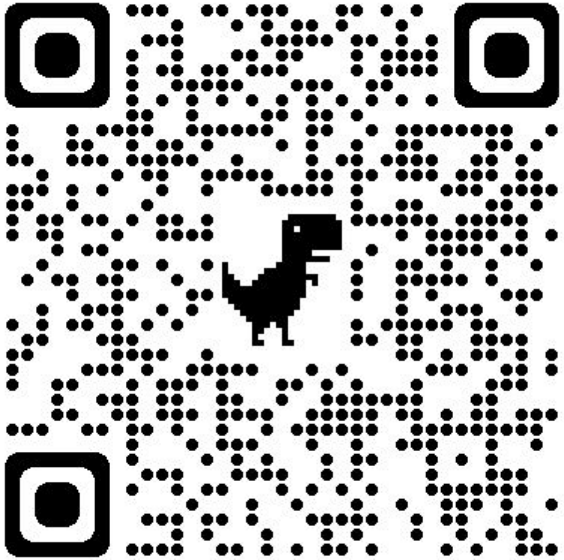
Cryptography with Formal Guarantees

Using HAX

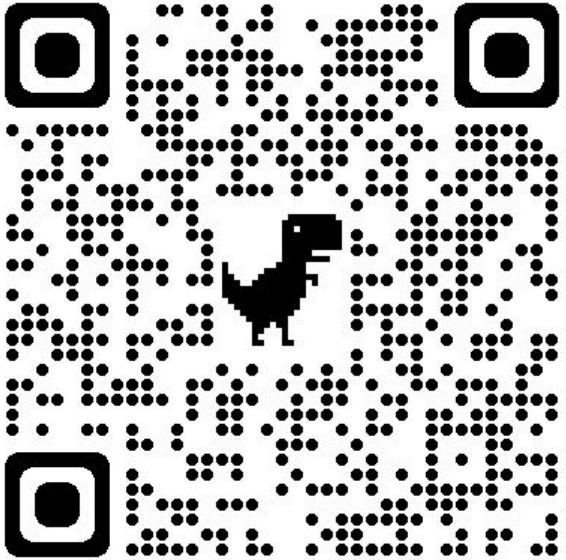
Outline

- Formal what and why?
- How?
- The hax toolchain
- Hands-on

hax demo



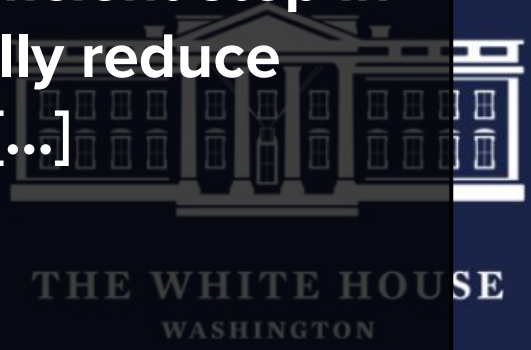
tutorial

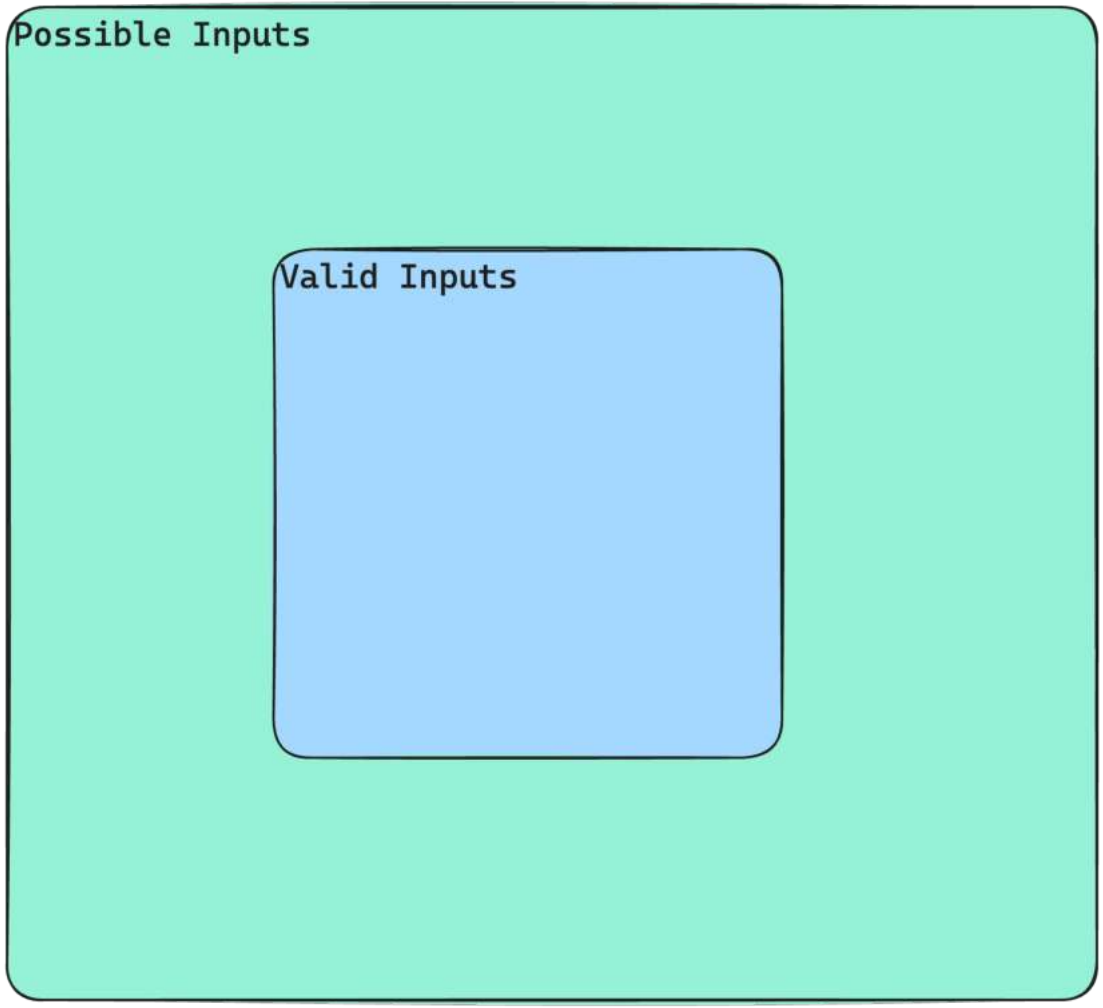


https://github.com/hacspec/hax/blob/franziskus/toronto-2024/examples/README.md

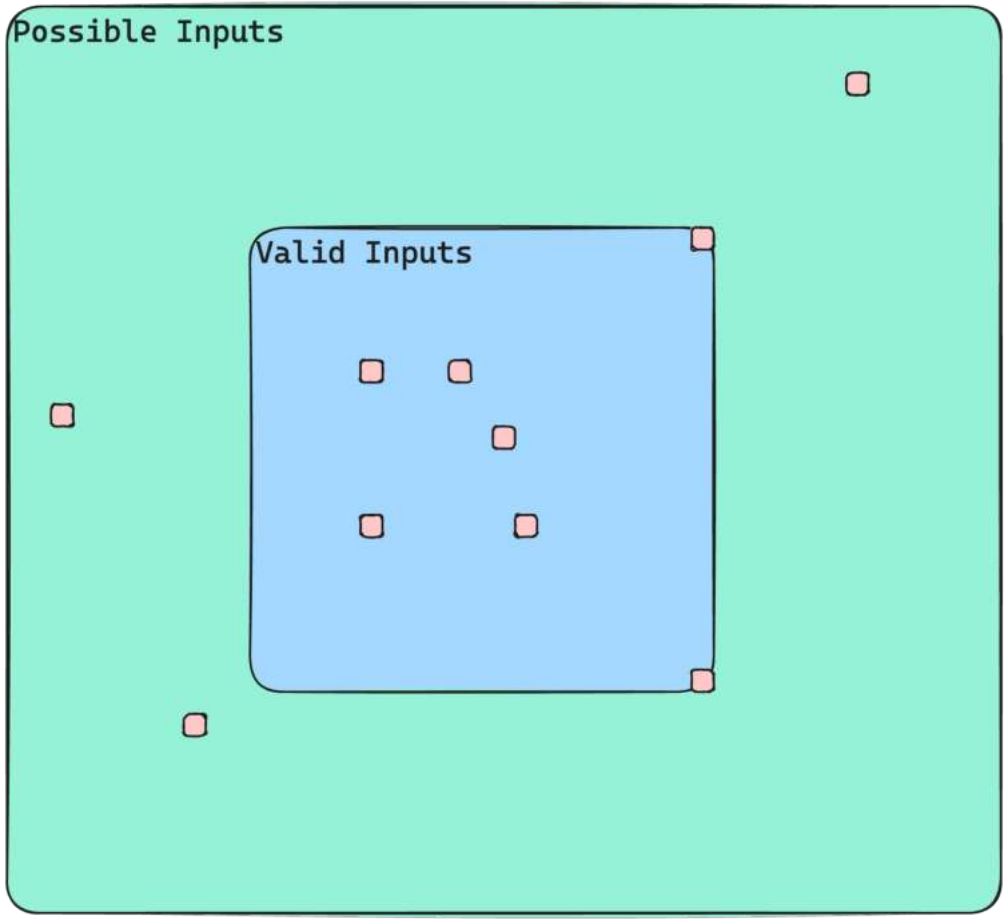
Why Formal Verification?

[...] testing is a necessary but insufficient step in the development process to fully reduce vulnerabilities at scale [...]




Possible Inputs
Valid Inputs

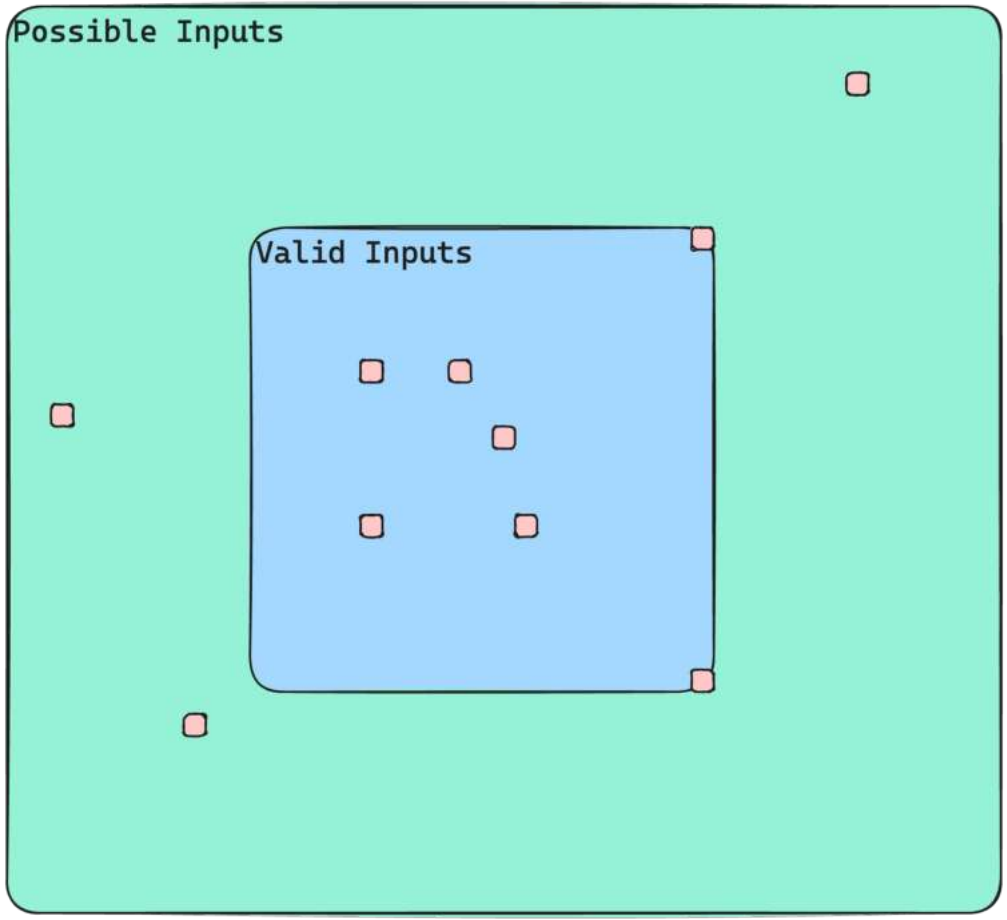
Testing



Testing

Wycheproof
ECDSA P256

471 Tests

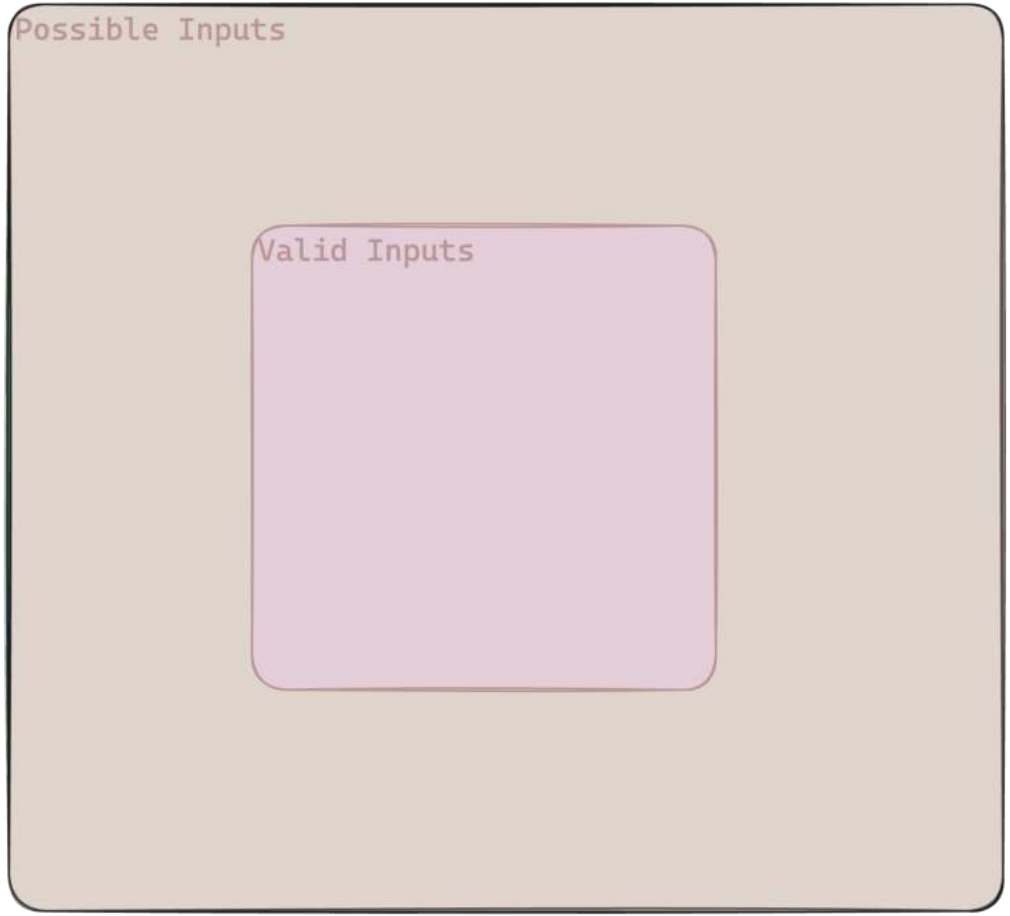


Possible
Inputs

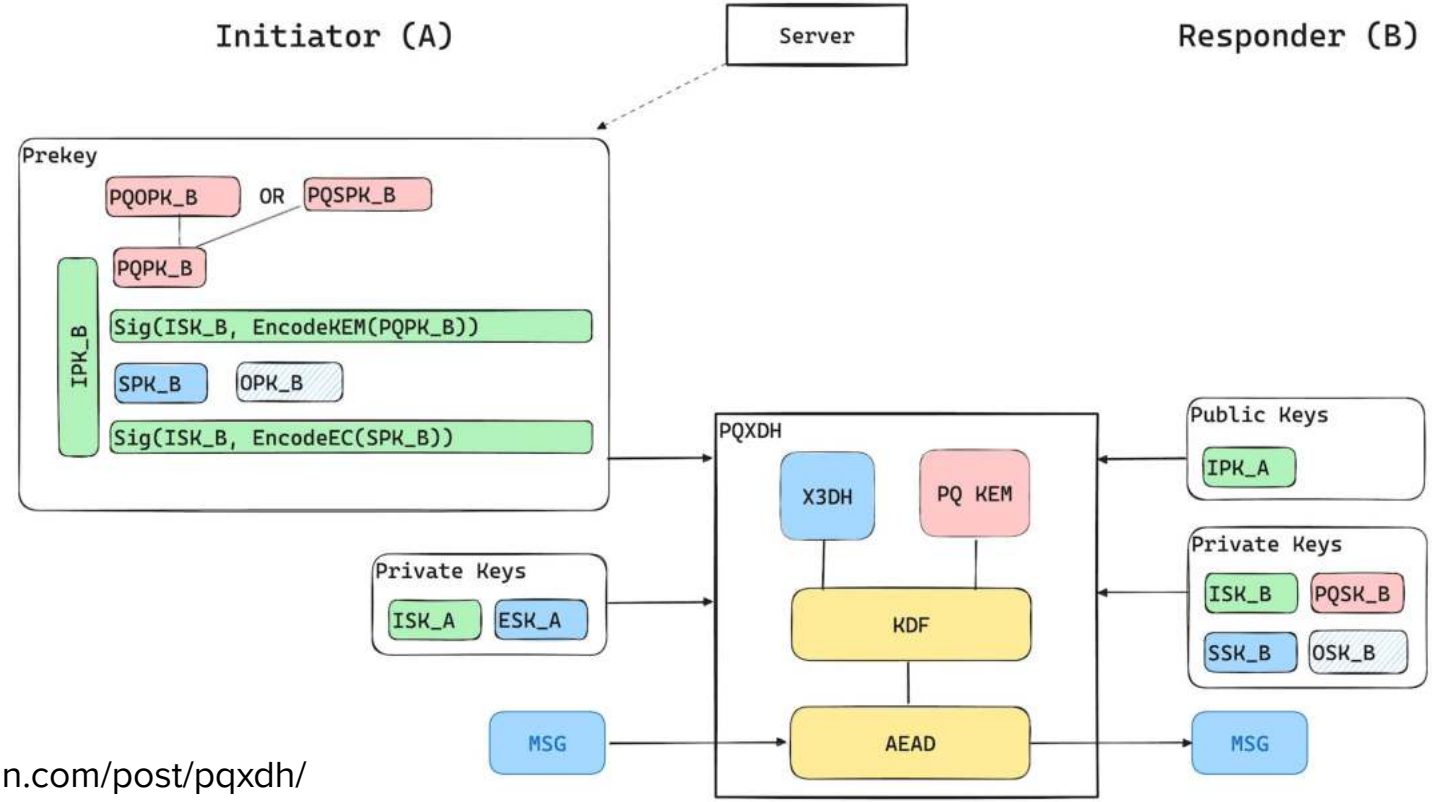
64 bytes
Signature

64 bytes
public key

Verification



PQXDH



https://cryspen.com/post/pqxdh/

TLS 1.3


```
let CompromiseServer() =
    in(c, server_name: bertie__tls13utils__t_Bytes);
    get server_dbs(=server_name, db) in
    event CompromisedServer(server_name);
    out(c, db).

process
    !CreateServer() | !Client() | !Server() | !CompromiseServer()
```

Formal Methods for Performance

```
// Due to the small coefficient bound, we can skip the first round of
// Montgomery reductions.
let mut zeta_i = 1;

for j in 0..128 {
    // Multiply by the appropriate zeta in the normal domain.
    let t = re.coefficients[j + 128] * -1600;

    re.coefficients[j + 128] = re.coefficients[j] - t;
    re.coefficients[j] = re.coefficients[j] + t;
}

re = ntt_at_layer_3(&mut zeta_i, re, 6);
re = ntt_at_layer_3(&mut zeta_i, re, 5);
re = ntt_at_layer_3(&mut zeta_i, re, 4);
re = ntt_at_layer_3(&mut zeta_i, re, 3);
re = ntt_at_layer_3(&mut zeta_i, re, 2);
re = ntt_at_layer_3(&mut zeta_i, re, 1);
```

Formal Methods for Performance

```
...
re = invert_ntt_at_layer(&mut zeta_i, re, 4);
re = invert_ntt_at_layer(&mut zeta_i, re, 5);
re = invert_ntt_at_layer(&mut zeta_i, re, 6);
re = invert_ntt_at_layer(&mut zeta_i, re, 7);

// Only reduce the first two entries
for i in 0..2 {
    re.coefficients[i] = barrett_reduce(re.coefficients[i]);
}
```

Verify when Review
& Exhaustive Testing
is impossible

Verify to optimize with confidence

How? 🤨

“[...] correctness is defined as the ability of a piece of software to meet a specific [...] requirement”

Algorithm 8 NTT(f)

Computes the NTT representation $\hat{f}$ of the given polynomial $f \in R_q$.

Input: array $f \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the input polynomial

Output: array $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the NTT of the input polynomial

1: $\hat{f} \leftarrow f$ ▷ will compute NTT in-place on a copy of input array
2: $k \leftarrow 1$
3: **for** ($len \leftarrow 128$; $len \geq 2$; $len \leftarrow len/2$)
4: **for** ($start \leftarrow 0$; $start < 256$; $start \leftarrow start + 2 \cdot len$)
5: $zeta \leftarrow \zeta^{\text{BitRev}_7(k)} \bmod q$
6: $k \leftarrow k + 1$
7: **for** ($j \leftarrow start$; $j < start + len$; j++)
8: $t \leftarrow zeta \cdot \hat{f}[j + len]$ ▷ steps 8-10 done modulo q
9: $\hat{f}[j + len] \leftarrow \hat{f}[j] - t$
10: $\hat{f}[j] \leftarrow \hat{f}[j] + t$
11: **end for**
12: **end for**
13: **end for**
14: **return** $\hat{f}$

Algorithm 8 $\mathsf{NTT}(f)$

Computes the NTT representation $\hat{f}$ of the given polynomial $f \in R_q$.

Input: array $f \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the input polynomial

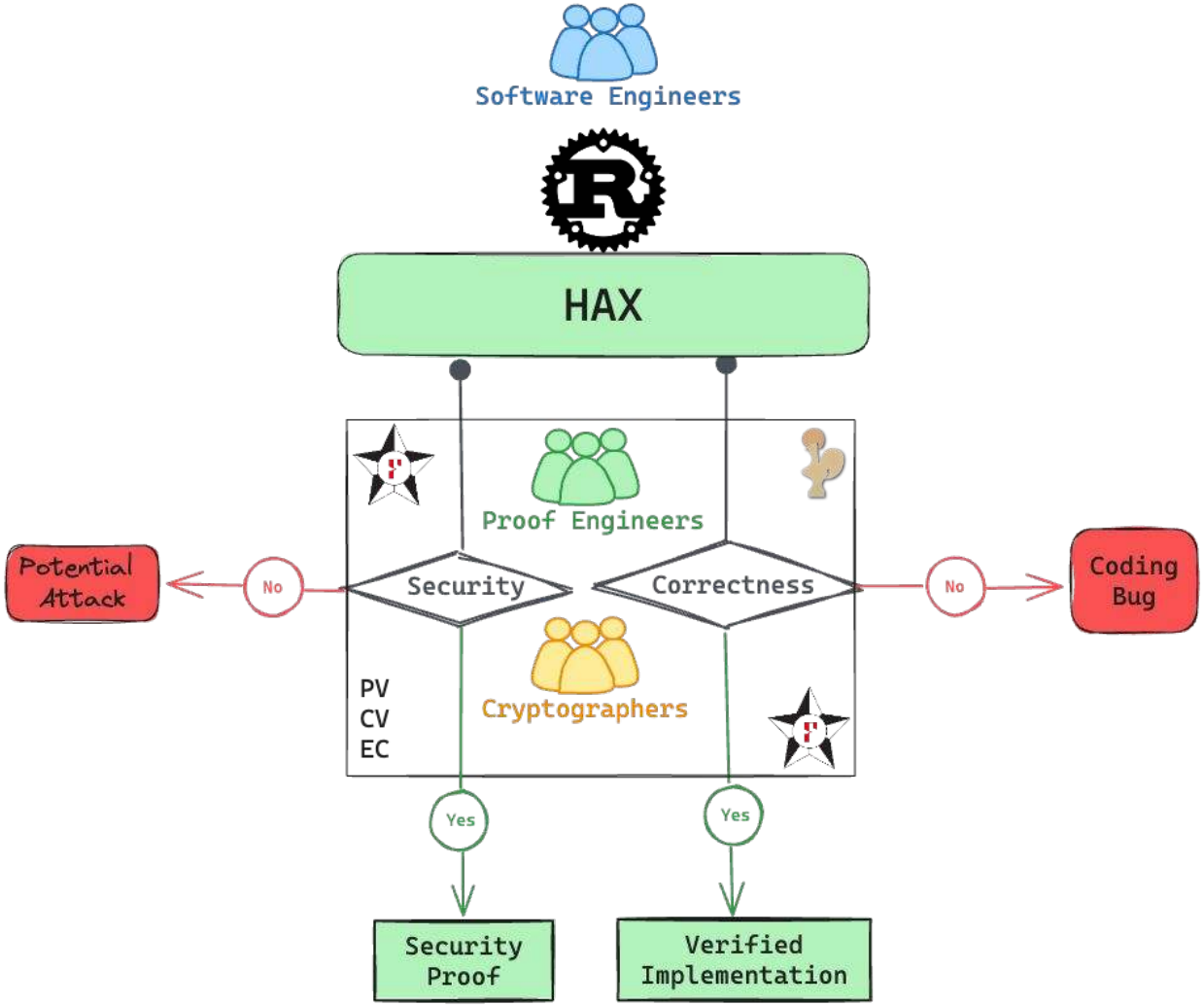
Output: array $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the NTT of the input polynomial

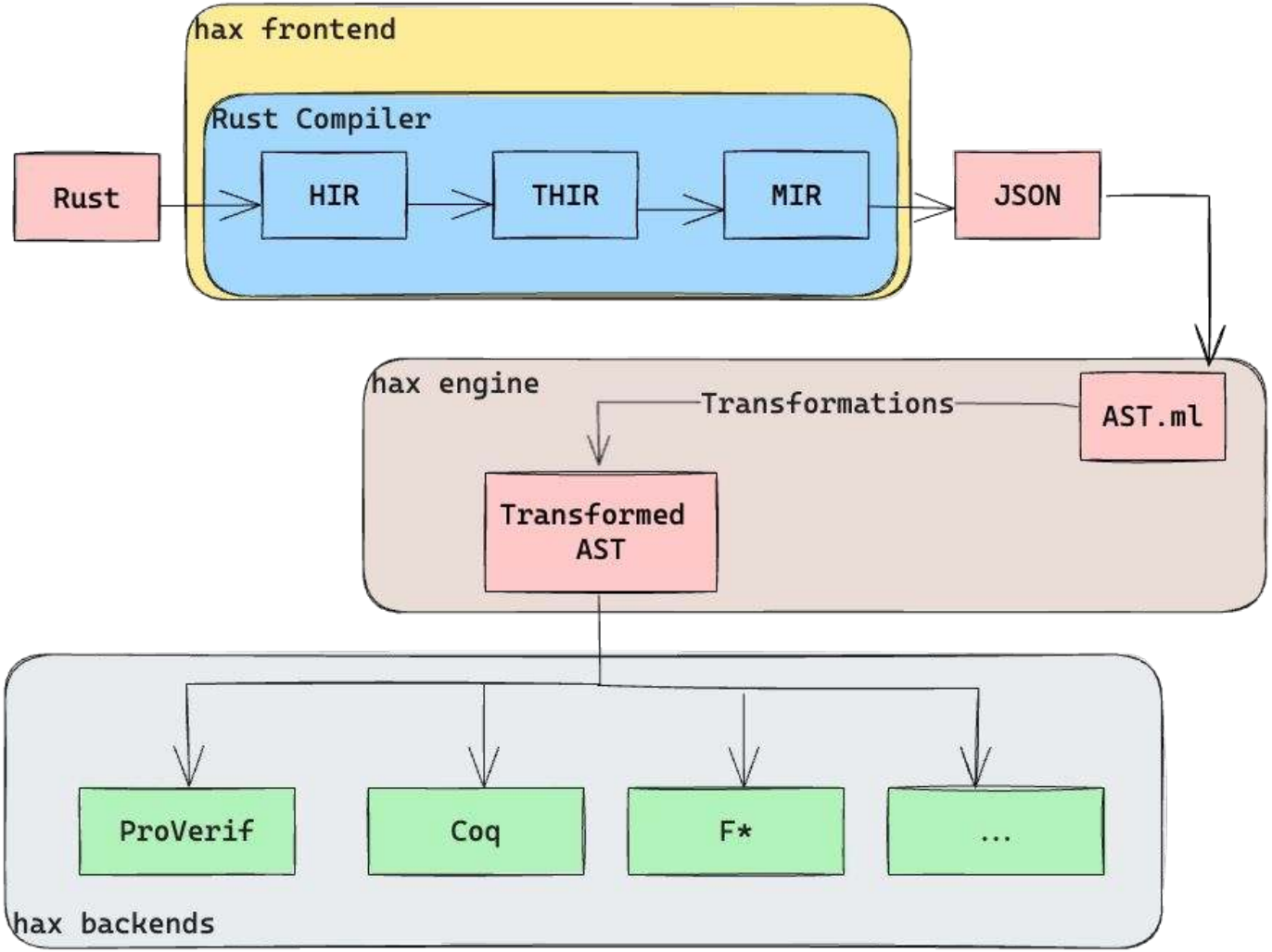
$\hat{f} \leftarrow f$ ▷ will compute NTT in-place on a copy of input array
$k \leftarrow 1$
for ($len \leftarrow 128$; $len \geq 2$; $len \leftarrow len/2$)
  for ($start \leftarrow 0$; $start < 256$; $start \leftarrow start + 2 \cdot len$)
    $zeta \leftarrow \zeta^{\mathsf{BitRev}_7(k)} \bmod q$
    $k \leftarrow k + 1$
    for ($j \leftarrow start$; $j < start + len$; j++)
      $t \leftarrow zeta \cdot \hat{f}[j + len]$
      $\hat{f}[j + len] \leftarrow \hat{f}[j] - t$
      $\hat{f}[j] \leftarrow \hat{f}[j] + t$
    end for
  end for
end for
return $\hat{f}$

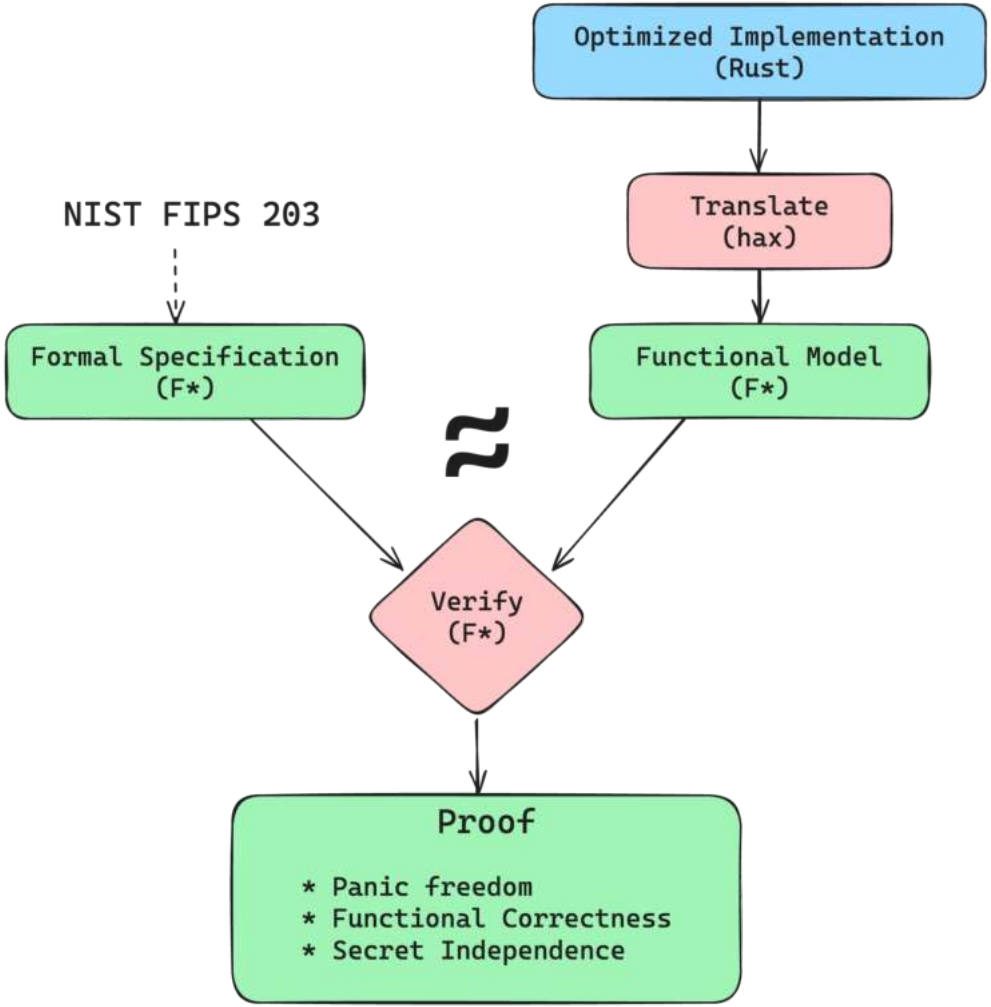
```
for round in 0..(128 >> layer) {
    *zeta_i += 1;

    let offset = round * step * 2;

    for j in offset..offset + step {
        let t = montgomery_multiply_fe_by_fer(
            re.coefficients[j + step],
            ZETAS_TIMES_MONTGOMERY_R[*zeta_i],
        );
        re.coefficients[j + step] = re.coefficients[j] - t;
        re.coefficients[j] = re.coefficients[j] + t;
    }
}
```

hax

High Assurance Translations


Software Engineers
HAX
Proof Engineers
Potential Attack
No
Security
Correctness
No
Coding Bug
PV
CV
EC
Cryptographers
Yes
Yes
Security Proof
Verified Implementation


hax frontend
Rust Compiler
Rust
HIR
THIR
MIR
JSON
hax engine
Transformations
AST.ml
Transformed AST
ProVerif
Coq
F*
...
hax backends


Optimized Implementation
(Rust)
Translate
(hax)
NIST FIPS 203
Formal Specification
(F*)
Functional Model
(F*)
≈
Verify
(F*)
Proof
* Panic freedom
* Functional Correctness
* Secret Independence

hax: A Tool Framework for Rust Verification

- Accepts a largish (and expanding) subset of safe Rust
  - Including hacspec, a purely functional spec language in Rust
- Translates it to formal models in F* or Coq
  - Upcoming backends for EasyCrypt, ProVerif, Lean, ...
- Usable and pragmatic design choices, not dogmatic
- Verify **panic-freedom**, **correctness**, **security**,...
  for the Rust code *you* care about,
  using the tool of *your choice*.

hax: ongoing projects

- **Rust Core:** an annotated version of the Rust Core library
- **Backends:** new backends for Lean, EasyCrypt, ProVerif

- **Verified**
  - **PQ Crypto:** verified Rust code for Kyber/ML-KEM, ...
  - **OS Modules:** verified kernel code for RIOT-OS
  - **Protocols:** verified code for EDHOC, MLS, TLS 1.3, ...
  - **Contracts:** verified canisters for Internet Computer

hax: Ongoing Challenges & Future Work

- Handle all the Rust code we want to handle
- Improve push-button verification
- Annotations in the proof assistant
- Proof assistant error messages
- IDE integration
- Correctness arguments of translations (research)

Can I use hax?

YES

Will there be bugs  ?

YES

core

1.77.0

(aedd173a2 2024-03-17)

All Items

Click or press ‘S’ to search, ‘?’ for

Crate core

[–] The Rust Core Library

hax: Ongoing Challenges & Future Work

- Handle all the Rust code we want to handle
- Improve push-button verification
- Annotations in the proof assistant
- Proof assistant error messages
- IDE integration
- Correctness arguments of translations (research)

hax: Process

```c
// reduce_once reduces 0 ⩽ x < 2*kPrime, mod kPrime.
static uint16_t reduce_once(uint16_t x) {
  assert(x < 2 * kPrime);
  const uint16_t subtracted = x - kPrime;
  uint16_t mask = 0u - (subtracted >> 15);
  // On Aarch64, omitting a |value_barrier_u16| results in a 2x speedup of Kyber
  // overall and Clang still produces constant-time code using `csel`. On other
  // platforms & compilers on godbolt that we care about, this code also
  // produces constant-time output.
  return (mask & x) | (~mask & subtracted);
}
```

```c
// constant time reduce x mod kPrime using Barrett reduction. x must be less
// than kPrime + 2×kPrime².
static uint16_t reduce(uint32_t x) {
  assert(x < kPrime + 2u * kPrime * kPrime);
  uint64_t product = (uint64_t)x * kBarrettMultiplier;
  uint32_t quotient = (uint32_t)(product >> kBarrettShift);
  uint32_t remainder = x - quotient * kPrime;
  return reduce_once(remainder);
}
```

hax: Process

```c
// reduce_once reduces 0 ⩽ x < 2*kPrime, mod kPrime.
static uint16_t reduce_once(uint16_t x) {
  assert(x < 2 * kPrime);
  const uint16_t subtracted = x - kPrime;
  uint16_t mask = 0u - (subtracted >> 15);
  // On Aarch64, omitting a |value_barrier_u16| results in a 2x speedup of Kyber
  // overall and Clang still produces constant-time code using `csel`. On other
  // platforms & compilers on godbolt that we care about, this code also
  // produces constant-time output.
  return (mask & x) | (~mask & subtracted);
}

// constant time reduce x mod kPrime using Barrett reduction. x must be less
// than kPrime + 2×kPrime².
static uint16_t reduce(uint32_t x) {
  assert(x < kPrime + 2u * kPrime * kPrime);
  uint64_t product = (uint64_t)x * kBarrettMultiplier;
  uint32_t quotient = (uint32_t)(product >> kBarrettShift);
  uint32_t remainder = x - quotient * kPrime;
  return reduce_once(remainder);
}
```

hax: Process

1. Asserts
2. Make the requirements formal
3. hax attributes for “design by contract”
4. F* statically checks that the properties hold

hax: Process

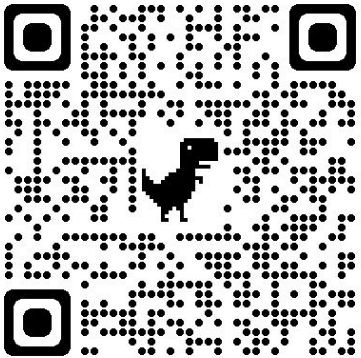
```
#[requires(coefficient_bits ≤ 11 && i32::from(fe) ≤ FIELD_MODULUS)]
#[ensures(|result| result ≥ 0 && result ≤ (1 << coefficient_bits) - 1)]
pub(super) fn compress_q(coefficient_bits: usize, fe: u16) → KyberFieldElement {
    let mut compressed: u32 = (fe as u32) << (coefficient_bits + 1);
    compressed += FIELD_MODULUS as u32;
    compressed ⧸= (FIELD_MODULUS << 1) as u32;

    (compressed & ((1u32 << coefficient_bits) - 1)) as KyberFieldElement
}
```

1. Asserts
2. Make the requirements formal
3. hax attributes for “design by contract”
4. F* statically checks that the properties hold

Verifying Libcrux's ML-KEM

Karthikeyan Bhargavan, Lucas Franceschino, Franziskus Kiefer, Goutam Tamvada

January 30, 2024



https://cryspen.com/post/ml-kem-verification/

Ensure specification is correct

- Specification is ground truth
- Manual inspection
  - Spec must be succinct & easy to read
- Tests
  - Test vectors
  - Test against other implementations

What do we prove?

- Panic freedom
- Secret independence
- Correctness (spec equivalence)
- Other properties when desirable
  - Examples

```
deserialize(serialize(x)) == x

decrypt(encrypt(y)) == y
```

Demo & Hands-on

Tasks

- Extract F*
- Lax typecheck F*
- Typecheck F* (panic freedom)
- Modify pre-conditions in Rust
  (and try to typecheck again)

Examples

- Chacha20
- Barrett reduction
- Compression in ML-KEM
- Write your own code!

Get together in small groups

Start docker and play with the examples

~/hax/examples

EXPLORER

OPEN EDITORS
- Welcome
- lib.rs examples/chacha20/...

UNTITLED (WORKSPACE)
- hax
  - .cache
  - .docker
  - .github
  - .hints
  - .utils
  - cli
  - engine
  - examples
    - chacha20
      - proofs
        - coq / extraction
        - fstar / extraction
        - hax.fst.config.json
        - Makefile
      - src
        - hacspec_helper.rs
        - lib.rs
      - tests
      - Cargo.toml
      - Makefile
    - limited-order-book
    - sha256
    - target
    - Cargo.lock
    - Cargo.toml
    - default.nix
    - hax.fst.config.json
    - Makefile
    - README.md
  - frontend
  - hax-lib
  - hax-lib-macros
  - proof-libs
  - target
  - test-harness
  - tests

OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE

Welcome | lib.rs

hax > examples > chacha20 > src > lib.rs > { } hacspec_helper

You, 19 hours ago | 2 authors (Lucas Franceschino and others)

```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 16 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State) -> State
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}
```

Lucas Franceschino, 4 weeks ago via PR #287 • feat(examples): init w

zsh - chacha20

~/repos/hax/examples/chacha20 main ±

EXPLORER

OPEN EDITORS
- Welcome
- lib.rs examples/chacha20/...

UNTITLED (WORKSPACE)
- hax
  - .cache
  - .docker
  - .github
  - .hints
  - .utils
  - cli
  - engine
  - examples
    - chacha20
      - proofs
        - coq
        - fstar / extraction
          - Chacha20.fst M
          - Chacha20.Hacspe... M
          - hax.fst.config.json
          - Makefile
      - src
        - hacspec_helper.rs
        - lib.rs
      - tests
      - Cargo.toml
      - Makefile
    - limited-order-book
    - sha256
    - target
  - Cargo.lock
  - Cargo.toml
  - default.nix
  - hax.fst.config.json
  - Makefile
  - README.md
  - frontend
  - hax-lib
  - hax-lib-macros
  - proof-libs
  - target

OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE

Welcome | lib.rs

hax > examples > chacha20 > src > lib.rs > { } hacspec_helper

You, 19 hours ago | 2 authors (Lucas Franceschino and others)

```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 16 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State) -> State
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}
```

zsh - chacha20

```
~/repos/hax/examples/chacha20  main ±  cargo test
   Compiling chacha20 v0.1.0 (/Users/franziskus/repos/hax/examples/chacha20)
    Finished test [unoptimized + debuginfo] target(s) in 0.80s
     Running unittests src/lib.rs (/Users/franziskus/repos/hax/examples/target/debug/deps/chacha20-6312cd2a88bf9d55)

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running tests/kat.rs (/Users/franziskus/repos/hax/examples/target/debug/deps/kat-da1c246bb0221d83)

running 1 test
test test_kat ... ok
```

cargo hax into fstar

```
~/repos/hax/examples/chacha20  main ±  cargo hax into fstar
   Compiling proc-macro2 v1.0.69
   Compiling unicode-ident v1.0.12
   Compiling syn v1.0.109
   Compiling version_check v0.9.4
   Compiling libc v0.2.150
   Compiling serde v1.0.192
   Compiling serde_json v1.0.108
   Compiling ryu v1.0.15
   Compiling itoa v1.0.9
   Compiling schemars v0.8.16
   Compiling cfg-if v1.0.0
   Compiling dyn-clone v1.0.16
   Compiling hax-lib v0.1.0-pre.1 (/Users/franziskus/repos/hax/hax-lib)
   Compiling proc-macro-error-attr v1.0.4
   Compiling proc-macro-error v1.0.4
   Compiling quote v1.0.33
   Compiling syn v2.0.39
   Compiling getrandom v0.2.11
   Compiling uuid v1.5.0
   Compiling serde_derive_internals v0.26.0
   Compiling schemars_derive v0.8.16
   Compiling serde_derive v1.0.192
   Compiling hax-lib-macros-types v0.1.0-pre.1 (/Users/franziskus/repos/hax/hax-lib-macros/types)
   Compiling hax-lib-macros v0.1.0-pre.1 (/Users/franziskus/repos/hax/hax-lib-macros)
   Compiling chacha20 v0.1.0 (/Users/franziskus/repos/hax/examples/chacha20)
note: Writing file "/Users/franziskus/repos/hax/examples/chacha20/proofs/fstar/extraction/Chacha20.fst"

note: Writing file "/Users/franziskus/repos/hax/examples/chacha20/proofs/fstar/extraction/Chacha20.Hacspec_helper.fst"

    Finished dev [unoptimized + debuginfo] target(s) in 7.79s
~/repos/hax/examples/chacha20  main ±
```

EXPLORER

OPEN EDITORS
GROUP 1
lib.rs examples/chacha20/...
GROUP 2
Chacha20.fst exam... M

UNTITLED (WORKSPACE)
hax
.cache
.docker
.github
.hints
.utils
cli
engine
examples
chacha20
proofs
coq
fstar / extraction
Chacha20.fst M
Chacha20.Hacspe... M
hax.fst.config.json
Makefile
src
hacspec_helper.rs
lib.rs
tests
Cargo.toml
Makefile
limited-order-book
sha256
target
Cargo.lock
Cargo.toml
default.nix
hax.fst.config.json
Makefile
README.md
frontend
hax-lib
hax-lib-macros
OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE

lib.rs

hax > examples > chacha20 > src > lib.rs > { } hacspec_helper

You, 19 hours ago | 2 authors (Lucas Franceschino and others)

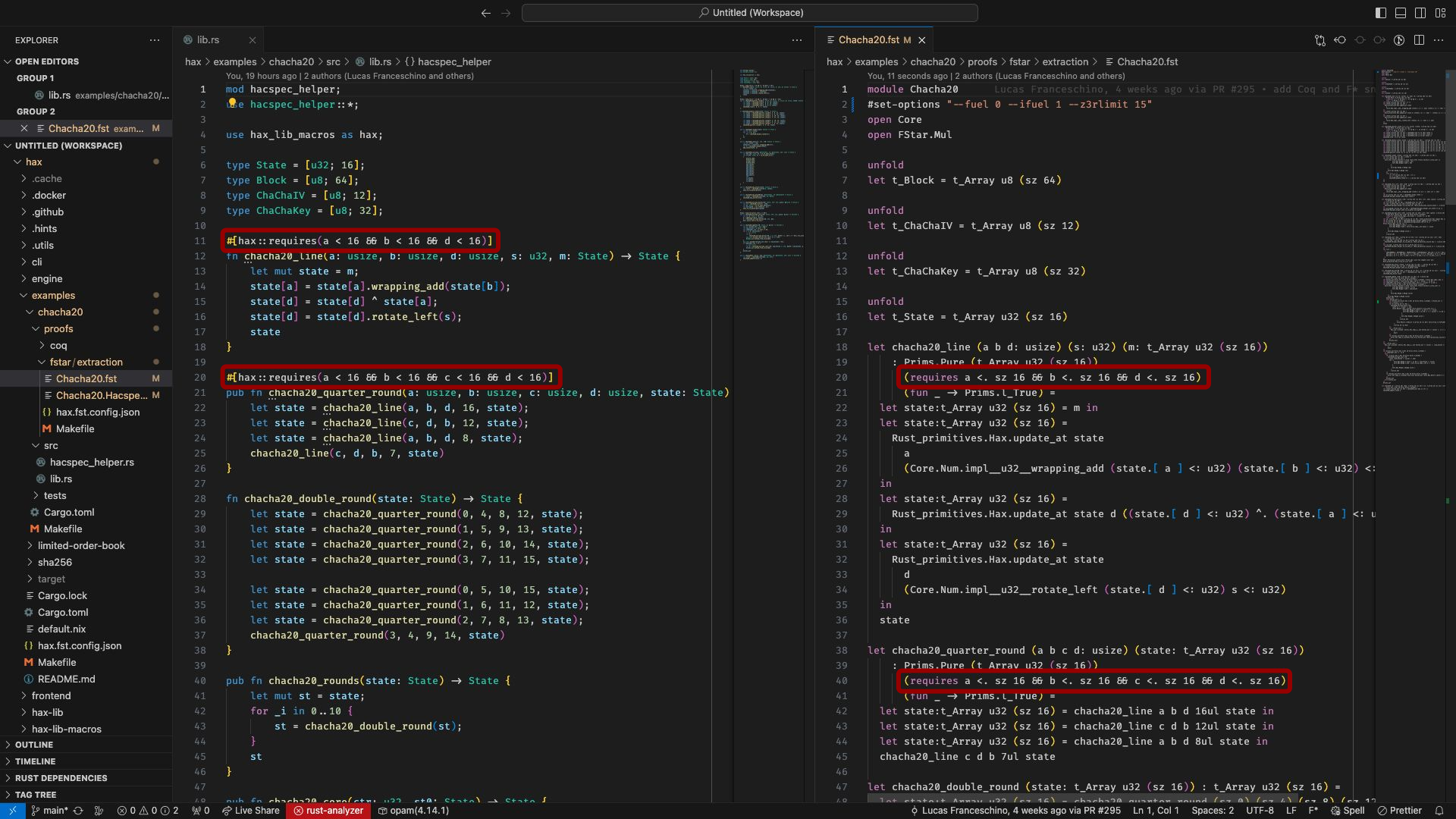
```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 16 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State)
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}
```

Chacha20.fst M

hax > examples > chacha20 > proofs > fstar > extraction > Chacha20.fst

You, 11 seconds ago | 2 authors (Lucas Franceschino and others)

```
module Chacha20      Lucas Franceschino, 4 weeks ago via PR #295 • add Coq and F* sr
#set-options "--fuel 0 --ifuel 1 --z3rlimit 15"
open Core
open FStar.Mul

unfold
let t_Block = t_Array u8 (sz 64)

unfold
let t_ChaChaIV = t_Array u8 (sz 12)

unfold
let t_ChaChaKey = t_Array u8 (sz 32)

unfold
let t_State = t_Array u32 (sz 16)

let chacha20_line (a b d: usize) (s: u32) (m: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 16 && b <. sz 16 && d <. sz 16)
      (fun _ -> Prims.l_True) =
  let state:t_Array u32 (sz 16) = m in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state
      a
      (Core.Num.impl__u32__wrapping_add (state.[ a ] <: u32) (state.[ b ] <: u32) <:
  in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state d ((state.[ d ] <: u32) ^. (state.[ a ] <: u
  in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state
      d
      (Core.Num.impl__u32__rotate_left (state.[ d ] <: u32) s <: u32)
  in
  state

let chacha20_quarter_round (a b c d: usize) (state: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 16 && b <. sz 16 && c <. sz 16 && d <. sz 16)
      (fun _ -> Prims.l_True) =
  let state:t_Array u32 (sz 16) = chacha20_line a b d 16ul state in
  let state:t_Array u32 (sz 16) = chacha20_line c d b 12ul state in
  let state:t_Array u32 (sz 16) = chacha20_line a b d 8ul state in
  chacha20_line c d b 7ul state

let chacha20_double_round (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
```


Untitled (Workspace)
EXPLORER
OPEN EDITORS
GROUP 1
lib.rs examples/chacha20/...
GROUP 2
Chacha20.fst exam... M
UNTITLED (WORKSPACE)
hax
.cache
.docker
.github
.hints
.utils
cli
engine
examples
chacha20
proofs
coq
fstar / extraction
Chacha20.fst M
Chacha20.Hacspe... M
hax.fst.config.json
Makefile
src
hacspec_helper.rs
lib.rs
tests
Cargo.toml
Makefile
limited-order-book
sha256
target
Cargo.lock
Cargo.toml
default.nix
hax.fst.config.json
Makefile
README.md
frontend
hax-lib
hax-lib-macros
OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE
lib.rs
hax > examples > chacha20 > src > lib.rs > { } hacspec_helper
You, 19 hours ago | 2 authors (Lucas Franceschino and others)
1 mod hacspec_helper;
2 use hacspec_helper::*;
3
4 use hax_lib_macros as hax;
5
6 type State = [u32; 16];
7 type Block = [u8; 64];
8 type ChaChaIV = [u8; 12];
9 type ChaChaKey = [u8; 32];
10
11 #[hax::requires(a < 16 && b < 16 && d < 16)]
12 fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
13 let mut state = m;
14 state[a] = state[a].wrapping_add(state[b]);
15 state[d] = state[d] ^ state[a];
16 state[d] = state[d].rotate_left(s);
17 state
18 }
19
20 #[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
21 pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State)
22 let state = chacha20_line(a, b, d, 16, state);
23 let state = chacha20_line(c, d, b, 12, state);
24 let state = chacha20_line(a, b, d, 8, state);
25 chacha20_line(c, d, b, 7, state)
26 }
27
28 fn chacha20_double_round(state: State) -> State {
29 let state = chacha20_quarter_round(0, 4, 8, 12, state);
30 let state = chacha20_quarter_round(1, 5, 9, 13, state);
31 let state = chacha20_quarter_round(2, 6, 10, 14, state);
32 let state = chacha20_quarter_round(3, 7, 11, 15, state);
33
34 let state = chacha20_quarter_round(0, 5, 10, 15, state);
35 let state = chacha20_quarter_round(1, 6, 11, 12, state);
36 let state = chacha20_quarter_round(2, 7, 8, 13, state);
37 chacha20_quarter_round(3, 4, 9, 14, state)
38 }
39
40 pub fn chacha20_rounds(state: State) -> State {
41 let mut st = state;
42 for _i in 0..10 {
43 st = chacha20_double_round(st);
44 }
45 st
46 }
47
Chacha20.fst M
hax > examples > chacha20 > proofs > fstar > extraction > Chacha20.fst
You, 11 seconds ago | 2 authors (Lucas Franceschino and others)
1 module Chacha20 Lucas Franceschino, 4 weeks ago via PR #295 • add Coq and F* sr
2 #set-options "--fuel 0 --ifuel 1 --z3rlimit 15"
3 open Core
4 open FStar.Mul
5
6 unfold
7 let t_Block = t_Array u8 (sz 64)
8
9 unfold
10 let t_ChaChaIV = t_Array u8 (sz 12)
11
12 unfold
13 let t_ChaChaKey = t_Array u8 (sz 32)
14
15 unfold
16 let t_State = t_Array u32 (sz 16)
17
18 let chacha20_line (a b d: usize) (s: u32) (m: t_Array u32 (sz 16))
19 : Prims.Pure (t_Array u32 (sz 16))
20 (requires a <. sz 16 && b <. sz 16 && d <. sz 16)
21 (fun _ -> Prims.l_True) =
22 let state:t_Array u32 (sz 16) = m in
23 let state:t_Array u32 (sz 16) =
24 Rust_primitives.Hax.update_at state
25 a
26 (Core.Num.impl__u32__wrapping_add (state.[a] <: u32) (state.[b] <: u32) <:
27 in
28 let state:t_Array u32 (sz 16) =
29 Rust_primitives.Hax.update_at state d ((state.[d] <: u32) ^. (state.[a] <: u
30 in
31 let state:t_Array u32 (sz 16) =
32 Rust_primitives.Hax.update_at state
33 d
34 (Core.Num.impl__u32__rotate_left (state.[d] <: u32) s <: u32)
35 in
36 state
37
38 let chacha20_quarter_round (a b c d: usize) (state: t_Array u32 (sz 16))
39 : Prims.Pure (t_Array u32 (sz 16))
40 (requires a <. sz 16 && b <. sz 16 && c <. sz 16 && d <. sz 16)
41 (fun _ -> Prims.l_True) =
42 let state:t_Array u32 (sz 16) = chacha20_line a b d 16ul state in
43 let state:t_Array u32 (sz 16) = chacha20_line c d b 12ul state in
44 let state:t_Array u32 (sz 16) = chacha20_line a b d 8ul state in
45 chacha20_line c d b 7ul state
46
47 let chacha20_double_round (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
main*
0 0 0 2
0
Live Share
rust-analyzer
opam(4.14.1)
Lucas Franceschino, 4 weeks ago via PR #295
Ln 1, Col 1
Spaces: 2
UTF-8
LF
F*
Spell
Prettier

EXPLORER

OPEN EDITORS
GROUP 1
lib.rs examples/chacha20/...
GROUP 2
Chacha20.fst exam... M

UNTITLED (WORKSPACE)
hax
.cache
.docker
.github
.hints
.utils
cli
engine
examples
chacha20
proofs
coq
fstar / extraction
Chacha20.fst M
Chacha20.Hacspe... M
hax.fst.config.json
Makefile
src
hacspec_helper.rs
lib.rs
tests
Cargo.toml
Makefile
limited-order-book
sha256
target
Cargo.lock
Cargo.toml
default.nix
hax.fst.config.json
Makefile
README.md
frontend
hax-lib
hax-lib-macros
OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE

lib.rs

hax > examples > chacha20 > src > lib.rs > { } hacspec_helper

You, 19 hours ago | 2 authors (Lucas Franceschino and others)

```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 16 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State)
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}
```

Chacha20.fst M

hax > examples > chacha20 > proofs > fstar > extraction > Chacha20.fst

```
  let chacha20_line (a b d: usize) (s: u32) (m: t_Array u32 (sz 16))

let chacha20_quarter_round (a b c d: usize) (state: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 16 && b <. sz 16 && c <. sz 16 && d <. sz 16)
      (fun _ -> Prims.l_True) =
  let state:t_Array u32 (sz 16) = chacha20_line a b d 16ul state in
  let state:t_Array u32 (sz 16) = chacha20_line c d b 12ul state in
  let state:t_Array u32 (sz 16) = chacha20_line a b d 8ul state in
  chacha20_line c d b 7ul state

let chacha20_double_round (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 0) (sz 4) (sz 8) (sz 12
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 1) (sz 5) (sz 9) (sz 13
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 2) (sz 6) (sz 10) (sz 1
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 3) (sz 7) (sz 11) (sz 1
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 0) (sz 5) (sz 10) (sz 1
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 1) (sz 6) (sz 11) (sz 1
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 2) (sz 7) (sz 8) (sz 13
  chacha20_quarter_round (sz 3) (sz 4) (sz 9) (sz 14) state

let chacha20_rounds (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
  let st:t_Array u32 (sz 16) = state in
  let st:t_Array u32 (sz 16) =
    Core.Iter.Traits.Iterator.f_fold (Core.Iter.Traits.Collect.f_into_iter ({
            Core.Ops.Range.f_start = 0l;
            Core.Ops.Range.f_end = 10l
          }
          <:
          Core.Ops.Range.t_Range i32)
      <:
      Core.Ops.Range.t_Range i32)
    st
    (fun st v__i ->
      let st:t_Array u32 (sz 16) = st in
      let v__i:i32 = v__i in
      chacha20_double_round st <: t_Array u32 (sz 16))
  in
  st

let chacha20_core (ctr: u32) (st0: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
  let state:t_Array u32 (sz 16) = st0 in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state
    (sz 12)
    (Core.Num.impl__u32__wrapping_add (state.[ sz 12 ] <: u32) ctr <: u32)
  in
  let k:t_Array u32 (sz 16) = chacha20_rounds state in
  Chacha20.Hacspec_helper.add state state k
```

Untitled (Workspace)

lib.rs M

hax > examples > chacha20 > src > lib.rs > chacha20_line

You, 1 second ago | 2 authors (Lucas Franceschino and others)

```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 17 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 16 && b < 16 && c < 16 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State) -> State {
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}

pub fn chacha20_core(ctr: u32, st0: State) -> State {
```

Chacha20.fst 3, M

hax > examples > chacha20 > proofs > fstar > extraction > Chacha20.fst

You, 2 seconds ago | 2 authors (Lucas Franceschino and others)

```
module Chacha20
#set-options "--fuel 0 --ifuel 1 --z3rlimit 15"
open Core
open FStar.Mul

unfold
let t_Block = t_Array u8 (sz 64)

unfold
let t_ChaChaIV = t_Array u8 (sz 12)

unfold
let t_
```

- Subtyping check failed; expected type i: usize{Core.Ops.Index.in_range state i}; got type usize
  - See also /Users/franziskus/repos/hax/proof-libs/fstar/rust_primitives/Rust_primitives.Hax.fst(16,32-16,44)

Rust_primitives.Hax.fst(16, 33): related location

- Subtyping check failed; expected type i: usize{Core.Ops.Index.in_range state i}; got type usize
  - SMT solver says:
    unknown because (incomplete quantifiers) (fuel=0; ifuel=1)
    Note:
    - 'canceled' or 'resource limits reached' means the SMT query timed

```
unfold
let ch
    :
    let
    let
      Ru
        a
        (Core.Num.impl__u32__wrapping_add (state.[ a ] <: u32) (state.[ b ] <: u32) <: u32)
    in
    let state:t_Array u32 (sz 16) =
      Rust_primitives.Hax.update_at state d ((state.[ d ] <: u32) ^. (state.[ a ] <: u32) <: u32)
    in
    let state:t_Array u32 (sz 16) =
      Rust_primitives.Hax.update_at state
        d
        (Core.Num.impl__u32__rotate_left (state.[ d ] <: u32) s <: u32)
    in
    state

let chacha20_quarter_round (a b c d: usize) (state: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 16 && b <. sz 16 && c <. sz 16 && d <. sz 16)
      (fun _ -> Prims.l_True) =
    let state:t_Array u32 (sz 16) = chacha20_line a b d 16ul state in
    let state:t_Array u32 (sz 16) = chacha20_line c d b 12ul state in
    let state:t_Array u32 (sz 16) = chacha20_line a b d 8ul state in
    chacha20_line c d b 7ul state

let chacha20_double_round (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
```

lib.rs M

hax > examples > chacha20 > src > lib.rs > chacha20_quarter_round

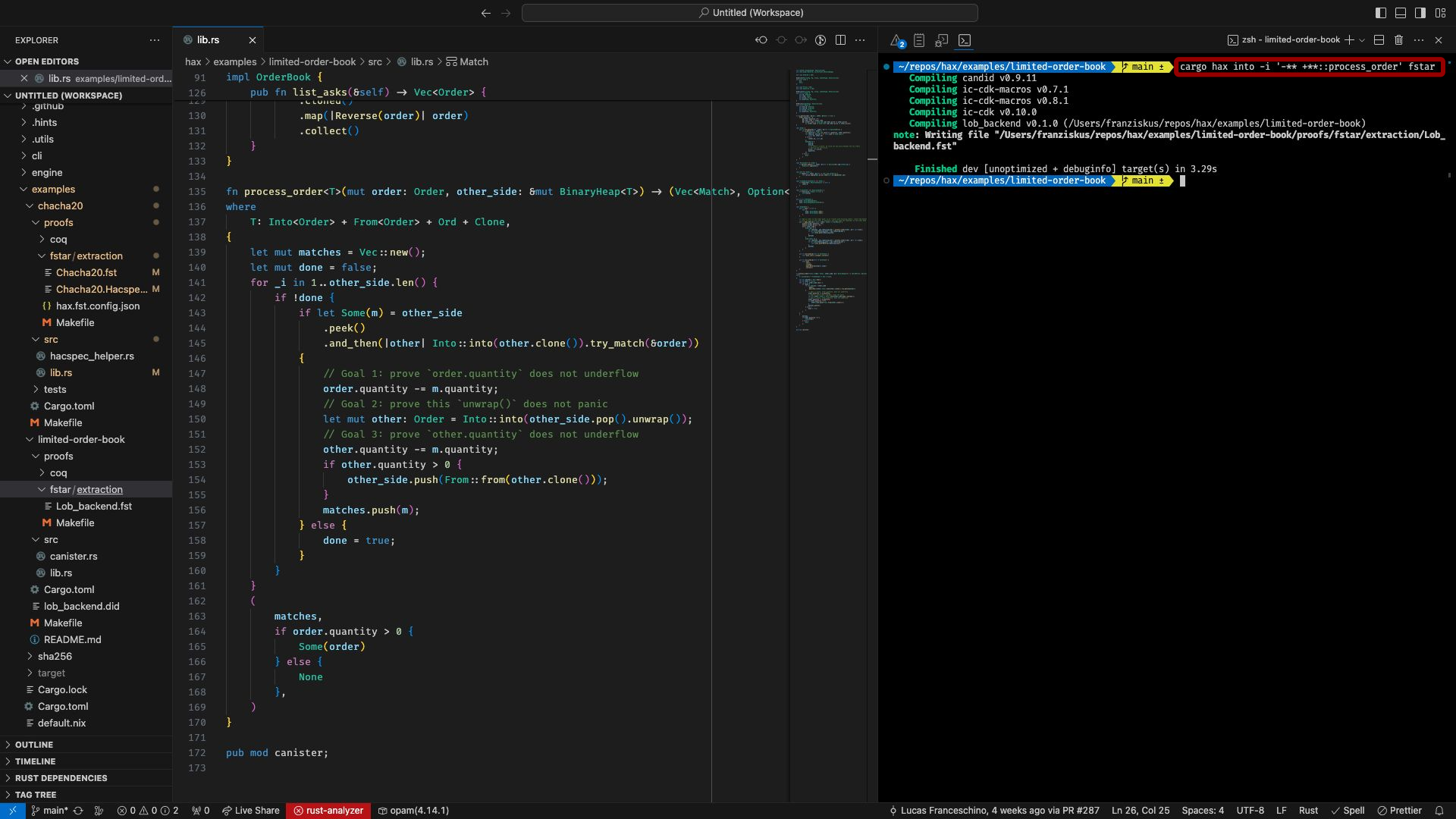
You, 1 second ago | 2 authors (Lucas Franceschino and others)

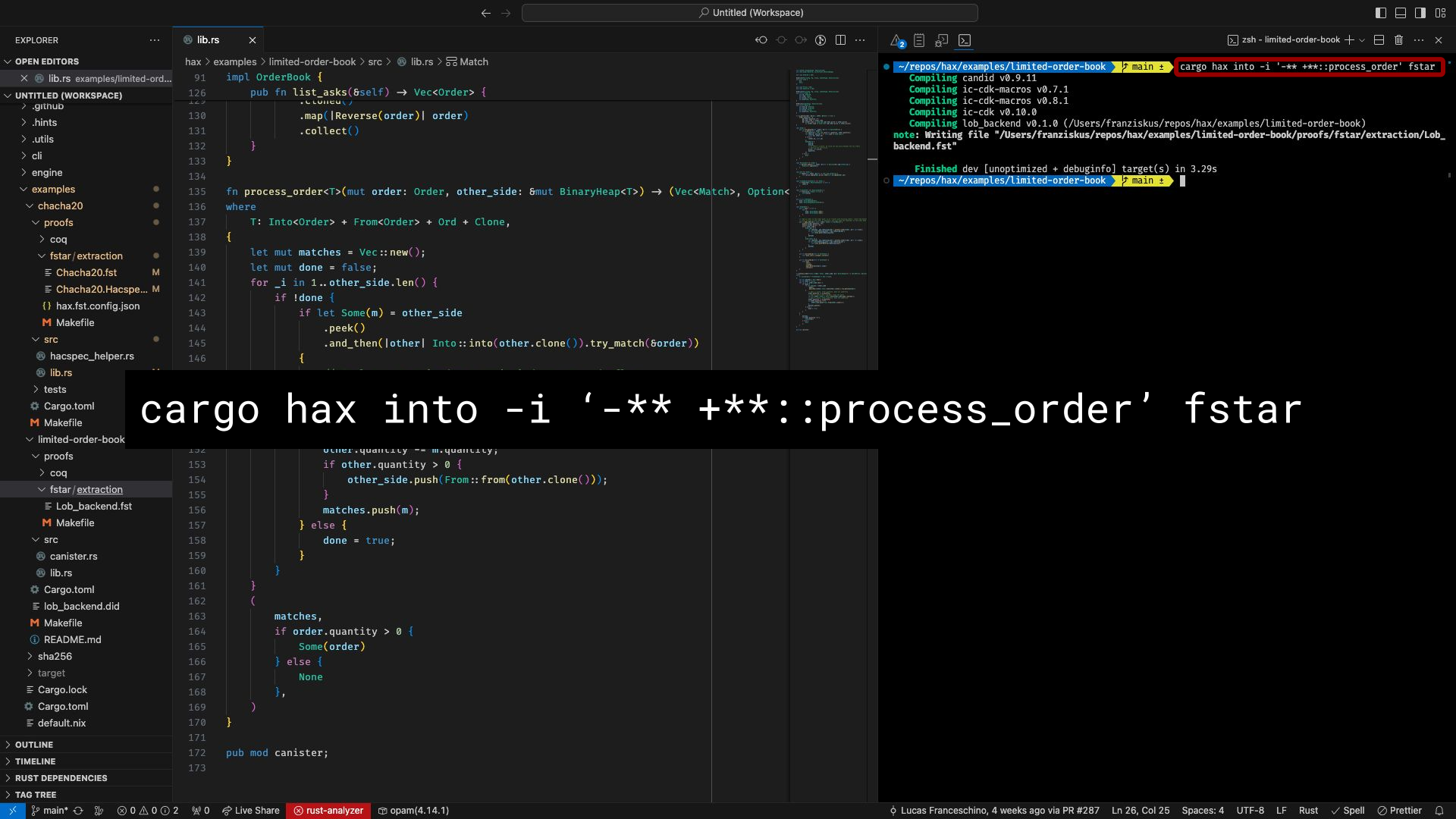
```rust
mod hacspec_helper;
use hacspec_helper::*;

use hax_lib_macros as hax;

type State = [u32; 16];
type Block = [u8; 64];
type ChaChaIV = [u8; 12];
type ChaChaKey = [u8; 32];

#[hax::requires(a < 12 && b < 16 && d < 16)]
fn chacha20_line(a: usize, b: usize, d: usize, s: u32, m: State) -> State {
    let mut state = m;
    state[a] = state[a].wrapping_add(state[b]);
    state[d] = state[d] ^ state[a];
    state[d] = state[d].rotate_left(s);
    state
}

#[hax::requires(a < 12 && b < 16 && c < 12 && d < 16)]
pub fn chacha20_quarter_round(a: usize, b: usize, c: usize, d: usize, state: State) -> State {
    let state = chacha20_line(a, b, d, 16, state);
    let state = chacha20_line(c, d, b, 12, state);
    let state = chacha20_line(a, b, d, 8, state);
    chacha20_line(c, d, b, 7, state)
}

fn chacha20_double_round(state: State) -> State {
    let state = chacha20_quarter_round(0, 4, 8, 12, state);
    let state = chacha20_quarter_round(1, 5, 9, 13, state);
    let state = chacha20_quarter_round(2, 6, 10, 14, state);
    let state = chacha20_quarter_round(3, 7, 11, 15, state);

    let state = chacha20_quarter_round(0, 5, 10, 15, state);
    let state = chacha20_quarter_round(1, 6, 11, 12, state);
    let state = chacha20_quarter_round(2, 7, 8, 13, state);
    chacha20_quarter_round(3, 4, 9, 14, state)
}

pub fn chacha20_rounds(state: State) -> State {
    let mut st = state;
    for _i in 0..10 {
        st = chacha20_double_round(st);
    }
    st
}

pub fn chacha20_core(ctr: u32, st0: State) -> State {
```

Chacha20.fst M

hax > examples > chacha20 > proofs > fstar > extraction > Chacha20.fst

```
unfold
let t_Block = t_Array u8 (sz 64)

unfold
let t_ChaChaIV = t_Array u8 (sz 12)

unfold
let t_ChaChaKey = t_Array u8 (sz 32)

unfold
let t_State = t_Array u32 (sz 16)

let chacha20_line (a b d: usize) (s: u32) (m: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 12 && b <. sz 16 && d <. sz 16)
      (fun _ -> Prims.l_True) =
  let state:t_Array u32 (sz 16) = m in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state
      a
      (Core.Num.impl__u32__wrapping_add (state.[ a ] <: u32) (state.[ b ] <: u32) <: u32)
  in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state d ((state.[ d ] <: u32) ^. (state.[ a ] <: u32) <: u32)
  in
  let state:t_Array u32 (sz 16) =
    Rust_primitives.Hax.update_at state
      d
      (Core.Num.impl__u32__rotate_left (state.[ d ] <: u32) s <: u32)
  in
  state

let chacha20_quarter_round (a b c d: usize) (state: t_Array u32 (sz 16))
    : Prims.Pure (t_Array u32 (sz 16))
      (requires a <. sz 12 && b <. sz 16 && c <. sz 12 && d <. sz 16)
      (fun _ -> Prims.l_True) =
  let state:t_Array u32 (sz 16) = chacha20_line a b d 16ul state in
  let state:t_Array u32 (sz 16) = chacha20_line c d b 12ul state in
  let state:t_Array u32 (sz 16) = chacha20_line a b d 8ul state in
  chacha20_line c d b 7ul state
      Lucas Franceschino, 4 weeks ago via PR #295 • add Coq and F* snapshots
let chacha20_double_round (state: t_Array u32 (sz 16)) : t_Array u32 (sz 16) =
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 0) (sz 4) (sz 8) (sz 12) state in
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 1) (sz 5) (sz 9) (sz 13) state in
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 2) (sz 6) (sz 10) (sz 14) state in
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 3) (sz 7) (sz 11) (sz 15) state in
  let state:t_Array u32 (sz 16) = chacha20_quarter_round (sz 0) (sz 5) (sz 10) (sz 15) state in
```


Untitled (Workspace)
EXPLORER
OPEN EDITORS
lib.rs examples/limited-ord...
UNTITLED (WORKSPACE)
.github
.hints
.utils
cli
engine
examples
chacha20
proofs
coq
fstar / extraction
Chacha20.fst M
Chacha20.Hacspe... M
hax.fst.config.json
Makefile
src
hacspec_helper.rs
lib.rs M
tests
Cargo.toml
Makefile
limited-order-book
proofs
coq
fstar / extraction
Lob_backend.fst
Makefile
src
canister.rs
lib.rs
Cargo.toml
lob_backend.did
Makefile
README.md
sha256
target
Cargo.lock
Cargo.toml
default.nix
OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE
lib.rs
hax > examples > limited-order-book > src > lib.rs > Match
impl OrderBook {
pub fn list_asks(&self) -> Vec<Order> {
.map(|Reverse(order)| order)
.collect()
}
}
fn process_order<T>(mut order: Order, other_side: &mut BinaryHeap<T>) -> (Vec<Match>, Option<
where
T: Into<Order> + From<Order> + Ord + Clone,
{
let mut matches = Vec::new();
let mut done = false;
for _i in 1..other_side.len() {
if !done {
if let Some(m) = other_side
.peek()
.and_then(|other| Into::into(other.clone()).try_match(&order))
{
// Goal 1: prove `order.quantity` does not underflow
order.quantity -= m.quantity;
// Goal 2: prove this `unwrap()` does not panic
let mut other: Order = Into::into(other_side.pop().unwrap());
// Goal 3: prove `other.quantity` does not underflow
other.quantity -= m.quantity;
if other.quantity > 0 {
other_side.push(From::from(other.clone()));
}
matches.push(m);
} else {
done = true;
}
}
}
(
matches,
if order.quantity > 0 {
Some(order)
} else {
None
},
)
}
pub mod canister;
zsh - limited-order-book
~/repos/hax/examples/limited-order-book ? main ±
cargo hax into -i '-** +**::process_order' fstar
Compiling candid v0.9.11
Compiling ic-cdk-macros v0.7.1
Compiling ic-cdk-macros v0.8.1
Compiling ic-cdk v0.10.0
Compiling lob_backend v0.1.0 (/Users/franziskus/repos/hax/examples/limited-order-book)
note: Writing file "/Users/franziskus/repos/hax/examples/limited-order-book/proofs/fstar/extraction/Lob_backend.fst"
Finished dev [unoptimized + debuginfo] target(s) in 3.29s
~/repos/hax/examples/limited-order-book ? main ±
main*
0 0 0 2
0
Live Share
rust-analyzer
opam(4.14.1)
Lucas Franceschino, 4 weeks ago via PR #287
Ln 26, Col 25
Spaces: 4
UTF-8
LF
Rust
Spell
Prettier


Untitled (Workspace)
EXPLORER
OPEN EDITORS
lib.rs examples/limited-ord...
UNTITLED (WORKSPACE)
.github
.hints
.utils
cli
engine
examples
chacha20
proofs
coq
fstar / extraction
Chacha20.fst M
Chacha20.Hacspe... M
hax.fst.config.json
Makefile
src
hacspec_helper.rs
lib.rs
tests
Cargo.toml
Makefile
limited-order-book
proofs
coq
fstar / extraction
Lob_backend.fst
Makefile
src
canister.rs
lib.rs
Cargo.toml
lob_backend.did
Makefile
README.md
sha256
target
Cargo.lock
Cargo.toml
default.nix
OUTLINE
TIMELINE
RUST DEPENDENCIES
TAG TREE
lib.rs
hax > examples > limited-order-book > src > lib.rs > Match
impl OrderBook {
pub fn list_asks(&self) -> Vec<Order> {
.map(|Reverse(order)| order)
.collect()
}
}
fn process_order<T>(mut order: Order, other_side: &mut BinaryHeap<T>) -> (Vec<Match>, Option<
where
T: Into<Order> + From<Order> + Ord + Clone,
{
let mut matches = Vec::new();
let mut done = false;
for _i in 1..other_side.len() {
if !done {
if let Some(m) = other_side
.peek()
.and_then(|other| Into::into(other.clone()).try_match(&order))
{
if other.quantity > 0 {
other_side.push(From::from(other.clone()));
}
matches.push(m);
} else {
done = true;
}
}
}
(
matches,
if order.quantity > 0 {
Some(order)
} else {
None
},
)
}
pub mod canister;
zsh - limited-order-book
~/repos/hax/examples/limited-order-book ? main ± cargo hax into -i '-** +**::process_order' fstar
Compiling candid v0.9.11
Compiling ic-cdk-macros v0.7.1
Compiling ic-cdk-macros v0.8.1
Compiling ic-cdk v0.10.0
Compiling lob_backend v0.1.0 (/Users/franziskus/repos/hax/examples/limited-order-book)
note: Writing file "/Users/franziskus/repos/hax/examples/limited-order-book/proofs/fstar/extraction/Lob_backend.fst"
Finished dev [unoptimized + debuginfo] target(s) in 3.29s
~/repos/hax/examples/limited-order-book ? main ±
cargo hax into -i '-** +**::process_order' fstar
main* 0 0 2 0 Live Share rust-analyzer opam(4.14.1)
Lucas Franceschino, 4 weeks ago via PR #287 Ln 26, Col 25 Spaces: 4 UTF-8 LF Rust Spell Prettier

lib.rs M

hax > examples > limited-order-book > src > lib.rs > { } impl Order > try_match

Lucas Franceschino, 4 weeks ago | 1 author (Lucas Franceschino)

```rust
#[derive(CandidType, Deserialize)]
pub struct Match {
    pub bid_id: OrderId,
    pub ask_id: OrderId,
    pub price: Price,
    pub quantity: Quantity,
}

fn is_match(order: &Order, other: &Order) -> bool {
    order.quantity > 0
        && other.quantity > 0
        && order.side != other.side
        && ((order.side == Side::Buy && order.price >= other.price)
            || (order.side == Side::Sell && order.price <= other.price))
}

impl Order {
    pub fn try_match(&self, other: &Self) -> Option<Match> {
        if is_match(self, other) {
            let quantity = std::cmp::max(self.quantity, other.quantity);
            let (bid_id, ask_id) = if self.side == Side::Buy {
                (self.id, other.id)
            } else {
                (other.id, self.id)
            };
            Some(Match {
                bid_id,
                ask_id,
                // If there's a match, we could use any price between the two orders.
                // Here we use self.price.
                price: self.price,
                quantity,
            })
        } else {
            None
        }
    }
}

impl PartialOrd for Order {
    fn partial_cmp(&self, other: &Self) -> Option<std::cmp::Ordering> {
        Some(self.cmp(other))
    }
}

impl Ord for Order {
    fn cmp(&self, other: &Self) -> std::cmp::Ordering {
```

Lob_backend.fst 2, M | Core.Cmp.fsti 1, M

hax > examples > limited-order-book > proofs > fstar > extraction > Lob_backend.fst

```
let process_order
  let done, matches, order, other_side:(bool & Alloc.Vec.t_Vec t_Match Alloc.Alloc.t_Global &
    Core.Iter.Traits.Iterator.f_fold (Core.Iter.Traits.Collect.f_into_iter ({
      (fun temp_0_ v__i ->
        let v__i:usize = v__i in
        if ~.done <: bool
        then
          match
            Core.Option.impl__and_then (Alloc.Collections.Binary_heap.impl_10__peek other_side
                <:
                Core.Option.t_Option v_T)
              (fun other ->
                let other:v_T = other in
                impl__Order__try_match (Core.Convert.f_into (Core.Clone.f_clone other <: v_T
                    <:
                    t_Order)
                  order
                <:
                Core.Option.t_Option t_Match)
            <:
            Core.Option.t_Option t_Match
          with
          | Core.Option.Option_Some m ->
            let order:t_Order =
              { order with f_quantity = order.f_quantity - m.f_quantity } <: t_Order
            in
            let tmp0, out:(Alloc.Collections.Binary_heap.t_BinaryHeap v_T &
              Core.Option.t_Option v_T) =
              Alloc.Collections.Binary_heap.impl_9__pop other_side
            in
            let other_side:Alloc.Collections.Binary_heap.t_BinaryHeap v_T = tmp0 in
            let hoist1:Core.Option.t_Option v_T = out in
            let hoist2:v_T = Core.Option.impl__unwrap hoist1 in
            let (other: t_Order):t_Order = Core.Convert.f_into hoist2 in
            let other:t_Order =
              { other with f_quantity = other.f_quantity - m.f_quantity } <: t_Order
            in
            let other_side:Alloc.Collections.Binary_heap.t_BinaryHeap v_T =
              if other.f_quantity >. 0uL
              then
                let other_side:Alloc.Collections.Binary_heap.t_BinaryHeap v_T =
                  Alloc.Collections.Binary_heap.impl_9__push other_side
                    (Core.Convert.f_from (Core.Clone.f_clone other <: t_Order) <: v_T)
                in
                other_side
              else other_side
            in
            let matches:Alloc.Vec.t_Vec t_Match Alloc.Alloc.t_Global =
              Alloc.Vec.impl_1__push matches m
```

Thanks

https://github.com/hacspec/hax

CRYSPEN