Assignment 2

Trusted Computing

TPM Architecture and TEE

Theory and Practical Implementation

Student Name: Yuvan Raj Krishna

Register Number: 22011102127

Course: Trusted Computing

Date: November 3, 2025

Contents

Part I
Theory Questions (10 marks)

1 Question 1: TPM Architecture and PCRs in Measured Boot

1.1 Introduction

The Trusted Platform Module (TPM) is a dedicated microcontroller designed to secure hardware through integrated cryptographic keys. As defined by the Trusted Computing Group (TCG), TPM 2.0 provides a hardware-based root of trust that enables secure generation of cryptographic keys, platform integrity measurements, and attestation capabilities. This section explores the internal architecture of TPM and the critical role of Platform Configuration Registers (PCRs) in implementing measured boot.

1.2 TPM 2.0 Internal Architecture

The TPM architecture comprises several key components working in concert to provide hardware-based security:

1.2.1 Root of Trust Components

Core Root of Trust for Measurement (CRTM) The CRTM represents the first code executed during platform boot and serves as the immutable trust anchor. Located in the BIOS Boot Block, it measures the BIOS firmware before execution and extends the measurement into PCR-0. The integrity of all subsequent measurements depends on the CRTM remaining uncompromised.

Root of Trust for Storage (RTS) The RTS is anchored by the Storage Root Key (SRK), a 2048-bit RSA key generated within the TPM and never exposed outside the chip. The SRK forms the root of a key hierarchy where child keys are encrypted by parent keys, creating a cryptographic chain. This hierarchy enables secure key storage with hardware binding, key migration policies, and delegation of authorization.

Root of Trust for Reporting (RTR) The RTR enables remote attestation through Attestation Identity Keys (AIK). Unlike the unique Endorsement Key (EK), multiple AIKs can be created to provide privacy-preserving attestation. AIKs sign attestation quotes containing PCR values, allowing remote verifiers to assess platform integrity without compromising user privacy.

1.2.2 Cryptographic Engine

The TPM includes a dedicated cryptographic processor implementing:

- **Random Number Generator**: Hardware RNG compliant with NIST SP 800-90A, providing entropy for cryptographic operations

- **Asymmetric Cryptography**: RSA (1024/2048 bits) and ECC (P-256) for key generation, signing, and encryption
- **Symmetric Cryptography**: AES-128/256 for efficient encryption/decryption
- **Hash Functions**: SHA-1 (legacy), SHA-256, SHA-384, SHA-512 for measurements and integrity verification

1.2.3 Memory Hierarchy

Non-Volatile Memory (2-8 KB) Persistent storage containing the Endorsement Key (EK) burned during manufacturing, Storage Root Key (SRK), owner authorization data, platform policies and authorization values, and monotonic counters for replay protection.

Volatile Memory (4-16 KB) Session-based storage reset on each boot containing current PCR values (24+ registers), loaded key handles, active authorization sessions, and temporary objects.

1.3 Platform Configuration Registers (PCRs)

1.3.1 PCR Architecture and Properties

PCRs are special-purpose registers within the TPM with unique characteristics:

- **Size**: 256 bits (SHA-256) or 384 bits (SHA-384) per register
- **Count**: Minimum 24 registers (PCR 0-23), implementations may provide more
- **Write-Once Property**: PCRs cannot be directly written; only extended through cryptographic operations
- **Initialization**: Set to known values (0x000...000) on platform reset
- **Tamper-Evidence**: Any modification produces cryptographically different values

1.3.2 PCR Allocation

The TCG PC Client Platform specification defines standard PCR usage:

PCR	Contents
0	CRTM, BIOS code, Host Platform Extensions
1-3	Platform Configuration, Option ROMs
4-5	Boot Loader (MBR, GRUB), Boot Configuration
6-7	State Transitions, Platform Manufacturer Specific
8-9	OS Kernel, initramfs
10-15	OS drivers, services, applications

Table 1: Standard PCR Allocation

1.4 PCRs in Measured Boot

1.4.1 The Extend Operation

The PCR extend operation is the fundamental mechanism for recording measurements:

$$\text{PCR}_{\text{new}} = \text{Hash}(\text{PCR}_{\text{old}} \,||\, \text{Data}) \tag{1}$$

Critical Properties:

- **One-way**: Computationally infeasible to reverse
- **Collision-resistant**: Extremely unlikely that two different boot sequences produce identical PCR values
- **Deterministic**: Same boot sequence always produces same final PCR values
- **Order-dependent**: The sequence of measurements matters

1.4.2 Measured Boot Chain

Measured boot implements a chain of trust where each component measures the next before transferring control:

1. **Power-On**: All PCRs initialized to 0x00...00
2. **CRTM Execution**: Immutable boot block measures BIOS firmware, extends to PCR-0, transfers control
3. **BIOS Execution**: Measures platform configuration → PCR-1, option ROMs → PCR-2/3, bootloader → PCR-4
4. **Bootloader**: Measures configuration → PCR-5, kernel/initramfs → PCR-8/9, launches kernel
5. **OS Kernel**: Measures loaded drivers → PCR-10+, system services extend additional PCRs

Key Principle: Each stage must measure the next before executing it, creating a tamper-evident log of the boot process.

1.5 Security Guarantees and Applications

1.5.1 Tamper Detection Mechanism

PCRs provide cryptographic evidence of platform integrity. Consider a system with legitimate BIOS producing PCR-0 = `0xA4B7...C3D2`. If an attacker modifies the BIOS, the modified code produces a different hash, causing PCR-0 to become `0x7F3E...9B1A`. Even single-bit modification produces an avalanche effect, guaranteeing detection through cryptographic properties.

1.5.2 Use Case 1: Sealed Storage (BitLocker)

Scenario: Windows BitLocker full-disk encryption
Implementation:

1. Disk encryption key generated and sealed to PCRs 0,1,2,3,4,5,6,7
2. Sealing binds key to current boot configuration
3. On boot, TPM checks PCR values
4. If PCRs match sealed values → Key released, disk decrypted
5. If PCRs differ → Key withheld, disk remains encrypted

Attack Prevention: Bootkit installation changes PCR-4 causing unseal failure. BIOS rootkit changes PCR-0 preventing key release. The attacker cannot extract the key, even with physical access to the disk.

1.5.3 Use Case 2: Remote Attestation

Scenario: Enterprise VPN requiring verified client integrity
Protocol:

1. Client requests VPN access
2. Server sends attestation challenge (nonce)
3. TPM generates quote: $\text{Quote} = \text{Sign}(\text{PCRs} \,||\, \text{nonce}, \text{AIK})$
4. Server verifies AIK certificate, quote signature, nonce freshness, and PCR values
5. Access granted only if verification succeeds

Benefits: Detects compromised clients before network access, prevents malware spread, provides cryptographic proof of compliance, and enables zero-trust architecture with continuous verification.

1.6 Conclusion

The TPM architecture provides a hardware-isolated secure subsystem with dedicated cryptographic capabilities. Platform Configuration Registers serve as tamper-evident measurement logs, enabling measured boot through cryptographic extend operations. The one-way, collision-resistant nature of PCR extends ensures that any boot-time modification produces detectably different values.

This foundation enables two critical security functions: sealed storage binds secrets to platform state (protecting against offline attacks), while remote attestation provides cryptographic proof of integrity to remote verifiers (enabling zero-trust architectures). Together, these mechanisms form the cornerstone of hardware-based trusted computing, defending against sophisticated boot-level attacks that evade traditional software-only security measures.

2 Question 2: TPM-Based Remote Attestation

2.1 Introduction

Remote attestation is a security mechanism that enables a platform to cryptographically prove its integrity state to a remote verifier. Using the TPM as a hardware root of trust, attestation provides verifiable evidence that a system is running known, unmodified software. This capability is fundamental to zero-trust security architectures, confidential computing, and compliance verification in distributed environments.

2.2 Remote Attestation Protocol

2.2.1 Key Cryptographic Components

Endorsement Key (EK) The Endorsement Key is a unique 2048-bit RSA key pair embedded in the TPM during manufacturing. The EK private key never leaves the TPM hardware and serves as the platform's cryptographic identity. The EK certificate, signed by the TPM manufacturer, provides a trust anchor linking the physical TPM to its provenance.

Attestation Identity Key (AIK) To preserve user privacy and prevent tracking across different attestation contexts, the TPM generates Attestation Identity Keys as pseudonyms for the EK. Multiple AIKs can be created for different relying parties, preventing correlation of a user's activities across services. AIKs are certified by a Privacy Certificate Authority (Privacy CA) after verifying the association with a legitimate EK.

Attestation Quote An attestation quote is a digitally signed data structure containing the current PCR values, a nonce provided by the verifier (for freshness), firmware version information, and additional metadata. The quote is signed with the AIK private key, creating unforgeable evidence of the platform's state at attestation time.

2.2.2 Two-Phase Attestation Protocol

Phase 1: AIK Provisioning (One-Time Setup)

1. Platform generates a new AIK key pair within the TPM
2. Platform obtains EK certificate from TPM and AIK public key
3. Platform submits certification request to Privacy CA containing:
    - EK certificate (proving genuine TPM)
    - AIK public key (to be certified)
    - Proof of AIK ownership (signed challenge)
4. Privacy CA validates the EK certificate chain against manufacturer root CA
5. Upon validation, Privacy CA issues AIK certificate binding the AIK to the verified TPM

Phase 2: Runtime Attestation

1. **Challenge**: Verifier generates random nonce and specifies PCR selection (e.g., PCRs 0-9)

2. **Quote Generation**: TPM creates attestation quote:

```
Quote_Structure = {
    magic: TPM_GENERATED,
    type: TPM_ST_ATTEST_QUOTE,
    qualifiedSigner: AIK_Name,
    extraData: Nonce,
    clockInfo: {clock, resetCount, restartCount},
    firmwareVersion: 0x....,
    pcrSelect: {sha256: [0,1,2,3,4,5,6,7,8,9]},
    pcrDigest: SHA256(PCR[0]||PCR[1]||...||PCR[9])
}
Signature = TPM2_Sign(Quote_Structure, AIK_Handle)

```

3. **Response Transmission**: Platform sends attestation evidence:
    - Attestation quote structure
    - AIK signature over the quote
    - AIK certificate
    - Optional: Event log detailing what was measured into each PCR

4. **Verification Process**: Verifier performs multi-step validation:
    - **Certificate Chain Validation**: Verify AIK certificate chains to trusted Privacy CA
    - **Signature Verification**: Validate signature using AIK public key from certificate
    - **Freshness Check**: Confirm nonce in quote matches sent challenge
    - **Integrity Assessment**: Compare PCR digest against reference ”golden” measurements
    - **Policy Evaluation**: Check additional constraints (firmware version, reset count limits)

5. **Authorization Decision**: Grant or deny access based on attestation result

2.3 Security Properties

2.3.1 Authenticity

The attestation evidence originates from a genuine TPM, verified through the manufacturer-signed EK certificate. Hardware-based key storage prevents attackers from forging attestation quotes, even with OS-level control.

2.3.2 Integrity

PCR values provide cryptographic proof of the measured boot sequence. Any modification to BIOS, bootloader, kernel, or drivers results in different PCR values, detectable during verification.

2.3.3 Freshness

The inclusion of a verifier-provided nonce ensures quotes cannot be replayed. An attacker capturing a valid attestation quote cannot reuse it, as the nonce will not match subsequent challenges.

2.3.4 Privacy Preservation

AIKs enable attestation without revealing the platform's unique identity (EK) to relying parties. Different AIKs can be used for different services, preventing cross-context tracking while maintaining trust in attestation.

2.4 Real-World Use Case: Azure Confidential Computing

2.4.1 Scenario

A financial services company (TechCorp) deploys critical transaction processing workloads on Microsoft Azure. Regulatory compliance (PCI-DSS) requires cryptographic proof that VMs are running only approved, unmodified software before accessing customer credit card data. The company must defend against both external attackers and potential insider threats with cloud infrastructure access.

2.4.2 Architecture and Implementation

Infrastructure Setup

- Azure provisions VMs with virtual TPM (vTPM) 2.0
- Secure Boot enabled; Measured Boot configured
- Boot measurements: UEFI firmware → PCR 0,7; Bootloader → PCR 4; Kernel → PCR 8,9; IMA driver measurements → PCR 10
- Reference PCR values for approved configuration stored in attestation policy

Attestation-Gated Access Flow

1. Transaction processing application starts and requests database credentials from Azure Key Vault
2. Key Vault intercepts request and triggers attestation via Azure Attestation Service (AAS)
3. AAS generates fresh nonce and sends attestation challenge to VM's vTPM
4. VM's vTPM generates quote containing current PCR values and signs with AIK

5. AAS performs comprehensive verification:
   - Validates AIK certificate issued by Azure Privacy CA
   - Verifies quote signature cryptographically
   - Confirms nonce freshness (within 60-second window)
   - Compares PCR values against attestation policy:

```
PCR[0,7]: UEFI firmware hash == approved_uefi_hash
PCR[4]:   Bootloader hash == grub2_signed_hash
PCR[8,9]: Kernel hash == ubuntu_22.04_kernel_5.15.0
PCR[10]:  IMA digest == policy_approved_binaries

```

6. **Success Path**: If all checks pass, AAS issues signed JWT token containing:
   - VM identity
   - Attestation timestamp
   - PCR claims
   - Token expiration (5 minutes)

   Application presents token to Key Vault, which releases database credentials.

7. **Failure Path**: If attestation fails:
   - Access denied; credentials withheld
   - Security alert generated with failure reason
   - VM automatically tagged for quarantine
   - SOC team notified for investigation

2.4.3 Attack Scenario: Bootkit Installation

Consider an attack where a malicious insider or compromised account attempts to subvert the VM:

1. **Attack**: Attacker gains access to Azure subscription and modifies VM's boot disk, injecting bootkit into GRUB bootloader to exfiltrate credit card data
2. **VM Reboot**: Modified bootloader executes; vTPM measures bootloader during measured boot
3. **PCR Deviation**: PCR[4] now contains hash of modified bootloader instead of legitimate GRUB:

```
Expected: PCR[4] = 0xa4f8c2...  (legitimate GRUB)
Actual:   PCR[4] = 0x3b9e71...  (infected bootkit)

```

4. **Attestation Trigger**: Application requests database credentials, triggering attestation

5. **Detection**: AAS detects PCR[4] mismatch during verification
6. **Denial**: **Access DENIED** - Application never receives credentials
7. **Response**: Automated quarantine, SOC alert, forensic snapshot captured

Impact: Without attestation, bootkit would successfully steal data. With attestation, attack detected immediately with zero data exposure, demonstrating defense-in-depth against sophisticated boot-level compromise.

2.4.4 Benefits Realized

- **Boot-Level Threat Detection**: Identifies BIOS rootkits, bootkits, and firmware malware that evade traditional endpoint security
- **Compliance Evidence**: Cryptographic audit trail proving only approved software accessed sensitive data (PCI-DSS 2.6, 6.5)
- **Zero-Trust Architecture**: Continuous verification eliminates implicit trust; "never trust, always verify"
- **Automated Remediation**: Policy violations trigger automated quarantine and alerting without human intervention
- **Insider Threat Mitigation**: Even cloud administrators cannot bypass attestation controls

2.5 Conclusion

TPM-based remote attestation provides a hardware-rooted mechanism for verifying platform integrity in distributed systems. By leveraging the TPM's cryptographic capabilities (EK for authenticity, AIK for privacy, PCRs for integrity), attestation enables verifiers to make trust decisions based on unforgeable evidence rather than implicit trust.

The Azure confidential computing use case demonstrates practical deployment at enterprise scale, showing how attestation prevents sophisticated boot-level attacks that evade traditional security controls. As cloud and edge computing expand, hardware-based attestation becomes essential for establishing trust boundaries in zero-trust architectures, supporting confidential computing workloads, and meeting regulatory requirements for cryptographic proof of system integrity.

3 Question 3: TEE Components - ARM TrustZone and Intel SGX

3.1 Introduction

A Trusted Execution Environment (TEE) is an isolated execution context that provides security services to applications running in an untrusted operating system. TEEs achieve isolation through hardware-enforced mechanisms, creating protected enclaves where sensitive code and data remain confidential even when the main OS is compromised. This section examines two prominent TEE implementations: ARM TrustZone, which provides system-wide dual-world isolation, and Intel SGX, which offers per-process enclave-based protection.

3.2 ARM TrustZone Architecture

ARM TrustZone technology extends the ARM processor architecture to provide system-wide hardware isolation between a "Secure World" for trusted operations and a "Normal World" for the standard operating environment. This approach enables security-critical functions to execute in complete isolation from the rich OS and applications.

3.2.1 Core Components

Processor Security Extensions **Security State Bit (NS)**: At the heart of TrustZone is a single hardware bit added to the processor pipeline that defines the current security state: NS=0 indicates Secure World execution, while NS=1 indicates Normal World execution. This bit is integrated throughout the processor, caches, memory management unit (MMU), and system bus, enforcing isolation at the hardware level.

Exception Level Architecture:

- **EL0**: Application level (both worlds)
- **EL1**: OS Kernel level (Normal World: Linux/Android; Secure World: Trusted OS like OP-TEE)
- **EL2**: Hypervisor level (virtualization)
- **EL3**: Secure Monitor - highest privilege level existing only in Secure state

Secure Monitor (EL3): The Secure Monitor operates at the highest privilege level and acts as the gatekeeper between worlds. It handles world switches via the Secure Monitor Call (SMC) instruction, saving and restoring processor context (registers, MMU configuration, interrupt state) to ensure complete isolation. The monitor validates all world-switch requests to prevent unauthorized access.

Memory Protection Infrastructure **TrustZone Address Space Controller (TZASC)**: The TZASC partitions physical memory into secure and non-secure regions through memory protection tables. Each DRAM region is tagged as secure or non-secure. Any attempt by the Normal World to access secure memory results in a bus error, enforced at the memory controller level before reaching DRAM. This hardware-level protection prevents DMA attacks and ensures memory isolation.

TrustZone Protection Controller (TZPC): The TZPC extends the security boundary to peripheral devices. Each peripheral (crypto accelerators, secure timers, secure storage controllers, touchscreen drivers) is assigned a security attribute. Secure peripherals can only be accessed from the Secure World, enabling protected I/O paths critical for applications like biometric authentication and secure display.

Trusted OS and Trusted Applications The Secure World runs a lightweight Trusted OS (e.g., OP-TEE, Trustonic Kinibi) that manages Trusted Applications (TAs). TAs are isolated modules providing security services: cryptographic key management, secure payment processing, DRM license verification, and biometric template matching. The Trusted OS provides:

- Secure storage API with encryption and integrity protection
- Cryptographic services (AES, RSA, HMAC, key derivation)
- Secure time and monotonic counters
- Inter-TA isolation and communication

3.2.2 Operational Flow: Mobile Payment System

Scenario: A banking application on Android must sign a payment transaction using cryptographic keys that must never be exposed to the Normal World OS, even if Android is compromised by malware.

Implementation:

1. **Initialization**: Banking app (Normal World) uses GlobalPlatform TEE Client API to open session with Payment TA in Secure World
2. **Transaction Request**: User initiates $500 payment; app constructs transaction data: `{recipient, amount, timestamp}`
3. **SMC Invocation**: App calls `TEEC_InvokeCommand(session, CMD_SIGN_TRANSACTION, params)`. This triggers SMC instruction, trapping to Secure Monitor at EL3
4. **World Switch**:
   - Secure Monitor saves complete Normal World state (registers, PC, stack pointer, MMU config)
   - Switches NS bit: NS=1 → NS=0
   - Configures Secure World MMU (secure memory mapping)
   - Transfers control to OP-TEE kernel in Secure World
5. **Secure Execution**:
   - OP-TEE validates session and dispatches to Payment TA
   - Payment TA retrieves private key from secure storage (encrypted blob decrypted inside Secure World)
   - TA uses secure crypto engine to compute: `Signature = HMAC-SHA256(key, transaction)`

- Private key never leaves Secure World memory

6. **Return World Switch**:
   - TA returns signed transaction to OP-TEE
   - OP-TEE triggers SMC to return to Normal World
   - Secure Monitor restores Normal World context
   - Signed transaction passed back to app
7. **Completion**: App sends signed transaction to bank server for verification

Security Guarantee: Even if Android is fully compromised (kernel rootkit, malicious app, debugger), the private signing key remains in Secure World protected memory. Attackers cannot access secure memory due to TZASC hardware enforcement, and the key only exists in plaintext inside the secure crypto engine.

3.3 Intel SGX Architecture

Intel Software Guard Extensions (SGX) provides a fundamentally different TEE model: per-process isolation through encrypted memory enclaves. Unlike TrustZone's system-wide approach, SGX allows individual applications to create protected memory regions (enclaves) where code and data are encrypted and integrity-protected, even from privileged software like the OS or hypervisor.

3.3.1 Core Components

Encrypted Page Cache (EPC) The EPC is a special region of DRAM (typically 128MB, expandable to 256MB in SGXv2) reserved exclusively for enclave pages. The Memory Encryption Engine (MEE) sits between the CPU and memory, transparently encrypting all data written to the EPC and decrypting data read from it.

Encryption Mechanism:

- AES-128-GCM for confidentiality and integrity
- Unique per-page encryption key derived from CPU master key
- Version tree prevents rollback attacks
- Integrity verified on every memory access

Security Properties:

- Plaintext data exists only in CPU package (registers, L1/L2/L3 cache)
- DRAM contains only ciphertext and authentication tags
- Protected from cold boot attacks, DMA attacks, physical memory probing
- OS and hypervisor cannot access enclave memory (bus encryption ensures this)

SGX Instruction Set Extensions **Privileged Instructions** (Ring 0 - OS/Hypervisor):

- `ECREATE`: Allocate EPC pages for new enclave, initialize SECS (SGX Enclave Control Structure)
- `EADD`: Add page to enclave, assign page type (code, data, TCS)
- `EEXTEND`: Incrementally measure page contents into MRENCLAVE
- `EINIT`: Finalize enclave, verify measurement, lock enclave into initialized state

Unprivileged Instructions (Ring 3 - User Application):

- `EENTER`: Enter enclave execution; switch to enclave code, enable EPC access
- `EEXIT`: Exit enclave; return to untrusted application code
- `EGETKEY`: Derive enclave-specific cryptographic keys from CPU fuses
- `EREPORT`: Generate attestation report proving enclave identity

Enclave Measurement (MRENCLAVE) MRENCLAVE is a 256-bit cryptographic hash that uniquely identifies an enclave's initial code and data:

```
MRENCLAVE = 0x000...000  // Initial value
For each page added during enclave build:
    For each 256-byte chunk of page:
        MRENCLAVE = SHA256(MRENCLAVE || chunk || offset || flags)
// Final MRENCLAVE frozen at EINIT
```

This measurement serves two purposes:

- **Attestation**: Remote parties verify they're communicating with the expected enclave code
- **Sealing**: Enclaves derive encryption keys bound to MRENCLAVE, ensuring sealed data can only be unsealed by the same enclave version

SGX Attestation **Local Attestation**: Enables one enclave to verify another enclave on the same platform using `EREPORT` and `EGETKEY`.

Remote Attestation: Proves enclave identity to remote parties:

1. Enclave generates `REPORT` containing MRENCLAVE and user data (e.g., enclave public key)
2. Quoting Enclave (Intel-signed) converts REPORT to remotely-verifiable QUOTE
3. QUOTE signed by Intel Attestation Service (IAS) root key
4. Remote verifier checks: Intel signature, MRENCLAVE matches expected value, report data (establishing secure channel)

3.3.2 Operational Flow: Confidential ML Inference

Scenario: A healthcare provider wants to use a cloud-based ML service for cancer detection from medical images, but patient data cannot be exposed to the cloud provider or untrusted cloud OS.

Implementation:

Phase 1 - Setup:

1. ML service provider develops enclave containing inference model
2. Enclave built with `ECREATE, EADD, EEXTEND, EINIT`
3. Provider publishes MRENCLAVE = `0xa3f8c2...b4d6` and source code for verification
4. Independent auditors verify code matches published MRENCLAVE

Phase 2 - Attestation:

1. Hospital client connects to cloud service
2. Client requests attestation quote from enclave
3. Enclave generates `EREPORT` with MRENCLAVE and enclave's ephemeral public key
4. Quoting Enclave signs report, producing Intel-verified QUOTE
5. Client verifies:
   - Intel signature (authentic SGX platform)
   - MRENCLAVE matches audited model (correct code)
   - Timestamp freshness
6. Client establishes TLS channel directly to enclave using attested public key

Phase 3 - Confidential Processing:

1. Client encrypts medical image with TLS session key
2. Encrypted image sent to cloud (cloud provider sees only ciphertext)
3. Application calls `EENTER` to enter enclave
4. **Inside Enclave**:
   - Decrypt image (plaintext exists only in EPC, encrypted in DRAM)
   - Run ML inference model
   - Generate cancer detection result
   - Encrypt result with TLS key
5. `EEXIT` returns encrypted result to untrusted application
6. Application forwards encrypted result to client
7. Client decrypts result

Security Guarantee: Medical images exist in plaintext only inside enclave (CPU + EPC). Cloud provider OS, hypervisor, and administrators cannot access plaintext data even with root/physical access. MEE encryption protects against DRAM probing, cold boot attacks, and compromised firmware.

3.4 Comparison: TrustZone vs SGX

Aspect	**ARM TrustZone**	**Intel SGX**
Isolation Model	System-wide dual worlds	Per-process enclaves
TCB Size	Larger: Trusted OS + TAs (MBs)	Smaller: Individual enclave (KBs-MBs)
Memory Protection	TZASC/TZPC physical partitioning	MEE encryption, all memory untrusted
Threat Model	Protects from compromised Normal World OS	Protects from OS, hypervisor, privileged malware, physical attacks
I/O Access	Secure peripherals (crypto, display, biometrics)	No direct I/O; OCALL to untrusted app
Attestation	Limited, vendor-specific	Built-in remote attestation (IAS/DCAP)
Performance Overhead	Low (1-5μs world switch)	Low-moderate (10-15% for memory-intensive workloads)
Use Cases	Mobile payments, DRM, biometrics, secure boot	Cloud confidential computing, secure multi-party computation

Table 2: Comprehensive Comparison of TrustZone and SGX

3.5 Conclusion

ARM TrustZone and Intel SGX represent two distinct philosophical approaches to trusted execution environments. TrustZone provides a system-wide security partition with a hardened Trusted OS managing security services, optimized for scenarios requiring secure I/O (payments, biometrics). SGX offers fine-grained per-application isolation with cryptographic memory protection, ideal for cloud scenarios where the infrastructure provider is untrusted.

TrustZone's strength lies in its integration with the platform's security peripherals and lower performance overhead for world switches, making it prevalent in mobile and embedded systems. SGX's advantage is its stronger threat model protecting against privileged attackers and physical access, combined with robust remote attestation enabling cloud confidential computing.

Modern security architectures increasingly combine multiple TEE technologies: TrustZone for platform security services, SGX for workload-specific confidential computing, and TPM for boot integrity and attestation. This defense-in-depth approach provides comprehensive protection across the system lifecycle.

4 Question 4: TPM vs TEE - Comparative Analysis

4.1 Introduction

Trusted Platform Module (TPM) and Trusted Execution Environment (TEE) represent two distinct approaches to hardware-based security, each designed for specific threat models and use cases. While both provide hardware-rooted trust, they differ fundamentally in architecture, capabilities, and deployment scenarios. This section provides a comprehensive comparison, examining trust boundaries, execution contexts, performance characteristics, and practical decision criteria for selecting between these technologies.

4.2 Fundamental Architectural Differences

4.2.1 Trust Boundary Comparison

The trust boundary defines what components are trusted versus untrusted, establishing the security perimeter that attackers must breach.

Aspect	**TPM**	**TEE**
Boundary Type	Physical: Discrete hardware chip	Logical: CPU isolation modes
Isolation Mechanism	Separate processor on LPC/SPI bus	CPU security states (TrustZone) or memory encryption (SGX)
Trusted Components	TPM chip internals: firmware, crypto engine, NV storage	Secure World OS + TAs (TrustZone) or Enclave code (SGX)
Untrusted Components	Everything external: main CPU, DRAM, OS, applications	Normal World OS (TrustZone) or entire software stack except enclave (SGX)
TCB Size	Minimal: ∼100-200 KB TPM firmware	Larger: MB-scale Trusted OS or individual enclaves
Attack Surface	Extremely narrow: command interface only	Moderate: Larger code base, more functionality
Side-Channel Vulnerability	Low: Physical isolation reduces timing/cache attacks	Higher: Shared CPU resources enable cache timing, speculative execution attacks
Physical Attacks	Tamper-resistant packaging, requires chip decapping	CPU package security; memory bus encryption (SGX)

Table 3: Trust Boundary Analysis

Key Insight: TPM's physical isolation provides a smaller, more defensible perimeter, while TEEs offer richer functionality at the cost of increased attack surface.

4.2.2 Execution Context and Capabilities

Analysis: TPM is a passive security anchor unsuitable for computation. TEEs provide active compute environments: TrustZone for I/O-rich scenarios, SGX for compute-intensive workloads with strong confidentiality.

Aspect	TPM	TEE (TrustZone)	TEE (SGX)
Processor	Dedicated 8-16 MHz CPU	Full-speed ARM CPU (GHz)	Full-speed x86 CPU (GHz)
Memory Capacity	2-8 KB NV, 4-16 KB volatile	GB-scale (limited by TZASC)	128-256 MB EPC, paging support
OS Environment	Bare-metal firmware	Trusted OS (OP-TEE, Kinibi)	Optional LibOS (Graphene, SGX-LKL)
Computation Model	Stateless operations only	Full execution environment	Full execution environment
Code Flexibility	Fixed firmware, vendor-signed	Dynamically loadable TAs	User-compiled enclaves
I/O Capabilities	None (isolated from peripherals)	Secure peripherals: crypto, display, biometrics, secure storage	None (OCALL to untrusted app)
Performance	100ms-seconds per operation	Near-native (<5μs world switch)	Near-native (10-15% memory overhead)
Primary Functions	Crypto, measurement, attestation, seal/unseal	Payment, DRM, biometrics, full apps	Confidential computing, secure computation
Persistence	Non-volatile storage survives power cycles	Volatile (requires secure storage API)	Volatile (sealing for persistence)

Table 4: Execution Context Comparison

4.3 Threat Model Comparison

4.3.1 TPM Threat Model

Protects Against:

- Software attacks on boot process (bootkits, BIOS rootkits)
- Unauthorized access to sealed data after boot-time modifications
- Offline attacks (disk encryption extraction)
- Software-based key extraction from storage

Does NOT Protect Against:

- Runtime attacks after successful boot attestation
- Execution of malicious code in Normal World
- Memory scraping of keys after unsealing
- Performance-intensive confidential computing

Trust Assumption: TPM assumes attackers cannot physically compromise the discrete chip (decapping, fault injection). Everything outside the TPM boundary is untrusted.

4.3.2 TEE Threat Model

TrustZone - Protects Against:

- Compromised Normal World OS (kernel rootkits)
- Malicious applications
- Direct memory access (DMA) attacks on secure memory
- Peripheral attacks (bus sniffing)

TrustZone - Does NOT Protect Against:

- Vulnerabilities in Trusted OS or TAs

- Physical attacks on DRAM (memory remains plaintext)
- Compromised firmware/secure boot process
- Side-channel attacks exploiting shared cache

SGX - Protects Against:

- Malicious OS/hypervisor (privileged software)
- Physical memory attacks (DRAM encryption)
- Other processes on same system
- Cloud provider infrastructure access

SGX - Does NOT Protect Against:

- Side-channel attacks (cache timing, speculative execution - Spectre, Foreshadow)
- Enclave code vulnerabilities
- Compromised input/output from untrusted application
- Denial-of-service (EPC exhaustion)

4.4 Use Case Decision Framework

4.4.1 When to Use TPM

1. Boot Integrity and Platform Attestation **Scenario**: Enterprise laptop fleet requiring verified boot state before VPN access

Requirements:

- Measure entire boot chain (UEFI, bootloader, kernel, drivers)
- Cryptographically prove boot integrity to remote IT infrastructure
- Persist measurements across reboots and power cycles
- Minimal performance impact on boot process

Why TPM:

- PCRs provide tamper-evident measurement log
- Hardware-rooted attestation via AIK prevents forgery
- Measurements persist across reboots (non-volatile storage)
- Passive measurement has negligible boot time impact
- Industry-standard protocol (TCG specifications)

Example: BitLocker full-disk encryption seals disk key to PCRs 0,1,2,3,4,5,6,7. If bootkit modifies bootloader, PCR-4 changes, key withheld, disk remains encrypted. IT admin can remotely attest employee laptops before granting network access.

2. Long-Term Cryptographic Key Storage **Scenario**: IoT device fleet requiring authentication keys that survive device capture

Requirements:

- Hardware-bound keys (cannot be copied)
- Survive firmware updates and reboots
- Protection against dictionary attacks on key authorization
- Low power consumption for battery-powered devices

Why TPM:

- SRK-based hierarchy binds keys to TPM hardware
- Non-volatile storage persists keys across power cycles
- Dictionary attack protection (lockout after N failed auth attempts)
- Minimal power draw from dedicated low-power chip
- Key attributes prevent export (`fixedTPM` flag)

Example: Smart meters authenticate to utility company using TPM-bound ECC keys. Even if attacker steals meter and extracts firmware, private key cannot be extracted from TPM chip. Key bound to specific device prevents cloning attacks.

4.4.2 When to Use TEE

1. Runtime Secure Processing with I/O (TrustZone) **Scenario**: Mobile banking application requiring secure PIN entry and transaction signing

Requirements:

- Secure UI path (trusted keyboard, display) to prevent screen overlays and keyloggers
- Real-time transaction processing (low latency)
- Cryptographic signing operations
- Protection even if Android OS compromised

Why TrustZone:

- Secure World can access secure touchscreen and display peripherals (TZPC-protected)
- World switch overhead is minimal ($<5\mu$s), enabling real-time responsiveness
- Trusted Application handles crypto operations with keys in secure storage
- Even kernel-level Android malware cannot access Secure World memory (TZASC enforcement)

Example: Google Pay / Samsung Pay use TrustZone for payment applets. PIN entry occurs in Secure World with secure touchscreen input. Transaction signing key never exposed to Android. Even sophisticated malware with root privileges cannot extract payment credentials.

2. Cloud Confidential Computing (SGX) **Scenario**: Healthcare provider processing sensitive patient data using untrusted cloud infrastructure

Requirements:

- Data confidentiality from cloud provider (OS, hypervisor, administrators)
- Cryptographic proof of code integrity (remote attestation)
- Computationally intensive ML inference
- Protection against physical memory access

Why SGX:

- MEE encryption protects data even from privileged software and physical DRAM access
- Remote attestation proves enclave identity (MRENCLAVE) to client before data transmission
- Full CPU performance for compute-intensive workloads (ML inference)
- No trust required in cloud provider infrastructure

Example: Microsoft Azure Confidential Computing. Medical imaging AI model runs in SGX enclave. Patient scans encrypted end-to-end; plaintext exists only in CPU/EPC. Azure administrators cannot access patient data even with hypervisor control. HIPAA compliance through cryptographic confidentiality.

4.4.3 Combined Deployment: Defense-in-Depth

Scenario: Secure Government Workstation Modern secure systems deploy **both TPM and TEE** for comprehensive protection:

Boot Phase:

- TPM measures boot chain (UEFI → GRUB → Linux kernel)
- Disk encryption key sealed to PCRs; released only if measurements match approved baseline
- Remote attestation to agency security infrastructure before network access

Runtime Phase:

- SGX enclaves process classified documents (redaction, encryption, access control)
- TrustZone handles smart card authentication and secure communications
- Sensitive keys stored in TPM; computational operations in enclaves

Attestation:

- Combined attestation: TPM quote (platform integrity) + SGX quote (application integrity)

- Server verifies: boot chain unmodified + approved enclave code + TrustZone-authenticated user
- Access granted only if all three security layers validate

Benefit: Defense-in-depth across entire system lifecycle: TPM protects boot, TrustZone protects peripherals, SGX protects runtime computation. Compromise requires breaking multiple hardware-isolated security boundaries.

4.5 Selection Criteria Matrix

Requirement	**TPM**	**TEE**
Verify boot integrity	✓	
Platform-wide attestation	✓	
Long-term key storage (survives reboot)	✓	
Seal data to platform state	✓	
Low power consumption	✓	
Execute arbitrary trusted code		✓
High-performance computation		✓
Secure user interface (PIN, biometrics)		✓(TZ)
Cloud confidential computing		✓(SGX)
Protection from physical memory attacks		✓(SGX)
Runtime secret protection		✓
Third-party code execution		✓

Table 5: Technology Selection Matrix

4.6 Conclusion

TPM and TEE are complementary security technologies designed for fundamentally different purposes. TPM provides a passive, hardware-isolated root of trust optimized for boot-time integrity measurement, platform attestation, and long-term key storage. Its discrete chip architecture offers minimal attack surface but limited computational capability. TEEs provide active execution environments for trusted applications, enabling runtime protection of sensitive code and data with full computational performance.

The choice between TPM and TEE depends on the security requirements:

- **Boot-time security**: TPM's measured boot and sealed storage protect against bootkits and offline attacks
- **Runtime secure processing**: TEEs enable confidential computing while system is operational
- **I/O-rich scenarios**: TrustZone provides secure peripheral access (payments, biometrics)
- **Cloud scenarios**: SGX protects against untrusted infrastructure

Modern secure systems increasingly deploy both technologies in concert: TPM establishes trust from first instruction (measured boot, attestation), while TEEs maintain isolation throughout runtime (confidential computing). This layered approach provides comprehensive protection spanning the entire system lifecycle, from power-on through complex application workloads, defending against the full spectrum of software and hardware attacks.

Part II
Practical Tasks (10 marks)

5 Part B: TPM Practical Demonstration

5.1 Objective

Demonstrate TPM 2.0 secure key storage and PCR-based data sealing: (1) Create TPM key, (2) Seal data to PCR values, (3) Unseal successfully when PCRs match, (4) Show unseal failure when PCRs change.

5.2 Environment

- **OS**: Ubuntu 22.04, **TPM**: Software TPM (swtpm) at localhost:2321, **Tools**: tpm2-tools v5.x

5.3 Task 1: Read Initial PCR Values

Commands:

```
tpm2_pcrread sha256:0,7
```

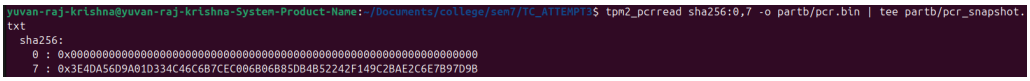

Figure 1: Initial PCR values (PCR 0: BIOS, PCR 7: Platform-specific)

PCR values represent baseline system state; will be used to create sealing policy.

5.4 Task 2: Create TPM Primary Key

Commands:

```
tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx \
  -a "restricted|decrypt|fixedtpm|fixedparent|sensitivedataorigin|userwithauth
    "
```

Primary key serves as root for key hierarchy; `fixedtpm` ensures hardware binding.

5.5 Task 3: Create PCR-Bound Policy & Seal Data

Commands:

```
# Create PCR policy
tpm2_startauthsession --policy-session -S policy.session
tpm2_policypcr --session policy.session --pcr-list sha256:0,7
tpm2_policygetdigest --session policy.session -o policy.digest
tpm2_flushcontext policy.session

```

Figure 2: Primary storage key (RSA 2048) in owner hierarchy

```
# Seal data
echo "Confidential lab secret for Assignment 2" > secret.txt
tpm2_create -C primary.ctx -L policy.digest -i secret.txt \
  -r seal.priv -u seal.pub -a "fixedtpm|fixedparent"
```

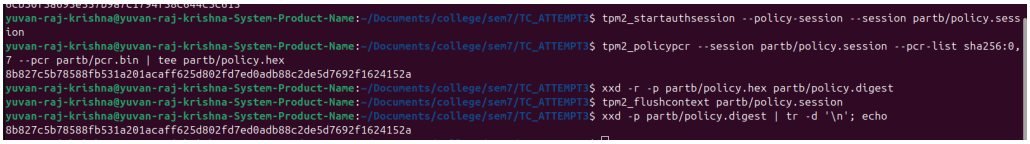

Figure 3: PCR policy digest bound to PCR 0 and 7

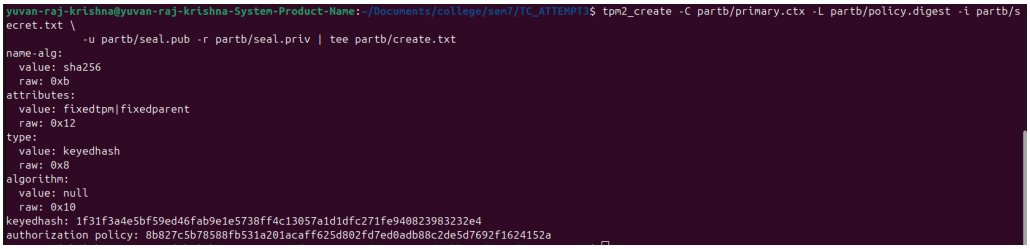

Figure 4: Sealed object with embedded PCR policy

Policy digest cryptographically binds to current PCR values; sealed object can only be unsealed when PCRs match.

5.6 Task 4: Load Sealed Object

Commands:

```
tpm2_load -C primary.ctx -r seal.priv -u seal.pub -c seal.ctx
```

5.7 Task 5: Unseal with Correct PCRs (Success)

Commands:

```
yuvan-raj-krishna@yuvan-raj-krishna-System-Product-Name:~/Documents/college/sem7/TC_ATTEMPT3$
      tpm2_load -C partb/primary.ctx -u partb/seal.pub -r partb/seal.priv -c partb/seal.ctx |
 tee partb/load.txt
name: 000bcb9cfbb61b41710939b10e0ac2cd4bf4f450185b3c1c8744774443b868ae2ece
yuvan-raj-krishna@yuvan-raj-krishna-System-Product-Name:~/Documents/college/sem7/TC_ATTEMPT3$
```

Figure 5: Sealed object loaded into TPM with unique object name

```
tpm2_startauthsession --policy-session -S policy.session
tpm2_policypcr --session policy.session --pcr-list sha256:0,7
tpm2_unseal -c seal.ctx -p session:policy.session
tpm2_flushcontext policy.session
```

```
yuvan-raj-krishna@yuvan-raj-krishna-System-Product-Name:~/Documents/college/sem7/TC_ATTEMPT3$
      tpm2_load -C partb/primary.ctx -u partb/seal.pub -r partb/seal.priv -c partb/seal.ctx |
 tee partb/load.txt
name: 000bcb9cfbb61b41710939b10e0ac2cd4bf4f450185b3c1c8744774443b868ae2ece
yuvan-raj-krishna@yuvan-raj-krishna-System-Product-Name:~/Documents/college/sem7/TC_ATTEMPT3$
 tpm2_getcap handles-transient
- 0x80000000
- 0x80000001
```

Figure 6: Policy session satisfying PCR requirements

```
yuvan-raj-krishna@yuvan-raj-krishna-System-Product-Name:~/Documents/college/sem7/TC_ATTEMPT3$
 tpm2_readpublic -c partb/seal.ctx
name: 000bcb9cfbb61b41710939b10e0ac2cd4bf4f450185b3c1c8744774443b868ae2ece
qualified name: 000b95dc396323b43c079980389ba114eefa340bcb22b0823604f383d81531fadb91
name-alg:
  value: sha256
  raw: 0xb
attributes:
  value: fixedtpm|fixedparent
  raw: 0x12
type:
  value: keyedhash
  raw: 0x8
algorithm:
  value: null
  raw: 0x10
keyedhash: 1f31f3a4e5bf59ed46fab9e1e5738ff4c13057a1d1dfc271fe940823983232e4
authorization policy: 8b827c5b78588fb531a201acaff625d802fd7ed0adb88c2de5d7692f1624152a
```

Figure 7: Successfully unsealed secret: "Confidential lab secret for Assignment 2"

Why Success: Current PCRs match sealed policy; TPM verifies and releases plaintext.

5.8 Task 6: Simulate Platform Tampering

Commands:

```
# Extend PCR 0 to simulate BIOS modification
echo "TAMPERED_BOOTLOADER" | openssl dgst -sha256 -binary | xxd -p > hash.txt
tpm2_pcrextend 0:sha256=$(cat hash.txt)
tpm2_pcrread sha256:0
```

PCR extend is irreversible: $\text{PCR}_{\text{new}} = \text{SHA256}(\text{PCR}_{\text{old}} \,||\, \text{"TAMPERED"})$
Simulates bootkit, BIOS modification, or unauthorized firmware change.

Figure 8: PCR 0 extended; value changed completely from baseline

5.9 Task 7: Unseal with Modified PCRs (Failure)

Commands:

```
# Attempt unseal (will fail)
tpm2_startauthsession --policy-session -S policy.session
tpm2_policypcr --session policy.session --pcr-list sha256:0,7
tpm2_unseal -c seal.ctx -p session:policy.session
```

Figure 9: Unseal failure: TPM_RC_POLICY_FAIL due to PCR mismatch

Why Failure: Current PCR 0 $\neq$ Sealed PCR 0 $\rightarrow$ Policy digest mismatch $\rightarrow$ TPM refuses to release secret.

Security Implication: Sealed data cryptographically bound to platform state; tampering prevents access.

5.10 Cleanup

Figure 10: Flushing transient handles and sessions

5.11 Conclusion

Results:

- ✓Created primary storage key (RSA 2048, owner hierarchy)
- ✓Created PCR-bound sealing policy
- ✓Sealed confidential data to PCR values
- ✓Successfully unsealed with matching PCRs
- ✓Demonstrated unseal failure after PCR modification (TPM_RC_POLICY_FAIL)

Key Learning: TPM provides hardware-backed protection through PCR-based sealing. Data cryptographically bound to platform state; any boot chain modification (detected via PCR changes) prevents unauthorized access. This forms the foundation for full-disk encryption (BitLocker, LUKS) and remote attestation in enterprise security.

Real-World Application: BitLocker seals disk key to PCRs 0,1,2,3,4,5,6,7. If BIOS or bootloader compromised, PCRs change, disk remains encrypted—protecting against offline attacks and bootkits.

References

1. Trusted Computing Group, “TPM 2.0 Library Specification”, Part 1: Architecture, Rev. 1.59, November 2019

2. Arthur, W., & Challener, D. (2015). *A Practical Guide to TPM 2.0*. Apress

3. TCG PC Client Platform Firmware Profile Specification, Family 2.0, Level 00 Revision 1.05

4. Sailer, R., et al. (2004). “Design and Implementation of a TCG-based Integrity Measurement Architecture”. *USENIX Security Symposium*

5. ARM, “ARM Security Technology - Building a Secure System using TrustZone Technology”, 2022

6. Ngabonziza, B., et al. (2016). “TrustZone Explained: Architectural Features and Use Cases”. *IEEE Cybersecurity Development Conference*

7. Costan, V., & Devadas, S. (2016). “Intel SGX Explained”. *IACR Cryptology ePrint Archive*

8. McKeen, F., et al. (2013). “Innovative Instructions and Software Model for Isolated Execution”. *HASP Workshop*

9. GlobalPlatform, “TEE Internal Core API Specification”, Version 1.3.1, 2022

10. Intel Corporation, “Intel SGX Developer Reference”, 2024