

Crypto-agility demystified

A project called *Sandwich*

Thomas B.¹

¹SandboxAQ

July 4, 2024

- 1 Today's cryptography
 - The developer who uses cryptography
 - Improving how cryptography is used
- 2 *crypto-agility*: do better cryptography
 - Definition
 - What are the benefits of it?
- 3 *Sandwich*: an experiment
 - What is *Sandwich*?
 - What is it offering?
 - API
 - Technical challenges

Quizz

Quizz

How many cryptography libraries are there in macOS/iOS?

Quizz

How many cryptography libraries are there in macOS/iOS?

1

Quizz

How many cryptography libraries are there in macOS/iOS?

5

Quizz

How many cryptography libraries are there in macOS/iOS?

10

Quizz

How many cryptography libraries are there in macOS/iOS?

14!

(and probably more!)

Cryptography libraries on macOS

<code>libboringsssl</code>	DL	<i>AES,ECDH,X509,TLS/SSL,MAC,DSA,CertVerif</i>
<code>libcrypto</code>	DL	<i>AES,ECDH,X509,TLS/SSL,MAC,DSA</i>
<code>libcommonCrypto</code>	DL	<i>AES,MAC</i>
<code>CryptoKit</code>	F	<i>AES,DSA</i>
<code>FindMyCrypto</code>	F	<i>PubKey Cryptography</i>
<code>libcorecrypto</code>	DL	<i>AES,ECDH,X509,MAC,DSA</i>
<code>libssl</code>	DL	<i>TLS/SSL,CertVerif,X509</i>
<code>libcoretls</code>	DL	<i>TLS/SSL,CertVerif,X509</i>
<code>libtls</code>	DL	<i>TLS/SSL,X509</i>
<code>Security</code>	F	<i>AES,ECDH,X509,TLS/SSL,Certverif</i>

DL: dylib, F: Framework

Which ones were found to be buggy once?

<code>libboringsssl</code>	DL	<i>AES,ECDH,X509,TLS/SSL,MAC,DSA,CertVerif</i>
<code>libcrypto</code>	DL	<i>AES,ECDH,X509,TLS/SSL,MAC,DSA</i>
<code>libcommonCrypto</code>	DL	<i>AES,MAC</i>
<code>CryptoKit</code>	F	<i>AES,DSA</i>
<code>FindMyCrypto</code>	F	<i>PubKey Cryptography</i>
<code>libcorecrypto</code>	DL	<i>AES,ECDH,X509,MAC,DSA</i>
<code>libssl</code>	DL	<i>TLS/SSL,CertVerif,X509</i>
<code>libcoretls</code>	DL	<i>TLS/SSL,CertVerif,X509</i>
<code>libtls</code>	DL	<i>TLS/SSL,X509</i>
<code>Security</code>	F	<i>AES,ECDH,X509,TLS/SSL,Certverif</i>

DL: dylib, F: Framework

What, bugs?

<code>libboringsssl</code>	DL	<i>cc OpenSSL</i>
<code>libcrypto</code>	DL	<i>2021-41581: stack based. b.o.</i>
<code>libcommonCrypto</code>	DL	<i>CVE-2016-1802: mishandles return values</i>
<code>libcorecrypto</code>	DL	<i>CVE-2024-23218: private key recovery</i>
<code>libssl</code>	DL	<i>cc OpenSSL/LibreSSL</i>
<code>libcoretls</code>	DL	<i>CVE-2015-4000: downgrade attack</i>
<code>libtls</code>	DL	<i>cc OpenSSL/LibreSSL</i>
<code>Security</code>	F	<i>CVE-2022-42793: code signing checks bypass</i>

DL: dylib, F: Framework

What, bugs?

libcrypto - CVE-2021-41581

- memory corruption bug
- cause: bug in code
- fix: patch the code

What, bugs?

libcommonCrypto - CVE-2016-4711

- cleartext disclosure.
- cause: weak / poorly documented API
- fix: more explicit API, rewrite the documentation, patch the code

What, bugs?

libcoreCrypto - CVE-2024-23218

- secret recovery
- cause: non constant-time computation
- fix: patch the code

Security - CVE-2022-42793

- code signing bypass
- cause: logical bug: root CA anchoring failure
- fix: more verification routine

What are the costs?

- more code to maintain
- more sw developers
- ++ attack surface
- inventory

What are the costs?

- more code to maintain
- more sw developers
- ++ attack surface
- inventory

Could we find a solution?

Single cryptography library, unified API, support for multiple programming languages

Cryptography in software engineering

The internet contains numerous cryptography libraries:

- Primitives written in various programming languages.

Examples

- `pq-crystals`
- `tiny-AES-c`

The internet contains numerous cryptography libraries:

- Primitives written in various programming languages.
- Libraries packaging primitives.

Examples

- `libsodium`
- `PyCryptodome`

The internet contains numerous cryptography libraries:

- Primitives written in various programming languages.
- Libraries packaging primitives.
- Libraries providing cryptosystems

Examples

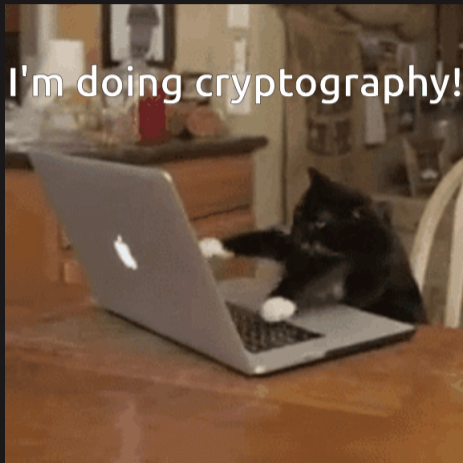
- `RustTLS`
- `python.ssl`

The internet contains numerous cryptography libraries:

- Primitives written in various programming languages.
- Libraries packaging primitives.
- Libraries providing cryptosystems

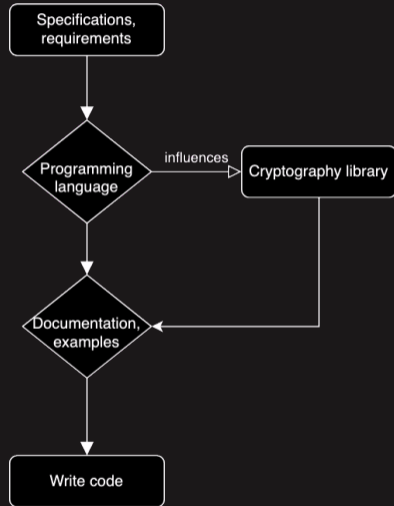
Which one should I use?

How do we use cryptography today?



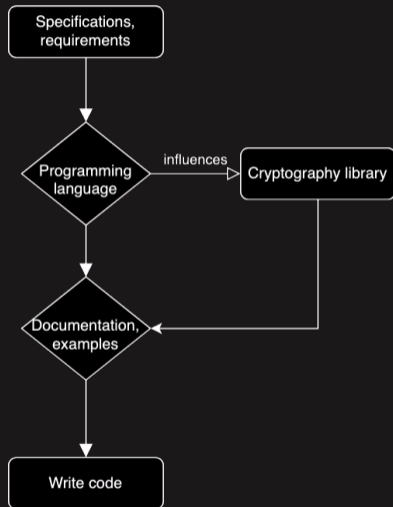
A (cryptography) software engineer's typical day

1 specifications, requirements



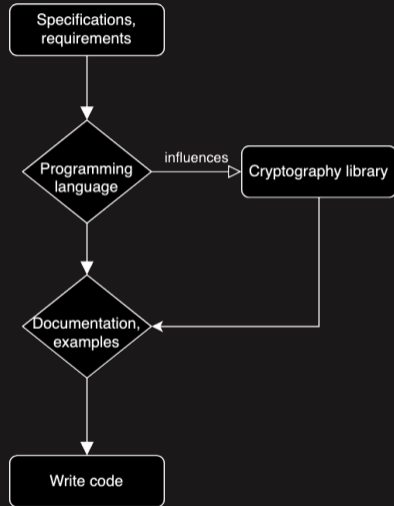
A (cryptography) software engineer's typical day

- 1 specifications, requirements
- 2 pick a library / programming language



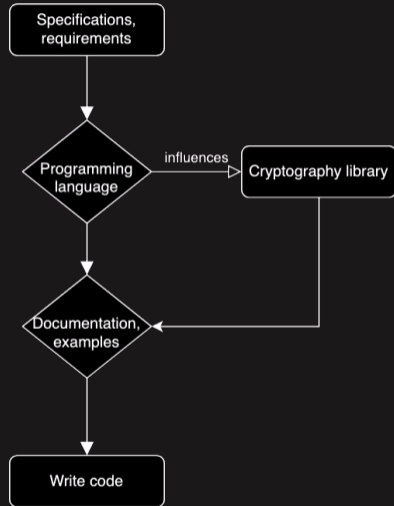
A (cryptography) software engineer's typical day

- 1 specifications, requirements
- 2 pick a library / programming language
- 3 read documentation, go through examples, etc.

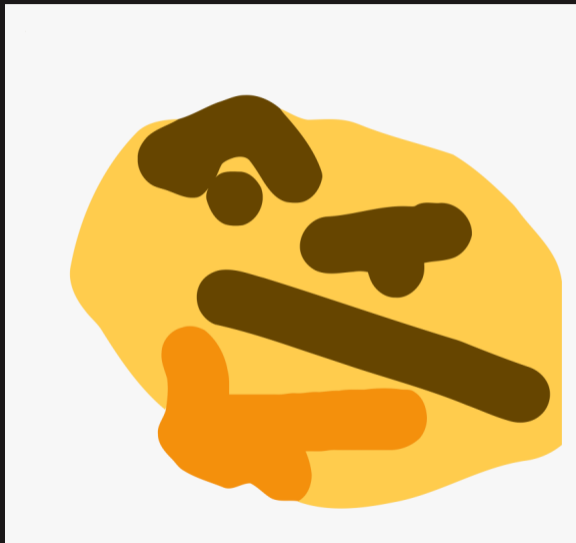


A (cryptography) software engineer's typical day

- 1 specifications, requirements
- 2 pick a library / programming language
- 3 read documentation, go through examples, etc.
- 4 write code



What difficulties can be anticipated?



Stage:

one

1. specifications, requirements

Stage:

one

1. specifications, requirements

- spec. mutate

Stage:

one

1. specifications, requirements

- spec. mutate
- new requirements

Stage:

one

1. specifications, requirements

- spec. mutate
- new requirements
- new uses

Stage:

two

2. pick a library / programming language

Stage:

two

2. pick a library / programming language

- no longer maintained

Stage:

two


2. pick a library / programming language

- no longer maintained
- new major

Stage:

two

2. pick a library / programming language

- no longer maintained
- new major
- bugs 

Stage:

two

2. pick a library / programming language

- no longer maintained
- new major
- bugs 🐛
- !FIPS

Stage:

two

2. pick a library / programming language

- no longer maintained
- new major
- bugs 🐛
- !FIPS
- lack of scheme support

Stage:

three

3. read documentation, go through examples, etc.

Stage:

three

3. read documentation, go through examples, etc.

- mistakes are likely

Stage:

three

3. read documentation, go through examples, etc.

- mistakes are likely
- APIs

Stage:

three

3. read documentation, go through examples, etc.

- mistakes are likely
- APIs
- documentation may be incomplete, missing or incorrect

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len);
```


Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len);
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len); // *still strongly discouraged* ...
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len); // *still strongly discouraged* ...  
X509 *y = d2i_X509(NULL, ppin, len);
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len); // *still strongly discouraged* ...  
X509 *y = d2i_X509(NULL, ppin, len); // *still strongly discouraged* ...
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len); // *still strongly discouraged* ...  
X509 *y = d2i_X509(NULL, ppin, len); // *still strongly discouraged* ...  
x = X509_new_ex(&libctx, ...);  
X509 *y = d2i_X509(&x, ppin, len);
```

Outdated documentation: example

Given the following API:

```
X509 *d2i_X509(X509 **a, const uint8_t **ppin, long length);
```

How does it behave in the following?:

```
X509 *x = X509_new();  
X509 *y = d2i_X509(&x, ppin, len); // *strongly discouraged* ...  
x = NULL;  
X509 *y = d2i_X509(&x, ppin, len); // *still strongly discouraged* ...  
X509 *y = d2i_X509(NULL, ppin, len); // *still strongly discouraged* ...  
x = X509_new_ex(&libctx, ...);  
X509 *y = d2i_X509(&x, ppin, len); // sounds good!
```

Stage:


four

4. write code

Stage:

four


4. write code

- bugs (mostly logical ones) 

Stage:

four

4. write code

- bugs (mostly logical ones) 
- usually tight to the programming language

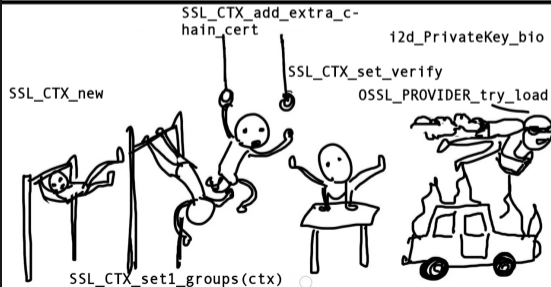
crypto-agility: a philosophy



What is *crypto-agility*?

crypto-agility:

```
let cfg = load("my_conf.proto")?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|...)?;
```



Key principles:

- configuration-oriented

What is *crypto-agility*?

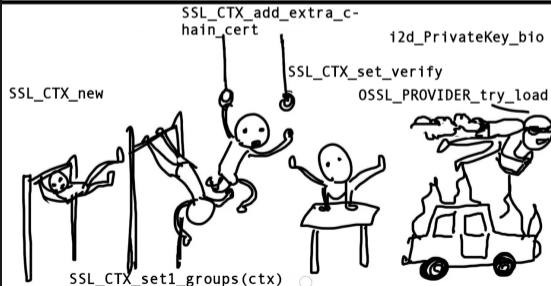
crypto-agility:

```
let cfg = load("my_conf.proto")?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|"...")?;
```



Key principles:

- configuration-oriented
- support for multiple backends



What is *crypto-agility*?

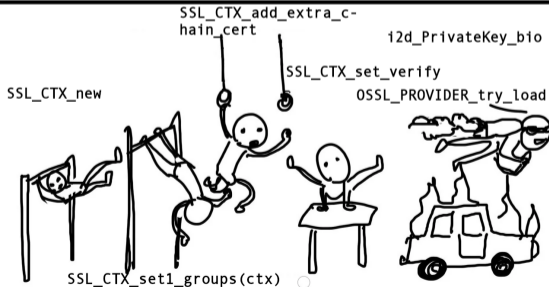
crypto-agility:

```
let cfg = load("my_conf.proto"?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|...)?;
```



Key principles:

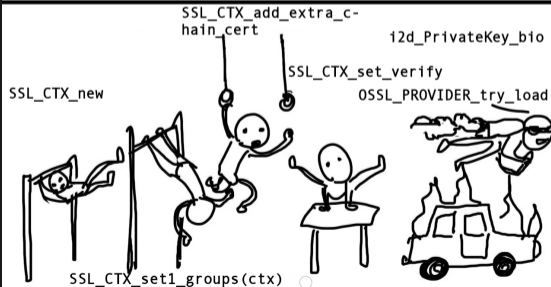
- configuration-oriented
- support for multiple backends
- backend agnostic



What is *crypto-agility*?

crypto-agility:

```
let cfg = load("my_conf.proto"?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|"...)?;
```



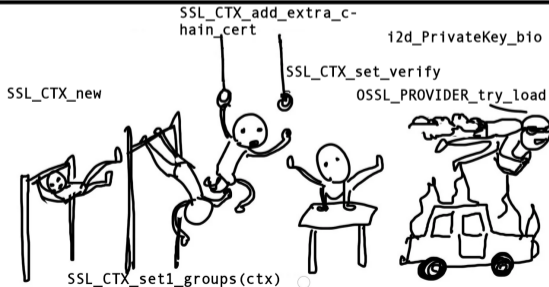
Key principles:

- configuration-oriented
- support for multiple backends
- backend agnostic
- unified API

What is *crypto-agility*?

crypto-agility:

```
let cfg = load("my_conf.proto"?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|"...)?;
```



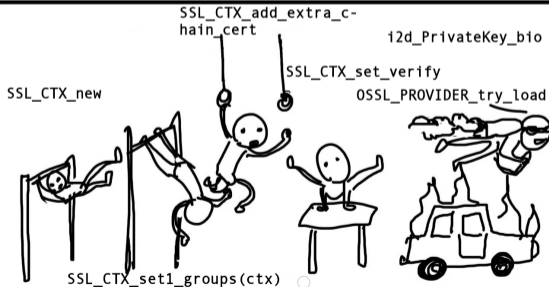
Key principles:

- configuration-oriented
- support for multiple backends
- backend agnostic
- unified API
- switching button for cryptographic primitives

What is *crypto-agility*?

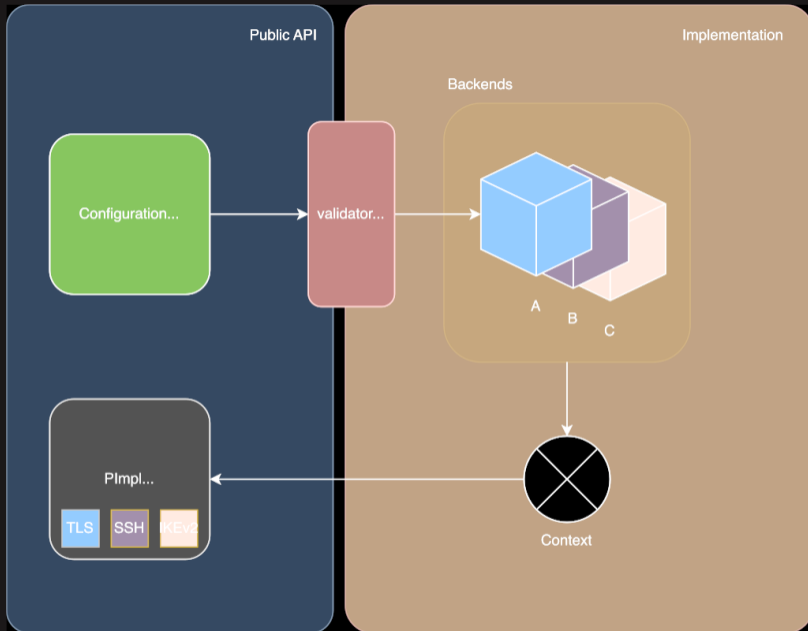
crypto-agility:

```
let cfg = load("my_conf.proto"?;  
do_tls(&cfg, "OpenSSL"|"BoringSSL"|"...)?;
```



Key principles:

- configuration-oriented
- support for multiple backends
- backend agnostic
- unified API
- switching button for cryptographic primitives
- ★ *bonus*: programming language agnostic



crypto-agility: benefits

Benefits

backed by multiple cryptography libraries

various support, modes, cryptosystems

Benefits

backed by multiple cryptography libraries

various support, modes, cryptosystems

less error prone

explicit, expressive, no default behavior, configuration-oriented

Benefits

backed by multiple cryptography libraries

various support, modes, cryptosystems

less error prone

explicit, expressive, no default behavior, configuration-oriented

unified API

- one API, many features
- configuration serves as documentation
- learn once, use everywhere

Benefits

backed by multiple cryptography libraries

various support, modes, cryptosystems

less error prone

explicit, expressive, no default behavior, configuration-oriented

unified API

- one API, many features
- configuration serves as documentation
- learn once, use everywhere

Polyglot

Invoked from various programming languages

one

1. pick a library / programming language

- >> **pick a library / programming language**
- read documentation, go through examples, etc.
- write code

one

1. pick a library / programming language

suggested solution

Not choosing is still choosing. - Sartre

- >> **pick a library / programming language**
- read documentation, go through examples, etc.
- write code

two

2. read documentation, go through examples, etc.

- pick a library / programming language
- >> **read documentation, go through examples, etc.**
- write code

Benefits: stage 2

two

2. read documentation, go through examples, etc.

suggested solution

Unified API, configuration serves as documentation

- pick a library / programming language
- >> **read documentation, go through examples, etc.**
- write code

three

3. write code

- pick a library / programming language
- read documentation, go through examples, etc.
- >> **write code**

three

3. write code

suggested solution

Most expressive, explicit and robust.

- pick a library / programming language
- read documentation, go through examples, etc.
- >> **write code**

three

3. write code

suggested solution

Most expressive, explicit and robust.

!

Remains a problem to solve.

- pick a library / programming language
- read documentation, go through examples, etc.
- >> **write code**

Sandwich



Authors: Duc Nguyen, Gaëtan Wattiau, Ibraheem Saleh, Jason Goertzen, Laurent Fousse, Mansi Sheth,

Thomas Bailleux, Timothy Harder

Sandwich: an experiment in *crypto-agility*

Goals

- meet the *crypto-agility* criteria

Sandwich: an experiment in *crypto-agility*

Goals

- meet the *crypto-agility* criteria
- open source

Goals

- meet the *crypto-agility* criteria
- open source
- integrate PQ/T hybrid and full PQ cryptography schemes

Alright, but why...?

yet another cryptography library...

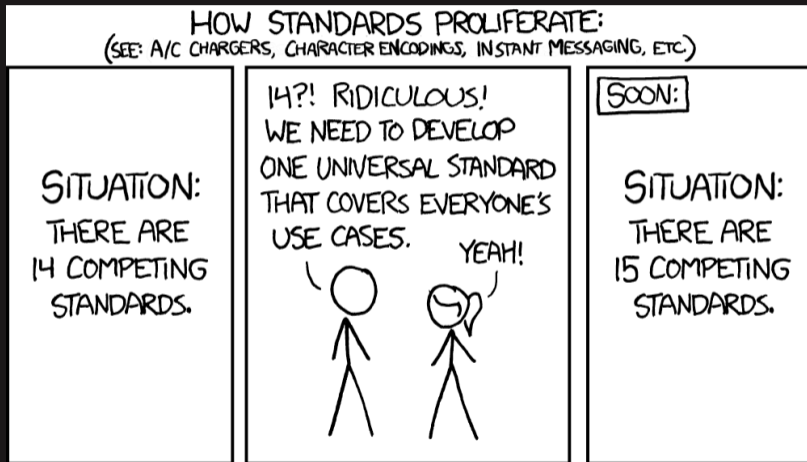


Figure: Standards - xkcd 927

Alright, but why...?

Quoting Carlos Aguilar-Melchor:

It Is an Opportunity!

Cryptography is managed today as it was in the 80s

- What is the list of all my certificates? Which ones are about to be invalidated? Which ones have automated renewal? Does the renewal process satisfy our standards?
- Which binaries in my company use a given version of OpenSSL? Will something break if I start disabling RC4 in my IT systems?
- Where do I have data at rest encryption? Are the keys rotated? In which country are they?

© SandboxAQ 2024

SandboxAQ Proprietary Material



Figure: Carlos' keynote at SSTIC 2024¹

- Core written in Rust, API also reachable from C, Python and Go

²<https://crates.io/crates/ring>

- Core written in Rust, API also reachable from C, Python and Go
- Three different backends: OpenSSL (3.3.1), BoringSSL, `ring`²

²<https://crates.io/crates/ring>

- Core written in Rust, API also reachable from C, Python and Go
- Three different backends: OpenSSL (3.3.1), BoringSSL, ring²
- API comprises protobuf definitions

²<https://crates.io/crates/ring>

- Core written in Rust, API also reachable from C, Python and Go
- Three different backends: OpenSSL (3.3.1), BoringSSL, `ring`²
- API comprises protobuf definitions
- TLS, asymmetric cryptography, signatures

Similar projects

Envoy (for the TLS part)

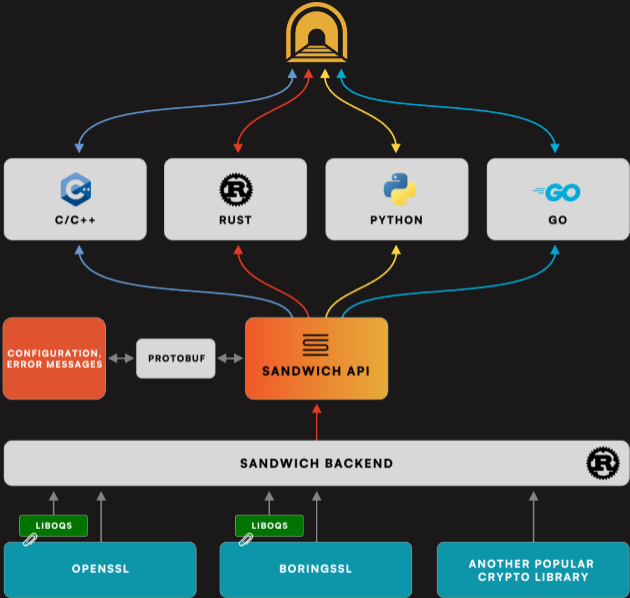
²<https://crates.io/crates/ring>

SANDWICH BINDINGS

- Simple API
- Misuse prevention
- Minimal overhead

SANDWICH BACKEND

- Link to a cryptography library at compile-time
- Support liboqs on top of OpenSSL and BoringSSL
- Easy to extend for other libraries



FORK QUANTUM-SAFE CLASSICAL

Reusable protobuf definitions

- Configuration based on versioned protobuf definitions
- Explicit: no default decisions, no hidden contracts

x25519_kyber768³

```
impl: IMPL_OPENSSL3_OQS_PROVIDER
client {
  tls {
    common_options {
      tls13 {
        ke: "x25519_kyber768"
        compliance {
          hybrid_choice: HYBRID_ALGORITHMS_ALLOW
          quantum_safe_choice: QUANTUM_SAFE_ALGORITHMS_ALLOW
          classical_choice: CLASSICAL_ALGORITHMS_ALLOW
          bit_strength_choice: BIT_STRENGTH_AT_LEAST_128
        }
      }
    }
  }
  x509_verifier {
    trusted_cas {
      static {
        data {
          filename: "server_fullchain.pem"
        }
        format: ENCODING_FORMAT_PEM
      }
    }
  }
}
```

³X25519Kyber768Draft00

API: example in Rust

```
use std::io::{
    Read as _,
    Write as _,
};
use std::net::TcpStream;
use sandwich::tunnel;

// Reads the configuration, for instance from a file.
let configuration: impl AsRef<str> = read_config()?;

// Instantiates a TLS context with the given configuration.
let client_context = tunnel::Context::try_from(configuration)?;

// Instantiates a configuration for TLS tunnels (SNI, SANs, etc.)
let tunnel_configuration = pb_api::TunnelConfiguration::parse(...)?;

// Connect to the server.
let mut tube = client_context.new_tunnel(
    TcpStream::connect("example.com:443"),
    &tunnel_configuration
)?;

// Performs the TLS handshake.
tube.handshake()?;
// `sandwich::Tunnel` implements [`std::io::Write`] and [`std::io::Read`].
tube.write(b"GET / HTTP/1.1\nHost: example.com\n\n")?;
```

Many third-parties:

- OpenSSL, BoringSSL, ring, protobuf

Many third-parties:

- OpenSSL, BoringSSL, ring, protobuf
- Python dependencies, Go dependencies, Rust dependencies

Many third-parties:

- OpenSSL, BoringSSL, ring, protobuf
- Python dependencies, Go dependencies, Rust dependencies

Solution: Bazel

Bazel build system



Many third-parties:

- OpenSSL, BoringSSL, ring, protobuf
- Python dependencies, Go dependencies, Rust dependencies

Solution: Bazel

Bazel build system



- Extensible
- Hermetic builds
- Fine-grained control over versions

Bazel is great, but. . .

Bazel is great, but... Rustaceans use Cargo!



Solution: nested build systems!

Nested build systems:

- Define placeholder libraries within a Cargo workspace

Solution: nested build systems!

Nested build systems:

- Define placeholder libraries within a Cargo workspace
- Call Bazel from a Build Script

Solution: nested build systems!

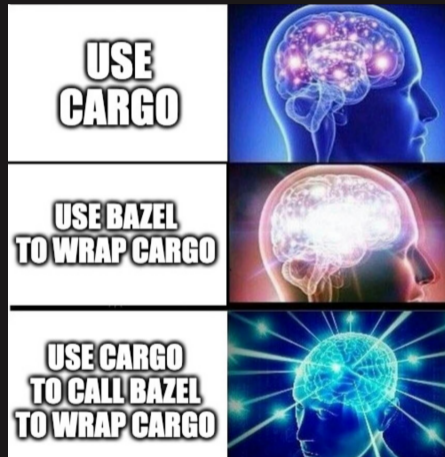
Nested build systems:

- Define placeholder libraries within a Cargo workspace
- Call Bazel from a Build Script
- Collect and dispatch artifacts across the libraries

Solution: nested build systems!

Nested build systems:

- Define placeholder libraries within a Cargo workspace
- Call Bazel from a Build Script
- Collect and dispatch artifacts across the libraries



feature flags

Let us choose the backend library at compile time.

feature flags

Let us choose the backend library at compile time.

- `curl-openssl`
- `curl-libressl`
- `curl-wolfssl`

feature flags

Let us choose the backend library at compile time.

- `curl-openssl`
- `curl-libressl`
- `curl-wolfssl`

Multiple libraries within the same build...

feature flags

Let us choose the backend library at compile time.

- `curl-openssl`
- `curl-libressl`
- `curl-wolfssl`

Multiple libraries within the same build...

- Symbol name collisions at link time...
- ...caused by: forks...

Agility at compile time / runtime

feature flags

Let us choose the backend library at compile time.

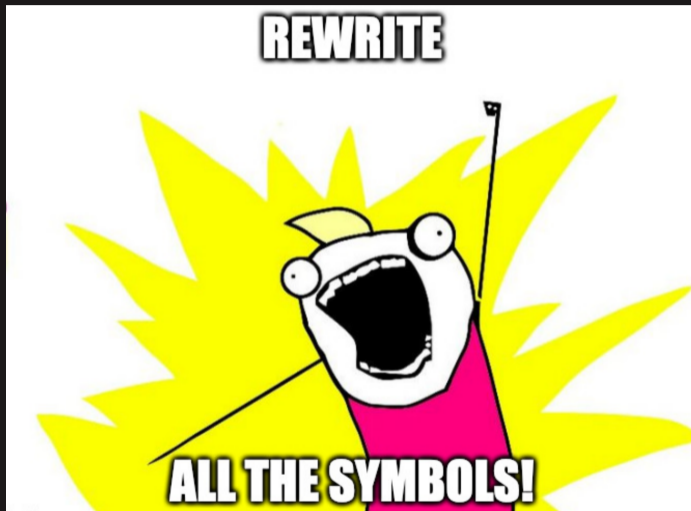
- `curl-openssl`
- `curl-libressl`
- `curl-wolfssl`

Multiple libraries within the same build...

- Symbol name collisions at link time...
- ...caused by: forks...
- ...caused by: C



Solution?



Bartleby: A symbol renaming toolkit

Bartleby:

- Collect symbols from input objects, archives



Bartleby: A symbol renaming toolkit

Bartleby:

- Collect symbols from input objects, archives
- Determine the ones that are defined (t|T)



Bartleby: A symbol renaming toolkit

Bartleby:

- Collect symbols from input objects, archives
- Determine the ones that are defined (t|T)
- Discard the others



Bartleby: A symbol renaming toolkit

Bartleby:

- Collect symbols from input objects, archives
- Determine the ones that are defined (t|T)
- Discard the others
- Prefix their names



Conclusion

- *crypto-agility* is a forerunner concept
- bring cryptography software engineering into 2024!
- will be essential for transitioning to *Post-Quantum Cryptography* seamlessly.
- *Sandwich* source code: <https://github.com/sandbox-quantum/sandwich>
- *Bartleby* source code: <https://github.com/sandbox-quantum/bartleby>

THE WORLD IF CRYPTOGRAPHY SOFTWARE WERE MORE MODERN



Thank you! Questions ?