Security Protocols

Deepak Puthal

Email: Deepak.Puthal@uts.edu.au

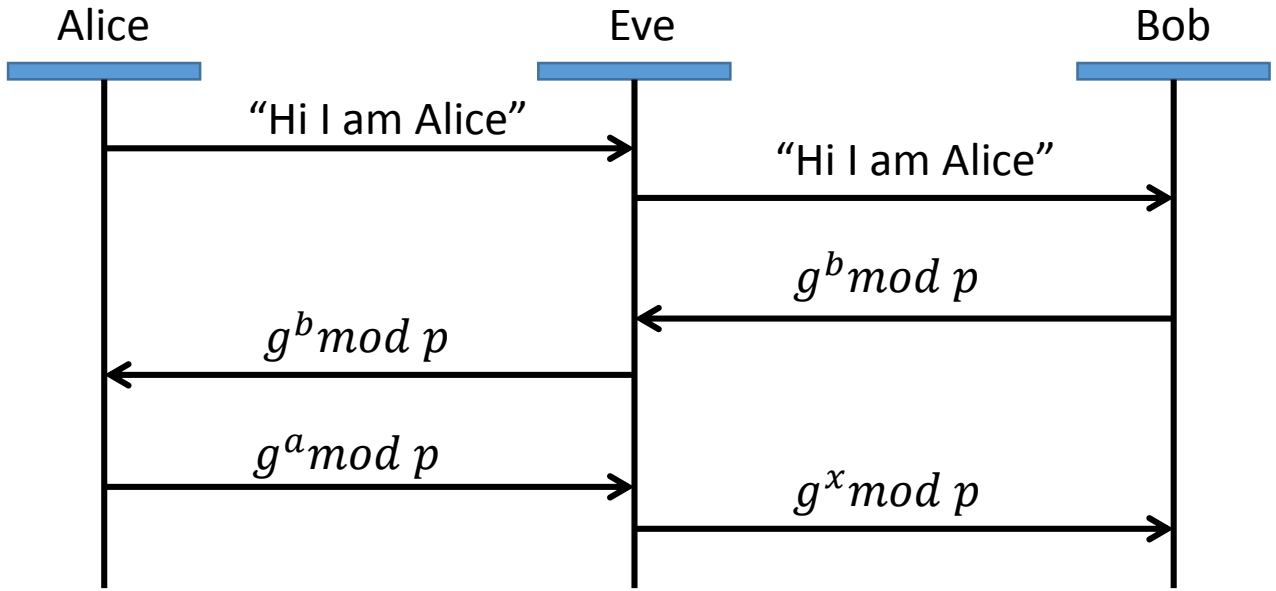
41900 – Fundamentals of Security

Overview

- **Problem with Diffie-Hellman**
  - Session Hijacking
  - Encrypted Key Exchange (EKE)
  - Definitions
- **Needham-Schroeder Protocol**
- **Public Key Management**
  - Certificate Authorities
  - Trust Models
- Secret Splitting and Sharing
  - Secret Splitting
  - Secret Sharing
- Commitment Protocols
  - Bit Commitment
  - Fair Coin Flipping
  - Mental Poker

Man-in-the-Middle

Suppose Alice and Bob are performing Diffie–Hellman key exchange, but a third party (Eve) actually owns the channel, and can intercept traffic.

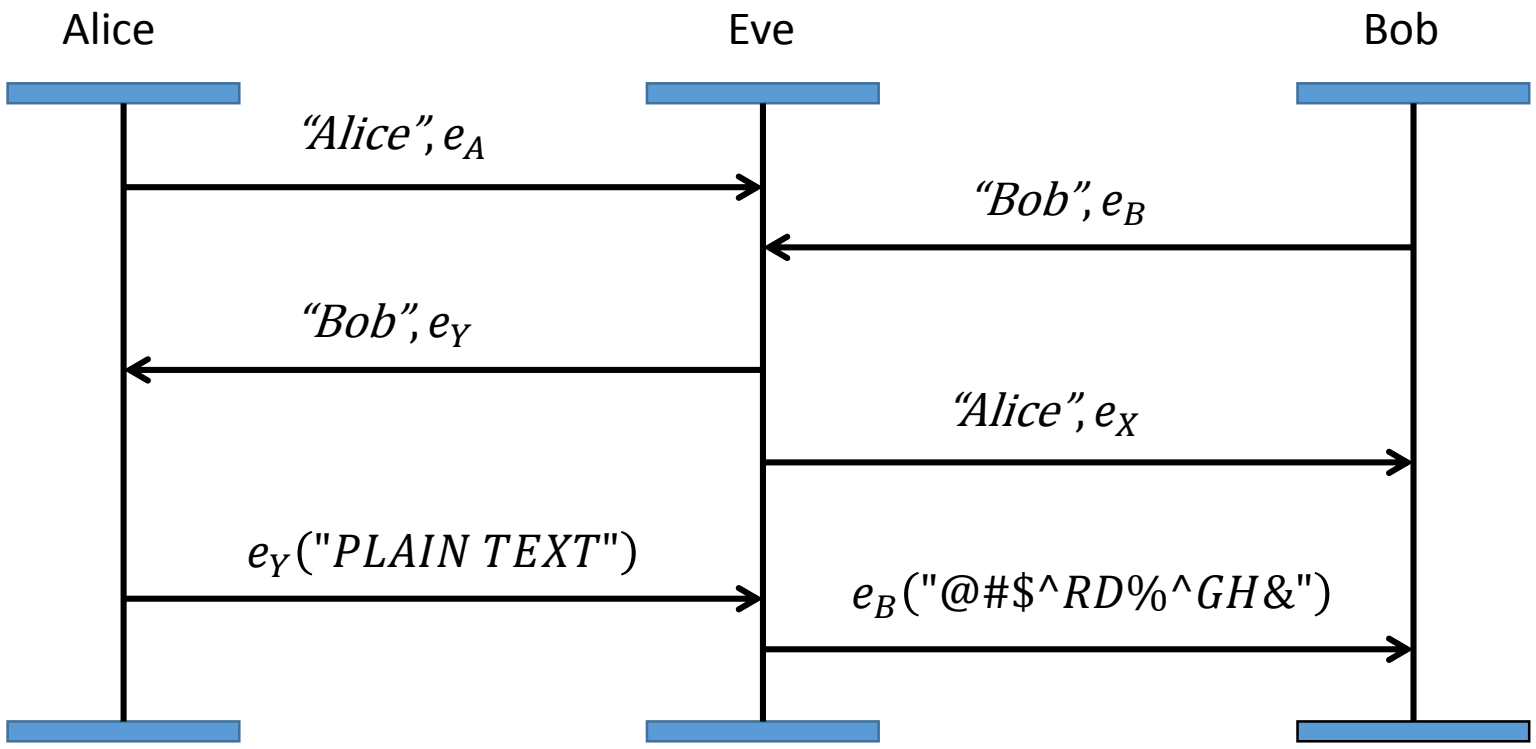


Shared secret for Alice/Eve is g^{ax} mod p.

Shared secret for Eve/Bob is g^{bx} mod p.

Eve owns the channel and neither Alice nor Bob know!

Man-in-the-Middle



Eve owns the channel!

Definitions

A protocol is said to have *perfect forward secrecy* if disclosure of long-term keys does not compromise past (short term) sessions.

- Ephemeral keys (such as are generated during Diffie–Hellman) give this automatically.
- Encrypting everything directly with an RSA public key does **not** have forward secrecy.

A protocol is vulnerable to a *known-key attack* if disclosure of past session keys allows an attacker to compromise future session keys (including actively impersonating).

Needham–Schroeder Protocol

The Needham–Schroeder protocol facilitates key exchange using a trusted third party (TPP).

Alice and Bob want to set up a session key for communication. All parties share a key with Trent, the TPP.

1. Alice sends Trent a request to talk to Bob: (A, B, r_A), where r_A is a nonce.
2. Trent sends Alice a session key, and a ticket to give to Bob: $E_{K_{AT}}\left(r_A, B, k_{AB}, E_{K_{AT}}(k_{AB}, A)\right)$
3. Alice sends Bob the ticket: $E_{K_{BT}}(k_{AB}, A), E_{K_{AB}}(s_A)$
4. Bob challenges Alice: $E_{K_{AB}}(s_A - 1, r_B)$, where r_B is a nonce.
5. Alice responds to Bob's challenge: $E_{K_{AB}}(r_B - 1)$

Needham–Schroeder Protocol

The Needham–Schroeder protocol facilitates key exchange using a trusted third party (TPP).

Alice and Bob want to set up a session key for communication. All parties share a key with Trent, the TPP.

1. Alice sends Trent a request to talk to Bob: (A, B, r_A), where r_A is a nonce.
2. Trent sends Alice a session key, and a ticket to give to Bob: $E_{K_{AT}}\left(r_A, B, k_{AB}, E_{K_{AT}}(k_{AB}, A)\right)$
3. Alice sends Bob the ticket: $E_{K_{BT}}(k_{AB}, A), E_{K_{AB}}(s_A)$
4. Bob challenges Alice: $E_{K_{AB}}(s_A - 1, r_B)$, where r_B is a nonce.
5. Alice responds to Bob's challenge: $E_{K_{AB}}(r_B - 1)$

What if Eve gets a hold of k_{AT} somehow?

Needham–Schroeder Protocol

Problem: Bob has no guarantee that k_{AB} is fresh. Old session keys are valuable, as they do not expire.

1. Suppose Eve manages to get k_{AT}.
2. She can now read all of Alice's messages and impersonate her to everyone else.
3. Alice needs to revoke her key, but the only person that can make this have an effect is Trent.
4. **Key revocation is a major problem.**

Needham-Schroeder's problem is that it assumes all users of the system are good guys, and the goal is to keep the bad guys from getting in — the "eggshell" model.

Kerberos

Kerberos is a protocol which builds on the Needham–Schroeder protocol, and allows nodes within an insecure network to prove identity to each other.

Extra reading: Designing an Authentication System: a Dialogue in Four Scenes.

Public Key Management using Certification Authorities

A public key certificate binds a public key to its owner:

1. Alice sends her public key to the CA (Trent).
2. The CA produces a certificate for Alice.
3. Alice sends her public key and certificate to Bob.
4. Bob verifies the certificate using Alice's public key.
5. Bob sends encrypted messages to Alice using the key.

Public Key Management using Certification Authorities

Certificate = sign_{CA}[X.500: name, org, address, pubkey, expires, ...]

- Everyone must be able to verify the CA's public key, so it is shipped with OS's, browsers, etc.
- The CA is a trusted party: it has the ability to issue signatures to whomever.

Public Key Certificate Generation

1. Alice generates a public/private keypair.
2. Alice sends the public key to the CA.
3. The CA challenges Alice to see if she knows the private key.
4. The CA generates a certificate and sends it to Alice.

Important Note:

1. The CA never learns Alice's private key.
2. Important for forward secrecy.
3. A compromise of the CA can still lead to people pretending to be Alice.

Certificate Revocation

Alice's certificate may need to be revoked.

- Her private key is stolen or otherwise compromised.
- She changes jobs (or trustworthiness).

This is a major problem with the CA system.

- May require daily certification-validation information (slow, cumbersome).
- Use expiration date field.
- Use of a certificate revocation list (CRL) which is circulated (like bad credit cards.)

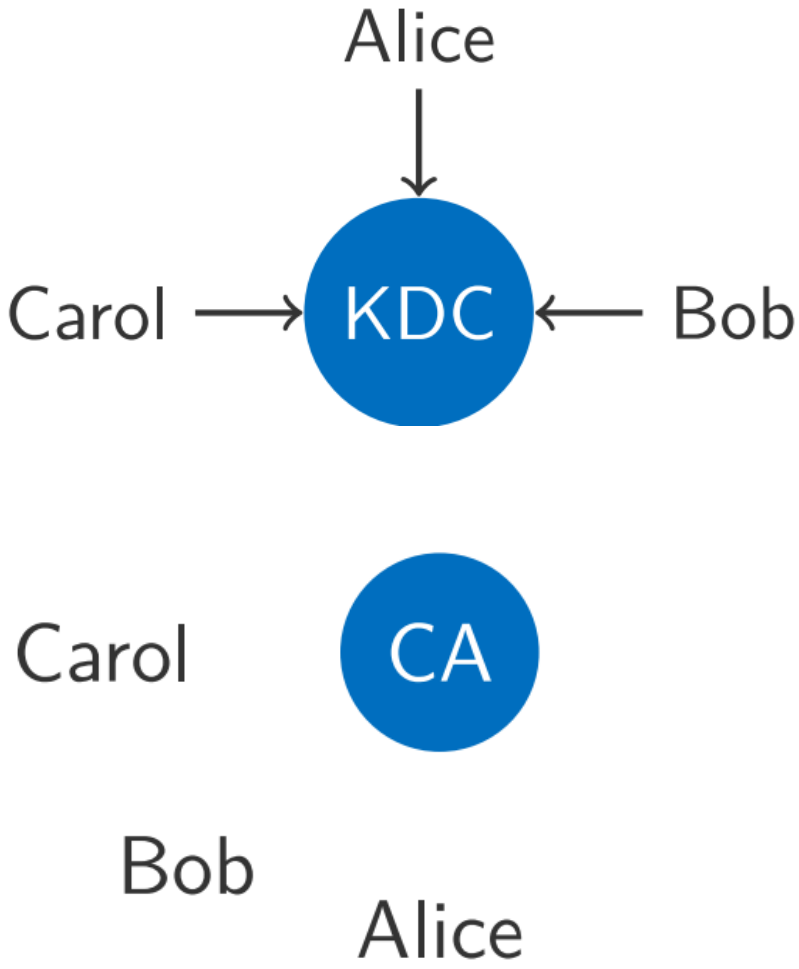
Trust model

Symmetric Keys

- TTP must be online (used every session).
- TTP is a juicy target (knows passwords).
- No forward Secrecy.

Asymmetric Keys

- TTP is offline (only used in generation).
- TTP only knows public keys.
- TTP has forward secrecy.
- Not as fast (e.g. SSL/TLS, PGP, ...)



Secret Splitting and Sharing

Secret Splitting

Problem: You are the CEO of Coca-Cola. You're responsible for keeping the formula secret from Pepsi's industrial spies. You could tell your most trusted employees, but ...

- They could defect to the opposition.
- They could fall to rubber hose cryptanalysis.

How can a secret be split among multiple parties such that each piece by itself is useless?

Secret Splitting with XOR

Suppose Trent wants to protect the message m:

1. Generate a random string r, the same length as m.
2. Compute $\mathrm{s} = \mathrm{m} \oplus \mathrm{r}$.
3. Give Alice r, and give Bob s.

Each piece r, s is called a shadow of the message m. To reconstruct m, Alice and Bob can XOR their shadows together:

$$\mathrm{s} \oplus \mathrm{r} = \mathrm{m}$$

If r is truly random, the system is perfectly secure (similar to One Time Pad).

The scheme may be extended to n people by generating $n - 1$ random strings $r_1, \ldots, r_{n-1}$. Give the first person r_1, the second person r_2, and so on up to r_{n-1}, and give the nth person $\mathrm{r}_1 \oplus \cdots \oplus \mathrm{r}_{\mathrm{n}-1} \oplus \mathrm{m}$.

Secret Splitting with XOR

Secret splitting aims to enhance reliability without increasing risk through distributing trust.

Issues: (with XOR)

- The system is adjudicated by Trent.
  - Trent can hand out rubbish and say it's the secret.
  - Trent can say he's splitting a secret 4 ways, but only splitting it between the first two people.
- All parties know the length of the message.
- The message is malleable: by flipping bits in any part, the recovered message changes.
- All parties are required to recover the message (*bus factor = 1*).

Secret Sharing

Problem: You are responsible for a small country's nuclear weapons.

- Ensure that no single lunatic can launch a missile.
- Ensure that no pair of lunatics can launch a missile.
- You want at least three of five officers to be lunatics before a missile can be launched.

This is called a (3, 5)-threshold scheme.

Secret Sharing

Shamir's Secret Sharing is an algorithm for dividing a secret into m pieces, where only n of them are required to reconstruct the original secret.

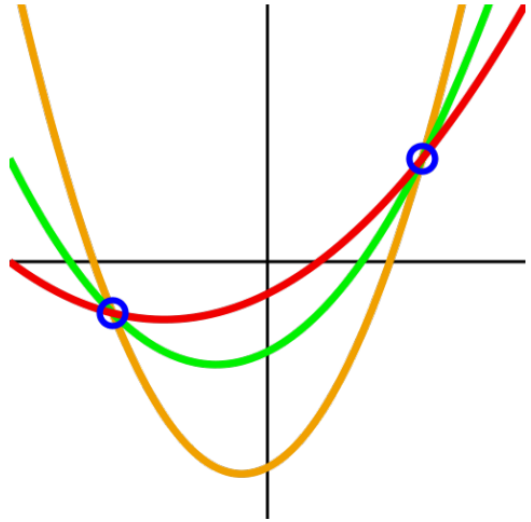
A polynomial of degree n can be uniquely defined by plotting $n + 1$ points on that polynomial.

Example

$$y = ax^2 + bx + c$$

Infinite possibilities for the coefficients with only 2 points.

3 points, and a, b, and c are uniquely determined.



Shamir's (t, n) threshold scheme

Fact: A polynomial $f(x)$ of degree $t - 1$ is uniquely determined by t distinct points $(x, f(x))$ lying on the curve. This works over Zp, not just R!

Trent wishes to distribute a message m amongst n users, where any group of t users $(1 \leqslant t \leqslant n)$ can recover m. (bus factor = $n - t + 1$)

Choose a prime $p > \max\{m, n\}$.

Create the polynomial $f(x) = m + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$, where the $0 \leqslant a_i < p$ are random and independent.

Trent selects n distinct points x_i, with $1 \leqslant x_i < p$.

Trent gives $(x_i, f(x_i))$ to person i.

Once t people pool their $(x_i, f(x_i))$ points, then the polynomial $f(x)$ can be reconstructed and m recovered. This can be done by Lagrange interpolation, for example.

Commitment Protocols

Bit Commitment

Problem:

- Alice wants to sell Bob information regarding police informants within his Mafia empire.
- Alice doesn’t trust Bob enough to tell him the rats without getting paid first.
- Bob thinks the deal is a police setup, and won’t give her the money until she commits to names.

Bit Commitment

Commitment:

1. Bob generates a random string r, and sends it to Alice.
2. Alice generates a random key k, and sends Bob $\mathrm{E_k(r \parallel m)}$.

Revelation:

1. Alice sends Bob the key k.
2. Bob decrypts the message with k, and verifies r.

Discussion:

1. r is needed for freshness, and to stop Alice from finding colliding messages, with $E_{k1}(m_1) = E_{k2}(m_2)$. She needs to commit at the time she receives r.
2. Bob does not know k until revelation, so cannot brute force the message space.

Bit Commitment with Hash Functions

Commitment:

1. Alice generates random strings r, s, and computes $x = h(r \| s \| m)$. (x is called a blob.)
2. Alice sends Bob (r, x).

Revelation:

1. Alice sends Bob the remaining data of (s, m).
2. Bob verifies that $h(r \| s \| m)$ is the same as the x he received.

Discussion:

1. Bob does not have to send any messages.
2. Alice sends a message to commit, and a message to reveal.
3. Since h is a crypto hash function, Alice cannot find t such that $h(r \| s \| m) = h(r \| t \| m)$.
4. s is kept secret so that Bob can't brute force the message space.

Fair Coin Flipping

Problem: Alice and Bob are arguing over the internet about who will be white (and therefore play first) in a game of online chess. They want to flip a coin to resolve the situation.

- Alice doesn't trust Bob to flip the coin.
- Bob doesn't trust Alice to flip the coin.

How can a coin be flipped fairly?

Fair Coin Flipping

To flip a coin fairly:

1. Alice commits to a bit b using a commitment scheme.
2. Bob tries to guess the bit.
3. Alice reveals the bit: if Bob guessed correctly, he wins the toss. Otherwise, Alice does.

Discussion:

The security of this algorithm lies in the security of the commitment scheme. In *particular*, the blob of the commitment scheme should not give away anything about the message inside (such as low-order bits).

Fair Coin Flipping using Public Key Crypto

We require a *commutative* public key cryptosystem, for example ElGamal, so that

$$E_A(E_B(m)) = E_B(E_A(m))$$

To perform a fair coin flip:

1. Alice and Bob generate keypairs A, B respectively.
2. Alice generates two random numbers r_T and r_H.
3. Alice sends Bob $m_1 = E_A(\text{heads}, r_H)$ and $m_2 = E_A(\text{tails}, r_T)$ in a random order.
4. Bob selects one of Alice's messages, call it x, and sends Alice $E_B(x)$.
5. Alice decrypts Bob's message and sends it back: $DA(EB(x))$.
6. Now Bob is left holding $E_B(m_1)$ or $E_B(m_2)$: he can decrypt this and send it back to Alice.
7. Alice verifies that Bob's response matches up with her r_T or r_H.

Fair Coin Flipping using Public Key Crypto

Discussion:

- The algorithm is self-enforcing: either party can detect the other cheating, without requiring a trusted third party.
- Bob learns the result of the coin flip before Alice. He can't change the result, but he may delay it ("flipping the coin into a well").
- Coin flipping has use in session key generation, as neither party can influence the result of each flip. For example, in Diffie-Hellman, one party selects an exponent after the first.

Mental Poker

Problem: Alice and Bob want to play poker over email.

- Alice doesn’t trust Bob.
- Bob doesn’t trust Alice.

How can Alice and Bob deal hands fairly?

Mental Poker

Use a commutative public key cryptosystem.

1. Alice and Bob generate keypairs A, B respectively.
2. Alice encrypts the 52 messages m_1 = (ace of spades, r_1), . . . using her public key, and sends these blobs x_1, . . . , x_{52} to Bob.
3. Bob picks 5 of the blobs at random (or however he pleases), encrypts them with his public key, and sends them back to Alice.
4. Alice decrypts the messages with her public key, and sends them back to Bob.
5. Bob decrypts the messages: this is his hand.
6. At the end of the game, Alice and Bob may reveal their keys to ensure no-one cheated.

How is Alice's hand dealt?

Attacks against the Poker Scheme

Cryptosystems (especially ones based in number theory) tend to leak small amounts of information, if not used in conjunction with hash functions.

For example, in RSA, if the number representing the card is a quadratic residue (a square number modulo the RSA modulus), then the encryption of the card is also a quadratic residue. This could be used by the dealer to “mark” certain cards.