The Python Cryptographic Authority's OpenSSL Experience

Alex Gaynor & Paul Kehrer

Hello!

Hello!

What is the Python Cryptographic Authority?

What is the Python Cryptographic Authority?

How do we use OpenSSL?

A brief history...

- Multiple backends

A brief history...

- Multiple backends

A brief history...

- Multiple backends
- **One backend: OpenSSL**

A brief history...

- Multiple backends
- One backend: OpenSSL
- **Along came the forks**

A brief history...

- Multiple backends
- One backend: OpenSSL
- Along came the forks
- **Please sir, I'd like some memory safety**

A brief history...

- Multiple backends
- One backend: OpenSSL
- Along came the forks
- **Please sir, I'd like some memory safety**

A brief history...

- Multiple backends
- One backend: OpenSSL
- Along came the forks
- Please sir, I'd like some memory safety
- **When it rains it pours**

Distribution

Distribution

What we want from a cryptography library

Desiderata

Desiderata

- Security

Desiderata

- Security
- **Correctness**

Desiderata

- Security
- **Correctness**

Desiderata

- Security
- Correctness
- **Performance**

Desiderata

- Security
- Correctness
- Performance
- **Generality**

Desiderata

- Security
- Correctness
- Performance
- Generality
- **Ergonomics**

What are our challenges with OpenSSL?

Testing

- Coverage

Testing

- Coverage

Testing

- Coverage
- **Unreliable CI hides issues**

Testing

- Coverage
- Unreliable CI hides issues
- **Bug fixes don’t always come with tests**

Testing

- Coverage
- Unreliable CI hides issues
- Bug fixes don’t always come with tests

How do we improve?

CI Improvements

- Combine and require coverage
- Improve the performance of each job so the feedback loop is faster
- CI reliability improvements
- Expand testing to new areas

CI Improvements

- Combine and require coverage
- Improve the performance of each job so the feedback loop is faster
- CI reliability improvements
- Expand testing to new areas

Performance

Performance

Performance

- In the beginning, it was fine

Performance

- In the beginning, it was fine
- **OpenSSL 3 regressed**

Performance

- In the beginning, it was fine
- OpenSSL 3 regressed
- **Complexity is growing**

Performance

- In the beginning, it was fine
- OpenSSL 3 regressed
- Complexity is growing
- **Root causes are unaddressed**

Performance

- In the beginning, it was fine
- OpenSSL 3 regressed
- Complexity is growing
- **Root causes are unaddressed**

API Design

API Design

- Mutable keys and other structures

API Design

- Mutable keys and other structures

API Design

- Mutable keys and other structures
- **Unclear ownership semantics**

API Design

- Mutable keys and other structures
- **Unclear ownership semantics**

API Design

- Mutable keys and other structures
- Unclear ownership semantics
- **General complexity**

API Design

- Mutable keys and other structures
- Unclear ownership semantics
- **General complexity**

```c
int mlkem_encapsulate(const unsigned char *pubkey_bytes, size_t pubkey_len,
                      unsigned char *shared_secret_out, unsigned char *ciphertext_out)
{
    EVP_PKEY *peer_pkey = NULL;
    int ret = 0;
    OSSL_PARAM params[2];
    // Create an EVP_PKEY from the raw public key bytes
    params[0] = OSSL_PARAM_construct_octet_string(OSSL_PKEY_PARAM_PUB_KEY,
                                                  (void *)pubkey_bytes, pubkey_len);
    params[1] = OSSL_PARAM_construct_end();
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_from_name(NULL, "ML-KEM-768", NULL);
    if (ctx == NULL) {
        goto cleanup;
    }
    if (EVP_PKEY_fromdata_init(ctx) <= 0) {
        goto cleanup;
    }
    if (EVP_PKEY_fromdata(ctx, &peer_pkey, EVP_PKEY_PUBLIC_KEY, params) <= 0) {
        goto cleanup;
    }
    // Free the old context and create new one for encapsulation
    EVP_PKEY_CTX_free(ctx);
    ctx = EVP_PKEY_CTX_new(peer_pkey, NULL);
    if (ctx == NULL) {
        goto cleanup;
    }
    if (EVP_PKEY_encapsulate_init(ctx, NULL) <= 0) {
        goto cleanup;
    }
    size_t ct_size = OSSL_ML_KEM_768_CIPHERTEXT_BYTES;
    size_t secret_size = OSSL_ML_KEM_SHARED_SECRET_BYTES;
    // Perform encapsulation
    if (EVP_PKEY_encapsulate(ctx, ciphertext_out, &ct_size, shared_secret_out,
&secret_size) <= 0) {
        goto cleanup;
    }
    ret = 1;

cleanup:
    EVP_PKEY_free(peer_pkey);
    EVP_PKEY_CTX_free(ctx);
    return ret;
}
```

```c
int mlkem_encapsulate(const uint8_t *serialized_pubkey,
                      size_t pubkey_len,
                      uint8_t out_shared_secret[MLKEM_SHARED_SECRET_BYTES],
                      uint8_t out_ciphertext[MLKEM768_CIPHERTEXT_BYTES]) {
    if (serialized_pubkey == NULL || out_shared_secret == NULL || out_ciphertext == NULL) {
        return 0;
    }

    // Parse the serialized public key
    struct MLKEM768_public_key public_key;
    CBS cbs;
    CBS_init(&cbs, serialized_pubkey, pubkey_len);

    if (!MLKEM768_parse_public_key(&public_key, &cbs)) {
        return 0;  // Parse error
    }

    // Check that there are no trailing bytes
    if (CBS_len(&cbs) != 0) {
        return 0;  // Unexpected trailing data
    }

    // Encapsulate a random shared secret
    MLKEM768_encap(out_ciphertext, out_shared_secret, &public_key);

    return 1;
}
```

Internal Complexity

- Source code readability
- Custom preprocessor
- Breaks grep
- Breaks go-to-definition

Internal Complexity

- Source code readability
- Custom preprocessor
- Breaks grep
- Breaks go-to-definition

Internal Complexity

- Source code readability
- Custom preprocessor
- Breaks grep
- Breaks go-to-definition

Strict Parsing

Strict Parsing

Divergence from forks

- LibreSSL
- BoringSSL
- aws-lc

What we think a modern cryptography library should drive towards

Memory safety

Memory safety

Memory safety

Extremely well tested

Extremely well tested

Extremely well tested

Narrow API surfaces

Narrow API surfaces

Performance by design

Performance by design

Thank you!