Università degli Studi di Milano
Department of Computer Science

SSL/TLS: cryptographic protocols and their weaknesses

Andrea Visconti

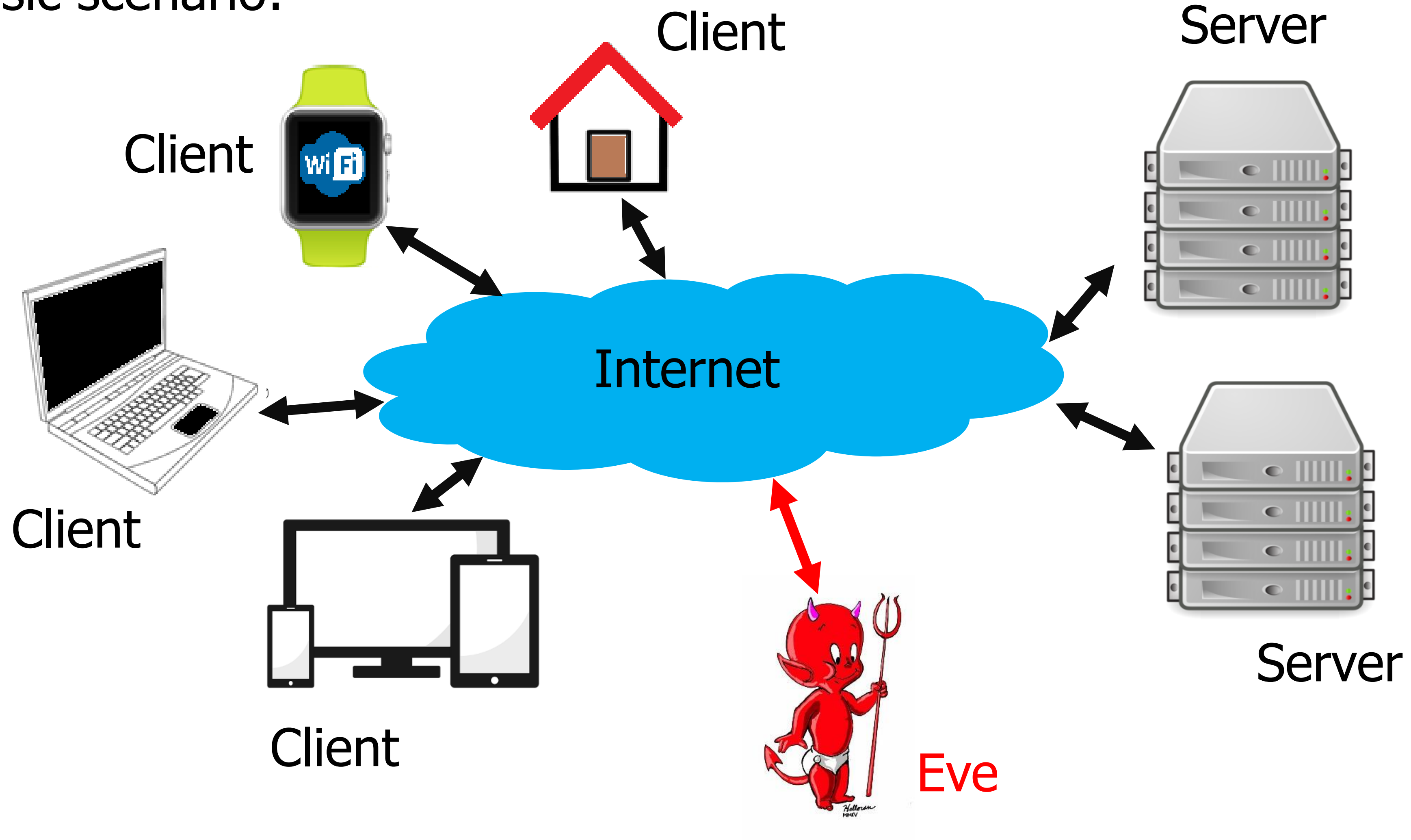
January 27th, 2020

Overview

1. Introduction to Secure Socket Layer and Transport Layer Security – SSL 2.0/3.0 and TLS 1.0/1.1/1.2
2. Security provided by SSL/TLS protocols
3. Vulnerabilities published in the literature (RFC 7457):
   - Null Prefix Attack
   - Renegotiation Attack
   - Browser Exploit Against SSL/TLS (BEAST)
   - Compression Ratio Info-leak Made Easy (CRIME)
   - Factoring RSA Export Keys (FREAK)
   - …
4. Introduction to TLS 1.3

Introduction

Basic scenario:



Introduction

SSL and TLS were meant to provide **a secure channel over untrusted networks**;

SSL and TLS use **X.509 certificates**;

SSL and TLS use **asymmetric** cryptography to:

- authenticate the actors;
- exchange a symmetric key;

A **symmetric** key is used to encrypt data flowing between client and server;

Introduction

1994: **Secure Sockets Layer** (SSL) protocol, created by Netscape;

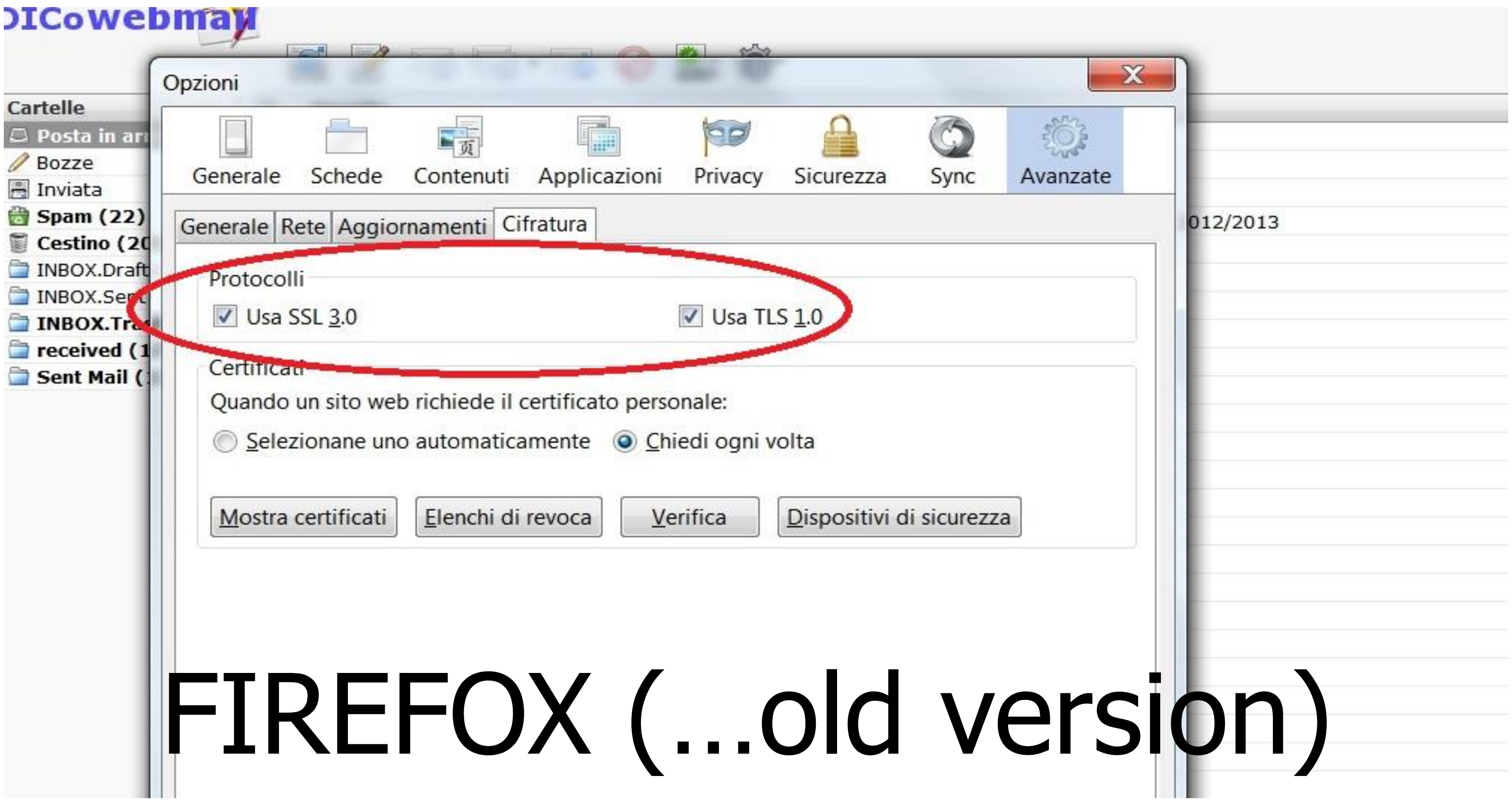
1996: **Transport Layer Security** (TLS), developed by the Internet Engineering Task Force (IETF);

SSL: SSL1 (never released), SSL2 (Feb 95), **SSL3 (Mar 96)**;
TLS: **TLS1.0 (Jan 99)**, TLS1.1 (Apr 06), TLS1.2 (Aug 08), TLS1.3 (Aug 2018);

How do you choose which one to use?

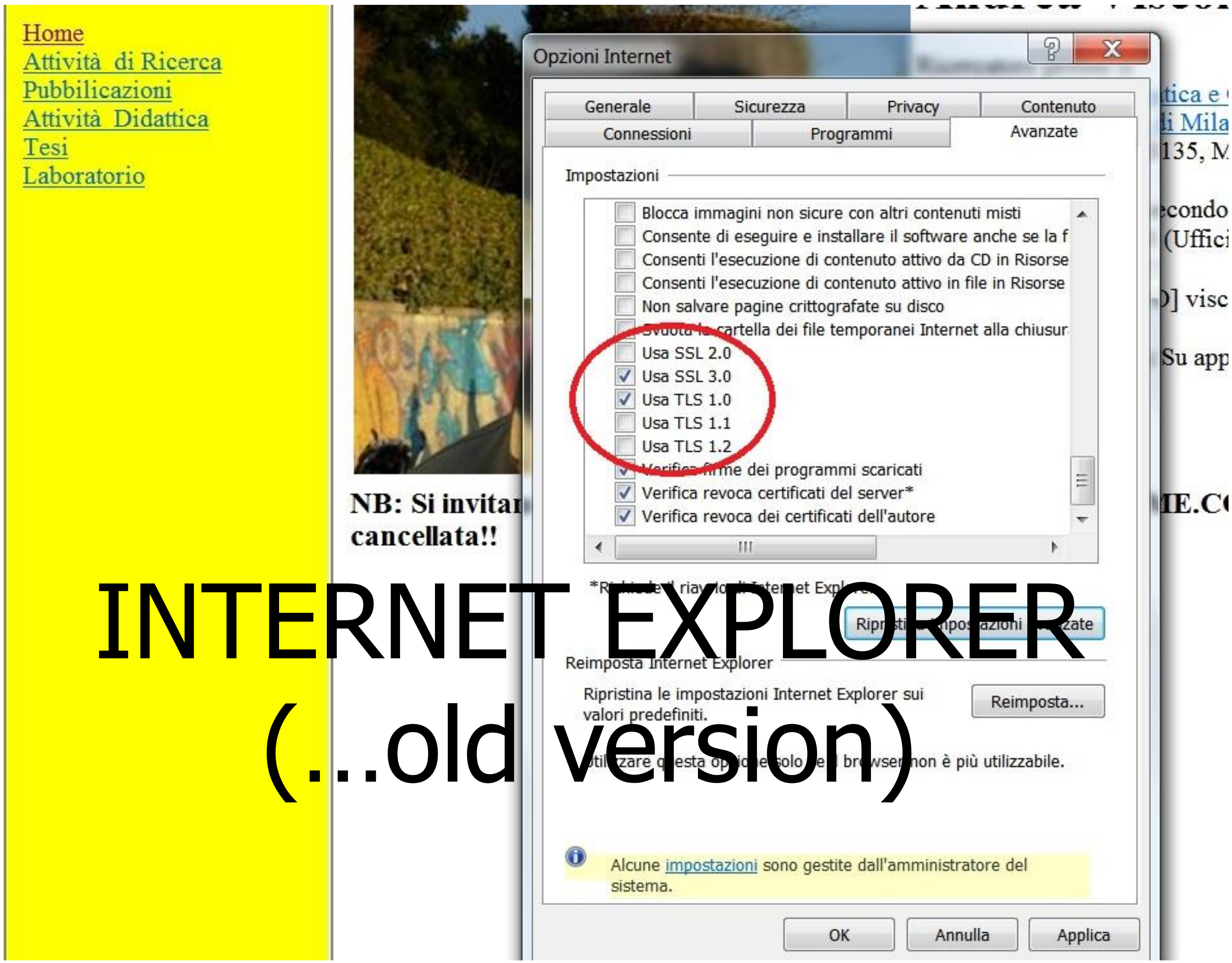
Introduction

Let your browser choose for you ...



FIREFOX (...old version)

Introduction



INTERNET EXPLORER

(...old version)

Introduction

- Firefox: TLS 1.0, ~~TLS 1.1~~, ~~TLS 1.2~~
- IE: TLS 1.0, TLS 1.1, TLS 1.2
- Chrome: TLS 1.0, ~~TLS 1.1~~, ~~TLS 1.2~~
- Opera: TLS 1.0, TLS 1.1, TLS 1.2
- Safari: TLS 1.0, ~~TLS 1.1~~, ~~TLS 1.2~~

Only Opera enables TLS 1.1 and TLS 1.2 by default.

(...old version)

Introduction

Firefox has not a user interface setting to disable or enable TLS/SSL protocol;

You can enable/disable protocols on the **about:config** page;

You can set the security.tls.version.**min** and security.tls.version.**max** preferences:

- 0 means SSL 3.0;
- 1 means TLS 1.0;
- 2 means TLS 1.1;
- 3 means TLS 1.2;
- 4 means TLS 1.3;

Introduction

The differences between SSL and TLS (TLS 1.3 excluded)… **Performaces**? **Security**?

TLS 1.0: the small differences between TLS 1.0 and SSL 3.0 preclude the interoperability between protocols;

TLS 1.1: This version include protection against CBC attacks;

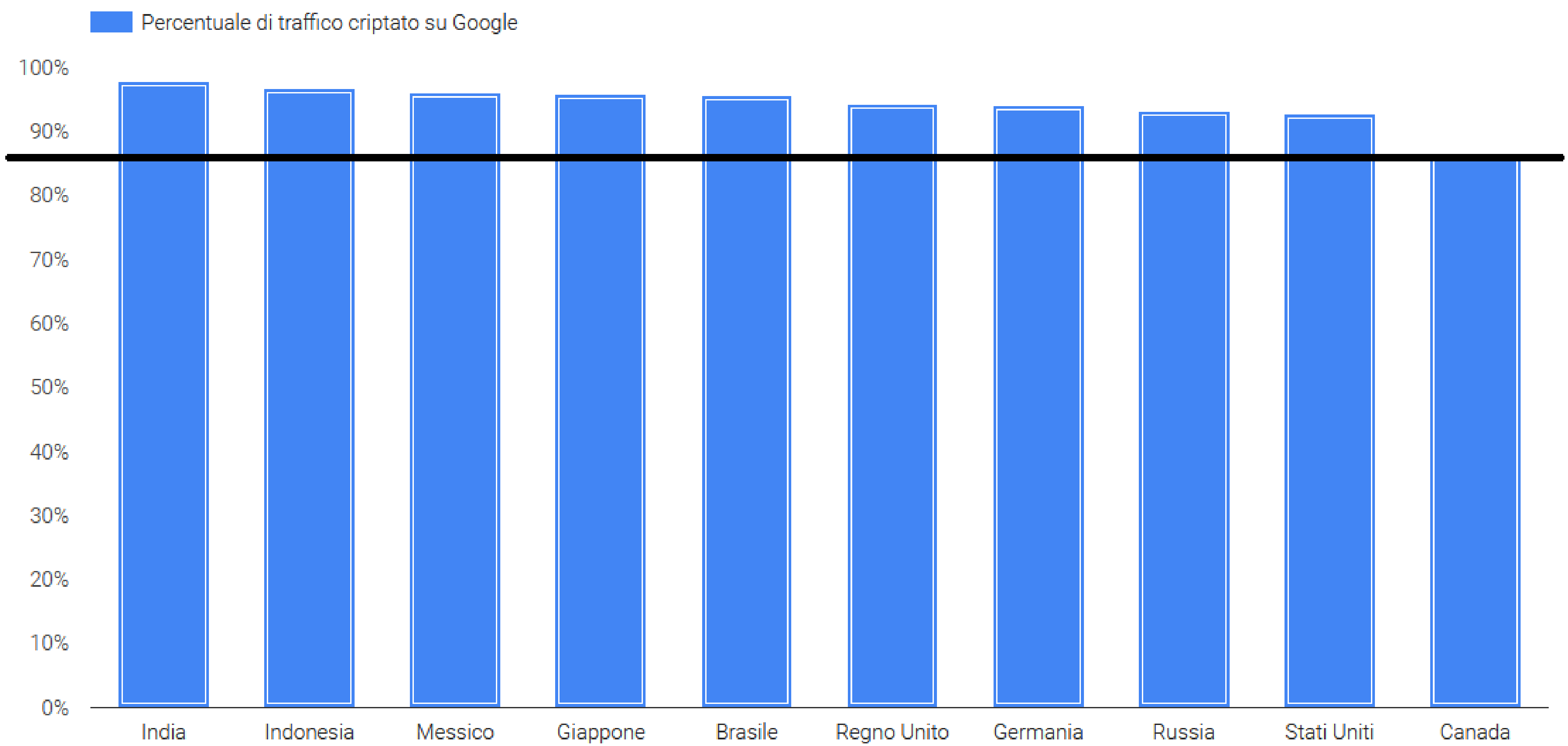
TLS 1.2: The combination of hash functions (MD5-SHA-1) has been replaced with SHA-256;

In 2015, IETF **deprecated** SSL 3.0 (RFC 7568);

In 2020, all major web browsers will **drop support** for TLS 1.0 and TLS 1.1.

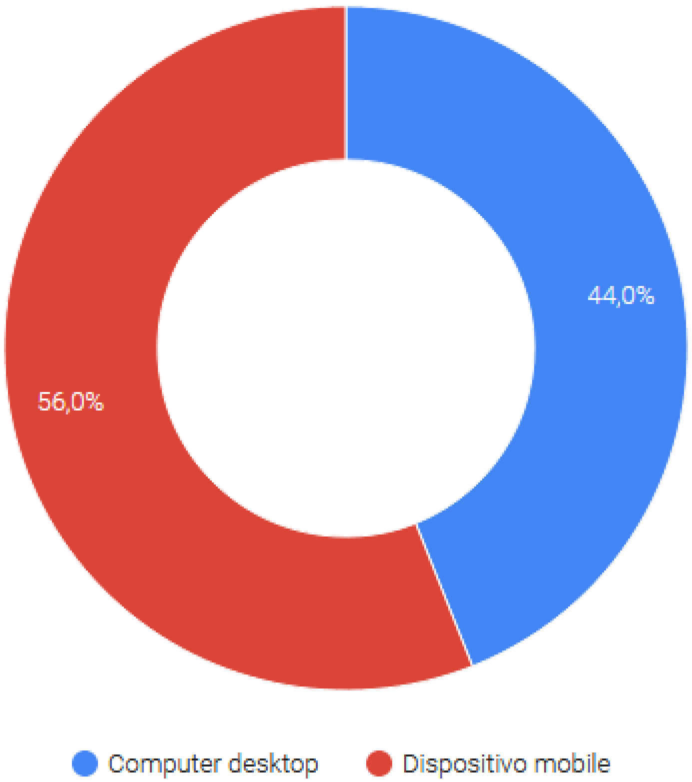
Introduction SSL/TLS

Statistics: HTTPS encryption on the web (GOOGLE, 2020)



Introduction SSL/TLS

Statistics: Unencrypted user traffic (GOOGLE, 2020)

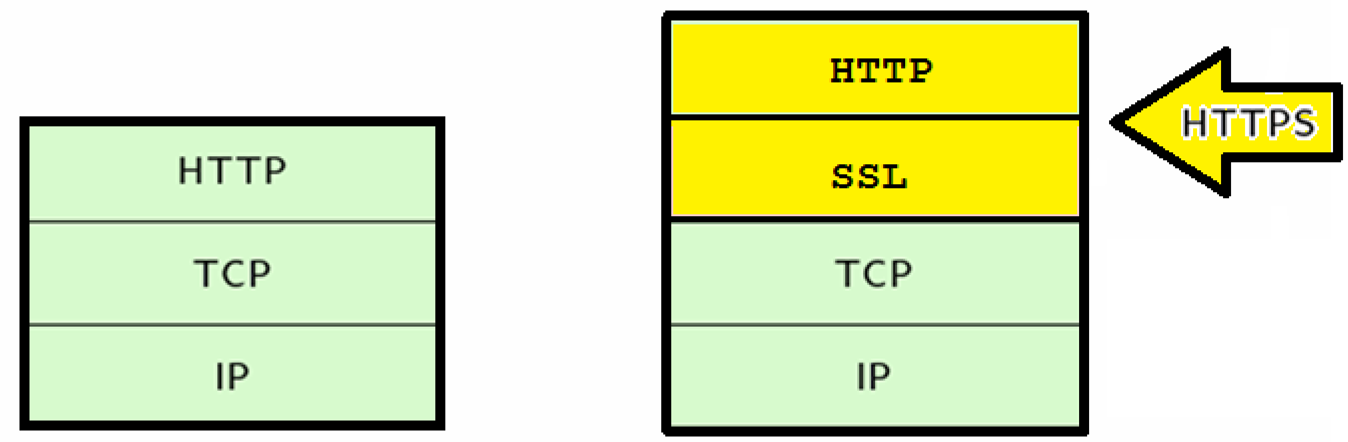


Introduction SSL/TLS

Statistics: HTTPS on top sites (GOOGLE, 2020)

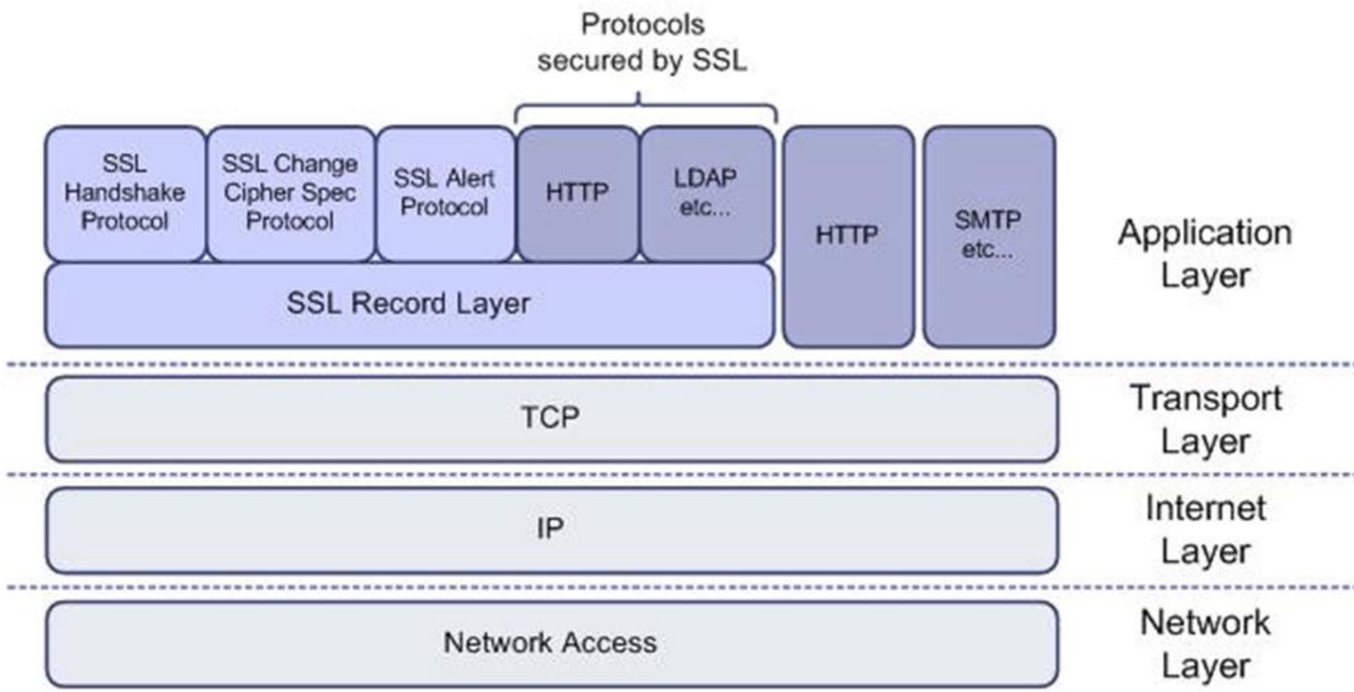
- Interestingly, the top 100 non-Google sites account for ≈25% of all website traffic;
- 96 out of 100 sites adopt HTTPS by default.

Introduction SSL/TLS



TCP/IP Protocol Stack

Introduction SSL/TLS



TCP/IP Protocol Stack

Introduction SSL/TLS

A description of SSL/TLS protocols (**TLS 1.3 excluded**) …

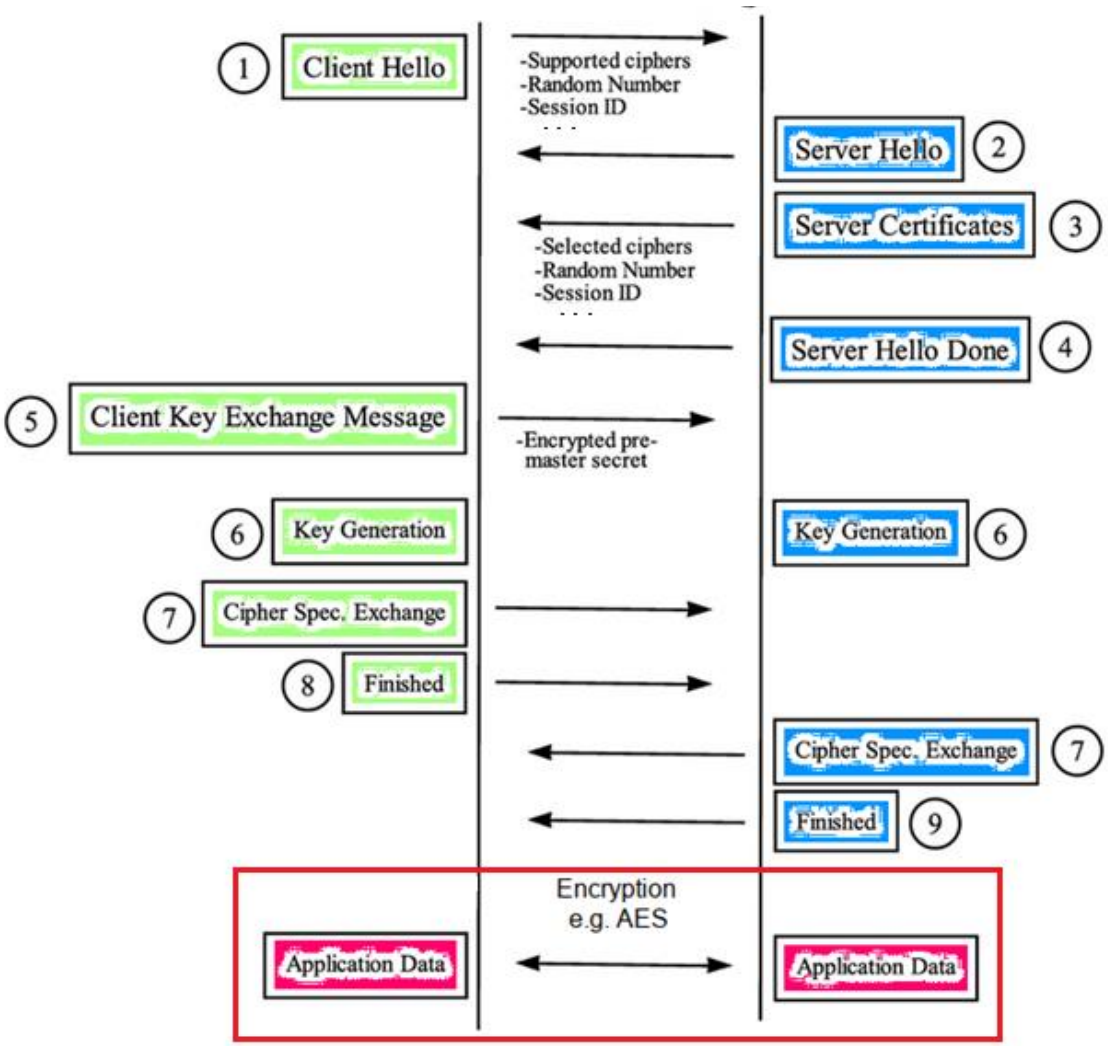
Session state: Session ID, Peer certificate, Compression method, Cipher spec, Master secret, Is resumable.

Connection state: Random sequences (client and server), Write MAC secret (client and server), IV (e.g. CBC mode), sequence numbers, …

Introduction SSL/TLS

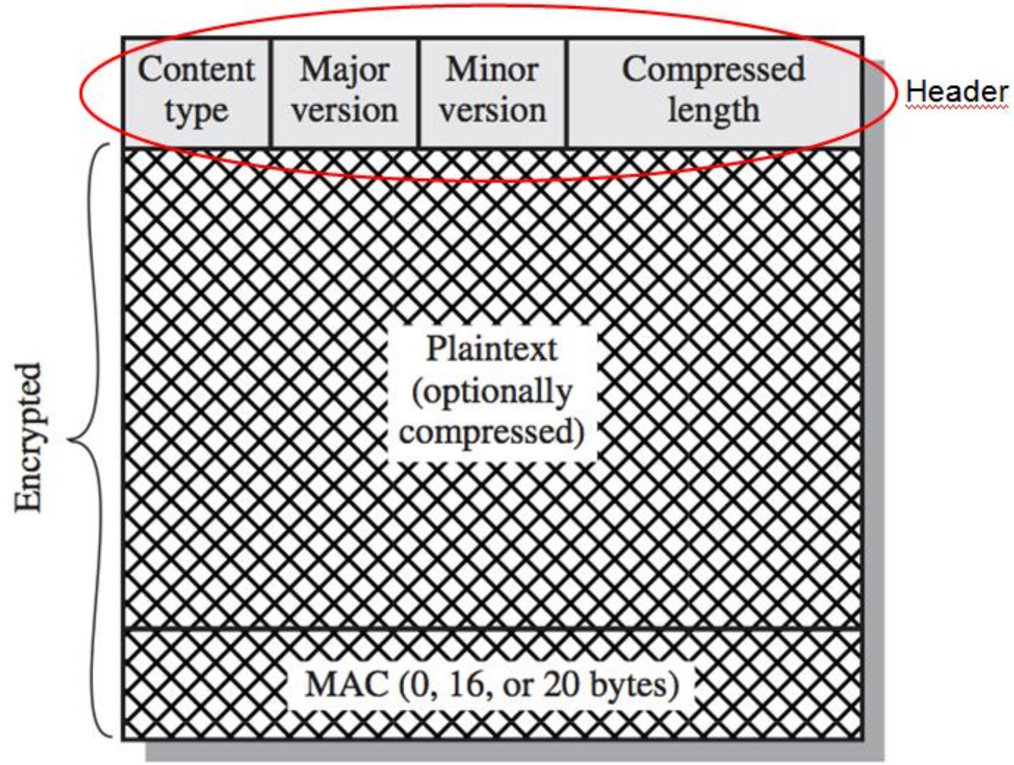
Handshake Protocol:

(TLS 1.3 excluded)



Introduction SSL/TLS

SSL Record Protocol:



Algo supported: AES, 3DES, DES, DES-40, RC4-128, RC2-40, …

Introduction SSL/TLS

Change Chiper Spec Protocol:

1 Byte (YES/NO)

Introduction SSL/TLS

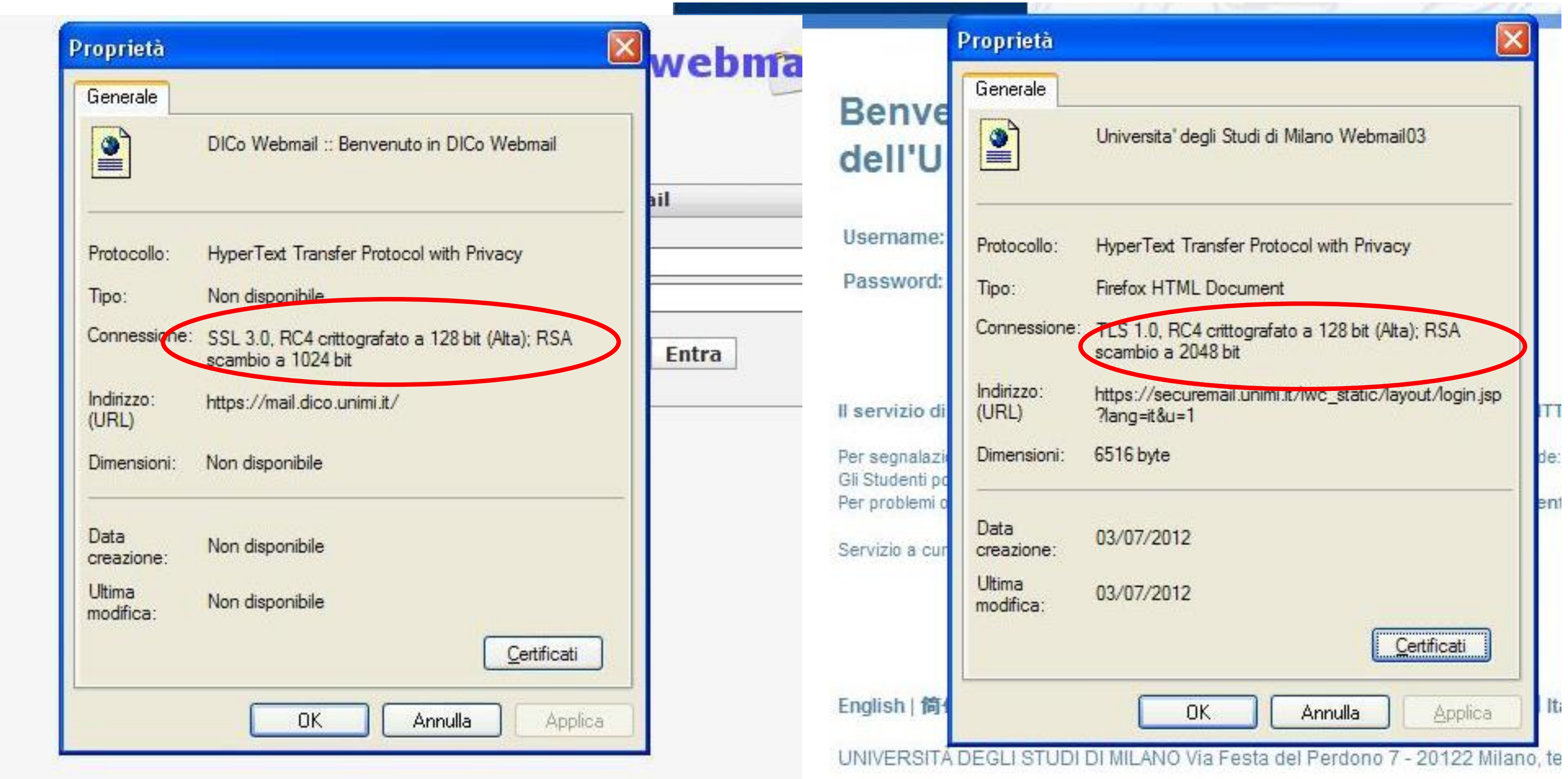
Alert Protocol:

WARNING: Close notify, No certificate, Unsupported certificate, Certificated revoked, Certificated expired, …

FATAL: Bad record MAC, Handshake failure, Decompression failure, Illegal parameter, …

Example SSL/TLS

The cipher suite includes algorithms for encrypting data, computing the MAC, and exchanging keys.



Null Prefix Attack (2009)

Null Prefix Attack

The problem is related to how browsers handle certificate fields with **null value character** (**\0**).

- **String format**: PASCAL VS. C;
- **Common name**: Main field checked for authentication;
- **Authentication**: Domain validation certificates rely on email checking;

Null Prefix Attack

Attackers generate and submit a fake certificate request to Certification Authorities;

www.my_email.com\0I_am_cheating_you.com

During validation, Certification Authorities do not check request content fully, ignoring the subdomains placed before the null value character;

www.~~my_email.com~~\0I_am_cheating_you.com

Null Prefix Attack

Unfortunately, most SSL/TLS implementations interpret the X.509 certificates as C-strings

Thus browsers consider the "\0" character as a terminating point:

www.my_email.com**\0**~~I_am_ch[illegible].com~~

hence

www.my_email.com

TLS Renegotiation Attack (2009)

RFC 5746: TLS Renegotiation Indication Extension

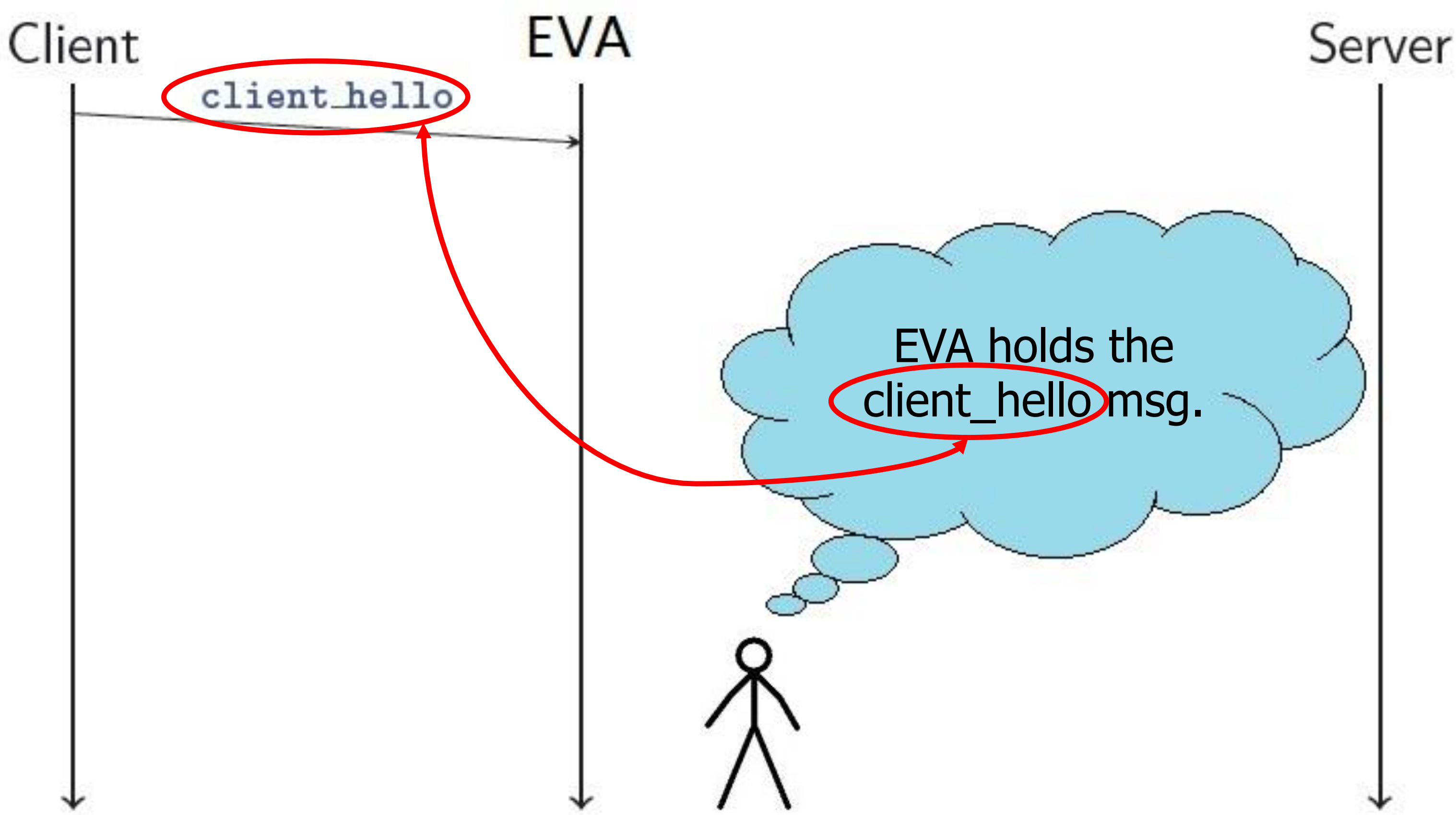
Feb 2010 – RFC 5746 – **Abstract**: Secure Socket Layer (**SSL**) and Transport Layer Security (**TLS**) **renegotiation are vulnerable to an attack** in which the attacker forms a TLS connection with the target server, injects content of his choice, and then splices in a new TLS connection from a client. The server treats the client's initial TLS handshake as a renegotiation and thus believes that the initial data transmitted by the attacker is from the same entity as the subsequent client data. **This specification defines a TLS extension** to cryptographically tie renegotiations to the TLS connections they are being performed over, thus **preventing this attack**.

RFC 5746: TLS Renegotiation Indication Extension

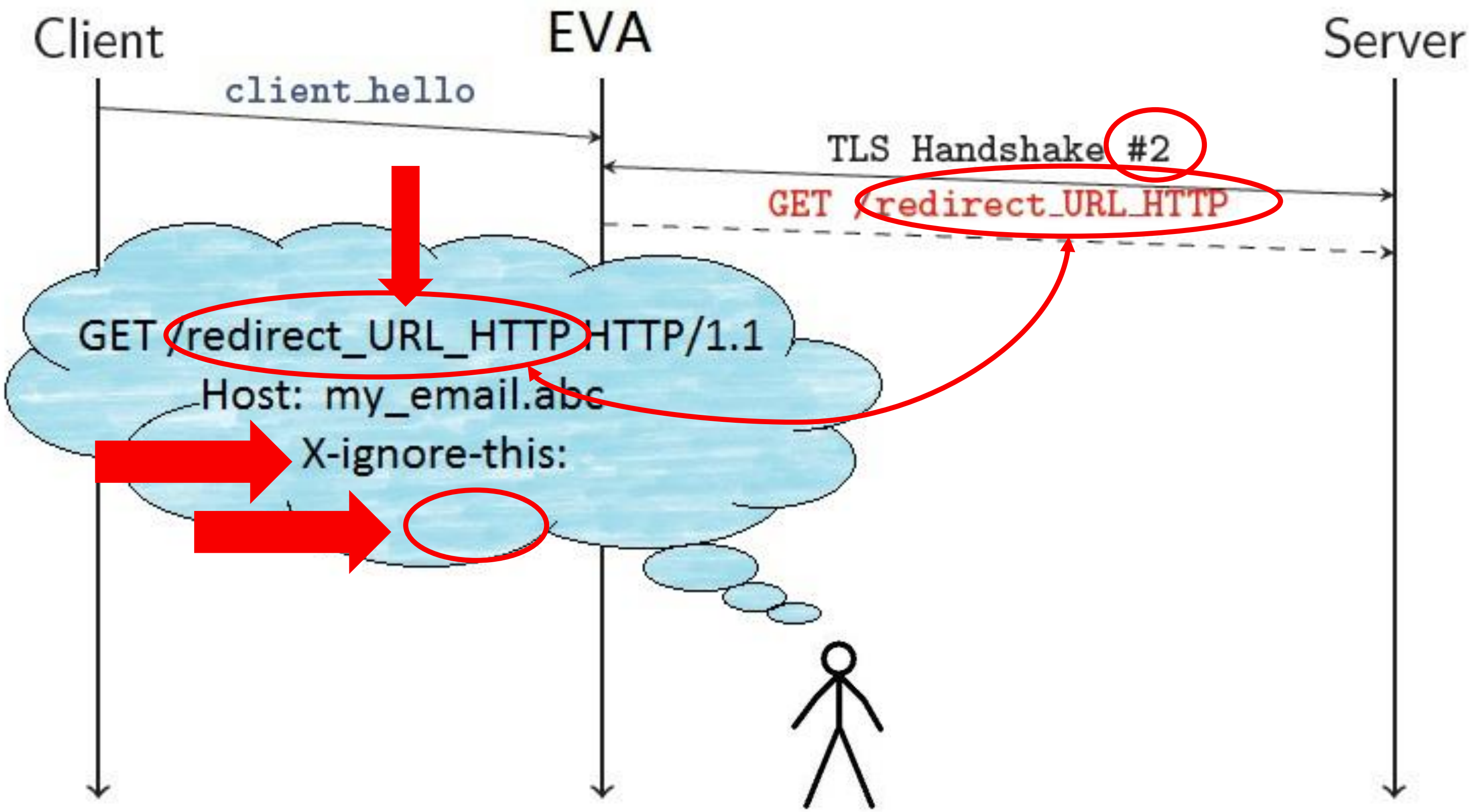
Feb 2010 – RFC 5746 – **Introduction**: … In some protocols **(notably HTTPS), no distinction is made between pre- and post-authentication stages** and the bytes are handled uniformly, **resulting in the server believing that the initial traffic corresponds to the authenticated client identity.** Even without certificate-based authentication, **a variety of attacks may be possible** in which the attacker convinces the server to accept data from it as data from the client.

For instance, **if HTTPS is in use with HTTP cookies, the attacker may be able to generate a request of his choice validated by the client's cookie.**

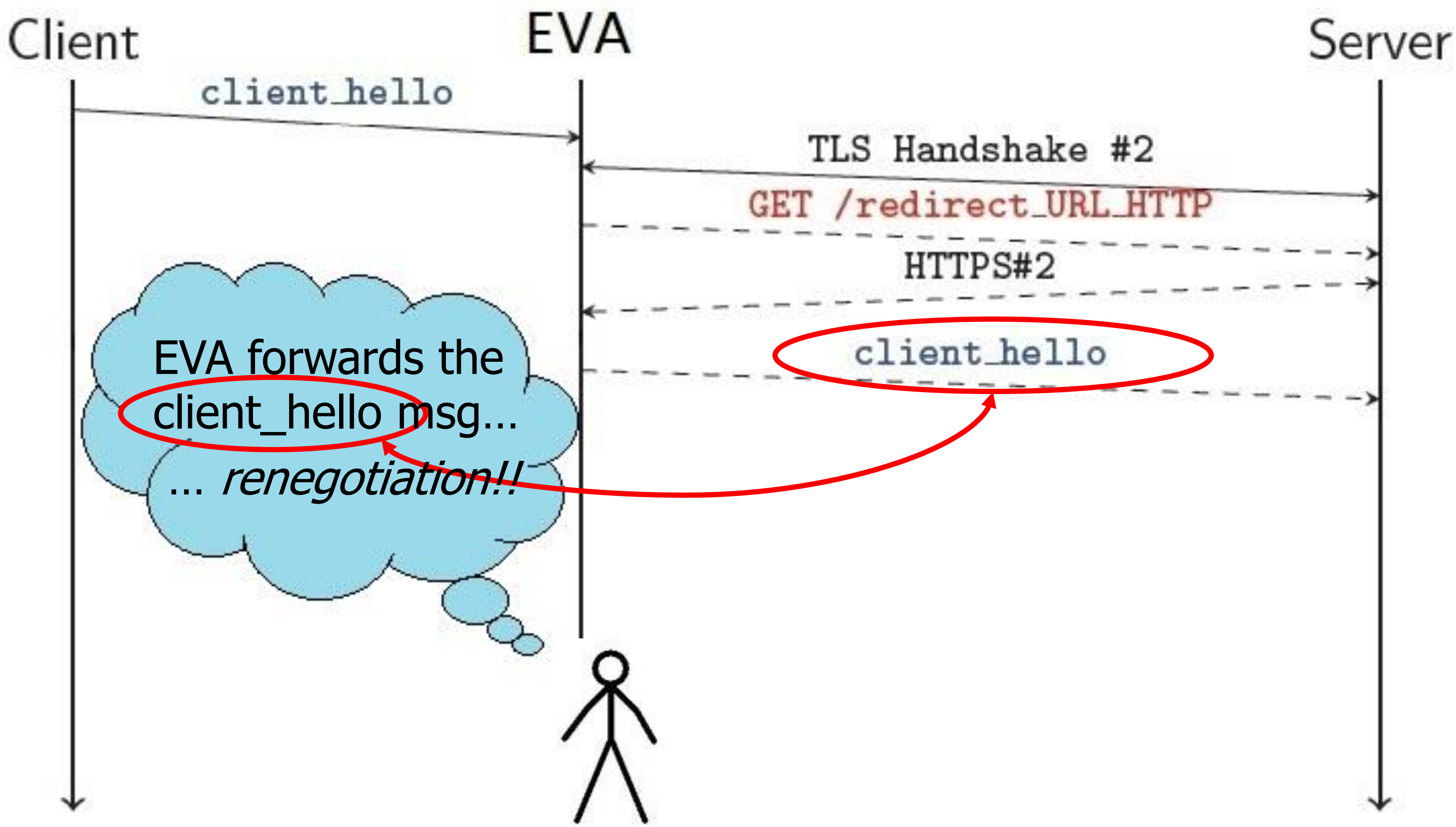
TLS Renegotiation Attack



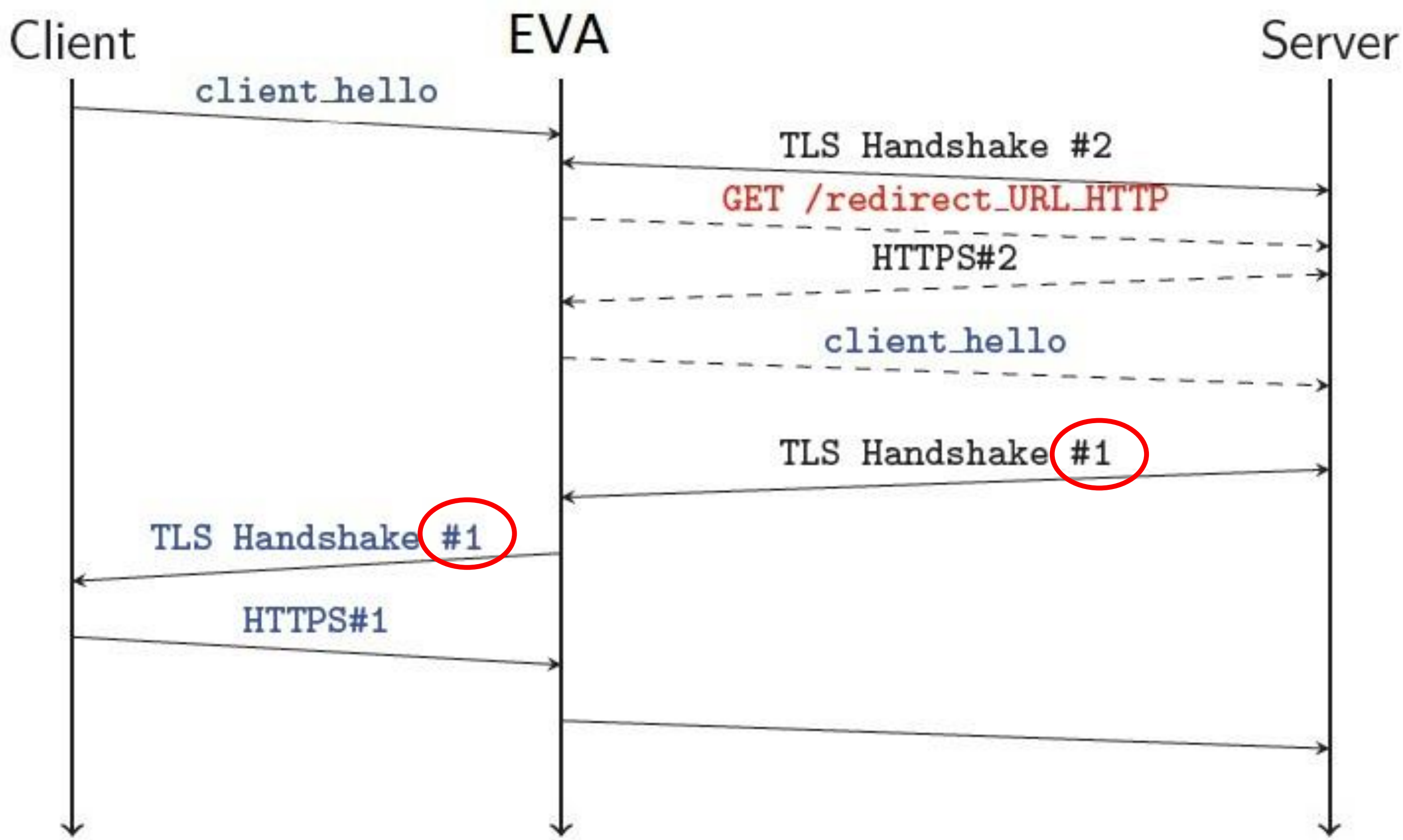
TLS Renegotiation Attack



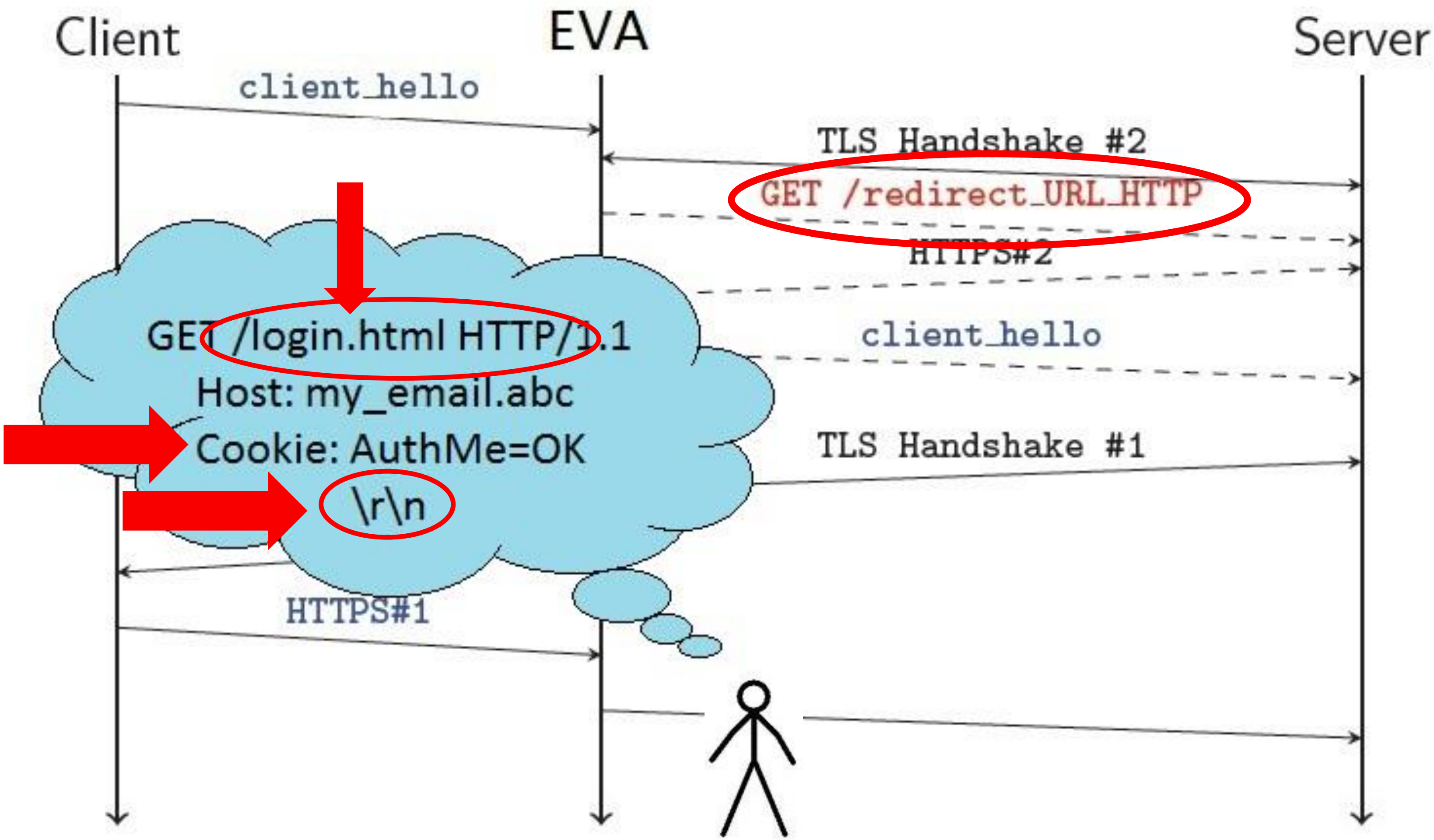
TLS Renegotiation Attack



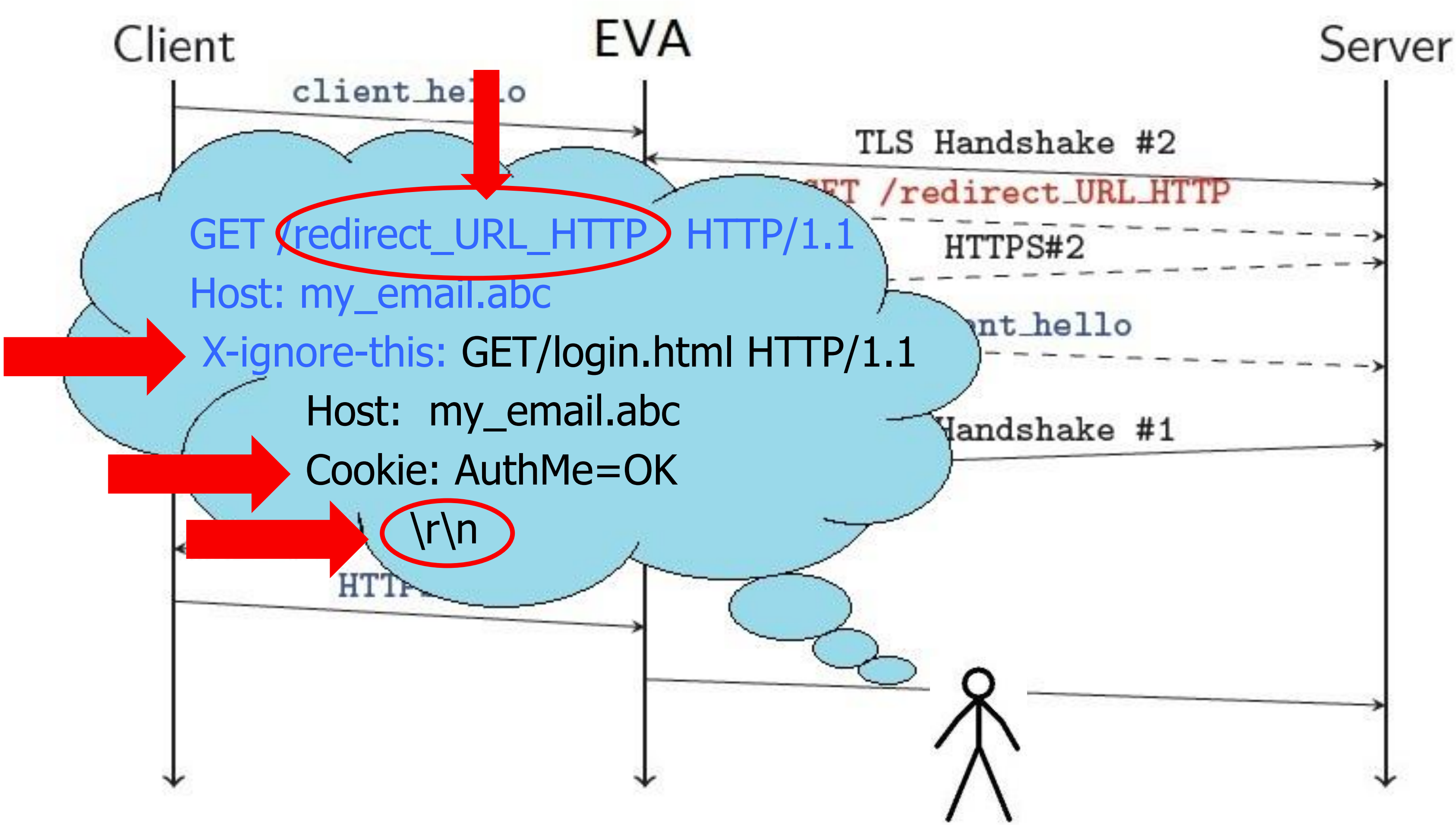
TLS Renegotiation Attack



TLS Renegotiation Attack



TLS Renegotiation Attack



BEAST: Browser Exploit Against SSL/TLS (2011)

Browser Exploit Against SSL/TLS

A security flaw: Rogaway (2002), Bard (2004), etc.;

Possible solutions: fix the bug, upgrading to TLS 1.1 or TLS 1.2, the ostrich solution, etc.

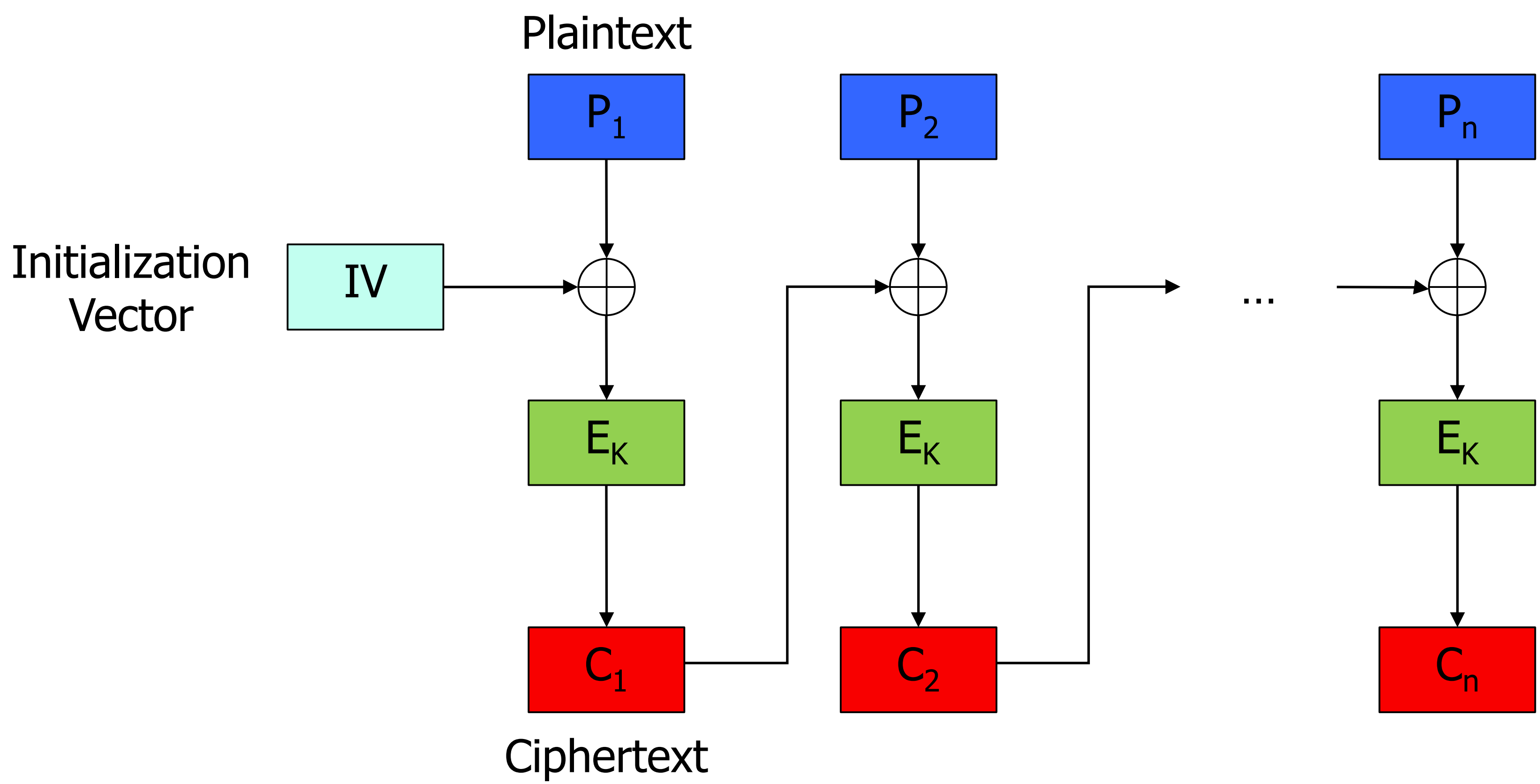
BEAST (2011): T. Duong and J. Rizzo exploit the vulnerability originally discovered in 2002;

TLS 1.1 has been widely adopted after the attack demonstration;

Unfortunately, the ostrich solution never works when it comes to security flaws.

Browser Exploit Against SSL/TLS

Cipher Block Chaining (CBC) mode encryption:



Browser Exploit Against SSL/TLS

- An attacker (Eve) can intercept network traffic;
- She will know the ciphertext;
- CBC mode encryption with chained initialization vectors;
- Initialization Vector (IV) is predictable;

An example:

Plaintext P=VISCONTIANDREA

P_1 = VISCONTI P_2 = ANDREA%%

Browser Exploit Against SSL/TLS

1. Block size B bytes (e.g. 8 bytes);

   $P_1 = V||I||S||C||O||N||T||I = 8$ bytes;

2. Eve chooses a random string R (B – 1) bytes);

   $R = A||A||A||A||A||A||A = 7$ bytes;

3. Eve prepends the string 'AAAAAAA' to P:

   $P_1^* = A||A||A||A||A||A||A||V = 8$ bytes;

4. Eve tries to guess P_1^*:

   $P_1' =$ Well-known String R|| Random character

   $= A||A||...||A||? = 8$ bytes;

Browser Exploit Against SSL/TLS

5. Eve tries to guess the last char:
   - Hp: $P_1' =$ AAAAAAAB
   - Hp: $P_1' =$ AAAAAAAC
   - Hp: $P_1' =$ AAAAAAAD
   - …
   - Hp: $P_1' =$ AAAAAAAV

5. If $P_1' = P_1^*$ then $C_1' = C_1^*$
   else $C_1' \neq C_1^*$

Browser Exploit Against SSL/TLS

7. Eve chooses a random string R (B – 2 bytes);

   $$R = A||A||A||A||A||A = 6 \text{ bytes};$$

8. She prepends the string 'AAAAAA' to P_1:

   $$P_1^* = A||A||A||A||A||A||V||I = 8 \text{ bytes};$$

9. She tries to guess P_1^*:

   $$P_1' = \text{Well-known String } R \;||\; V \;||\; \text{Random character} =$$

   $$= A||A||\ldots||A||V||? = 8 \text{ bytes};$$

Browser Exploit Against SSL/TLS

10. Again, Eve tries to guess the last char:
    - Hp: $P_1' =$ AAAAAAVB
    - Hp: $P_1' =$ AAAAAAVC
    - Hp: $P_1' =$ AAAAAAVD
    - …
    - Hp: $P_1' =$ AAAAAAVV
11. If $P_1' = P_1^*$ then $C_1' = C_1^*$

    else $C_1' \neq C_1^*$

Browser Exploit Against SSL/TLS

- Deterministic algorithm;
- An attacker tries to guess the encoding of a byte instead of a block;
- 256 iterations (worst case);
- 128 iterations (average case);

CRIME:
Compression Ratio Info-leak Made Easy (2012)

Compression Ratio Info-leak Made Easy

A security flaw: J. Kelsey (2002);

Compression Ratio Info-leak Made Easy: T. Duong, J. Rizzo (2012);

The attacker observes the **change in size of the compressed** request payload.

When the size of the compressed content is reduced, it can be inferred that it is probable that some part of the injected content matches some part of the secret content that the attacker desires to discover.

A possible solution: **CRIME can be defeated by preventing the use of compression**.

POODLE:
Padding Oracle On Downgraded Legacy Encryption
(2014)

POODLE attack

Published by Google researchers;

It is a man-in-the-middle attack;

It is **a padding oracle attack**;

It takes advantage of Internet and security **software clients' fallback to SSL 3.0**

If attackers successfully exploit this vulnerability, on average, they only need to make 256 SSL 3.0 requests to reveal one byte of encrypted messages;

Heartbleed (2014)

Heartbleed

It is a **security bug** (**OpenSSL** crypto library);

It is a **buffer over-read**;

No input validation (due to **a missing bounds check**);

An attacker can read the memory of the systems protected by the vulnerable versions of the OpenSSL software;

An attacker is able to steal secret keys of certificates, user passwords, business critical documents, ...

FREAK:
Factoring RSA Export Keys
(2015)

Factoring RSA Export Keys

Flaw known since 1990s but exploited in 2015.

Main idea:

- to manipulate the initial cipher suite negotiation (MITM);
- the compliance with **U.S. cryptography export regulations** (RSA moduli of 512 bits).

1. The client asks for a “standard RSA” ciphersuite;
2. The attacker changes such a message with “export RSA” ciphersuite;
3. The server provide a 512-bit export RSA key, while the client accepts it;

Factoring RSA Export Keys

4. The attacker factors the weak RSA key;
5. When the client sends the *encrypted pre-master secret*, the attacker can decrypt it;
6. The next step is to get the *master secret.*

36.7% of the HTTPS servers with browser-trusted certificates (14 million sites) were vulnerable to FREAK, included nsa.gov, whitehouse.gov, irs.gov, tips.fbi.gov, connect.facebook.net, ...

26.3% of all HTTPS servers;

Several browsers were vulnerable to the FREAK attack.

TLS 1.3

Introduction TLS 1.3

In 2018, IETF states that

- about 81% of communications are encrypted (TLS or SSL)
- about 11% of hosts in the Internet use SSL
- about 89% of hosts in the Internet use TLS (TLS 1.3 excluded)

The differences between TLS 1.3 and TLS 1.0/1.1/1.2: **performances** and **security**.

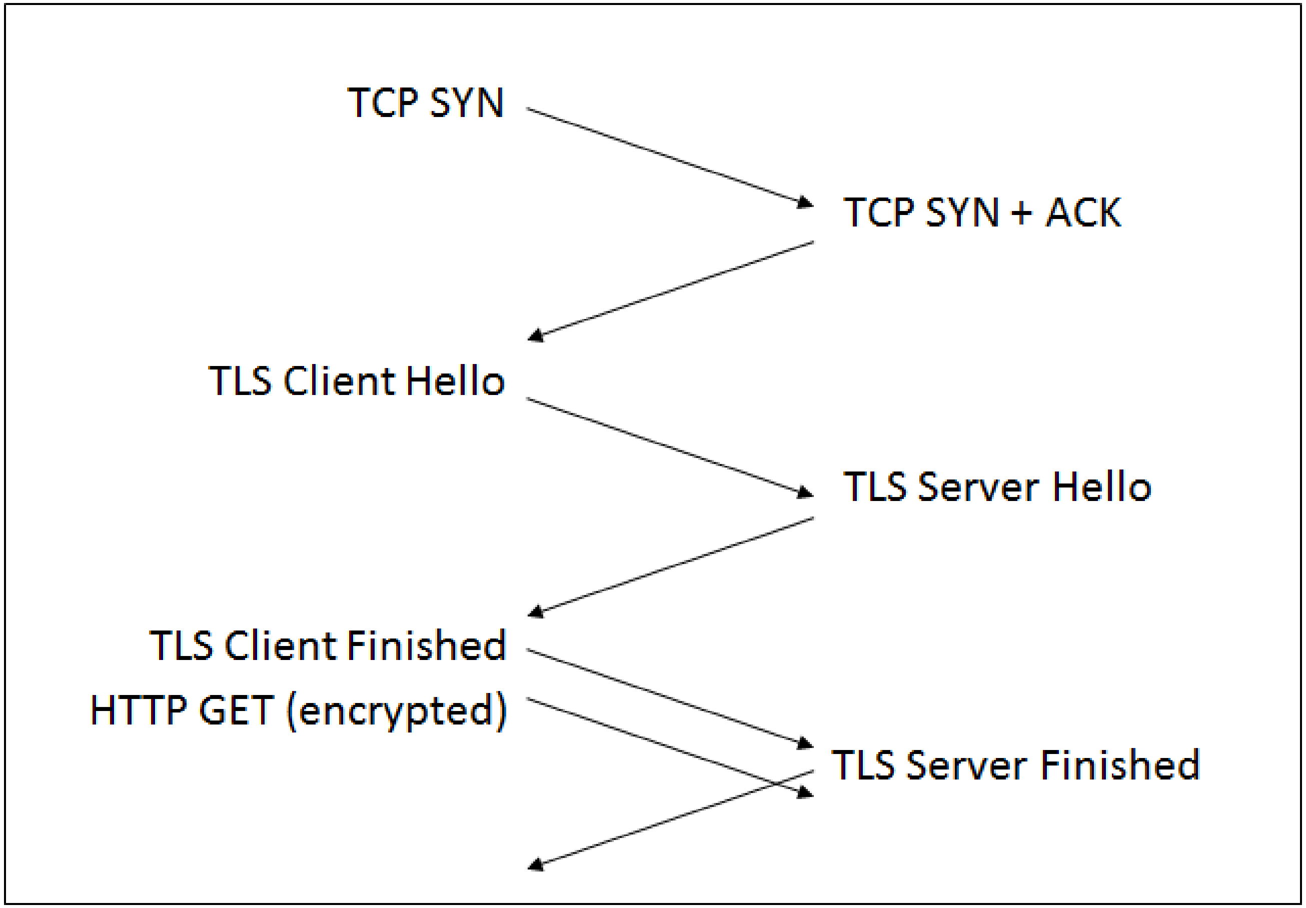
Introduction TLS 1.3

Performaces:

- TLS false start
- TCP fast open
- Zero-One Trip Time (0-RTT)

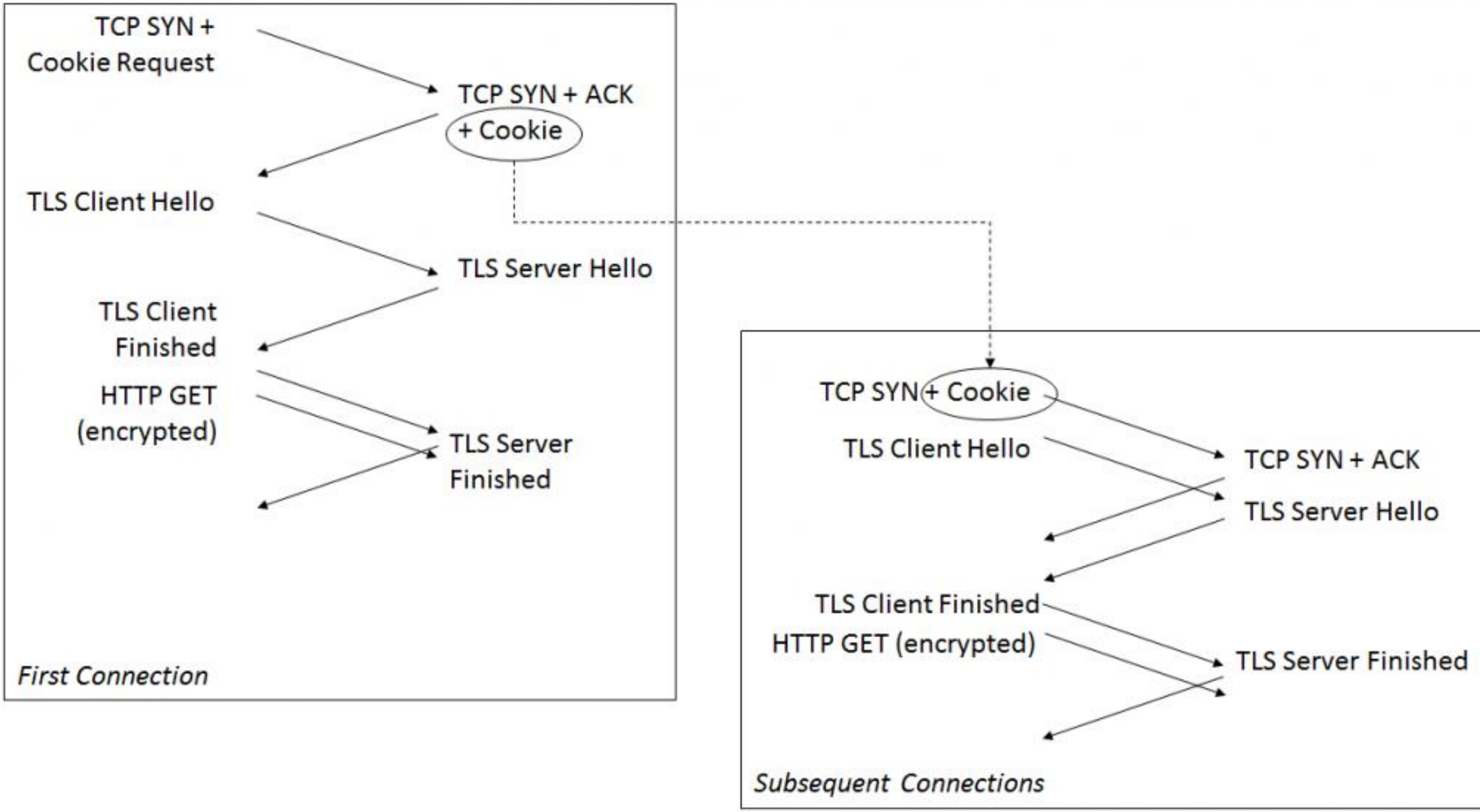
Introduction TLS 1.3

TLS false start:



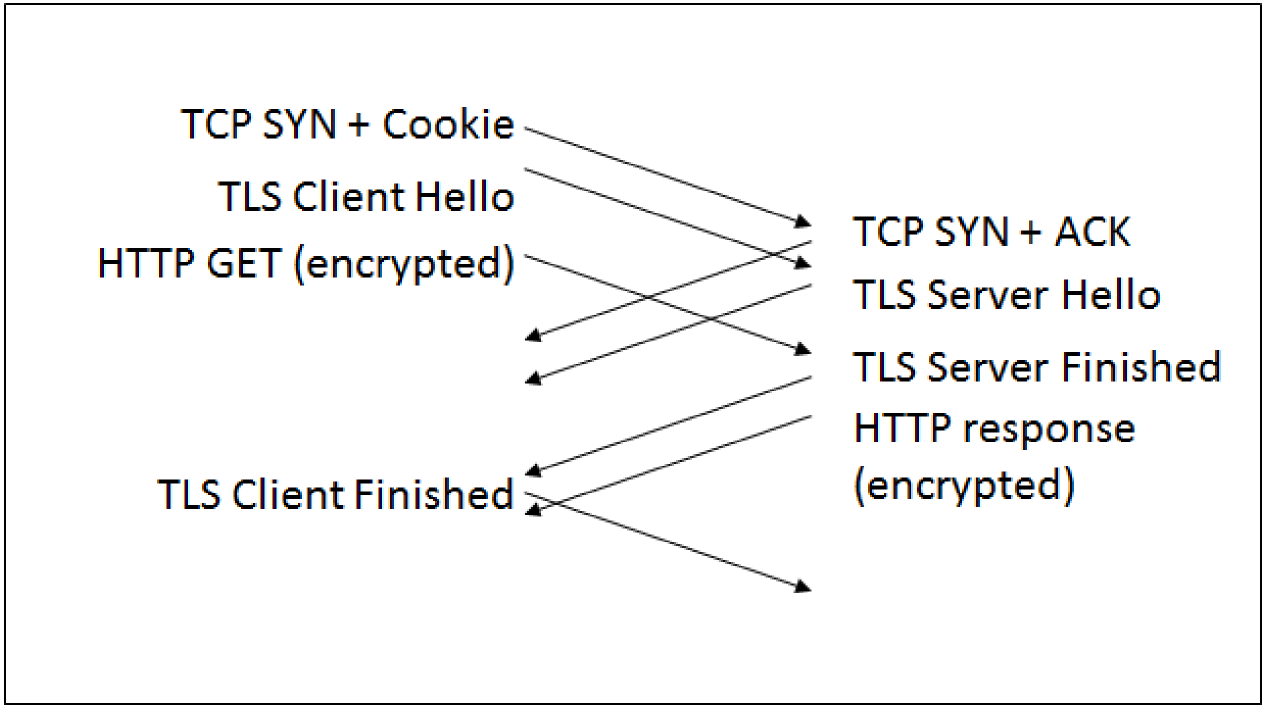
Introduction TLS 1.3

TCP fast open:



Introduction TLS 1.3

Zero-One Trip Time (0-RTT):



Introduction TLS 1.3

Security:

- TLS 1.2 must be configured properly;
- TLS 1.3 removes the deprecated ciphers and features by default;

Ciphers removed: MD5, SHA-1 (backward compatibility), SHA-224, RC4, DES, 3DES, AES-CBC, export-strength cipher, …

Thank your for your attention

andrea.visconti@unimi.it
www.di.unimi.it/visconti