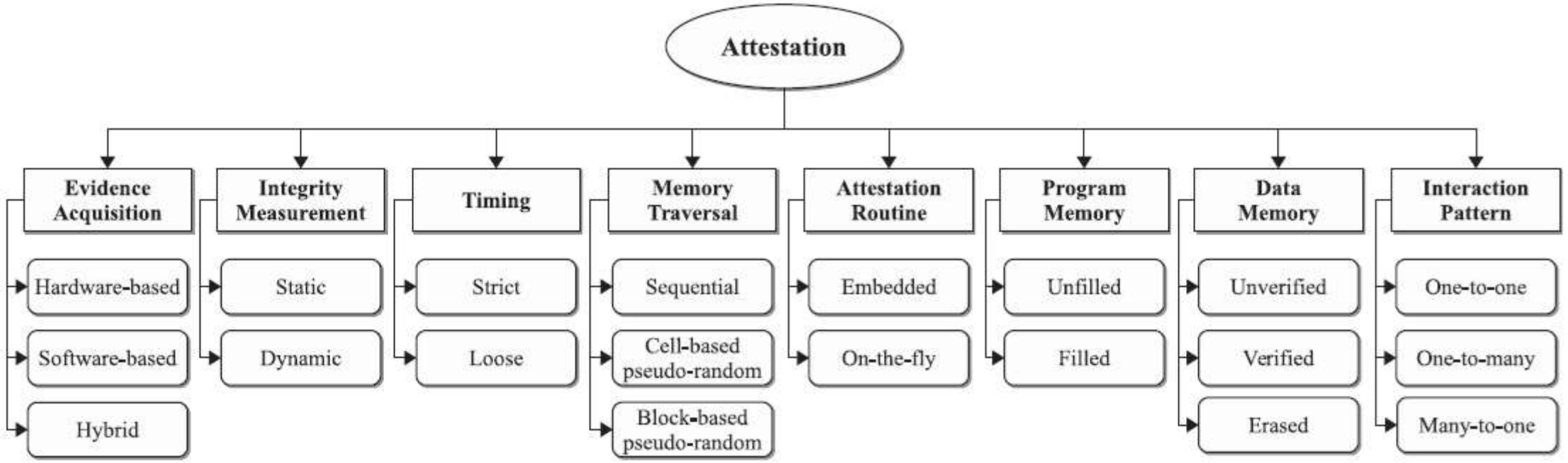
Taxonomy for Attestation



Evidence Acquisition Approaches

Approaches	Hardware	Software	Hybrid
Description	Relies on tamper-resistant hardware like Trusted Platform Module (TPM) or Physical Unclonable Functions (PUFs) to secure preloaded secrets and provide root of trust.	No secure hardware required. Prover executes attestation routine producing allegedly unforgeable results through careful design and timing constraints.	Doesn't depend on tamper-resistant hardware but requires specific hardware like ROM to store attestation routines, preventing adversary modification.
Advantages	Provides root of trust, reliable information and services.	Works on legacy devices, no additional cost or size increase.	Balances security and practicality, prevents routine modification.
Disadvantages	Cannot use on legacy devices, increases cost and energy consumption, vulnerable to side-channel attacks.	Difficult to design correctly, debated feasibility, vulnerable to various attacks.	Requires sufficient ROM space, not applicable to all legacy devices.

Integrity Measurement

Static Measurement

Verifies only the static part of memory whose contents never change during normal software execution. Provides straightforward verification but vulnerable to Return-Oriented Programming attacks that use existing code without modification.

Dynamic Measurement

Checks runtime properties representing behavior that must be satisfied during normal program execution. Addresses ROP vulnerabilities but faces challenges in identifying all valid states of dynamic objects.

Static memory regions provide a straightforward verification path—the Verifier challenges the Prover to calculate a checksum over these regions. However, verifying that code is being executed as intended is equally important, which is the aim of dynamic integrity measurement approaches.

Timing Constraints

Strict Timing

Relies on accurate measurements of attestation routine execution time. Assumes time-optimal implementation where any modifications result in detectable delays. Does not rely on tamper-resistant hardware but requires proving runtime optimality—an open research problem.

Loose Timing

Does not depend on precise execution time measurements. Either relies on tamper-resistant hardware or makes assumptions limiting adversary capabilities, such as filling all memory to prevent malware storage.

Theoretically, larger time limits allow more attacks. However, strict timing approaches face challenges: difficult to design time-optimal routines, error-prone assembly implementation, and inability to perform multi-hop attestation due to unpredictable network latency.

Memory Traversal Strategies

01

Sequential

Iterates through memory from start to end. Simple, efficient, provides complete coverage but predictable—adversaries can move malware around to avoid detection.

02

Cell-Based Pseudo-Random

Accesses memory addresses in unpredictable order one at a time. Unpredictable but requires O(n log n) memory reads for probabilistic coverage, with some addresses accessed multiple times.

03

Block-Based Pseudo-Random

Accesses blocks of addresses with XOR operations within blocks. Reduces overhead to O((n log n)/b) iterations while maintaining unpredictability. Block size b represents security-performance tradeoff.

Attestation Routine

- **Embedded Routines**: Most approaches embed the attestation routine in the Prover's memory prior to network deployment. These provide security by design—the routine is conceived to be secure such that modified memory or routines result in invalid responses with high probability.
- **On-the-Fly Generation**: The Verifier generates and sends different routines for each attestation round. Provides security through obscurity—since new routines are generated each time, attackers cannot predict instructions or outcomes, making vulnerabilities unlikely to be exploited timely.

Program Memory

The Problem

If a sensor node's program doesn't occupy entire program memory, empty spaces remain. Adversaries can exploit this space to store data used to overcome attestation.

The Solution

Some approaches fill empty space with incompressible random noise, preventing adversaries from having space to hide malicious code without overwriting original contents.

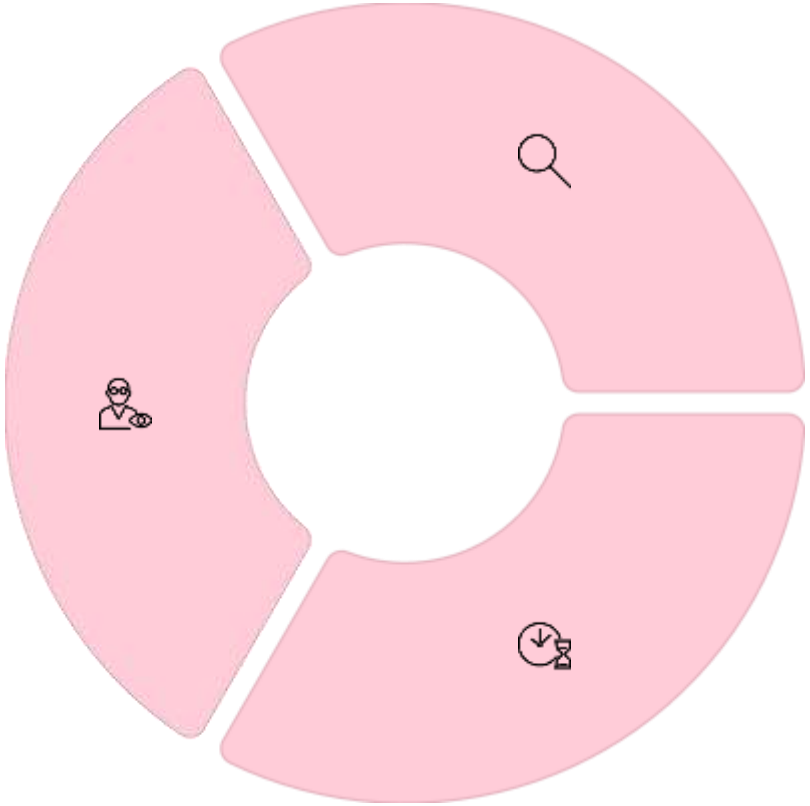
The Challenge

Physical memory contents are typically low entropy and compressible. Adversaries can still compress original program code to gain space for malicious code, even when empty spaces are filled.

Data Memory

Unverified

Some approaches don't verify data memory, particularly in Harvard architecture where data memory is smaller and non-executable. However, this leaves systems vulnerable to ROP attacks.



Verified

Dynamic attestation mechanisms attempt to verify data memory through data boundary integrity or other techniques. However, due to dynamic data behavior, only partial coverage is typically achieved.

Erased

Some approaches overwrite all data memory contents, eliminating any malicious data. While safer, this also destroys all legitimate data the node has collected prior to attestation.

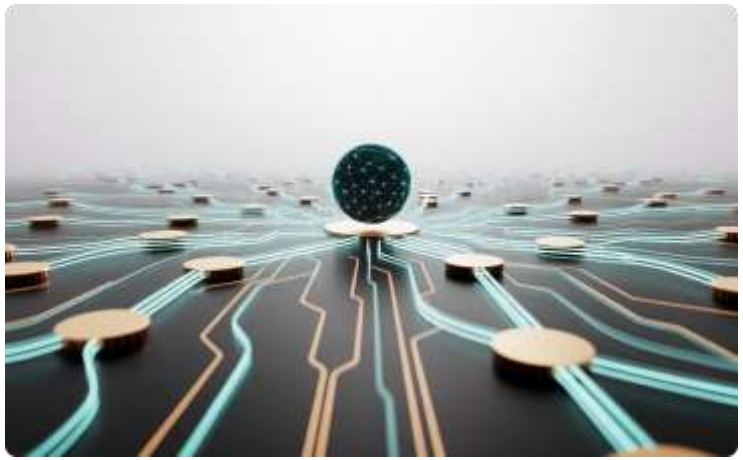
Interaction Patterns

One-to-One

Single verifier attests single prover per round. Simple but time increases linearly with number of nodes. Requires trusted verifier entity.

One-to-Many

Single verifier attests multiple provers simultaneously. Reduces overall computation time through parallelization but compromised node can discredit entire attestation chain.



Many-to-One

Multiple neighbors collaborate to attest single node. Enables distributed attestation without trusted verifiers but requires minimum network density and more vulnerable to compromised nodes.