# Trusted 3rd parties

Key management. N users → storing mutual secret keys is difficult

Each user would have to store O(n) keys.

Solution: Online Trusted 3rd party (TTP)

The TTP provides a key for both users.

A toy protocol: Eavesdropping security only

Alice wants a shared key with Bob.

1. If Alice wants to establish a communication with Bob she first ask the TTP for a shared key.
2. TTP responds with two messages:
   * E(KA, “A,B”||KAB) ← This is the shared Key encrypted with Alice’s key
   * E(KB, “A,B”||KAB) ← This is the ticket, just the shared Key encrypted with Bob’s key
3. When Alice wants to start the communication she first sends the ticket to Bob
4. Bob decrypts the ticket and gets the shared key
5. At this point both Alice and Bob have the shared key and they can use it to communicate securely which one another

(E,D) is a CPA-secure cipher

This mechanism is eavesdropping secure because the eavesdropper sees:

E(kA, “A,B”||kAB), E(KB, “A,B”||kAB)

As (E,D) is CPA-secure the eavesdropper learns nothing about the keys. For him the messages is indistinguishable from junk messages.

The TTP needed for every key exchange and knows all session keys.

The usability of this mechanism depends on the context being used. For example for WWW could be difficult to determine who will be the TTP, but inside a company it is feasible to designate a machine as a TTP and use it as a single point of trust.

This is a very simple shared key protocol. It is secure against eavesdropping, but it is insecure against replay attacks. It should never be used on the real world.

The alternative to a TTP is public-key cryptography:

* Merkle (1974)
* Diffie-Hellman (1976)
* RSA (1977)

Modern ideas:

* ID-based encryption (2001)
* Functional encryption (2011)

# Merkle puzzles

Goal: Alice and Bob want shared key, unknown to eavesdropper

For now: security against to eavesdropping (no tampering)

This can be accomplished with symmetric encryption, but very inefficiently

Main tool: puzzles

Puzzle: a problem that can be solved with some effort

Example: E(k,m) a symmetric cipher with k in {0,1}128

- puzzle(P) = E(P,”message”) where P = 096 || b1...b32

- Goal: find P br trying all 232 possibilities

**Alice**: prepare 232 puzzles

* For i = 1,...,232 choose random Pi in {0,1}32 and xi,ki in {0,1}128 set

puzzlei ← E( 096 || Pi , “Puzzle # xi” || ki )

* Send puzzle1..puzzle2^32 to Bob

**Bob**: choose a random puzzlej and solve it. Obtain (xj, kj).

* Send xj to Alice

**Alice**: lookup puzzle with number xj. Use kj as shared key

Alice’s work: O(n) → prepare n puzzles

Bob’s work: O(n) → solve one puzzle

Eavesdropper’s work: O(n2) → solve n puzzles

It is unknown if this quadratic gap can be improved using symmetric cipher.

It have been proved that if the participants treat the cipher as a black box oracle, and don’t take into consideration the implementation of the cipher, the quadratic gap is the best possible solution (and there always be a quadratic order attack).

# The Diffie-Hellman protocol

The goal is to achieve an exponential gap.

The Diffie-Hellman protocol informally:

Fix a large prime p (e.g. 600 digits)

Fix an integer g in {1,...,p}

**Alice**: choose a random a in {1,...,p-1}

**Bob**: choose a random b in {1,...,p-1}

**Alice** sends: A ← ga (mod p)

**Bob** sends: B ← gb (mod p)

The key will be kAB = gab (mod p)

**Alice** takes the B received from Bob and computes: Ba (mod p) = (gb)a = kAB

**Bob** takes the A received from Alice and computes Ab (mod p) = (ga)b = kAB

Eavesdropper sees: p, g, A=ga (mod p), and B=gb (mod p)

He wants to compute gab (mod p)

Define DHg(ga, gb) = gab (mod p)

How hard is the DH function mod p?

Suppose prime p is n bits long

Best known algorithm (GNFS): run time )

Similar security:

|  |  |  |
| --- | --- | --- |
| **cipher key size** | **modulus size** | **Elliptic Curve size** |
| 80 bits | 1024 bits | 160 bits |
| 128 bits | 3072 bits | 256 bits |
| 256 bits (AES) | 15360 bits | 512 bits |

As a result: slow transition away from (mod p) to elliptic curves

modulos size of 1024 bits → complex aprox = e10 ← it’s not exactly because other factor on the equation → e80

As it is the DH protocol is insecure against man-in-the-middle attacks (active attacks).

The MiTM intercepts the first message from Alice A = ga and sends an A’ = ga’ to Bob

The MiTM intercepts the message from Bob B=gb and sends a B’ = bb’ to Alice

Alice is gonna compute her part of the key with the message received from MiTM:

KAM=(B’)a = gab’

Bob is gonna compute her part of the key with the message received from MiTM:

KBM=(A’)b = gba’

The MiTM knows both keys. He will receive the ciphered messages, decrypts them and reencrypts them with the other key and resends to their destination.

# Public-Key encryption

Key pair: Public key for the Encryption algorithm and a Secret key for the Decryption algorithm

**Def**: a public-key encryption system is a triple of algorithms (G, E, D)

G(): randomized algorithm that outputs a key pair (pk, sk)

E(pk, m): randomized algorithm that takes m in M and outputs c in C

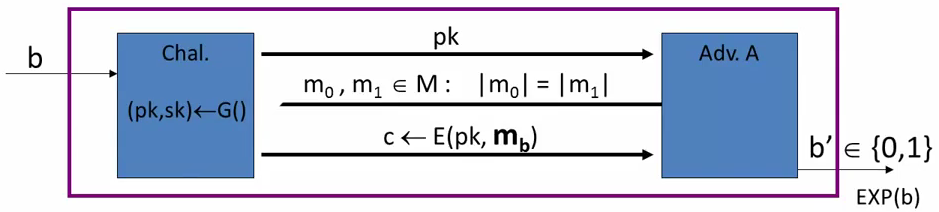
D(sk, c): deterministic algorithm that takes c in C and outputs m in M or bottom

**Consistency**: for all (pk, sk) output by G:

for all m in M: D(sk, E(pk,m)) = m

## Semantic security:

For b = 0,1 define experiments EXP(0) and EXP(1) as:



The attacker is given the public key, so he can encrypt all the messages he want. There is no Chosen Plaintext Attack.

**Def**: EBIG = (G,E,D) is semantically secure (a.k.a IND-CPA) if for all efficient algorithms:

AdvSS[A,EBIG] = | Pr[EXP(0) = 1] - Pr[EXP(1) = 1] | is negligible

## Establishing a shared secret

Alice generates her key pair (pk, sk) with G()

Alice sends her pk to Bob

Bob receives the pk from Alice

Bob chooses a random number x in {0,1}128

Bob encrypts the random x with pk from Alice: c ← E(pk, x)

Alice decrypts the c message from Bob with her secret key sk and obtains x

x is the shared secret key between Alice and Bob

## Security (eavesdropping only)

Adversary sees pk, E(pk,x) and wants x in M

Semantic security ⇒ Adversary cannot distinguish:

{ pk, E(pk,x), x} from {pk, E(pk, x), rand in M}

⇒ can derive session key from x

Protocol is still vulnerable to man-in-the-middle

The MiTM intercepts the messages and sends his public key (pk’) instead of the original public key (pk).

Construction of public key encryption rely on hard problems from number theory and algebra.