

Towards authentication of transparent systems

Fredrik Strömberg @ OSFC Oct 11th, 2023

“All models are wrong, but some are useful.”

– George Box (Science and statistics)

System Transparency Authentication Mechanism

STAM

is (or rather will be)

an authentication mechanism for transparent systems.

A system is transparent if its reachable state space

- *can be cryptographically proven by induction starting from the platform's procurement and provisioning manifest,*
- *has enough conceptual integrity to be comprehended by the auditor and*
- *is globally discoverable by a transparency log monitor.*

Work in progress!

STAM

builds on

- secure network communication protocols with entity authentication mechanisms
- remote attestation
- reproducible builds
- Sigsum (transparency logging with witness cosigning) and
- ~ “state space constraint auditability” (?)

Secure communication

Secure communication protocol design

10–20 years ago:

Cryptographically porous

Today:

Cryptographically verifiable (e.g. WireGuard)

Secure communication protocol design

1. Pre-shared public key ->
2. ECDH ->
3. KDF ->
4. Symmetric enc/dec & sign/verify (e.g. HMAC):

Authenticated, forward secret, confidential and integrity-protected communication channel.

**TLS, SSH and VPN protocols
are**

secure communication protocols

with

entity authentication mechanisms.

TLS

(and other secure communication protocols)

- Authentication is done by establishing trust in a certificate.
- The certificate contains an identity and a public key.
- The authenticated key is used to establish secure communication.
- Secure communication can be established assuming the remote system's private key has not been compromised.

STAM

(and remote attestation mechanisms)

- Authentication is done by establishing trust in a certificate.
- The certificate contains an identity, a public key, **and other claims about the remote system's attributes.**
- The authenticated key is used to establish secure communication.
- Secure communication can be established assuming the remote system's private key has not been compromised.

STAM

will assure a local system of a remote system's

- platform provenance (signed provisioning manifest)
- platform state (signed TPM measurements)
- software authenticity (signed code)
- source code traceability (signed reproducible builds)
- attestation freshness (measure a recent Sigsum STH)
- certificate transparency (Sigsum log all signatures)
- human-readable identity (e.g. X.509 TLS Certificate)

STAM

signature details

- DC engineer signs (signed provisioning manifest)
- TPM signs (signed TPM measurements)
- Developer signs (signed code)
- Build chain signs (signed reproducible builds)
- Log, witnesses & TPM sign (measure a recent Sigsum STH)
- Log & witnesses sign (Sigsum log all signatures)
- Web CA signs (e.g. X.509 TLS Certificate)

STAM

signature details, cont.

- DC engineer signs : TPM event log during provisioning
- TPM signs : build artefacts in boot chain
- Developer signs : build artefacts
- Build chain signs : build artefacts, source code
- Log, witnesses & TPM sign : Sigsum STH
- Log & witnesses sign : all signatures mentioned
- Web CA signs : domain name

STAM

does not guarantee prevention of yours or others

- design mistakes
- implementation mistakes
- configuration mistakes
- vulnerabilities
- backdoors

Nor does it guarantee protection from a malicious person who procures and provisions the platform.

Conceptual models

STAM cryptographically connects

1. a procured and provisioned platform,
2. enrolled in your fleet with an identity and
3. software

to a discoverable and auditable state space of the running system.

| | Conceptual understanding | Human-readable | Machine-usable | Running system |
|----------|--|----------------|-----------------------|----------------|
| Software | !!! RCE, • etc, • misconfig. | source code | build artifact | |
| Hardware | !!! yarch • control hazards • pipelines Spectre, Rowhammer | blueprints | physical construction | |

ROM (also instr. integrity)

↓
Memory map



instr ~

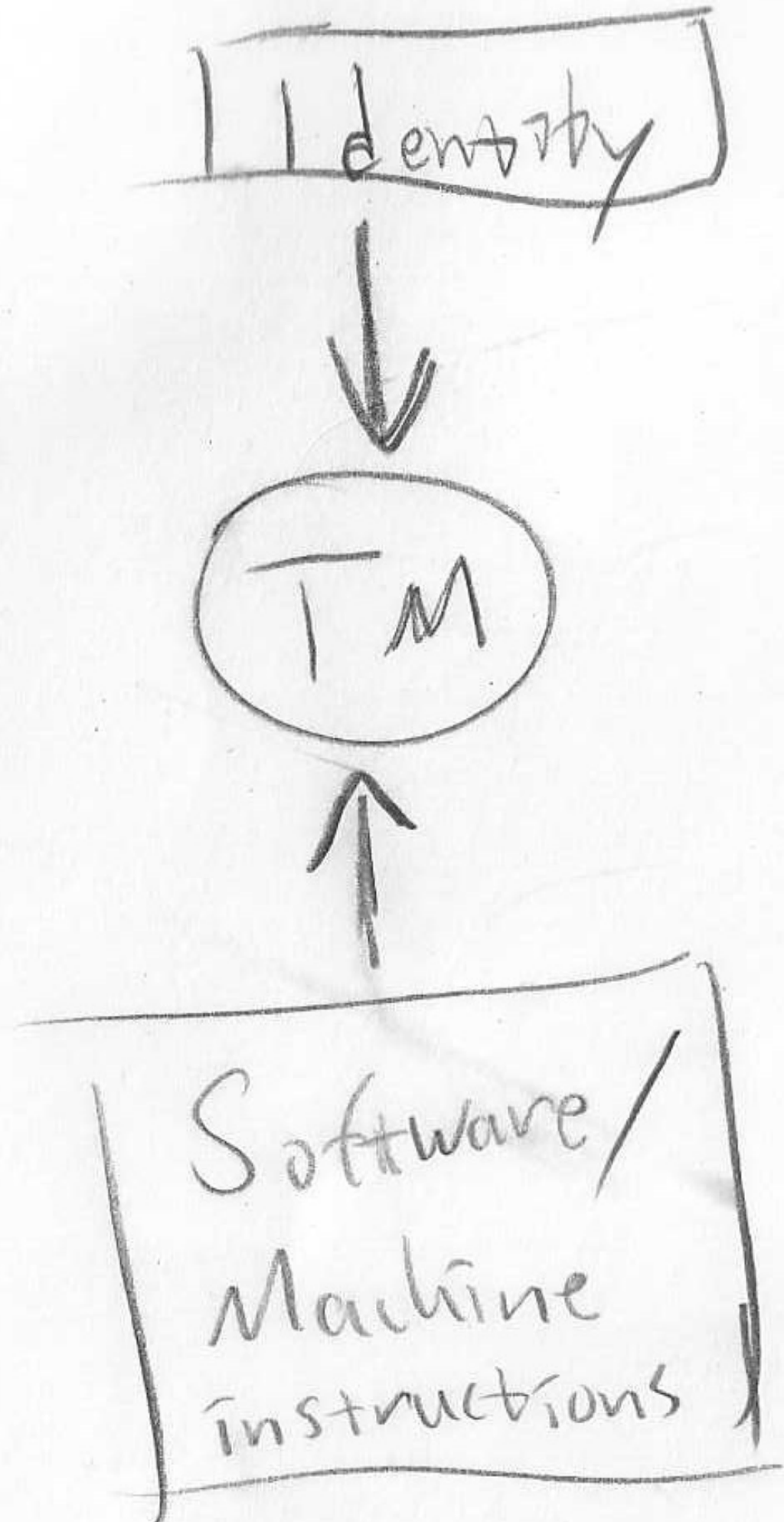
| | |
|----|----|
| AF | |
| 03 | FB |
| | |
| | |

....

} infinite loop
while;
true

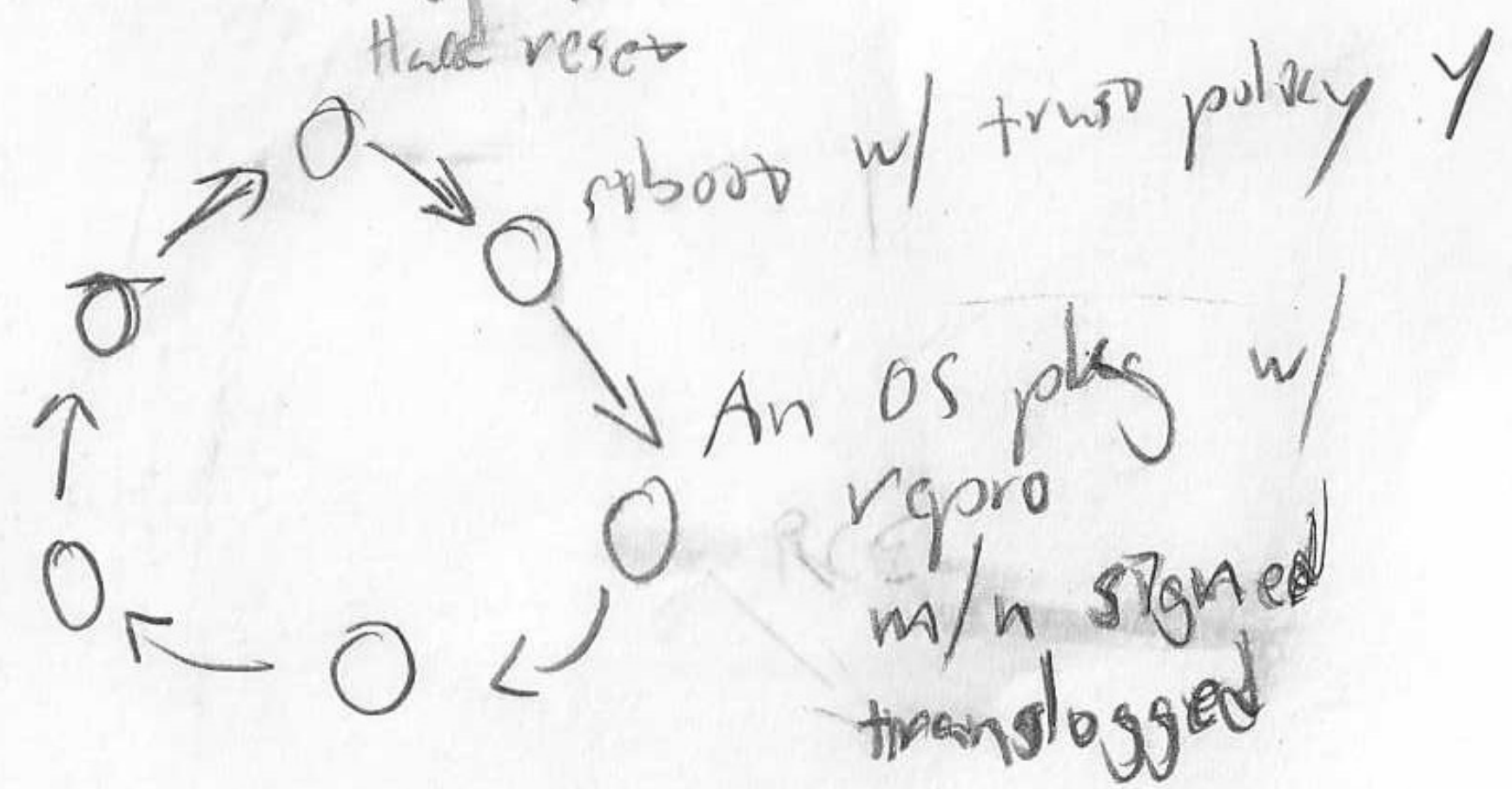
State graph

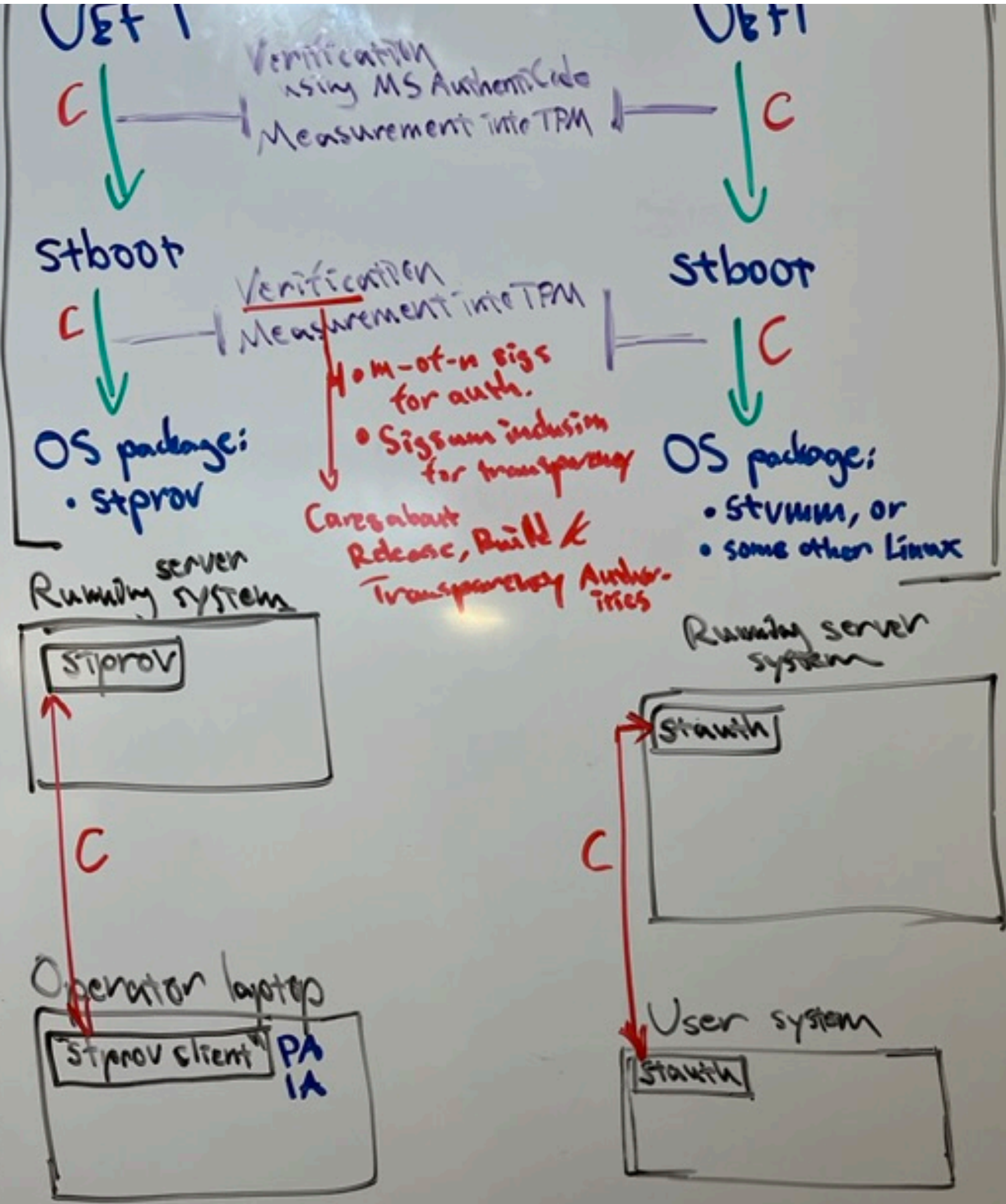




"Remote system 'Id' has state graph X"

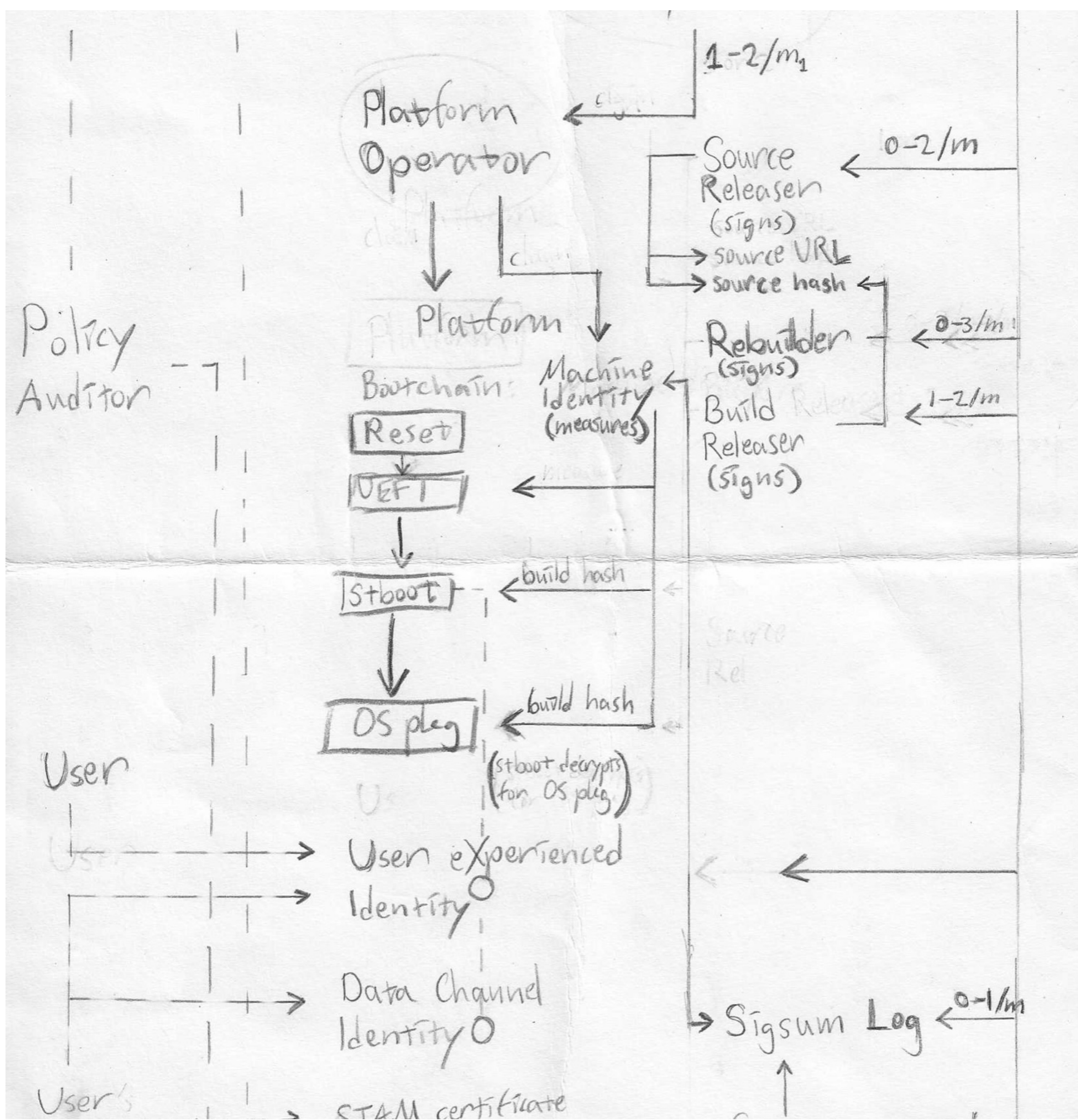
State graph





Verification Measurement

- Other systems
- C Build Release Auth
 - C Repro Build Auth
 - C Source Release Auth
 - C Sigsum log
 - C -u- witnesses
 - C -h- monitor
- OS package CDN



System Transparency
design philosophy

“Exploitation of vulnerabilities can be modeled as the practical exploration of the space of states not intended by the designer of the system.”

– langsec.org

Complexity

Attack
surface

Actual
behaviour

Expected behaviour



“Life is complex, as are the tasks we encounter. Our tools must match the tasks.”

“Complexity is good; It is confusion that is bad.”

“The most important principle for taming complexity is to provide a good conceptual model.”

– Don Norman (The design of everyday things)

“Dealing with complexity is the most important challenge in software design.”

– John Ousterhout (A philosophy of software design)

“The most fundamental problem in computer science is problem decomposition: how to take a complex problem and divide it up into pieces that can be solved independently.”

– John Ousterhout (A philosophy of software design)

“There are two general approaches to fighting complexity:

1. Make code simpler and more obvious

2. Encapsulate it so that programmers can work on a system without being exposed to all of its complexity at once”

– John Ousterhout (A philosophy of software design)

**“Thinking clearly is hard: we
can use all the help we can
get.”**

– Leslie Lamport (Specifying systems)

“Conceptual integrity is the most important consideration in system design.”

– Fred Brooks (The mythical man-month)

“One of the most important elements of good software design is separating what matters from what doesn't matter.”

– John Ousterhout (A philosophy of software design)

“To decide what matters, look for leverage, where the solution to one problem also allows many other problems to be solved, or where knowing one piece of information make it easy to understand many other things.”

– John Ousterhout (A philosophy of software design)

Complexities that matter

Computational complexity is the foundation of modern cryptography.

What would the Internet look like without TLS?

Conceptual complexity is the root cause of most(?) security vulnerabilities.

High-leverage defences

(increase confidence that intended and actual system behaviour are the same)

1. Maximise use of cryptographic defences

(as cryptography relies on computational complexity)

2. Maximise use of hardware defences

(as hardware defines the rules of software)

3. Maximise conceptual integrity

(see quotes)

4. Constrain the reachable state space whenever possible

5. Distribute trust assumptions

Measured and verified boot

1. Load untrusted artefact into memory
2. Hash untrusted artefact
3. **Verify** (given some trust policy)
4. Execute artefact (if it passed verification)
3. **Measure** (i.e. append to append-only platform state)
4. Execute artefact unconditionally

Uncond. measured vs verified boot

- 2nd hand value
- e-waste
- key compromise
- design complexity of "1st instruction integrity"
- boot performance
- attack resilience
- so when is verified boot needed?

Measured boot variants

- Different CPU mode contexts (eg sm, um)
- Different cores and/or ICs (TPM, AMD SEV SNP, etc)
- Different stages of the bootchain (DICE, Tllitis TKey)

Tillitis TKey

uses software measurement to derive key material

$CDI = \text{Hash}(\text{UDS}, \text{Hash}(\text{software}), \text{USS})$

- Compound Device Identifier
- Unique per Device Secret
- User-Supplied Secret

Tillitis TKey

flexibility

- while true
- Ed25519 sign
- Verified boot
- M-of-n verified boot
- M-of-n verified boot and Sigsum inclusion proof check

Tillitis TKey

flexibility, cont.

1. Measured boot followed by
2. M-of-n verified boot and Sigsum inclusion proof check
3. Sigsum inclusion proof check of hash to be signed
4. Ed25519 sign

Tillitis TKey

flexibility, cont. 2

1. Measured boot followed by
2. M-of-n verified boot and Sigsum inclusion proof check
- 3. Secure communication channel**
4. Sigsum inclusion proof check of hash to be signed
5. Ed25519 sign

Tillitis TKey
flexibility, cont. 3