Some thoughts on SSL/TLS from a (nearly) 6-year PhD student

Olivier Levillain

ANSSI

2015-11-23

Who am I?

Olivier Levillain (`@pictyeye`)

- M2 internship in cryptography: study of a hash function
- member of the systems lab at ANSSI (2007-2012)
- head of the network lab at ANSSI (2012-2015)
- head of the training center (CFSSI) (2015-)

Research

- part of the low-level x86 security work (SMM/ACPI)
- PhD student working on SSL/TLS since 2011
- Participation to languages studies since 2007
- some work on binary *parsers*

Teaching

- cryptography: hash function and cryptanalysis
- systems module for the CFSSI
- courses on SSL/TLS, and more recently on secure development

ANSSI

ANSSI (French Network and Information Security Agency) has InfoSec (and no Intelligence) missions:

- detect and early react to cyber attacks
- prevent threats by supporting the development of trusted products and services
- provide reliable advice and support
- communicate on information security threats and the related means of protection

These missions concern:

- governmental entities
- companies
- the general public

TLS: a quick tour

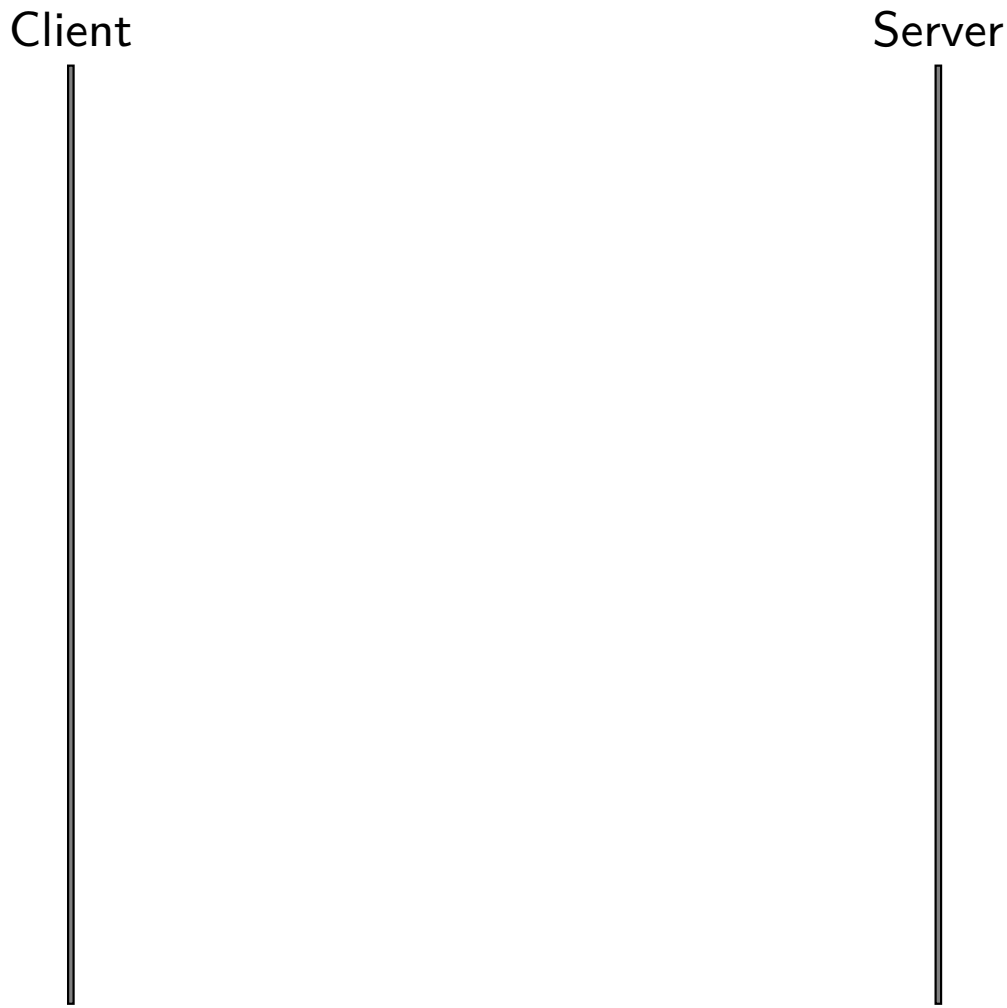
SSL/TLS: an essential building block of Internet

- `https://` invented by Netscape in 1995
  - the beginning of the e-commerce
- Massive usage of SSL/TLS today
  - HTTPS, well beyond e-commerce websites
  - A way to secure other protocols (SMTP, IMAP, LDAP...)
  - SSL VPN
  - EAP TLS

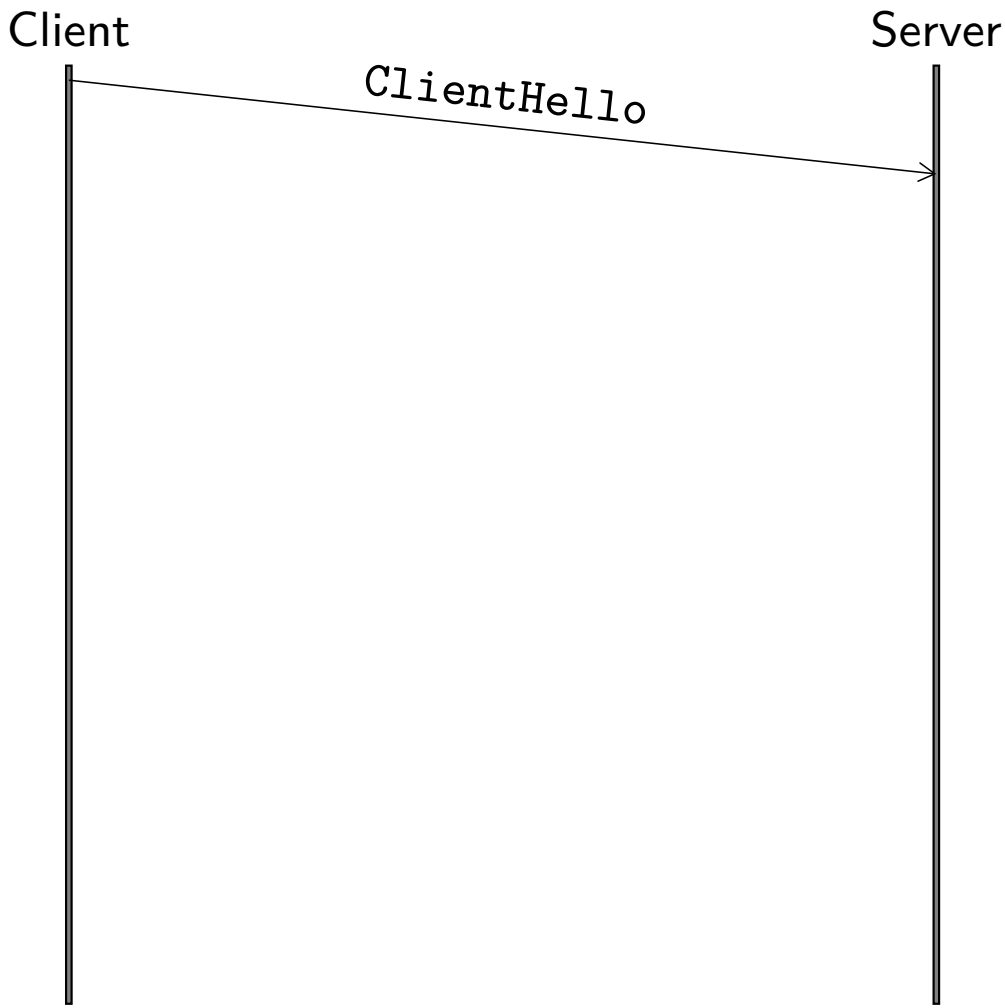
SSL/TLS: an essential building block of Internet

- `https://` invented by Netscape in 1995
  - the beginning of the e-commerce
- Massive usage of SSL/TLS today
  - HTTPS, well beyond e-commerce websites
  - A way to secure other protocols (SMTP, IMAP, LDAP...)
  - SSL VPN
  - EAP TLS
- SSL (*Secure Sockets Layer*) or TLS (*Transport Layer Security*) ?
  - SSLv2 (1995) and v3 (1996) designed by Netscape
  - TLS 1.0 (2001) a.k.a. SSLv3.1, handled by IETF
  - New revisions since: 1.1 (2006), 1.2 (2008) and 1.3 (2016?)

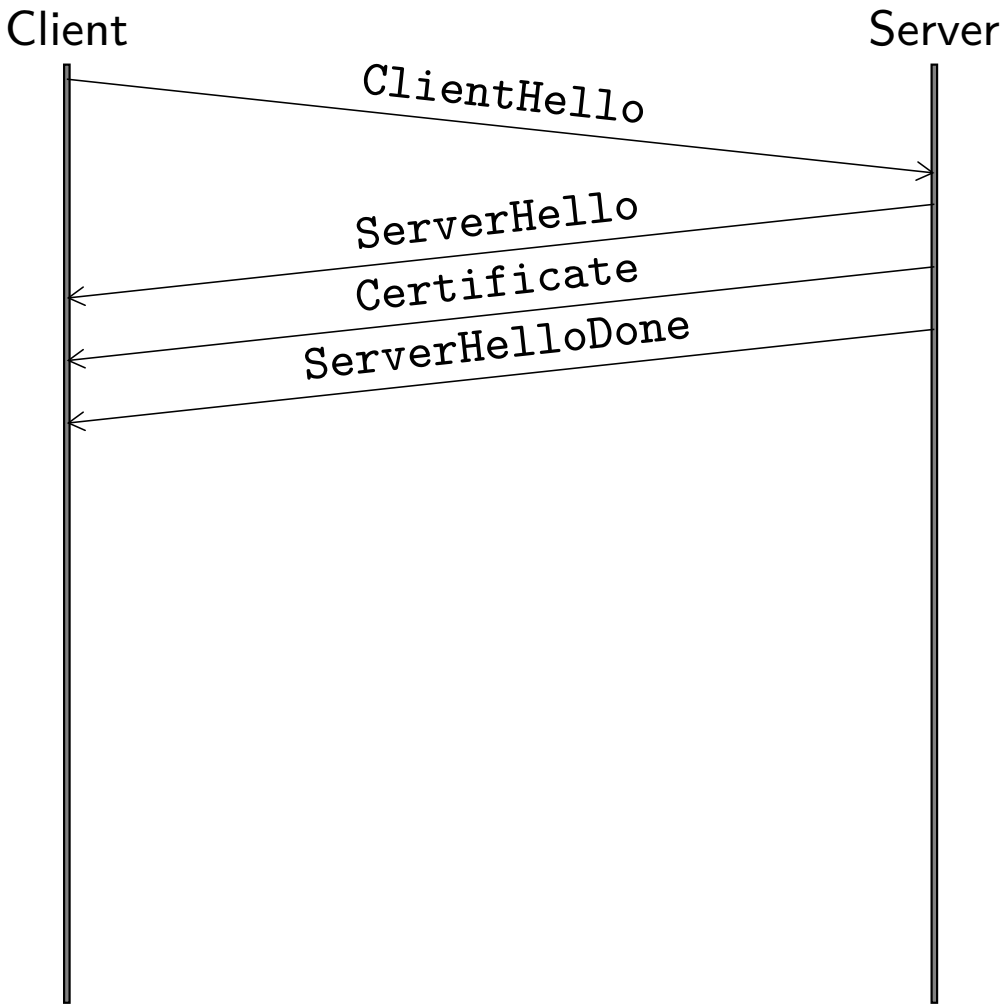
Fonctionnement du protocole



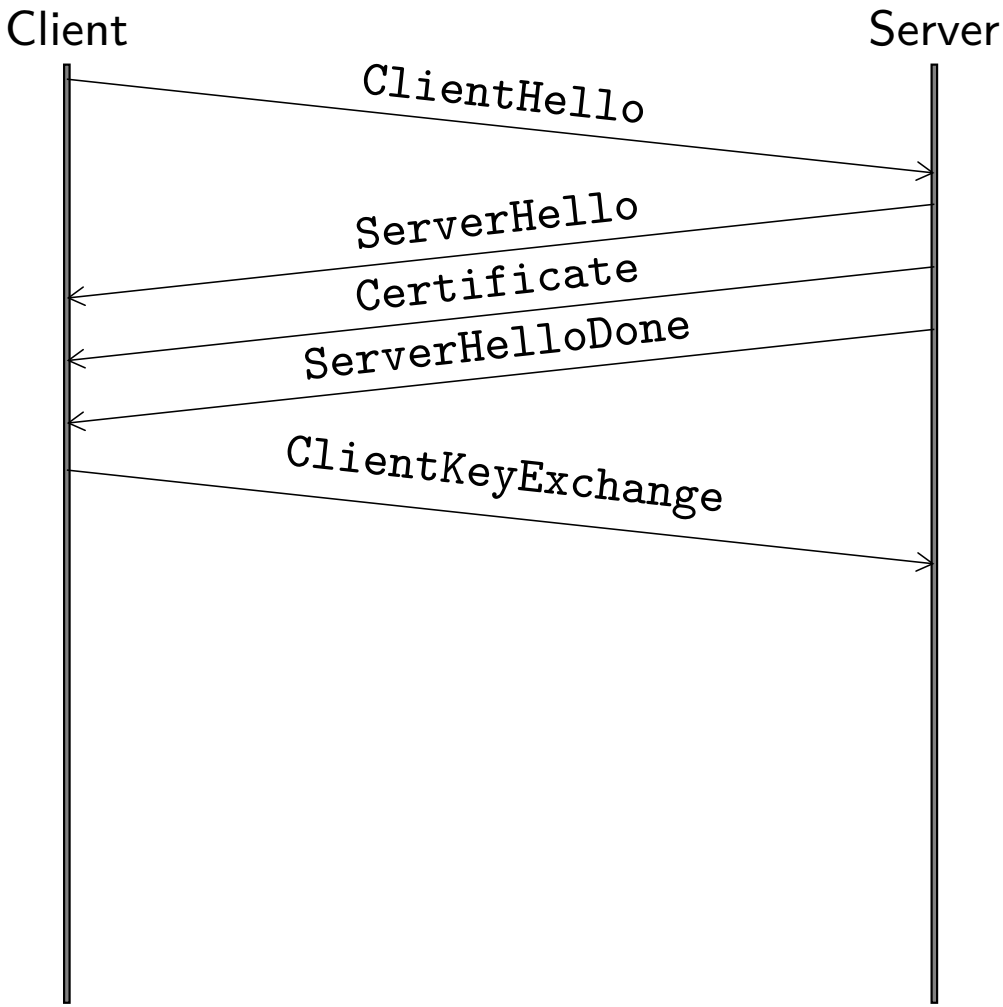
Fonctionnement du protocole



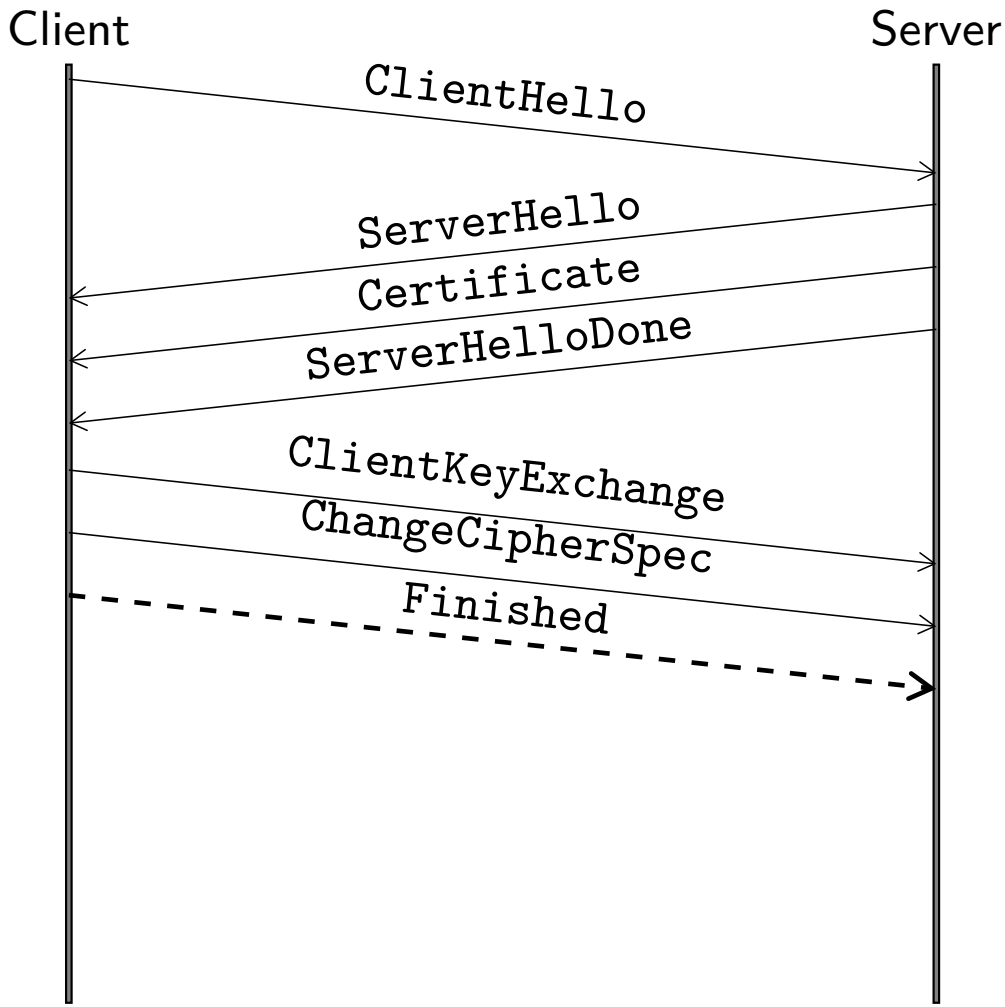
Fonctionnement du protocole



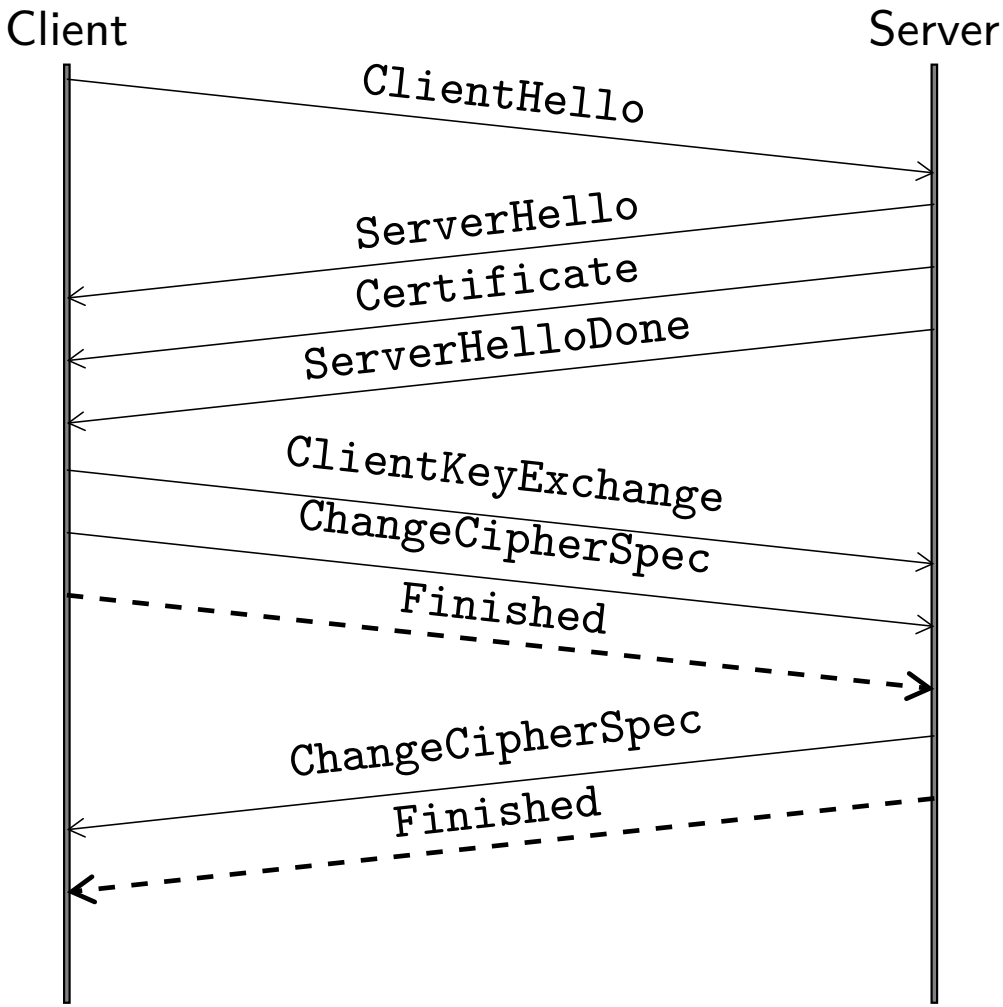
Fonctionnement du protocole



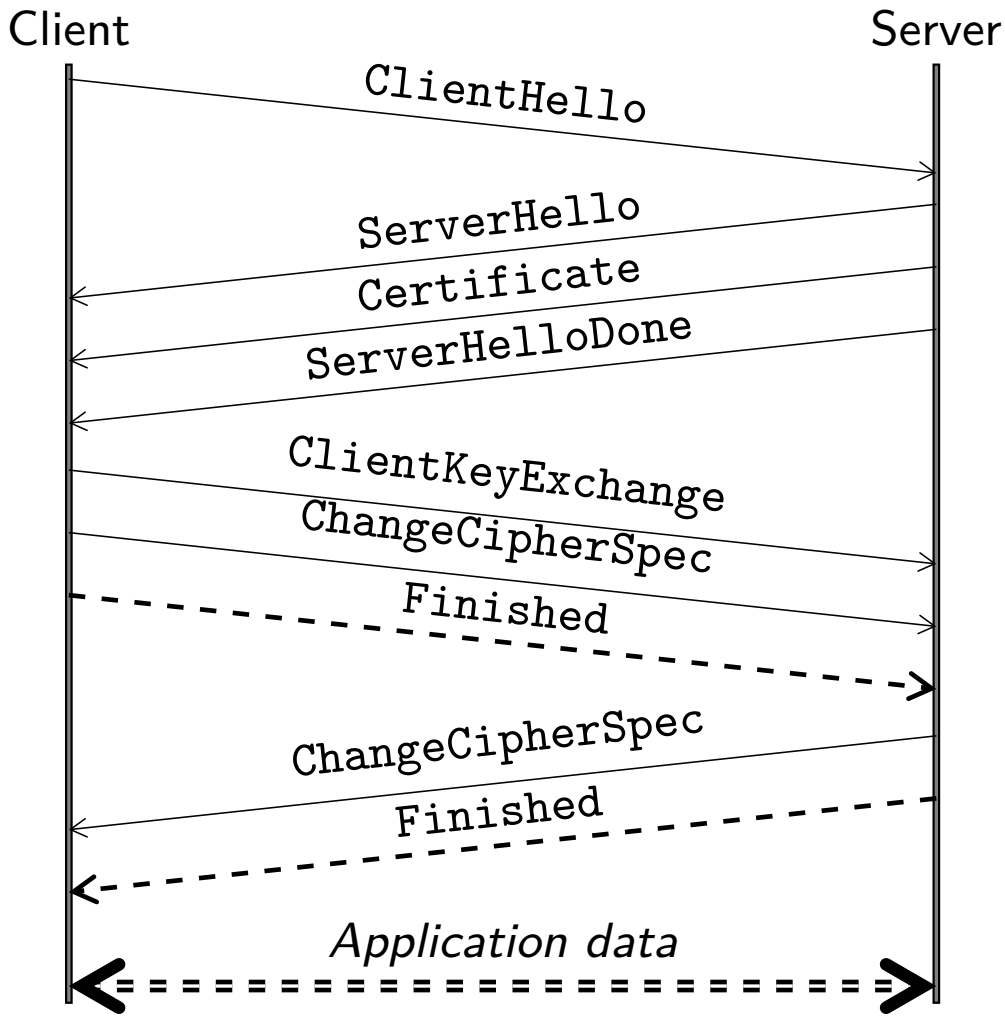
Fonctionnement du protocole



Fonctionnement du protocole



Fonctionnement du protocole



Some figures about SSL/TLS

- More than 50 RFC
- 5 protocol versions for the moment
- More than 300 ciphersuites
- More than 20 extensions
- Some *interesting* features
  - compression
  - renegotiation
  - session resumption (2 methods)
- A dozen well known implementations
- How many home-made implementations ?

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D something else (`RC4_MD5`)

Home-made SSL/TLS stacks (1/3)

What can a TLS server answer to a client proposing the following ciphersuites: `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D something else (`RC4_MD5`)

The explanation is a little sad:

- a ciphersuite is a 16-bit integer
- until (relatively) recently, all ciphersuites were of the form `00 XX`
- so why bother with the most significant byte?

Home-made SSL/TLS stacks (2/3)

- In 2010, Google proposed some extensions, *False Start* and *Snap Start*
- After several months, the Internet seemed intolerant to *Snap Start*
- The proposal was withdrawn in 2012

Home-made SSL/TLS stacks (2/3)

- In 2010, Google proposed some extensions, *False Start* and *Snap Start*
- After several months, the Internet seemed intolerant to *Snap Start*
- The proposal was withdrawn in 2012

- One year later, the same problem reappears in another context
- On the WG mailing list (`tls@ietf.org`), someone speaks up and explains the issue: the `ClientHello` is too big...

Home-made SSL/TLS stacks (3/3)

Here is the beginning of a 258-byte long `ClientHello`

16	03	01	01	02

TLS

Type	Version	Length
HS	*TLS 1.0*	*258*

SSLv2

Length	Pad.	Type
5635	...	*CH*

A TLS `ClientHello` with a size between 256 and 511 can be seen as an SSLv2 `ClientHello`!

Home-made SSL/TLS stacks (3/3)

Here is the beginning of a 258-byte long `ClientHello`

16	03	01	01	02

TLS

Type	Version	Length
HS	*TLS 1.0*	*258*

SSLv2

Length	Pad.	Type
5635	...	*CH*

A TLS `ClientHello` with a size between 256 and 511 can be seen as an SSLv2 `ClientHello`!

In the end, all is well

- Google's new proposal: an extension to pad `ClientHello`...

A brief history of SSL/TLS vulnerabilities

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2

- 2009: renegotiation attack

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 2009: renegotiation attack
- 2014: Triple Handshake (attack mixing renegotiation and session resumption)

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 1998: Bleichenbacher attack on PKCS#1 v1.5

- 2009: MD5 collision on real certificates
- 2009: renegotiation attack

- 2011: BEAST (implicit IV in CBC mode)

- 2013: Lucky 13 (CBC padding oracle) + statistical biases in RC4

- 2014: Triple Handshake (attack mixing renegotiation and session resumption)

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 1998: Bleichenbacher attack on PKCS#1 v1.5
- 2002: wrong interpretation of X.509 *Basic Constraints* extension (IE)

- 2009: MD5 collision on real certificates
- 2009: renegotiation attack
- 2009: confusion in handling null characters in certificates
- 2011: BEAST (implicit IV in CBC mode)

- 2013: Lucky 13 (CBC padding oracle) + statistical biases in RC4
- 2014: Apple's `goto fail`
- 2014: certificate validation bypass in GnuTLS (the other `goto fail`)
- 2014: Triple Handshake (attack mixing renegotiation and session resumption)
- 2014: Heartbleed
- 2014: EarlyCCS
- 2015: FREAK and LogJam

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 1998: Bleichenbacher attack on PKCS#1 v1.5
- 2002: wrong interpretation of X.509 *Basic Constraints* extension (IE)
- 2008: certificate validation bypass in OpenSSL
- 2009: MD5 collision on real certificates
- 2009: renegotiation attack
- 2009: confusion in handling null characters in certificates
- 2011: BEAST (implicit IV in CBC mode)

- 2013: Lucky 13 (CBC padding oracle) + statistical biases in RC4
- 2014: Apple's `goto fail`
- 2014: certificate validation bypass in GnuTLS (the other `goto fail`)
- 2014: Triple Handshake (attack mixing renegotiation and session resumption)
- 2014: Heartbleed
- 2014: EarlyCCS
- 2015: FREAK and LogJam

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 1998: Bleichenbacher attack on PKCS#1 v1.5
- 2002: wrong interpretation of X.509 *Basic Constraints* extension (IE)
- 2008: certificate validation bypass in OpenSSL
- 2009: MD5 collision on real certificates
- 2009: renegotiation attack
- 2009: confusion in handling null characters in certificates
- 2011: BEAST (implicit IV in CBC mode)
- 2011: wrong interpretation of X.509 *Basic Constraints* extension (iOS)

- 2013: Lucky 13 (CBC padding oracle) + statistical biases in RC4
- 2014: Apple's `goto fail`
- 2014: certificate validation bypass in GnuTLS (the other `goto fail`)
- 2014: Triple Handshake (attack mixing renegotiation and session resumption)
- 2014: Heartbleed
- 2014: EarlyCCS
- 2015: FREAK and LogJam

A brief history of SSL/TLS vulnerabilities

- 1995: down-negotiation in SSLv2
- 1998: Bleichenbacher attack on PKCS#1 v1.5
- 2002: wrong interpretation of X.509 *Basic Constraints* extension (IE)
- 2008: certificate validation bypass in OpenSSL
- 2009: MD5 collision on real certificates
- 2009: renegotiation attack
- 2009: confusion in handling null characters in certificates
- 2011: BEAST (implicit IV in CBC mode)
- 2011: wrong interpretation of X.509 *Basic Constraints* extension (iOS)
- 2012: *Mining your Ps and Qs* (bad random used in RSA key generation)
- 2013: Lucky 13 (CBC padding oracle) + statistical biases in RC4
- 2014: Apple's `goto fail`
- 2014: certificate validation bypass in GnuTLS (the other `goto fail`)
- 2014: Triple Handshake (attack mixing renegotiation and session resumption)
- 2014: Heartbleed
- 2014: EarlyCCS
- 2015: FREAK and LogJam

SSL/TLS design flaws in the first phase

The first step in an SSL/TLS connection is to authenticate the server and share a common secret

Some examples of what could go wrong

SSL/TLS design flaws in the first phase

The first step in an SSL/TLS connection is to authenticate the server and share a common secret

Some examples of what could go wrong

- SSLv2 down-negotiation: only the random values exchanged were authenticated

SSL/TLS design flaws in the first phase

The first step in an SSL/TLS connection is to authenticate the server and share a common secret

Some examples of what could go wrong

- SSLv2 down-negotiation: only the random values exchanged were authenticated
- Renegotiation / Triple Handshake: different sessions/epochs were not bound in a secure manner

About *master secret* and key derivation (1/2)

The *master secret* is derived from

- the result of the key exchange algorithm
- random values sent in plaintext

Integrity is only guaranteed afterwards when `Finished` messages are exchanged

About *master secret* and key derivation (1/2)

The *master secret* is derived from

- the result of the key exchange algorithm
- random values sent in plaintext

Integrity is only guaranteed afterwards when `Finished` messages are exchanged

Several attacks rely on this limited coverage in the derivation

- theoretical cross-protocol attacks were presented in 1999
- (almost) practical instanciations were published in 2012
- Triple Handshake used this in 2014
- SMACK/FREAK 2015
- LogJam in 2015

About *master secret* and key derivation (2/2)

Fix proposal

- session-hash extension: the derivation should hash all the previously exchanged messages
- RFC 7627 for TLS 1.{0,1,2}
- native feature of TLS 1.3

About *master secret* and key derivation (2/2)

Fix proposal

- session-hash extension: the derivation should hash all the previously exchanged messages
- RFC 7627 for TLS 1.{0,1,2}
- native feature of TLS 1.3

However, this is not enough, as *LogJam* proved it

- the DH group is chosen by the server using unauthenticated information (possibly a weak one such as 512-bit DH group)
- the chosen group is signed along with the exchanged random values
- an attacker can break the discrete logarithm and reuse this message to control the communication

About *master secret* and key derivation (2/2)

Fix proposal

- session-hash extension: the derivation should hash all the previously exchanged messages
- RFC 7627 for TLS 1.{0,1,2}
- native feature of TLS 1.3

However, this is not enough, as *LogJam* proved it

- the DH group is chosen by the server using unauthenticated information (possibly a weak one such as 512-bit DH group)
- the chosen group is signed along with the exchanged random values
- an attacker can break the discrete logarithm and reuse this message to control the communication

The real fix: sign all the previously exchanged messages, not only the random values

Attacks against the Record Protocol

Since 2011, a lot of practical attacks targetting the Record Protocol

- BEAST: against CBC with implicit IV (versions before TLS 1.1)
- CRIME: refined with TIME and BREACH, using TLS (or HTTP) compression as a side-channel
- Lucky 13: padding oracle against CBC
- RC4 biases: statistical biases allowing for plaintext recovery
- POODLE: more efficient padding oracle against SSLv3

Attacks against the Record Protocol

Since 2011, a lot of practical attacks targetting the Record Protocol

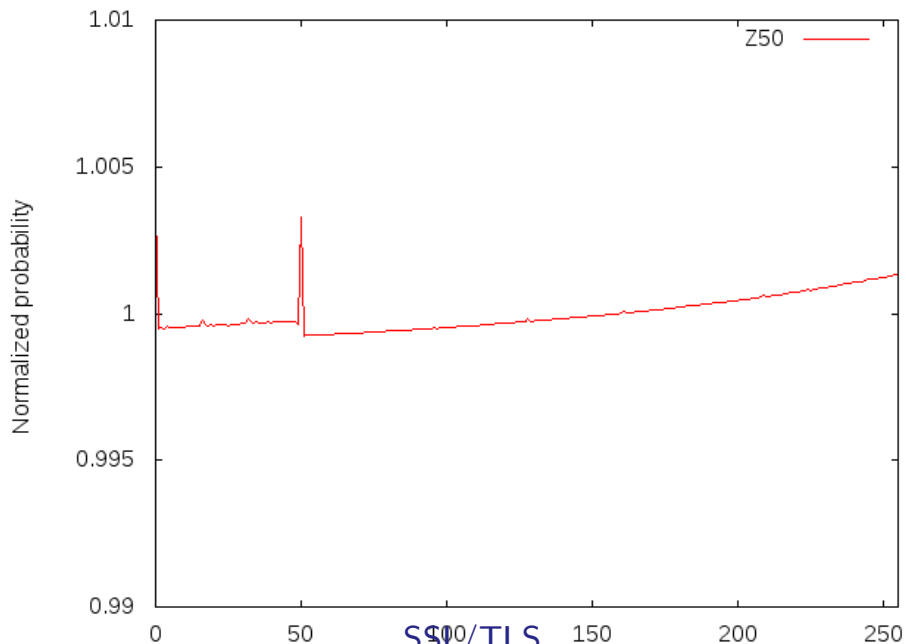
- BEAST: against CBC with implicit IV (versions before TLS 1.1)
- CRIME: refined with TIME and BREACH, using TLS (or HTTP) compression as a side-channel
- Lucky 13: padding oracle against CBC
- RC4 biases: statistical biases allowing for plaintext recovery
- POODLE: more efficient padding oracle against SSLv3

All these have similarities: they target HTTP authenticated cookies

- indeed, cookies are repeated across TLS sessions, making some attacks possible

Focus on RC4

- 1987: RC4, a proprietary streamcipher designed by Rivest
- 1994: RC4 leaks under the name ARCFOUR
- 1995-2000: biases related to the first bytes of the keystream
- 2001: WEP (which uses correlated keys) is broken by Flührer, Mantin, and Shamir
- 2013: exploitable biases are discovered in the TLS use case
- 2014: more, better, attacks



Some thoughts about CBC mode

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

Some thoughts about CBC mode

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

- Beware of padding oracles with CBC mode in TLS
- CBC mode with implicit IV may be subject to attacks
- With SSLv3, the padding is not well specified, which aggravates padding oracles

Some thoughts about CBC mode

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller (2004-05-20)

- Beware of padding oracles with CBC mode in TLS
- CBC mode with implicit IV may be subject to attacks
- With SSLv3, the padding is not well specified, which aggravates padding oracles
- ... he was describing Lucky 13, BEAST and POODLE

2014: terrible year for TLS

In 2014, all major TLS stacks were affected by a critical vulnerability

- February : `goto fail` in Apple
- February : `goto fail` in GnuTLS
- April : *Heartbleed* in OpenSSL
- June : *Early CCS* in OpenSSL
- September : Universal signature forgery (Berserk?) in NSS (Mozilla)
- September : Universal signature forgery (Berserk?) in CyaSSL
- September : Universal signature forgery (Berserk?) in PolarSSL (now mbedTLS)
- November : remote code execution in SChannel (MS)

2014: terrible year for TLS

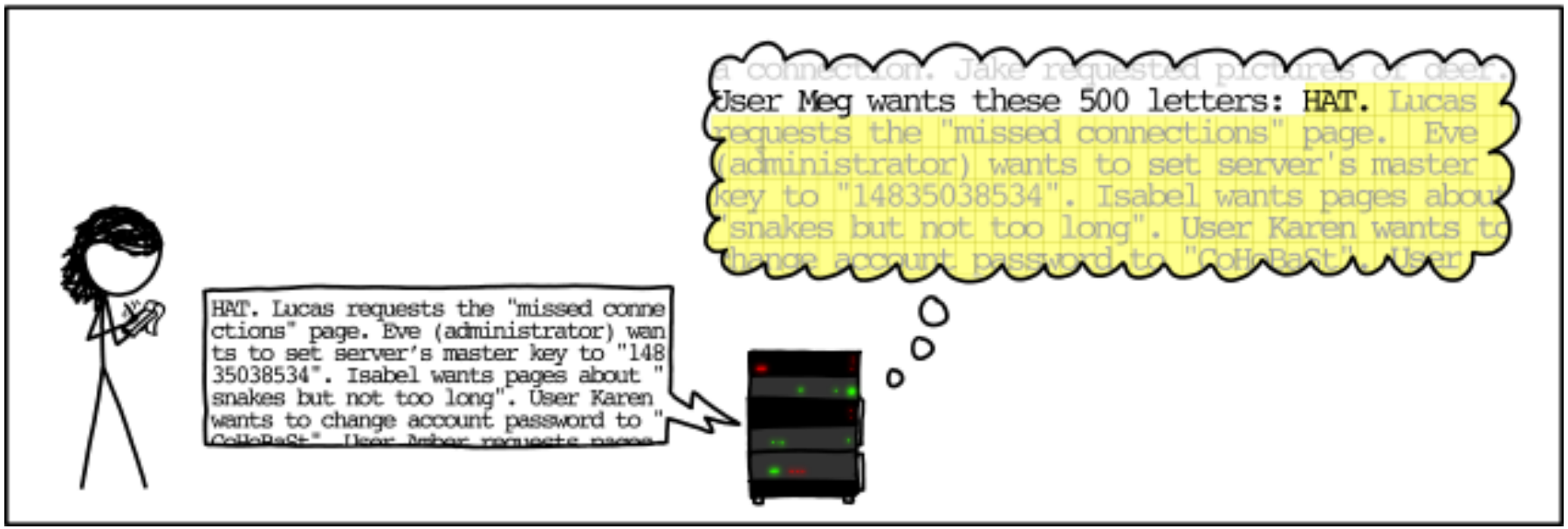
In 2014, all major TLS stacks were affected by a critical vulnerability

- February : `goto fail` in Apple
- February : `goto fail` in GnuTLS
- April : *Heartbleed* in OpenSSL
- June : *Early CCS* in OpenSSL
- September : Universal signature forgery (Berserk?) in NSS (Mozilla)
- September : Universal signature forgery (Berserk?) in CyaSSL
- September : Universal signature forgery (Berserk?) in PolarSSL (now mbedTLS)
- November : remote code execution in SChannel (MS)

Quizz

- What also happened on Heartbleed's day (April 8th, 2014)?
- What also happened on Berserk's day (September 24th, 2014)?

A typical buffer overflows: Heartbleed



Source: `http://xkcd.com/1354`

```
+        /* Read type and payload length first */
+        if (1 + 2 + 16 > s->s3->rrec.length)
+                return 0; /* silently discard */
```

Another silly buffer overflow: WinShock

- Client authentication using Elliptic Curve certificates rely on two more messages
- The `Certificate` message, containing the certificate chain
  - it contains the used curve
  - in particular, it indicates the length of the coordinates L
- The `CertificateVerify`, containing a signature covering previous messages
  - the signature contains coordinates, with a length l
  - what if SChannel copied l bytes in a L-byte buffer with no checks...

Another silly buffer overflow: WinShock

- Client authentication using Elliptic Curve certificates rely on two more messages
- The `Certificate` message, containing the certificate chain
  - it contains the used curve
  - in particular, it indicates the length of the coordinates L
- The `CertificateVerify`, containing a signature covering previous messages
  - the signature contains coordinates, with a length l
  - what if SChannel copied l bytes in a L-byte buffer with no checks...

But wait?! Nearly noone uses TLS client authentication with certificates?

Another silly buffer overflow: WinShock

- Client authentication using Elliptic Curve certificates rely on two more messages
- The `Certificate` message, containing the certificate chain
  - it contains the used curve
  - in particular, it indicates the length of the coordinates L
- The `CertificateVerify`, containing a signature covering previous messages
  - the signature contains coordinates, with a length l
  - what if SChannel copied l bytes in a L-byte buffer with no checks...

But wait?! Nearly noone uses TLS client authentication with certificates?

Teaser: all vulnerable server stacks were nevertheless exploitable

Apple's goto fail

```
/* Extract from Apple's sslKeyExchange.c */
if ((err=SSLHashSHA1.update(&hashCtx,&serverRandom))!=0)
    goto fail;
if ((err=SSLHashSHA1.update(&hashCtx,&signedParams))!=0)
    goto fail;
    goto fail;
if ((err=SSLHashSHA1.final(&hashCtx,&hashOut))!=0)
    goto fail;
```

Syntax doesn't help, but the compiler doesn't seem concerned about signaling obviously dead code...

True, False, FILE_NOT_FOUND

Focus CVE-2014-0092 on GnuTLS, in March 2014:

> *But this bug is arguably much worse than* Apple*'s, as it has allowed crafted certificates to evade validation check for all versions of* GnuTLS *ever released since that project got started in late 2000.[...]*
> *The* `check_if_ca` *function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

True, False, FILE_NOT_FOUND

Focus CVE-2014-0092 on GnuTLS, in March 2014:

> *But this bug is arguably much worse than* Apple*'s, as it has allowed crafted certificates to evade validation check for all versions of* GnuTLS *ever released since that project got started in late 2000.[...]*
> *The* `check_if_ca` *function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

By the way, a similar bug was found in OpenSSL... in 2008 (CVE-2008-5077).

Basic constraints omitted

A bug found in 2002 by Marlinspike in Microsoft Internet Explorer:

- the X509 stack did not check the `Basic Constraints` extension
- every end certificate could be reused as a CA

Basic constraints omitted

A bug found in 2002 by Marlinspike in Microsoft Internet Explorer:

- the X509 stack did not check the `Basic Constraints` extension
- every end certificate could be reused as a CA

The same bug was found, in 2010, in Apple iOS...

Basic constraints omitted

A bug found in 2002 by Marlinspike in Microsoft Internet Explorer:

- the X509 stack did not check the `Basic Constraints` extension
- every end certificate could be reused as a CA

The same bug was found, in 2010, in Apple iOS...

Maybe we should not always blame the developer... and improve our languages/tools/specs

Bleichenbacher

RSA PKCS#1 v1.5

- RSA encryption relies on a padding scheme
- how to handle an invalid padding at decryption time?

Bleichenbacher

RSA PKCS#1 v1.5

- RSA encryption relies on a padding scheme
- how to handle an invalid padding at decryption time?

Bleichenbacher attack (1998)

- principle: send an altered ciphertext
- if the attacker can distinguish a valid padding from an invalid one, plaintext recovery is possible
- application to TLS: the so-called Million Message Attack

Bleichenbacher

RSA PKCS#1 v1.5

- RSA encryption relies on a padding scheme
- how to handle an invalid padding at decryption time?

Bleichenbacher attack (1998)

- principle: send an altered ciphertext
- if the attacker can distinguish a valid padding from an invalid one, plaintext recovery is possible
- application to TLS: the so-called Million Message Attack

The attack reappeared in 2014

- In Java crypto library, a padding error leads to an exception
- To avoid a timing attack, one must reimplement the algorithm
- A TLS implementer has to choose between code reuse and security

MAC-then-Encrypt

Padding issues also exist in symmetric crypto

- Using *MAC-then-CBC* is vulnerable
- 2002: Vaudenay presents the attack principle
- 2011: *XML Encryption is broken* paper
- 2013: Lucky 13 proves applicability to TLS

MAC-then-Encrypt

Padding issues also exist in symmetric crypto

- Using *MAC-then-CBC* is vulnerable
- 2002: Vaudenay presents the attack principle
- 2011: *XML Encryption is broken* paper
- 2013: Lucky 13 proves applicability to TLS

Fix?

- duct-tape (add an implementation note in the standard)
- a sordid patch to ensure constant-time decryption
- use `Encrypt-then-MAC` (RFC 7366) or secure AEAD schemes

Again, one had to choose between modularity and security

SMACK and FREAK

In 2015, many attacks were published against nearly all TLS stacks

- In Java, sending an early `Finished` allows an attacker to skip all the negotiation (including the server authentication)

SMACK and FREAK

In 2015, many attacks were published against nearly all TLS stacks

- In Java, sending an early `Finished` allows an attacker to skip all the negotiation (including the server authentication)
- In several stacks, confusion between elliptic points used in ECDHE and ECDSA when some messages are skipped

SMACK and FREAK

In 2015, many attacks were published against nearly all TLS stacks

- In Java, sending an early `Finished` allows an attacker to skip all the negotiation (including the server authentication)
- In several stacks, confusion between elliptic points used in ECDHE and ECDSA when some messages are skipped
- OpenSSL accepted RSA-EXPORT key exchange instead of plain RSA key exchange (FREAK)

SMACK and FREAK

In 2015, many attacks were published against nearly all TLS stacks

- In Java, sending an early `Finished` allows an attacker to skip all the negotiation (including the server authentication)
- In several stacks, confusion between elliptic points used in ECDHE and ECDSA when some messages are skipped
- OpenSSL accepted RSA-EXPORT key exchange instead of plain RSA key exchange (FREAK)

In general, the state machine is drived by the received messages, whereas the implementation should keep track of the expected messages

Other problems with state machines

- Early CCS, also found by formal methods!
- In the previous SChannel vulnerability, even unsollicited client-sent `Certificate` and `CertificateVerify` messages were interpreted

All TLS stacks more or less vulnerable to similar attacks:

- Maybe the specs are too complicated?
- Maybe we need more tests?

TLS 1.3: a new hope?

Among the discussed flaws, a lot of them are *by design*

- Obsolete crypto, finally removed: PKCS#1 v1.5, RC4, CBC...
- State machine: the negotiation phase has been cleaned up
- In parallel to the specification efforts, the protocol is modeled and tested using formal methods (TRON workshop early 2016)

Yet, TLS 1.3 will not solve

- compatibility issues: we will still need to speak TLS 1.2 and earlier for some time...
- the complexity introduced by the new constructions
  - 0-RTT
  - Client authentication and key refresh are not stable yet...

Languages and methodology

Should not programming languages help us?

- Strong typing can help avoid simple bugs
- Memory mangagement done right

Tools can help us

- activate more warnings
- and handle them (`-Werror`)

Add more tests

- non-regression tests
- negative tests

Questions

Thank you for your attention

olivier.levillain@ssi.gouv.fr