QUBIP

OpenSSL Conference
Prague 2025

OpenSSL Providers in Rust (for the PQC transition): a workshop

Tampere University

Nicola Tuveri
Doctoral Researcher
Tampere University

The QUBIP project is funded by the European Union under the Horizon Europe framework programme [Grant Agreement No. 101119746]

$ doas -u openssl whoami

- a member of the Academics Community (https://openssl-communities.org/hub-academics/)
  - Side messages:
    - **Join the communities you self-identify with!**
    - Start discussions: there are no silly questions, we need more point-of-views, the more diverse the better!
    - Challenge your BAC/TAC representatives
    - Nominate yourself in the upcoming elections
    - **We need more Academics and new elected representatives to advise the OpenSSL Projects!**

- I have been contributing to OpenSSL for several years as an external contributor, Committer, OTC member, SRT volunteer, and most recently as C/TAC & F/BAC+TAC

See all the ways to reach me at https://romen.dev

👋 *Please reach out!!!* 👋

$ doas -u researcher whoami

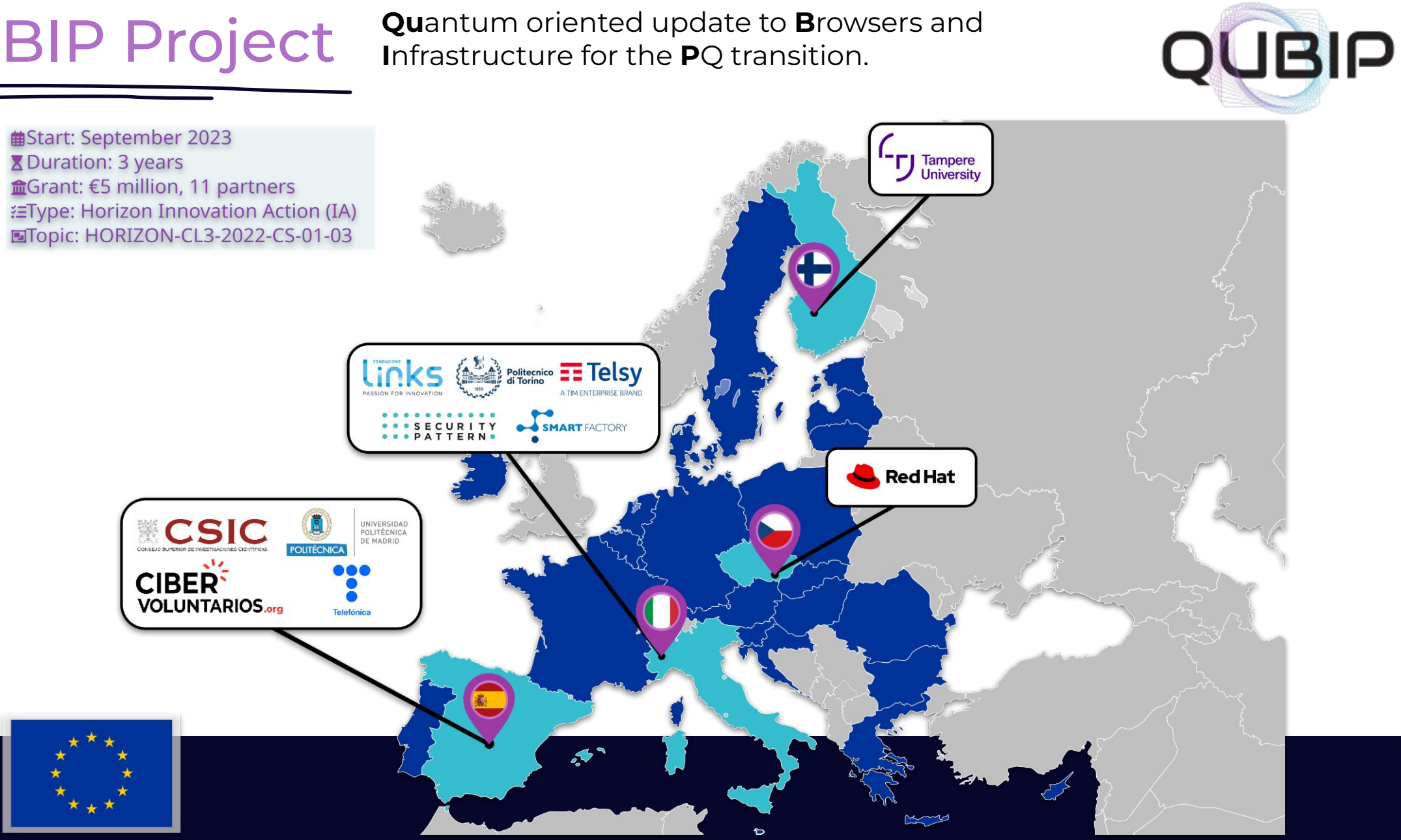
- I have been focusing my research on
  - μ-architectural side-channels in cryptographic software implementations: defects and countermeasures
  - bridging the gap between bleeding-edge state-of-the-art crypto implementations and mainstream adoption
- On the last topic, I have been boring reviewers for years about using `ENGINE`s and `Provider`s to bridge these gaps
  - `libsuola` [ia.cr/2018/354]
  - *Batch Binary Weierstrass* [ia.cr/2019/829]
  - https://openssIntru.cr.yp.to/
  - QUBIP `aurora` (this talk!)

If you are curious about my first research interest---and you want to know more from someone that actually knows what they are doing---do not miss the material from Billy Brumley's presentation at the OpenSSL Conference Prague 2025! https://openssl-conference.org/speaker-sessions/detail-151_1292941

OpenSSL Conference Prague 2025

QUBIP Project

Quantum oriented update to **B**rowsers and **I**nfrastructure for the **P**Q transition.



QUBIP Methodology

Applications

Transparent and seamless integration across the rich and diverse ecosystem of higher level applications and libraries.

Existing applications do not require changes in their source code to transition to PQC: they rely on the cryptographic agility provided by the APIs of cryptographic libraries.

Crypto Libraries

Widespread crypto libraries are deeply integrated into modern OSes, acting as a foundation for user applications as well as system tools, runtimes, and libraries.

They often can be extended by pluggable loadable modules, to support HW and SW security modules. We can use these capabilities to transition towards PQC.

Loadable Modules

Loadable modules can be plugged at runtime in crypto libraries to support HW or SW implementations.

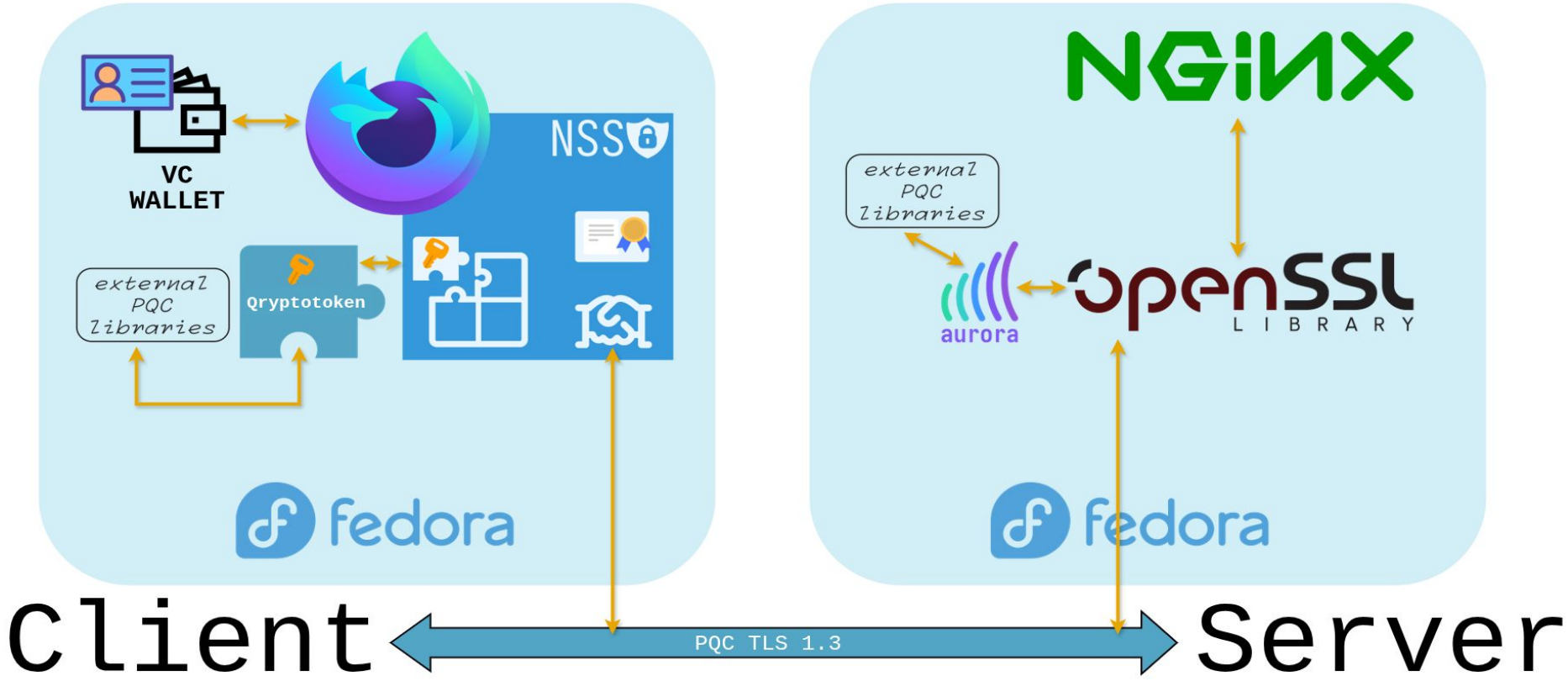
We propose shallow modules—i.e., void of cryptographic implementations—as a means to separate from the fast-paced development in the ecosystem of optimized implementations for PQ primitives, and to decouple external implementations from the data types and patterns required for compatibility with specific cryptographic libraries.

Implementations

We do NOT plan to design new cryptographic algorithms or implementations, but to integrate state-of-the-art optimized implementations for PQ/T hybrids and primitives.

The shallow module approach allows to explore a number of solutions, including SW and HW alternatives, different programming languages, implementations with formal verification guarantees, hybrid schemes, and more.

Internet Browsing Pilot Demonstrator



Check the slides and recording from Akif's and Francesco's presentation at the OpenSSL Conference Prague 2025!

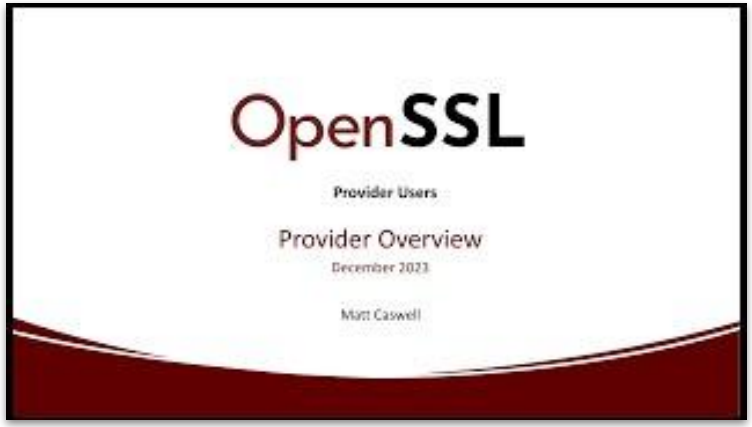
https://openssl-conference.org/speaker-sessions/detail-147_1292922

OpenSSL
Conference
Prague 2025

🤦: Didn’t you promise a workshop?

🤦‍♀️: What abo ?

Provider ???



- Manpages:
  - `provider(7ossl)`

    *A **provider**, in OpenSSL terms, is a unit of code that provides one or more implementations for various operations for diverse algorithms that one might want to perform.*
  - `provider-base(7ossl)`

    How does a Provider interact with OpenSSL's `core` (and viceversa)?

- Check OpenSSL's youtube channel for precious gems!
  - There are more than the 2 videos linked here as an example

<TODO/AI prompt:
You are a knowledgeable long-time OpenSSL developer; you learned all @levitte ever said and wrote about `ENGINE`s and `Providers`.

Insert a funny diagram showing the complex interdependencies among a gazillion cryptic manpages>

Provider ???

Hello, world! (in C)

```c
#include <openssl/core.h>
#include <openssl/core_dispatch.h>
#include <stdio.h>

static const OSSL_DISPATCH provfns[] = {{0, NULL}};

/* Check function signature */
OSSL_provider_init_fn OSSL_provider_init;

int OSSL_provider_init(const OSSL_CORE_HANDLE *handle,
                       const OSSL_DISPATCH *in,
                       const OSSL_DISPATCH **out,
                       void **provctx) {
  fprintf(stderr, "nonsense: Hello, world!\n");
  *out = provfns;
  return 1;
}
```

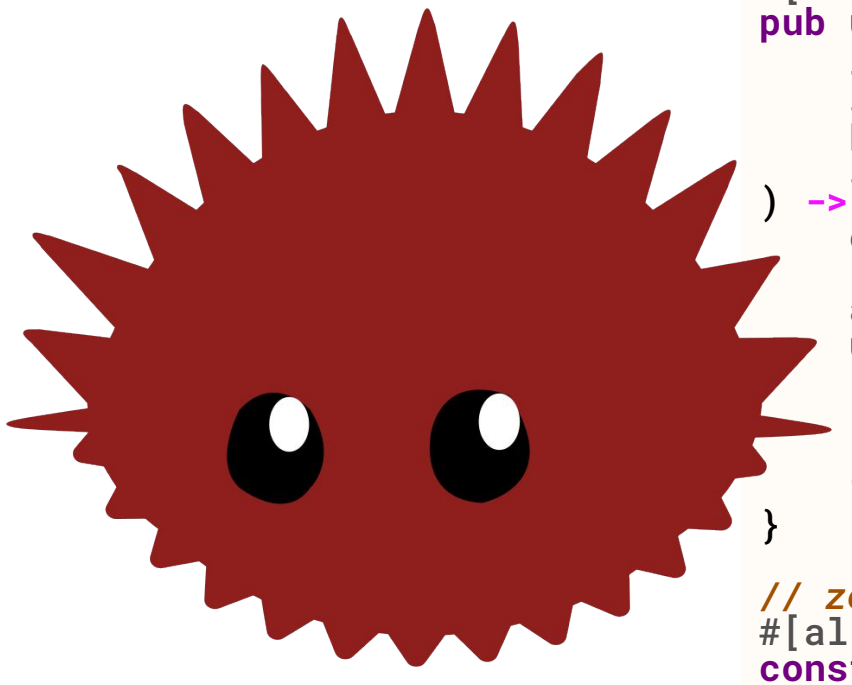
```
test: helloworld.so
	OPENSSL_MODULES=. \
	OPENSSL_CONF=/dev/null \
	openssl list \
	  -provider helloworld \
	  -providers \
	  -verbose
.PHONY: test

helloworld.so: helloworld.o
	clang -shared \
	  -Wl,--export-dynamic-symbol=OSSL_provider_init \
	  -o $@ $<

helloworld.o: helloworld.c
	clang -fPIC -c -o $@ $<
```

(credits: I stole all that made sense from @levitte, I only own the typos)

Demo

Provider ???

Hello, world! (in rust)



```rust
mod bindings; // <-- can you spot the handwaving?
use bindings::*;

static PROVFNS: [OSSL_DISPATCH; 1] = [OSSL_DISPATCH {
    function_id: 0,
    function: None,
}];

#[allow(non_snake_case)]
#[unsafe(no_mangle)]
pub unsafe extern "C" fn OSSL_provider_init(
    _handle: *const OSSL_CORE_HANDLE,
    _core_dispatch: *const OSSL_DISPATCH,
    provider_dispatch: *mut *const OSSL_DISPATCH,
    _provctx: *mut *mut c_void,
) -> c_int {
    eprintln!("nonsense {}: Hello, world!\n", "🦀");

    assert!(!provider_dispatch.is_null());
    unsafe {
        *provider_dispatch = PROVFNS.as_ptr();
    }

    1
}

// zero-cost compile-time checks for consistency
#[allow(clippy::unnecessary_operation, clippy::identity_op)]
const _: () = {
    {
        let _check: OSSL_provider_init_fn = Some(OSSL_provider_init);
    }
};
```

Demo

Thank you!

aurora

An OpenSSL Provider for the PQC transition written in Rust

- Anti-goals
  - do not pick a winner algorithm
  - do not pick a winner implementation
- Project structure (https://github.com/QUBIP/aurora/)
  - src/
    - lib.rs
    - init.rs
    - forge.rs *(more on this later)*
    - adapters.rs
      - libcrux
      - pqclean
      - rustcrypto (pure rust)
      - slhdsa_c

Demo

Q/A

Thank you!