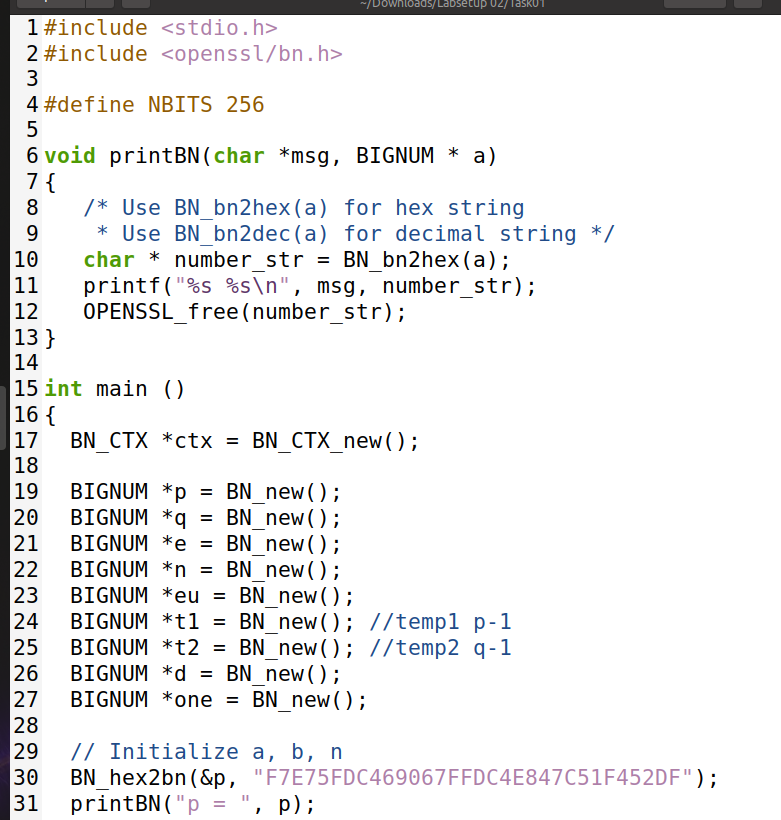
**IS Assignment 02**

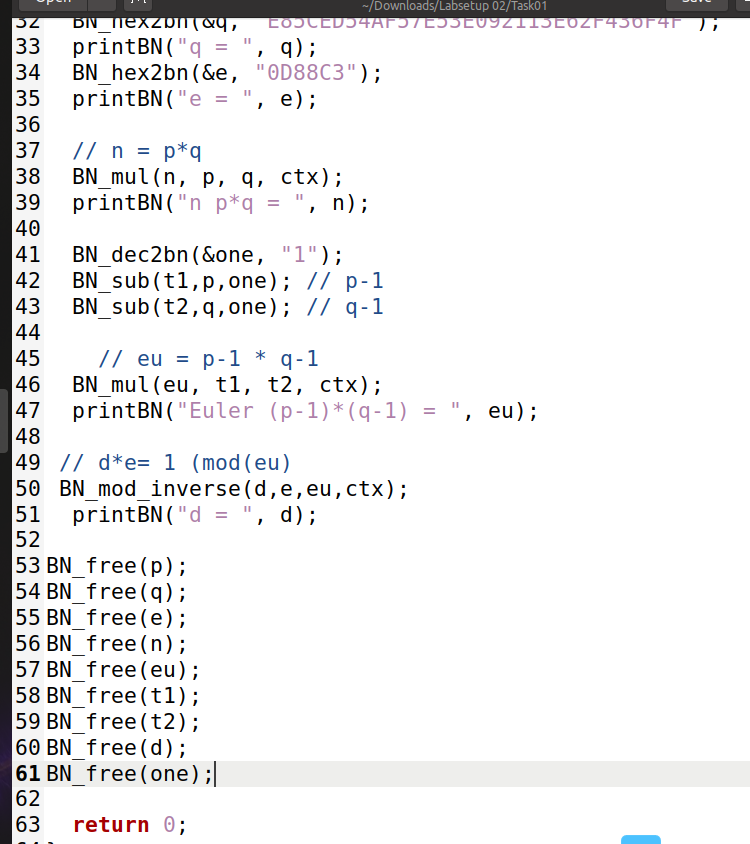
**Insha Javed 21k-3279**

**Muhammad Tahir 21k-4503**

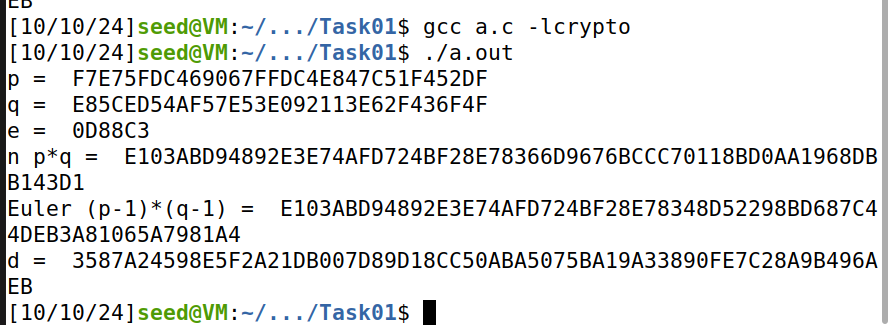
**Task01**

**Code:**



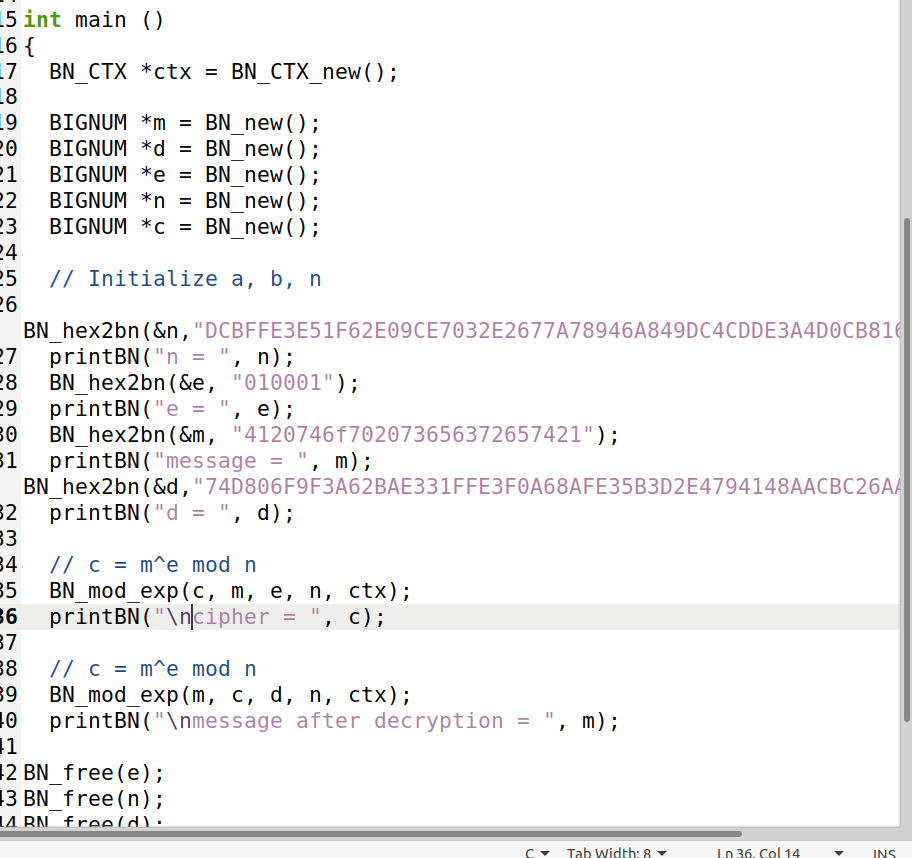


**OutPut:**

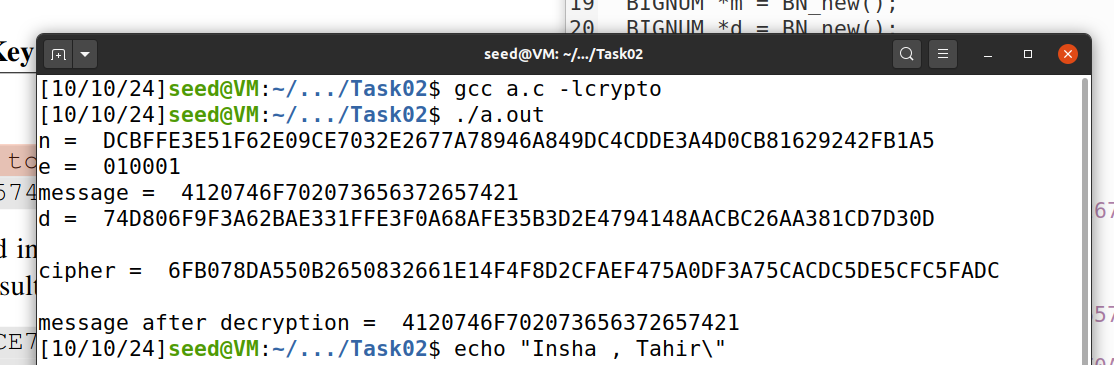


**Task02**

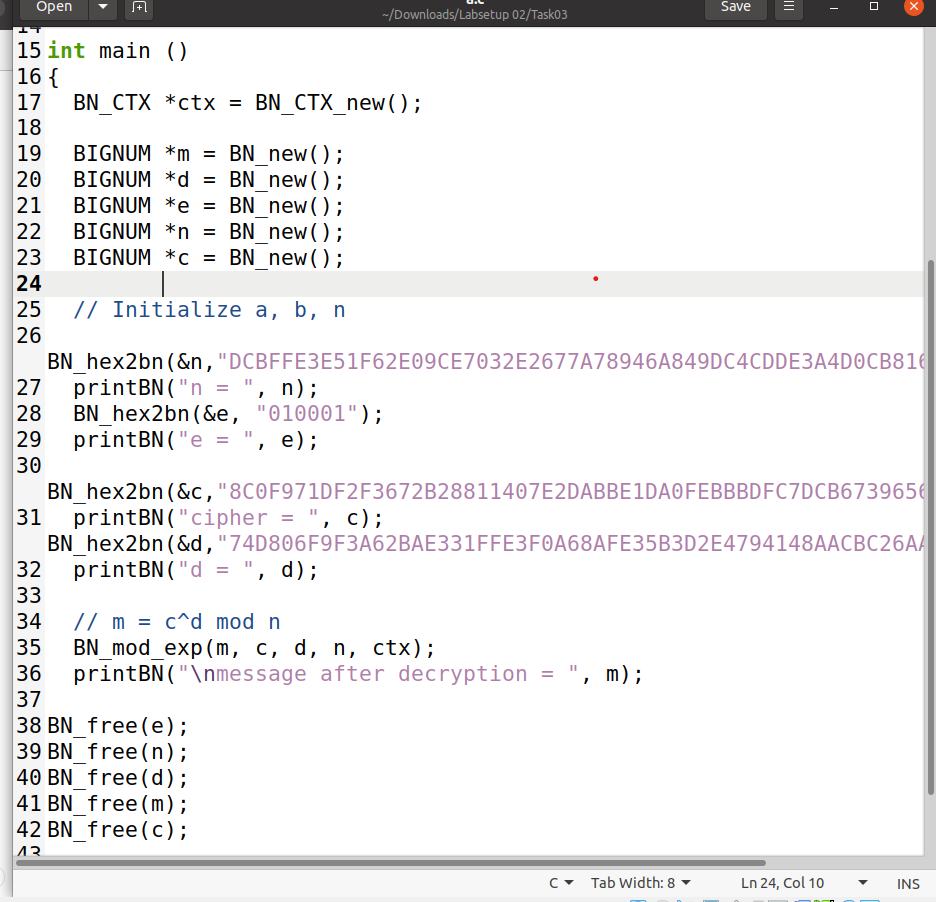
**Code:**

****

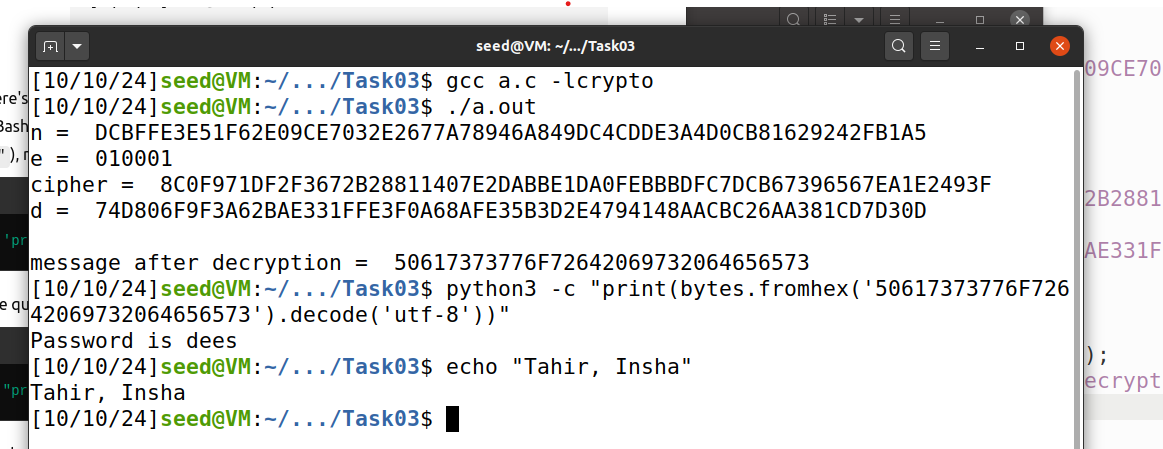
**Output:**

****

**Task03**

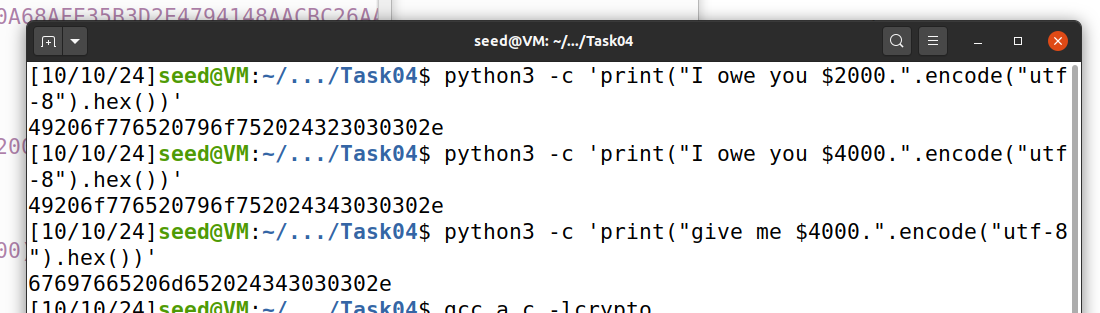
**Code:  
**

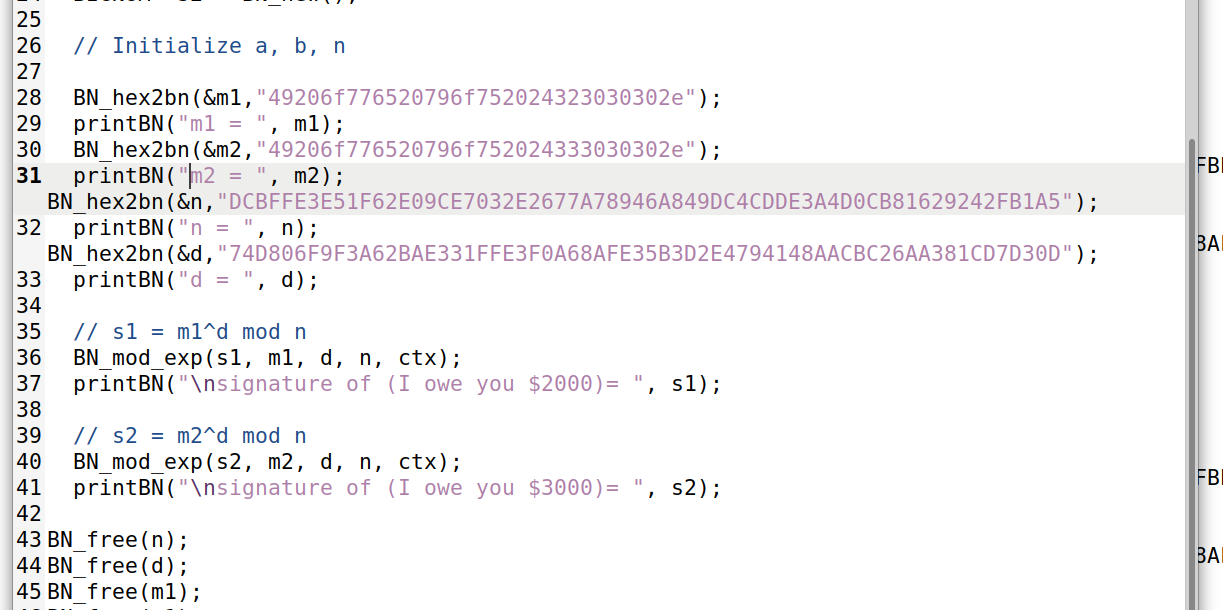
**Output:**

****

**Task04**

**Code:**

****

****

**Output:**

****

**Observation:**

When signing the messages:

**Original Message1:** "I owe you $2000."

**Signature:** 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

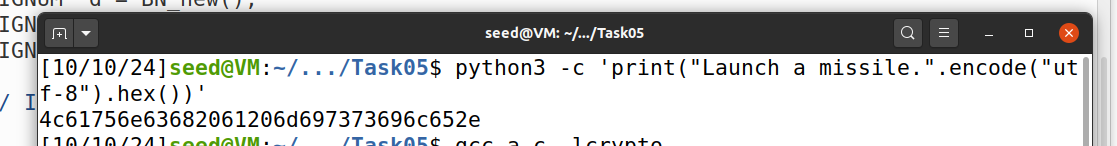
**Modified Message2:** "I owe you $3000."

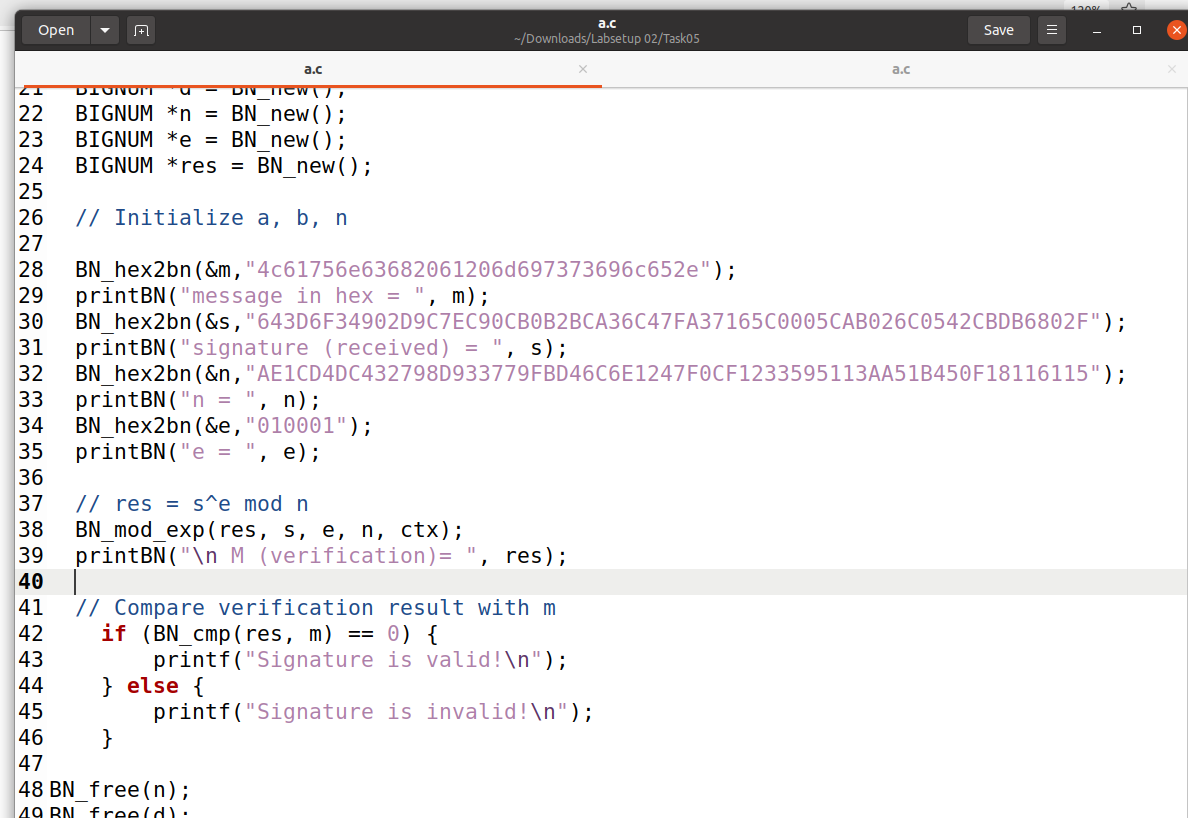
**Signature:** BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822

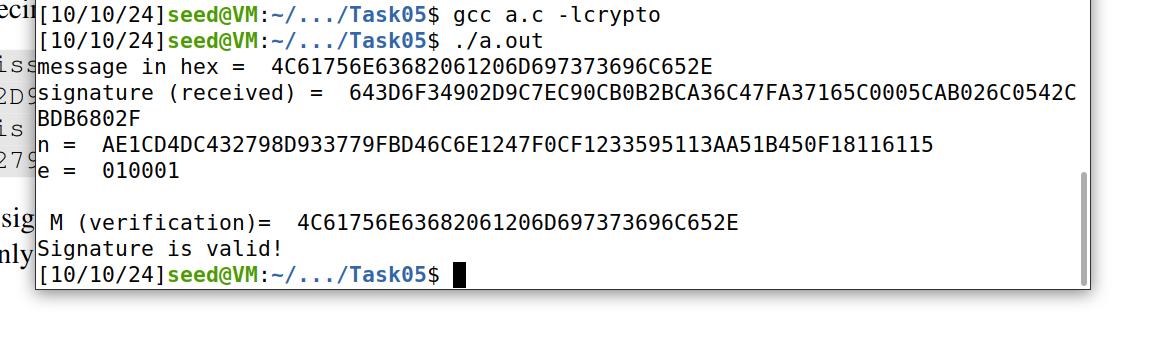
1. **Unique Signatures:** Each message generates a completely different signature, even with a minor change (from $2000 to $3000). This shows the sensitivity of digital signatures to message content.
2. **Integrity Assurance:** The different signatures ensure that any modification to the message invalidates the original signature, confirming the message's integrity.
3. **Non-reusability:** Signatures are not interchangeable; altering the message results in a distinct signature, preventing fraud.

**Task05**

**Code:**

****

****

**Output:  
**

Modifying the last 2 bits of the signature:

BN\_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

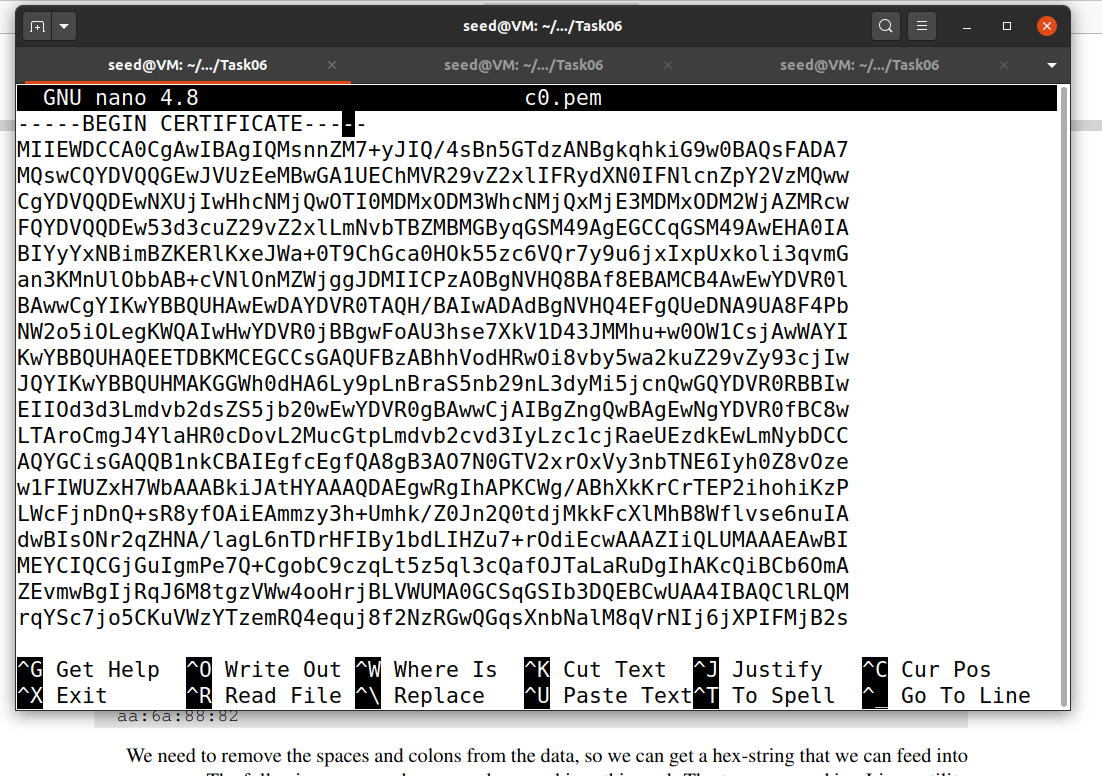
To

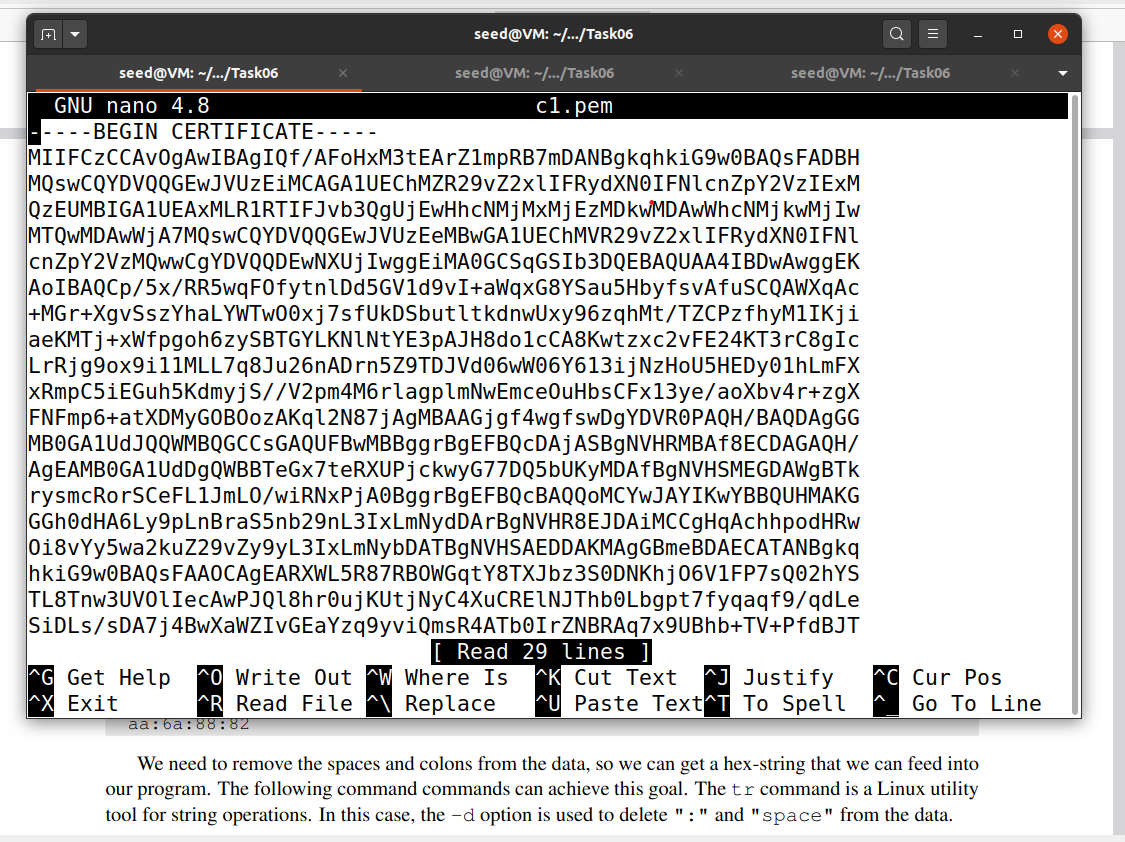
BN\_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

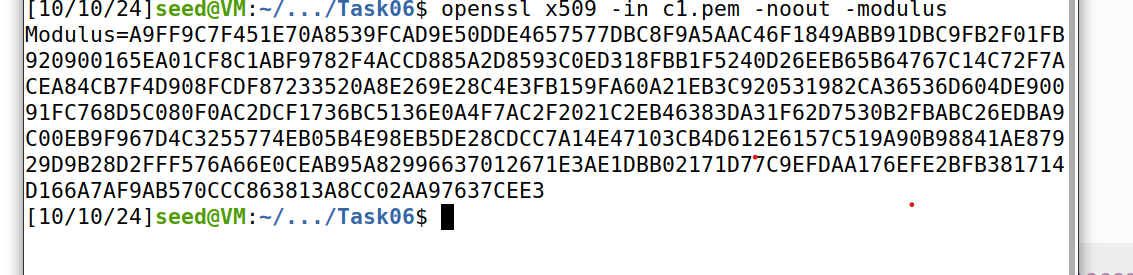


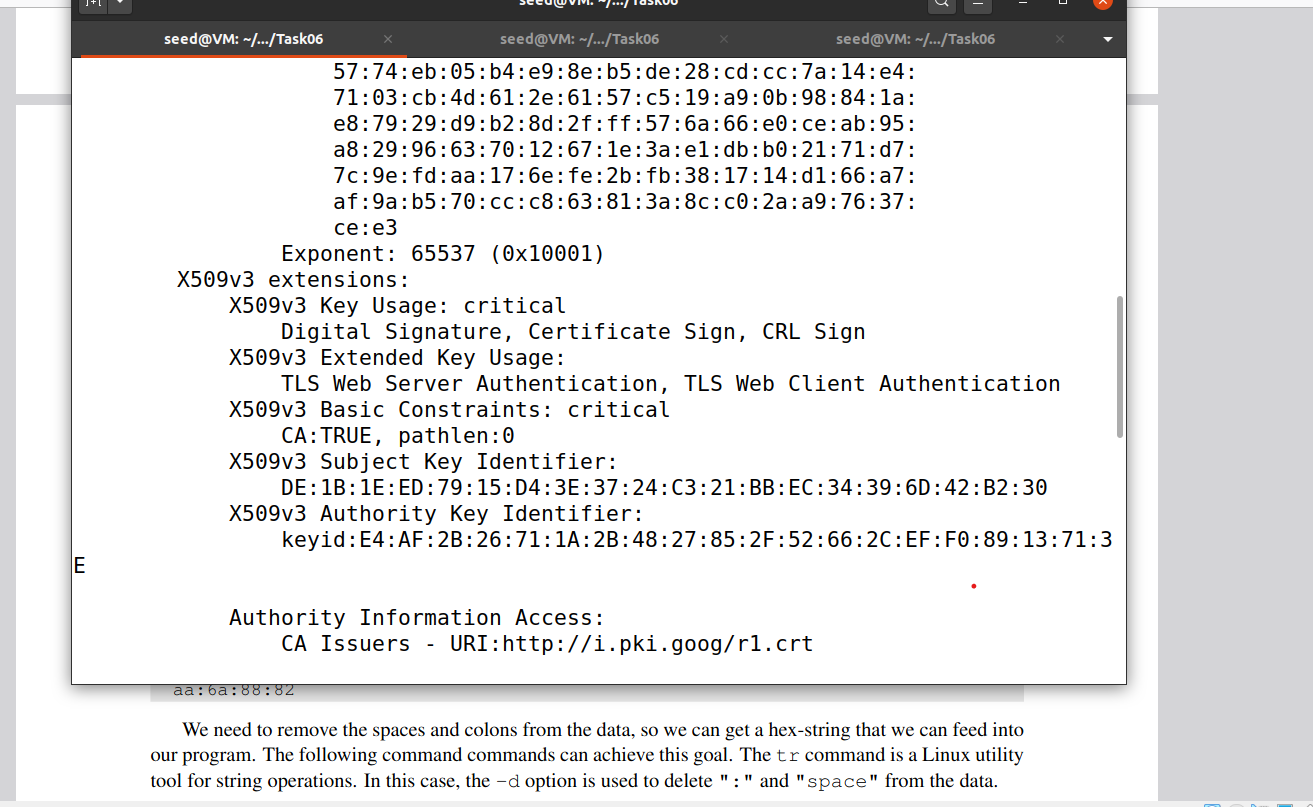
**Task 06:**

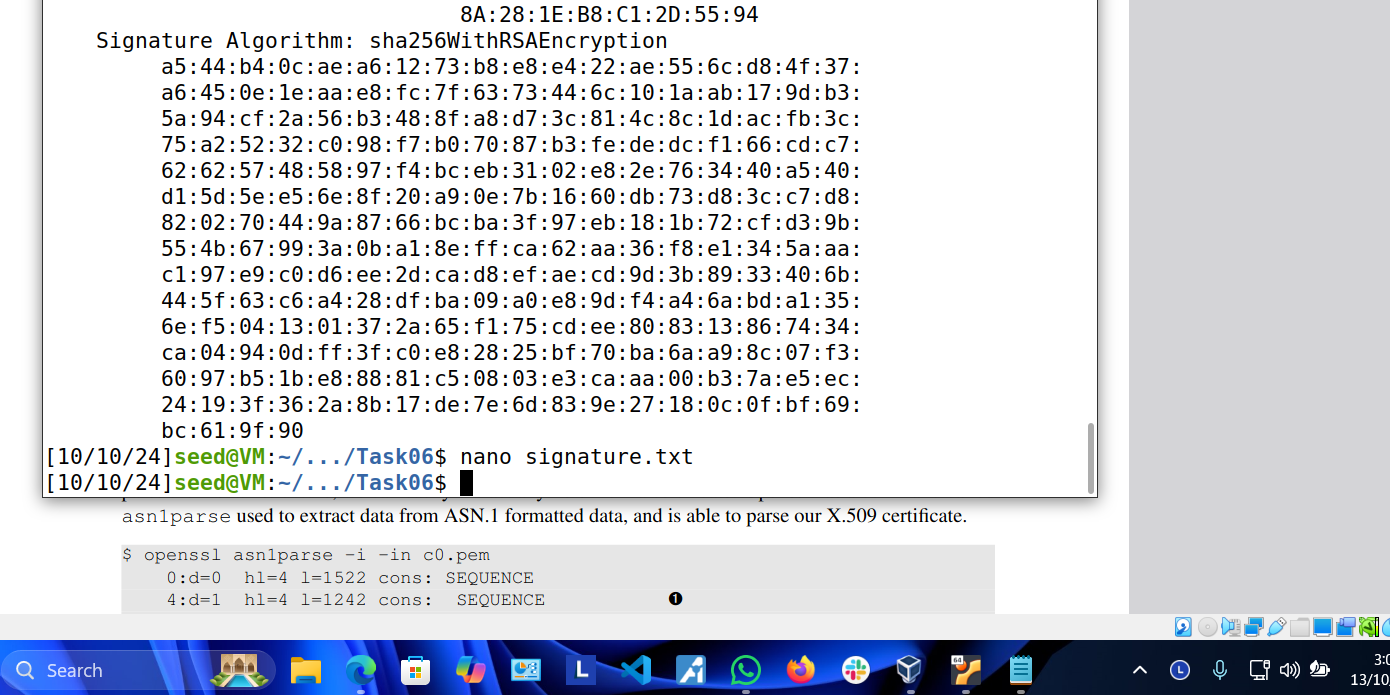
**Code:  
C0:**

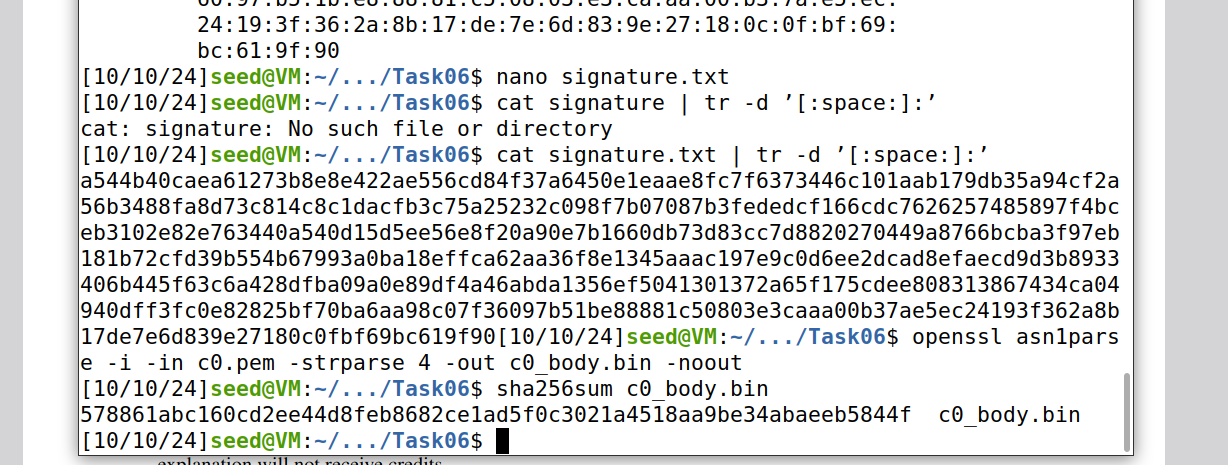
****

**C1:  
**

****

****

****

****

**CODE:**

**Explanation:  
Here's a minimal explanation of the code:**

**1. \*\*Initialization\*\*:**

**- Several `BIGNUM` variables are initialized to hold the modulus (`mod\_n`), public exponent (`e`), signature, and body hash.**

**- The public key (`mod\_n`, `e`), the signature, and the expected hash (`body\_hash`) are loaded using `BN\_hex2bn`.**

**2. \*\*Signature Decryption\*\*:**

**- The RSA signature is decrypted using `BN\_mod\_exp` (which computes `signature^e mod n`).**

**3. \*\*Hash Extraction\*\*:**

**- The last 32 bytes of the decrypted signature (SHA-256 hash) are extracted using `BN\_bn2binpad` directly into a fixed-size array.**

**4. \*\*Hash Comparison\*\*:**

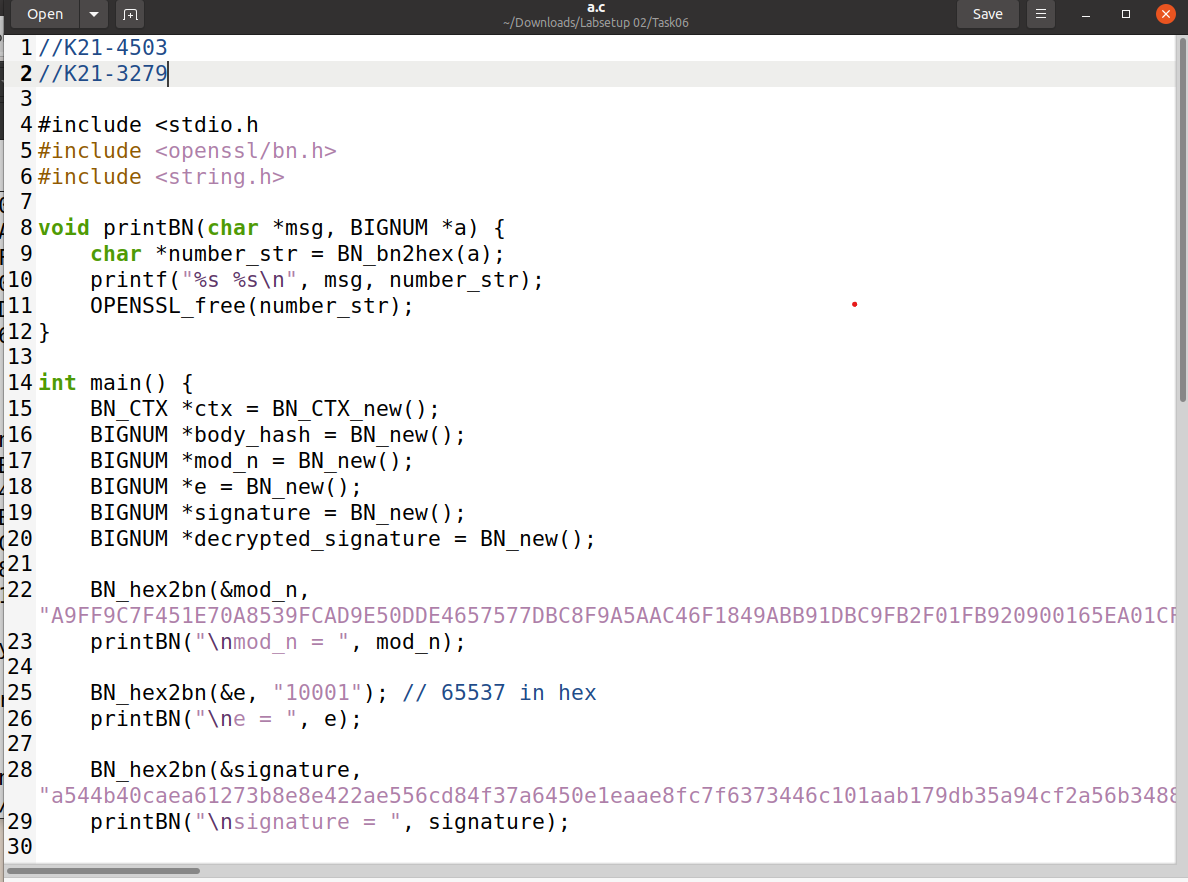
**- The extracted hash from the decrypted signature is compared with the actual hash (`body\_hash`).**

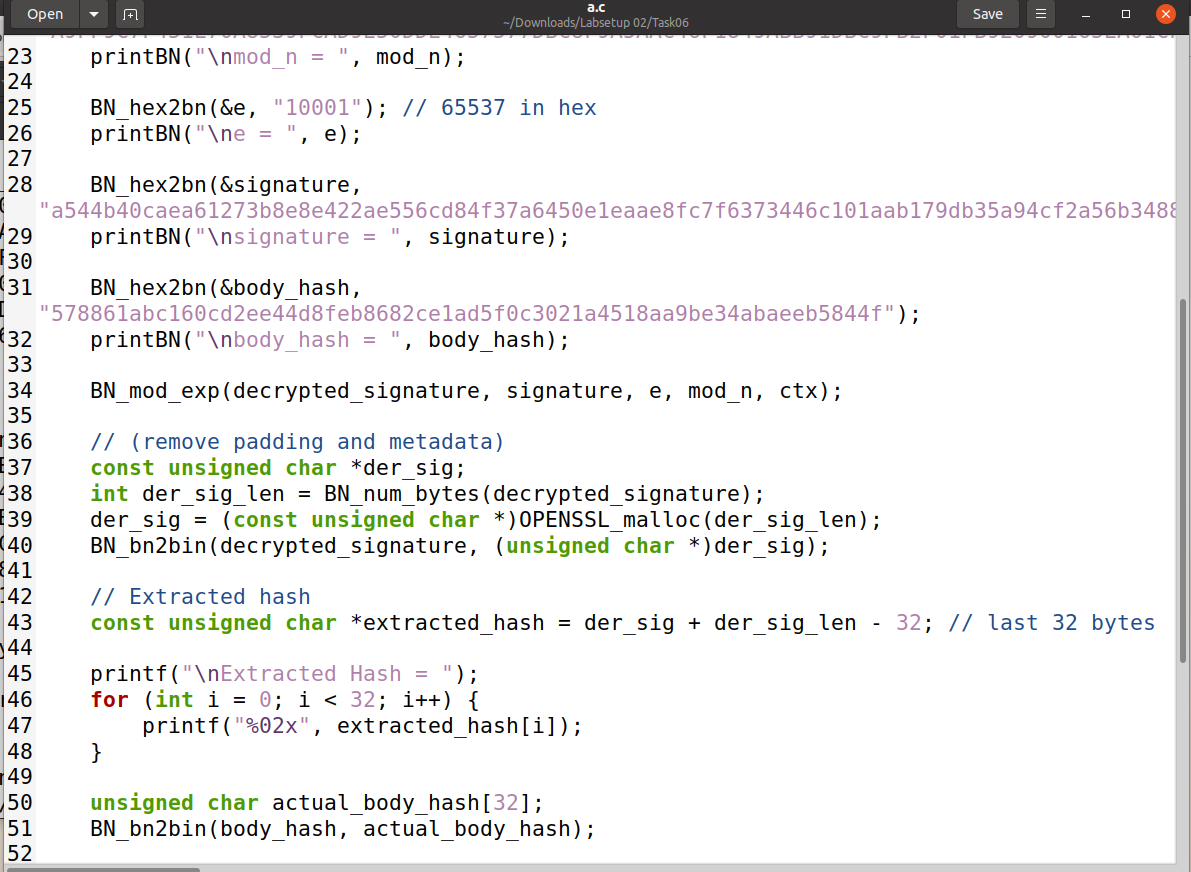
**- If they match, the signature is valid; otherwise, it is invalid.**

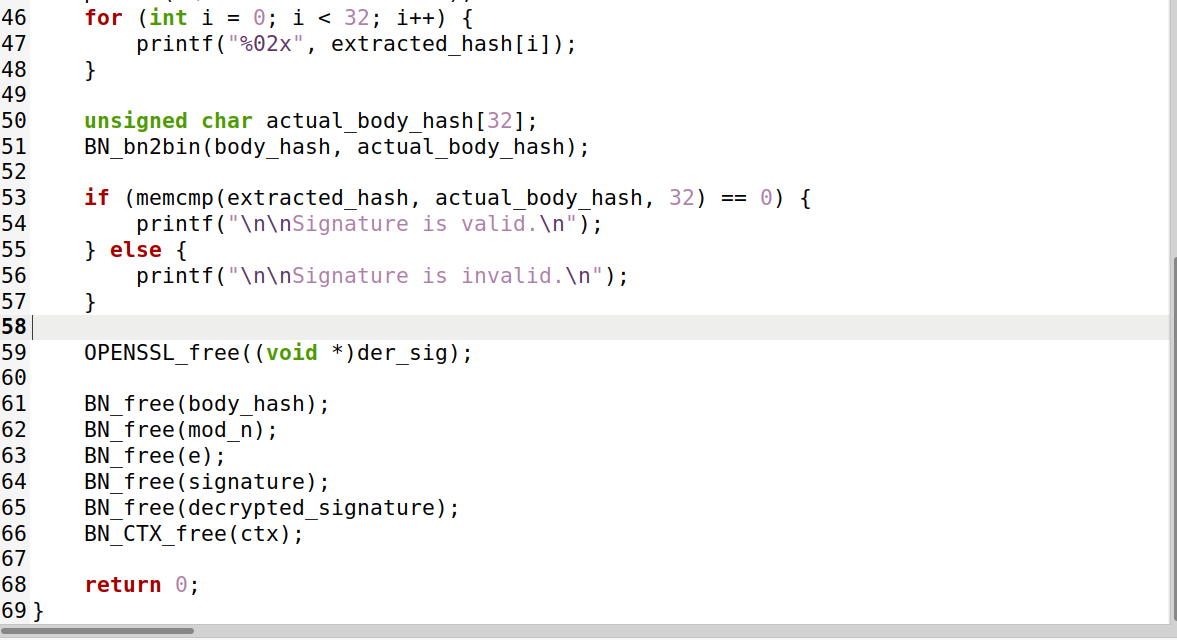
**5. \*\*Memory Cleanup\*\*:**

**- All allocated `BIGNUM` variables are freed to avoid memory leaks.**

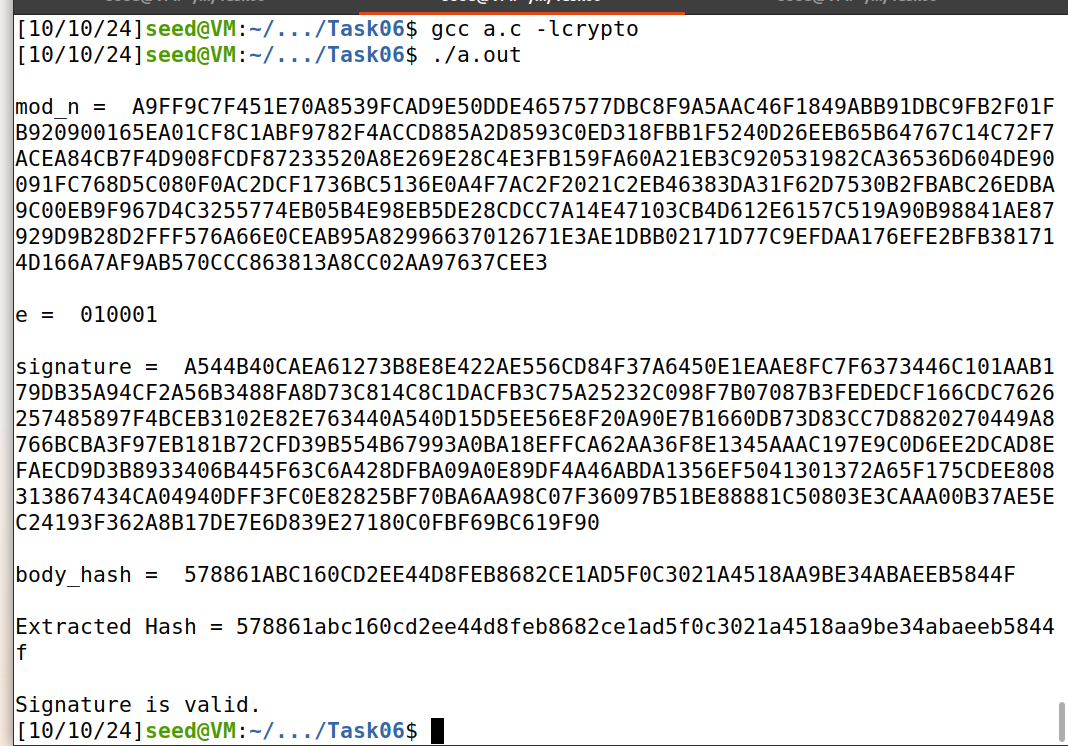
**This code verifies an RSA signature by decrypting it and comparing the hash in the signature with the expected SHA-256 hash.**

****

****

****

**Output:**

****