Mechanized Cryptographic Proofs of Protocols and their Link with Verified Implementations

Benjamin Lipp — PhD Defence

PhD Advisors: Bruno Blanchet, Karthikeyan Bhargavan

June 28, 2022

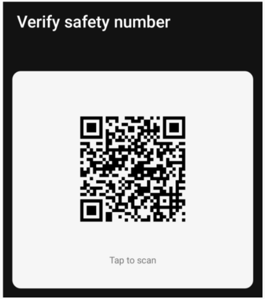
Inria Paris

Cryptology is a fascinating science, concerned with security and privacy of information, communication, and computation in the presence of adversaries.

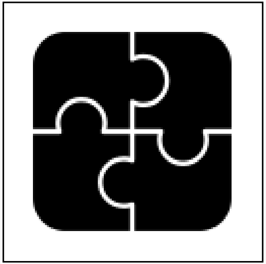
— adapted from iacr.org

Cryptology is a fascinating science, concerned with security and privacy of information, communication, and computation in the presence of adversaries.

— adapted from iacr.org



Cryptographic Security



Strongest assurance for specification level of cryptosystems:
Provable Security

In practice: reduction proofs in the computational model.

Game-Based Proofs

Like in a board game:

- Participants:
- Actions:
- Winning Condition

Game-Based Proofs

Like in a board game:

- Participants: Adversary
- Actions:
- Winning Condition

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions:
- Winning Condition

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

We do not play the game, we reason about it!

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

We do not play the game, we reason about it!

encode security properties here

initial game

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

We do not play the game, we reason about it!

encode security properties here

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

We do not play the game, we reason about it!

encode security properties here

$$\text{initial game} \quad \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \quad \text{intermediate games} \quad \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \quad \text{final game}$$

Game-Based Proofs

Like in a board game:

- Participants: Adversary, Alice, Bob, Client, Server
- Actions: Encrypt, Reveal, Corrupt
- Winning Condition

We do not play the game, we reason about it!

encode security properties here

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

Critique I: Relevance to Real-World Systems

Common critique [Hv17]: Assumptions and abstractions are **too far from reality**!

Critique I: Relevance to Real-World Systems

Common critique [Hv17]: Assumptions and abstractions are **too far from reality**!

Practice-Oriented Provable Security

Critique I: Relevance to Real-World Systems

Common critique [Hv17]: Assumptions and abstractions are **too far from reality**!

Practice-Oriented Provable Security tries to find useful answers to:

- Which key size is secure?
- How many users should be allowed?

Critique II: When is a proof convincing?

Critique II: When is a proof convincing?

Bellare and Rogaway: many “essentially unverifiable” proofs, “crisis of rigor” [BR06]

Critique II: When is a proof convincing?

Bellare and Rogaway: many “essentially unverifiable” proofs, “crisis of rigor” [BR06]

Halevi: some reasons are social, but “our proofs are truly complex” [Hal05]

Critique II: When is a proof convincing?

Bellare and Rogaway: many “essentially unverifiable” proofs, “crisis of rigor” [BR06]

Halevi: some reasons are social, but “our proofs are truly complex” [Hal05]

Call for “automated tools, that can help write and verify game-based proofs”.
— [Hal05; BR06]

Game-Based Proofs with the CryptoVerif Proof Assistant

encode security properties here

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

Game-Based Proofs with the CryptoVerif Proof Assistant

encode security properties here

$$\text{initial game} \quad \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \quad \text{intermediate games} \quad \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \quad \text{final game}$$

security properties easily provable here

- CryptoVerif constructs a sequence of computationally indistinguishable games
- built-in proof strategy, and detailed guidance by user

Game-Based Proofs with the CryptoVerif Proof Assistant

encode security properties here

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

- CryptoVerif constructs a sequence of computationally indistinguishable games
- built-in proof strategy, and detailed guidance by user
- supports indistinguishability, secrecy, authentication properties
- computes exact security probability bound

Thesis Goals

Increase **applicability** and **visibility** of computer-aided proofs

Thesis Statement: Cryptographic proofs of practical usefulness are feasible using automated proof assistants.

Outline

Part I. Case studies on real-world protocols.

- The **Hybrid Public Key Encryption** standard
- The **WireGuard VPN** protocol

Outline

Part I. Case studies on real-world protocols.

- The **Hybrid Public Key Encryption** standard
- The **WireGuard VPN** protocol

Part II. Linking Cryptographic Proofs to Implementations.

- **cv2fstar**: translate CryptoVerif models to executable F* specifications

Part I: The Hybrid Public Key Encryption Standard

Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. “Analysing the HPKE Standard”. EUROCRYPT 2021.

Richard L. Barnes, Karthik Bhargavan, Benjamin Lipp, and Christopher A. Wood. “Hybrid Public Key Encryption”. RFC 9180. February 2022.

Hybrid Public Key Encryption (HPKE)

- *Hybrid* in the spirit of the KEM/DEM paradigm:
  asymmetric building block as Key Encapsulation Mechanism,
  symmetric building block as Data Encapsulation Mechanism

Hybrid Public Key Encryption (HPKE)

- *Hybrid* in the spirit of the KEM/DEM paradigm:
  asymmetric building block as Key Encapsulation Mechanism,
  symmetric building block as Data Encapsulation Mechanism

- RFC 9180 by the Crypto Forum Research Group (CFRG)
  of the Internet Research Task Force (IRTF)

- Usage in TLS 1.3's Encrypted Client Hello (ECH) extension, and
  the Messaging Layer Security (MLS) group messaging protocol, amongst others.

Hybrid Public Key Encryption (HPKE)

- *Hybrid* in the spirit of the KEM/DEM paradigm:
  asymmetric building block as Key Encapsulation Mechanism,
  symmetric building block as Data Encapsulation Mechanism

- RFC 9180 by the Crypto Forum Research Group (CFRG)
  of the Internet Research Task Force (IRTF)

- Usage in TLS 1.3's Encrypted Client Hello (ECH) extension, and
  the Messaging Layer Security (MLS) group messaging protocol, amongst others.

- Requirements: modern crypto, provable security, test vectors, freely implementable.

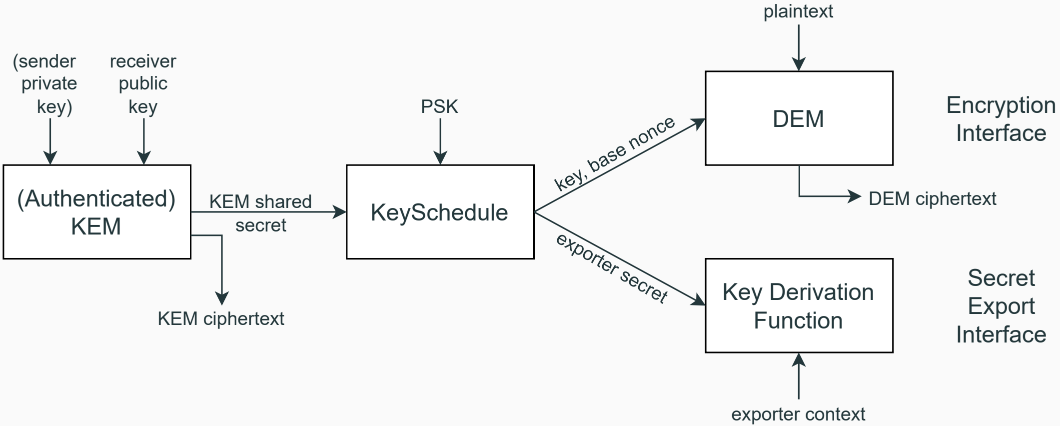
HPKE Specifies Different APIs

- Single-Shot Encryption: encrypt one message
- Single-Shot Secret Export: provide one secret of (almost) *arbitrary* length
- Multi-Shot: multiple messages, and multiple secrets

HPKE Specifies Four Different Modes

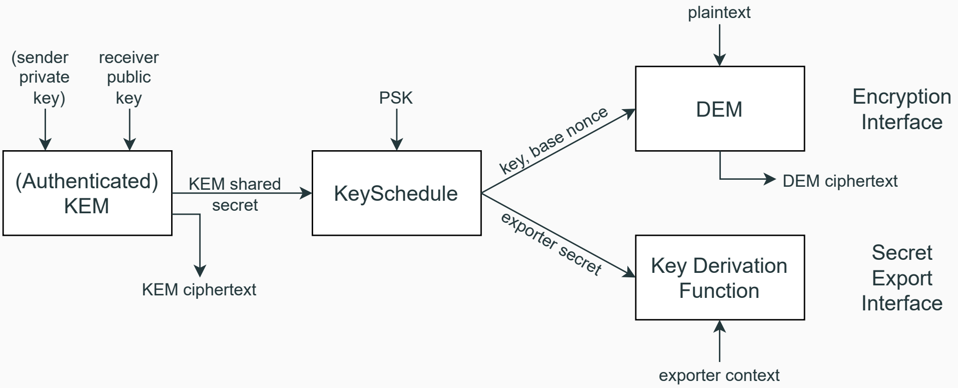
	receiver key pair	sender key pair	pre-shared key
Base	y	n	n
Auth	y	y	n
PSK	y	n	y
AuthPSK	y	y	y

Overview of the Construction



Contributions to the HPKE Standard

1. Preliminary cryptographic analysis **of all modes and interfaces** leading to
   - **redesign of DHKEM** and the key schedule such that DHKEM is CCA-secure on its own
   - introduce **proper oracle separation** using labels



Contributions to the HPKE Standard

2. Updating HPKE specification in the HACL* library
    - has been upstreamed
    - used to compute input parameter length limits, indicated in the RFC

Contributions to the HPKE Standard

3. Detailed cryptographic analysis of the **Auth mode** leading to
    - composition theorems about Auth mode's security
    - with exact security bounds
    - development of the **nominal groups framework** for modeling elliptic curves

Security Notions for AKEM and APKE

Chosen-Ciphertext Indistinguishability (CCA)
confidentiality of AKEM and APKE ciphertexts

Authenticity (Auth)
unforgeability of AKEM and APKE ciphertexts

Security Notions for AKEM and APKE

Chosen-Ciphertext Indistinguishability (CCA)
confidentiality of AKEM and APKE ciphertexts

Authenticity (Auth)
unforgeability of AKEM and APKE ciphertexts

Both of them in two variants:

Outsider adversary can choose from the honest key pairs when calling oracles,
no honest key pair is compromised

Security Notions for AKEM and APKE

Chosen-Ciphertext Indistinguishability (CCA)
confidentiality of AKEM and APKE ciphertexts

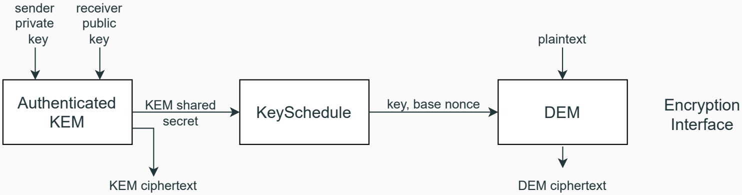
Authenticity (Auth)
unforgeability of AKEM and APKE ciphertexts

Both of them in two variants:

Outsider adversary can choose from the honest key pairs when calling oracles,
no honest key pair is compromised

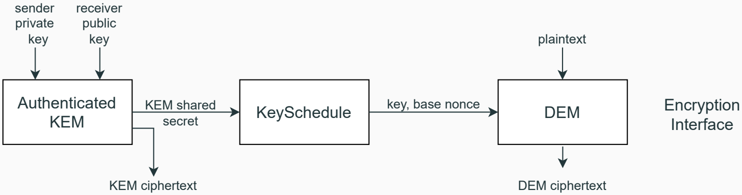
Insider adversary can *provide* sender or receiver *secret* key,
this is stronger than compromise of honestly generated key pairs

Proofs About HPKE's Auth Mode



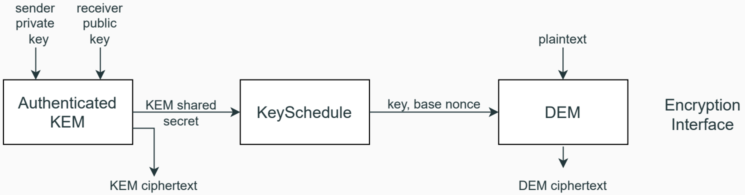
- CryptoVerif: Outsider-CCA, Insider-CCA, Outsider-Auth
  of the standard's Diffie-Hellman-based instantiation of AKEM

Proofs About HPKE's Auth Mode



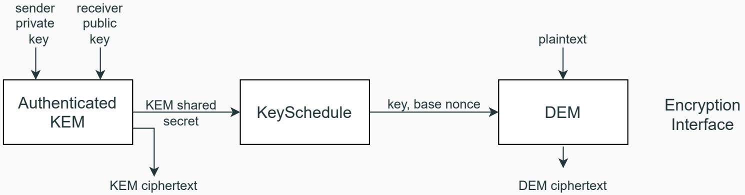
- CryptoVerif: Outsider-CCA, Insider-CCA, Outsider-Auth of the standard's Diffie-Hellman-based instantiation of AKEM
- CryptoVerif: PRF-security of HPKE's KeySchedule

Proofs About HPKE's Auth Mode



- CryptoVerif: Outsider-CCA, Insider-CCA, Outsider-Auth of the standard's Diffie-Hellman-based instantiation of AKEM
- CryptoVerif: PRF-security of HPKE's KeySchedule
- CryptoVerif: **composition theorems** for Outsider-CCA, Insider-CCA, and Outsider-Auth of the AKEM/DEM construction

Proofs About HPKE's Auth Mode



- CryptoVerif: Outsider-CCA, Insider-CCA, Outsider-Auth of the standard's Diffie-Hellman-based instantiation of AKEM
- CryptoVerif: PRF-security of HPKE's KeySchedule
- CryptoVerif: **composition theorems** for Outsider-CCA, Insider-CCA, and Outsider-Auth of the AKEM/DEM construction
- Hand-written non-tight proof of **single-user/two-user** $\Rightarrow$ **multi-user security notions** for AKEM, to close gap to proofs of, e.g., PQ KEMs

Elliptic Curves and Nominal Groups

The HPKE standard allows for different elliptic curves, in particular the NIST curves P-256, P-384, P-521, as well as Curve25519 and Curve448.

Elliptic Curves and Nominal Groups

The HPKE standard allows for different elliptic curves, in particular the NIST curves P-256, P-384, P-521, as well as Curve25519 and Curve448.

- The NIST curves are **prime-order groups**.
- Curve25519 and Curve448 are **not prime-order groups**.

Elliptic Curves and Nominal Groups

The HPKE standard allows for different elliptic curves, in particular the NIST curves P-256, P-384, P-521, as well as Curve25519 and Curve448.

- The NIST curves are **prime-order groups**.
- Curve25519 and Curve448 are **not prime-order groups**.
  For each honestly generated public key, there is a small number of equivalent public keys.

Elliptic Curves and Nominal Groups

The HPKE standard allows for different elliptic curves, in particular the NIST curves P-256, P-384, P-521, as well as Curve25519 and Curve448.

- The NIST curves are **prime-order groups**.
- Curve25519 and Curve448 are **not prime-order groups**.
  For each honestly generated public key, there is a small number of equivalent public keys.

We define a framework of **(rerandomisable) nominal groups** to cover both prime-order and non-prime-order groups in one model.

In short: We do not assume a group structure, but only an exponentiation function with certain properties.

Exact Security of HPKE

Algorithm choices with their security level:

- Elliptic curves from 128 to 256 bits
- Hash functions from 256 to 512 bits
- AEAD with key length from 128 to 256 bits,
  the auth tag length is always 128 bits.

Proof result: the length of the auth tag limits the overall security level to 128 bits.

HPKE: Conclusion and Future Work

HPKE Auth mode satisfies its desired security properties with a **maximum security level of 128 bits**.

Nominal Groups cover prime-order and non-prime-order groups in one model.

HPKE: Conclusion and Future Work

HPKE Auth mode satisfies its desired security properties with a **maximum security level of 128 bits**.

Nominal Groups cover prime-order and non-prime-order groups in one model.

Future Work:

- open question: multi-key security of current AEAD schemes
- detailed analysis of (Auth)PSK mode

Part I: The WireGuard VPN Protocol

Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. "A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol". IEEE EuroS&P 2019.

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

- uses modern cryptography
- no ciphersuite negotiation
  (unlike e. g., TLS)

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

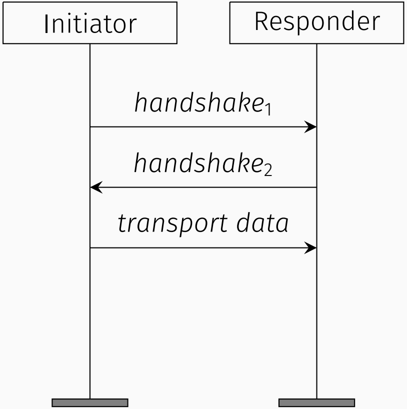
- uses modern cryptography
- no ciphersuite negotiation
  (unlike e. g., TLS)
- aims to replace OpenVPN and IPsec
  - works directly over UDP
  - only a few thousand lines of code

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

- uses modern cryptography
- no ciphersuite negotiation (unlike e. g., TLS)
- aims to replace OpenVPN and IPsec
  - works directly over UDP
  - only a few thousand lines of code
- integration into the **Linux kernel** in 2020
- organizations and VPN providers started adopting it

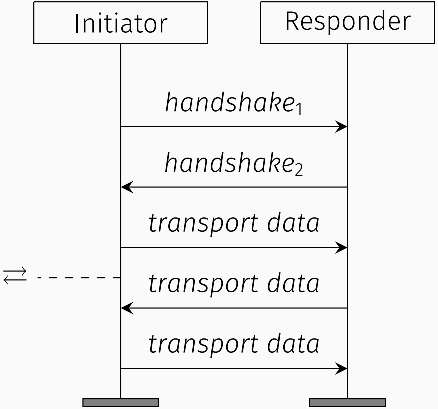
WireGuard's Main Protocol



Based on protocol IKpsk2 from the Noise Protocol Framework

1st transport data msg must come from initiator

WireGuard's Main Protocol



Based on protocol IKpsk2 from the Noise Protocol Framework

1st transport data msg must come from initiator

Our Contributions

Mechanized cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- usual properties for secure channels
- identity hiding
- resistance against denial of service

Our Contributions

Mechanized cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- usual properties for secure channels
- identity hiding
- resistance against denial of service

Reusable contributions:

- Precise model of the Curve25519 elliptic curve for Diffie-Hellman
- Indifferentiability lemmas for chains of random oracle calls

Our Contributions

Mechanized cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- usual properties for secure channels
- identity hiding
- resistance against denial of service

Reusable contributions:

- Precise model of the Curve25519 elliptic curve for Diffie-Hellman
- Indifferentiability lemmas for chains of random oracle calls

Related work:

- WireGuard: DowlingPaterson'18, DonenfeldMilner'18
- IKpsk2: KobeissiNicolasBhargavan'19, Suter-Dörig'18, Girol'19

Analyzed Properties

Usual secure channel properties:

Confidentiality
- Secrecy and
- Forward secrecy of transport data messages

Analyzed Properties

Usual secure channel properties:

Confidentiality
- Secrecy and
- Forward secrecy of transport data messages

Agreement
- Mutual authentication (as of 2nd protocol msg)
- Session uniqueness
- Channel binding
- Resistance against key compromise impersonation
- Resistance against identity mis-binding
  (except theoretical attack)

Analyzed Properties

Usual secure channel properties:

Confidentiality
- Secrecy and
- Forward secrecy of transport data messages

Agreement
- Mutual authentication (as of 2nd protocol msg)
- Session uniqueness
- Channel binding
- Resistance against key compromise impersonation
- Resistance against identity mis-binding (except theoretical attack)

Additional properties in WireGuard:

- Resistance against denial of service (no replay of 1st msg, cookie enforces round-trip)
- Identity hiding (weak)

Chain of Random Oracle Calls

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \text{const} \\
C &\leftarrow \text{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \text{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \text{hkdf}(C, v_2) \\
C &\leftarrow \text{hkdf}(C, v_3) \\
C &\leftarrow \text{hkdf}(C, v_4) \\
C &\leftarrow \text{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \text{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \text{hkdf}(C, \epsilon)
\end{aligned}
$$

Game Size Explosion in CryptoVerif

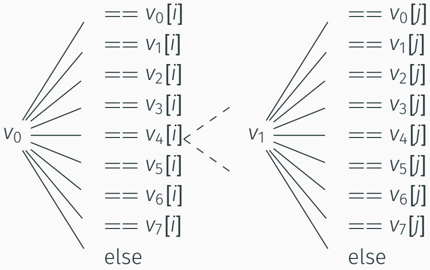
Considering all collision cases leads to exponential growth of number of branches:

v_0

$== v_0[i]$
$== v_1[i]$
$== v_2[i]$
$== v_3[i]$
$== v_4[i]$
$== v_5[i]$
$== v_6[i]$
$== v_7[i]$
else

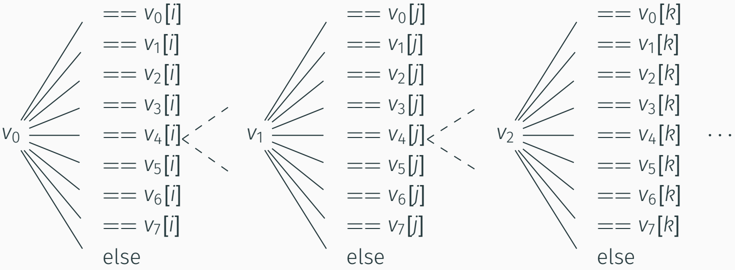
Game Size Explosion in CryptoVerif

Considering all collision cases leads to exponential growth of number of branches:



Game Size Explosion in CryptoVerif

Considering all collision cases leads to exponential growth of number of branches:



Simplification of the Random Oracle Chain

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \mathrm{const} \\
C &\leftarrow \mathrm{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \mathrm{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \mathrm{hkdf}(C, v_2) \\
C &\leftarrow \mathrm{hkdf}(C, v_3) \\
C &\leftarrow \mathrm{hkdf}(C, v_4) \\
C &\leftarrow \mathrm{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \mathrm{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{hkdf}(C, \epsilon)
\end{aligned}
$$

Simplification of the Random Oracle Chain

8 chained calls to
one random oracle.

$$\begin{aligned}
C &\leftarrow \text{const} \\
C &\leftarrow \text{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \text{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \text{hkdf}(C, v_2) \\
C &\leftarrow \text{hkdf}(C, v_3) \\
C &\leftarrow \text{hkdf}(C, v_4) \\
C &\leftarrow \text{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \text{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \text{hkdf}(C, \epsilon)
\end{aligned}$$

3 independent calls to
3 *independent* random oracles.

$$\begin{aligned}
k_1 &\leftarrow \text{chain}_1(v_0, v_1) \\
k_2 &\leftarrow \text{chain}_2(v_0, v_1, v_2) \\
\pi \| k_3 \| T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \text{chain}_6(v_0, v_1, v_2, v_3, v_4, v_5, v_6)
\end{aligned}$$

Simplification of the Random Oracle Chain

Indifferentiable in any context:

8 chained calls to
one random oracle.

$$\begin{aligned}
C &\leftarrow \text{const} \\
C &\leftarrow \mathrm{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \mathrm{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \mathrm{hkdf}(C, v_2) \\
C &\leftarrow \mathrm{hkdf}(C, v_3) \\
C &\leftarrow \mathrm{hkdf}(C, v_4) \\
C &\leftarrow \mathrm{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \mathrm{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{hkdf}(C, \epsilon)
\end{aligned}$$

3 independent calls to
3 *independent* random oracles.

$$\begin{aligned}
k_1 &\leftarrow \mathrm{chain}_1(v_0, v_1) \\
k_2 &\leftarrow \mathrm{chain}_2(v_0, v_1, v_2) \\
\pi \| k_3 \| T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{chain}_6(v_0, v_1, v_2, v_3, v_4, v_5, v_6)
\end{aligned}$$

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

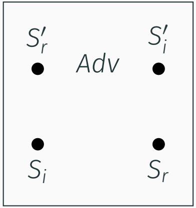
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

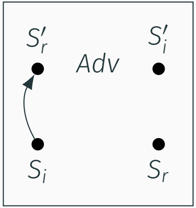
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

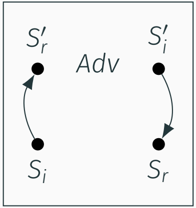
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

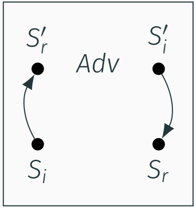
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

$\rightarrow$ The two sessions derive the same traffic keys but are between different parties.

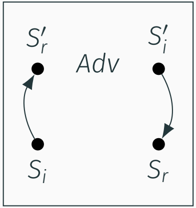
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or an honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

$\rightarrow$ The two sessions derive the same traffic keys but are between different parties.

Mitigation: include static public keys S_i^{pub} and S_r^{pub} into key derivation

WireGuard: Conclusion and Future Work

WireGuard protocol is cryptographically secure

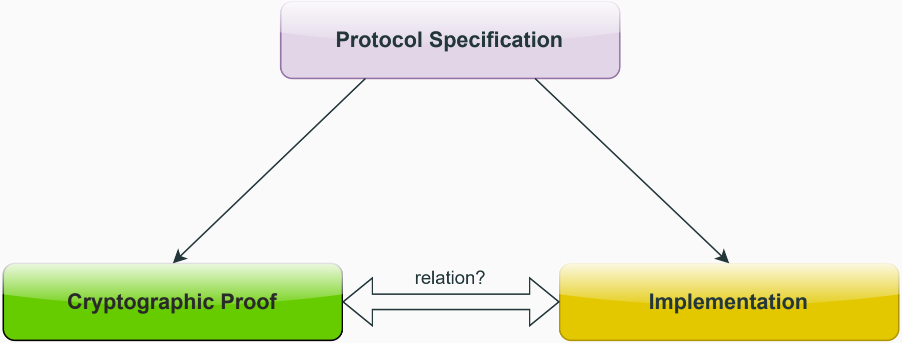
- theoretical identity mis-binding attack
- weak identity hiding

Possible future work

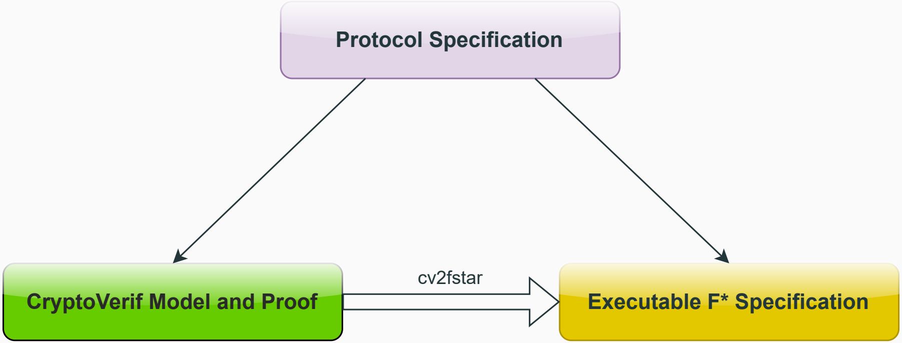
- more Noise protocols
- use PRF-ODH assumption

Part II: Translate CryptoVerif Models to Executable $F^\star$ Specifications

Problem Statement



Proposal: cv2fstar



Output Language: F⋆

> "F⋆ is a general-purpose functional programming language aimed at program verification"
>
> — www.fstar-lang.org

- rich type system
- interactive proofs and discharging to SMT

Output Language: F⋆

> “F⋆ is a general-purpose functional programming language aimed at program verification”
>
> — www.fstar-lang.org

- rich type system
- extracts to OCaml and F#;
  C, WebAssembly (KaRaMeL)
- interactive proofs and discharging to SMT
- HACL⋆ High-Assurance Cryptographic Library

  used in Mozilla’s NSS, Windows Kernel, Linux Kernel, WireGuard, Microsoft’s QUIC implementation, Tezos blockchain

Detailed Motivation

1. obtain implementation with cryptographic security guarantees
   $\rightarrow$ cv2fstar provides an automatic translation from CryptoVerif

2. provably instantiate non-cryptographic assumptions of the CryptoVerif model
   $\rightarrow$ cv2fstar generates lemmas as proof obligations

3. reuse CryptoVerif theorems for further proofs
   $\rightarrow$ future work: translate as assumed lemmas

Related Work

Large body of research, in three groups:

1. Model $\rightarrow$ Implementation
2. Implementation $\rightarrow$ Model
3. Proofs on Code

Related Work

Large body of research, in three groups:

1. Model $\rightarrow$ Implementation
2. Implementation $\rightarrow$ Model
3. Proofs on Code

cv2fstar uses approach (1). Builds upon cv2ocaml.

F⋆ Specifications in Pure State-Passing Style

state = entropy $*$ tables $*$ sessions $*$ events

entropy: explicitly track randomness used for random sampling

tables: each table has an append-only list of entries

sessions: map of session ID $\rightarrow$ list of session entries.
enforces oracle order, implements variable scope in oracle sequences

events: append-only list of events

F⋆ Specifications in Pure State-Passing Style

state = entropy $*$ tables $*$ sessions $*$ events

entropy: explicitly track randomness used for random sampling

tables: each table has an append-only list of entries

sessions: map of session ID $\rightarrow$ list of session entries.
enforces oracle order, implements variable scope in oracle sequences

events: append-only list of events

```
val oracle: state -> nat -> t -> state * option (nat * T)
```

Translating Oracle Sequences Using Sessions

```
O1(...) :=
  a <- ...;
  ...
  return(...);

O2(...) :=
  b <- a;
  ...
  return(...);

O3(...) :=
  c <- f(a, b);
  ...
  return(...).
```

CryptoVerif semantics:

- Oracles can only be called in order
- Variables stay in scope for following oracles

Translating Oracle Sequences Using Sessions

```
O1(...) :=
  a <- ...;
  ...
  return(...);

O2(...) :=
  b <- a;
  ...
  return(...);

O3(...) :=
  c <- f(a, b);
  ...
  return(...).
```

CryptoVerif semantics:

- Oracles can only be called in order
- Variables stay in scope for following oracles

F⋆ implementation:

- before returning: store free variables of following oracles in session entry

Translating Oracle Sequences Using Sessions

```
O1(...) :=
  a <- ...;
  ...
  return(...);

O2(...) :=
  b <- a;
  ...
  return(...);

O3(...) :=
  c <- f(a, b);
  ...
  return(...).
```

CryptoVerif semantics:

- Oracles can only be called in order
- Variables stay in scope for following oracles

F⋆ implementation:

- before returning: store free variables of following oracles in session entry
- return a session ID

Translating Oracle Sequences Using Sessions

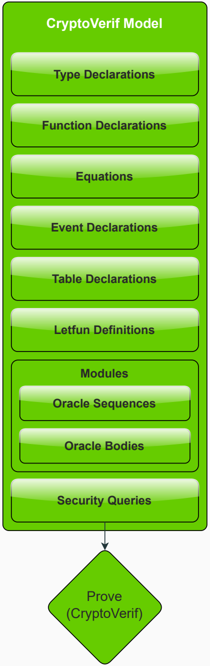
```
O1(...) :=
  a <- ...;
  ...
  return(...);

O2(...) :=
  b <- a;
  ...
  return(...);

O3(...) :=
  c <- f(a, b);
  ...
  return(...).
```

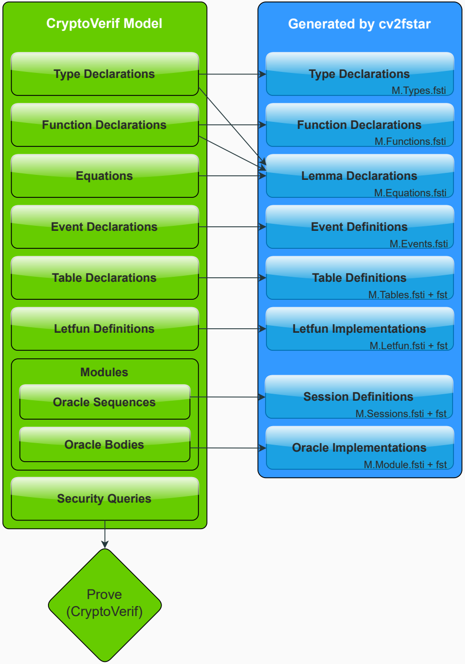
CryptoVerif semantics:

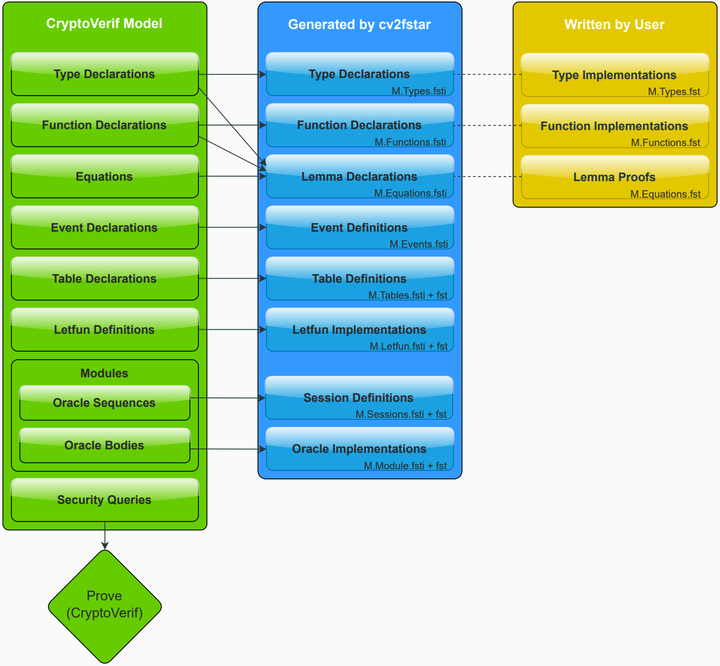
- Oracles can only be called in order
- Variables stay in scope for following oracles

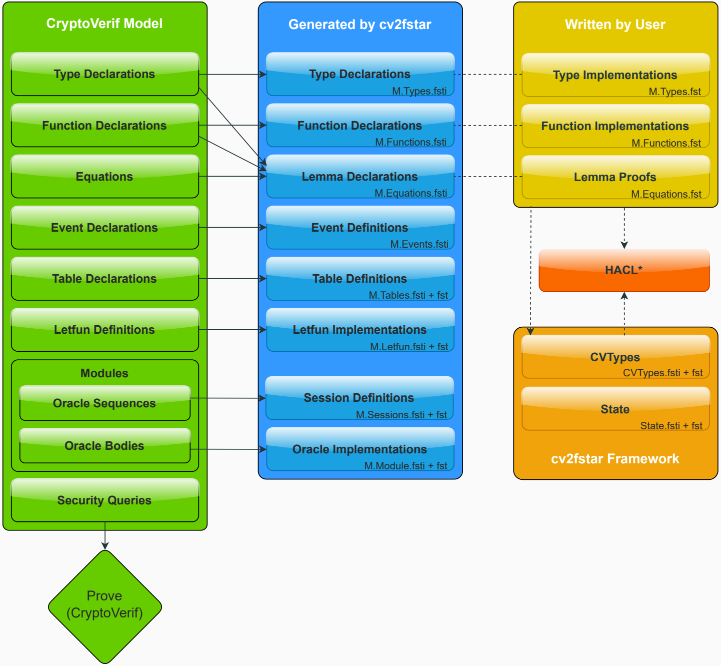
F* implementation:

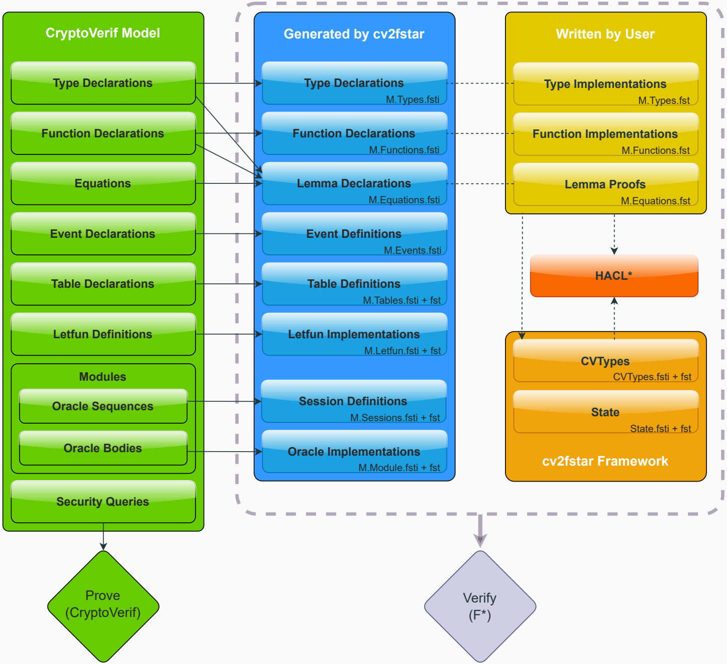
- before returning: store free variables of following oracles in session entry
- return a session ID
- following oracle is called with session ID, retrieves values of free variables


CryptoVerif Model
Type Declarations
Function Declarations
Equations
Event Declarations
Table Declarations
Letfun Definitions
Modules
Oracle Sequences
Oracle Bodies
Security Queries
Prove
(CryptoVerif)


CryptoVerif Model
Type Declarations
Function Declarations
Equations
Event Declarations
Table Declarations
Letfun Definitions
Modules
Oracle Sequences
Oracle Bodies
Security Queries
Prove (CryptoVerif)
Generated by cv2fstar
Type Declarations
M.Types.fsti
Function Declarations
M.Functions.fsti
Lemma Declarations
M.Equations.fsti
Event Definitions
M.Events.fsti
Table Definitions
M.Tables.fsti + fst
Letfun Implementations
M.Letfun.fsti + fst
Session Definitions
M.Sessions.fsti + fst
Oracle Implementations
M.Module.fsti + fst


CryptoVerif Model
Type Declarations
Function Declarations
Equations
Event Declarations
Table Declarations
Letfun Definitions
Modules
Oracle Sequences
Oracle Bodies
Security Queries
Prove (CryptoVerif)
Generated by cv2fstar
Type Declarations
M.Types.fsti
Function Declarations
M.Functions.fsti
Lemma Declarations
M.Equations.fsti
Event Definitions
M.Events.fsti
Table Definitions
M.Tables.fsti + fst
Letfun Implementations
M.Letfun.fsti + fst
Session Definitions
M.Sessions.fsti + fst
Oracle Implementations
M.Module.fsti + fst
Written by User
Type Implementations
M.Types.fst
Function Implementations
M.Functions.fst
Lemma Proofs
M.Equations.fst


CryptoVerif Model
Type Declarations
Function Declarations
Equations
Event Declarations
Table Declarations
Letfun Definitions
Modules
Oracle Sequences
Oracle Bodies
Security Queries
Prove (CryptoVerif)
Generated by cv2fstar
Type Declarations
M.Types.fsti
Function Declarations
M.Functions.fsti
Lemma Declarations
M.Equations.fsti
Event Definitions
M.Events.fsti
Table Definitions
M.Tables.fsti + fst
Letfun Implementations
M.Letfun.fsti + fst
Session Definitions
M.Sessions.fsti + fst
Oracle Implementations
M.Module.fsti + fst
Written by User
Type Implementations
M.Types.fst
Function Implementations
M.Functions.fst
Lemma Proofs
M.Equations.fst
HACL*
CVTypes
CVTypes.fsti + fst
State
State.fsti + fst
cv2fstar Framework


CryptoVerif Model
Type Declarations
Function Declarations
Equations
Event Declarations
Table Declarations
Letfun Definitions
Modules
Oracle Sequences
Oracle Bodies
Security Queries
Prove
(CryptoVerif)
Generated by cv2fstar
Type Declarations
M.Types.fsti
Function Declarations
M.Functions.fsti
Lemma Declarations
M.Equations.fsti
Event Definitions
M.Events.fsti
Table Definitions
M.Tables.fsti + fst
Letfun Implementations
M.Letfun.fsti + fst
Session Definitions
M.Sessions.fsti + fst
Oracle Implementations
M.Module.fsti + fst
Written by User
Type Implementations
M.Types.fst
Function Implementations
M.Functions.fst
Lemma Proofs
M.Equations.fst
HACL*
CVTypes
CVTypes.fsti + fst
State
State.fsti + fst
cv2fstar Framework
Verify
(F*)

Non-Cryptographic Assumptions to Lemmas

4 sources from which lemmas are generated as proof obligations:

1. explicit equations

```
equation forall v_1:t_1, ..., v_n:t_n; M if M'.
```

Non-Cryptographic Assumptions to Lemmas

4 sources from which lemmas are generated as proof obligations:

1. explicit equations

   ```
   equation forall v_1:t_1, ..., v_n:t_n; M if M'.
   ```

2. built-in equational theories
   e.g., commutativity, associativity, groups

   ```
   equation builtin commut_group(f, inv, n).
   ```

Non-Cryptographic Assumptions to Lemmas

4 sources from which lemmas are generated as proof obligations:

1. explicit equations

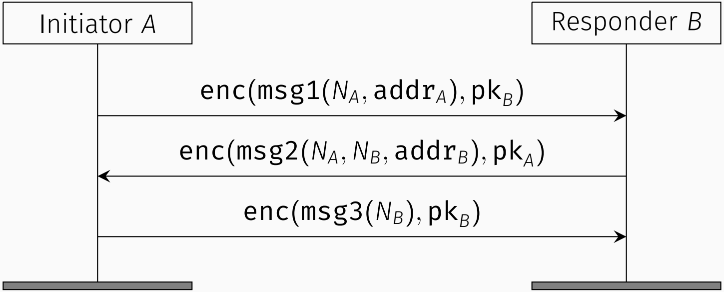
   ```
   equation forall v_1:t_1, ..., v_n:t_n; M if M'.
   ```

2. built-in equational theories
   e.g., commutativity, associativity, groups

   ```
   equation builtin commut_group(f, inv, n).
   ```

3. correctness of inverses for functions declared `[data]`
4. correctness of inverses for (de)serialization functions

Case Study: The Needham-Schroeder-Lowe Protocol (NSL)



NSL: Source Code Statistics

		# files	# SLOC	# PLOC
cv2fstar handwritten framework	F*	12	870	150
NSL handwritten	CryptoVerif	1	170	0
NSL generated by cv2fstar	F*	20	910	0
NSL handwritten	F*	3	370	100

51 lemmas generated for NSL
42 trivial for F*: proven by `()`
18 coming from CryptoVerif's standard library (all trivial for F*)

≈ 100 lines of proof code in NSL.Equations.fst, 150 in CVTypes.fst(i) combined.

cv2fstar: Conclusion

We have shown that cv2fstar

- allows to translate CryptoVerif models to executable code
- interoperable with HACL⋆
- allows to fill in implementation details in F⋆,
  and to prove that they fulfill the CryptoVerif model's assumptions

cv2fstar: Conclusion

We have shown that cv2fstar

- allows to translate CryptoVerif models to executable code
- interoperable with HACL⋆
- allows to fill in implementation details in F⋆,
  and to prove that they fulfill the CryptoVerif model's assumptions

Future Work (excerpt)

- Translate CryptoVerif theorems to F⋆:
  correspondance, secrecy, indistinguishability properties
- WireGuard case study, link with Noise* implementation

Thesis Conclusion

Thesis Conclusion: Part I

- Writing highly-detailed cryptographic proofs using CryptoVerif is feasible for real-world protocols.
- More detailed models of elliptic curves than any handwritten analysis before: use Curve25519 with confidence.
- WireGuard: carefully modeled very close to its implementation.
- HPKE: concrete security bounds; influence on standard.

Thesis Conclusion: Part II

cv2fstar compiles CryptoVerif models to executable F* specifications.

- cryptographic properties on executable code
- prove properties about implementation details

State of the User Base of Cryptographic Proof Assistants?

State of the User Base of Cryptographic Proof Assistants?

> "I believe that if a tool like that is built well, it will be adopted and used by many. Wouldn't you like to be cited by half of the papers appearing in CRYPTO 2010? Here is your chance..."
>
> — Halevi in the year 2005 [Hal05]

State of the User Base of Cryptographic Proof Assistants?

> "I believe that if a tool like that is built well, it will be adopted and used by many. Wouldn't you like to be cited by half of the papers appearing in CRYPTO 2010? Here is your chance..."
>
> — Halevi in the year 2005 [Hal05]

> "formalistic approaches are gone [from cryptography's tier-1 venues] (unless they claim to bridge to 'real' crypto)"
>
> — Rogaway in the year 2015 [Rog15]

State of the User Base of Cryptographic Proof Assistants?

> "I believe that if a tool like that is built well, it will be adopted and used by many. Wouldn't you like to be cited by half of the papers appearing in CRYPTO 2010? Here is your chance..."
>
> — Halevi in the year 2005 [Hal05]

> "formalistic approaches are gone [from cryptography's tier-1 venues] (unless they claim to bridge to 'real' crypto)"
>
> — Rogaway in the year 2015 [Rog15]

Interest and demand is clearly there, from cryptographers and standardization bodies.

Challenges

Multitude of proof methodologies and security notions:
game-based, simulation-based, UC, state-separating proofs.

“does not have a very appealing ‘business case”’
— Halevi [Hal05]

Challenges:

- small user base
- incentives and funding for researchers and developers?

Specific Goals?

- Teaching material
- More accessible user interfaces
- Artifact evaluation at more publication venues. (CHES is leading by example since 2021!)

→ Build up trust in the tools within the community.

Future Work

Generally: continue using and developing cryptographic proof assistants in the area of practice-oriented provable security.

- Collaborate with cryptographers, support standardization efforts
- Usability of proof assistants
- Consider quantum adversaries

Computer-aided cryptographic proofs are an exciting and in-demand research effort, with many promising open problems!