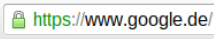
Transport Layer Security

Christian Grothoff

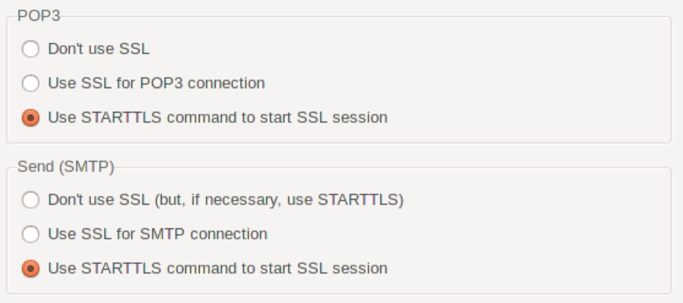
Berner Fachhochschule

14.05.2018

TLS is everywhere



https://www.google.de/



POP3

○ Don't use SSL

○ Use SSL for POP3 connection

● Use STARTTLS command to start SSL session

Send (SMTP)

○ Don't use SSL (but, if necessary, use STARTTLS)

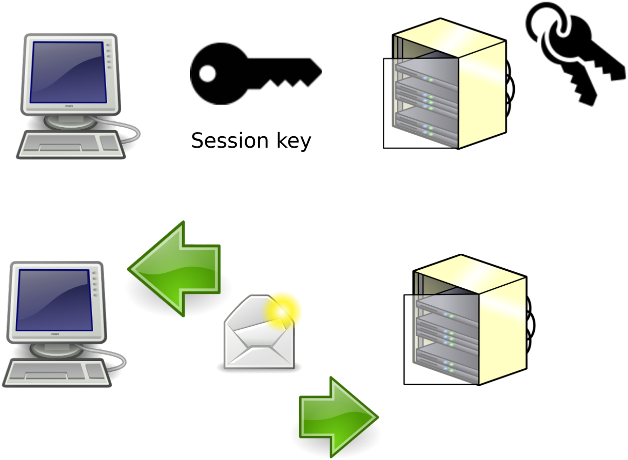
○ Use SSL for SMTP connection

● Use STARTTLS command to start SSL session

TLS versions

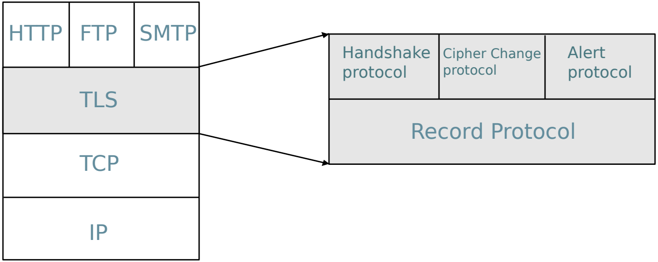
1994	SSL v2
1995	SSL v3
1999	TLS v1.0
2006	TLS v1.1
2008	TLS v1.2
2018	TLS v1.3

TLS overview



Session key

TLS Protocol Stack



Maximum record payload is 16kB.

Why Records?

Why not encrypt data in constant stream as we write to TCP?

Why Records?

Why not encrypt data in constant stream as we write to TCP?

- Where would we put the MAC?
- If at the end, we get no integrity until all data is processed!
- Most applications process/display data incrementally!

Records allow us to:

- Break stream into series of records
- Each record carries a MAC
- Receiver can act on record as it arrives!

Attacks on records

Attacker could re-order or replay records!

Attacks on records

Attacker could re-order or replay records!

- Put sequence number into MAC.

Attacker could truncate TCP stream!

Attacks on records

Attacker could re-order or replay records!

- Put sequence number into MAC.

Attacker could truncate TCP stream!

- Use record types.
- Have special record type to indicate end of stream.

Protocol and Software

- TLS protocol is way too complex
- Many implementations in use
- Vulnerabilities in protocol design and implementations

Attacks on TLS and implementations

2011	BEAST
2012	CRIME
2013	BREACH, Lucky Thirteen
2014	Heartbleed, BERserk, POODLE
2015	FREAK, Logjam, MACE, RSA-CRT, Mar Mitzvah
2016	SLOTH, DROWN
2017	ROBOT
2018	CVE-2018-0488, CVE-2018-1000151

No news for cryptographers

Rivest: DSA weakness (1992)	Playstation 3 broken (2010), Mining Ps and Qs (2012)
Dobbertin: MD5 weak (1996), Wang: MD5 collission, SHA1 weak (2004/2005)	MD5 CA attack (2008), Flame (2012), SLOTH (2016)
Lenstra: RSA-CRT weakness (1996)	RSA-CRT attack (2015)
Bleichenbacher: Million Message attack (1998)	DROWN (2016)
Biehl: Fault attacks on ECC (2000)	Invalid curve attacks (2015)
Fluhrer/McGrew: RC4 biases (2000)	RC4 TLS attacks (2013-2016), Bar Mitzvah (2016)
Vaudenay: Padding Oracle (2002)	Lucky Thirteen (2013)
Bard: Implicit IV vuln (2004)	BEAST (2011)
Bleichenbacher: Signature forgery (2004)	BERserk (2014), ROBOT (2017)

Security is hard

"In order to defend against this attack, implementations MUST ensure that record processing time is essentially the same whether or not the padding is correct. [...] **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." (TLS 1.2, RFC 5246, 2008)

Modes

- Many SSL/TLS modes built “authenticted encryption” by combining authentication and encryption
- Many attacks would have been avoided by using primitive that implements both in one, such as AES-GCM or ChaCha20-Poly1305
- Anything using ECB, CBC, CFB, OFB, CTR is likely broken
- GCM needs a nonce $\Rightarrow$ another major failure mode

Primitives

SSL started with many primitives we now know consider insecure:

- RC4
- SHA1
- MD5
- 1024 bit DH with fixed parameters
- “export” ciphers

Deprecation

Evolution is slow as deprecation *blocks* connections:

- What percentage of clients is it OK to block?
- What percentage of servers is it OK to block?
- Many middleboxes *require* insecure versions!
- If old versions are supported, downgrade attacks are possible!

Origins of Complexity

1. We have a version negotiation mechanism
2. Servers have broken TLS implementations on version negotiation
3. Browsers implement workaround ("protocol dance")
4. Workaround introduces security issue (downgrade)
5. Workaround for security issue introduced by workaround gets standardized.

TLS Usability

To use TLS securely, you need at least:

- Secure implementation
- Secure protocol configuration (cipher suite)
- X.509 certificate(s)
- Tell client you support TLS: `Strict-Transport-Security` header
- Secure certificate chains against bad CA:
    - HTTP Public Key Pinning (HPKP)
    - Certificate Patrol
    - Certificate Transparency (CT)

Security by Default?

Security by Default?

You wish:

```
SSLProtocol -SSLv2 -SSLv3 -TLSv1 TLSv1.1 +TLSv1.2
SSLHonorCipherOrder on
SSLCompression off
SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:\
 ECDHE-RSA-AES256-GCM-SHA384:ECDH-RSA-AES256-\
 GCM-SHA384:ECDH-ECDSA-AES256-GCM-SHA384:ECDH\
 -RSA-RC4-SHA:RC4-SHA:TLSv1:!AES128:!3DES:!CA\
 MELLIA:!SSLv2:HIGH:MEDIUM:!MD5:!LOW:!EXP:!NUL\
 L:!aNULL
```

It is 2018 and our TLS configurations **still** look like this!

The Future

TLS 1.3

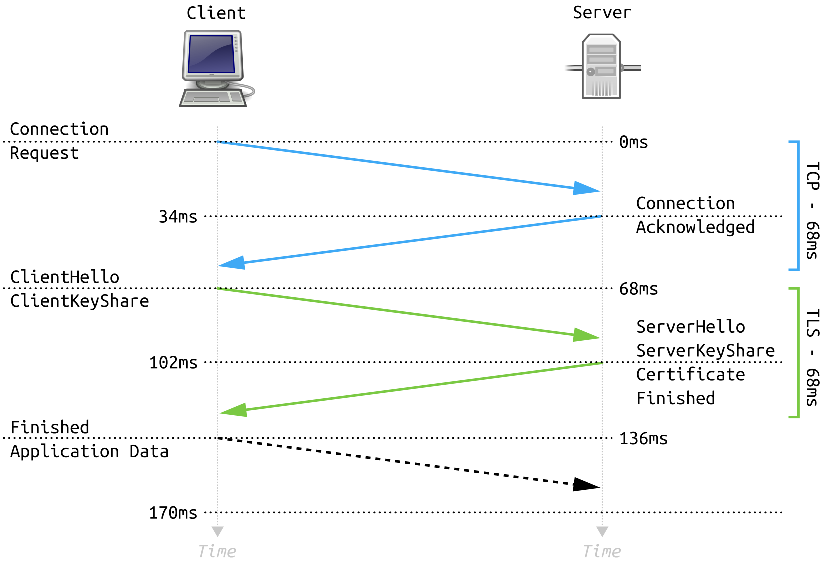
TLS 1.3

- Attempt to break away from attack-patch-attack-patch design cycle
- Research community more involved

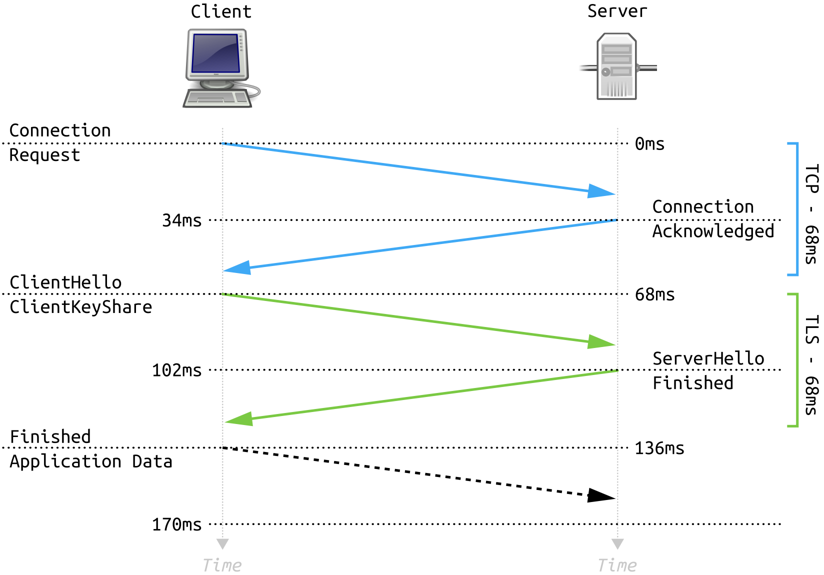
⇒ Formal security proofs (value?)

- Protocol differs significantly from previous versions
- Still lots of extensions, lots of modes
- Client still begins negotiation with ClientHello

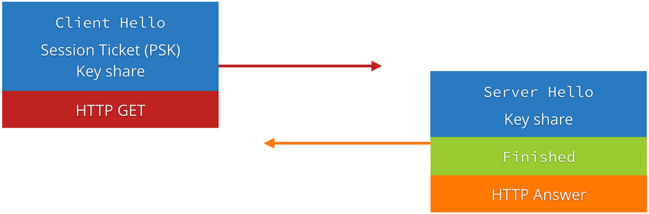
TLS 1.3: Full Handshake



TLS 1.3: Abbreviated Handshake



TLS 1.3: 0.5 RTT Handshake



Client Hello

Session Ticket (PSK)
Key share

HTTP GET

Server Hello

Key share

Finished

HTTP Answer

TLS 1.3

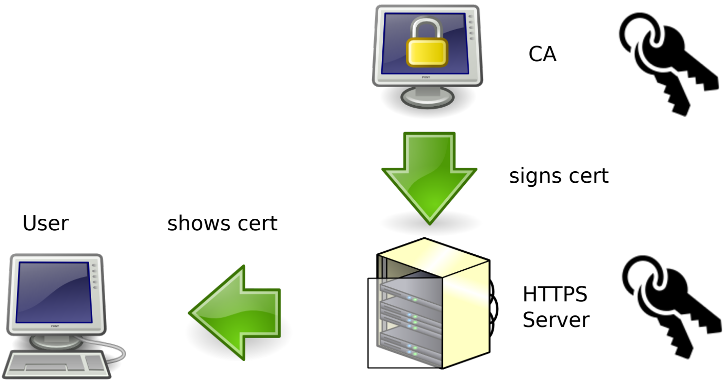
- Also deprecates many insecure ciphers
- Again has downgrade attack problem
- Still uses X.509 certificates

Break

X.509

- TLS servers (and sometimes clients) are identified by public key
- Public keys are *certified* by certificate authorities
- X.509 certificates are the format used for certificates
- Any certificate authority can certify keys for any domain

X.509 overview



Certificate validation is hard

- BERserk was a catastrophic failure in the certificate validation of the NSS library (used by Firefox / Chrome)
- Most TLS libraries had a chain validation issue at some point

Certificate validation is hard

- BERserk was a catastrophic failure in the certificate validation of the NSS library (used by Firefox / Chrome)
- Most TLS libraries had a chain validation issue at some point

Let's assume TLS is correct and correctly implemented...

CA issues all the time

- June 2013: ANSSI issues certs for Google
- March 2014: India CCA intermediate compromised and issued certs for Yahoo and Google
- Feb 2015: Superfish / Privdog / Komodia breaking certificate authentication
- March 2015: Comodo cert for live.fi through hostmaster@live.fi
- March 2015: Same thing for xs4all
- March 2015: Google found bad certs issued by MCS Holdings / CNNICa
- April 2015: Google and Mozilla remove CNNIC

Too many CAs

- There are hundreds of browser-accepted CAs and an unknown number of subordinate CAs
- Each of them can break TLS security
- It does not matter how good your CA is — the only thing that matters is the worst CA of them all

CNNIC

- CNNIC issued intermediate certificate to Egyptian company MCS Holdings
- MCS used it in a Man-in-the-Middle-TLS-Proxy in violation of policy
- Google and Mozilla kick CNNIC out

Domain Validation via E-Mail

- Domain Validation: CA sends mail to defined aliases (admin, administrator, webmaster, hostmaster, postmaster, see Baseline Requirements)
- If you offer E-Mail you **must** make sure that nobody can register such an address
- One can argue if this is sane system, but it is documented (Baseline Requirements)
- live.fi / xs4all.nl issues were their fault

Revocation is broken

- Two revocation mechanisms:
  - Certificate Revocation Lists (CRL), and
  - Online Certificate Status Protocol (OCSP)
- Browsers used insecure soft-fail mode in the past
- Chrome and Firefox distribute their own blocklists, but they don't scale
- OCSP stapling could help, but needs a mechanism to indicate its use (muststaple draft)

Man in the Middle Proxies

- Superfish: Created a TLS Man in the Middle Proxy, private key was static and part of the Software (Komodia)
- Privdog: Just disabled TLS verification completely (Privdog is founded by the CEO of Comodo)
- Several Antiviruses do the same. Not fully broken, but all decrease the security of TLS
- This is not directly a problem of CAs or TLS

Alternatives: DNSSEC/DANE

DANE will not provide you any security today

It is very uncertain if it will ever do that

DNSSEC — too many pieces

For DNSSEC to work you need:

- A signed root
- A signed Top Level Domain
- A domain broker that supports DNSSEC
- A DNS operator that supports DNSSEC
- A client that verifies DNSSEC

Only if you have all five you have security.

Working DNSSEC deployment is near zero

- DNSSEC propaganda: "xx % of all TLDs are signed", "there are already XX.XXX signed domains"
- Completely irrelevant statements
- Cryptographic signatures are not worth anything if nobody is checking them
- Once you enable checking, you find out signatures are invalid
- Even if they are valid today, that may not be true after key rollover
- Same issue as with TLS: How many users are you willing to burn?

⇒ Client deployment of DNSSEC is very close to zero

DNSSEC client

- So how exactly does a client verify DNSSEC signatures? (Most common today: Not at all)
- DNSSEC verification happens in the DNS resolver — but clients usually do not have full DNS resovlers

DNSSEC client

- Should we trust our providers? (No!)
- Should operating systems ship DNS resolvers?
- Should applications ship their own DNS resolvers?
- Not clear how DNSSEC should be deployed on clients!

So what is DANE?

- Idea of DANE: If we already have a secure DNS through DNSSEC we can add certificate information to the DNS
- The problem: We do not have working DNSSEC
- Building something on top of something that does not work is pointless
- Also, to secure DNS, IETF proposed putting DNS-over-TLS

Does anyone see a chicken-and-egg problem here?

TLS and HTTP: HTTP Public Key Pinning (HPKP)

- Webpage sends a header with hashes of public keys for the browser to pin
- Browser stores these hashes
- Always needs at least two keys - because you need to be able to change your certificates in the future
- Adds a “Trust on First Use” (ToFU) protection

HTTP Public Key Pinning (HPKP)

- HPKP header:
- max-age=31536000;pin-sha256="HD3EpAqgxJWKGiSuuXPyipmL33IwYIwhLUgF1gKYOuc= sha256="dwUkkREEnv6pEtNJoRzIBHJm3IlUvPhgy0mdYFOM6V8= includeSubDomains; report-uri="/hpkp.php"
- Browser pins the two hashes for [max-age] seconds
- report-uri is unimplemented today

HPKP deployment

- HPKP is supported by Chrome/Chromium and Firefox
- Needed for deployment: Software change in browsers and configuration change on servers
- Large webpages have pre-loaded pins in the browsers

HPKP: Only for HTTPS

- One big drawback: It is only for the Web
- As HPKP is implemented via HTTP headers it does not work on other protocols
- There is a proposal called TACK to do something similar on the TLS layer

HPKP Warning

HPKP improves confidentiality, but can be dangerous to availability:

- If you loose your keys you may lock out your visitors!
- Needs careful planning of key management.

Certificate Transparency

- Public logs with all certs in them
- Certificate can contain log proof confirming that it has been added to a log
- When a browser sees a certificate that is not in the log it can raise alarm

Certificate Transparency

- Certificate Transparency runs in soft-fail mode, it cannot prevent misuse
- But it makes it hard to use malicious certificates without being noticed

Free certificates

- StartSSL, free for noncommercial use, 1 year validity
- WoSign, 2 years validity
- Let's encrypt (EFF, Mozilla), 3 months validity, auto-renewal

HTTP Strict Transport Security (HSTS)

- HSTS tells the browser to mark a page as HTTPS only for a defined timeframe
- Further prevents stripping attacks
- You can even pre-load your webpage as HTTPS only into Chrome and Firefox

HSTS attack through NTP

- HSTS protects a page for a defined timeframe
- System time is considered trustworthy, but it is not!
- Delorean-Attack circumvents HSTS with NTP
- NTP provides no security (solutions: tlsdate, openntpd)

Acknowledgements

- Based on materials and inspiration taken from talks by Hanno Böck

Further reading I

- How broken is TLS? `http://media.ccc.de/browse/conferences/eh2014/EH2014_-_5744_-_de_-_shack-seminarraum_-_201404201530_-_wie_kaputt_ist_tls_-_hanno.html`
- Google on CNNIC `http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html`
- Mozilla on CNNIC `https://blog.mozilla.org/security/2015/04/02/distrusting-new-cnnic-certificates/`
- live.fi bad cert `https://technet.microsoft.com/en-us/library/security/3046310`
- xs4all bad cert `https://raymii.org/s/blog/How_I_got_a_valid_SSL_certificate_for_my_ISPs_main_website.html`

Further reading II

- OCSP muststaple https://tools.ietf.org/html/draft-hallambaker-muststaple-00
- Superfish https://noncombatant.org/2015/02/21/superfish-round-up/
- Privdog https://blog.hboeck.de/archives/865-Software-Privdog-worse-than-Superfish.html
- Why not DNS records (Ryan Sleevi) https://lists.w3.org/Archives/Public/public-webappsec/2014Dec/0264.html
- DNSSEC is dead (Alex Stamos) http://www.slideshare.net/astamos/appsec-is-eating-security
- Against DNSSEC (Thomas Ptacek) http://sockpuppet.org/blog/2015/01/15/against-dnssec/
- HPKP https://developer.mozilla.org/en-US/docs/Web/Security/Public_Key_Pinning

Further reading III

- HPKP draft https://tools.ietf.org/html/draft-ietf-websec-key-pinning-21
- HPKP script for spki hashes https://github.com/hannob/hpkp
- Certificate Transparency http://www.certificate-transparency.org/
- StartSSL https://www.startssl.com/
- Wosign https://wosign.com/
- Let's encrypt https://letsencrypt.org/
- POODLE bites again https://www.imperialviolet.org/2014/12/08/poodleagain.html
- TLS 1.2 / RFC 5246 https://www.ietf.org/rfc/rfc5246.txt
- Encrypt-then-MAC / RFC 7366 https://tools.ietf.org/html/rfc7366

Further reading IV

- RC4 attacks 2013 `http://www.isg.rhul.ac.uk/tls/`
- RC4 attacks 2015 IMAP / HTTP Basic Auth `http://www.isg.rhul.ac.uk/tls/RC4mustdie.html`
- RC4 Bar Mitzvah attack `http://www.crypto.com/papers/others/rc4_ksaproc.pdf`
- POODLE `https://www.openssl.org/~bodo/ssl-poodle.pdf`
- Dancing protocols, POODLEs and other tales from TLS `https://blog.hboeck.de/archives/858-Dancing-protocols,-POODLEs-and-other-tales-from-TLS.html`
- BERserk `http://www.intelsecurity.com/advanced-threat-research/berserk.html`
- BERserk PoC `https://github.com/FiloSottile/BERserk`

Further reading V

- Bleichenbacher Signature Forgery 2006 https://www.ietf.org/mail-archive/web/openpgp/current/msg00999.html
- miTLS - formally verified `http://www.mitls.org/`
- ocaml-tls `https://github.com/mirleft/ocaml-tls`
- Quote on gmail TLS performance `https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html`
- SSL Strip `http://www.thoughtcrime.org/software/sslstrip/`
- HSTS Preload `https://hstspreload.appspot.com/`
- Bypassing HTTP Strict Transport Security `https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-w pdf`

Further reading VI

- Delorean NTP MitM
  `https://github.com/PentesterES/Delorean`
- Ring Learning With Errors / post-quantum key exchange
  `http://www.douglas.stebila.ca/research/papers/bcns15`
- SPHINCS / post quantum signatures
  `http://sphincs.cr.yp.to/`
- Qualys SSL Labs Test
  `https://www.ssllabs.com/ssltest/`