RESEARCH ARTICLE OPEN ACCESS

Multi-Agent Based PGP Architecture

Babak Nouri-Moghaddam[1], Mohammad Ismaeil Shahabian[2], Hamid Reza Naji[3]
[1]Graduate University of Advanced Technology, Kerman, Iran
[2]Graduate University of Advanced Technology, Kerman, Iran
[3]Graduate University of Advanced Technology, Kerman, Iran

Abstract
Pretty Good Privacy (PGP) is a package for securing emails, files communications. It is an open-source package, which is available online for users. PGP provides some of the most important security services like Authentication, Confidentiality, and Integrity. PGP Also applies compression techniques for compressing messages and reducing their size. Also it uses Radix-64 encoding/decoding scheme for email compatibility.
The classic PGP has been formed by independent components and uses a hierarchal structure in which each component is responsible for providing one of the services or features in PGP. This hierarchal structure forces all the components, even the independent ones to be executed in a linear way. Because of this structure, each component waits idle for long a time. As a result, the classic PGP has low performance and high execution time. By studying this structure, we find out that we can redesign the architecture by using Multi-Agent systems to eliminate bottlenecks. With this new design, we can achieve higher performance and faster execution time than the classic PGP. In the proposed scheme, each Agent handles one of the PGP's components and in the implementation semaphores will be used to handle each agent. By using this technique, we will have concurrency between the agents and as a result the idle time will decrease and the proposed scheme will get higher performance and lower execution time than the classic PGP. The experimental results show that our scheme runs 30% faster than the classic PGP with different configurations of computer hardware.

Keywords: Pretty Good Privacy, Multi-agent systems, Email Communications, Authentication, Confidentiality

I. Introduction

PGP is a well-known security package, which provides authentication and confidentiality along with other security features. Commonly PGP users use this package for signing and encrypting/decrypting emails and files to increase their communication security. For non-commercial users PGP is a free package and available online, but for commercial use, it has a low-cost version. PGP is on the Internet Standards Track, it is under active development, and its current specification is RFC 4880[1,2].

PGP applies the chain of actions like hash functions, compressing algorithms, symmetric cryptography, and public-key cryptography on a message to achieve its goals. PGP considers the algorithms like a black box and this feature makes it independent of the algorithm implementation. The users are free, whether to use the suggested algorithms or to use their own algorithms. There is one condition that says the sender and receiver ought to use same algorithms.

In 1991, Phil Zimmermann introduced the first design of PGP. He was a member of a community against nucleus technology. He wanted to design the architecture for securing their community members' communication (e.g. E-mail). The first version of PGP included the symmetric algorithm. Zimmermann and his team established their own company in 1992 and started to adjust PGP. For developing PGP's architecture, he has done the following [3,4]:

1) First, he had designed the PGP architecture using independent blocks. For each block, he selected the best available algorithms.
2) The architecture he had designed was completely independent from platforms (e.g. Operating systems and processors).
3) He declared the whole package, source code and its documentation free, and made it available online.
4) Finally, he entered into an agreement with a company to release low-cost commercial version of PGP.

Users from around the world started to use PGP, and in short PGP became very popular. Some of the reasons for this explosive growth are [3,4]:

1) The most powerful reason for this growth is that PGP is a free service, and available for all platforms.
2) PGP, based on the algorithms that have a high history of studies and reviews in public is considered extremely secure. More specifically, for the public-key encryption, it includes RSA,

DSS, and Diffie-Hellman. For symmetric encryption it has an option to choose between CAST-128, IDEA, and 3DES, and it includes SHA-1 for hash coding.
3) PGP covers wide range of application e.g. encrypting files or securing message which would be used for communication over the Internet or networks.
4) PGP is now on an Internet standards tracking (RFC 3156; MIME Security with OpenPGP [2]). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

From what published until now, there is not any report of security bridges in PGP[5]. Actually, in theory, the first versions have some security problems, but they had been solved in the later versions, so the standard suggests using later versions instead of the old ones.

PGP officially has been used for email contents and attachments encryptions. After 2002, various versions of PGP have been released to support different application like: distribute encryption to manage central servers. PGP can provide a wide range of services like email and attachment security, digital signature, encrypting of the whole hard disk, security of files and folders, encrypt and batch file transferring, encrypted HTTP request/ response on the client server architecture [3,5].

By studying this structure, we find out that we can redesign the architecture by using Multi-Agent systems to eliminate bottlenecks. With this new design, we can achieve higher performance and faster execution time than the classic PGP. In the proposed scheme, each Agent handles one of the PGP's components and in the implementation semaphores will be used to handle each agent. By using this technique, we will have concurrency between the agents and as a result the idle time will decrease and the proposed scheme will get higher performance and lower execution time than the classic PGP. The experimental results show that our scheme runs 30% faster than the classic PGP with different configurations of computer hardware.

The rest of the paper has been structured as follows. In section 2, we first survey the Classic PGP's structure. Section 3 shows the theoretical implementation and time analysis of classic PGP. Section 4 describes our proposed scheme and then we will discuss about its time analysis. In Section 5, we discussed some practical implementation issues. Section 6 shows the results of the implementation and compression between two schemes' results. Finally, Section 7 concludes this paper and points out the future works.

II. Related works

2.1 Notation

We use the following notation throughout this paper:

- S: Sender
- R: Receiver
- M: Message
- MD: Message Digest
- K_s: Session key used in symmetric encryption scheme
- PR_a: Private key of user A, used in public-key cryptography
- PU_a: Public key of user A, used in public-key cryptography
- EP: Public-key encryption
- DP: Public-key decryption
- EC: Symmetric encryption
- DC: Symmetric decryption
- H: hash function
- || : concatenation
- Z: compression has been using the ZIP algorithm
- R64: conversion to Radix 64 ASCII format

2.2 Classic PGP

PGP Package is a combination of different cryptographic techniques; it uses symmetric and asymmetric cryptography to provide various security services. The main objective of PGP is to provide authentication and confidentially in the best way possible. Digital signature and Public key cryptography schemes have been used to achieve authentication. First, it computes a message digest using Hash function and then encrypts the MD with sender's Private Key. At the last step, the results prepended to the message. RSA and SHA-1 are recommended by PGP's standard for Public key encryption and message digest computation. For confidentiality, PGP applies symmetric encryption and Digital envelope schemes. After concatenating phase, PGP chooses random key as a session key and encrypts the message with symmetric key algorithm using the random session key. For session key transmission, the sender encrypts the random session key with receiver's Public Key, and then the result will prepended to the encrypted message. CAST, IDEA, or 3DES is recommended by PGP's standard for symmetric key encryption. In addition, the standard recommends using RSA or Diffie-Hellman for key exchanging.

Actually the steps we talked about, was half of the protocol. Another half of the action takes place on the receiver side. Signature verification, key exchange, message decryption, message Integrity check, and other phases take place on the receiver side. For this reason, we separate two side's actions and explain them in separate sections.

2.2.1 On the sender side:

Sender will do the following actions to create and send PGP packets (Figure 1) [3]:

a) The sender generates the message.
b) Compute the MD by using a Hash function, e.g. SHA-1.
c) The MD will be encrypted with senders PR_a and using a Public Key algorithm e.g. RSA.
d) The signed MD prepended to the Message.
e) Compression algorithm (Z) like ZIP function will be used to reduce the size of the new message.
f) After step 5, K_s is generated by the random session generator function. Note that the each Ks will be used just one time.
g) The EP e.g. CAST will be applied to the compressed message.
h) For transmitting the K_s, it will be encrypted by applying RSA or Diffie-Hellman using receiver's PU_b
i) The result prepended to encrypted message and then sender will transmit the whole packet to the receiver.

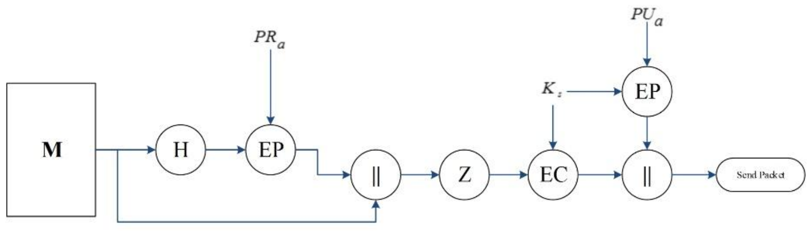

Figure 1. PGP sender side flowchart

2.2.2 On the sender side:

The receiver does the following actions to get the original M, authenticate sender and check M integrity (Figure 2)[3]:

a) Get the encrypted $K_s$by decrypting it. For this, propose receiver uses PR_b with the Public Key algorithm.
b) After getting the K_s, receiver decrypts the encrypted zip message.
c) The result will be decompressed. After decompressing, the receiver will get the original message and signed MD.
d) For authentication and integrity check, receiver decrypts the signed MD with senders PU_a and gets MD.
e) In addition, receiver computes the MD' using the Hash function.
f) Finally, the receiver compressed MD and MD'. If the compressed MD is equal to MD', the sender is considered as valid user and the message will be approved too.

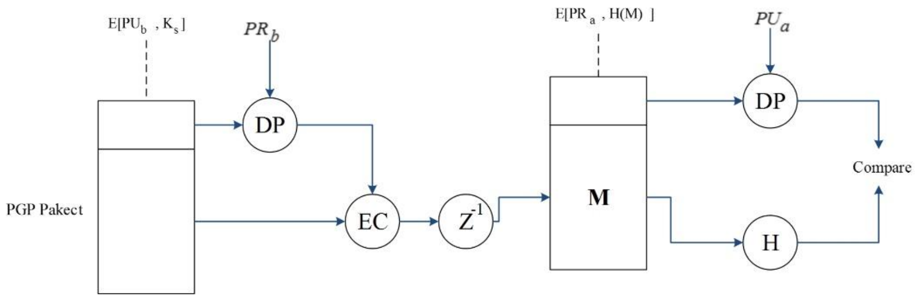

Figure 2. PGP receiver side flowchart

PGP package performs integrity check on the message to make sure the message did not change on the way. Also by validating the digital signature, the receiver will ensure that the message created by valid sender.

There is one necessary condition in using PGP; each entity in PGP communication (e.g. Sender or Receiver) should know the other entity's Public Key. In practice, using Public Keys like the way we talked has some issues, because Public Keys can be forged or eavesdropped in the communication. In the latest versions of PGP, they solve this problem by using some kind of certificate mechanism.

Classic PGP's architecture uses the hierarchal design and does not talk about how each component in PGP will be executed. In the next section, we will analyze the Classic PGP's execution model and time consumption.

III. Theoretical implementation and time analysis of classic PGP

3.1 Sender Side Implementation:

As it shown in figure 3, the PGP hierarchy has been implemented by applying number 1 to 9 for each operation in PGP.

1) *Message digest generator function*: we use SHA-1 hash function for computing message digest. SHA-1 takes a string with maximum length of 2^{64} bits as input and then returns 160-bit message digest.
2) *Public key encryption*: for signing message digest, the sender will encrypt the message digest by applying the RSA function with its own private key.
3) *Merge function*: this function prepends the singed message digest to message.
4) *Compression function*: result of the Merge function, will be compressed by using ZIP function.
5) *Session key generator function*: this function generates 128-bit random key.
6) *Public key encryption*: the session key will be encrypted with RSA using the receiver's Public key.
7) *Symmetric encryption function*: CAST function will be used for encryption of the compressed message.
8) *Merge function*: encrypted session key will be prepended to the encrypted message.
9) *Encoding function*: the result of step (h) will be encoded with Radix64 function.

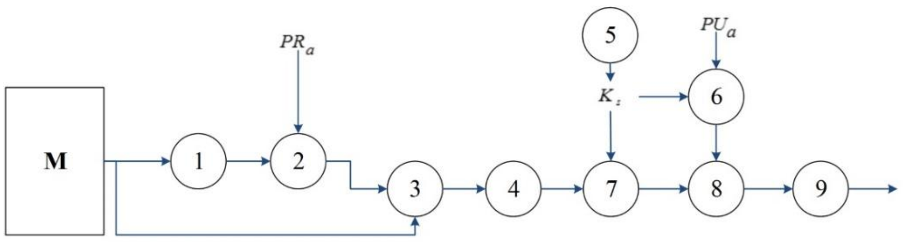

Figure 3. Sender side implementationOperations 1, 2, and 3 cover the authentication and integrity services; Operations 5 to 9 cover confidentiality.

3.2 Time analysis of Sender side operations:

If we assume T is the average execution time of each operation and functions are executed sequentially and separately, then the total execution time will take 9T for sending one message (Figure 4):

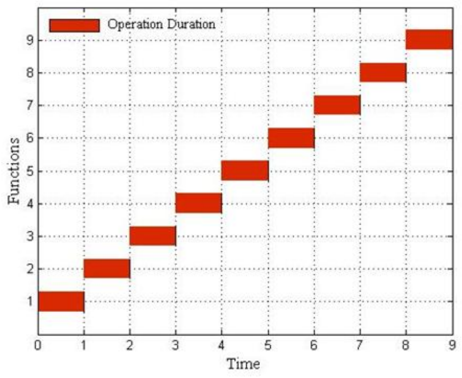

Figure 4. Time analysis of Sender side operations

3.3 Receiver Side Implementation:

According to theFigure 5, PGP's hierarchy has been implemented with operations 1 to 9. Each operation is explained as follow:

1) *Decoder function*: it takes an incoming packet as input and decodes it using Radix64 decoder. The output will be used as input to theSeparator function.
2) *Separator function*: incoming packets consist of encrypted session key and encrypted message; Separator function separates these two parts from each other.
3) *Public key decryption function*: for extracting session key from an incoming packet, Receiver decrypts the encrypted session key with its private key.
4) *Symmetric decryption function*: after extracting the session key, the receiver decrypts the rest of the packet with CAST function using the session key.
5) *Decompress function*: the result of the step (4) will be decompressed with Unzip function.
6) *Separator function*: the decompressed packet consists of two parts. Part one is the original message and part is the digital signature; this function separates these two parts from each other and each part will be used as input to the different function.
7) *Public key decryption function*: for signature validation, the receiver will decrypt the digital signature with the sender's public key.

8) *Message digest generator function*: SHA-1 hash function will be used for computing message digest.
9) *Comparison function*: the result of the steps 7 and 8 will be compared to each other, if these two match each other the sender is valid one. Otherwise, message has been modified or sender is not the one who has claimed.

Operations 2 to 4 are covering confidentiality service; and operations 7 to 9 will cover authentication and integrity services.

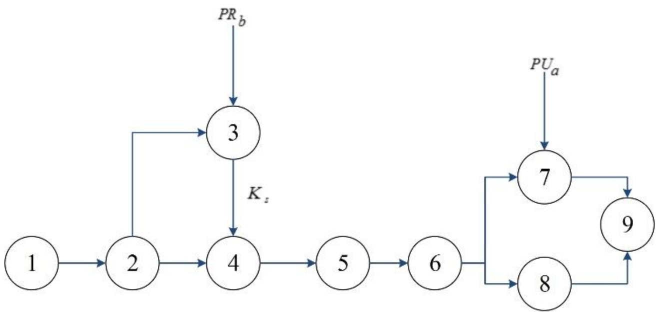

Figure 5. Receiver Side Implementation

3.4 Time analysis of Receiver side operations:
Same as the sender side operations, we assume T is the average execution time of each operation functions are executed sequentially and separately, then the total execution time will take 9T for getting the original message (Figure 6):

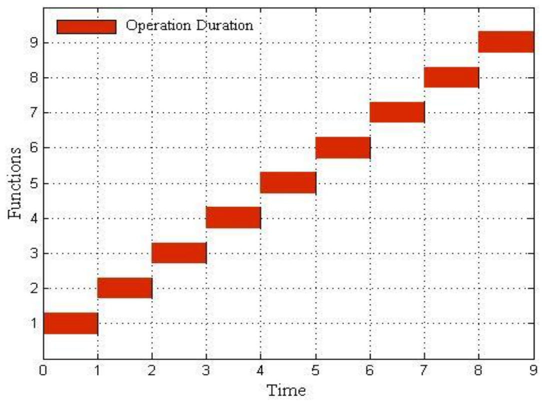

Figure 6. Time analysis of Receiver side operations

IV. THE PROPOSED SCHEME: Fast PGP using Multi-Agent systems

In the previous section, we presented classic PGP's implementation. The results of analyses show some PGP functions are independent from others which can be run simultaneously. In follow, we discuss how to use Multi-Agent systems for running and handling independent function simultaneously.

4.1 Applying Multi-Agent architecture to Sender side:

Figure 7shows that how we can use Multi-Agent architecture to run independent functions simultaneously. In the following, we will explain our proposed scheme:

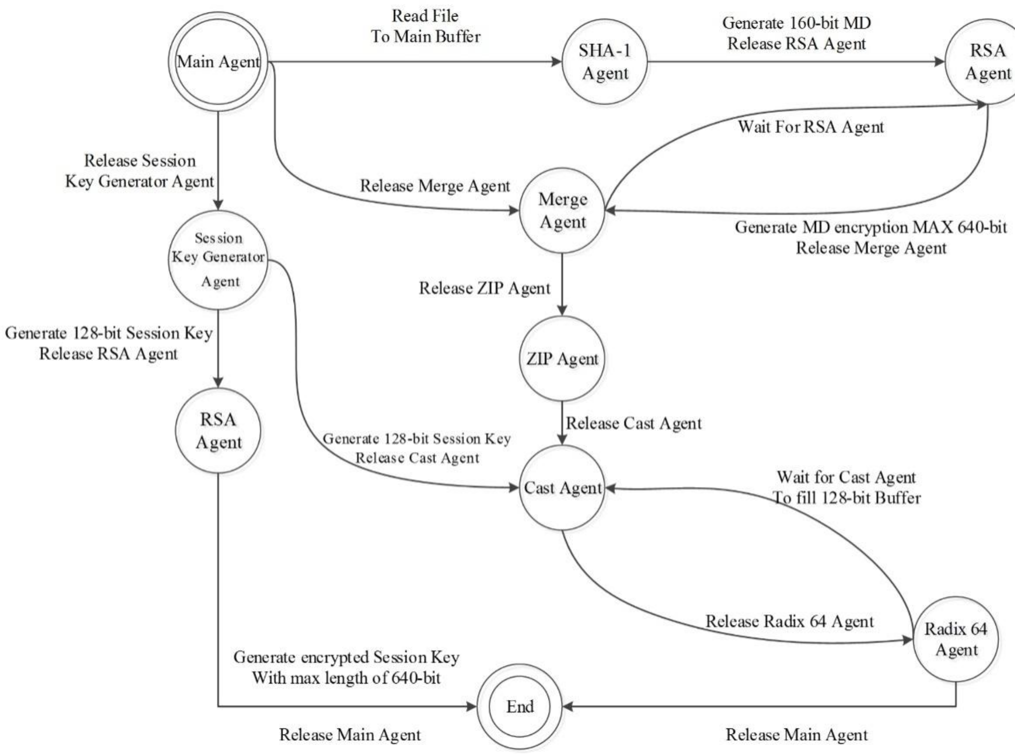
Main Agent
Read File To Main Buffer
SHA-1 Agent
Generate 160-bit MD Release RSA Agent
RSA Agent
Wait For RSA Agent
Release Session Key Generator Agent
Release Merge Agent
Merge Agent
Generate MD encryption MAX 640-bit Release Merge Agent
Session Key Generator Agent
Release ZIP Agent
Generate 128-bit Session Key Release RSA Agent
ZIP Agent
RSA Agent
Release Cast Agent
Generate 128-bit Session Key Release Cast Agent
Cast Agent
Wait for Cast Agent To fill 128-bit Buffer
Release Radix 64 Agent
Radix 64 Agent
Generate encrypted Session Key With max length of 640-bit
End
Release Main Agent
Release Main Agent

Figure 7. Data flaw graph of the proposed scheme on the sender side

Message digest generator function uses the SHA-1 Agent to compute a 160-bit message digest. Thus for signing message digest, the RSA Agent should begin after SHA-1 Agent completed its result. Signing message digest can run concurrently with Merge Agent; and the digital signature will be prepended at the end of the original message. Zip Agent starts depends on the result of merging phase then it should start after the Merge Agent. The random session key generator Agent could run concurrently with operation (a) to (b), because the operation of this Agent is independent of them; but the result of the key generator Agent should be ready before CAST started.

CAST Agent will be started after the compression phase. Encrypting session key with RSA depends on the result of the random session generator Agent. As a result, it should be started after this Agent but it can run concurrently with functions 1, 2, 3, d and 7; because there is not any dependency between them. Note that the Public key encryption needs more time than any other operation.

The Merge Agent can be removed, as we can use the result of the Agents 6 and 7directly as input to the Radix64 encoding Agent and by this way Radix64 can run concurrently with Agent 6 and 7. Radix64 encoding Agent writes the output directly into the buffer.

4.2 Time analysis of applying Multi-Agent architecture to Sender side:

By analyzing proposed scheme on the sender side, we can see that in theory execution time of PGP will be decreased to 3.5T. Compared to the classic PGP, the execution time of the proposed scheme is less than half of the classic PGP. Figure 8shows the time analysis of the proposed scheme:

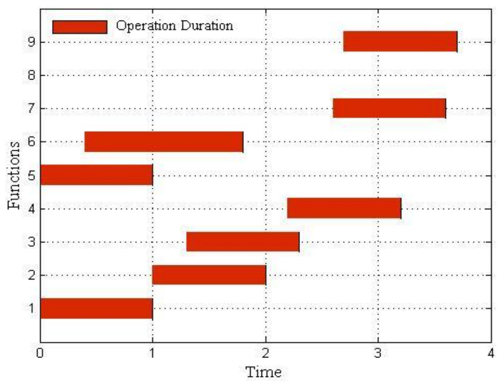

Figure 8. Time analysis of applying Multi-Agent architecture to Sender side

4.3 Applying Multi-Agent architecture to Receiver Side:

We are proposing new design for the receiver side using multi agent systems; the proposed scheme has been shown inFigure 9.In the following, we explain our proposed scheme on Receiver Side:

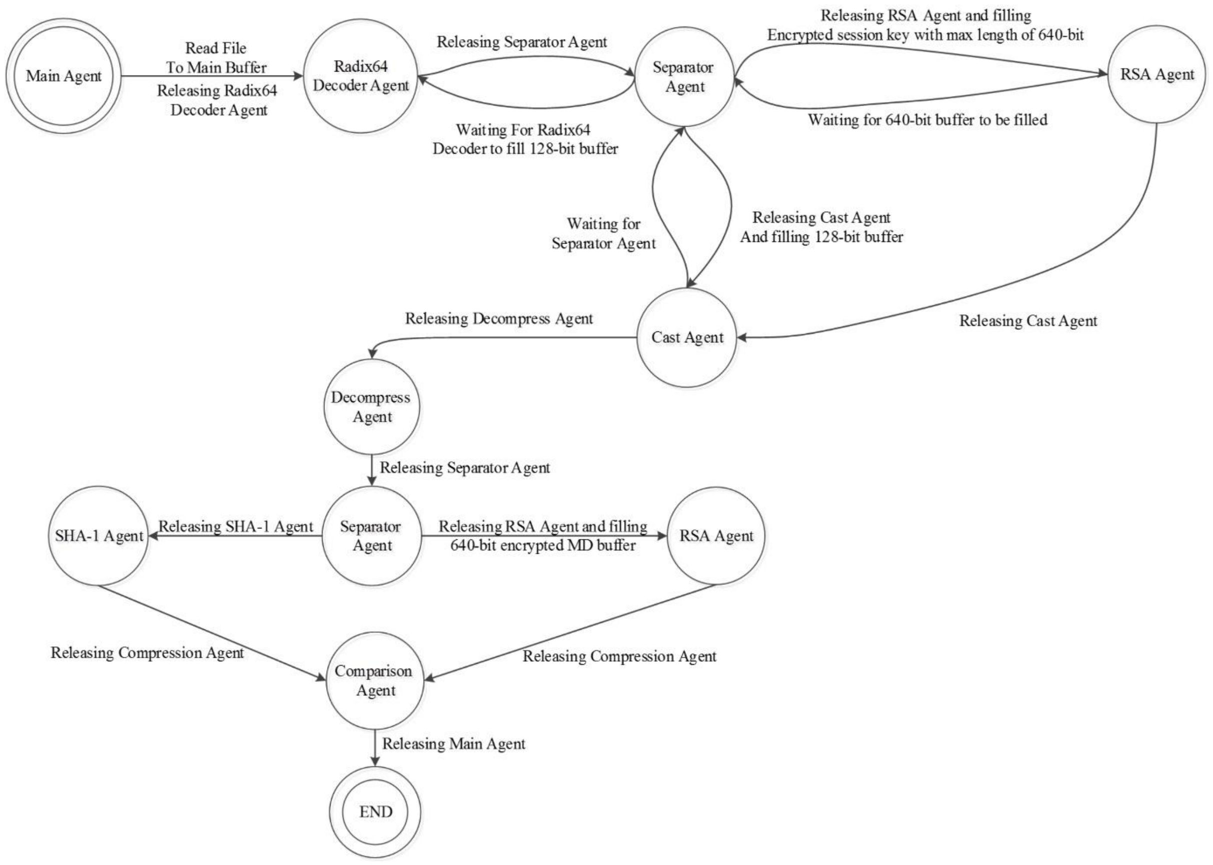
Main Agent
Read File
To Main Buffer
Releasing Radix64
Decoder Agent
Radix64
Decoder Agent
Releasing Separator Agent
Waiting For Radix64
Decoder to fill 128-bit buffer
Separator
Agent
Releasing RSA Agent and filling
Encrypted session key with max length of 640-bit
RSA Agent
Waiting for 640-bit buffer to be filled
Waiting for
Separator Agent
Releasing Cast Agent
And filling 128-bit buffer
Releasing Cast Agent
Cast Agent
Releasing Decompress Agent
Decompress
Agent
Releasing Separator Agent
SHA-1 Agent
Releasing SHA-1 Agent
Separator
Agent
Releasing RSA Agent and filling
640-bit encrypted MD buffer
RSA Agent
Releasing Compression Agent
Releasing Compression Agent
Comparison
Agent
Releasing Main Agent
END

Figure 9. Data flaw graph of the proposed scheme on the receiver side

Radix64 Agent receives incoming packets and starts decoding the packet; the result of this operation will be a data in ASCII format.

By knowing the length of the encrypted session key, the Separator Agent can be executed simultaneously with the Radix64 Agent. Separator Agent can separate the encrypted session key and the encrypted compressed message from each other while the Radix64 Agent decodes the incoming packet. Immediately after filling the encrypted session key buffer by Separator Agent, RSA Agent starts to decrypt Session key using the receiver's private key.

CAST Agent initiates after the RSA Agent should complete extraction of the session key. If the random session key got ready, the CAST Agent will wait for the Separator Agent to fill the CAST Agent buffer with compressed message, any time that the buffer is filled, the CAST Agent will start to decrypt it.

When CAST256 Agent finished decrypting, Decompress Agent starts to decompress the output. By completing decompressing phase, Separator Agent starts to separate original message and digital signature from each other.

After the Separator Agent finished its job, the RSA Agent and SHA-1 Agent will start simultaneously; the SHA-1 Agent will compute message digest of the original message and RSA Agent will decrypt the digital signature using sender's public key. Finally, after SHA-1 and RSA Agents finished their computations, the Comparison Agent compares the two results, if they are equal then the message will be accepted, otherwise the packet is discarded.

4.4 Time analysis of applying Multi-Agent architecture to Receiver side:

The time analysis of new scheme on receiver side is shown in figure 10. As we can see in this figure and also theoretical analyze because on operations' nature the time reduction is not the same on the receiver and sender sides. Time reduction on the receiver side by applying multi agent technique is about 4.7 T; and total time to finish its job is about 5.3 T.

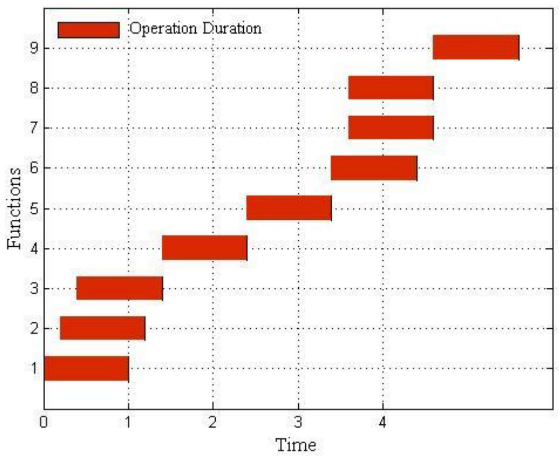

Figure 10. Time analysis of applying Multi-Agent architecture to Receiver side

V. Implementation of the Proposed Scheme

We have implemented the Classic PGP and proposed scheme, both using C language. There are some important issues about Implementation of the Proposed Scheme that should be considered. First, we should solve the mutual exclusion problem, because Agents should work together, wait for other agents' results, inform other agents to start their procedure, and agents should never write shared buffers at the same time. We used one of the operating systems techniques known as semaphores and we assigned a semaphore to each agent. Any agent who has to work with other agents, which they have a shared value in use, should use "Wait" or "Release" command to work with the shared value. For example, in sender side the CASTAgent waits for the results of Session key generator Agent and Compression Agent, CAST Agent uses "Wait" command and waits on the CAST256 semaphore. When the other two agents' results completed, they use "Release" command to release the CAST256 semaphore and as a result,CAST agent finds out these agents had finished their work and it starts its procedure.

In the implementation, we used seven semaphores and assigned one semaphore for each Agent; The total number of the semaphores are fewer than Agents, because we have some agents like Merge Agent and RSA Agent which appeared in different places but just running the same procedure. We considered one extra semaphore for Main agent to handle other agents. The Main Agent is used for managing other agents, initializing start values like sender/receivers' Public keys, reading the original message into the main buffer; and forwarding PGP Packets and etc.

In the proposed scheme, there are a number of buffers, which are used as shared values between agents. We have a main agent buffer, this buffer is filled with the original message, and Main-Semaphore handles accessing to it; the other agents will use these buffer to gain access to the original message. There is 160 bit message digest buffer, which will be used to hold message digest that computed by SHA-1 Agent. SHA-1-Semaphore handles accessing to this buffer. A 640-bits buffer has been used for storing the RSA Agent's results; RSA-Semaphore is responsible for access controlling of this buffer. By releasing or waiting on these semaphores, the agent will be aware of the other agents' conditions, and when the semaphore is released by one agent, the agents who have waited on that semaphore will start their procedure. The complete data flow and semaphores' actions that are shown in the

Figure 7andFigure 9.

To implement different components of the PGP like hashing, compression, etc. we used the standard C libraries that are available for free. For hash function, we choose SHA-1 algorithm. For symmetric key cryptography, CAST-256 algorithm has been selected. RSA algorithm has been selected as Public key cryptography. For ASCII format encoding and decoding, we chose the Radix64 standard algorithm. Lempel-Ziv compression function selected for compression.

VI. Experimental evaluation

We have implemented the proposed scheme on the Intel's multi core CPU and AMD phoneme. C programming language is used for implementation. After implementing schemes, we ran different tests on both schemes and studied the results. For testing, we used packets with different size, the packets size are 20 KB, 40 KB, 80 KB, 120 KB, 160 KB, and 200 KB. Figure 11and Figure 12 show the test result on different samples.

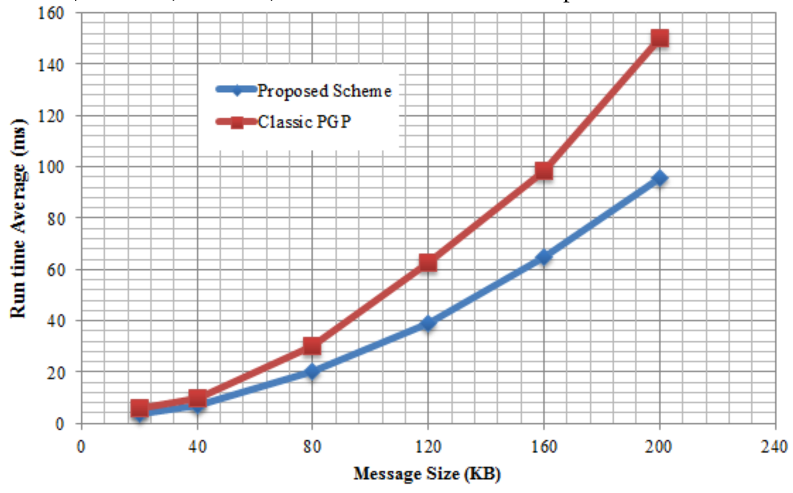

Figure 11. Sender Side's result

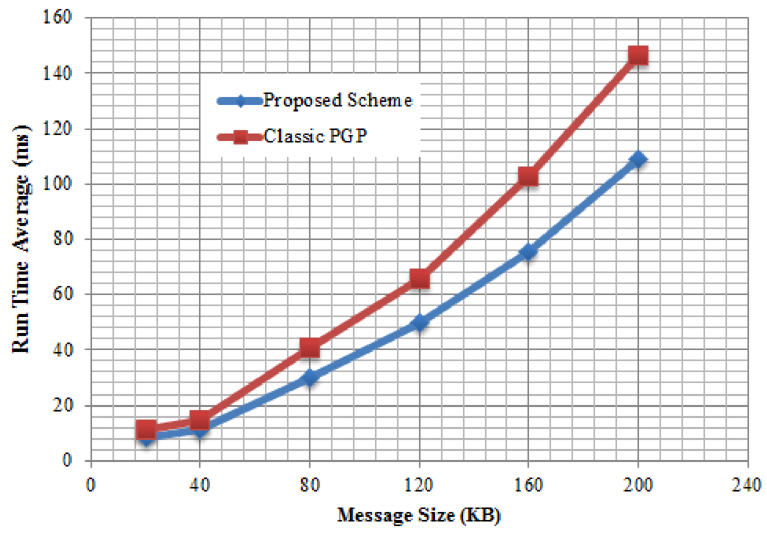

Figure 12. Receiver Side's result

The results show that the proposed scheme reduces execution time at sender side about 35%, and 26% in receiver side. The total, the time reduction is about 30.5%.

The experiments' results show that the time reduction does not match with theoretical predictions, because we haven't take into account the overhead of handling agents. This overhead decreases the proposed scheme's efficiency. The results show when we are using small message, there is a little difference between the two schemes because the execution time is very small, so the reduced time can be neglected. The sender side results show the proposed scheme has more linear behavior than Classic PGP, because we have good concurrency between Agents and as a result the time reduction of the proposed scheme is about 35%. Time saving at receiver side of the proposed scheme, has similar behavior more like to Classic PGP because we have a little concurrency in this form and as a result the time reduction of the receiver side is about 26%.

The major benefit of the proposed scheme is in using large messages or files. By using large messages the agent's handling overhead can be neglect compared to the time reduction. The time reduction is very considerable that is about 36 ms in the 200 KB message size and using larger message the scheme will save more time.

Based on the results, we can reach time reduction around 30.5%, and this time saving can have a major benefit on Mail servers that sends or receives many messages and because of the security services, Mail servers should do a lot of encryption/decryption on messages. The main goal of the experiment is to demonstrate the performance of our new approach. It is also highly desirable to investigate the performance of the proposed scheme when we have predefined compression Code Book and multiple Merge Agents for handling merge procedure. This also can be considered for our future research.

VII. Conclusion

PGP is a very popular security package for email communication and other file transfer protection. The PGP architecture has a hierarchical style and this style is not changed from the start. In this paper, we proposed a new architecture for PGP using Multi Agent systems. We selected the actions that can be executed concurrently with other actions. Each agent handles an independent action. For managing Agents, we consider a main agent that handles the concurrency and job flow between different agents.

We have analyzed the proposed scheme and our analysis shows that the proposed scheme can reduce the execution time about 50%. For demonstrating the real performance of the proposed scheme, we have implemented it and the Classic PGP. The results show that total time saving in both sender and receiver side is about 30%. The agent handling overheads reduce the performance of the proposed scheme. For the future work, we will examine caching techniques and predefined compression Code Book to increase performance of the proposed scheme.

References

[1] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP message format", RFC 2440, November1998.
[2] OpenPGP, RFC4880, http://tools.ietf.org/html/rfc4880
[3] PCWorld. "PGP Encryption Proves Powerful". 2003-05-26. Retrieved 2010-02-08.
[4] R. K. Nichols, ICSA guide to cryptography: McGraw-Hill Professional, 1998.
[5] P. R. Zimmermann, "An introduction to cryptography", Network Associates Inc., PGP, version, vol. 6, 1995.
[6] W. Stallings, Network security essentials vol. 4: Prentice Hall, 2010.