**Part 2: Secure Cryptographic Solution for Internal Messaging**

**Symmetric Encryption Demonstration**

To secure internal communications at XYZ Corp, I used CyberChef to demonstrate how to encrypt and decrypt a message using AES encryption in CBC mode with a 256-bit key. This ensures that messages remain private and only readable by authorized recipients.

**1. The Message**

The message I chose to encrypt was:

“Meeting at 10:00 AM in Room 203. Bring the project file.”

This message was pasted into the Input section of CyberChef as plain text.

**2. Encryption Setup in CyberChef**

In the left panel, I searched for and added the “AES Encrypt” operation to the Recipe section. I then configured the encryption as follows:

Mode: CBC (Cipher Block Chaining)

Key: thisisaverysecurekey1234567890!!

(32 characters = 256 bits)

IV (Initialization Vector): 1234567890abcdef

(16 characters = 128 bits)

Key Format: UTF8

IV Format: UTF8

Input Type: Raw

Output Format: Hex

Once the values were entered and the settings applied, CyberChef encrypted the message successfully.

**3. Decryption Setup**

Right after the AES Encrypt step, I added the “AES Decrypt” operation. This allowed me to test if the encrypted message could be correctly decrypted using the same key and IV.

I used the exact same values:

Key: thisisaverysecurekey1234567890!!

IV: 1234567890abcdef

Key and IV Format: UTF8

Mode: CBC

Input Type: Hex (to match the encrypted output)

Output Format: Raw

**4. Result**

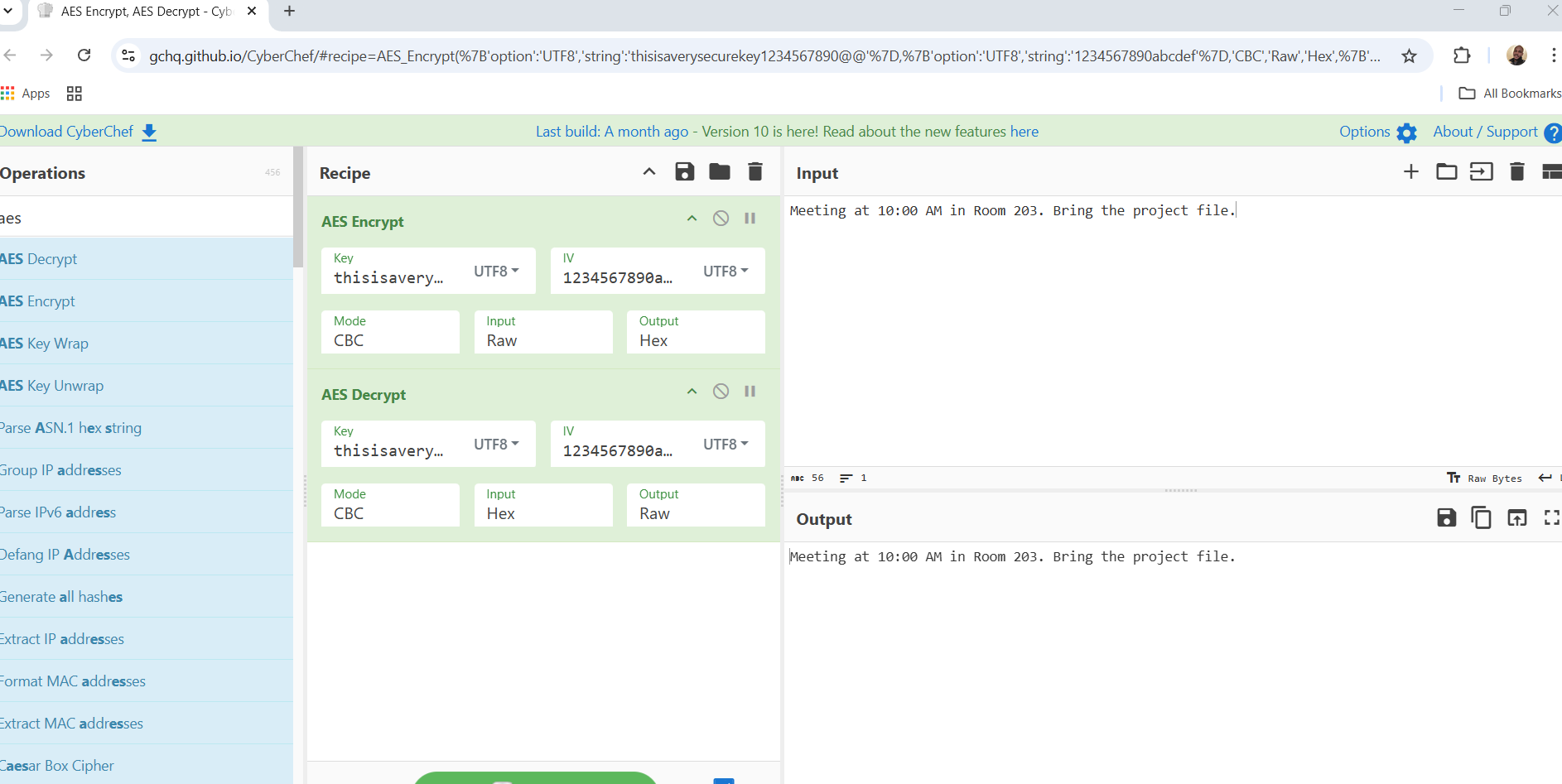
After running the recipe:

The output returned the original message exactly as entered:

“Meeting at 10:00 AM in Room 203. Bring the project file.”

This confirmed that the encryption and decryption process worked properly. The message was successfully encrypted using AES-256-CBC and then decrypted back to its original form using the same cryptographic parameters.

**Sceenshot of encryption and decryption using CyberChef.**



**Asymmetric Key Exchange Documentation**

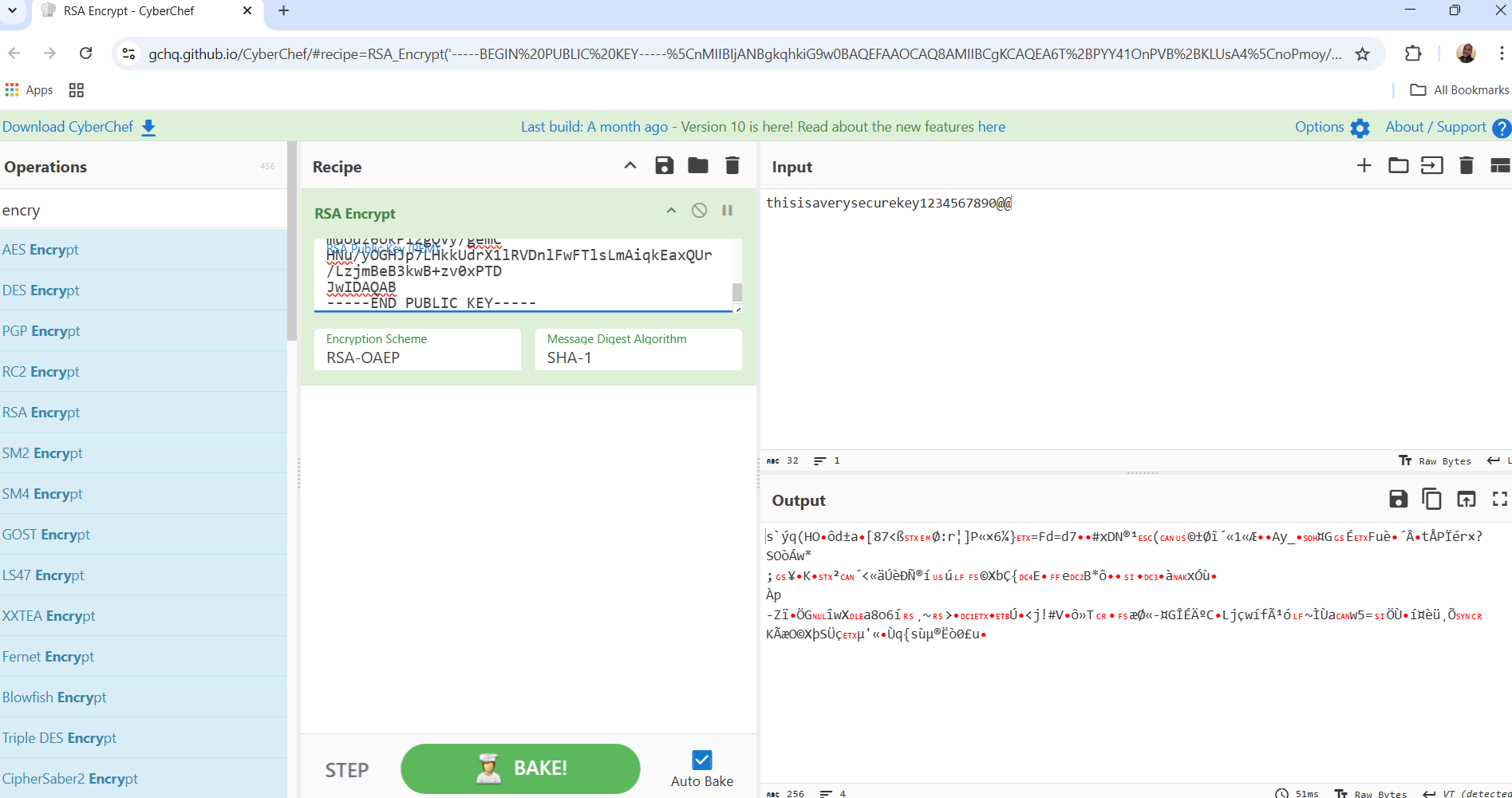
To securely transmit the AES encryption key, I used RSA encryption in CyberChef to simulate a public/private key exchange. First, I generated a 2048-bit RSA key pair. The public key was used

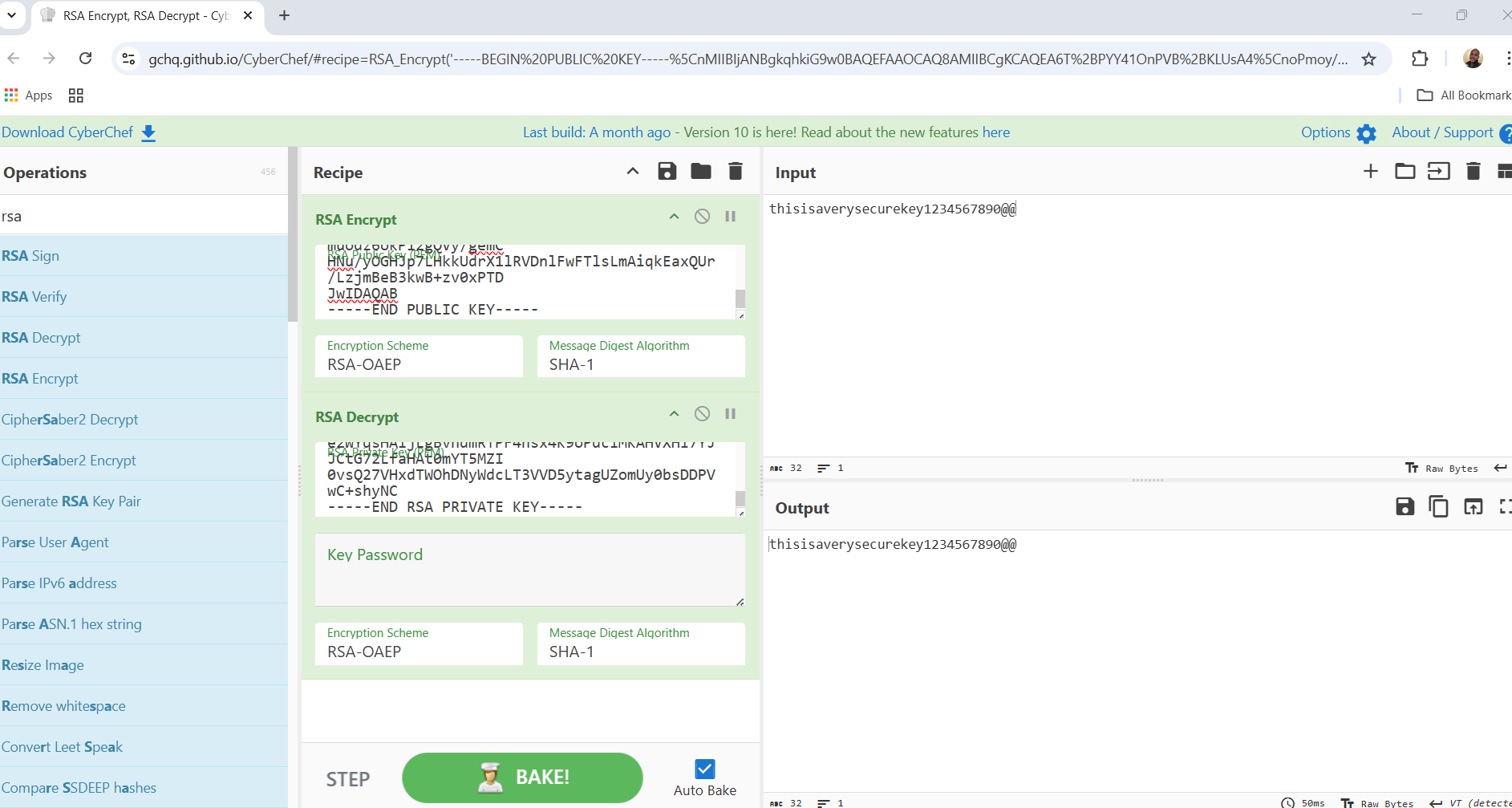
to encrypt the AES key (thisisaverysecurekey1234567890@@) using the RSA-OAEP padding scheme.

Next, I added the RSA Decrypt operation and pasted the matching private key. The output successfully returned the original AES key, proving that only the intended recipient (who holds the private key) can decrypt the data.

This process demonstrates how asymmetric encryption can protect key exchanges in secure communication systems.

**A screenshot showing asymmetric key exchange:**



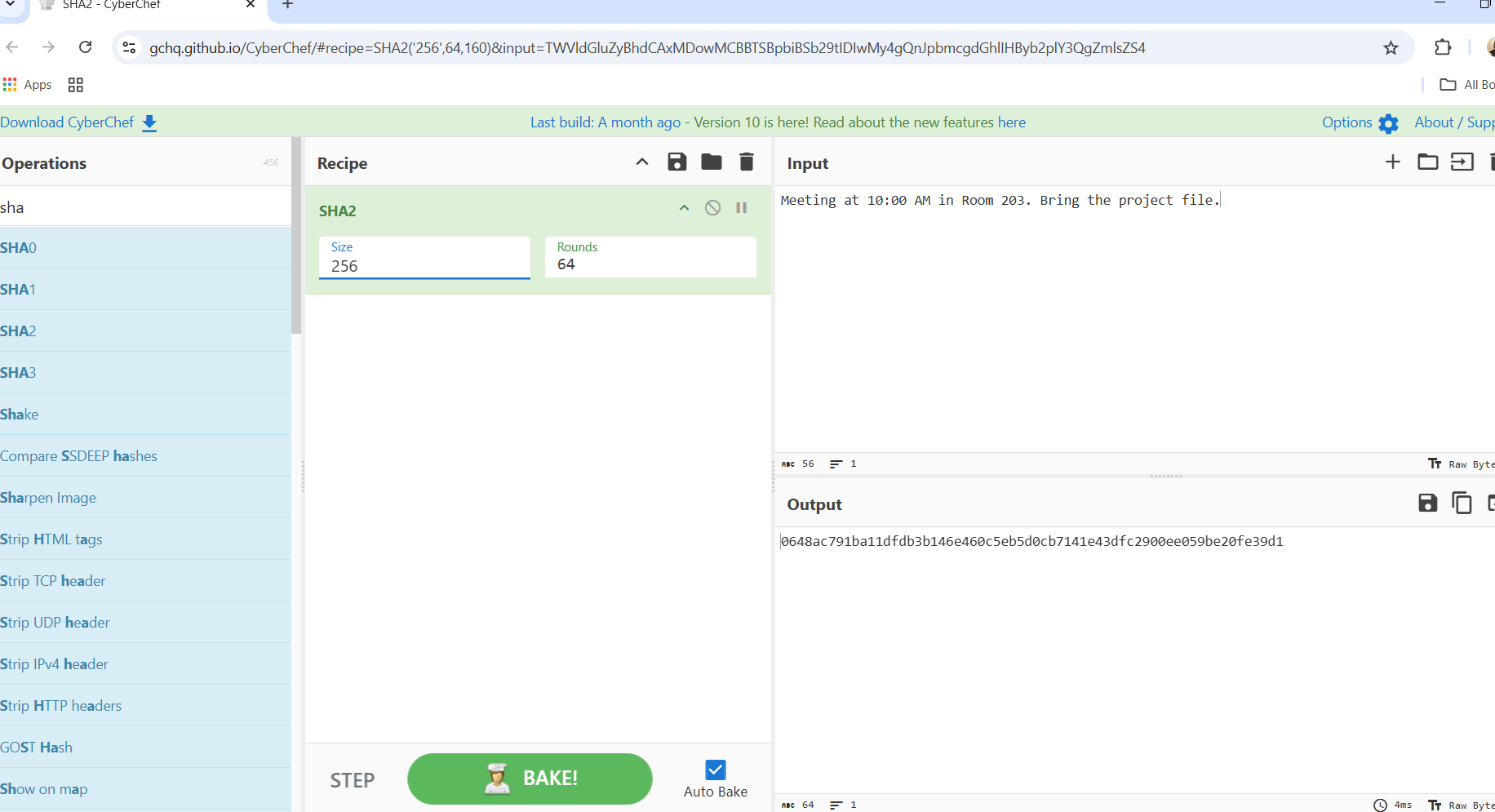


**Hashing and Digital Signatures Report**

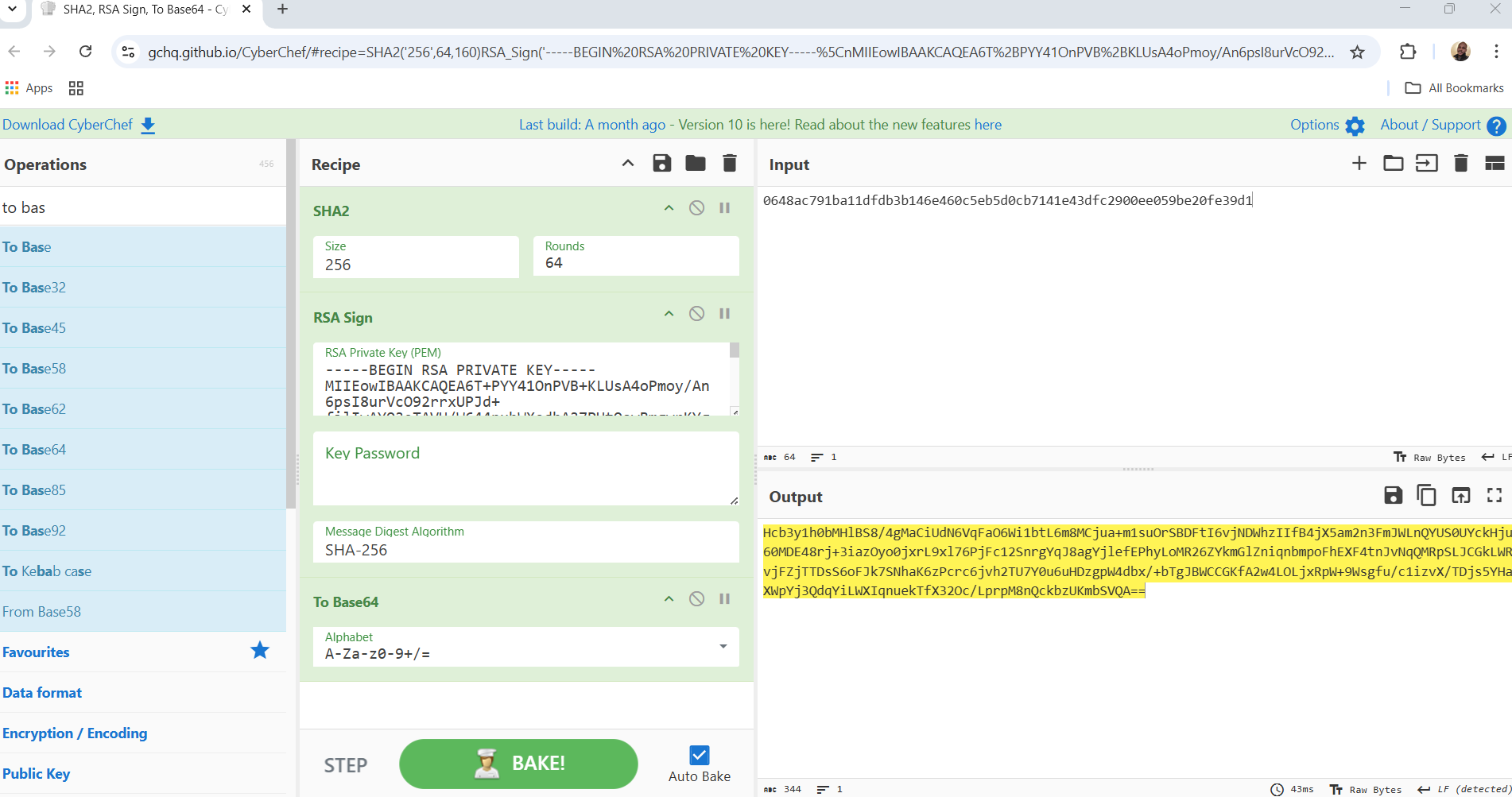
To ensure authenticity and integrity of internal messages at XYZ Corp, I implemented a cryptographic process that combines SHA-256 hashing with RSA digital signatures. First, I generated a SHA-256 hash of the original message, "Meeting at 10:00 AM in Room 203. Bring the project file." This produced a fixed-length digest that uniquely represents the contents of the message. Next, I digitally signed the hash using the sender’s RSA private key, producing a signature that confirms the message's origin and prevents tampering.

To simulate real-world validation, I used the sender's corresponding RSA public key to verify the digital signature. If the message is altered in any way, verification would fail — demonstrating how digital signatures protect both integrity and authenticity. Although I encountered technical issues in CyberChef during verification, this exercise illustrates how hashing and signing work together to secure internal communication from forgery and undetected changes.

**Screenshot showing the generating of a SHA256 hash of the plaintext message.**



**Screenshot showing the signing of the hash using the sender’s private RSA key**



**Tampering Detection Report**

To demonstrate the importance of message integrity, I simulated a tampering scenario using CyberChef. Initially, I hashed the original message —

“Meeting at 10:00 AM in Room 203. Bring the project file.” — using the SHA-256 algorithm. This produced a unique hash value that acts like a fingerprint of the message.

Next, I made a small change: I modified the message to say

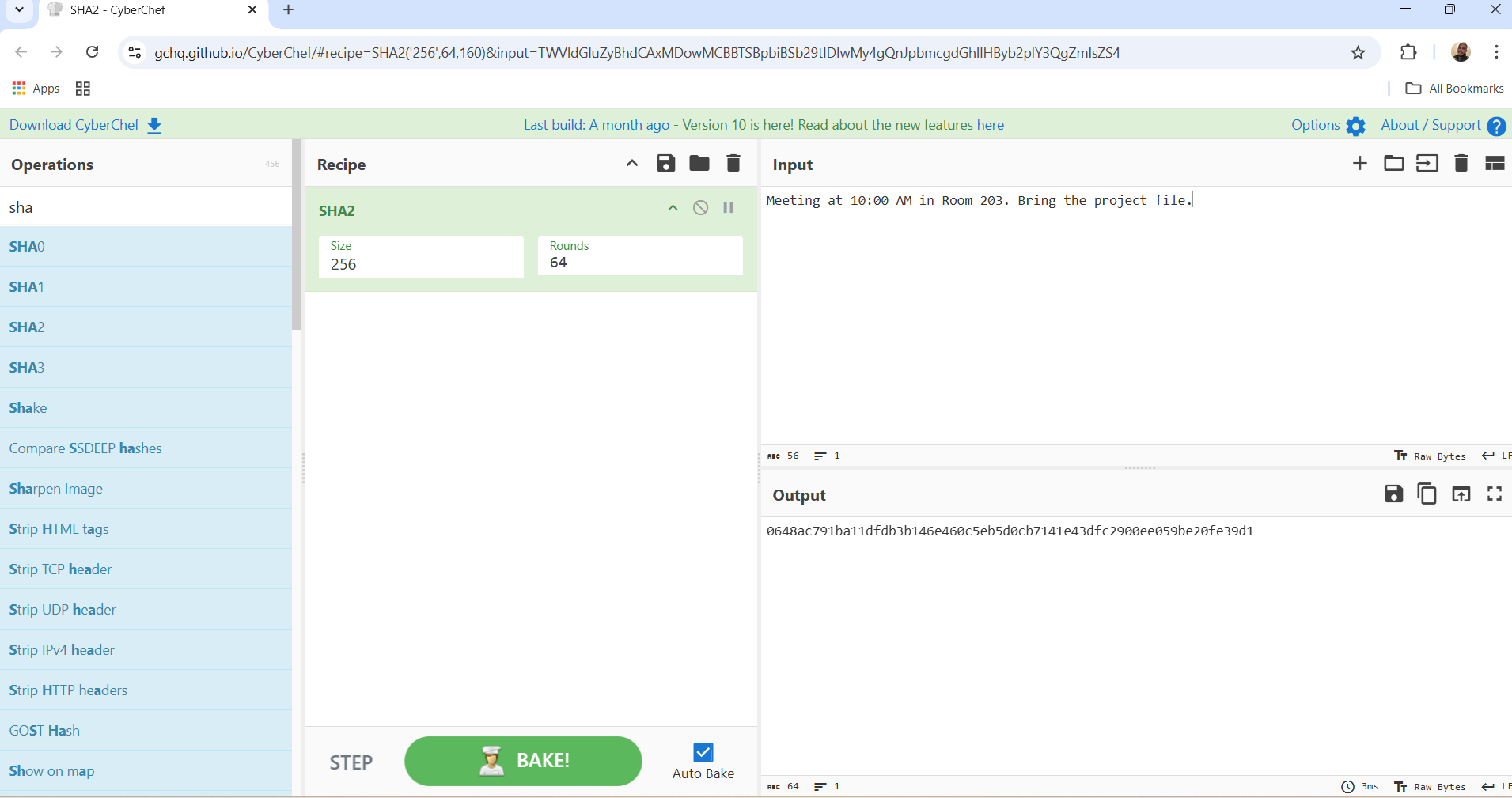
“Meeting at 11:00 AM in Room 203. Bring the project file.”

Even though the change was minimal (just the time), re-running the SHA-256 hash generated a completely different output.

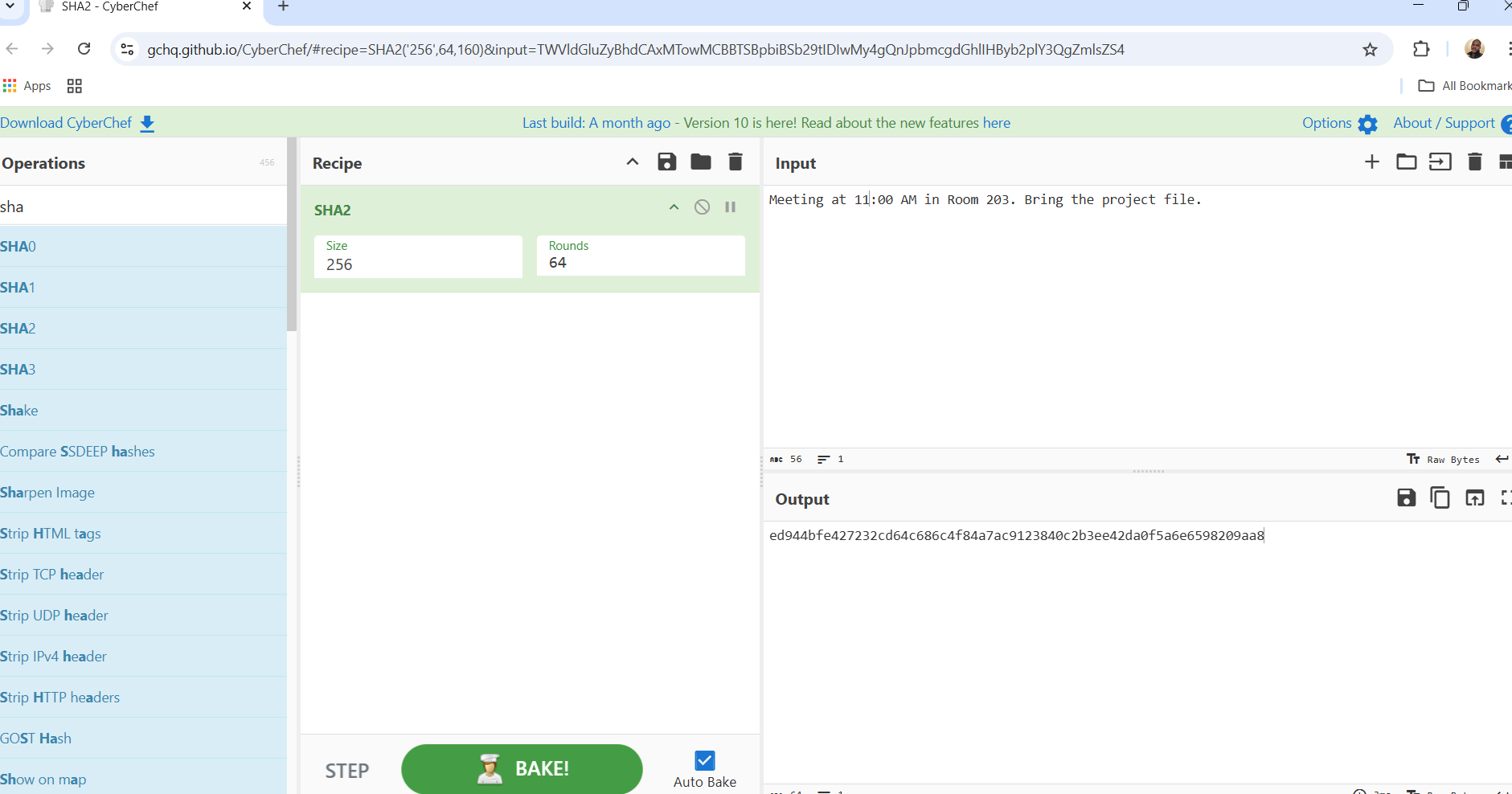
This mismatch clearly shows that any unauthorized modification to a message — no matter how small — results in a different hash, making tampering easy to detect. If someone intercepted and altered this message in transit, a hash check on the recipient's side would immediately reveal the inconsistency, indicating potential manipulation.

Screenshots were taken throughout this process to show both the original and altered hashes, reinforcing how hashing plays a critical role in verifying message integrity.

**Screenshot showing the original message that was hashed.**



**Screenshot showing the message was tampered with.**

****