

Secure Channel for File Encryption using User Keys stored on JAVA CARD

Milan Patnaik¹, Suresh Baddipudi¹, Mayank Samadhiya¹, Mohammad Akhtar¹

¹*Department of Informatics, Masaryk University, Brno, Czech Republic*

Abstract

Java Cards can be used to store user keys and passwords for use in encryption and decryption of the files on the PC. An application can access the Java Card to extract the key and perform the encryption or decryption of files. However, it is imperative to establish a secure channel between the Java Card and the PC application. This proposal describes two such secure channel protocols viz, *SymSec* and *AsymSec* to access the key from the Java Card for file encryption and decryption on a PC. The *SymSec* and *AsymSec* protocols use symmetric and asymmetric key cryptography for the secure channel protocol. It also proposes to compare the two protocols with respect to time for encryption and decryption of files.

Keywords: Symmetric Key Cryptography, Asymmetric Key Cryptography, Java Card Security, Secure Channels

I. INTRODUCTION

Java cards are tamperproof smart cards which can be used to store users secret passwords and even certificates and private keys. One can deploy java cards to provide stronger user authentication and nonrepudiation for a range of security solutions, including key transfer over a network, secure web communication etc. However, all such applications require establishing a secure channel between the application and the Java Card for secure transfer of the key/password from Java Card to the PC application. This work will use a Java Card to store user secret key/password to be used for file encryption and decryption on a PC. The work will propose two secure channel protocols and compare their timings for file encryption and decryption. Some of the major highlights of this work will be as under:-

- Propose a Symmetric Key Cryptography based secure channel protocol between Java Card and PC application.
- Propose an Asymmetric Key Cryptography based secure channel protocol between Java Card and PC application.
- Compare the timings for file encryption and decryption using the above two secure channel protocols.

Before describing the protocols, it is to be noted that the encryption of an *OBJECT* using a *KEY* is denoted as $Enc[KEY, (OBJECT)]$ in rest of the proposal.

II. PROPOSED SECURE CHANNEL PROTOCOLS

A. *SymSec Protocol*

This protocol will use the symmetric key cryptography for establishment of the secure channel using secure Deffi Hellman algorithm. It is assumed that an User PIN (*PIN*), Pre-Shared Secret (K_p), Deffi Hellman parameters ie, prime p and the primitive root modulo p ie, g and the Counter value (C) has been set on the Java Card during the administrative phase. A session key K_s is generated by Deffi Hellman algorithm during each session. The secure channel protocol (*SymSec*) for retrieval of User Key (P) is as given in Figure 1.

During the implementation of the *SymSec* protocol, the following inferences were made :-

- The generation of Deffi Hellman keys have to be done using RSA encryption APIs which incur equal delay.
- The use of RSA encryption for finding exponential modulus did not succeed on the JAVA CARD.

In view of the above, the *SysSec* protocol was modified by changing the Deffi Hellman Key generation into a Key Derivation protocol using Cryptographic Hash function. The same protocol is described next.

B. *SysSec.mod Protocol*

This protocol used symmetric key cryptography. It is assumed that an User PIN (*PIN*), Pre-Shared Secret (K_p), and the Counter value (C) has been set on the Java Card during the administrative phase. Two keys viz, K_{enc_B} and $K_{enc_{BA}}$ using two random numbers A and B generated by the PC and the JAVA CARD were generated using Hash functions as under:-

- $K_{enc_B} = MAC(B)$
- $K_{enc_{BA}} = MAC(BA)$

where, $MAC(X)$ is the MAC of X and BA are concatenation of B and A in the order given. The secure channel protocol (*SymSec.mod*) for retrieval of User Key (P) is as given in Figure 2.

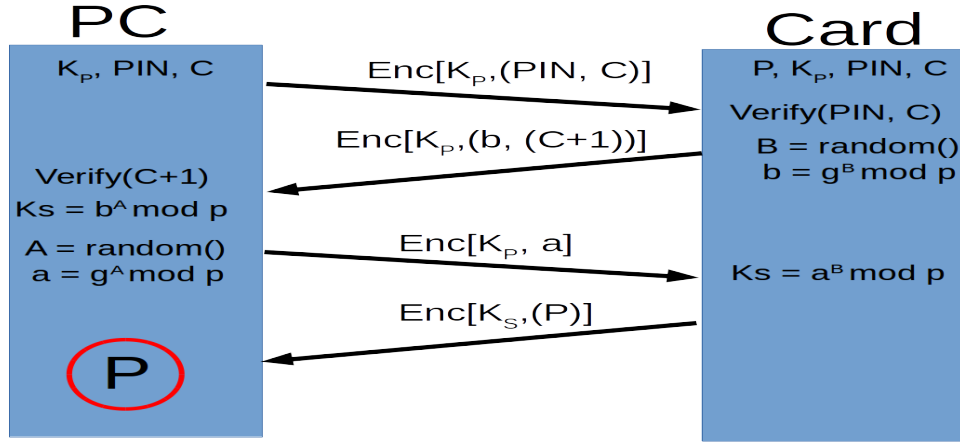


Fig. 1. Secure Channel Protocol : SymSec

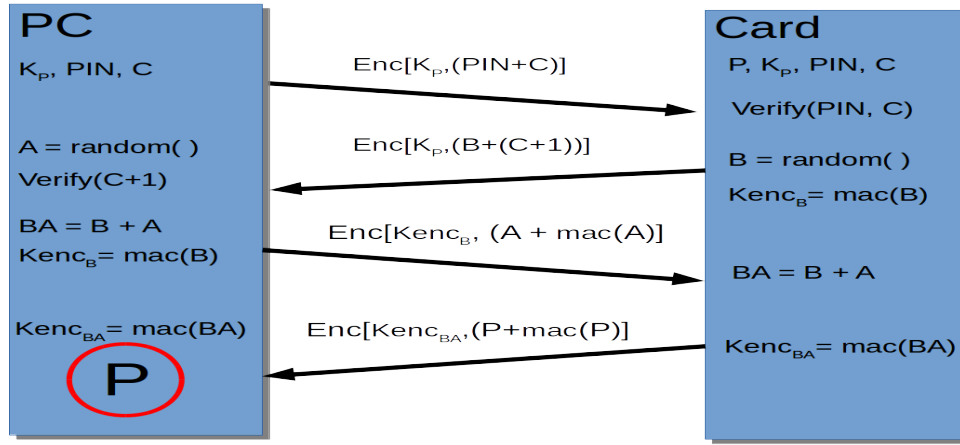


Fig. 2. Secure Channel Protocol : SymSec_mod

C. AsymSec Protocol

This protocol uses asymmetric key cryptography for establishment of a secure channel between the Java Card and the PC application. The PC application proposed to be used is as given in [1]. This secure channel protocol will be used for transmission of User Key (P) from the Java Card to the PC application [1] for encryption and decryption of files. It is assumed that an User PIN (PIN), Pre-Shared Secret (K_p), and the Counter value (C) has been set on the Java Card during the administrative phase. The one time public/private key pair of Card ($K_{B_{pub}}$ and $K_{B_{pvt}}$) and PC application ($K_{A_{pub}}$ and $K_{A_{pvt}}$) are generated for each session. The secure channel protocol (*SymSec*) for retrieval of User key (P) is as given in Figure 3.

The various observations and inferences learned during the implementation of RSA are as under:-

- The RSA Public Key can be exported from PC to Card and vice versa by extracting the Modulus and Exponent using `<public key>.getModulus` and `<public key>.getExponent` functions.
- The `<public key>.getModulus` function returns one extra byte as the sign byte of the Modulus. For example in case of 1024 bit RSA it returns 129 bytes.
- The exporting of Modulus and Exponent between Java Card and PC Application was successfully implemented.
- The `<public key>.setModulus` required setting the 129 byte Modulus for successful decryption by the Private Key.
- The proof of concept implementation of the above was carried out successfully in PC application.
- The `<public key>.setModulus` of 129 bytes in Java Card failed.

III. EXPERIMENTS AND RESULTS

Experiment 1 : A secure protocol for transfer of Key from the Java Card to the PC application will be designed and implemented by using Symmetric Key Cryptography. The same will involve the following initial *One – time* tasks viz, 1) Setting of User PIN (PIN); 2) Setting of Counter (C) and the User Key (P); and, 3) Setting of Pre Shared Secret (K_p). During the process of encryption and decryption, a secure channel will be established and the User Key (P) will be retrieved by the *SymSec_mod* protocol described earlier. The timings for User Key (P) retrieval and encryption of test files were noted

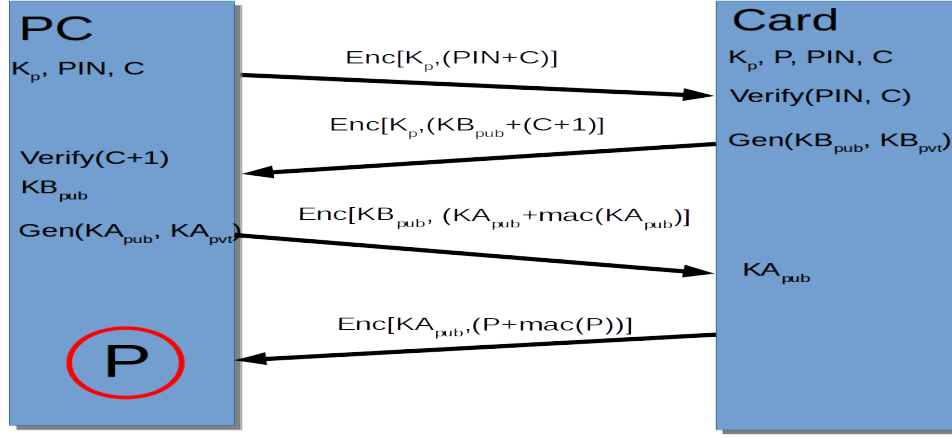


Fig. 3. Secure Channel Protocol : AsymSec

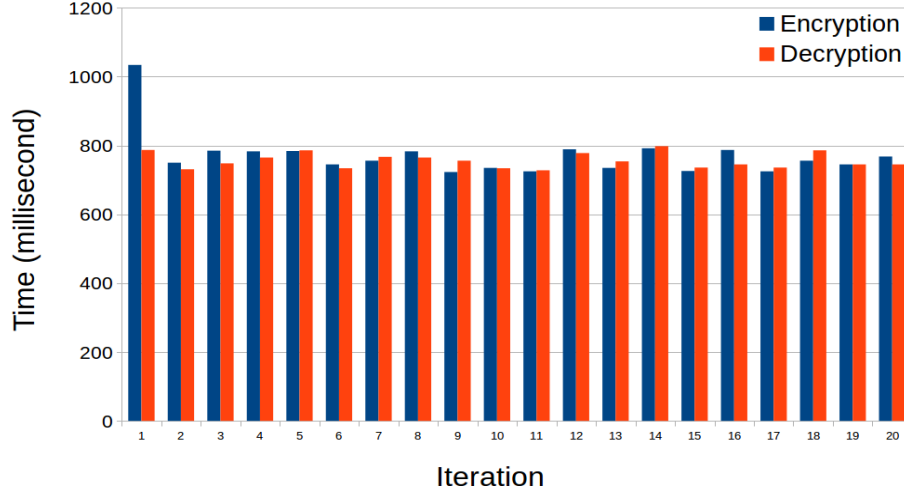


Fig. 4. Time for File Encryption and Decryption : SymSec_mod

for a set of 100 test files. Figure 4 gives the time for file encryption and decryption including retrieval of password from card. Figure 5 gives the time for retrieval of password from the card.

Experiment 2 : A secure protocol for transfer of Key from the Java Card to the PC application [1] will be designed and implemented by using Asymmetric Key Cryptography. The same will involve the following initial *One-time* tasks viz, 1) Setting of User PIN (P); 2) Setting of Counter (C) and the User Key (P); 3) Setting of Pre Shared Secret (K_p); and, 4) Setting of Diffie Hellman parameters p and g . During the process of encryption and decryption, a secure channel will be established and the User Key (P) will be retrieved by the *AsymSec* protocol described earlier. The timings for User Key (P) retrieval and encryption of a test file will be noted for a set of 100 test files.

IV. CONCLUSION

Java Cards are extensively used for storing user keys/passwords. These keys can be used for file encryption and decryption on a PC application [1]. However, it is imperative to establish a secure channel between the Java Card and the PC application for retrieval of secret key from the Java Card. This work will propose two such secure channel protocol with symmetric and asymmetric key cryptography. It will also compare the proposed protocols for timings of retrieval of key and file encryption and decryption.

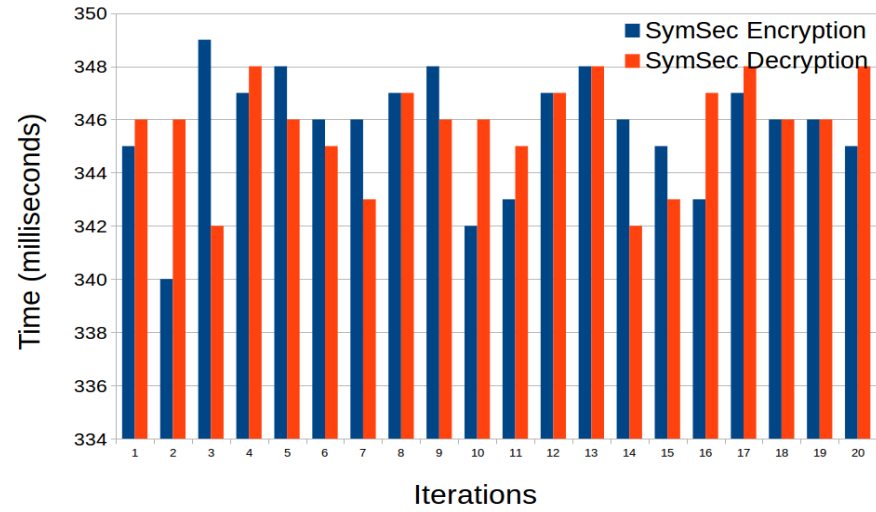


Fig. 5. Time for Password Retrieval : SymSec_mod

REFERENCES

- [1] https://sourceforge.net/projects/fileencoderapplication/files/20160130_FileEncoderApplication.v1.1.zip/download