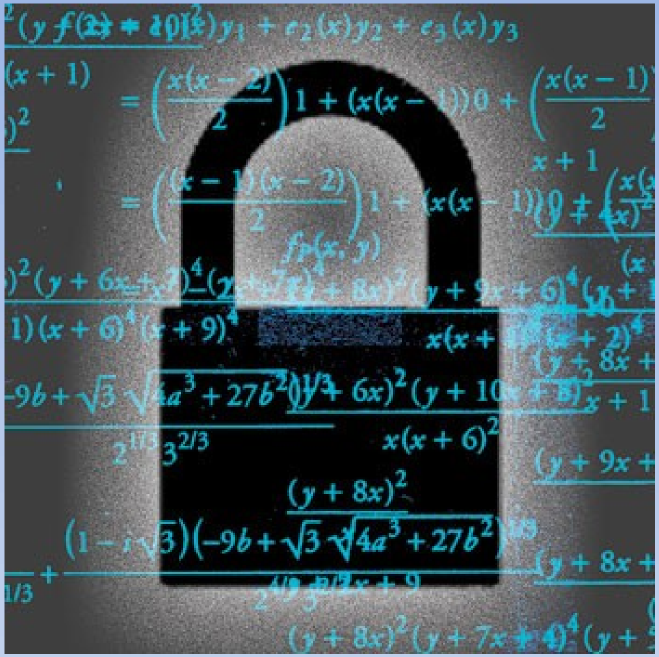


Cryptography
Lecture 8

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

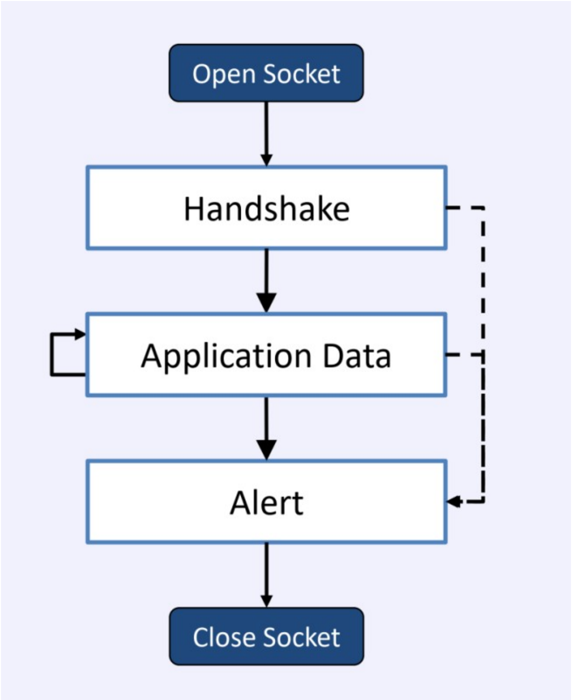
TLS 1.3 (example)

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

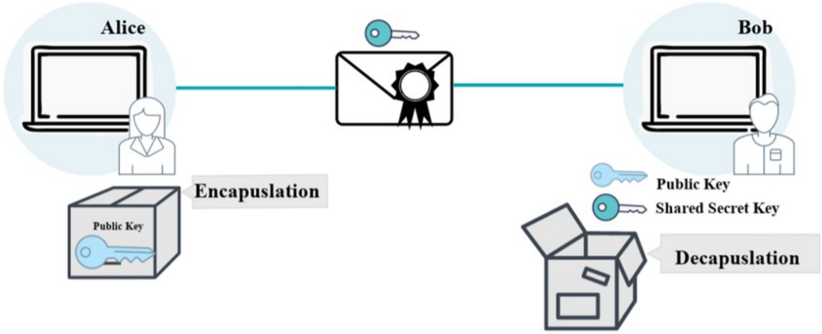


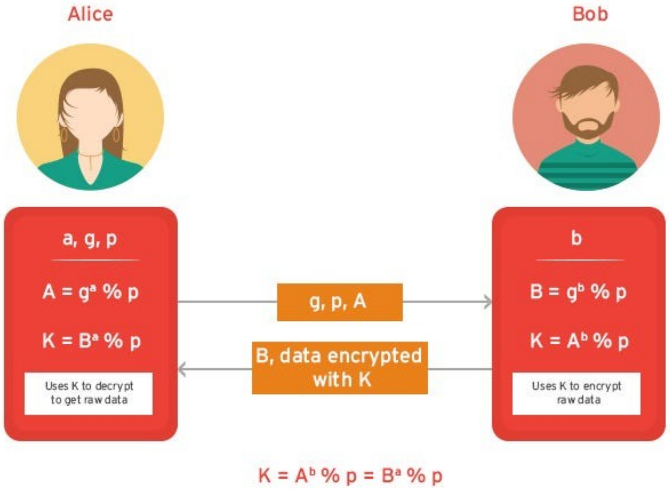
(OWASP presentation)

KEY 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

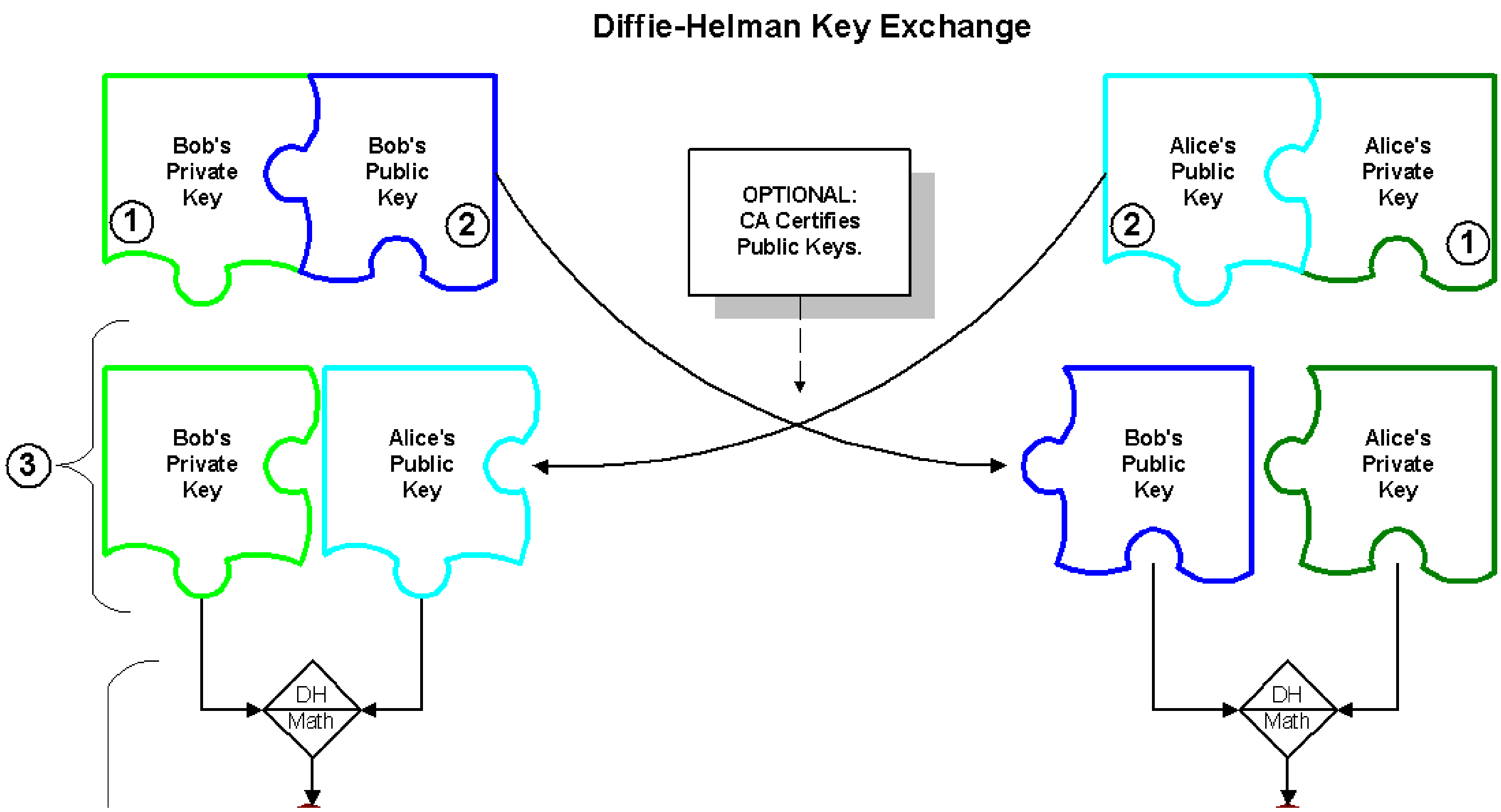
Key Encapsulation Mechanism (KEM)

- 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

$x = g^a \bmod p$ →

← $y = g^b \bmod p$

Bob

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

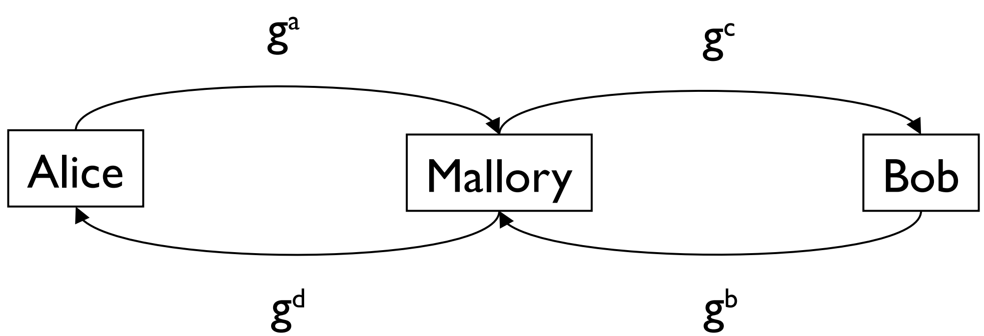
$K = k_b = x^b \bmod p$

- 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

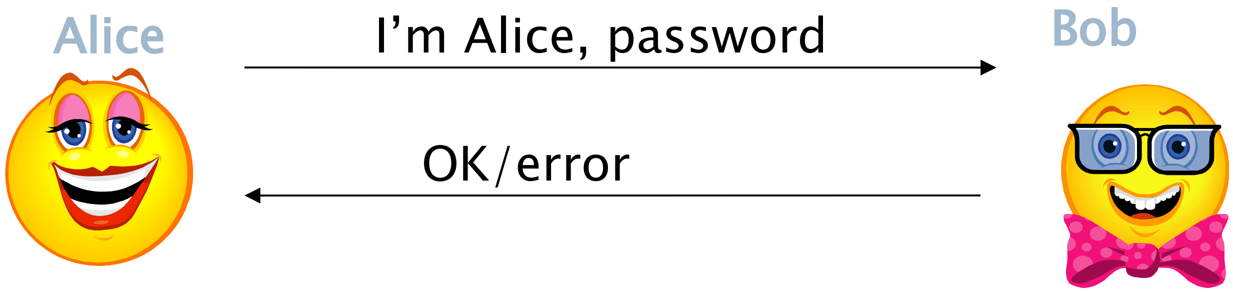
- Authentication:
  - Use public key encryption (and usually certificates)
  - Use pre-shared keys (like passwords)

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

Authentications

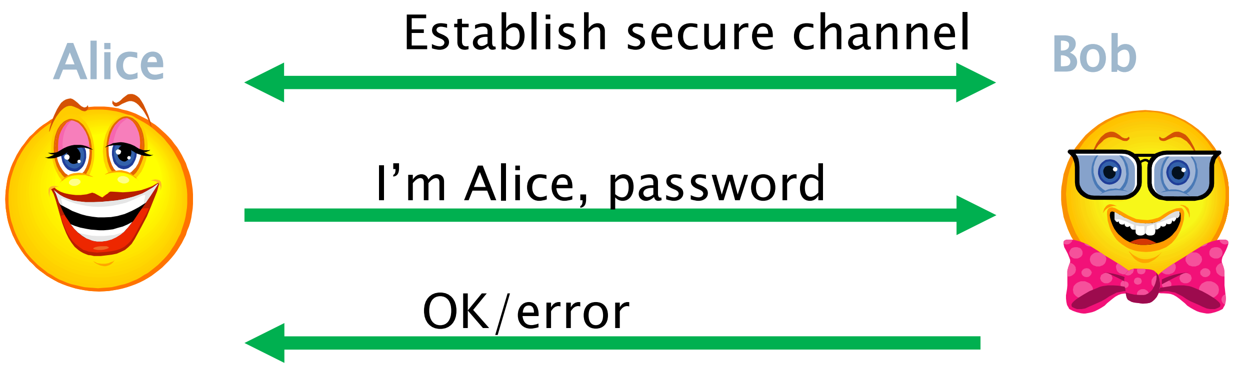
- Use public key encryption (and usually certificates)
- Use pre-shared keys (like passwords or master key of the last session)

Simple Transmission (PSK)

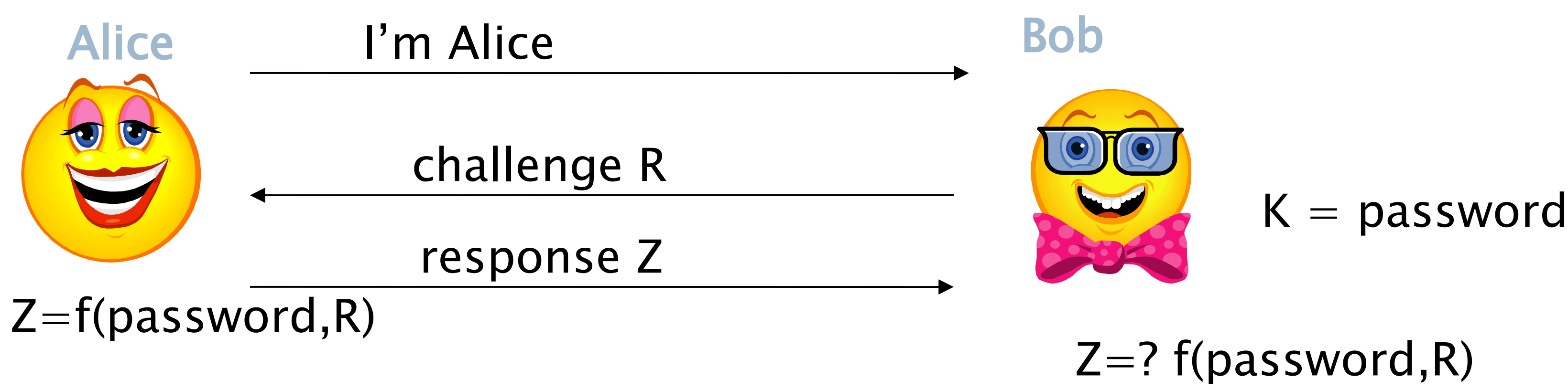


- Insecure!
- Can be easily eavesdroped

Secure simple Transmission (PSK)



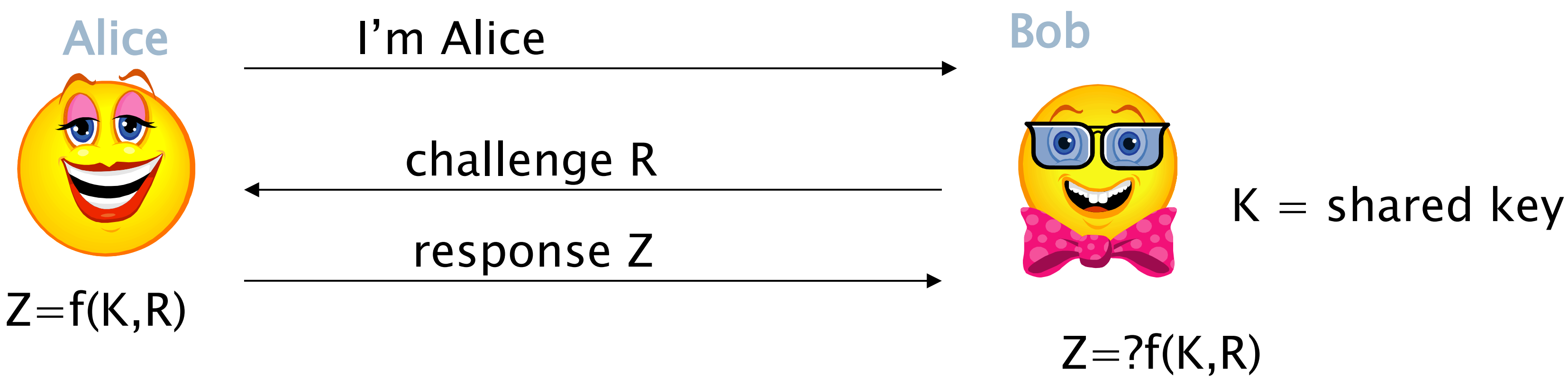
One-way Challenge-Response



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

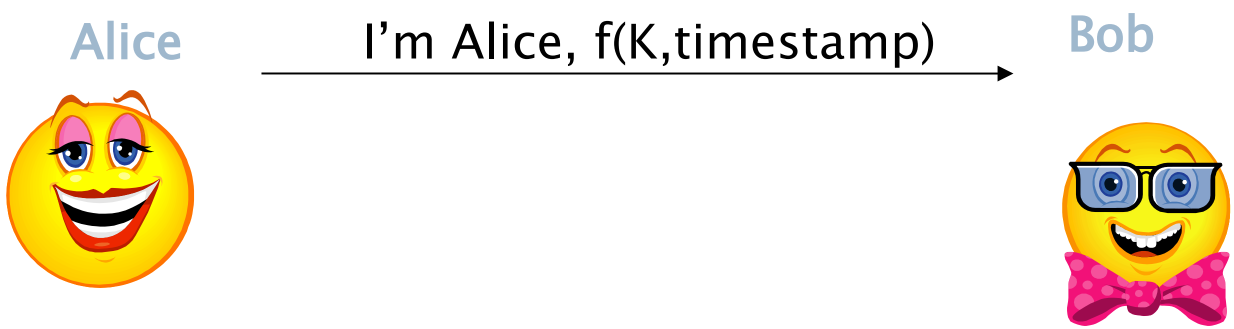
One-way Challenge-Response (PSK)



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)

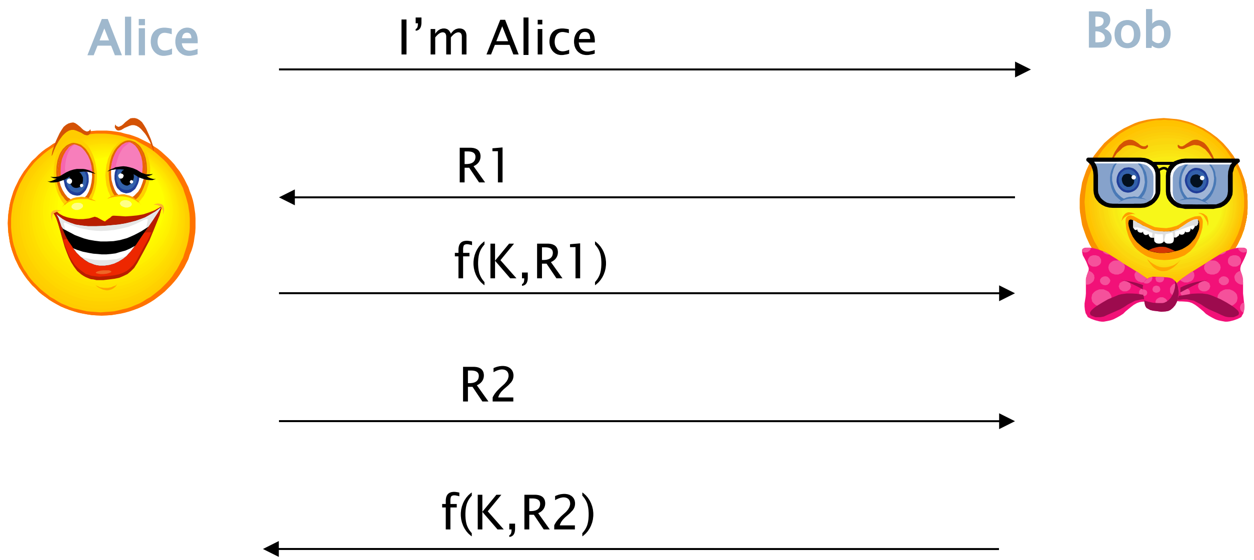


- Problems?
  - Impersonate Alice if intercept and send message – race condition
  - If use same K with multiple servers, could send message to another server and impersonate Alice
  - Clock skew/synchronization

2-Way Authentication

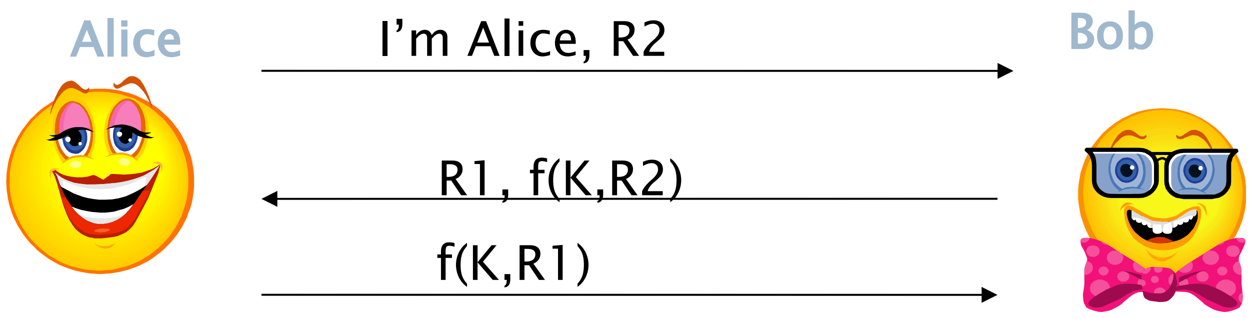
- Authentication often needed in both directions
- Server trusting user is not only concern
  - User must trust server
  - Ex. User accessing online bank account

Mutual Authentication with Secret Key



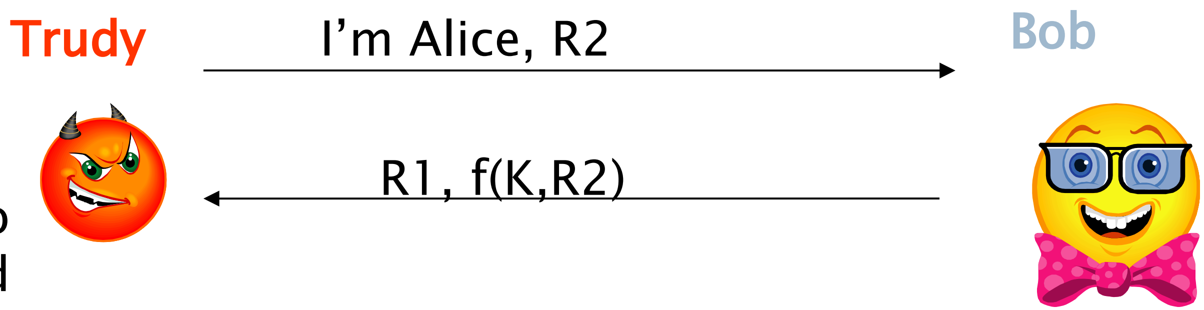
Mutual Authentication with Secret Key

More efficient version:

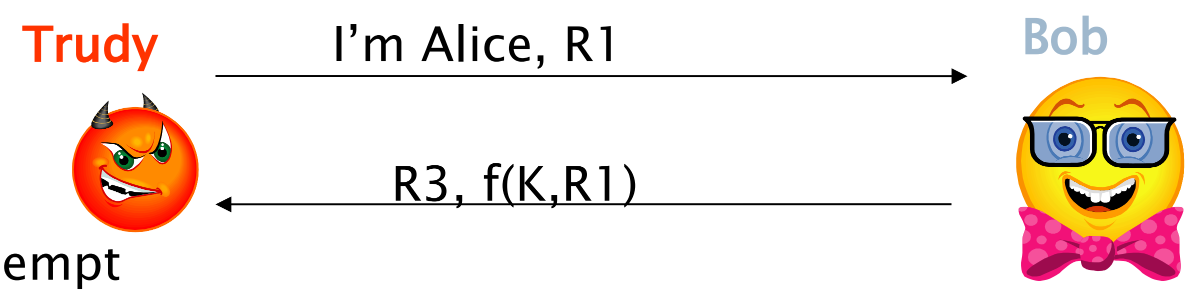


Mutual Authentication with Secret Key

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

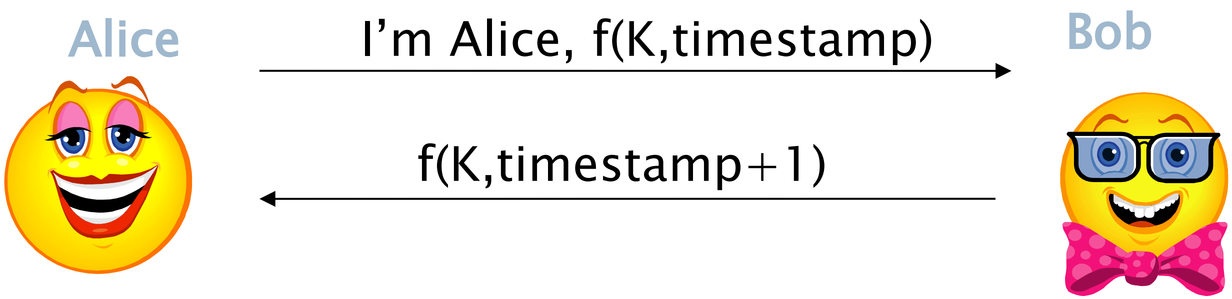
- Solutions:
  - Separate keys for each direction/different passwords
  - Requirements on R values: odd in one direction, even in the other, concatenate with senders' name

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

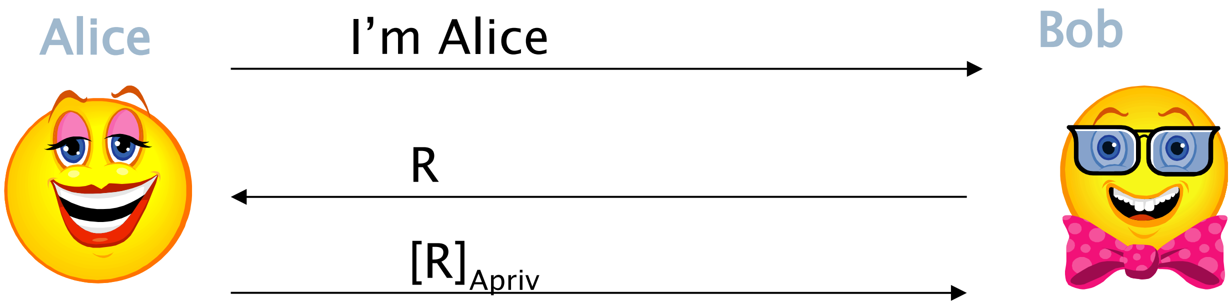


- Same issues as before plus clock skew
- Any modification to timestamp will work

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



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

One-way Using Public Key

Alice

I’m Alice

Bob

R

$[R]_{Apriv}$

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

Alice

I’m Alice

Bob

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

$[R]_{Apub}$

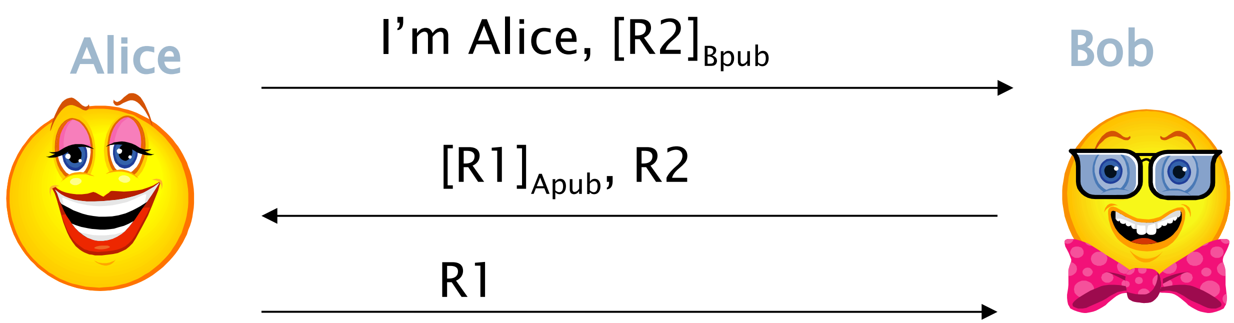
R

$[R]_{Ax}$ = 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) ←

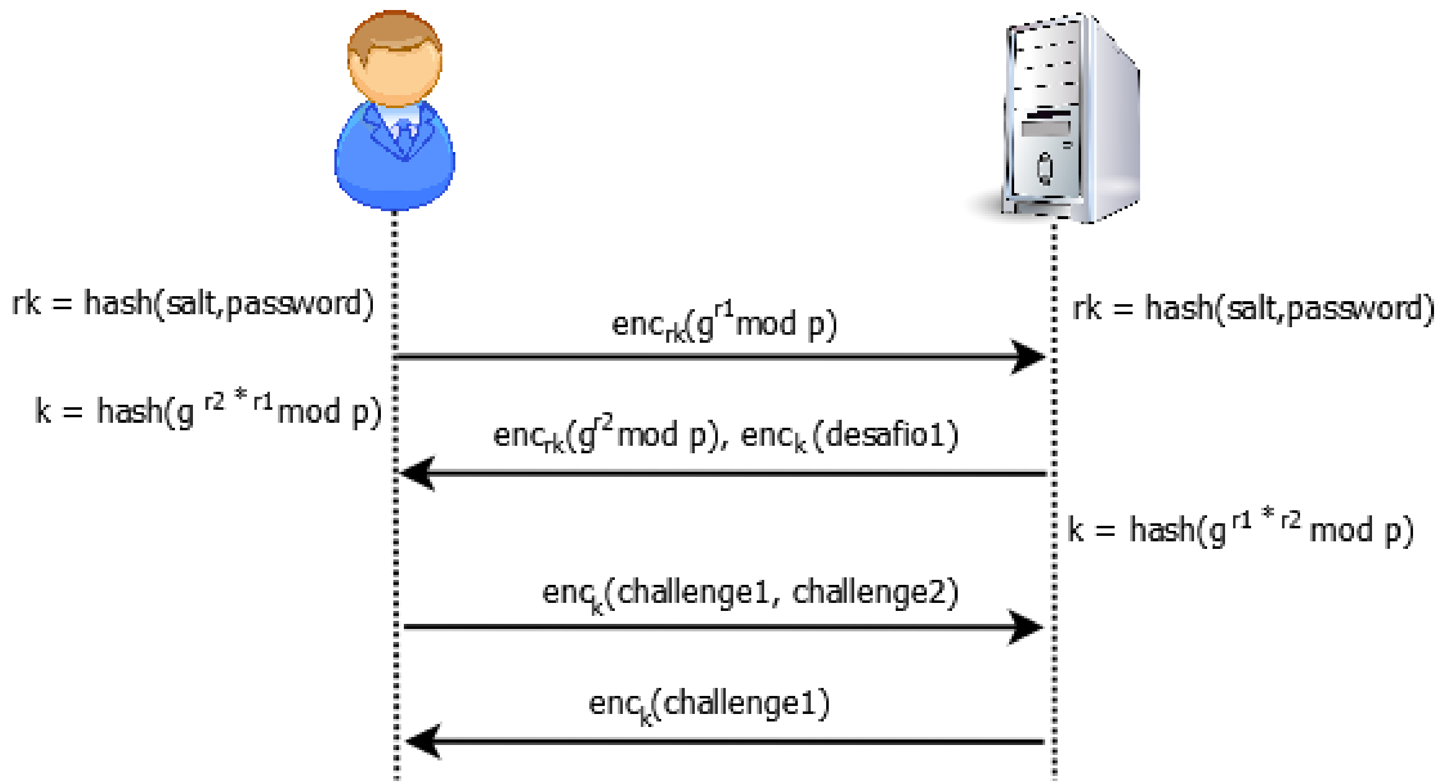
Sig_A(Bob||g^a|| g^b) →

K = g^{ab}

K = g^{ab}

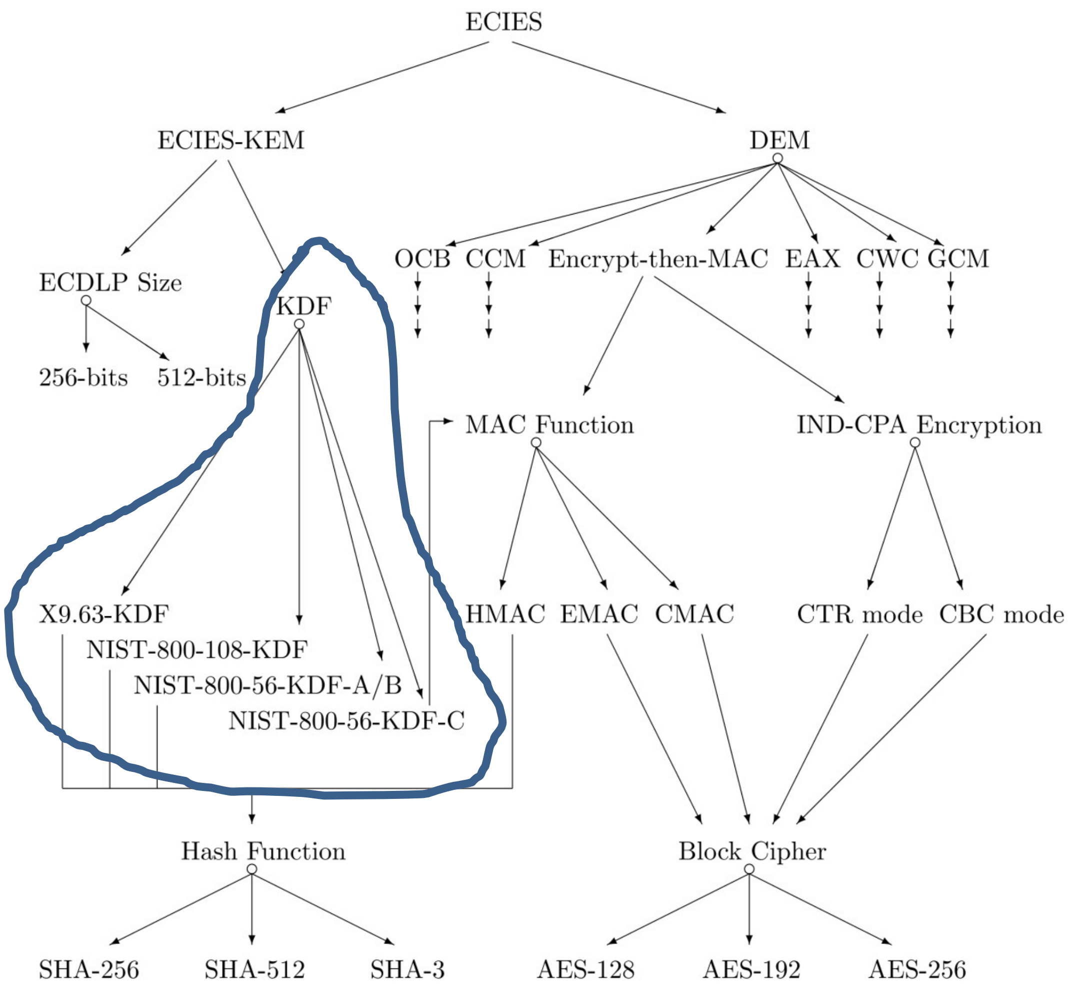
Password-based Authenticated Key Exchange (PAKE)

- 1992, Bellovin and Merritt
- Encrypted Key Exchange (EKE)



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) and from an entropy source (in the case of key generation)
2. from passwords

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

Deriving many keys from one

Typical scenario. a single <u>source key</u> (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

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, (**CTX II 0**)) II F(SK, (**CTX II 1**)) II ⋯ II F(SK, (**CTX II L**))

CTX: a string that uniquely identifies the application

KDF(SK, CTX, L) :=

F(SK, (**CTX II 0**)) II F(SK, (**CTX II 1**)) II ··· II F(SK, (**CTX II 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

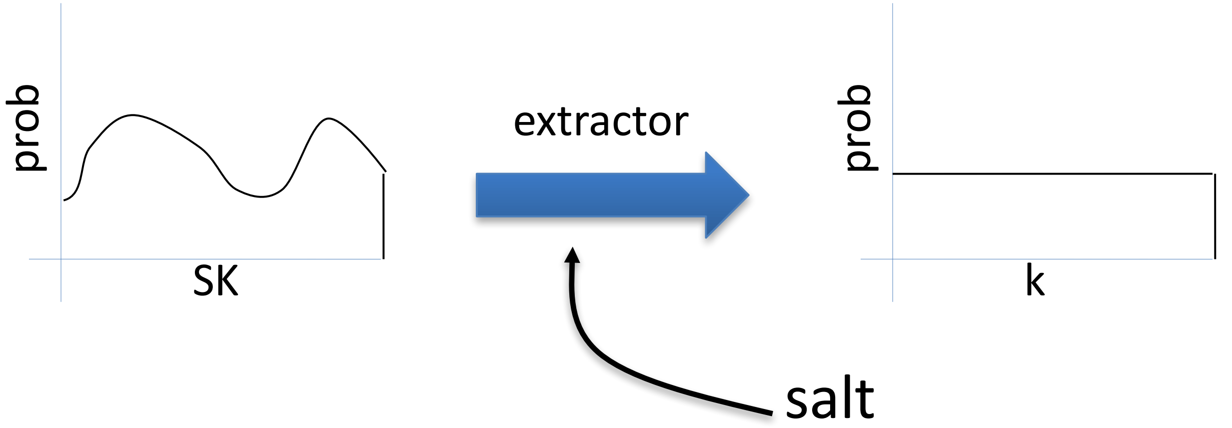
Source key often not uniformly random:

- 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

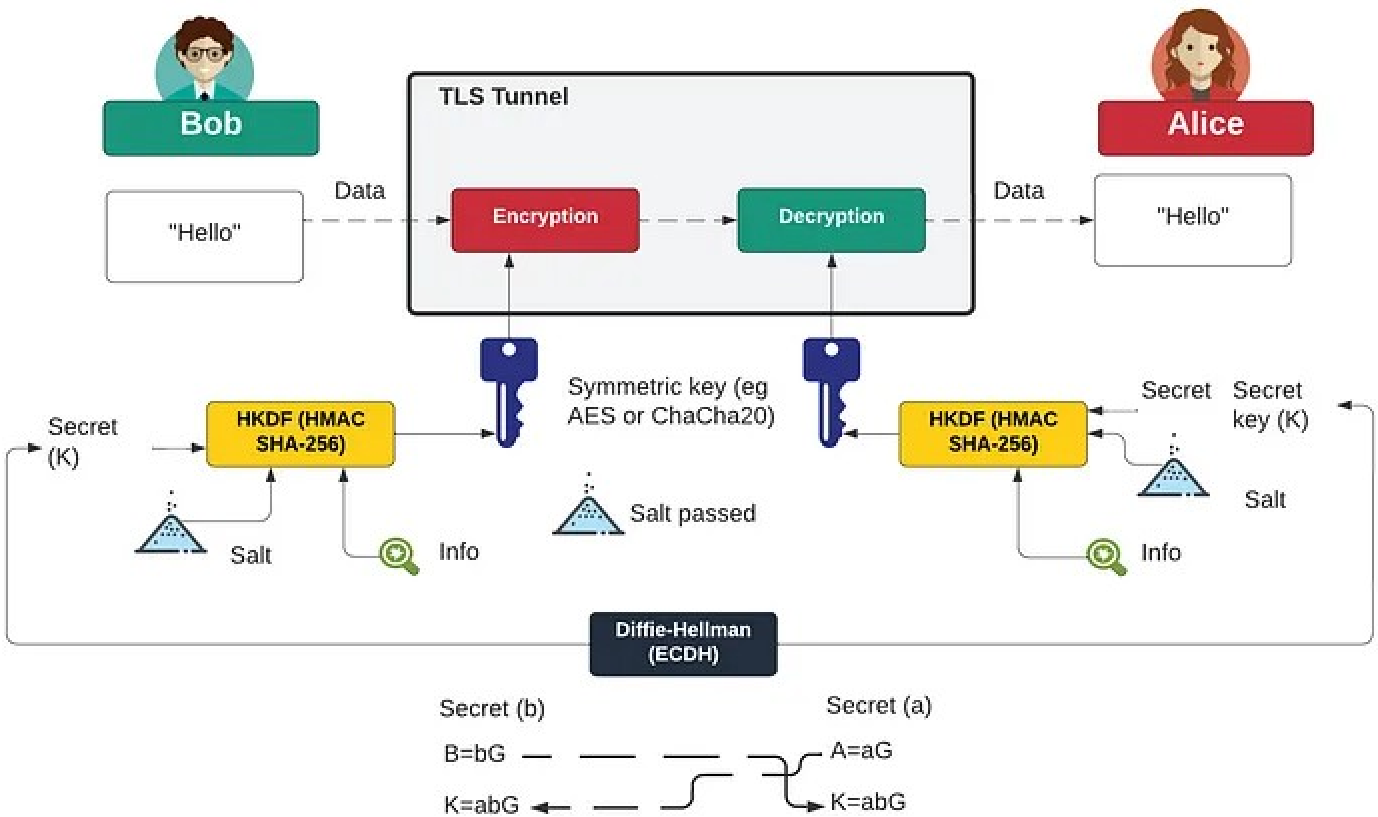
HKDF: a KDF from HMAC

Implements the extract-then-expand paradigm:

- extract: use **k = 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)

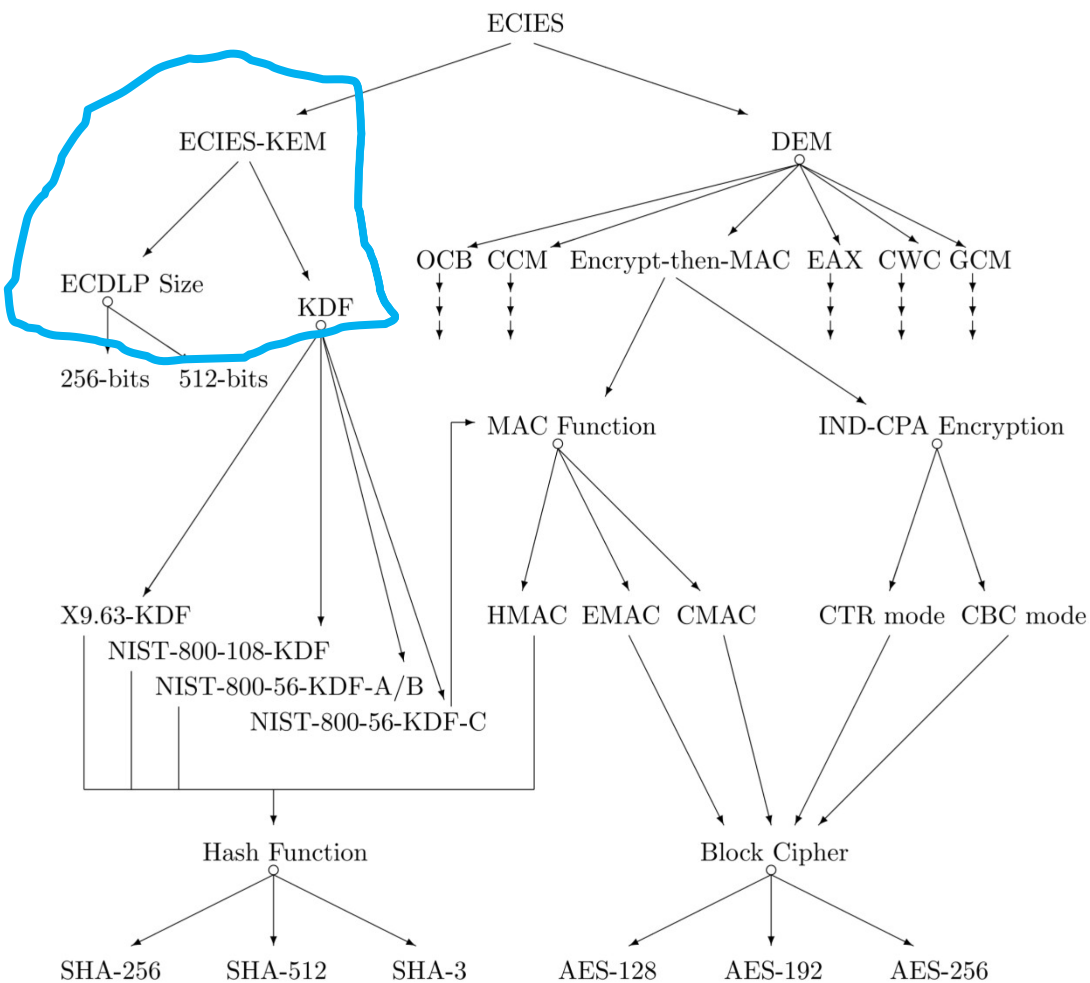
$H^{(c)}$(pwd ll salt): iterate hash function c times

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

Overview



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

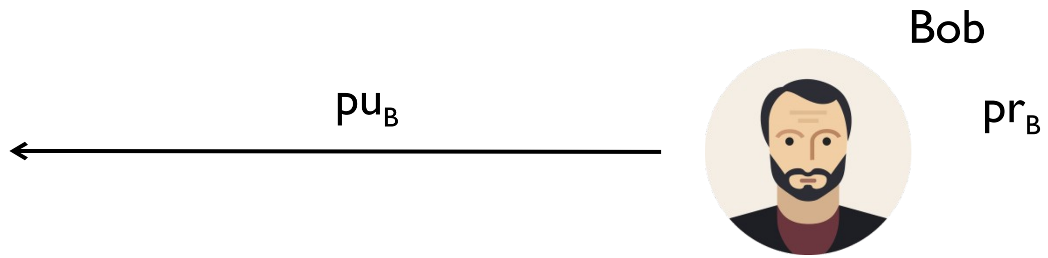
ECIES

- EC Integrated Encryption Scheme (ECIES)
- KEM = Key Encapsulation Mechanism
- DEM = Data Encapsulation Mechanism

- ECIES = ECIES-KEM + DEM

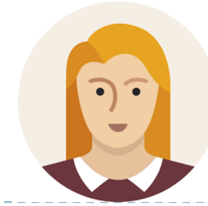
ECIES

- Bob generates public/private keys (EC) DH.
- Bob publishes public key.



ECIES

Alice



- Ephemeral key pair (pr_A, pu_A) (EC)DH
- Compute common key

$$K = ECDH(pr_A, pu_B)$$

- Compute Session Key

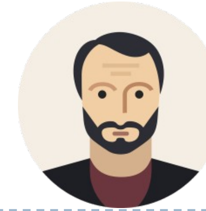
$$K' = KDF(salt, info, K)$$

- Use K' to protect confidentiality and integrity of message M.
- Send to Bob
  - (C, tag, pu_A, other aux info)

ECIES

Bob

pr_B



- Compute common key

$$K = ECDH(pr_B, pu_A)$$

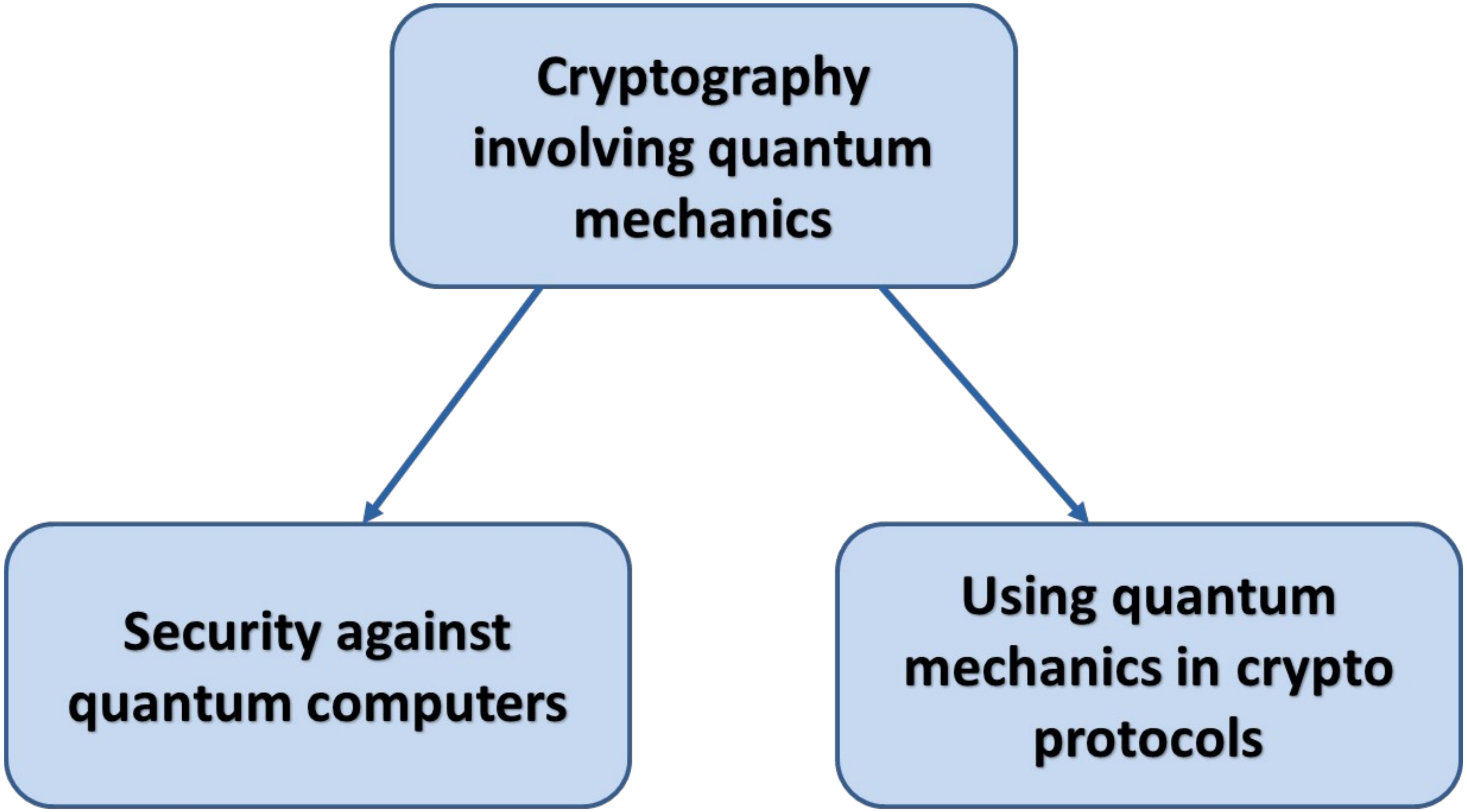
- Compute Session Key

$$K' = KDF(salt, info, K)$$

- Use K’ to verify integrity and decrypt C ot retrieve message M.

QUANTUM KEY DISTRIBUTION

quantum cryptography



Quantum Key distribution

- Symmetric key
- It is based on quantum mechanics
- Two physically separated parties can create and share random secret keys
  - Allows them to verify that the key has not been intercepted.

- Establish an unconditionally secure communication channel

1. Quantum Key distribution
2. Switch to one-time-pad

Basic Idea

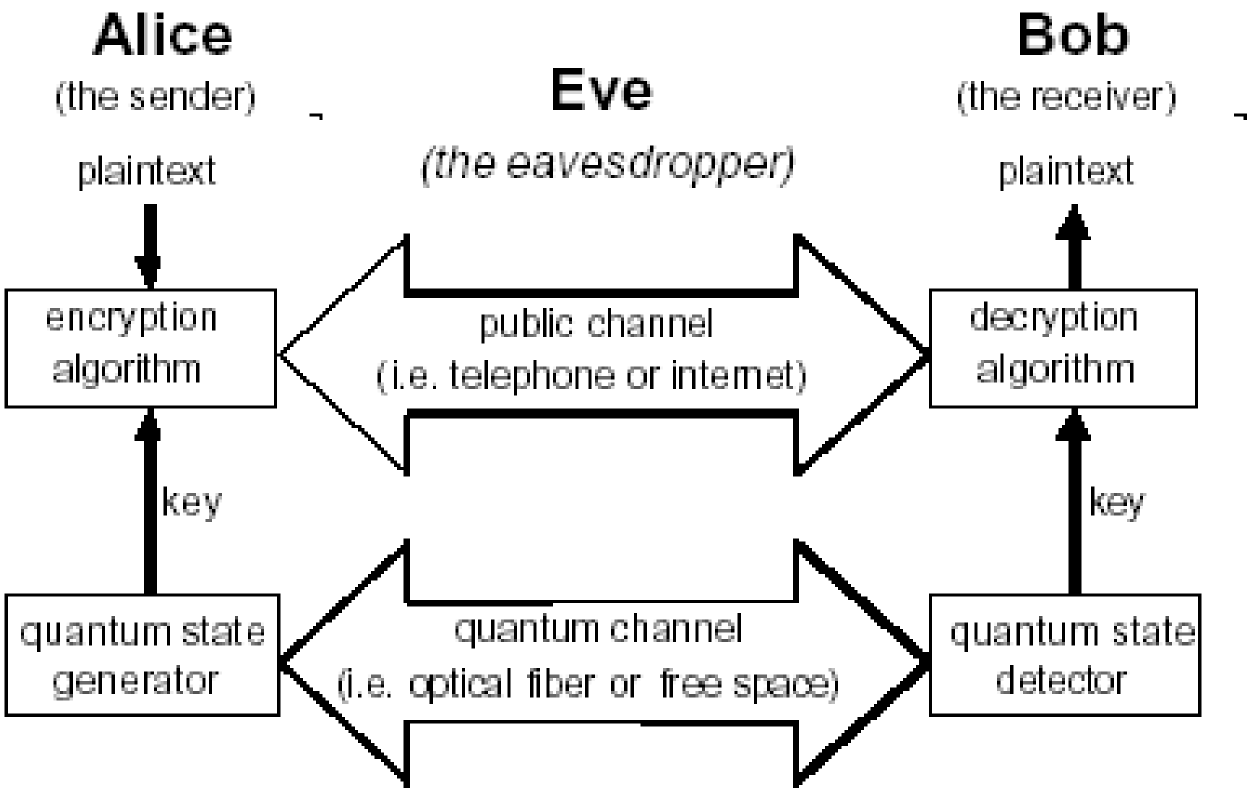


Figure 1. Quantum Key Distribution.

fundamentals

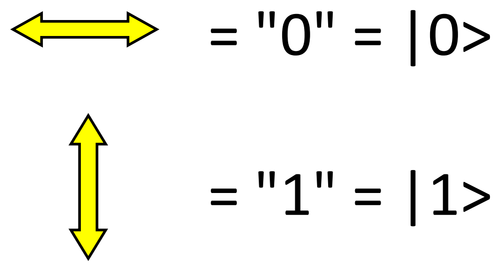
- Measurement causes perturbation

- No Cloning Theorem
- An unknown quantum state CANNOT be cloned. Therefore, eavesdropper, Eve, cannot have the same information as Bob.
- Single-photon signals are secure.

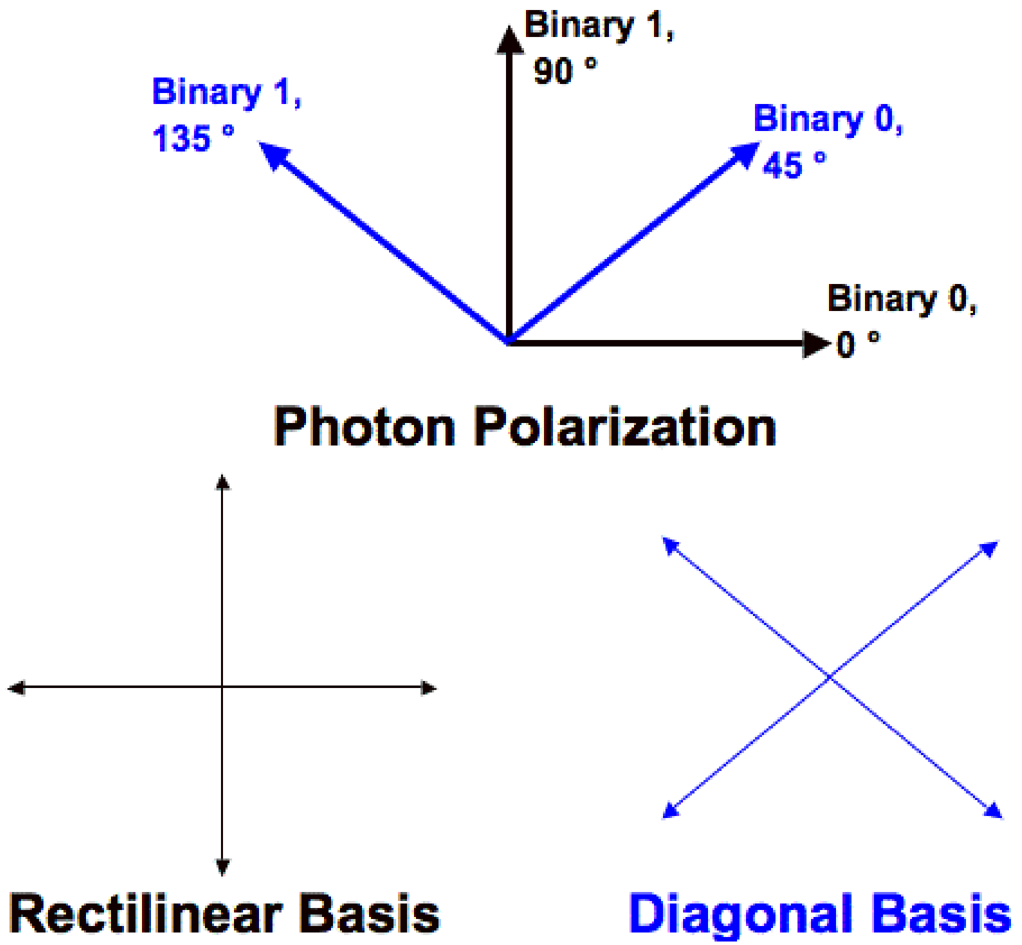
- Thus, measuring the qubit in the wrong basis destroys the information

Quantum communications

- Transmitting information with a single-photon
- Use a quantum property to carry information



Two basis


Binary 1,
90 °
Binary 1,
135 °
Binary 0,
45 °
Binary 0,
0 °
Photon Polarization
Rectilinear Basis
Diagonal Basis

BB84 - Set-up

- Paper by Charles Bennett and Gilles Brassard in 1984 is the basis for QKD protocol BB84. Prototype developed in 1991.

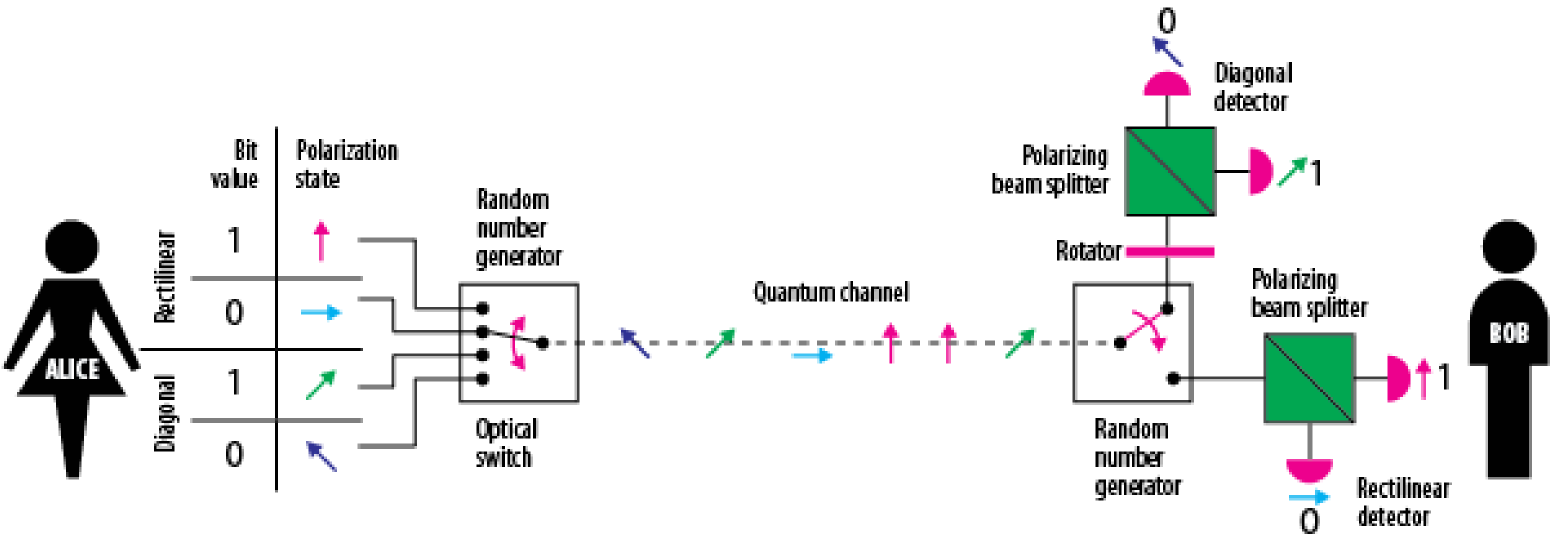
- Alice

Has the ability to create qubits in two orthogonal bases

- Bob

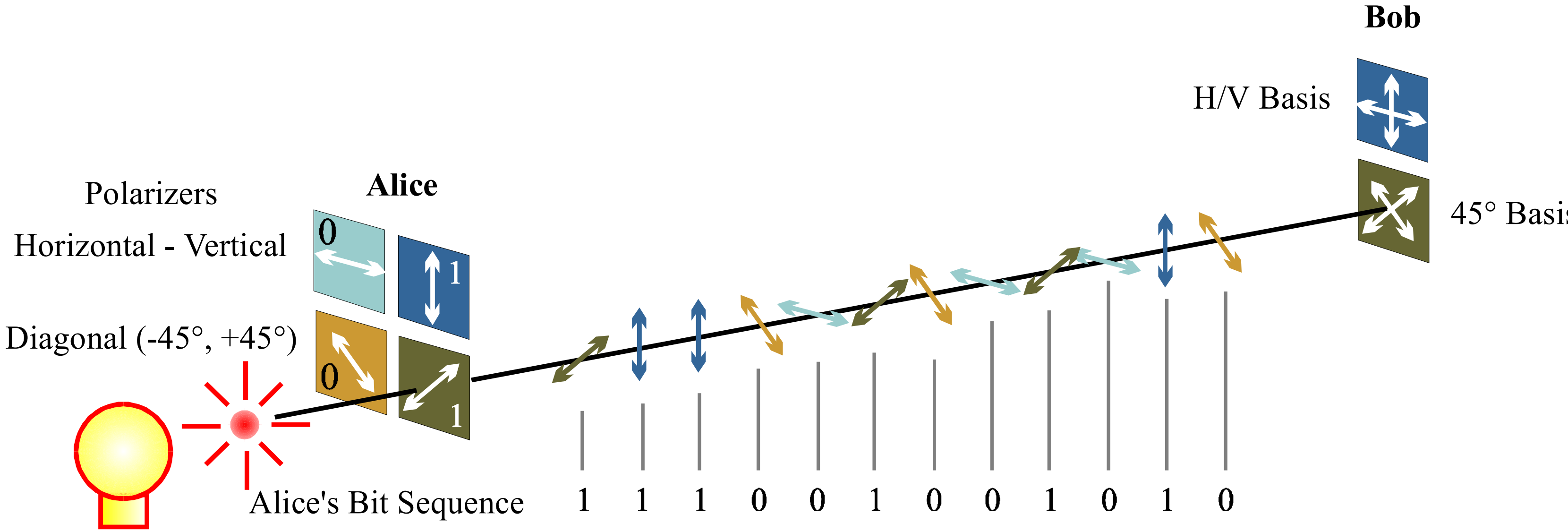
Has the ability to measure qubits in those two bases.

BB84



ALICE
Bit value
Polarization state
Rectilinear
Diagonal
1
0
1
0
Random number generator
Optical switch
Quantum channel
0
Diagonal detector
Polarizing beam splitter
1
Rotator
Polarizing beam splitter
Random number generator
1
Rectilinear detector
0
BOB

BB84



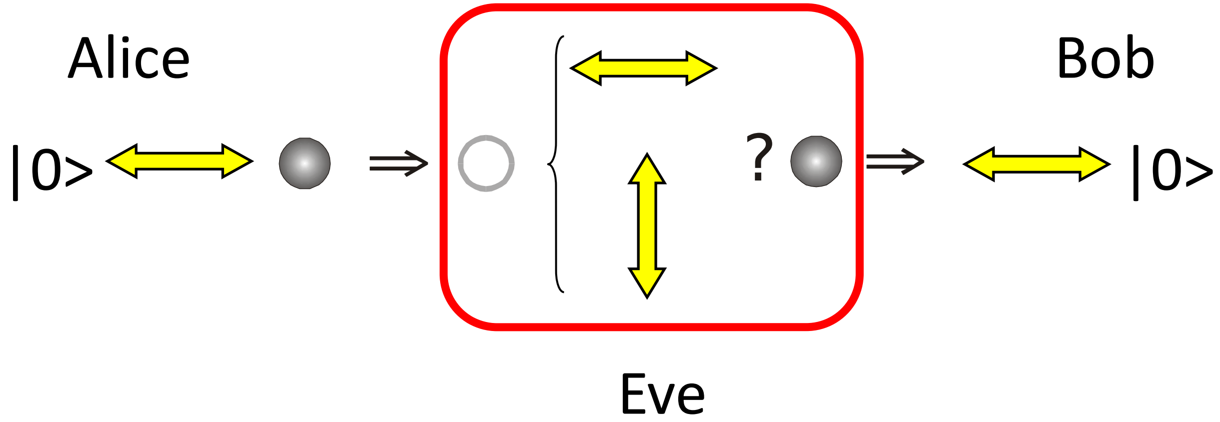
Alice's Bit Sequence	1	1	1	0	0	1	0	0	1	0	1	0
Bob's Bases	+	+	×	×	+	×	+	×	×	+	+	×
Bob's Results	0	1	0	-	0	1	1	1	1	-	1	0
Key	-	1	-	-	0	1	-	-	1	-	1	0

Example

Alice's bit	0	1	1	0	1	0	0	1
Alice's basis	+	+	X	+	X	X	X	+
Alice's polarization	↑	→	↖	↑	↖	↗	↗	→
Bob's basis	+	X	X	X	+	X	+	+
Bob's measurement	↑	↗	↖	↗	→	↗	→	→
Public discussion								
Shared Secret key	0		1			0		1

Eavesdropping

- Communication interception



- Use quantum physics to force spy to introduce errors in the communication
- The errors are detected

ASSSUMPTIONS

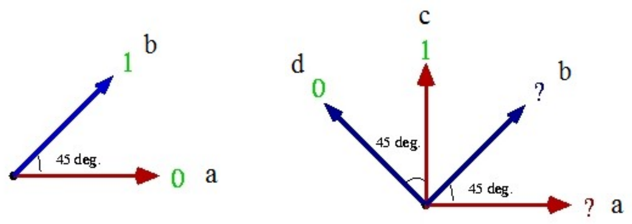
- Source: Emits perfect single photons. (No multi-photons)
- Channel: noisy but lossless. (No absorption in channel)
- Detectors: Perfect detection efficiency. (100 %)
- Basis Alignment: Perfect. (Angle between X and Z basis is exactly 45 degrees.)

- Conclusion: QKD is secure in theory.
- (Assumptions lead to security proofs)

Other schemes

- EPR
- Uses entangled qubits sent from a central source
- Alice and Bob measure qubits randomly and independently
- After measuring, they compare measurement bases and proceed as in BB84
- Advantage over BB84 is that Eve can now be detected using rejected qubits

- B92
- Uses only two non-orthogonal states
- Each bit is either successfully
- received or an “erasure”



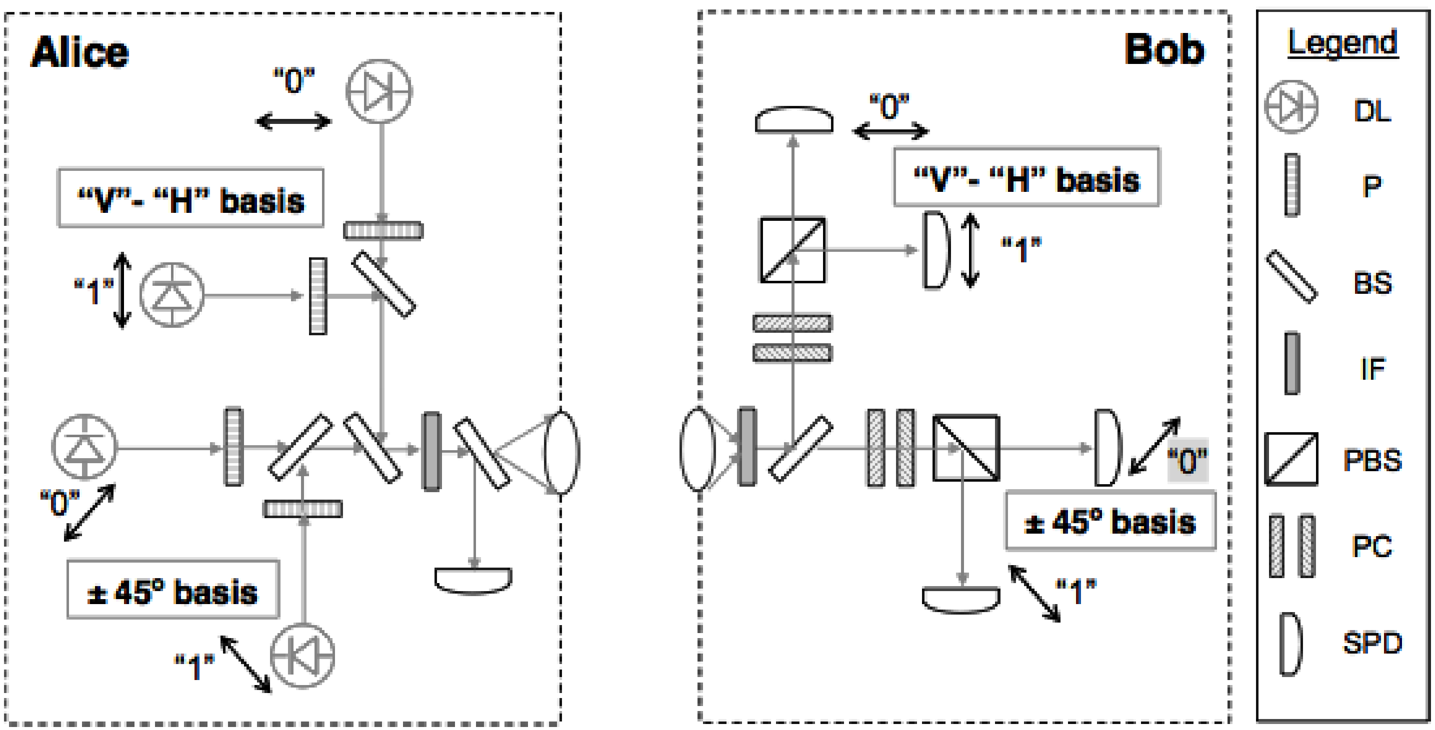
Current State of Affairs

- Commercial quantum key distribution products exist

- Current fiber-based distance
- record: 200 km

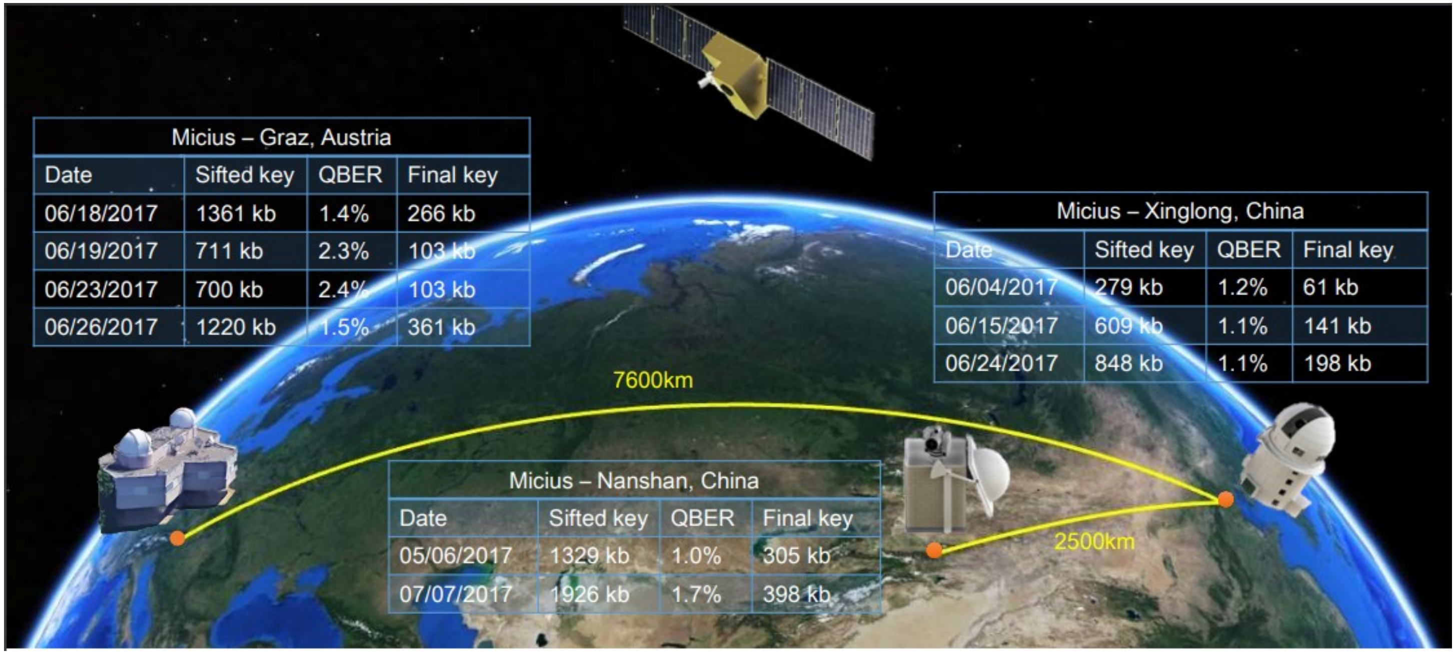
Current State of Affairs

- Demonstrated free-space link: 10 km



Satellite-to-ground quantum key distribution

- Micius satellite
- Use QKD and symmetric encyrption
- ESA signed a contract with SES Techcom S.A. (LU) to develop the Quantum Cryptography Telecommunication System (QUARTZ)



KEY LENGTH

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)

Minimum symmetric key-size in bits for various attackers

Attacker	Budget	Hardware	Min security
"Hacker"	0	PC	58
	$< \$400$	PC(s)/FPGA	63
	0	"Malware"	77
Small organization	\$10k	PC(s)/FPGA	69
Medium organization	\$300k	FPGA/ASIC	69
Large organization	\$10M	FPGA/ASIC	78
Intelligence agency	\$300M	ASIC	84

ECRYPT II, Yearly Report on Algorithms and Keysizes (2011-2012)

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

Security levels (symmetric equivalent)

Security Level	Security (bits)	Protection	Comment
1.	32	Attacks in "real-time" by individuals	Only acceptable for auth. tag size
2.	64	Very short-term protection against small organizations	Should not be used for confidentiality in new systems
3.	72	Short-term protection against medium organizations, medium-term protection against small organizations	
4.	80	Very short-term protection against agencies, long-term prot. against small organizations	Smallest general-purpose level, $\leq$ 4 years protection (E.g. use of 2-key 3DES, $< 2^{40}$ plaintext/ciphertexts)
5.	96	Legacy standard level	2-key 3DES restricted to $\sim 10^6$ plaintext/ciphertexts, $\approx$ 10 years protection
6.	112	Medium-term protection	$\approx$ 20 years protection (E.g. 3-key 3DES)
7.	128	Long-term protection	Good, generic application-indep. recommendation, $\approx$ 30 years
8.	256	"Foreseeable future"	Good protection against quantum computers unless Shor's algorithm applies.

ECRYPT II, Yearly Report on Algorithms and Keysizes (2011-2012)

Key length

- Keys are getting older (with use)

✓ There are time/memory/pre-processing (generic) attacks.

✓ Based on the birthday paradox

✓ Use session keys

No. of keys (Data) 2^d	Time 2^t	Memory 2^m	Pre-processing 2^p
$2^{n/4}$	$2^{n/2}$	$2^{n/2}$	$2^{3n/4}$
$2^{n/3}$	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$
$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$

ECRYPT II, Yearly Report on Algorithms and Keysizes (2011-2012)

❑ There is an attack against AES-128. It has only 85-bit security

- if 2^{43} encryptions of an (arbitrary) fixed text under different keys are available to the attacker.

Key length

k	$\ell(N)$	$\ell(q)$	k	$\ell(N)$	$\ell(q)$	k	$\ell(N)$	$\ell(q)$	k	$\ell(N)$	$\ell(q)$	k	$\ell(N)$	$\ell(q)$
Lenstra–Verheul 2000 [369] ⋆														
80	1184	142	112	3808	200	128	5888	230	192	20160	350	256	46752	474
Lenstra 2004 [366] ⋆														
80	1329	160	112	3154	224	128	4440	256	192	12548	384	256	26268	512
IETF 2004 [444] ⋆														
80	1233	148	112	2448	210	128	3253	242	192	7976	367	256	15489	494
SECG 2009 [520]														
80	1024	160	112	2048	224	128	3072	256	192	7680	384	256	15360	512
NIST 2012 [437]														
80	1024	160	112	2048	224	128	3072	256	192	7680	384	256	15360	512
ECRYPT2 2012 [187]														
80	1248	160	112	2432	224	128	3248	256	192	7936	384	256	15424	512

- *D5.4. Algorithms, Key Size and Protocols Report (2018). ECRYPT-CSA*

Key usage

- Principle of key separation: the cryptographic kerys must only be used for their intended purpose.
- For example to use a symmetric AES key as both the key to an application of AES in an encryption scheme, and also for the use of AES within a MAC scheme
- Using an RSA private key as both a decryption key and as a key to generate RSA signatures.
- Use the same encryption key on a symmetric channel between Alice and Bob for two way communication (as opposed to two unidirectional keys).
- Such usage can often lead to unexpected system behavior.
- It must be well investigated

- It is difficult to enforce the principle

Key deletion/backup/archive

- Key backup
  - Critical operation
  - If you lose the key, you lose encrypted data
  - Key escrow

- Key archival
  - Special type of backup
  - Usually a legal requirement
  - Public keys to verify signatures

- Key deletion
  - Data sanitization
  - Not a trivial task
  - Repeatedly overwriting
  - Follow standards


Questions?

OTHER KEY MANAGEMENT RELATED ISSUES

What is the best we can hope foR

1. The primitive is solid
2. The algorithm and the protocol are secure
3. The implementation flawless

- Then, it is all about the secret keys.

- Manage the circle of life of a key
- (generate the key, establish, use, store, delete/archive)

- Much more difficult than it sounds!!

Key management

- ✓ Secure design and implementation
- ✓ We are shifting the problem
- ✓ The key can be seen as special type of data

- Secure administration of (cryptographic) keys

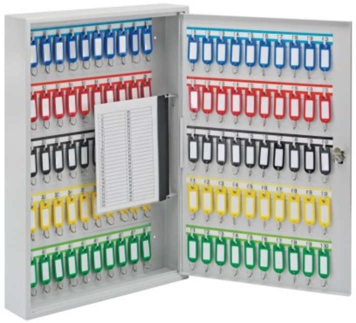
- Standards
- NIST Special Publication 800-57. Recommendation for key management, Part 1: General (Revision 3). National Institute of Standards and Technology, 2012
- NIST Special Publication 800-130. A framework for designing cryptographic key management systems. National Institute of Standards and Technology, 2013.

Key lifecycle

1. Key Generation
2. Key Registration/Certification
3. Key Distribution and Installation
4. Key Storage and backup
5. Key Use
6. Revocation/Change
7. Key Archive/Destruction

Key storage

- Store a password, symmetric key, private part of the public-key pair.



- ➢ No storage
- ➢ In the clear
- ➢ In hardware
- ➢ Encrypted
- Store locally
- Use a web/cloud based secret manager
- ➢ Off-line (pen and paper)

Key storage - Secure hardware

- ✓ Secure tokens (eg. Smart cards)

- ✓ Hardware Security Module (HSM)

- ➢ FIPS PUB 140-2. SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES
- Four levels of security
- ➢ FIPS 140-3 was approved on March 22, 2019 and will become effective on September 22, 2019
- ➢ NIST maintains a DB of validated cryptographic modeules

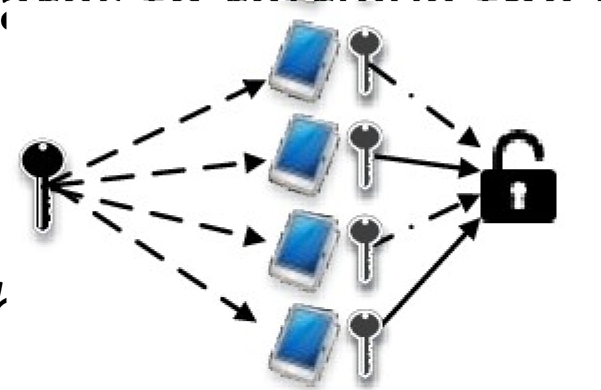
Key storage - Key wrapping

- Use a block cipher to encrypt a secret key.

Scheme	Categorisation Legacy	Categorisation Future
KW	✓	✗
TKW	✓	✗
KWP	✓	✗
AESKW	✓	✗
TDKW	✓	✗
AKW1	✓	✗
AKW2	✗	✗
SIV	✓	✓

Key storage - Secret sharing schemes

- Main concept:
  - ➢ Produce shares from the secret
  - ➢ Use distributed storage for the shares
  - ➢ Each share looks random (no information leakage)
  - ➢ Delete the secret key
  - ➢ Use the sh[illegible]ecret key when needed



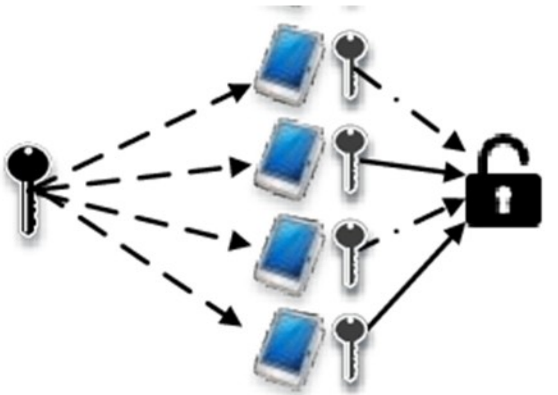
- *key sh*

Threshold scheme

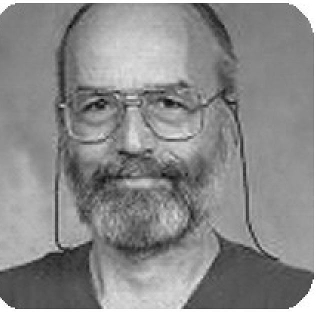
- A (k,n) threshold scheme has the following properties:

✓ From the secret n shares are produced.
✓ Any group of k share owners can reconstruct the secret,
✓ Any a group of (k-1) or less shares cannot!

- Most of the schemes are based on Shamir's scheme.



Shamir threshold scheme



- Invented by Adi Shamir in 1979.
- It is based on the fact that k points uniquely determine a polynomial of degree k-1.

- <u>The algorithm:</u>
- Pick a random polynomial of degree k-1

$$q(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

- where the secret S is the constant term S=q(0)=a_0, and the shares S_i are given by

$$S_1 = q(1), S_2 = q(2), \ldots, S_n = q(n)$$

Shamir Approach (continued)

- ➢ represent each share as a point $(x_i, q(x_i) = y_i)$
- ➢ All arithmetic done modulo a prime number p (integer ring)
- ➢ All the coefficients are randomly chosen from a uniform distribution over the integers in [0,p)

$$P(x) = \sum_{i=1}^{k} y_i \prod_{j=1, j \neq i}^{k} \frac{x - x_j}{x_i - x_j}$$

- With k shares we reconstruct the polynomial using the Lagrange Interpolation

$$P(x) = y_1 \frac{(x - x_2)(x - x_3)\cdots(x - x_k)}{(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_k)} + y_2 \frac{(x - x_1)(x - x_3)\cdots(x - x_k)}{(x_2 - x_1)(x_2 - x_3)\cdots(x_2 - x_k)} + \cdots + y_k \frac{(x - x_1)(x - x_2)\cdots(x - x_{k-1})}{(x_k - x_1)(x_k - x_2)\cdots(x_k - x_{k-1})}$$

Example: (3,5)threshold scheme

$n = 5$

$k = 3$

$S = 7$

$a_0 = S$

$a_1 = 3$

$a_2 = 5$

$p = 11$

$$q(x) = 5x^2 + 3x + 7 \pmod{11}$$

$$S_1 = q(1) = 5(1)^2 + 3(1) + 7 \pmod{11} \equiv 4$$

$$S_2 = q(2) = 5(2)^2 + 3(2) + 7 \pmod{11} \equiv 0$$

$$S_3 = q(3) = 5(3)^2 + 3(3) + 7 \pmod{11} \equiv 6$$

$$S_4 = q(4) = 5(4)^2 + 3(4) + 7 \pmod{11} \equiv 2$$

$$S_5 = q(5) = 5(5)^2 + 3(5) + 7 \pmod{11} \equiv 4$$

- Using the shares S1, S2, and S4 we have

$$P(x) = [4\frac{(x-2)(x-5)}{(1-2)(1-5)} + 0\frac{(x-1)(x-5)}{(2-1)(2-5)} + 4\frac{(x-1)(x-2)}{(5-1)(5-2)}] \pmod{11}$$

$$P(x) = [(x-2)(x-5) + 4(x-1)(x-2)] \pmod{11} = 5x^2 + 3x + 7 \pmod{11}$$

$$S = P(0) = 7$$

Exercise

1. Let $H:\{0,1\}^* \rightarrow T$ be a collision resistant hash function. Is the following hash function collision resistant?

$$H'(m)=H1(H2(m))$$

Proof (sketch)

- Let n1<n2
- Let H'(m)=H1(H2(m)) and let's assume that H'(m) is not collision resistant. Thus, there is a polynomial algorithm A that can compute a pair of messages m1 and m2, more efficiently than O($2^{n1/2}$), such that:

H'(m1)=H'(m2)

Thus, it holds H1(H2(m1))=H1(H2(m2)). We distinguish two cases:

1. H2(m1)=H2(m2). Then, the algorithm A can compute collisions for H2(m), more efficiently than O($2^{n2/2}$). This is a contradiction.
2. H2(m1)≠H2(m2). Then, the messages y1=H2(m1) and y2=H2(m2)

H1(H2(m1))=H1(H2(m2)) <=> H1(y1)=H1(y2)

are collisions for H1(m). That is that, the algorithm A can compute collisions for H1(m), more efficiently than O($2^{n1/2}$). This is a contradiction.

Proof (sketch)

- Let n1>n2.
- Our goal is to find collisions for H'(m)=H1(H2(m)) more efficiently than $O(2^{n1/2})$.
- Let H2(m1)=H2(m2). We can find them with random trials in $O(2^{n2/2})$.
- Also , H'(m1)=H1(H2(m1))= H1(H2(m2))= H'(m2). Thus, we have a collision for H' more efficiently than than $O(2^{n1/2})$, since n1>n2

- When n1=n2, then it is secure (the same proof as for H(H(m)))

Key lifetime

- A key is valid (can be used) for a specified period of time. When that period has expired, it is either destroyed or archived.

- ❑ Key compromise
- ❑ Future attacks
- ❑ Key exposure
- ❑ Flexibility (key length/key lifetime)
- ❑ Key management failures
- ❑ Key management cycles

Key generation/derivation

- We want to generate/compute
  - ➢ asymmetric key-pairs,
  - ➢ symmetric keys,
  - ➢ initialization vectors (IVs)
  - ➢ Challenge-response protocols

- Generate the key
  - ✓ Hardware based source of randomness
  - ✓ Software based source of randomness

- Key derivation
  - ✓ From other keys
  - ✓ Password based derivation functions

Key generation

- Difficult to find random sources
- Random Number/bit Generators (RNGs) or True Random Number Generators (TRNGs)
- TRNG device
- special-purpose hardware (e.g. electronic circuits, quantum devices)
- post-processing (noise whitening)
- operate at low output rates