The Rocky Road to

Hanno Böck

https://hboeck.de

@hanno

Transport Layer Security

A protocol to create an encrypted and authenticated layer around other protocols

TLS 1.3 was published in August 2018

How did we get there?

In 1995 Netscape introduced Secure Socket Layer or SSL version 2

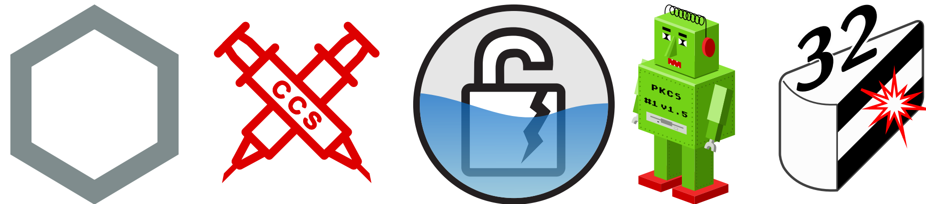
In 1996 it was followed up with SSL version 3

In 1999 the IETF took over and renamed it to TLS

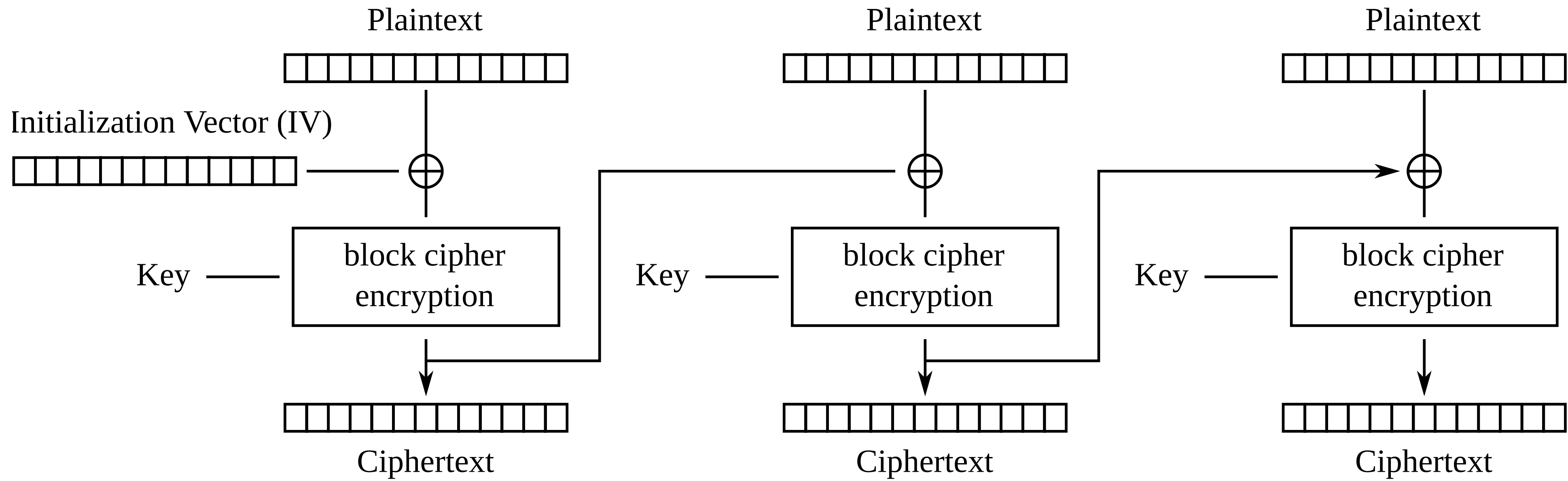
SSL/TLS History

- 1995: SSL 2
- 1996: SSL 3
- 1999: TLS 1.0
- 2006: TLS 1.1
- 2008: TLS 1.2
- 2018: TLS 1.3

Vulnerabilities



Padding Oracles in CBC mode



WhiteTimberwolf, Wikimedia Commons, Public Domain

CBC Padding for Block Ciphers (AES)

Encryption of data blocks means we have to fill up space

CBC in TLS

MAC-then-Pad-then-Encrypt

Valid Padding

00

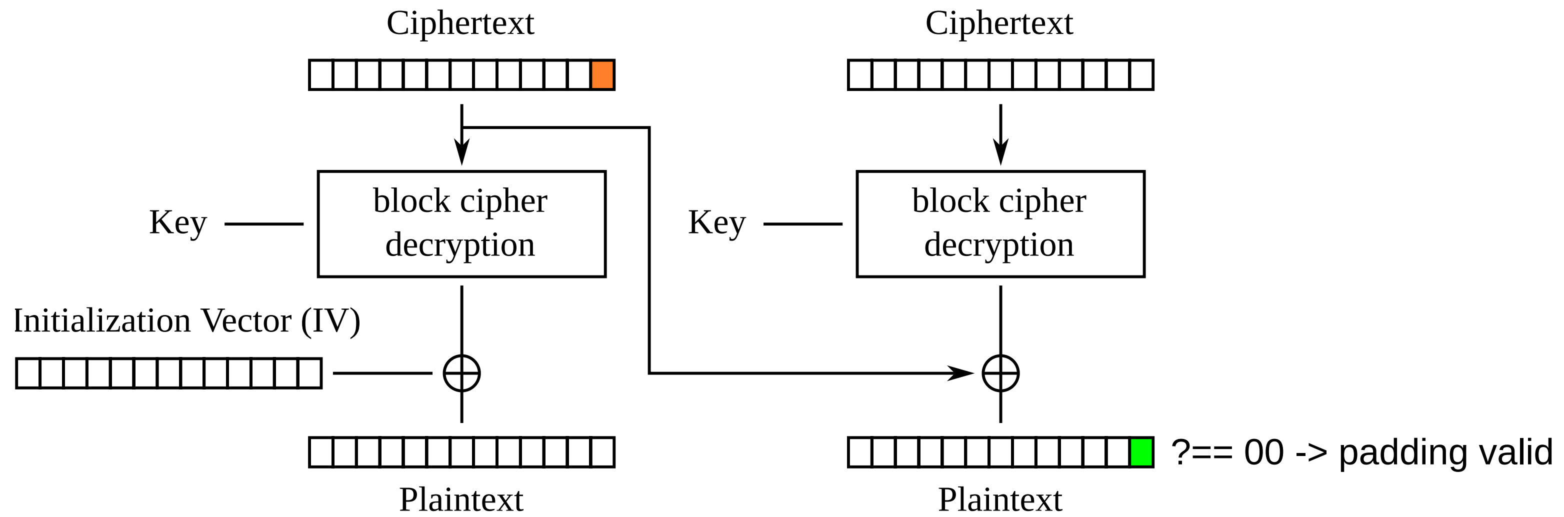
01 01

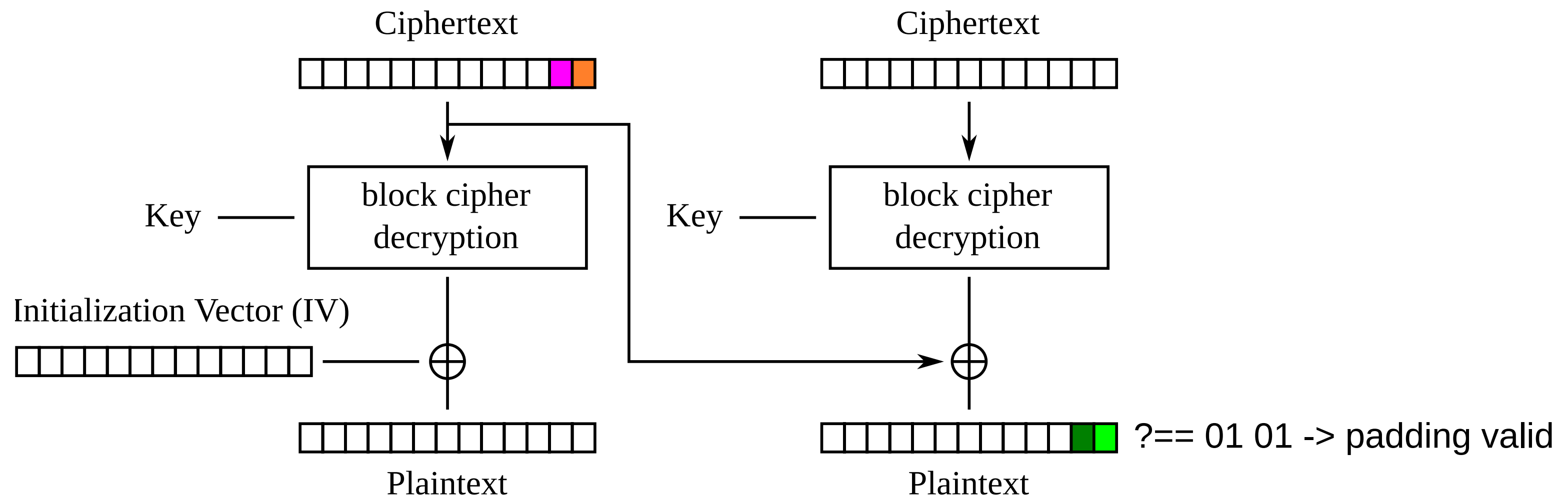
02 02 02

03 03 03 03

...

We assume a situation where the attacker can see whether the padding is valid


Ciphertext
Ciphertext
Key
block cipher decryption
Key
block cipher decryption
Initialization Vector (IV)
Plaintext
Plaintext
?== 00 -> padding valid
Attacker wants to decrypt
Attacker manipulates / XOR with guess


Ciphertext
Ciphertext
Key
block cipher decryption
Key
block cipher decryption
Initialization Vector (IV)
?== 01 01 -> padding valid
Plaintext
Plaintext
Attacker knows
Attacker manipulates to ⊕ ⊕ 01
Attacker wants to know
Attacker manipulates to guess⊕1

2002: Serge Vaudenay discovers Padding Oracle

Vaudenay, 2002

TLS errors

decryption_failed

bad_record_mac

If an attacker can see the TLS error he can use a padding oracle

However TLS errors are encrypted:

Attack is not practical

2003: Timing attack allows practical padding oracle attack

Canvel et al, 2003

TLS 1.2 fixed it (kind of)

This leaves a small timing channel, *since MAC performance depends to some extent on the size of the data fragment,* ***but it is not believed to be large enough to be exploitable,*** *due to the large block size of existing MACs and the small size of the timing signal.*

Lucky Thirteen (2013)

Actually it is large enough to be exploitable

AlFardan, Paterson 2013

It is possible to make TLS with CBC timing safe, but it adds a lot of complexity to the code

POODLE (2014)

SSLv3 has a padding oracle flaw by design

Möller et al, 2014

POODLE-TLS (2014)

Implementations fail to check the padding, making TLS vulnerable to POODLE, too

Langley, 2014

Lucky Microseconds in s2n (2015)

Sorry Amazon, your fix for Lucky Thirteen doesn't work

Albrecht, Paterson, 2015

LuckyMinus20 in OpenSSL (2016)

When OpenSSL tried to fix Lucky Thirteen they introduced another padding oracle

Somorovsky, 2016

The original attack didn't work in practice, because TLS errors are encrypted

But what if there are implementations that create other errors that an attacker can see? For example TCP errors, connection resets or timeouts?

Yes, you can find servers doing that

Bleichenbacher attacks

RSA Encryption

Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1

Daniel Bleichenbacher

Bell Laboratories
700 Mountain Ave., Murray Hill, NJ 07974
bleichen@research.bell-labs.com

Abstract. This paper introduces a new adaptive chosen ciphertext attack against certain protocols based on RSA. We show that an RSA private-key operation can be performed if the attacker has access to an oracle that, for any chosen ciphertext, returns only one bit telling whether the ciphertext corresponds to some unknown block of data encrypted using PKCS #1. An example of a protocol susceptible to our attack is SSL V.3.0.

Keywords: chosen ciphertext attack, RSA, PKCS, SSL

Bleichenbacher, 1998

RSA PKCS #1 1.5 Encryption

```
00 | 02 | [random] | 00 | 03 | 03 | [secret]
```

A valid decryption always starts with 00 02

What shall a server do if it doesn't?

Send an error?

Sending an error tells the attacker something:

Decrypted data does not start with 00 02

Attacker can send modified ciphertext and learn enough to decrypt data

So TLS 1.0 introduced some countermeasures

2003: Klima-Pokorny-Rosa attack

Countermeasures were incomplete

Klima et al, 2003

2014: Java is vulnerable to Bleichenbacher attacks

And OpenSSL via timing

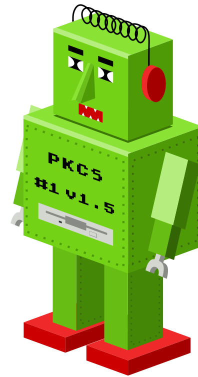
Meyer et al, 2014

2016: DROWN

SSL 2 is vulnerable to Bleichenbacher attacks by design

Aviram et al, 2016

2017: Return of Bleichenbacher's Oracle Threat (ROBOT)



~1/3 of top webpages and at least 15 different implementations vulnerable

Böck, Somorovsky, Young, 2017

2018: 9 Lives of Bleichenbacher's CAT

Cache sidechannels that work against almost most RSA implementations

Ronen et al, 2018

Bleichenbacher attack countermeasures

TLS 1.0

```
Note: An attack discovered by Daniel Bleichenbacher [BLEI] can be used
      to attack a TLS server which is using PKCS#1 encoded RSA. The
      attack takes advantage of the fact that by failing in different
      ways, a TLS server can be coerced into revealing whether a
      particular message, when decrypted, is properly PKCS#1 formatted
      or not.

      The best way to avoid vulnerability to this attack is to treat
      incorrectly formatted messages in a manner indistinguishable from
      correctly formatted RSA blocks. Thus, when it receives an
      incorrectly formatted RSA block, a server should generate a
      random 48-byte value and proceed using it as the premaster
      secret. Thus, the server will act identically whether the
      received RSA block is correctly encoded or not.
```

TLS 1.1

```
Note: An attack discovered by Daniel Bleichenbacher [BLEI] can be
      used to attack a TLS server that is using PKCS#1 v 1.5 encoded
      RSA.  The attack takes advantage of the fact that, by failing
      in different ways, a TLS server can be coerced into revealing
      whether a particular message, when decrypted, is properly
      PKCS#1 v1.5 formatted or not.

      The best way to avoid vulnerability to this attack is to treat
      incorrectly formatted messages in a manner indistinguishable
      from correctly formatted RSA blocks.  Thus, when a server
      receives an incorrectly formatted RSA block, it should generate
      a random 48-byte value and proceed using it as the premaster
      secret.  Thus, the server will act identically whether the
      received RSA block is correctly encoded or not.

      [PKCS1B] defines a newer version of PKCS#1 encoding that is
      more secure against the Bleichenbacher attack.  However, for
      maximal compatibility with TLS 1.0, TLS 1.1 retains the
      original encoding.  No variants of the Bleichenbacher attack
      are known to exist provided that the above recommendations are
      followed.

Implementation Note: Public-key-encrypted data is represented as an
                     opaque vector <0..2^16-1> (see Section 4.7).
                     Thus, the RSA-encrypted PreMasterSecret in a
                     ClientKeyExchange is preceded by two length
                     bytes.  These bytes are redundant in the case of
                     RSA because the EncryptedPreMasterSecret is the
                     only data in the ClientKeyExchange and its
                     length can therefore be unambiguously
                     determined.  The SSLv3 specification was not
                     clear about the encoding of public-key-encrypted
                     data, and therefore many SSLv3 implementations
                     do not include the length bytes, encoding the
                     RSA encrypted data directly in the
                     ClientKeyExchange message.

                     This specification requires correct encoding of
                     the EncryptedPreMasterSecret complete with
                     length bytes.  The resulting PDU is incompatible
                     with many SSLv3 implementations.  Implementors
                     upgrading from SSLv3 must modify their
                     implementations to generate and accept the
                     correct encoding.  Implementors who wish to be
                     compatible with both SSLv3 and TLS should make
                     their implementation's behavior dependent on the
                     protocol version.

Implementation Note: It is now known that remote timing-based attacks
                     on SSL are possible, at least when the client
                     and server are on the same LAN.  Accordingly,
                     implementations that use static RSA keys SHOULD
                     use RSA blinding or some other anti-timing
                     technique, as described in [TIMING].

Note: The version number in the PreMasterSecret MUST be the version
      offered by the client in the ClientHello, not the version
      negotiated for the connection.  This feature is designed to
      prevent rollback attacks.  Unfortunately, many implementations
      use the negotiated version instead, and therefore checking the
      version number may lead to failure to interoperate with such
      incorrect client implementations.  Client implementations, MUST
      and Server implementations MAY, check the version number.  In
      practice, since the TLS handshake MACs prevent downgrade and no
      good attacks are known on those MACs, ambiguity is not
      considered a serious security risk.  Note that if servers
      choose to check the version number, they should randomize the
      PreMasterSecret in case of error, rather than generate an
      alert, in order to avoid variants on the Bleichenbacher attack.
      [KPR03]
```

TLS 1.2

```
Note: Attacks discovered by Bleichenbacher [BLEI] and Klima et al.
[KPR03] can be used to attack a TLS server that reveals whether a
particular message, when decrypted, is properly PKCS#1 formatted,
contains a valid PreMasterSecret structure, or has the correct
version number.

As described by Klima [KPR03], these vulnerabilities can be avoided
by treating incorrectly formatted message blocks and/or mismatched
version numbers in a manner indistinguishable from correctly
formatted RSA blocks.  In other words:

   1. Generate a string R of 46 random bytes

   2. Decrypt the message to recover the plaintext M

   3. If the PKCS#1 padding is not correct, or the length of message
      M is not exactly 48 bytes:
         pre_master_secret = ClientHello.client_version || R
      else If ClientHello.client_version <= TLS 1.0, and version
      number check is explicitly disabled:
         pre_master_secret = M
      else:
         pre_master_secret = ClientHello.client_version || M[2..47]

Note that explicitly constructing the pre_master_secret with the
ClientHello.client_version produces an invalid master_secret if the
client has sent the wrong version in the original pre_master_secret.

An alternative approach is to treat a version number mismatch as a
PKCS-1 formatting error and randomize the premaster secret
completely:

   1. Generate a string R of 48 random bytes

   2. Decrypt the message to recover the plaintext M

   3. If the PKCS#1 padding is not correct, or the length of message
      M is not exactly 48 bytes:
         pre_master_secret = R
      else If ClientHello.client_version <= TLS 1.0, and version
      number check is explicitly disabled:
         premaster secret = M
      else If M[0..1] != ClientHello.client_version:
         premaster secret = R
      else:
         premaster secret = M

Although no practical attacks against this construction are known,
Klima et al. [KPR03] describe some theoretical attacks, and therefore
the first construction described is RECOMMENDED.

In any case, a TLS server MUST NOT generate an alert if processing an
RSA-encrypted premaster secret message fails, or the version number
is not as expected.  Instead, it MUST continue the handshake with a
randomly generated premaster secret.  It may be useful to log the
real cause of failure for troubleshooting purposes; however, care
must be taken to avoid leaking the information to an attacker
(through, e.g., timing, log files, or other channels.)

The RSAES-OAEP encryption scheme defined in [PKCS1] is more secure
against the Bleichenbacher attack.  However, for maximal
compatibility with earlier versions of TLS, this specification uses
the RSAES-PKCS1-v1_5 scheme.  No variants of the Bleichenbacher
attack are known to exist provided that the above recommendations are
followed.

Implementation note: Public-key-encrypted data is represented as an
opaque vector <0..2^16-1> (see Section 4.7).  Thus, the RSA-encrypted
PreMasterSecret in a ClientKeyExchange is preceded by two length
bytes.  These bytes are redundant in the case of RSA because the
EncryptedPreMasterSecret is the only data in the ClientKeyExchange
and its length can therefore be unambiguously determined.  The SSLv3
specification was not clear about the encoding of public-key-
encrypted data, and therefore many SSLv3 implementations do not
include the length bytes -- they encode the RSA-encrypted data
directly in the ClientKeyExchange message.

This specification requires correct encoding of the
EncryptedPreMasterSecret complete with length bytes.  The resulting
PDU is incompatible with many SSLv3 implementations.  Implementors
upgrading from SSLv3 MUST modify their implementations to generate
and accept the correct encoding.  Implementors who wish to be
compatible with both SSLv3 and TLS should make their implementation's
behavior dependent on the protocol version.

Implementation note: It is now known that remote timing-based attacks
on TLS are possible, at least when the client and server are on the
same LAN.  Accordingly, implementations that use static RSA keys MUST
use RSA blinding or some other anti-timing technique, as described in
[TIMING].
```

With every new TLS version the countermeasures became more complicated

Many more attacks on poor choices in TLS 1.2 and earlier

SLOTH, FREAK, Logjam, SWEET32, Triple Handshake

Fixing bugs like TLS 1.2 and earlier

Use workarounds for known security issues

If workarounds are insufficient use more workarounds

Create optional secure modes, but keep the insecure ones

Fixing bugs like TLS 1.3

Remove insecure cryptographic constructions

TLS 1.3 Deprecations

- CBC-Modes, RC4, Triple-DES
- GCM with explicit nonces
- RSA Encryption, PKCS #1 1.5
- MD5, SHA1
- Diffie Hellman with custom or small parameters
- Obscure, custom and insecure Elliptic Curves

Formal Verification

Researchers have started to formally analyze TLS in recent years

Many vulnerabilities were found during protocol analysis

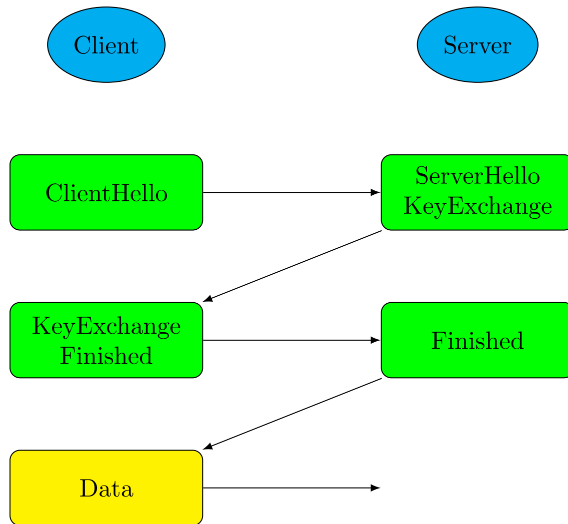
These analyses have contributed to and guided the design of TLS 1.3

Security is nice, but there's something else we care about:

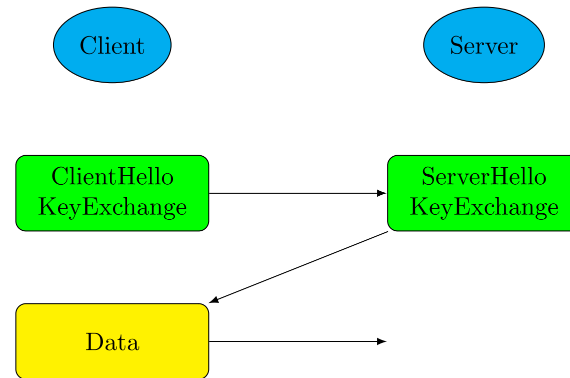
Speed!

TLS Fresh Handshake

TLS 1.2



TLS 1.3



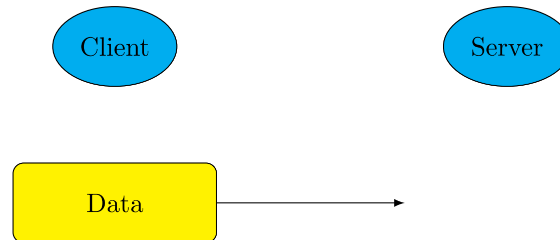
TLS 1.3 handshake removes one round trip from fresh handshakes

Handshake improves forward secrecy on session resumption and protects more data

TLS 1.3 has a faster and more secure handshake

Watch 33C3 talk

TLS 1.3 Zero Round Trip (0-RTT)



If we previously connected we can use a pre-shared Key (PSK) to send data without any round trip

More speed!

But 0-RTT is not for free

Replay attacks

0-RTT should only be used where it's safe

Example HTTPS

GET Request: Idempotent

POST Request: Not Idempotent

In theory HTTP GET requests are idempotent and safe for 0-RTT

Do web developers know what idempotent means?

0-RTT does not have strong forward secrecy

Many speculate that future TLS 1.3 attacks will exploit 0-RTT

0-RTT is optional

If it turns out being too bad we can disable it

Deployment

It's not enough to design a faster, more secure TLS protocol, you also have to deploy it

On the Internet

The real Internet

The version number

This may sound trivial, but one other new thing that TLS 1.3 brings is a new version number

```
▾ Transport Layer Security
  ▾ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301) ←
      Length: 311
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 307
        Version: TLS 1.2 (0x0303) ←
        Random: a9a0303cfc3067c3915f5e3471d03975a62ab22664c18ed9…
        Session ID Length: 32
        Session ID: b229cd21e91d1d37d3fe0c126383e03a227d83500553f905…
        Cipher Suites Length: 62
      ▸ Cipher Suites (31 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 172
      ▸ Extension: server_name (len=19)
      ▸ Extension: ec_point_formats (len=4)
      ▸ Extension: supported_groups (len=12)
      ▸ Extension: session_ticket (len=0)
      ▸ Extension: encrypt_then_mac (len=0)
      ▸ Extension: extended_master_secret (len=0)
      ▸ Extension: signature_algorithms (len=48)
      ▾ Extension: supported_versions (len=9)
          Type: supported_versions (43)
          Length: 9
          Supported Versions length: 8
          Supported Version: TLS 1.3 (0x0304) ←
          Supported Version: TLS 1.2 (0x0303)
          Supported Version: TLS 1.1 (0x0302)
          Supported Version: TLS 1.0 (0x0301)
      ▸ Extension: psk_key_exchange_modes (len=2)
      ▸ Extension: key_share (len=38)
```

```
▾ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301) ←
    Length: 311
  ▾ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 307
      Version: TLS 1.2 (0x0303) ←
```

```
▾ Extension: supported_versions (len=9)
    Type: supported_versions (43)
    Length: 9
    Supported Versions length: 8
    Supported Version: TLS 1.3 (0x0304)
    Supported Version: TLS 1.2 (0x0303)
    Supported Version: TLS 1.1 (0x0302)
    Supported Version: TLS 1.0 (0x0301)
```

TLS 1.0 came after SSL 3

SSL 3	03 00
TLS 1.0	03 01
TLS 1.1	03 02
TLS 1.2	03 03
TLS 1.3	It's complicated

TLS record layer

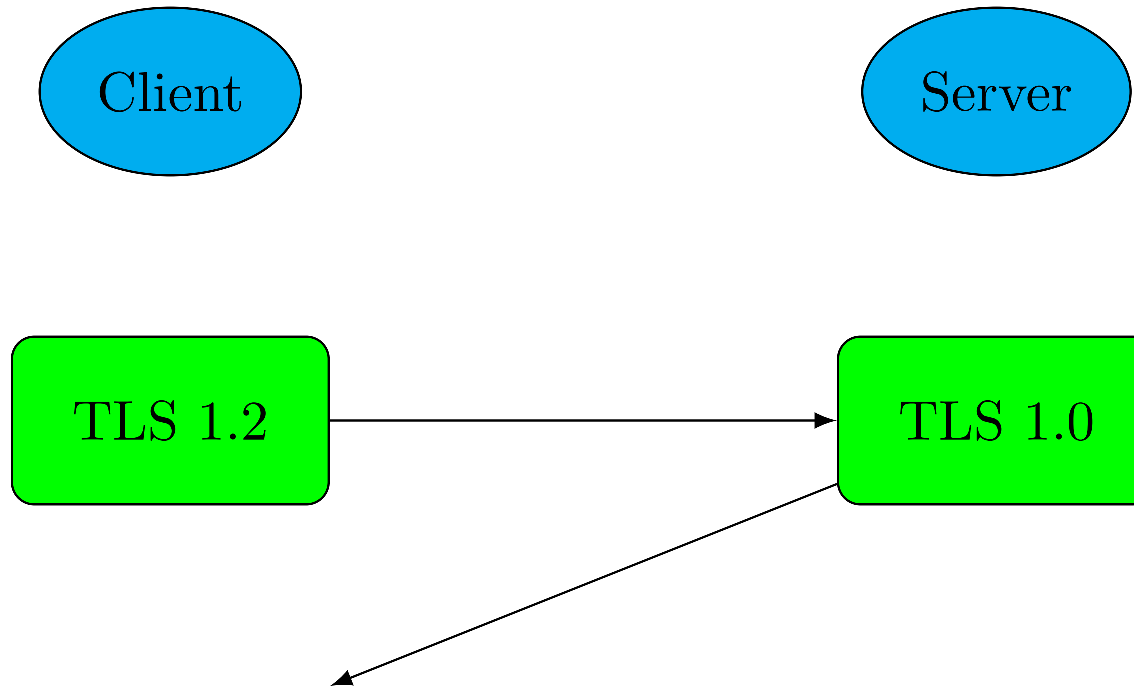
A protocol inside the protocol which has its own meaningless version number

We can't update the whole Internet at once

When we deploy a new version of TLS we need to still support old versions

Let's assume we have a client supporting TLS 1.2 and a server supporting TLS 1.0

TLS Version Negotiation



This is very simple

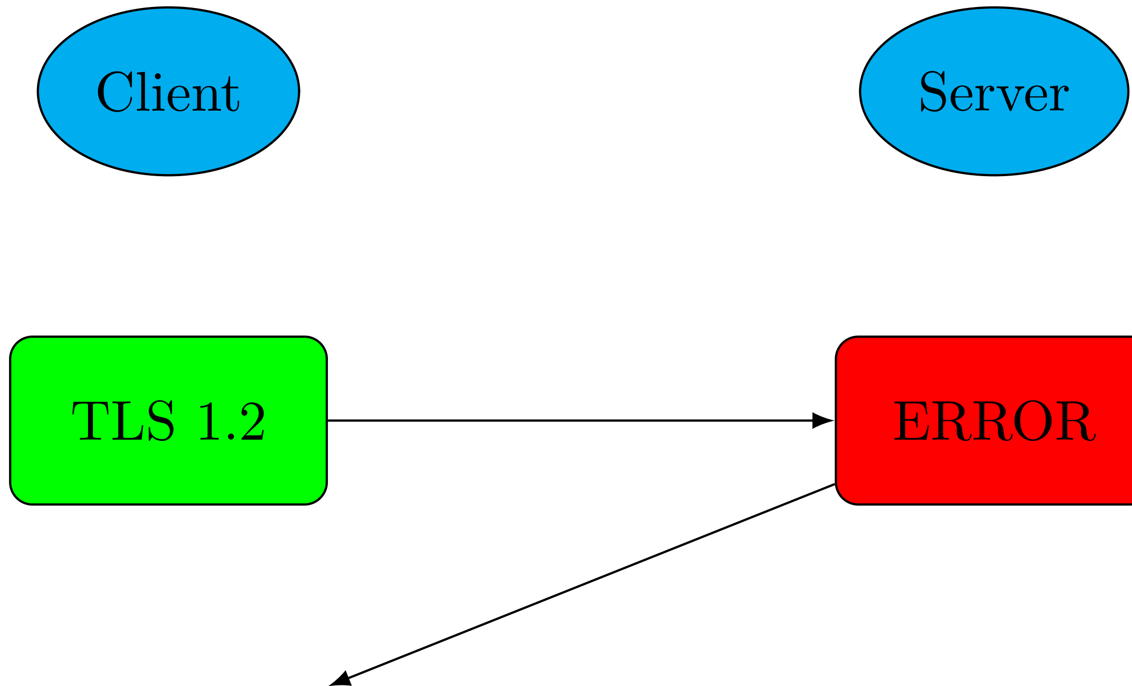
```
if (client_max_version < server_max_version) {
        connection_version = client_max_version;
} else {
        connection_version = server_max_version;
}
```

There's no way anyone could possibly get that wrong

Okay, we were talking about the real Internet

There are Enterprise Products

TLS Version Negotiation Enterprise Edition

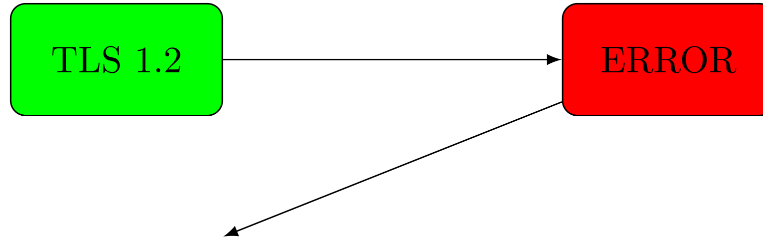


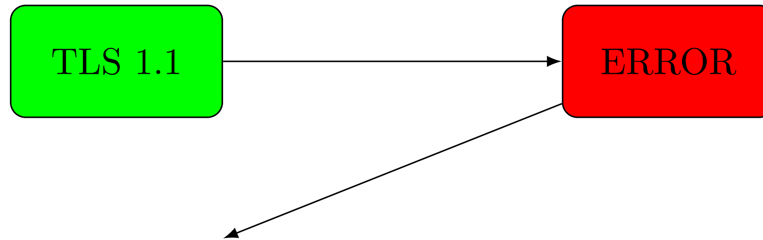
Version intolerance

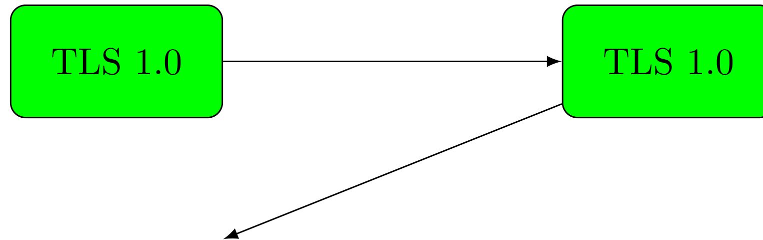
Version intolerance shows up every single time a new TLS version is introduced

What did browsers do?

Client Server







Remember POODLE (2014)?

Guanaco, Wikimedia Commons, CC0

POODLE was a Padding Oracle in SSL 3

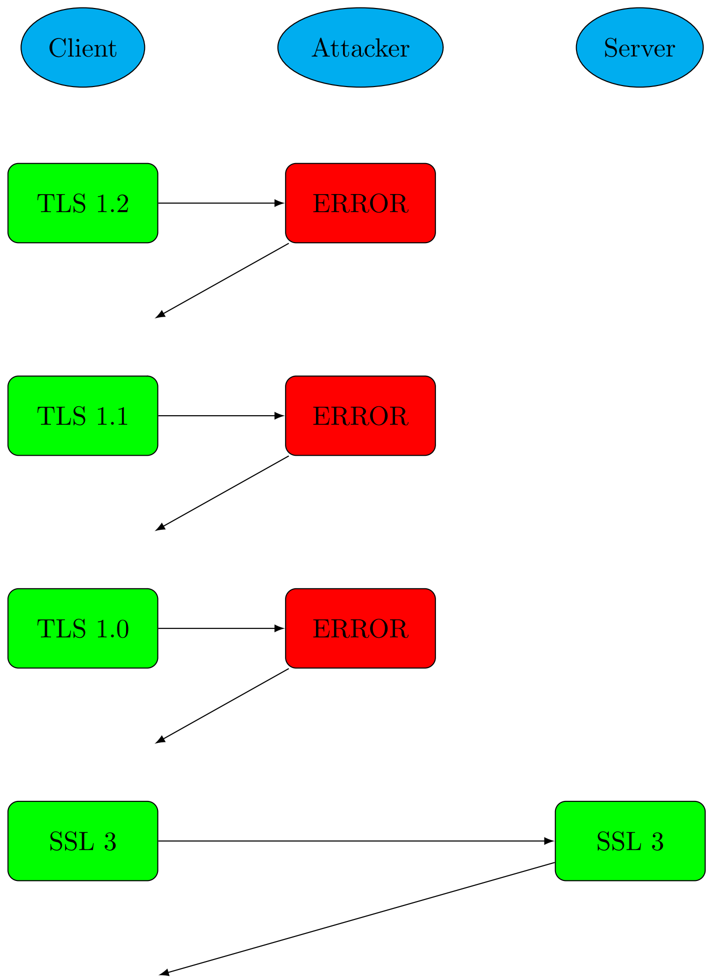
Who used SSL 3 in 2014?

It was deprecated for 16 years

Nokia Phones with Windows Mobile (built 2011)

Image: Petar Milošević, CC by 4.0

But most browsers and most servers used at least TLS 1.0


Client
Attacker
Server
TLS 1.2
ERROR
TLS 1.1
ERROR
TLS 1.0
ERROR
SSL 3
SSL 3

So how to fix these insecure downgrades?

Let's add another workaround

SCSV: Introduce a mechanism that lets well-behaving servers detect when clients did a downgrade

At some point Enterprise servers had fixed version intolerance and browsers stopped these downgrades

Have I said they fixed version intolerance?

Of course not!

They fixed version intolerance for TLS 1.2, not for 1.3

New version negotiation in TLS 1.3

Old version field (legacy_version) stays at TLS 1.2

New extension (supported_versions) signals support for future TLS versions.

Does that mean we will have the same problem again with TLS 1.4?

GREASE

(Generate Random Extensions And Sustain Extensibility)

Servers should ignore unknown versions in *supported_versions*

Let's train servers to actually do that

GREASE values are reserved, bogus TLS versions that will never be used for real TLS versions

Clients can randomly send GREASE values in the TLS handshake

Implementors with broken version negotiation will hopefully notice that before shipping their product

Okay, so with the new version negotiation and GREASE we can ship TLS 1.3?

The Middlebox disaster

In summer 2017 TLS 1.3 was almost finished and ready to go, but it took another year until it was actually finalized

Browser vendors noticed a high number of connection failures when trying to deploy TLS 1.3

The reason: Devices analyzing traffic and trying to be smart

"Let's look at this TLS package. I've never seen something like that... let's better discard it."

These were largely passive middleboxes that should just pass traffic through

How to fix

Browser vendors proposed some changes to TLS 1.3 that made it look more like TLS 1.2

ChangeCipherSpec in TLS 1.2

The ChangeCipherSpec (CCS) message signals the change from unencrypted to encrypted content

Let's send a bogus CCS early in the handshake and hope this will confuse "smart" middleboxes into thinking that everything afterwards is encrypted and shouldn't be touched

MiNe, Wikimedia Commons, CC by 2.0

NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA

Dual EC DRBG

The NSA created a random number generator with a backdoor and convinced NIST to standardize it

With a generous offer of 10 Million Dollar they convinced RSA security to use Dual EC DRBG

Extended Random

There exists a draft for a TLS extension that adds some extra random numbers to the TLS handshake

Why?

In 2014 researchers figured out that Extended Random makes the Dual EC DRBG backdoor much more effective

Checkoway et al, 2014

Coincidentally RSA's BSAFE library also contained support for Extended Random - but it was switched off by default, so everyone thought it's no big deal

Canon Pixma printers had a local HTTPS server, implemented with RSA BSAFE and Extended Random switched on

Extended Random was only a draft, so it had no official Extension number, RSA just used one of the next available numbers

This number collided with one of the new extensions in TLS 1.3, resulting in connection failures of TLS 1.3 supporting browsers and these Canon printers

There were many more TLS deployment issues and they continue

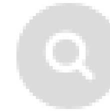
What about future TLS versions?

We have GREASE, which helps a bit

There's even a proposal to regularly roll out temporary TLS versions every few months

My prediction: These deployment problems are going to get worse

LOCKHEED MARTIN
CYLANCE
ERAGING
AR CIAL
INT IGENCE
VERAC



Security

Detecting Encrypted Malware Traffic (Without Decryption)

In the future we may have AI-supported TLS change intolerance, and that may be much harder to fix

Speaking of Enterprise environments

TLS removed the RSA encryption handshake very early

It doesn't have Forward Secrecy and it suffers from Bleichenbacher attacks

An E-Mail to the TLS Working Group from the Banking Industry

[tls] Industry Concerns about TLS 1.3

I recently learned of a proposed change that would affect many of my organization's member institutions: the deprecation of RSA key exchange.

Deprecation of the RSA key exchange in TLS 1.3 will cause significant problems for financial institutions, almost all of whom are running TLS internally and have significant, security-critical investments in out-of-band TLS decryption.

BITS/TLS list

My view concerning your request: no.

Rationale: We're trying to build a more secure internet.

Kenny Paterson

You're a bit late to the party. We're metaphorically speaking at the stage of emptying the ash trays and hunting for the not quite empty beer cans.

More exactly, we are at draft 15 and RSA key transport disappeared from the spec about a dozen drafts ago. I know the banking industry is usually a bit slow off the mark, but this takes the biscuit.

Kenny Paterson

This led to several proposals to add a "visibility" mode to TLS 1.3, which were all rejected by the IETF TLS working group

The prevailing opinion in the TLS working group was that the goal of monitoring traffic content is fundamentally at odds with the goal of TLS

So the industry went to ETSI, the European standardization organization

They published Enterprise TLS (ETLS)

The IETF wasn't happy about the abuse of the name TLS

TLS 1.3

What's left?

Many attacks aren't against the cryptography of the protocol itself

Despite all the protocol issues the biggest TLS security flaw is probably that people aren't using it

SSL Stripping

We should use HTTPS by default

We also need to enforce it with HSTS (HTTP Strict Transport Security)

E-Mail

Server-to-Server STARTTLS is usually optional and unauthenticated

MTA-STS

Publishing a TLS policy for SMTP via HTTPS

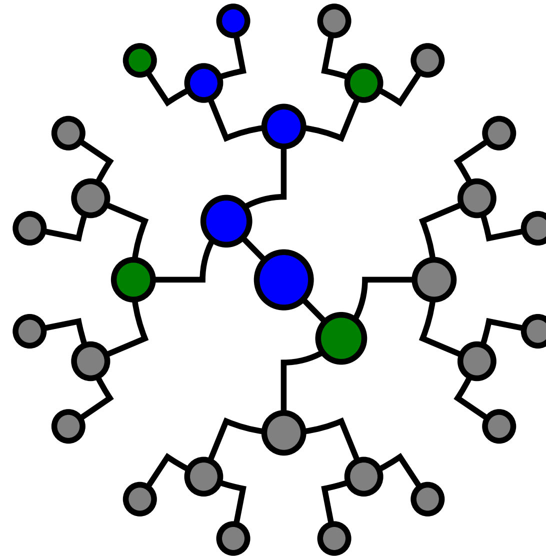
Certificates

Popular Hacker Opinion

"The whole Certificate Authority system is broken"

Things have improved considerably, yet not everyone wants to recognize that

Certificates Transparency



CAs that repeatedly violate rules get distrusted

No CA is too big to fail

If you don't believe it ask Symantec

Future attacks

Compression attacks

CRIME, BREACH, TIME, HEIST

There's yet no satisfying fix for compression attacks

Domain Validation

Certificates are issued based on checks of domain ownership, yet these checks happen over an unencrypted Internet

Getting Certificates via BGP Hijacking

This is definitely possible, but hasn't been seen in the real world yet

No, Extended Validation does not help

Summary

TLS 1.3 deprecates many insecure constructions

TLS 1.3 is faster

Deploying new things on the Internet is a mess

Encrypt your connections!