

## Project 2 - Email Cryptography

Kevin Lin, Kong Huang

May 4, 2015

# Contents

<b>1</b>	<b>Programming Report</b>	<b>3</b>
1.1	Design of the Email Application . . . . .	3
1.2	Cryptographic attacks and their defenses . . . . .	5
	<b>References</b>	<b>8</b>

# Chapter 1

## Programming Report

### 1.1 Design of the Email Application

Our email application allows a user to encrypt an email message (using Gmail), and then send it securely to another user. The other user can then use our application to decrypt the received message.

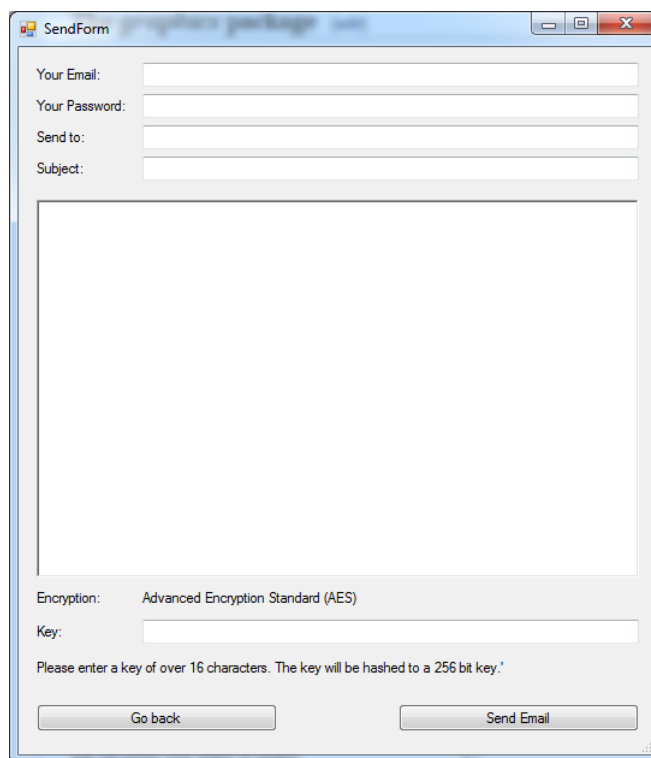
The image shows a screenshot of a Java Swing window titled "SendForm". The window has a standard Mac OS X-style title bar with minimize, maximize, and close buttons. The main content area is divided into several sections. At the top, there are four labeled text input fields: "Your Email:", "Your Password:", "Send to:", and "Subject:". Below these fields is a large, empty rectangular area, likely for the email body. At the bottom of the window, there is a section for encryption settings. It includes a label "Encryption:" followed by the text "Advanced Encryption Standard (AES)". Below this is a "Key:" label and a corresponding text input field. A small note below the key field reads: "Please enter a key of over 16 characters. The key will be hashed to a 256 bit key." At the very bottom of the window, there are two buttons: "Go back" on the left and "Send Email" on the right.

Figure 1.1: Sending an email

The main menu of the application allows the user to choose between sending or decrypting an email. Once the user has chosen, the application switches to

the appropriate interface. Figure 1.1 is an example of the user interface for sending an email.

After the user enters the information, a variety of steps are taken. First, the key that they have provided is hashed and salted using the SHA-256 algorithm. Afterwards, we format the email in order to let the other user know that they are receiving an encrypted email. Finally, we use implicit TLS in order to have a secure network connection.

We used implicit TLS as the primary cryptographic protocol in sending the email because we wanted full encryption for the connection from the start of the session. In implicit TLS, the following approach is used:

1. Client connects to a different port. For example, non encrypted SMTP client connects to port 25 but for implicit TLS, client connects to port 465.
2. The client establishes an encrypted TLS session immediately upon connecting to the server before exchanging any SMTP data, including reading the server's initial greeting.
3. The username and password are sent encrypted.
4. The session remains encrypted throughout the entire connection.

The encrypted session uses TLS 1.2, following the normal TLS handshake protocol. This involves seven general steps:

1. The application sends a "client hello" message to the Gmail server. In this message the supported cipher suites are expressed in the standard OpenSSL notation: "ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH". The application supports all CipherSuites excluding anonymous Diffie-Hellman cipher suites, cipher suites utilizing 64 or 56 bit encryption algorithms, cipher suites utilizing export encryption schemes, and cipher suites utilizing the MD5 encryption scheme. Additionally, a random byte string is generated and sent in this message.
2. The Gmail server will respond with a "server hello" message, choosing one of the cipher suites that the application supports. Along with the message, a session ID, a random byte string, and the server's digital certificate is sent.
3. The application verifies the server's digital certificate.
4. The application creates a premaster secret using the two random byte strings from steps 1 and 2, and encrypts it using the server's public key. Subsequently, it is sent to the server. After this step, both the server and the application can generate the shared secret key using all of the transmitted messages.
5. The application sends a "finished" message encrypted using the shared secret key.
6. The server sends a "finished" message encrypted using the shared secret key.

7. Using the shared secret key, the application and the server can communicate privately (sending emails and receiving confirmation of emails being sent).

However, since encryption always consumes more bandwidth and computational resources, there may be instances when you'll want to encrypt only one channel. In that case, you can use explicit TLS which allows you to choose which channel to encrypt. You can even choose to revert back to a regular (unencrypted) connection and not encrypt any channel at all.

Using TLS provides us with strong authentication, message privacy, and integrity. TLS can help to secure transmitted data using encryption. TLS also authenticates servers and clients to prove the identities of parties engaged in secure communication. It also provides data integrity through an integrity check value. In addition to protecting against data disclosure, the TLS security protocol can be used to help protect against masquerade attacks, man-in-the-middle attacks, and replay attacks.

## 1.2 Cryptographic attacks and their defenses

### Data Eavesdropping

*How it works:*

The majority of network communications occur in an unsecured or plaintext format. This means that an attacker who has gained access to data paths in your network can listen in and interpret the network traffic. In short, eavesdropping is the act of intercepting communications between two points. A specialized program can be used to sniff and record packets of data communications from a network. By using cryptographic tools for analysis and decryption, these recorded packets can be easily read and examined afterward. As an example, Voice over IP (VoIP) calls made using IP-based communication can be picked up and recorded using protocol analyzers and then converted to audio files using other specialized software.

*Our defense:*

Our email application uses Transport Layer Security (TLS), and since TLS encrypts the transmission, the contained data cannot be eavesdropped. However, TLS cannot prevent eavesdropping if the certificate authority (CA) is not safe, which is not our concern since we are properly checking for valid signed certificates from trusted CAs. In addition, TLS will not protect against attacks against the endpoints itself. That is, it will not help you if there are bugs in the used TLS stacks, buffer overflows or bugs in the application logic (aka cross-site-scripting). If the attacker manages to compromise the endpoint in some ways (which is unlikely since they have to compromise Gmail) he will be able to inject itself into the application to get access to the unencrypted data or to the encryption keys.

### Data Modification

*How it works:*

After an attacker has read your data, the next logical step is to alter it. A data modification attack occurs when an attacker modifies the data in the

packet without the knowledge of the sender or receiver. Even if you do not require confidentiality for all communications, you do not want any of your messages to be modified in transit. For example, if you go shopping online, you do not want the items, amounts, or billing information to be modified.

*Our defense:*

Since our transmission is encrypted, the attacker cannot alter the data without the message becoming invalid. Through the standard TLS protocol, encryption involves the usage of HMACs (keyed-hash message authentication codes), so if the attacker wanted to modify the email, the attacker would need to know the secret and the message (amongst others), which is impossible given the usage of the public key infrastructure in TLS. Without knowing these, the attacker can only modify the message arbitrarily, causing the server to deem the email invalid when confirming the authenticity of the sender. For extra clarity, the TLS 1.2 specification Appendix F.2 states: "To prevent message replay or modification attacks, the MAC is computed from the MAC key, the sequence number, the message length, the message contents, and two fixed character strings."

## Data Replay

*How it works:*

A data replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. Suppose Alice wants to prove her identity to Bob. Bob requests her password as proof of identity, which Alice dutifully provides. Meanwhile, Eve is eavesdropping on the conversation and keeps the password. After the interchange is over, Eve (posing as Alice) connects to Bob; when asked for a proof of identity, Eve sends Alice's password (or hash) read from the last session, which Bob accepts thus granting access to Eve.

*Our defense:*

The TLS channel itself is protected against replay attacks using the HMAC in the same fashion as stated in the previous section. The TLS 1.2 specification Appendix F.2 states: "To prevent message replay or modification attacks, the MAC is computed from the MAC key, the sequence number, the message length, the message contents, and two fixed character strings." Also, TLS requires the client and server to exchange a nonce in the hello message. The implementation should verify that the nonce is never repeated in order to prevent the replay attacks.

## Masquerade Attacks/Identity Spoofing

*How it works:*

Most networks and operating systems use the IP address of a computer to identify a valid entity. In certain cases, it is possible for an IP address to be falsely assumed (identity spoofing). The attacker then masquerades as another by falsifying data and gaining an illegitimate advantage. An attacker might also use special programs to construct IP packets that appear to originate from valid addresses. After gaining access to the network with a valid IP address, the attacker can modify, reroute, or delete your data.

*Our defense:*

This type of attack is not much of a problem for TLS connections as TLS

authenticates all parties and encrypts all traffic. Using TLS prevents an attacker from performing IP address spoofing on a specific connection (for example, mutual TLS connections). However, an attacker could still spoof the address of the DNS server. Additionally, if the attacker has access to the email and password of the recipient of a delivered message, the attacker cannot decrypt the message because of the additional AES encryption.

## **Man-in-the-Middle Attack**

*How it works:*

In this attack, an attacker places himself in between a visitor and a web site, impersonating both. With this attack, the browser thinks it is talking to the server on an encrypted channel, and the server thinks it is talking to the browser, but they are both talking to the attacker who is sitting in the middle. All traffic passes through this man-in-the-middle, who is able to read and modify any of the data.

*Our defense:*

The certificate authority system is designed to stop the man-in-the-middle attacks. In TLS, the server uses the private key associated with their certificate to establish a valid connection. The server keeps the key secret, so the man-in-the-middle can't use the site's real certificate; they have to use one of their own. The attacker has to either convince a certificate authority to sign their certificate, or just use it, as is. A man-in-the-middle trying to use a certificate that is not validated by a known trusted CA will be caught immediately.

## **Compromised-Key Attack**

*How it works:*

A compromised-key attack occurs when the attacker determines the key, which is a secret code or number used to encrypt, decrypt, or validate secret information. This key corresponds to the certificate associated with the server. When the attacker is successful in determining the key, the attacker uses the key to decrypt encrypted data without the knowledge of the sender of the data.

*Our defense:*

Depending on the compromised key, the message is still secure based on several factors. Assuming that the compromised key is the key used in the AES encryption scheme, the attacker only needs access to the message delivered to the email of the recipient. As a result, in such a situation, the security relies on the security of the recipient's email service itself. Assuming that the compromised key is the shared secret key used in the TLS protocol, nothing can be determined as the message will always be encrypted with solely an AES scheme, or with an AES scheme and the scheme chosen through the TLS protocol. Assuming that both keys are compromised, the attacker could intercept the encrypted message as it is being sent through the TLS protocol, and decrypt it twice to recover the message. Intercepting this message would be simple for the attacker as long as the attacker can locate the communication channel being used.

# References

- [1] "Common Types of Network Attacks." *Common Types of Network Attacks*. Microsoft, n.d. Web. 03 May 2015.
- [2] Northwood, Chris. "Cryptography, Attacks and Countermeasures." *Cryptography, Attacks and Countermeasures* N.p., n.d. Web. 03 May 2015.
- [3] "Transport Layer Security." *Wikipedia*. Wikimedia Foundation, n.d. Web. 03 May 2015.
- [4] "What Is TLS/SSL?" *Logon and Authentication*. Microsoft, n.d. Web. 03 May 2015.