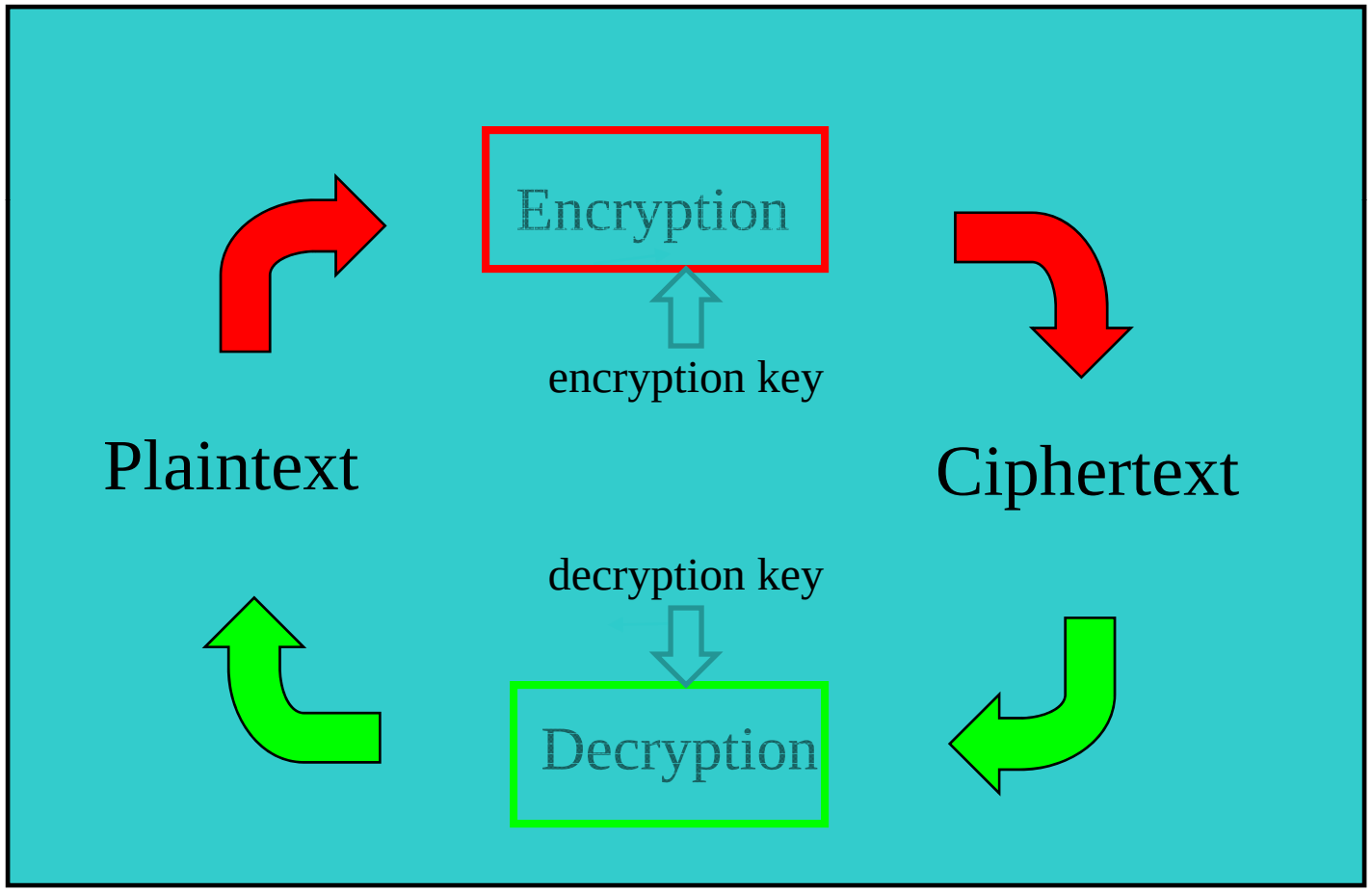
CIS 5371 Cryptography

2. Safeguard and Attack

Encryption


Encryption
encryption key
Plaintext
Ciphertext
decryption key
Decryption

Encryption algorithms

Notation

- Message M
- Algorithm A
- Key K
- Cryptographic transformation: $M' = A(K,M)$

If A is an encryption algorithm with key K and A' is the corresponding decryption algorithm with key K', we require that:

$$M = A'(K', A(K,M))$$

So $A'(K',\bullet) = A(K,\bullet)^{-1}$

Dolev-Yao threat model

Notation

- Principal or entity or user: a computer, device, resource, person, etc
- Attacker (adversary, enemy. Intruder, eavesdropper, imposter)

Setting

- A large open network
- The adversary is very powerful and clever in manipulating communication
- The adversaries techniques are unpredictable

Dolev-Yao threat model

The attacker,

- Can obtain any message passing through the network.
- May be a legitimate user and can initiate any conversation with any other user (insider)
- Has the opportunity to become any receiver to any user
- Can send messages to any user by impersonating another user.

Dolev-Yao threat model

Unless explicitly stated,
The attacker is not omnipotent. In particular,

- The attacker cannot guess a random number selected from a set with X with probability better than $1/|X|$.
- He cannot encrypt data, or decrypt encrypted data without the correct key (for sufficiently large key space)
- Has cannot find the correct encryption/decryption key.
- Typically, the actions and resources of the adversary are *polynomially* bounded –to be explained later.

Authentication Servers
Trust Management systems

- Alice and Bob wish to communicate privately over an insecure network---in the presence of the adversary, Malice.
- They have to encrypt the messages they send each other.
- Suppose they have never met before: how do they exchange keys?
- One way is to get the keys from a Trusted Third Party (TTP), Trent.
- The TTP can be an Authentication Server.
- In general parties may use different Servers, which must then trust each other.
- Trust Management (TM) systems will support security applications for entities that have no direct trust links.

Trust Management
The $n(n\text{-}1)/2$ problem

- If Alice and Bob meet physically they can exchange a secret key (or knowledge about each others public keys).
- If there are n users in the communication system then $n(n\text{-}1)/2$ need to be exchanged.
- For many applications the cost of storing this many keys may be excessive.
- Not to mention the complexity of private meetings –-although this could be done initially at a key pre-distribution stage, with the help of a TTP.
- In this Section shall assume that Alice and Bob have a common TTP.

Authenticated Key Establishment, AKE

Let *K* be the key to be established between Alice and Bob.

We require that at the end of an AKE protocol:

1. Only Alice and Bob share *K*
2. Alice and Bob each know that the other (and *only the other*) knows *K*
3. Alice and Bob know that *K* is newly generated

The first requirement captures the meaning of *authentication*;

The second, *entity authentication*;

The third, the basic principle of *key freshness* (*key management*).

AKE protocols
Protocols for message confidentiality

AKE with a trusted third party

- Both Alice and Bob *trust* Trent
- Alice shares an encryption key K_{AT} with Trent
- Bob shares an encryption key K_{BT} with Trent

AKE protocols
Protocols for message confidentiality

An AKE protocol with trusted third party

1. Alice generates K at random and sends to Trent:

   $$Alice, Bob, \{K\}_{K_{AT}}$$

1. Trent finds the keys K_{AT}, K_{BT}, decrypts $\{K\}_{K_{AT}}$ to reveal K, creates $\{K\}_{K_{BT}}$ and sends to Bob:

   $$Alice, Bob, \{K\}_{K_{BT}}$$

1. Bob decrypts $\{K\}_{K_{BT}}$ to reveal K, and sends to Alice:

   $$\{Hello\ Alice, I'm\ Bob\}_K$$

Security properties for AKE protocols

At the end of a protocol run:

- *Authentication*
  (Other than Trent) Alice and Bob and *only they* should know the key *K*.
- *Entity Authentication*
  Alice and Bob should each know that the other knows the key *K*.
- *Key-freshness* (or -*management*)
  Alice and Bob should know that the key *K* is newly generated.

Protocols for message confidentiality

A weakness of the AKE protocol with TTP

- In the protocol Bob must be satisfied that Alice has generated the key *K* sufficiently at random.
- But maybe Alice doesn't bother.
- This is a design flaw.

AKE protocols –a fix

AKE 2

1. Alice sends to Trent: *Alice*, *Bob*
2. Trent finds the keys K_{AT}, K_{BT}, generates K at random and sends Alice:

$$\{K\}_{K_{AT}}, \{K\}_{KB_T}$$

1. Alice decrypts $\{K\}_{K_{AT}}$ and sends to Bob:

$$Trent, Alice, \{K\}_{K_{BT}}$$

1. Bob decrypts $\{K\}_{K_{BT}}$ to reveal K, forms and sends to Alice:

$$\{Hello\ Alice, I'm\ Bob\}_K$$

Attack, Fix, Attack, Fix, …
The way ahead

- Even this fix is not good enough.
- One way to learn is by improvising, and then waiting for somebody to break your scheme …

AKE protocols

Attack on the AKE 2 protocol

1. Alice's first message (*Alice*, *Bob*) is intercepted by Malice
2. Malice sends to Trent: *Alice*, *Malice*.
3. Trent finds keys K_{AT}, K_{MT}, generates K_{AM} at random and sends Alice:

$$\{K_{AM}\}_{K_{AT}}, \{K_{AM}\}_{K_{MT}}$$

1. Alice decrypts $\{K_{AM}\}_{K_{AT}}$ to get K_{AM}, and sends to Malice (Bob):

$$Trent, Alice, \{K_{AM}\}_{K_{MT}}$$

1. Malice (Bob) sends to Alice:

$$\{Hello\ Alice, I'm\ Bob\}_{K_{AM}}$$

AKE protocols

Attack on the AKE 2 protocol

Result

Alice thinks she is sharing a key with Bob, while actually sharing a key with Malice.

AKE protocols

A fix

Alice sends to Trent: *Alice*, $\{Bob\}_{K_{AT}}$.

Not good enough:

Malice (Alice) can now send to Trent: *Alice*, $\{\text{Malice}\}_{K_{AT}}$.

(Malice can get $\{\text{Malice}\}_{K_{AT}}$ from an earlier session with Alice in which he was involved (replay attack), or get it by tampering with encryptions)

Even this is not good enough.

What is needed is to guard against tampering of messages, that is protocols with message authentication and use time stamps.

Message Authentication

A Message Authentication protocol with TTP

1. Alice sends to Trent: *Alice*, *Bob*
2. Trent finds keys K_{AT}, K_{BT}, generates K at random and sends Alice:

$$\{Bob, K\}_{K_{AT}}, \{Alice, K\}_{K_{BT}}$$

1. Alice decrypts $\{Bob, K\}_{K_{AT}}$, extracts K, checks Bob's ID, and sends to Bob:

$$Trent, Alice, \{Alice,K\}_{K_{BT}}$$

1. Bob decrypts $\{Alice, K\}_{K_{BT}}$, extracts K, checks Alice's ID, and sends to Alice:

$$\{Hello\ Alice, I'm\ Bob\}_K$$

Attack on Message Authentication

A Replay attack

1. Alice sends to Trent: *Alice*, *Bob*
2. Malice (Trent) sends to Alice:

$$\{Bob, K'\}_{K_{AT}}, \{Alice, K'\}_{KB_T}$$

1. Etc

Here $\{Bob, K'\}_{K_{AT}}$, $\{Alice, K'\}_{KB_T}$ are a replay of old messages from an earlier execution.

This implies that the same key *K'* is used again.

A Challenge-Response protocol

The Needham-Schroeder protocol

1. Alice generates at random a "nonce" N_A and sends to Trent:

   $Alice, Bob, N_A$

1. Trent creates a random K, and sends Alice:

   $\{N_A, K, Bob, \{K, Alice\}_{K_{BT}}\}_{K_{AT}}$

1. Alice decrypts, checks N_A and Bob's ID and sends to Bob:

   $Trent, \{K, Alice\}_{K_{BT}}$

1. Bob decrypts, checks Alice's ID, creates a random N_B, and sends to Alice: $\{I'm\ Bob!\ N_B\}_K$

1. Alice sends to Bob: $\{I'm\ Alice!\ N_B - 1\}_K$

A replay attack

The Denning-Sako attack

1. Alice generates at random N_A and sends to Trent:

   Alice, *Bob*, N_A

1. Trent creates a random K, and sends Alice:

   $\{N_A, K, Bob, \{K, Alice\}_{K_{BT}}\}_{K_{AT}}$

1. Alice decrypts, checks N_A and Bob's ID and sends to Malice (Bob):

   Trent, $\{K, Alice\}_{K_{BT}}$

3' Malice (Alice) sends to Bob (*replay attack*):

   Trent, $\{ K', Alice\}_{K_{BT}}$

1. Bob decrypts, checks Alice's ID, creates a random N_B, and sends to Alice: $\{I'm\ Bob!\ N_B\}_{K'}$,

2. Malice (Alice) sends to Bob: $\{I'm\ Alice!\ N_B - 1\}_{K'}$

A replay attack

Result

Bob thinks he is sharing a new session key with Alice while actually sharing an old key, which may be known to Malice.

The Denning-Sako fix

1. Alice generates at random N_A and sends to Trent:

   $Alice, Bob, N_A$

1. Trent creates a random K_A, and sends Alice:

   $\{N_A, K, T, Bob, \{K, Alice, T\}_{K_{BT}}\}_{K_{AT}}$,

   where T is a timestamp

1. Alice decrypts, checks N_A, T and Bob's ID and sends to Bob:

   $\{K, Alice, T\}_{K_{BT}}$

1. Bob decrypts, checks Alice's ID and T, creates a random N_B, and sends to Alice: $\{I'm\ Bob!\ N_B\}_K$

2. Alice sends to Bob: $\{I'm\ Alice!\ N_B - 1\}_K$

A protocol using Public-key Cryptosystems

Public-key cryptosystems

1. Each user U has a *public encryption key* K_U and a corresponding *secret decryption key* K_U^{-1} such that: *for any message M,*

   $C = \{M\}_{K_U}$ *is the encryption of message M*

   $M = \{C\}_{K_U^{-1}}$ *is the plaintext decryption of C*

1. We assume that only U knows the secret decryption key K_U^{-1}

2. We can regard $\{X\}_{K_U^{-1}}$ as a *digital signature* on X, which can be easily verified by any user using the public key K_U, since: $X = \{\{X\}_{K_U^{-1}}\}_{K_U}$

Needham-Schroeder public-key authentication protocol

1. Alice sends to Trent: *Alice*, *Bob*
2. Trent sends to Alice: $\{K_B, Bob\}_{K_T\text{-}1}$
3. Alice verifies Trent's signature on K_B, *Bob*, creates her random nonce N_A, and sends to Bob: $\{N_A, Alice\}_{K_B}$
4. Bob decrypts, checks Alice's ID and sends to Trent: *Bob*, *Alice*
5. Trent sends Bob: $\{K_A, Alice\}_{K_T\text{-}1}$
6. Bob verifies Trent's signature, creates his random nonce N_B, and sends to Alice: $\{N_A, N_B\}_{K_A}$
7. Alice decrypts, and sends to Bob: $\{N_B\}_{K_B}$

Low's interleaving attack

The protocol can be considered as the interleaving of two logically disjoint protocols:

Steps 1, 2, 4, 5 involve getting public keys while

Steps 3,6,7 are concerned with the authentication.

Therefore we may assume that each principal initially has copies of each other's public keys, and focus on the steps:

1. Alice sends to Bob: $\{N_A, Alice\}_{K_B}$
2. Bob sends to Alice: $\{N_A, N_B\}_{K_A}$
3. Alice sends to Bob: $\{N_B\}_{K_B}$

Low's interleaving attack

The attack involves two runs:

1-3. 1-6, 1-7: Alice establishes a valid session with Malice

2-3. 2-6, 2-7: Malice impersonates Alice to establish a bogus session with Bob:

2-3 Malice → Bob: $\{N_A, Alice\}_{K_B}$

2-6 Bob → Malice: $\{N_A, N_B\}_{K_A}$

Malice can't decrypt this to get N_B so gets Alice to do it for him

1-6 Malice → Alice: $\{N_A, N_B\}_{K_A}$

1-7 Alice → Malice: $\{N_B\}_{K_M}$

2-7 Alice → Malice: $\{N_B\}_{K_B}$

A simple fix

2-6 Bob sends to Alice: $\{\text{Bob}, N_A, N_B\}_{K_A}$

Security of the fixed N-S Public-key Authentication Protocol

- One should refrain from claiming that this fix results in a secure protocol.
- We shall see later that even with the fix there are several additional problems due to the design feature that message authentication is achieved via "decryption-and-checking".
- The error-prone nature of AKE protocols has inspired the systematic approach to the developing secure AKE protocols and the formulation of a security framework for such protocols.