**Test Plan: Cryptographic Scripts**

1. **cryptographic\_hash.py**

**Test Cases:**

1. **Hash Consistency:**
   * **Scenario:** Verify that the hash remains consistent for the same input.
   * **Test Steps:**
     1. Hash a set of input data.
     2. Hash the same input data again.
   * **Expected Result:** The hash values must be the same.
2. **Order Independence:**
   * **Scenario:** Ensure the hash remains the same regardless of the order of input arguments.
   * **Test Steps:**
     1. Hash input data with arguments in one order.
     2. Hash the same input data with arguments in a different order.
   * **Expected Result:** The hash values must be the same.

2. **hex\_to\_binary.py**

**Test Cases:**

1. **Conversion Accuracy:**
   * **Scenario:** Convert a hexadecimal number to binary and back to decimal.
   * **Test Steps:**
     1. Convert a decimal number to hexadecimal.
     2. Use **hex\_to\_binary** to convert the hexadecimal to binary.
     3. Convert the binary number back to decimal.
   * **Expected Result:** The final decimal number should match the original number.

3. **cryptographic\_hash\_example.py**

**Test Cases:**

1. **Hash Generation:**
   * **Scenario:** Generate a hash using the provided examples.
   * **Test Steps:**
     1. Use the **cryptographic\_hash** function with Example 1.
     2. Use the function with Example 2.
   * **Expected Result:** Confirm that the generated hashes match the expected results.

4. **hex\_to\_binary\_example.py**

**Test Cases:**

1. **Conversion Accuracy:**
   * **Scenario:** Verify the accuracy of hexadecimal to binary conversion.
   * **Test Steps:**
     1. Convert a decimal number to hexadecimal.
     2. Use **hex\_to\_binary** to convert the hexadecimal to binary.
   * **Expected Result:** The converted binary should be accurate.

5. **crypto\_dependencies.py**

**Test Cases:**

1. **Version Compatibility:**
   * **Scenario:** Ensure that the specified versions of dependencies are installed.
   * **Test Steps:**
     1. Verify that pytest version matches the specified version.
     2. Verify flask version.
     3. Verify pubnub version.
     4. Verify requests version.
     5. Verify cryptography version.
     6. Verify Werkzeug version.
   * **Expected Result:** All dependencies should be installed with the specified versions.

6. **pubsub\_example.py**

**Test Cases:**

1. **PubNub Initialization:**
   * **Scenario:** Verify the correct initialization of PubNub.
   * **Test Steps:**
     1. Initialize a PubSub instance.
     2. Verify that PubNub is configured with the correct keys.
   * **Expected Result:** PubNub should be properly configured.
2. **Message Publication:**
   * **Scenario:** Ensure messages can be successfully published.
   * **Test Steps:**
     1. Publish a test message to a channel.
     2. Confirm that the message is received.
   * **Expected Result:** The message should be received successfully.

Notes:

* Ensure that the tests cover both normal use cases and edge cases.
* Make sure to run the tests in different environments to validate compatibility.

**Additional Considerations:**

* **Test Automation:** Consider automating tests using testing frameworks such as pytest.
* **Security Testing:** Conduct security testing to identify vulnerabilities.
* **Performance Testing:** Evaluate the performance of cryptographic functions, especially in scenarios with large data.

This test plan provides a structured approach to testing the cryptographic scripts, covering various scenarios and ensuring the correctness of the implemented functionality.