Security Protocols

Kostas Papagiannopoulos
University of Amsterdam
k.papagiannopoulos@uva.nl

Contents

Introduction

Simple Authentication Protocols

Symmetric Key Protocols

Public Key Protocols

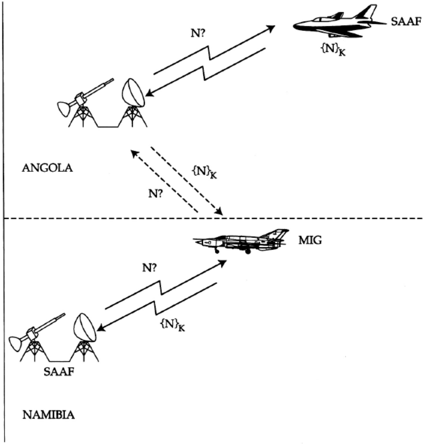
Zero-Knowledge Proofs

TLS Protocol

Renegotiation Attack on TLS 1.0

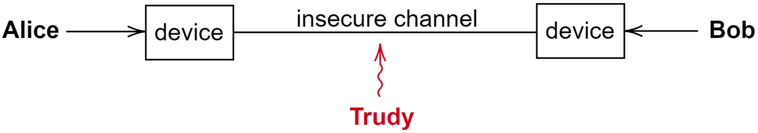
Introduction

Introduction



- MiG-in-the-middle attack during the Namibian war of independence

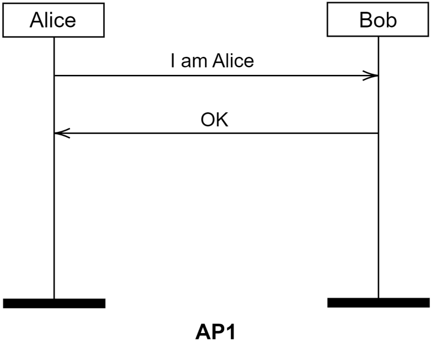
Introduction



- Alice and Bob try to communicate over an insecure channel e.g. over the internet
- The attacker Trudy is able to:
  - passively observe messages
  - replay past messages
  - actively insert, delete or change messages

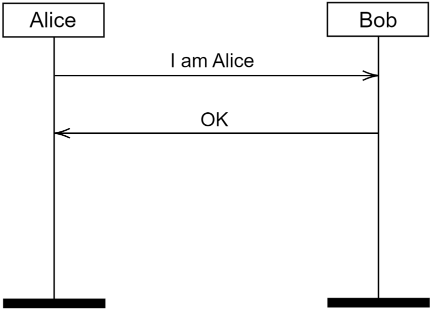
Simple Authentication Protocols

Simple Authentication Protocols



1. Alice initiates contact with Bob and claims that she is Alice
2. Bob accepts it and now Alice is authenticated

Simple Authentication Protocols

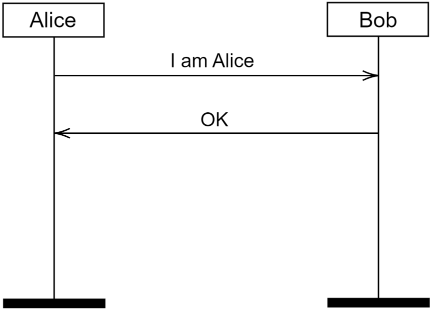


AP1

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob accepts it and now Alice is authenticated

- What is the issue here?

Simple Authentication Protocols

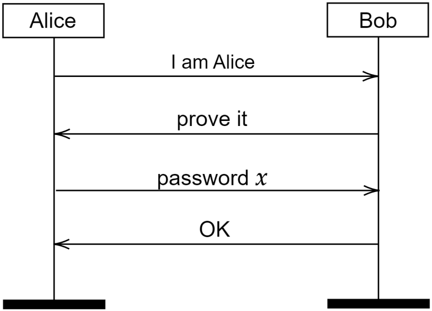


AP1

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob accepts it and now Alice is authenticated

- What is the issue here?
- Anyone can claim to be Alice! Bob has no means of verifying this claim

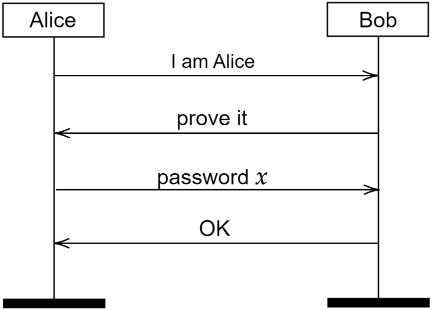
Simple Authentication Protocols



AP2

0. Before authentication takes place, Alice and Bob establish a secret password x between them

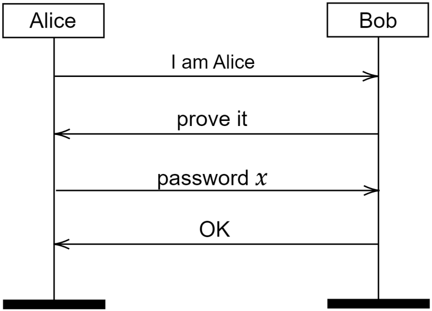
Simple Authentication Protocols



AP2

0. Before authentication takes place, Alice and Bob establish a secret password x between them
1. Alice initiates contact with Bob and claims that she is Alice

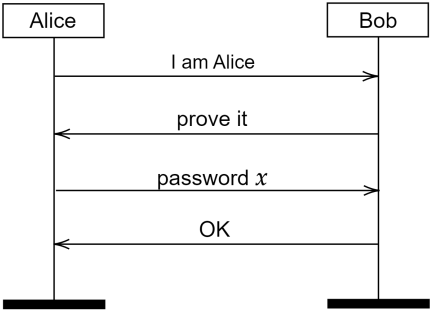
Simple Authentication Protocols



AP2

0. Before authentication takes place, Alice and Bob establish a secret password x between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob asks proof of her identity

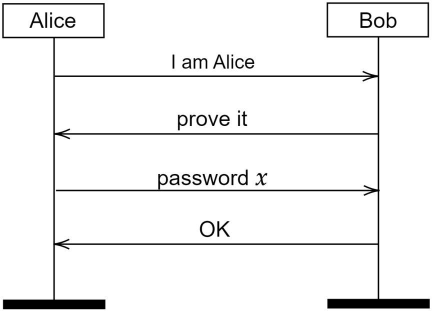
Simple Authentication Protocols



AP2

0. Before authentication takes place, Alice and Bob establish a secret password x between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob asks proof of her identity
3. Alice provides the secret password

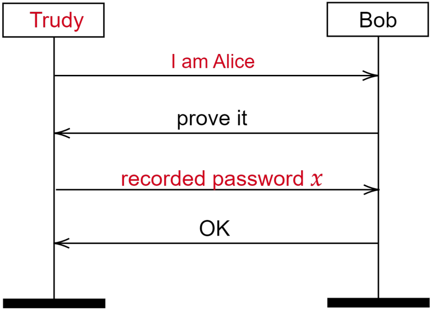
Simple Authentication Protocols



AP2

0. Before authentication takes place, Alice and Bob establish a secret password x between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob asks proof of her identity
3. Alice provides the secret password
4. Only Alice and Bob know the password x, thus Alice is authenticated

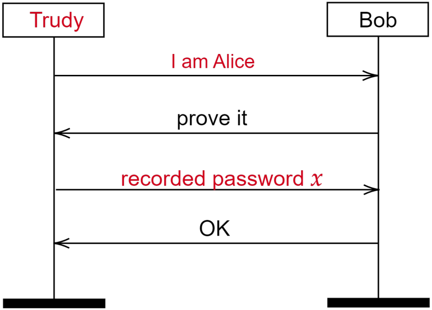
Simple Authentication Protocols



AP2 replay attack

- What are the issues here?

Simple Authentication Protocols

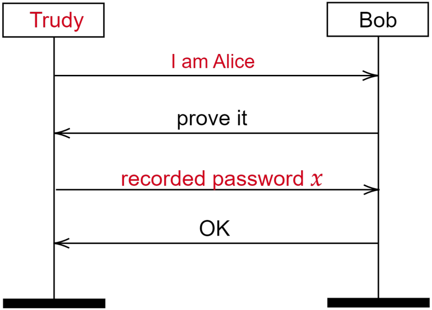


AP2 replay attack

- What are the issues here?

1. Trudy can observe the communication between Alice and Bob and learn the secret password x. Later she can **replay** it to authenticate as Alice.

Simple Authentication Protocols

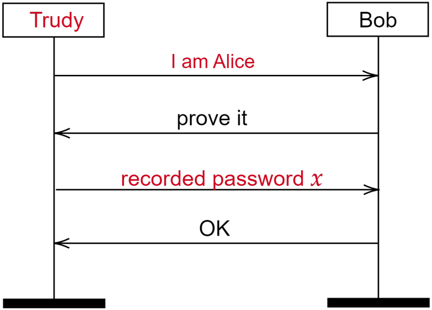


AP2 replay attack

- What are the issues here?

1. Trudy can observe the communication between Alice and Bob and learn the secret password x. Later she can **replay** it to authenticate as Alice.
2. Alice and Bob must share the password securely (chicken-egg problem)

Simple Authentication Protocols

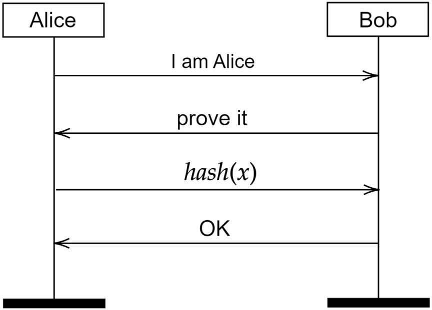


AP2 replay attack

- What are the issues here?

1. Trudy can observe the communication between Alice and Bob and learn the secret password x. Later she can **replay** it to authenticate as Alice.
2. Alice and Bob must share the password securely (chicken-egg problem)
3. One-sided authentication (Alice to Bob) instead of mutual authentication

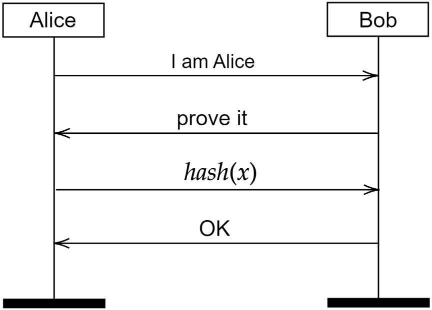
Simple Authentication Protocols



AP3

- What is the issue here?

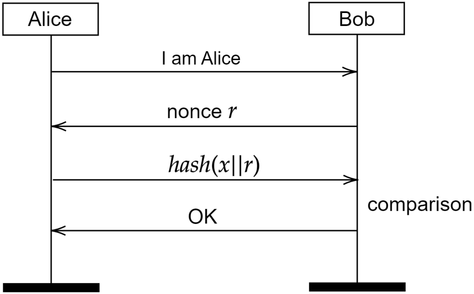
Simple Authentication Protocols



AP3

- What is the issue here?
- Hashing the password and ensuring the password's integrity does not prevent the replay attack
- Trudy will simply replay the *hash*(x) instead of the password x i.e. she does not need to know x

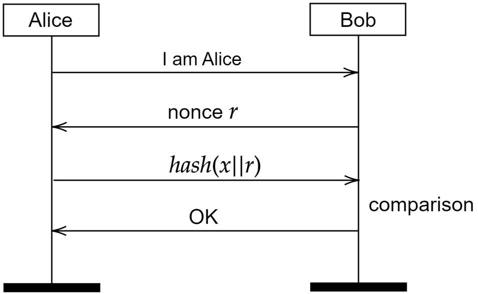
Simple Authentication Protocols



AP4

- Bob will use a **challenge-response** protocol

Simple Authentication Protocols

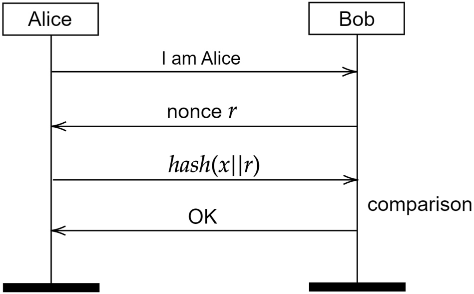


AP4

- Bob will use a **challenge-response** protocol

1. Bob sends a **nonce** r to Alice i.e. a number used once

Simple Authentication Protocols

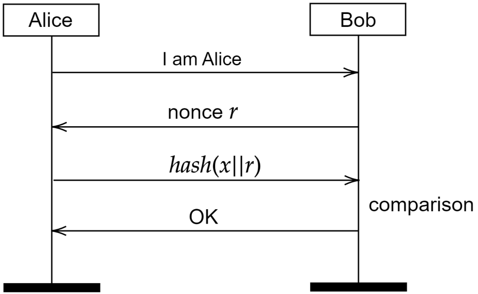


AP4

- Bob will use a **challenge-response** protocol

1. Bob sends a **nonce** r to Alice i.e. a number used once
2. Alice replies with the hash of the password concatenated with the nonce i.e. she computes $hash(x||r)$ and sends it to Bob

Simple Authentication Protocols

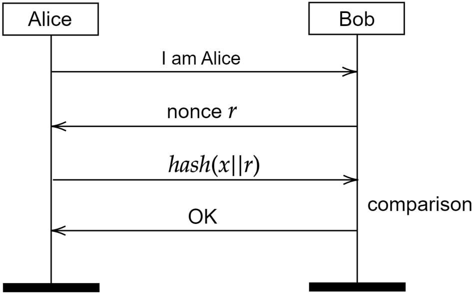


AP4

- Bob will use a **challenge-response** protocol

1. Bob sends a **nonce** r to Alice i.e. a number used once
2. Alice replies with the hash of the password concatenated with the nonce i.e. she computes $hash(x||r)$ and sends it to Bob
3. Bob knows x and r. He will compute the same hash $hash(x||r)$ and compare it to the one received from Alice

Simple Authentication Protocols



AP4

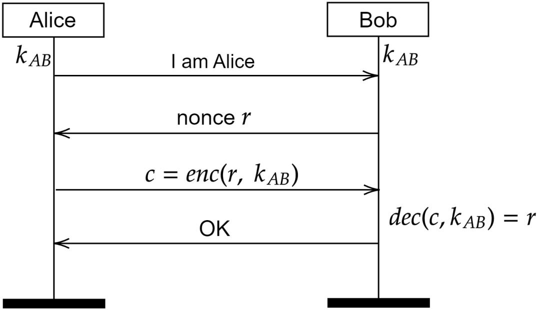
- Bob will use a **challenge-response** protocol

1. Bob sends a **nonce** r to Alice i.e. a number used once
2. Alice replies with the hash of the password concatenated with the nonce i.e. she computes $hash(x||r)$ and sends it to Bob
3. Bob knows x and r. He will compute the same hash $hash(x||r)$ and compare it to the one received from Alice

- The nonce makes the reply fresh and Bob can detect a replay attack

Authentication using Symmetric Cryptography

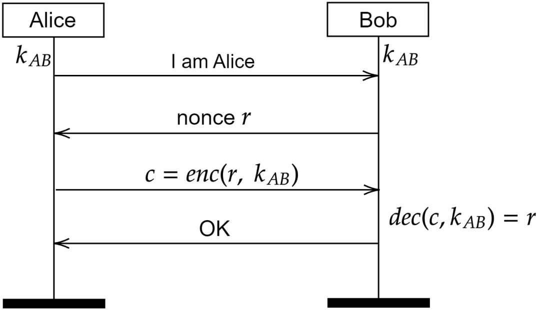
Symmetric Key Protocols



AP5

0. Before authentication takes place, Alice and Bob establish a secret symmetric key k_{AB} between them

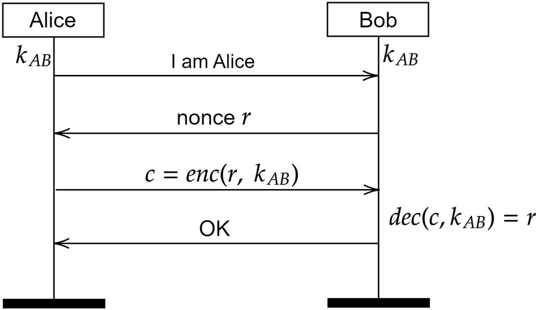
Symmetric Key Protocols



AP5

0. Before authentication takes place, Alice and Bob establish a secret symmetric key k_{AB} between them
1. Alice initiates contact with Bob and claims that she is Alice

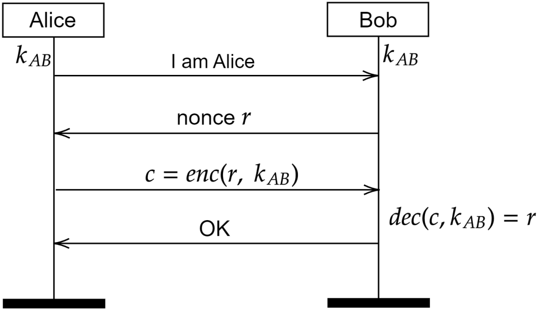
Symmetric Key Protocols



AP5

0. Before authentication takes place, Alice and Bob establish a secret symmetric key k_{AB} between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends a nonce r to Alice

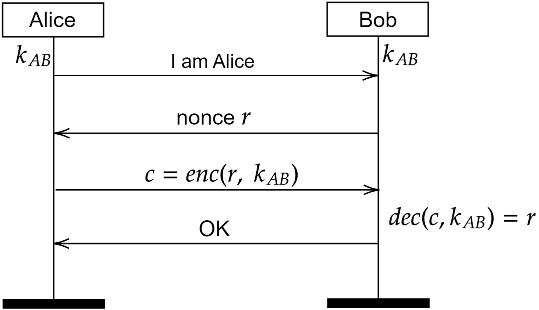
Symmetric Key Protocols



AP5

0. Before authentication takes place, Alice and Bob establish a secret symmetric key k_{AB} between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends a nonce r to Alice
3. Alice encrypts r with the key and sends ciphertext $c = enc(r, k_{AB})$ to Bob

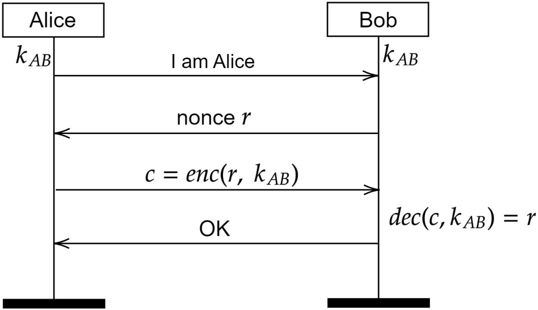
Symmetric Key Protocols



AP5

0. Before authentication takes place, Alice and Bob establish a secret symmetric key k_{AB} between them
1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends a nonce r to Alice
3. Alice encrypts r with the key and sends ciphertext $c = enc(r, k_{AB})$ to Bob
4. Bob decrypts c and verifies that $dec(c, k_{AB})$ is equal to r

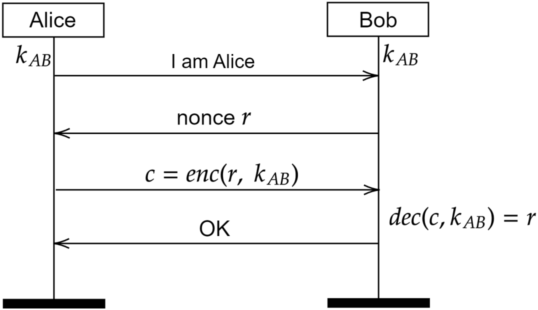
Symmetric Key Protocols



AP5

- This protocol can also prevent the replay attack
- What is the issue?

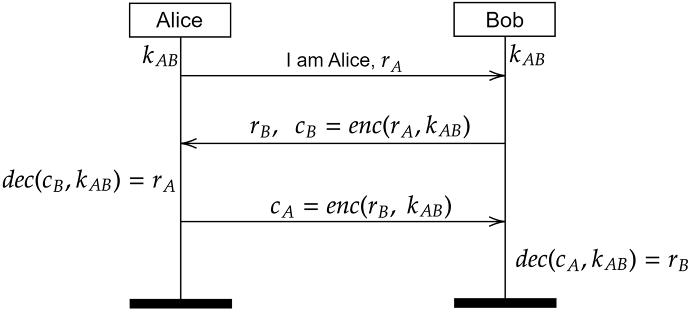
Symmetric Key Protocols



AP5

- This protocol can also prevent the replay attack
- What is the issue?
- One-sided authentication (Alice to Bob) instead of mutual authentication

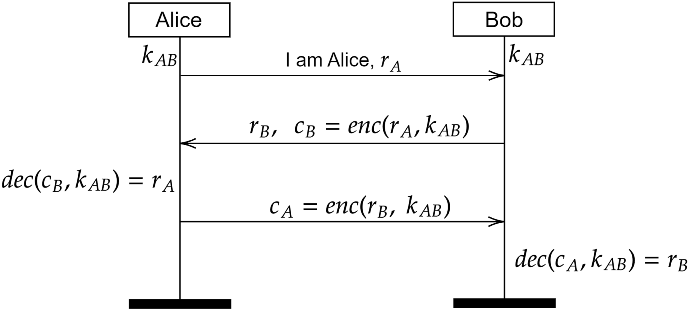
Symmetric Key Protocols



AP6

- We will perform the challenge-response protocol for both parties to achieve **mutual authentication**

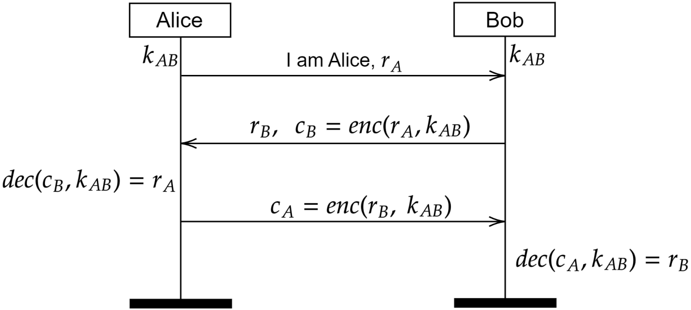
Symmetric Key Protocols



AP6

1. Alice sends nonce r_A to Bob
2. Bob encrypts r_A with the key and sends ciphertext $c_B = enc(r_A, k_{AB})$ and a nonce r_B to Alice
3. Alice decrypts c_B and verifies that $dec(c_B, k_{AB})$ is equal to her nonce r_A

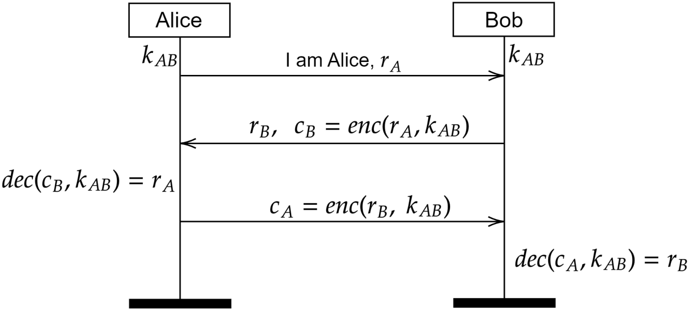
Symmetric Key Protocols



AP6

1. Alice sends nonce r_A to Bob
2. Bob encrypts r_A with the key and sends ciphertext $c_B = enc(r_A, k_{AB})$ and a nonce r_B to Alice
3. Alice decrypts c_B and verifies that $dec(c_B, k_{AB})$ is equal to her nonce r_A
4. Alice encrypts r_B with the key and sends ciphertext $c_A = enc(r_B, k_{AB})$ to Bob
5. Bob decrypts c_A and verifies that $dec(c_A, k_{AB})$ is equal to his nonce r_B

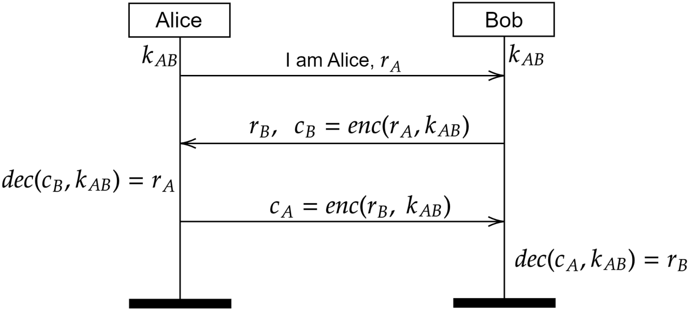
Symmetric Key Protocols



AP6

- What is the issue?

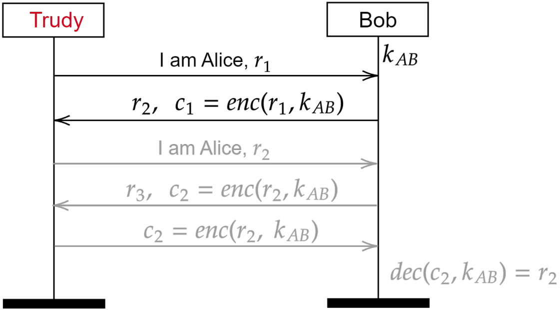
Symmetric Key Protocols



AP6

- What is the issue?
- Trudy can perform a different **replay** attack

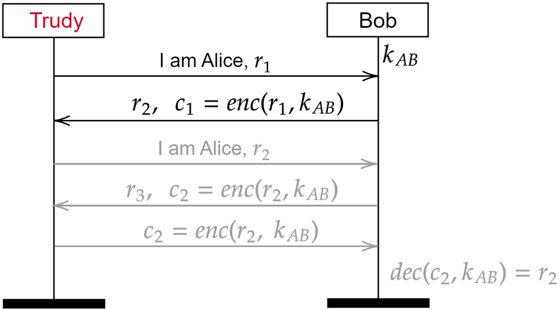
Symmetric Key Protocols



AP6 replay attack

1. Trudy claims to Bob that she is Alice and sends nonce r_1
2. Bob thinks he is talking to Alice so he encrypts the nonce and sends to Trudy ciphertext $c_1 = enc(r_1, k_{AB})$ and a nonce r_2
3. Trudy should encrypt r_2 with the key k_{AB} and send $enc(r_2, k_{AB})$ back to Bob

Symmetric Key Protocols

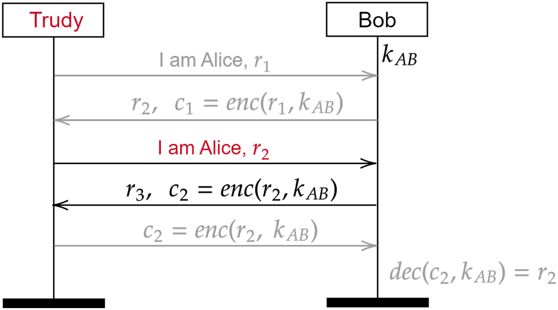


AP6 replay attack

1. Trudy claims to Bob that she is Alice and sends nonce r_1
2. Bob thinks he is talking to Alice so he encrypts the nonce and sends to Trudy ciphertext $c_1 = enc(r_1, k_{AB})$ and a nonce r_2
3. Trudy should encrypt r_2 with the key k_{AB} and send $enc(r_2, k_{AB})$ back to Bob

- Trudy doesn't know the key k_{AB} so the protocol appears to be secure
- To break it she must somehow recover $enc(r_2, k_{AB})$

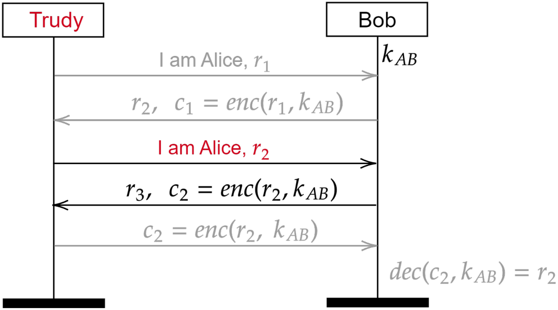
Symmetric Key Protocols



AP6 replay attack

4. Trudy claims again to Bob that she is Alice. She must send a nonce and chooses to send Bob's old nonce r_2 (from her previous attempt)
5. Bob doesn't remember his old nonce, so he encrypts it and sends back to Trudy the ciphertext $c_2 = enc(r_2, k_{AB})$ and another nonce r_3

Symmetric Key Protocols

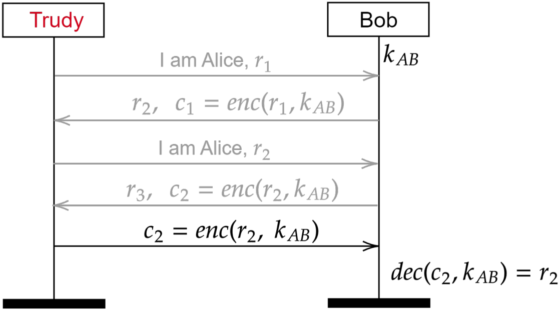


AP6 replay attack

4. Trudy claims again to Bob that she is Alice. She must send a nonce and chooses to send Bob's old nonce r_2 (from her previous attempt)
5. Bob doesn't remember his old nonce, so he encrypts it and sends back to Trudy the ciphertext $c_2 = enc(r_2, k_{AB})$ and another nonce r_3

- Trudy now has the missing value $c_2 = enc(r_2, k_{AB})$
- She can use it in her **first authentication attempt** to trick Bob

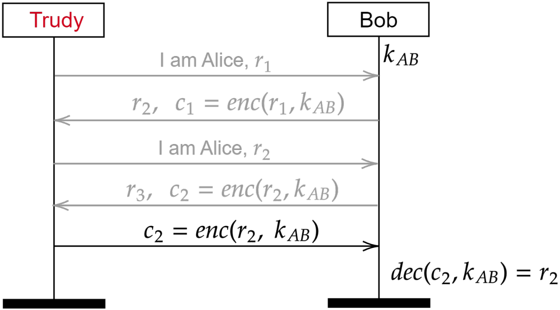
Symmetric Key Protocols



AP6 replay attack

7. Trudy sends to Bob $c_2 = enc(r_2, k_{AB})$
8. Bob decrypts c_2 and verifies that $dec(c_2, k_{AB})$ is equal to r_2

Symmetric Key Protocols

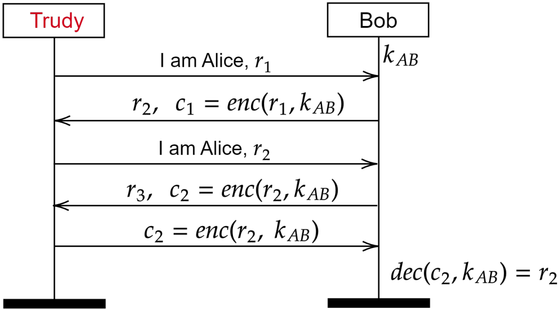


AP6 replay attack

7. Trudy sends to Bob $c_2 = enc(r_2, k_{AB})$
8. Bob decrypts c_2 and verifies that $dec(c_2, k_{AB})$ is equal to r_2

- Trudy is now authenticated as Alice

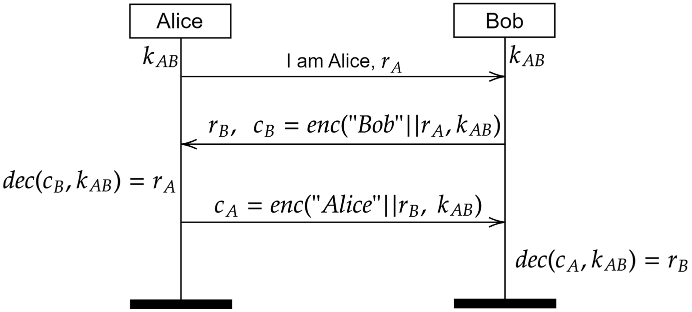
Symmetric Key Protocols



AP6 replay attack

- Trudy never learned the secret key yet tricked the protocol
- Repeating the process for both protocol users (for mutual authentication) does not guarantee security
- Seemingly benign changes in the protocol can break the protocol!

Symmetric Key Protocols



AP7

- Encrypting the user's name together with the nonce can prevent this attack

Authentication with Public Key Cryptography

Public Key Protocols

- For symmetric cryptography to work, Alice and Bob must share the k_{AB} in advance. Can we use public key cryptography to avoid this problem?
- Alice has a public key t_A and a private key s_A

$$\text{Alice's key pair: } (t_A, s_A)$$

- Everyone knows the public key t_A but only Alice knows the private key s_A

Public Key Protocols

- To send to Alice a message m

$$\text{You encrypt } m \text{ with } t_A : \quad c = enc(m, t_A)$$

$$\text{Alice is the only one that can decrypt } c \text{ with } s_A\text{:} \quad m = dec(c, s_A)$$

Public Key Protocols

- To send to Alice a message m

$$\text{You encrypt } m \text{ with } t_A: \quad c = enc(m, t_A)$$

$$\text{Alice is the only one that can decrypt } c \text{ with } s_A: \quad m = dec(c, s_A)$$

- Alice can also sign a message m

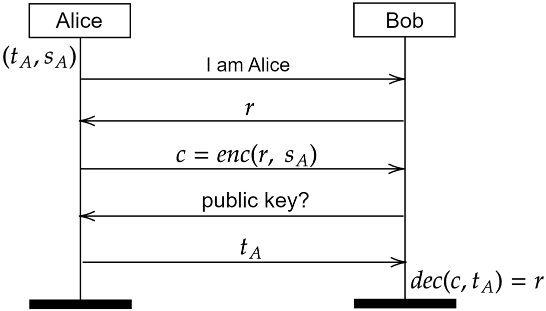
$$\text{Alice is the only one that can encrypt } m \text{ with } s_A: \quad d = enc(m, s_A)$$

She produces (m, d) i.e. the message m together with its signature d

Anyone can verify the signature d on message m using the public key t_A:

$$m = dec(d, t_A)$$

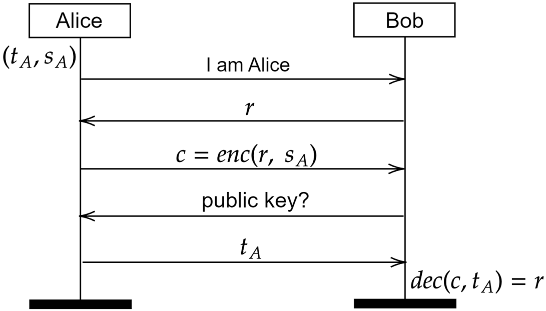
Public Key Protocols



AP8

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Alice

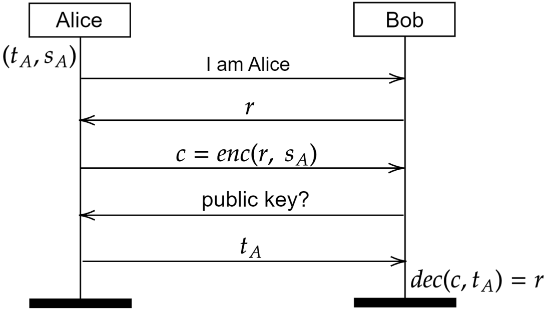
Public Key Protocols



AP8

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Alice
3. Alice encrypts r with her private key s_A and sends $c = enc(r, s_A)$ to Bob

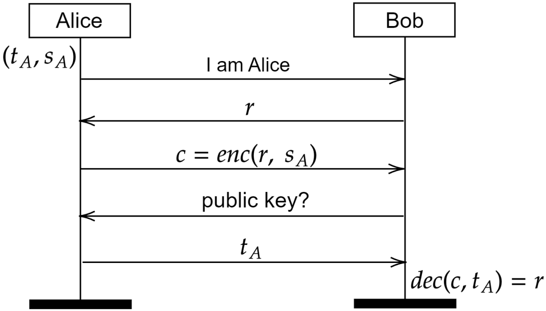
Public Key Protocols



AP8

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Alice
3. Alice encrypts r with her private key s_A and sends $c = enc(r, s_A)$ to Bob
4. Bob asks for her public key t_A
5. Alice sends the public key t_A to Bob

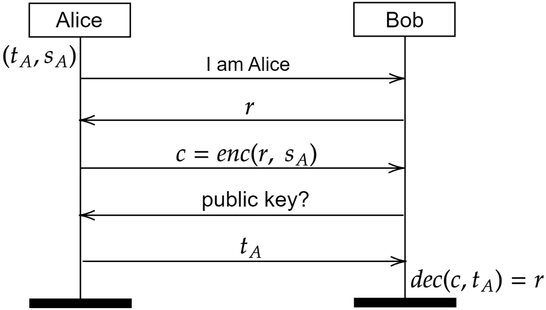
Public Key Protocols



AP8

1. Alice initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Alice
3. Alice encrypts r with her private key s_A and sends $c = enc(r, s_A)$ to Bob
4. Bob asks for her public key t_A
5. Alice sends the public key t_A to Bob
6. Bob decrypts c and verifies that $dec(c, t_A)$ is equal to his nonce r

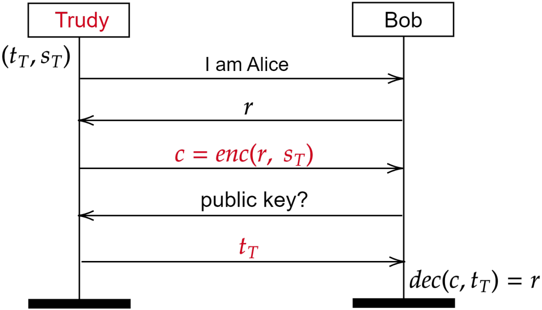
Public Key Protocols



AP8

- No need to share keys in advance!

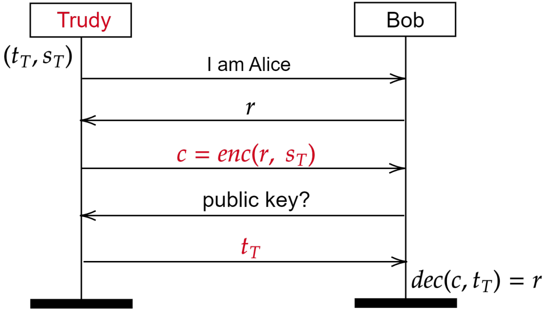
Public Key Protocols



AP8 man-in-the-middle attack

1. Trudy initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Trudy

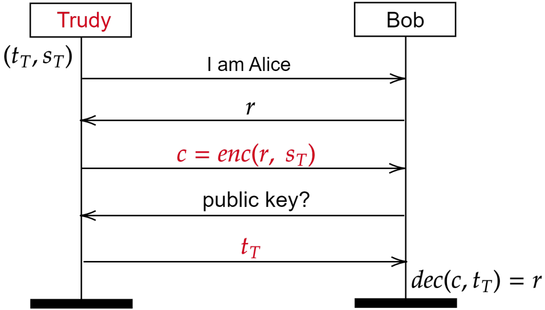
Public Key Protocols



AP8 man-in-the-middle attack

1. Trudy initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Trudy
3. Trudy encrypts r **with her own private key** s_T i.e. computes $c = enc(r, s_T)$

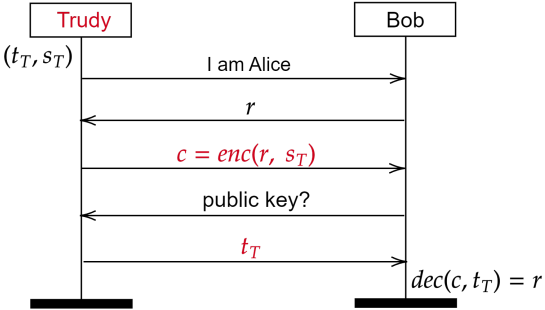
Public Key Protocols



AP8 man-in-the-middle attack

1. Trudy initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Trudy
3. Trudy encrypts r **with her own private key** s_T i.e. computes $c = enc(r, s_T)$
4. Bob asks for the public key of Alice
5. Trudy replies **with her own public key** t_T

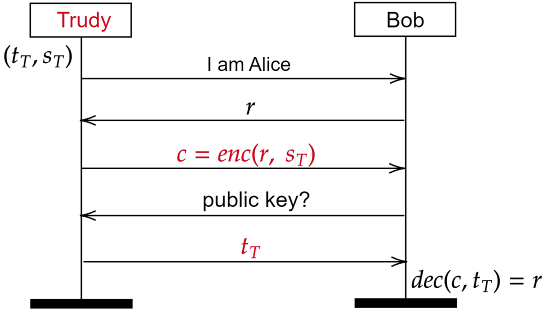
Public Key Protocols



AP8 man-in-the-middle attack

1. Trudy initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Trudy
3. Trudy encrypts r **with her own private key** s_T i.e. computes $c = enc(r, s_T)$
4. Bob asks for the public key of Alice
5. Trudy replies **with her own public key** t_T
6. Bob decrypts c and verifies that $dec(c, t_T)$ is equal to his nonce r

Public Key Protocols

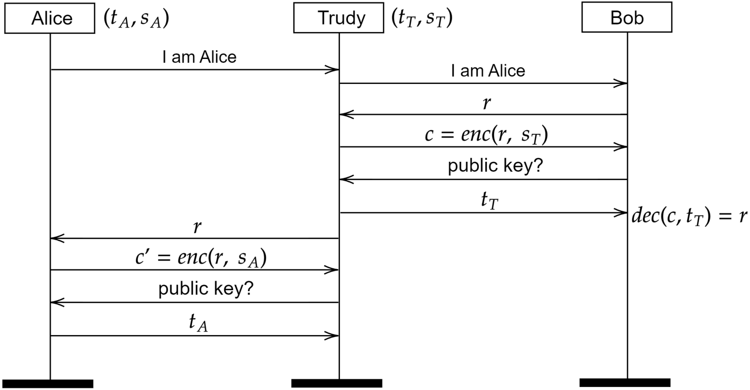


AP8 man-in-the-middle attack

1. Trudy initiates contact with Bob and claims that she is Alice
2. Bob sends nonce r to Trudy
3. Trudy encrypts r **with her own private key** s_T i.e. computes $c = enc(r, s_T)$
4. Bob asks for the public key of Alice
5. Trudy replies **with her own public key** t_T
6. Bob decrypts c and verifies that $dec(c, t_T)$ is equal to his nonce r

- Trudy in now authenticated as Alice

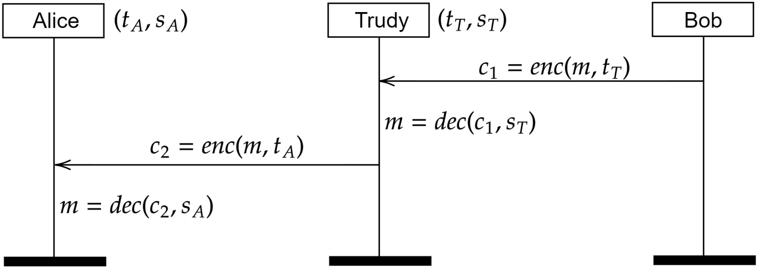
Public Key Protocols



AP8 man-in-the-middle attack

- Trudy mirrors this process on the side of Alice i.e. she pretends to be Bob to her
- Trudy convinced Alice that she is Bob and Bob that she is Alice

Public Key Protocols



AP8 man-in-the-middle attack

- Trudy can relay messages between Bob and Alice
- She can monitor the entire communication between them while being undetected i.e. she performs a **man-in-the-middle** attack

Public Key Protocols

- The underlying problem is that virtually anyone can publish a public key and claim to be Alice
- We use **certificate authorities (CA)** to link people to their public keys

Public Key Protocols

- The underlying problem is that virtually anyone can publish a public key and claim to be Alice
- We use **certificate authorities (CA)** to link people to their public keys

1. Alice wants to link herself to her public key t_A

Public Key Protocols

- The underlying problem is that virtually anyone can publish a public key and claim to be Alice
- We use **certificate authorities (CA)** to link people to their public keys

1. Alice wants to link herself to her public key t_A
2. She contacts a trustworthy CA and the CA signs her public key t_A using its own private key s_{CA}

$$\text{Signature: } d = enc(t_A, s_{CA}), \quad \text{Signed public key: } (t_A, d)$$

Public Key Protocols

- The underlying problem is that virtually anyone can publish a public key and claim to be Alice
- We use **certificate authorities (CA)** to link people to their public keys

1. Alice wants to link herself to her public key t_A
2. She contacts a trustworthy CA and the CA signs her public key t_A using its own private key s_{CA}

$$\text{Signature: } d = enc(t_A, s_{CA}), \quad \text{Signed public key: } (t_A, d)$$

3. Now anyone can verify the CA's signature using the CA's public key t_{CA} and be certain that t_A is Alice's public key

$$\text{Verify that: } t_A = dec(d, t_{CA})$$

Public Key Protocols

- The underlying problem is that virtually anyone can publish a public key and claim to be Alice
- We use **certificate authorities (CA)** to link people to their public keys

1. Alice wants to link herself to her public key t_A
2. She contacts a trustworthy CA and the CA signs her public key t_A using its own private key s_{CA}

$$\text{Signature: } d = enc(t_A, s_{CA}), \quad \text{Signed public key: } (t_A, d)$$

3. Now anyone can verify the CA's signature using the CA's public key t_{CA} and be certain that t_A is Alice's public key

$$\text{Verify that: } t_A = dec(d, t_{CA})$$

- This solution implies that the CA is **trusted** by all communicating parties

Zero-Knowledge Proofs

ZKP

- Assume that Peggy wants to prove to Victor that she knows something
- ...but she does not want to reveal this to anyone, Victor included!

ZKP

- Assume that Peggy wants to prove to Victor that she knows something
- ...but she does not want to reveal this to anyone, Victor included!

e.g. she wants to prove that she lives in this country without showing her identity and revealing name/surname or BSN

e.g. she wants to reveal that her age is greater or equal to 18 but not the exact age in years

e.g. she want to vote in an anonymous yet verifiable manner

ZKP

- Assume that Peggy wants to prove to Victor that she knows something
- ...but she does not want to reveal this to anyone, Victor included!

  e.g. she wants to prove that she lives in this country without showing her identity and revealing name/surname or BSN

  e.g. she wants to reveal that her age is greater or equal to 18 but not the exact age in years

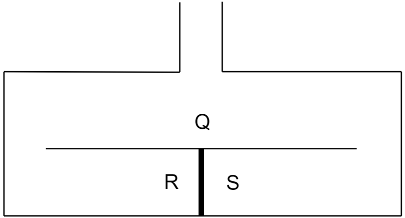
  e.g. she want to vote in an anonymous yet verifiable manner

- This seemingly impossible problem can be solved using an interactive **zero-knowledge proof (ZKP)**

ZKP

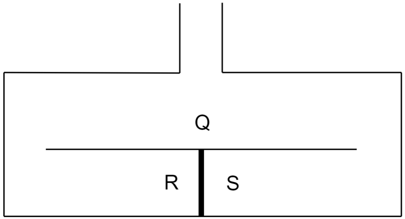
The zero-knowledge cave

- Consider a cave with 3 points: Q, R, S
- Between R and S there is a door that opens only when the right passphrase is uttered ("open sesame")
- If someone stands at point Q they cannot hear the passphrase uttered at point R or point S



ZKP

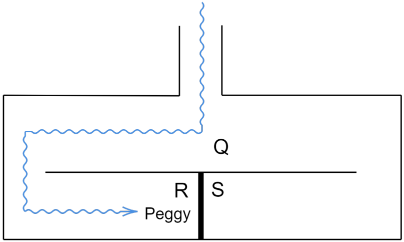
- Peggy (the prover) knows the secret passphrase but Victor (the verifier) doesn't
- Peggy must show to Victor that she knows the passphrase that opens the door



ZKP

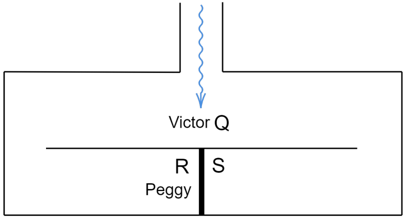
1. Peggy enters the cave and flips a coin. On ‘heads’ she goes to point R and on ‘tails’ she goes to point S

   e.g. below Peggy’s coin flip is ‘heads’ so she is positioned at R



ZKP

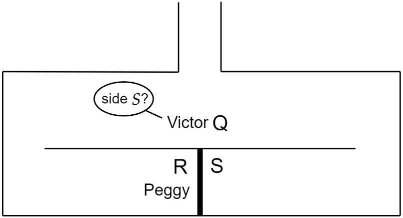
2. Victor enters the cave and positions himself on point Q where he cannot see or hear Peggy



ZKP

3. Victor flips a coin. On 'heads' he asks Peggy to appear from side R and on 'tails' he asks Peggy to appear from side S

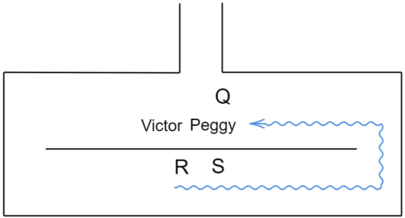
   e.g. below Victor's coin flip is 'tails' thus Peggy must appear from side S



ZKP

4. If needed, Peggy can use the passphrase to open the door and appear form the side Victor is asking

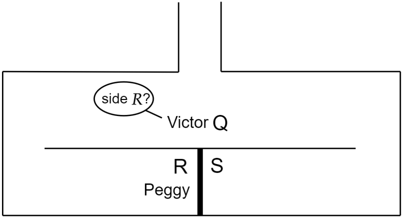
   e.g. below Victor is expecting her from side S thus she utters the passphrase, opens the door and meets Victor in Q



ZKP

- With probability 50% Peggy may not need to utter the passphrase

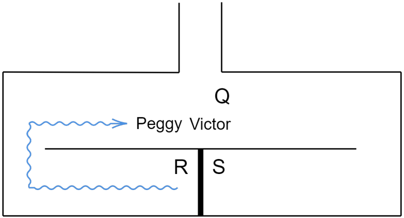
  e.g. below Peggy is placed at position R. Victor's coin flip is 'heads' so Peggy can just return from side R without opening the door



ZKP

- With probability 50% Peggy may not need to utter the passphrase

  e.g. below Peggy is placed at position *R*. Victor's coin flip is 'heads' so Peggy can just return from side *R* without opening the door



ZKP

- Victor and Peggy will repeat the protocol n times
- Since Peggy knows the passphrase, she will always appear from the side Victor asked her to

- The probability of tricking Victor is $(1/2)^n$ which can be made arbitrarily small
- Thus Victor is convinced that Peggy knows the secret passphrase

TLS Protocol

TLS Protocol

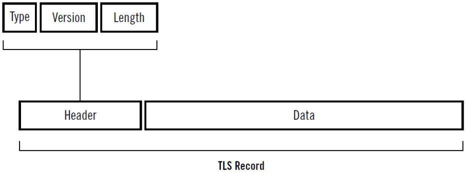
Record Protocol

- The core of TLS is the record protocol which transports and encrypts all connection messages

TLS Protocol

Record Protocol

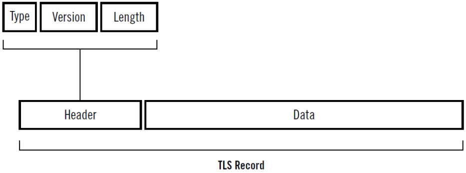
- The core of TLS is the record protocol which transports and encrypts all connection messages
- Structure of TLS 1.2 records:



TLS Protocol

Record Protocol

- The core of TLS is the record protocol which transports and encrypts all connection messages
- Structure of TLS 1.2 records:

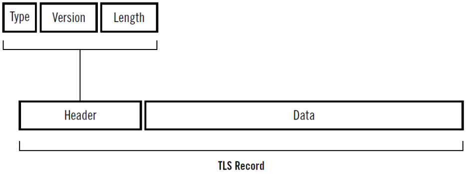


- The `Version` specifies the current TLS version
- The `Length` specifies the record length ($\leq 2^{14}$ bytes)

TLS Protocol

Record Protocol

- The core of TLS is the record protocol which transports and encrypts all connection messages
- Structure of TLS 1.2 records:



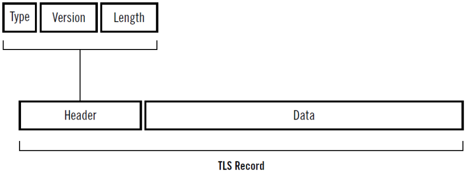
- The `Version` specifies the current TLS version
- The `Length` specifies the record length ($\leq 2^{14}$ bytes)
- The `Type` specifies the subprotocol

```
enum { change_cipher_spec (20), alert (21),
handshake (22), application_data (23) } ContentType;
```

TLS Protocol

Record Protocol

- The core of TLS is the record protocol which transports and encrypts all connection messages
- Structure of TLS 1.2 records:



- The `Version` specifies the current TLS version
- The `Length` specifies the record length ($\leq 2^{14}$ bytes)
- The `Type` specifies the subprotocol

```
enum { change_cipher_spec (20), alert (21),
handshake (22), application_data (23) } ContentType;
```

- The record keeps also a unique 64-bit sequence number to avoid replays

TLS Protocol

Record protocol

- Transports opaque data buffers submitted to it by other protocol, fragmenting and combining if needed

TLS Protocol

Record protocol

- Transports opaque data buffers submitted to it by other protocol, fragmenting and combining if needed
- Provides encryption and integrity validation. New TLS sessions, initially transport messages without any protection. Once the handshake is complete, the record layer provides security.

TLS Protocol

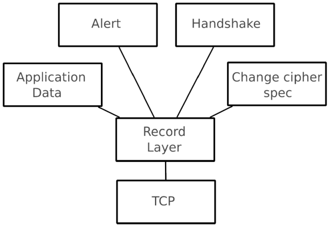
Record protocol

- Transports opaque data buffers submitted to it by other protocol, fragmenting and combining if needed
- Provides encryption and integrity validation. New TLS sessions, initially transport messages without any protection. Once the handshake is complete, the record layer provides security.
- Can provide Compression (removed in TLS 1.3) and Extensions

TLS Protocol

Record protocol

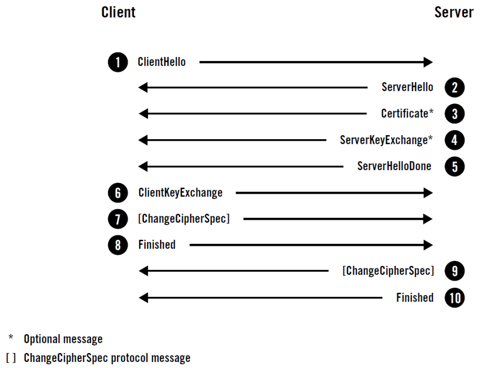
- Transports opaque data buffers submitted to it by other protocol, fragmenting and combining if needed
- Provides encryption and integrity validation. New TLS sessions, initially transport messages without any protection. Once the handshake is complete, the record layer provides security.
- Can provide Compression (removed in TLS 1.3) and Extensions
- Utilizes four main subprotocols: handshake, alert, application data, change cipher spec



TLS Protocol

Handshake protocol

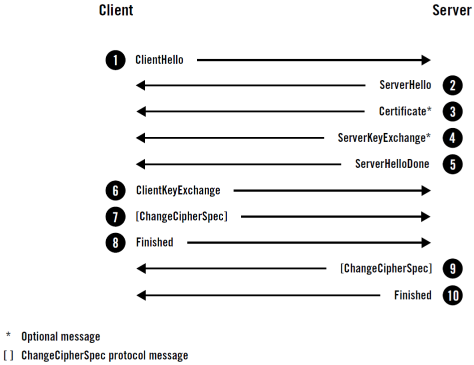
- A full handshake starts with `ClientHello` where the Client submits its connection and security parameters to the Server
- The Server replies with `ServerHello` that selected the parameters



TLS Protocol

Handshake protocol

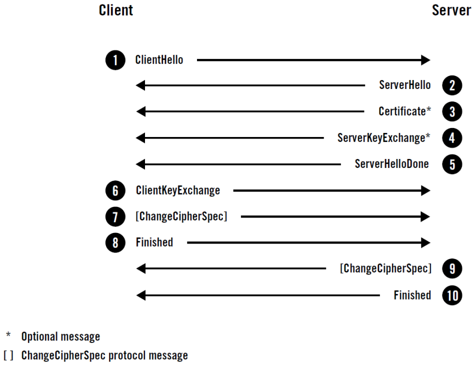
- The Server sends a signed `Certificate` for authentication and the Client verifies the signature
- The Client and Server use public key cryptography to exchange keys. The Server sends the needed information via `ServerKeyExchange` and notifies that is done.



TLS Protocol

Handshake protocol

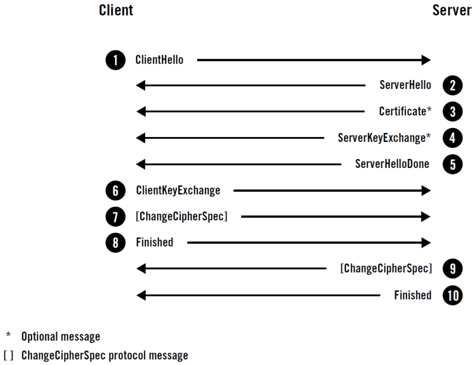
- The Client sends the needed information via `ClientKeyExchange`, derives the `premaster secret` and the `master secret`
- Now the Client switches encryption on and notifies the Server via `ChangeCipherSpec`



TLS Protocol

Handshake protocol

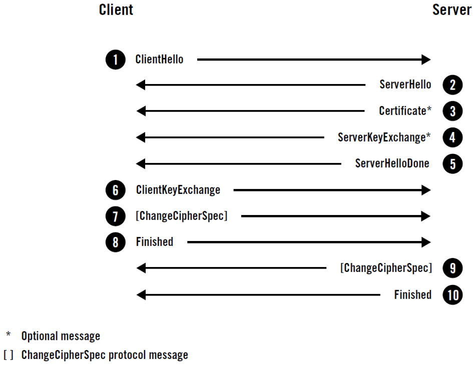
- To verify the integrity of the hanshake so far, the Client computes a `MAC(handshake messages)` and sends it to the Server via `Finished`
- The Server derives the `premaster secret` and the `master secret`, switches encryption on and notifies the Client via `ChangeCipherSpec`



TLS Protocol

Handshake protocol

- To verify the integrity of the hanshake, the Server computes a `MAC(handshake messages)` and sends it to the Client via `Finished`
- A TLS session between Client-Server may now begin using a symmetric algorithm for bulk encryption of application data



Renegotiation Attack on TLS 1.0

Renegotiation Attack

Session Resumptions and Renegotiation

- Session Resumption can resume a previous TLS session identified by its Session ID, using the same previously generated keys
- Session resumptions improve performance by avoiding a new handshake with expensive public key encryption/decryption
- For session resumption, the client sends a custom `ClientHello` message

Renegotiation Attack

Session Resumptions and Renegotiation

- Session Resumption can resume a previous TLS session identified by its Session ID, using the same previously generated keys
- Session resumptions improve performance by avoiding a new handshake with expensive public key encryption/decryption
- For session resumption, the client sends a custom `ClientHello` message

- Session renegotiation performs a new negotiation of algorithms and generation of new keys between client and server
- Session renegotiation is a useful feature to change the security parameters of a TLS session
- For session renegotiation, the client sends a custom `ClientHello` message

Renegotiation Attack

Session Resumptions and Renegotiation

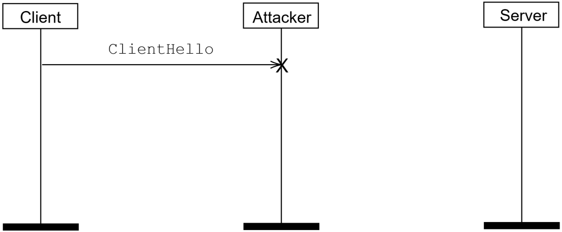
- Session Resumption can resume a previous TLS session identified by its Session ID, using the same previously generated keys
- Session resumptions improve performance by avoiding a new handshake with expensive public key encryption/decryption
- For session resumption, the client sends a custom `ClientHello` message

- Session renegotiation performs a new negotiation of algorithms and generation of new keys between client and server
- Session renegotiation is a useful feature to change the security parameters of a TLS session
- For session renegotiation, the client sends a custom `ClientHello` message

- The renegotiation feature is problematic and caused the Renegotiation Attack

Renegotiation Attack

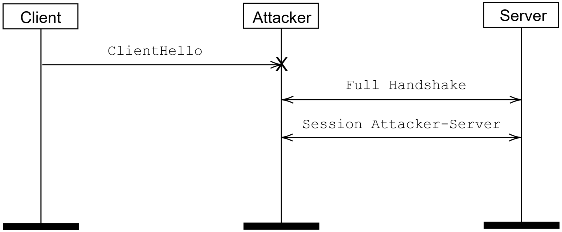
Step 1: Intercept TLS connection



- The Client (victim) initiates a TLS connection to the Server
- The Client sends a `ClientHello` message to initiate the Handshake
- The Attacker intercepts the `ClientHello` message and stops it temporarily

Renegotiation Attack

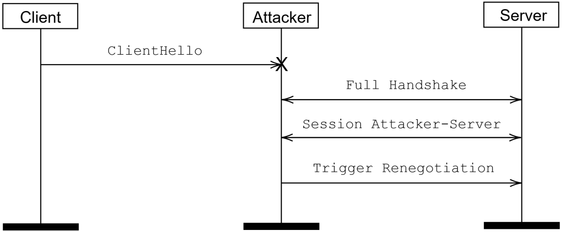
Step 2: Open new TLS connection



- The Attacker initiates a different TLS connection to the Server
- The Attacker sends a `ClientHello` message and performs a full handshake to establish the Attacker-Server TLS session

Renegotiation Attack

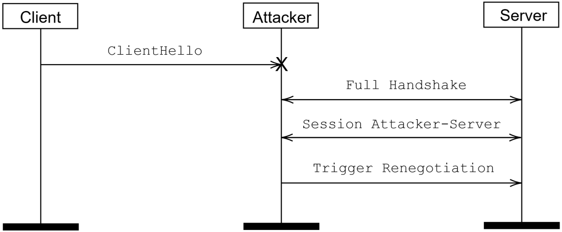
Step 3: Renegotiate the TLS connection



- The Attacker can trigger a renegotiation of the Attacker-Server TLS session. How?
  - Some servers support client-initiated renegotiations
  - Some servers have 2 layers of authentication: server-only and server-client (mutual) authentication
  - Long-term sessions renegotiate to refresh the keys

Renegotiation Attack

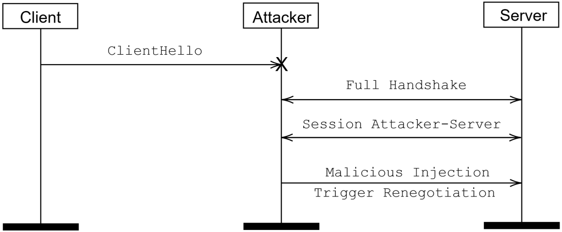
Step 3: Renegotiate the TLS connection



- The Attacker can trigger a renegotiation of the Attacker-Server TLS session. How?
  - Some servers support client-initiated renegotiations
  - Some servers have 2 layers of authentication: server-only and server-client (mutual) authentication
  - Long-term sessions renegotiate to refresh the keys
- Side note: TLS renegotiation offers also opportunity for DoS (slow public key crypto and RTTs) thus should be limited

Renegotiation Attack

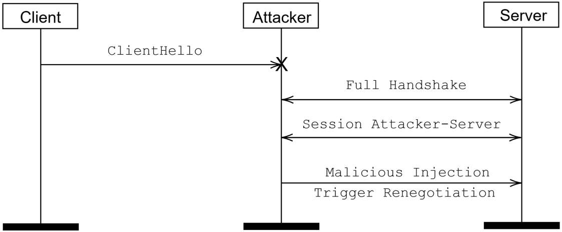
Step 4: Malicious Injection



- Together with the renegotiation trigger the Attacker injects a malicious application request
  - e.g. reset the password of the Client or transfer funds from the Client's account on the Server to the Attacker

Renegotiation Attack

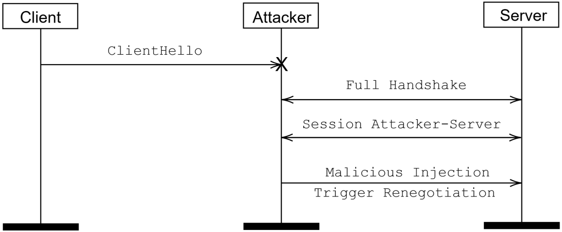
Step 4: Malicious Injection



- Together with the renegotiation trigger the Attacker injects a malicious application request
  - e.g. reset the password of the Client or transfer funds from the Client's account on the Server to the Attacker
- Shouldn't such malicious requests fail?

Renegotiation Attack

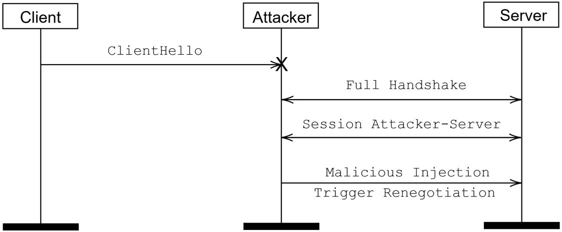
Step 4: Malicious Injection



- Together with the renegotiation trigger the Attacker injects a malicious application request
  - e.g. reset the password of the Client or transfer funds from the Client's account on the Server to the Attacker
- Shouldn't such malicious requests fail? Yes! Such requests would need to be authorized by the Client

Renegotiation Attack

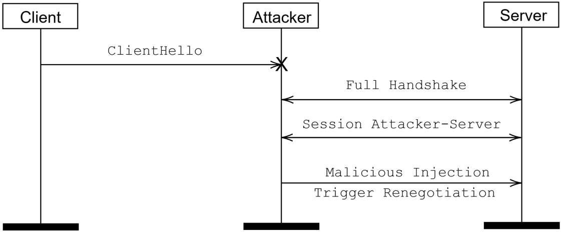
Step 4: Malicious Injection



- Together with the renegotiation trigger the Attacker injects a malicious application request
  - e.g. reset the password of the Client or transfer funds from the Client's account on the Server to the Attacker
- Shouldn't such malicious requests fail? Yes! Such requests would need to be authorized by the Client
- When will the malicious requests be executed by the Server?

Renegotiation Attack

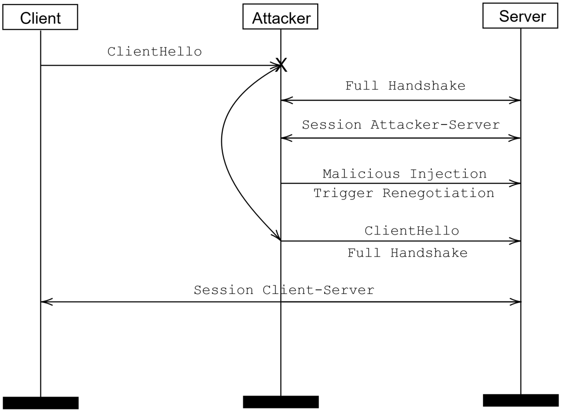
Step 4: Malicious Injection



- Together with the renegotiation trigger the Attacker injects a malicious application request
  - e.g. reset the password of the Client or transfer funds from the Client's account on the Server to the Attacker
- Shouldn't such malicious requests fail? Yes! Such requests would need to be authorized by the Client
- When will the malicious requests be executed by the Server? After the renegotiation concludes! The renegotiation takes precedence over application data and the malicious request gets buffered in the Server.

Renegotiation Attack

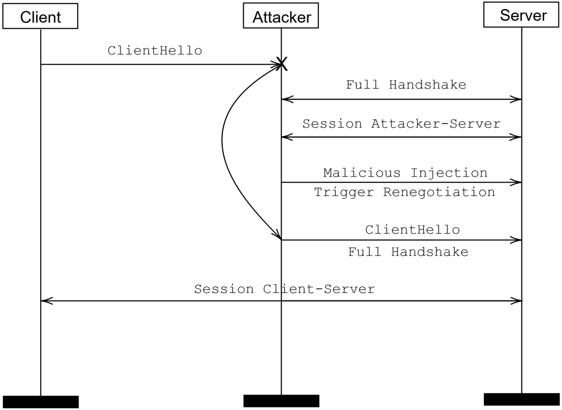
Step 5: Replay ClientHello



- The Attacker replays the original `ClientHello` message from the Client

Renegotiation Attack

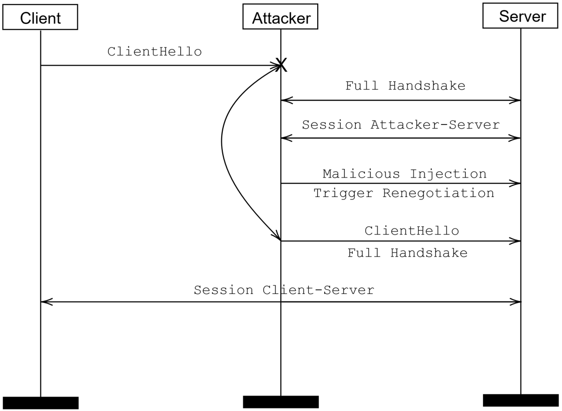
Step 5: Replay ClientHello



- The Attacker replays the original `ClientHello` message from the Client
- The Client and the Server perform a full Handshake and establish their own TLS session. The Client believes that an initial handshake was executed.

Renegotiation Attack

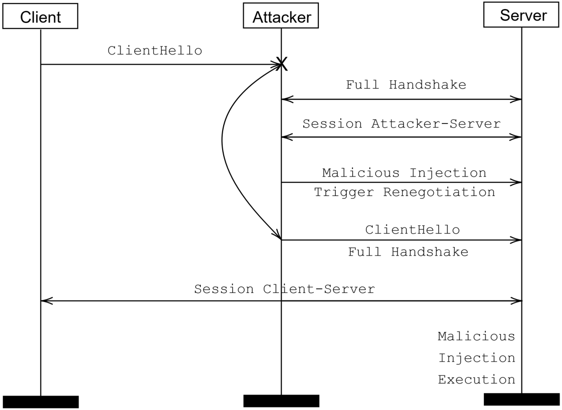
Step 5: Replay ClientHello



- The Attacker replays the original `ClientHello` message from the Client
- The Client and the Server perform a full Handshake and establish their own TLS session. The Client believes that an initial handshake was executed.
- Note that the Attacker has no access to the Client-Server session

Renegotiation Attack

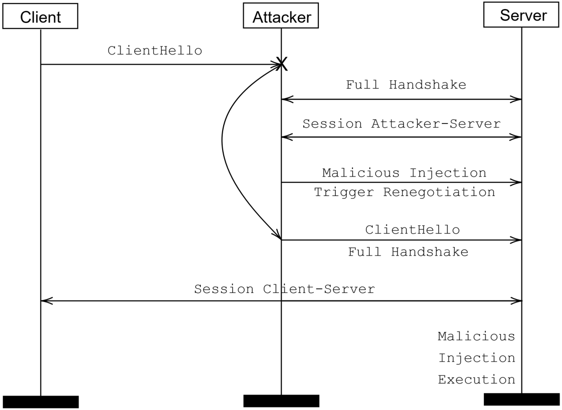
Step 6: Execute the Malicious Injection



- The renegotiation concluded so the Server will execute the buffered application requests

Renegotiation Attack

Step 6: Execute the Malicious Injection



- The renegotiation concluded so the Server will execute the buffered application requests
- The Server is tricked to believe that the buffered malicious request comes from the Client-Server session thus gets executed with the Client's authorization!

Renegotiation Attack

Vulnerability Causes

- The `ClientHello` message is indistinguishable between initial handshakes and renegotiation handshakes
- The application layer does not get notified when the encryption layer changes

Renegotiation Attack

Vulnerability Causes

- The `ClientHello` message is indistinguishable between initial handshakes and renegotiation handshakes
- The application layer does not get notified when the encryption layer changes

Vulnerability Impact

- Renegotiation attacks break TLS 1.0 security and integrity
- The attack affected mostly HTTP but also SMTP, FTPS protocols
- Exploitation requires custom work on the application protocol but produced strong results such as execution of arbitrary HTTP requests, credential theft, user redirection to malicious websites, making the client appear to attack the server with malicious injections
  `https://www.g-sec.lu/practicaltls.pdf`
- Renegotiation attacks were followed by more potent versions like the TripleHandshake attack

Renegotiation Attack

Vulnerability Mitigation

- TLS 1.3 does not support session renegotiation
- Disable renegotiation in older TLS versions

Renegotiation Attack

Vulnerability Mitigation

- TLS 1.3 does not support session renegotiation
- Disable renegotiation in older TLS versions

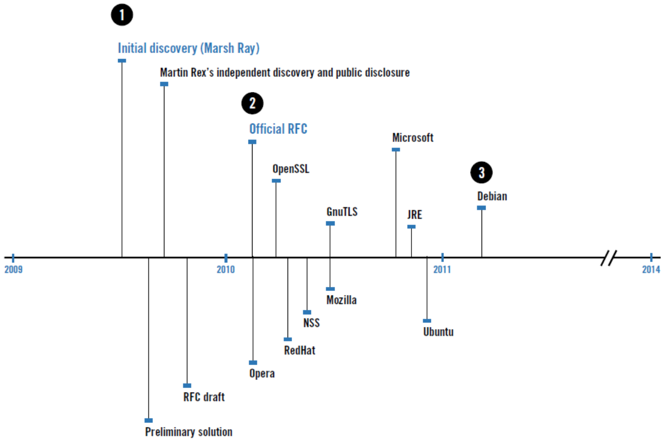
- Use the Renegotiation Indication Extension on TLS [RFC 5746] so that the server can:
  - Distinguish between the initial handshake and the renegotiation
  - Prevent the buffered requests from "splicing" across sessions

$$\texttt{RenegotiationInfo} = \begin{cases} \texttt{empty}, & \text{if initial handshake} \\ \texttt{MAC(previous handshakes)}, & \text{if renegotiation} \end{cases}$$

  - Replaying the client's original `ClientHello` will be detected since the `RenegotiationInfo` will be empty
  - Setting the `RenegotiationInfo` to the `MAC(Attacker-Server handshake)` will be detected during the client-server handshake

Renegotiation Attack

Vulnerability Mitigation Timeline



- 6 months to fix the protocol
- 12 additional months to patch TLS libraries and operating systems
- 24 additional months to patch almost everything