Authentication:

Part B:
First, break down the JWT token into the **header, payload,** and **signature.**

I took the "**abcdefghijklmnopqrstuvwxyz0123456789**" as the alphanumeric character set for running the brute-force to find the 5-character secret key because decoded payload contains the hint: "**lowercase-alphanumeric-length-5".**

For every 5-character combination, I signed the decoded payload & header using 'sha256' (algorithm visible in the decoded header) to get the corresponding signature, if a signature matches with the signature in the provided JWT token I broke the loop and printed the secret key.
Found secret key: **p1gzy**

```
charset = "abcdefghijklmnopqrstuvwxyz0123456789"

for combo in itertools.product(charset, repeat=5):
   secret = ''.join(combo)
   decoded_payload = json.loads(base64.urlsafe_b64decode(payload +
'===').decode('utf-8'))
   decoded_header = json.loads(base64.urlsafe_b64decode(header +
'===').decode('utf-8'))
   calculated_signature = hmac.new(secret.encode(),
f"{header}.{payload}".encode(), 'sha256').digest()
   calculated_signature =
base64.urlsafe_b64encode(calculated_signature).decode('utf-8').rstrip('=')
   if calculated_signature == signature:
       print(secret)
       break
```

Part C:
To prevent widespread damage if a JWT signing secret is compromised:

- **Use asymmetric signing** (RSA/ECDSA) so only the private key can sign JWTs.
- **Implement key rotation** with versioning (kid in JWT header) and periodic updates.
- **Use short-lived access tokens** with secure refresh tokens.
- **Maintain a token revocation list** for compromised tokens.
- **Limit token scope and permissions** to minimize damage.