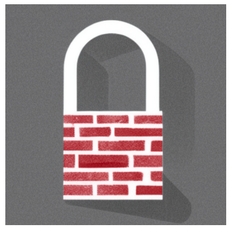


Carnegie Mellon University
Security and Privacy Institute
CyLab

Northeastern University

OwlC: Compiling Security Protocols to Verified, Secure, High-Performance Libraries

Pratap Singh
CMU

Joshua Gancher
Northeastern

Bryan Parno
CMU

Crypto Protocols Break

ALPACA —

Hackers can mess with HTTPS connections by sending data to your email server

Cross-protocol attacks could potentially steal login cookies or execute malicious code.

DAN GOODIN - 6/9/2021, 8:00 AM

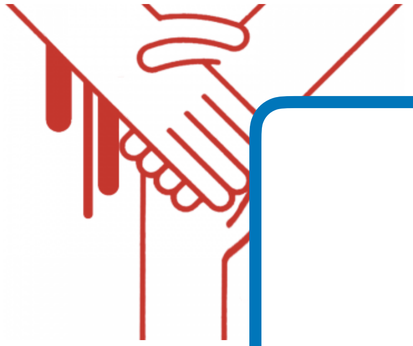
SSL/TLS data compression l
cookies, researchers say

Last Revised: September 30, 2016

Alert Code: TA14-290A

caused by US companies being forced to sell weakened encryption software to overseas customers

Crypto Protocols Break



Crypto protocols are only as secure as the code that runs them!

ALPACA —

Hackers sending

Cross-protocol at

DAN GOODIN - 6/9/2021,

SSL/TLS data compr

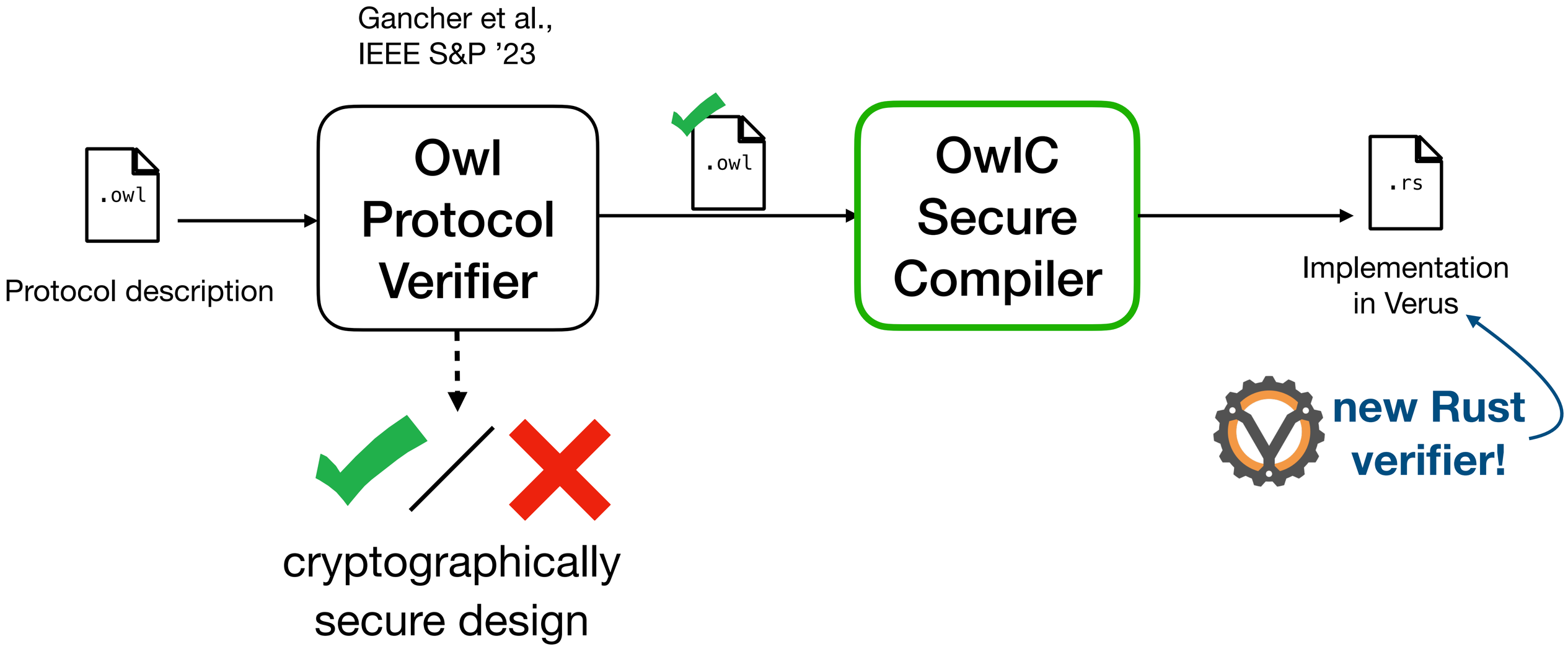
cookies, researchers say

software to overseas customers

Last Revised: September 30, 2016 **Alert Code:** TA14-290A

OwlC:

- security-preserving compiler for protocols
- guarantees no **explicit** or **implicit** information leakage
- novel mechanism for controlling effects in code
- generates **performant, interoperable** implementations of **realistic protocols**


Gancher et al.,
IEEE S&P '23
.owl
Protocol description
Owl
Protocol
Verifier
.owl
OwlC
Secure
Compiler
.rs
Implementation
in Verus
new Rust
verifier!
cryptographically
secure design

Example Protocol in Owl

```
def client_main () @ Client : Unit =
    let i = input in
    case dec(get(psk), i) {
      | Some k_x =>
        let c = enc(k_x, get(x)) in
        output c
      | None => ()
    }
```

Compiling Owl to Rust

```
def client_main () @ Client : Unit =
    let i = input in
    case dec(get(psk), i) {
     | Some k_x =>
        let c = enc(k_x, get(x)) in

        output c

     | None => ()
    }
```

Owl

```
fn client_main(cfg: ClientCfg) {
  let i = input();
  match dec(cfg.psk, i) {
    Some(k_x) => {
      let mut c = [0; 32];
      enc(k_x, cfg.x, &mut c);
      output(c)
    },
    None => ()
  }
}
```

Rust

Problem: Explicit Leakage

```
def client_main () @ Client : Unit =
    let i = input in
    case dec(get(psk), i) {
     | Some k_x =>
        let c = enc(k_x, get(x)) in

        output c

     | None => ()
    }
```

Owl

```
fn client_main(cfg: ClientCfg) {
  let i = input();
  match dec(cfg.psk, i) {
    Some(k_x) => {
      let mut c = [0; 32];
      enc(k_x, cfg.x, &mut c);
      output(k_x);
      output(c)
    },
    None => ()
  }
}
```

Rust

Problem: Explicit Leakage

```
fn client_main(cfg: ClientCfg) {
  let i = input();
  match dec(cfg.psk, i) {
    Some(k_x) => {
      let mut c = [0; 32];
      enc(k_x, cfg.x, &mut c);
      output(k_x);
      output(c)
    },
    None => ()
  }
}
```

Rust

1. How to specify the protocol's input-output effects in Verus?
2. How to restrict the compiled code's input-output effects?

Solution 1: Interaction Trees

```
enum ITree<A> {
    Ret    (A),
    Input  (bytes -> ITree<A>),
    Output (bytes, ITree<A>),
    Sample (nat, bytes -> ITree<A>),
}
```

“an effectful computation returning values of type A”

[Xia et al, POPL ’20]

Solution 1: Interaction Trees

```
enum ITree<A> {
    Ret    (A),
    Input  (bytes -> ITree<A>),
    Output (bytes, ITree<A>),
    Sample (nat, bytes -> ITree<A>),
}
```

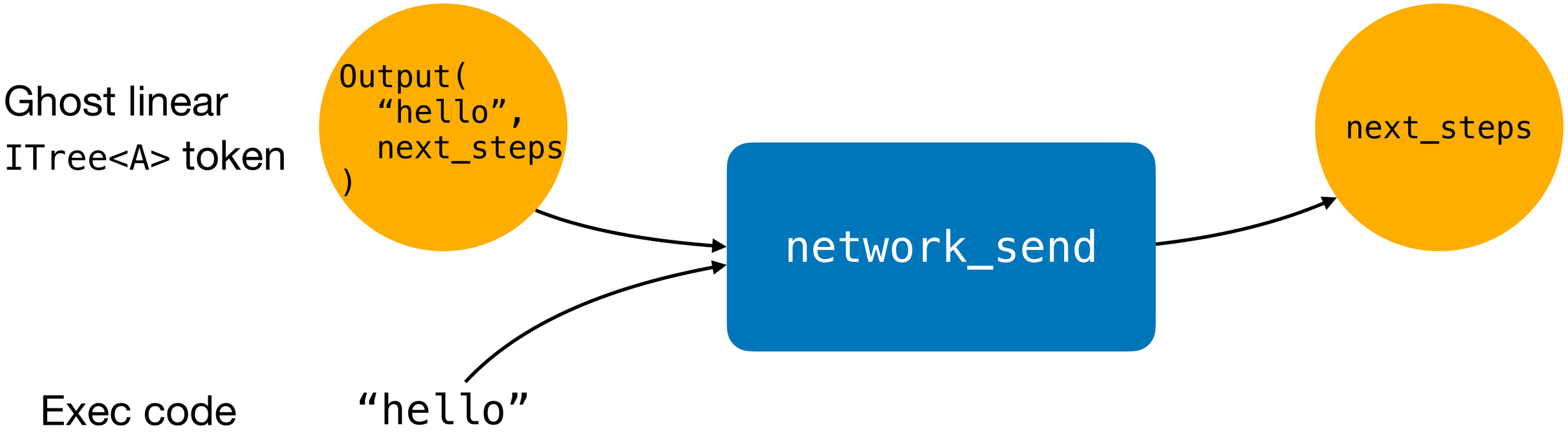
```
let i = input in
case dec(get(psk), i)
 | Some k_x =>
    let c = enc(k_x, get(x)) in
    output c to Server
 | None => ()
```

```
Input(λi.
  match dec(cfg.psk, i) {
    Some(k_x) =>
      Output(enc(k_x, cfg.x),
        Ret(())
      ),
    None => Ret(())
  }
)
```

Solution 2:
Ghost Linear Permission Tokens

Solution 2:
Ghost Linear Permission Tokens

```
enum ITree<A> {
    Ret    (A),
    Input  (bytes -> ITree<A>),
    Output (bytes, ITree<A>),
    Sample (nat, bytes -> ITree<A>),
}
```

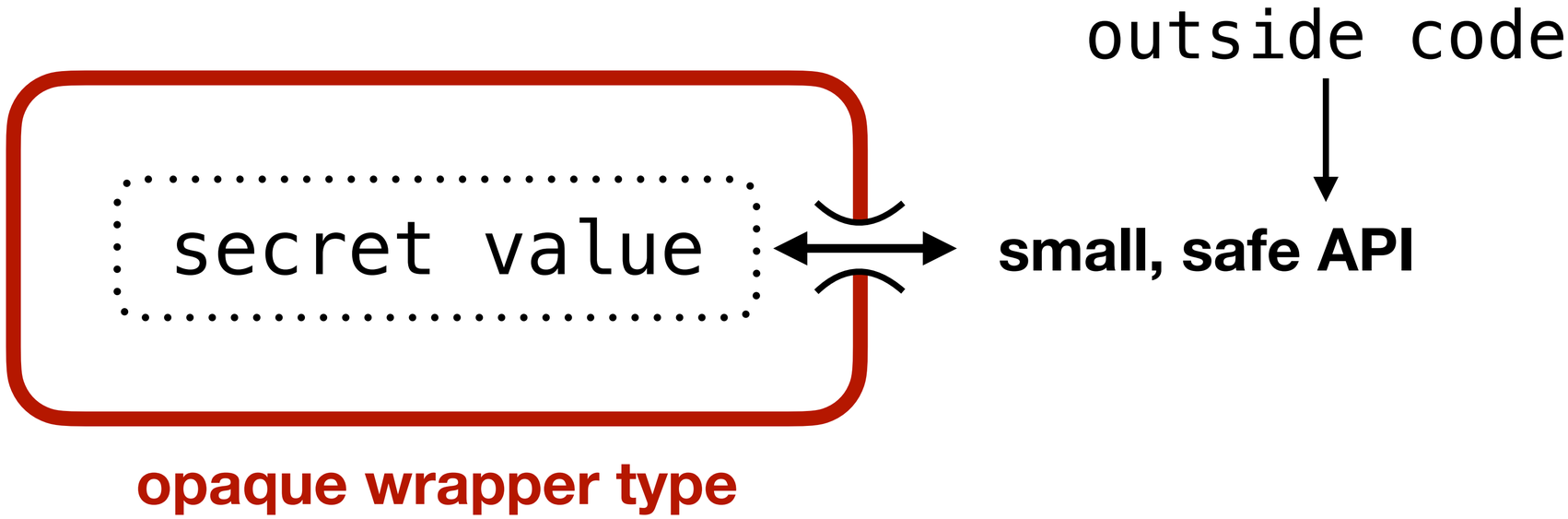


Problem: Implicit Leakage

```
fn client_main(cfg: ClientCfg) {
  let i = input();
  match dec(cfg.psk, i) {
    Some(k_x) => {
      if k_x[0] == 0 { sleep(10); }
      let mut c = [0; 32];
      enc(k_x, cfg.x, &mut c);
      output(c)
    },
    None => ()
  }
}
```

Preventing Implicit Leakage: Type Abstraction

Type abstraction: **secret data can’t influence control flow**



Type Abstraction is Not Enough!

```
def client_main () @ Client : Unit =
    let i = input in
    case dec(get(psk), i) {
      | Some k_x =>
          let c = enc(k_x, get(x)) in
          output c
      | None => ()
    }
```

declassification

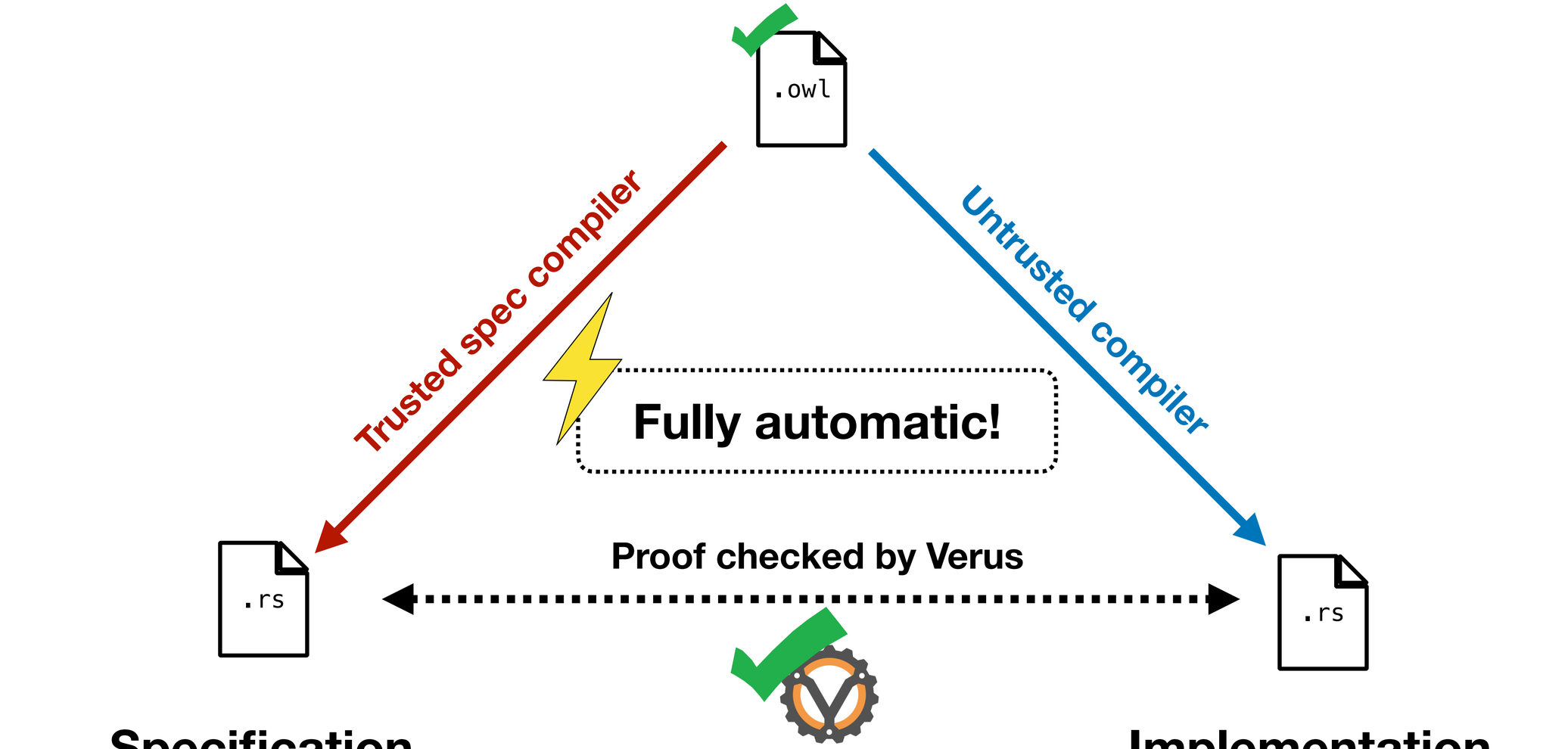
Solution: Declassification as an Effect

Owl's cryptographic analysis tells us which values are safe to declassify!

```
enum ITree<A> {
    Ret     (A),
    Input   (bytes -> ITree<A>),
    Output  (bytes, ITree<A>),
    Sample  (nat, bytes -> ITree<A>),
    Declassify (value, ITree<A>),
}
```

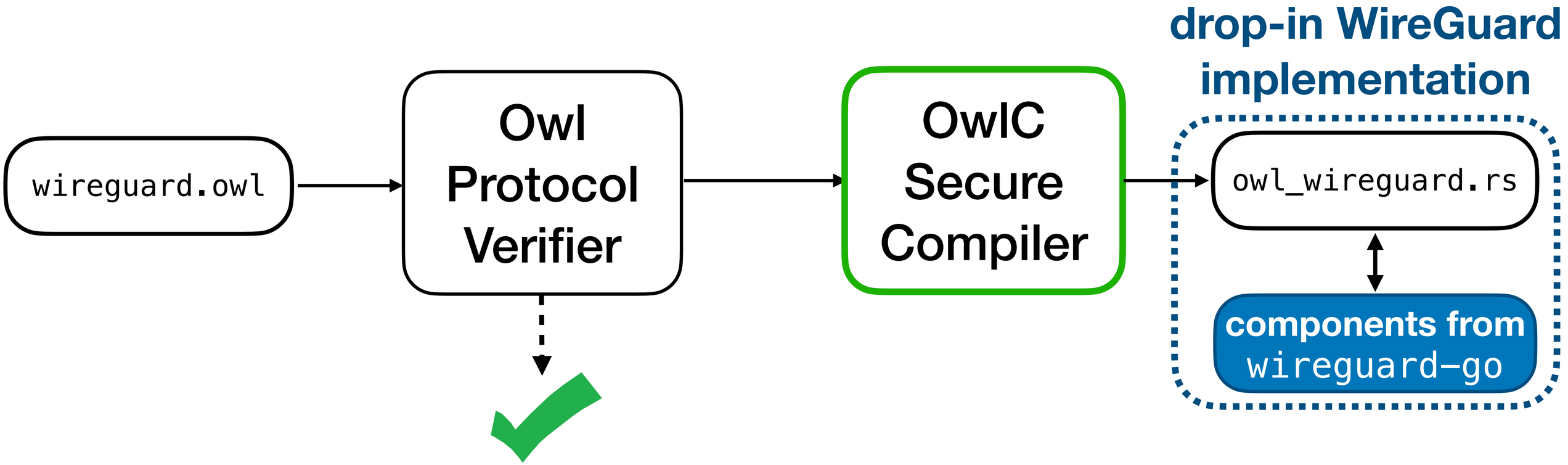
permission to reveal the value

OwlC Compiler Pipeline



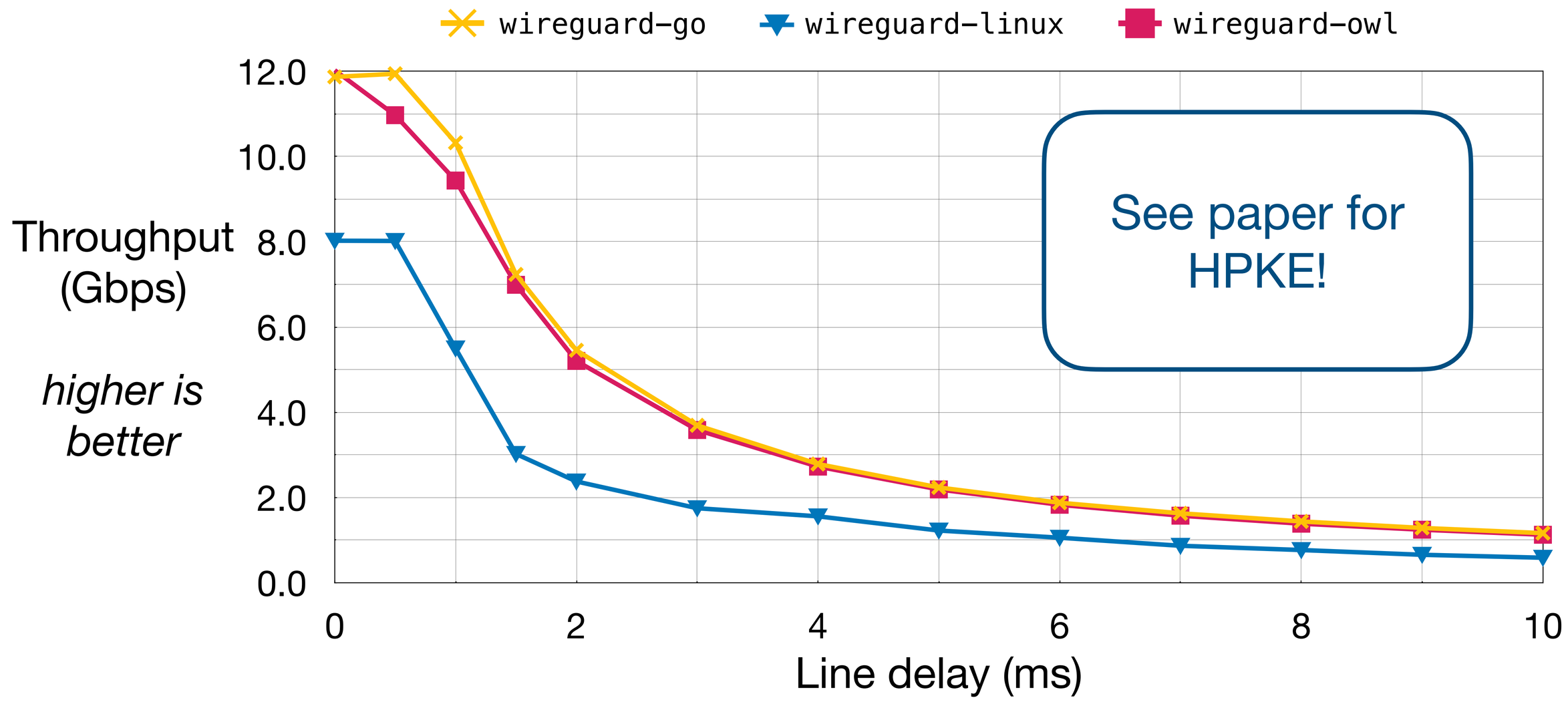
Case study: WireGuard

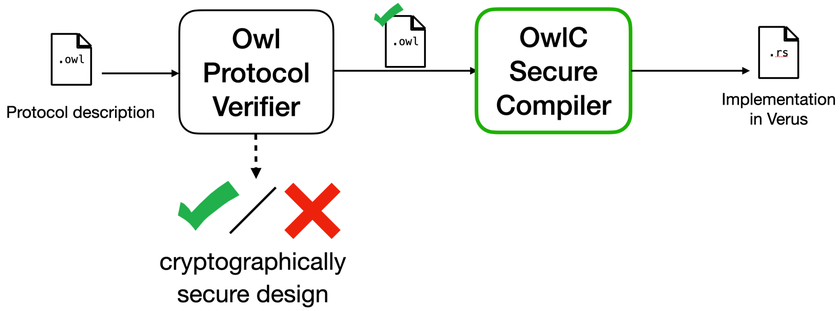
FAST, MODERN, SECURE VPN TUNNEL

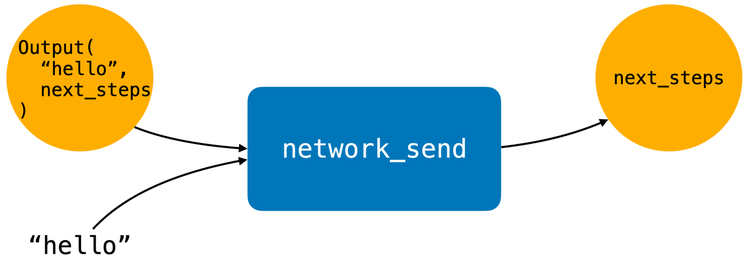


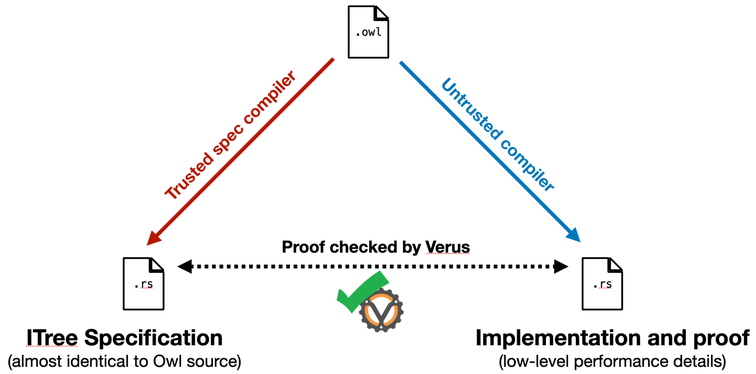
Case study: WireGuard

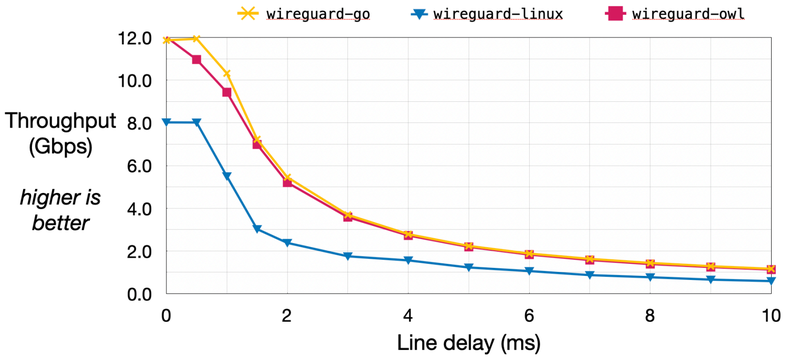
FAST, MODERN, SECURE VPN TUNNEL











OwlC:

Security-preserving compiler for protocols

Prevents explicit leakage via ITrees & tokens

Prevents implicit leakage via controlled declassification

Performant, interoperable case studies

pratapsingh@cmu.edu

owl-lang.org