**Stegogram Encryption and Decryption**

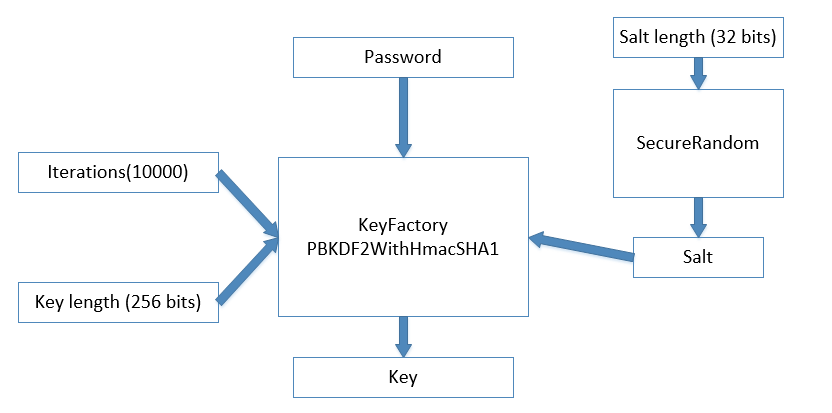
**Standards and Specifications:** Stegogram conforms to the following standards and specifications using the java.crypto library. This information can be found in the java 8 documentation at the following URL.

<https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html>

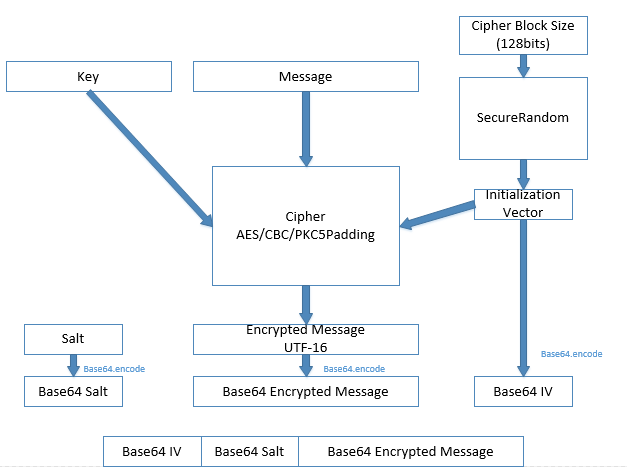
* **FIPS PUB 197** – AES-256 bit block cipher
* **FIPS PUB 81 –** CBC (cipher algorithm mode)
* **RSA Laboratories, "PKCS #5: Password-Based Cryptography Standard," version 2.0, September 2000 -** PBEWithHmacSHA1 (Password based encryption specification including PKCS5 padding)
* **FIPS PUB 184 -**  SHA1

**Encryption:** Stegogram uses a password based encryption scheme defined in the above standards and specifications.

**Key Generation:**



**Cipher Text:**

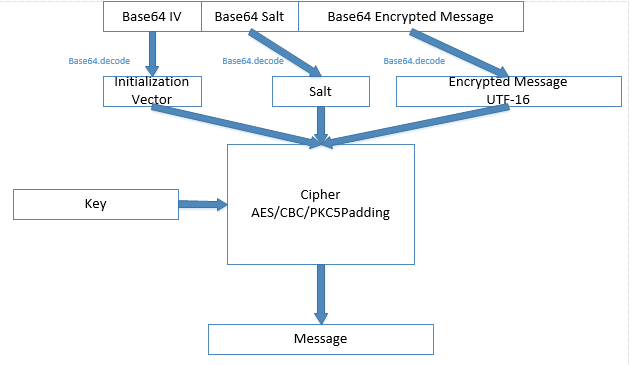


**Summary:**

Stegogram uses a password based encryption scheme called Public-Key Cryptography Standard #5. To generate a key with this scheme you must provide the password, the iterations for the algorithm, the desired key length and the salt value. Stegogram uses a hashed key with UTF-16 encoding. We decided to use UTF-16 encoding because of the increased number of characters that would need to be tried in a brute force attack. Specifically an attacker would have to try 2 to the power of 16 raised to the power of 32 or . The brute force attack is slowed down further by the number of iterations needed to produce the key. The number of iterations is set to 10000 in Stegogram. The iterations are set to 10,000 because this is equal to the number of iterations android 4.0 phones use to encrypt back-up data. The PBKCS#5 only specifies 1000 iterations but we decided to use 10 thousand since the speed of computers have drastically increased since 2001, when the specification was first published. The key length we provide to the cipher generator is 256 bits because this is the length needed for AES-256 encryption. The salt is created by a pseudo-random byte generator. A salt is used to protect against rainbow table attacks and it is currently set to 32 bits. Once the key is generated it is used along with the message and a pseudo-random 128bit initialization vector to generate the encrypted message using the Advanced Encryption Standard in Cipher Block Chaining Mode and PKCS#5 padding. The initialization vector, the salt, and the encrypted message are encoded in Base64 and then concatenated this is then embedded in an image. The salt is safe to send because it is only used to protect against precomputed rainbow tables. If an attacker extracted the salt value correctly they would have to compute another rainbow table based on that salt value. Also each hash would have to be computed using 10000 iterations. Depending on how many words are in there dictionary the computation time to generate these hashes would be infeasible. It is also unnecessary to keep the initialization vector secret due to the protections already in place that have been described above.

**Decryption:** Stegogram uses a decryption scheme defined in the standards and specifications above. The key is generated from a user entered password and then decrypted as shown by the following figure.

**Extracting Message:**



**Summary:**

The decryption method is similar to the encryption method. A key is generated as explained in the encryption summary using the supplied password. The initialization vector, salt and encrypted message is then extracted from the Stegogram and decoded from Base64. The key is then used with the extracted salt and initialization vector to decrypt the message. If the decryption fails because of an incorrect password the user is notified.

**Conclusion:**

Stegogram uses a password based encryption scheme where neither the password nor the hashed password is ever stored or transmitted. This protects against one of the main vulnerabilities in AES encryption, which is poor key management (NIST Special Publication 800-57). Stegogram’s encryption/decryption method is currently used to protect confidentiality of the user. In future releases we hope to also protect the integrity of the data being transmitted. Or in other words we hope to be able to identify if the message has been altered in any way during transmission.