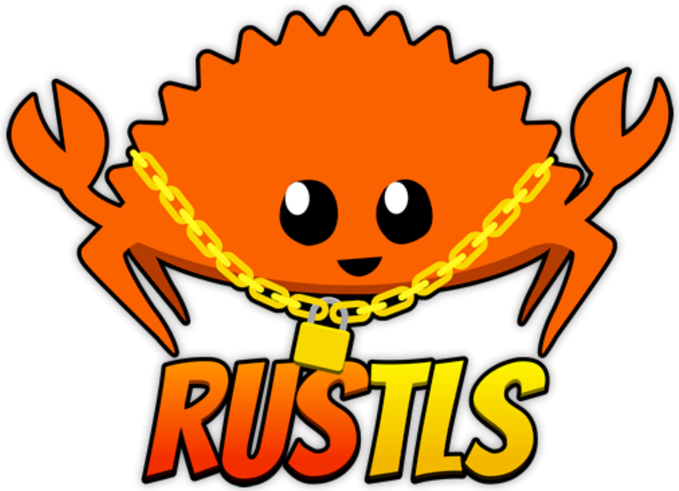
rustls: TLS in Rust



Dirkjan Ochtman, november 2022

Who am I?

- CTO at Instant Domains
- rustls co-maintainer since 2020
  - Worked on rustls on an ISRG contract for 6 months
- Maintainer of many other crates (chrono, trust-dns, indicatif, ...)
- Started working on rustls in 2018 in order to enable QUIC support
  - You may have heard of the Quinn crate as well

What is TLS?

- Transport Level Security
  - Version 1.3 was published as RFC 8446 in 2018
- Protocol used to protect TCP/IP traffic
  - Authentication: peer is the one you expected
  - Confidentiality: no one else can your data
  - Integrity: data cannot be modified without attackers
- Also used in QUIC

What is rustls?

- Rust implementation of the TLS protocol, started by Joe Pixton-Barr
  - Building on webpki, a web PKI certificate validation library (by Brian Smith)
  - And *ring*, core cryptography library also created by Brian
- Supports TLS 1.2 and TLS 1.3 (older versions are deprecated)
  - Does not support some older cipher suites
- Safe Rust only

Why use rustls? (1)

- Nov 2022: CVE-2022-3786 (High)
- Nov 2022: CVE-2022-3602 (High)
- Sep 2022: CVE-2022-3358 (Low)
- Jul 2022: CVE-2022-2274 (High)
- Jul 2022: CVE-2022-2097 (Moderate)
- Jun 2022: CVE-2022-2068 (Moderate)
- May 2022: CVE-2022-1473 (Low)
- May 2022: CVE-2022-1434 (Low)
- May 2022: CVE-2022-1343 (Moderate)
- May 2022: CVE-2022-1292 (Moderate)
- Mar 2022: CVE-2022-0778 (High)
- Jan 2022: CVE-2021-4160 (Moderate)

Why use rustls?

- It is more efficient (compared to OpenSSL, July 2019)
  - rustls is 15% quicker to send data
  - rustls is 5% quicker to receive data
  - rustls is 20-40% quicker to set up a client connection
  - rustls is 10% quicker to set up a server connection
  - rustls is 30-70% quicker to resume a client connection
  - rustls is 10-20% quicker to resume a server connection
  - rustls uses less than half the memory of OpenSSL

Type state: configuration (1)

```
ServerConfig::builder()
    .with_safe_default_cipher_suites()
    .with_safe_default_kx_groups()
    .with_safe_default_protocol_versions()
    .unwrap()
    .with_no_client_auth()
    .with_single_cert(certs, private_key)
    .expect("bad certificate/key");
```

This may be shortened to:

```
ServerConfig::builder()
    .with_safe_defaults()
    .with_no_client_auth()
    .with_single_cert(certs, private_key)
    .expect("bad certificate/key");
```

```
ClientConfig::builder()
    .with_safe_default_cipher_suites()
    .with_safe_default_kx_groups()
    .with_safe_default_protocol_versions()
    .unwrap()
    .with_root_certificates(root_certs)
    .with_single_cert(certs, private_key)
    .expect("bad certificate/key");
```

This may be shortened to:

```
ClientConfig::builder()
    .with_safe_defaults()
    .with_root_certificates(root_certs)
    .with_no_client_auth();
```

Type state:

configuration

(2)

```
impl<S: ConfigSide> ConfigBuilder<S, WantsCipherSuites>                    source

pub fn with_safe_defaults(self) -> ConfigBuilder<S, WantsVerifier>         source
```

Start side-specific config with defaults for underlying cryptography.

If used, this will enable all safe supported cipher suites (`DEFAULT_CIPHER_SUITES`), all safe supported key exchange groups (`ALL_KX_GROUPS`) and all safe supported protocol versions (`DEFAULT_VERSIONS`).

These are safe defaults, useful for 99% of applications.

```
pub fn with_cipher_suites(                                                 source
    self,
    cipher_suites: &[SupportedCipherSuite]
) -> ConfigBuilder<S, WantsKxGroups>
```

Choose a specific set of cipher suites.

```
pub fn with_safe_default_cipher_suites(self) -> ConfigBuilder<S, WantsKxGroups>   source
```

Choose the default set of cipher suites (`DEFAULT_CIPHER_SUITES`).

Note that this default provides only high-quality suites: there is no need to filter out low-, export- or NULL-strength cipher suites: rustls does not implement these.

```
§impl<S: ConfigSide> ConfigBuilder<S, WantsKxGroups>                       source

pub fn with_kx_groups(                                                     source
    self,
    kx_groups: &[&'static SupportedKxGroup]
) -> ConfigBuilder<S, WantsVersions>
```

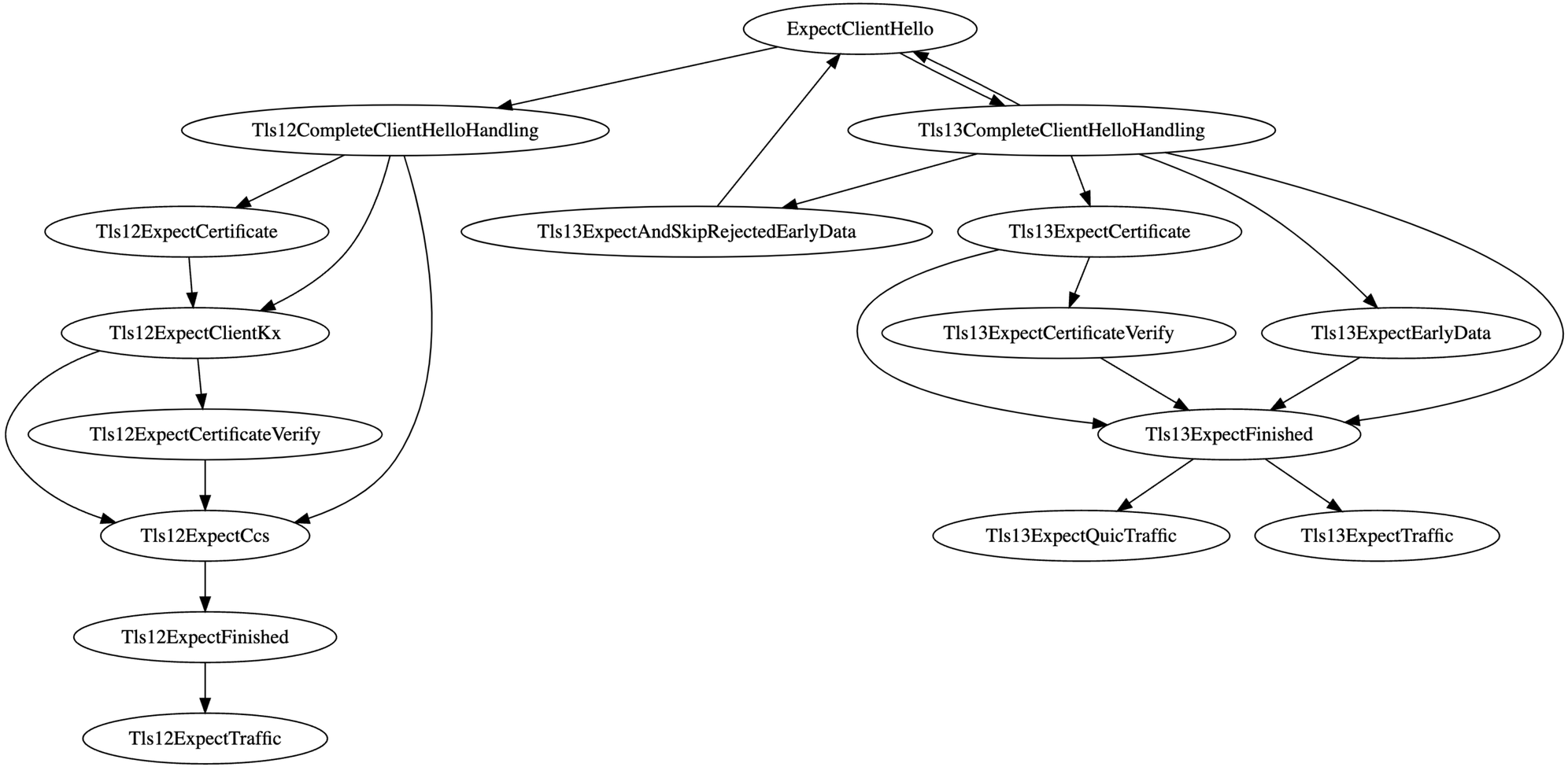
Choose a specific set of key exchange groups.

```
pub fn with_safe_default_kx_groups(self) -> ConfigBuilder<S, WantsVersions>   source
```

Choose the default set of key exchange groups (`ALL_KX_GROUPS`).

This is a safe default: rustls doesn't implement any poor-quality groups.

Type state: handshake



State machine

```rust
pub(crate) trait State<Data>: Send + Sync {
    fn handle(
        self: Box<Self>,
        cx: &mut Context<'_, Data>,
        message: Message<'_>,
    ) -> Result<Box<dyn State<Data>>, Error>;

    fn export_keying_material(
        &self,
        _output: &mut [u8],
        _label: &[u8],
        _context: Option<&[u8]>,
    ) -> Result<(), Error> {
        Err(Error::HandshakeNotComplete)
    }

    #[cfg(feature = "secret_extraction")]
    fn extract_secrets(&self) -> Result<PartiallyExtractedSecrets, Error> {
        Err(Error::HandshakeNotComplete)
    }

    fn perhaps_write_key_update(&mut self, _cx: &mut CommonState) {}
}
```

0 implementations

```rust
pub(crate) struct Context<'a, Data> {
    pub(crate) common: &'a mut CommonState,
    pub(crate) data: &'a mut Data,
}
```

Type state: 1.3 key schedule

- KeyScheduleEarly
- KeySchedulePreHandshake
- KeyScheduleHandshakeStart
- KeyScheduleHandshake
- KeyScheduleTrafficWithClientFinishedPending
- KeyScheduleTraffic

"Type confusion"

- Handshake states
- Connection state
- Cipher suite states

Certificate verification

- webpki maintenance
  - rustls/webpki
- rustls-native-certs
- webpki-roots
- New: rustls-platform-verifier

Why not use rustls?

- No IP addresses in certificates
- Portability
  - *ring* maintenance?
- Platform verification
- Support for TLS 1.1 and older

Future work

- IP address support
- Decrease memory usage/allocations
- Improve session cache API
- Provide access to more client certificate details
- Encrypted ClientHello
- FIPS?

Questions?