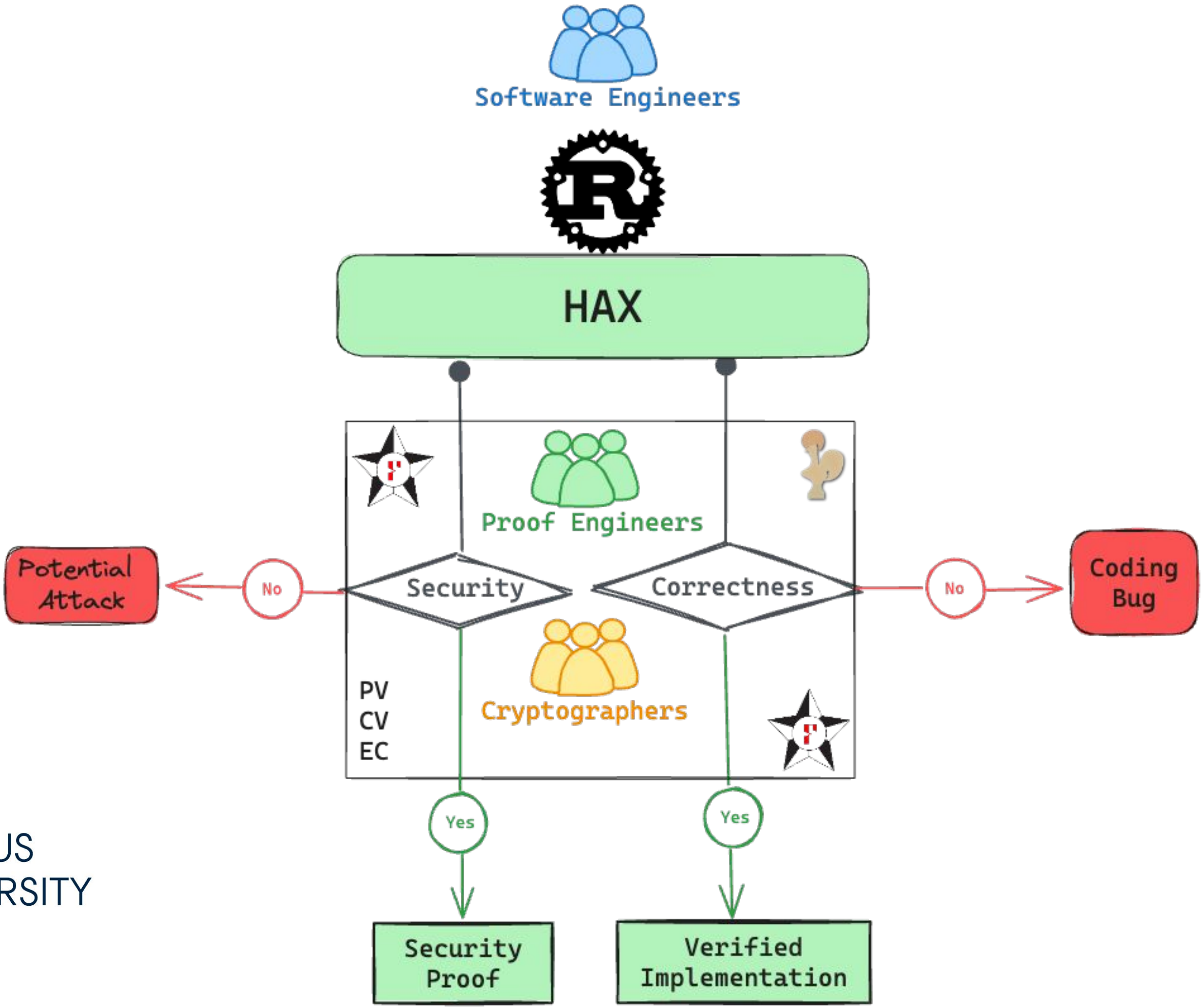
Enabling High Assurance Cryptographic Software

hax

Karthikeyan Bhargavan, Lucas Franceschino, Lasse Letager Hansen, Franziskus Kiefer, Jonas Schneider-Bensch, and Bas Spitters



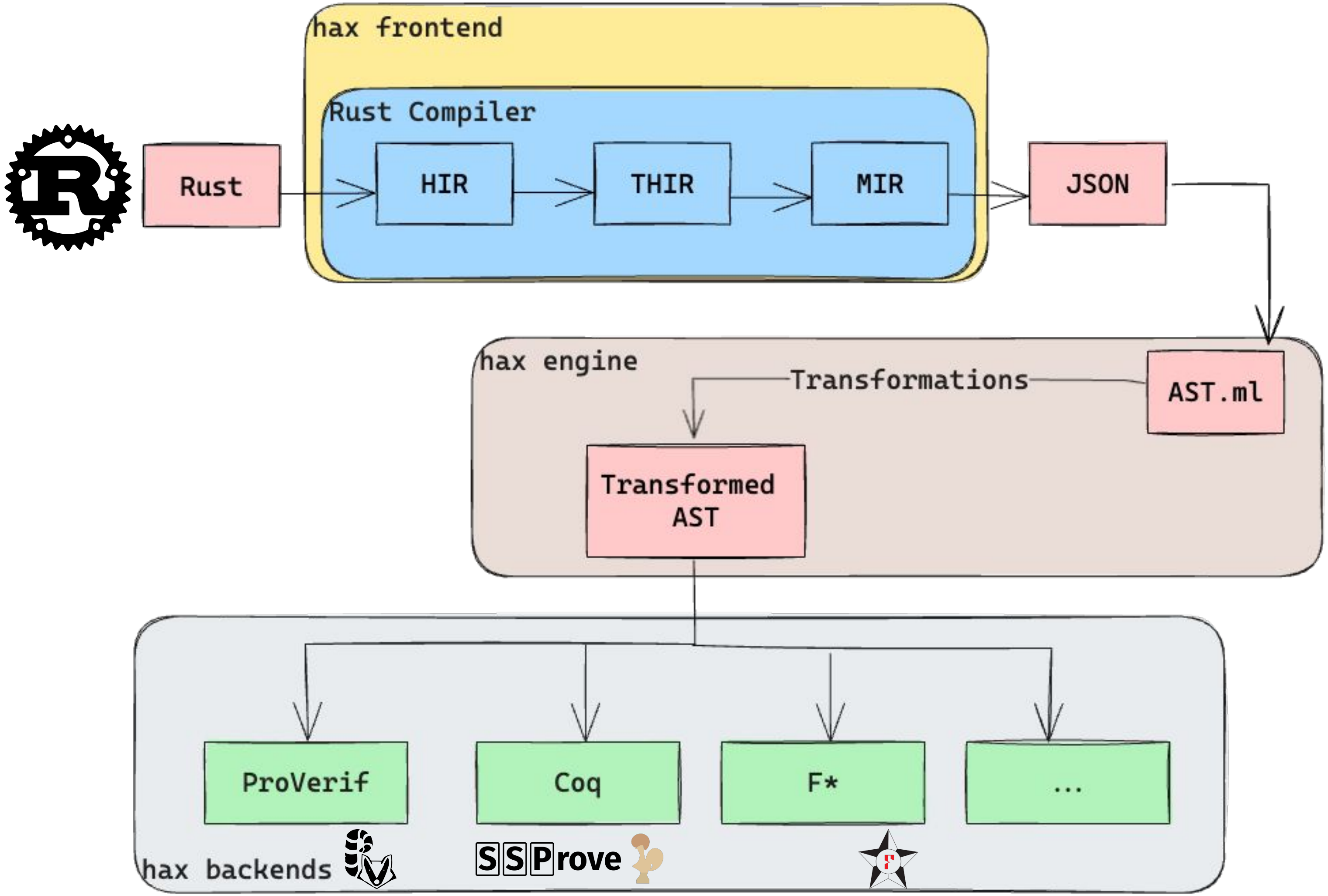
U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

AARHUS UNIVERSITY

Inria

hax: A Tool Framework for Rust Verification

- Accepts a large subset of safe Rust
  - Including hacspec, a purely functional spec language in Rust (presented at RustVerify'21)
- Translates it to formal models in F* or Coq, SSProve, ProVerif
  - Upcoming backends for EasyCrypt, 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*.

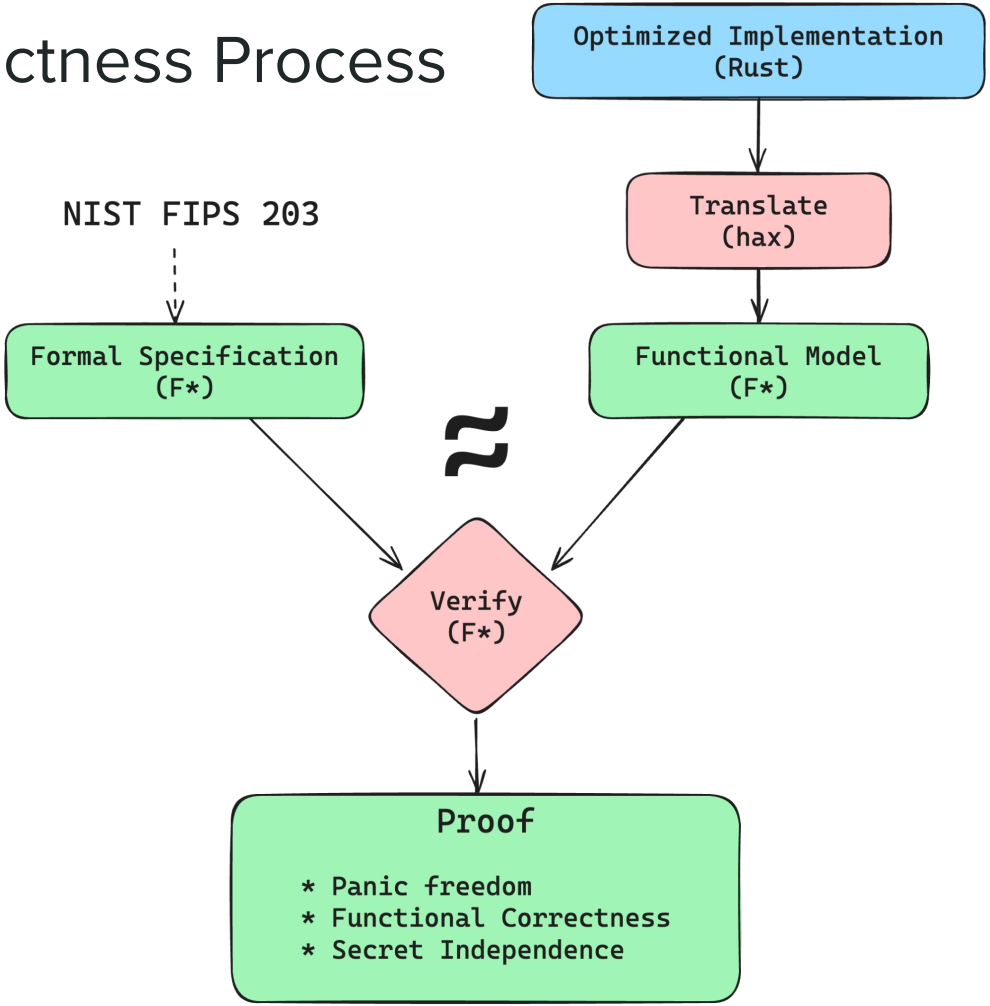

hax frontend
Rust Compiler
Rust
HIR
THIR
MIR
JSON
hax engine
Transformations
AST.ml
Transformed AST
ProVerif
Coq
F*
...
hax backends
SSProve

Coming Up

- A portable, verified Rust implementation of NIST’s ML-KEM
- Versatility of the Coq / SSProve backends:
    - Verifiable equivalence between Rust code and a verified assembly implementation
    - Formally verified guarantees for smart contracts in Rust
- The first TLS implementation in Rust with symbolic security guarantees based on the implementation itself, not a separate specification

Verified ML-KEM using F*

hax: Correctness Process



hax: Correctness Process

```
#[requires((i64::from(value) >= -BARRETT_R && i64::from(value) <= BARRETT_R))]
#[ensures(|result| result > -FIELD_MODULUS && result < FIELD_MODULUS &&
                   result % FIELD_MODULUS == value % FIELD_MODULUS)]
pub fn barrett_reduce(value: FieldElement) -> FieldElement {
    let t = i64::from(value) * BARRETT_MULTIPLIER;
    let t = t + (BARRETT_R >> 1);

    let quotient = t >> BARRETT_SHIFT;
    let quotient = quotient as i32;

    let sub = quotient * FIELD_MODULUS;

    value - sub
}
```

```
let barrett_reduce (value: i32)
    : Prims.Pure i32
      (requires
        (Core.Convert.f_from value <: i64) >=. (Core.Ops.Arith.Neg.neg v_BARRETT_R <: i64) &&
        (Core.Convert.f_from value <: i64) <=. v_BARRETT_R)
      (ensures
        fun result ->
          let result:i32 = result in
          result >. (Core.Ops.Arith.Neg.neg v_FIELD_MODULUS <: i32) && result <. v_FIELD_MODULU
          (result %! v_FIELD_MODULUS <: i32) =. (value %! v_FIELD_MODULUS <: i32)) =
  let t:i64 = (Core.Convert.f_from value <: i64) *! v_BARRETT_MULTIPLIER in
  let t:i64 = t +! (v_BARRETT_R >>! 1l <: i64) in
  let quotient:i64 = t >>! v_BARRETT_SHIFT in
  let quotient:i32 = cast (quotient <: i64) <: i32 in
  let sub:i32 = quotient *! v_FIELD_MODULUS in
  value -! sub
```

1. hax attributes for “design by contract”
2. F* statically checks that the properties hold

Secret Independence

Static analysis of forbidden operations

- **arithmetic operations** with input-dependent timing (e.g. division) over secret integers
- **comparison** over secret values
- **branching** over secret values
- **array** or vector **accesses** at secret indices

Timing issues in PQ-Crystals reference code

```c
void poly_tomsg(uint8_t msg[KYBER_INDCPA_MSGBYTES], const poly *a)
{
  unsigned int i,j;
  uint16_t t;

  for(i=0;i<KYBER_N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t  = a->coeffs[8*i+j];
      t += ((int16_t)t >> 15) & KYBER_Q;
      t  = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
      msg[i] |= t << j;
    }
  }
}
```

Timing issues in PQ-Crystals reference code

```
let compress_ciphertext_coefficient coefficient_bits fe =
  let _:Prims.unit = () <: Prims.unit in
  let _:Prims.unit = () <: Prims.unit in
  let compressed:u32 = (cast (fe <: u16) <: u32) <<! (coefficient_bits +! 1uy <: pub_u8) in
  let compressed:u32 =
    compressed +! (cast (Libcrux.Kem.Kyber.Constants.v_FIELD_MODULUS <: i32) <: u32)
  in
  (* Potential Timing Leak: division is not secret indepdnent *)
  let compressed:u32 =
    compressed /! (cast (Libcrux.Kem.Kyber.Constants.v_FIELD_MODULUS <<! 1l <: i32) <: u32)
  in
  let res = cast (Libcrux.Kem.Kyber.Arithmetic.get_n_least_significant_bits coefficient_bits compressed <: u32
    )
  <:
  i32
  in
  res
```

Verifying Libcrux's ML-KEM

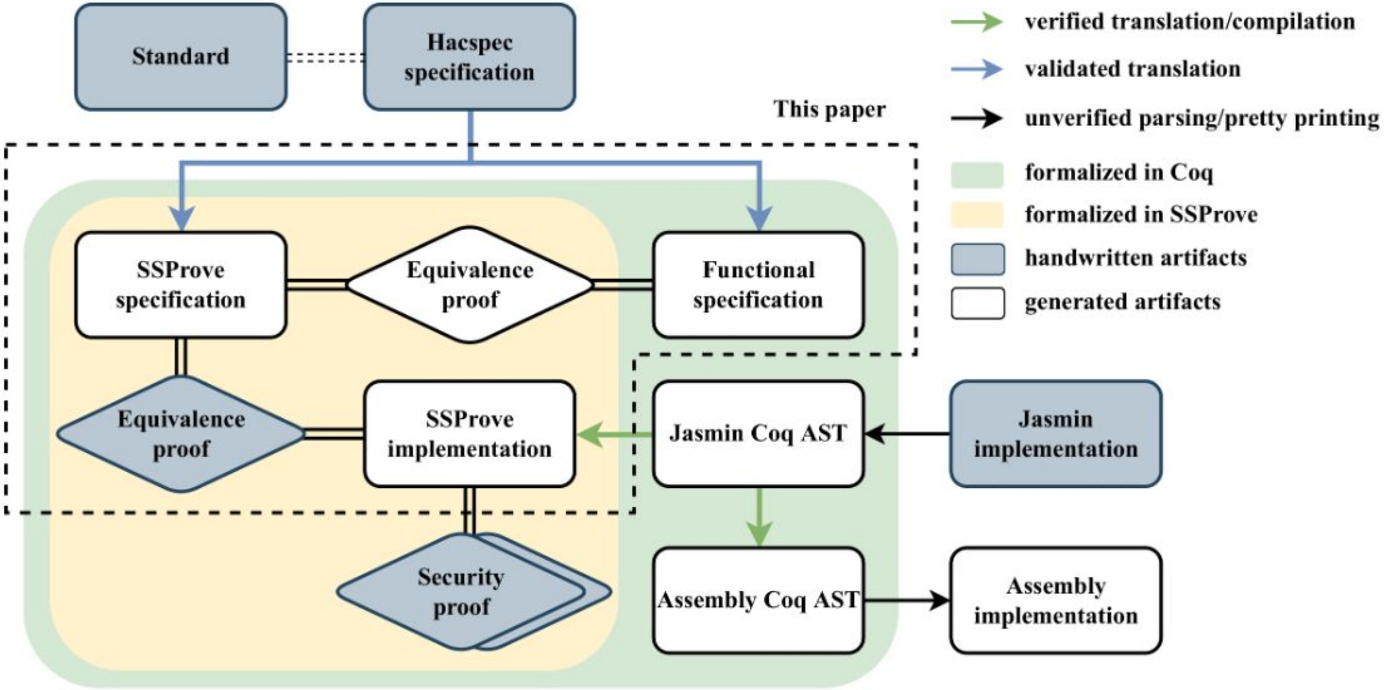
Karthikeyan Bhargavan, Lucas Franceschino, Franziskus Kiefer, Goutam Tamvada

January 30, 2024

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

Leverage existing work: Coq & SSProve

The Last Yard



[The Last Yard: Foundational End-to-End Verification of High-Speed Cryptography](). *CPP'24.*
Philipp G. Haselwarter, Benjamin Salling Hvass, Lasse Letager Hansen, Théo Winterhalter, Cătălin Hrițcu, Bas Spitters.

Verification of Smart Contracts

- Model smart contracts as state machines, using hax attribute
- Translate to SSProve
- Leverage existing work: ConCert a smart contract verification framework in Coq

```
#[hax::receive(contract = "name_of_contract",
               name = "name_of_entry_point",
               parameter = "name_of_data_structure")]
pub fn some_function_name<A: HasActions>(
    ctx: &impl HasReceiveContext,
    state: SomeContractState,
) -> Result<(A, SomeContractState), ParseError> {
    let params: name_of_data_structure =
        ctx.parameter_cursor().get()?;
    ..
    Ok((A::accept(), state_ret))
}
```

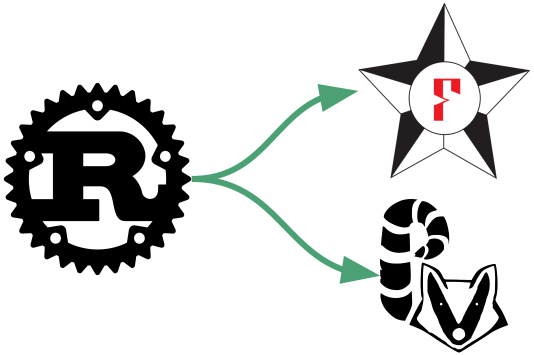
Specifying Smart Contract with Hax and ConCert. *CoqPL'24.*
Lasse Letager Hansen, Bas Spitters.

Protocol Verification using ProVerif

Multi-Backend use of hax: TLS 1.3

- Bertie is a minimal, high-assurance implementation of TLS 1.3 that aims to be verification friendly
- An ongoing example of simultaneous translation to F* as well as ProVerif

https://github.com/cryspen/bertie



for Correctness

for Security

What’s ProVerif?

- Fully automated protocol verification tool, which accepts a variant of Pi-Calculus as input language which is translated to Horn clauses
- Allows analysis of Reachability, Correspondence and Observational Equivalence properties in a purely symbolic (Dolev-Yao) security model
- Has previously been used to analyze many real world protocols for security issues, based on their specifications, e.g. TLS1.3, Signal …

https://bblanche.gitlabpages.inria.fr/proverif/

Applied Pi-Calculus

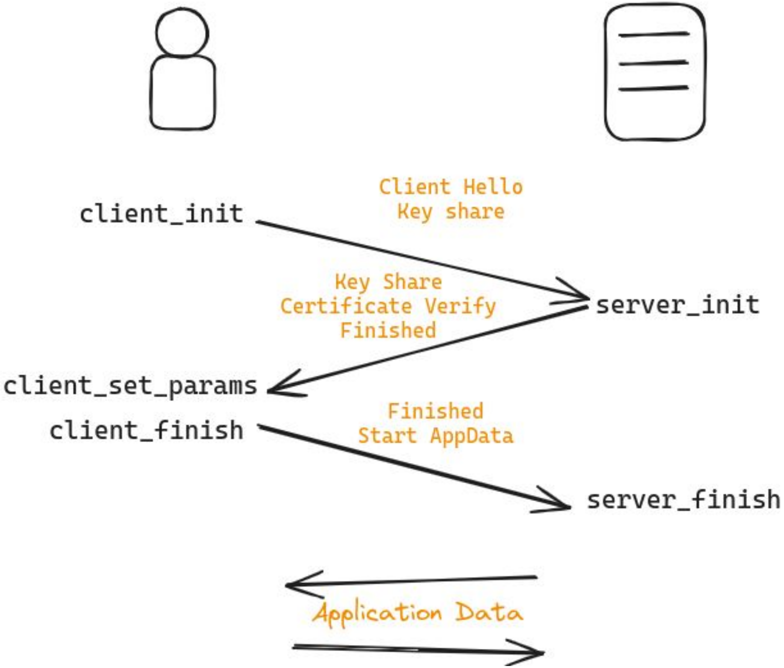
```
type secret_key.
type public_key.
type message.

fun pk_from_sk(secret_key): public key.
fun enc(public_key, message): ciphertext.
reduc forall sk: secret_key, msg: message;
    dec( sk, enc(pk_from_sk(sk), msg)) = msg.
```

- No data structures, only constructors and destructors on terms
- No support for associative operations
  - `concat( A, concat(B, C)) != concat(concat(A, B), C)`
- No Result / Option types, model blocks on failure to destruct term
- No polymorphism of any kind

TLS 1.3 Handshake in Applied Pi-Calculus

```
let Client() =
     let (CH,KS) =
          client_init() in
     out(c, (CH, KS));
     in(c, (KS, CV, SF));
…
```

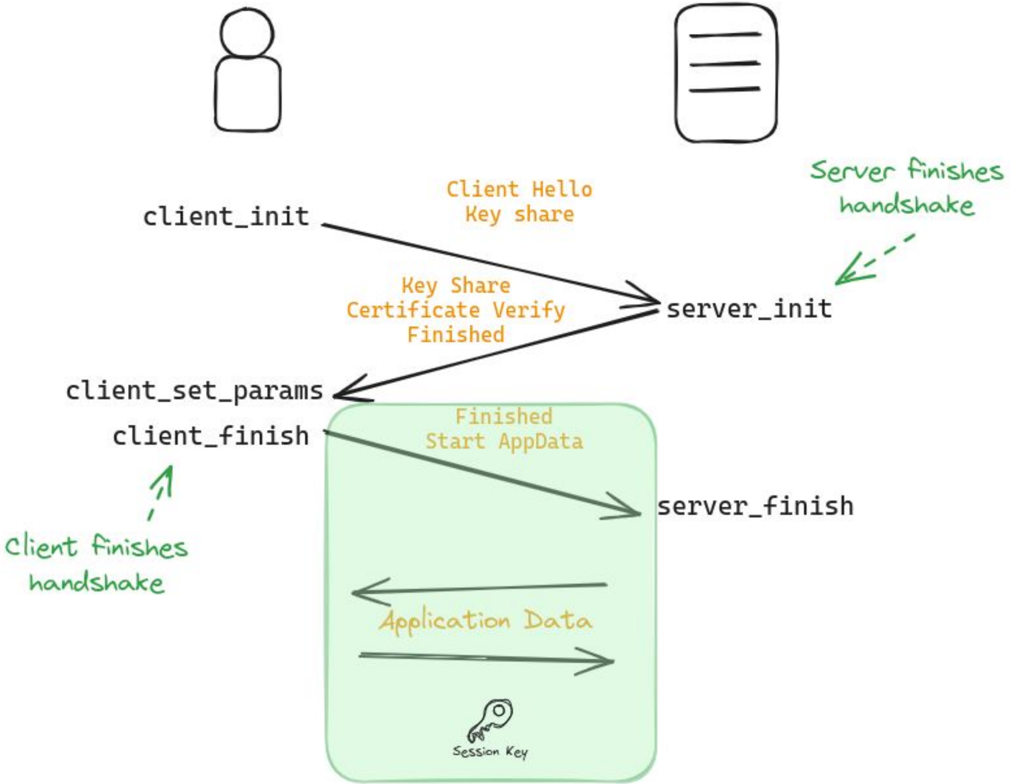



```
let Server() =
     in(c, (CH,KS));
     let (KS, CV, SF) =
          server_init() in
     out(c, (KS, CV, SF));

…
```

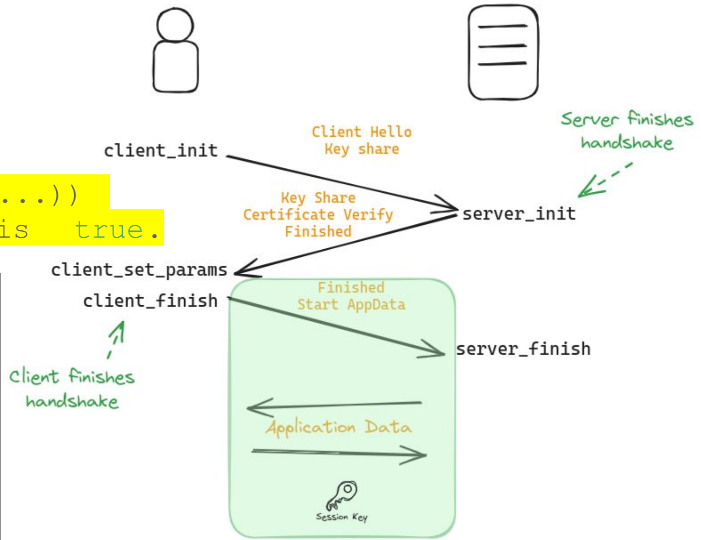
```
process
     Client() | Server()
```

TLS 1.3 Handshake Security



ProVerif Queries

```
Query event(ClientFinished(...)) ==> event(ServerFinished(...))
|| CompromisedServerCertSK(...) || CompromisedServerPSK(...) is  true.
```

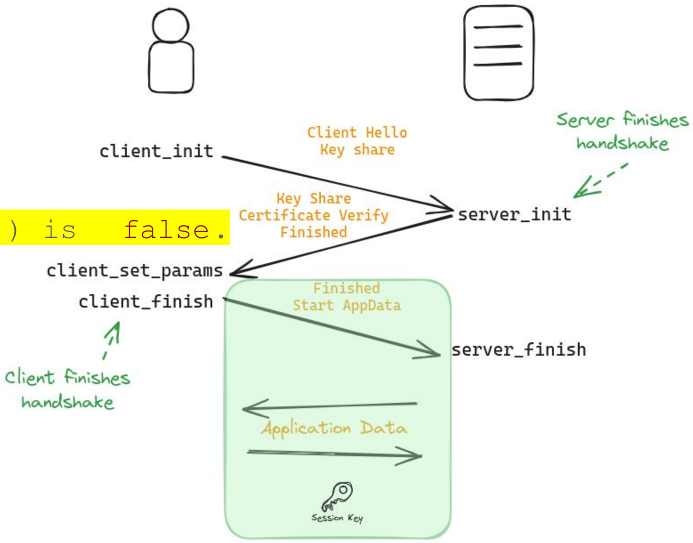



```
query server_name:bertie__tls13utils__t_Bytes,
      alg: bertie__tls13crypto__t_AeadAlgorithm,
      c2skiv: bertie__tls13crypto__t_AeadKeyIV,
      c2sctr:nat,
      s2ckiv:bertie__tls13crypto__t_AeadKeyIV,
      s2cctr:nat,
      exp:bertie__tls13utils__t_Bytes,
      ccipher:bertie__tls13record__t_DuplexCipherState1,
      cst:bertie__tls13handshake__t_ClientPostClientFinished,
      sst:bertie__tls13handshake__t_ServerPostServerFinished;

      event(ClientFinished(server_name,
                           bertie__tls13record__DuplexCipherState1_c(alg,c2skiv,c2sctr,s2ckiv,s2cctr,exp),
                           cst))
      ==> event(ServerFinished(server_name,
                               bertie__tls13record__DuplexCipherState1_c(alg,s2ckiv,s2cctr,c2skiv,c2sctr,exp),
                               sst))
          || event(CompromisedServerCertSK(server_name))
          || event(CompromisedServerPSK(server_name)).
```

ProVerif Queries

Query event(ClientFinished(...)) ==> event(ServerFinished(...)) is false.




```
query server_name:bertie__tls13utils__t_Bytes,
      alg: bertie__tls13crypto__t_AeadAlgorithm,
      c2skiv: bertie__tls13crypto__t_AeadKeyIV,
      c2sctr:nat,
      s2ckiv:bertie__tls13crypto__t_AeadKeyIV,
      s2cctr:nat,
      exp:bertie__tls13utils__t_Bytes,
      ccipher:bertie__tls13record__t_DuplexCipherState1,
      cst:bertie__tls13handshake__t_ClientPostClientFinished,
      sst:bertie__tls13handshake__t_ServerPostServerFinished;

      event(ClientFinished(server_name,
                          bertie__tls13record__DuplexCipherState1_c(alg,c2skiv,c2sctr,s2ckiv,s2cctr,exp),
                          cst))
      ==> event(ServerFinished(server_name,
                              bertie__tls13record__DuplexCipherState1_c(alg,s2ckiv,s2cctr,c2skiv,c2sctr,exp),
                              sst)).
```

Overview of ProVerif Extraction

- Ca. 4000 lines of extracted ProVerif code for 900 lines of Rust handshake code (+ its dependencies):
    - Reachability of all parts of handshake
    - Server authentication
    - Secrecy of derived session key
    - (given handwritten models for handshake data en-/decoding & crypto primitives)
- No security issues found during verification, but 1 logic bug
- On F* side: Panic freedom, scope excludes certificate parsing and HTTPS application code but complements ProVerif analysis

Takeaways for this Talk

- hax offers an undogmatic approach to Rust Verification
  - Driven by real implementations
  - Allows you to use the right tool for the given job
  - Different verification methodologies can complement each other using hax
- Ongoing Efforts & Improvements
  - Bubble-up proof assistant error messages
  - IDE integration
  - More backends…
  - For *your* favorite verification tool?

https://github.com/hacspec/hax

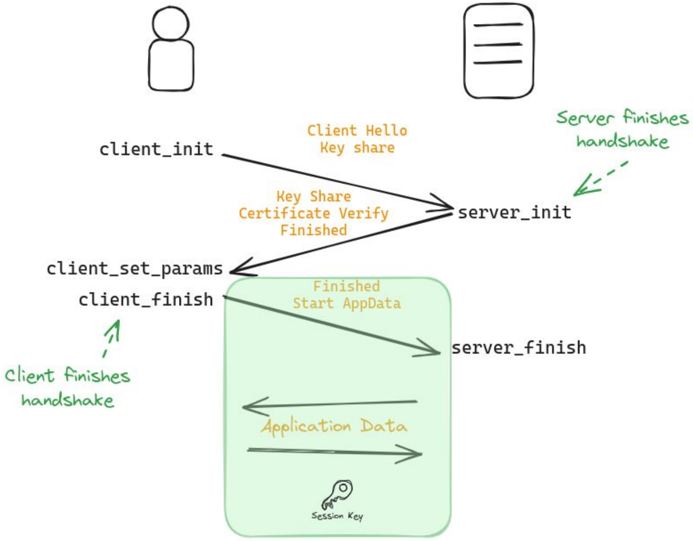
jonas@cryspen.com

Translating Functions to Process Macros

```rust
/// Derive an AEAD key and iv.
pub(crate) fn derive_aead_key_iv(
    hash_algorithm: &HashAlgorithm,
    aead_algorithm: &AeadAlgorithm,
    key: &Key,
) -> Result<AeadKeyIV, TLSError> {
    let sender_write_key: Bytes = hkdf_expand_label(
        hash_algorithm,
        key,
        label: bytes(&LABEL_KEY),
        context: &Bytes::new(),
        aead_algorithm.key_len(),
    )?;
    let sender_write_iv: Bytes = hkdf_expand_label(
        hash_algorithm,
        key,
        label: bytes(&LABEL_IV),
        context: &Bytes::new(),
        aead_algorithm.iv_len(),
    )?;
    Ok(AeadKeyIV::new(
        key: AeadKey::new(bytes: sender_write_key,
        alg: *aead_algorithm),
        sender_write_iv,
    ))
}
```

```
letfun bertie__tls13handshake__derive_aead_key_iv(
    hash_algorithm : bertie__tls13crypto__t_HashAlgorithm,
    aead_algorithm : bertie__tls13crypto__t_AeadAlgorithm,
    key : bertie__tls13utils__t_Bytes
  ) =
  let sender_write_key = bertie__tls13handshake__hkdf_expand_label(
    hash_algorithm,
    key,
    bertie__tls13utils__bytes(bertie__tls13formats__v_LABEL_KEY),
    bertie__tls13utils__impl__Bytes__new(()),
    bertie__tls13crypto__impl__AeadAlgorithm__key_len(aead_algorithm)
  ) in let sender_write_iv = bertie__tls13handshake__hkdf_expand_label(
    hash_algorithm,
    key,
    bertie__tls13utils__bytes(bertie__tls13formats__v_LABEL_IV),
    bertie__tls13utils__impl__Bytes__new(()),
    bertie__tls13crypto__impl__AeadAlgorithm__iv_len(aead_algorithm)
  ) in bertie__tls13crypto__impl__AeadKeyIV__new(
    bertie__tls13crypto__impl__AeadKey__new(
      sender_write_key, aead_algorithm
    ),
    sender_write_iv
  )
  else bertie__tls13crypto__t_AeadKeyIV_err()
  else bertie__tls13crypto__t_AeadKeyIV_err().
```

ProVerif Events




```
let
  (server_hello: bertie__tls13formats__handshake_data__t_HandshakeData,
   flight: bertie__tls13formats__handshake_data__t_HandshakeData,
   cipher0: Option,
   cipher_hs: bertie__tls13record__t_DuplexCipherStateH,
   cipher1: bertie__tls13record__t_DuplexCipherState1,
   server_state: bertie__tls13handshake__t_ServerPostServerFinished)
= server_init_output in        You, 1 second ago • Uncommitted changes
event ServerFinished(server_name,cipher1,server_state);
```

```
let (client_finished: bertie__tls13formats__handshake_data__t_HandshakeData,
     cipher: bertie__tls13record__t_DuplexCipherState1,
     client_state: bertie__tls13handshake__t_ClientPostClientFinished)
    = bertie__tls13handshake__client_finish(server_name,
                                            flight,
                                            client_state)
in
event ClientFinished(server_name,cipher,client_state);
```

Auto-generating Constructors

```
(* marked as constructor *)
fun bertie__tls13formats__server_hello(
    bertie__tls13crypto__t_Algorithms,
    bertie__tls13utils__t_Bytes,
    bertie__tls13utils__t_Bytes,
    bertie__tls13utils__t_Bytes
  )
  : bertie__tls13formats__handshake_data__t_HandshakeData [data
```

```
/// Build the server hello message.
#[cfg_attr(feature = "hax-pv", pv_constructor)]
pub(crate) fn server_hello(
    algs: &Algorithms,
    sr: Random,
    sid: &Bytes,
    gy: &KemPk,
) -> Result<HandshakeData, TLSError> {
    let ver: Bytes = bytes2(3, 3);
    let sid: Bytes = encode_length_u8(sid.as_raw())?;
    let cip: Bytes = algs.ciphersuite()?;
    let comp: Bytes = bytes1(0);
    let ks: Bytes = server_key_shares(algs, gy.clone())?;
    let sv: Bytes = server_supported_version(algs)?;
    let mut exts: Bytes = ks.concat(sv);
    match algs.psk_mode() {
        true => exts = exts.concat(server_pre_shared_key(algs)?),
        false => {}
    }
    let encoded_extensions: Bytes = encode_length_u16(exts)?;
    let sh: HandshakeData = HandshakeData::from_bytes(
        HandshakeType::ServerHello,
        &bytes_concat!(ver, sr, sid, cip, comp, encoded_extensions),
    )?;
    Ok(sh)
}
```

```rust
#[cfg_attr(feature = "hax-pv", pv_handwritten)]
pub(crate) fn parse_server_hello(
    algs: &Algorithms,
    server_hello: &HandshakeData,
) -> Result<(Random, KemPk), TLSError> {
    let HandshakeData(server_hello: Bytes) =
        server_hello.as_handshake_message(HandshakeType::ServerHello)?;
    let ver: Bytes = bytes2(3, 3);
    let cip: Bytes = algs.ciphersuite()?;
    let comp: Bytes = bytes1(0);
    let mut next: usize = 0;
    match check_eq(&ver, &server_hello.slice_range(next..next + 2)) {
```

```
(* REPLACE by handwritten model *)
letfun bertie__tls13formats__parse_server_hello(
         algs : bertie__tls13crypto__t_Algorithms,
         server_hello : bertie__tls13formats__handshake_data__t_HandshakeData
       ) =
       bitstring_default().
```

```
reduc forall algs: bertie__tls13crypto__t_Algorithms,
             server_random: bertie__tls13utils__t_Bytes,
             sid: bertie__tls13utils__t_Bytes,
             gy: bertie__tls13utils__t_Bytes;
   bertie__tls13formats__parse_server_hello(
     algs,
     bertie__tls13formats__server_hello(algs, server_random, sid, gy)
) = (server_random, gy).
```

ProVerif Boilerplate

```rust
#[derive(Debug, Clone, Default)]
4 implementations
pub struct ServerDB {
    pub(crate) server_name: Bytes,
    pub(crate) cert: Bytes,
    pub(crate) sk: SignatureKey,
    pub(crate) psk_opt: Option<(Bytes, Psk)>,
}
```

```
type bertie__server__t_ServerDB.

fun bertie__server__t_ServerDB_to_bitstring(bertie__server__t_ServerDB)
    : bitstring [typeConverter].
fun bertie__server__t_ServerDB_from_bitstring(bitstring)
    : bertie__server__t_ServerDB [typeConverter].
const bertie__server__t_ServerDB_default_value: bertie__server__t_ServerDB.
letfun bertie__server__t_ServerDB_default() =
       bertie__server__t_ServerDB_default_value.
letfun bertie__server__t_ServerDB_err() =
       let x = construct_fail() in bertie__server__t_ServerDB_default_value.
fun bertie__server__ServerDB_c(
      bertie__tls13utils__t_Bytes,
      bertie__tls13utils__t_Bytes,
      bertie__tls13utils__t_Bytes,
      Option
    )
    : bertie__server__t_ServerDB [data].
reduc forall
  bertie__server__ServerDB_f_server_name: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_cert: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_sk: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_psk_opt: Option
;
    accessor_bertie__server__ServerDB_f_server_name(
      bertie__server__ServerDB_c(
        bertie__server__ServerDB_f_server_name,
        bertie__server__ServerDB_f_cert,
        bertie__server__ServerDB_f_sk,
        bertie__server__ServerDB_f_psk_opt
      )
    ) = bertie__server__ServerDB_f_server_name.
reduc forall
  bertie__server__ServerDB_f_server_name: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_cert: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_sk: bertie__tls13utils__t_Bytes,
  bertie__server__ServerDB_f_psk_opt: Option
```