Towards Formal Verification in Cryptographic Web Applications

A Three Year Evolution

Nadim Kobeissi

About Us

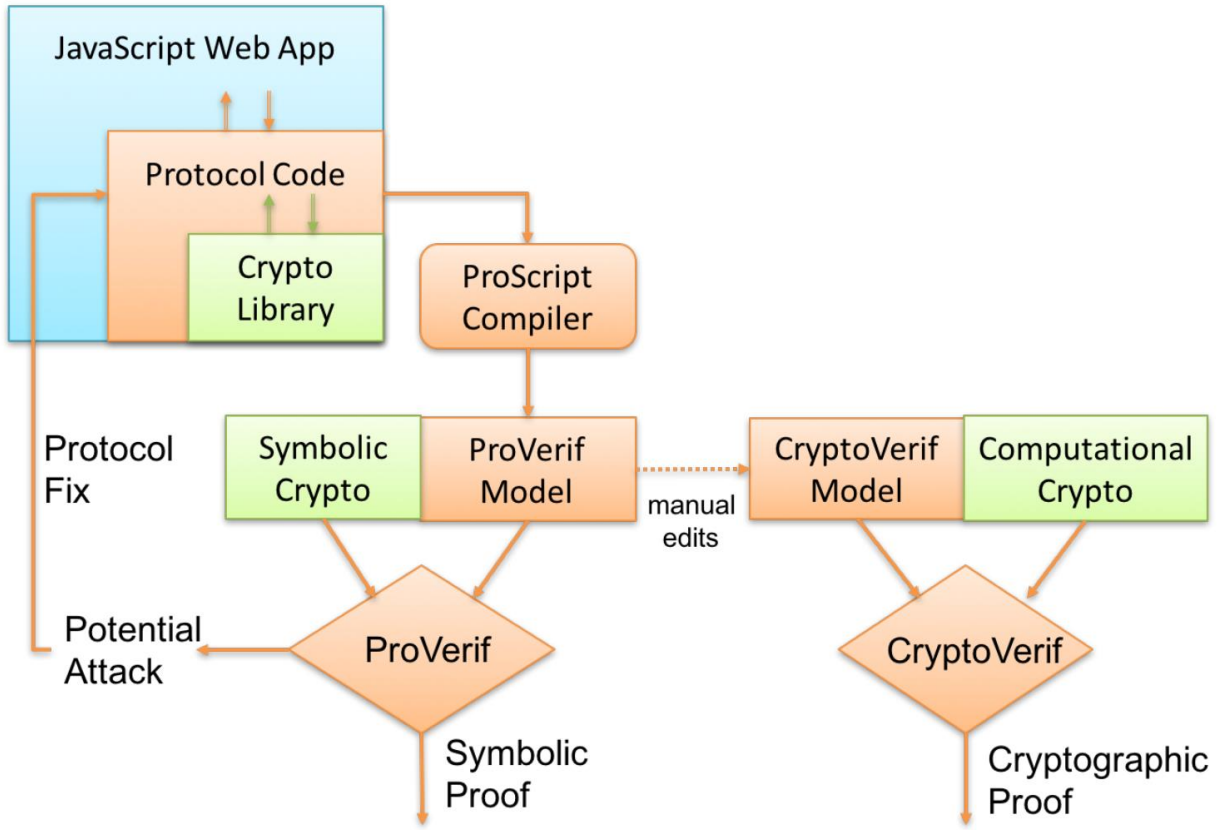
- PROSECCO: **Pro**gramming **Sec**urely with **C**rypt**o**graphy.
- Team at INRIA Paris specializing in applied cryptography and formal verification.
- Goals:
  - Formally delineate the patterns in which cryptographic flaws occur across all the world's important protocols.
  - Develop technologies to minimize these flaws occurring again in the future, based on what we've learned.

Technologies

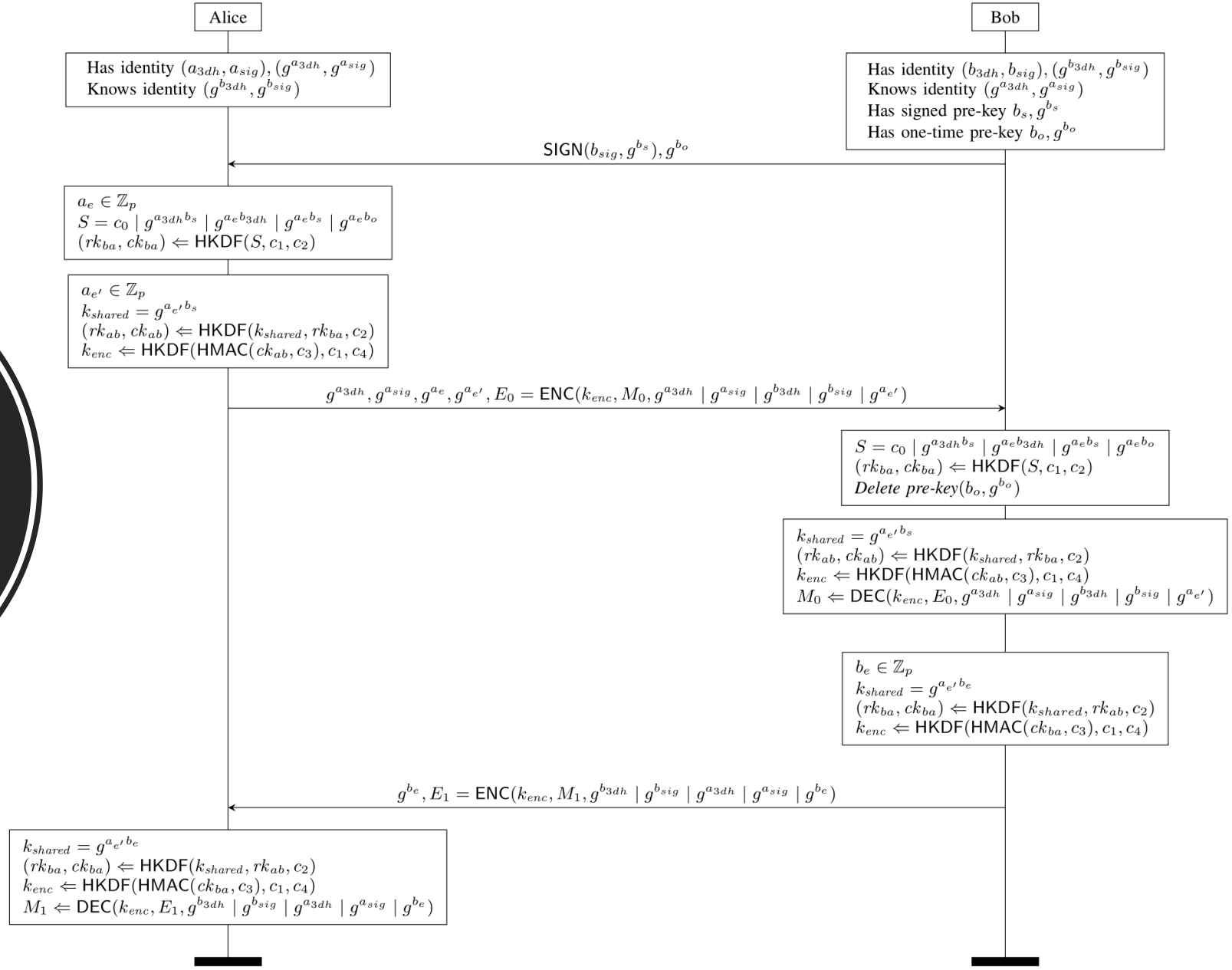
- Major projects:
  - **F*:** ML programming language that lends itself to formal verification.
    - Dependent types, refinements, etc.
    - HACL* verified cryptography library, miTLS verified TLS implementation.
  - **ProVerif**: Automated protocol verification in the symbolic model.
    - Network execution under a Dolev-Yao attacker.
    - ProScript, TLS, Signal, ACME, Capsule, LDL…
  - **CryptoVerif**: Guided protocol verification with proofs in the computational model.
    - TLS, Signal, WireGuard…

Cryptographic Web Applications

- Radical propulsion in market share:
  - Cryptocat: end-to-end encrypted chat with OTR (2011)
  - WhatsApp Web: end-to-end encrypted view into mobile device (2016)
  - Signal Desktop: Electron App (2017)
  - Skype: Electron App (2018)

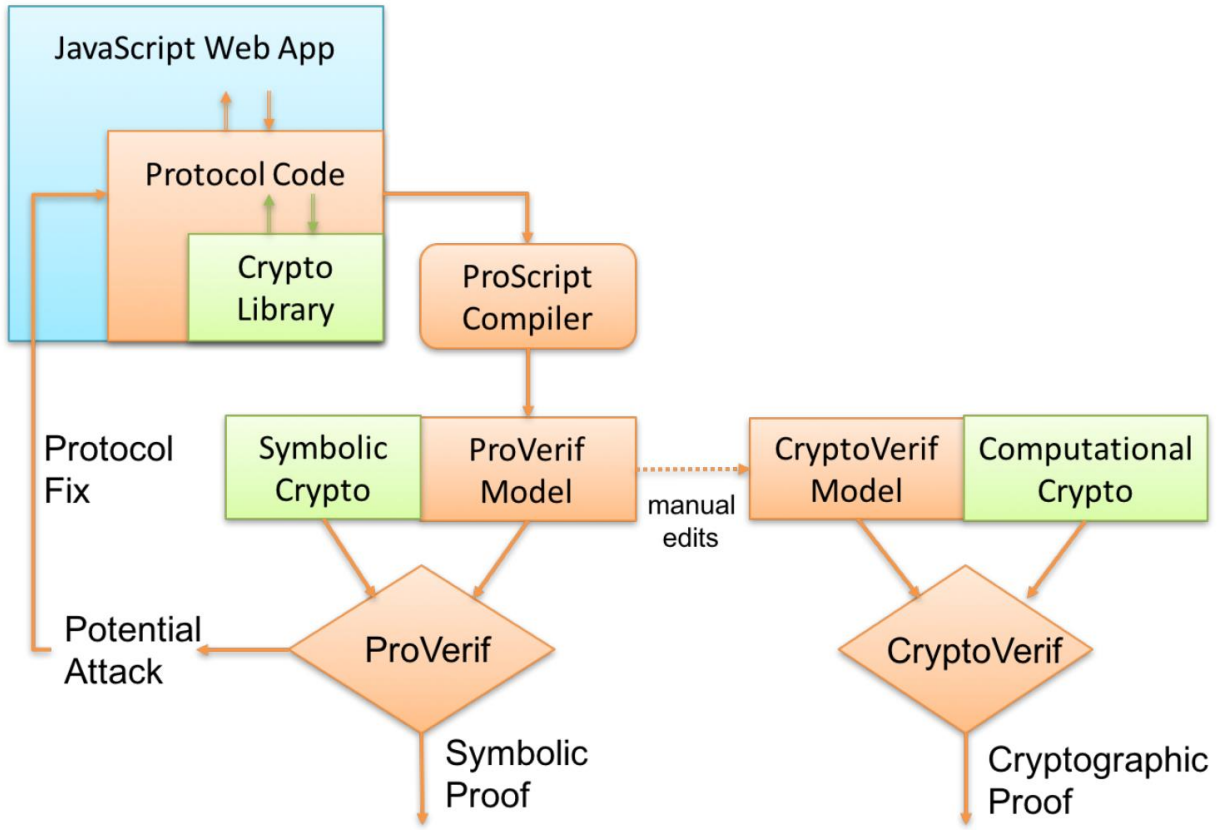


Signal Protocol



Linking JavaScript Implementations to Verification Frameworks

- ProScript: evolution from Defensive JavaScript (Antoine Delignat-Lavaud, 2014) into a full language: subset of JavaScript -> ProVerif



```
315 const RATCHET = {
316     deriveSendKeys: function(them, myEphemeralKeyPriv) {
317         const kShared = ProScript.crypto.DH25519(
318             myEphemeralKeyPriv, them.ephemeralKey
319         )
320         const sendKeys = UTIL.HKDF(
321             kShared, them.recvKeys[0], 'WhisperRatchet'
322         )
323         const kKeys = UTIL.HKDF(
324             ProScript.crypto.HMACSHA256(sendKeys[1], '1'),
325             Type_key.construct(),
326             'WhisperMessageKeys'
327         )
328         return {
329             sendKeys: sendKeys,
330             kENC:     kKeys[0]
331         }
332     },
```

```
443 letfun fun_deriveSendKeys(them:object_them, myEphemeralKeyPriv:key) =
444     let kShared = ProScript_crypto_DH25519(
445         myEphemeralKeyPriv, Object_them_get_ephemeralKey(them)
446     ) in
447     let sendKeys = fun_HKDF(
448         kShared, Array_1_get_e_0(Object_them_get_recvKeys(them)),
            string_101)
449     in
450     let kKeys = fun_HKDF(
451         ProScript_crypto_HMACSHA256(Array_1_get_e_1(sendKeys),
            string_105),
452         Type_key_construct(),
453         string_108
454     ) in
455     Object_112(
456         Array_1_get_e_0(kKeys),
457         sendKeys
458     ).
```

ProScript to ProVerif: Quick Example

Verification in ProVerif

- Define a top-level process.
- Define queries.
- Execute over a network with an active attacker.

- **Protocol bugs:** Key Compromise Impersonation. If Bob's long-term secret and Bob's signed pre-key is compromised, attacker can impersonate Alice to Bob.
- **Implementation bugs:** missing HMAC check.

```
free secMsg1:bitstring [private].
free secMsg2:bitstring [private].
free secMsg3:bitstring [private].
query attacker(secMsg1).
event Send(key, key, bitstring).
event Recv(key, key, bitstring).
query a:key,b:key,m:bitstring; event(Recv(a, b, m)) ==> event(Send(a, b, m)).
query a:key,b:key,m:bitstring; event(Recv(a, b, m)).
query a:key,b:key,m:bitstring; event(Send(a, b, m)).

let Initiator(
        initiatorIdentityKey:object_keypair,
        initiatorSignedPreKey:object_keypair,
        initiatorPreKey:object_keypair,
        responderIdentityKeyPub:key,
        responderIdentityDHKeyPub:key
) =
        out(io, (
                Object_keypair_get_pub(initiatorSignedPreKey),
                ProScript_crypto_ED25519_signature(
                        Type_key_toBitstring(Object_keypair_get_pub(initiatorSignedPreKey)),
                        Object_keypair_get_priv(initiatorIdentityKey),
                        Object_keypair_get_pub(initiatorIdentityKey)
                ),
                Object_keypair_get_pub(initiatorPreKey)
        ));
        in(io, (
                responderSignedPreKeyPub:key,
                responderSignedPreKeySignature:bitstring,
```

Verification in ProVerif

- We verify:
  - Confidentiality.
  - Authenticity.
  - Forward secrecy.
  - Future secrecy.
  - Indistinguishability.
  - Absence of replay attacks.

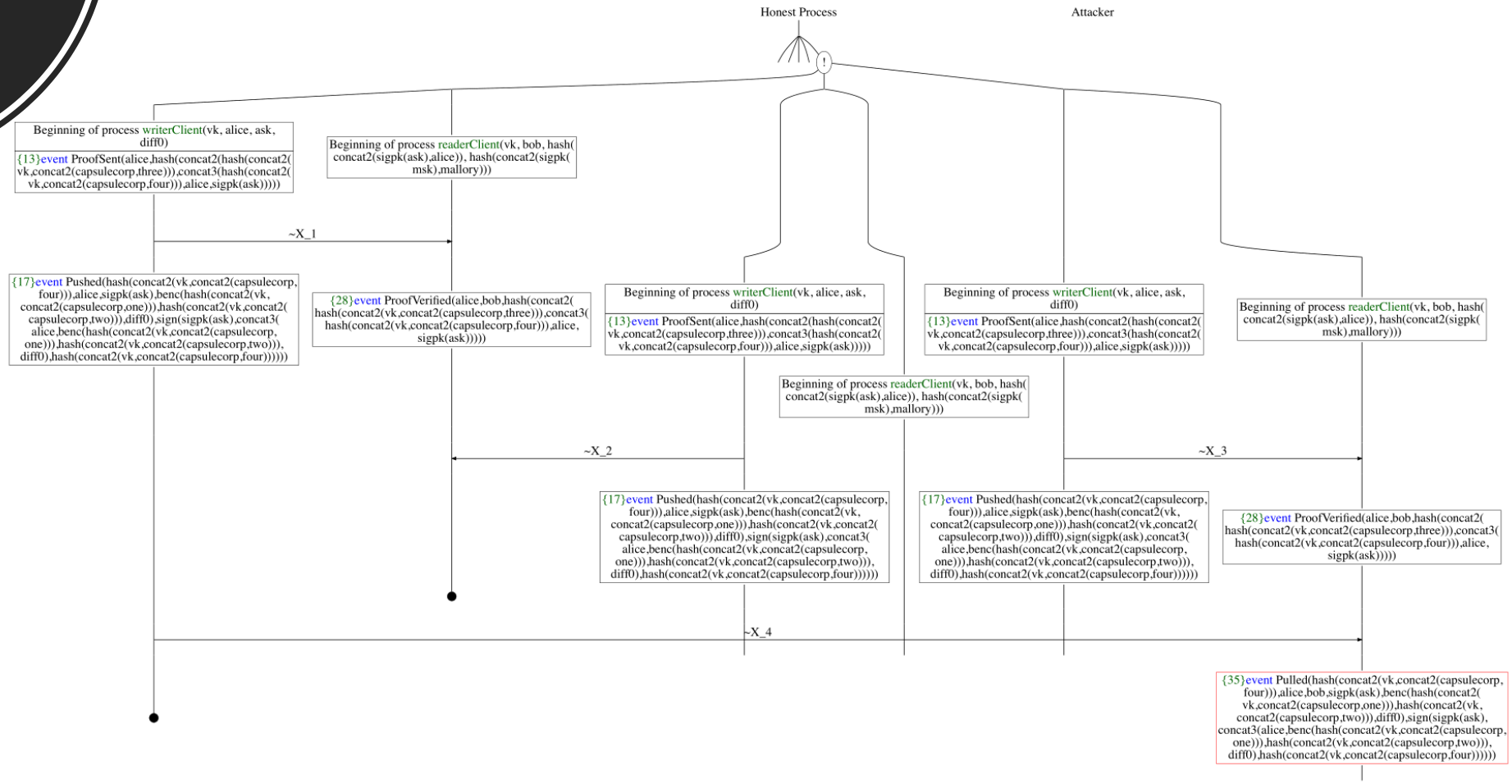
```
free secMsg1:bitstring [private].
free secMsg2:bitstring [private].
free secMsg3:bitstring [private].
query attacker(secMsg1).
event Send(key, key, bitstring).
event Recv(key, key, bitstring).
query a:key,b:key,m:bitstring; event(Recv(a, b, m)) ==> event(Send(a, b, m)).
query a:key,b:key,m:bitstring; event(Recv(a, b, m)).
query a:key,b:key,m:bitstring; event(Send(a, b, m)).

let Initiator(
        initiatorIdentityKey:object_keypair,
        initiatorSignedPreKey:object_keypair,
        initiatorPreKey:object_keypair,
        responderIdentityKeyPub:key,
        responderIdentityDHKeyPub:key
) =
        out(io, (
                Object_keypair_get_pub(initiatorSignedPreKey),
                ProScript_crypto_ED25519_signature(
                        Type_key_toBitstring(Object_keypair_get_pub(initiatorSignedPreKey)),
                        Object_keypair_get_priv(initiatorIdentityKey),
                        Object_keypair_get_pub(initiatorIdentityKey)
                ),
                Object_keypair_get_pub(initiatorPreKey)
        ));
        in(io, (
                responderSignedPreKeyPub:key,
                responderSignedPreKeySignature:bitstring,
```

ProVerif Trace: Capsule

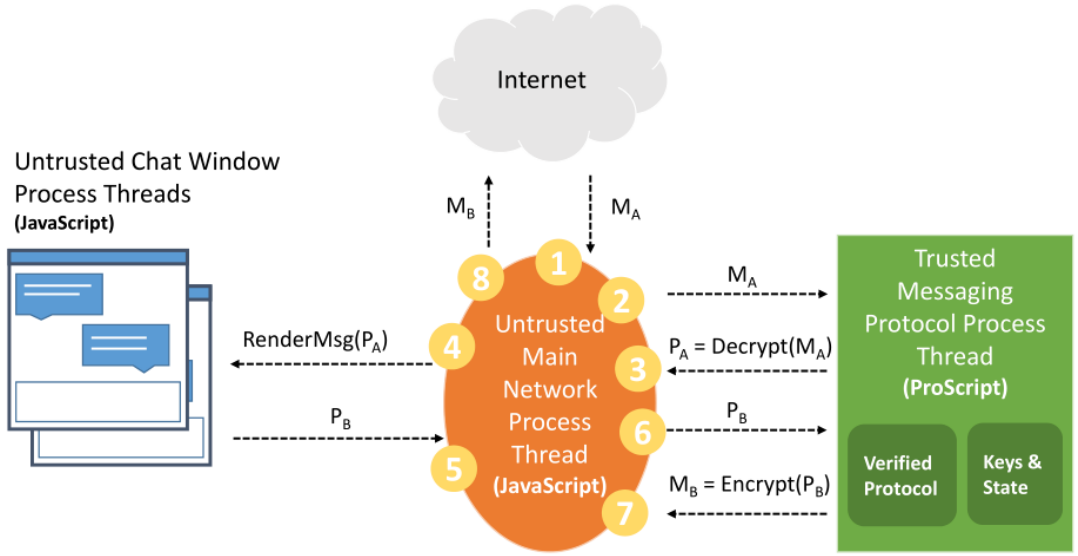
A trace has been found.

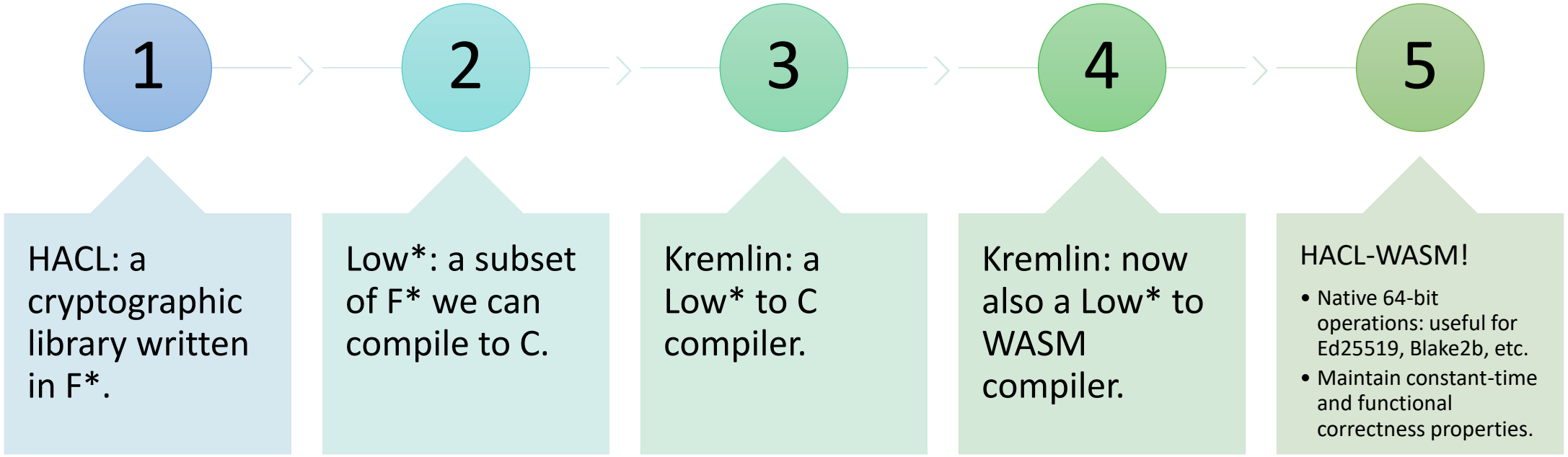
Abbreviations
~X_1 = (hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask),hash(concat2(hash(concat2(vk,concat2(capsulecorp,three))),concat3(hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask)))))
~X_2 = (hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask),hash(concat2(hash(concat2(vk,concat2(capsulecorp,three))),concat3(hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask)))))
~X_3 = (hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask),hash(concat2(hash(concat2(vk,concat2(capsulecorp,three))),concat3(hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask)))))
~X_4 = (hash(concat2(vk,concat2(capsulecorp,four))),alice,sigpk(ask),benc(hash(concat2(vk,concat2(capsulecorp,one))),hash(concat2(vk,concat2(capsulecorp,two))),diff0),sign(sigpk(ask),concat3(alice,benc(hash(concat2(vk,concat2(capsulecorp,one))),hash(concat2(vk,concat2(capsulecorp,two))),diff0),hash(concat2(vk,concat2(capsulecorp,four))))))



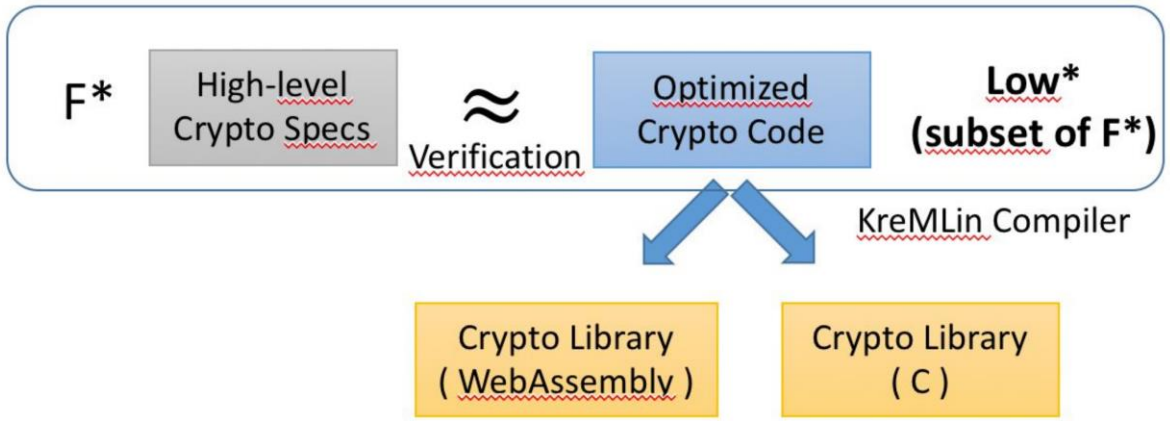
Cryptographic Web Applications

- Cryptocat (2016):
  - ProScript protocol core (Signal)
  - Translates and verifies in ProVerif
  - Manually proven in CryptoVerif
  - Trusted cryptographic core
  - The structure is there, but can we improve upon the individual components?





HACL-WASM: F* Primitives in WebAssembly



- HACL-WASM gives us perhaps the most high-assurance cryptographic primitives for the web.
- Can we pair this with a protocol implementation from F*?
- Integration: Signal, Skype, Cryptocat, Capsule.

SignalStar and HACL-WASM

Conclusion

Three years of following different complimentary approaches: advances in one branch leads to conclusions useful for another.

In the future: generating full applications that are formally verified: protocol, primitives, etc. and facilitating availability to provers.