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

I Theory Questions (10 marks) 3

1 Question 1: TPM Architecture and PCRs in Measured Boot 3
- 1.1 TPM Internal Architecture . . . 3
  - 1.1.1 Root of Trust Components . . . 3
  - 1.1.2 Cryptographic Engine . . . 3
  - 1.1.3 Memory . . . 3
- 1.2 Platform Configuration Registers (PCRs) . . . 3
  - 1.2.1 PCR Architecture . . . 3
- 1.3 PCRs in Measured Boot . . . 4
  - 1.3.1 Extend Operation . . . 4
  - 1.3.2 Measured Boot Chain . . . 4
- 1.4 Security Guarantees and Use Cases . . . 4
  - 1.4.1 Tamper Detection . . . 4
  - 1.4.2 Use Case 1: Sealed Storage . . . 4
  - 1.4.3 Use Case 2: Remote Attestation . . . 5
- 1.5 Conclusion . . . 5

2 Question 2: TPM-Based Remote Attestation 6
- 2.1 Remote Attestation Protocol . . . 6
  - 2.1.1 Key Components . . . 6
  - 2.1.2 Attestation Flow . . . 6
- 2.2 Security Properties . . . 7
- 2.3 Use Case: Azure Cloud VM Attestation . . . 7
  - 2.3.1 Scenario . . . 7
  - 2.3.2 Implementation . . . 7
  - 2.3.3 Attack Scenario Prevented . . . 7
  - 2.3.4 Benefits . . . 8
- 2.4 Conclusion . . . 8

3 Question 3: TEE Components - ARM TrustZone and Intel SGX 9
- 3.1 ARM TrustZone Architecture . . . 9
  - 3.1.1 Core Components . . . 9
  - 3.1.2 Operational Flow: Mobile Payment . . . 9
- 3.2 Intel SGX Architecture . . . 10
  - 3.2.1 Core Components . . . 10
  - 3.2.2 Operational Flow: Confidential ML Inference . . . 10
- 3.3 Comparison: TrustZone vs SGX . . . 11
- 3.4 Conclusion . . . 11

4 Question 4: TPM vs TEE - Comparison 12
- 4.1 Trust Boundary . . . 12
- 4.2 Execution Context . . . 12
- 4.3 Use Case Analysis . . . 12
  - 4.3.1 Use TPM For: . . . 12
  - 4.3.2 Use TEE For: . . . 13

4.3.3 Use Both (TPM + TEE): . . . 13
4.4 Selection Criteria . . . 13
4.5 Conclusion . . . 13

II Practical Tasks (10 marks) 14

5 Part B: TPM Practical Demonstration 14
5.1 Objective . . . 14
5.2 Environment . . . 14
5.3 Task 1: Read Initial PCR Values . . . 14
5.4 Task 2: Create TPM Primary Key . . . 14
5.5 Task 3: Create PCR-Bound Policy & Seal Data . . . 15
5.6 Task 4: Load Sealed Object . . . 15
5.7 Task 5: Unseal with Correct PCRs (Success) . . . 16
5.8 Task 6: Simulate Platform Tampering . . . 16
5.9 Task 7: Unseal with Modified PCRs (Failure) . . . 16
5.10 Cleanup . . . 17
5.11 Conclusion . . . 17

References 19

Part I
Theory Questions (10 marks)

1 Question 1: TPM Architecture and PCRs in Measured Boot

1.1 TPM Internal Architecture

A Trusted Platform Module (TPM) is a hardware security chip providing cryptographic root of trust. Key components include:

1.1.1 Root of Trust Components

- **Core Root of Trust for Measurement (CRTM)**: First code executed after boot; measures BIOS/UEFI and stores in PCR-0
- **Root of Trust for Storage (RTS)**: Based on Storage Root Key (SRK) - 2048-bit RSA key never exported; protects key hierarchy
- **Root of Trust for Reporting (RTR)**: Uses Attestation Identity Key (AIK) for signing attestation quotes

1.1.2 Cryptographic Engine

Hardware-based processor providing:

- **Key Generation**: Hardware RNG (NIST SP 800-90A); RSA/ECC key generation
- **Crypto Operations**: RSA sign/verify, ECC operations, AES encryption
- **Hash Engine**: SHA-1/256/384 for measurements and PCR extends

1.1.3 Memory

- **Non-Volatile (2-8 KB)**: Stores Endorsement Key (EK), SRK, owner authorization, platform policies
- **Volatile (4-16 KB)**: Active PCR values, loaded keys, session data (reset on reboot)

1.2 Platform Configuration Registers (PCRs)

1.2.1 PCR Architecture

Properties:

- 24+ registers, each 256 bits (SHA-256) or 384 bits (SHA-384)
- **Write-once per boot**: Cannot be directly written, only extended
- **Reset on boot**: Initialized to known values (0x00...00)

- **Tamper-evident**: Any change produces completely different values

Usage Convention:

- PCR 0: CRTM, BIOS, Platform Extensions
- PCR 1-3: Platform Configuration, Option ROMs
- PCR 4-5: Boot Loader (GRUB), Boot Configuration
- PCR 8-9: OS Kernel, initramfs
- PCR 10-15: System drivers and services

1.3 PCRs in Measured Boot

1.3.1 Extend Operation

The core PCR operation:

$$\text{PCR}_{\text{new}} = \text{SHA256}(\text{PCR}_{\text{old}} \,||\, \text{data\_to\_measure}) \tag{1}$$

Properties: One-way, collision-resistant, deterministic (same boot → same PCR values)

1.3.2 Measured Boot Chain

1. **Power-On**: PCRs initialized to 0x00...00
2. **CRTM**: Measures BIOS → Extend PCR-0
3. **BIOS**: Measures bootloader → Extend PCR-4
4. **Bootloader**: Measures kernel → Extend PCR-8
5. **Kernel**: Measures drivers → Extend PCR-10+

Key Principle: Each stage measures next stage *before* executing it.

1.4 Security Guarantees and Use Cases

1.4.1 Tamper Detection

Any modification changes PCR completely:

- Original: PCR[0] = 0xAAAA...1111
- Modified BIOS: PCR[0] = 0xBBBB...2222 (completely different)

1.4.2 Use Case 1: Sealed Storage

BitLocker Example:

- Seal disk encryption key to PCRs 0,1,2,3,4,5,6,7
- Key released only if current PCRs match sealed values
- If BIOS/bootloader modified → PCR mismatch → Disk stays encrypted

1.4.3 Use Case 2: Remote Attestation

Corporate VPN Example:

- TPM signs PCR values with AIK: Quote = $\text{Sign}(\text{PCRs} \,||\, \text{nonce}, \text{AIK}_{\text{priv}})$
- VPN server verifies quote and compares PCRs to approved values
- Grant access only if platform integrity verified

1.5 Conclusion

TPM's hardware-isolated architecture with tamper-evident PCRs enables measured boot by creating irreversible hash chains of boot components. This provides cryptographic proof of platform integrity for both local (sealed storage) and remote (attestation) verification, forming the foundation of hardware-based trusted computing.

2 Question 2: TPM-Based Remote Attestation

2.1 Remote Attestation Protocol

Remote attestation enables a platform to cryptographically prove its integrity to a remote verifier using TPM as hardware root of trust.

2.1.1 Key Components

- **Endorsement Key (EK)**: Unique 2048-bit RSA key embedded during manufacturing; never leaves TPM
- **Attestation Identity Key (AIK)**: Privacy-preserving pseudonym for EK; signs attestation quotes
- **Quote**: Signed statement of PCR values including verifier's nonce for freshness

2.1.2 Attestation Flow

Phase 1: One-Time AIK Setup

1. Platform generates AIK in TPM
2. Privacy CA verifies EK certificate chain
3. CA issues AIK certificate

Phase 2: Attestation

1. **Challenge**: Verifier sends random nonce + PCR selection
2. **Quote Generation**: TPM executes:

```
Quote = {
    PCR_values,
    Nonce,
    Firmware_version
}
Signature = Sign(Quote, AIK_private)

```

3. **Response**: Platform sends {Quote, Signature, AIK_cert}
4. **Verification**: Verifier:
    - Verifies AIK certificate (from trusted CA)
    - Verifies quote signature with AIK public key
    - Validates nonce matches (freshness check)
    - Compares PCR values to "golden" approved values
5. **Decision**: Grant/Deny access based on results

2.2 Security Properties

- **Authenticity**: Evidence from genuine TPM (EK verified by manufacturer)
- **Integrity**: PCR values prove unmodified boot sequence
- **Freshness**: Random nonce prevents replay attacks
- **Privacy**: AIK protects user identity across contexts

2.3 Use Case: Azure Cloud VM Attestation

2.3.1 Scenario

Financial services company (TechCorp) uses Azure VMs for critical workloads. Must ensure VMs run only verified, unmodified software before accessing sensitive data.

2.3.2 Implementation

Setup

- Azure provisions VM with virtual TPM (vTPM)
- Measured boot: UEFI → PCR 0,7; Bootloader → PCR 4; Kernel → PCR 8,9

Attestation Flow

1. Finance app requests database credentials from Azure Key Vault
2. Key Vault triggers attestation: generates nonce, requests quote from VM's vTPM
3. VM's vTPM generates quote with current PCR values
4. Azure Attestation Service verifies:
    - AIK certificate validity
    - Quote signature
    - Nonce freshness
    - PCRs match attestation policy
5. **If trusted**: Issue token → Key Vault releases secrets
6. **If untrusted**: Deny access + security alert + VM quarantine

2.3.3 Attack Scenario Prevented

1. Attacker compromises Azure account, modifies VM bootloader with malicious code
2. VM reboots, bootloader measured differently: PCR[4] changes
3. App requests credentials → Attestation triggered
4. PCR[4] mismatch detected → **Access DENIED**

5. Security team alerted, VM isolated

Outcome: Without attestation, attacker steals data. With attestation, attack detected immediately with zero data loss.

2.3.4 Benefits

- Detects boot-level malware (bootkits, BIOS rootkits)
- Cryptographic proof of compliance for regulations
- Zero-trust architecture: Always verify, never trust
- Automated response to compromises

2.4 Conclusion

TPM-based remote attestation provides cryptographic proof of platform integrity using hardware roots of trust (EK), privacy-preserving credentials (AIK), and tamper-evident measurements (PCRs). The enterprise cloud use case demonstrates prevention of sophisticated boot-level attacks that evade traditional security, enabling zero-trust architectures in distributed systems.

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

3.1 ARM TrustZone Architecture

TrustZone creates two parallel execution environments on a single CPU:

3.1.1 Core Components

Processor Security Extensions

- **Secure Monitor (EL3)**: Highest privilege level; mediates transitions between worlds via SMC instruction
- **Security State Bit (NS)**: NS=0 (Secure World), NS=1 (Normal World)
- **Dual Worlds**:
  - Normal World: Rich OS (Android, Linux), untrusted apps
  - Secure World: Trusted OS (OP-TEE), Trusted Applications (TAs)

Memory Protection

- **TZASC (Address Space Controller)**: Marks memory as Secure/Non-Secure; Normal World access to Secure memory → Bus error
- **TZPC (Protection Controller)**: Tags peripherals (e.g., Crypto engine = SECURE, UART = NON-SECURE)

3.1.2 Operational Flow: Mobile Payment

Scenario: Banking app signs payment transaction without exposing key to Normal World OS.

1. User initiates payment (Normal World)
2. App calls `TEEC_InvokeCommand()` → SMC instruction
3. **World Switch**: Secure Monitor saves Normal World context, switches to Secure World
4. OP-TEE dispatches to Payment TA
5. TA: Validates transaction, loads key from secure storage, signs with crypto engine
6. **World Switch Back**: Return to Normal World with signed transaction
7. App sends to bank server

Security: Private key never exposed to Normal World; even if Android compromised, key remains safe.

3.2 Intel SGX Architecture

SGX provides per-application isolation via encrypted memory regions (enclaves).

3.2.1 Core Components

Encrypted Page Cache (EPC)

- Special DRAM region (128-256MB) managed exclusively by CPU
- Encrypted by Memory Encryption Engine (MEE) using AES-128
- Plaintext exists only inside CPU package; DRAM contains only ciphertext
- Protected from malicious OS, hypervisor, physical memory attacks

SGX Instructions

- **Management** (Ring 0): `ECREATE` (create enclave), `EADD` (add page), `EEXTEND` (measure page), `EINIT` (finalize)
- **Execution** (Ring 3): `EENTER` (enter enclave), `EEXIT` (exit enclave)

Enclave Measurement (MRENCLAVE)

```
For each page added:
    For each 256-byte chunk:
        MRENCLAVE = SHA256(MRENCLAVE || chunk || offset)
```

256-bit hash uniquely identifying enclave code/data; used for attestation and sealing.

3.2.2 Operational Flow: Confidential ML Inference

Scenario: Cloud ML service processes sensitive medical images without exposing to provider.

1. **Setup**: Provider builds enclave, publishes MRENCLAVE = a3f8c2...b4d6
2. **Runtime**:
    - Server creates enclave (`ECREATE, EADD, EINIT`)
    - Client requests attestation
    - Enclave generates report; Quoting Enclave signs → Quote
    - Client verifies Intel signature + MRENCLAVE match
    - Secure channel established (TLS to enclave)
3. Client encrypts medical image, sends to server
4. `EENTER`: Decrypt inside enclave, run ML inference, encrypt result, `EEXIT`
5. Client decrypts result

Security: Data encrypted in DRAM; cloud provider cannot see plaintext.

Aspect	**ARM TrustZone**	**Intel SGX**
Isolation	System-wide (two worlds)	Per-application (enclaves)
TCB Size	Larger (Trusted OS)	Smaller (enclave only)
Memory	TZASC/TZPC (MB-GB)	MEE encryption (limited EPC)
Attestation	Limited (vendor-specific)	Built-in (Intel IAS)
Secure UI	Yes (secure display/touch)	No (no I/O in enclave)
Use Cases	Mobile payments, DRM, biometrics	Cloud confidential computing

Table 1: TrustZone vs SGX Comparison

3.3 Comparison: TrustZone vs SGX

3.4 Conclusion

ARM TrustZone and Intel SGX represent distinct TEE approaches. TrustZone provides system-wide isolation suitable for mobile scenarios requiring secure UI. SGX offers fine-grained isolation with strong cryptographic guarantees, ideal for cloud confidential computing. Both enable secure execution in untrusted environments with different trust models and performance characteristics.

4 Question 4: TPM vs TEE - Comparison

4.1 Trust Boundary

Aspect	TPM	TEE
Boundary Type	Physical (discrete hardware chip)	Logical (CPU isolation modes)
Isolation	Separate processor, LPC/SPI bus	CPU security states (TrustZone) or encrypted memory (SGX)
Trusted Components	TPM chip internals only	Secure World/Enclave code
Untrusted	Everything outside TPM (CPU, OS, DRAM)	Normal World OS or untrusted application code
TCB Size	Very small (∼100KB firmware)	Larger (MB-scale Trusted OS or enclave)
Attack Surface	Minimal (command interface)	Moderate (execution environment)
Side-Channel Risk	Low (separate chip)	Higher (shared CPU resources)

Table 2: Trust Boundary Comparison

4.2 Execution Context

Aspect	TPM	TEE (TrustZone)	TEE (SGX)
CPU Speed	8-16 MHz	GHz (full CPU)	GHz (full CPU)
Memory	KB-scale (NV + volatile)	MB-scale	MB-scale (EPC limited)
OS Environment	None (firmware)	Trusted OS (OP-TEE)	Library OS (optional)
Arbitrary Code	No	Yes (Trusted Apps)	Yes (Enclave code)
I/O Access	None	Secure peripherals	None (via OCALL)
Performance	Very slow (100ms-seconds)	Near-native (1-5μs switch)	Near-native (10-15% overhead)
Capabilities	Crypto operations, key storage, PCR, seal/unseal	Full computation, secure UI, crypto	Full computation, memory encryption, no I/O

Table 3: Execution Context Comparison

4.3 Use Case Analysis

4.3.1 Use TPM For:

1. Measured Boot & Attestation

- **Example**: Enterprise laptop with BitLocker
- **TPM Role**: Measures boot chain, seals disk key to PCRs, remote attestation to IT admin
- **Why TPM**: Persists through reboots, platform-wide view, hardware root of trust

2. Long-Term Key Storage

- **Example**: IoT device authentication
- **TPM Role**: SRK-based key hierarchy, TPM-bound keys, dictionary attack protection
- **Why TPM**: Hardware-bound (cannot be exported), survives reboot

4.3.2 Use TEE For:

1. Runtime Secure Processing (TrustZone)

- **Example**: Mobile payment (Google Pay)
- **TEE Role**: Secure UI for PIN entry, transaction signing in Secure World
- **Why TEE**: Secure I/O, real-time performance, protects runtime secrets

2. Confidential Cloud Computing (SGX)

- **Example**: Medical ML inference on encrypted patient data
- **TEE Role**: Decrypt and process data inside enclave, cloud provider cannot access
- **Why TEE**: Protects data from cloud provider, full computational capability

4.3.3 Use Both (TPM + TEE):

Secure Enterprise Workstation

- **Boot**: TPM measures boot chain, seals disk key
- **Runtime**: SGX enclave processes classified documents
- **Attestation**: Combined TPM quote (boot integrity) + SGX quote (app integrity)
- **Benefit**: Defense-in-depth covering full stack (boot to runtime)

4.4 Selection Criteria

4.5 Conclusion

TPM and TEE are complementary technologies addressing different security needs. TPM provides a passive hardware root of trust for platform integrity, measured boot, and secure key storage with minimal attack surface. TEE provides active execution environment for trusted applications with full computational capabilities.

TPM's physical trust boundary (discrete chip) differs from TEE's logical boundary (CPU isolation). TPM excels at boot-time verification and long-term storage, while TEE enables runtime processing of sensitive data. Modern secure systems deploy both: TPM establishes trust from first instruction, TEE maintains isolation throughout runtime, providing comprehensive protection across the full system lifecycle.

Requirement	TPM	TEE
Verify boot integrity	✓	
Long-term key storage	✓	
Platform attestation	✓	
Execute trusted code at runtime		✓
High computational throughput		✓
Secure UI (PIN entry, display)		✓(TrustZone)
Cloud confidential computing		✓(SGX)
Low power budget	✓	

Table 4: Selection Criteria

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
```

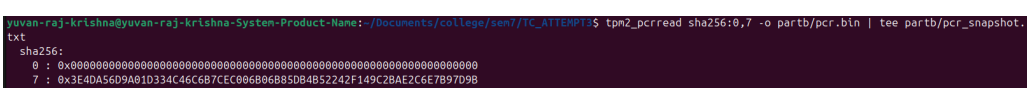

Figure 1: Initial PCR values (PCR 0: BIOS, PCR 7: Platform-specific)

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
    "
```

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

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

# 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"
```

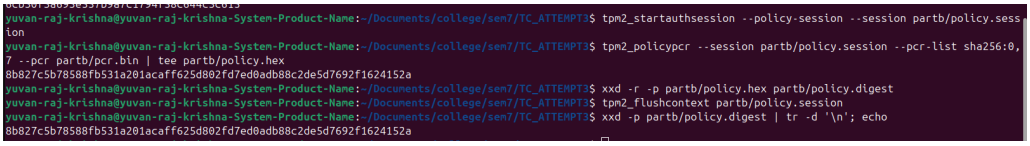

Figure 3: PCR policy digest bound to PCR 0 and 7

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
```

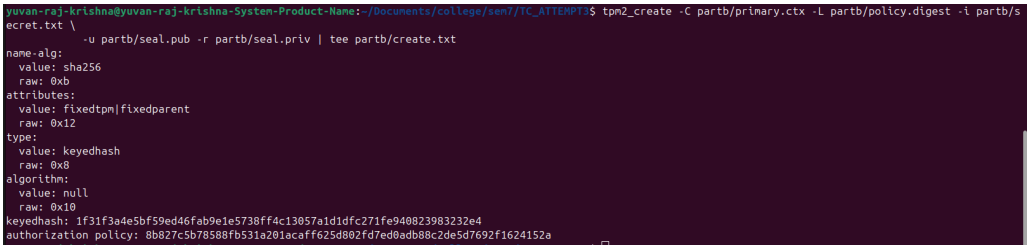

Figure 4: Sealed object with embedded PCR policy

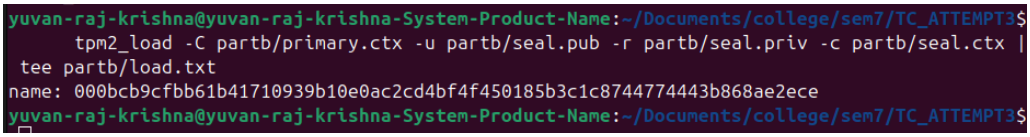

Figure 5: Sealed object loaded into TPM with unique object name

5.7 Task 5: Unseal with Correct PCRs (Success)

Commands:

```
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
```

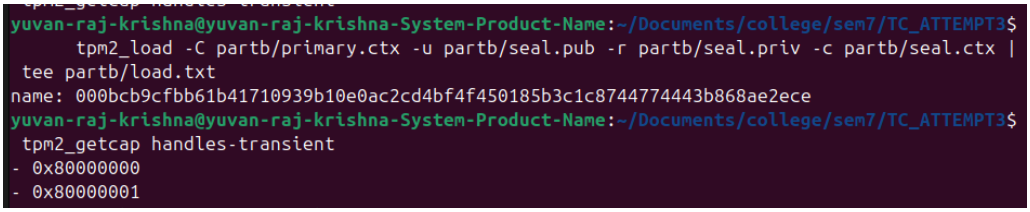

Figure 6: Policy session satisfying PCR requirements

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.

5.9 Task 7: Unseal with Modified PCRs (Failure)

Commands:

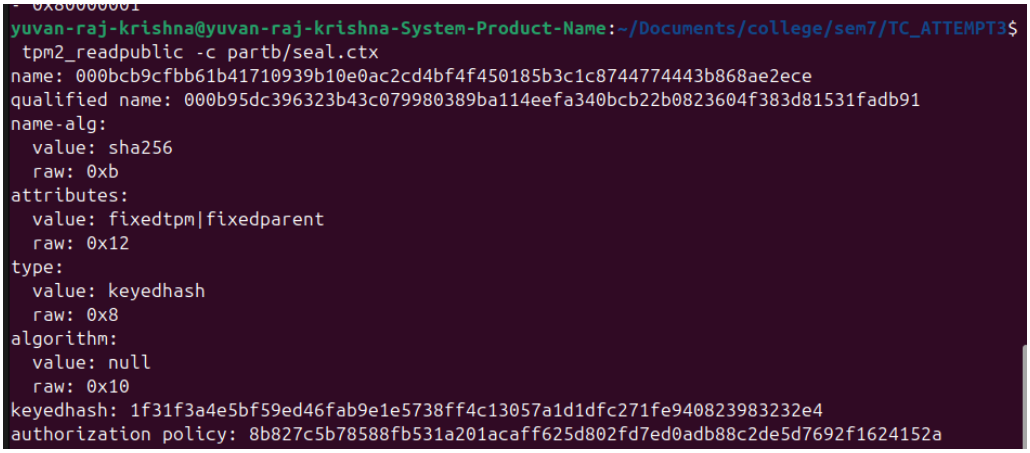

Figure 7: Successfully unsealed secret: "Confidential lab secret for Assignment 2"

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

```
# 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

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

Figure 10: Flushing transient handles and sessions

- ✓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