Don’t trust the Internet; Verify it!

Pierre Louis Aublin
2026 年 8 月 4 日

What is Formal Verification?

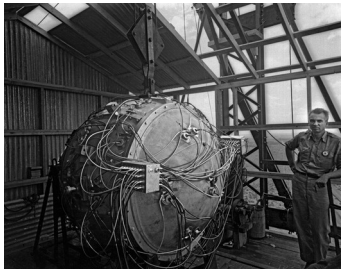
- **Mathematical proof** that a system **satisfies** its **specification**
- Different flavours
  - Dynamic (testing at runtime)
  - Static
- Different approaches
  - Model checking
  - Deductive verification
  - ...

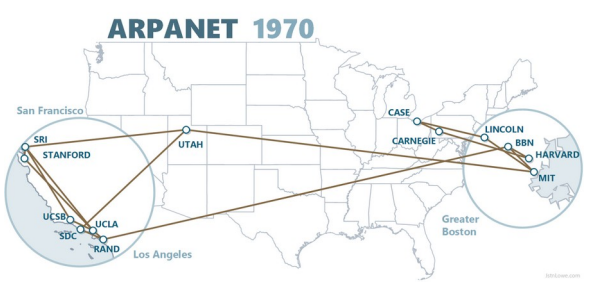
Testing: **hope** to find bugs

Formal Verification: **prove** there are no bugs

How we think of Formal Verification

From Specialized Fields to the General Public





Why Verify the Internet

- **Correctness**
  - Check for specification compliance and protocol errors
- **Secu**
  -
  -
- **Relia**
  - Avoid bugs or misconfigurations
- **Interoperability**
  - Ease standardization and development of various implementations

Formal Verification Increases Trust in the Infrastructure

Why Verify now?

- Protocols become increasingly complex and difficult to maintain
- Attacks increase both in quantity and sophistication

Internet Outages vs Formal Verification

AWS, October 2025

- Impact on critical services: DynamoDB, EC2, Lambda…
- Day-long disruption for Amazon, Disney+, Reddit, Signal, Zoom…
- Root cause: DNS race condition
- **Formal Verification** would have prevented this issue when **designing** the system

Cloudflare, November 2025

- ~3h global outage
- CloudFlare handles ~20% of web traffic
- Root cause: panic in Rust code
- **Formal Verification** would have avoided the situation while **writing** the code

Formal Verification for QUIC

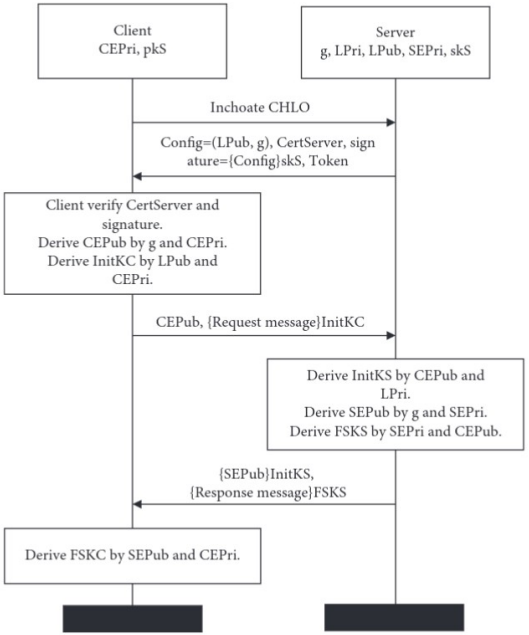


FIGURE 3: The detailed steps of the QUIC handshake process.

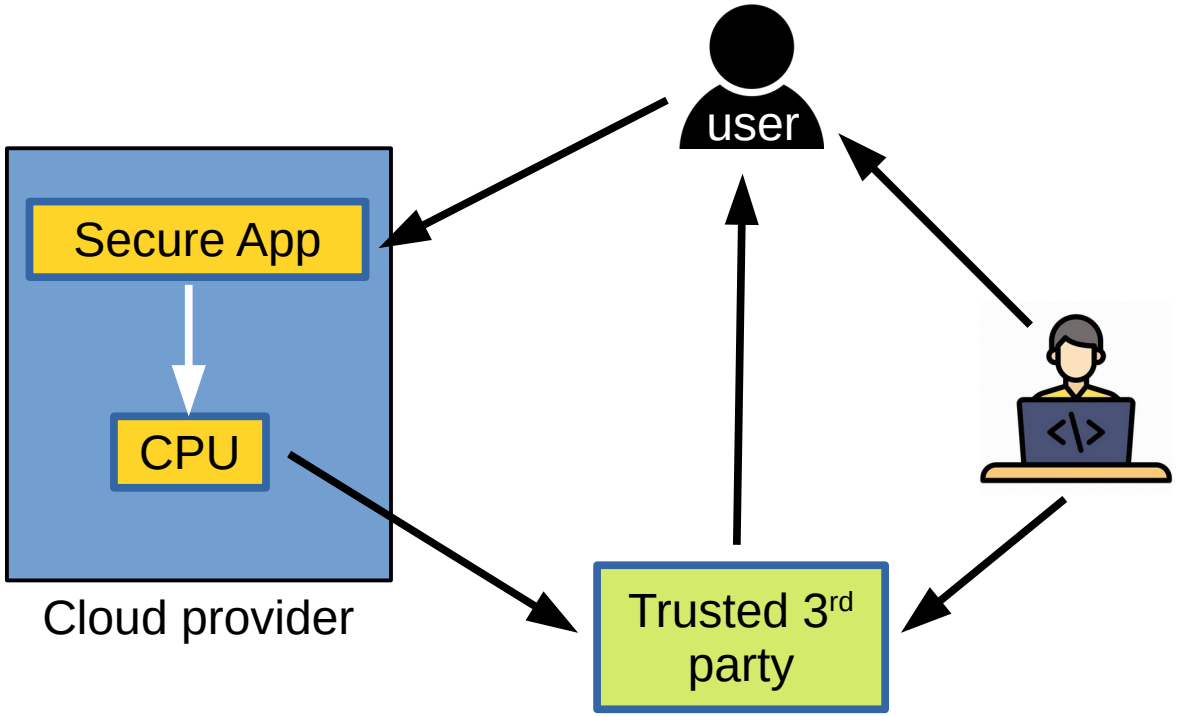
Handshake protocol**:**
Fast but **Complex**

- What could go wrong?
  - Specification errors
  - Attacks
  - …

Formal verification proves the protocol is correct:

- "it is not possible that a packet encrypted with an old key will be accepted after a key update"
- "an attacker cannot replay a valid packet to deceive the server

Formal Verification for Trusted Execution



Need to **trust** many participants

- Developer
- Application code & dependencies
- Application image repository
- CPU vendor
- Cloud provider

Formal Verification Tools (not exhaustive)

Target
Language
Proof Engine
Learning Curve
Abstraction Level
Link with code
Use-cases

Formal Verification Tools (not exhaustive)

	TLA+
Target	Design and architecture
Language	Temporal Logic
Proof Engine	Model Checking
Learning Curve	Medium
Abstraction Level	High
Link with code	None
Use-cases	Distributed Protocols

Formal Verification Tools (not exhaustive)

	TLA+	Verus
Target	Design and architecture	Implementation
Language	Temporal Logic	Rust w/ Annotations
Proof Engine	Model Checking	SMT solver
Learning Curve	Medium	Medium
Abstraction Level	High	Medium
Link with code	None	Direct
Use-cases	Distributed Protocols	Systems code

Formal Verification Tools (not exhaustive)

	TLA+	Verus	Lean
Target	Design and architecture	Implementation	Mathematical Proof
Language	Temporal Logic	Rust w/ Annotations	Dedicated Proof Language
Proof Engine	Model Checking	SMT solver	Proof Assistant
Learning Curve	Medium	Medium	High
Abstraction Level	High	Medium	Low
Link with code	None	Direct	Separated or Translation (e.g., Aeneas)
Use-cases	Distributed Protocols	Systems code	Cryptographic proofs

Formal Verification Tools (not exhaustive)

Target	Desi cture	In n	Mathematical Proof
Language	c	Rus ons	Dedicated Proof Language
Proof Engine	Model Checking		Proof Assistant
Learning Curve	Medium	Medium	High
Abstraction Level	High	Medium	Low
Link with code	None	Direct	Separated or Translation (e.g., Aeneas)
Use-cases	Distributed Protocols	Systems code	Cryptographic proofs

Formal Verification Examples

min() function in Verus

```
use builtin::*;

verus! {

fn min(x: i32, y: i32) -> (m: i32)
  ensures
    m == x || m == y,
    m <= y,
    m <= x,
{
    if x <= y {
        y
    } else {
        y
    }
}

} // verus!

 example.rs ▓▓ [rust] utf-8 84% 21:0
```

```
$ ./verus example.rs
note: verifying root module

error: postcondition not satisfied
  --> example.rs:12:1
   |
11 |         m <= x,
   |         ------ failed this postcondition
12 | / {
13 | |     if x <= y {
14 | |         y
15 | |     } else {
16 | |         y
17 | |     }
18 | | }
   | |_^ at the end of the function body

error: aborting due to previous error

verification results:: verified: 0 errors: 1
$
```

TLA+ Modeling of AWS outage DNS race condition

- November 05, 2025

```
CONSTANTS
    ENACTORS,           \* set of concurrent Enactors
    PLAN_AGE_THRESHOLD, \* number of plans after which a plan can be deleted
    MAX_PLAN            \* maximum number of plans (bounded model checking)

VARIABLES
    current_plan,       \* currently active Route53 plan
    latest_plan,        \* last plan generated by the Planner
    highest_plan_applied, \* highest version ever applied
    plan_deleted,       \* set of deleted plans
    plan_channel,       \* queue of plans awaiting enactment
    enactor_processing, \* which plan each Enactor is processing
    enactor_pc          \* Enactor program counter

(*************************************************************************)
(* Initial state                                                         *)
(*************************************************************************)

Init ==
    /\ current_plan = 0
    /\ latest_plan = 0
    /\ highest_plan_applied = 0
    /\ plan_deleted = [i \in 1..MAX_PLAN |-> FALSE]
    /\ plan_channel = << >>
    /\ enactor_processing = [e \in ENACTORS |-> 0]
    /\ enactor_pc = [e \in ENACTORS |-> 0]

(*************************************************************************)
(* Planner behavior                                                      *)
(*************************************************************************)

PlannerStep ==
    /\ latest_plan < MAX_PLAN
    /\ latest_plan' = latest_plan + 1
    /\ plan_channel' = Append(plan_channel, latest_plan + 1)
    /\ UNCHANGED <<current_plan, highest_plan_applied, plan_deleted,
                   enactor_processing, enactor_pc>>

(*************************************************************************)
(* Enactor behavior                                                      *)
(*************************************************************************)

EnactorReceiveStep(self) ==
  /\ plan_channel # << >>
  /\ enactor_pc[self] = 0
  /\ enactor_processing' =
       [enactor_processing EXCEPT ![self] = Head(plan_channel)]
  /\ enactor_pc' =
       [enactor_pc EXCEPT ![self] = 1]
  /\ plan_channel' = Tail(plan_channel)
  /\ UNCHANGED <<current_plan, latest_plan,
                 highest_plan_applied, plan_deleted>>
```

```
            THEN
                /\ current_plan' = enactor_processing[self]
                /\ highest_plan_applied' =
                    IF enactor_processing[self] > highest_plan_applied
                      THEN enactor_processing[self]
                      ELSE highest_plan_applied
                /\ enactor_pc' =
                    [enactor_pc EXCEPT ![self] = 2]
                /\ UNCHANGED <<latest_plan, plan_deleted,
                               plan_channel, enactor_processing>>
            ELSE
                /\ enactor_processing' =
                    [enactor_processing EXCEPT ![self] = 0]
                /\ enactor_pc' =
                    [enactor_pc EXCEPT ![self] = 0]
                /\ UNCHANGED <<current_plan, latest_plan, highest_plan_applied,
                               plan_deleted, plan_channel>>

Cleanup(my_plan, old) ==
    [i \in 1..MAX_PLAN |->
        IF (my_plan - i >= PLAN_AGE_THRESHOLD)
        THEN TRUE
        ELSE old[i]]

EnactorCleanupStep(self) ==
    /\ enactor_pc[self] = 2
    /\ plan_deleted' = Cleanup(enactor_processing[self], plan_deleted)
    /\ enactor_processing' =
        [enactor_processing EXCEPT ![self] = 0]
    /\ enactor_pc' =
        [enactor_pc EXCEPT ![self] = 0]
    /\ UNCHANGED <<current_plan, latest_plan,
                   highest_plan_applied, plan_channel>>

(*************************************************************************)
(* Next-state relation                                                   *)
(*************************************************************************)
Next ==
    \/ PlannerStep
    \/ \E self \in ENACTORS : EnactorReceiveStep(self)
    \/ \E self \in ENACTORS : EnactorApplyStep(self)
    \/ \E self \in ENACTORS : EnactorCleanupStep(self)


(*************************************************************************)
(* Invariants                                                            *)
(*************************************************************************)

NeverDeleteActive ==
    current_plan > 0 => ~plan_deleted[current_plan]


(*************************************************************************)
(* Specification                                                         *)
(*************************************************************************)

Spec ==
    Init /\ [][Next]_<<current_plan, latest_plan, highest_plan_applied,
                       plan_deleted, plan_channel, enactor_processing, enactor_pc>>

============================= END MODULE AwsDNSRace ============================
\* Example constants
\* MAX_PLAN == 5
\* PLAN_AGE_THRESHOLD == 2
\* NUM_ENACTORS == 2
```

Formal Verification: Not Difficult

TLA+ is not hard to learn. Amazon engineers report that:

> "Engineers from entry level to principal have been able to learn TLA+ from scratch and get useful results in two to three weeks, in some cases in their personal time on weekends and evenings, without further help or training."

THOUGHT LEADERSHIP

Learn Formal the Easy Way

August 31, 2021 • 3 MIN READ

TLA+ By Example

Learn TLA+ specifications through interactive examples. Write specs and run the TLC model checker directly in your browser.

June 5, 2025

AI is a gamechanger for TLA+ users

There has never been a better time to learn formal specification.

TLA+ Made Simple with ChatGPT

Sep 24, 2023

Conclusion: Please verify your systems

- Internet is too important and complex for testing only
- Formal Verification is now more accessible than ever
- Make the Internet safe and robust
  - **For you**: **less headaches** during **development**
  - **For everyone**: more **security**, more **trust**

The verification has already started!

TECHNOLOGY

Signal's protocol gets glowing reviews in first security audit

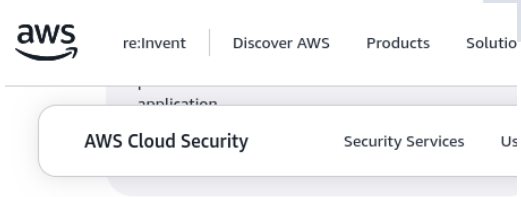
Signal is widely considered the gold standard of secure encrypted messaging apps but it hasn't been subject to a fine-toothed audit. What did researchers have to say?

BY PATRICK HOWELL O'NEILL • NOVEMBER 8, 2016

Revolutionizing Network Protocol Testing: A New Era of Validation

"Towards verification of QUIC and its extensions"

ElNiak Follow 10 min read · Jan 26, 2024



Continuous Formal Verification of Amazon s2n

We describe formal verification of s2n, the open-source TLS implementation used in numerous Amazon services. A key aspect of this proof infrastructure is continuous checking to ensure that properties remain proven during the lifetim…

Internet Security Research Group -- Memory Safety project