The script I wrote upholds all thee areas of the CIA Triad.

**Confidentiality**

* In an effort to protect the data from unauthorized access, this script not only hashes the user’s input, but it also encrypts that hashed value with the use of the Fernet key that is generated.

**Integrity**

* In order to confirm that the data hasn’t been manipulated and that is in fact accurate, the script does a comparison at the end via an if/else conditional. The screen prints “Integrity Verified!” if the conditional statement is satisfied.

**Availability**

* This script hashes and encrypts the inputted data with hashlib and cryptography.fernet libraries, both of which are the most used in Python when it comes to securing data, making it a reliable tool. The required resources are minimum, making it much easier for anyone to be able to add input. The while loop also keeps the user in the program via continuously asking for input that satisfies the current five character minimum. This ensures that it remains usable.

Entropy is a measure of randomness, uncertainty, or disorder within a system. With that said, this script guarantees a high level of entropy but not only hashing the user’s input but then generating an encryption key from the hashed value of the provided input.

Key generation is satisfied by the use of Fernet.generate\_key() as it creates a randomized key each time this script is run. Even if the key was somehow compromised, when it is used it would still only result in the hashed text, not the true value provided by the user themselves.

First, the program asks the user for an input message and checks that it’s at least five characters long. This ensures we have valid data to work with.

Next, the input is hashed using the SHA-256 algorithm. Hashing converts the message into a fixed-length hexadecimal string that cannot be reversed back into the original text. I print this so we can see the hashed version of the user’s input.

After that, the program generates a random encryption key using Fernet and creates an encryption object with it. With this key, the hashed message is encrypted into ciphertext, which I also display.

Then, the encrypted value is decrypted back to its original hashed form. Notice that we don’t get the raw user input back — we get the hash that we encrypted earlier.

Finally, the script compares the original hashed value with the decrypted one. If they match, it proves the process preserved the data’s integrity, and we print a confirmation message.