E-mail Forensics with DKIM

CSE 565 Fall 2015

Computer Security

Project 3

Computer Science and Engineering
University at Buffalo
Buffalo, NY-14214

Aniruddh Adkar
UB Person number: 50169486
aadkar@buffalo.edu

Aniket Thigale
UB Person number: 50168090
athigale@buffalo.edu

Saumitra Vaidya
UB Person number: 50168093
svaidya@buffalo.edu

Tushar Kulkarni
UB Person number: 50168831
tkulkarn@buffalo.edu

Pranav Jain
UB Person number: 50169659
pranavja@buffalo.edu

Phase 1: Investigate E-Mail spoofing

Section 1.1: Explain the full header of both e-mails received in your UB mailbox. Try to give as much details as possible. Try to describe it using the timestamps provided on the header.

The E-Mail header of the mail sent from Gmail account to UB account:

Delivered-To: aadkar@g-mail.buffalo.edu
Received: by 10.194.120.195 with SMTP id le3csp1647229wjb;
Wed, 18 Nov 2015 18:51:39 -0800 (PST)
X-Received: by 10.140.109.54 with SMTP id k51mr4982126qgf.83.1447901499369;
Wed, 18 Nov 2015 18:51:39 -0800 (PST)
Return-Path: <aadkar@buffalo.edu>
Received: from localmailg.acsu.buffalo.edu (localmailg.acsu.buffalo.edu. [128.205.4.25])
by mx.google.com with ESMTP id u133si5095958qka.104.2015.11.18.18.51.39
for <aadkar@g-mail.buffalo.edu>;
Wed, 18 Nov 2015 18:51:39 -0800 (PST)
Received-SPF: pass (google.com: best guess record for domain of aadkar@buffalo.edu designates 128.205.5.226 as permitted sender) client-ip=128.205.5.226;
Authentication-Results: mx.google.com;
spf=pass (google.com: best guess record for domain of aadkar@buffalo.edu designates 128.205.5.226 as permitted sender) smtp.mailfrom=aadkar@buffalo.edu
Received: from smtp.buffalo.edu (smtp3.acsu.buffalo.edu [128.205.5.226])
by localmailg.acsu.buffalo.edu (Prefixe) with ESMTP id 21498E09A5
for <aadkar@buffalo.edu>; Wed, 18 Nov 2015 21:51:39 -0500 (EST)
Received: from mail-lf0-f51.google.com (mail-lf0-f51.google.com [209.85.215.51])
(Authenticated sender: aadkar@buffalo.edu)
by smtp.buffalo.edu (Postfix) with ESMTPSA id C78C48F99B7
for <aadkar@buffalo.edu>; Wed, 18 Nov 2015 21:51:38 -0500 (EST)
Received: by lfdo63 with SMTP id o63so39277652lfd.2
for <aadkar@buffalo.edu>; Wed, 18 Nov 2015 18:51:37 -0800 (PST)
MIME-Version: 1.0
X-Received: by 10.25.163.131 with SMTP id m125mr2272096lfe.0.1447901497453;
Wed, 18 Nov 2015 18:51:37 -0800 (PST)
Received: by 10.25.73.195 with HTTP; Wed, 18 Nov 2015 18:51:37 -0800 (PST)
Date: Wed, 18 Nov 2015 21:51:37 -0500
X-Gmail-Original-Message-ID: <CALL2GvjbjygMYz-eGNN_=9_6sM-5hLkTifZTN7cf8wYxGyFmsg@mail.gmail.com>
Message-ID: <CALL2GvjbjygMYz-eGNN_=9_6sM-5hLkTifZTN7cf8wYxGyFmsg@mail.gmail.com>
Subject: Test email
From: Aniruddh A <aadkar@buffalo.edu>
To: aadkar@buffalo.edu
Content-Type: multipart/alternative; boundary=001a114108346cf8010524dbd65a

--001a114108346cf8010524dbd65a
Content-Type: text/plain; charset=UTF-8
Hi

Test email sent from Gmail account aadkarub@gmail.com to aadkar@buffalo.edu
using email id as aadkar@buffalo.edu.
Aniruddh Adkar
--001a114108346cf8010524dbd65a
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr"> Hi

 Test email sent from Gmail accou=
nt aadkarub@gmail.com to <a href=
=3D"mailto:aadkar@buffalo.edu">aadkar@buffalo.edu using email id as <a =
href=3D"mailto:aadkar@buffalo.edu">aadkar@buffalo.edu.

</div=
>Aniruddh Adkar

--001a114108346cf8010524dbd65a—

The E-Mail header of the mail sent from UB account to UB account:

MIME-Version: 1.0
Received: by 10.194.221.34 with HTTP; Wed, 18 Nov 2015 18:53:47 -0800 (PST)
Date: Wed, 18 Nov 2015 21:53:47 -0500
Delivered-To: aadkar@buffalo.edu
Message-ID: <CAFFcRSkf01Uq9cLYQLYnL4Okhnms47DCpRi-UFd-Mj-yC6W5Gg@mail.gmail.com>
Subject: Test email 2
From: Aniruddh Ramesh Adkar <aadkar@buffalo.edu>
To: Aniruddh Ramesh Adkar <aadkar@buffalo.edu>
Content-Type: multipart/alternative; boundary=e89a8fb1fd4033e9c20524dbde96

--e89a8fb1fd4033e9c20524dbde96
Content-Type: text/plain; charset=UTF-8
Hi
Test email sent from UB account aadkar@buffalo.edu to aadkar@buffalo.edu
using email id as aadkar@buffalo.edu.
--
Aniruddh Adkar

--e89a8fb1fd4033e9c20524dbde96
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr"> Hi

 Test email sent from UB account <a hr=
ef=3D"mailto:aadkar@buffalo.edu">aadkar@buffalo.edu to <a href=3D"mailt=
o:aadkar@buffalo.edu">aadkar@buffalo.edu using email id as <a href=3D"m=
ailto:aadkar@buffalo.edu">aadkar@buffalo.edu.<br clear=3D"all"> <di=
v>
--
<div class=3D"gmail_signature"><div dir=3D"ltr"> <div=
 dir=3D"ltr">Aniruddh Adkar

--e89a8fb1fd4033e9c20524dbde96—

Received: To detect the path by which emails were sent, we need to check Received header in the bottom up fashion. This field includes the sender's IP address, optional details on Transport Layer Security, the server that processed the message and a timestamp. It is the most important and reliable field of an E-Mail header. It gives the list of servers through which the message travelled from the sender to the receiver. Timestamp helps in tracking the flow of message from the sender to the receiver. The data header on a message will be the local time where the message was sent. In the Received field, the timestamp is local to the computer that handles it. The following key is useful in interpreting the timestamps: PST: Pacific Standard Time, EDT: Eastern Daylight Time, GMT: Greenwich Mean Time.

The path of the message from the Gmail account to the UB account is as follows:

- The Web server 10.25.73.195 receives the mail through Web (HTTP) protocol at time Wed, 18 Nov 2015 18:51:37 -0800 (PST).
- The google server 10.25.163.131 receives the message through SMTP protocol with timestamp Wed, 18 Nov 2015 18:51:37 -0800 (PST) and SMTP ID m125mr2272096lfe.0.1447901497453.
- The google server lfdo63 receives the message through SMTP protocol with timestamp Wed, 18 Nov 2015 18:51:37 -0800 (PST) and SMTP id o63so39277652lfd.2.
- UB's SMTP server, smtp.buffalo.edu receives the message from google server mail-lf0-f51.google.com with timestamp Wed, 18 Nov 2015 21:51:38 -0500 (EST) and with ESMTPSA id C78C48F99B7. A delay of 1 second is introduced at this point.
- The message goes to localmailD.acsu.buffalo.edu from smtp3.acsu.buffalo.edu through ESMTP protocol with timestamp Wed, 18 Nov 2015 21:51:39 -0500 (EST) and ESMTP id 21498E09A5.
- The message is directed to mx.google.com using ESMTP protocol with timestamp Wed, 18 Nov 2015 18:51:39 -0800 (PST) and with ESMTP id u133si5095958qka.104.2015.11.18.18.51.39. A delay of 1 second is found.
- The message is directed to the server 10.140.109.54 with timestamp Wed, 18 Nov 2015 18:51:39 -0800 (PST) and with SMTP id k51mr4982126qgf.83.1447901499369.
- The message is directed to the server 10.194.120.195 with timestamp Wed, 18 Nov 2015 18:51:39 -0800 (PST) and with SMTP id le3csp1647229wjb.

Authentication-Results: This is a trace header where records of email authentication are stored by the receiver. Mx.google.com is the authentication server's ID. Spf field corresponds to the Sender Policy Framework and checks the domain of the mail initiator in the list of authorized domains published in DNS records. Spf=pass indicates that domain name is authenticated by the mx.google.com's mail exchange server.

Received-SPF: MTA adds the above advisory header to message so that SPF results can be verified. These headers must be added above "Received" header for the given MTA so that it becomes clear that trusted MTA added header to the message.

Return-Path: Email address for the return mail. Same as reply-to.

Content-Type: It describes the data in the body such that the receiving user agent can handle the data in the right way. The Content-type header includes *type* and *sub-type* fields to indicate the actual nature of the data. The sub-type field specifies the data format. Content-type: Plain text indicates plain-text with

UTF-8 encoding character set. Content-type: multipart/alternative indicates that the body has multiple parts along with boundary to identify different parts.

To: It contains the recipient's name and address

From: It contains the sender's name and address.

Subject: It contains the actual mail content which the sender would like the receiver to know.

Message-ID and X-Gmail-Original-Message-ID: It is the unique identifier of an E-Mail message. It includes time and date stamp along with the local host's domain name. No two messages can have the same Message-ID. In email 1, message ID : CALL2GvjbjygMYz-eGNN_=9_6sM-5hLkTifZTN7cf8wYxGyFmsg@mail.gmail.com.

Delivered-To: It contains the E-Mail address of the receiver to whom the E-Mail is being addressed.

Date: It gives the date and time when the E-Mail message was created or composed.

MIME-Version: 1.0 – version of MIME supported.

Section 1.2: What are the unique identifiers of these messages contained on the header? Are these fields spoof-vulnerable? Justify your answer.

Message-ID and X-Gmail-Original-Message-ID are the unique identifiers of these messages. Every message is uniquely identified by the 'Message ID' inserted into the header. This is done by the first MTA (Mail Transfer Agent) or MUA (Mail User Agent). It is not a compulsory field but it is recommended by RFC 2822 to uniquely identify the sender of the mail. The message IDs are split into parts, the one preceding @ and the part following @. The part preceding @ contains date, time, process id and a few random numbers. The other part contains the domain name. The message IDs for the 2 messages are given below:

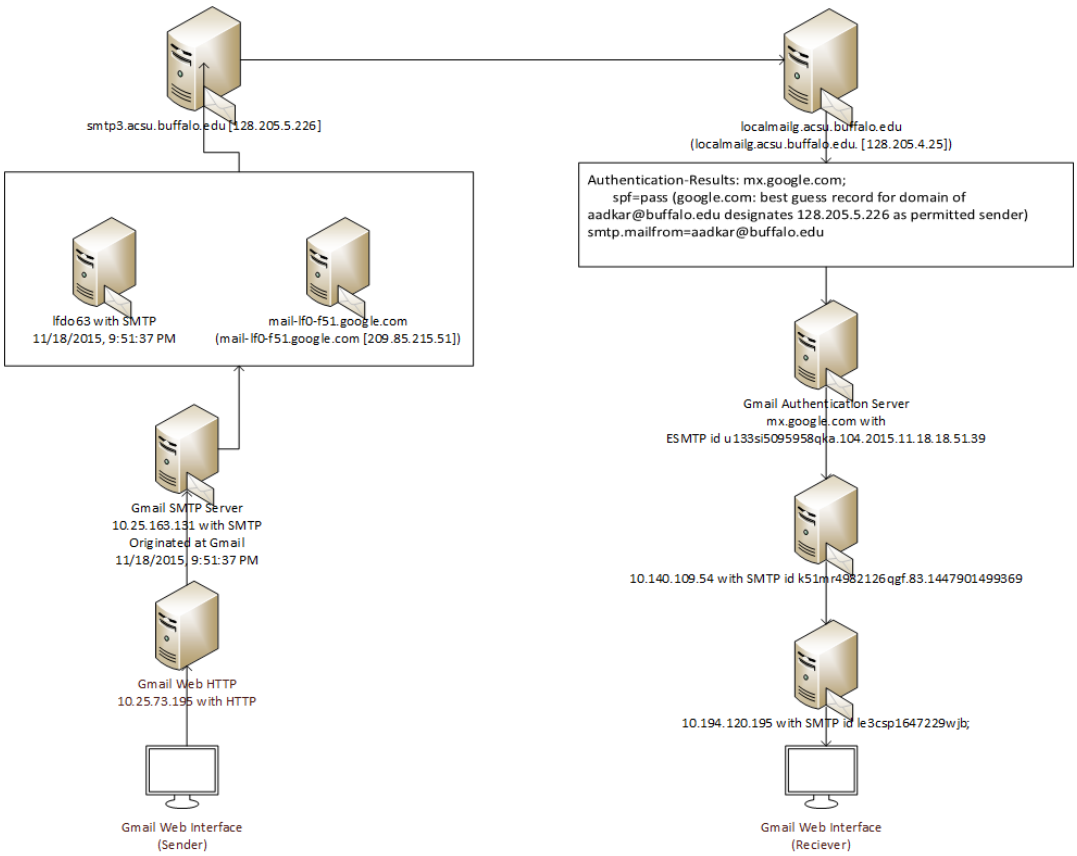
G-Mail to UB Account: CALL2GvjbjygMYz-eGNN_=9_6sM-5hLkTifZTN7cf8wYxGyFmsg@mail.gmail.com

UB to UB Account: CAFFcRSkf01Uq9cLYQLYnL4Okhnms47DCpRi-UFd-Mj-yC6W5Gg@mail.gmail.com

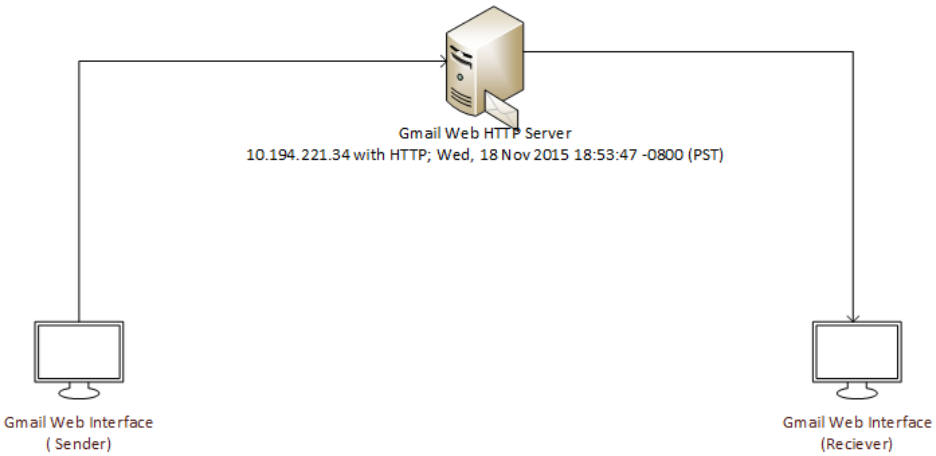
Yes, the Message ID is spoof – vulnerable. But one needs special technical knowledge to spoof the message ID. If the attacker has knowledge of how message IDs are created, he/she can replicate the steps and create a spoofed message using this new message ID. It can be done by observing a few email headers where the MTA is sendmail, making it possible to make a message ID that looks legitimate. Spoofed messages can be identified using forensic analysis of the message ID. Knowledge of message ID construction comes into play here.

Section 1.3: Show diagrammatically the network path traversed by both the e-mails. Use IP address as well as server name to identify intermediate nodes. Are the paths same or different? Justify your answer.

Network Diagram for the flow of message from Gmail to UB account:



Network diagram for the flow of message from UB account to UB account:



The paths traversed by both the emails are different because while sending a message from Gmail account to UB account, the From field contains the Gmail ID. The message hence has to move from the path shown in the first image above, where it moves from Gmail web interface to HTTP client to google servers and then UB SMTP server and so on. Form UB server, it is then routed to the UB mail ID of the recipient. On the other hand, when we send a message from UB account to UB account, since the message does not have to pass through any other server, it is directly routed from UB mail server back to the UB user ID that belongs to the same UB server.

Section 1.4: Can we characterize one of these e-mails as a spoofed e-mail? Why or Why not? Justify the cases of both e-mails.

None of these e-mails can be characterized as spoofed e-mails.

In case 1, Gmail authenticates the UB email ID before adding it to an existing account. ESMTP will not forward any emails originating from a blacklisted domain or if the mail seems to be forged. Since these emails were not tampered with or replicated when they were travelling from MTA/MUA, they can be tagged as original ones. Spoofing can be detected if the source IP address remains the same with change in authenticated domain names. Spoofing can also be detected using inconsistencies in timestamp and message ID.

In case 2, the message is travelling inside the same server after being sent from UB email ID, from where it is redirected back to the UB mail. Since it does not travel through any other network entity, it is not forged or spoofed.

Section 1.5: Using your knowledge from Section 1.1 through Section 1.4, describe briefly how you can deduce a conclusion on a suspected e-mail spoofing case using the header information.

There are a number of factors that can help in determining if an email is spoofed:

1. If the difference in sent and received timestamps is large, the possibility of the email being spoofed increases.

2. An unusually large number of hops between the sender and the receiver mean the email could be spoofed.

3. If the message is relayed via an unusual and random location, it could mean that the message was spoofed/modified there and then sent.

4. If the message header shows the use of some script to make the message instead of an email client application.

5. If the email travels through some server other than the gmail server.

6. The 2nd part of the message ID should be the domain through which the mail was sent, otherwise it's mostly spoofed.

7. If the friendly name is the sender's name, which the receiver knows or has a general idea about, but the e-mail in the from field is unusual like spammer@spoof.com, the email is spoofed.

8. The SPF record helps identify the mail server's authenticity to send emails on behalf of a domain. If the value of SPF field is pass, it means the client is authorized to inject a given mail with the given identity. Otherwise, the email could be spoofed if it shows fail.

Phase 2: Separation of HAM and SPAM

Section 2.1:

Domain Keys Identified Mail

DKIM allows an organization to take responsibility for a message in a way that can be verified by a recipient. Domain Keys Identified Mail uses cryptographic technique to validate authorization and attaches new domain identifier. It is independent from any other identifier that message might have. It is a domain level authentication framework for emails. Every administrative domain (such as Gmail, Yahoo, any local network etc. from which the e-mail originates) generates public/private key pair. The public key is added in DNS (Domain Name System) resource records and private key is used for signing user's e-mail message and the selected RFC 5322 headers. DKIM has two standard word specifications. One, it permits the domain to use the digital signatures for transferring of emails. Second, the domain can publish its methods or techniques how it applies those digital signatures. Thus, DKIM assist the receivers in identifying the trusted senders.

DKIM Components

MUA (Message User Agent): At the sending side, this formats the message and submits it to MHS via MSA. On the other hand i.e. at the receiving side, it processes the message for storage or delivery to the user.

MHS (Message Handling service): A network consisting of the various components namely MSA, MDA and MTAs is together called a MHS.

MSA (Message Submission Agent): This is signer of the message. It enforces the message into Internet standards and policies of hosting domain before relaying to MTA.

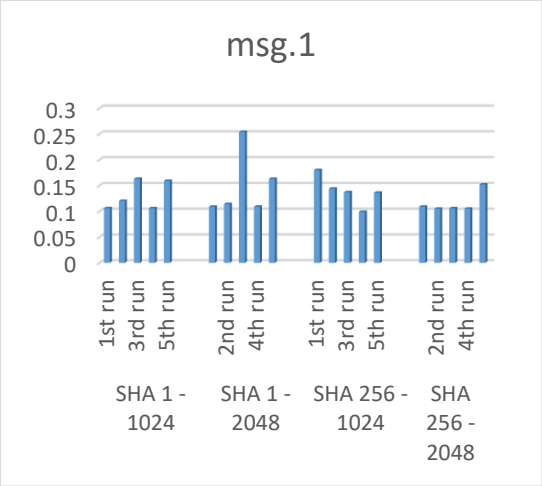
MTA (Message Transfer Agent): This takes care of relaying message from MSA to MDA through one or more MTAs.

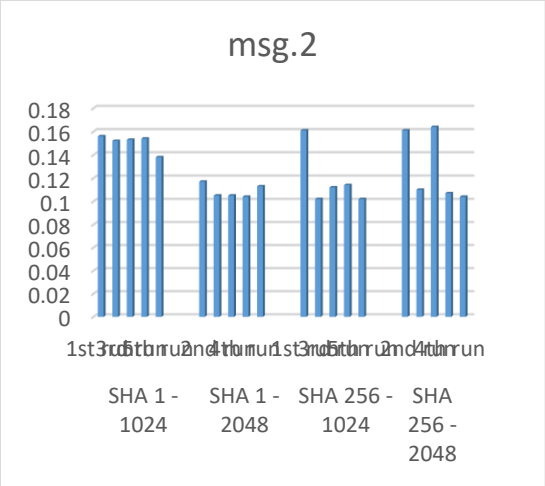
MDA (Message Digest Agent): This is the verifier of the message. It performs the verification test on the message and if the test is successful, it transfers the message from MHS to MS

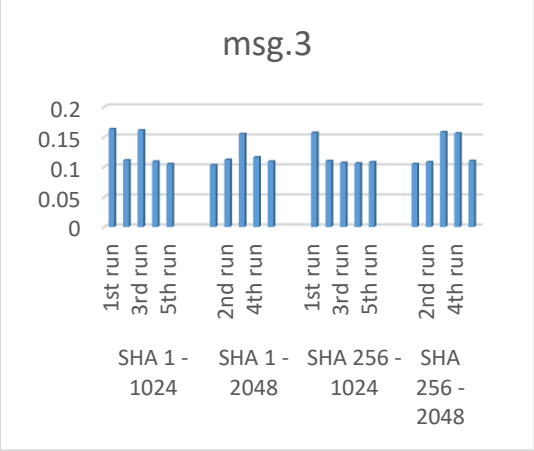
MS (Message Store): This is used to store all the messages received by the associated MUA. It can be on the remote machine or on the local machine where the MUA is present.

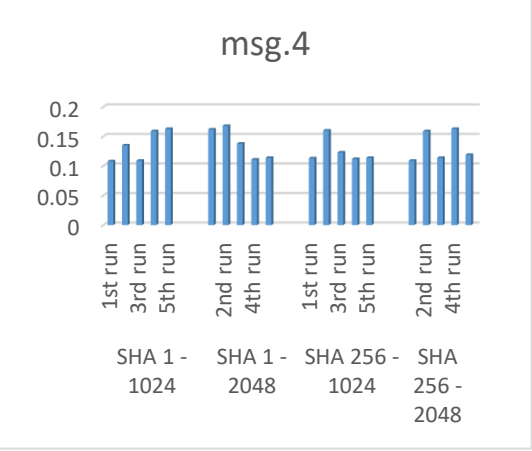
Section 2.2:

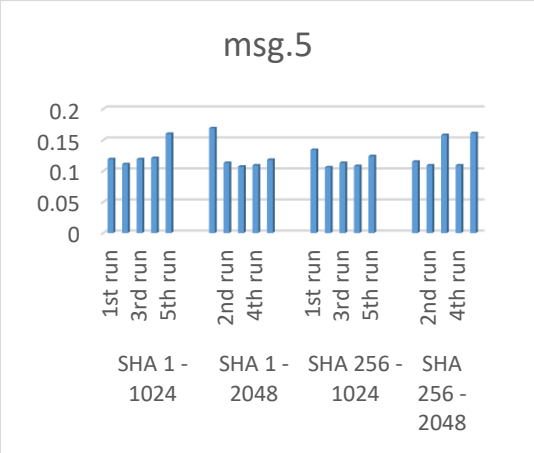
Signing

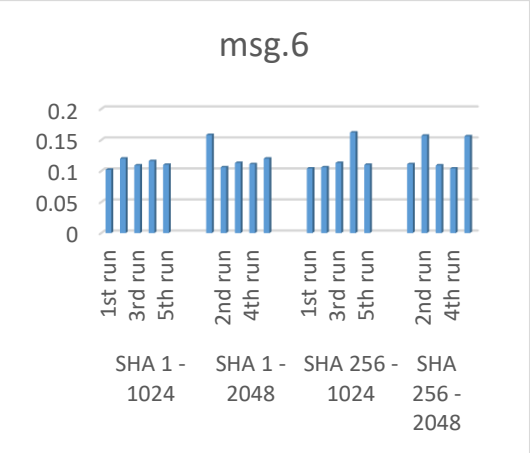


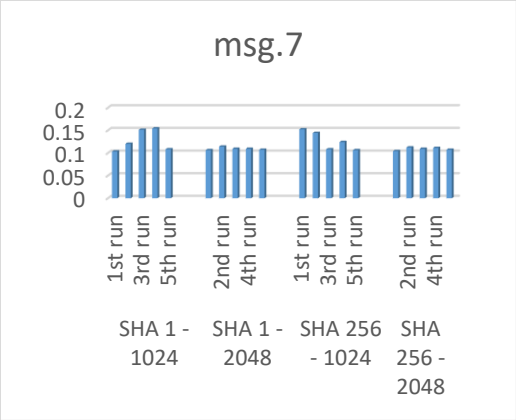


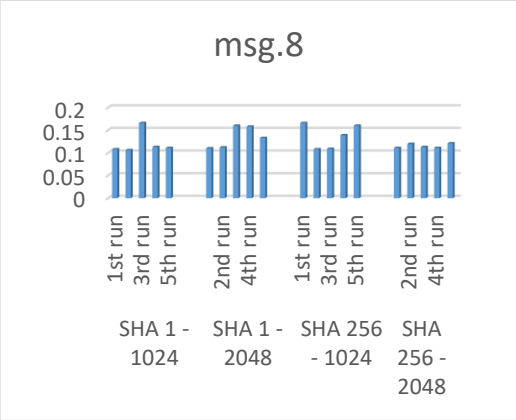


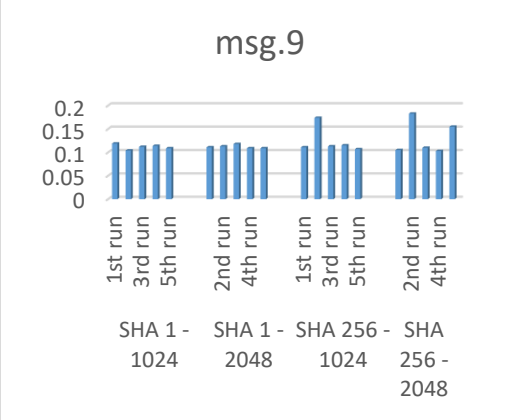


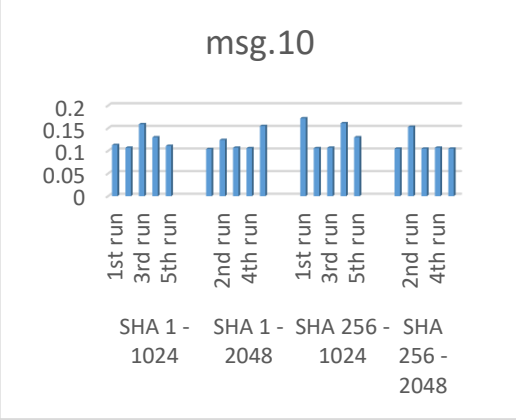




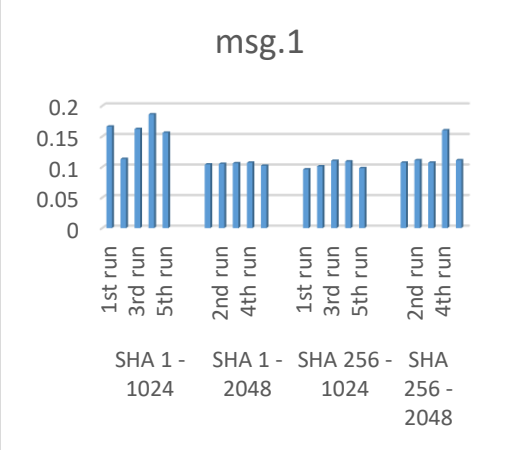


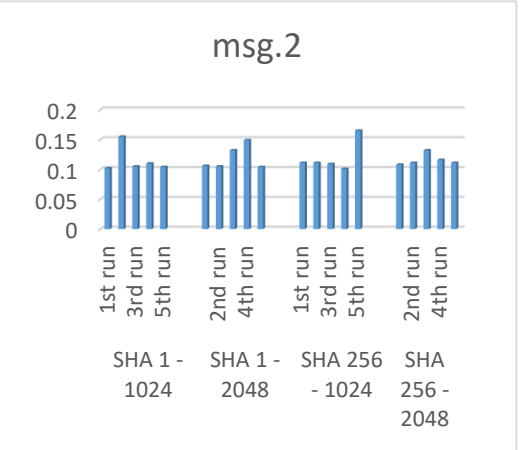


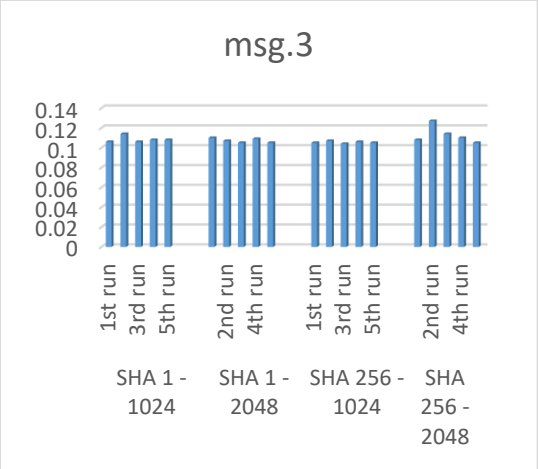


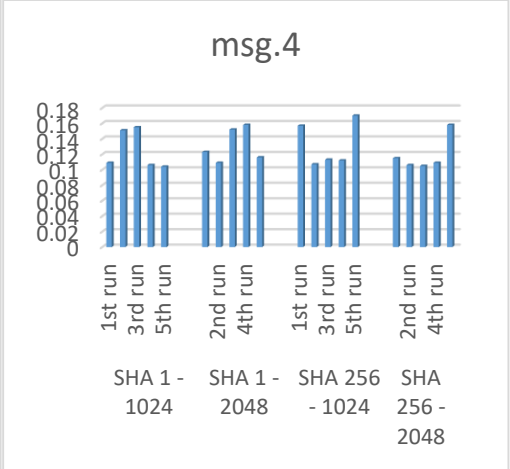


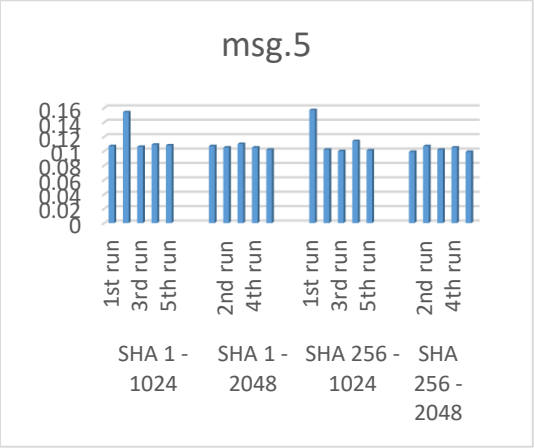
Verification

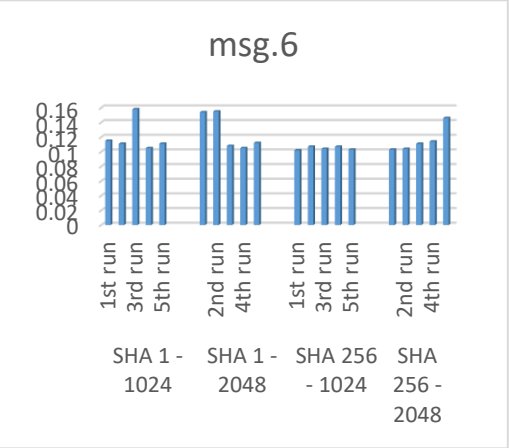


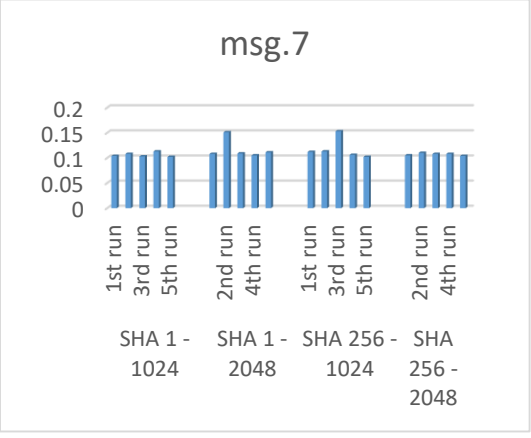


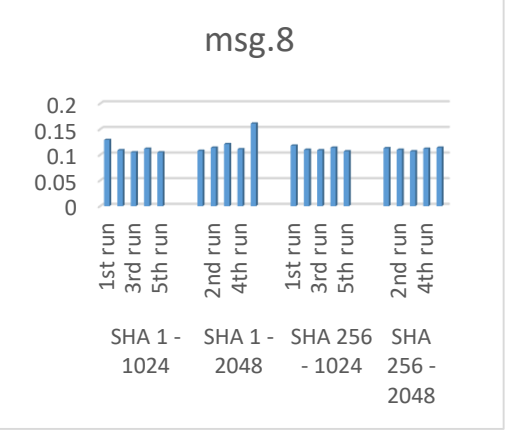

msg.3
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048

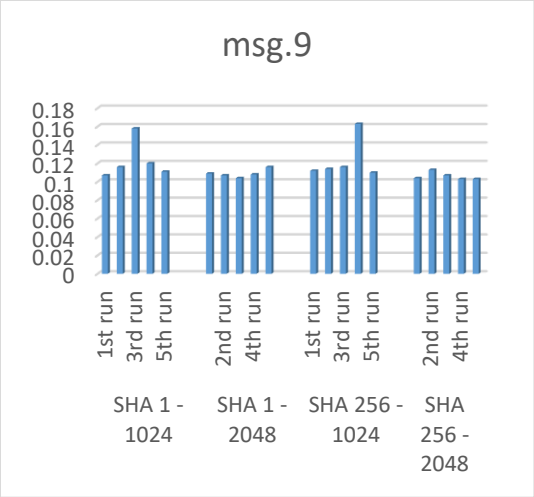

msg.4
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048

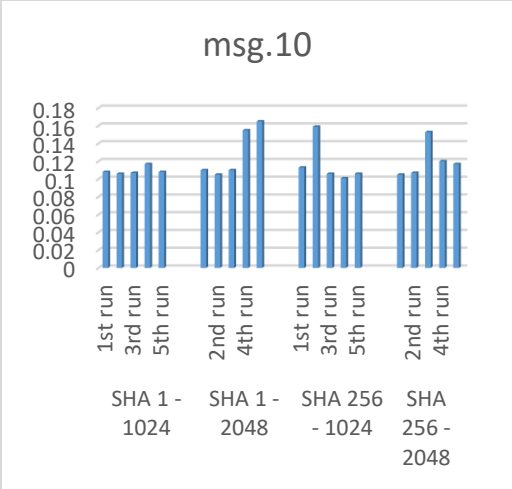

msg.5
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048


msg.6
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048


msg.7
0.2
0.15
0.1
0.05
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048


msg.8
0.2
0.15
0.1
0.05
0
1st run
3rd run
5th run
2nd run
4th run
1st run
3rd run
5th run
2nd run
4th run
SHA 1 - 1024
SHA 1 - 2048
SHA 256 - 1024
SHA 256 - 2048





Section 2.3:

Out of the four algorithms, 1024 RHA + SHA 1 has a better performance when run on the same length of data with smaller key size and digest.

Section 2.4:

1. S/MIME (Secure/Multipurpose Internet Mail Extensions) is a standard for public key encryption signing of MIME data and PGP (Pretty Good Privacy) is used in data communication for providing privacy and authentication.

But it has following shortcomings

   a. It needs an understanding of Public-Private key generation. And it needs public key to be shared.
   b. S/MIME requires that you keep certificates on file (or use public key infrastructure) for every organization you exchange data with.
   c. It requires S/MIME configuration both at the sending and receiving end. Which makes communication with user not using S/MIME encryption difficult.
   d. Message body needs to be modified in order to provide a packaging to a message.

2. Difference between DKIM and other email signature schemes.
   a. Signing of the mail is done by administrator in DKIM while it is done by the user in other email signature schemes.
   b. DKIM uses DNS-based self-certified keys. it does not require generalized, powerful and long-term certificates, issued by separate authorities.

c. DKIM uses less expensive mechanism since it is meant to be used for short-term use. Other signature schemes such as S/MIME or PGP use expensive encryption mechanisms.
d. S/MIME modifies the message body to provide a packaging. DKIM does not alter original message.

3. Broad categories of Domain Validations:
   Two broad categories for domain validators is **Based on IP addresses** and **Bases on digital signatures.** DKIM is based on digital signature validation.

4. DKIM helps a signer to prevent itself from forgery or improper use. Only signer can generate particular message because only he has the access to the private key. This helps in protecting its reputation at the receiving side. On the other side it helps receiver to verify if the message received is from the legitimate user or not. If the verification test fails it denies the message from the signer and if the test passes then the reputation builds and is used for further communication by avoiding extra processing.

5. The digital signature is generated by sending MTA by an applying an algorithm on the signed fields and generates a unique sequence of characters, a hash value. The public key used to generate it stored at the listed domain. The recipient MTA on receiving, verifies DKIM signature by getting public key through DNS. This key is used to decrypt the hash value in the header and also the hash value is recalculated for the mail it receives. If both those match then the header fields are not forged, otherwise forged. Thus DKIM signature signifies if all the fields in a header are forged or not.

Code Snippet

```
#include <fcntl.h>
#include<string.h>
#include <openssl/sha.h>
#include<stdio.h>
#include <time.h>
#include <sys/time.h>
void SHA1Digest1024Sign(char *);
void SHA256Digest1024Sign(char *);
void SHA1Digest2048Sign(char *);
void SHA256Digest2048Sign(char *);

int main(){
int  pid,i,status;
char *filename[] =
{"msg.1","msg.2","msg.3","msg.4","msg.5","msg.6","msg.7","msg.8","msg.9","msg
.10"};

struct timeval begin, end;
unsigned int t;


clock_t tic,toc;
for(i=0;i<10;i++){
        tic = clock();
        gettimeofday(&begin, NULL);
        pid = fork();
        if (pid == 0){
                printf("\nInside child process\n");
                SHA1Digest1024Sign(filename[i]);

        }

        waitpid(pid,&status,0);
        printf("\nInside parent process\n");
        gettimeofday(&end, NULL);

        //get the total number of ms that the code took:

        t = end.tv_usec - begin.tv_usec;

        toc = clock();

        printf("Time taken to generate SHA1 Digest for file : %s and Sign with
1024 bit rsa : %f milliseconds %fmsright \n",filename[i], (double)(toc -
                tic)*1000 / CLOCKS_PER_SEC,(double) t/1000);
        printf("Process 1 - SHA1Digest1024Sign complete. Iteration : %d
\n",i);

        tic = clock();
        pid = fork();
        if (pid == 0){
                SHA256Digest1024Sign(filename[i]);
        }
        waitpid(pid,&status,0);
        toc = clock();
```

```
        printf("Time taken to generate SHA256 Digest for file : %s and Sign
with 1024 bit rsa : %f milliseconds\n",filename[i], (double)(toc -
        tic)*1000 / CLOCKS_PER_SEC);
        printf("Process 2 - SHA256Digest1024Sign complete. Iteration : %d
\n",i);


        tic = clock();
        pid = fork();
        if (pid == 0){
                SHA1Digest2048Sign(filename[i]);
        }
        waitpid(pid,&status,0);
        toc = clock();
        printf("Time taken to generate SHA1 Digest for file : %s and Sign with
2048 bit rsa : %f milliseconds\n",filename[i], (double)(toc -
        tic)*1000 / CLOCKS_PER_SEC);
        printf("Process 3 - SHA1Digest2048Sign complete. Iteration : %d
\n",i);

        tic = clock();
        gettimeofday(&begin, NULL);
        pid = fork();
        if (pid == 0){
                SHA256Digest2048Sign(filename[i]);
        }
        waitpid(pid,&status,0);
        gettimeofday(&end, NULL);
        t = end.tv_usec - begin.tv_usec;

        toc = clock();
        printf("Time taken to generate SHA256 Digest for file : %s and Sign
with 2048 bit rsa : %f milliseconds %fmsright\n",filename[i], (double)(toc -
                tic)*1000 / CLOCKS_PER_SEC,(double) t/1000);
        printf("Process 4 - SHA256Digest2048Sign complete. Iteration : %d
\n\n*********************************************\n",i);


}
return 0;
}

void  SHA1Digest1024Sign(char *filename)
{

char outputFileName[200]="";
strcat(outputFileName,filename);
strcat(outputFileName,".sha1.1024");
printf("%s \n",outputFileName);
char *args[] = {"openssl", "dgst", "-sha1", "-verify",
"rsapublickey1024.pem", "-signature", outputFileName, filename, NULL};
//char *args[] = {"openssl", "dgst", "-sha1", "-sign",
"rsaprivatekey1024.pem", "-out", outputFileName, filename, NULL};
execvp(args[0],args);

}
```

```
void  SHA256Digest1024Sign(char *filename)
{
//printf("Process  - SHA256Digest1024Sign Started. Filename : %s
\n",filename);
clock_t tic = clock();
char outputFileName[200]="";
strcat(outputFileName,filename);
strcat(outputFileName,".sha256.1024");
char *args[] = {"openssl", "dgst", "-sha256", "-verify",
"rsapublickey1024.pem", "-signature", outputFileName, filename, NULL};
//char *args[] = {"openssl", "dgst", "-sha256", "-sign",
"rsaprivatekey1024.pem", "-out", outputFileName, filename, NULL};
execvp(args[0],args);
//clock_t toc = clock();
//printf("Time taken by to generate SHA1 Digest for file : %s and Sign with
1024 bit rsa : %f milliseconds\n", (double)(toc - tic)*1000 /
}

void  SHA1Digest2048Sign(char *filename)
{
//printf("Process  - SHA1Digest2048Sign Started. Filename : %s \n",filename);
char outputFileName[200]="";
strcat(outputFileName,filename);
strcat(outputFileName,".sha1.2048");
char *args[] = {"openssl", "dgst", "-sha1", "-verify",
"rsapublickey2048.pem", "-signature", outputFileName, filename, NULL};
//char *args[] = {"openssl", "dgst", "-sha1", "-sign",
"rsaprivatekey2048.pem", "-out", outputFileName, filename, NULL};
execvp(args[0],args);
}


void  SHA256Digest2048Sign(char *filename)
{
printf("Process  - SHA256Digest2048Sign Started. Filename : %s \n",filename);
char outputFileName[200]="";
strcat(outputFileName,filename);
strcat(outputFileName,".sha256.2048");
char *args[] = {"openssl", "dgst", "-sha256", "-verify",
"rsapublickey2048.pem", "-signature", outputFileName, filename, NULL};
//char *args[] = {"openssl", "dgst", "-sha256", "-sign",
"rsaprivatekey2048.pem", "-out", outputFileName, filename, NULL};
execvp(args[0],args);
}
```

REFERENCES:

https://blog.returnpath.com/10-tips-on-how-to-identify-a-phishing-or-spoofing-email/

http://www.w3.org/Protocols/rfc1341/4_Content-Type.html
http://forensicswiki.org/wiki/Using_message_id_headers_to_determine_if_an_email_has_been_forged
https://en.wikipedia.org/wiki/Message-ID
https://www.google.com/support/enterprise/static/postini/docs/admin/en/admin_ee_cu/header_received.html
https://mediatemple.net/community/products/dv/204643950/understanding-an-email-header
http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1048&context=adf
https://support.google.com/a/answer/33786?hl=en
http://www.meridiandiscovery.com/articles/email-forgery-analysis-in-computer-forensics/
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.5284&rep=rep1&type=pdf