EPiC Series in Computing

Volume 63, 2019, Pages 197–210

Proceedings of 32nd International Conference on Computer Applications in Industry and Engineering

SAFEST: Secure Actions for FTP Environment with Smart Token

Tarek S. Sobh[1] and Awad H. Khalil[2]

[1]The Higher Institute of Computer and Information Technology, El Shorouk Academy, Cairo, Egypt
[2]Department of Computer Science & Engineering, The American University in Cairo, Egypt
tarekbox2000@yahoo.com, akhalil@aucegypt.edu

Abstract

Nowadays, with the wide applications of distributed systems, web-based applications, and communications systems over the Internet for carrying data between users such as terminal client and computer/server or communications between different devices using a computer network, network security has become crucial requirement to ensure authentic received data during transmission. Authentication and encryption are basic procedures to ensure secure communications over a public network due to tamper-resistance and convenience in dealing with a password file. Most of the used protocols; HTTP, FTP, and SMTP of the Internet applications use text stream that is more and more vulnerable to attacks. Encryption represents the main security for the most computer applications.

This work proposes enhanced secure actions for transferring data using FTP protocol by using a smart token. A smart token has the capabilities of the smart card, but more secured beside some interesting operations. A practical and secure user scheme, based on a smart token device, is proposed. A Secure Platform has been developed using implemented APIs and PKCS#11 as RSA standard interface. The proposed API is called SAFEST (Secure Actions for FTP Environment with Smart Token). SAFEST API wraps a standard protocol for implementing the communication between a token and the application using it. This API is a platform independent, scalable to support more functionality, optimizing token usage and adding more security for accessing token objects. The smart token can process the cryptographic key operations on its own rather than on the host computer, which supports high-level platform independence. In addition, through the proposed SAFEST API, standard interfacing to such token devices from any vendor can be implemented through using PKCS#11 interfaces, developed by RSA labs.

Key words: **Smart Token, Security Protocols, Web Applications, PKCS#11, FTP**

Q. Yuan, Y. Shi, L. Miller, G. Lee, G. Hu and T. Goto (eds.), CAINE 2019 (EPiC Series in Computing, vol. 63), pp. 197–210

1 Introduction

USB keys (tokens) and smart cards are commonly used for distributed environments that are deployed in insecure systems.

As the user information and keys are stored on the smart card but processed on the host computer for intended applications, consequently, this leads to high risk of identity theft. This problem has been solved by smart token [3] [4] [12] through developing API allowing applications to use a smart token that can process the key operations on its own rather than on the host computer.

A smart token should contain an API for the purpose of working with the cryptographic keys and allows key management operations.

Such API is important and it should be designed to prevent malicious operations. In addition, the stored ciphering keys stored should remain secret. Consequently, it is hard to design such an API and to implement key recovery processes to face 'security APIs' attacks [1] [5] [17] [18].

RSA PKCS#11 is a common standard that is used for designing token interfaces [12] [16]. The API described by RSA PKCS#11 is known as 'Cryptoki'.

The FTP protocol is available for every client without the need for any additional requirements. Here, our aim is to propose secure actions for transferring data using FTP and leveraging a smart token. For the purpose of data transfer, we use a standard FTP protocol that supports no security measures instead of a protocol that does, like SCP that uses SSH. In this work, a smart secure FTP application can be supported via the API where a user can upload and download encrypted files securely.

The proposed system supports the features of securing login authentication on the server using smart token, securing encrypted uploading and downloading and storing files on the server. In addition, a new feature is introduced: securing the information that can be accessed by a third party.

The proposed system is called Secure Actions for FTP Environment with Smart Token (SAFEST). SAFEST is smart and secure FTP application and it is a client/server architecture. It can provide users with a secure transfer of data and strong authentication protocol [4] [7] [20] [21]. Secure transfer meaning that the transferred data are encrypted, come from an authorized person signed with digital signature, and stored encrypted on the server. The proposed API can be accessed from any computer through a web application or installed as a desktop application. In addition, the proposed system is a platform independent.

Standard and customized protocols such as PKCS#11 developed by RSA [5] [9] [12] [16] [18] are used to keep the application more secure. We added some enhancements to the building processes of the security system that led to systems which can be handled efficiently. The results of testing the SAFEST using available tokens have shown that every token that is offering the necessary functionalities to import and export encryption keys, and the required actions for standard key management, has been working in a secure way allowing the FTP application to call to the APIs, user's keys with less vulnerability to theft or misuse by unauthorized persons. The Smart Token manages access to the keys by passwords, hence, identity security requirements are met [8] [10] [14].

2 THREAT MODEL

The threat model of security smart token focuses on Smart Token devices that are used for authorization and authentication.
For designing a token-based authorization and authentication scheme, we have to analyze all possible threats. A smart token is mainly used to:

- Accomplish strong client authentication when token holder uses services such as FTP or others through a web application. It is accomplished by having the client sign in data. The token only produces the digital signature after some form of token holder's verification through using the Personal Identification Number (PIN).
- Generating digital signatures with a certificate from a certificate authority (CA) instead of handwritten signatures. This advanced digital signature relies on the token holder verification using PIN validation. Both generation and verification of the digital signature are complicated tasks.
- Getting information on the smart token-holder (e.g. Id, gender, date of birth, address, … etc). It is common to gather such information without any smart token-holder verification.
- Decipher confidential data that is intended for the smart token-holder only.
- Smart token integrity: the main target after creating tokens is to prevent unauthorized entities from forging or changing the content of a valid token.
- Token theft: It is mandatory to have a theft detection mechanism to prevent unauthorized entities from using stolen smart tokens.

The threats against entities in the framework are as follows:

- Users/clients may be heterogeneous. They may be smart devices, laptops, smartphones, cloud services or other such as services of Internet of Things (IoT). The stored data related to smart token content on these devices should be protected from leakage. Authentication is required for all clients during the smart token exchanges.
- Successful authentication is required to authorization server in order to issue smart tokens for the clients.

The smart token may contain user data, private keys to sign or decrypt information, and a genuine copy of the root certificate as reference data. The smart token is connected to the user's device. The smart token communicates with a web application on the user's computing device, and by using PIN; the user authenticates himself to his smart token.

3 Smart Token & PKCS#11 Attacks

Smart Token is a flash-like device a user can easily insert into the USB port and access his account on the SAFEST application only when it is present.

3.1 Why Smart Token

Standards for interfacing to Smart Token devices are introduced at some level. For instance, the mechanical characteristics and electrical connections are well defined, as are the methods for supplying commands and receiving results [6] [11] [12] [18].

A smart token can create, delete, modify and search for objects. It can control objects in order to do cryptographic functions because smart token sometimes has an interior irregular number generator.

3.2 PCKS#11 Attacks

The originators of PKCS #11 portrayed the design objectives as follows: to "give a standard interface amongst applications and cryptographic token/cards" and in the meantime to "permit asset sharing" (i.e. many-to-many connection amongst applications and devices). It was not expected to be a general interface to cryptographic activities or security administrations. Instead, it could be used to build such activities, services or appropriate APIs.

Various late papers have demonstrated attacks against sensitive keys [1] [5] [14] [17] [18]. Huge numbers of attacks are 'key separation', where the key attributes are set so as to give a key conflicting role. Other related works [1] [5] explain the case of a key with the attributes set for the deciphering of decrypt texts, and for 'wrapping', i.e. encryption of different keys for secure transport. According to Matteo et. al. (2010), in order to decide the estimated value of a sensitive key, the attacker basically wraps it and then decrypts, as follows:

Initially: the attacker knows h (n1, k1) and

h (n2, k2). The name n2 has the attributes wrap and

decrypt set whereas n1 has the attribute sensitive and

extractable set.

Trace:

Wrap: h (n2, k2), h (n1, k1) → {|k1|} k2

SDecrypt: h (n2, k2), {|k1|} k2 → k1

Matteo et. al. (2010) present notation for PKCS#11 based APIs, characterizing it more formally as follows: h (n1, k1) is a predicate expressing that there is a handle n1 for a key k1 put away on the device. The same key for both sender and receiver are used for symmetric encryption (i.e. in this case k1 under key k2 is defined by {|k1|} k2). Moreover, as shown by the PKCS#11 formats, the smart token cannot state whether an underlying string is a cryptographic key or not. Hence when it starts the decipher command, it has no chance to get off saying that the packet it is deciphering contains a key.

It is hard to prevent the types of attacks listed above, but we can do some actions as follows: Limit the commands that might be used to prevent certain conflicting attributes from being identified for the same object, but at the same time show more attacks [1] [5]. However, we are not sure that any standard real devices that follow the standard actually implement key management as such, because a large portion of the functionality is optional. This is one of the reasons for this work. Furthermore, we need to check the PKCS#11 standard with appropriate configuration still safe without the need for some new mechanisms. This is another motivation for this work.

3.3 SAFEST Planning

Based on existing products, multiple applications would be allowed to use a smart token for security not as before when each application needs to deal directly with one token. Hence, the SAFEST facilitates this process through the designed API.

Also, as the market for smart token applications is evolving and fulfills major requirements in the industry, many have shown interest in the product when completed.

The smart token API and applications are intended to market segments concerned with securing keys to secure data transfer and encryption of data for the following purposes:

- Cost-effective one-time password for the use of multiple employees,
- Strong authentication for private key implementation for each individual user token, or,
- Single Sign-On (SSO) [12] [13] for complete password management with more complex passwords to accomplish day-to-day processes [2] [15] [16] [19].

Figure 1 shows SAFEST ordered layers which include smart token API and applications, Java-compatible platform, SAFEST API, and existing FTP software. All these layers are working together in order to serve and deal with the target application.

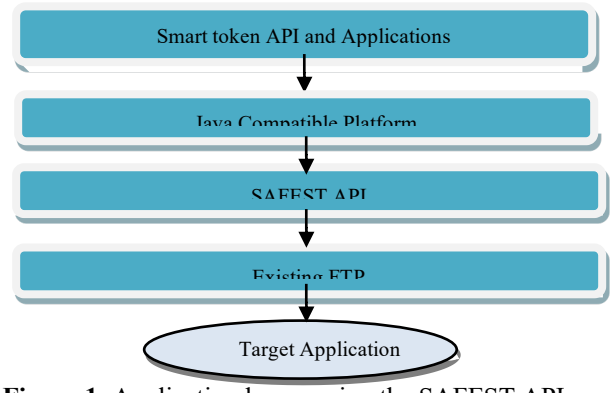

Figure 1: Application layers using the SAFEST API

Each user in our application framework can hold his/her secure keys and data almost immune to misuse as most applications use a smart token to secure user data [17] [22].

The SAFEST mission statement is developing smart token API. Each developer can work directly without the need to deal directly with token and applications that demonstrate that API.

4 Solution overview and SAFEST architecture

After recognizing the importance of the token and its limitation being just a microcontroller with small memory storage, an identity-based application has been conceived. In real-time, these capabilities are not sufficient to encrypt a file and it could take hours to just encrypt 1 MB of data with a symmetric algorithm. This was the strong motivation to develop an API to facilitate the interfacing. This API is called SAFEST API.

For performing its security goals, SAFEST focuses on the following sensitive targets:
1. PIN code enabling cryptographic operations with the smart token device;
2. Cryptographic steps are independent of the knowledge of the PIN code;
3. Prevent cryptographic keys leakage in its clear shape out of the device.

4.1 SAFEST API Architecture

SAFEST API wraps some standard protocols for implementing the communication between a token and the application which uses it. The API is platform independent, and scalable to hold more functionalities. In addition, it optimizes token usage, and adds more security on accessing token objects. As shown in Figure 2, the required components for dealing with SAFEST APIs are Token with device contention and synchronization, operating system, and PKCS#11.

In this work, the secure application is a secure file transfer between a client and FTP server; these security actions will cover the authentication operation, secure data transfer, and confidentiality. The SAFEST API provides the application with a rich set of functions and/or features that facilitate the development of the secured application. Some of such user interface functions and/or features are listed as follows:

Login: the user can access SAFEST web page then inserts his/her token and types his/her PIN code, username, and password to login with his/her account.
View folder contents: the user can click on the folder to view a list of the contents such as folders and files.
Upload file: the user can select the file to be uploaded from the file chooser.
Download file: the user can select the file to be downloaded and its location on his/her terminal.

Invoke permission: the user can select the file to invoke permission, then a list containing the options (read, write, read/write or execute) will appear for each of both owner and group.
Create a new folder: the user can choose the location on the server and create a new folder with its intended name.
Edit file/folder name: the user can choose the file/folder on server and press "Edit" then type its intended name.
Move file/folder location: the user can choose the file/folder on a server and move to the desired location.

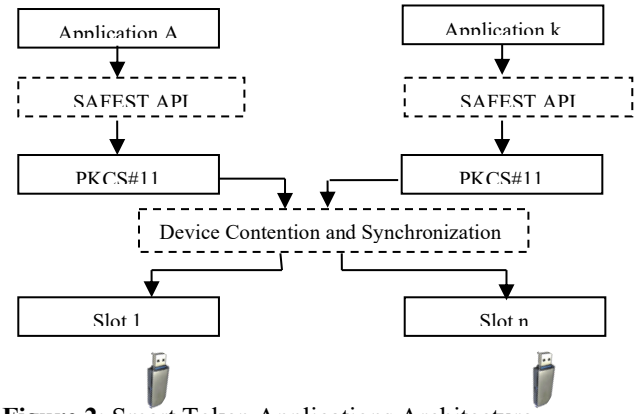

Figure 2: Smart Token Applications Architecture

SAFEST supports encryption of large files and imports and exports sensitive data from the token, encrypts some objects, and generates keys. In addition, it works in a client/server environment for authentication purpose using the smart token as shown in Figure 3, while, Figure 4 introduces a use case diagram of the proposed secure FTP application as a case study.

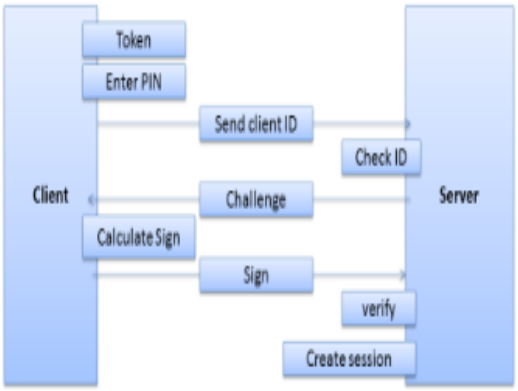

Figure 3: Authentication Calling Sequence application

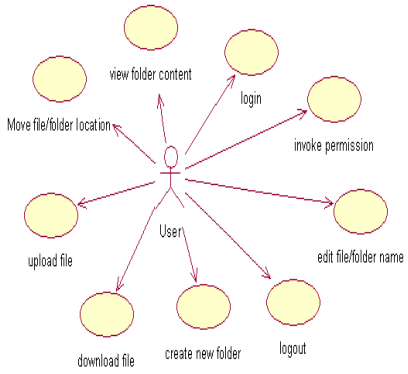

Figure 4: Use case diagram of the case

4.2 Security Transparency to Application User

The development of secured application over SAFEST API allows applications to use smart token once it can process the cryptographic key operations on its own rather than on the host computer or PIN code. Our SAFEST API will provide the conceived application with transparency to use the smart token on any platform. It uses the standard interfacing to such token devices of any vendor. Such interface, developed by RSA labs, is called PKCS#11.

A Smart token is a flash-like device that a user can easily insert to the USB port and accessing his account on our application only when it is present. It operates as smart card except encryption and decryption are embedded inside. Our intended system uses smart token over the web to secure file transfer between users and server using our API to deal with a token.

File transfer application demonstrates the SAFEST API where a user can upload and download files in a secure manner. Such an application can be used in a very friendly way. A user just opens the website and uses the application from the available browser on any platform, and plug in the token into a USB port and transfer files in a secure way.

The cryptographic keys will be less vulnerable to theft or misuse from unauthorized persons since the smart token has its PIN and the stored cryptographic keys are either encrypted or not extractable. In addition, the SAFEST API authenticates the objects created by the smart token for any more additional security purposes.

4.3 SAFEST Requirements and Capabilities

The conceived application uses the smart token as a main cryptographic device which will be the main component.

- Acceptance and Operational Criteria

Our intended system uses smart token over file system to secure file storage using our API to deal with a token with the following requirements

Token: The token is needed to be connected to the server in order the user can connect
Java compatible machine: As Java is platform independent we just need a computer with Java installed.
Nonfunctional requirements: Platform independence - Web and desktop - Multilingual.
Functional requirements: Encrypt a file - Decrypt a file - Export public Key

It is clear from the above requirements SAFEST can provide an application to the user with the following capabilities:

- Scalability and platform independence that is a key feature of our approach due to using Java as a platform independent and the above listed nonfunctional requirements.
- Secure application with strong authentication due to the above listed functional requirements with PIN code usage.

5 Experimental Case Study

In order to test the conceived SAFEST, an experimental tool or application has been developed that heavily uses encryption and decryption of files. This tool provides means for ensuring that the file is not modified by another person, and ensures that the file comes from a trusted person. The tool provides a transparent way to encrypt and decrypt files with easy key distribution. Also, it supports users with transparent and concrete secured data transfer with variant mechanisms.

5.1 Case Study

General users just need an easy and friendly way, free of technical compatibility problems, in order to secure their data and information. The normal user would be allowed to access private objects on the token, and that access is granted only after the normal user has been authenticated using SAFEST API.

Simple scenario:

- User1 wants to encrypt file private.txt
  1. He/she will open file encryption and choose from file menu encrypt, then encrypt dialogue will appear to him
  2. He/she will choose the file to be encrypted and will specify the location of the encrypted file.
  3. Then he/she will choose the public key, which will encrypt the file.
  4. Then he/she will press the encrypt button.
- User1 wants to decrypt file private.txt.enc
  1. He/she will open file encryption and choose from file menu decrypt, then decrypt dialogue will appear to him/her.
  2. He/she will choose the file to be decrypted and will specify the location of the decrypted file.
  3. Then he will choose the public key or the certificate to verify the digital signature of the sender.
  4. Then he/she will press the decrypt button.

A security officer is allowed to access the private objects and editing them on the token, and that access is granted only after the normal user has been authenticated using the SAFEST API.

Table 1: Case study decomposition

Subsystem ID	Subsystem Name	Description
1	SAFEST-API	API to deal with token directly
2	PKCS11	The PKCS library
3	File Encryptor	Perform cipher/decipher files and handle interaction with users

5.2 Case Study Classes, Sequence Diagrams, State Diagram, and User Interface

In this section, a case study is displayed in terms of main and detailed classes, sequence diagrams, state diagrams and the screenshot of the graphical user interface. Tables 2 and 3 show the main classes and detailed classes of the proposed case study associated with a brief description.

Table 2: Main classes of the proposed case study

Class ID	Class Name	Description
1	File Encryptor	Perform encryption/decryption through SAFEST API using key, certificate or online web service
2	SAFESTsingleton	Check when creating a new object if there is an object already created to be used, if not it will create a new one
3	SAFEST API	Functionality to use token utilities within the application

Table 3: Detailed classes of the proposed case study

Class ID	Class Name	Description
1	Token	Interface with token through SAFEST API
2	FTP client	Handle connection initiation and processing that follows
3	FileTransferClient	Control all download upload encrypt and decrypt operations

4	FileTransferInputStream	Download file to user specified location
5	FTPInputStream	Get remote file
6	FTPOutputStream	Transfer file to remote destination
7	FileTransferOutputStream	Upload file
8	WebService	maintain connection
9	SAFESTlogin	Login performed through SAFEST API
10	AbstractCommand	Generic command to allow application expendability
11	ChallengeCommand	Initiate communication channel
12	AuthenticationCommand	Authenticate user
13	SAFEST API	Functionality to use token utilities within the application

Figures 5, 6, and 7 present sequence diagrams of some basic actions such as Authentication Command Execution, Encrypt File, and Command Execution.

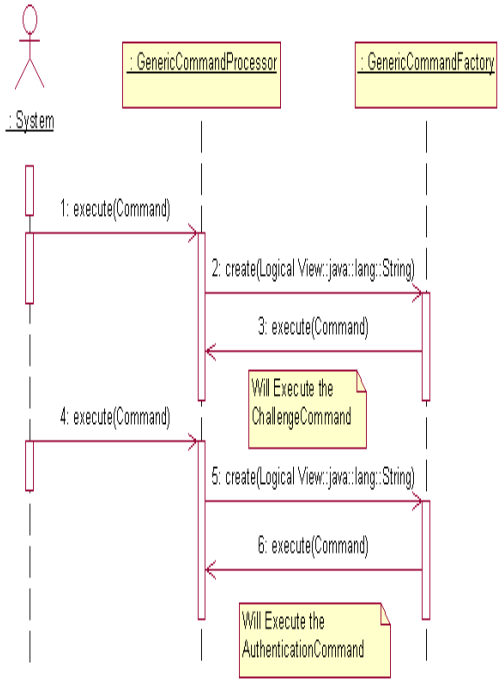

Figure 5: Sequence Diagram of Authentication Command Execution

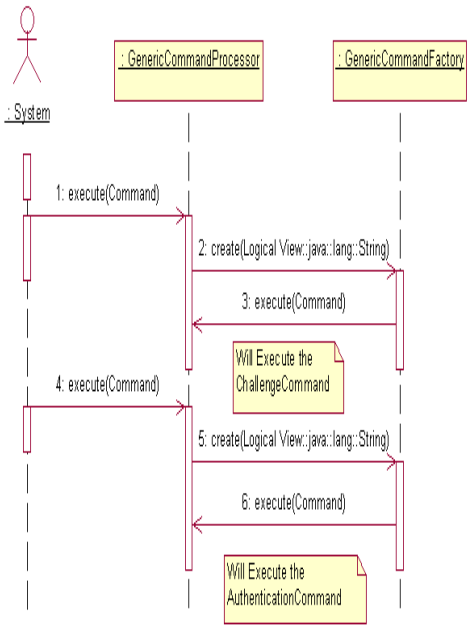

Figure 6: Encrypt File Sequence Diagram

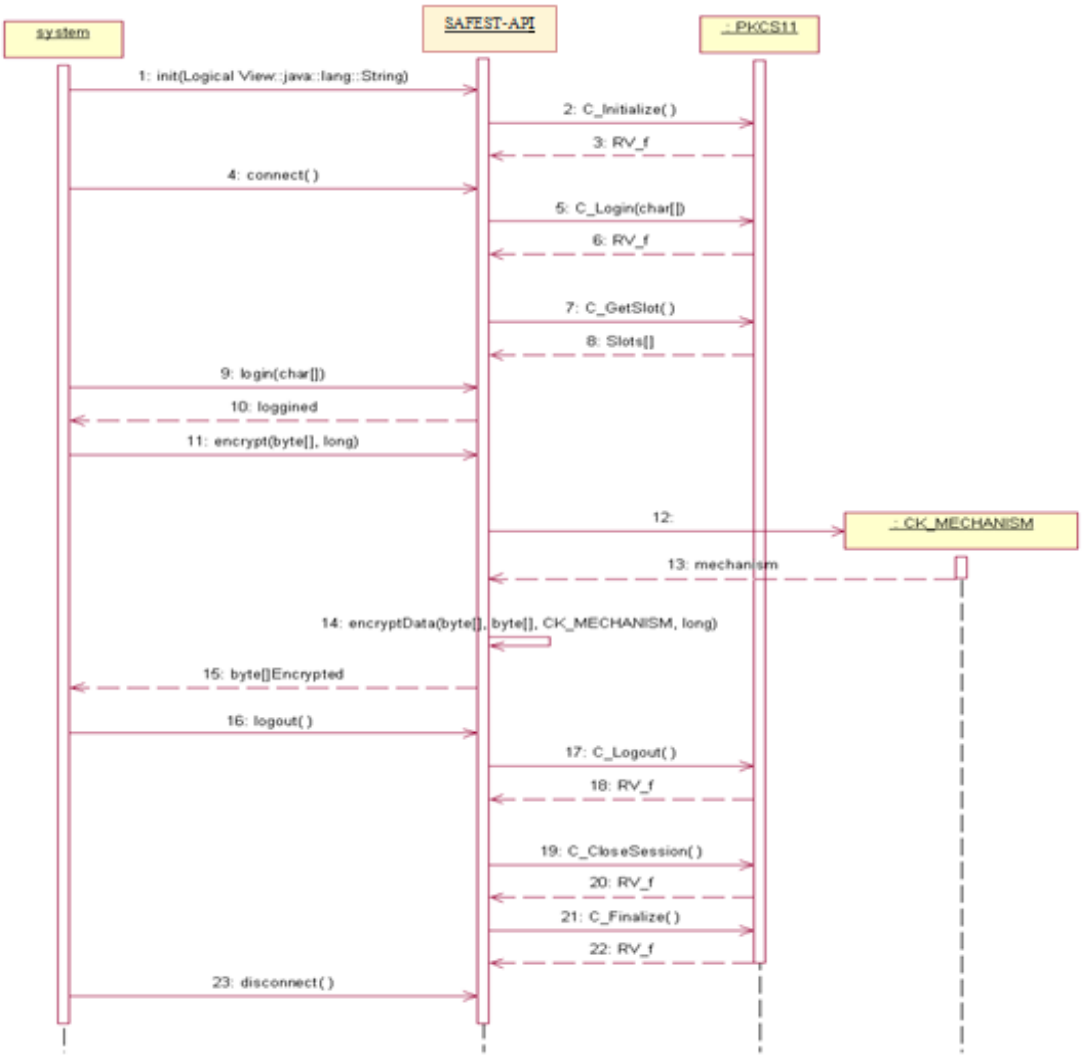

Figure 7: Command Execution Sequence Diagram

In this work, our case study has two sides: the server side and the client side. Figure 8 shows State diagram of the proposed case study from the application client side.

Token Interface (SAFEST-API): this interfaces the token (over PKCS#11).

Client Interface: is a web-based interface to manage and edit the files on the FTP server.

Administrative Interface: is a desktop application and web application to manage and register all application accounts, also manage the process of the token configuration.

Main interface: the user types his/her id then tabs to the password field and types the password and the token PIN code that will be transferred to the user file system interface. The user clicks log in, and if the information is correct he/she will be directed to the user interface otherwise he/she will be obligated to re-enter his/her credentials.

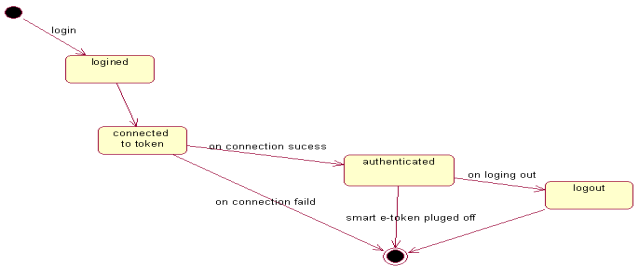

Figure 8: State Diagram of the Application Client Side

The user interface is accessed through any internet explorer that has no special requirements except a java enabled machine where a user at first must login in order to view his/her files and folders on the server. In order to upload a file, he/she has to select the particular file from the local machine then click the button upload. A message appears for him/her to wait until the process is completed. The download operation goes the same way. Figure 9 shows the client application main screen.

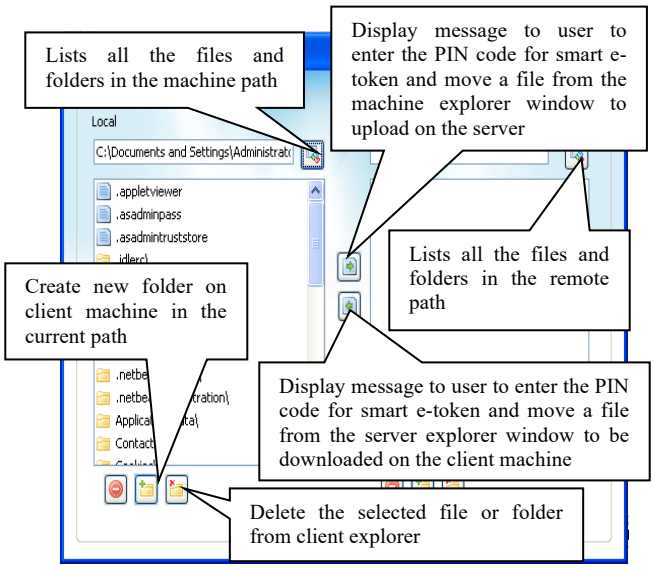

Figure 9: Client application main screen

5.3 Implementation Details

Java is a major tool for our application to run in addition to the USB port of the smart token.

The application is pure software since the developed API to deal with token would be accessible by any other application.

1. Software/Hardware Development Tools

We used C++/Java IDE to develop the SAFEST API and two Tokens for testing.

A. C++/Java IDE to develop the SAFEST API C++ IDEs (visual studio 2005 and Eclipse C++) was developed as the PKI (Public Key Infrastructure) was originally developed in C++.
Java IDEs (Netbeans6.X and Eclipse) as Java is platform independent (the requirement for the project)

B. This work has been implemented using two tokens for testing purposes. The used tokens are products supplied from Egyptian company called "SoftLock".

2. Hardware Interfaces

The system interfaces with the smart token device through vendor's PKCS#11 implementation.

3. Software Interfaces

- PKCS#11 implementation
- SAFEST API
- Browser able to run java applets for client-server, and setting up the database, any platform that supports the following can be used: A) Web server, B) MySql database and C) Java Runtime Environment

4. Communication Protocols

This system uses no custom communication interfaces as it works over the FTP protocol.

Assumptions and Dependency:

The application requires the existence of the PKCS#11's implementation to the environment on which the user works.

The token must implement and provide RSA and AES Cryptographic algorithms.

Token authentication using a PIN, not the one which uses a fingerprint.

Finally, well-known methods inherited from classes such as "java.lang.Object" is used by SAFEST-API implementation. Such methods are "clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait". These methods load a native library of the implementation of the smart Token using the libPath parameter. This parameter is the path of the implementation of the PKCS#11 library.

5.4 SAFEST Performance

Normally, the performance of hardware token processors is low. This work optimizes the performance of the used hardware token. The main delay for executing operations with a token application is caused by the switch into the environment of the token application itself.

We examined the time required for a PKCS#11 signing operation using some existing tokens as opposed to using our SAFEST solution. The signature operation is considered the most widely recognized operation for security tokens, as it is used for authentication and signing of the document that contains the data length. The particular data length is insignificant in this situation, as most applications simply hash the document themselves and just sign the hash digest, to decrease the amount of data that some way or another must be transferred and handled by the token.

Table 4: SAFEST Performance for a PKCS#11 utilizing RSA-1024 signature

Hardware Token	Tokens Solutions	Time
Aladdin	eToken Pro 32k	3.9s
MARX	CrypToken MX2048	2.5s
SoftLock	SAFEST	0.25s

As shown in Table 4, we examined, in particular, an eToken Pro 32k and CrypToken MX2048 as hardware tokens against our SAFEST solution with SoftLock as hardware token on a standard Core i5 PC from HP with an Intel 3.2 GHz. For the signature operation, we use the PKCS#11 C_Sign command utilizing RSA-1024 as the signature component. As shown in Table 4, the SAFEST is faster than the eToken Pro 32k and CrypToken MX2048.

6 Conclusion

This work proposes enhanced secure actions for transferring data using FTP protocol by applying smart token as a case study. A smart token has the capabilities of the smart card, but it is more secured beside some interesting operations. A practical and secure user scheme, based on a smart token device, is proposed.

A secure platform has been developed using implemented APIs and PKCS#11 as RSA standard interface. The proposed API is called SAFEST (Secure Actions for FTP Environment with Smart Token). SAFEST API wraps a standard protocol for implementing the communication between the token and the application that uses it. This API is platform independent, scalable to support more functionality. In addition, it optimizes token usage and adds more security for accessing token objects.

The smart token can process the cryptographic key operations on itself rather than on the host computer which supports high-level of platform independence. Here, through the proposed SAFEST API, standard interfacing to such token devices from any vendor can be implemented through using PKCS#11 interface, developed by RSA labs.

Finally, we can find more than one standard which address key management, though, there is no one of these standards aims at cryptographic tokens. Currently, it is clear there is a move towards additional standards and research that construct a secure interface based on PKCS#11 with cryptographic tokens.

References

[1] A. Gkaniatsou, F. McNeill, A. Bundy, G. Steel, R. Focardi, C. Bozzato, "Getting to know your card: Reverse-engineering the smart-card application protocol data unit", In: Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, December 7-11, pp. 441-450, 2015.

[2] Alfredo Pironti, Davide Pozza, and Riccardo Sisto. 'FormallyBased Semi-Automatic Implementation of an Open Security Protocol.' In: Journal of Systems and Software (2012), pp. 835– 849, 2012.

[3] Athanasios Moralis, Vassiliki Pouli, Symeon Papavassiliou, and Vasilis Maglaris, "A Kerberos security architecture for web services based instrumentation grids", Future Generation Computer Systems 25 (2009), PP. 804-818, 2009

[4] Catherine S. Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack, "User perceptions of security, convenience and usability for ebanking authentication tokens", computers & security 28 (2009), PP. 47- 62, 2009.

[5] Claudio Bozzato, Riccardo Focardi, Francesco Palmarini, and Graham Steel, "APDU-Level Attacks in PKCS#11 Devices", In proceeding of RAID 2016, Paris, France, PP. 97-117, Settembre 2016.

[6] Costas Lambrinoudakis, "Evaluating and enriching information and communication technologies compliance frameworks with regard to privacy", Information Management & Computer Security, Vol. 21 No. 3, PP. 177-190, 2013

[7] Hyun Sook Rhee, Jeong Ok Kwon, and Dong Hoon Lee, "A remote user authentication scheme without using smart cards", Computer Standards & Interfaces, vol. 31 (2009), PP. 6–13, 2009

[8] Igor Bilogrevic, Mohammad Hossein Manshaei, Maxim Raya, and Jean-Pierre Hubaux, "OREN: Optimal revocations in ephemeral networks", Computer Networks 55 (2011), PP. 1168–1180, 2011.

[9] Josef Hertl, "Verifying and improving cryptographic key security in PKCS#11 implementations", DIPLOMA THESIS, MASARYK UNIVERSITY FACULTY OF INFORMATICS, 2014.

[10] Kalpana Singh and Shefalika Ghosh Samaddar, "Enhancing Koyama Scheme Using SelectiveEncryption Technique in RSA-based Singular Cubic Curve with AVK", International Journal of Network Security, Vol.14, No.3, PP. 164-172, May 2012.

[11] Li Wang, Ali Ghorbani, and Yao Li, "Automatic Multi-Step Attack Pattern Discovering", International Journal of Network Security, Vol.11, No.1, PP.32–42, July 2010.

[12] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel, "Attacking and Fixing PKCS#11 Security Tokens", In Proceeding CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

[13] Mohd Nazri Ismail and Mohd Taha Ismail "Analyzing of Virtual Private Network over Open Source Application and Hardware Device Performance", European Journal of Scientific Research, Vol.28 No.2, pp.215-226, 2009.

[14] M. Zaki and Tarek S. Sobh, "NCDS: data mining for discovering interesting network characteristics", Information and Software Technology, Volume 47, Issue 3, Pages 189-198, 2005.

[15] Ruey-Shun Chen, Change-Jen Hsu, Chan-Chine Chang, S.W. Yeh "A Web-based monitor and management system architecture for enterprise virtual private network", Computers and Electrical Engineering, Vol. 31 (2005), PP. 503–524

[16] RSA Security Inc., v2.20. PKCS #11: Cryptographic Token Interface Standard, June 2004.

[17] RSA Security Inc., Draft v2.30. PKCS #11: Cryptographic Token Interface Standard (July 2009), http://www.rsa.com/rsalabs/node.asp?id=2133

[18] S. Fr¨oschle and G. Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In P. Degano and L. Vigan`o, editors, Revised Selected Papaers of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09), volume 5511 of Lecture Notes in Computer Science, pages 92–106, York, UK, Aug. 2009. Springer.

[19] Tarek S. Sobh, Mohamed I. Amer, "PGP Modification for Securing Digital Envelope Mail Using COM+ and Web Services", International Journal of Network Security (IJNS), Vol.13, No.2, PP.79–91, 2011.

[20] Tarek S. Sobh and Yasser Aly, "Effective and Extensive Virtual Private Network", Journal of Information Security, Volume 2, Issue 1, Pages 39-49, 2011.

[21] Vincent Cheval and Bruno Blanchet. 'Proving More Observational Equivalences with ProVerif.' In: Principles of Security and Trust. Springer, 2013, pp. 226–246.

[22] Xiaozhuo Gu, Jianzu Yang, Julong Lan, and Zhenhuan Cao, "Huffman-based join-exit-tree scheme for contributory key management", Computers & Security, Vol. 28 (2009), PP. 29–39, 2009.