Some Lessons from Deploying Communications Security at Scale

Eric Rescorla

Mozilla

ekr@rtfm.com

Our Problem Statement

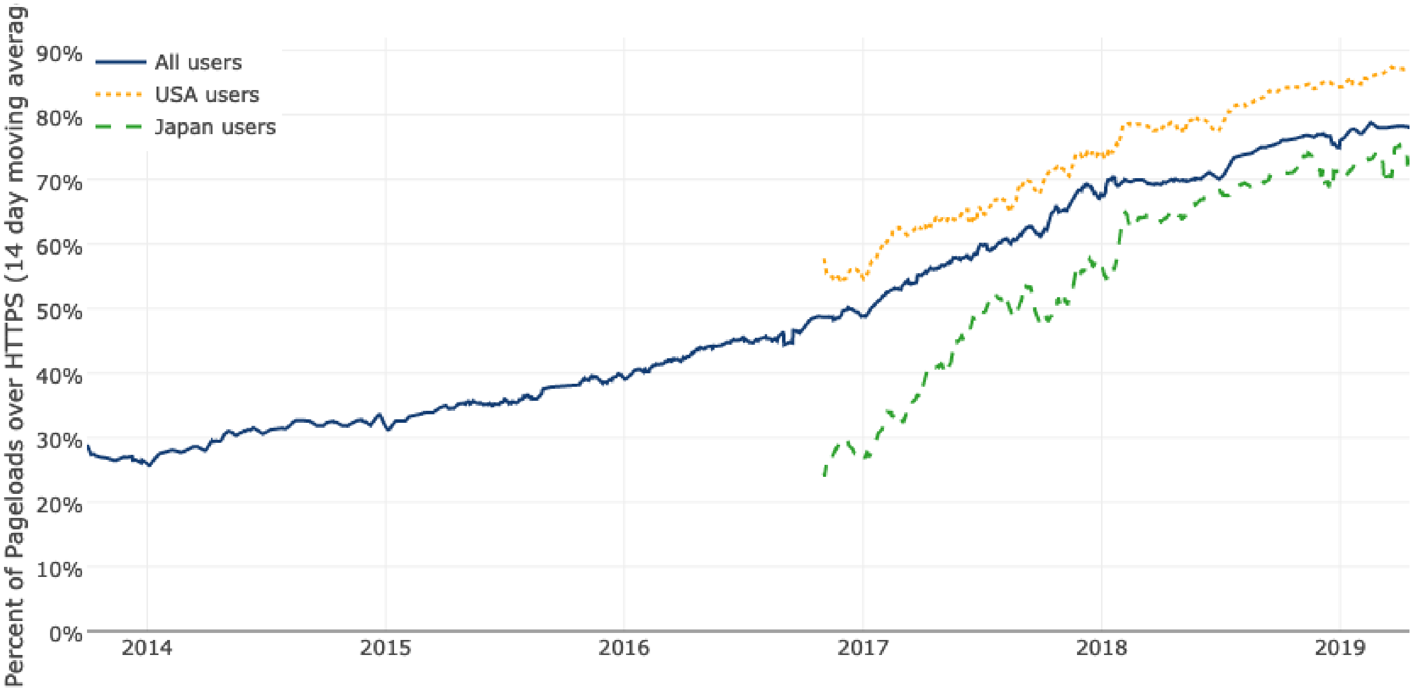
Individuals security and privacy on the internet are fundamental and must not be treated as optional.
— *Mozilla Manifesto, Principle #4*

[W]e assume that the attacker has nearly complete control of the communications channel over which the end-systems communicate. This means that the attacker can read any PDU (Protocol Data Unit) on the network and undetectably remove, change, or inject forged packets onto the wire.
— *RFC 3552*

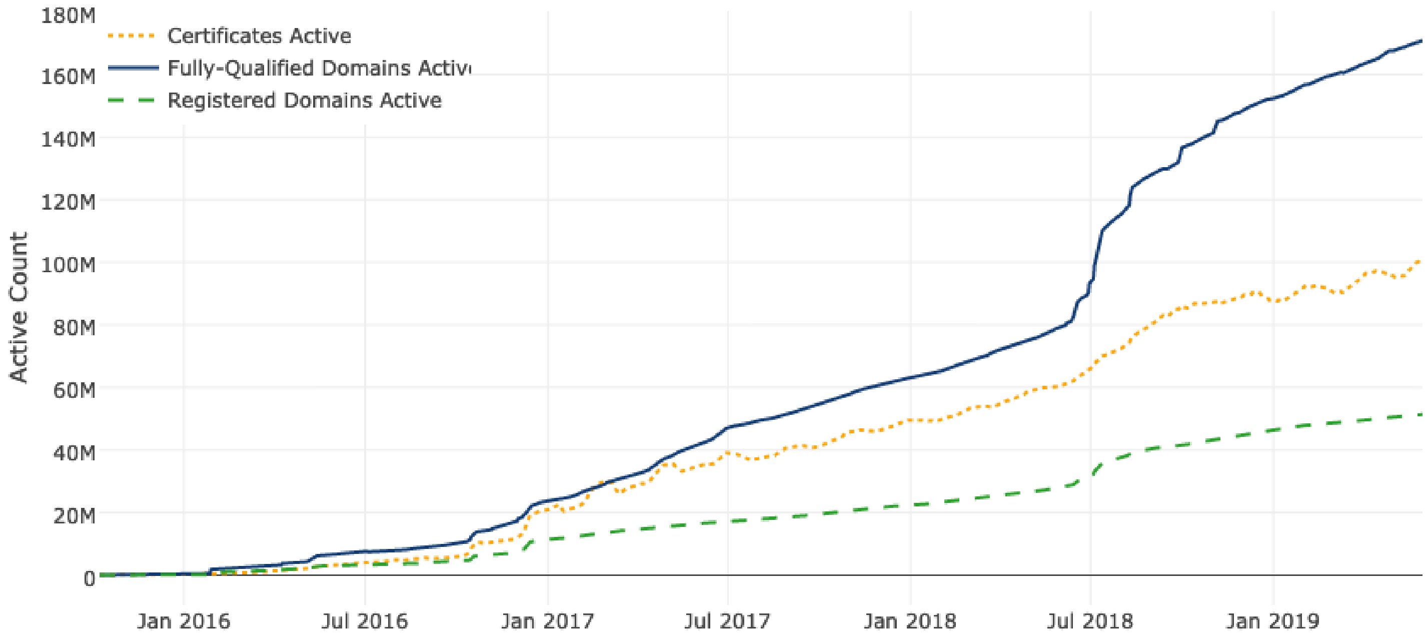
Historical Situation

- Good news
  - Cryptography offers a way out of this box
  - We have solutions for endpoint authentication, confidentiality, message integrity, etc.
- Bad news
  - Early Internet built almost entirely without cryptography
  - Why? Patents, computational cost, export controls, missing authentication infrastructure
- Need to somehow retrofit security onto this system
  - Whoever touched things last gets blamed

HTTPS Deployment

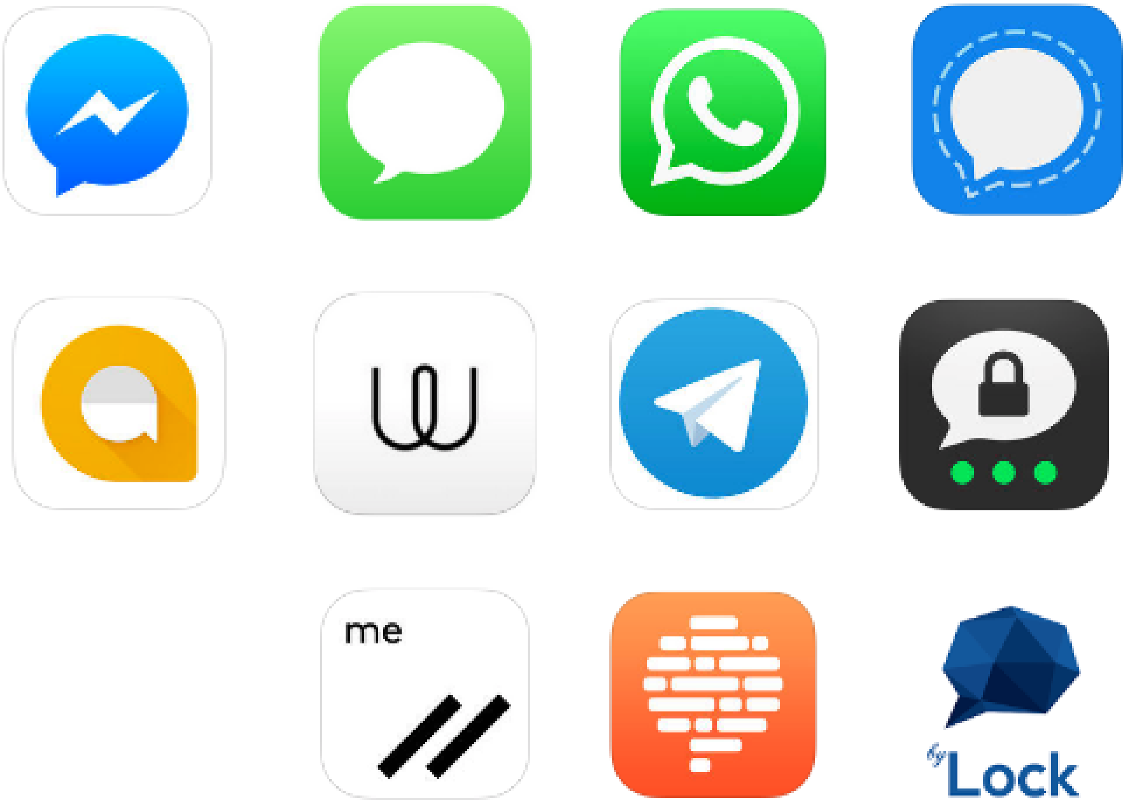


WebPKI



Certificates Active
Fully-Qualified Domains Activ
Registered Domains Active
Active Count
180M
160M
140M
120M
100M
80M
60M
40M
20M
0
Jan 2016
Jul 2016
Jan 2017
Jul 2017
Jan 2018
Jul 2018
Jan 2019

Messaging Security



What is Transport Layer Security?

- Probably the Internet's most important security protocol
- Designed over 20 years ago by Netscape for Web transactions
  - Back then, called Secure Sockets Layer
- But used for just about everything you can think of
  - HTTP
  - SSL-VPNs
  - E-mail
  - Voice/video
  - IoT
- Maintained by the Internet Engineering Task Force*
- Really showing its age as of 2015

*`https://www.ietf.org/`, `https://tlswg.org/`

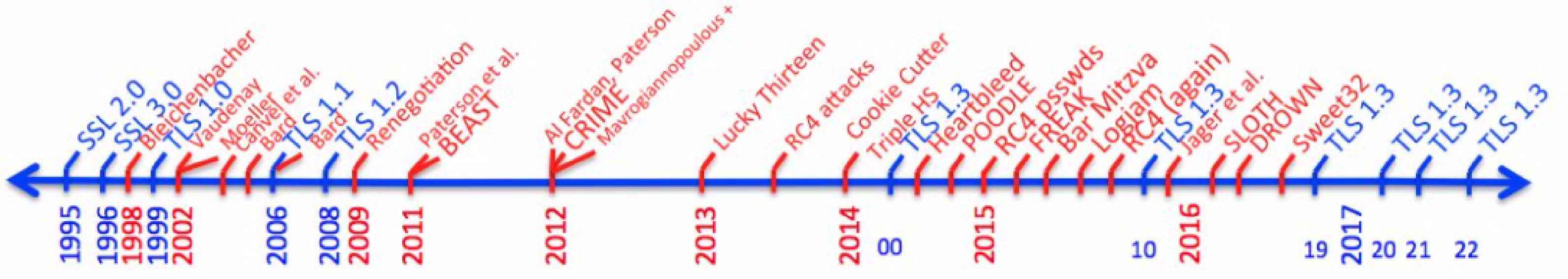
TLS 1.2 Attacks*

NETSCAPE

SSL 2.0 (1995) → SSL 3.0 (1996)

IETF

TLS 1.0 (1999) → TLS 1.1 (2006) → TLS 1.2 (2008)



*Slide from van der Merwe and Paterson

Goals for TLS 1.3

Clean up: Remove unused or unsafe features

Improve privacy: Encrypt more of the handshake

Improve latency: Target: 1-RTT handshake for naïve clients;
0-RTT handshake for repeat connections

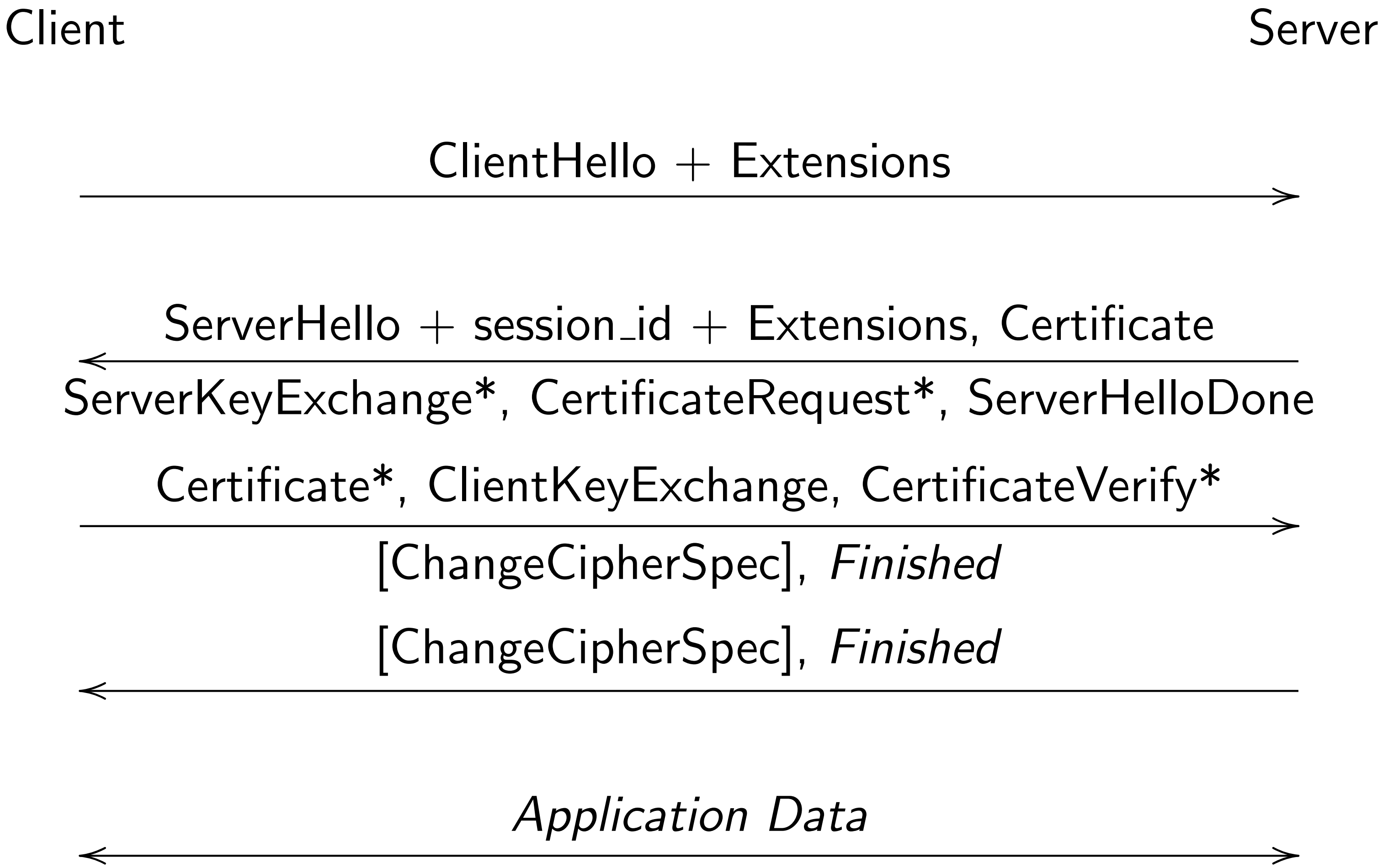
Continuity: Maintain existing important use cases

Security Assurance: Have analysis to support our work

TLS Structure

- Handshake protocol
  - Establish shared keys (typically using public key cryptography)
  - Negotiate algorithms, modes, parameters
  - Authenticate one or both sides
- Record protocol
  - Carry individual messages
  - Protected under symmetric keys
- This is a common design (SSH, IPsec, etc.)

Reminder: TLS 1.2 Full Handshake



Client

Server

ClientHello + Extensions

ServerHello + session_id + Extensions, Certificate

ServerKeyExchange*, CertificateRequest*, ServerHelloDone

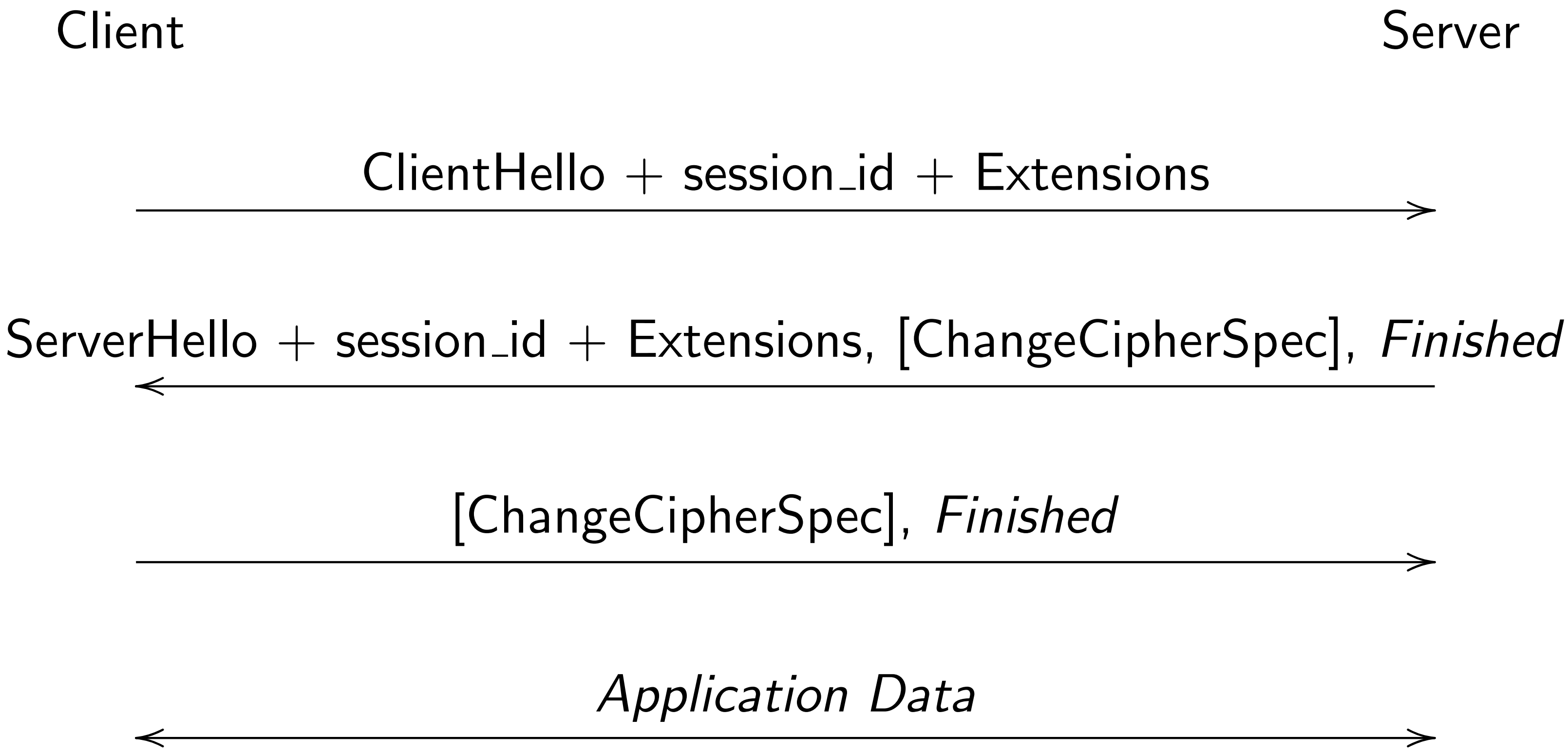
Certificate*, ClientKeyExchange, CertificateVerify*

[ChangeCipherSpec], *Finished*

[ChangeCipherSpec], *Finished*

Application Data

Reminder: TLS 1.2 Resumed Handshake



Client

Server

ClientHello + session_id + Extensions

ServerHello + session_id + Extensions, [ChangeCipherSpec], *Finished*

[ChangeCipherSpec], *Finished*

Application Data

Removed Features

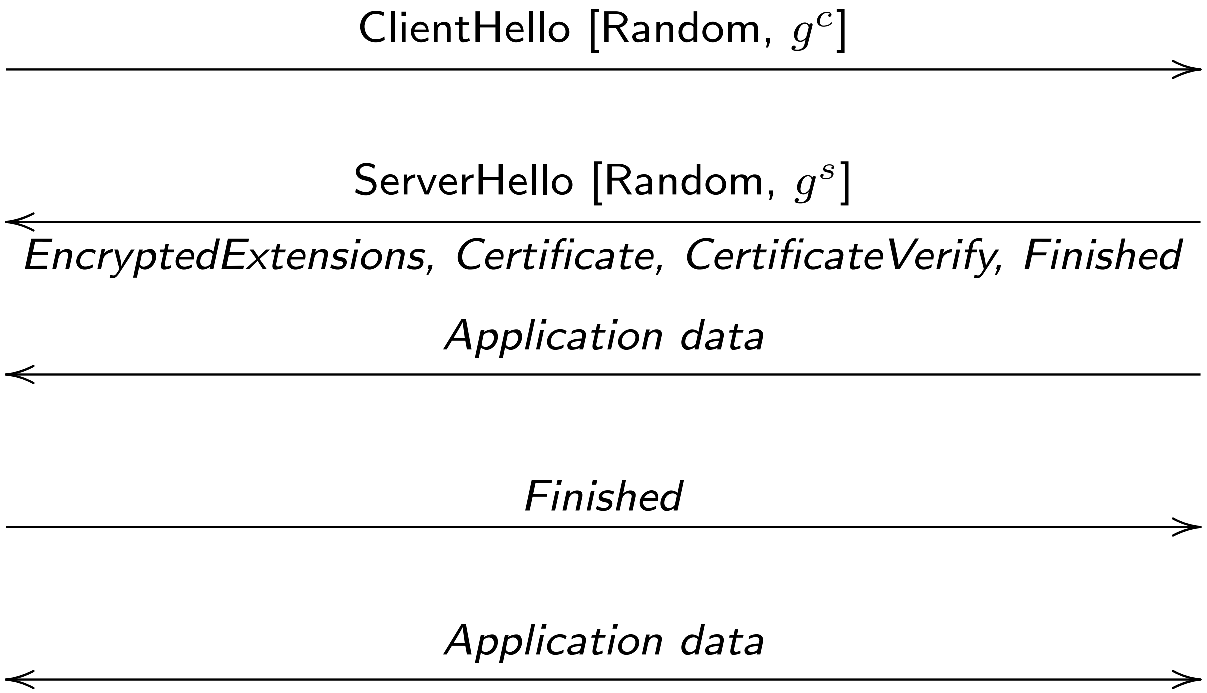
- Static RSA
- Custom (EC)DHE groups
- Compression
- Renegotiation*
- Non-AEAD ciphers
- Simplified resumption

*Special accommodation for inline client authentication

Optimizing Through Optimism

- TLS 1.2 assumed that the client knew nothing
  - First round trip mostly consumed by learning server capabilities
- TLS 1.3 narrows the range of options
  - Only (EC)DHE
  - Limited number of groups
- Client can make a good guess at server's capabilities
  - Pick its favorite groups and send DH share(s)

TLS 1.3 1-RTT Handshake Skeleton



ClientHello [Random, g^c]

ServerHello [Random, g^s]

EncryptedExtensions, Certificate, CertificateVerify, Finished

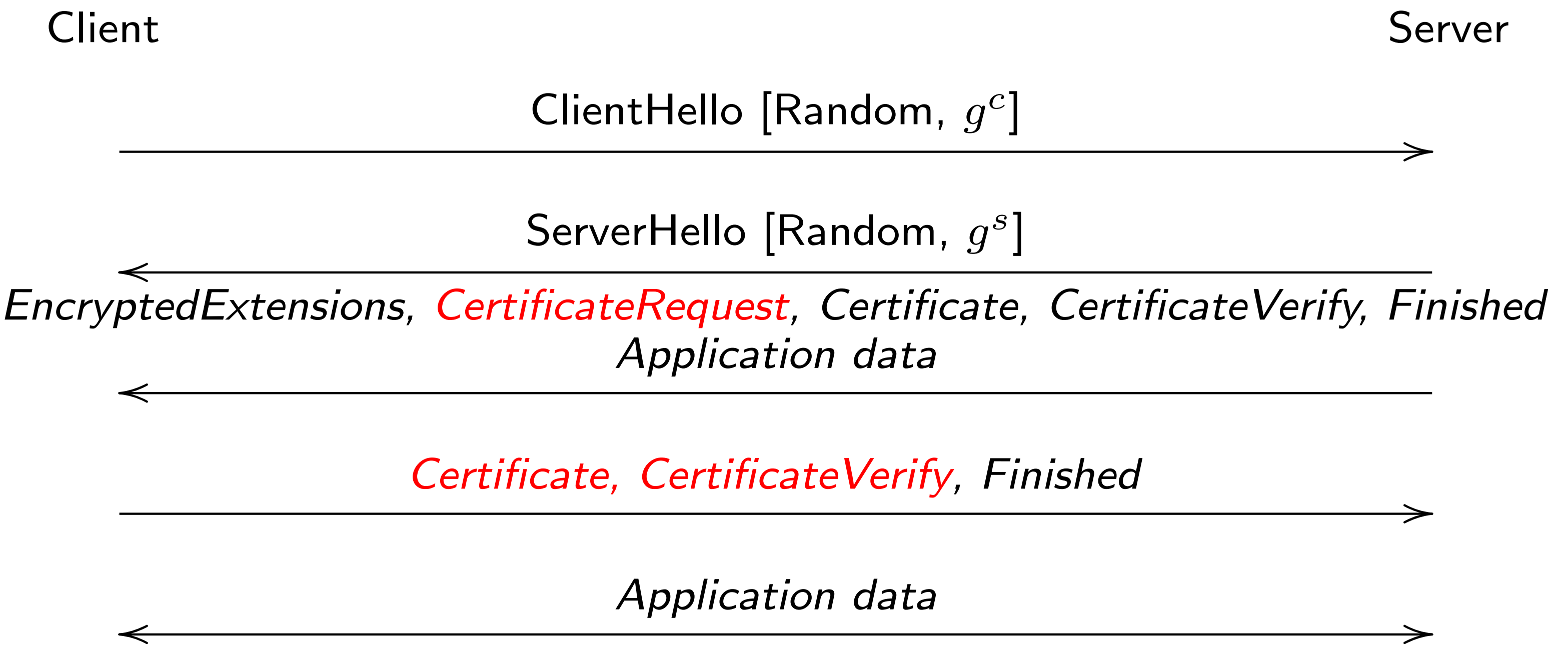
Application data

Finished

Application data

- Server can write on its first flight (e.g., banners or H2 SETTINGS)
- Client can write on second flight
- Server certificate is encrypted
  - Only secure against passive attackers

TLS 1.3 1-RTT Handshake w/ Client Authentication Skeleton



Client

Server

ClientHello [Random, g^c]

ServerHello [Random, g^s]

EncryptedExtensions, CertificateRequest, Certificate, CertificateVerify, Finished

Application data

Certificate, CertificateVerify, Finished

Application data

- Client certificate is encrypted
- Secure against an active attacker

Pre-Shared Keys and Resumption

- TLS 1.2 already supported a Pre-Shared Key (PSK) mode
  - Used for IoT-type applications
- TLS 1.3 merges PSK and resumption
  - Server provides a key label
  - ... bound to a key derived from the handshake
  - Label can be a “ticket” (encryption of the key)
- Two major modes
  - Pure PSK
  - PSK + (EC)DHE

```
Initial Handshake:
       ClientHello
        + key_share              -------->
                                                       ServerHello
                                                               ...
                                                        {Finished}
                                 <--------     [Application Data*]
       ...
       {Finished}                -------->
                                 <--------      [NewSessionTicket]
       [Application Data]        <------->      [Application Data]

Subsequent Handshake:
       ClientHello
         + pre_shared_key
         + key_share*            -------->
                                                       ServerHello
                                                  + pre_shared_key
                                                      + key_share*
                                             {EncryptedExtensions}
                                                        {Finished}
                                 <--------     [Application Data*]
       {Finished}                -------->
       [Application Data]        <------->      [Application Data]
```

0-RTT Handshake

- Basic observation: once we have established a ticket we have a shared key
  - With someone we have authenticated
- We can send *application data* on the first flight
- TLS 1.3 used to have a DH-based 0-RTT mode
  - Got stripped out due to academic and implementor feedback

TLS 1.3 0-RTT Handshake Skeleton

```
ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
(Application Data*)     -------->
                                                ServerHello
                                           + pre_shared_key
                                               + key_share*
                                      {EncryptedExtensions}
                                              + early_data*
                                                 {Finished}
                        <--------       [Application Data*]
(EndOfEarlyData)
{Finished}              -------->
[Application Data]      <------->        [Application Data]
```

Server Version Intolerance

- TLS 1.2 uses a simple version negotiation scheme
  - Client provides it's maximum version in ClientHello
  - Server chooses $min(ClientVersion, ServerVersion)$
- Unfortunately, about 1% of servers are intolerant of versions > 1.2
  - This makes it unsafe to offer TLS 1.3
- Fix
  - ClientHello.Version $=$ 1.2
  - Include a TLS extension that lists all versions the client supports
  - Nearly all servers ignore unknown extensions

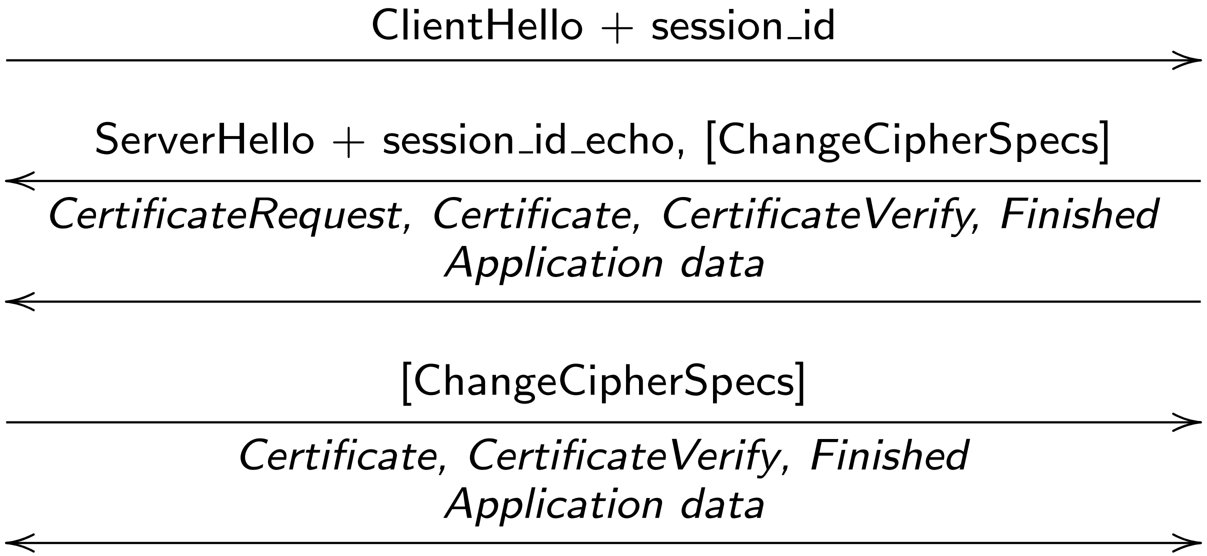
The Great Middlebox Mess

- Some middleboxes break when you negotiate TLS 1.3
- Error rates (Firefox Beta versus Cloudflare)
  - 2.2% for TLS 1.2
  - 3.9% for TLS 1.3
- What's happening?
  - They're trying to look at handshake details
  - Even when they don't know the version
- This means you need fallback to deploy TLS 1.3
- ... which also breaks anti-downgrade
- Only found this out right when everything else was done
  - Only see it when you try to deploy

What's going on here?

- Not totally clear...
  - A lot of different vendors (so probably a lot of things)
  - Chrome got a few devices in the lab
  - ... but not all of them
- Some things we know
  - Incomplete MITM
  - Protocol enforcement ("this doesn't look like TLS 1.2"...)

The fix: TLS 1.3 looks like TLS 1.2 Resumption



ClientHello + session_id

ServerHello + session_id_echo, [ChangeCipherSpecs]

CertificateRequest, Certificate, CertificateVerify, Finished
Application data

[ChangeCipherSpecs]

Certificate, CertificateVerify, Finished
Application data

- CCS is just a dummy and doesn't affect the state machine
  - Recipient ignores it
- Middlebox expects everything after CCS to be encrypted
  - And doesn't try to look inside
- This gives comparable error rates between 1.2 and 1.3 → No fallback

Incomplete MITM Problems Remain

- A MITM device is really a back-to-back proxy
- Some MITMs try to do less
  - Reuse pieces of the ClientHello
  - Filter based on server certificate
  - ... this usually ends badly
- Example: Cisco Firepower
  - TLS 1.3 uses the server Random value for anti-downgrade
  - Firepower devices forwarded the server Random value, but negotiate TLS 1.2
  - This looks like an attack → Fail
  - Reported Dec 2017, fixed in 2018

Static RSA, Passive Inspection, and You

- A lot of enterprises do TLS passive inspection
  - Inspection box attached to a span port
  - You give the RSA private key to the inspection box
  - Decrypt the EPMS and hence the whole connection?*
- TLS 1.3 breaks this (no static RSA)
- Lot of requests from enterprises to do something
  - But we didn't.
  - (they don't really need our help)

*Don't forget to disable (EC)DHE cipher suites

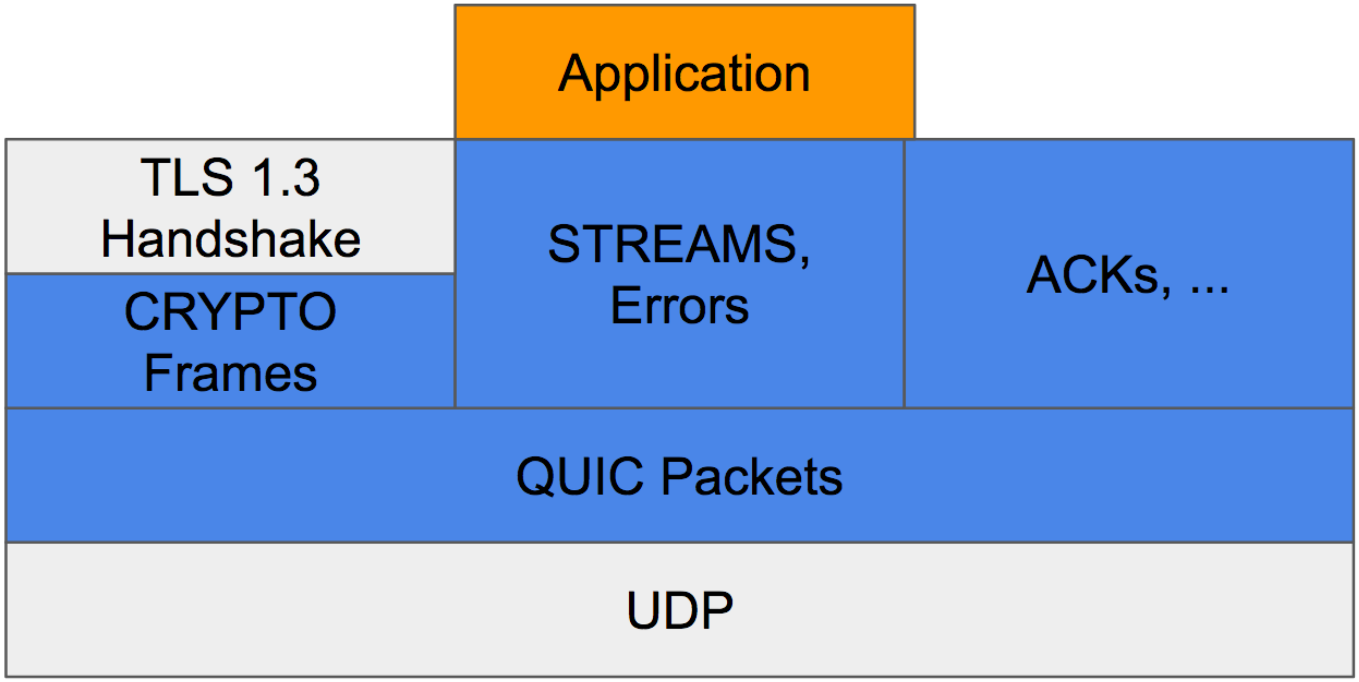
Where are we now

- RFC Published August 10, 2018
- Browsers: Firefox, Chrome, Safari
- Server operators: Akamai, Cloudflare. Facebook, Google, Apple
- Libraries: OpenSSL, BoringSSL, NSS, Fizz, PicoTLS, ...
- $\approx 20\%$ of Firefox connections
- $> 50\%$ of Facebook connections!

QUIC

- TLS 1.3 is a big improvement
  - But it still runs over TCP
- A new transport protocol can do better
  - Iterate more quickly
  - Shorten the handshake (TFO only sort-of works)
  - Multiplexing without head-of-line blocking
  - Protect more of the protocol from attack

QUIC Architecture



Quick iteration

- QUIC can be implemented in user space
- This means we can roll out new versions quickly
  - Without waiting for the operating system
  - Chrome and Firefox ship every 6-8 weeks
- This capability got used extensively for TLS 1.3 and is expected for QUIC

True 0-RTT

- We want to send *data* in the first flight
  - TLS 1.3 lets you send application data with the first TCP data
  - ... but this is after the TCP handshake
  - TCP Fast Open in principle allows this
  - ... but middleboxes get in the way
- Layering on top of UDP helps
  - Can just send data in first flight
  - Middleboxes don't try to "help"
  - ... Though sometimes they block stuff

Multiplexing without head-of-line blocking

- HTTP/2 had multiplexing (streams)
  - But all the streams run over the same TCP/TLS channel
  - This means you get head-of-line blocking on packet loss
- QUIC runs over UDP and provides its own reliability
  - This means no head-of-line blocking in typical scenarios*
  - Biggest improvement in cases of high packet loss

*Some exceptions may apply when one stream depends on another; also the handshake

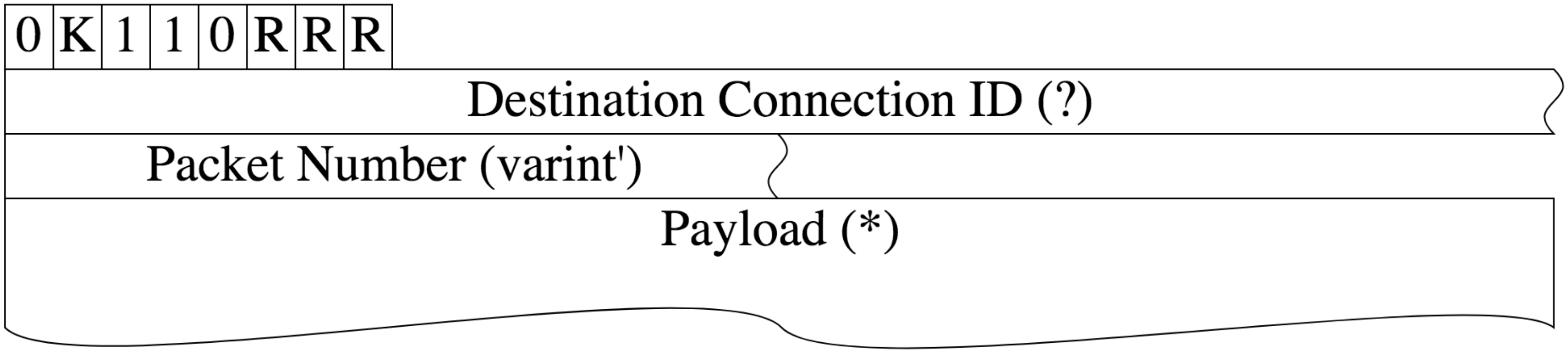
Protect More of the Protocol From Attack

- TLS 1.3 runs over TCP
  - People can still attack the TCP channel
  - ... e.g., RST attacks
- Everything in QUIC is encrypted
  - Including the transport meta-information (packet numbers, stream offsets, ACKs, errors, etc.)
  - Attackers (or network operators) can't see connection state
  - ... or tear down the connection

Ossification Defenses

- Network middleboxes tend to assume protocols are invariant
  - ... and fail unpredictably when those invariants are violated (cf. TLS 1.3 version problem)
- QUIC ossification countermeasures
  - Encrypt as much as possible
  - Publish explicit protocol invariants
  - “Grease” reserved bits

QUIC Packet Headers*

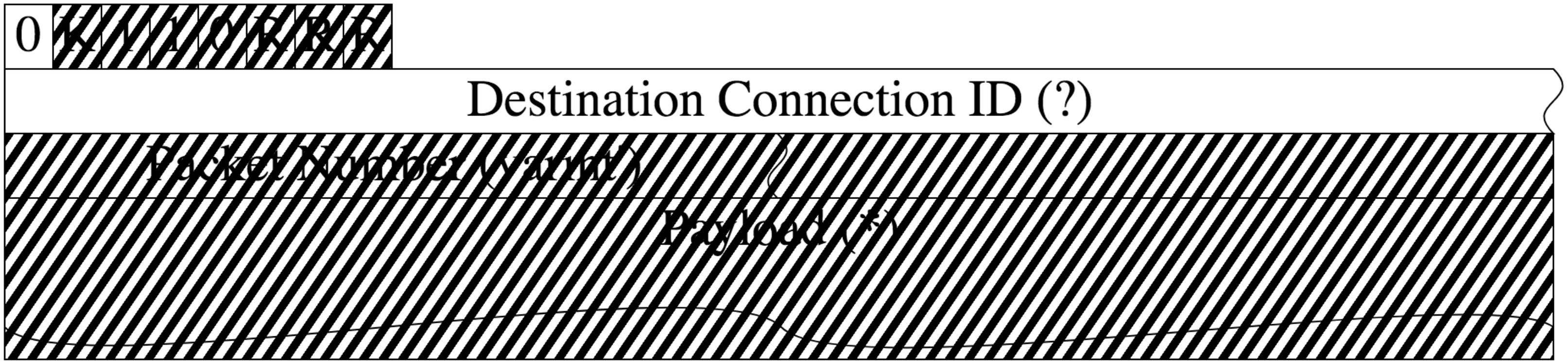


0 K 1 1 0 R R R

Destination Connection ID (?)

Packet Number (varint')

Payload (*)



0 K 1 1 0 R R R

Destination Connection ID (?)

Packet Number (varint')

Payload (*)

*Slightly out of date...

Really, it's all encrypted

- Handshake is encrypted with a deterministic key
  - Derived from the connection IDs
  - And a per-QUIC version constant
  - Middleboxes can't decrypt future unknown versions of QUIC
- Most exposed reserved bits are "greased"
  - Send random bits in their place
  - Ensures that endpoints and middleboxes don't depend on them
  - Authenticated so they can't be changed

What about the QUIC version number?

- The version number in the handshake is in the clear
  - Concerns that middleboxes will enforce that
  - ... and terminate QUIC connections with other versions
- Potential approaches
  - Remove the version number and use trial decryption to detect version
  - Distribute "alternative" versions somehow
  - Distribute keys to encrypt more of the handshake somehow
  - Do nothing?
- This is currently an unsolved problem

  `https://github.com/quicwg/base-drafts/issues/2496`

DNS Security is Bad

- Most clients get DNS from their network
  - Server delivered over unauthenticated DHCP
  - Unencrypted DNS transport to resolver
  - No way to know resolver's security or Mprivacy policy
- Lots of security and privacy problems here
  - On-network attackers
  - Attacks by the resolver
    - ∗ Surveillance
    - ∗ Censorship
    - ∗ Typo "correction"
  - Privacy-hostile behaviors by the resolver (EDNS0-Client-Subnet, no QMIN, ...)

An aside: Why not DNSSEC?

- Reminder: DNSSEC is a PKI for domain names
  - Rooted in the DNS root
- DNSSEC doesn't provide privacy
- Still possible to do blocking
  - Forge an NXDOMAIN
  - Non-DNSSEC clients (almost everyone) are fooled
  - DNSSEC clients can see something is wrong
    - ∗ But they still can't recover

DNSSEC Deployment Issues

- Almost all current DNSSEC validation is by the resolver
  - Comcast, Google, Cloudflare, Quad9 all do this
- Our threat model includes the resolver
  - So validation has to be at the endpoint
- Problem: too many false positives
  - Many middleboxes tamper with DNS – or can't do large records correctly
    - EDNS(0) and DNS/TCP not universally supported
    - In 2015 TXT records failed about 4-5% of the time*
  - This is indistinguishable from an attack
  - Hard-failing on DNSSEC validation failure is infeasible
- Maybe DoH will fix this?

*https://www.imperialviolet.org/2015/01/17/notdane.html

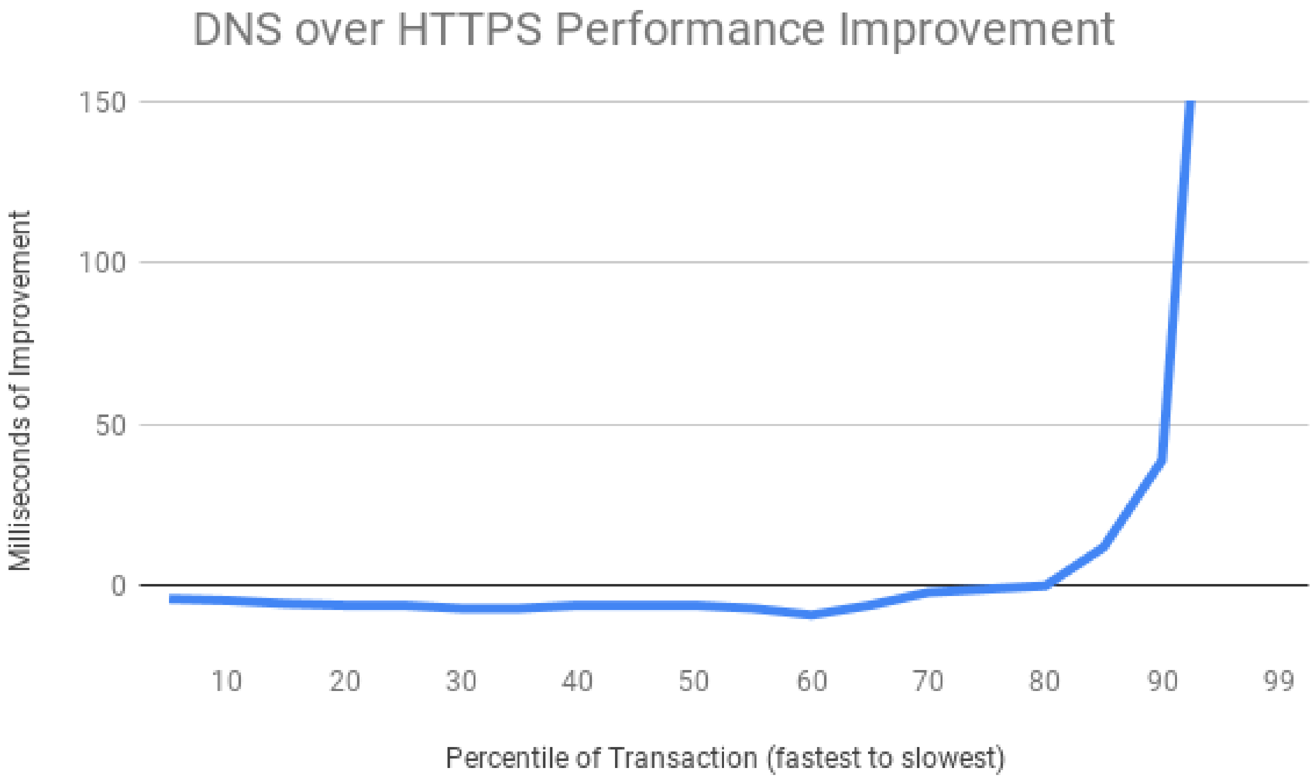
DNS over HTTPS

- What it sounds like
  - DNS packets over HTTPS
- Technically just a new transport for DNS
  - Harder to block
  - Can mux HTTP and DNS traffic
- But often conflated with Trusted Recursive Resolvers
  - Specific DoH deployment model
  - *Application* picks a resolver
  - ... based on application developer's relationship with resolver

DoH/TRR in Firefox

- DoH support in Firefox (disabled by default)
- Currently performing experiments to determine viability
  - Things are looking pretty good so far
  - Plan to ship it by default once we're confident
- Currently use Cloudflare's resolver
  - Cloudflare signed up to a strong privacy policy
  - Looking for other partners (especially outside the US)

DOH Performance



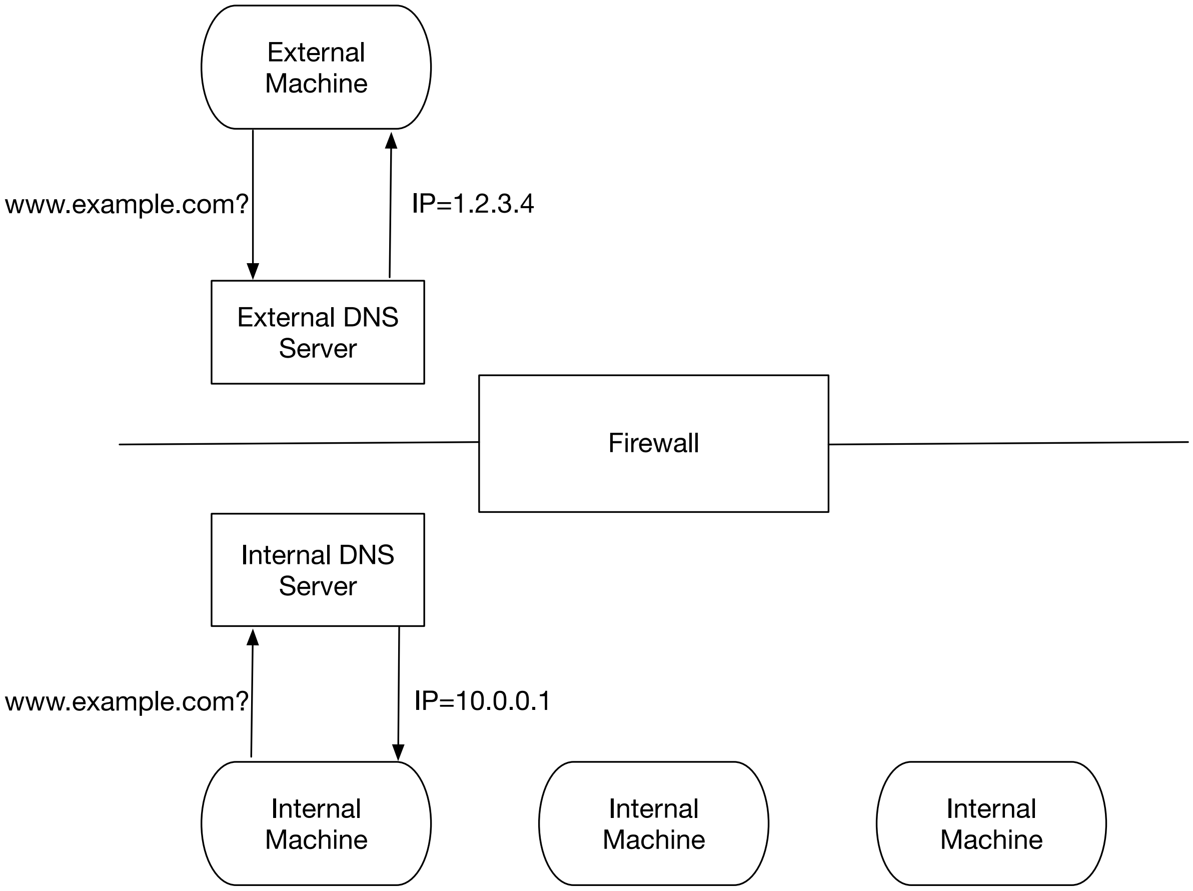
One small step...

- This is an improvement
  - ... but it still doesn't fix everything
- And comes with costs
  - Increased centralization
  - No competition for DoH service
  - Potentially suboptimal routing
  - Makes network filtering much harder

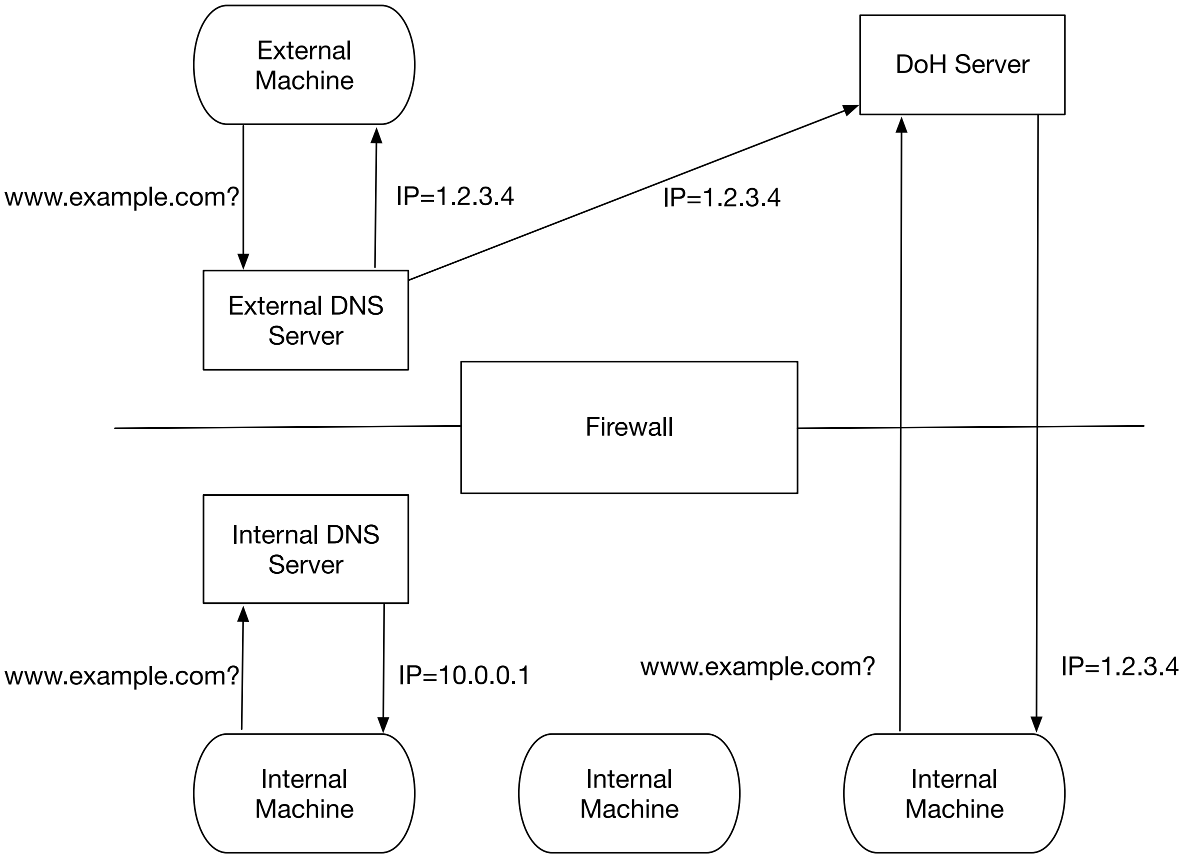
DNS Filtering

- A lot of networks filter DNS
  - Enterprise policy enforcement
  - Malware and C&C blocking
  - Parental controls (typically on adult content)
  - National level blocking
- This looks just like an attacker
  - And in some cases (e.g., censorship) it is
  - But sometimes it's what the user wanted

Split Horizon



Split Horizon after DoH



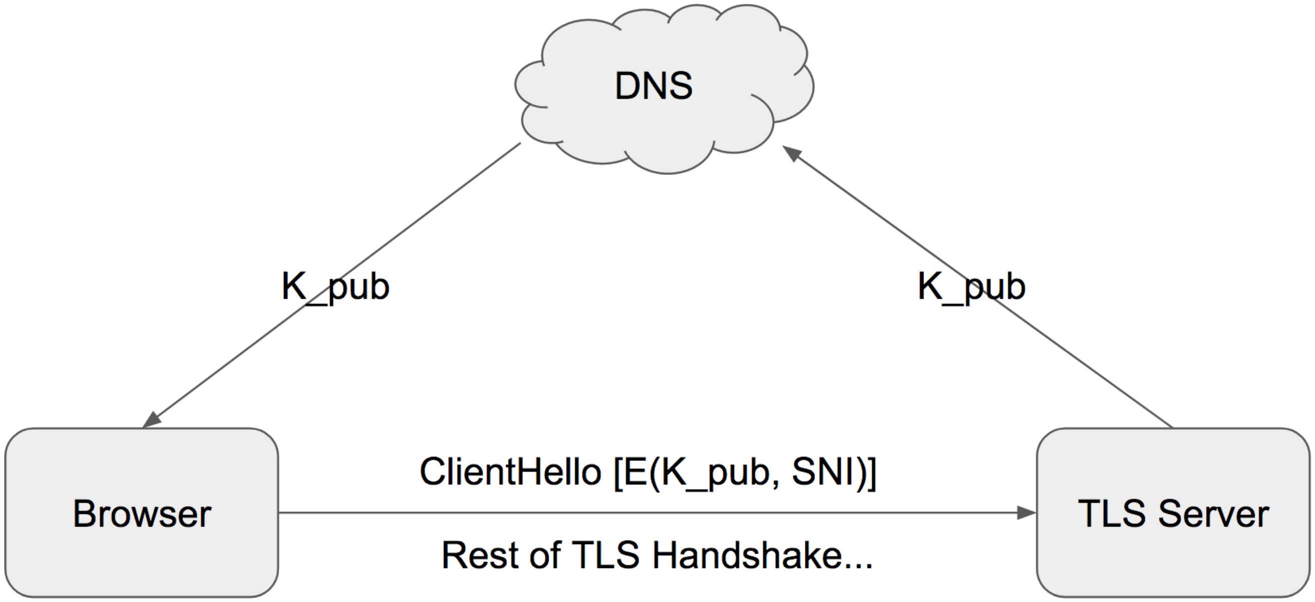
Unexpected Behaviors

- Ideally enable DoH by default
  - Allow the user to choose a different server or disable DoH
  - Allow “enterprise” configuration or disabling of DoH
  - Allow networks to pick out of the trusted resolver set
- Unfortunately machines aren’t configured this way now
  - So this breaks filtering whether the user wants that or not
  - Heuristically disable DoH?
    - When devices are under central management
    - When we detect blocking
      - But this makes blocking (and hence censorship) easy
- Still working on our rollout plan

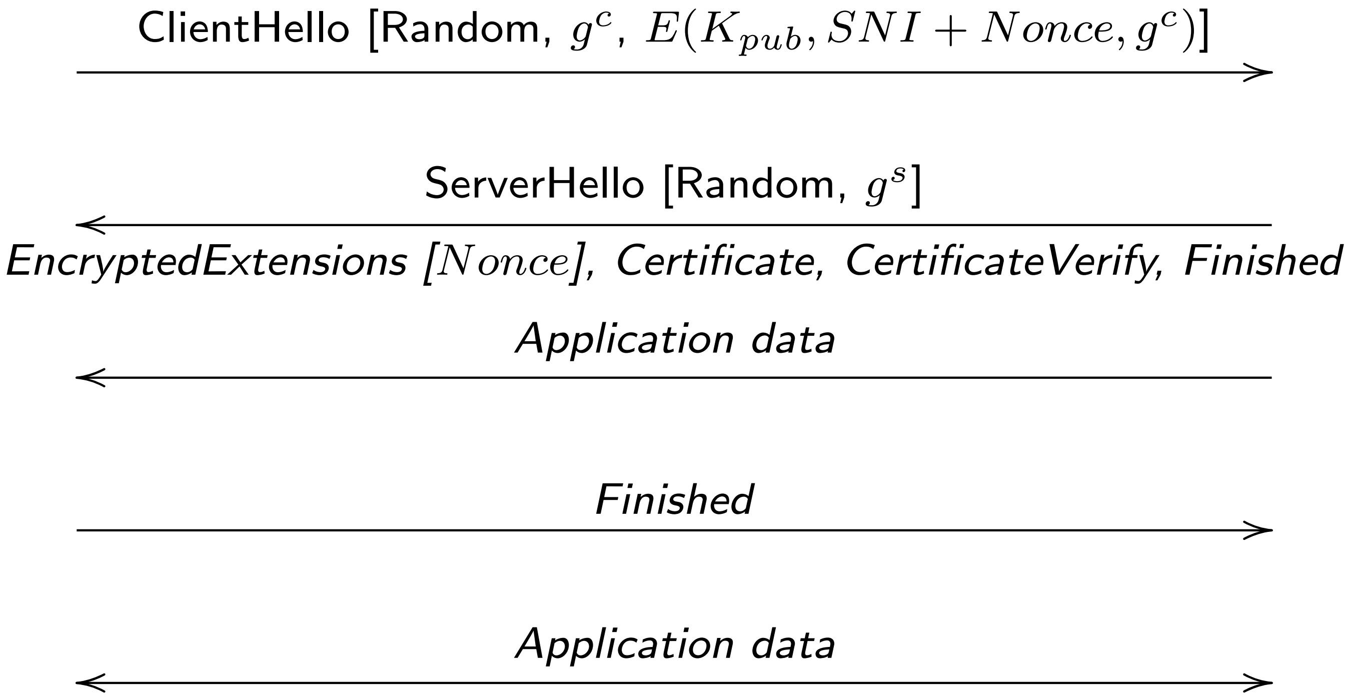
Encrypted SNI

- Server Name Indication (SNI) enables TLS virtual hosting
  - ... but leaks your destination to the network
  - even when multiple servers on the same IP
- TLS 1.3 encrypts the server certificate but not the SNI
  - Not because we didn't try
  - Just couldn't figure out how to do it well
  - Some good ideas about six months ago

ESNI Architecture



ESNI in TLS 1.3



ClientHello [Random, g^c, $E(K_{pub}, SNI + Nonce, g^c)$]

ServerHello [Random, g^s]

EncryptedExtensions [$Nonce$], Certificate, CertificateVerify, Finished

Application data

Finished

Application data

- Client sends SNI, nonce encrypted under server public key
- Server echoes nonce
- This is TLS 1.3 only (for real!)

Multi-CDN Issues

- Many sites are served by multiple CDNs
  - Use a third-party service to switch between them
  - Usually uses a CNAME record which points to either `cdn1.com` or `cdn2.com`
- Possible to get inconsistent records
  - ESNI keys for CDN1 and addresses (A records) for CDN2
  - This will cause hard failure
- No good fixes
  - Combined record with ESNI keys and A record
  - Carry A record “filters” with ESNI keys
    - Retry on filter failure
- A lot more coordination between DNS and TLS than we would like

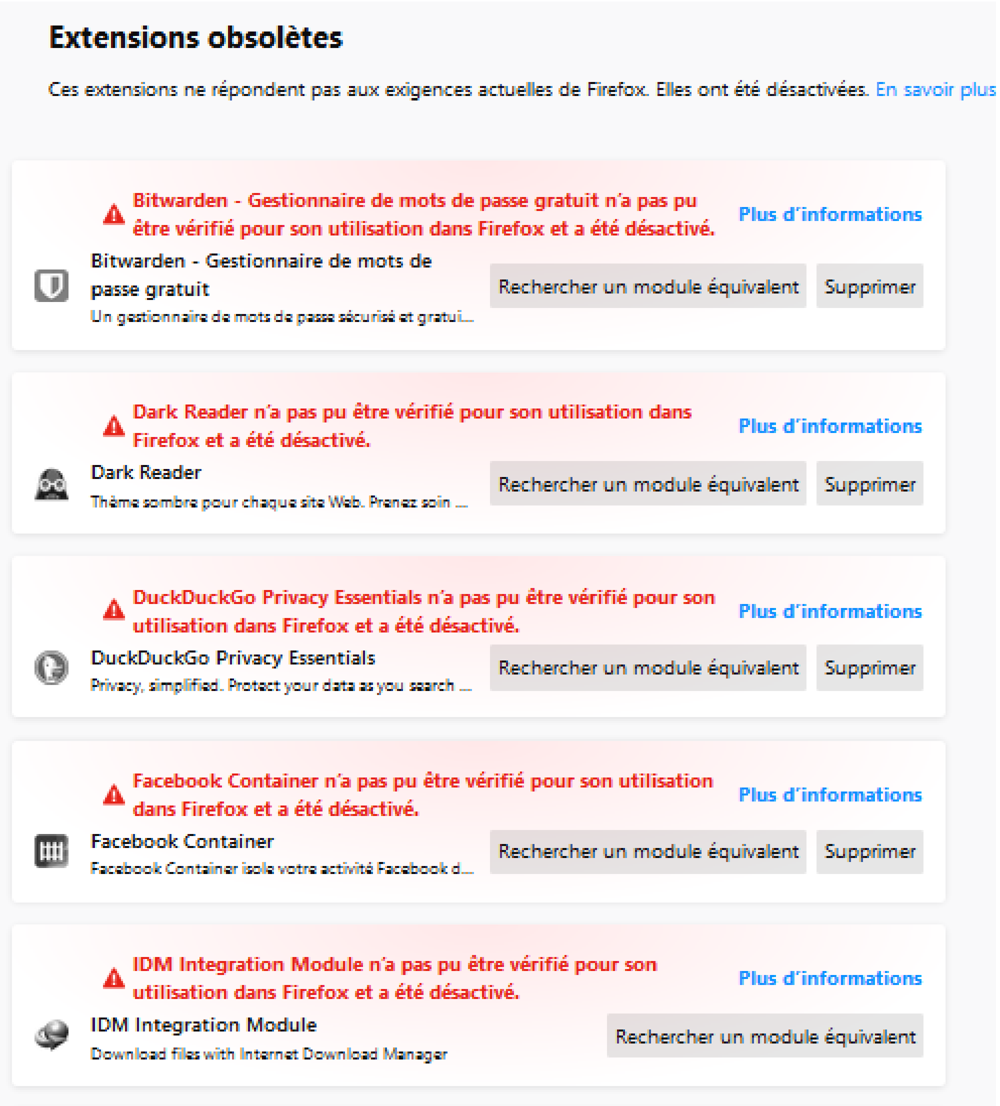
ESNI Status

- IETF WG draft
- Already live on Cloudflare
- Available in Firefox Nightly
- Probably still a lot of churn before it’s done
- Can also be used with QUIC

A Recent Emergency

- Firefox is an extensible browser
  - Users can download *add-ons* that extend the behavior of Firefox
- All add-ons have to be *signed* by Mozilla
  - Enforce policies
  - Allow for blocklisting extensions which we know to be bad
- Signatures authorized by a certificate chain tied to a trust anchor in the browser
  - May 4, just after midnight UTC, one of the intermediate certificates expired
  - ... oops

This is what failure looks like



Extensions obsolètes

Ces extensions ne répondent pas aux exigences actuelles de Firefox. Elles ont été désactivées. En savoir plus

⚠ Bitwarden - Gestionnaire de mots de passe gratuit n'a pas pu être vérifié pour son utilisation dans Firefox et a été désactivé.

Bitwarden - Gestionnaire de mots de passe gratuit

Un gestionnaire de mots de passe sécurisé et gratui...

⚠ Dark Reader n'a pas pu être vérifié pour son utilisation dans Firefox et a été désactivé.

Dark Reader

Thème sombre pour chaque site Web. Prenez soin ...

⚠ DuckDuckGo Privacy Essentials n'a pas pu être vérifié pour son utilisation dans Firefox et a été désactivé.

DuckDuckGo Privacy Essentials

Privacy, simplified. Protect your data as you search ...

⚠ Facebook Container n'a pas pu être vérifié pour son utilisation dans Firefox et a été désactivé.

Facebook Container

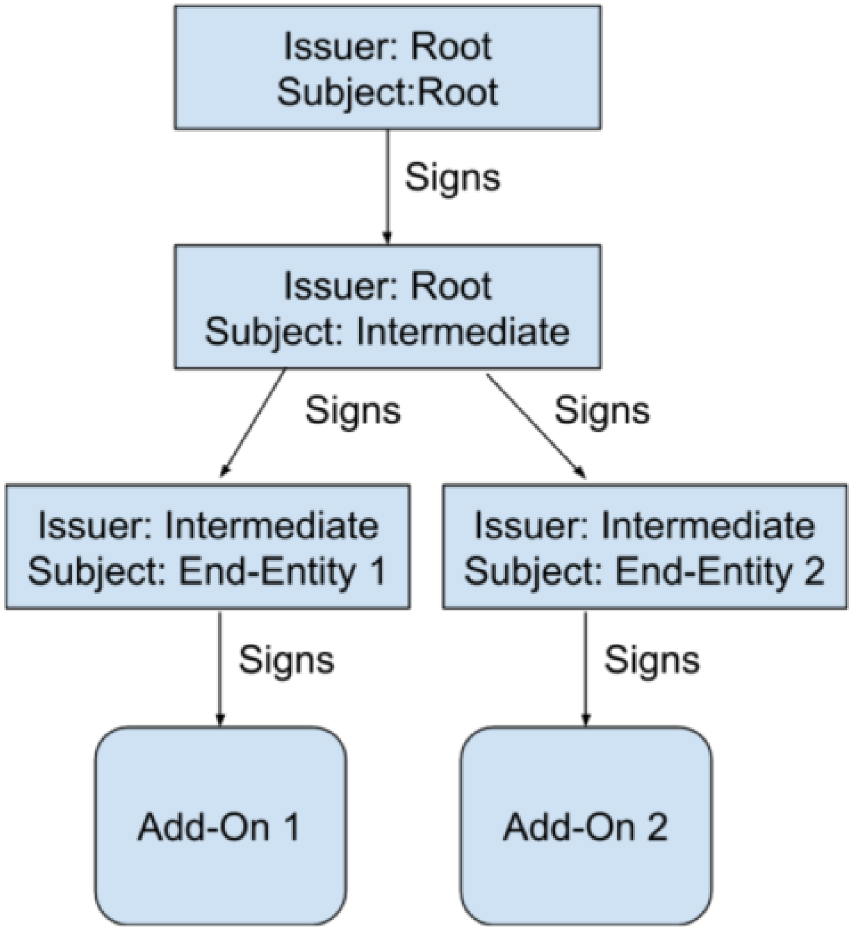
Facebook Container isole votre activité Facebook d...

⚠ IDM Integration Module n'a pas pu être vérifié pour son utilisation dans Firefox et a été désactivé.

IDM Integration Module

Download files with Internet Download Manager

Add-on Certificate Hierarchy



Damage Limitation

- Add-ons are re-checked on a 24-hour clock
  - So many users still had working add-ons
  - This would get worse as time went by
- First step: remotely disable add-on checking
  - This stabilizes the situation for unaffected users

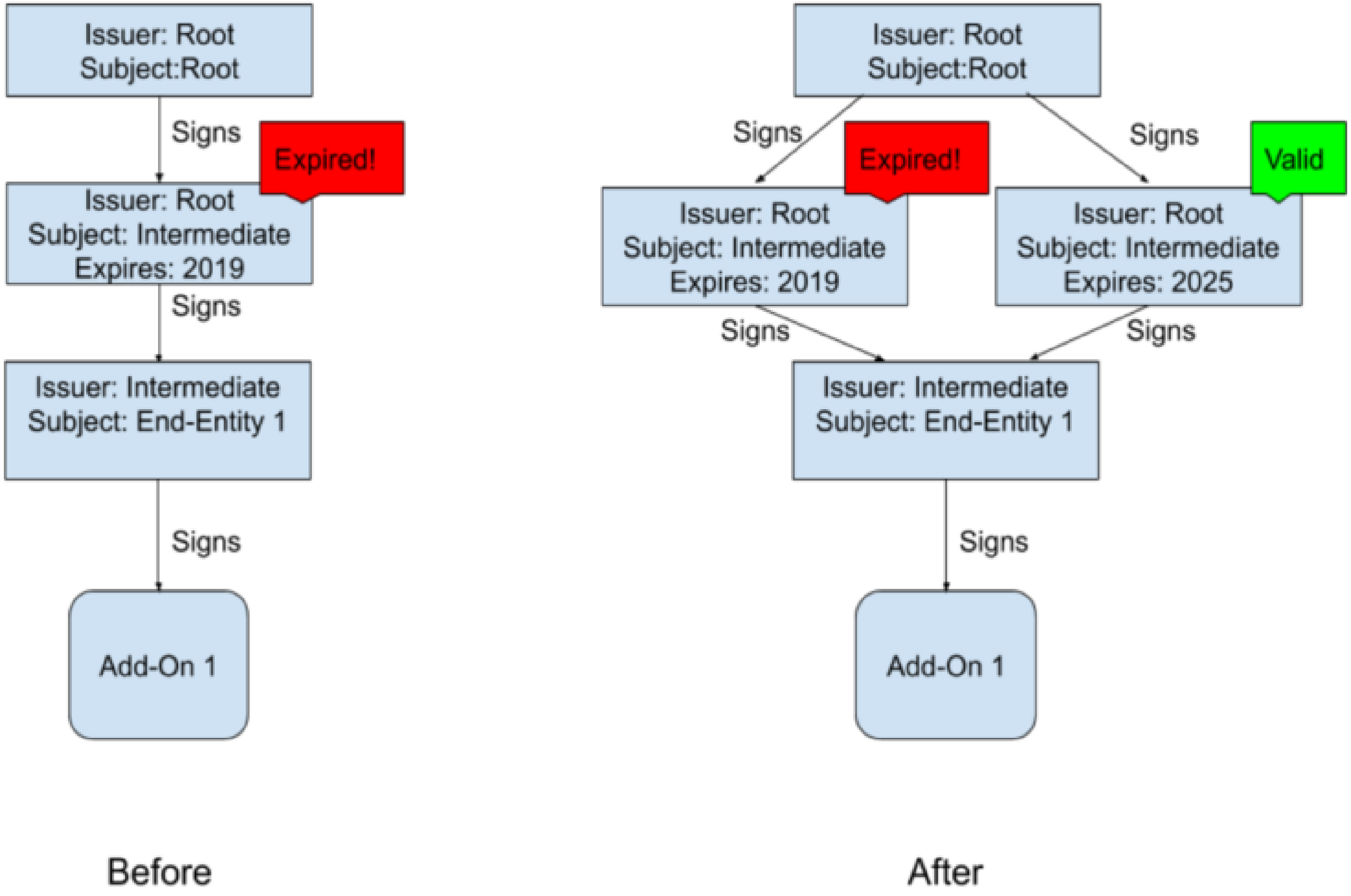
Why not just re-sign everything?

- Too slow
  - About 15,000 add-ons
  - The signing system isn’t designed for bulk signing
- Too hard to distribute the new add-ons
  - Add-ons update on a 24-hour schedule
  - Some add-ons are manually installed
- Needed an alternative approach

Some surprising facts about certificate validation

- Each add-on comes with all the certificates you need to validate it
- But these aren't used directly
  - All the certificates are inserted into a database
  - Then we try to construct a chain working back from the leaf
    - Using *all* available certificates
    - ... and trying multiple paths in parallel
- This implies a potential fix
  - Make a new valid certificate *with the same name and key*
  - Remotely install it in Firefox
  - Profit

Repaired Certificate Hierarchy



Remote installation

- Use a new add-on ("system add-on")
  - Signed with the *new* certificate
- Add-on does two things
  - Installs new certificate in the permanent database*
  - Re-verifies every add-on
    - Which should re-activate them
- Fix developed and deployed in 9 hours
  - Using our "Studies" system

*This isn't specially trusted, it's just there

Mostly a success

- Not all users have Studies enabled
  - People who disabled Telemetry/Studies (especially in enterprised)
  - Firefox on Android
  - Some downstream builds
  - People behind MITM proxies*
  - Very old versions of Firefox
- Need a dot release to fix most of these
- We had some bugs (remember, this was all done in 9 hours)

*They run everything

An interesting bug

- We install the certificate and then re-check all add-ons
- What happens if the certificate installation fails?
- Result: add-on check fails and all add-ons are disabled
  - No-op for people who were unaffected
  - But breaks everyone we had protected by disabling re-checking
- This is a case we hadn't anticipated

Final Thoughts

- The deployment universe is incredible hostile
  - Almost anything you do will probably break something
  - Need extensive measurement and experiment/testing to keep breakage within acceptable limits
- Many network elements take advantage of plaintext
  - This makes it very hard to change things
  - ... even when they're not trying to stop you
  - Solution is to encrypt as much as possible
- Many of these issues aren't about communications security per se
  - But about network protocol design... and politics
- We're making progress anyway

Questions?

You might be interested in

- IETF main page: `https://www.ietf.org/`
- TLS WG: `https://tlswg.org/`
- QUIC WG: `https://quicwg.org/`
- DOH WG: `https://datatracker.ietf.org/wg/doh/about/`