Email Security

A SECURE CLIENT FOR EXISTING EMAIL SERVICES

Motivation

”When you upload, submit, store, send or receive content to or through our services, you give Google (and those we work with) a world- wide license to use, host, store, reproduce, modify, create derivative works (such as those resulting from translations, adaptations or other changes we make so that your content works better with our services), communicate, publish, publicly perform, publicly display and distribute such content.”

-Google End User License Agreement[5]

Motivation

This means that all your communication is monitored/read by google.

Even if you trust Google (which all of do!), its not safe since it is also shared with companies that collaborate with Google.

It is possible for them to legally steal your ideas and reproduce them.

Bottom line: there’s no privacy from the email service providers. (True for many others along with Google)

Tools Used

JavaMail API from Oracle

Bouncy Castle Library for JAVA

Advance Encryption Standard (AES)

- CBC mode of operation
- PKCS7 padding
- 128 bit key
- 128 bit random IV

RSA Encryption

- Block Cipher mode with OAEP
- 1024 bit keys
- PKCS8 encoding for storing

Tools Used

SHA256

- 128 bit key
- Digest size 256 bit

PBKDF2 (Password Based Key Derivation Function)

- 128 bit random salt
- 10,000 rounds of operation
- Generated key size 128 bit

Solution

Local email client that connects with Gmail for sending receiving email

Securely share public keys via Gmail (Using PBKDF based Symmetric Keys)

Encrypt body of email using session keys

Encrypt session keys using Asymmetric Key Cryptography (RSA)

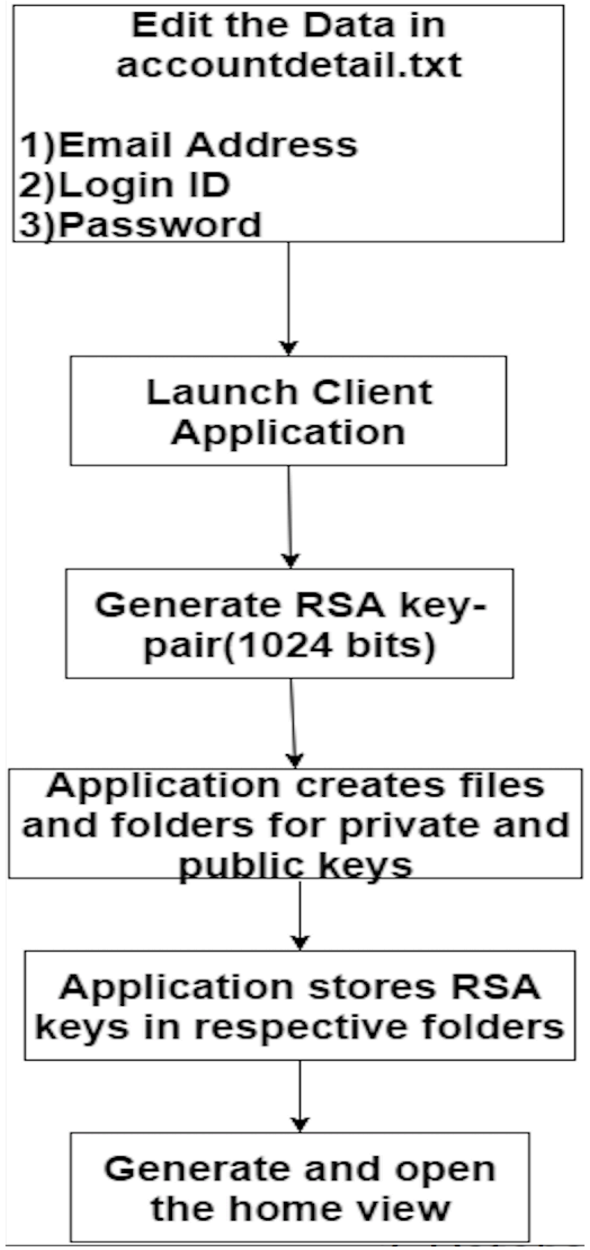
Send both cipher texts via Gmail

Receiver recovers session keys using his private key

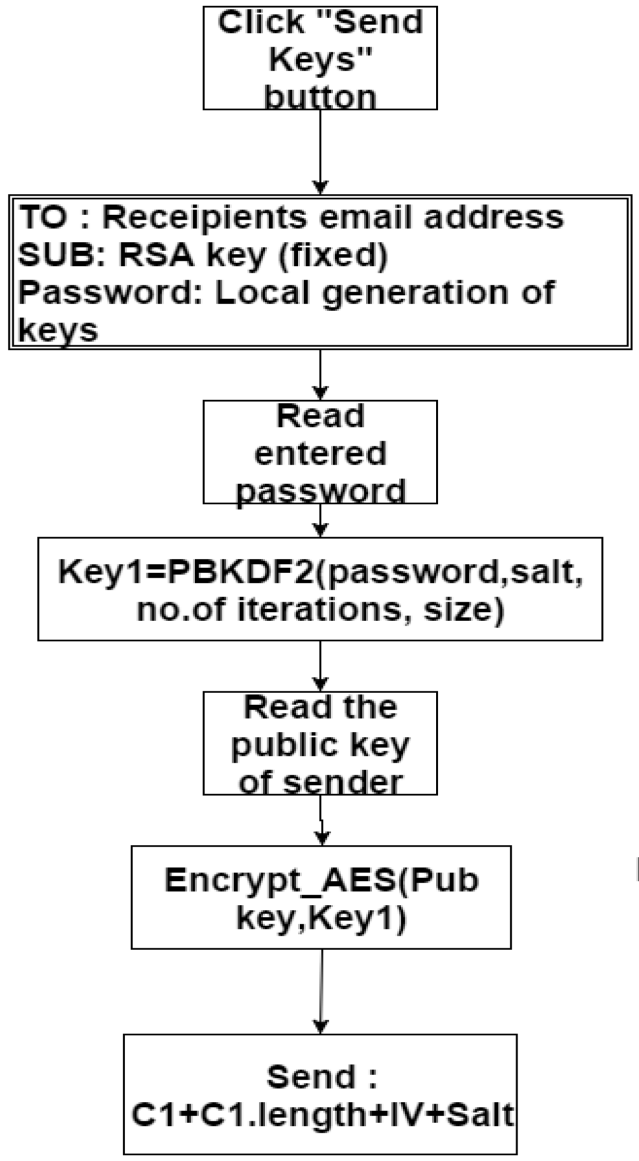
Receiver recovers message using the decoded symmetric key

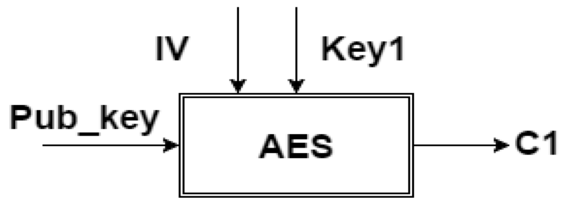
All communications with Google servers use TLS for security

Setup

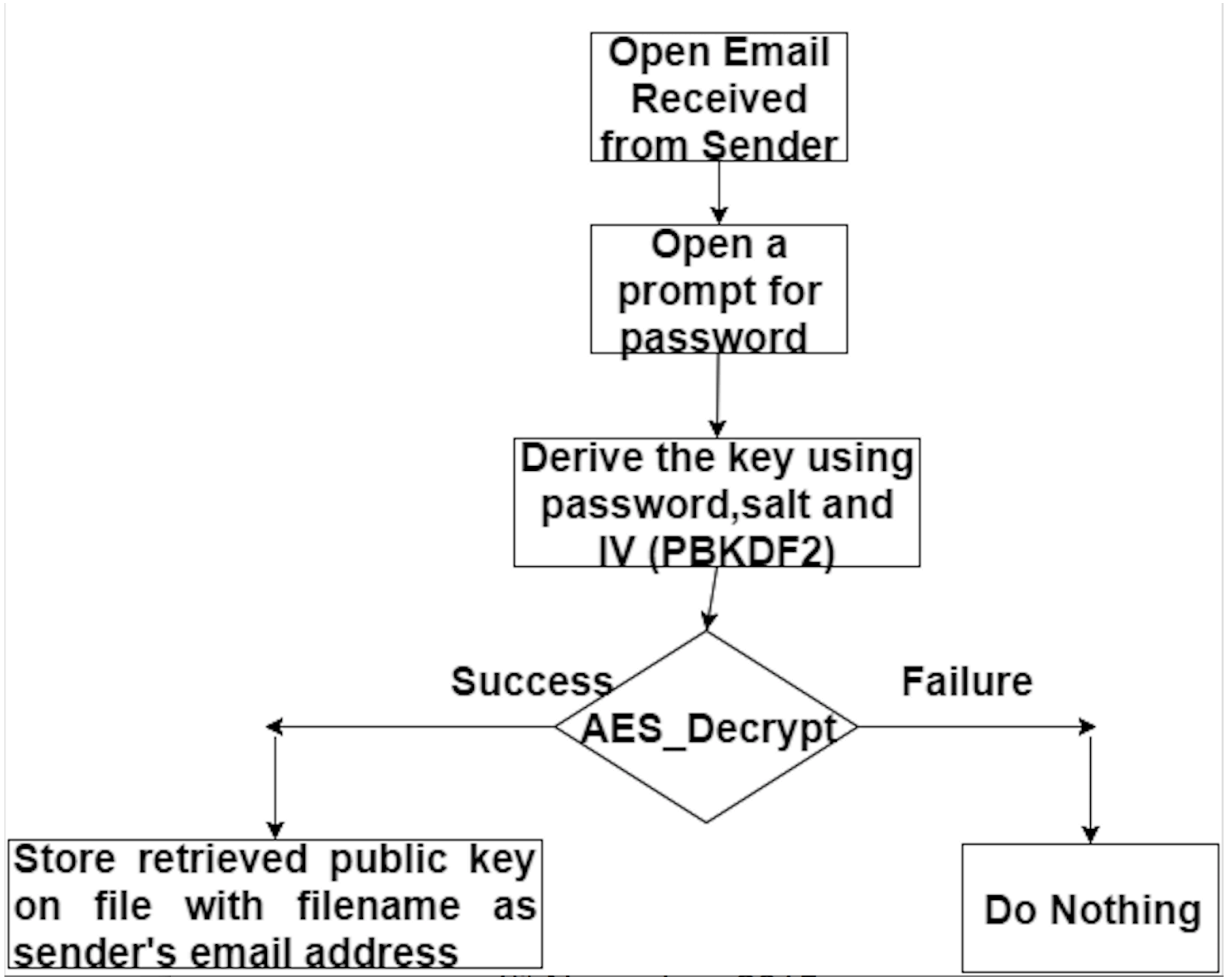

Edit the Data in accountdetail.txt
1)Email Address
2)Login ID
3)Password
Launch Client Application
Generate RSA key-pair(1024 bits)
Application creates files and folders for private and public keys
Application stores RSA keys in respective folders
Generate and open the home view

Key Sharing (Send)


Click "Send Keys" button
TO : Receipients email address
SUB: RSA key (fixed)
Password: Local generation of keys
Read entered password
Key1=PBKDF2(password,salt, no.of iterations, size)
Read the public key of sender
Encrypt_AES(Pub key,Key1)
Send : C1+C1.length+IV+Salt

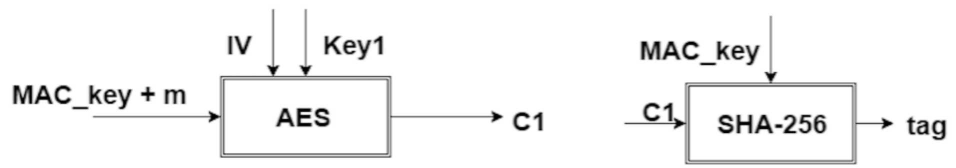


Key Sharing (Receive)

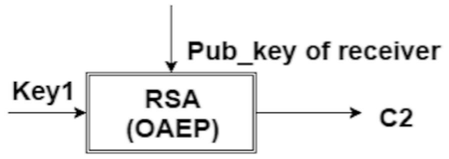

Open Email Received from Sender
Open a prompt for password
Derive the key using password,salt and IV (PBKDF2)
AES_Decrypt
Success
Failure
Store retrieved public key on file with filename as sender's email address
Do Nothing

Email (Send)

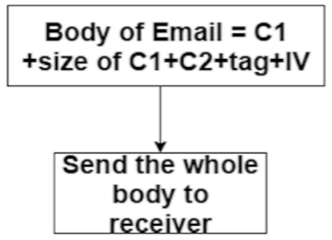
Step 1 : Encrypt Message and Mac_key and create a tag



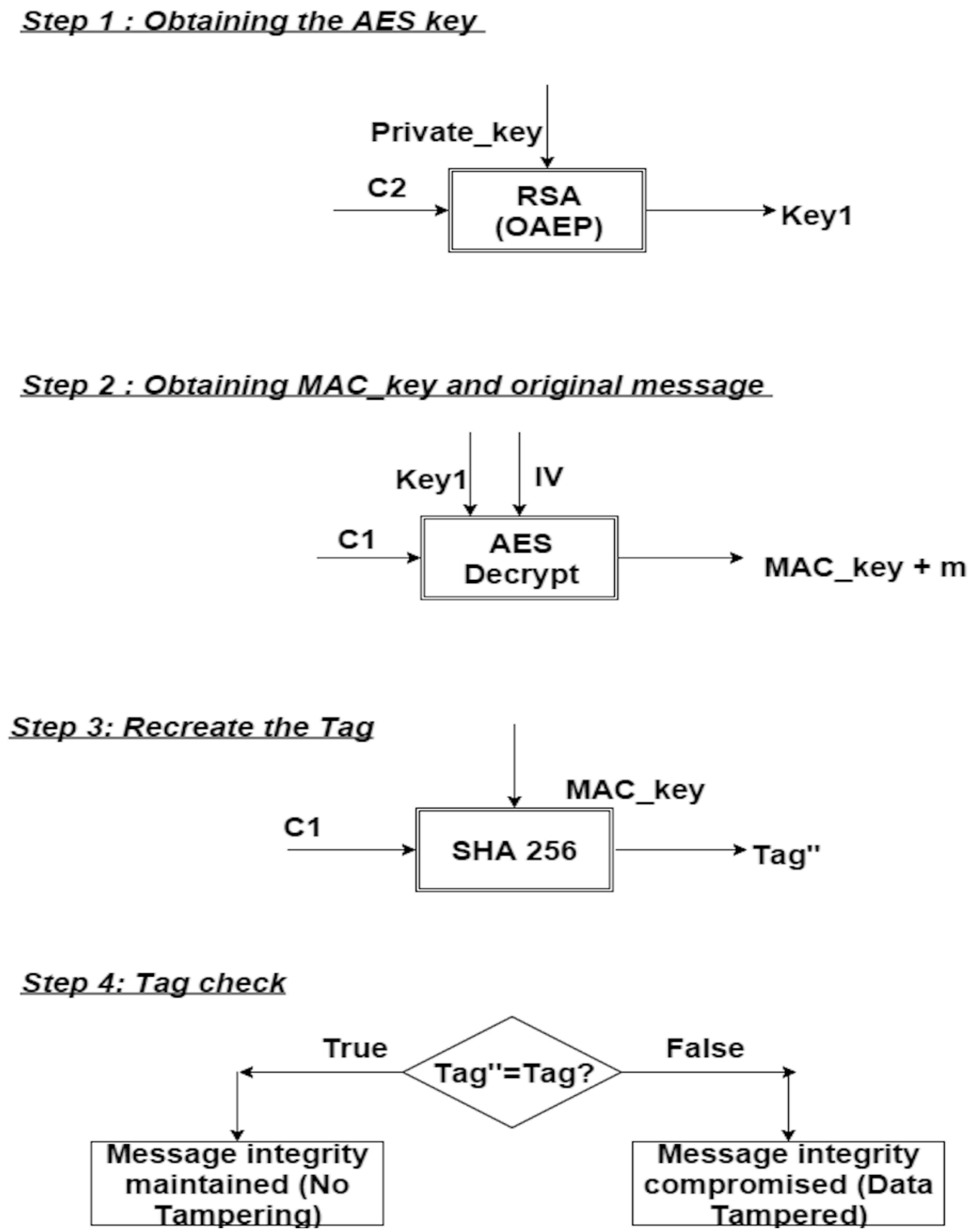
Step 2: Encrypt AES key (key1) with public key of receiver



Step 3: Sending the Email


Body of Email = C1 +size of C1+C2+tag+IV
Send the whole body to receiver

Email (Receive)


Step 1 : Obtaining the AES key
Private_key
C2
RSA (OAEP)
Key1
Step 2 : Obtaining MAC_key and original message
Key1
IV
C1
AES Decrypt
MAC_key + m
Step 3: Recreate the Tag
MAC_key
C1
SHA 256
Tag''
Step 4: Tag check
True
Tag''=Tag?
False
Message integrity maintained (No Tampering)
Message integrity compromised (Data Tampered)

Assumptions

Assumes email address of intended recipient to be correct

Assumes File System of the user to be safe (not infected with any malware or virus)

Assumes password will be shared securely

Assumes TLS connection to be a safe mode of communication

Assumes AES in CBC mode to be non-deterministic and secure with 128 bit keys

Assumes RSA with 1024 bit key to be secure

Assumes SHA256 to be collision resistant

Adversarial Model

TLS based connection from User device to Google server provides security against

- COA (Eavesdropper)
- Known Plain-text attack
- Chosen Plain-text attack
- Chosen Cipher-text attack

Google as adversary

- Has access to cipher text only
- Cannot decrypt without private key of receiver
- Public key cannot be recovered from cipher-text without password

Possible Attacks

Replay attacks

- Google has valid cipher-text of messages on its servers, which can be sent over and over to the recipient

Disruptive attacks

- Google can filter all messages that have “RSA Key” as subject (Thus, preventing key exchange from happening)
- Google can filter messages based on length (i.e. prevent key exchange messages from reaching the recipient)

Denial of service attack

- Google can decide to remove your account or block your IP from accessing their servers

Live Demo!

Conclusion

Implemented Solution is light weight with almost no setup process or learning curve.

Utilizes existing infrastructure such as TLS for communication and Google servers for data storage.

Provides complete privacy from Email service providers when used.

Cross-platform support since entire code base is in Java.

Basically, no more creepy ads about 'cars' when you write an email with the word 'car' in it.

References

[1] Bouncy Castle *http://www.bouncycastle.org*

[2] JavaMail API *http://www.oracle.com/technetwork/java/javamail/index.html*

[3] Stack Overflow *http://stackoverflow.com*

[4] Crypto and Javahelp subreddits *http://reddit.com/r/crypto* & *http://reddit.com/r/javahelp*

[5] Google End User Lisence Agreement *http://www.google.com/intl/en/policies/terms/*

THANK YOU!

AKASH SINGH | ABHIJEET RANADIVE