Funded by the European Union under Grant Agreement No. 101087529. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Co-funded by the European Union

Quantum-Resistant Security Libraries

CHESS

Cyber-security Excellence Hub in Estonia and South Moravia

Workshop on Post-Quantum Cryptography: Principles and Challenges

27.3.2025, FEKT VUT, Brno

Petr Muzikant

Information Security Research Institute @ Cybernetica AS, Estonia

CYBERNETICA

Public

Introduction - Who Am I?

- Local Information-Security graduate
- Security Engineer
  - focus on PQC implementation and engineering issues
- **Estonian e-government systems** (PoCs)
  - PQ Web-eID (authentication framework)
  - PQ CDOC2 (encryption framework)
  - PQ eID / PKI (CA, OCSP, TSA)
  - PQ IVXV (internet voting framework)
  - supporting side-projects

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*
  - *standards*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*
  - *standards*
  - *hybrid modes, key combiners*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*
  - *standards*
  - *hybrid modes, key combiners*
  - *crypto agility*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
    - *confidentiality first*
    - *standards*
    - *hybrid modes, key combiners*
    - *crypto agility*
    - *pre-hashing dilemma, externalMu modes*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*
  - *standards*
  - *hybrid modes, key combiners*
  - *crypto agility*
  - *pre-hashing dilemma, externalMu modes*
  - *interoperability issues*

Assumptions and Preliminary Topics

- *Developer/Engineer who aims to make an application quantum safe*
  - *confidentiality first*
  - *standards*
  - *hybrid modes, key combiners*
  - *crypto agility*
  - *pre-hashing dilemma, externalMu modes*
  - *interoperability issues*
- **PQC migration is not a simple switch of algorithms / libraries!**
  - **But,** you don't need to know these topics for innocent experimenting

Migration Steps

- *Made the decision*
  - *→ PQC migration plan*
    - *→ crypto discovery/inventory*
      - *→ preliminary analysis of PQC impact*
- Now what?
  - **Start implementing PQC** (*in SW*)
  - Two options:
    - Upgrade existing cryptography libraries
    - Introduce new dependency for PQ-only library

Pure PQ Libraries

One simple note / use case

https://github.com/PQClean/PQClean

PQClean (C)

- Cleaned aggregation of NIST (to-be) standards
  - Unified API and code style
  - Platform-specific optimizations
  - All submitted algorithms at one place
- **Source of source-code** (i.e. not a library)
- No security guarantees

Embedded devices, single algorithm usage https://github.com/open-quantum-safe/liboqs

libOQS (C)

- **Most well-maintained library** out there
- Language wrappers:
  - C++, Python, Java, Go, .NET, Rust, and PHP
- Applications built with libOQS
  - **OpenSSL**, OpenSSH, OpenVPN forks
- No security guarantees
  - Common choice for companies? (including Amazon, Meta, ...)

Default choice for almost everything

https://github.com/cloudflare/circl

CIRCL (Go)

- Developed, maintained, and used by Cloudflare
- Offers same interfaces as `go/crypto` library
- Cloudflare has high influence on upcoming standards
- Joy to pick-up and use :)
- No security guarantees

Great integration with Go applications

Other random projects you might find

- https://github.com/rustpq/pqcrypto (Rust)
  - Bindings to PQClean
- https://github.com/paulmillr/noble-post-quantum (JS)
  - Claims high security
- https://github.com/mupq/pqm4 (C)
  - Optimized for ARM Cortex-M4

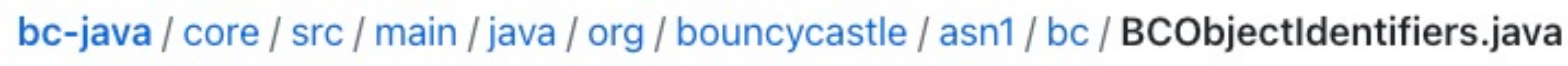
High-assurance Implementations

- PQC Alliance → **PQ Code Package** (https://github.com/pq-code-package)
  - Promises high security guarantees
  - Not so much traction yet
- Formosa Crypto → **Libjade** (https://formosa-crypto.org/tools/libjade)
  - Specially crafted toolboxes for high-assurance and quality code
  - *"If we should do cryptography again, we should make it right"*
- **KyberLib** (https://kyberlib.com)
  - Claims strong security guarantees in Rust

If you need PQC in real-world product: **Libjade**

PQC Support In Existing Libraries

BouncyCastle (Java)

- Not well documented
  - 
    bc-java / core / src / main / java / org / bouncycastle / asn1 / bc / BCObjectIdentifiers.java
  - `org.bouncycastle.pqc.*` packages
- Different workflow from others
- Useful "Java Keytool" benefits too

If you are heavily integrated in Java ecosystem

PQ Java Keytool

- keytool = command for managing a keystore of cryptographic objects
- PQ BouncyCastle → PQ Java Keytool
- e.g. to generate .p12 with Dilithium keypair and self-signed certificate:

```
keytool \
    -providerpath bcprov-jdk18on-175.jar \
    -provider org.bouncycastle.pqc.jcajce.provider.BouncyCastlePQCProvider \
    -genkeypair \
    -keyalg Dilithium5 \
    -alias cdoc20-client-pqc-CA \
    -keystore cdoc20clientpqcCA.p12 \
    -storepass passwd \
    -sigalg Dilithium5 \
    -dname "CN=cdoc20-client-pqc-CA,OU=ISRI,O=CyberneticaAS,L=Brno,S=Czechia,C=CZ"
```

Go/crypto, Python/cryptography, Botan

- go/crypto
  - **Only hybrid TLS by default**
  - ML-KEM implementation is internal
- python/cryptography
  - Depends on OpenSSL
  - No intention to develop anything until OpenSSL is mature
- Botan
  - PQC included

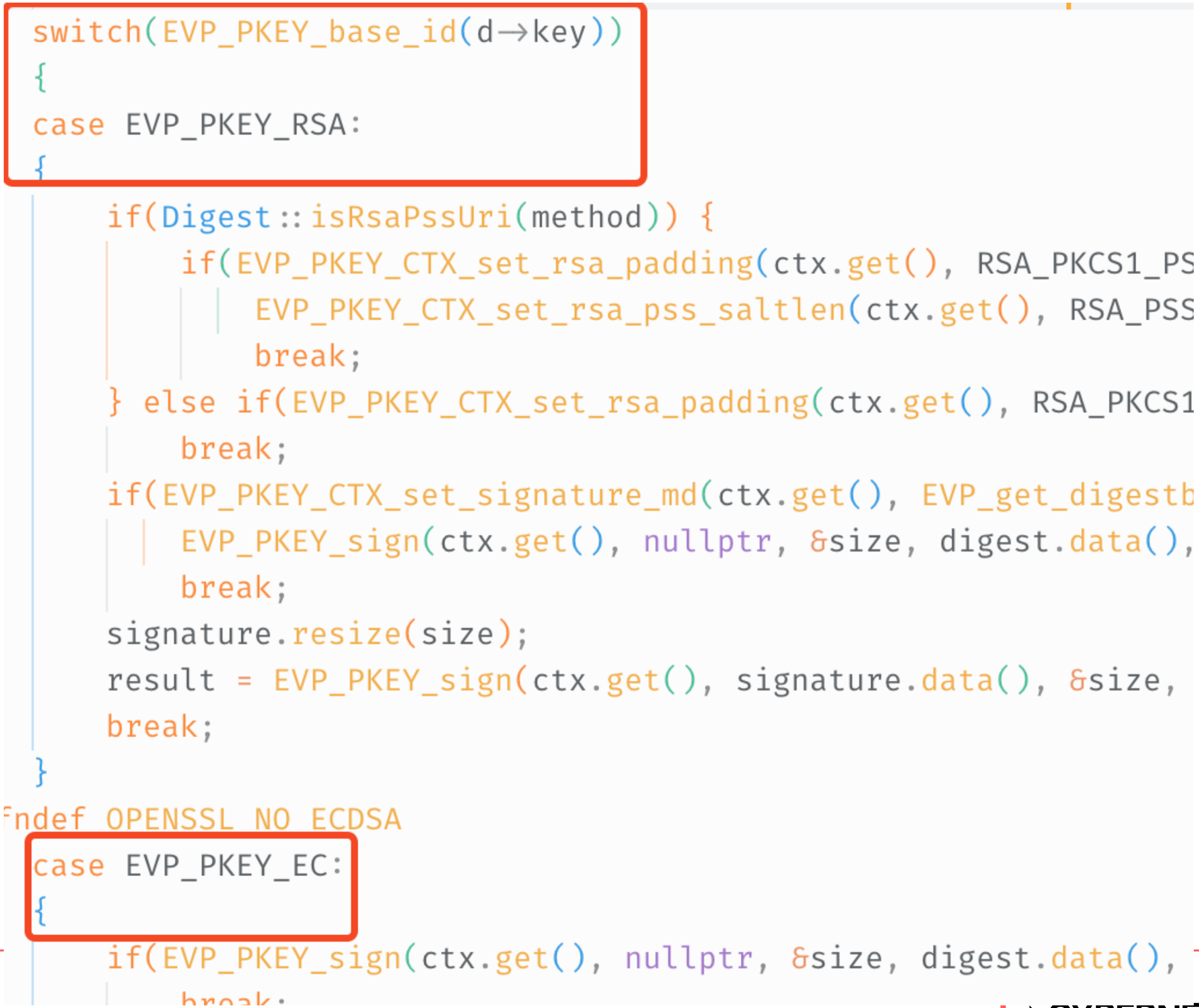
go/crypto: if you just need TLS in Go, **Botan:** if you used it before https://github.com/open-quantum-safe/oqs-provider

OpenSSL

- OpenQuantumSafe - **oqs-provider**
  - integrates **libOQS into OpenSSL v3+**
  - → TLS, SSH, certificates, CA, OCSP, TSA, basically everything
- Michael Baentsch (OQS maintainer and committee member) helps OpenSSL to introduce PQC

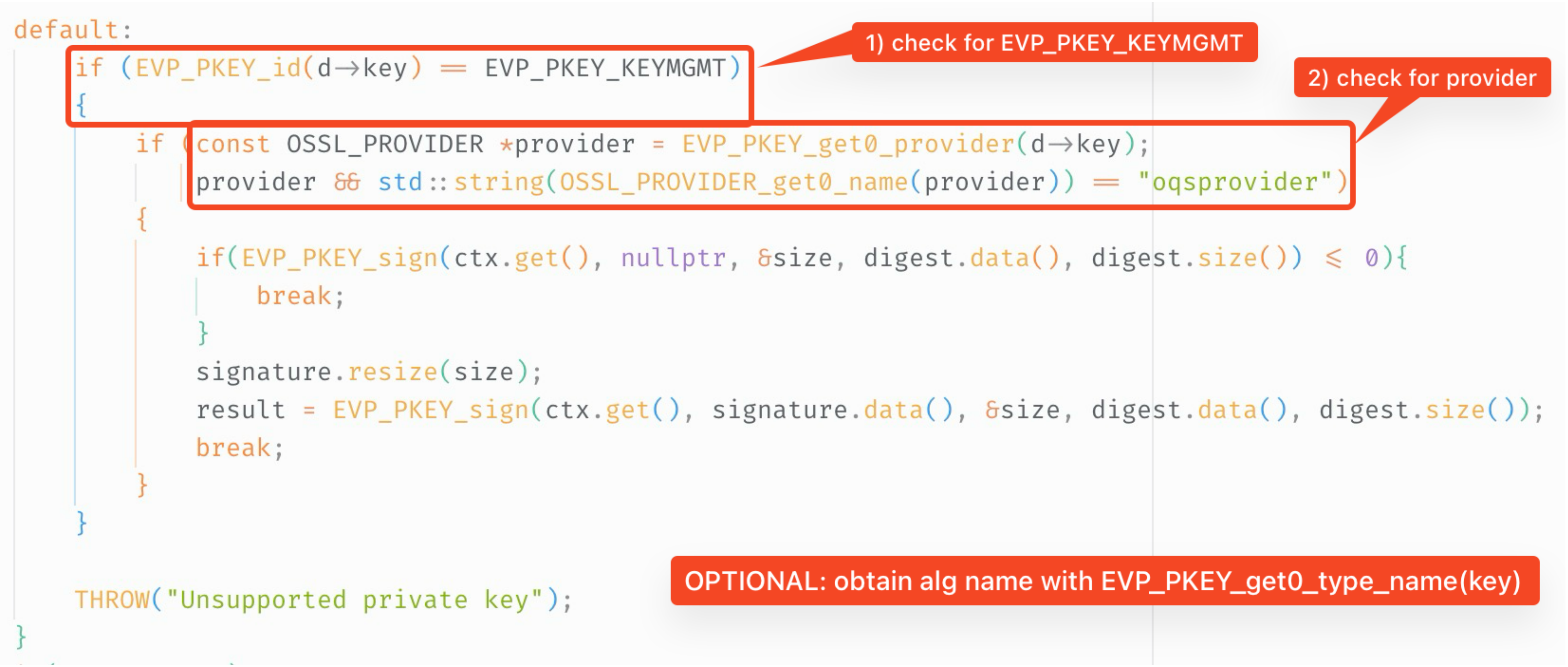
For high-level applications

PQ OpenSSL



```
switch(EVP_PKEY_base_id(d->key))
{
case EVP_PKEY_RSA:
{
    if(Digest::isRsaPssUri(method)) {
        if(EVP_PKEY_CTX_set_rsa_padding(ctx.get(), RSA_PKCS1_PS
            EVP_PKEY_CTX_set_rsa_pss_saltlen(ctx.get(), RSA_PSS
            break;
    } else if(EVP_PKEY_CTX_set_rsa_padding(ctx.get(), RSA_PKCS1
        break;
    if(EVP_PKEY_CTX_set_signature_md(ctx.get(), EVP_get_digestb
        EVP_PKEY_sign(ctx.get(), nullptr, &size, digest.data(),
        break;
    signature.resize(size);
    result = EVP_PKEY_sign(ctx.get(), signature.data(), &size,
    break;
}
fndef OPENSSL_NO_ECDSA
case EVP_PKEY_EC:
{
    if(EVP_PKEY_sign(ctx.get(), nullptr, &size, digest.data(),
        break;
```

PQ OpenSSL



```
default:
    if (EVP_PKEY_id(d->key) == EVP_PKEY_KEYMGMT)
    {
        if (const OSSL_PROVIDER *provider = EVP_PKEY_get0_provider(d->key);
            provider && std::string(OSSL_PROVIDER_get0_name(provider)) == "oqsprovider")
        {
            if(EVP_PKEY_sign(ctx.get(), nullptr, &size, digest.data(), digest.size()) <= 0){
                break;
            }
            signature.resize(size);
            result = EVP_PKEY_sign(ctx.get(), signature.data(), &size, digest.data(), digest.size());
            break;
        }
    }

    THROW("Unsupported private key");
}
```

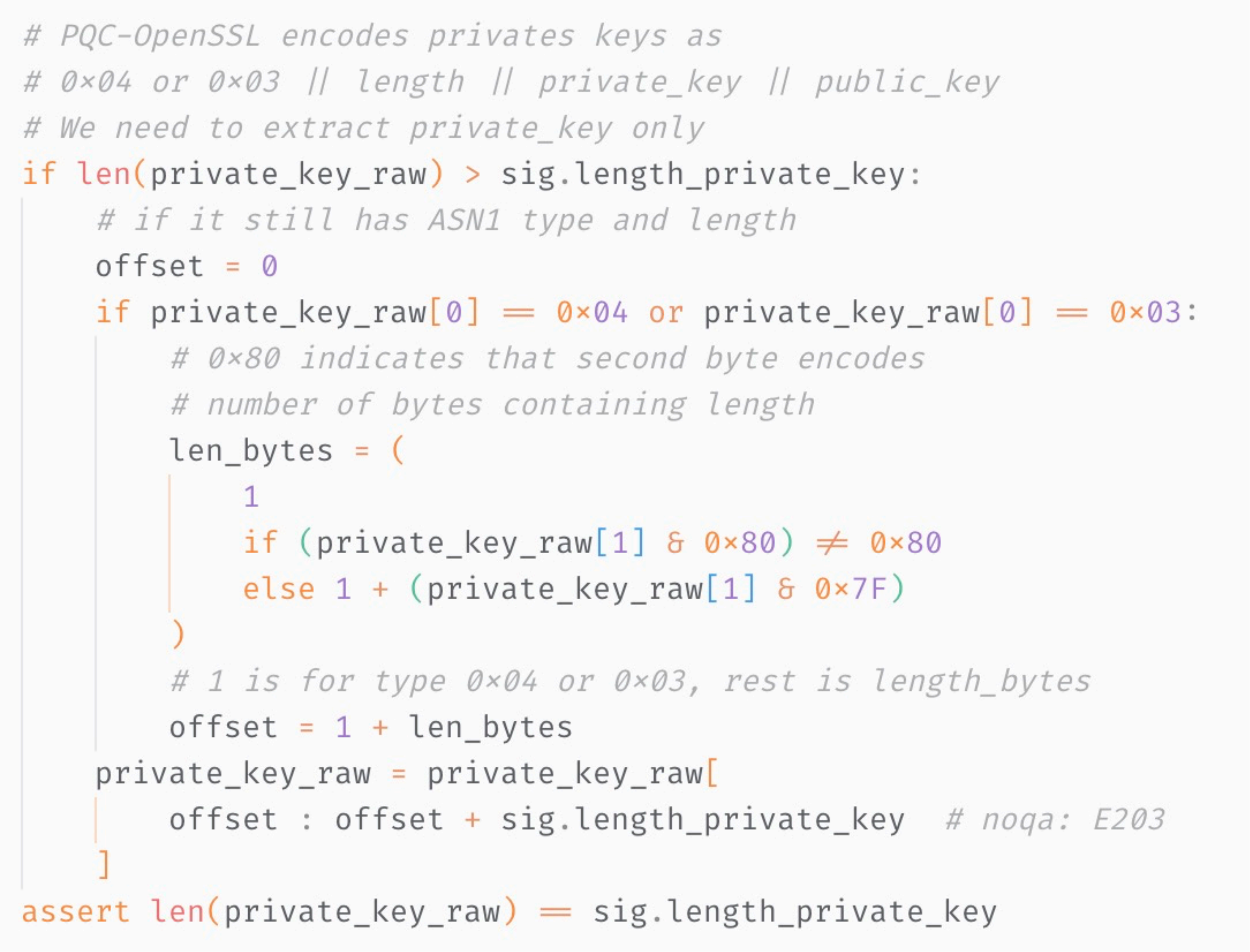
1) check for EVP_PKEY_KEYMGMT

2) check for provider

OPTIONAL: obtain alg name with EVP_PKEY_get0_type_name(key)

PQ OpenSSL
Private key encoding

- OpenSSL outputs the private key as:
    - **privateKey || publicKey**
- This concatenated format is put into the **PrivateKeyInfo** structure



```
# PQC-OpenSSL encodes privates keys as
# 0x04 or 0x03 || length || private_key || public_key
# We need to extract private_key only
if len(private_key_raw) > sig.length_private_key:
    # if it still has ASN1 type and length
    offset = 0
    if private_key_raw[0] == 0x04 or private_key_raw[0] == 0x03:
        # 0x80 indicates that second byte encodes
        # number of bytes containing length
        len_bytes = (
            1
            if (private_key_raw[1] & 0x80) != 0x80
            else 1 + (private_key_raw[1] & 0x7F)
        )
        # 1 is for type 0x04 or 0x03, rest is length_bytes
        offset = 1 + len_bytes
    private_key_raw = private_key_raw[
        offset : offset + sig.length_private_key  # noqa: E203
    ]
assert len(private_key_raw) == sig.length_private_key
```

https://developers.google.com/tink

Bonus: Google's Tink

- Shift in how we understand cryptography
  - Focus on **cryptographic agility and key rotation**
  - Data models are **revolving around the keys, not algorithms**
- *"Think in terms of keys and primitives, not algorithms"*
- Lot of articles on PQC
- Google will ONLY use Tink and BoringSSL from now

If you want to be the cool kid

Bonus: standards

- (BSI, ANSSI, NÚKIB, etc: recommendations and guidelines)
- NIST: algorithms
- IETF: internet protocols (certificates, PKI,T ASN.1 structures)
- ETSI: digital signature legal singing

Standards are useful, libraries will depend on them

Conclusions

- Plenty of options
- **PQC migration is a process** → be interested in it
  - Monitor news (https://groups.google.com/a/list.nist.gov/g/pqc-forum)
  - Watch recordings of conferences
    - PKI Consortium PQC Conference
    - Real World PQC
    - NIST PQC Standardization Conference

Great opportunity to learn about applied cryptography in general

Conclusions

- **Prototype stuff**
  - Pilot project / proof-of-concept is referenced a lot as a great approach
- **Lot of opportunities to help**
  - PQC open-source is quite welcoming community
  - Raise issues, ask about edge cases

PQ engineering = baby that recently learned how to walk, stumbles a lot

Thank you for listening!

- https://cyber.ee/
- @ info@cyber.ee
- cybernetica
- CyberneticaAS
- cybernetica_ee
- Cybernetica

Petr Muzikant, petr.muzikant@cyber.ee