11/24/2015

E-mail Forensics with DKIM

CSE 565- Project 3

Submitted by-

Ahut Gupta 50169273

Aditi Asthana 50169677

Rachna Shivangi 50169616

Shaleen Mathur 50170060

Nikhil Ayyagiri 50169485

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.

1. *First e-mail (Sent from Gmail account to UB account using UB e-mail ID)*

Delivered-To: ahutgupt@g-mail.buffalo.edu
The e-mail address to which the message was delivered to.

Return-Path: ahutgupt@buffalo.edu
This is the email ID to which any reply to this message will be sent to.

The following headers list the servers which receive and transmit the message. When read from the bottom to the top, we can obtain the path travelled by the message. (also represented by a diagram in Section 1.3)

```
Received: by 10.107.168.18 with SMTP id r18csp1060050ioe;
        Sun, 22 Nov 2015 10:12:40 -0800 (PST)
X-Received: by 10.140.178.146 with SMTP id y140mr24577033qhy.100.1448215960959;
        Sun, 22 Nov 2015 10:12:40 -0800 (PST)
Received: from localmailg.acsu.buffalo.edu (localmailg.acsu.buffalo.edu.
[128.205.4.25])
        by mx.google.com with ESMTP id a73si8410130qkb.126.2015.11.22.10.12.40
        for <ahutgupt@g-mail.buffalo.edu>;
        Sun, 22 Nov 2015 10:12:40 -0800 (PST)
Received-SPF: pass (google.com: best guess record for domain of ahutgupt@buffalo.edu
designates 128.205.5.229 as permitted sender) client-ip=128.205.5.229;
```

The message is authenticated at mx.google.com and tested for the authenticity of the sender.
In this case, the message and the sender pass the test.

```
Authentication-Results: mx.google.com;
       spf=pass (google.com: best guess record for domain of ahutgupt@buffalo.edu
designates 128.205.5.229 as permitted sender) smtp.mailfrom=ahutgupt@buffalo.edu
Received: from smtp.buffalo.edu (smtp4.acsu.buffalo.edu [128.205.5.229])
  by localmailg.acsu.buffalo.edu (Prefixe) with ESMTP id BBB23E0999
  for <ahutgupt@buffalo.edu>; Sun, 22 Nov 2015 13:12:40 -0500 (EST)
Received: from mail-lf0-f50.google.com (mail-lf0-f50.google.com [209.85.215.50])
  (Authenticated sender: ahutgupt@buffalo.edu)
  by smtp.buffalo.edu (Postfix) with ESMTPSA id 6E0FE9C9E71
  for <ahutgupt@buffalo.edu>; Sun, 22 Nov 2015 13:12:40 -0500 (EST)
Received: by lfs39 with SMTP id 39so95560705lfs.3
 for <ahutgupt@buffalo.edu>; Sun, 22 Nov 2015 10:12:38 -0800 (PST)
```

Note here that there was a **delay of 2 seconds**, while the message was transferred from the google server (`mail-lf0-f50.google.com`) to the UB mail server (`smtp.buffalo.edu`)

```
MIME-Version: 1.0
X-Received: by 10.25.28.70 with SMTP id c67mr8026993lfc.95.1448215958954; Sun,
 22 Nov 2015 10:12:38 -0800 (PST)
Received: by 10.112.146.106 with HTTP; Sun, 22 Nov 2015 10:12:38 -0800 (PST)
Date: Sun, 22 Nov 2015 13:12:38 -0500
```

The following headers store the attributes of the message like, the unique message ID, From, To, body of the message and other fields.

```
X-Gmail-Original-Message-ID:
<CAGzTRh2zteAAysPQrpQ4gwqfFXpu9r6wMtOFOaBAMW5HSa9OSA@mail.gmail.com>
Message-ID: <CAGzTRh2zteAAysPQrpQ4gwqfFXpu9r6wMtOFOaBAMW5HSa9OSA@mail.gmail.com>
Subject: Security Project 3
From: Ahut Gupta <ahutgupt@buffalo.edu>
To: ahutgupt@buffalo.edu
Content-Type: multipart/alternative; boundary=001a11402258caaec20525250d52
```

```
--001a11402258caaec20525250d52
Content-Type: text/plain; charset=UTF-8

This is a test message.

--001a11402258caaec20525250d52
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr">This is a test message.=C2=A0</div>
```

2. Second e-mail (Sent from UB account to UB account using UB e-mail ID

```
MIME-Version: 1.0
```

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that helps extend the limited capabilities of email by allowing insertion of images, sounds and text in a message.

```
Received: by 10.107.23.5 with HTTP; Sun, 22 Nov 2015 10:14:22 -0800 (PST)
Date: Sun, 22 Nov 2015 13:14:22 -0500
Delivered-To: ahutgupt@buffalo.edu
Message-ID: <CANdbVBMRbXObMx+SB_2Q6_EDs7RacrAThA1bnOJMgZ3130yB=A@mail.gmail.com>
Subject: Security Project 3
From: Ahut Gupta <ahutgupt@buffalo.edu>
To: ahutgupt@buffalo.edu
Content-Type: multipart/alternative; boundary=001a113f8efaf3dc0c0525251385

--001a113f8efaf3dc0c0525251385
Content-Type: text/plain; charset=UTF-8
```

Section 1.2

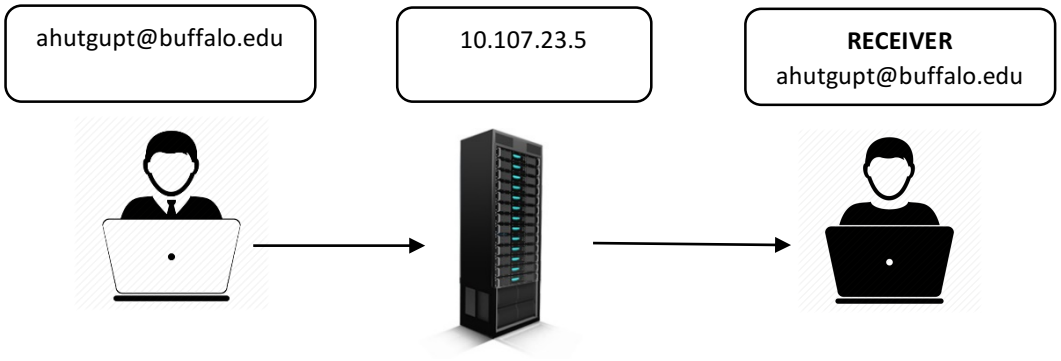
Unique Identifiers of the message:

Message-ID: This is a unique string assigned by the email server when the message is first created. As this is available in plaintext to anyone who has access to the message headers, this field can be spoofed easily.

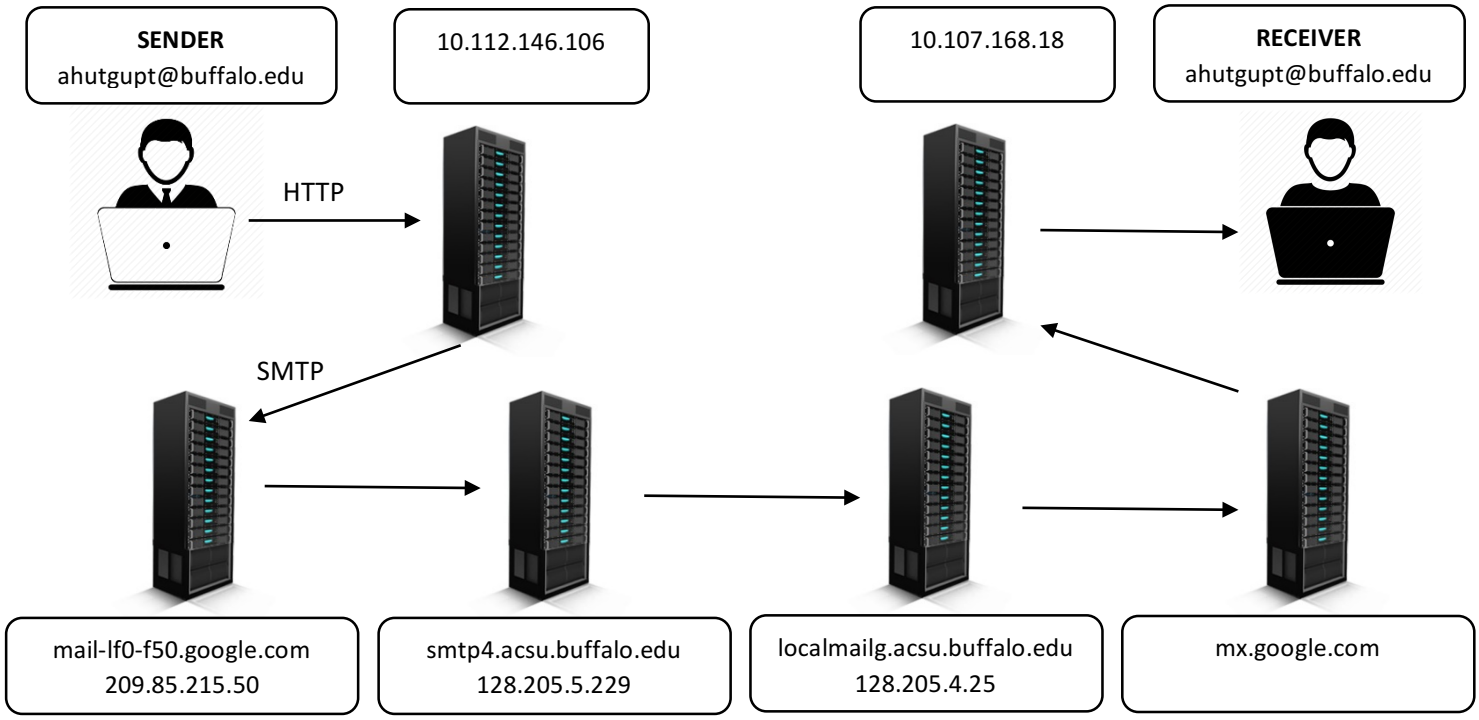
Date: This is the exact date and time at which the message was created. Again, as in Message ID, this field can be modified by any adversary and spoofed easily.

Section 1.3

Second email *(sent from UB mail)*



First email (sent from Gmail)



As expected, the message travels through a longer path in the first case, when we use Gmail to send the message. Whereas in the second case, it's just a one-step process.

Section 1.4

E-mail spoofing is the forgery of an e-mail header so that the message appears to have originated from someone or somewhere other than the actual source. In this case, let's consider the first e-mail. In this e-mail, we see that all the fields, "Sender", "Return-Path" and "From" have the same value: ahutgupt@buffalo.edu. Thus we can conclude that the **first e-mail is not spoofed.** Similarly for the second e-mail, we see that all the above mentioned fields have the same email address. Therefore, we can conclude that the **second e-mail is not spoofed**.

Section 1.5

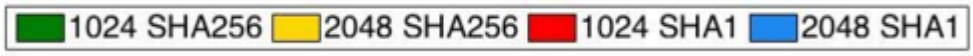
We should compare the fields "Sender", "Return-Path" and "From" in the message headers. If all the fields have the same email-id, then we can conclude that the message is coming from the original sender and has not been tampered with or spoofed. If any of those fields are different, particularly, "Return-Path", then the email has been spoofed by some third-party user.

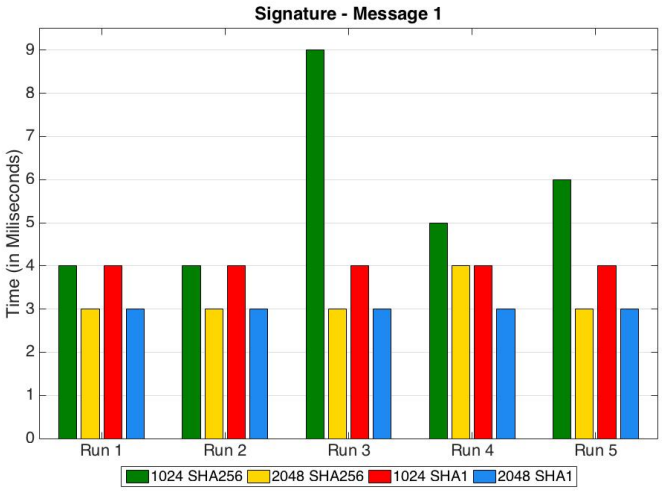
Section 2.1

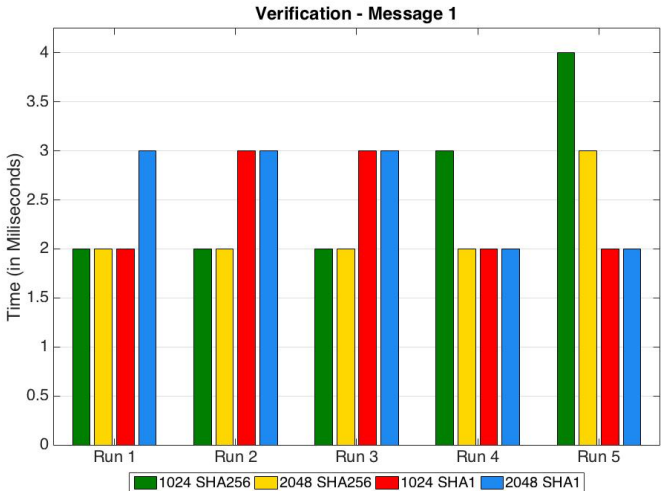
DKIM stands for Domain Keys Identified Mail which is used to authenticate sender or receiver. It is primarily responsible to evaluate whether email spoofing has been done or not and ensures signing domain is responsible for a message. In DKIM, we associate a domain name to an email message, thereby allowing a person or organization to claim responsibility for the message. It adds a digital signature to the message where, signing will be done by message originator's Administrative Management Domain (ADMD). Signing is typically performed by any of the functional components in that environment, including a Mail User Agent (MUA), a Mail Submission Agent (MSA), or an Internet Boundary Message Transfer Agent (MTA). After a message has been signed, any agent (MSA or MTA or MUA) in the message transit path can choose to validate the signature. Receiver can verify the signature by querying the signer's domain directly to retrieve the appropriate public key, and thereby confirm that the message was attested to by a party in possession of the private key for the signing domain. Usually, Validation will be done by an agent in the ADMD of the message recipient. The functional components are Mail User Agent (MUA), a Mail Submission Agent (MSA), or an Internet Boundary Message Transfer Agent (MTA). Basic components of ADMD includes Edge, User and Transit. The signature includes a number of fields like DKIM version, algorithm used to

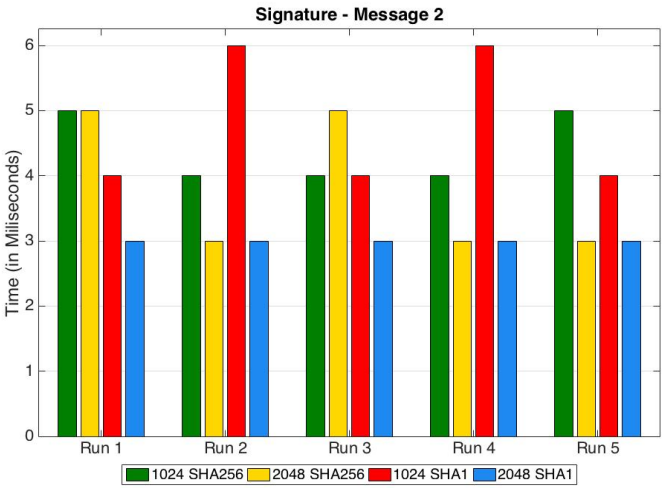
generate signature, domain name, Canonicalization method used on the header and the body, signed header fields, the hash of the canonicalized body part of the message and so on, where each field begins with a tag consisting of a tag code.

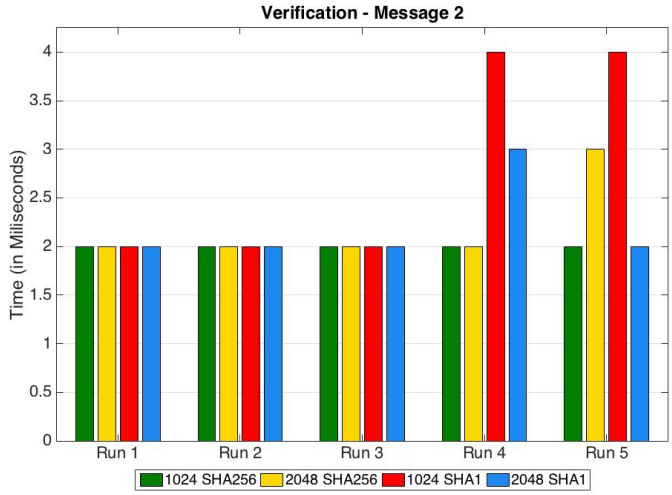
Section 2.2

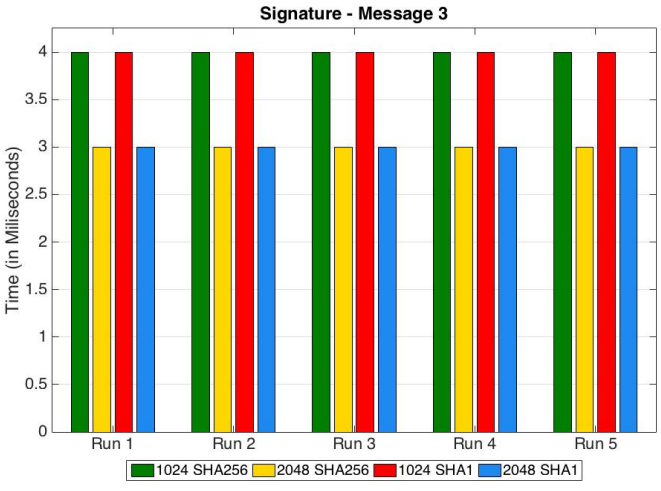
Color legend: 

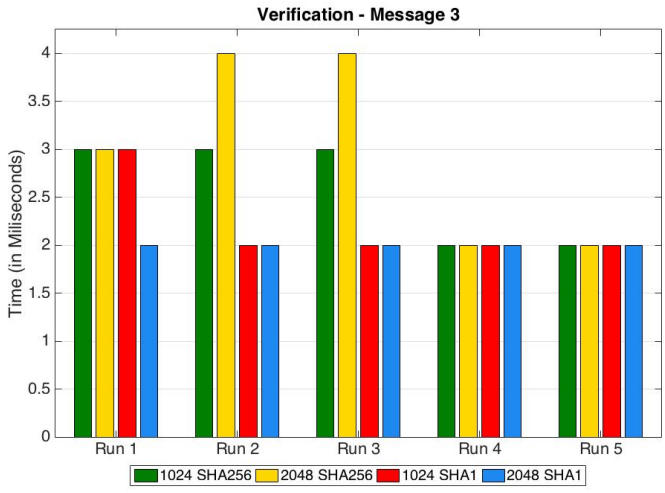


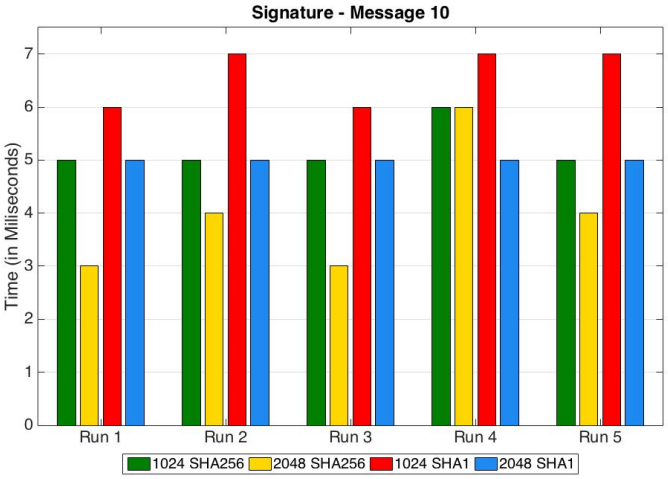


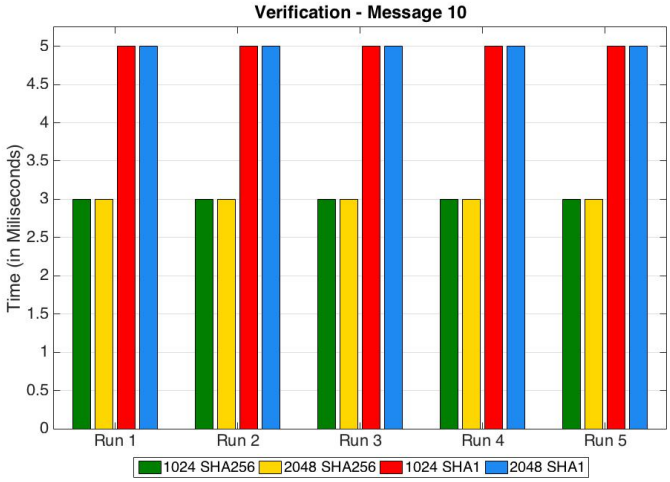












Section 2.3

	2048 SHA1	1024 SHA1	2048 SHA256	1024 SHA256
Sign (msg1 - 3kB)	3	4	3.2	5.6
Sign (msg10 – 538kB)	5	6.6	4	5.2
Verification (msg1)	2.6	2.4	2.1	2.6
Verification (msg10)	5	5	3	3

On comparing (average of 5 runs) the time taken by 4 techniques for different size messages (1 - smallest size and 10 - largest size among the 10 messages we are working on) , we can see that SHA256 with 2048 bit key performs best as it takes the least time for both signature and verification.

Section 2.4:

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

PGP:

- No forward secrecy and doesn't have plausible deniability – If the sender's or receivers key is compromised by any means. All our past messages and future messages can be decrypted.
- Both sender and receiver must be using PGP for PGP encryption to work.
- Sender and receiver should have software capable of reading the encrypted material and apply the corresponding decryption algorithm. In order to avoid conflict, both the sender and receiver must check and compare their PGP versions before exchanging encrypted data
- Messaging to a group is unwieldy as each copy has to be encrypted.
- Complexity of PGP is high compared to symmetric encryption, which uses one key, or asymmetric encryption, which uses two because PGP uses symmetric encryption with two keys.
- PGP is helpless if password is forgotten or lost as the encryption algorithms is strong, so forgotten passwords results in lost messages or lost files.
- Growth of different versions and sources of PGP further complicates the administration.

S/MIME:

- S/MIME requires the use of trusted certificate authorities but sometimes the receiver has difficulty in opening the encrypted message which is because of untrusted root CAs, intermediate CAs that can't be validated, or CRLs that are not available. It

- S/MIME provides an end to end security, so a third party malware detector can't be used in between the process. If a malware is ingested with the message, it can only be detected after decryption at receiver's side which might be an issue to the receivers end.
- Only some email software can handle S/MIME signatures, which results in attaching smime.p7s to those software which cannot handle. This might be confuse some users.
- As S/MIME requires certificate for implementation, not all users would like to trust certificate authorities. They may want to encrypt message with public/private key pair.
- S/MIME sometimes considered unsuitable for use via webmail clients. Some security practices require the private key to be kept accessible to the user but inaccessible from the webmail server, which further complicates the key advantage of webmail: providing ubiquitous accessibility.

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

DKIM is different from other email signature schemes such as S/MIME and PGP in following ways

- S/MIME and PGP signature scheme assumes that the receiver's mail system knows how to deal with the signed messages whereas notifies signers how to create the signatures and include them in their messages, and informs verifiers how to interpret and verify the signatures.
- DKIM allows the signer to include selected headers.
- S/MIME and OpenPGP might not involve the participation of the domain owner. This which does not provide control over the use of addresses in the domain but whereas, DKIM is designed to provide domain owner with the control over the use of addresses in the domain.
- DKIM does not depend on public and private key pairs which are issued by trusted certificate authorities.
- DKIM uses signing mechanism to put the signature in the message header. Whereas, in S/MIME or PGP modification of the body of message is required.
- DKIM can track a given email whether it really came from the mail server which is specified in received headers of the mail whereas OpenPGP (for signed messages), does not care about what servers a given email went through.
- Deployment of new internet protocols is easy as there is no dependency.
- Unlike PGP and S/MIME, DKIM satisfies the short-term needs of email transit authentication which make it less costly to use because DKIM uses DNS-based self-certified keys.
- DKIM can also be entirely invisible to end-users whose software does not support DKIM, whereas S/MIME and OpenPGP are quite intrusive.
- DKIM is based on domain names rather than complete email address, which implies signing is controlled by domain server and not individual email users.

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

Domain validation falls under two categories

- Validation which relies on IP address
- Validation which relies on digital signature.

Domain validation in DKIM is done using digital signatures.

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

- Signer- Signer is responsible for having placed a message into the network. Signing practices are added in order to allow sending domain to convey information to verifiers about how it chooses to sign the mail. So, domain has the power to defend its name against improper use, and to protect its reputation.
- Receiver- DKIM helps receiver to determine whether it has come from the domain it says it did. This further helps receiver to trust this domain and helps to create a list of trustable domains which then would be allowed to bypass strict inspection process which consumes lot of time and resources.

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

No, DKIM signature does not signify that all the fields in the header information are not forged. DKIM does not guarantee that signed message will be uncorrupted. It tries to minimize the chance of corruption by signing the significant header fields. We know header fields are the most vulnerable fields of the message. Header fields are of two types significant and insignificant. These insignificant header fields are vulnerable to modification while in transit. Sometimes, we do not sign insignificant header fields because it increases performance and the chance that the signature can be successfully verified. Hence, it does not guarantee that DKIM signature signifies that header information are not forged.

References

1. http://science.opposingviews.com/disadvantages-pgp-encryption-2300.html
2. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.114&rep=rep1&type=pdf
3. https://en.wikipedia.org/wiki/Email_authentication
4. Cryptography and Network Security Principles and Practice Sixth Edition by William Stallings.
5. http://security.stackexchange.com/questions/41289/please-help-verify-my-understanding-of-domain-validation-dv-ssl-certificate
6. http://security.stackexchange.com/questions/23180/is-dkim-needed-for-emails-encrypted-with-openpgp
7. http://www.dkim.org/info/dkim-faq.html
8. https://en.wikipedia.org/wiki/S/MIME
9. https://tools.ietf.org/html/draft-ietf-dkim-overview-01

Appendix

KeyGeneration.C

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void main(){
 pid_t pID = fork(); // creates copy of the parent process - in this case 1 chid process have been created
 if (pID ==0) //code executed by child1
  {
   char *args[] = {"genrsa","-out","rsaprivatekey2048.pem","2048",NULL };
   execvp("openssl", args);
  }
  else if(pID<0) // Case to handle exception when fork failed to create child process
  {
    printf("Failed to fork") ;
    exit(1);
  }
  else
   {
     // code executed by parent process
                    int returnStatus;
                    waitpid(pID, &returnStatus, 0); //waits for child process to terminate(in this case child1)
     pid_t pID1 = fork();
     if (pID1 ==0) //code executed by child2
     {
        char *args1[] = {"genrsa", "-out", "rsaprivatekey1024.pem", "1024", NULL };
        execvp("openssl", args1);
     }
     else //code executed by Parent Process
     {
                                int returnStatus1;
                                waitpid(pID1, &returnStatus1, 0);
                                pid_t pID2=fork();
```

```
        if(pID2==0){ //code executed by child3
           char*args2[]={"rsa","-in","rsaprivatekey2048.pem","-out","rsapublickey2048.pem","-outform","PEM","-pubout"};
           execvp("openssl",args2);
        }
        else //code executed by Parent Process
        {
           pid_t pID3=fork();
           if(pID3==0){ //code executed by child4
                                                    char*args3[]={"rsa","-in","rsaprivatekey1024.pem","-
out","rsapublickey1024.pem","-outform","PEM","-pubout"};
                                                    execvp("openssl",args3);
                                             }
        }
     }
  }
}
```

Signature.C

```
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
int main(void)
{
 int i;
 for(i =1;i<=10;i++)
 {
    printf("Message : %d\n", i);
    struct timeval timeVar1, timeVar2, timeVar3, timeVar4, timeVar5, timeVar6, timeVar7, timeVar8;
    char command2048SHA1[200];
    sprintf(command2048SHA1, "openssl dgst -sha1 -sign rsaprivatekey2048.pem msg%d.txt >
./Output/2048SHA1/cipher%d.txt.sig" , i,i);
    gettimeofday(&timeVar1, NULL);
    system(command2048SHA1);
    gettimeofday(&timeVar2, NULL);
    printf("     2048sha1 - %ld\n", (timeVar2.tv_usec-timeVar1.tv_usec)/1000 );
    char command1024SHA1[200];
    sprintf(command1024SHA1, "openssl dgst -sha1 -sign rsaprivatekey1024.pem msg%d.txt >
./Output/1024SHA1/cipher%d.txt.sig" , i,i);
    gettimeofday(&timeVar3, NULL);
    system(command1024SHA1);
    gettimeofday(&timeVar4, NULL);
    printf("     1024sha1 - %ld\n", (timeVar4.tv_usec-timeVar3.tv_usec)/1000 );
    char command2048SHA256[200];
    sprintf(command2048SHA256, "openssl dgst -sha256 -sign rsaprivatekey2048.pem msg%d.txt >
./Output/2048SHA256/cipher%d.txt.sig",i,i);
    gettimeofday(&timeVar5, NULL);
    system(command2048SHA256);
    gettimeofday(&timeVar6, NULL);
    printf("     2048sha256 - %ld\n", (timeVar6.tv_usec-timeVar5.tv_usec)/1000 );
    char command1024SHA256[200];
    sprintf(command1024SHA256, "openssl dgst -sha256 -sign rsaprivatekey1024.pem msg%d.txt >
./Output/1024SHA256/cipher%d.txt.sig",i,i);
    gettimeofday(&timeVar7, NULL);
```

```
    system(command1024SHA256);
    gettimeofday(&timeVar8, NULL);
    printf("      1024sha256 - %ld\n", (timeVar8.tv_usec-timeVar7.tv_usec)/1000 );
 }
}
```

Verify.C

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
int main()
{
 int i;
 for(i=1; i<=10 ; i++)
 {
    struct timeval timeVar1, timeVar2, timeVar3, timeVar4, timeVar5, timeVar6, timeVar7, timeVar8;
    char command2048SHA1[200];
    sprintf(command2048SHA1, "openssl dgst -sha1 -verify rsapublickey2048.pem -signature
./Output/2048SHA1/cipher%d.txt.sig msg%d.txt" , i,i);
    gettimeofday(&timeVar1, NULL);
    system(command2048SHA1);
    gettimeofday(&timeVar2, NULL);
    printf("2048sha1  %ld\n", (timeVar2.tv_usec-timeVar1.tv_usec)/1000 );
    char command1024SHA1[200];
    sprintf(command1024SHA1, "openssl dgst -sha1 -verify rsapublickey1024.pem -signature
./Output/1024SHA1/cipher%d.txt.sig msg%d.txt" , i,i);
    gettimeofday(&timeVar3, NULL);
    system(command1024SHA1);
    gettimeofday(&timeVar4, NULL);
    printf("1024sha1  %ld\n", (timeVar4.tv_usec-timeVar3.tv_usec)/1000 );
    char command2048SHA256[200];
    sprintf(command2048SHA256,"openssl dgst -sha256 -verify rsapublickey2048.pem -signature
./Output/2048SHA256/cipher%d.txt.sig msg%d.txt" ,i ,i);
    gettimeofday(&timeVar5, NULL);
    system(command2048SHA256);
    gettimeofday(&timeVar6, NULL);
    printf("2048sha256  %ld\n", (timeVar6.tv_usec-timeVar5.tv_usec)/1000 );
    char command1024SHA256[200];
    sprintf(command1024SHA256 , "openssl dgst -sha256 -verify rsapublickey1024.pem -signature
./Output/1024SHA256/cipher%d.txt.sig msg%d.txt" , i, i);
    gettimeofday(&timeVar7, NULL);
    system(command1024SHA256);
    gettimeofday(&timeVar8, NULL);
    printf("1024sha256  %ld\n", (timeVar8.tv_usec-timeVar7.tv_usec)/1000 );
 }
}
```