IoT Security and Privacy: Remote Attestation

CPS and IoT Security

Alessandro Brighente

Master Degree in Cybersecurity

UNIVERSITÀ DEGLI STUDI DI PADOVA

SPRITZ SECURITY & PRIVACY RESEARCH GROUP

The Software Problem

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- IoT devices have their own software and this can be compromised by malicious entities
- It is not easy to detect attacks in on-field devices, as Stuxnet showed
- We need solutions to attest the legitimacy of (software and hardware) components by distantly looking at them and their behavior
- We need to account for the constrained resources of these devices

Remote Attestation

- Attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim
- An attester is a party performing the attestation activity
- An attestation protocol is a cryptographic protocol involving a target an attester, an appraiser and possibly other principals serving as trust proxies
- Target: supply evidence that will be considered authoritative by the appraiser while respecting privacy goals of its target

Remote Attestation

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- We denote as remote attestation a protocol whereby a challenge (Chal) verifies the internal state of a device called a prover (Prov)
- This protocol is performed *remotely*, i.e., over the Internet
- Goal: an honest Prov should create an authentication token that convinces Chal that the former is some well-defined expected state
- If Prov has been compromised by an adversary, the authentication token must reflect this

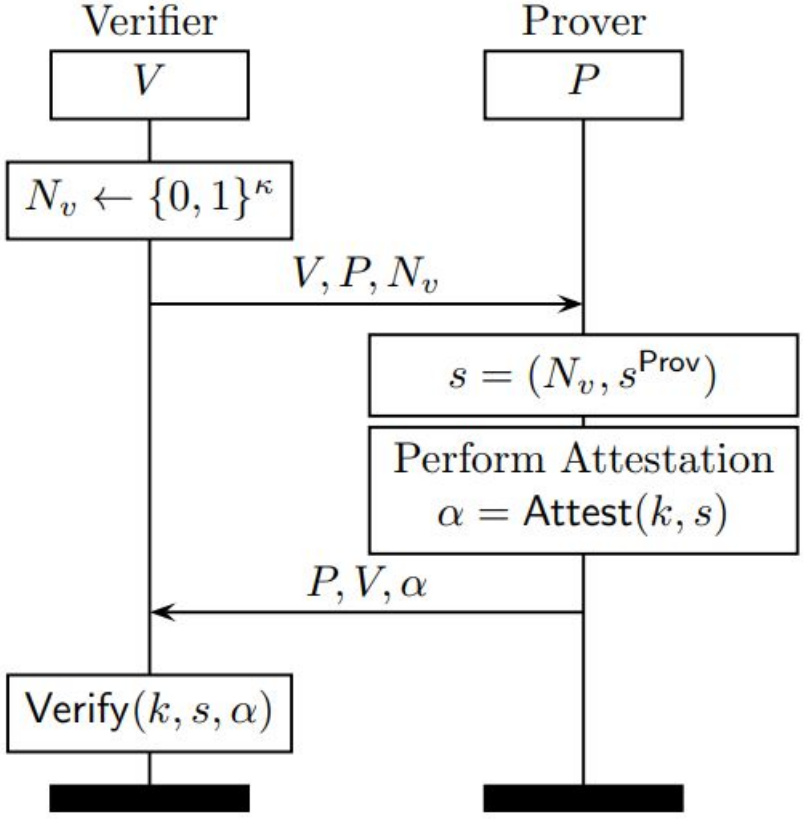
Attestation Protocol

An attestation protocol P is comprised by the following components

- Steup() - a probabilistic algorithm that, given a security parameter 1^κ outputs a long-term key k;
- Attest(k,.) - a deterministic algorithm that, given a key k and a device state s, outputs an attestation toke a
- Verify(k,.,.) - a deterministic algorithm that, given a key k, a device state s, and an attestation token a, outputs 1 iff a corresponds to s, i.e., iff Attest(k,s) = a, and outputs 0 otherwise

Attestation Protocol

- The verifier challenges the prover with a fresh nonce (uniformly random and from a large pool)
- The prover attests its state with the key and creates a token
- The verifier receives the token and decides whether to accept it



Att-Forgery Security

Let us define the following game

Game 1 (Att-Forgery$_{Chal,Prov}$(k)): Chall running P interacts with Prov as follows

1. Chal runs k ← Setup(1^k) and outputs s^{Chal} to Prov
2. Prov is given oracles access to Attest, i.e., adaptively submit q device states and receive the corresponding token
3. Eventually Prov outputs a; the game outputs 1 iff Verify(k,s,a) = 1, i.e., iff a corresponds to s = (s^{Chal},s^{Prov})

Att-Forgery Security

- An honest node has no problem winning the previous game
- If instead Prov has been compromised, its s^{Prov} has changed and must attempt to simulate the operation of Attest
- We can define the following security notion for an attestation protocol

Definition: Att-Forgery Security. An attestation protocol P = (Setup, Attest, Verify) is Att-Forgery-secure if there exists a negligible function negl such that for any probabilistic polynomial time prover Prov and sufficiently large k it holds $Pr[\text{Att-Forgery}_{\text{Chal Prov}}(k) = 1] \leq \text{negl}(k)$

Adversary Model

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- We assume the adversary can compromise Prov at any time
- Once it is compromised, the adversary has control over the prover device
- There however needs to be a key that the adversary cannot access to although being in control of the prover
- We assume that the adversary cannot modify the hardware components of the compromised device
- We also assume that there is a way to protect Attest against side channel attacks

Properties Required

- Attest needs to have specific security properties
- We assure there exists a secure algorithm to compute a based on the prover's state s and a prover-specific key k (e.g., via HMAC)
- Attest must satisfy the following security properties
  - Only Attest can compute a valid token a
  - A accurately captures s, i.e., Attest(k,s) = Attest(k,s') with negligible probability
- Two ways to attacker remote attestation
  - Attack 1: The adversary simulates Attest and correctly computes a
  - Attack 2: returned a does not reflect s, i.e., escape detection

Properties Required

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The key k is the only secret held by Prov, and access to k allows the adversary to simulate Attest (i.e., type 1 attack)
- Exclusive access: attest must have exclusive access to k.
- No leaks: Attest leaks no function of k other than a, i.e., after Attest completes, the entire state of Prov is statistically independent from k
- Immutability: Attest code is immutable. This means that it needs to be executed in-place from immutable memory
- Uninterruptibility: the attacker has no means to interrupt the execution of attest

Types of Remote Attestation

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Remote attestation can be performed in several ways, with different requirements in terms of device capabilities, equipment, and security guarantees
- At a high level, we can distinguish between software-based attestation and root of trust-based attestation

Software-Based Attestation

SPRITZ
Security & Privacy
Research Group

Università
degli Studi
di Padova

- In software-based attestation, typically we use timing information to allow the verifier to assess the correctness of the firmware running on the prover
- These approaches generally require strict timing requirements on the network, which might not always be feasible in generic IoT
- They also generally consider a one-hop communication between verifier and prover, which makes it hard to be realized in large IoT networks

Software-Based Attestation

- A Naive approach for verifying the prover's memory content would be to challenge the prover in computing a MAC of the memory content with a verifier-provided key
- The verifier knows the memory content of the legitimate device
- However, an attacker could easily cheat on this
- Indeed, the attacker could save the original memory content and move it to an empty portion of the memory or to an external device that could be accessible when needing to compute the MAC-based proof

Pseudo-Random Memory Traversal

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The embedded device has a memory-content verification procedure that can be remotely activated by the verifier
- The procedure uses a pseudorandom memory traversal
- In particular, the challenge is used as seed for a pseudorandom number generator that generates a list of memory addresses to be checked
- The adversary has no mean to know in advance the portion of the memory that will be checked

Pseudo-Random Memory Traversal

SPRITZ
Security & Privacy
Research Group

Università
degli Studi
di Padova

- In case the verification procedure is heading towards a portion of the memory that has been altered, the attacker needs to divert it to the a memory location where the correct copy is stored
- This however causes an increase in the time needed to compute the verification
- Thus the verifier will either see a non valid authentication token or a suspiciously long time needed to compute the authentication token

Attestation Based On Root of Trust

SPRITZ
Security & Privacy
Research Group

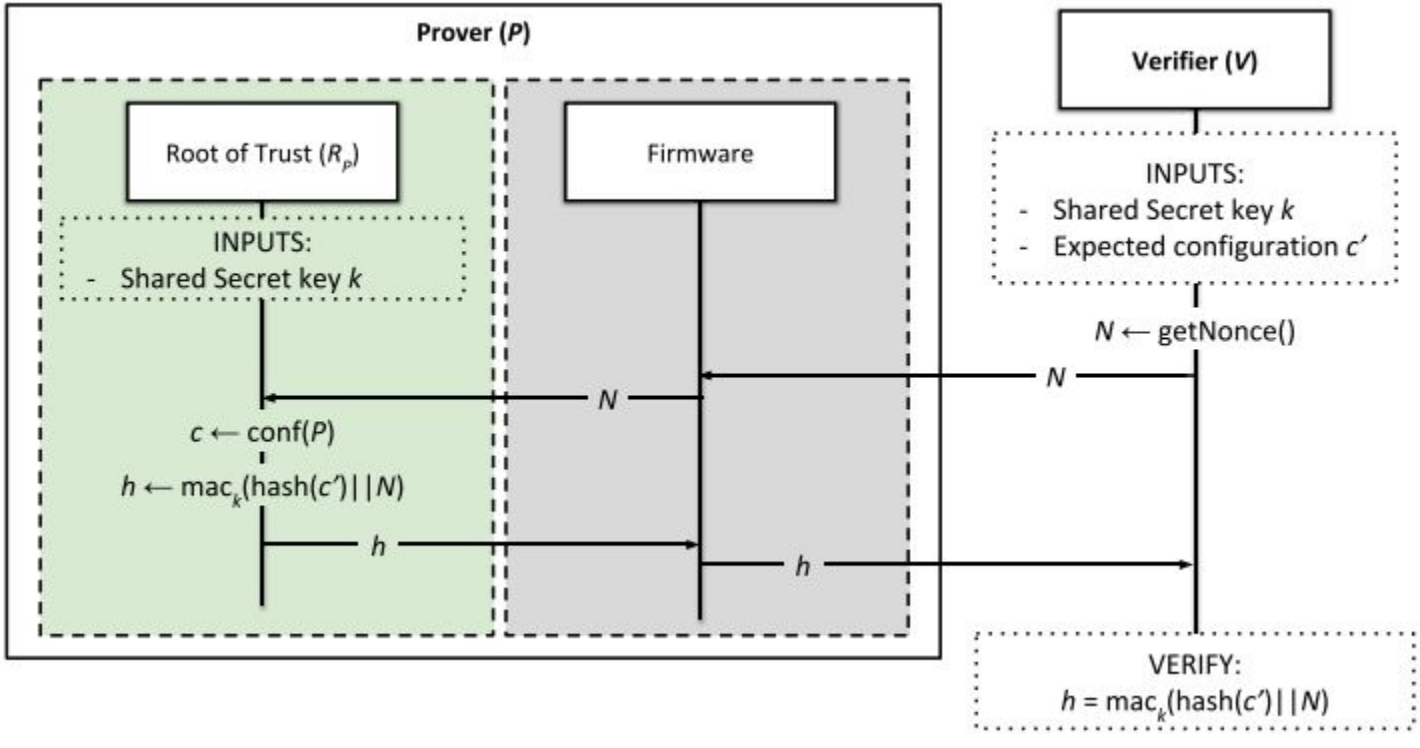
Università
degli Studi
di Padova

- These schemes leverage a root of trust residing inside the prover
- This component is a assumed to be trusted and is the endpoint of the attestation protocol
- It usually comprises a combination of hardware and software
- The value to be attested is stored inside the root of trust
- In IoT devices, the root of trust is realized using hardware with minimal security capabilities, such as code and memory isolation
- Four strategies: interactive RA, Interactive Self-RA, Non-interactive RA, and non-interactive self-RA

Interactive RA

- It consists of an interactive protocol between prover P and verifier V
- V sends a challenge N to P's root of trust R_p, which responds with a proof of the device's configuration, e.g., c = hash(conf(P))||N
- The proof h is either signal (public key crypto) or tagged via MAC (symmetric key)
- V verifies the integrity of P by verifying the authenticity of h

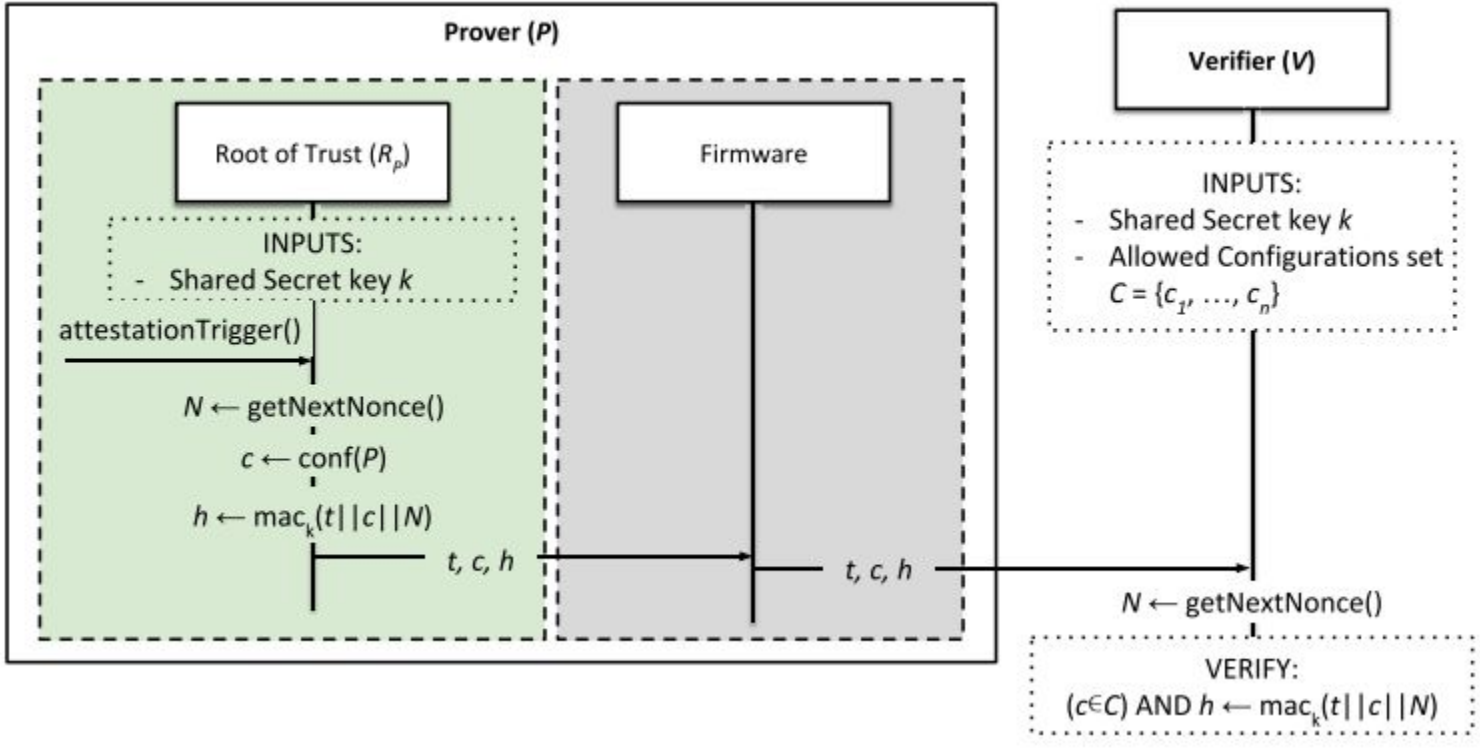
Interactive RA



Interactive Self-RA

- Leveraging the capabilities of Trusted Execution Environments, it is possible to verify c at P's side given the list of potential allowed configurations securely stored and accessible by R_p
- After receiving N from V anc computing c, R_p produces a signed/MACed token h authenticating a binary result r (true of false)
- This is then delivered to V as a customized token

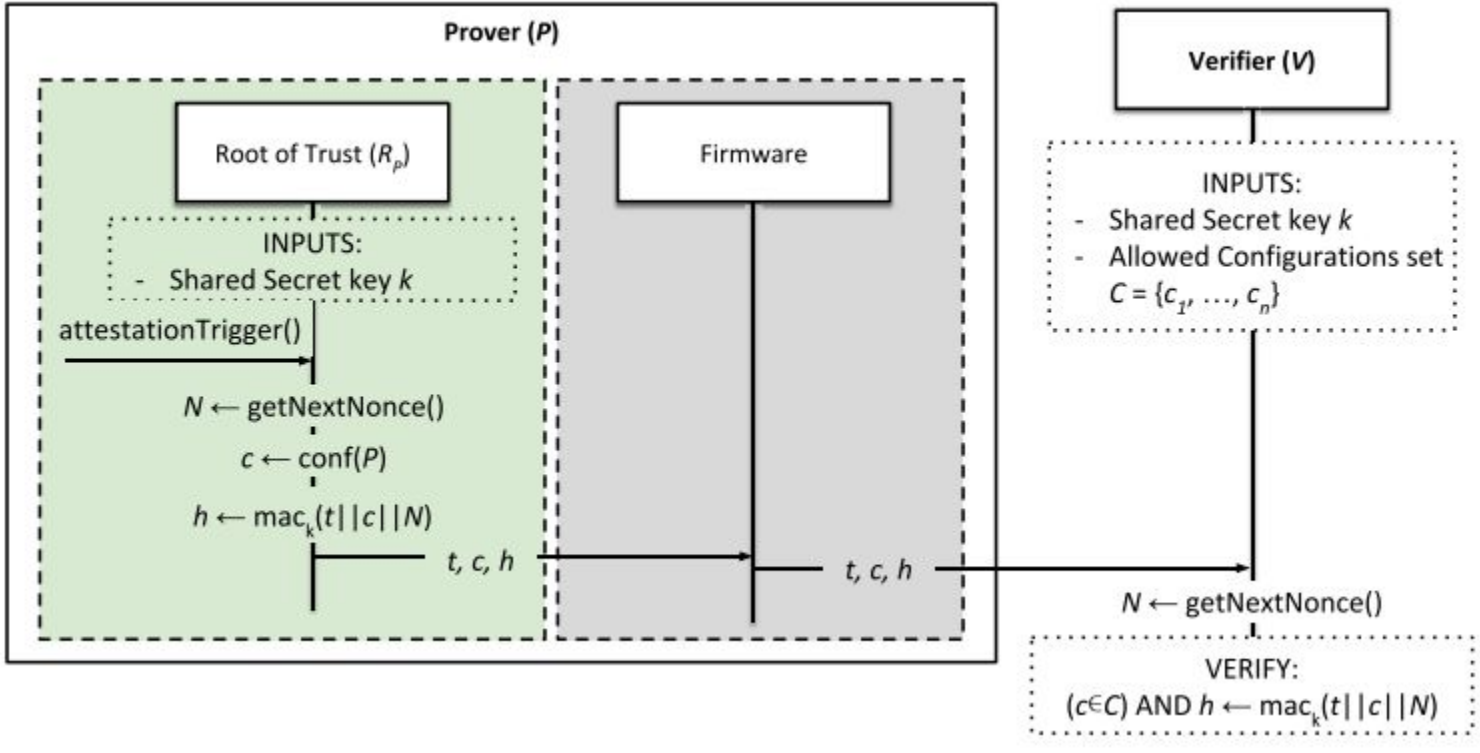
Interactive Self-RA



Non-Interactive RA

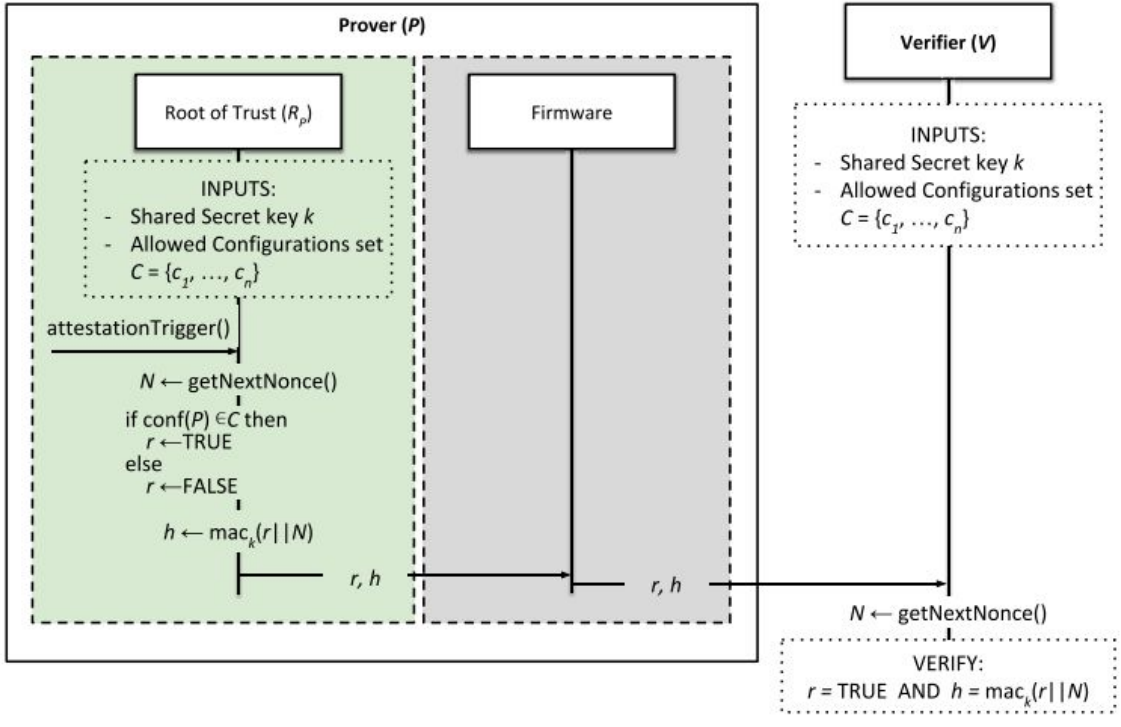
- The prover P autonomously decides the time at which attestation should happen and locally generates a pseudo-random nonce N
- Remove the need for V to start the process, but require additional hardware, e.g., secure source of time
- Examples of additional needs include Real Time Clocks, Attestation Trigger Circuit, or Reliable Read-Only Clocks

Non-Interactive RA



Non-Interactive Self-RA

- Variation of the previous one, where P's TEE know the set of possible configurations and can therefore perform self attestation



Collective RA

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- In large networks, it might be convenient to assess the status of multiple nodes instead of performing single attestations
- Collective remote attestation has been introduced to limit the time needed to perform attestation of multiple interconnected devices
- We consider a large network of low-end devices which are heterogeneous in terms of software and hardware configurations
- These are provers, and of course we have the verifier
- We need an additional device, i.e., the aggregator

An Example of Collective RA

- Let us consider an IoT distributed system where nodes communicate in an asynchronous manner via a publish/subscribe pattern
- The verifier performs attestation in two steps: initialization at time T_0 and attestation at time T_1
- During initialization time, the verifier initiates the attestation procedure with one or more services (publishers)
- The publisher then performs the local attestation and publishes the result together with the data it produced

An Example of Collective RA

- Every subscriber service receives the publish the data and also perform the attestation
- At attestation time, verifier sends an attestation request to one or more subscribers, which act as prover for the whole network
- Subscribers report an attestation result that includes the result of all the previous services that were directly or indirectly involved in triggering a given event to which the subscriber was registered

An Example of Collective RA

Verifier

Publisher

Time T_0 $R \leftarrow \{0,1\}^n$;
$\sigma_{Vrf} \leftarrow sig(SK_{Vrf}; P \,||\, R)$; ① $Ch = \{P, R, \sigma_{Vrf}\}$ →

if $(vrfsig(PK_{Vrf}; P \,||\, R, \sigma_{Vrf}))$ **then**
Begin
If $(timestamp_P$ is **null**) **then**
$timestamp_P[P] = 0$;
$ServID \leftarrow P$;
$GHV_{prev} \leftarrow 0$;
$Input_P \leftarrow read()$;
$Output_P \leftarrow exec(ServID, Input)$;

② **attest()**
Begin
$LHV_P \leftarrow$ **checksum**(P);
$timestamp_P[P] \leftarrow timestamp_P[P] + 1$;
$\tau \leftarrow ServID \,||\, timestamp_P \,||\, LHV_P \,||\, Output_P \,||\, Input_P \,||\, GHV_{prev}$;
$GHV_P \leftarrow Enc(PK_{Vrf}; \tau)$;
End

③ **publish()**
Begin
$msg_P \leftarrow Output_P \,||\, GHV_P \,||\, timestamp_P$;
$\sigma_P \leftarrow sig(SK_P; msg_P)$;
End

Else
Reject *Ch*;
End.

An Example of Collective RA

Verifier | Publisher | Subscriber

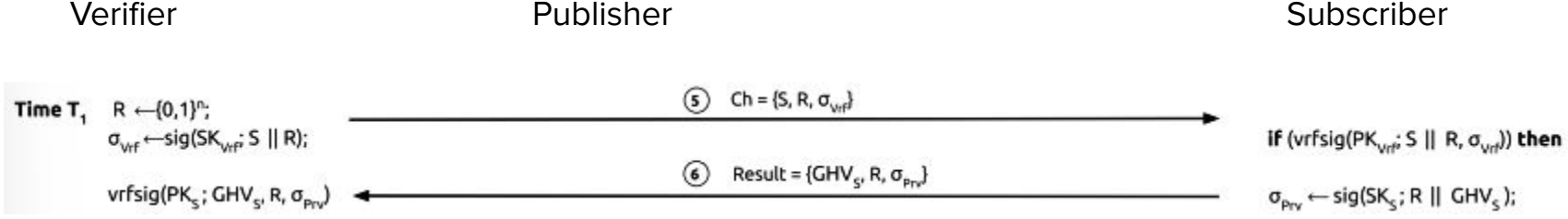
$data = \{msg_P, \mu_P\}$

```
If (vrfsig(PK_P; msg_P, σ_P)) then
Begin
If (timestamp_S is null) then
   timestamp_S [S] = 0;
 ServID ← S;
 Output_P || GHV_P || timestamp_P ← msg_P;
 for (i=0; i< length(timestamp_P); i++) {
    timestamp_S [i] =  max(timestamp_P [i], timestamp_S [i]);
 }
 Input_S ← Output_P;
 GHV_prev ← GHV_P;
 Output_S ← exec (ServID, Input_S);
```

④

```
attest()
Begin
  LHV_S ← checksum(S);
 timestamp_S [S] ←  timestamp_S [ S] + 1;
  τ← ServID || timestamp || LHV_S || Output_S || Input_S || GHV_prev;
  GHV_S ← Enc(PK_Vrf ; τ);
End
Else
    Reject data;
End.
```

An Example of Collective RA



- This is the final attestation procedure, where the verifier retrieves the attestation result GHV_S, singed, and containing the received nonce