

Report on AESCrypt file Encryption Software JavaCard based security over Secure Channel

Balaji Kommuru UCO 459208
Surendra Sharma UCO 459205
Ashwin Yakkundi UCO 459202

May 4, 2017

1 Introduction

1.1 AESCrypt

AES Crypt is a file encryption software available on several operating systems that uses the industry standard Advanced Encryption Standard (AES) to easily and securely encrypt files.

On the command line, one can execute the "aescrypt" command with name of the file



and password to use to encrypt or decrypt. For Java and C# developers, there is also a Java and C# library available that can read and write AES-encrypted files from within your application.

Using a powerful 256-bit encryption algorithm, AES Crypt can safely secure your most sensitive files. Once a file is encrypted, you do not have to worry about a person reading your sensitive information, as an encrypted file is completely useless without the password. It simply cannot be read.

AES Crypt is the perfect tool for anyone who carries sensitive information with them while traveling, uploads sensitive files to servers on the Internet, or wishes to protect sensitive information from being stolen from the home or office. AES Crypt is also the perfect solution for those who wish to backup information and store that data at a bank, in a cloud-based storage service, and any place where sensitive files might be accessible by someone else.

1.2 Security Issues of AESCrypt

The main problem of the AESCrypt is it takes the password from the user in plaintext format and processes as it is. This is not advisable as it can leak the information without the knowledge of the user. This is shown in the figure: *AESCrypt-Existing*.

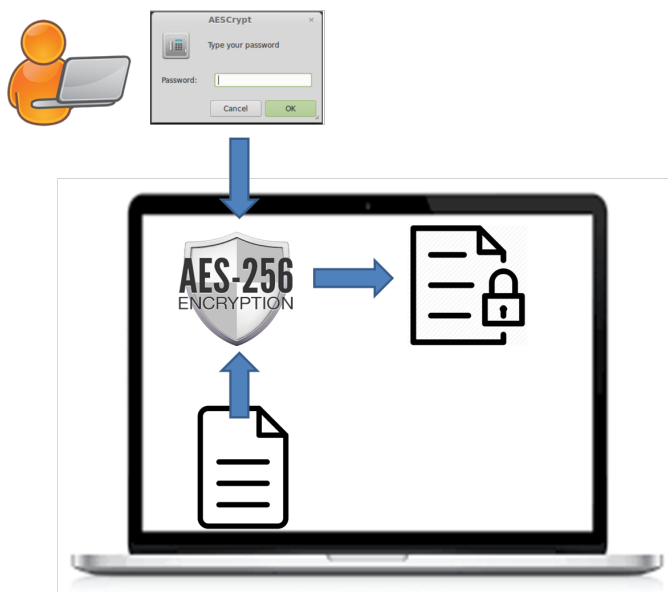


Figure 1: AESCrypt-Existing

The AESCrypt prompts for a password from the user. Alternatively, the application reads a password file stored on the computer. The file is encrypted with AES-256-CBC and the encrypted file with .aes extension is stored back on the computer. The password stored in the memory of the user is liable for loss while the password stored in the password file is vulnerable.

Although the AES-256 encryption is used, the confidentiality of the file relies only on the strength of the Password chosen by the user, which may be with much less entropy. The application is vulnerable to brute force attacks against smart dictionary attacks considering also common usage of digits or special characters in human-created passwords. The password stored in the memory of the user is liable for loss while the password stored in the

password file is vulnerable.

1.3 Security Enhancement using JavaCard HSM

JavaCard based HSM(Hardware Security Module) solution is implemented to improve the security of the AESCrypt software. The design of this solution ensures the security in storages, security in transit, Authentication and Integrity of the application. The password handling is done through the Java Card as shown in the figure *AESCrypt-modified*.

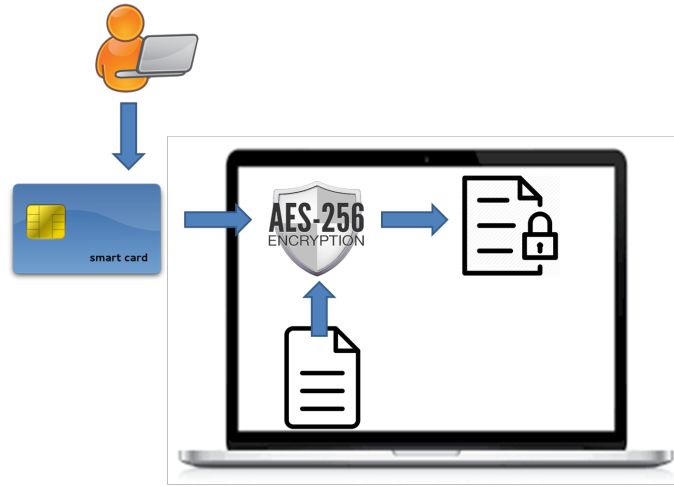


Figure 2: AESCrypt-Modified

The details of the design and implementation is explained in the following sections.

2 Setting the Keys in Trusted Environment

2.1 Setting of PIN

2.1.1 PIN setting with APDU

PIN of 4 digit size is set using the APDU. This is done in trusted environment. This is done before the card is given to particular user. Generally it is done by administrators or manufacturers of card.

2.1.2 PIN Verification with APDU

The PIN already set is verified for the correct using this process. We are checking the PIN set earlier by us as a verification.

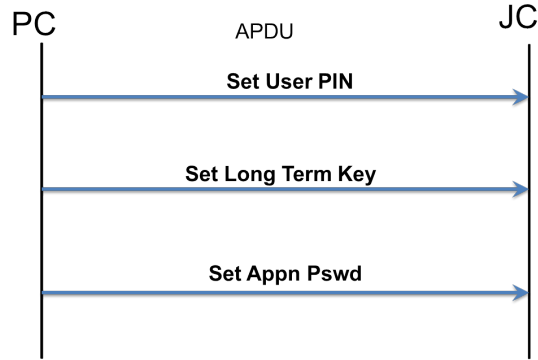


Figure: Setting Long Term Key & App Password in JavaCard

3 How many ports have been used by attacker to launch the attack?

3.1 Setting of the Long Term Key

Long Term Key of 16 bytes size is set in the trusted environment. This Key is statically defined in the code. It is once set is permanent for the card life time. It is read only Key. This is done using the **keytype.setKey** function. Like wise reading the key is done using the **keytype.getKey** function. This process is depicted in the figure *Setting the Long term key and app password in javacard*.

4 Session Key establishment using the Secure Channel

Using the Long Term Key, Session Key establishment is done securely. Initially Nonces are generated at both PC(Personel Computer) and JC(JavaCard). Both components has unique IDs as PCID and JCID. They are used for mutual authentication.

PC generates the nonce, encrypts including with PCID with IV_1 using the long term Key and sends it to JC. At other end JC decrypts with long term key, verifies the PCID and extracts the PC nonce.

JC generates the nonce, encrypts including with PCID and PC nonce with IV_2 using the long term Key and sends it to PC. At other end PC decrypts with long term key, verifies the PCID and PC nonce recieved and extracts the JC nonce.

Now both ends got the nonces of both and generates session keys by exclusive-or of them. And now PC sends the generated session Key with long term key with IV_3 and sends to JC. JC decrypts and verify the sessions key recieved with its generated one. If they tally,

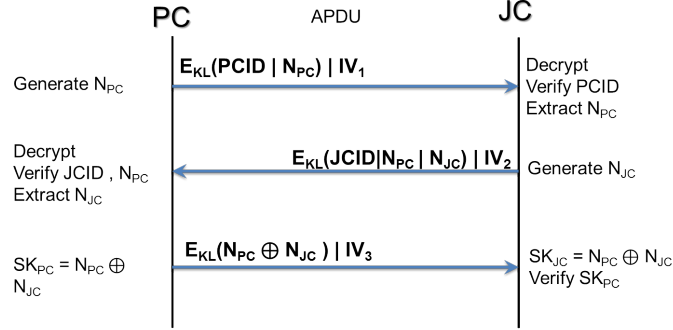


Figure: Secure Session Key Establishment

communication can be done securely with using this session keys by both ends. This process is depicted in the figure *SessionKeyEstablishment*

5 Password retrieval through the secure channel

The generated session Keys are used to exchange the password securely. For getting the password, PIN verification is done. PIN and Password are encrypted through the secure channel always. Integrity of the Channel protected by SHA2 based HMAC.

For getting the password, PC initially encrypts PIN with session key, and calculates HMAC with IV_1 and sends along with APDU. JC decrypts and verifies with HMACed value and PIN. Then JC encrypts the Password with session key, and calculates HMAC with IV_2 and sends along with APDU. PC decrypts and verifies with HMACed value and extracts the password.

6 Conclusion

The design of components Trusted environment setup for security in storage, Secure channel for security in transit, mutual authentication and integrity checking with HMAC all together enhances the security of the AESCrypt.

7 References

1. <https://github.com/FreedomBen/aescrypt/tree/master/java>
2. <https://docs.oracle.com/javacard/3.0.5/api/javacardx/crypto/package-frame.html>
3. <https://docs.oracle.com/javacard/3.0.5/api/javacardx/crypto/Cipher.html>
4. <https://docs.oracle.com/javacard/3.0.5/api/javacard/security/package-summary.html>

5. https://en.wikipedia.org/wiki/Secure_channel
6. https://en.wikipedia.org/wiki/Hash-based_message_authentication_code
7. https://en.wikipedia.org/wiki/Key_exchange
- 9. <https://github.com/maxashwin/pv204/tree/Mod-ss> - url of Code of Modified AESCrypt**