**Digital envelope approach for Secure Real Time Chats Application**

**Abstract**

The utilization of information technology in our daily life has increased faster than we could imagine. Information or data are transferred through the internet every millisecond. End-to-end encryption has become a key feature of any online messenger application. In this paper, we proposed a lightweight protocol that allows real time encryption and decryption while maintaining computationally impossible to break through. We deployed symmetric and asymmetric encryption in simple chats applications. We were able to get real time performance but we did not provide comparison with other methods. Although we use chats application as our test case, this protocol is also applicable for other use cases that involve secure data transactions.

**Introduction**

As the people's awareness of his/her own privacy increased, any information system that exchanges data between machines through the wire or wireless medium should be in compliance with the CIA which is commonly known as the computer security triad(Confidentiality, Authorization, Authentication).   
  
**Confidentiality**

Confidentiality is keeping the data of another person or entity private.

**Integrity**

Integrity is keeping the data unmodified from sender to receiver end.

**Availability**

Availability is to ensure data are accessible by authorized users or entities.

**Related work**

A number of popular algorithms were proposed like DES(Data Encryption Standard), Triple DES, RSA(Rivest, Shamir, and Adleman), AES (Advanced Encryption Standard) and many more.

**DES** is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology(NIST). It takes 64-bit long block plain text and 56-bit long key as input and output cipher text.

As the computation power improved, DES became too easy to break through with brute force. **Triple DES** also known as 3DES was published in the year of 1995 to address the issue arising from modern cryptanalytic techniques and supercomputing power. 3DES operate by doing DES three times, as a result three sets of keys are required. There are three option available:

Key option 1: k1, k2 and k3 are independent

Key option 2: k1 and k2 are independent and k3 = k1

Key option 3: k1, k2 and k3 are identical

This method is widely used because many hardware were designed to accelerate the process of DES. This method could extend the longevity of those machines.

**Advanced Encryption Standard (AES)** is an approved symmetric key cryptographic algorithm that is a block cipher. The AES algorithm is capable of using cryptographic keys of 128,192, and 256 bits. It was established by the U.S.National Institute of Standards and Technology (NIST) in 2001.

**RSA** is a public-key cryptosystem (PKC) named after Rivest, Shamir, and Adleman, who publicly described the algorithm in 1977. A pair of distinct keys are required, only the encryption key is made publicly available while the decryption key is kept secretly by it’s owner. The key generation algorithm involves two very large prime numbers and the algorithm is very light weight. Reversal of it has to do with a problem called prime factorization. Brute force is even computationally expensive because the minimum key length is 2048-bits.

**Method**

In this paper we attempt to solve two issues:

1. Efficiency
   1. Both encryption and decryption of RSA involve calculation of large numbers and exponent. Therefore, we opt to encrypt the message with no fixed length with AES.
2. Frequent key exchange to prevent cryptanalysis
   1. Approach that only uses AES has to ensure random symmetric keys used are frequently updated to prevent other parties from understanding the ciphertext through similarity of ciphertext. Moreover, each session of key exchange also exposes us to tracking. Therefore, we proposed to encrypt the random symmetric key through RSA using the public key of the receiver. Through this approach, keys used during AES encryption can be exchanged privately each time. Since the random symmetric key used is 128-bits long, the probability of collision is sufficiently low.

**Implementation**

The system is written in python 3.8 and tested on Windows 10 desktops. The desktops used are equipped with Intel 9th generation core i7 CPU with 16 GB RAM. We used TCP as a transfer protocol. We have written three programs, they are server, client and another one to simulate Certificate Authorities (CAs). Beginning of each conversation or connection, the client will request the public key of the receiver from CA, after the server has accepted the connection it will request the caller's public key from CA based on the user id received from the caller.

In order to create a secure digital envelope we first AES encrypts the message using a random symmetric key to convert plaintext into an encrypted message, then RSA encrypts this key using the public key of the expected receiver to generate the encrypted key. Both ciphertext and encrypted key are sent together to the receiver.

To interpret the digital envelope received, first RSA decrypts the encrypted key using the receiver's private key to retrieve the random symmetric key. Lastly, AES decrypts the encrypted message using the random symmetric key to obtain the message.

Link to complete code: <https://github.com/JonahYeoh/digital-document-sharing>

Figure 1 shows the creation and interpretation of the digital envelope.

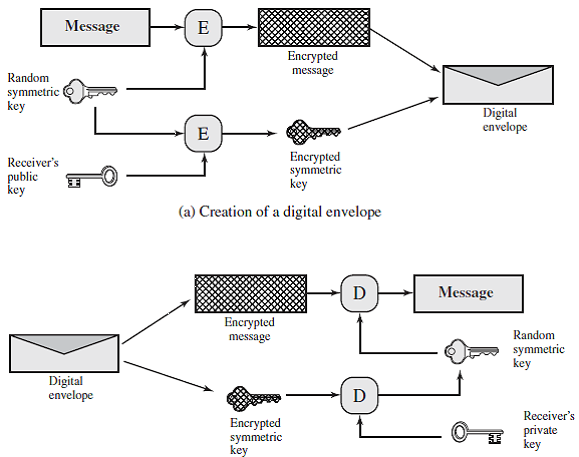


Figure 1

**Experiment**

Users are able to send and receive encrypted messages in real time speed using our system. Messages were not modified from sender to receiver end. For simplicity's sake , cryptanalysis and stress tests were not performed.

**Conclusion**

The protocol we proposed was able to achieve our goal of secure real time communication. However, many aspects like scalability and vulnerability of our protocol were not tested. Besides this, we did not compare the effectiveness of multiple protocols. Future research effort will focus at testing the robustness of the protocol.