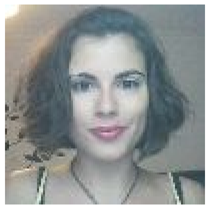
Network and Computer Security
ALAMEDA CAMPUS
GROUP 3
1st semester, 2015/2016
Secure Calendar

NAME	STUDENT NUMBER	EMAIL
Tasneem Akhthar	424480	tasneem.akhthar@tecnico.ulisboa.pt
Rui Botelho	67077	rui.botelho@tecnico.ulisboa.pt
Laura Gouveia 	73202	laura.gouveia@tecnico.ulisboa.pt

Problem

When using a scheduler, a user can create calendars, add, remove, edit events from a calendar, and share calendars with other users. A user would access a server where the calendar data is stored, and could modify the data stored on server.

The main threats encountered are summarized on the table below. These are the STRIDE threats to which our system would be vulnerable to, and to which we must protect.

There is a significant spoofing threat on both ends, that is, both the user and the server identities could be spoofed. On the user end, the user identity can be spoofed to access the user data stored and encrypted in the server, while on the server end the server identity could be spoofed to access the user's data, however without compromising the encrypted data on server.

Tampering with the data is also a major threat. The decrypted data on server is vulnerable to being accessed and changed, given that the connection between server and client would not be secured. While this would not allow for the accessing of user data on server, it would compromise its validity.

A shared secure calendar is also vulnerable to forms of repudiation, since an attacker posing as a trusted third party could access and control the shared events.

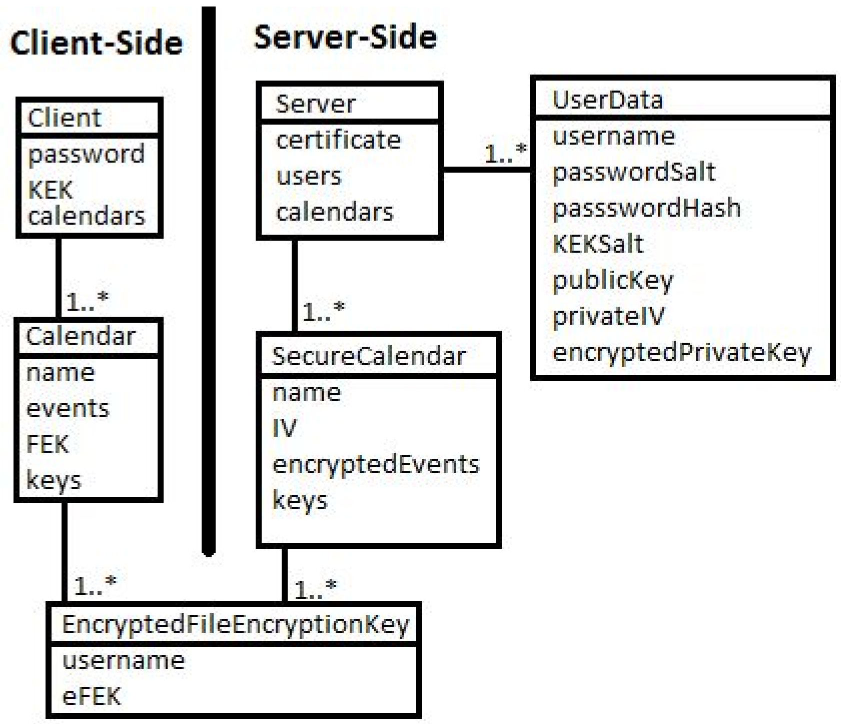
One of the main threats in this type of calendar would be information disclosure, since user privacy is a main concern. Without proper encryption of the server data, an attacker who would gain access to the server, physical or otherwise, would have access to all user data. All the user data should remain private and properly secured.

From these STRIDE threats, both denial of service and elevation of privileges do not have a significant impact on this implementation.

STRIDE	
Spoofing identity	Spoofing another user identity Spoofing the server identity
Tampering with data	Calendars information modification
Repudiation	Repudiation of the confirmation of a shared scheduled event
Information Disclosure	Calendars information privacy breach
Denial of service	No identifiable threats

Elevation of privileges	No identifiable threats

Solution Design



The system will be composed of multiple client applications running and communicating with a single server application. When the client is installed, the server certificate is also installed. The client connects to the server through a secure stream where the server is authenticated using a certificate.

The user registation data generation process is done in the client, which means the password is never sent to the server, because the password is used both for authentication and to decrypt the private key, so that even if the server is compromised the calendars data would still be secret. The user registers by submiting the username and password in the client, then the client generates the passwordHash and passwordSalt that will be used to authenticate the user. Based on the password it generates a KEK(Key Encrypting Key) and KEKSalt, and a RSA private and public keys, so that it encrypts the private key using the KEK and a generated privateIV (Initialization Vector).
Finally the client sends to the server a message containing the information represented in UserData. Whenever a client creates a calendar, a File Encryption Key(KEK) is generated and is encrypted using the client public key, so that the new calendar events are encrypted using the FEK.

When a user wants to use the application, first he authenticates by typing the username and password, requests a calendar by typing the calendar name, and after finishing editing the

calendar the client shares the calendar with another user and this information is saved on the server. When this process begins the client connects to the server through SSL, sends the username and receives the passwordSalt, computes and sends the passwordHash and the user authentication is complete. The client then receives the KEKSalt, the encrypted private key and respective private IV and the list of usernames and respective public Keys. The KEK is computed using the password and the received KEKSalt, the encrypted private key is decrypted using the KEK and the private IV.
The client requests the calendar by sending the calendar name and if the server asserts that the user has access permission to that calendar, the client receives the calendar, named SecureCalendar in the diagram. The FEK is obtained by decrypting with the private key, the calendar events are decrypted using the FEK and calendar's IV. After changing the events, they are encrypted using the FEK. After requesting to share with user B, the client uses B's public key to encrypt the FEK and adds the resulting ciphertext and B's username to the calendar's keys list. Now when B wants to access the same calendar, he obtains the FEK by decrypting the ciphertext using his private key.

To access any calendar data, first the KEK which is not stored anywhere needs to be generated based on the password each time a user authenticates, this way only the user who knows the password can read his calendar, but there is also the problem of being subject to dictionary attacks. On the server side all the private information is encrypted, which means that the server is used simply used for authentication, access control and storage but not sensitive data handling which is done only on the client side.
To allow the sharing of a calendar the user had to somehow save the FEK encrypted together with the calendar. If the FEK encryption was done using symmetric encryption it would need the other user key, but that would expose all data encrypted with that key, for this reason public keys are used to encrypt the FEK.
To garantee integrity, all encrypted information stored together with the respective MAC(Message Authentication Code).

Solution Implementation

The solution consists of three C# language, .NET Visual Studio 2015 projects , SecureCalendarClient, SecureCalendarServer and SecureCalendarLib on which the first 2 projects depend.
Used openssl to generate the server RSA 1024 bits public and private keys, and then create a self-signed X509 certificate. The communication channel uses TLS 1.0 security protocol, key exchange algorithm ECDH ephemeral, hash algorithm SHA1, cipher algorithm AES 256 bits.
Uses PBKDF2 with 128 bits and 10000 iterations, for the passwordHash and KEK generation.
The FEK encryption and users' private and public key, used RSA 2024 bits.The encryption and decryption of calendar events used cipher AES was used with 128 bits key, 128 bits block, PKCS7 padding.

The data model, the server authentication, the user authentication, the calendar encryption and decryption were finished. But the remaining implementation of the integrity assurance procedures using MACs and the sharing of calendars was not finished.
Our solution, altough using medium strengh key sizes, provides a good framework for a remote encrypted calendar storage server that also allow calendars sharing while having good usability, requiring the user to only know a password.

Conclusion

Building a secure system depends on the security goals, on our specific system the privacy of users calendar data was the leading goal. For this reason we choose to separate the data manipulation from the data storage, by designing a client application that only handles private data, and a server application that only handles public data and stores encrypted data.
After arriving at a solution design we started by trying to implement the system as a web application, but it turned out to be unfeasible doing cryptography in Javascript. We then focused on using Windows Crypto API which turned out to be good enough.