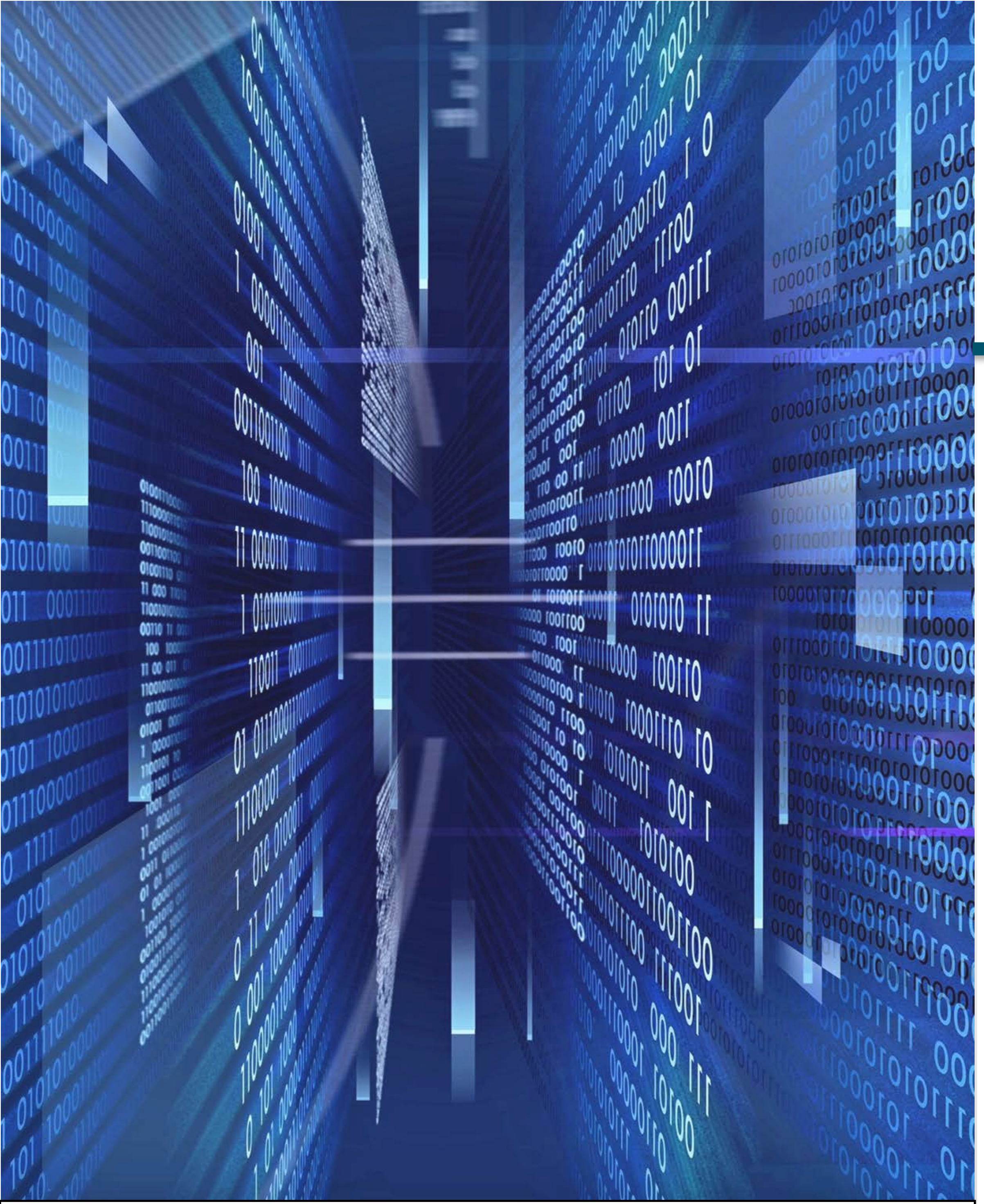


cybersecurity centre

Academic Centre of Excellence

COMP6224
Secure Communication part 2 – Security Protocols: SSL/TLS and Kerberos

Dr Federico Lombardi

f.lombardi@soton.ac.uk

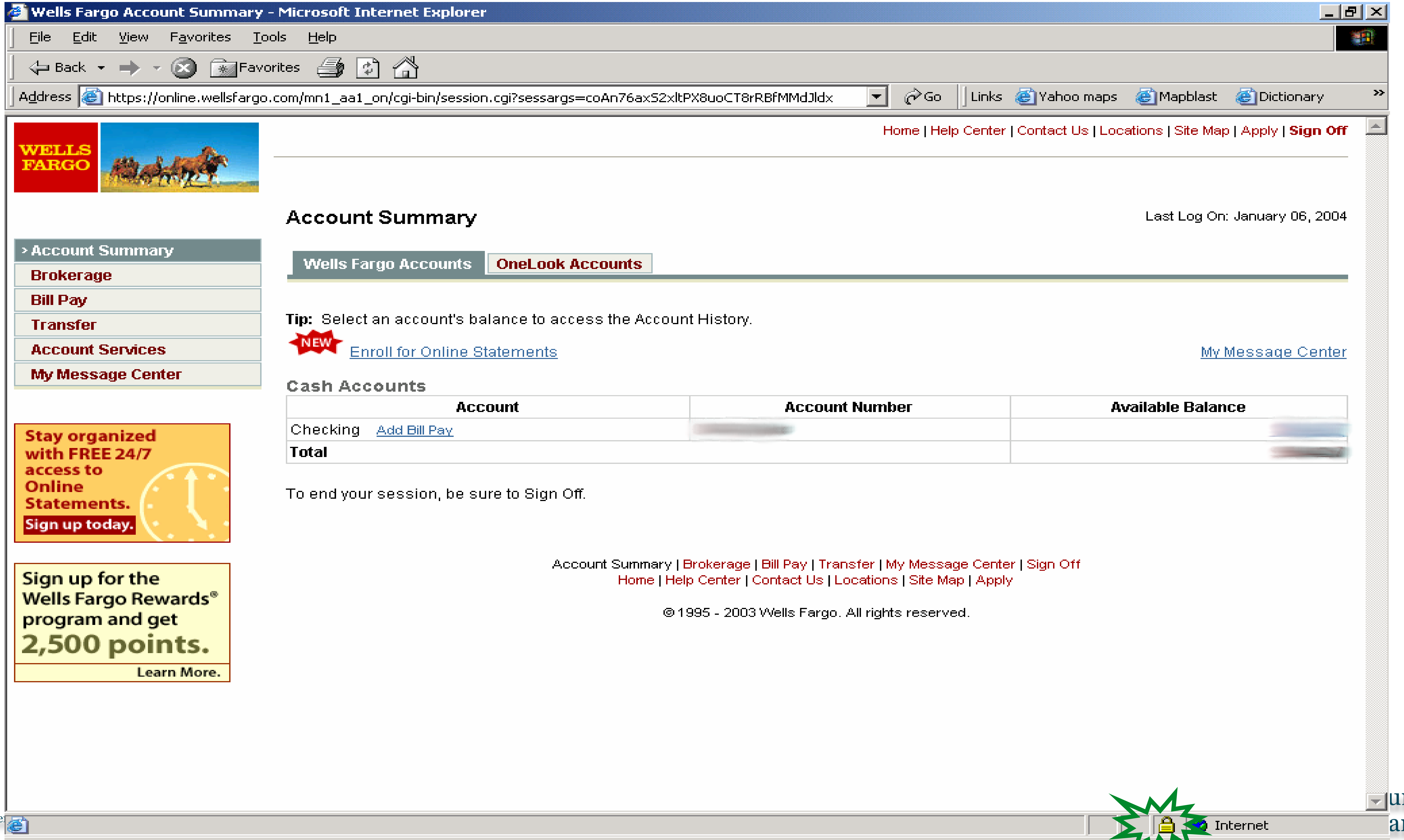
CyberSecuritySoton.org [w]

@CybSecSoton [fb & tw]

SSL/TLS

- Transport Layer Security Protocol
  - De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
  - In practice, used to protect information transmitted between browsers and Web servers
  - Deployed in nearly every Web browser

SSL/TLS in the real world



TLS Basics

- TLS consists of two protocols

- **Handshake protocol**
  - Use public-key cryptography to establish a shared secret key between the client and the server

- **Record protocol**
  - Use the secret key established in the handshake protocol to protect communication between the client and the server

The Handshake protocol

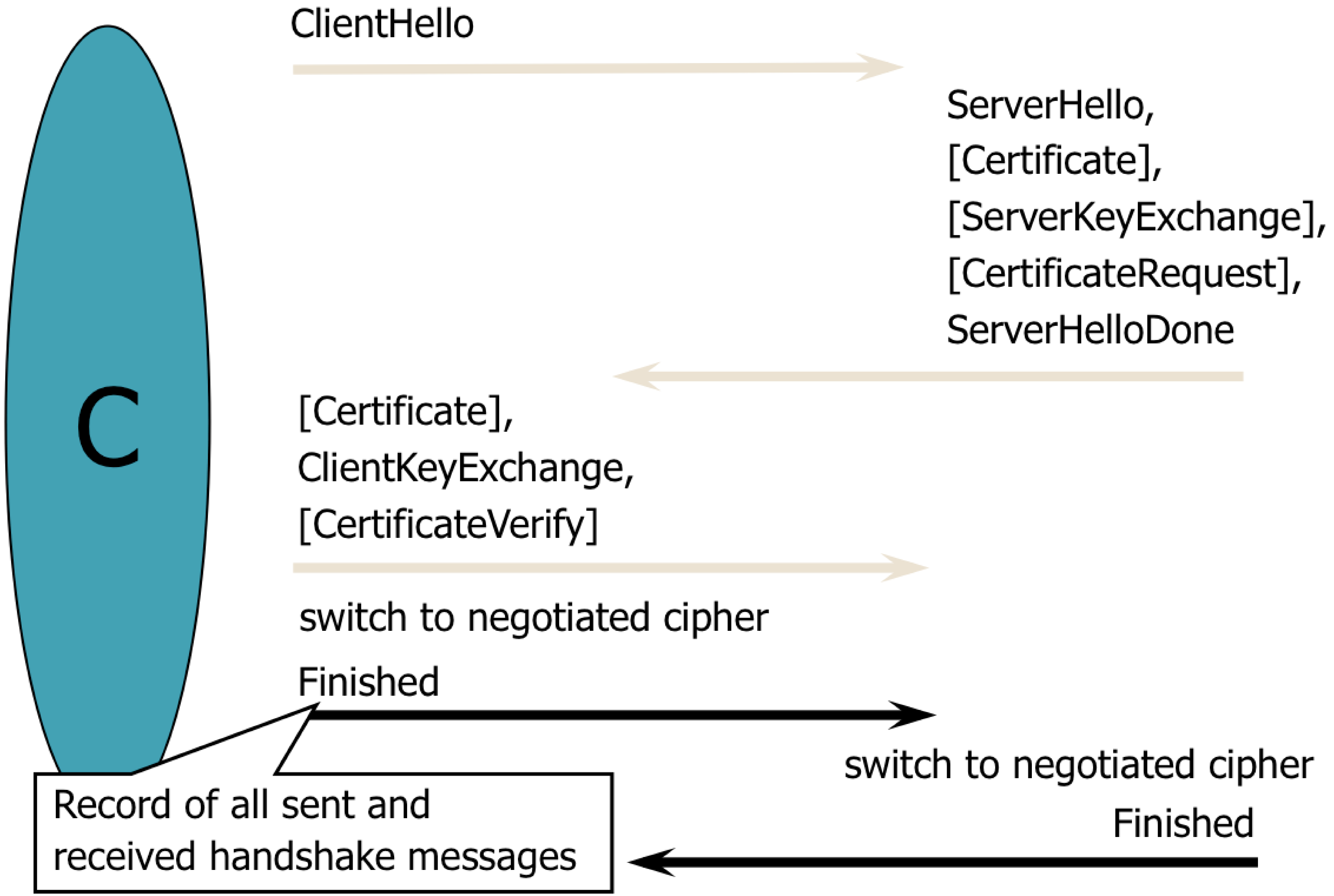
- Two parties:
  - **client** (browser)
  - **server** (Web site)

- Negotiate version of the protocol and the set of cryptographic algorithms to use
  - Interoperability between different implementations of the protocol

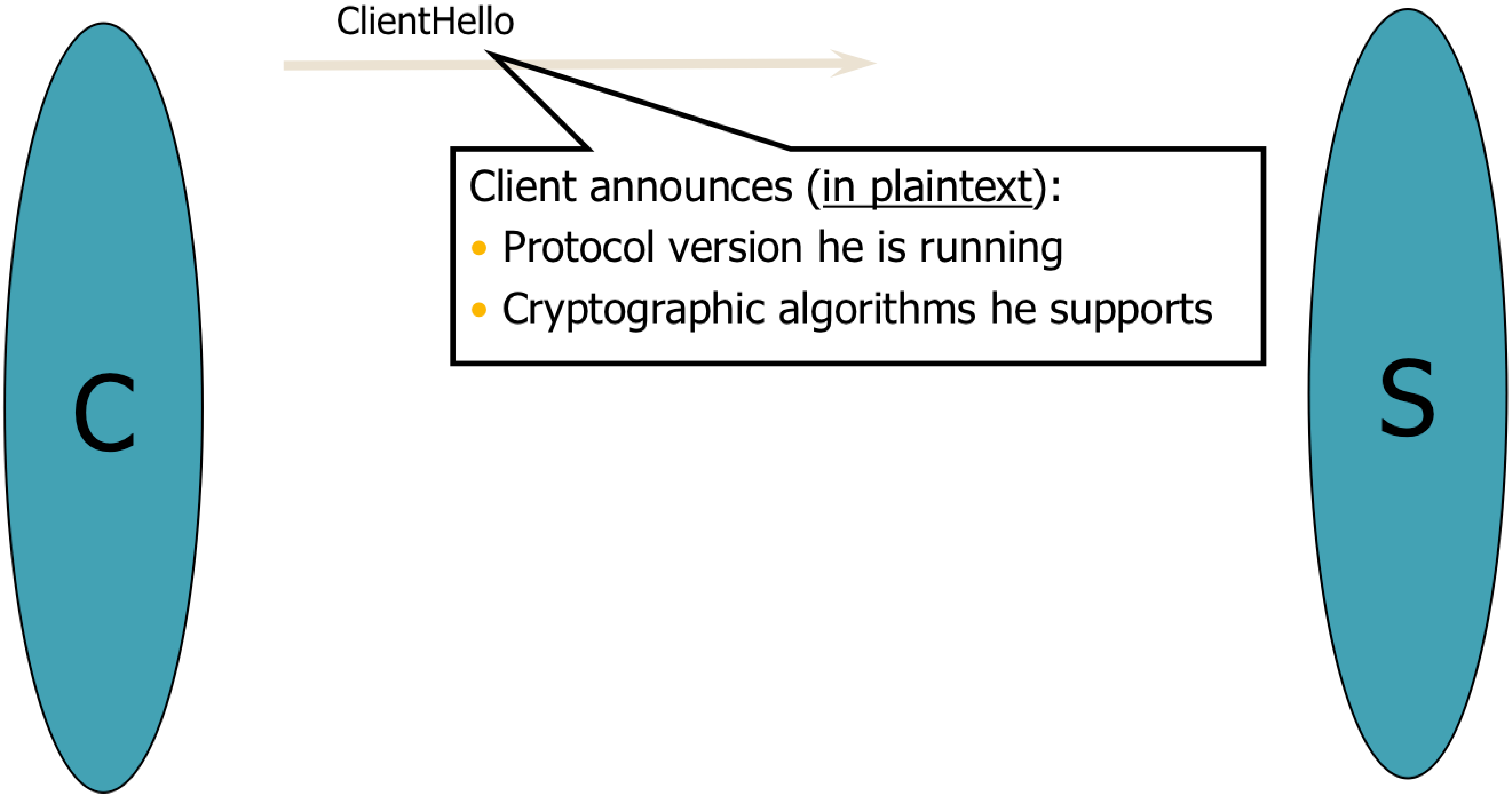
- Authenticate server and client (optional)
  - Use digital certificates to learn each other public keys and
  - Verify each other’s identity

- Use public keys to establish a shared secret

The Handshake prtocol



The Handshake protocol



Client Hello

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

Session id (if the client wants to resume an old session)

Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

Server Hello

C

Version$_c$, suite$_c$, N$_c$ →

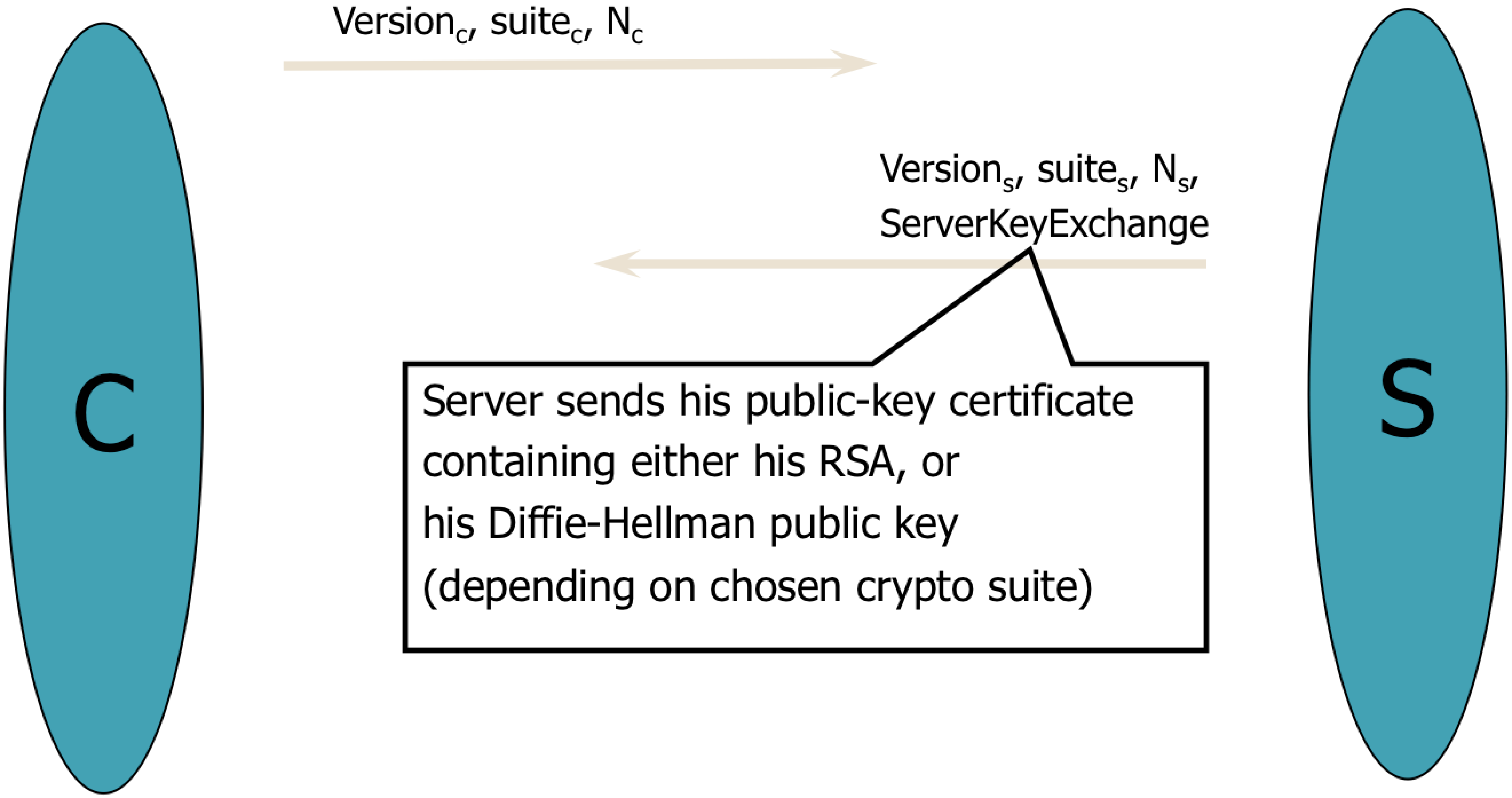
← ServerHello

S

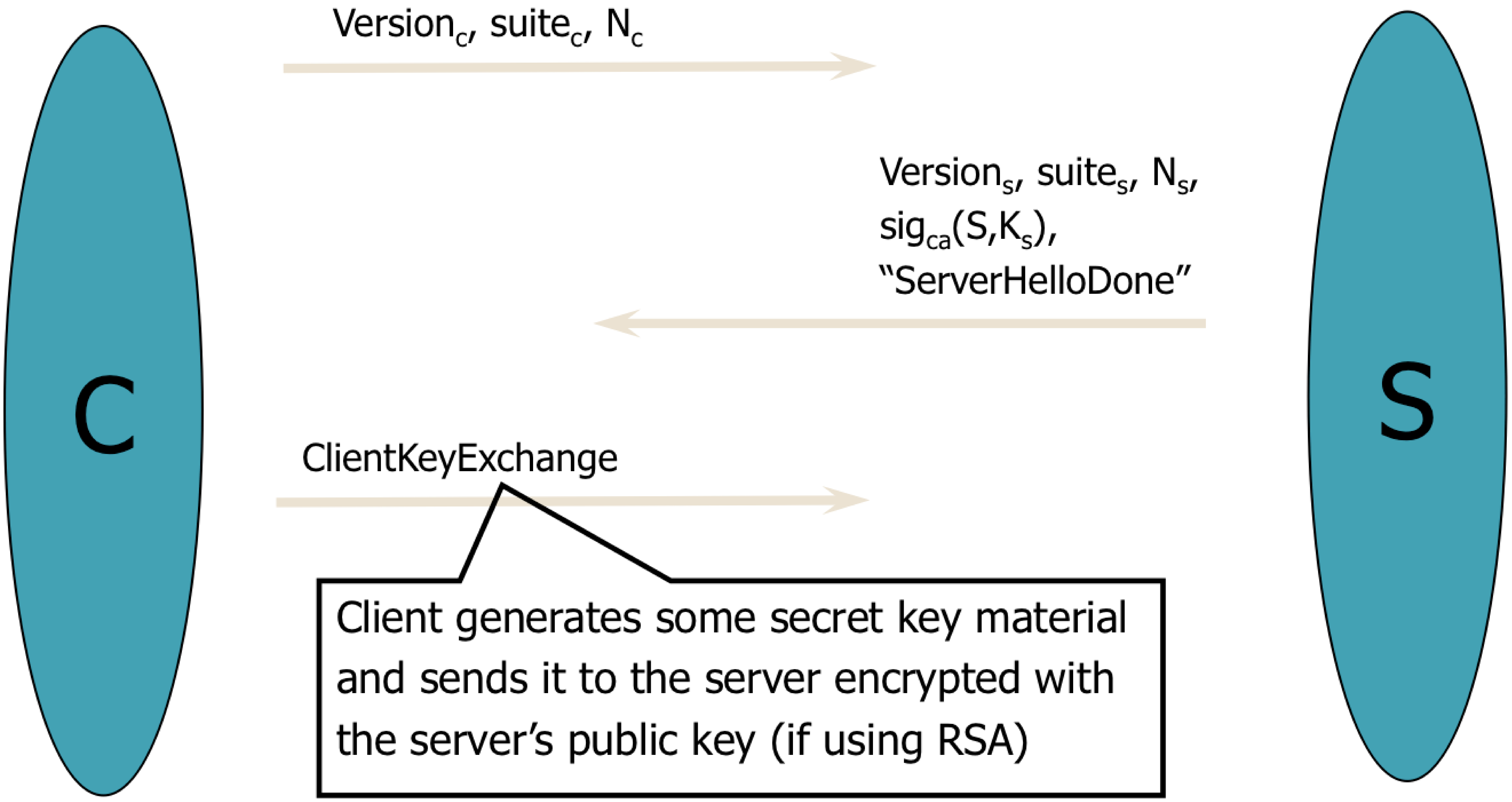
Server responds (<u>in plaintext</u>) with:

- Highest protocol version supported by both client and server
- Strongest cryptographic suite selected from those offered by the client

ServerKeyExchange



ClientKeyExchange



How the client (browser) check the certificates?

1. The web browser (client) receive the **server's certificate**, containing the **public key** of the web server.
2. This certificate is signed with the **private key** of a trusted **CA**.
3. The browser has installed (by default) the **public keys** of all the major **CA**.
4. It uses such **CA’s public key** to verify that the web server's certificate was signed by the trusted CA.

Heartbleed

- Hearthbleed was a bug into **OpenSSL**, a common used implementation of SSL/TLS v1.0.1 and 1.0.1f

- The bug was discovered in **2014** but it was present in released code **since 2012**

- A carefully crafted packet causes OpenSSL to read and return portions of the vulnerable server's server memory
  - Encryption keys, passwords, and other sensitive information
  - Confidentialty problem!

- Recent Shodan report says 200,000 web sites are still vulnerable

Heartbleed explained

- The **Heartbeat** protocol allows the two parties (client and server) to know if they are still alive
- How the protocol works:
  - The client sends to the server a message saying: “***if you are still alive reply*** xxx”, xxx # letters
  - The client check the response of the server
  - This is repeated periodically

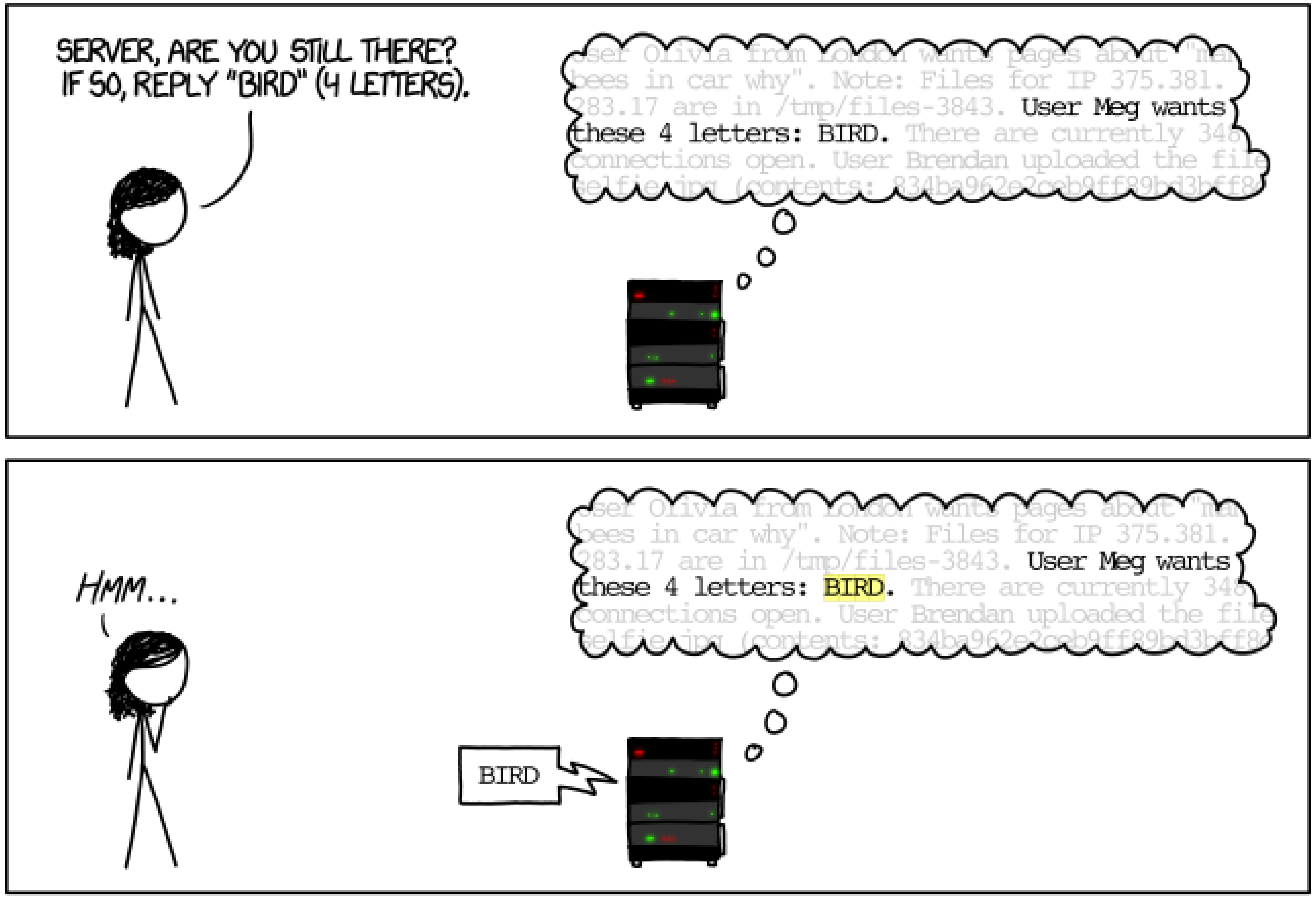


Image from ***xkcd.com***

Heartbleed explained

- They continue to keep the connection opened through the Heartbeat protocol

- What is the problem?

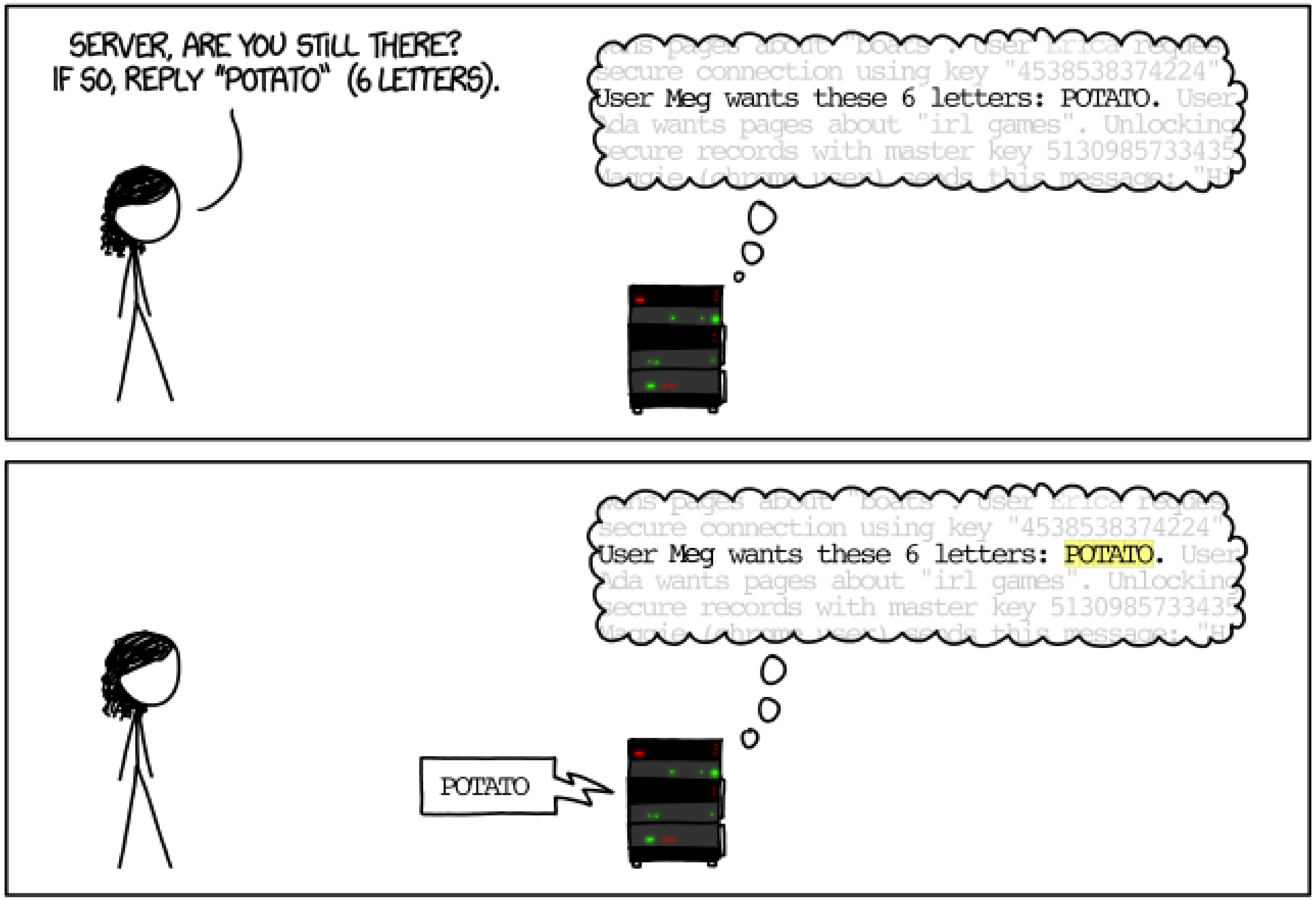


Image from ***xkcd.com***

Heartbleed explained

- **Problem:** The server does not check if the length suggested by the client matches with the word the client asked

- Example:
  word = hat
  length = 500

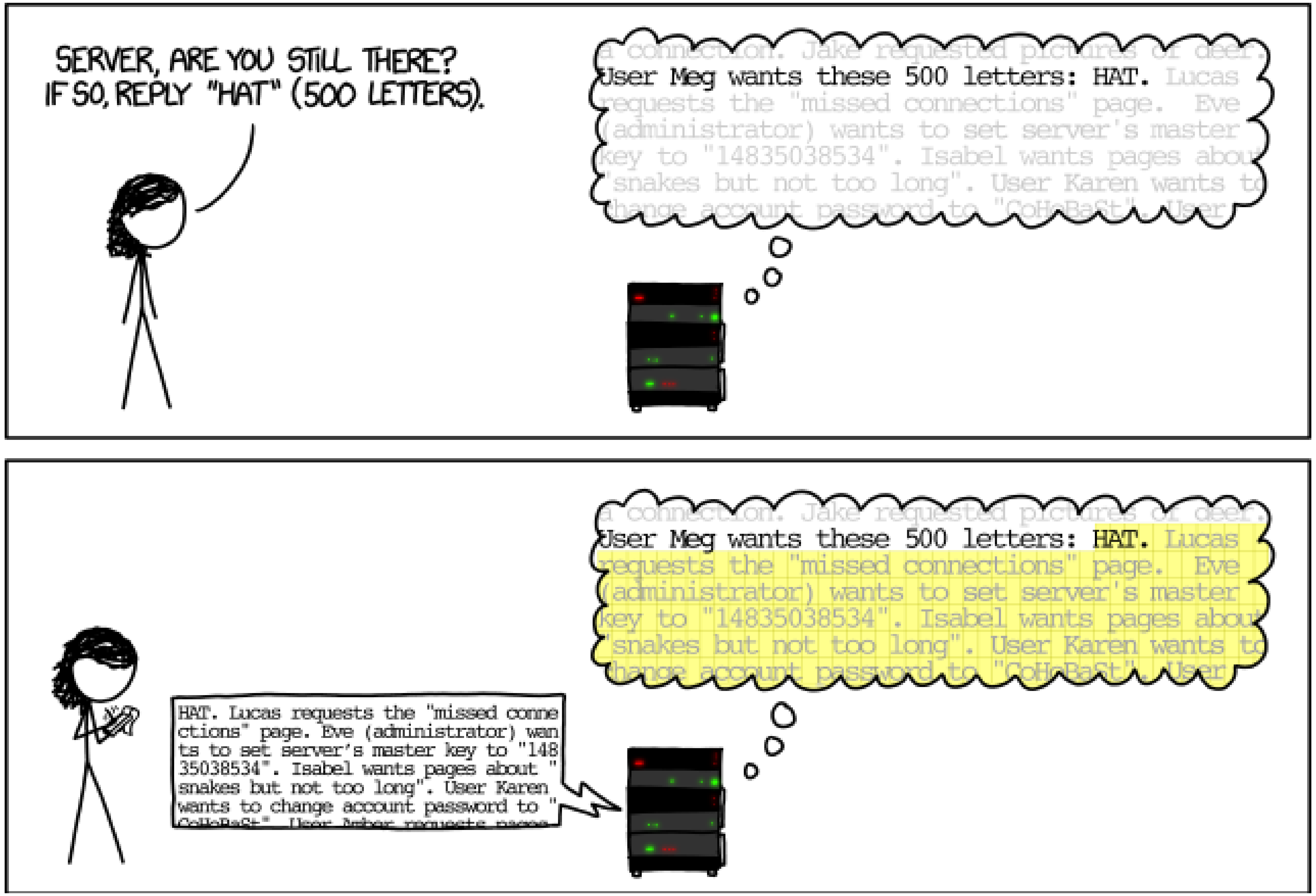
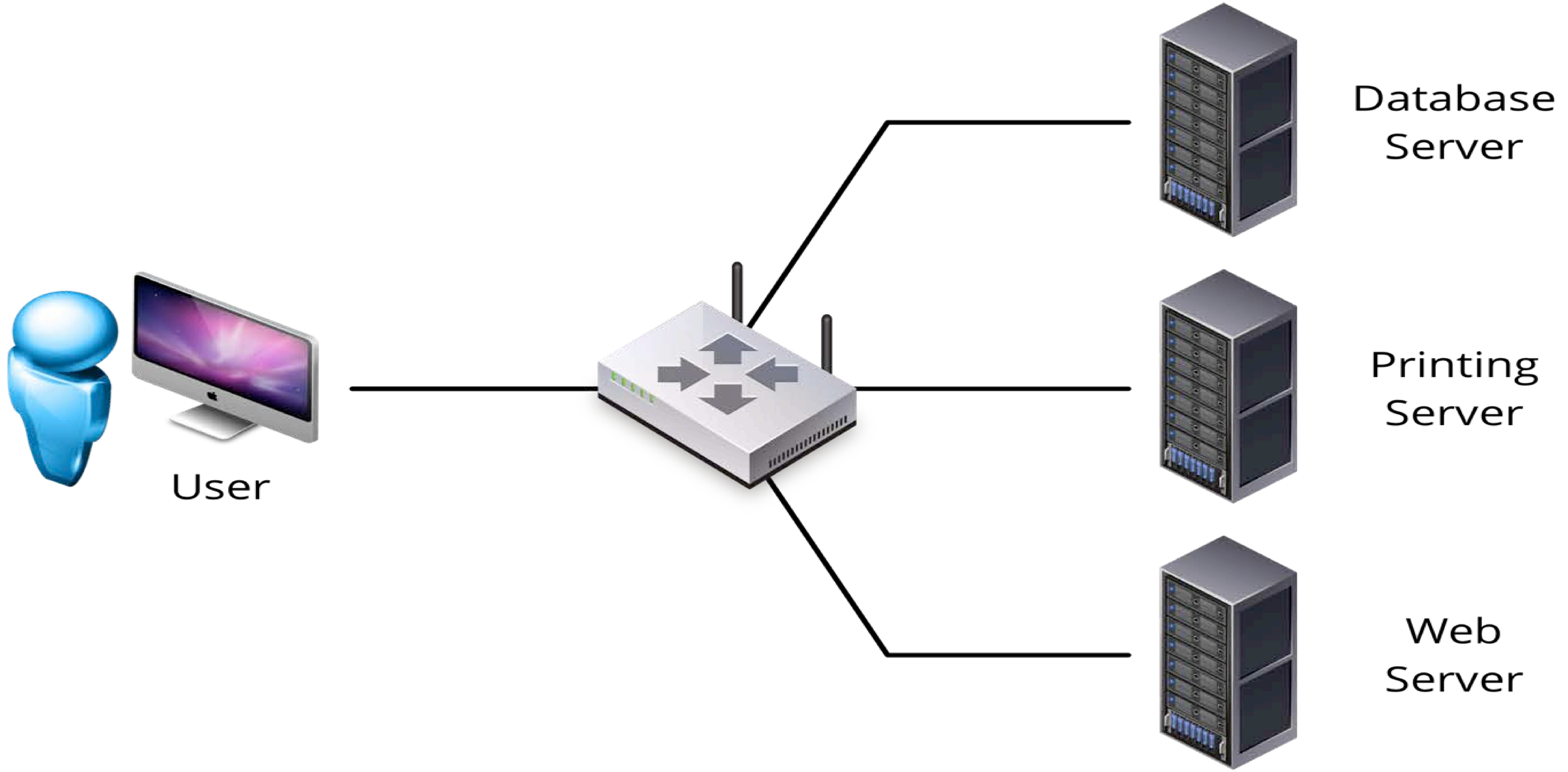


Image from ***xkcd.com***

Recommended Readings

- Stallings, W. Cryptography and Network Security. Chapter 17
- Hearthbleed – hearthbleed.com

Network Authentication



Kerberos

- Network authentication protocol
- Invented at M.I.T in late 1980’s
- Allow users to access services distributed throughout a network
- Built on the concepts of
  - Ticket
  - Centralized trusted server known as **Key Distribution Center** (KDC)
- Relies upon **symmetric encryption**

Kerberos Goals

- The user's password should **not travel** over the network
- The user's password is **never stored** on the client's machine
- The user is asked to enter the password only once per work session (**single-sign on**)
- The **authentication** information management is **centralized** and resides at the authentication server
- Clients and application servers **mutually authenticate**

Kerberos Concepts

- **Realm**
  - Set of users and application servers that are authenticated by the same Key Distribution Server
- **Principal**
  - User, Application Server, Service on the Network
- **Ticket**
  - Proof presented by a user to an application server to demonstrate his/her identity
  - Issued by the Authentication Server
  - Encrypted with secret key of the service it is intended for

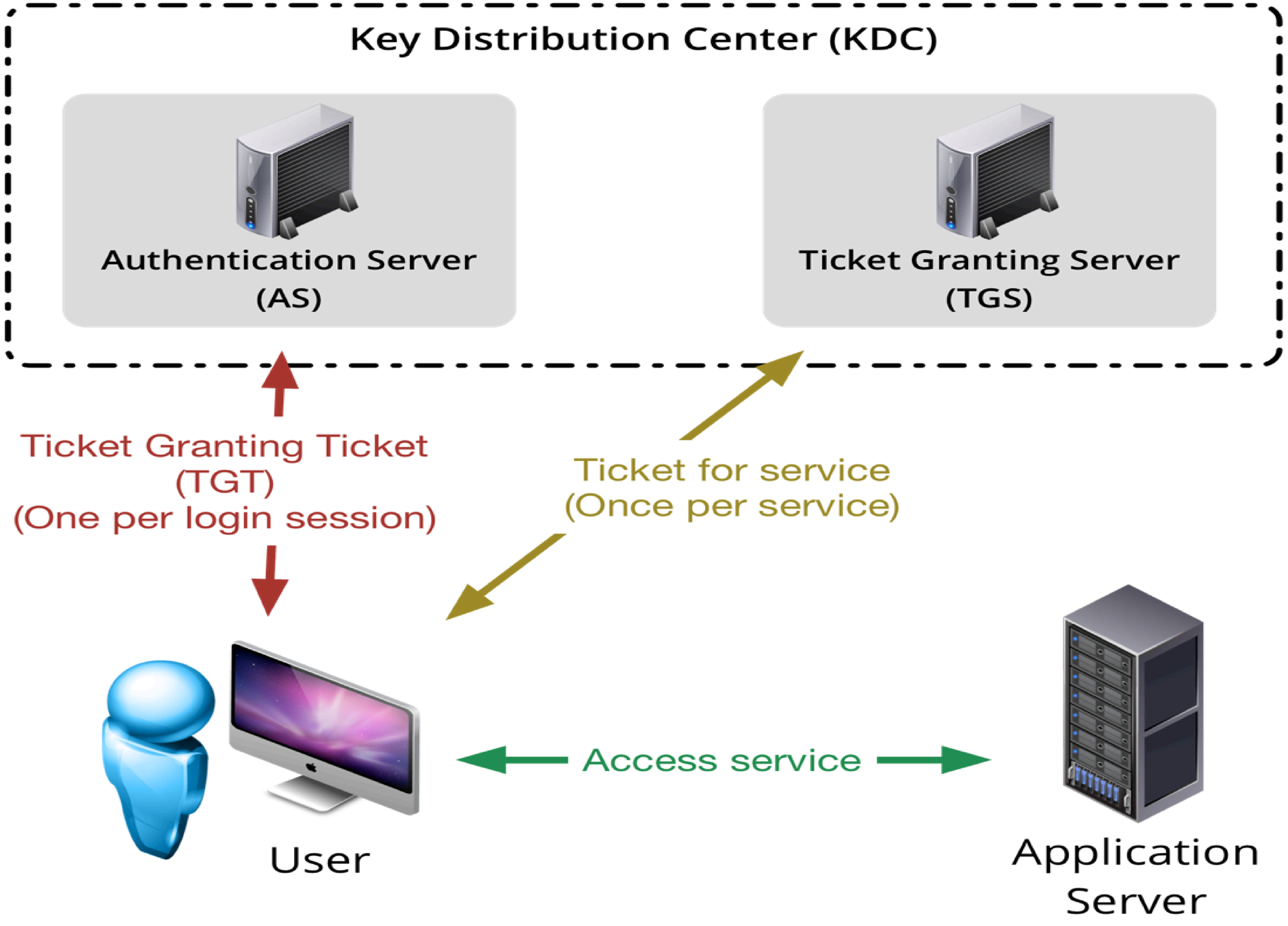
Kerberos Key Distribution Center

- Centralized Trust Model
  - Each client and server trusts the KDC
  - Each client and server shares a **master key** with the KDC
- Maintains a database of principals and their master keys
- The principals' master keys are stored encrypted with the master key of the KDC (K_{KDC})
- The master key for principals is derived from their passwords

Kerberos Key Distribution Center

- It consists of three components
  - **Database**
    - Maintains master key for each principal
  - **Authentication Server**
    - Issues a Ticket Granting Ticket (TGT)
  - **Ticket Granting Server**
    - Issues Ticket to access a service

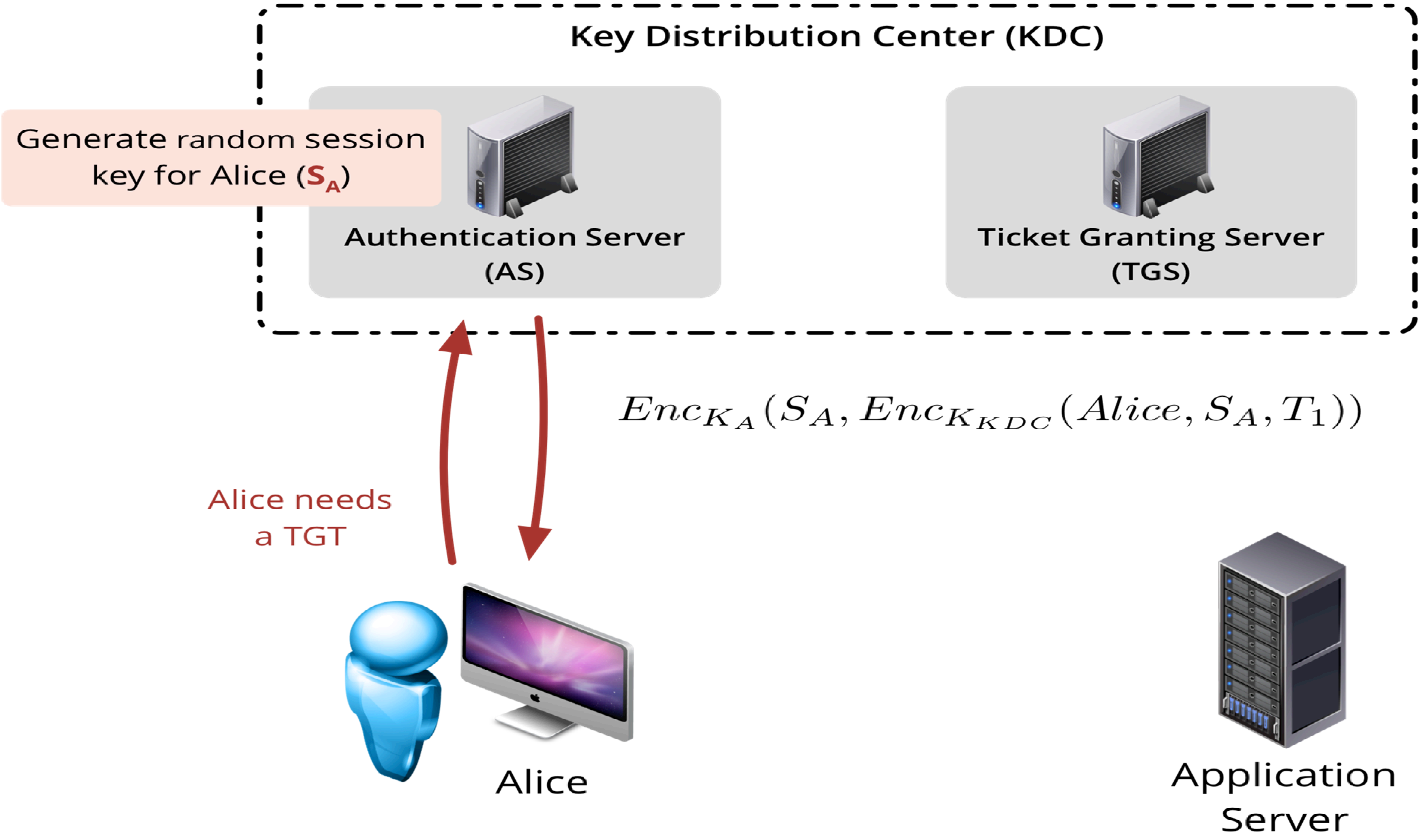
Kerberos Overview



Kerberos Overview

- Assume that Alice wants to access Service S
- **Phase 1**: Alice gets a Ticket Granting Ticket (TGT) from the Authentication Server
- **Phase 2**: Alice uses the TGT to obtain a ticket to access Service S from the Ticket Granting Server
- **Phase 3**: Alice accesses Service S

Phase 1: Obtaining a TGT



Phase 1: Obtaining a TGT

1. Client requests a Ticket Granting Ticket to the Authentication Server
2. The Authentication Server generates a session key S_A
3. The Authentication Server generates the Ticket Granting Ticket for Alice

$$Enc_{K_{KDC}}(Alice, S_A, T_1)$$

- Alice is the username
- SA is the session key for Alice and the TGS
- T1 is the timestamp
- K_{KDC} is the secret key shared between the Authentication Server and the Ticket Granting Server

Phase 1: Obtaining a TGT

4. The Authentication Server sends to the Client

$$Enc_{K_A}(S_A, Enc_{K_{KDC}}(Alice, S_A, T_1))$$

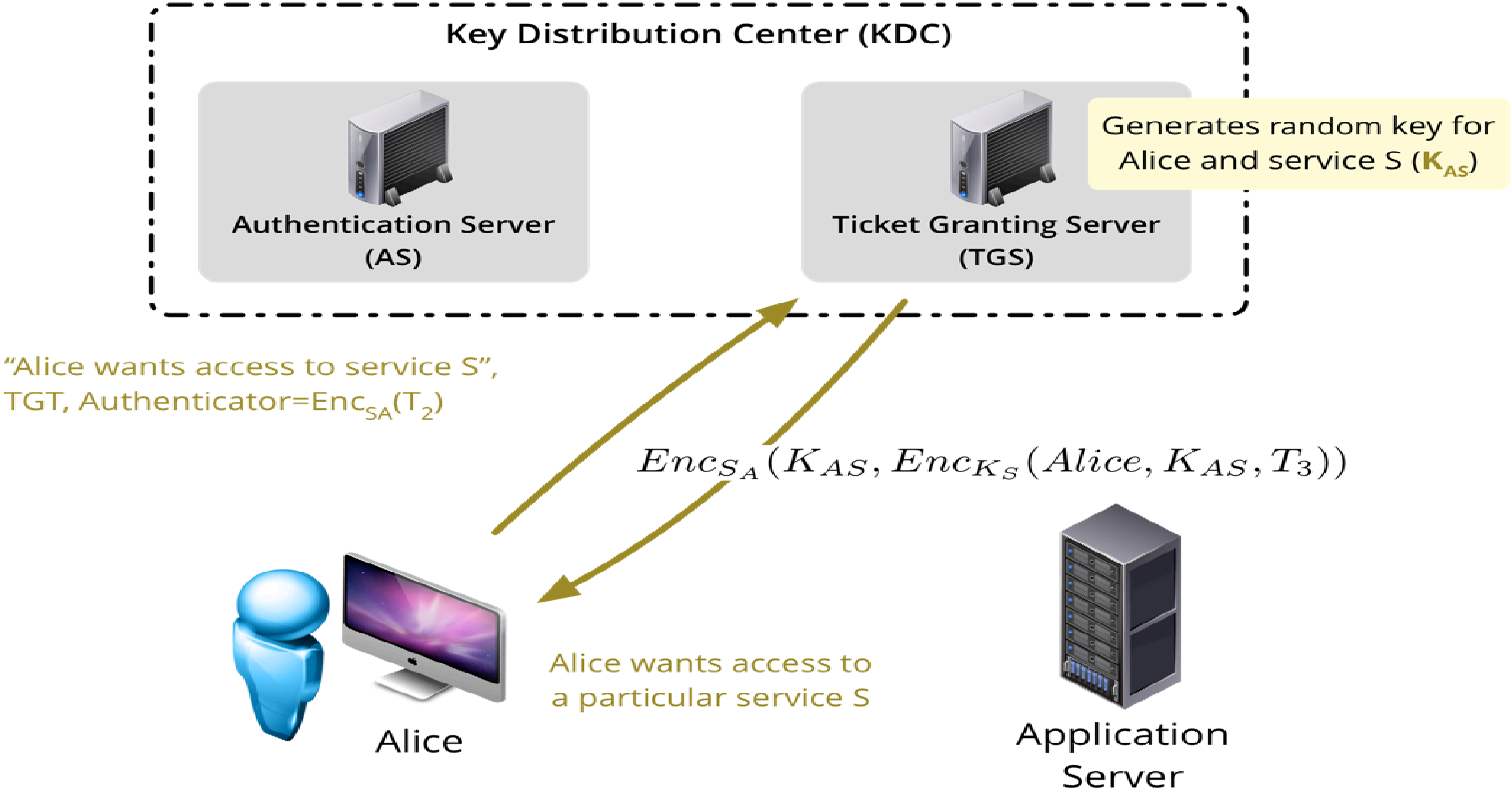
   - **K_A** is the master key for user Alice

5. The Client asks Alice to provide login and password
6. The Client generates K_A from Alice's password
7. The Client uses KA to decrypt

$$Enc_{K_A}(S_A, Enc_{K_{KDC}}(Alice, S_A, T_1))$$

8. The Client obtains the session key SA and the TGT

Phase 2: Obtaining a Service Ticket



Phase 2: Obtaining a Service Ticket

1. The Client sends a request to issues a service ticket for service S with the TGT and an authenticator $Enc_{S_A}(T_2)$
2. The Ticket Granting Server decrypts TGT with its master key K_{KDC} and obtains the session key $\mathbf{S_A}$ and timestamp $\mathbf{T_1}$
3. The Ticket Granting Server decrypts the authenticator and obtains timestamp $\mathbf{T_2}$
4. The Ticket Granting Server verifies that $T_2 - T_1 < 5$ min

Phase 2: Obtaining a Service Ticket

5. The Ticket Granting Server generates a session key $\mathbf{K_{AS}}$
6. The Ticket Granting Server issues the Ticket for service S

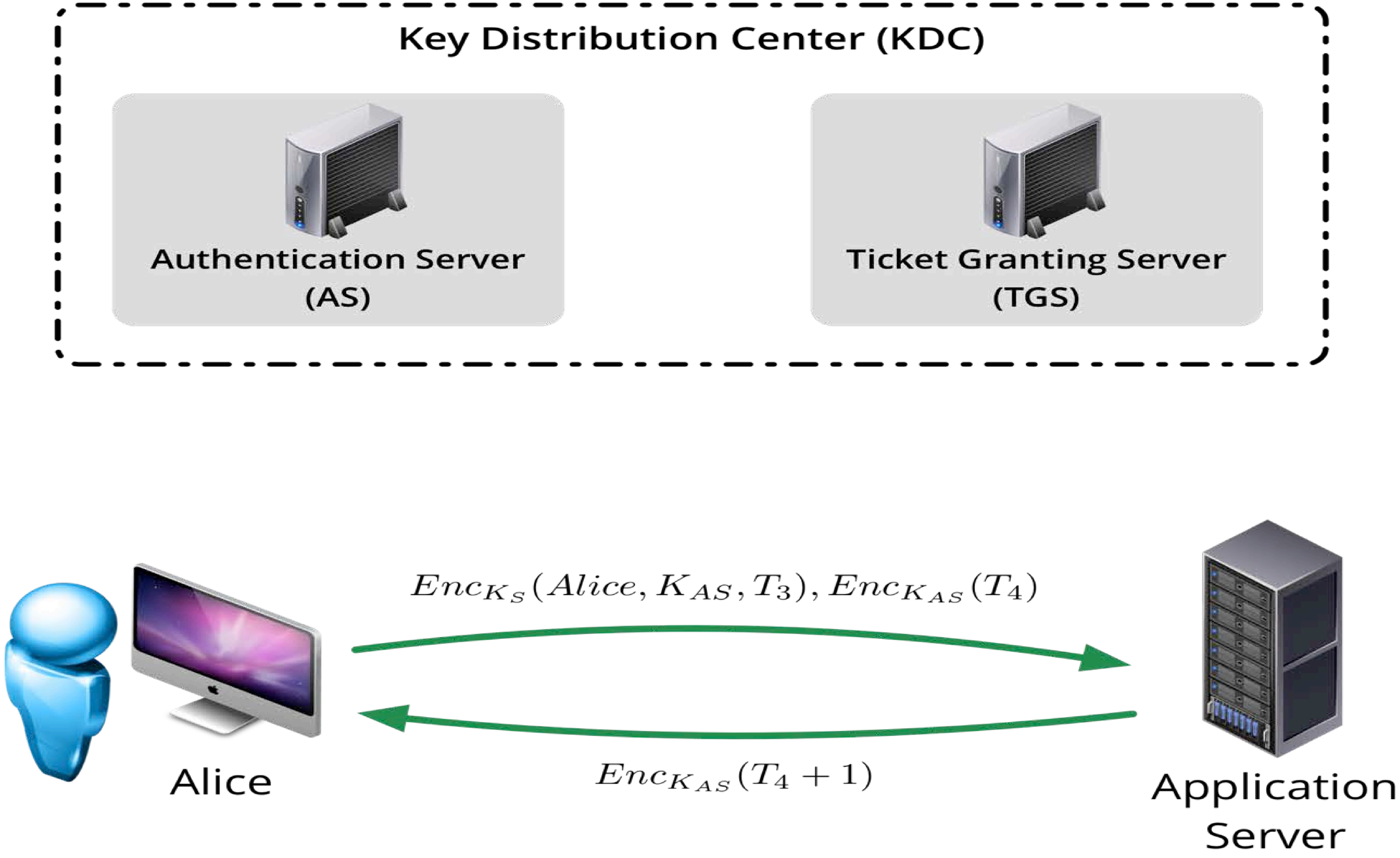
- Alice is the username
- T_3 is the timestamp
- K_{AS} is the session key for Alice and the service S
- K_S is the master key of the service S

$$Enc_{K_S}(Alice, K_{AS}, T_3)$$

7. The Ticket Granting Server sends to the client

$$Enc_{S_A}(K_{AS}, Enc_{K_S}(Alice, K_{AS}, T_3))$$

Phase 3: Using the Service Ticket



Phase 3: Using the Service Ticket

1. The Client sends a request to access the service S with
   - the ticket for service S $Enc_{K_S}(Alice, K_{AS}, T_3)$
   - an authenticator $Enc_{K_{AS}}(T_4)$
2. The Service S decrypts the service ticket for S with its master key $\mathbf{K_S}$ and obtains $\mathbf{K_{AS}}$ and $\mathbf{T_3}$
3. The Service S decrypts the authenticator and obtains $\mathbf{T_4}$
4. The Service S verifies that $T_4 - T_3 < 5$ min
5. The Service S sends $Enc_{K_{AS}}(T_4 + 1)$

Kerberos Limitations

- Single point of failure: requires continuous availability of KDC
  - When the KDC server is down, no one can log in
  - Can be mitigated by using multiple KDC servers
- Requires the clocks of the involved entities to be synchronized
  - Tickets have time availability period

    If the host clock is not synchronized with the clock of Kerberos server, the authentication will fail.
- Assumes the user’s workstation is secure
- It is vulnerable to password guessing attacks

Recommended Readings

- The Kerberos Network Authentication Server RFC 4120
  - http://www.ietf.org/rfc/rfc4120.txt
- Black Hat 2014 – Windows: Abusing Microsoft Kerberos Sorry You Guys Don’t Get It
  - https://www.youtube.com/watch?v=-IMrNGPZTl0
- Kerberos tutorial from M.I.T
  - http://www.kerberos.org/software/tutorial.html