Using Hax for Correct and Secure Zero-Knowledge Implementations

Lasse Letager Hansen (letager@cs.au.dk), Eske Hoy Nielsen, Nikolaj Sidorenco, and Bas Spitters

AARHUS UNIVERSITY

Department of Computer Science, Aarhus University, Denmark

Can we trust online voting?

How do we ensure the security of online voting implemented by smart contracts?

EU DATA ACT SMART CONTRACTS
(30a) robustness and access control:

> ensure that the smart contract has been designed to offer [...] a very high degree of robustness to avoid functional errors and to withstand manipulation by third parties.

SWISS E-VOTING REGULATION

> Requires formal security proofs in the computational model

Why is voting relevant for blockchains

Smart Contract Voting

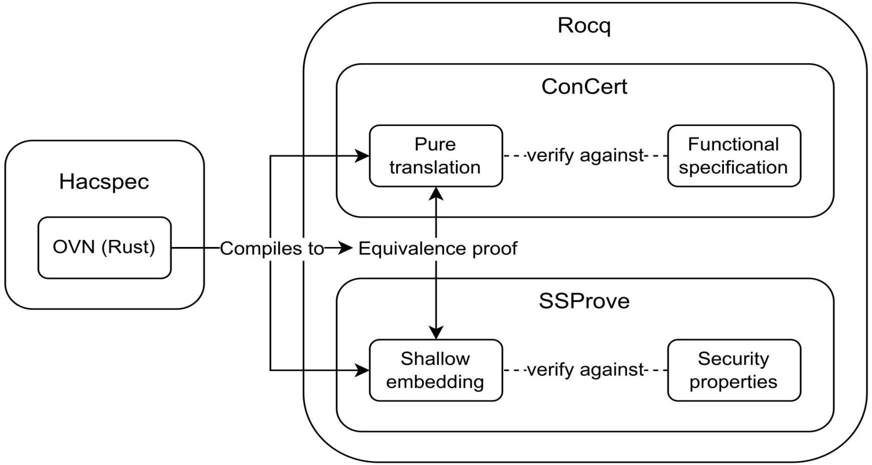
- Voting is used in blockchains for
  - consensus, governance, and decentralized organizations (DAOs).
- Moreover, for blockchains, the adversarial model is complex:
  - The adversary has complete knowledge of the system and full access to the network.
- The stakes are high, both financial and societal.

There can be bugs in

- the specification, the cryptographic proofs and/or the implementation.

Process

- Implement an executable specification of a protocol in safe Rust (Hax)
- Translate it into a proof assistant (Rocq)
- Prove security properties (SSProve)
- Prove functional correctness and trace properties (ConCert)
- Re-extract the code and run it
    - Efficient implementation of secure primitives (libcrux)

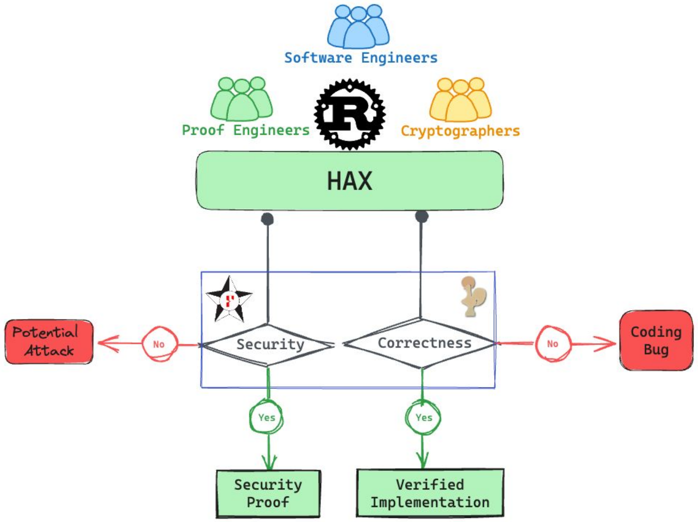


Formalization

- Rocq vs Lean
    - Rocq is the main implementation of dependent type theory
    - Lean is a newer proof assistant, primarily used by mathematicians
    - Goal of Hax: collaboration between proof assistants and tools
- NIST
- Formal methods
- Standardizations

Hax

- a subset of safe Rust with translations to proof assistants
- makes internet standards (e.g. IETF and NIST) machine-readable.
- executable specification in safe Rust
  - efficient implementation when building on the libcrux library of verified cryptographic primitives



SSProve

- a foundational framework for modular cryptographic proofs in Rocq
- formal way of doing State Separating Proofs (SSP)
- a language with monadic state and probability
- a program logic derived from the categorical Dijkstra monad framework
- game hopping style proofs in the computational model
- many useful examples (e.g. the Joy of Crypto)

ConCert

- A smart contract certification framework in Rocq
- Models an abstract account-based blockchain with pure smart contracts
- Verified compilation to a number of targets including WebAssembly (WASM)

Online voting

Most online voting protocols use

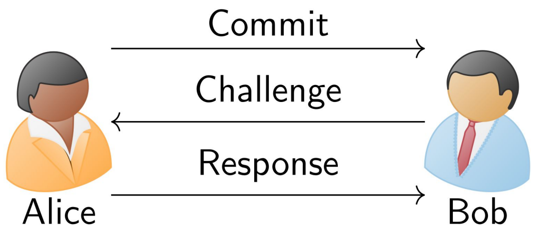
- Commitment schemes
- Zero Knowledge Proofs

The OVN protocol is a small and simple voting protocol and the parts are reusable

A version of OVN runs in production on the Concordium blockchain

Zero-Knowledge Proof (ZKP)

- Schnorr
  - Proof that I know the exponents of an expression, without revealing them
- OR proof
  - Proof that I know one of two statements is correct, without revealing which.
  - e.g. vote is 0 or 1
- These examples are Σ-protocols
  - A three-step protocol



Σ-protocol - Security Properties

- Correctness of protocol
- Special Honest Verifier Zero-Knowledge (SHVZK)
  - A simulator that can construct a transcript given the response and challenge
- Simulation Sound Extractability
  - An extractor that can construct the witness given two valid runs of the same commit

Open Vote Network (OVN)

Protocol (using a group where the Decisional Diffie-Hellman (DDH) problem is hard):

- Round 1 (register vote):
  - Each public key is put on the blockchain and committed to (Schnorr ZKP)
- Round 2 (commit to vote):
  - Verify commitment from round 1
  - Compute commitment to vote
- Round 3 (Cast vote):
  - Build an OR-proof (0 or 1) and cast vote
- Round 4 (tally):
  - Verify the OR-proof and commitments
  - Tally result

Open Vote Network (OVN)

Properties

- Self-tallying: After all ballots have been cast anyone can compute the result
- Maximum ballot secrecy: Each ballot is indistinguishable from random input
- Universal verifiability: Anyone can verify the protocol was done correctly

Security

- Commitment (SSProve)
- Schnorr ZK protocol and the OR-construction (SSProve)
- Functional correctness (ConCert)

Maximum Ballot Secrecy

// Register vote

$x_i \in_R \mathbb{Z}_q$

$Schnorr_{zkp_i} \leftarrow \texttt{Schnorr}(g^{x_i}, x_i)$

$\texttt{Publish} : Schnorr_{zkp_i}, g^{x_i}$

// Commit to vote

$\texttt{validate}(Schnorr_{zkp_j}) \quad \forall j \in (1, n)$

$g^{y_i} \leftarrow \dfrac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{n} g^{x_j}}$

$vote_i := (g^{y_i})^{x_i} \cdot g^{v_i}$

$commit_i := \mathcal{H}(vote_i) : \mathbb{Z}_q$

$\texttt{Publish} : commit_i$

// Cast vote

$OR_i = \texttt{CDS}(g^{y_i}, x_i, v_i)$

$\texttt{Publish} : vote_i, OR_i$

// Tally

$\texttt{CDSvalidate}(g^{y_j}, OR_j) \quad \forall j \in (1, n)$

$\texttt{CheckCommit}(commit_j, vote_j) \quad \forall j \in (1, n)$

$$g^{tally} = \prod_{j=1}^{n} vote_j \overset{bruteforce}{\Longrightarrow} tally$$

// Register vote	// Register vote
$x_i \in_R \mathbb{Z}_q$	
$h \leftarrow DL_{real}(x_i)$	$h \leftarrow DL_{ideal}(\_)$
	$x_i \in_R \mathbb{Z}_q$
$Schnorr_{zkp_i} \leftarrow \texttt{Schnorr}(h, x_i)$	$Schnorr_{zkp_i} \leftarrow \texttt{Schnorr}(h, x_i)$
$\texttt{Publish} : Schnorr_{zkp_i}, h$	$\texttt{Publish} : Schnorr_{zkp_i}, h$
...	...

$x_i \in_R \mathbb{Z}_q$	
// Commit to vote	// Commit to vote
$\texttt{validate}(Schnorr_{zkp_j}) \quad \forall j \in (1, n)$	$\texttt{validate}(Schnorr_{zkp_j}) \quad \forall j \in (1, n)$
$g^{y_i} \leftarrow \frac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{n} g^{x_j}}$	$g^{y_i} \leftarrow \frac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{n} g^{x_j}}$
	$x_i \in_R \mathbb{Z}_q$
$vote_i := (g^{y_i})^{x_i} \cdot g^{v_i}$	$vote_i := (g^{y_i})^{x_i} \cdot g^{v_i}$
$commit_i := \mathcal{H}(vote_i) : \mathbb{Z}_q$	$commit_i := \mathcal{H}(vote_i) : \mathbb{Z}_q$
$\texttt{Publish} : commit_i$	$\texttt{Publish} : commit_i$
...	...

// Commit to vote	// Commit to vote
$\texttt{validate}(Schnorr_{zkp_j}) \quad \forall j \in (1, n)$	$\texttt{validate}(Schnorr_{zkp_j}) \quad \forall j \in (1, n)$
$g^{y_i} \leftarrow \dfrac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{n} g^{x_j}}$	
$x_i \in_R \mathbb{Z}_q$	
$vote_i := DL_{real}(y_i \cdot x_i) \cdot g^{v_i}$	$vote_i := DL_{ideal}(\_) \cdot g^{v_i}$
$commit_i := \mathcal{H}(vote_i) : \mathbb{Z}_q$	$commit_i := \mathcal{H}(vote_i) : \mathbb{Z}_q$
$\texttt{Publish} : commit_i$	$\texttt{Publish} : commit_i$
	$x_i \in_R \mathbb{Z}_q$
...	...

$x_i \in_R \mathbb{Z}_q$	
// Cast vote	// Cast vote
$OR_i = \texttt{CDS}_{real}(g^{y_i}, x_i, v_i)$	$OR_i = \texttt{CDS}_{ideal}(g^{y_i}, x_i, v_i)$
$\texttt{Publish} : vote_i, OR_i$	$\texttt{Publish} : vote_i, OR_i$
	$x_i \in_R \mathbb{Z}_q$
...	...

$x_i \in_R \mathbb{Z}_q$	
// Tally	// Tally
$\texttt{CDSvalidate}(g^{y_j}, OR_j) \quad \forall j \in (1, n)$	$\texttt{CDSvalidate}(g^{y_j}, OR_j) \quad \forall j \in (1, n)$
$\texttt{CheckCommit}(commit_j, vote_j) \quad \forall j \in (1, n)$	$\texttt{CheckCommit}(commit_j, vote_j) \quad \forall j \in (1, n)$
$g^{tally} = \prod_{j=1}^{n} vote_j \overset{bruteforce}{\Longrightarrow} tally$	$g^{tally} = \prod_{j=1}^{n} vote_j \overset{bruteforce}{\Longrightarrow} tally$
	$x_i \in_R \mathbb{Z}_q$

Related work

Verification process

- EasyCrypt: Not foundational

Unmaintained, but part of the inspiration for SSProve:

- CertiCrypt, Foundational Cryptography Framework (FCF), CryptHOL

Symbolic proofs and provers: Present in Hax

- using e.g. Squirrel, Tamarin, ProVerif

Alternative voting protocol: ElectionGuard

- is more off-chain but uses similar building blocks.

Conclusion

- A formalization and implementation of ZK proofs as part of a larger protocol
- First time showing both the correctness and security of a smart contract
- Illustrate possibilities for formal methods as requirements of online voting
- Can be made efficient with Libcrux library of verified crypto primitives