RFC-014: Coq Extraction for TCB

Status: Proposed **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.n et

---

Abstract

This RFC specifies the use of Coq proof assistant for verified implementation of the Trusted Computing Base, with extraction to OCaml for production use. Prove once, trust forever.

Motivation

The Prime Directive (RFC-002):

> *If it's in the TCB, it's in OCaml. Otherwise it's in Chicken Scheme.*

But even OCaml can have bugs. The TCB handles: - Ed25519 signatures - SHA-512 hashing - Signature chain verification

A single bug breaks everything.

Coq provides:

1. **Machine-checked proofs**: Theorems verified by computer
2. **Extraction**: Generate OCaml from proofs
3. **Correctness by construction**: Implementation matches specification
4. **Eternal validity**: Proofs don't expire

From the Coq motto:

> *The proof is in the code.*

Specification

Trusted Computing Base

```
┌─────────────────────────────────────────────────────────────────┐
│                        CYBERSPACE TCB                           │
│                                                                 │
│   ┌─────────────┐   ┌─────────────┐   ┌─────────────┐          │
│   │  Ed25519    │   │  SHA-512    │   │  Verify     │          │
│   │  Proven     │   │  Proven     │   │  Proven     │          │
│   └──────┬──────┘   └──────┬──────┘   └──────┬──────┘          │
│          │                 │                 │                  │
│          └─────────────────┼─────────────────┘                  │
│                            │                                    │
```

```
│                        │                             │
│                  Coq Extraction                      │
│                        │                             │
│                  ┌─────□─────┐                       │
│                  │   OCaml   │                       │
│                  │ ~1000 LOC │                       │
│                  └───────────┘                       │
│                                                      │
│   Proven in Coq. Extracted to OCaml. Called from Scheme.   │
│                                                      │
└──────────────────────────────────────────────────────┘
```

Coq Specifications

Types

```
(* Byte arrays *)
Definition bytes := list byte.

(* Keys *)
Record ed25519_public_key := {
  pk_bytes : bytes;
  pk_length : length pk_bytes = 32
}.

Record ed25519_private_key := {
  sk_bytes : bytes;
  sk_length : length sk_bytes = 64
}.

(* Signatures *)
Record ed25519_signature := {
  sig_bytes : bytes;
  sig_length : length sig_bytes = 64
}.

(* Hashes *)
Record sha512_hash := {
  hash_bytes : bytes;
  hash_length : length hash_bytes = 64
}.
```

Signature Specification

```
(* Abstract signature scheme *)
Module Type ED25519_SPEC.
  Parameter sign : ed25519_private_key -> bytes -> ed25519_signature.
  Parameter verify : ed25519_public_key -> bytes -> ed25519_signature -> bool.

  (* Correctness: valid signatures verify *)
  Axiom sign_verify_correct :
    forall sk pk msg,
      pk = derive_public sk ->
      verify pk msg (sign sk msg) = true.

  (* Security: cannot forge without private key *)
  Axiom unforgeability :
    forall pk msg sig,
      verify pk msg sig = true ->
      exists sk, pk = derive_public sk /\ sig = sign sk msg.
End ED25519_SPEC.
```

Hash Specification

```
Module Type SHA512_SPEC.
  Parameter hash : bytes -> sha512_hash.

  (* Determinism *)
  Axiom hash_deterministic :
    forall x, hash x = hash x.

  (* Collision resistance (assumed) *)
  Axiom collision_resistant :
    forall x y, hash x = hash y -> x = y. (* Idealized *)
End SHA512_SPEC.
```

Chain Verification

```
(* Certificate chain verification *)
Fixpoint verify_chain
  (root : ed25519_public_key)
  (certs : list signed_cert)
  (target_tag : tag) : bool :=
  match certs with
  | nil => false
  | cert :: rest =>
      let issuer_key := cert_issuer cert in
      let subject_key := cert_subject cert in
      let cert_tag := cert_tag cert in
```

```
      (* Check issuer matches current key *)
      andb (key_eq root issuer_key)
      (* Check signature valid *)
      (andb (verify issuer_key (cert_content cert) (cert_signature cert))
      (* Check tag grants permission *)
      (andb (tag_implies cert_tag target_tag)
      (* Continue chain *)
      (match rest with
       | nil => true
       | _ => verify_chain subject_key rest target_tag
       end)))
  end.

(* Theorem: Valid chain implies authorization *)
Theorem chain_authorization :
  forall root certs tag,
    verify_chain root certs tag = true ->
    authorized root tag.
Proof.
  (* Proof by induction on chain length *)
  ...
Qed.
```

Extraction to OCaml

Extraction Directives

```
Require Import ExtrOcamlBasic.
Require Import ExtrOcamlString.

(* Extract to OCaml types *)
Extract Inductive bool => "bool" ["true" "false"].
Extract Inductive list => "list" ["[]" "(::)"].

(* Link to libsodium *)
Extract Constant ed25519_sign => "Sodium.Ed25519.sign".
Extract Constant ed25519_verify => "Sodium.Ed25519.verify".
Extract Constant sha512_hash => "Sodium.SHA512.hash".

(* Generate OCaml *)
Extraction "tcb.ml" verify_chain sign verify hash.
```

Generated OCaml

```ocaml
(* tcb.ml - Extracted from Coq *)

let rec verify_chain root certs target_tag =
  match certs with
  | [] -> false
  | cert :: rest ->
      let issuer_key = cert_issuer cert in
      let subject_key = cert_subject cert in
      key_eq root issuer_key &&
      Sodium.Ed25519.verify issuer_key (cert_content cert) (cert_signature cert) &&
      tag_implies (cert_tag cert) target_tag &&
      (match rest with
       | [] -> true
       | _ -> verify_chain subject_key rest target_tag)
```

Integration with Scheme

FFI Layer

```ocaml
(* tcb_ffi.ml - FFI bindings for Chicken Scheme *)

let () = Callback.register "tcb_verify_chain" verify_chain
let () = Callback.register "tcb_sign" sign
let () = Callback.register "tcb_verify" verify
let () = Callback.register "tcb_hash" hash
```

Scheme Bindings

```scheme
;; crypto-ffi.scm
(module crypto-ffi
  (ed25519-sign ed25519-verify sha512-hash verify-chain)

  (import (chicken foreign))

  ;; Call into verified OCaml
  (define ed25519-sign
    (foreign-lambda* blob ((blob key) (blob msg))
      "return tcb_sign(key, msg);"))

  (define verify-chain
    (foreign-lambda* bool ((blob root) (pointer certs) (pointer tag))
      "return tcb_verify_chain(root, certs, tag);"))
```

...)

Proof Obligations

What We Prove

1. **Signature correctness**: Valid signatures verify
2. **Chain soundness**: Valid chain implies authorization
3. **Hash properties**: Determinism, length preservation
4. **Type safety**: No buffer overflows, no null pointers

What We Assume

1. **Cryptographic hardness**: Ed25519 unforgeability
2. **libsodium correctness**: Implementation matches spec
3. **OCaml runtime**: Extraction target is correct
4. **Hardware**: CPU executes instructions correctly

Trust Chain

```
Mathematical proof (Coq)
    ↓
Extraction (verified by Coq)
    ↓
OCaml code (typed, memory-safe)
    ↓
libsodium (audited, widely deployed)
    ↓
CPU instructions (trust hardware)
```

Development Workflow

1. Specify in Coq

```
(* Define types and operations *)
(* State properties and theorems *)
```

2. Prove Correctness

```
(* Prove theorems *)
(* Coq checks proofs mechanically *)
```

3. Extract to OCaml

```
$ coqc -R . Cyberspace tcb.v
$ coqc -R . Cyberspace extract.v
$ ls *.ml
tcb.ml tcb_types.ml
```

4. Compile and Link

```
$ ocamlfind ocamlopt -package sodium -linkpkg \
    tcb.ml tcb_ffi.ml -o tcb.cmxa
```

5. Call from Scheme

```
(import crypto-ffi)
(ed25519-sign key message)  ; Calls verified code
```

Existing Verified Libraries

Fiat-Crypto

- Verified elliptic curve implementations
- Used by BoringSSL, Chrome
- Extraction to C, Java, Go

HACL*

- Verified cryptographic library
- Written in F* (similar to Coq)
- Used by Firefox, Wireguard

Potential Use

```
Require Import Fiat.Crypto.Ed25519.
(* Use pre-verified Ed25519 implementation *)
```

Security Considerations

Verified Components

- Signature operations
- Hash operations
- Chain verification logic

Unverified (Trusted)

- FFI layer (small, auditable)
- libsodium bindings
- Scheme runtime

Audit Surface

```
Total TCB:     ~1000 lines OCaml
Verified:      ~800 lines (extracted from ~2000 lines Coq)
Trusted:       ~200 lines (FFI, bindings)
```

References

1. Coq Development Team. The Coq Proof Assistant Reference Manual.
2. Erbsen, A., et al. (2019). Simple High-Level Code for Cryptographic Arithmetic.
3. Protzenko, J., et al. (2017). Verified Low-Level Programming Embedded in F*.
4. Chlipala, A. (2013). Certified Programming with Dependent Types.
5. RFC-002: Cyberspace Architecture

Changelog

- **2026-01-06** - Initial specification

Implementation Status: Proposed **Proof Assistant:** Coq **Extraction Target:** OCaml **TCB Size:** ~1000 lines