# Verification of subsets of RUST

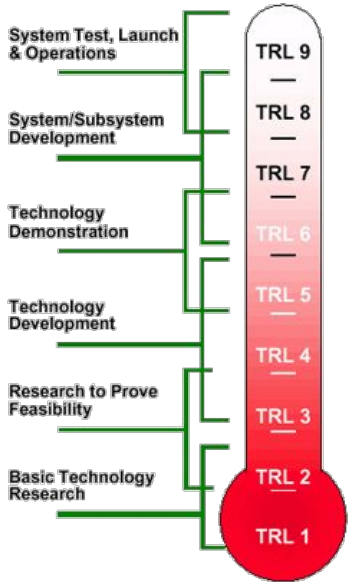
Bas Spitters

COBRA
CONCORDIUM BLOCKCHAIN
RESEARCH CENTER AARHUS

# CoBRA

secure distributed systems, zero-knowledge, …

formal verification, smart contracts, high assurance cryptography




Concordium: based in science
DevX concordium/cobra collab

# Formal verification

Want to avoid bugs:

- Bugs in cryptography allow people to print money
- Bugs in smart contracts allow people to steal millions

Formal methods (RFVIG)

- **Type checkers**
- Unit tests, property based testing (quickcheck, proptest)
- **Formal verification** (automatic, interactive)

# Interactive formal verification

Automatic tools can proof some properties of software
Advanced proofs need a human to interact with a *computer proof assistant*

Examples: compilers, hypervisor, smart contracts, …
State of the art in programming language *research*

~40% of papers at POPL conference come with a formal proof

E.g. functional correctness, properties of PL:
type safety, preservation of semantics, ...

# Coq proof assistant

Functional programming language
Logic/Type theory for specifications

Small kernel of logical rules

Used by 10ks users

```
Definition hd (default:A) (l:list A) ≜
  match l with
    | [] ⇒ default
    | x :: _ ⇒ x
  end.

Definition hd_error (l:list A) ≜
  match l with
    | [] ⇒ None
    | x :: _ ⇒ Some x
  end.

Definition tl (l:list A) ≜
  match l with
    | [] ⇒ nil
    | a :: m ⇒ m
  end.

(** The [In] predicate *)
Fixpoint In (a:A) (l:list A) : Prop ≜
  match l with
    | [] ⇒ ⊥
    | b :: m ⇒ b = a ∨ In a m
  end.
```

[Software foundations book]

# (Biased) State of the art of FV in rust

Unlike C, no precise rust semantics (but ferrocene)

Wasm has a precise and formalized semantics


LLVM (vellvm)


RustBelt (~MIR)

A very long term goal: verified semantics for rust

# Rust as an onion

From hacspec to unsafe rust

First steps:

- Hacspec (pure)
- fiat-rust/bedrock (small imperative language with expressions, e.g. for crypto)
- Pure (functional) rust for smart contracts

# HacSpec

HAC (high assurance cryptography)

Functionally correct, cryptographically secure, fast, constant time

- Rust as a specification language:
  From IETF pseudocode to rust
- Subset of rust with a precise (operational) semantics, type checker
  Not yet blessed by the rust community
- Backends in proof assistants: F*,easycrypt, **Coq**
  E.g. prove group laws for ECC instead of testing them
- Ex: SHA-256, ... , **BLS** (IETF proposal, Concordium ID-layer)

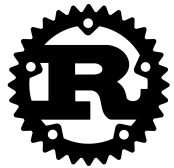
# Fiat/bedrock

Generating platform independent, correct implementations:

Fiat-cryptography (MIT):

Verified partial evaluation from Coq to a small imperative language
printed to rust

Straight-line code … constant time

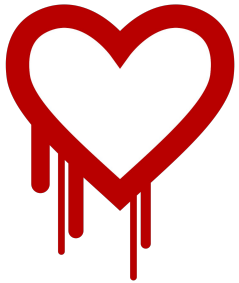


Bedrock adds loops and function calls

WIP generating of efficient rust implementations (w/Diego)

E.g. Field inversion, BLS

# Protocols: TLS, Noise



Popular verification target after Heartbleed

- Everest: functional correctness C code in F*
- HMAC in Coq: verified C code, cryptographically secure

Modular analysis of TLS1.3 cryptographic security uses State Separating proofs.
Need to connect pseudo-code to code: SSProve.

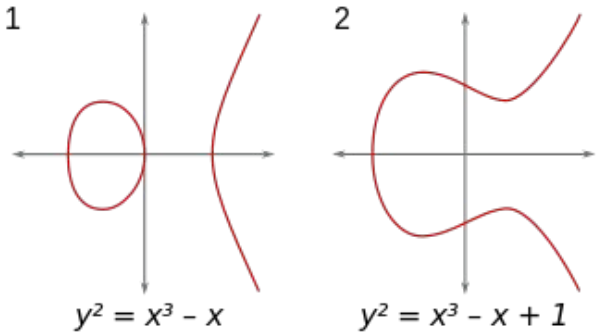
- Possible building blocks for verification of RustTLS, rust-noise and other complex protocols
- Sigma-protocols for Zero-Knowledge in SSProve (ID-layer)
- First formal proof of Safety and liveness of Nakamoto consensus

# From specification to implementation

pseudocode -> Hacspec -> Coq -> fiat library -> rust

Case study: From the BLS specification,

generate an efficient implementation



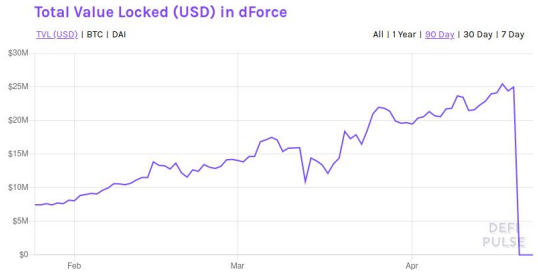

Concordium, linux kernel, wireguard, mirageos,

… are using HAC in production

# Smart contracts

Small programs on the blockchain. Concordium: wasm on chain, rust to generate wasm

Can automate simple banking tasks
Examples: deFi: tokens, exchange, escrow, voting, …




Big hacks: DAO, uniswap, burgerswap, dForce,...
of contracts written in *legacy* solidity (js) language
Losing 10s of millions due to simple programming errors.
ConCert: Writing a specification, we've found bugs using quickcheck.
Prove adherence to the specification in Coq.
Player 1 has a winning strategy in TicTacToe.

# ConCert

Mathematical model of a smart contract as **interacting** pure programs

But there's more … *verified* extraction to rust (Coq workshop)

Proving functional programs correct:
Write them in a dependently typed language (Coq)
and erase the complex types, keeping the lambda terms. Cf. refinement types in haskell (liquidhaskell)

Reverse: {l:list | len l =n} -> {l:list | len l =n}

Meta-coq meta-programming

For smart contracts we extract to rust's Arenas.
Memory is freed after program terminates. Running on concordium stagenet!

# Conclusions

- From specification to implementation in rust
- High assurance cryptography in rust (w/RFVIG, RCIG)
- Verified smart contracts to rust

Concordium:

- HAC in production
- Wasm formalized semantics
  verified smart contracts on stagenet

COBRA
CONCORDIUM BLOCKCHAIN
RESEARCH CENTER AARHUS