

Verified Secure Routing

Peter Müller

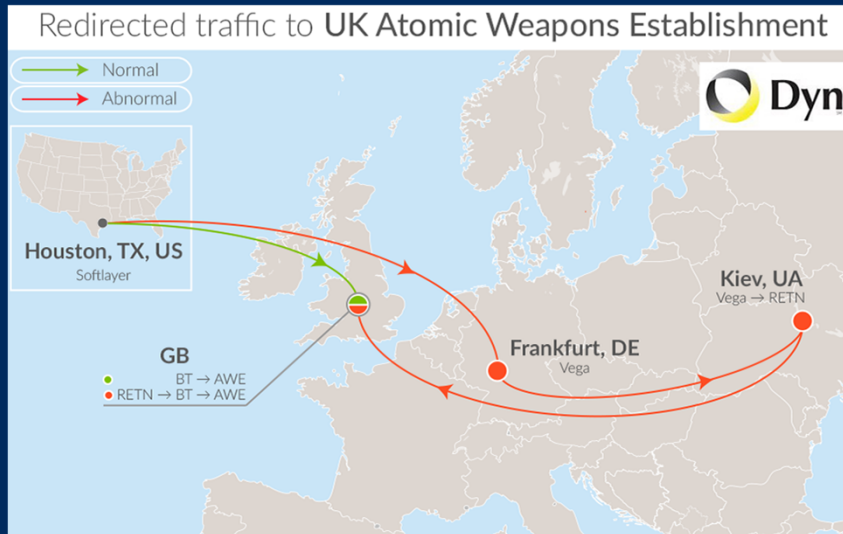
Joint work with the Verified Scion Team at ETH

ETH zürich



- Internet is a network of Autonomous Systems (AS)
- Each AS is a network of routers run by an institution
- Routes between AS are discovered using Border Gateway Protocol (BGP)
- Based on trust, for instance, any AS can announce any IP address range

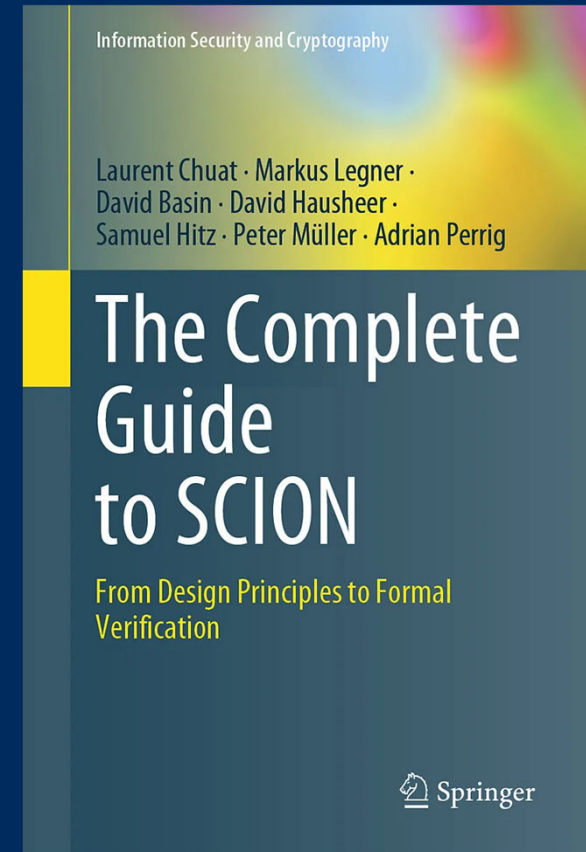
- There are numerous ways to attack Internet routing



- In 2013, Ukrainian ISP announced route prefixes to British Telecom AS
 - Traffic of some UK customers was redirected to Ukraine, including UK's Atomic Weapons Establishment
- Senders have no control over the taken routes
 - Routers on path can read and modify data

Scion Internet Architecture

- Scion is a new architecture for inter-domain routing
 - Path control, e.g., geofencing
 - Multipath communication
 - DDoS protection
- Research and commercial deployments

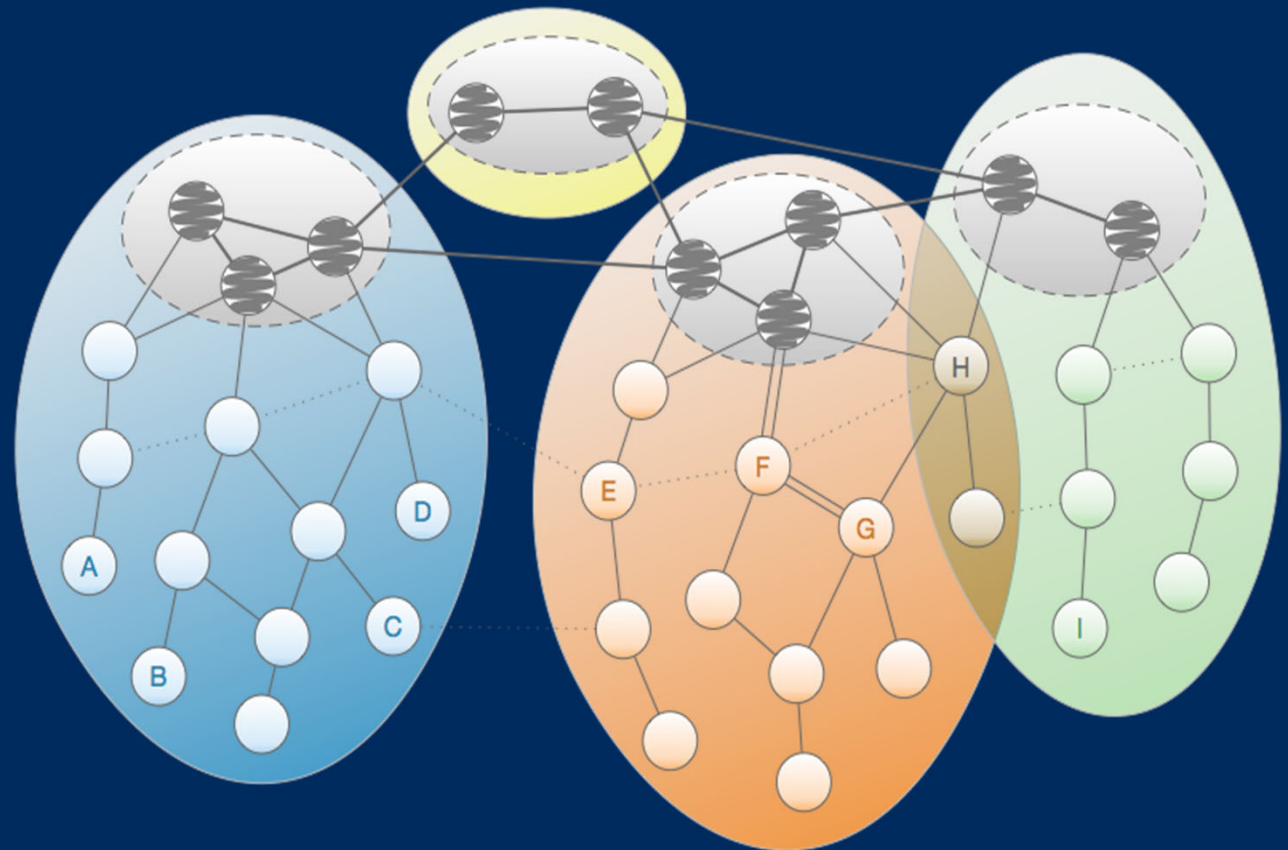




**Formal end-to-end verification
of security and correctness**

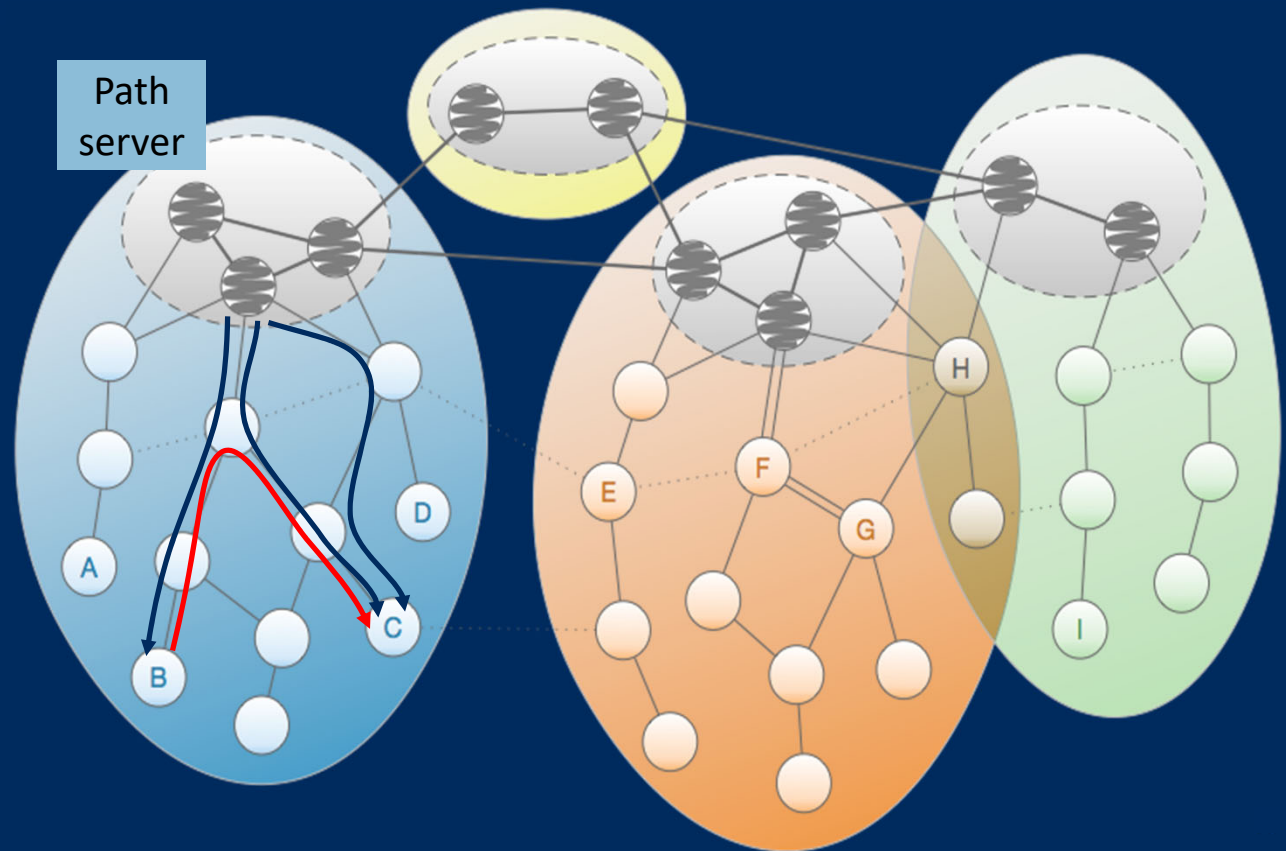
Isolation Domains

ASes are organized into isolation domains with independent control planes and root of trust



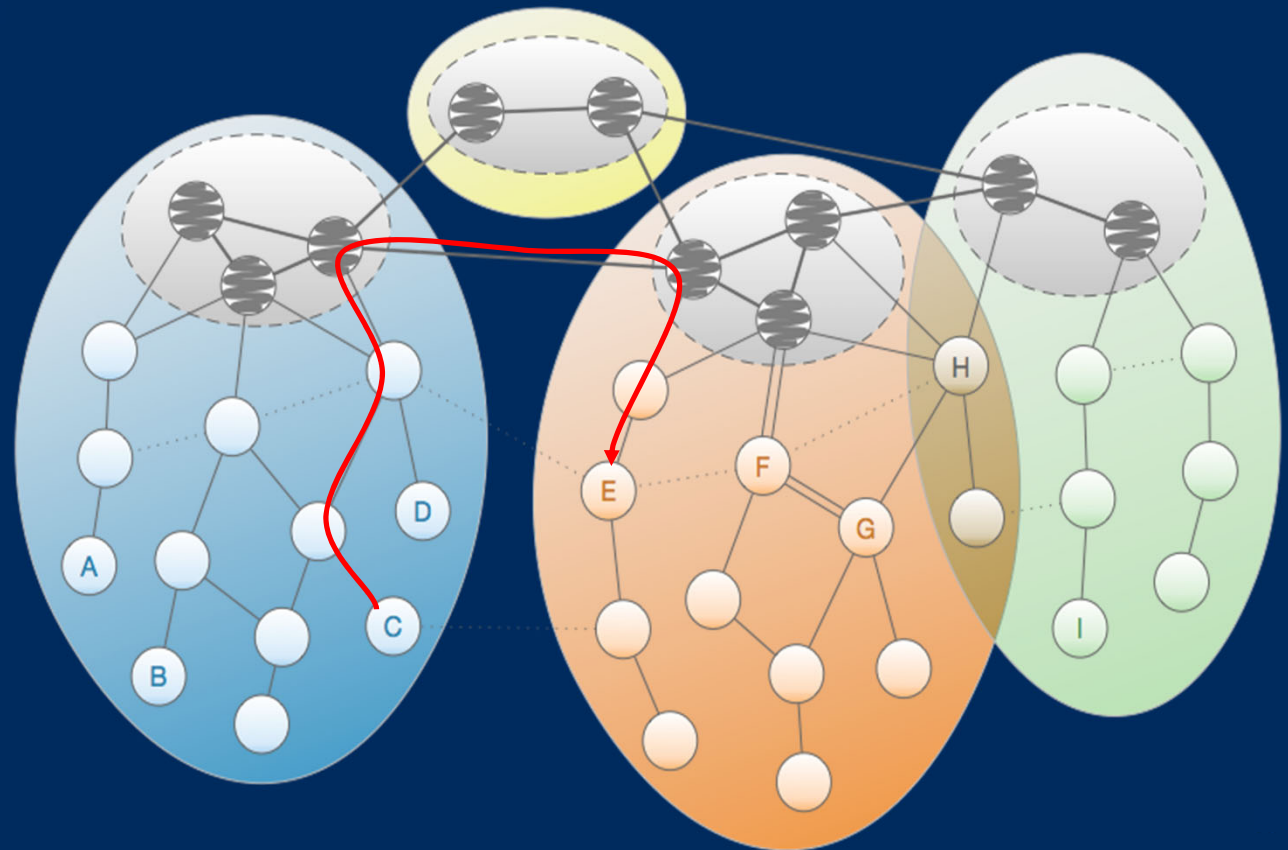
Scion Routing

- Path exploration
 - Paths are sequences of signed hop fields
 - Each hop field carries routing information for one AS (input and output ports)
- Path registration with path server
- Path selection
 - Path is stored in packet header



Scion Forwarding

- Path is stored in packet header
- Consisting of up segment, core segment, and down segment



Security and Correctness

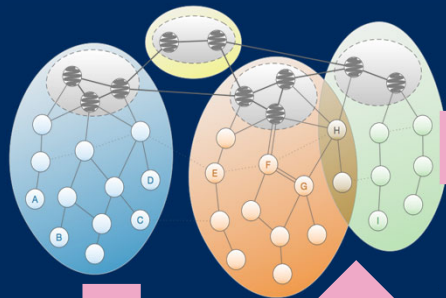
■ Protocol-level properties

- **Path validity:** Constructed paths are valid and reflect the routing decisions by on-path ASes.
- **Path authorization:** Packets travel only along previously authorized paths
- **Detectability:** An active attacker cannot hide their presence on the path

■ Code-level properties

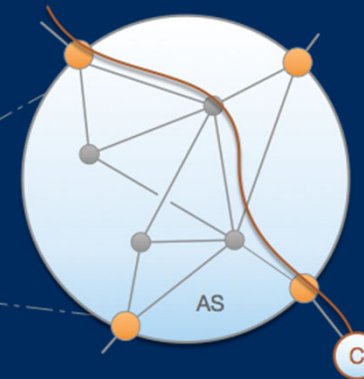
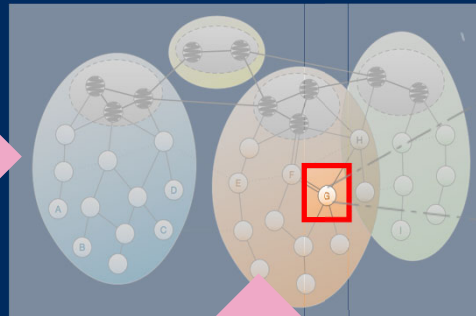
- **Safety:** No run-time errors
- **Correctness:** Routers and servers implement protocol correctly
- **Progress:** Required I/O happens eventually
- **Secure information flow:** Code does not leak information about crypto keys

Mathematical model of entire network



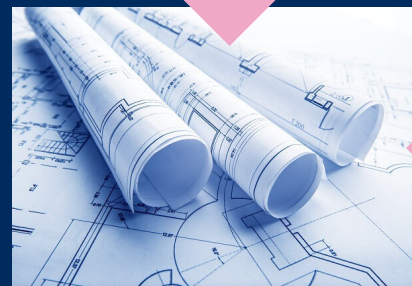
Refinement

Mathematical model of border router



Equivalence

Refinement



Router specification

Verification



Router implementation

Design Model

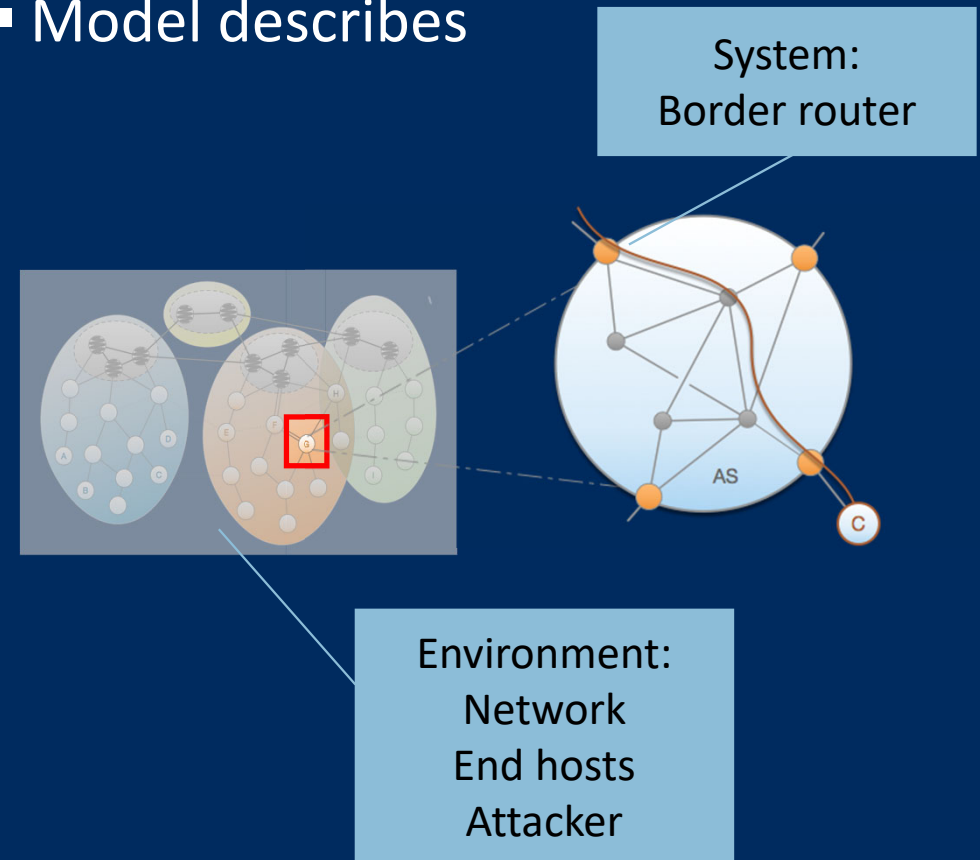
- Formalize the design model as transition system

```
var in: Multiset<Message>
var out: Multiset<Message>

initially
  in = { }
  out = { }

event process(M, M')
  guard M ∈ in ∧ valid(M) ∧
        reply(M, M')
  action
    in := in \ { M }
    out := out ∪ { M' }
```

- Model describes



Stepwise Refinement

- Design model is developed by stepwise refinement
- Prove properties of most abstract model
- Each refinement
 - Incorporates additional system requirements or environment assumptions
 - Preserves properties of more-abstract system
 - Is tool-checked in Isabelle
- Strategy: strengthen attacker while increasing protection of paths

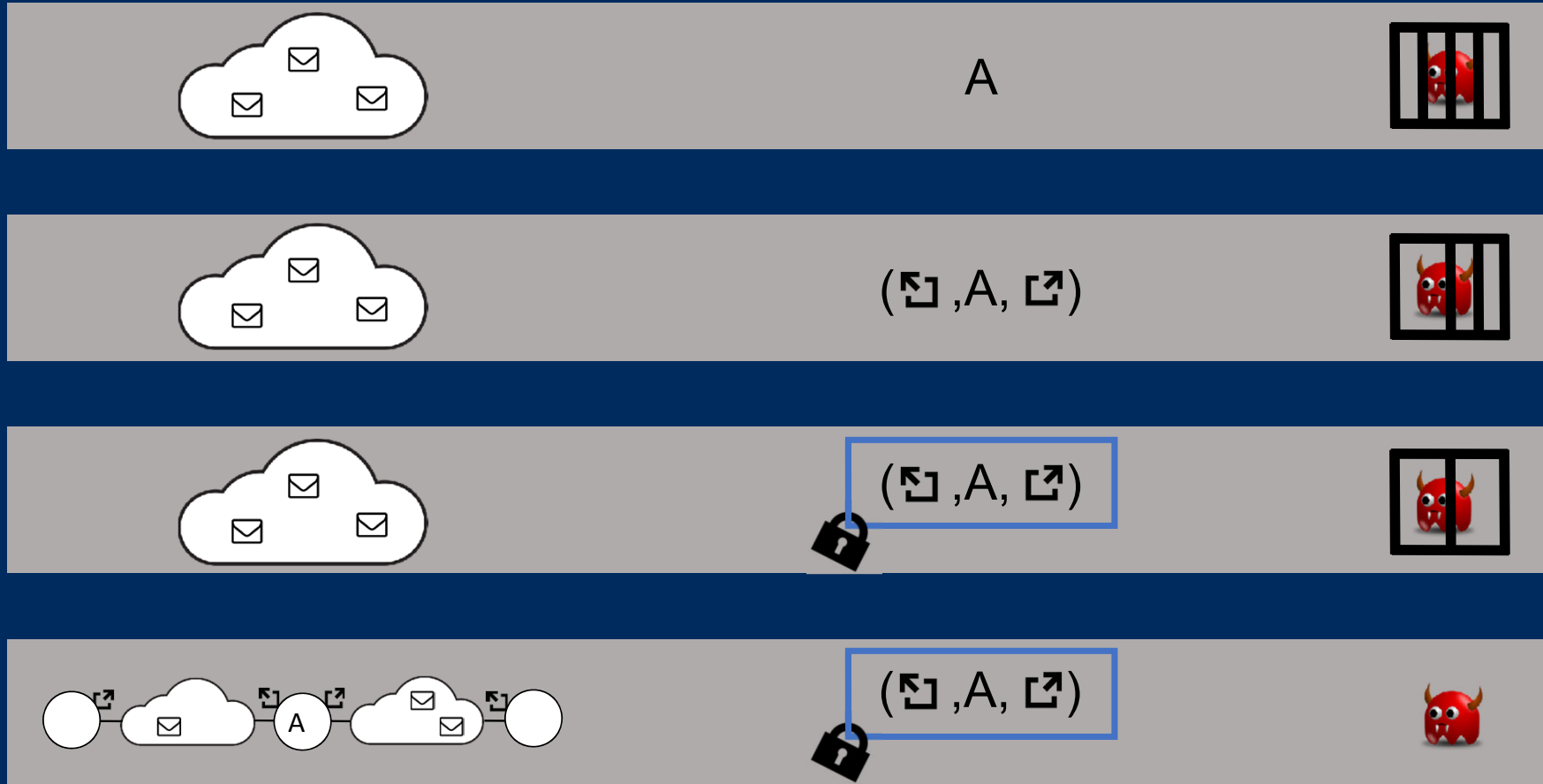


Communication channels

Hop field format

Attacker

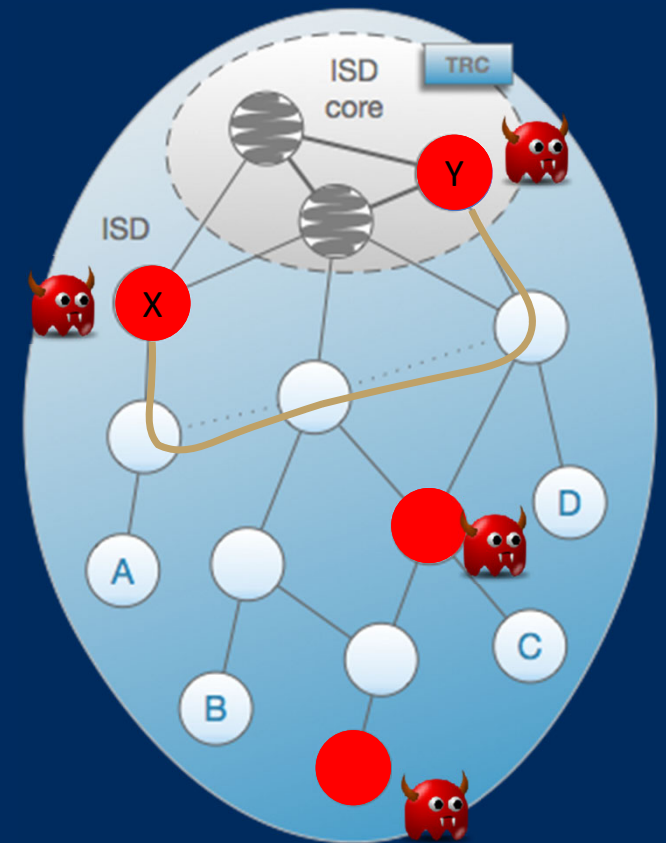
Refinement



: Message set Neighbor Asets : Fields protected by MAC

Attacker Model

- Localized, colluding Dolev-Yao attacker model
- Attacker:
 - Actively controls some ASes
 - Can observe, block, and inject messages
 - Can eavesdrop globally
- Cryptography is assumed to be perfect

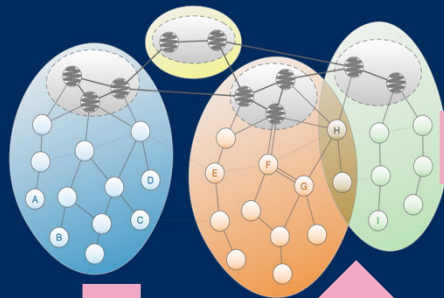


Results of Protocol Verification

- A formal model of the network components and their environment
- Model serves as formal specification for the implementation
- Proofs of the desired properties under the assumption that each component satisfies its specification
- 16,100 lines (models and proofs)

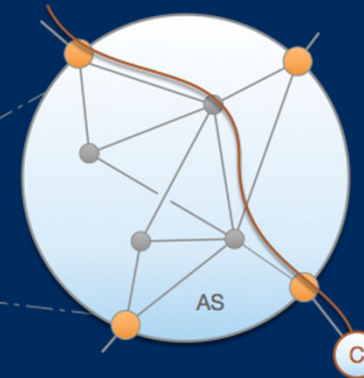
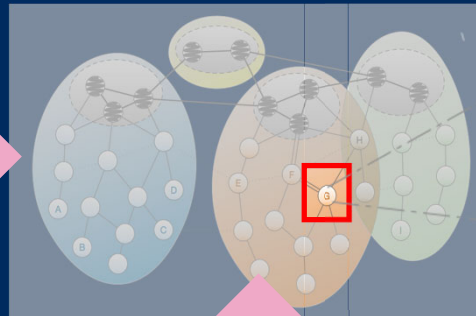
- Improved understanding of protocols and properties
- Revealed design flaws that enabled five different security attacks
- Issues were found during modeling and formalization

Mathematical model of entire network



Refinement

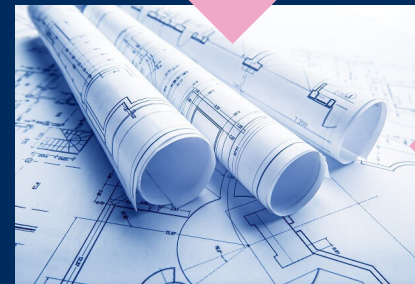
Mathematical model of border router



Equivalence

Refinement

Verification



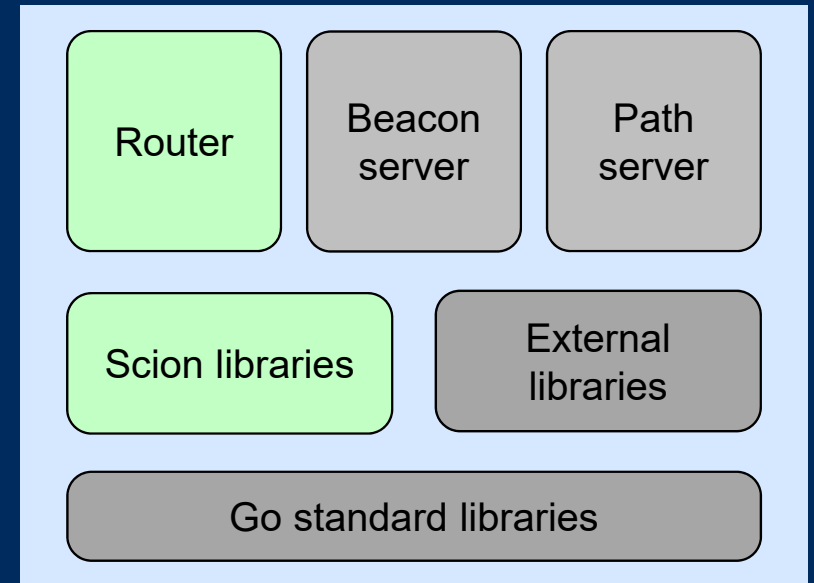
Router specification



Router implementation

Scion Implementation

- Open-source implementation
 - 35kloc of Go (Router: 4.7kloc)
 - Uses concurrency, async, globals
- Verify safety, functional correctness, progress, secure information flow
- Assume correctness of external libraries, Go compiler, OS, hardware



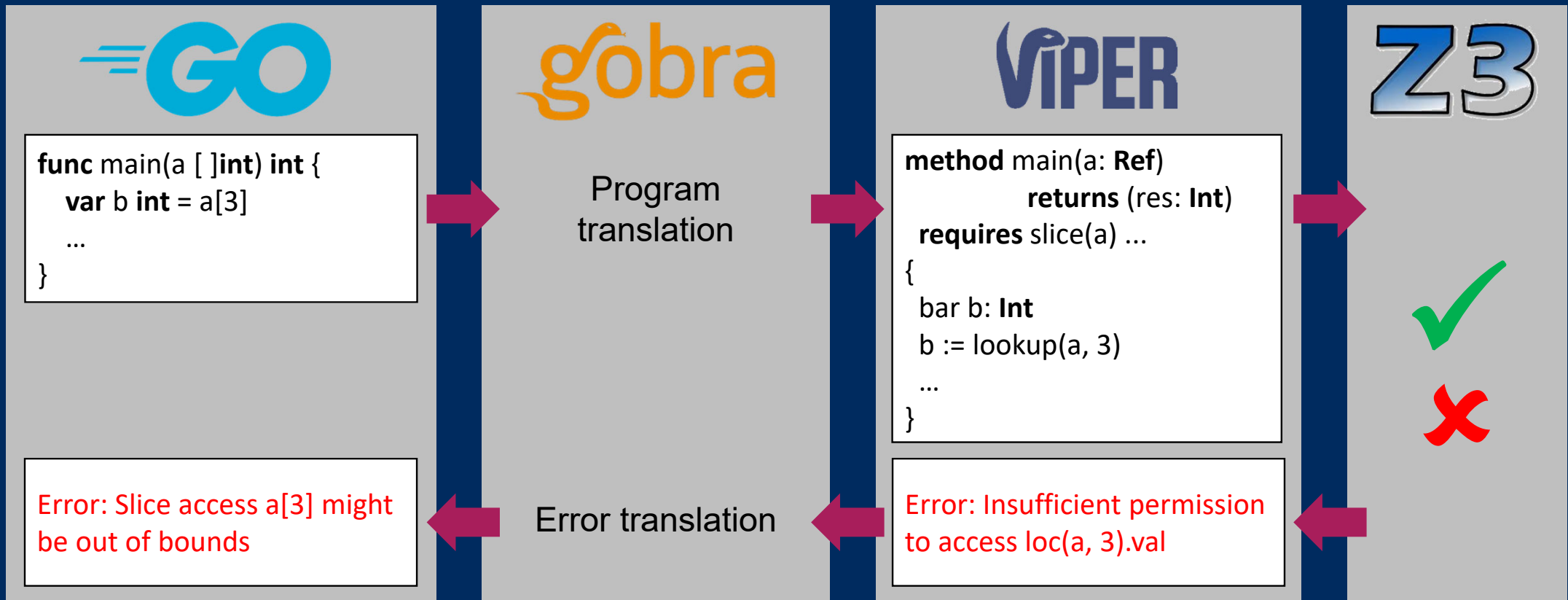
Gobra: Sound Verification for Go

```
func indexOf(l []int, i, val int) (res int)
requires 0 <= i && i < len(l)
requires forall j int :: i <= j && j < len(l) ==> acc(&l[j])
decreases len(l) - i

ensures forall j int :: i <= j && j < len(l) ==> acc(&l[j])
ensures res != -1 ==> i <= res && res < len(l) &&
  (forall j int :: i <= j && j < res ==> l[j] != val) && l[res] == val
{
  if l[i] == val           { return i }
  else if i >= len(l) - 1 { return -1 }
  else                     { return indexOf(l, i+1, val) }
}
```

- No run-time errors
- No data races
- Functional properties
- Termination
- I/O behavior
- Secure information flow

Gobra Toolchain



Permissions

- Associate each heap location with a permission
- Permissions are held by method executions
- Access to a memory location requires permission

```
func indexOf(l [ ]int, i, val int) (res int)  
  requires forall j int :: i <= j && j < len(l) ==> acc(&l[ j ])  
  ensures forall j int :: i <= j && j < len(l) ==> acc(&l[ j ])
```

- Permissions can be transferred, but not duplicated or forged
- Predicates abstract over concrete permissions

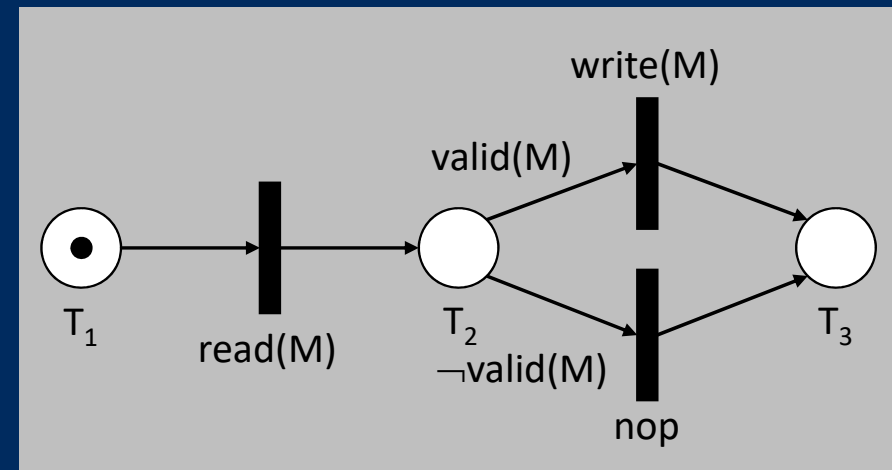
I/O Permissions

- Permissions and predicates can be used to reason about resources
- Here: permission to perform an I/O operation

```
func write(value string)  
  requires writeIO(value)
```

I/O Behavior as Petri Nets

- Adaptation of work by Penninckx et al.
- Petri nets specify permitted I/O behavior
 - Traces of basic I/O operations
 - Sequences, parallelism, non-determinism
- Petri nets are encoded as (recursive) predicates



```
predicate router( $T_1$ ) {  
   $\forall M \exists T_2, T_3 \bullet$   
  readIO( $T_1, M, T_2$ ) *  
  ( $valid(M) \Rightarrow writeIO(T_2, M, T_3)$ ) *  
  ( $\neg valid(M) \Rightarrow nop(T_2, T_3)$ ) *  
  router( $T_3$ )  
}
```

Specification of I/O Behavior

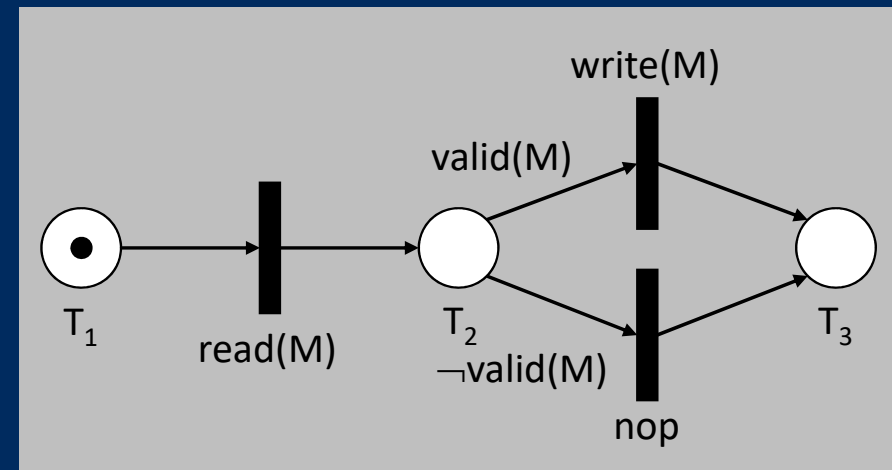
Basic I/O operations

- Require I/O permission
- Require token in appropriate place
- Advance token

```
func write(value string)
  requires token(T) * writeIO(T, value, T')
  ensures token(T')
```

Method precondition characterizes permitted I/O behavior

```
func main()
  requires token(T) * router(T)
```



```
predicate router(T1) {
  ∀M ∃T2, T3 •
    readIO(T1, M, T2) *
    (valid(M) ⇒ writeIO(T2, M, T3)) *
    (¬ valid(M) ⇒ nop(T2, T3)) *
    router(T3)
}
```

From Design Model to I/O Specification

- Refine design model to have an event for each basic I/O operation

```
I/O event write(val)
```

```
func write(value string)
```

- Encode entire event system as recursive predicate

```
event drop(M)  
guard ¬valid(M)  
action  
  buf := buf \ { M }
```

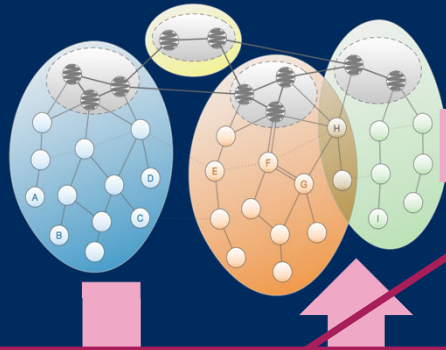
```
predicate system(T1, state) {  
  (∀args • guard(args, state) ⇒  
    ∃T2 • opIO(T1, args, T2) * system(T2, state')) *  
  ...
```

```
predicate router(T1, buf) {  
  (∀M • ¬valid(M) ⇒ router(T1, buf \ { M }))) *  
  ...
```


Status of Code Verification

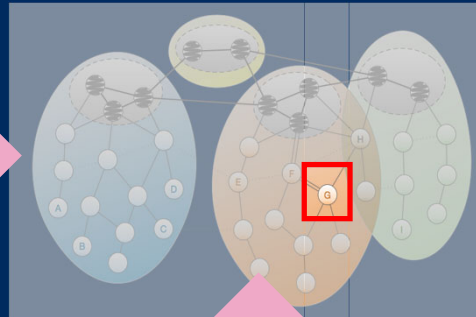
- Completed verification of SCION router (4,700 LoC)
 - Memory safety
 - Functional correctness
 - I/O behavior
 - Termination
- 13,400 lines of annotations (2.8 LoS per LoC)
- Required only three code changes
- Identified 13 confirmed issues related to memory safety, functional correctness, and I/O behavior (plus 2 performance issues)
- Despite extensive code reviews, testing, and fuzzing
- Verification is currently being introduced into the continuous integration

Mathematical model of entire network



Refinement

Mathematical model of border router

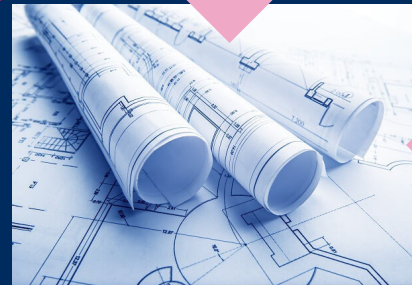


Equivalence

- Developed by stepwise refinement
- Verified in Isabelle

- Properties encoded in permission logic
- Verified in Gobra/Viper

- Transition system encoded as I/O specification
- Verified in Isabelle



Router specification

Verification



Router implementation