CS565 Computer Security Project 3

E-mail Forensics with DKIM

Team members:

Sumedha Salil Nashte (502085998)
Vipin Kumar (50208397)
Surabhi Singh Ludu (50207139)
Gideon Paul Arugollu (50207129)

Phase 1

Section 1.1: Full header of both e-mails received in your UB mailbox

Header for email 1: (sent from surabhisinghludu@gmail.com to sludu@buffalo.edu)

```
Delivered-To: sludu@g-mail.buffalo.edu
Received: by 10.55.115.131 with SMTP id o125csp958777qkc;
        Thu, 17 Nov 2016 14:33:15 -0800 (PST)
X-Received: by 10.55.93.68 with SMTP id r65mr6143326qkb.84.1479421995870;
        Thu, 17 Nov 2016 14:33:15 -0800 (PST)
Return-Path: <sludu@buffalo.edu>
Received: from localmailf.acsu.buffalo.edu (localmailf.acsu.buffalo.edu. [128.205.4.17])
        by mx.google.com with ESMTP id d93si3264118qkh.92.2016.11.17.14.33.15
        for <sludu@g-mail.buffalo.edu>;
        Thu, 17 Nov 2016 14:33:15 -0800 (PST)
Received-SPF: pass (google.com: best guess record for domain of sludu@buffalo.edu designates 128.205.5.229 as permitted sender)
client-ip=128.205.5.229;
Authentication-Results: mx.google.com;
       spf=pass (google.com: best guess record for domain of sludu@buffalo.edu designates 128.205.5.229 as permitted sender)
smtp.mailfrom=sludu@buffalo.edu
Received: from smtp.buffalo.edu (smtp4.acsu.buffalo.edu [128.205.5.229]) by localmailf.acsu.buffalo.edu (Prefixe) with ESMTP id
875F380086 for <sludu@buffalo.edu>; Thu, 17 Nov 2016 17:33:15 -0500 (EST)
Received: from mail-qk0-f173.google.com (mail-qk0-f173.google.com [209.85.220.173]) (Authenticated sender: sludu@buffalo.edu) by
smtp.buffalo.edu (Postfix) with ESMTPSA id 6F18D1992264 for <sludu@buffalo.edu>; Thu, 17 Nov 2016 17:33:15 -0500 (EST)
Received: by mail-qk0-f173.google.com with SMTP id q130so241514796qke.1
        for <sludu@buffalo.edu>; Thu, 17 Nov 2016 14:33:15 -0800 (PST)
X-Gm-Message-State: AKaTC02EzKhkNU0jhnYE4hIxIaePXdFpDubfKC3QoZgtmq/2CN8Gx4Q4cZ8wsN7UBYDBI/y+ZH8JlVHsd1lwpA==
X-Received: by 10.55.162.83 with SMTP id l80mr6463331qke.168.1479421994928; Thu, 17 Nov 2016 14:33:14 -0800 (PST)
MIME-Version: 1.0
Received: by 10.237.39.7 with HTTP; Thu, 17 Nov 2016 14:33:14 -0800 (PST)
From: Surabhi Singh Ludu <sludu@buffalo.edu>
Date: Thu, 17 Nov 2016 17:33:14 -0500
X-Gmail-Original-Message-ID: <CAExHD2PWbVu+c8EZGovVhUqJMHjmTD6bxVFq_A0q5Y3Dj=2URg@mail.gmail.com>
Message-ID: <CAExHD2PWbVu+c8EZGovVhUqJMHjmTD6bxVFq_A0q5Y3Dj=2URg@mail.gmail.com>
Subject: Email 1 for project 3 of Computer Security
To: sludu@buffalo.edu
Content-Type: multipart/alternative; boundary=001a114faecc7b1970054186c6d9
```

Fig 1. Header for the mail sent from Gmail

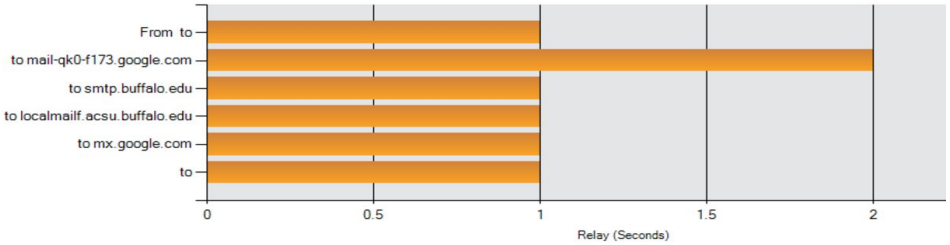

Fig 2. Graph showing time taken for all message hops

Header for email 2: (sent from sludu@buffalo.edu to sludu@buffalo.edu)

```
MIME-Version: 1.0
Received: by 10.55.162.212 with HTTP; Thu, 17 Nov 2016 14:34:38 -0800 (PST)
Date: Thu, 17 Nov 2016 17:34:38 -0500
Delivered-To: sludu@buffalo.edu
Message-ID: <CANPRFoiQMtxKg+n2zoVQYjermuiq_Ezkofie0bEoG5_9J3vzww@mail.gmail.com>
Subject: Email 2 for project 3 of Computer Security
From: Surabhi Singh Ludu <sludu@buffalo.edu>
To: sludu@buffalo.edu
Content-Type: multipart/alternative; boundary=001a113f453070aaec054186cb27

--001a113f453070aaec054186cb27
Content-Type: text/plain; charset=UTF-8
```

Fig 3. Header for the mail sent from buffalo ID

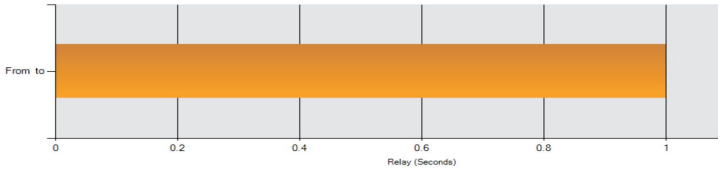

Fig 4. Graph showing time taken for all the hops

➢ **Delivered-To:** This field gives the email ID of the user to which the final email was delivered. The value of this field was sludu@g-mail.buffalo.edu for the mail sent from Gmail while for the one sent from UB account it was sludu@buffalo.edu.

➢ **Received:** These fields define all the intermediate nodes through which the email went before reaching the receiver. For the mail sent by Gmail, we have more intermediate nodes while for UB account mail we have only one intermediate node.

➢ **Return-Path:** This field defines the return path for this email i.e the id that reply would be sent to.

➢ **Authentication-Results:** It contains values for the authentication results for the sender's server. It may include SPF or DKIM results. It's status is pss for the mail sent by Gmail.

➢ **MIME-Version:** This field gives the MIME version being used for this particular email.

➢ **From:** This field defines the name and email ID of the sender for the email.

➢ **Date:** This field is the timestamp for the email, telling when the email was sent, including its date and time.

➢ **Message-ID:** This is unique field, an ID that is assigned to every email that is ever sent on the internet. It follows a specific set of rules based on the timestamp and the sender's domain.

➢ **Subject:** It defines the subject of the email being sent. Given by the sender of the email.

➢ **To:** It contains the intended recipient of the email. This may be different from the Delivered-To field as it is in the mail sent from Gmail.

➢ **Content-Type:** This field defines the type of content that this email contains. It makes it easy for the end user's system to read the email and present it to the receiver.

We can see from comparison that the email sent from Gmail took more hops to reach the destination and took more time as sending date is 17 Nov 2016 17:33:14 and received 1 second later at Nov 2016 14:33:15(PST) which is equivalent to Nov 2016 17:33:15.
While mail sent from buffalo account to buffalo mail id was received in the same second my perhaps some microsecond difference as sending time is 17 Nov 2016 17:34:38 and receiving time is 17 Nov 2016 17:34:38.

Section 1.2: Unique identifiers of these messages contained on the header

Unique identifiers in the headers above are message-id. Message-IDs are required to have a specific format which is a subset of an email address and to be globally unique. That is, no two different messages must ever have the same Message-ID. A common technique used by many message systems is to use a time and date stamp along with the local host's domain name
These are supposed to be unique but can easily be forged. It is quite easy to spoof this message ID because the message ID has certain set of rules that it follows, but by spoofing just this message ID the email can not be simply classified as spoofed because other fields may not be easy to spoof. There also may be SPF or DKIM authentication sent along with the mail, which would make spoofing harder.

Section 1.3: Diagrammatical representation the network path traversed by both the e-mails

Mail 1 (sent from Gmail ID) :

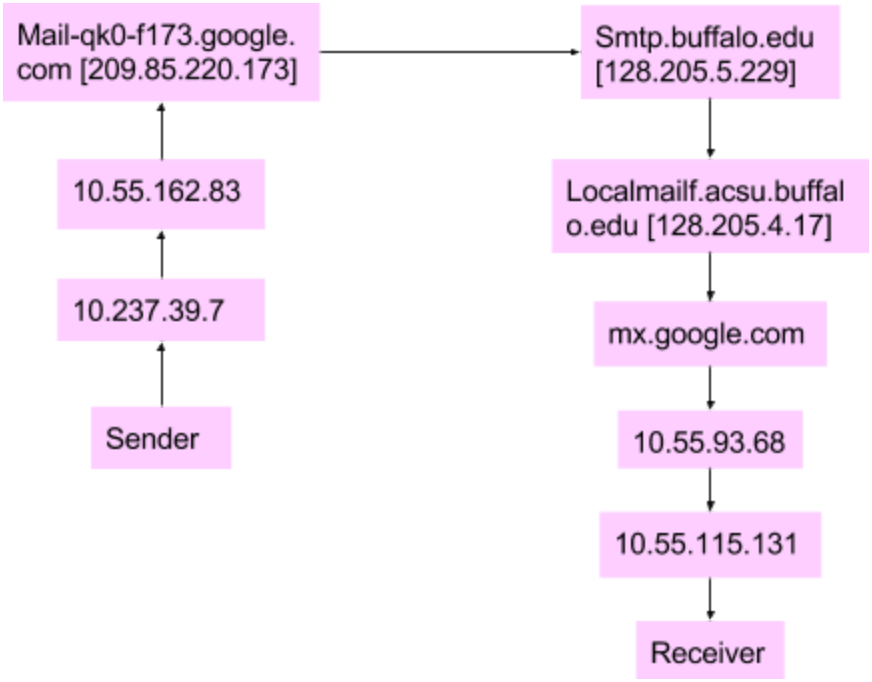

Fig 5. A directed graph representing the network path for mail sent from Gmail

Mail 2 (sent from UB account) :

Fig 6. A directed graph representing the network path for mail sent from UB account

Section 1.4: Can we characterize one of these e-mails as a spoofed e-mail?

Neither of the emails sent can be classified as email spoofs.

What we did in this section for mail 1 is authorize another email client i.e. surabhisinghludu@gmail.com to send emails as sludu@buffalo.edu. When we set this up as alias, we receive an authentication code mail, which in turn authenticates that the user trying to add a certain mail as alias in another mail owns both mail ids.

Fig 5. Verification page for setting up an alias with gmail account

Hence, this cannot be considered as email spoofing. Rather this is a case of email aliasing.
Moreover the Sender Policy Framework(SPF) for this email is pass, so we can somewhat trust the sender.
For email 2 we sent a mail to our own email address that is from sludu@buffalo.edu to sludu@buffalo.edu, hence even this could not be considered as email spoofing. Though this mail did not contain any authentication test but that may be due to the fact that mail was sent from @buffalo.edu domain to @buffalo.edu domain. This can be exploited by an adversary, posing as a user of the @buffalo.edu email and then send an email to other users on the same server. This email might be spoofed because of the absence of any authentication data.

Section 1.5: A brief description of how we can deduce a conclusion on a suspected e-mail spoofing case using the header information

We can deduce a conclusion on a suspected e-mail spoofing case using the header information by analyzing the received email headers. We can trace the hops and come to the IP/server name of the original sender by analyzing received field of the header.
We can also check the reply-to field to check which email id the reply to that email will be routed to. If this email id seems suspicious with spelling mistakes or unknown domain name, we can't trust this sender.
We can also use either the SPF(Sender Policy Framework) test or the DKIM(Domain Key Identified Mail) test to see if the email is spoofed or not.

Phase 2

Section 2.1: A brief description of how DKIM works. What are the components?

DomainKeys Identified Mail (DKIM) defines a mechanism by which email messages can be cryptographically signed, permitting a signing domain to claim responsibility for the introduction of a message into the mail stream i.e. it allows sender to attach a domain with an email message. It provides email users with an additional level of protection against email forgery. DKIM places a signature on the email header, this includes 3 related fields:

- A digital signature
- A definition of the fields over which the digital signature was calculated
- The sending domain

DKIM publishes the public key and policies of the sending organization to the Domain Name System (DNS). The receiving organization verifies the DKIM signature by comparing it with the sender's public key made available through DNS. After a DKIM signature has been placed on a message, an agent in the ADMD (Administrative Management Domain), which is a public email service, will validate the signature. Any functional component in the message transit path can validate the signature. The recipient end-user does not have to make any validations, instead the recipients ADMD does the validations.
Functional Components of DKIM:

- **Signing:** Signing will be done by a service agent within the authority of the message originator's Administrative Management Domain (ADMD). Signing might be performed by any of the functional components, in that environment, including: Mail User Agent (MUA), or Mail Submission Agent (MSA), Internet Boundary MTA. DKIM permits signing to be performed by authorized third-parties.
- **MUA (Message User Agent):** Message is formatted and submitted to MHS via MSA. It processes the message for storage or delivery to the receiver.
- **MSA (Message Submission Agent):** It performs signing before enforcing the message into Internet standards and policies of hosting domain before relaying to MTA.

- **MTA (Message Transfer Agent):** This deals with the relaying message from MSA to MDA through one or more MTAs.
- **Verifying:** Verifying is performed by an authorized module within the verifying ADMD. Within a delivering ADMD, verifying might be performed by an MTA, MDA, or MUA.
- **MDA (Message Digest Agent):** This is the verifier of the message. It performs the verification test on the message and if the test is successful, it transfers the message from MHS to MS
- **MHS (Message Handling service):** A network consisting of the various components namely MSA, MDA and MTAs is together called a MHS.

Section 2.2: Graph plots along with the source code written. Attach the source code as an appendix.

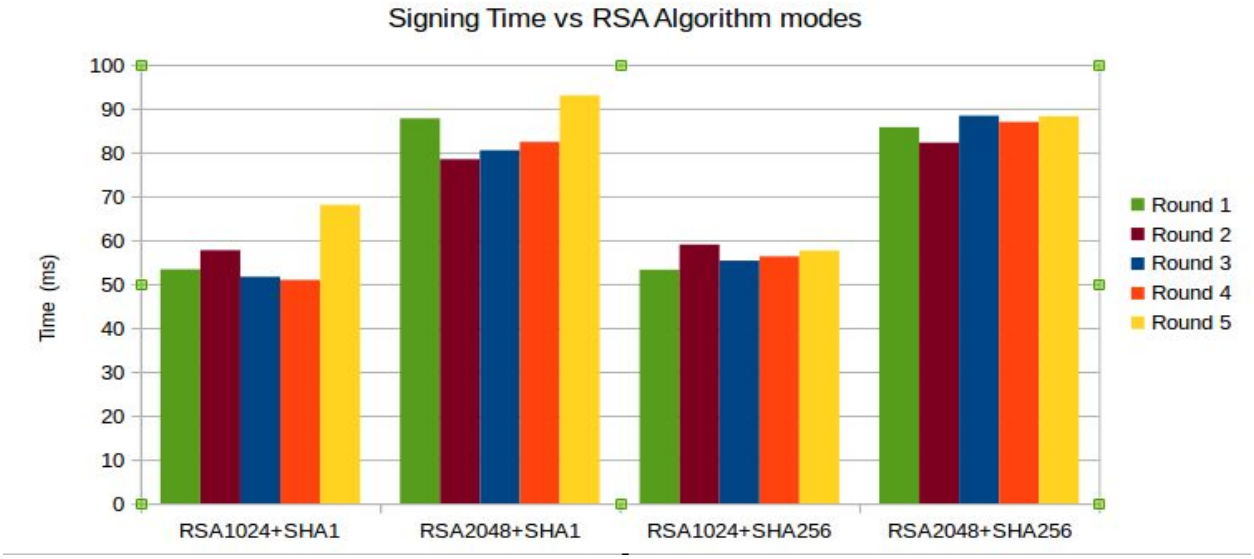

Fig 6. Graph showing the time taken in milliseconds for applying each algorithm(signing) for 5 test runs

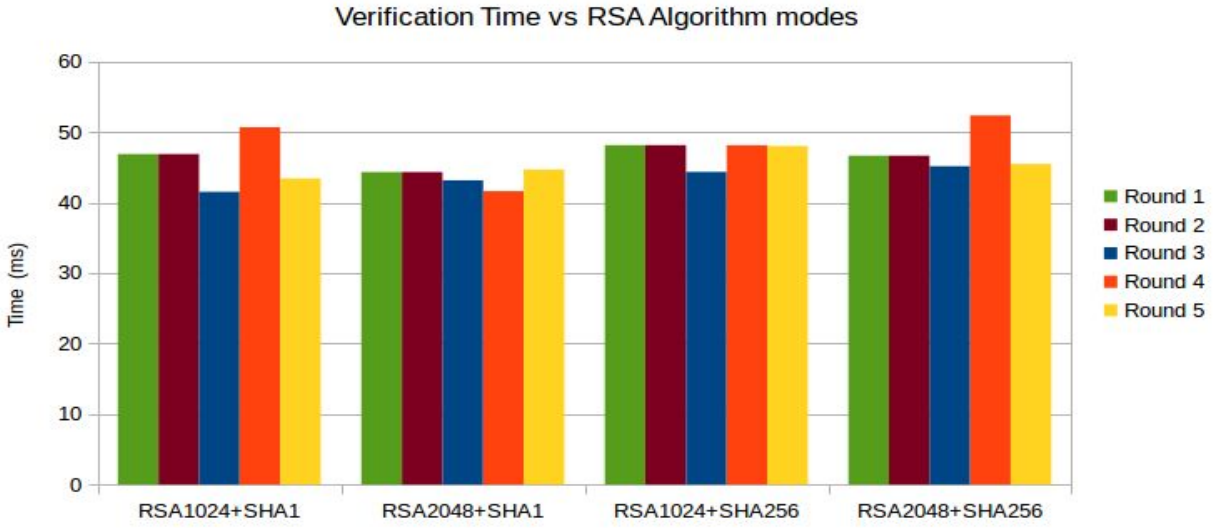

Fig 7. Graph showing the time taken in milliseconds for applying each algorithm(verification) for 5 test runs

Source code attached as an appendix.

Section 2.3: Explain which of the combination of core DKIM algorithms provides the best performance using email message size as the criteria.

Comparing all the given conditions and time taken by the different algorithms RSA1024+SHA1 takes less time in signing and verification of the given data on an average. However, this algorithm is ideal only if the security is not critical.

Section 2.4:

1. Briefly describe the problems with using S/MIME or PGP in emails.

- S/MIME and PGP are intended for longer-term, personal protection and therefore use costlier mechanisms and per-user identification, with the requirement that key and certification information, needed for a given message, be available essentially forever.
- Also they require modification of the message body, to provide their packaging as MIME body-part types.
- S/MIME involves encryption, and when the user encrypts e-mail, it is no longer searchable. Users can no longer retrieve past e-mails based upon message text keyword searches, although the e-mail subject line and some other information, such as file attachment name, may remain visible.
- Outgoing S/MIME-encrypted email can be anti-virus scanned before encrypting and sending, but it's more difficult to scan incoming S/MIME messages, where the scanning is done on a gateway or by an external service provider. Most of the risk from e-mail malware isn't from the data sent, anyway. It's from the data received. If a user uses S/MIME and doesn't have client-side malware detection for e-mail, he/she has a problem.
- Before we can communicate via PGP, we first need to exchange keys. PGP makes this a problem for the users. It takes very long to encrypt/decrypt messages.
- Encryption and decryption time also increases with the key size. A 2048 bits key will take much longer to work with than, for example, a 512 bits key.

2. How is DKIM different from these email signature schemes?

- DKIM is based on domain names, rather than complete email addresses like in S/MIME or PGP. Hence, signing is controlled by the administrator of the domain name, rather than by individual email users.
- DKIM uses DNS-based self-certified keys. Because the scope of DKIM is limited, it does not need generalized, powerful and long-term certificates, issued by separate authorities.
- Unlike S/MIME and PGP, DKIM does not modify the body. Rather it places its parametric information into header fields that are typically not shown to the recipient. Therefore DKIMs can be entirely invisible to recipients.
- Using domain names, rather than email addresses, provides advantage.
- Unlike PGP and S/MIME, the DKIM doesn't provides neither non-repudiation nor confidentiality.

3. What are other broad categories of Domain Validations used? What does DKIM fall under?

There are two broad categories of Domain Validations used:

- Based on IP addresses.
- Bases on digital signatures. DKIM falls under this category.

4. Briefly explain what does DKIM do for the signer and for the receiver?

The responsible organization adds a digital signature to the message, associating it with a domain name of that organization. Typically, signing will be done by a service agent within the authority of the message originator's Administrative Management Domain (ADMD). Signing might be performed by any of the functional components, in that environment, including: Mail User Agent (MUA), or Mail Submission Agent (MSA), Internet Boundary MTA. DKIM permits signing to be performed by authorized third-parties.

After a message has been signed, any agent in the message transit path can choose to validate the signature. Typically, validation will be done by an agent in the ADMD of the message recipient. Again, this may be done by any functional component within that environment. Notably this means that the signature can be used by the recipient ADMD's filtering software, rather than requiring the recipient end-user to make an assessment.

5. Does DKIM signature signify that all the fields in the header information are not forged?

No. In the case of messages signed using DKIM signatures or other message signing methods that sign headers, this may invalidate one or more signature on the message if they included the header field to be removed at the time of signing. However, signing agents may therefore elect to omit these header fields from signing to avoid this situation. DKIM allows the sender to choose to sign some or all of the message header fields. Since many of the fields do not contain significant information, signers may choose not to sign them. Headers fields are most vulnerable to change in transit, so leaving insignificant fields unsigned will increase the chance of successful verification. However, these fields could be tampered in transit. Hence, a DKIM signature does not signify that all fields in the header are not forged.

REFERENCES:

[1] http://mxtoolbox.com/Public/Tools/EmailHeaders.aspx
[2] https://www.spamgrate.com/
[3] http://www.webopedia.com/TERM/E/e_mail_aliasing.html
[4] http://www.dkim.org/specs/draft-kucherawy-sender-auth-header-14.html
[5] http://en.wikipedia.org/wiki/DomainKeys_Identified_Mail
[6] http://www.dkim.org/info/dkim-faq.html
[7] http://www.emailonacid.com/blog/details/C13/what_is_dkim_everything_you_need_to_know_about_digital_signature
[8] http://en.wikipedia.org/wiki/S/MIME
[9] http://secushare.org/PGP

APPENDIX

Source Code :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/time.h>
#include<sys/types.h>
//signing the email with RSA and SHA
double signMessage(char **hashName, char **message, char *rsaKey, char *shaType){
	double time = 0;
	int i = 0;
	int pid;
	struct timeval start,end;
	gettimeofday(&start,NULL);
	while(i<10){
	pid = fork();
	if(!pid){

execl("/usr/bin/openssl","openssl","dgst",shaType,"-sign",rsaKey,"-out",hashName[i],message[i],NULL);
	}else{
	waitpid(pid,NULL, 0);
	}
	i++;
	}
	gettimeofday(&end,NULL);
	time
=(((double)end.tv_sec-(double)start.tv_sec)*1000)+(((double)end.tv_usec-(double)start.tv_usec)/((double)10
00));
return time;
}

//Verify the signed email with RSA and SHA
double verifyMessage(char **hashName, char **message, char *rsaKey, char *shaType){
	double time = 0;
	int i = 0;
	int pid;
	struct timeval start,end;
	gettimeofday(&start,NULL);
	while(i<10){
	pid = fork();
	if(!pid){

execl("/usr/bin/openssl","openssl","dgst",shaType,"-verify",rsaKey,"-signature",hashName[i],message[i],NUL
L);
```

```
        }else{
        waitpid(pid,NULL, 0);
        }
        i++;
        }
        gettimeofday(&end,NULL);
        time
=(((double)end.tv_sec-(double)start.tv_sec)*1000)+(((double)end.tv_usec-(double)start.tv_usec)/((double)10
00));
return time;
}

void main(){
   int i =0;
   double timeTaken = 0;
   char *inputMsg[10];
   char *hash1024SHA1[10];
   char *hash1024SHA256[10];
   char *hash2048SHA1[10];
   char *hash2048SHA256[10];

   for(i=0;i<10;i++){
        inputMsg[i]= (char*)malloc(30);
        hash1024SHA1[i]= (char*)malloc(30);
        hash1024SHA256[i]= (char*)malloc(30);
        hash2048SHA1[i]= (char*)malloc(30);
        hash2048SHA256[i]= (char*)malloc(30);
        sprintf(inputMsg[i],"mails/mail %d.msg",i+1);
        sprintf(hash1024SHA1[i],"encrypted/1024SHA1/cipher.%d",i+1);
        sprintf(hash1024SHA256[i],"encrypted/1024SHA256/cipher.%d",i+1);
        sprintf(hash2048SHA1[i],"encrypted/2048SHA1/cipher.%d",i+1);
        sprintf(hash2048SHA256[i],"encrypted/2048SHA256/cipher.%d",i+1);
   }


   for (i=1;i<=5;i++){

        // sha1 + RSA 1024 signing
        timeTaken = 0;
        timeTaken = signMessage(hash1024SHA1,inputMsg,"rsaprivatekey1024.pem","-sha1");
        printf("Time Taken for round %d of 1024+SHA1 signing is %lf \n",i,timeTaken);

        // sha1 + RSA 2048 signing
```

```
        timeTaken = 0;
        timeTaken = signMessage(hash2048SHA1,inputMsg,"rsaprivatekey2048.pem","-sha1");
        printf("Time Taken for round %d of 2048+SHA1 signing is %lf \n",i,timeTaken);

        // sha256 + RSA 1024 signing
        timeTaken = 0;
        timeTaken = signMessage(hash1024SHA256,inputMsg,"rsaprivatekey1024.pem","-sha256");
        printf("Time Taken for round %d of 1024+SHA256 signing is %lf \n",i,timeTaken);

        // sha256 + RSA 2048 signing
        timeTaken = 0;
        timeTaken = signMessage(hash2048SHA256,inputMsg,"rsaprivatekey2048.pem","-sha256");
        printf("Time Taken for round %d of 2048+SHA256 signing is %lf \n\n",i,timeTaken);
    }


    for(i=1;i<=5;i++){

        // sha1 + RSA 1024 verification
        timeTaken = 0;
        timeTaken = verifyMessage(hash1024SHA1,inputMsg,"rsapublickey1024.pem","-sha1");
        printf("Time Taken for round %d of 1024+SHA1 verification is %lf \n",i,timeTaken);

        // sha1 + RSA 2048 verification
        timeTaken = 0;
        timeTaken = verifyMessage(hash2048SHA1,inputMsg,"rsapublickey2048.pem","-sha1");
        printf("Time Taken for round %d of 2048+SHA1 verification is %lf \n",i,timeTaken);

        // sha256 + RSA 1024 verification
        timeTaken = 0;
        timeTaken = verifyMessage(hash1024SHA256,inputMsg,"rsapublickey1024.pem","-sha256");
        printf("Time Taken for round %d of 1024+SHA256 verification is %lf \n",i,timeTaken);

        // sha256 + RSA 2048 verification
        timeTaken = 0;
        timeTaken = verifyMessage(hash2048SHA256,inputMsg,"rsapublickey2048.pem","-sha256");
        printf("Time Taken for round %d of 2048+SHA256 verification is %lf \n",i,timeTaken);
    }

}
```