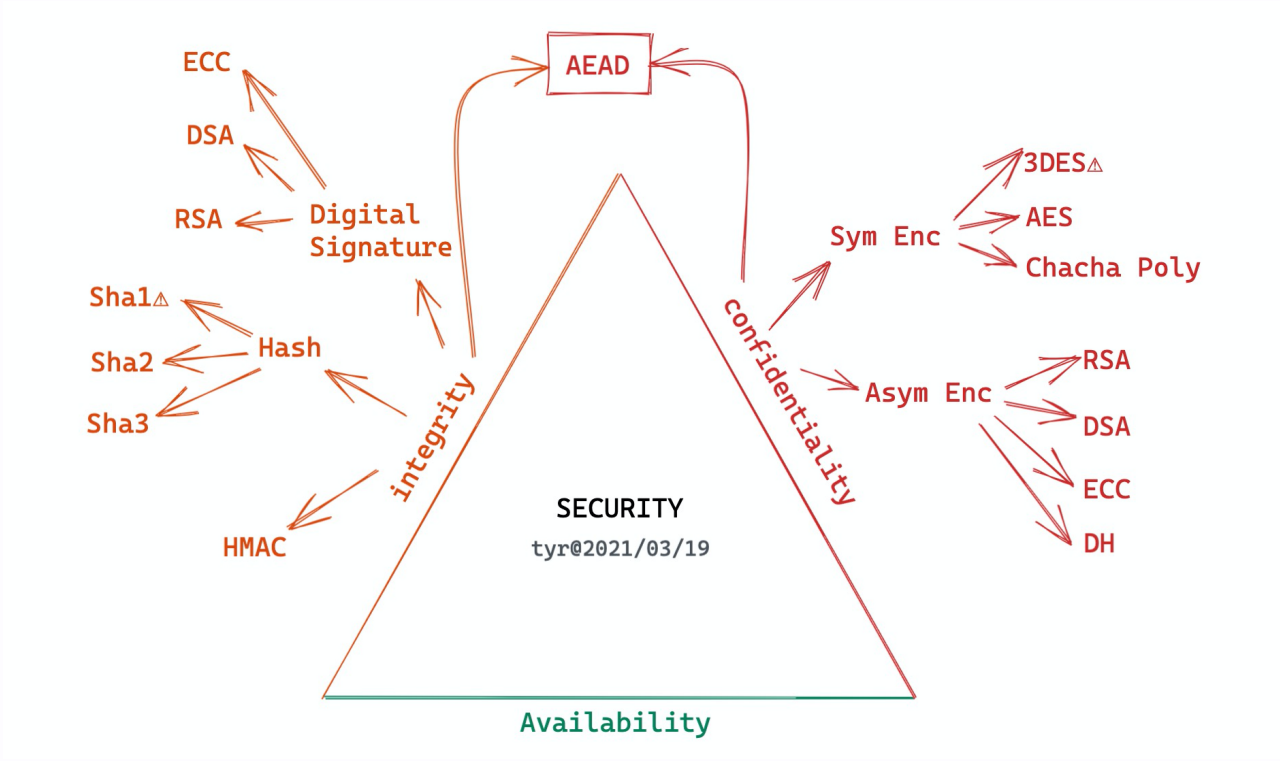
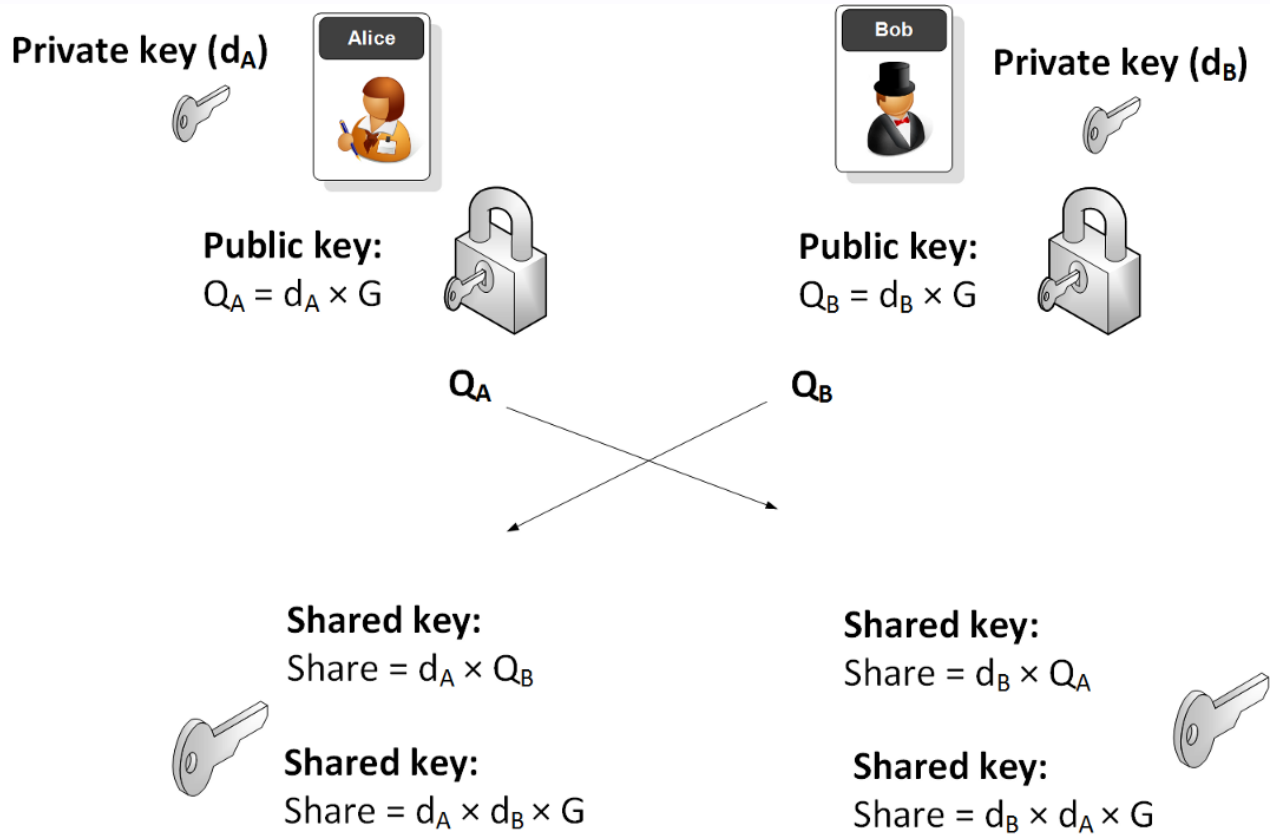
Rust: 构建安全高性能的网络应用

当我们谈论网络安全的时候，我们在谈论什么？



应用层安全

- 使用标准协议 - TLSv1.3
- 构建你自己的安全协议 - Noise Protocol
- 应用层安全的基石：DH 算法



Rust TLS 支持

- openssl
- rustls (基于 ring)
- tokio-tls-helper

配置

- 客户端：domain，CA cert
- 服务器：cert / key

```
# client configuration

domain = "localhost"

[cert]
pem = """-----BEGIN CERTIFICATE-----
MIIBeTCCASugAwIBAgIBKjAFBgMrZXAwNzELMAkGA1UEBgwCVVMxFDASBgNVBAoM
C0RvbWFpbiBJbmMuMRIwEAYDVQQDDAlEb21haW4gQ0EwHhcNMjEwMzE0MTg0NTU2
WhcNMzEwMzEyMTg0NTU2WjA3MQswCQYDVQQGDAJVUzEUMBIGA1UECgwLRG9tYWlu
IEluYy4xEjAQBgNVBAMMCURvbWFpbiBDQTAqMAUGAytlcAMhAAzhorM9IPsXjBTx
ZxykGl5xZrsj3X2XqKjaAVutnf7po1wwWjAUBgNVHREEDTALgglsb2NhbGhvc3Qw
HQYDVR0OBBYEFD+NqChBZDOs5FMgefHJSIWiRTHXMBIGA1UdEwEB/wQIMAYBAf8C
ARAwDwYDVR0PAQH/BAUDAwcGADAFBgMrZXADQQA9sligQcYGaBqTxR1+JadSelMK
Wp35+yhVVuu4PTL18kWdU819w3cVlRe/GHt+jjlbk1i22TvfO5AaNmdxySkO
-----END CERTIFICATE-----"""

# server configuration

[identity]
key = """-----BEGIN PRIVATE KEY-----
MFMCAQEwBQYDK2VwBCIEII0kozd0PJsbNfNUS/oqI/Q/enDiLwmdw+JUnTLpR9xs
oSMDIQAtkhJiFdF9SYBIMcLikWPRIgca/Rz9ngIgd6HuG6HI3g==
-----END PRIVATE KEY-----"""

[identity.cert]
pem = """-----BEGIN CERTIFICATE-----
MIIBazCCAR2gAwIBAgIBKjAFBgMrZXAwNzELMAkGA1UEBgwCVVMxFDASBgNVBAoM
C0RvbWFpbiBJbmMuMRIwEAYDVQQDDAlEb21haW4gQ0EwHhcNMjEwMzE0MTg0NTU2
WhcNMjIwMzE0MTg0NTU2WjA5MQswCQYDVQQGDAJVUzEUMBIGA1UECgwLRG9tYWlu
IEluYy4xFDASBgNVBAMMC0dSUEMgU2VydmVyMCowBQYDK2VwAyEALZISYhXRfUmA
SDHC4pFj0SIHGv0c/Z4CIHeh7huhyN6jTDBKMBQGA1UdEQQNMAuCCWxvY2FsaG9z
dDATBgNVHSUEDDAKBggrBgEFBQcDATAMBgNVHRMEBTADAQEAMA8GA1UdDwEB/wQF
AwMH4AAwBQYDK2VwA0EAy7EOIZp73XtcqaSopqDGWU7Umi4DVvIgjmY6qbJZP0sj
ExGdaVq/7MOlZl1I+vY7G0NSZWZIUilX0CoOkrn0DA==
-----END CERTIFICATE-----"""
```

代码

- 服务器
  - 加载配置 `ServerTlsConfig`
  - 准备好 `TLS acceptor`
  - `acceptor.accept(tcp_stream)`
- 客户端
  - 加载配置 `ClientTlsConfig`
  - 准备好 `TLS connector`
  - `connector.connect(tcp_stream)`

Server:

```rust
// you could also build your config with cert and identity separately. See tests.
let config: ServerTlsConfig = toml::from_str(config_file).unwrap();
let acceptor = config.tls_acceptor().unwrap();
let listener = TcpListener::bind(addr).await.unwrap();
tokio::spawn(async move {
    loop {
        let (stream, peer_addr) = listener.accept().await.unwrap();
        let stream = acceptor.accept(stream).await.unwrap();
        info!("server: Accepted client conn with TLS");

        let fut = async move {
            let (mut reader, mut writer) = split(stream);
            let n = copy(&mut reader, &mut writer).await?;
            writer.flush().await?;
            debug!("Echo: {} - {}", peer_addr, n);
        }

        tokio::spawn(async move {
            if let Err(err) = fut.await {
                error!("{:?}", err);
            }
        });
    }
});
```

Client:

```rust
let msg = b"Hello world\n";
let mut buf = [0; 12];

// you could also build your config with cert and identity separately. See tests.
let config: ClientTlsConfig = toml::from_str(config_file).unwrap();
let connector = config.tls_connector(Uri::from_static("localhost")).unwrap();

let stream = TcpStream::connect(addr).await.unwrap();
let mut stream = connector.connect(stream).await.unwrap();
info!("client: TLS conn established");

stream.write_all(msg).await.unwrap();

info!("client: send data");

let (mut reader, _writer) = split(stream);

reader.read_exact(buf).await.unwrap();

info!("client: read echoed data");
```

Noise Protocol

- TLS vs Noise protocol：动态协商 vs 静态协商
- Noise_IKpsk2_25519_ChaChaPoly_BLAKE2s：
  - I：发起者的固定公钥未加密就直接发给应答者
  - K：应答者的公钥发起者预先就知道
  - psk2：把预设的密码（Pre-Shared-Key ）放在第 2 个握手包之后
  - ChaChaPoly：对称加密算法
  - BLAKE2s：哈希算法
- 协议最少 0-RTT (x 或者 xpsk），之后就建立好加密通道，可以发送数据

Noise Protocol 接口

- build：根据协议变量和固定私钥，初始化 HandshakeState
- write(msg, buf): 根据当前的状态，撰写协议报文或者把用户传入的 buffer 加密
- read(buf, msg)：根据当前的状态，读取用户传入的 buffer，处理握手状态机或者把用户传入的 buffer 解密
- into_transport_mode：将 HandshakeState 转为 CipherState
- rekey：在传输模式下，用户可以调用 rekey 来更新密钥

代码（0-RTT）

- Initiator:
  - 构建 `HandshakeState`
  - 发送握手数据
  - 进入传输模式
- Responder:
  - 构建 `HandshakeState`
  - 接收握手数据
  - 进入传输模式

```rust
pub fn new(config: SessionConfig) -> Result<Self, ConcealError> {
    let mut header: Header = config.header;
    let noise_params: NoiseParams = header.to_string().parse()?;
    // in handshake mode this should be enough
    let mut buf: [u8; _] = [0u8; 256];

    if header.handshake_message.is_empty() {
        // initiator
        let mut noise: HandshakeState = if !header.use_psk {
            Builder::new(noise_params): Builder
                .remote_public_key(pub_key: &config.rs.unwrap()): Builder
                .local_private_key(&config.keypair.private): Builder
                .build_initiator()?
        } else {
            Builder::new(noise_params): Builder
                .remote_public_key(pub_key: &config.rs.unwrap()): Builder
                .local_private_key(&config.keypair.private): Builder
                .psk(location: 1, key: &config.psk.unwrap()): Builder
                .build_initiator()?
        };

        let len: usize = noise.write_message(payload: &[0u8; 0], message: buf.as_mut())?;
        let handshake_message: Vec<u8> = buf[..len].to_vec();
        header.handshake_message = handshake_message;
        let state: TransportState = noise.into_transport_mode()?;
        Ok(Self { state, header })
    } else {
        // responder
        let mut noise: HandshakeState = if !header.use_psk {
            Builder::new(noise_params): Builder
                .local_private_key(&config.keypair.private): Builder
                .build_responder()?
        } else {
            Builder::new(noise_params): Builder
                .local_private_key(&config.keypair.private): Builder
                .psk(location: 1, key: &config.psk.unwrap()): Builder
                .build_responder()?
        };
        let _len: usize = noise.read_message(&header.handshake_message, payload: &mut buf)?;
        let state: TransportState = noise.into_transport_mode()?;
        Ok(Self { state, header })
    }
}
```

如何 安全 地，确定性 地生成

密码/密钥/证书?

Cellar

- 受 Bitcoin HD wallet 启发（BIP-32 Hierachical Deterministic Wallets）
- 密码可以从一个根密码一层层派生下去
- 每个父密码可以派生子密码，结果跟从根密码派生一样

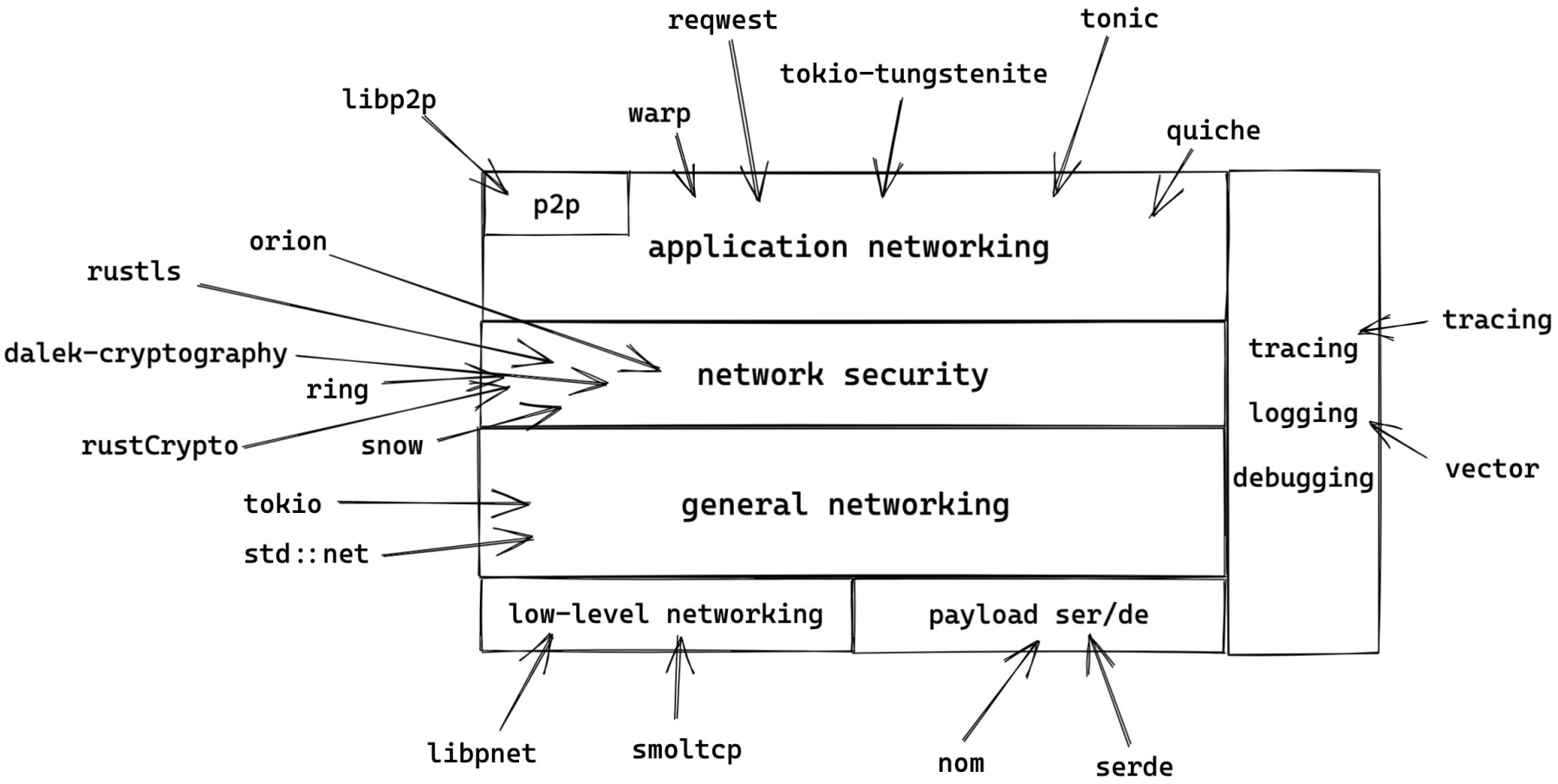
```
salt            = Secure-Random(output_length=32)
stretched_key   = Argon2(passphrase=user_passphrase, salt=salt)

auth_key        = HMAC-BLAKE2s(key=stretched_key, "Auth Key")
c1              = HMAC-BLAKE2s(key=stretched_key, "Master Key")
c2              = Secure-Random(output_length=32)
encrypted_c2    = ChaCha20(c2, key=auth_key, nonce=salt[0..CHACHA20_NONCE_LENGTH])

master_key      = HMAC-BLAKE2s(key=c1, c2)
application_key = HMAC-BLAKE2s(key=master_key, "app info, e.g. yourname@gmail.com")
```

Cellar 代码

- 生成密码或 Ed25519 密钥
- 生成 x509 证书

```rust
#[test]
Debug
fn generate_key_by_path_should_work() -> Result<(), CellarError> {
    let passphrase: &str = "hello";
    let aux: AuxiliaryData = init(passphrase)?;
    let key: Zeroizing<[u8; _]> = generate_master_key(passphrase, &aux)?;
    let parent_key: Vec<u8> = generate_app_key(passphrase, &aux, info: b"apps", KeyType::Password)?;
    let app_key: Vec<u8> = generate_app_key_by_path(parent_key: key, path: "apps/my/awesome/key", KeyType::Password)?;
    let app_key1: Vec<u8> = generate_app_key_by_path(
        parent_key: as_parent_key(app_key: &parent_key),
        path: "my/awesome/key",
        KeyType::Password,
    )?;
    assert_eq!(app_key, app_key1);
    Ok(())
}

▶ Run Test | Debug
#[test]
Debug
fn generate_ca_cert_should_work() -> Result<(), CellarError> {
    let info: CertInfo = CertInfo::new(domains: &["localhost"], ips: &[], country: "US", org: "Domain Inc.", cn: "Domain CA", days: None);
    let (_, parent_key: Vec<u8>, cert_pem: CertificatePem) = generate_ca(info.clone())?;

    load_ca(&cert_pem.cert, key: &cert_pem.sk)?;

    let cert1: Vec<u8> = generate_app_key_by_path(
        parent_key: as_parent_key(app_key: &parent_key),
        path: "localhost/ca",
        KeyType::CA(info),
    )?;

    let cert_pem1: CertificatePem = bincode::deserialize(bytes: &cert1)?;

    assert_eq!(&cert_pem.sk, &cert_pem1.sk);
    assert_eq!(&cert_pem.cert, &cert_pem1.cert);

    Ok(())
}

fn generate_ca(info: CertInfo) -> Result<(Key, Vec<u8>, CertificatePem), CellarError> {
    let passphrase: &str = "hello";
    let aux: AuxiliaryData = init(passphrase)?;
    let key: Zeroizing<[u8; _]> = generate_master_key(passphrase, &aux)?;
    let parent_key: Vec<u8> = generate_app_key(passphrase, &aux, info: b"apps", KeyType::Password)?;

    let cert: Vec<u8> = generate_app_key_by_path(parent_key: key.clone(), path: "apps/localhost/ca", KeyType::CA(info))?;
    let cert_pem: CertificatePem = bincode::deserialize(bytes: &cert)?;
    Ok((key, parent_key, cert_pem))
}
    You, seconds ago • Uncommitted changes
```

构建高性能安全的网络


reqwest
tonic
tokio-tungstenite
libp2p
warp
quiche
p2p
application networking
orion
rustls
dalek-cryptography
ring
rustCrypto
snow
network security
tracing
tracing
logging
debugging
vector
tokio
std::net
general networking
low-level networking
payload ser/de
libpnet
smoltcp
nom
serde

参考资料

- tokio tls helper
- Noise 框架：构建安全协议的蓝图
- Cellar: 分层确定性密钥管理
- Conceal：使用 Noise protocol 做文件加密

May the Rust be with you