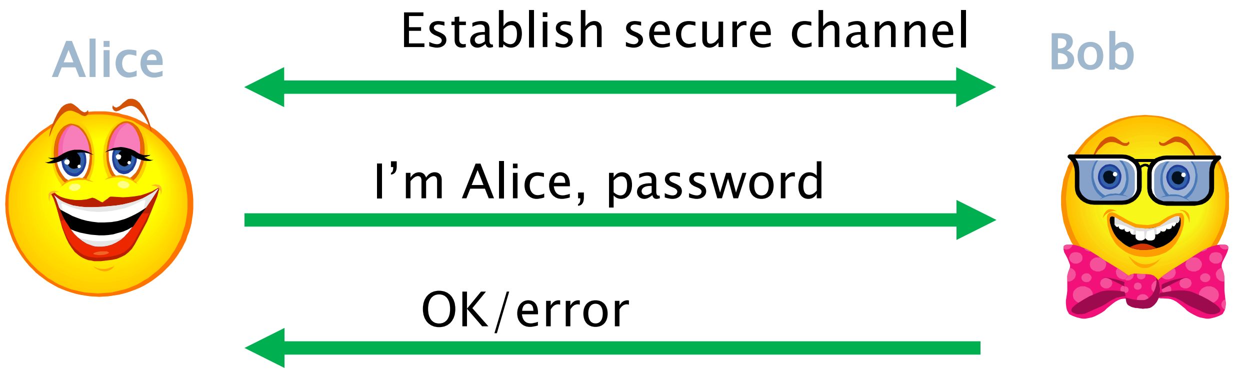
- Use public key encryption (and usually certificates)
- Use pre-shared keys (like passwords or master key of the last session)

Simple Transmission (PSK)

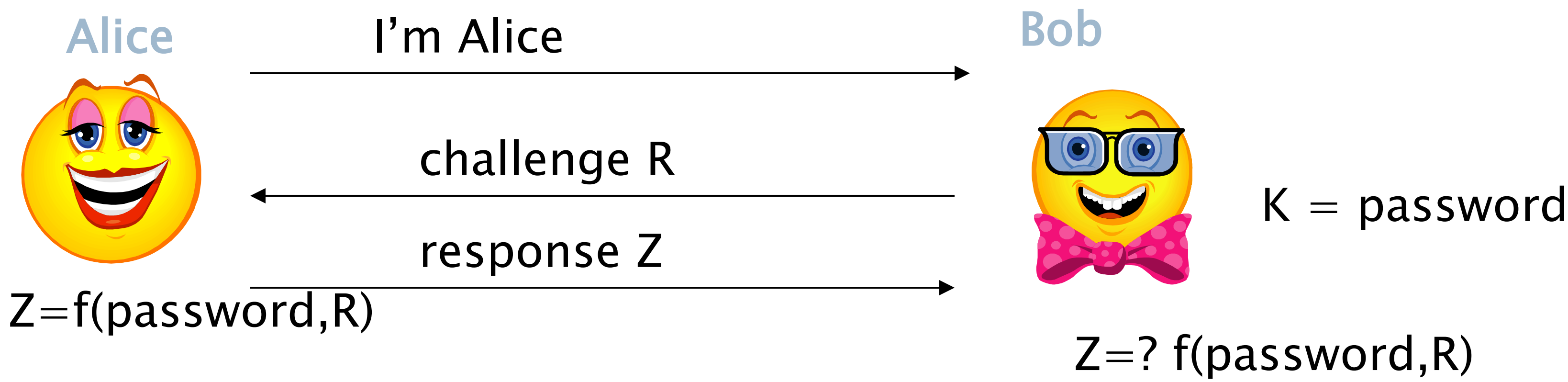


- Insecure!
- Can be easily eavesdroped

Secure simple Transmission (PSK)



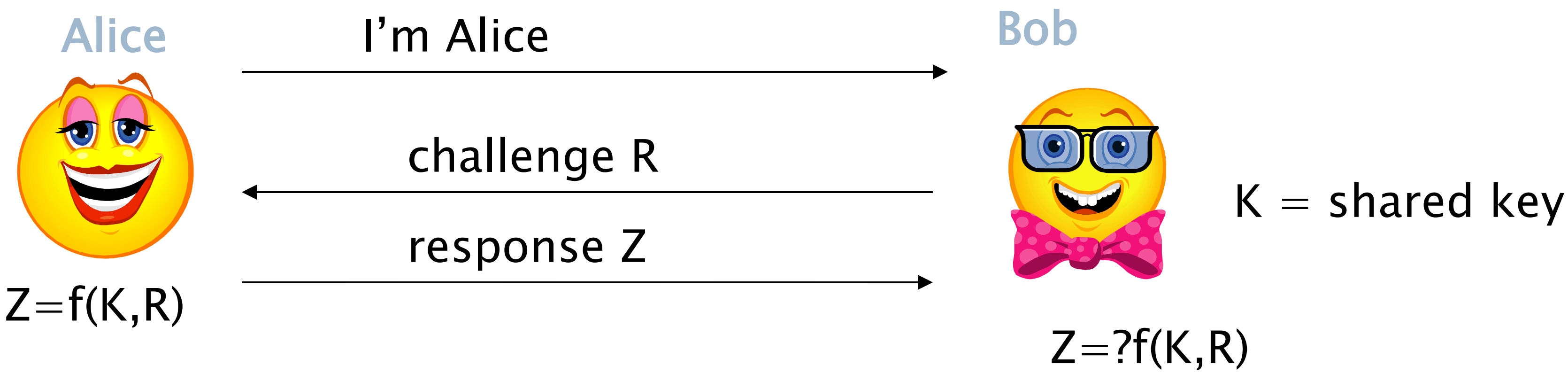
One-way Challenge-Response



f() can be:

- encryption function – Bob just decrypts and verifies time in within allowed skew
- hash – Bob needs to hash all times in allowable interval or Alice sends time

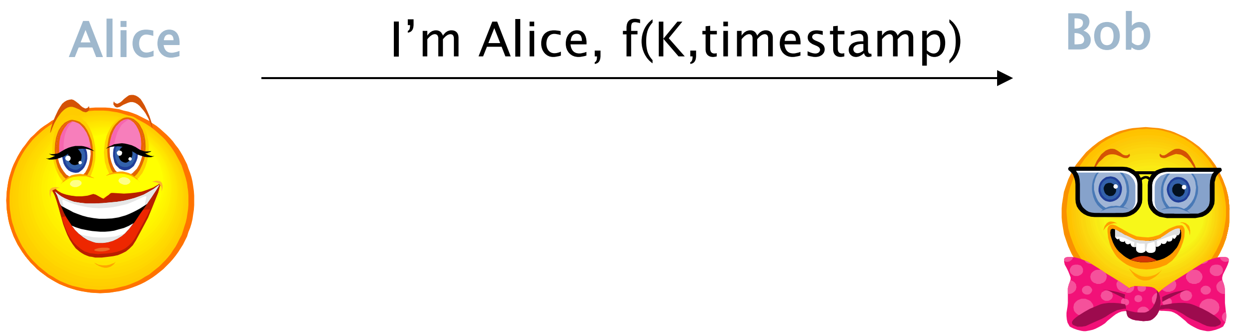
One-way Challenge-Response (PSK)



f() can be:

- encryption function – Bob just decrypts and verifies time in within allowed skew
- hash – Bob needs to hash all times in allowable interval or Alice sends time
- It is better to use MAC (usually HMAC)

One-Way using Timestamp (PSK)

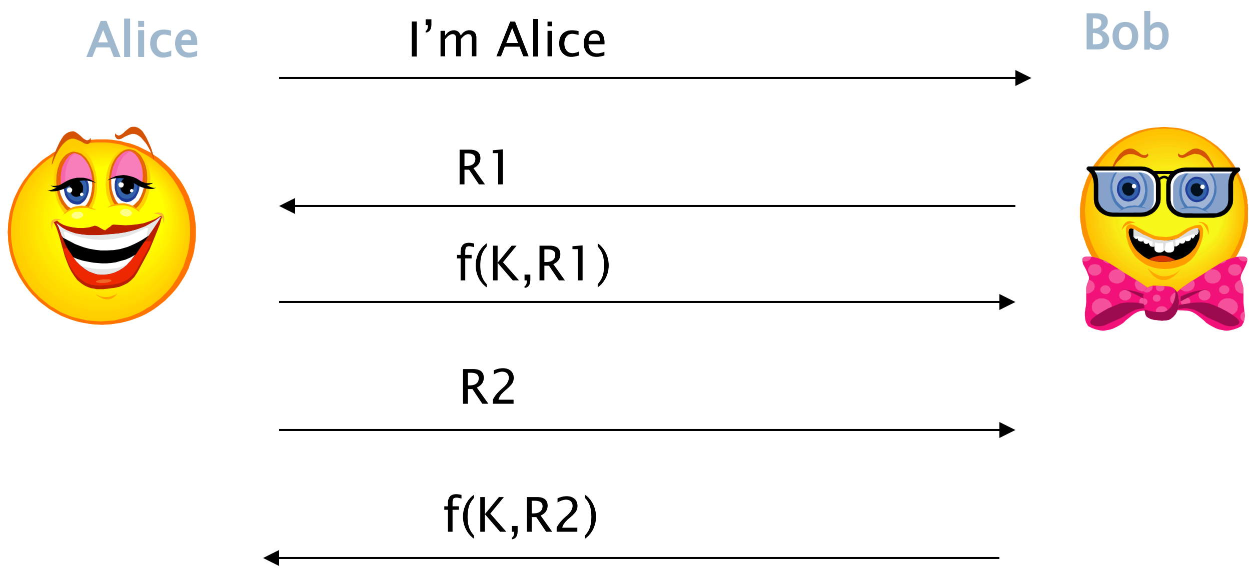


- Problems?
  - Impersonate Alice if intercept and send message – race condition
  - If use same K with multiple servers, could send message to another server and impersonate Alice
  - Clock skew/synchronization

2-Way Authentication

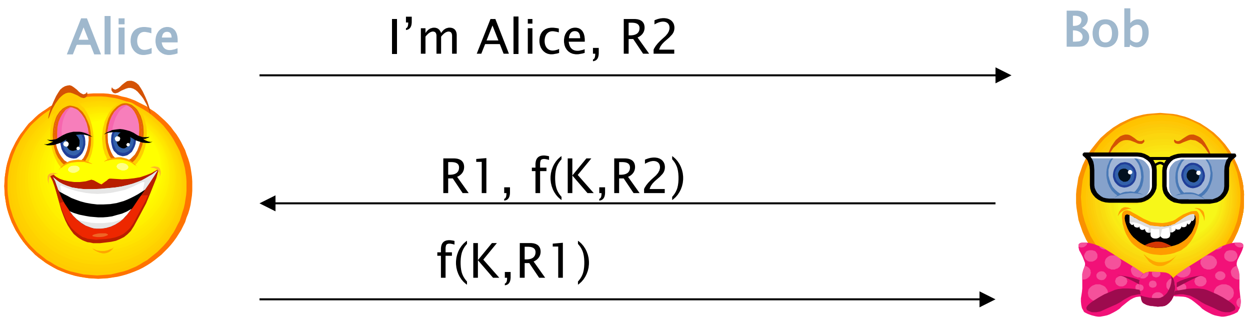
- Authentication often needed in both directions
- Server trusting user is not only concern
  - User must trust server
  - Ex. User accessing online bank account

Mutual Authentication with Secret Key



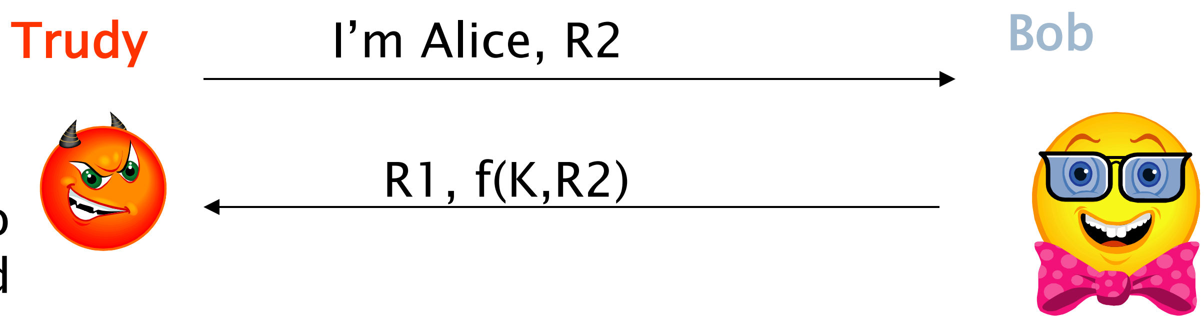
Mutual Authentication with Secret Key

More efficient version:

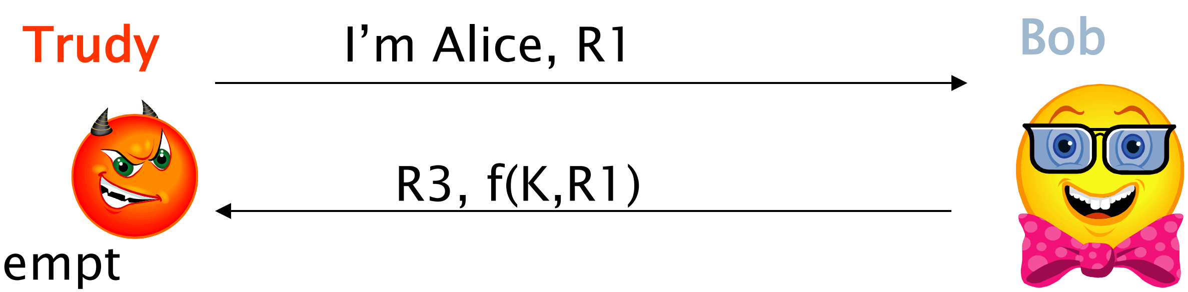


Mutual Authentication with Secret Key

Reflection attack:



Doesn’t know K so can’t send f(K,R1)



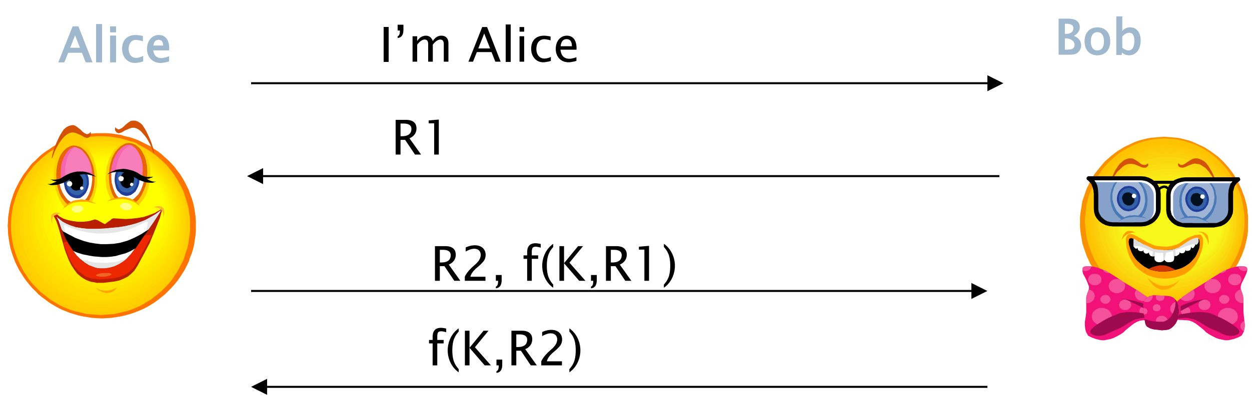
Now use f(K,R1) in above attempt

Mutual Authentication with Secret Key

- Solutions:
  - Separate keys for each direction/different passwords
  - Requirements on R values: odd in one direction, even in the other, concatenate with senders' name

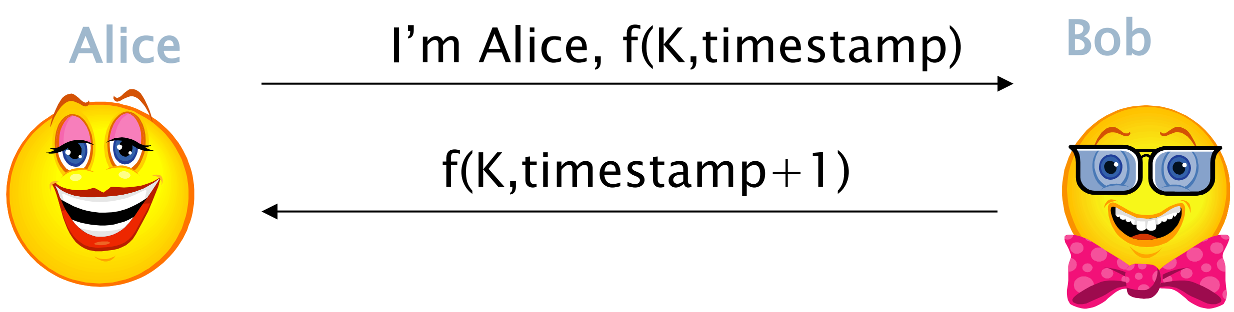
Password/Key Guessing

- Also note, Trudy can get Bob to encrypt a value (or a several of values) and then try an offline attack to guess K
- Have Bob return R1 value for Alice to encrypt



Now Bob would have to reuse R1 in order for Trudy, who eavesdrops, to be able to use f(K,R1)

Timestamps



- Same issues as before plus clock skew
- Any modification to timestamp will work

Certification based

- We use public key cryptography
- Prove the possession of a public key
- Usually it is based on certificates
- Very popular

One-way Using Public Key

Alice

I’m Alice →

← R

$[R]_{Apriv}$ →

Bob

Bob decrypts with Alice’s public key and verifies R was returned.

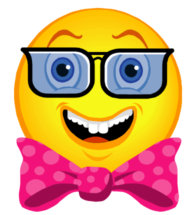
One-way Using Public Key

Alice 

I'm Alice →

← R

$[R]_{Apriv}$ →

Bob 

Bob decrypts with Alice's public key and verifies R was returned.

 Alice

Alice proves to Bob she has her private key by returning R

I'm Alice →

← $[R]_{Apub}$

R →

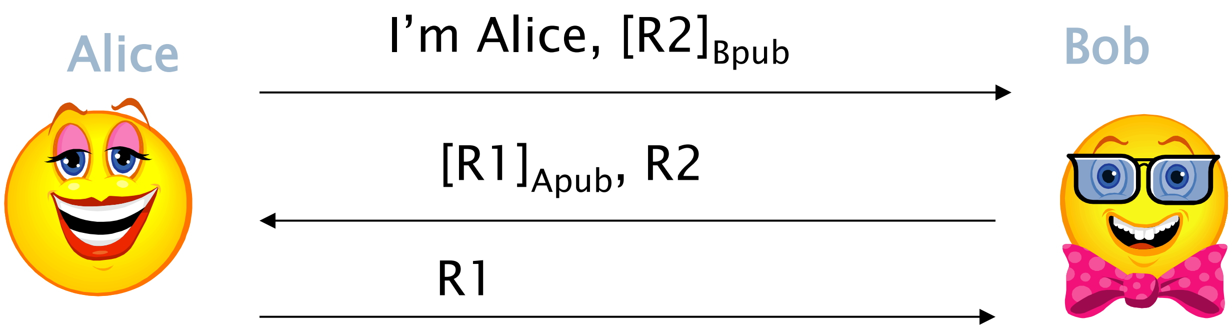
Bob 

$[R]_{Ax}$ = R signed with Alice's x key, where x is private (priv) or public (pub) key

One-way Problems

- First case:
  - Can send anything to Alice as R and get Alice to sign it
- Second case:
  - Intercepted an encrypted message for Alice, send it and get Alice to decrypt it

Mutual Authentication with Public Keys



- Always the same issue!
  - how to obtain/store/validate Bob’s public key

Ake based on DH: Station-to-station protocol

g^amod p||CertA

g^bmod p||CertB||Sig_B(Alice|| g^b ||g^a)

Sig_A(Bob||g^a|| g^b)

K = g^{ab}

K = g^{ab}

Key length

- Difference between symmetric and public key cryptography
- ❑ Symmetric key: best attack (must be) exhaustive search
- ❑ Public key: more efficient attacks due to the mathematical algorithms

  - Several reports exist with recommendations: (www.keylength.com)

- Lenstra and Verheul Equations (2000)
- Lenstra Updated Equations (2004)
- ECRYPT-CSA Recommendations (2018)
- NIST Recommendations (2016)
- ANSSI Recommendations (2014)
- IAD-NSA CNSA Suite (2016)
- Network Working Group RFC3766 (2004)
- BSI Recommendations (2018)

Key-size Equivalence

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512


Questions?