Verifying Security Protocols
End-to-End with Owl

Joshua Gancher
Carnegie Mellon University
Owl

new language for building security protocols

• TLS, WireGuard, Signal, …

• Supports verified implementations
new language for building security protocols

- TLS, WireGuard, Signal, …
- Supports verified implementations

Our vision:
formally verified, drop-in replacements of protocol implementations
Owl

new language for building security protocols

• TLS, WireGuard, Signal, …

• Supports verified implementations

Our vision:
formally verified, drop-in replacements of protocol implementations
Owl

new language for building security protocols

- TLS, WireGuard, Signal, …
- Supports **verified implementations**

Our vision:

**formally verified**, drop-in replacements of protocol implementations

{functional correctness, memory safety, secure designs}
Protocol Description

Owl

Type-based security analysis

✅ / ✗
Protocol Description

Owl

Type-based security analysis

Verified Extraction Pipeline

- functional correctness
- memory safety
- side-channel resistance
- performance

[Checkmark] / [X]
Owl

Protocol Description

Type-based security analysis

Verified Extraction Pipeline

- functional correctness
- memory safety
- side-channel resistance
- performance

fully automated
OWL: Compositional Verification of Security Protocols via an Information-Flow Type System

Joshua Gancher  Sydney Gibson  Pratap Singh  Samvid Dharanikota  Bryan Parno
Carnegie Mellon University

Protocol Description

Type-based security analysis

Verified Extraction Pipeline

- functional correctness
- memory safety
- side-channel resistance
- performance
Symbolic Security

Computational Security
Symbolic Security

Crypto modeled by abstract terms, equations on functions

Computational Security
Symbolic Security

Crypto modeled by abstract terms, equations on functions

functions: enc/2, dec/2
equations: dec(enc(m, k), k) = m
Symbolic Security

Crypto modeled by abstract terms, equations on functions

functions: \( \text{enc}/2, \text{dec}/2 \)
equations: \( \text{dec}(\text{enc}(m, k), k) = m \)

Computational Security

attacker can only use specified functions, equations
Symbolic Security

Crypto modeled by abstract terms, equations on functions

functions: $\text{enc}/2$, $\text{dec}/2$
equations: $\text{dec}(\text{enc}(m, k), k) = m$

attacker can only use specified functions, equations

Computational Security

Crypto is given by algorithms on bytes

Cryptography specified by security properties:
secrecy of messages, unforgeability of ciphertexts,
...
Symbolic Security

Crypto modeled by abstract terms, equations on functions

functions: \( \text{enc}/2, \text{dec}/2 \)
equations: \( \text{dec} (\text{enc}(m, k), k) = m \)

attacker can only use specified functions, equations

Computational Security

Crypto is given by algorithms on bytes

Cryptography specified by security properties:
secrecy of messages, unforgeability of ciphertexts, ...

Strong attacker model; closer to implementations
Types for Security Protocols
Types for Security Protocols

\[ K : \text{enckey for } T \quad m : T \quad \implies \quad \text{enc}(K, m) : \text{public} \]
Types for Security Protocols

\[ K : \text{enckey for } T \quad \iff \quad m : T \Rightarrow \text{enc}(K, m) : \text{public} \]

security via type checking: \( \vdash P \implies P \text{ secure} \)
Types for Security Protocols

\[ K : \text{enckey for } T \]
\[ m : T \quad \implies \quad \text{enc}(K, m) : \text{public} \]

security via type checking:

\[ \vdash P \implies P \text{ secure} \]

one time proof effort
Types for Security Protocols

\[ K : \text{enckey for } T \quad \Rightarrow \quad \text{enc}(K, m) : \text{public} \]

security via type checking:

\[ \vdash P \quad \Rightarrow \quad P \text{ secure} \]

one time proof effort

developer only

uses type system!
Types for Security Protocols

\[ K : \text{enckey for } T \quad \iff \quad \text{enc}(K, m) : \text{public} \]

security via type checking: \[ \vdash P \iff P \text{ secure} \]

one time proof effort

developer only uses type system!

automatic, modular proof effort

6
Protocol-Level Modularity

Handshake +

Data Transfer
Protocol-Level Modularity

(only specs)

Handshake

uses

Data Transfer
Protocol-Level Modularity

Data Transfer

Handshake

(only specs)

uses

Handshake w/ PKI

Handshake w/ Pre-Shared Key

instantiates
Owl

Type-based security analysis

Protocol Description

Verified Extraction Pipeline

- memory safety
- functional correctness
- side-channel resistance
- performance

9
Protocol Description

Owl

Type-based security analysis

Verified Extraction Pipeline

- memory safety
- functional correctness
- side-channel resistance
- performance

Ongoing work!
Verus: Verifying Rust Programs using Linear Ghost Types
Verus: Verifying Rust Programs using Linear Ghost Types

Verus Extraction

functional specification

executable Rust implementation
Verus: Verifying Rust Programs using Linear Ghost Types

functional specification

(Almost) directly equal to input Owl code

Verus Extraction

Executable Rust implementation
Verus: Verifying Rust Programs using Linear Ghost Types

Verus Extraction

- Functional specification

  (almost) directly equal to input Owl code

- Gory low-level details
  verified parser library,
  zero-copy ciphers,

- Executable Rust implementation

10
Verus: Verifying Rust Programs using Linear Ghost Types

Verus Extraction

- functional specification
- autogenerated proof script

Executable Rust implementation

- memory safety
- functional equivalence
- side-channel resistance
Ongoing Work: a verified VPN
Ongoing Work: a verified VPN

widely used:
inside Linux kernel
Ongoing Work: a verified VPN

widely used: inside Linux kernel

very lean: implementable in 4K LoC
Ongoing Work: a verified VPN

widely used:
inside Linux kernel

very lean:
implementable in 4K LoC

Goal: verified, drop-in replacement
Owl:
End-to-End Verification of Security Protocols via a Secure Type System

New tool for modular, automated proofs of security protocols

• Novel use of type systems for constructing secure cryptographic protocols
  • Security is proved once-and-for-all;
  • Protocols checked via type checker
• Ongoing work: verified extraction and drop-in implementation of WireGuard

owl-lang.org
jgancher@andrew.cmu.edu