Optimizing Proof of Aliveness in Cyber-Physical Systems

Zheng Yang[1], **Chenglu Jin**[2], Xuelian Cao[1], Marten van Dijk[2], Jianying Zhou[3]

[1] Southwest University,

[2] CWI Amsterdam,

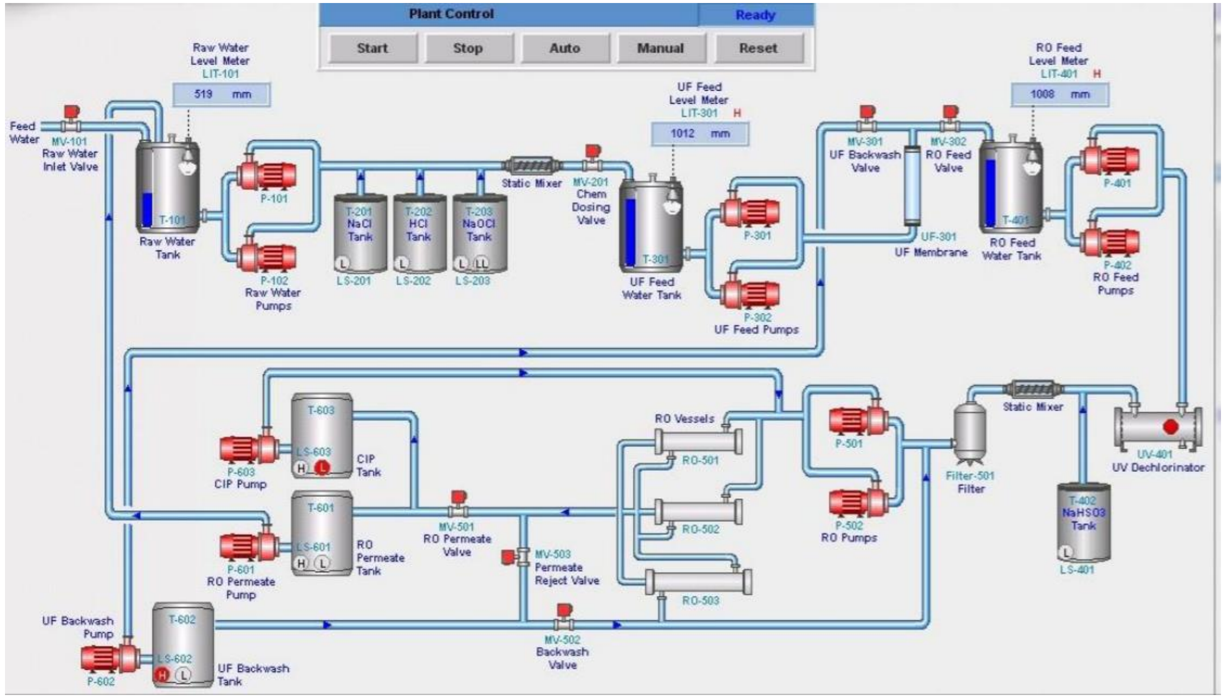
[3] Singapore University of Technology and Design

*Yang and Jin share the first authorship.

Published at IEEE IEEE Transactions on Dependable and Secure Computing 2024

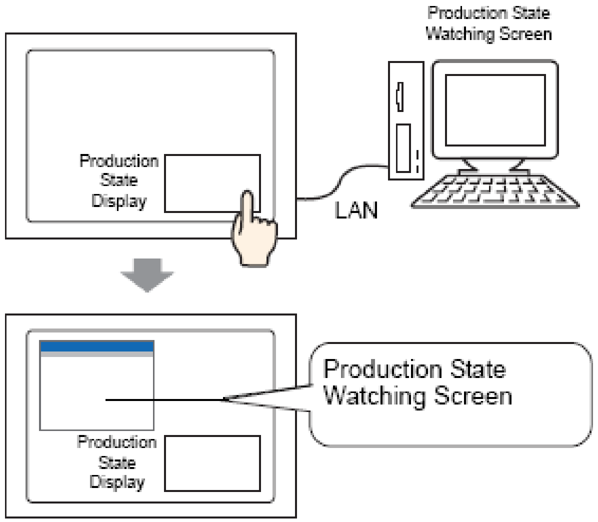
Aliveness of devices in cyber-physical systems

- Aliveness ≈ **Continue** functioning as designed
- Importance of Aliveness:
  - Work collaboratively
  - Critical components
    - Blackout
  - Safety critical components
    - Triton targeting safety instrumented systems (SIS)



Check the aliveness

- Track the running status of the devices
- Immediately raise alarm, and fix it

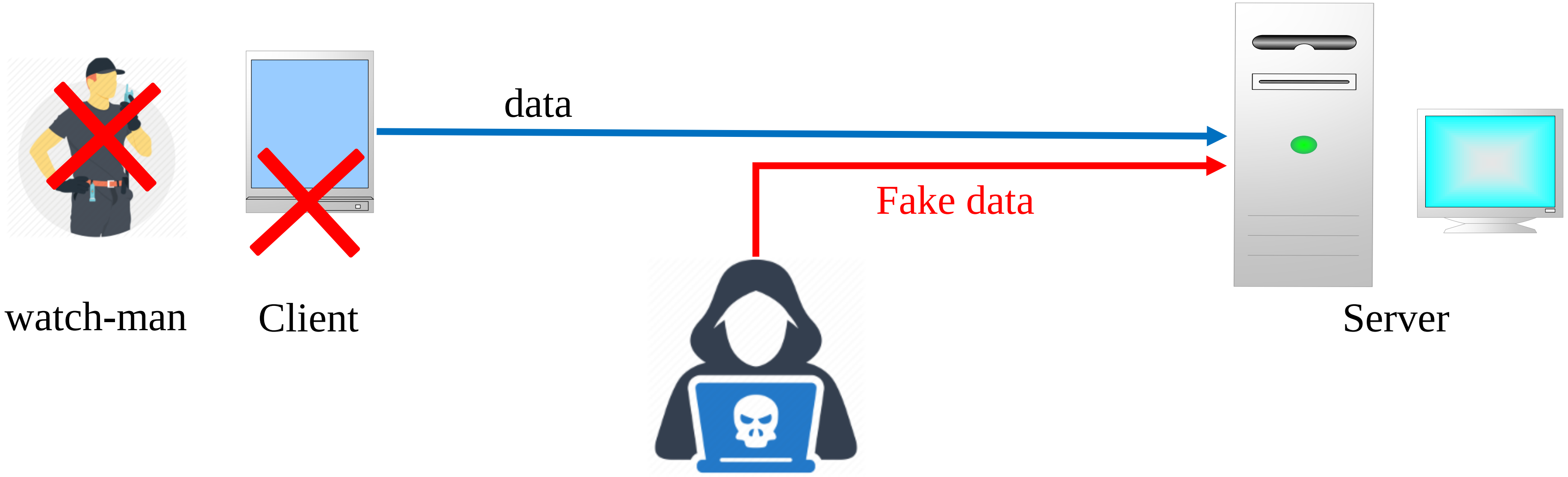


On-site check

Remote Monitor

Challenge in checking the aliveness remotely

- **Inject fake data** against automatic check



- Hard to identify the death promptly

This work: Proof of Aliveness

- Cryptographic notion - PoA
  - **Two-party protocol**: prover (client), verifier (server)
  - **Heartbeat pattern**: the prover periodically sends proofs to a verifier with a fixed time interval Δ_s, e.g., every Δ_s=30 seconds
  - **Dead** if no valid proof within *aliveness tolerance time* T_{att}, e.g., T_{att}=3 minutes

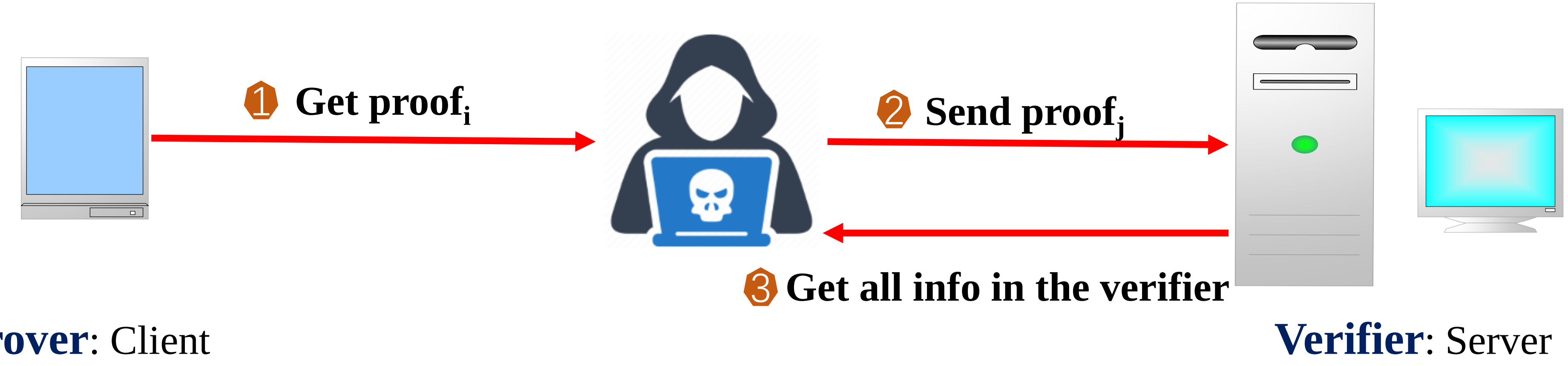
Prover: Client

Verifier: Server

Δ_s

Security model for PoA

- **Adversary model: network attacker**
  - **Eavesdropping, injecting, and replay attack are allowed**
  - **Server can be compromised**



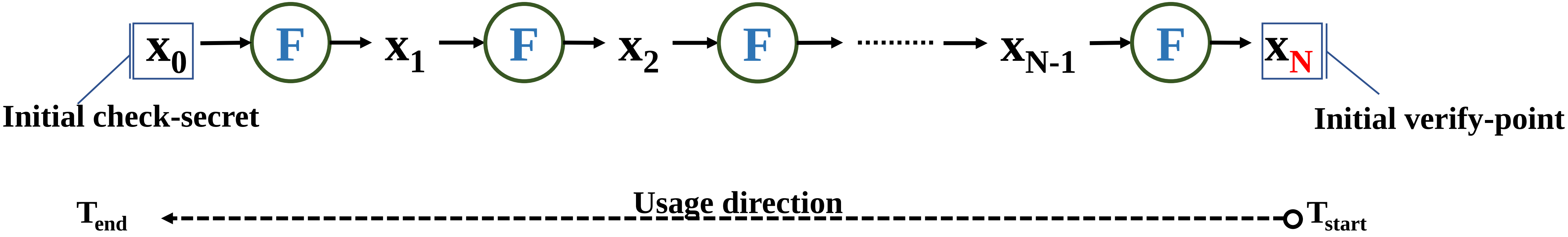
- **Security goal**: no adversary can forge a valid aliveness proof (especially when the prover is dead)

How to realize PoA

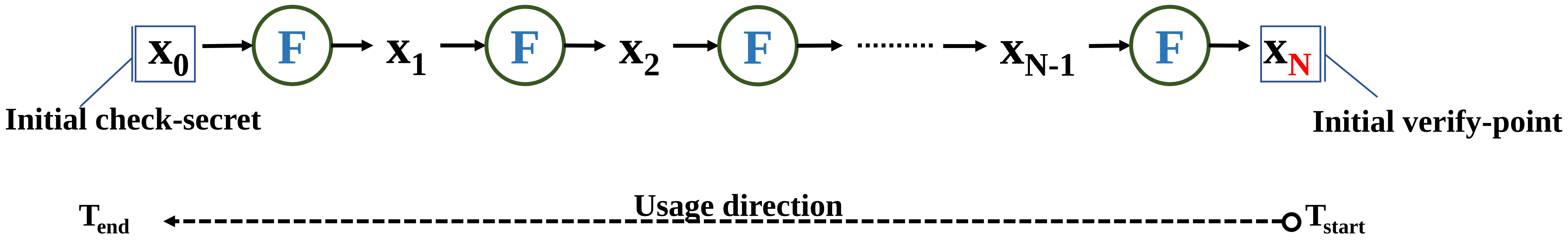
- Digital signature
  - Inefficient for resource-constrained devices
- Message authentication code
  - Subject to Server compromise attack
- **Time-based one-time password**
  - Lightweight, rely on hash or one-way function (OWF)
  - Server compromise resilience, e.g., T/Key [DMB17]
  - Passwords=Proofs sent in a constant pace, every Δ_s seconds

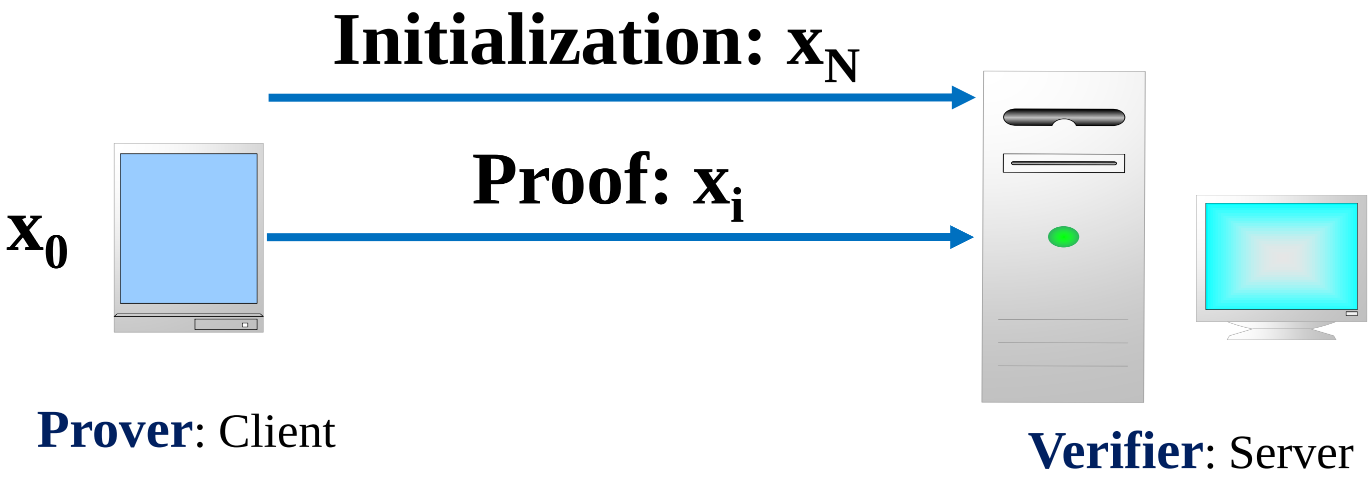
Single-chain PoA Π_{OWF} from [Lam81]

- One-way function **F:** $\{0,1\}^m \rightarrow \{0,1\}^m$
  - **Easy to compute F, but very hard to compute F^{-1}**
  - **One-way function chain: $X_i = F^i(X_0)$, where X_0 is random**



Single-chain PoA Π_{OWF} from [Lam81]





Verify-point x_j $(i+1 \leqslant j \leqslant N)$

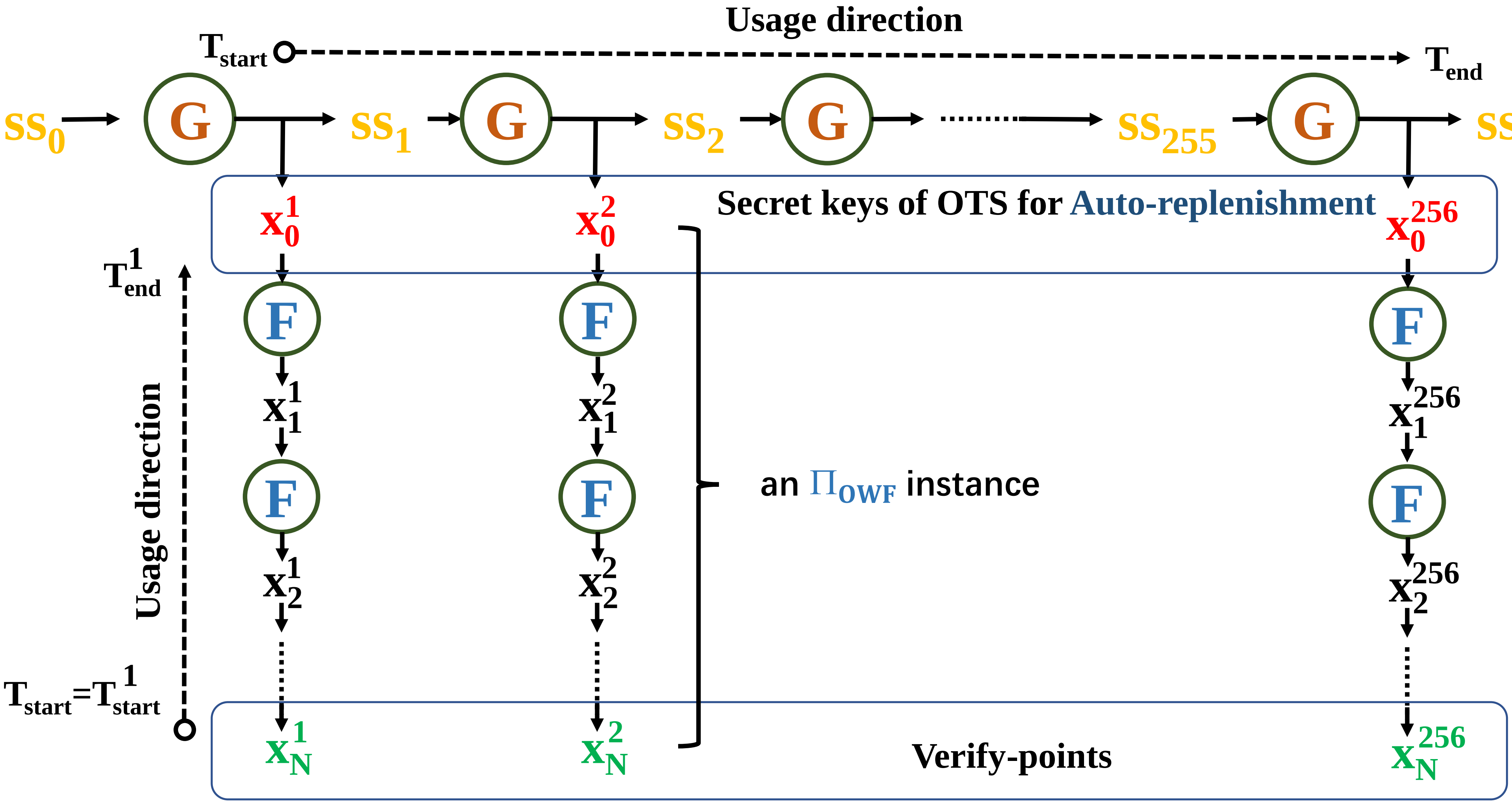
Check x_j ?= F (x_i)

Update $x_j := x_i$

Π_{OWF}: limitation

- Finite number of proofs
  - Does not match with the super long life time of CPS devices
- The total number of proofs N=1 million $\rightarrow$ 1 years with Δ_s =30 seconds intervals
- We need to auto-replenish the proofs by the protocol itself
  - Assuming no long-term/master keys

Multiple-chain PoA $\Pi_{\mathrm{OWF}}^{\mathrm{PRG}}$

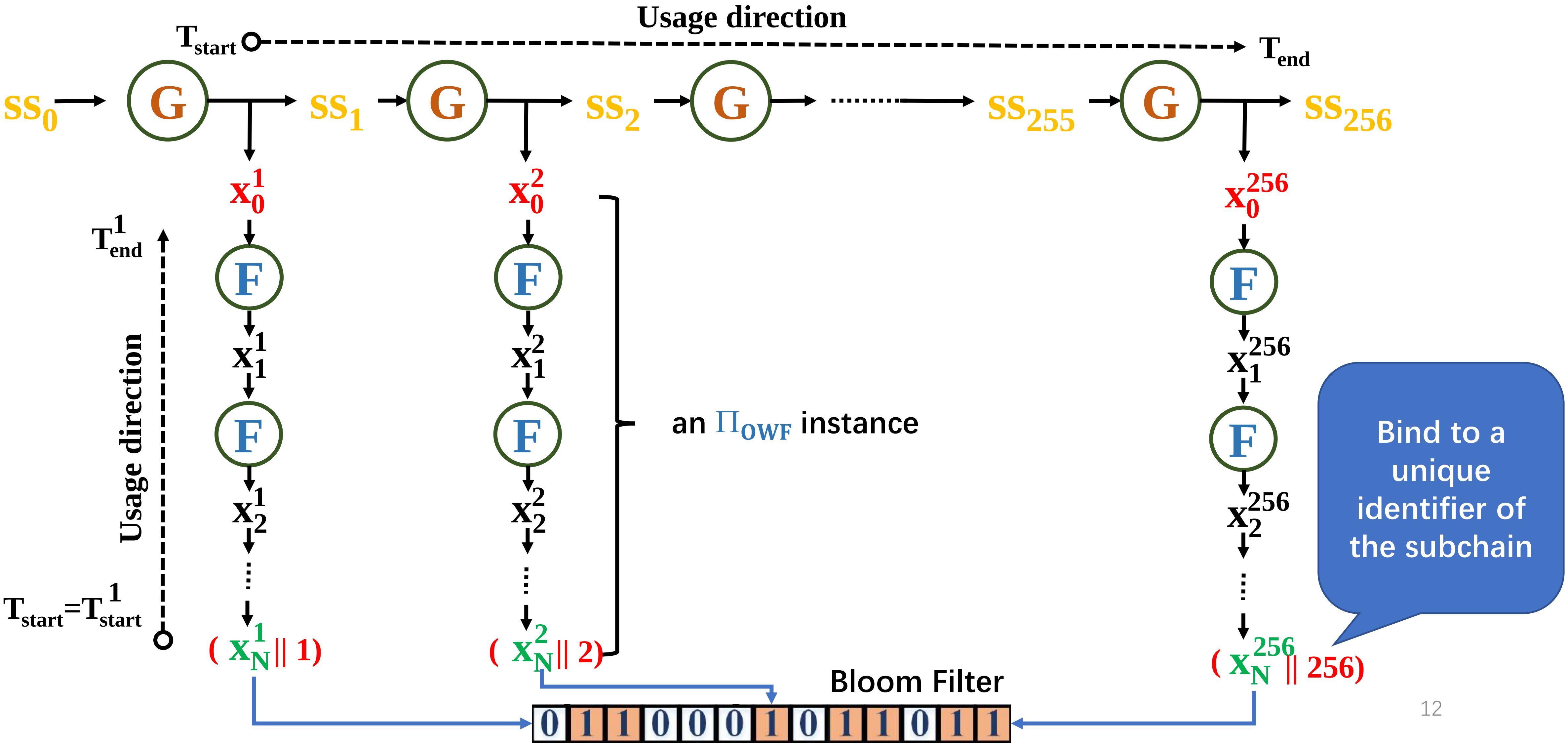


G: Pseudorandom number generator

F: One-way function

OTS: Lamport One-Time Signature [Lam79] – based on one-way functions

PoA Π_{OWF}^{PRG} - BF

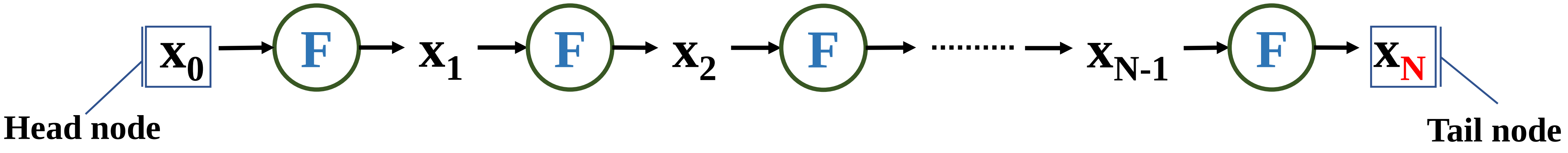


Commitment-Based Replenishment

- OTS is secure forever, but can we use something weaker and more efficient?
  - Yes. Hash-based commitment scheme, only secure before the commitment is open.

1. When sending $\mathbf{X_1}$, the prover also sends **H($\mathbf{X_0}$, New_Instance)** to the verifier. **H** is a collision resistant hash function.
2. In the end of the life time of the chain, the prover sends $\mathbf{X_0}$ and **New_Instance** to the verifier
3. The verifier verifies $\mathbf{X_0}$ with the known info and then verifies New Instance with **H($\mathbf{X_0}$, New_Instance)** received previously.

- This replenishment also works on multi-chain structures



Optimal Caching Strategy

- Consider a memory sufficient device (more discussion on memory insufficient devices in the paper)
- A memory efficient implementation that minimizes the proof generation time: one F call per proof generation
- Break an N-node chain into $\sqrt{N}$ segments of $\sqrt{N}$ nodes.
- Memory requirement: 2 $\sqrt{N}$ nodes: $\sqrt{N}$ checkpoints and $\sqrt{N}$ cached nodes
- When the i-th segment is being used in the reverse order, the (i-1)-th segment is being computed in the forward order from its checkpoint and overwrite the proof just used.

Caching Example

Suppose N = 100. Then we need $2\sqrt{N} = 20$ node storage.

Checkpoints

X_0	X_{10}	X_{20}	X_{30}	X_{40}	X_{50}	X_{60}	X_{70}	X_{80}	X_{90}

One Segment

X_{90}	X_{91}	X_{92}	X_{93}	X_{94}	X_{95}	X_{96}	X_{97}	X_{98}	X_{99}
X_{90}	X_{91}	X_{92}	X_{93}	X_{94}	X_{95}	X_{96}	X_{97}	X_{98}	X_{80}
X_{90}	X_{91}	X_{92}	X_{93}	X_{94}	X_{95}	X_{96}	X_{97}	X_{81}	X_{80}
X_{90}	X_{91}	X_{92}	X_{93}	X_{94}	X_{95}	X_{96}	X_{82}	X_{81}	X_{80}
X_{90}	X_{91}	X_{92}	X_{93}	X_{94}	X_{95}	X_{83}	X_{82}	X_{81}	X_{80}

Performance evaluation

- Client – Raspberry Pi 3, server – laptop , N=2^{22} (4 million)
- Random oracle (RO), Hash – SHA256, PRG – AES-CTR
- Standard model (STD), OWF – Subset-sum, PRG – [YLW13]

Protocol	Setup	Proof Generation average/worst	Verification	Replenishment
Π_{OWF} STD	185.33 s	44.19 μs / 44.19 μs	4.12 μs	N/A
Π_{OWF} RO - C	15.69 s	3.74 μs / 3.74 μs	0.47 μs	11.22 μs
$\Pi_{\mathrm{OWF}}^{\mathrm{PRG}}$-BF RO	17.11 s	5.50 μs / 18.00 μs	0.47 μs	2.65 ms
$\Pi_{\mathrm{OWF}}^{\mathrm{PRG}}$-BF STD	192.48 s	45.5 μs / 10.46 ms	4.12 μs	5.28 s

Best on a memory sufficient prover

Summary

- Cryptographic notion of Proof of Alievness
    - Security model (not detailed in the talk)
    - New security bounds in the standard model (not discussed in the talk)
- Optimized PoA constructions and implementations
    - Reduce the overall chain size: auto-replenishment
    - Minimize the proof generation time: optimal caching strategy
    - Reduce the server storage: Bloom filter
    - Reduce the replenishment time: commitment scheme
- Performance evaluation on Raspberry Pi.