HMAC Example and Explanation

Python Code

```
import hmac
import hashlib

SECRET_KEY = b'supersecretkey'  # Same key, now used securely

def generate_mac(message: bytes) -> str:
    return hmac.new(SECRET_KEY, message, hashlib.md5).hexdigest()

def verify(message: bytes, mac: str) -> bool:
    expected_mac = generate_mac(message)
    return mac == expected_mac
```

What is HMAC?

HMAC (Hash-based Message Authentication Code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It provides both data integrity and authentication. The strength of HMAC comes from the cryptographic strength of the underlying hash function and the size and secrecy of the key.

Why the Attack Would Fail on HMAC

Without knowing the secret key, an attacker cannot generate a valid HMAC for a modified message. Unlike simple hashes, HMAC includes the secret key in its calculation, which protects against tampering and forgery. Even if an attacker knows the hash function used (e.g., MD5), they cannot create a valid HMAC without the secret key. This makes HMAC secure against length extension and other attacks that affect some hash functions.

How Does HMAC Work?

HMAC works by combining a secret key with the message, hashing the result, then combining the hash again with the secret key and hashing once more. Specifically, it performs the following steps:
1. The key is shortened or padded to a standard length.
2. Two fixed-length pads (inner and outer) are created by XOR-ing the key with specific constants.

3. The message is concatenated with the inner pad and hashed.
4. The result is then concatenated with the outer pad and hashed again.
This dual application of the hash function ensures resistance to certain cryptographic attacks.

Additional Details About HMAC

HMAC is defined in RFC 2104, and is used in:
- HTTPS / TLS
- AWS API authentication
- JWT tokens

It is the correct and recommended way to implement message authentication.