rustls: modern, faster, safer TLS

RustFest Paris
26th May 2018

This talk

1

A quick introduction to TLS

2

TLS support in the Rust ecosystem

3

rustls

This talk

1

A quick introduction to TLS

2

TLS support in the Rust ecosystem

3

rustls

Transport Layer Security

Previously known as SSL - Secure Sockets Layer

https://blog.mozilla.org/security/

A quick introduction to TLS

Goals

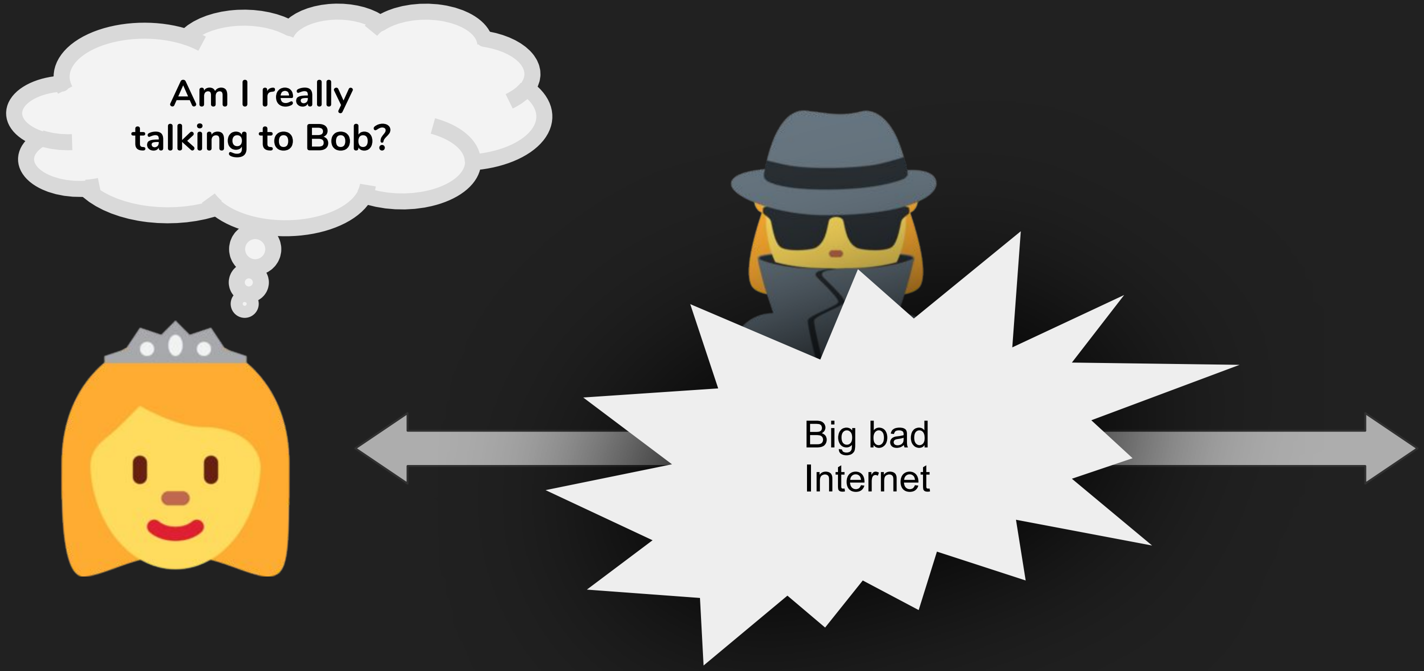
History

Security

TLS goals: Confidentiality

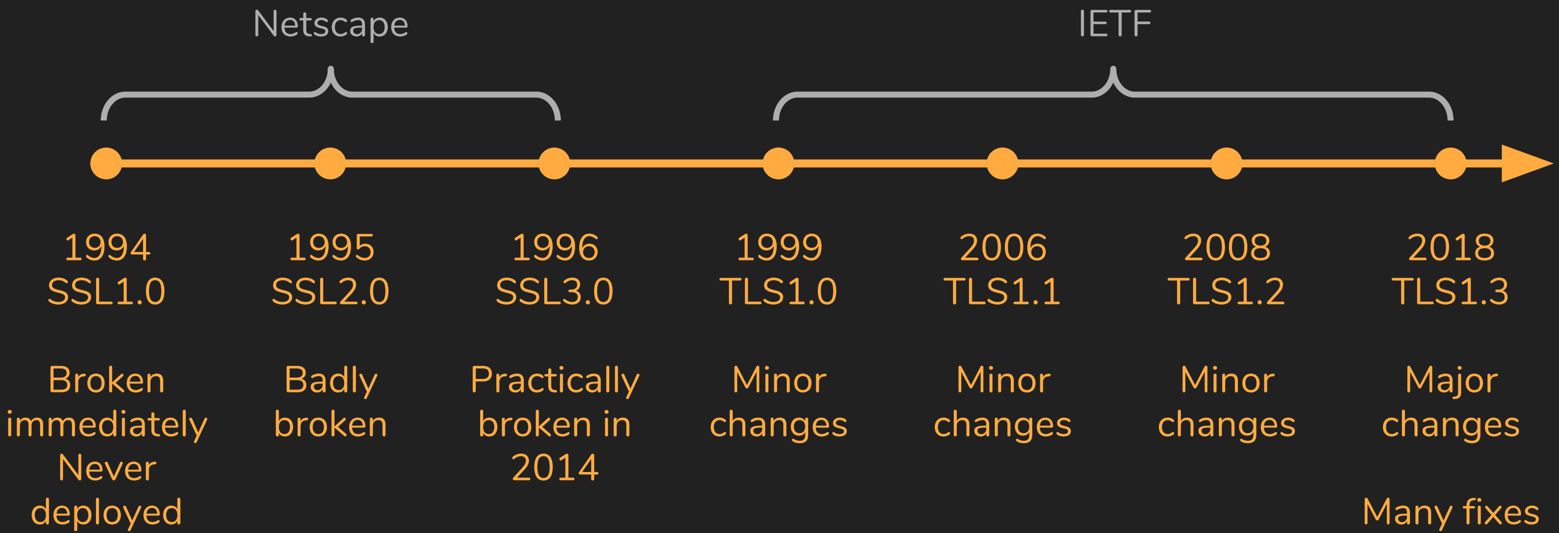
What can Eve learn from my traffic?
Big bad Internet

TLS goals: Authenticity



TLS goals: Integrity

Is what I received precisely what Bob sent?
Big bad Internet
Is what I received precisely what Alice sent?



Netscape

IETF

1994
SSL1.0

Broken immediately
Never deployed

1995
SSL2.0

Badly broken

1996
SSL3.0

Practically broken in 2014

1999
TLS1.0

Minor changes

2006
TLS1.1

Minor changes

2008
TLS1.2

Minor changes

2018
TLS1.3

Major changes

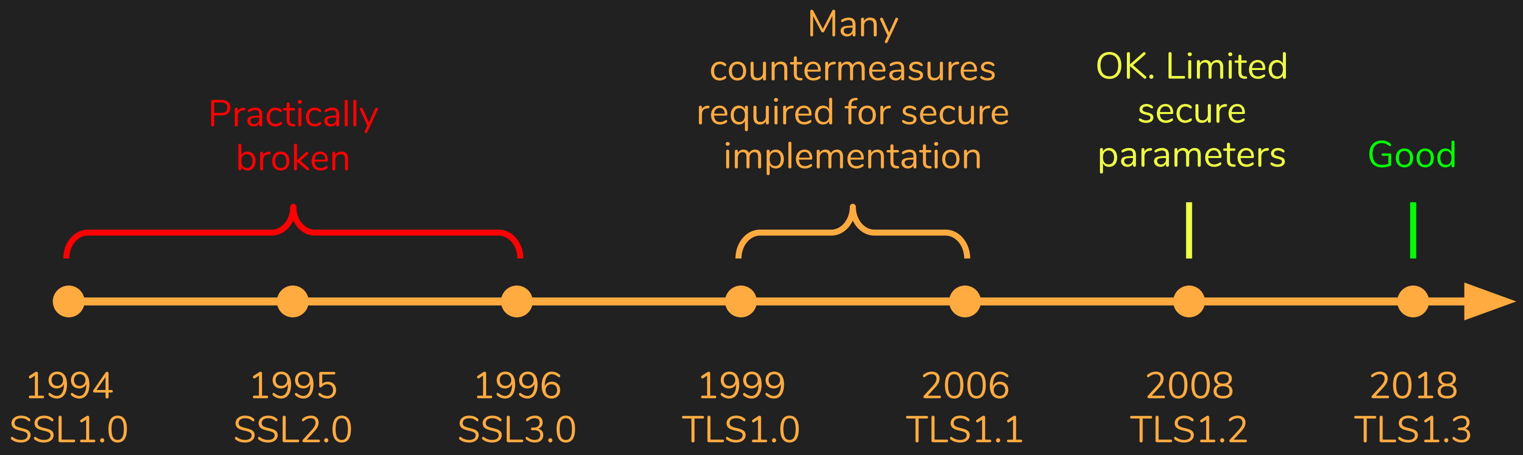
Many fixes

Brief history of SSL/TLS

	1994	1995	1996	1999	2006	2008	2018
	SSL1.0	SSL2.0	SSL3.0	TLS1.0	TLS1.1	TLS1.2	TLS1.3
		2.9%	11.0%	85.5%	84.9%	91.9%	
April-May Δ		-0.1%	-0.3%	-1.4%	-0.3%	+0.4%	

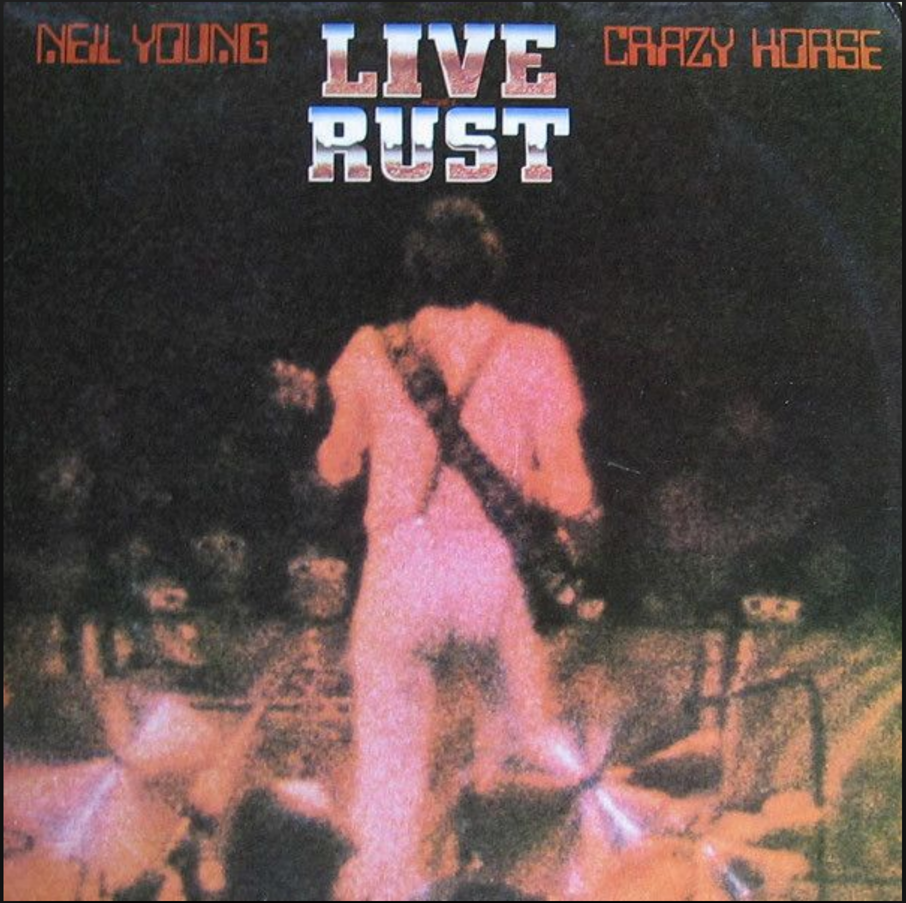
Current support for SSL/TLS by version

Source: Qualys SSL Pulse, May 2018
Data covers servers only



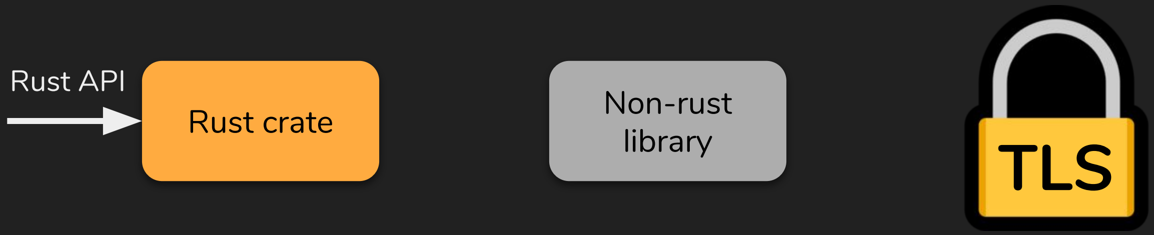
Current security results for SSL/TLS

So, what about Rust?



Rust ecosystem TLS support

Legend



Rust API

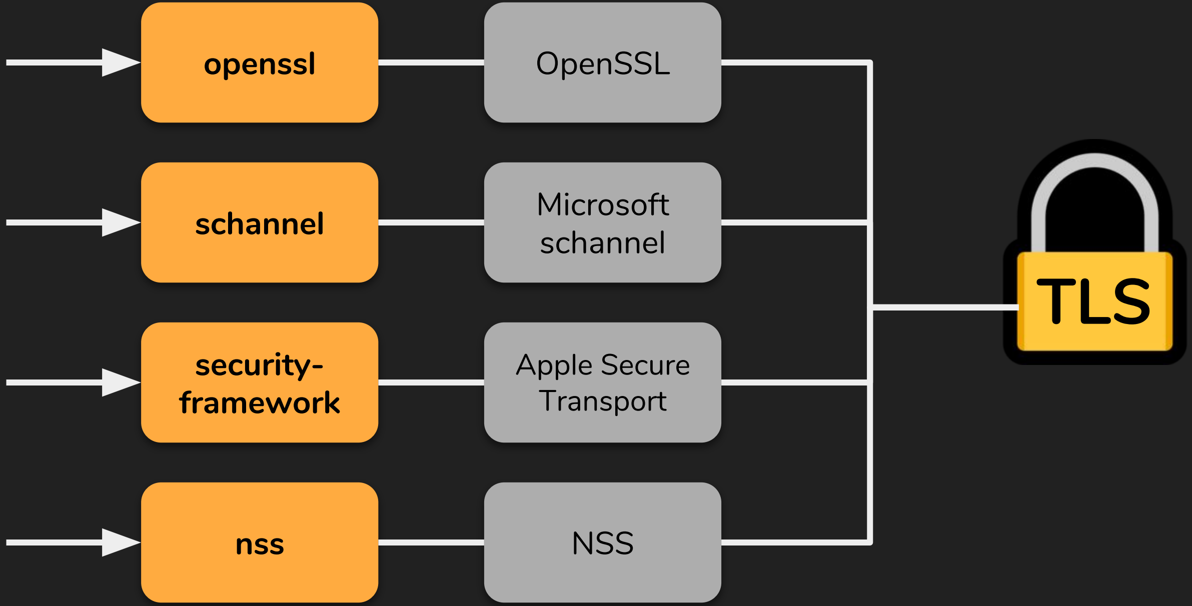
Rust crate

Non-rust library

TLS

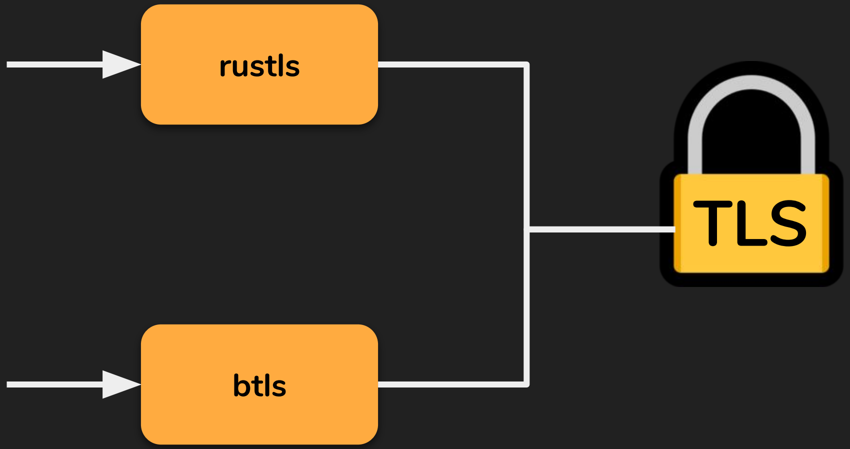
Rust ecosystem TLS support

Bindings



Rust ecosystem TLS support

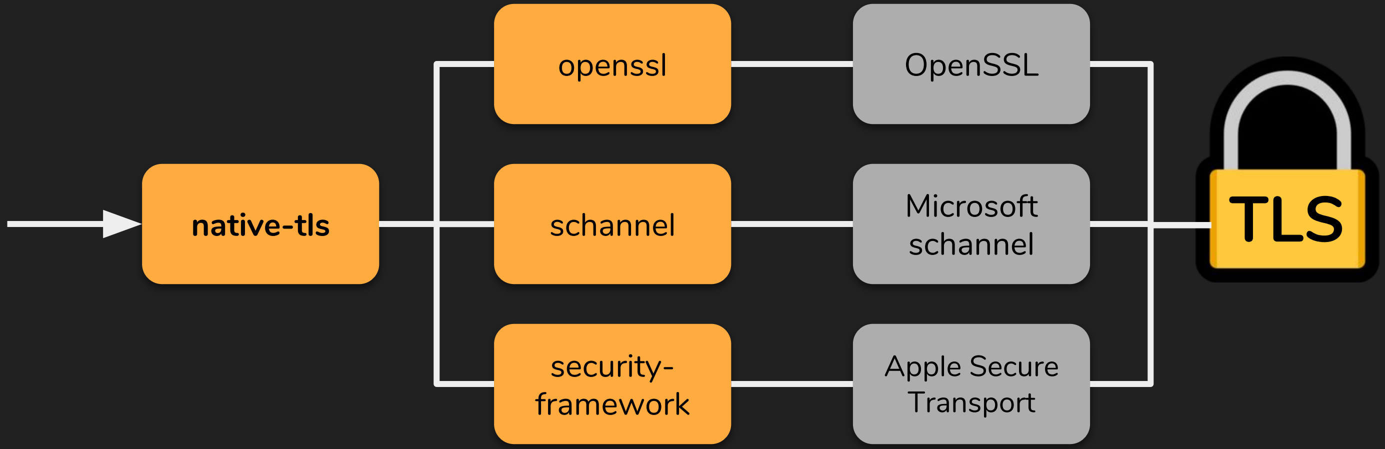
Implementations



btls: https://gitlab.com/ilari_l/btls

Rust ecosystem TLS support

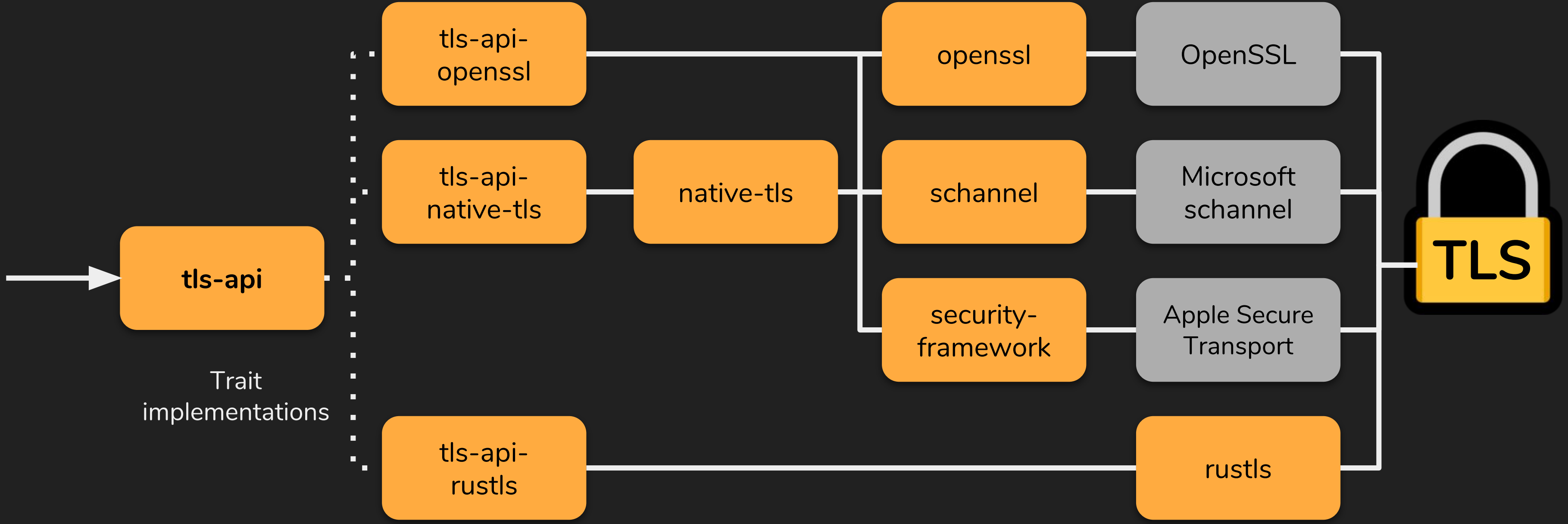
Abstractions



Automatically selected depending on platform

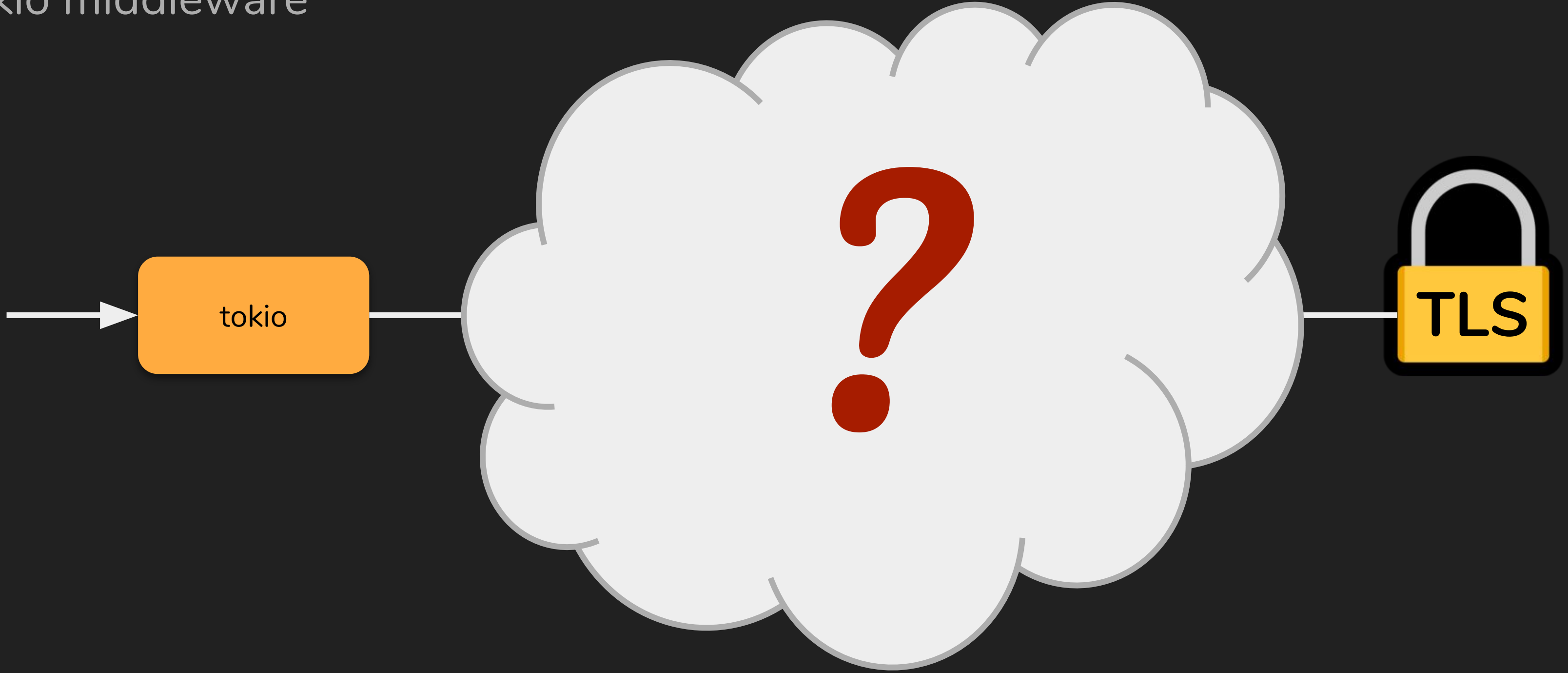
Rust ecosystem TLS support

Abstractions



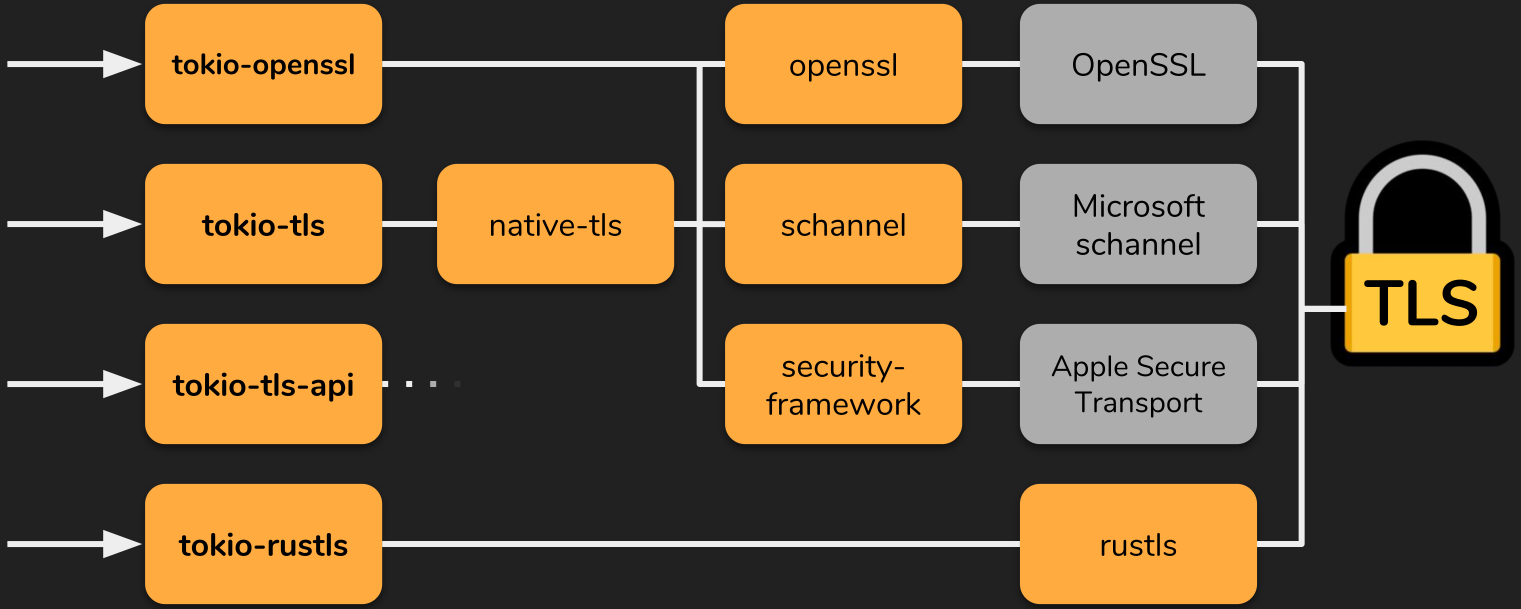
Rust ecosystem TLS support

Tokio middleware



Rust ecosystem TLS support

Tokio middleware



So, what about rustls?

https://github.com/ctz/rustls

- implements TLS1.2 and TLS1.3
- in safe subset of Rust
- Apache2.0/MIT/ISC triple-licensed

rustls aims

- ✓ modern cryptography
- ✓ no security configuration needed
- ✓ simple, pipe-y, IO-agnostic API
- ✓ no unsafe features *
- ✓ target of ~95% compatibility

* in the rust and TLS senses

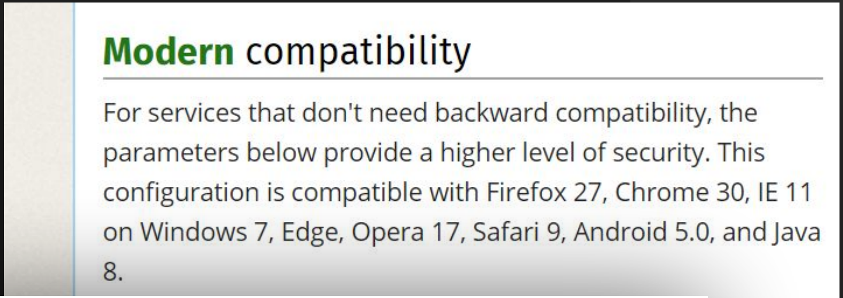
rustls
brief history

first commit	2nd May 2016
first connection	27th May 2016 🎂
TLS1.3 added	~November 2016

23 contributors so far -- thanks!

what does "Modern TLS" actually mean?

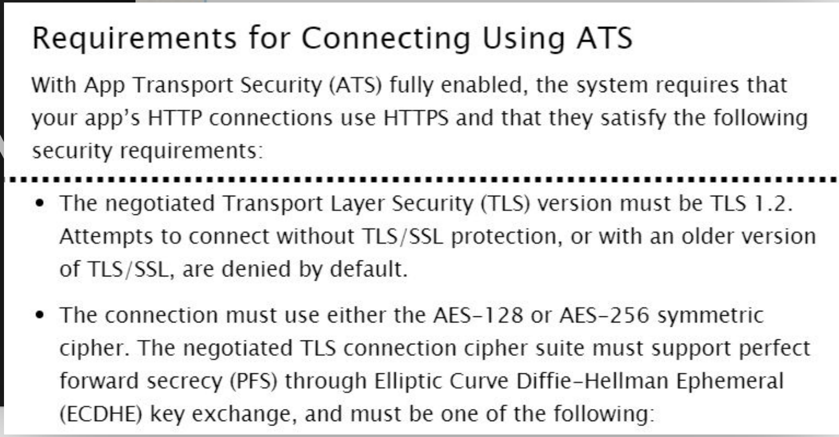
- TLS1.2 and later only
- Strong cryptography only
- At some compatibility cost



Modern compatibility

For services that don't need backward compatibility, the parameters below provide a higher level of security. This configuration is compatible with Firefox 27, Chrome 30, IE 11 on Windows 7, Edge, Opera 17, Safari 9, Android 5.0, and Java 8.

mozilla



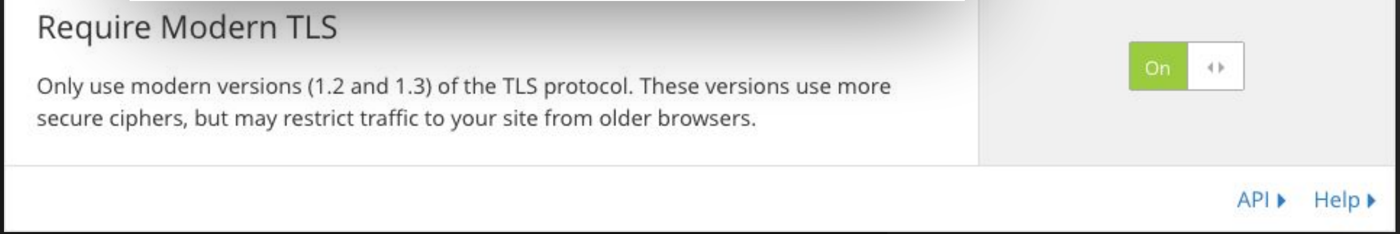
Requirements for Connecting Using ATS

With App Transport Security (ATS) fully enabled, the system requires that your app's HTTP connections use HTTPS and that they satisfy the following security requirements:

- The negotiated Transport Layer Security (TLS) version must be TLS 1.2. Attempts to connect without TLS/SSL protection, or with an older version of TLS/SSL, are denied by default.
- The connection must use either the AES-128 or AES-256 symmetric cipher. The negotiated TLS connection cipher suite must support perfect forward secrecy (PFS) through Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange, and must be one of the following:

- TLS1.2 and later only
- Strong cryptography only
- At some compatibility cost

apple



Require Modern TLS

Only use modern versions (1.2 and 1.3) of the TLS protocol. These versions use more secure ciphers, but may restrict traffic to your site from older browsers.

cloudflare

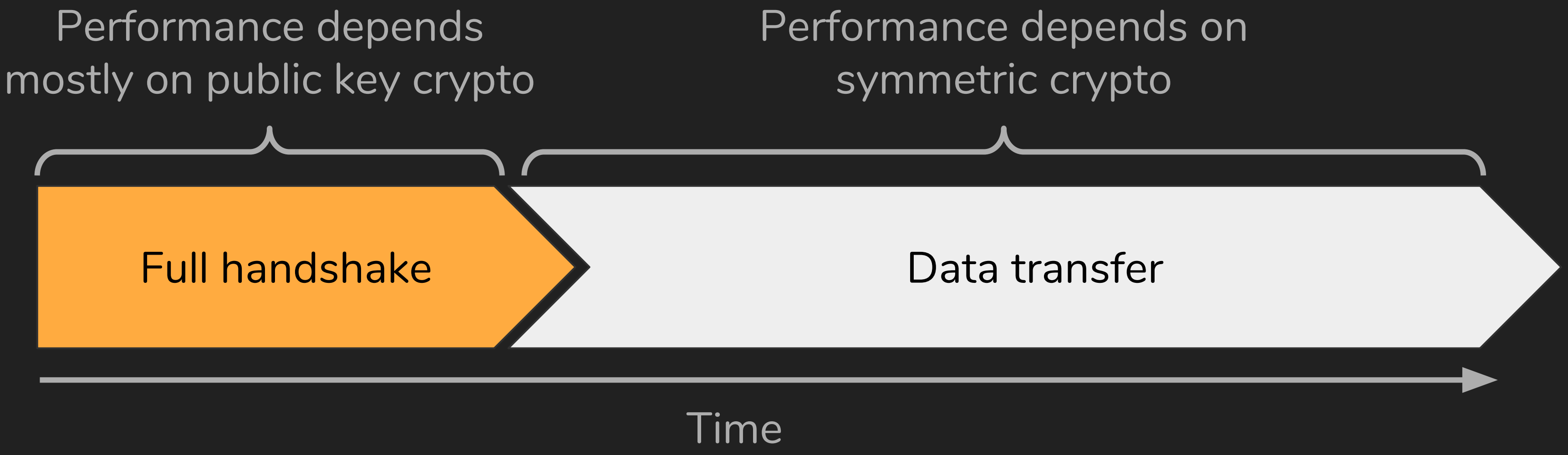
Testing

Automated testing:

- Integration tests against openssl and some public web servers
- Top-level API tests
- Unit tests of library internals
- 'bogo' - the BoringSSL test suite
- Performance benchmarks

Currently 97% line coverage

Performance

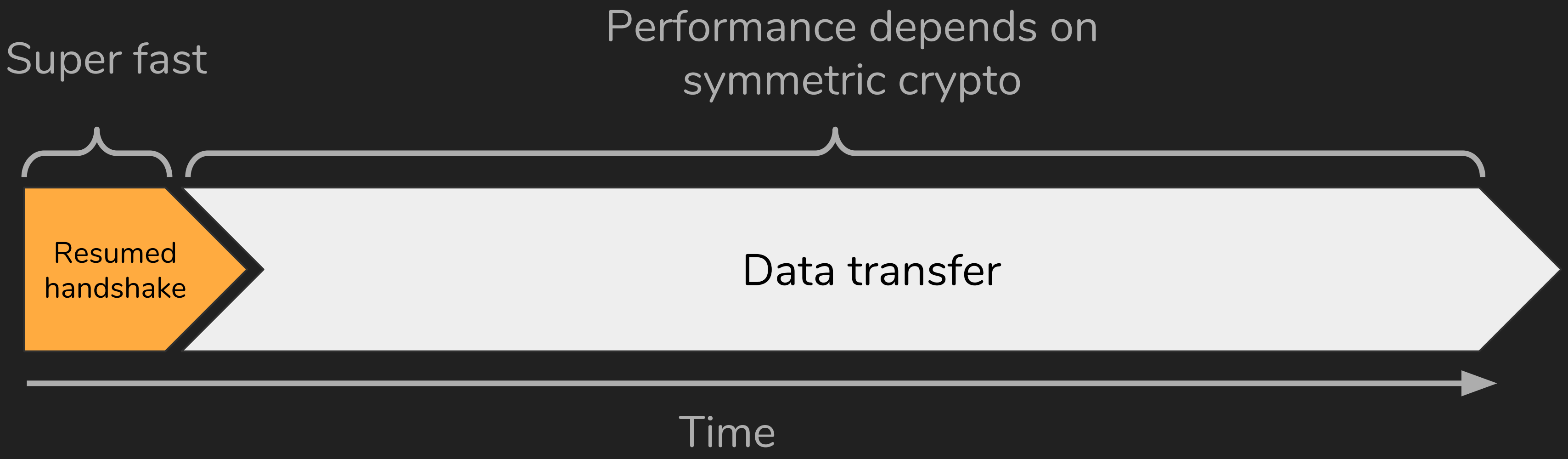


Performance depends mostly on public key crypto

Performance depends on symmetric crypto

Life of a TLS connection

Performance



Super fast

Performance depends on symmetric crypto

Resumed handshake

Data transfer

Time

Life of a TLS resumed connection

Performance

Data transfer

Direction	OpenSSL	rustls	vs.
Sending	3365.56 megabytes/sec	3591.31 megabytes/sec	+6.7%
Receiving	3738.02 megabytes/sec	3727.86 megabytes/sec	-0.3%

both libraries can saturate 25gbit ethernet with a single core
assuming no other overhead

Using ECDHE-RSA-AES128-GCM-SHA256, TLS1.2, per 3.20GHz i5-6500 core
https://jbp.io/2018/01/07/rustls-vs-openssl-performance-1.html

Performance

Resumed handshake

Direction	OpenSSL	rustls	vs.
Client	18905 conn/sec	28200 conn/sec	1.5x faster
Server	18933 conn/sec	25019 conn/sec	1.3x faster

Using session ID resumption, TLS1.2, per 3.20GHz i5-6500 core
Full results & writeup to come

Performance

Full handshake

Direction	OpenSSL	rustls	vs.
Client	1679 conn/sec	207 conn/sec	8x slower
Server	1175 conn/sec	690 conn/sec	1.7x slower

coming improvements to *ring* tested as giving ~6x speedup

TLS1.2, server authentication only, per 3.20GHz i5-6500 core
Full results & writeup to come

"goto fail"

The "goto fail" bug -- a security failure in Apple Secure Transport:

```
...
     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;

     err = sslRawVerify(ctx,
                        ctx->peerPubKey,
                        dataToSign,           /* plaintext */
                        dataToSignLen,        /* plaintext length */
                        signature,
                        signatureLen);
     if(err) {
...
```

"Marker types"

(please note: unlikely to be a new idea)

- How to make code robust against this kind of catastrophe?
- Problem is fundamentally: ***absence of an error is a poor indicator of signature validity***

This idea:

- Unique, zero-sized, behaviour-less, explicitly constructed type
- Represents **positive** outcome of signature verification
- Weave this type into protocol states

"Marker types"

In rustls:

- Protocol states after important verifications require values of these types
- This binds entering those states to the verification
- The compiler then checks we didn't skip verification somehow
- Code review task: are these types only constructed at precisely the right point?

Zero run-time cost

"Marker types"

```
let fin = constant_time::verify_slices_are_equal(&expect_verify_data, &finished.0)
    .map_err(|_| {
         sess.common.send_fatal_alert(AlertDescription::DecryptError);
         TLSError::DecryptError
    })
    .map(|_| verify::FinishedMessageVerified::assertion())?;


...


struct ExpectTLS12Traffic {
    _cert_verified: verify::ServerCertVerified,
    _sig_verified: verify::HandshakeSignatureValid,
    _fin_verified: verify::FinishedMessageVerified,
}
```

rustls in the future

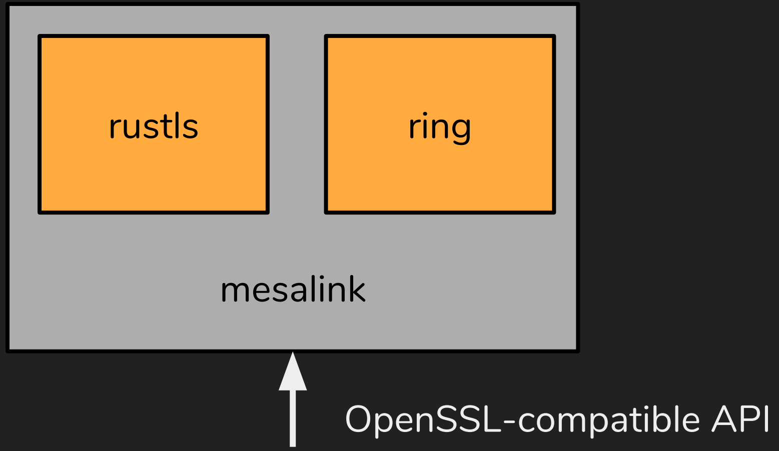
Write some glue for use from non-rust programs

rustls in the future

~~Write some glue for use from non-rust programs~~

MesaLink

https://github.com/mesalock-linux/mesalink



rustls in the future

Work on verification:

- upstream bug in Galois Inc. verification tools filed so they can process LLVM bitcode output by rustc
- aim to reuse verification from s2n (Amazon's in-house TLS library)
- this should show that rustls implements the TLS protocol faithfully

thanks

Repo:	https://github.com/ctz/rustls
Test server:	https://rustls.jbp.io/
Twitter:	@jpixton
Mail:	jbp@jbp.io
Slides:	https://github.com/ctz/talks