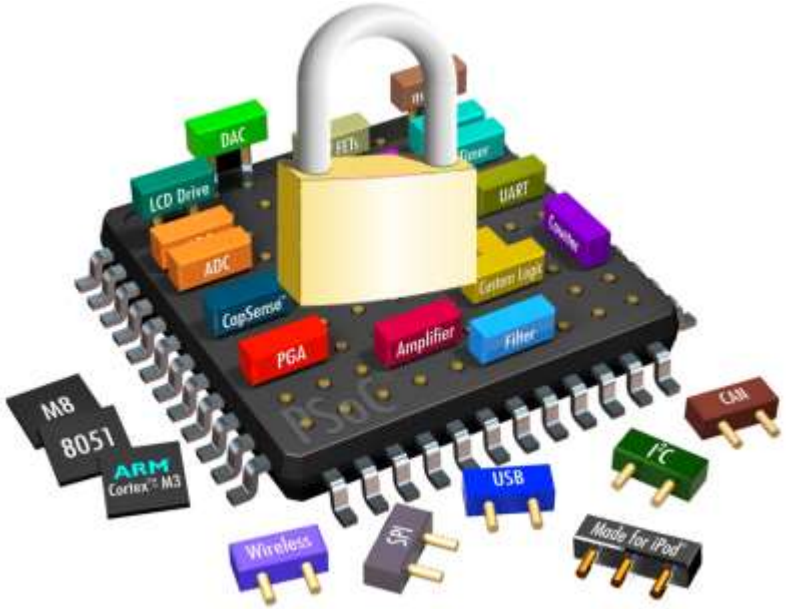
Hardware Security

Trusted Execution Environment



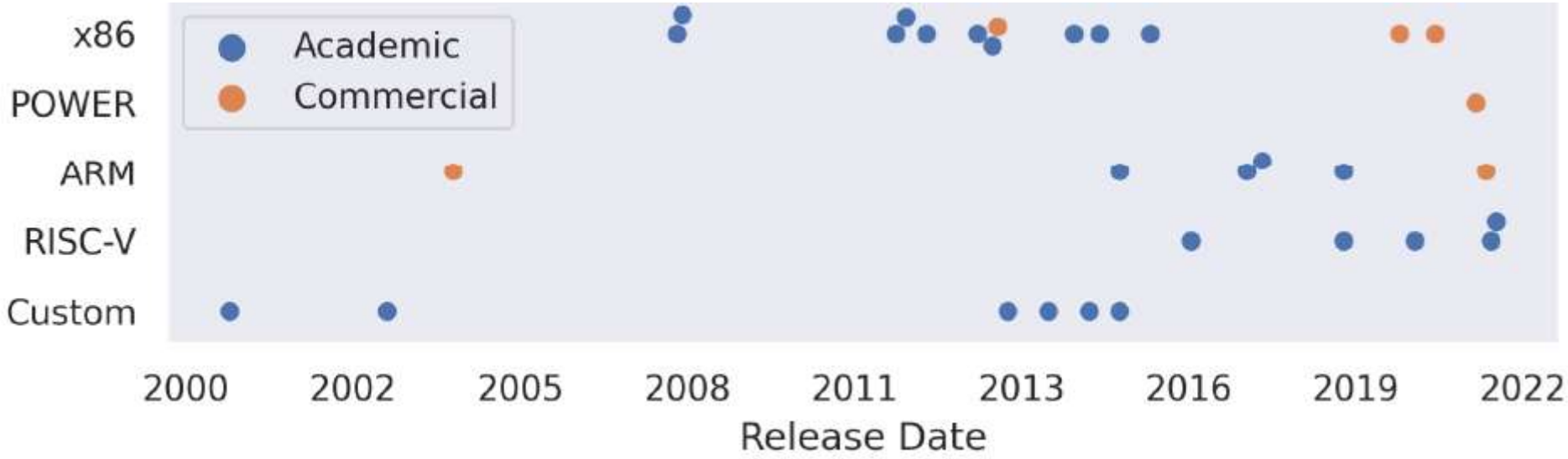
Trusted Execution Environment

- A Trusted Execution Environment (TEE) is a secure area within a processor that ensures confidentiality and integrity of code and data loaded into it. TEEs provide isolated execution of trusted software components, protecting them from the rest of the system.
  - **Verifiable launch** of the execution environment for the sensitive code and data so that a remote entity can ensure that it was set up correctly.
  - **Run-time isolation** to protect the confidentiality and integrity of sensitive code and data.
  - **Trusted IO** to enable secure access to peripherals and accelerators.
  - **Secure storage** for TEE data that must be stored persistently and made available only to authorized entities at a later point in time.

Why TEEs Matter

- Protect sensitive data against co-located attackers
- Enable secure computation in untrusted environments
- Support for multi-tenant computing platforms
- Reduce the trusted computing base (TCB)

Evolution of TEEs

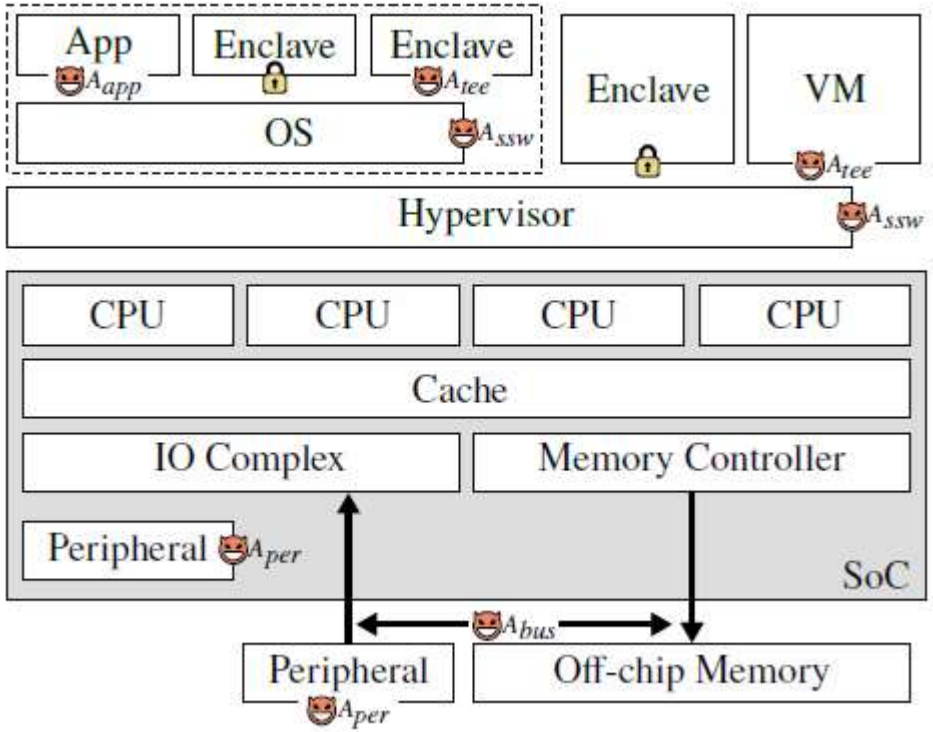


- The timeline shows the evolution of TEEs across different instruction set architectures.
- We observe a significant increase in TEE proposals in recent years, spanning x86, ARM, POWER, and RISC-V architectures, reflecting growing industry and academic interest in hardware security.

Terminology

1	2	3
Enclave	**TCB**	**TEE**
A TEE instance that provides an isolated execution environment (Intel calls these "enclaves," AMD uses "Secure Encrypted VMs," ARM CCA uses "Realms" or "trustlets")	Trusted Computing Base - all hardware and software components that must be trusted for the security of the system	The entire architecture that enables the creation and operation of enclaves

System Model

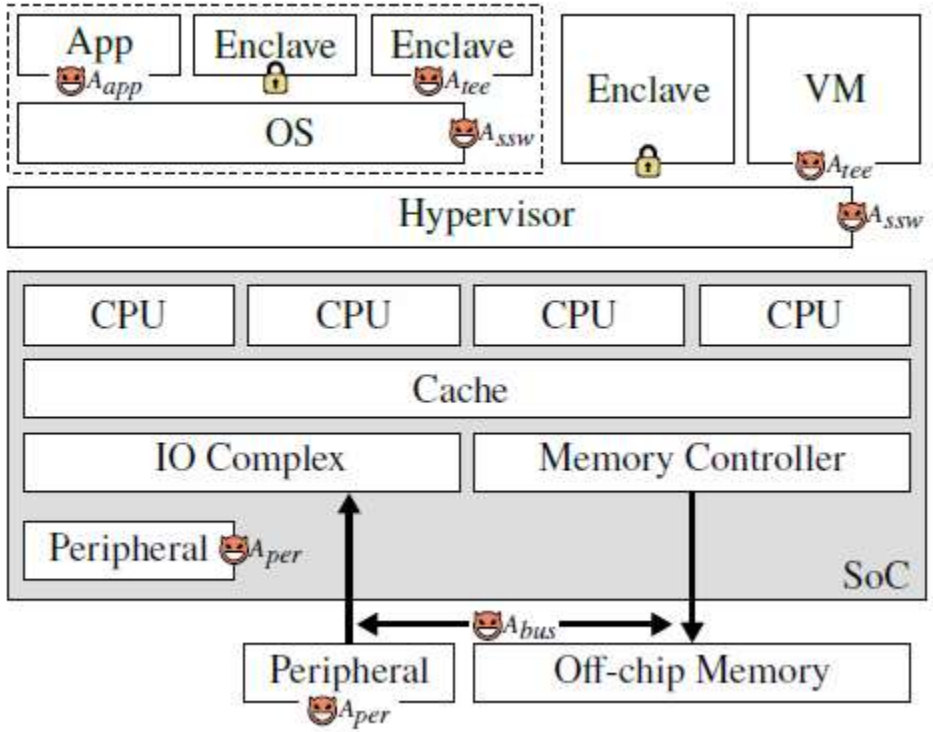


Most TEE solutions target a modern computing platform consisting of a System-on-Chip (SoC) with off-chip memory and peripherals. The SoC contains cores, caches, and fabric connecting to memory controllers and IO complex.

The software stack typically includes an OS and userspace applications, and in virtualized environments, a hypervisor with multiple VMs.

App App App App	Ring 3	EL0	U	PR	PL3
Guest OS Guest OS	Ring 0	EL1	S	OS	PL2
Hypervisor	Ring -1	EL2	H	Hyp	PL1
Firmware	Ring -2	EL3	M	-	PL0
	x86	*ARM*	*RISC-V*	*PowerPC*	*PL*

Adversary Model



- Co-located Enclave (A_{tee})
  - Adversary controlling other enclaves on the same platform, relevant in multi-tenant environments
- Unprivileged Software (A_{app})
  - Adversary running unprivileged software at the same privilege level as the victim enclave
- System Software (A_{ssw})
  - Adversary controlling system management software like OS or hypervisor
- Peripheral (A_{per})
  - Adversary controlling untrusted peripherals that can launch nefarious IO transactions
- Fabric (A_{bus})
  - Adversary with physical access to intercept off-chip communications
- Startup (A_{boot})
  - Adversary controlling system boot process and configuration

Verifiable Launch

Verifiable launch ensures that an enclave's execution environment is configured correctly and its initial state is as expected. This process involves:

Root of Trust

Establishing a trusted starting point (RTM) for the measurement process

Measurement

Creating a cryptographic fingerprint of the enclave's initial state

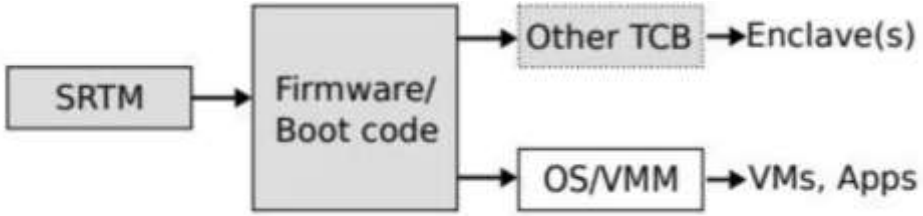
Attestation

Providing proof of the enclave's state to a verifier

This process establishes trust in the enclave before any sensitive operations are performed.

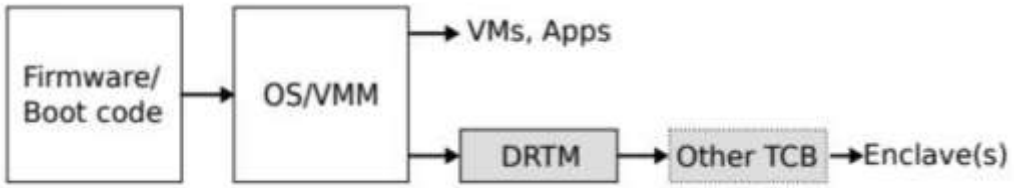
Root of Trust for Measurement (RTM)

Static RTM (SRTM)



Unbroken chain of trust from system reset to enclave execution

Dynamic RTM (DRTM)



Establishes a new RTM without trusting previously executed code

Most TEEs use SRTM. Only a few systems from Intel and AMD leverage DRTM, likely because the main motivation—excluding boot code from the TCB—only applies to platforms with legacy boot code.

Measurement Process

The measurement process creates a cryptographic fingerprint of the enclave's initial state:

Initial Measurement

Starting at the RTM, each component in the chain of trust measures the next component before transferring control

Integrity Checking

Components are not only measured but also integrity-checked (e.g., signature verification) before execution

Secure Storage

Measurements are stored securely in protected registers or memory regions to prevent tampering

TEEs use similar measurement techniques, typically involving cryptographic hashes over the initial memory of the enclave.

Attestation

Local Attestation

- For verifiers co-located on the same platform
- Typically uses symmetric cryptography
- More efficient than remote attestation
- Limited to intra-platform verification

Remote Attestation

- For verifiers not on the same platform
- Uses asymmetric cryptography
- Often involves certificate chain verification
- Supported by most TEE solutions

Attestation allows a verifier to check that the enclave has been launched correctly and its initial state matches expected reference values.

	Name	ISA	RTM	Attestation	
				Local	Remote
Industry	Intel SGX [7], [20]	x86	DRTM	●	●
	Intel TDX [8]	x86	DRTM	●	●
	AMD SEV-SNP [3]	x86	SRTM	◔	●
	ARM TZ [21]	ARM	SRTM	◔	◔
	ARM Realms [9]	ARM	SRTM	◔	●
	IBM PEF [22]	POWER	SRTM	●	●
Academia	Flicker [5]	x86	DRTM	◔	●
	SEA [6]	x86	DRTM	◔	●
	SICE [23]	x86	SRTM	◔	●
	PodArch [24]	x86	SRTM	◔	◔
	HyperCoffer [25]	x86	SRTM	◔	◔
	H-SVM [26], [27]	x86	SRTM	◔	◔
	EqualVisor [18]	x86	SRTM	◔	◔
	xu-cc15 [28]	x86	SRTM	◔	◔
	wen-cf13 [29]	x86	SRTM	◔	◔
	Komodo [30]	ARM	SRTM	◔	●
	SANCTUARY [31]	ARM	SRTM	●	●
	TrustICE [32]	ARM	SRTM	◔	◔
	HA-VMSI [33]	ARM	SRTM	◔	●
	Sanctum [4]	RISC-V	SRTM	●	●
	TIMBER-V [34]	RISC-V	SRTM	●	●
	Keystone [35]	RISC-V	SRTM	◔	●
	Penglai [36]	RISC-V	SRTM	◔	●
	CURE [37]	RISC-V	SRTM	◔	●
	Iso-X [38]	OpenRISC	SRTM	◔	●
	HyperWall [39]	SPARC	SRTM	◔	●
	Sancus [40], [41]	MSP430	HW	◔	●
	TrustLite [42]	Custom	SRTM	●	●
	TyTan [43]	Custom	SRTM	●	●
	XOM [44]	Custom	SRTM	◔	◔
	AEGIS [45]	Custom	SRTM	◔	◔

Attestation Report Contents

Basic Information

- Enclave measurements
- TCB measurements
- Nonce for freshness

Extended Information

- Runtime attributes (e.g., SMT enabled/disabled)
- Software version numbers
- Migration policies
- TCB version information

Custom Data

- Public key certificates
- Channel establishment data
- Application-specific information

Modern TEEs include more comprehensive information in attestation reports compared to early designs that only included measurements and nonces.

Secret Provisioning

Provisioning secrets into enclaves is often the final step during launch. Two main approaches exist:

Pre-Attestation Provisioning

Some TEEs allow enclaves to be provisioned with secret data prior to attestation:

- Secret data included in initial enclave state
- Reflected in the measurement
- Encrypted before delivery to the platform
- Examples: IBM PEF, AMD SEV-SNP, PodArch

Post-Attestation Provisioning

Other TEEs require attestation before any secret data can be provisioned:

- Enclave appends custom data to attestation report
- Data authenticated during attestation
- Used to establish secure communication channel
- Based on key exchange protocols

Entities of Attestation Protocols

Verifier (V)

The trusted entity that validates integrity. Often the network's base station, though it can be any wireless sensor node with potentially more computational power than regular nodes.

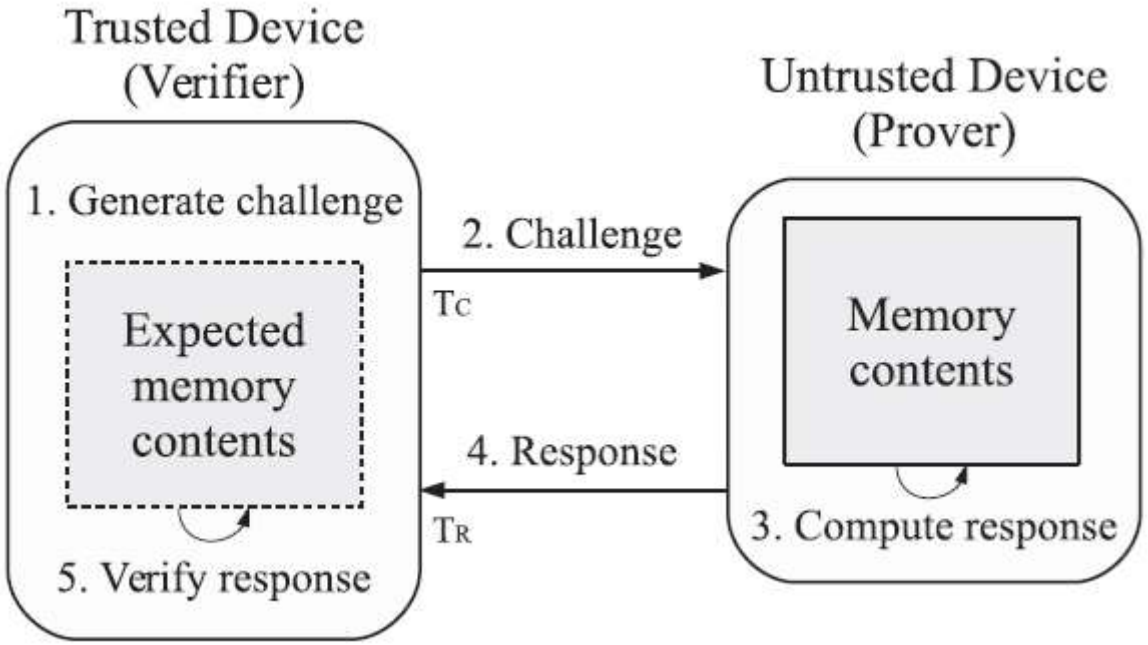
Prover (P)

The device being verified. Has an internal state reflecting its memory contents including program memory, data memory, registers, MMIO, and potentially external memories.

Adversary (A)

The attacker launching attacks remotely or using an already-compromised network node. Aims to compromise nodes without detection by the attestation procedure.

Attestation Overview



$$
\begin{aligned}
& V : c \xleftarrow{R} Challenge() \\
V \rightarrow P & : c \\
& V : T_C \leftarrow current\ time \\
& P : r \leftarrow Attest(S, c) \\
P \rightarrow V & : r \\
& V : T_R \leftarrow current\ time \\
& V : Verify(S, c, r, T_A, T_C, T_R)
\end{aligned}
$$

Key Assumptions

- About the Verifier
  - Cannot be compromised by the attacker
  - Knows the expected state of the Prover
  - Knows the hardware architecture of the Prover
- About the Adversary
  - Can reverse engineer Prover's software and hardware
  - Has full control of Prover's memory
  - Cannot modify Prover's hardware

Requirements

Authenticity

Allows the Verifier to confirm the source of the response

Atomicity

Guarantees uninterrupted execution, preventing memory modification or parallel computation

Unforgeability

Prevents adversaries from producing the same response faster than ATTEST

Dynamicity

Reflects the actual running system, not just static memory

Determinism

Enables the Verifier to reach the same result independently

Target

- Code integrity
  - The integrity of the code is verified
  - The computation result could be distorted by data manipulation
- Code + data integrity
  - The data regions are initialized to known values and verified
  - It is hard to verify the integrity during run-time
- Hardware-based attestation
  - The hardware measures the integrity of the code and data
  - The integrity metric is typically the hash value of the memory signed by the hardware
  - The run-time integrity is guaranteed by isolation

Common Attacks: Computational Exploits

1

Precomputation

Adversary precomputes operations not influenced by the challenge, gaining time for other operations. If challenges are predictable, valid responses can be precomputed entirely.

2

Forgery

Adversary attempts to generate valid responses despite memory modifications by executing modified attestation routines or altering memory so modifications neutralize each other during computation.

3

Return-Oriented Programming

Uses existing code without alteration to execute malicious operations by linking together small instruction sequences called "gadgets," circumventing defenses that assume code modification.

Common Attacks: Memory Manipulation

Memory Copy

1

If sufficient free space exists, adversary stores malicious data while keeping a copy of original contents, modifying the attestation routine to compute responses over the original copy location

Data Substitution

2

Special case when insufficient space exists for full copy—adversary modifies part of original contents and keeps copy only of overwritten data, redirecting memory reads during attestation

Compression

3

Adversary compresses original contents to gain space for malicious data, then decompresses on the fly during attestation to compute expected responses

Common Attacks: Network-Based Threats

Replay

Eavesdrop valid attestation responses from non-compromised nodes and retransmit when challenged. Only works if nodes execute the same program and receive identical challenges.

Collusion

Compromised nodes collaborate to compute valid responses, exchanging messages to recover original memory contents or dividing attestation operations across multiple devices.

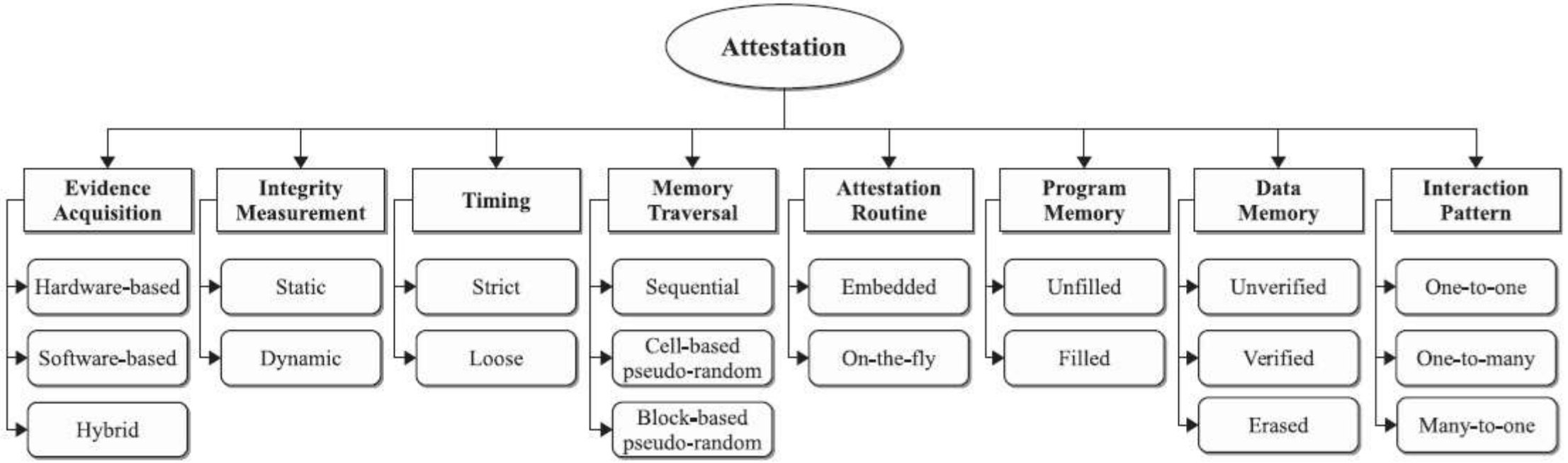
Impersonation

Adversary takes multiple identities (Sybil attack), masquerading as genuine nodes during attestation or impersonating the base station to forward challenges.

Proxy

Special collusion case using a device with better computing capabilities that keeps copies of original memory contents and computes valid responses faster than common nodes.

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.

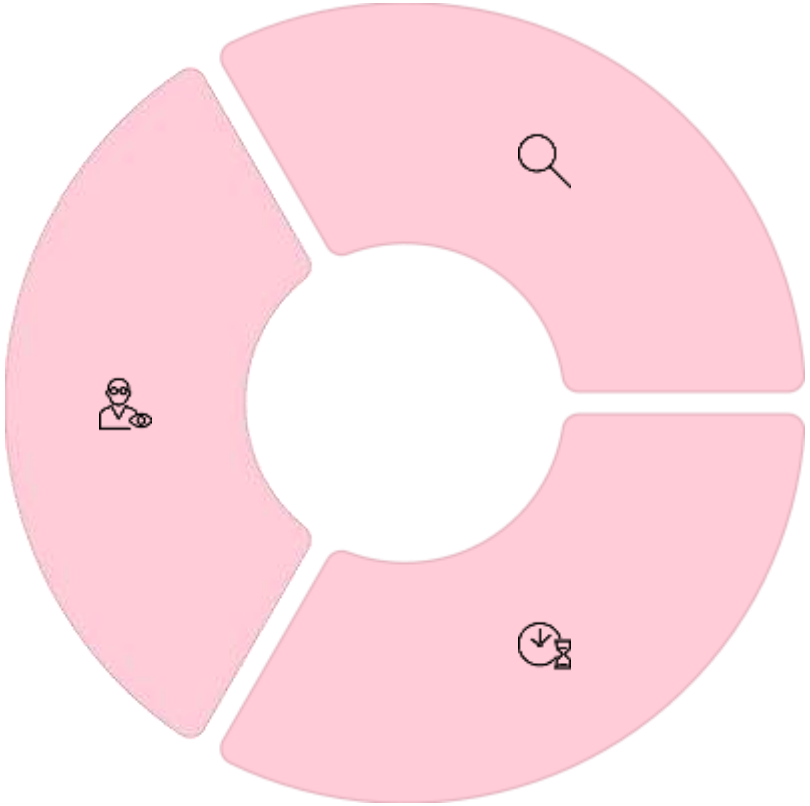
The Challenge

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.

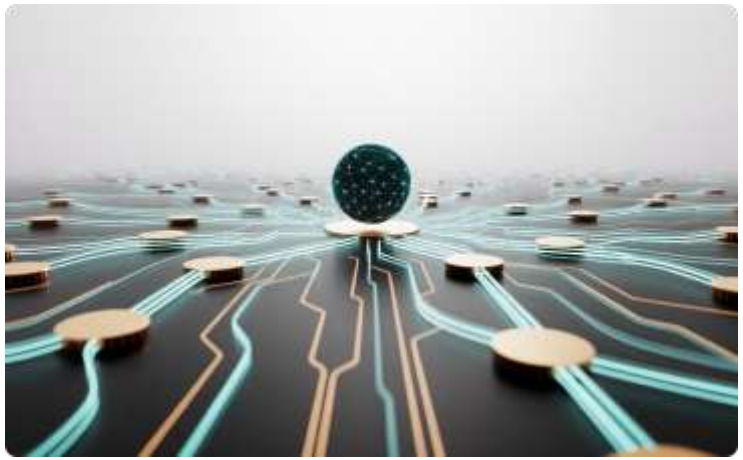
Interaction Patterns

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.



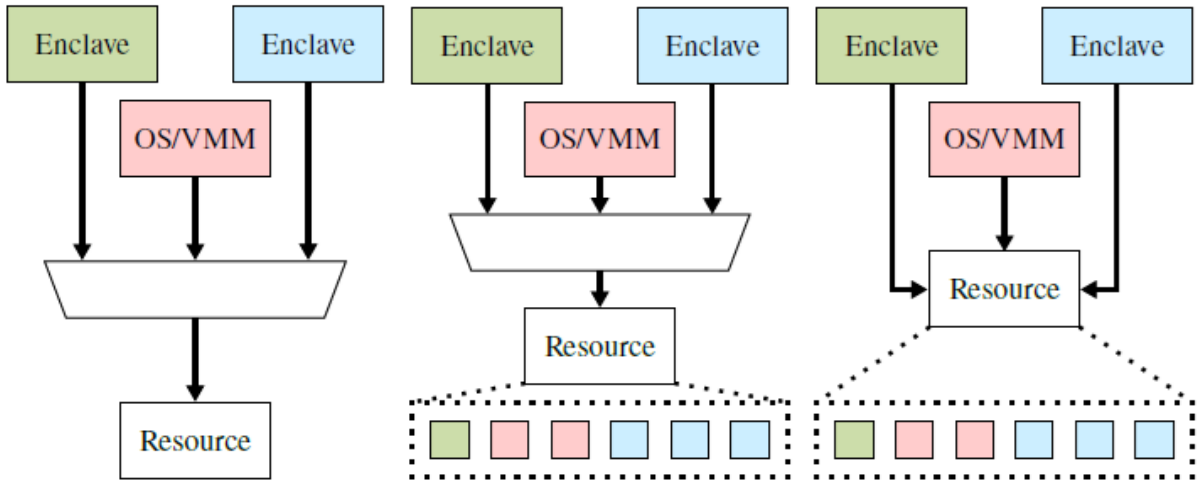
Many-to-One

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

Run-time Isolation

- Run-time isolation protects the confidentiality and integrity of sensitive computations and data during enclave execution.
- This requires isolating both CPU and memory resources.
- Isolation strategies
  - **Resource Partitioning**:
    How resources are divided (temporal, spatial, or spatio-temporal)
  - **Isolation Enforcement**:
    How isolation is enforced (logical or cryptographic)

Resource Partitioning Strategies



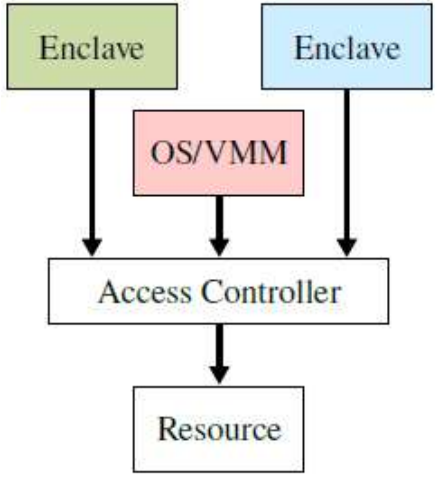
Temporal Partitioning

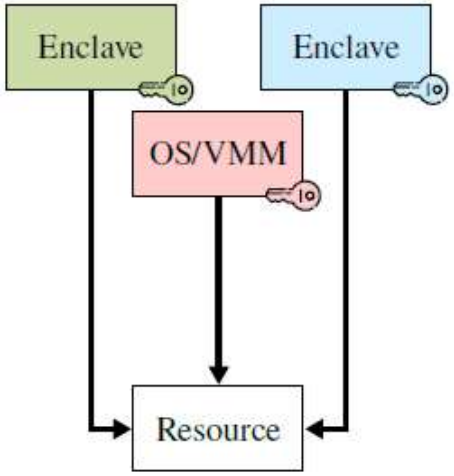
Spatio-temporal Partitioning

Spatial Partitioning

- Temporal
  - Securely multiplexes the same resource among multiple execution contexts over time.
  - At any point, a single context has exclusive access.
- Spatio-temporal
  - Leverages both temporal and spatial aspects.
  - Resources can be spatially partitioned but these partitions may change over time.
- Spatial
  - Resources are split so trusted and untrusted contexts use separate, dedicated partitions.
  - Enables concurrent access without interference.

Isolation Enforcement Strategies





- Logical Isolation
  - Uses access control mechanisms to prohibit unauthorized access
  - Intercepts data accesses and checks against access control information
  - Access control information must be protected
  - Efficient for software adversaries

- Cryptographic Isolation
  - Uses encryption for confidentiality
  - Uses MACs for integrity protection
  - Requires anti-replay schemes for complete integrity
  - Effective against physical adversaries

CPU Isolation

Existing TEEs use temporal partitioning with logical enforcement for CPU state isolation. This approach:

Implementation

Secure context switch routine that saves, purges, and restores execution contexts

Requirements

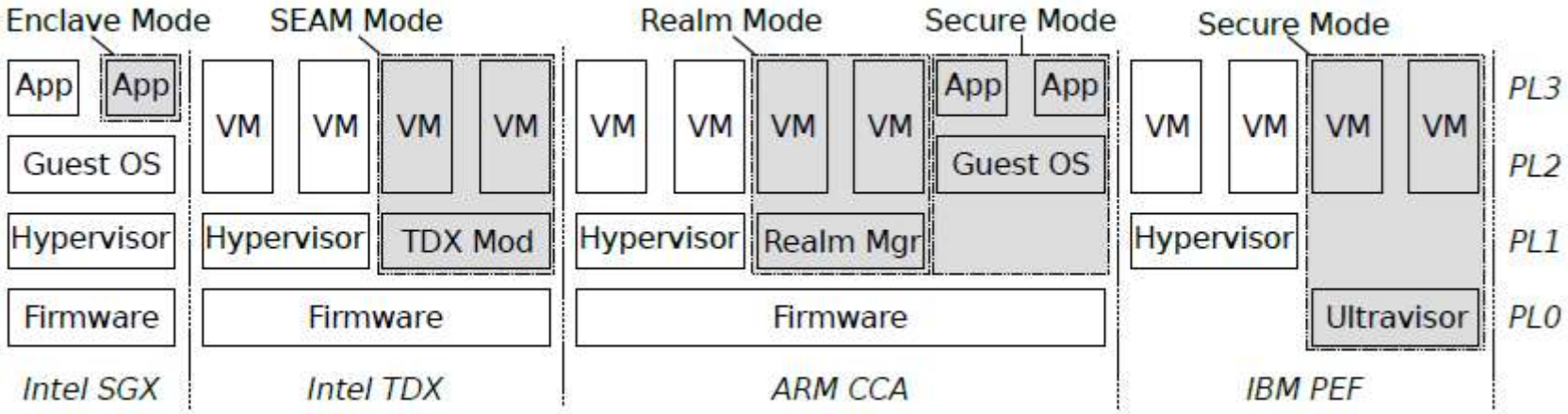
TCB must fully mediate every context switch using CPU modes and privilege levels

Advantages

Minimal performance overhead compared to other approaches

Other approaches like spatial partitioning (dedicating cores to enclaves) or cryptographic enforcement have significant downsides in terms of resource utilization or performance.

CPU Modes

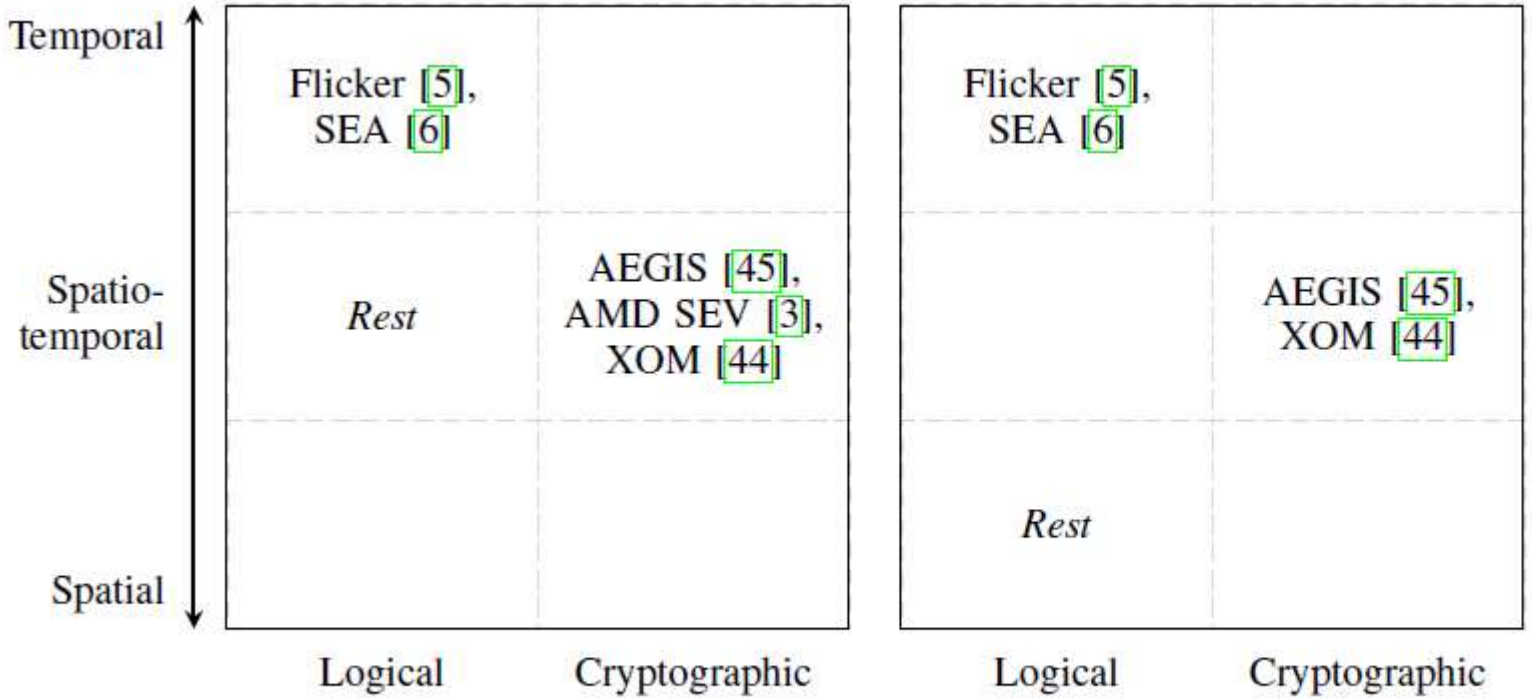


- Commercial processors often add new execution modes to support TEEs.
- In contrast, most academic TEEs rely on existing privilege levels and firmware for secure context switching.

Memory Isolation

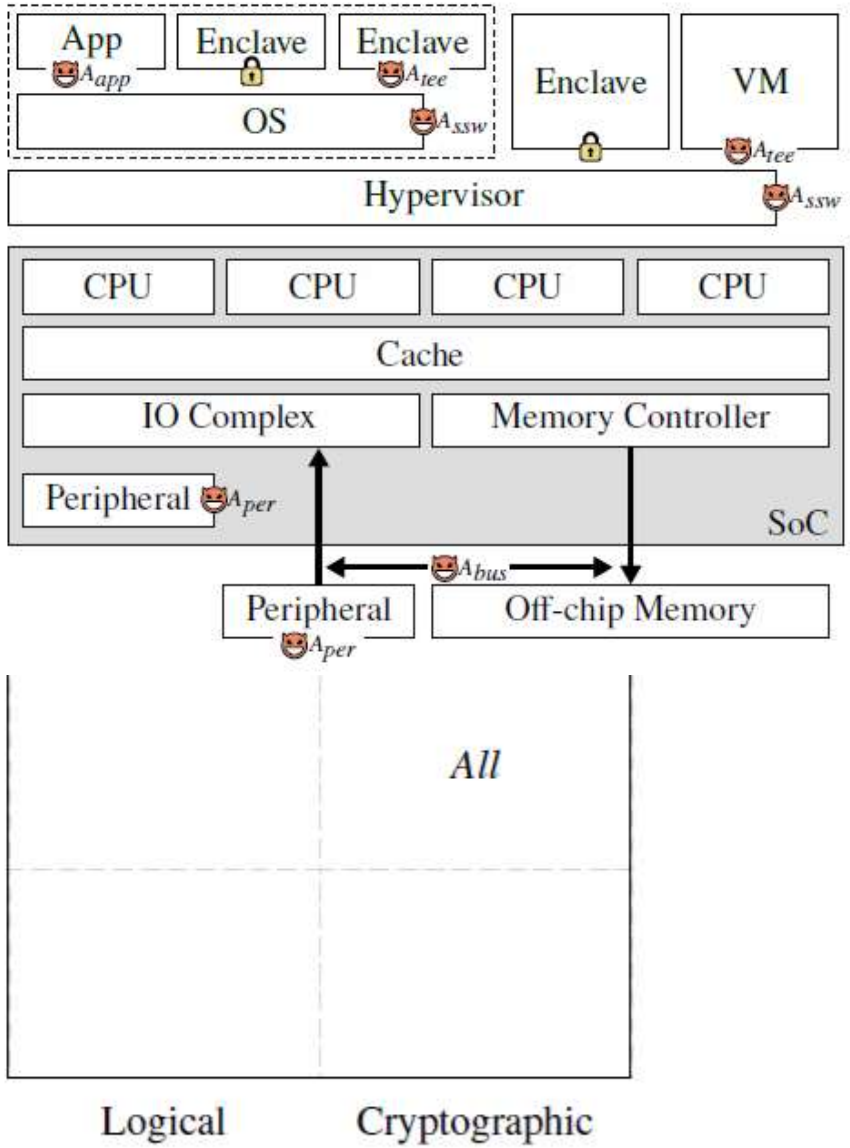
- Memory isolation is particularly challenging as it must protect:
  - Off-chip memory
  - On-chip microarchitectural structures (caches)
  - Translation structures (page tables)
  - Translation look-aside buffers (TLBs)
- Unlike CPU isolation where all TEEs use the same approach, memory isolation strategies are diverse.
  - Many TEEs use different strategies based on the type of attacker being considered.

Memory Isolation



(a) Enclave memory isolation against A_{app}, A_{tee}, and A_{ssw}.

(b) TCB memory isolation against A_{app}, A_{tee}, and A_{ssw}.



(c) Memory isolation strategy against A_{bus}.

Memory Protection Mechanisms

Memory Protection Unit (MPU)

- Checks physical address and access type against access control information
- Supports limited number of memory regions
- Suitable for coarse-grained protection
- Used in many academic TEEs

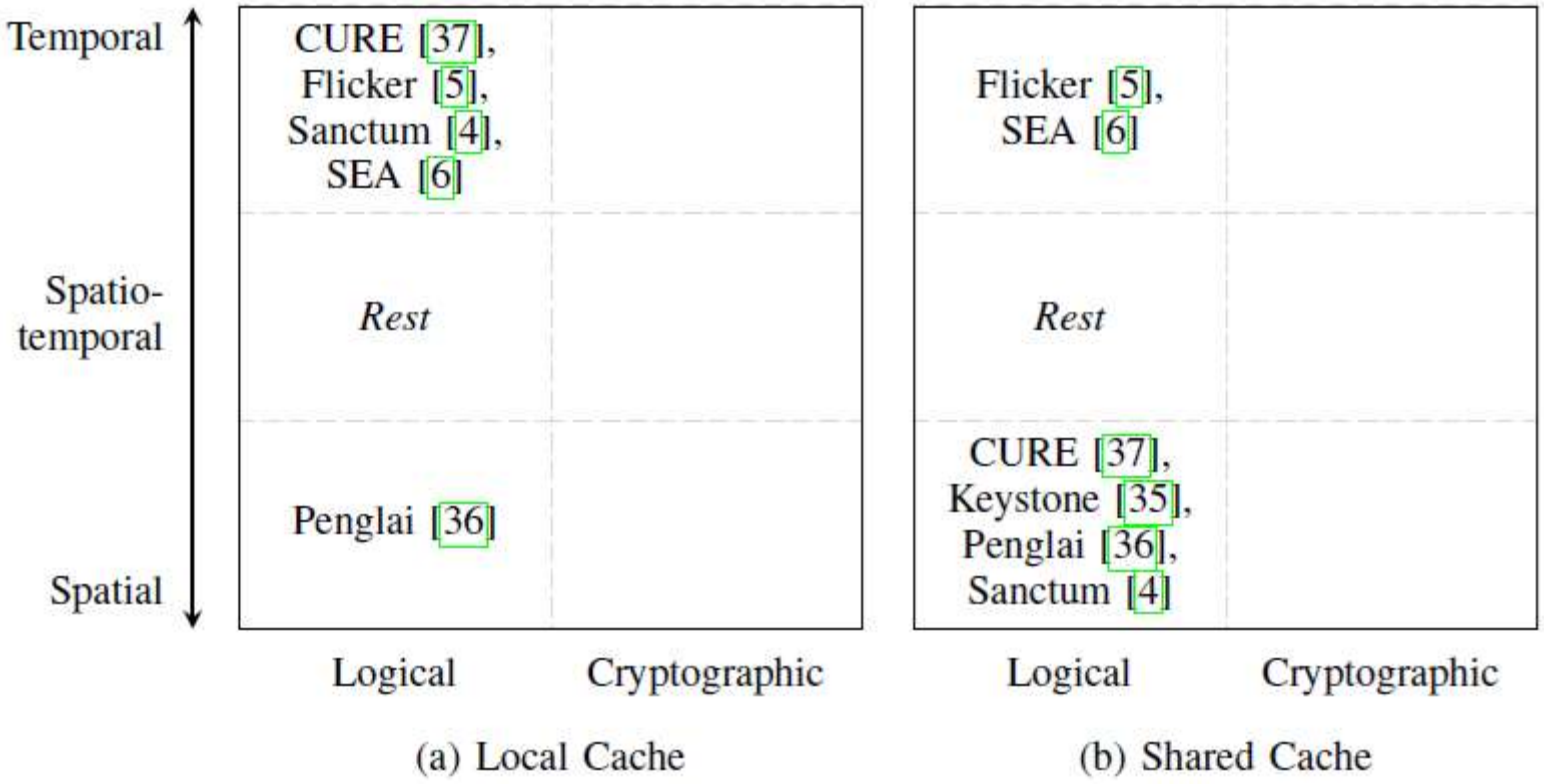
Memory Management Unit (MMU)

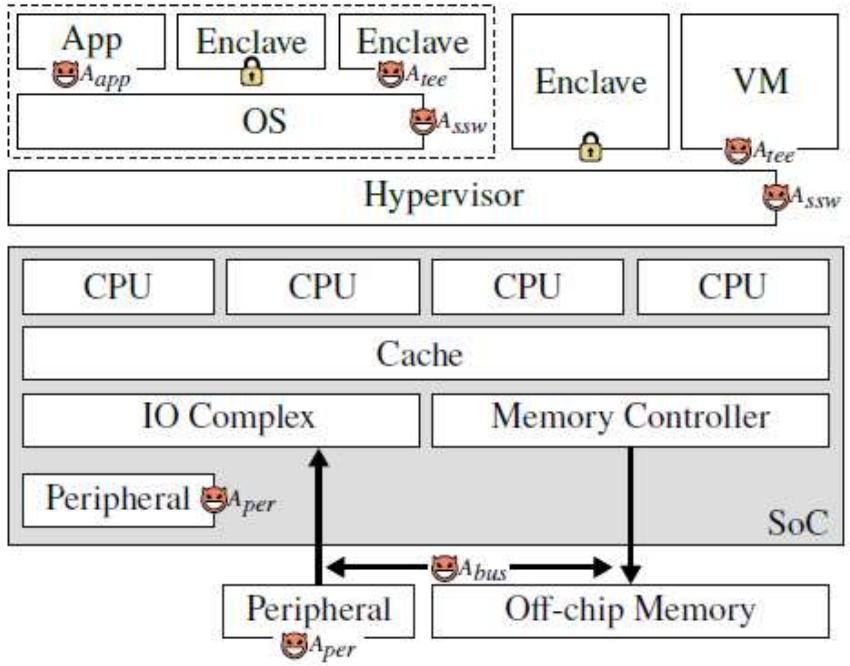
- Converts virtual addresses to physical addresses using page tables
- Stores permissions alongside mappings
- Offers fine-grained protection
- Used in many commercial TEEs

The access control information used by MPUs and MMUs must itself be protected against unauthorized access, typically through spatial partitioning.

Cache Isolation

Against software adversaries (A_{app}, A_{tee}, and A_{ssw}).





Translation Look-aside Buffers (TLBs)

TLBs hold recent virtual-to-physical address translations and must be protected to prevent leakage:

Spatial Partitioning

Dedicated TLBs per logical processor provide inherent spatial partitioning

Temporal Partitioning

TLBs are flushed on context switches to prevent reuse of stale translations

Optimization

Modern processors support partial TLB flushes using context identifiers

Cryptographic isolation is not used for TLBs due to performance overheads.

Trusted IO

- Trusted IO enables secure interactions between enclaves and external devices.
- Trusted path
  - Ensures confidentiality and integrity for the enclave's accesses to the device.
- Trusted device architecture
  - Protects enclave data on the device itself
  - Ensures device cannot leak sensitive data
  - Applies isolation principles to device resources
  - Particularly important for accelerators

Trusted Path Types

Logical Trusted Path

Uses access control mechanisms to allow/deny accesses based on origin or destination

- Access control filters for memory-mapped IO (MMIO)
- Trusted memory mappings

Cryptographic Trusted Path

Establishes a secure channel between enclave and device

- End-to-end encryption
- Authentication and attestation
- Can protect against physical attackers (Abus)

Some solutions combine both approaches, using different types for MMIO and DMA.

Trusted Device Architectures

For devices that process user data (e.g., accelerators), the device itself must protect data confidentiality and integrity.

Temporal Partitioning

Sharing device among multiple contexts over time with secure context switching

Spatio-temporal Partitioning

Multiple enclaves access device concurrently with hardware-enforced isolation

Cryptographic Protection

Applied to device-side memory resources similar to CPU DRAM protection

The isolation strategies used for CPU and memory can be applied to device resources as well.

Trusted IO Examples

GPU Protection

- Graviton: Hardware-enforced isolation
- Telekine: Cryptographic protection
- HIX: Logical path for MMIO, cryptographic for DMA
- ZeroKernel: Temporal isolation

FPGA Protection

- MeetGo: Secure remote applications
- ShEF: Shielded enclaves
- Trustore: Multi-tenant isolation

Secure Storage

Secure storage ensures that sensitive data persists across different enclave invocations and is only available to authorized entities.

Sealing

Process of encrypting data before storing it persistently

Unsealing

Process of decrypting data, accounting for enclave state

Binding Policies

Rules determining which enclaves can unseal previously sealed data

Only about a third of existing TEE solutions explicitly discuss sealing support, with most implementations resembling the original TPM-based approach.

Sealing Approaches

TPM-based Sealing

- Generate asymmetric key pair
- Encrypt data such that it can only be decrypted when system configuration matches
- Uses measurements in TPM's Platform Configuration Registers (PCRs)
- Examples: Flicker, SEA, IBM-PEF

Software TCB Sealing

- TCB exposes interface to create sealing keys
- No additional hardware required
- Examples: OP-TEE, Keystone, TIMBER-V, Sanctuary

Some TEEs like Intel SGX expose special CPU instructions in hardware to enable sealing, with different binding types (developer identity, enclave measurement).

Binding Policies for Sealed Data

Different TEEs implement various policies for determining which enclaves can unseal data:

Same Measurement

Only an enclave with identical measurement on the same platform can unseal the data

Same Developer

All enclaves signed by the same developer can unseal each other's data

Migration Support

Enclaves can come with a migration policy to transfer sealed data to a different host

The architecture must ensure that only the TCB and the owner enclave have access to the sealing keys.

TCB Composition

The Trusted Computing Base (TCB) includes all hardware and software components that must be trusted for security. TCB components can be:

Immutable Components

- Cannot be changed post-manufacture
- Typically implemented in hardware
- Examples: RTM, hardware-based isolation

Mutable Components

- Can be updated during platform lifetime
- Usually implemented in software
- Updates reflected in attestation reports
- Examples: measurement code, attestation

Most TEEs use a mix of mutable and immutable components in their TCB.

TCB Components by Function

Verifiable Launch

RTM is immutable in all TEEs, while measurement and attestation are typically mutable

Run-time Isolation

Most TEEs use mutable software TCB for CPU context switching and memory isolation

Trusted IO

Most TEEs with trusted IO rely on mutable software TCB

Secure Storage

Implemented either through TPM, software TCB, or hardware instructions

The ability to update TCB components is crucial for addressing vulnerabilities without product recalls.

TCB Size Considerations

TCB size is often measured in lines of code (LoC) for mutable components:

0-1K

Minimal TCBs

Academic proposals like Flicker, Sanctum, and Komodo focus on minimal TCB size

2-10K

Moderate TCBs

RISC-V TEEs like Keystone and Penglai have moderate TCB sizes

50K+

Commercial TCBs

ARM TrustZone and IBM PEF have larger TCBs with more features

TCB size alone is not a good metric for comparing TEEs, as it doesn't account for feature differences, implementation quality, or verification efforts.

Comparative Analysis: TCB Size

75K

IBM PEF software monitor size (lines of code)

50K

ARM TrustZone OP-TEE implementation (lines of code)

16.5K

Keystone secure boot code (lines of code)

5K

Sanctum security monitor (lines of code)

0.8K

Komodo security monitor (lines of code)

TCB size varies widely, with commercial implementations generally having larger codebases than academic proposals. However, TCB size alone is not a good metric for security or functionality.

Comparative Analysis: RTM and Attestation

Name	ISA	RTM	Local Attestation	Remote Attestation
Intel SGX	x86	DRTM	✓	✓
Intel TDX	x86	DRTM	✓	✓
AMD SEV-SNP	x86	SRTM	✓	✓
ARM TZ	ARM	SRTM	-	✓
ARM Realms	ARM	SRTM	-	✓
Keystone	RISC-V	SRTM	-	✓

Most TEEs use SRTM, with only Intel solutions using DRTM. Remote attestation is universally supported, while local attestation is less common.

Comparative Analysis: CPU Isolation

Name	Isolation Strategy	Enclave Type	Software TCB Level
Intel SGX	Temporal-Logical	Application	-
Intel TDX	Temporal-Logical	VM	PL1
AMD SEV-SNP	Temporal-Logical	VM	Co-processor
ARM TZ	Temporal-Logical	App/VM	PL0+(PL1/2)
ARM CCA	Temporal-Logical	VM	PL0+(PL1)
Sanctum	Temporal-Logical	Application	PL0

All TEEs use temporal-logical isolation for CPU state. Enclaves run as either applications (PL3) or VMs (PL2), while software TCB components run at higher privilege levels.

Comparative Analysis: Memory Isolation

Name	Software Adversaries	Physical Adversary	Access Control
Intel SGX	Spatio-temporal Logical	Cryptographic	Page Tables
Intel TDX	Spatio-temporal Logical	Cryptographic	Page Tables
AMD SEV-SNP	Spatio-temporal Cryptographic	Cryptographic	Page Tables
ARM TZ	Spatio-temporal Logical	Cryptographic (optional)	Page Tables
Keystone	Spatio-temporal Logical	-	MPU
AEGIS	Spatio-temporal Cryptographic	Cryptographic	Extra Metadata

Memory isolation strategies vary widely, with different approaches for software vs. physical adversaries. Commercial TEEs tend to use page tables, while academic TEEs often use MPUs.

Comparative Analysis: Trusted IO and Secure Storage

Name	Trusted IO	Secure Storage
Intel SGX	-	Hardware instructions
Intel TDX	-	-
AMD SEV-SNP	-	Primitives only
ARM TZ	TrustZone Protection Controller	Software TCB
ARM CCA	Realm Management Extension	-
CURE	IO filters	-
Keystone	-	Software TCB

Trusted IO and secure storage are less universally supported than other security goals. Only about a third of TEEs explicitly discuss sealing support.

Comparative Analysis: TCB Composition

Name	RTM	Measurement	Attestation	CPU Isolation	Memory Isolation
Intel SGX	Immutable	Mutable	Mutable	Mutable	Mutable + Immutable
Intel TDX	Immutable	Mutable	Mutable	Mutable	Mutable
AMD SEV-SNP	Immutable	Unknown	Mutable	Unknown	Unknown
Sanctum	Immutable	Mutable	Mutable	Mutable	Mutable
Iso-X	Immutable	Immutable	Immutable	Immutable	Immutable

Most TEEs use immutable RTM but mutable components for other security functions. Few designs like Iso-X implement everything in immutable hardware.