VERZEO MAY BATCH

A
Project Report
On
SECURITY ,DISPOSABLE E-MAILS AND USE A SECURE EMAIL SYSTEM

For the course
Cybersecurity

SUBMITTED BY

CS05B6
Alamuru Sainath Reddy
Bulagakula Ruthvick
Epuru Saiteja Reddy
Kanika Mulchandani
Vaibhav Sharma
Ananya Panda
Lavanya CM
Rakshitha Lokesh
Chaitanya Shrivastav
Akshara Magesh
Anjali Verma
Aditi Gireesh
Anirudh Kathikar

SUBMITTED TO

event@verzeo.in
assignments@anir0y.live

CONTENTS

Objectives:

Identify security issues related to common email systems.

Email security issues plague every single business or organisation. Why? Because it's an easy point of failure for scammers and hackers to exploit.

We've experienced our fair share of email problems throughout the years. We've also helped organisations clean up the mess after a problem while helping them better secure their email.

1. Sending Confidential Data

If emails aren't encrypted, sending confidential data is like telling hackers "please, come steal our information." We don't want that to happen to any business, but it happens all too often. Have a clear policy about what should and shouldn't be sent over email and ensure any confidential data is encrypted.

2. Malware

We see businesses scrambling to recover after malware attacks all the time. Malware sneaks in and wreaks havoc before anyone realizes anything's wrong. Where does malware love to hide? Emails. Why? It's easy to make an email seem legitimate, such as sending a malicious attachment but making it seem as if it came from a co-worker.

Unless employees check the sender address or double-check with the co-worker to ensure the email is from them, they open the malware infected attachment and the damage is done. With over 600 millions forms of malware and 1 in 131 emails being infected, it's likely one of the biggest threats. We recommend training employees on email security issues and how to avoid them. Plus, having strong anti-malware and firewall protection in places helps too.

3. Phishing

Outside of malware, phishing scams are one of the biggest email security threats. We've seen numerous employees who handed over their credentials to hackers, who then used those credentials to compromise a business's network. It's all too easy to click on a link in an email to go to an account. The only problem is those links are often phishing scams looking for user data.

4. Stolen Devices

We realize this might not sound like an email security problem, but when a smartphone or tablet is stolen, thieves simply tap to view all emails. If the device isn't secured with a passcode or biometric security, it takes seconds for thieves to compromise a business. We recommend securing devices and being able to remotely wipe them quickly.

5. Weak Passwords

No matter how strong a business's network is, a weak email password still poses a threat. We always advise businesses to use strong passwords for everything, but often, those passwords aren't easy to remember. Implementing stronger password policies and even adding two-factor authentication can help.

guerrillamail.com

Guerrilla Mail gives you a disposable email address. There is no need to register, simply visit Guerrilla Mail and a random address will be given. You can also choose your own address.

You can give your email address to whoever you do not trust. You can view the email on Guerrilla Mail, click on any confirmation link, then delete it. Any future spam sent to the disposable email will be zapped by Guerrilla Mail, never reaching your mailbox, keeping your mailbox safe and clean.

What is Disposable Temporary Email?

Disposable email - is a service that allows you to receive email at a temporary address that self-destructs after a certain time elapses. It is also known by names like : tempmail, 10minutemail, throwaway email, fake-mail or trash-mail. Many forums, Wi-Fi owners, websites and blogs ask visitors to register before they can view content, post comments or download something. Temp-Mail - is the most advanced throwaway email service that helps you avoid spam and stay safe.

Principle of Secure EMail

We use many of the tools to create a high-level design of a secure email system. We create this high-level design in an incremental manner, at each step introducing new security services. When designing a secure email system, let us keep in mind the racy example introduced, the illicit love affair between Alice and Bob. In the context of e-mail, Alice wants to send an e-mail message to Bob, and Trudy wants to intrude.

Before plowing ahead and designing a secure email system for Alice and Bob, we should first consider which security features would be most desirable for them. First and foremost is secrecy. Neither Alice nor Bob wants Trudy to read Alice's e-mail message. The second feature that Alice and Bob would most likely want to see in the secure e-mail system is sender authentication. In particular, when Bob receives the message from Alice, "I don't love you anymore. I never want to see you again. Formerly yours, Alice" , Bob would naturally want to be sure that the message came from Alice and not from Trudy. Another feature that the two lovers would appreciate is message integrity, i.e., assurance that the message Alice sends is not modified while enroute to Bob. Finally, the e-mail system should provide receiver authentication, i.e., Alice wants to make sure that she is indeed sending the letter to Bob and not to someone else (e.g., Trudy) who is impersonating as Bob.

So let's begin by addressing the foremost concern of Alice and Bob, namely, secrecy. The most straightforward way to provide secrecy is for Alice to encrypt the message with symmetric key technology (such as DES) and for Bob to decrypt the message on message receipt. If the symmetric key is long enough, and if only Alice and Bob have the key, then it is extremely difficult for anyone else (including Trudy) to read the message. Although this approach is straightforward, it has a fundamental problem as it is difficult to distribute a symmetric key so that only Alice and Bob have copies of the key. So we naturally consider an alternative approach, namely, public key cryptography (using, for example, RSA). In the public-key approach, Bob makes his public key publicly available (for example, in a public-key server or on his personal Web page), Alice encrypts her message with Bob's public key, and sends the encrypted message to Bob's e-mail address. (The encrypted message is encapsulated with MIME headers and sent over ordinary SMTP) When Bob receives the message, he simply decrypts it with his private key. Assuming that Alice knows for sure that the public key is Bob's public key (and that the key is long enough), then this approach is an excellent means to provide the desired secrecy. One problem, however, is that public-key encryption is relatively inefficient, particularly for long messages. (Long e-mail messages are now commonplace in the Internet, due to increasing use of attachments, images, audio and video.) To overcome the efficiency problem, let's make use of a session key. In particular, Alice (1) selects a symmetric key, K_S, at random, (2) encrypts her message, m, with the symmetric key, K_S,(3) encrypts the symmetric key with Bob's public key, e_B, (4) concatenates the encrypted message and the encrypted symmetric key to form a "package", and (5) sends the package to Bob's e-mail address. The steps are illustrated in Figure below. (In this and the subsequent figures, the "+" represents concatenation and the "-" represents de-concatenation.) When Bob receives the package, he (1) uses his private key d_B to obtain the symmetric key, S, and (2) uses the symmetric key S to decrypt the message m.

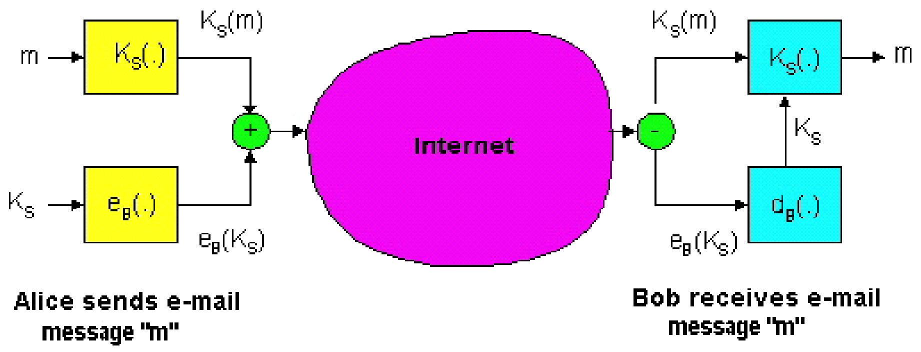


Figure: Alice uses a symmetric session key, K_S, to send a secret e-mail to Bob.

Having designed a secure e-mail system that provides secrecy, let's now design another system that provides both sender authentication and integrity. We'll suppose, for the moment, that Alice and Bob are no longer concerned with secrecy (they want to share their feelings with everyone!), and are only concerned about sender authentication and message integrity. To accomplish this task, we use digital signatures and message digests. Specifically, Alice (1) applies a hash function, H (e.g., MD5), to her message m to obtain a message digest, (2) encrypts the result of the hash function with her private key, d_A, to create a digital signature, (3) concatenates the original (unencrypted message) with the signature to create a package, (4) and sends the package to Bob's e-mail address. When Bob receives the package, he (1) he applies Alice's public key, e_A, to the electronic signature and (2) compares the result of this operation to his own hash, H, of the message. The steps are illustrated in Figure below. If the two results are the same, Bob can be pretty confident that message came from Alice and is unaltered.

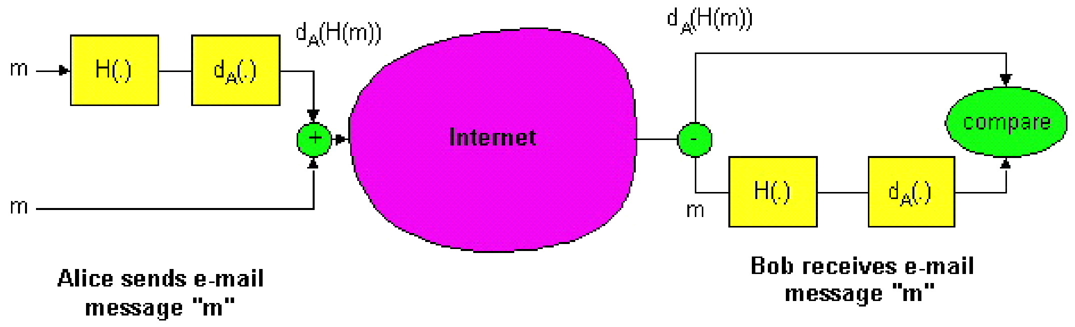

m
H(.)
dA(.)
dA(H(m))
+
m
Internet
dA(H(m))
-
compare
m
H(.)
dA(.)
Alice sends e-mail message "m"
Bob receives e-mail message "m"

Figure: Using hash functions and digital signatures to provide sender authentication and message integrity.

Now let's consider designing an email system that provides secrecy, sender authentication and message integrity. This can be done by combining the procedures in above both figures. Alice first creates a preliminary package, exactly as in the second Figure, which consists of her original message along with a digitally-signed hash of the message. She then treats this preliminary package as a message in itself, and sends this new message through the sender steps in first Figure, creating a new package that is sent to Bob. The steps applied by Alice are shown in Figure below. When Bob receives the package, he first applies his side of first Figure and then his side of second Figure. It should be clear that this design achieves the goal of providing secrecy, sender authentication and message integrity. Note in this scheme that Alice applies public key encryption twice: once with her own private key and once with Bob's public key. Similarly, Bob applies public key encryption twice - once with his private key and once with Alice's public key.

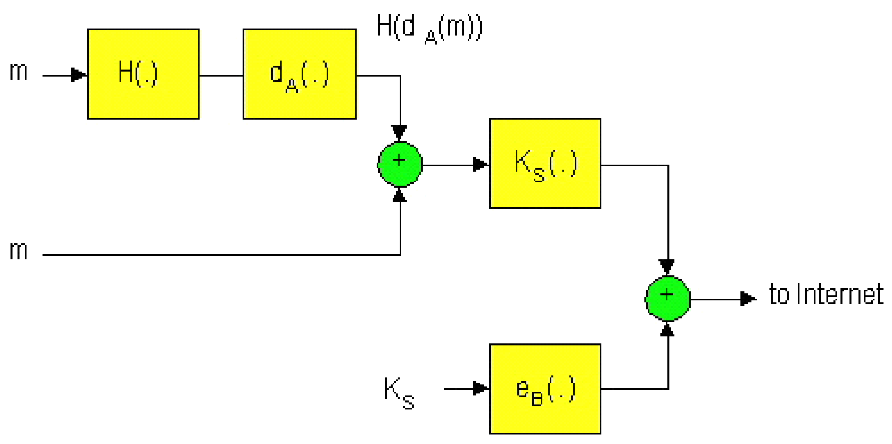


Figure: Alice uses symmetric-key cryptography, public-key cryptography, a hash function and a digital signature to provide secrecy, sender authentication and message integrity.

The secure email design outlined in this Figure probably provides satisfactory security for most email users for most occasions. But there is still one important issue that remains to be addressed. The design in this Figure requires Alice to obtain Bob's public key, and requires Bob to obtain Alice's public key. The distribution of these public keys is a non-trivial problem. For example, Trudy might masquerade as Bob and give Alice her own public key while saying that it is Bob's public key. A popular approach for securely distributing public keys is to certify the public keys.

PGP

Originally written by Phil Zimmerman in 1991, pretty good privacy (PGP) is e-mail, an encryption scheme that has become a de-facto standard, with thousands of users all over the globe. Versions of PGP are available in the public domain; for example, you can find the PGP software for your favorite platform as well as lots of interesting reading at the International PGP Home Page. PGP is also commercially available, and is also available as a plug-in for many e-mail user agents, including Microsoft's Exchange and Outlook, and Qualcomm's Eudora.

The PGP design is, in essence, the same as the design shown in third Figure. Depending on the version, the PGP software uses MD5 or SHA for calculating the message digest; CAST, Triple-DES or IDEA for symmetric key encryption; and RSA for the public key encryption. In addition, PGP provides data compression.

When PGP is installed, the software creates a public key pair for the user. The public key can be posted on the user's Web site or placed in a public key server. The private key is protected by the use of a password. The password has to be entered every time the user accesses the private key. PGP gives the user the option of digitally signing the message, encrypting the message, or both digitally signing and encrypting. Below Figure shows a PGP signed message. This message appears after the MIME header. The encoded data in the message is $d_A(H(m))$, i.e., the digitally signed message digest. As we discussed above, in order for Bob to verify the integrity of the message, he needs to have access to Alice's public key.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Bob:
My husband is out of town tonight.

Passionately yours, Alice

-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv

yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2

-----END PGP SIGNATURE-----
```

Figure: A PGP signed message.

Below Figure: shows a PGP secret message. This message also appears after the MIME header. Of course, the plaintext message is not included within the secret e-mail message. When a sender (such as Alice) wants both secrecy and integrity, the PGP would contain a message like that of Figure below contained within the message of above Figure.

```
-----BEGIN PGP MESSAGE-----

Version: PGP for Personal Privacy 5.0

u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdSlOjuHgbcfDssWe7/K=lKhnMikLo0+l/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp

-----END PGP MESSAGE
```

Figure: A secret PGP message.

PGP also provides a mechanism for public key certification, but the mechanism is quite different from the conventional certification authority that we examined. PGP public keys are certified by a web of trust. Alice can certify any pair of key and user name for which she believes the pair really belongs together. In addition, PGP permits Alice to say that she trusts another user to vouch for the authenticity of more keys. Some PGP users sign each other's keys by holding key signing parties. Users physically gather, exchange floppy disks containing public keys, and certify each other's keys by signing them with their private keys. PGP public keys are also distributed by PGP public key servers on the Internet. When a user submits a public key to such a server, the server stores a copy of the key, sends a copy of the key to all the other public-key servers, and serves the key to anyone who requests it. Although key signing parties and PGP public key servers actually exist, by far the most common way for users to distribute their public keys is posting them on their personal Web pages. Of course, keys on personal Web pages are not certified by anyone, but they are easy to access.

<u>Activity:</u>

Send E-mail using python

Python comes with the built-in smtplib module for sending emails using the Simple Mail Transfer Protocol (SMTP). smtplib uses the RFC 821 protocol for SMTP. The examples in this tutorial will use the Gmail SMTP server to send emails, but the same principles apply to other email services. Although the majority of email providers use the same connection ports as the ones in this tutorial, you can run a quick Google search to confirm yours.
To get started with this tutorial, set up a Gmail account for development, or set up an SMTP debugging server that discards emails you send and prints them to the command prompt instead. Both options are laid out for you below. A local SMTP debugging server can be useful for fixing any issues with email functionality and ensuring your email functions are bug-free before sending out any emails.
Option 1: Setting up a Gmail Account for Development
If you decide to use a Gmail account to send your emails, I highly recommend setting up a throwaway account for the development of your code. This is because you'll have to adjust your Gmail account's security settings to allow access from your Python code, and because there's a chance you might accidentally expose your login details. Also, I found that the inbox of my testing account rapidly filled up with test emails, which is reason enough to set up a new Gmail account for development.
A nice feature of Gmail is that you can use the + sign to add any modifiers to your email address, right before the @ sign. For example, mail sent to my+person1@gmail.com and my+person2@gmail.com will both arrive at my@gmail.com. When testing email functionality, you can use this to emulate multiple addresses that all point to the same inbox.
To set up a Gmail address for testing your code, do the following:
Create a new Google account.
Turn *Allow less secure apps* to *ON*. Be aware that this makes it easier for others to gain access to your account.

If you don’t want to lower the security settings of your Gmail account, check out Google’s documentation on how to gain access credentials for your Python script, using the OAuth2 authorization framework.

Option 2: Setting up a Local SMTP Server

You can test email functionality by running a local SMTP debugging server, using the smtpd module that comes pre-installed with Python. Rather than sending emails to the specified address, it discards them and prints their content to the console. Running a local debugging server means it’s not necessary to deal with encryption of messages or use credentials to log in to an email server.

You can start a local SMTP debugging server by typing the following in Command Prompt:

```
$ python -m smtpd -c DebuggingServer -n localhost:1025
```

On Linux, use the same command preceded by sudo.

Any emails sent through this server will be discarded and shown in the terminal window as a bytes object for each line:

```
---------- MESSAGE FOLLOWS ----------
b'X-Peer: ::1'
b''
b'From: my@address.com'
b'To: your@address.com'
b'Subject: a local test mail'
b''
b'Hello there, here is a test email'
------------ END MESSAGE ------------
```

For the rest of the tutorial, I’ll assume you’re using a Gmail account, but if you’re using a local debugging server, just make sure to use localhost as your SMTP server and use port 1025 rather than port 465 or 587. Besides this, you won’t need to use login() or encrypt the communication using SSL/TLS.

Sending a Plain-Text Email

Before we dive into sending emails with HTML content and attachments, you’ll learn to send plain-text emails using Python. These are emails that you could write up in a simple text editor. There’s no fancy stuff like text formatting or hyperlinks. You’ll learn that a bit later.

Starting a Secure SMTP Connection

When you send emails through Python, you should make sure that your SMTP connection is encrypted, so that your message and login credentials are not easily accessed by others. SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are two protocols that can be used to encrypt an SMTP connection. It’s not necessary to use either of these when using a local debugging server.

There are two ways to start a secure connection with your email server:

Start an SMTP connection that is secured from the beginning using SMTP_SSL().

Start an unsecured SMTP connection that can then be encrypted using .starttls().

In both instances, Gmail will encrypt emails using TLS, as this is the more secure successor of SSL. As per Python’s Security considerations, it is highly recommended that you use create_default_context() from the ssl module. This will load the system’s trusted CA certificates, enable host name checking and certificate validation, and try to choose reasonably secure protocol and cipher settings.

If you want to check the encryption for an email in your Gmail inbox, go to *More → Show original* to see the encryption type listed under the *Received* header.

smtplib is Python’s built-in module for sending emails to any Internet machine with an SMTP or ESMTP listener daemon.

I’ll show you how to use SMTP_SSL() first, as it instantiates a connection that is secure from the outset and is slightly more concise than the .starttls() alternative. Keep in mind that Gmail requires that you connect to port 465 if using SMTP_SSL(), and to port 587 when using .starttls().

Option 1: Using SMTP_SSL()

The code example below creates a secure connection with Gmail’s SMTP server, using the SMTP_SSL() of smtplib to initiate a TLS-encrypted connection. The default context of ssl validates the host name and its certificates and optimizes the security of the connection. Make sure to fill in your own email address instead of my@gmail.com:

```
import smtplib, ssl

port = 465  # For SSL
password = input("Type your password and press enter: ")

# Create a secure SSL context
context = ssl.create_default_context()

with smtplib.SMTP_SSL("smtp.gmail.com", port, context=context) as
server:
    server.login("my@gmail.com", password)
    # TODO: Send email here
```

Using with smtplib.SMTP_SSL() as server: makes sure that the connection is automatically closed at the end of the indented code block. If port is zero, or not specified, .SMTP_SSL() will use the standard port for SMTP over SSL (port 465).

It’s not safe practice to store your email password in your code, especially if you intend to share it with others. Instead, use input() to let the user type in their password when running the script, as in the example above. If you don’t want your password to show on your screen when you type it, you can import the getpass module and use .getpass() instead for blind input of your password.

Option 2: Using .starttls()

Instead of using .SMTP_SSL() to create a connection that is secure from the outset, we can create an unsecured SMTP connection and encrypt it using .starttls().

To do this, create an instance of smtplib.SMTP, which encapsulates an SMTP connection and allows you access to its methods. I recommend defining your SMTP server and port at the beginning of your script to configure them easily.

The code snippet below uses the construction server = SMTP(), rather than the format with SMTP() as server: which we used in the previous example.

To make sure that your code doesn’t crash when something goes wrong, put your main code in a try block, and let an except block print any error messages to stdout:

```
import smtplib, ssl

smtp_server = "smtp.gmail.com"
port = 587  # For starttls
sender_email = "my@gmail.com"
password = input("Type your password and press enter: ")

# Create a secure SSL context
context = ssl.create_default_context()

# Try to log in to server and send email
try:
    server = smtplib.SMTP(smtp_server,port)
    server.ehlo() # Can be omitted
    server.starttls(context=context) # Secure the connection
    server.ehlo() # Can be omitted
    server.login(sender_email, password)
    # TODO: Send email here
except Exception as e:
    # Print any error messages to stdout
    print(e)
finally:
    server.quit()
```

To identify yourself to the server, .helo() (SMTP) or .ehlo() (ESMTP) should be called after creating an .SMTP() object, and again after .starttls(). This function is implicitly called by .starttls() and .sendmail() if needed, so unless you want to check the SMTP service extensions of the server, it is not necessary to use .helo() or .ehlo() explicitly

Sending Your Plain-text Email

After you initiated a secure SMTP connection using either of the above methods, you can send your email using .sendmail(), which pretty much does what it says on the tin:

```
server.sendmail(sender_email, receiver_email, message)
```

I recommend defining the email addresses and message content at the top of your script, after the imports, so you can change them easily:

```
sender_email = "my@gmail.com"
receiver_email = "your@gmail.com"
message = """\
Subject: Hi there

This message is sent from Python."""

# Send email here
```

The message string starts with "Subject: Hi there" followed by two newlines (\n). This ensures Hi there shows up as the subject of the email, and the text following the newlines will be treated as the message body.

The code example below sends a plain-text email using SMTP_SSL():

```
import smtplib, ssl

port = 465  # For SSL
smtp_server = "smtp.gmail.com"
sender_email = "my@gmail.com"  # Enter your address
receiver_email = "your@gmail.com"  # Enter receiver address
password = input("Type your password and press enter: ")
message = """\
Subject: Hi there

This message is sent from Python."""

context = ssl.create_default_context()
with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, message)
```

For comparison, here is a code example that sends a plain-text email over an SMTP connection secured with .starttls(). The server.ehlo() lines may be omitted, as they are called implicitly by .starttls() and .sendmail(), if required:

```
import smtplib, ssl

port = 587  # For starttls
smtp_server = "smtp.gmail.com"
sender_email = "my@gmail.com"
receiver_email = "your@gmail.com"
password = input("Type your password and press enter:")
message = """\
Subject: Hi there

This message is sent from Python."""

context = ssl.create_default_context()
with smtplib.SMTP(smtp_server, port) as server:
    server.ehlo()  # Can be omitted
    server.starttls(context=context)
    server.ehlo()  # Can be omitted
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, message)
```

Sending Fancy Emails

Python’s built-in email package allows you to structure more fancy emails, which can then be transferred with smtplib as you have done already. Below, you’ll learn how use the email package to send emails with HTML content and attachments.

Including HTML Content

If you want to format the text in your email (**bold**, *italics*, and so on), or if you want to add any images, hyperlinks, or responsive content, then HTML comes in very handy. Today’s most common type of email is the MIME (Multipurpose Internet Mail Extensions) Multipart email, combining HTML and plain-text. MIME messages are handled by
Python’s email.mime module. For a detailed description, check the documentation.
As not all email clients display HTML content by default, and some people choose only to receive plain-text emails for security reasons, it is important to include a plain-text alternative for HTML messages. As the email client will render the last multipart attachment first, make sure to add the HTML message after the plain-text version.
In the example below, our MIMEText() objects will contain the HTML and plain-text versions of our message, and
the MIMEMultipart("alternative") instance combines these into a single message with two alternative rendering options:

```
import smtplib, ssl
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

sender_email = "my@gmail.com"
receiver_email = "your@gmail.com"
password = input("Type your password and press enter:")

message = MIMEMultipart("alternative")
message["Subject"] = "multipart test"
message["From"] = sender_email
message["To"] = receiver_email

# Create the plain-text and HTML version of your message
text = """\
Hi,
How are you?
Real Python has many great tutorials:
www.realpython.com"""
```

```
html = """\
<html>
  <body>
    <p>Hi,<br>
       How are you?<br>
       <a href="http://www.realpython.com">Real Python</a>
       has many great tutorials.
    </p>
  </body>
</html>
"""

# Turn these into plain/html MIMEText objects
part1 = MIMEText(text, "plain")
part2 = MIMEText(html, "html")

# Add HTML/plain-text parts to MIMEMultipart message
# The email client will try to render the last part first
message.attach(part1)
message.attach(part2)

# Create secure connection with server and send email
context = ssl.create_default_context()
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as
server:
    server.login(sender_email, password)
    server.sendmail(
        sender_email, receiver_email, message.as_string()
    )
```
In this example, you first define the plain-text and HTML message as string literals, and then store them as plain/html MIMEText objects. These can then be added in this order to the MIMEMultipart("alternative") message and sent through your secure connection with the email server. Remember to add the HTML message after the plain-text alternative, as email clients will try to render the last subpart first.

Adding Attachments Using the email Package

In order to send binary files to an email server that is designed to work with textual data, they need to be encoded before transport. This is most commonly done using base64, which encodes binary data into printable ASCII characters.

The code example below shows how to send an email with a PDF file as an attachment:

```
import email, smtplib, ssl

from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

subject = "An email with attachment from Python"
body = "This is an email with attachment sent from Python"
sender_email = "my@gmail.com"
receiver_email = "your@gmail.com"
password = input("Type your password and press enter:")

# Create a multipart message and set headers
message = MIMEMultipart()
message["From"] = sender_email
message["To"] = receiver_email
message["Subject"] = subject
message["Bcc"] = receiver_email  # Recommended for mass emails

# Add body to email
message.attach(MIMEText(body, "plain"))

filename = "document.pdf"  # In same directory as script

# Open PDF file in binary mode
with open(filename, "rb") as attachment:
    # Add file as application/octet-stream
    # Email client can usually download this automatically as attachment
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())

# Encode file in ASCII characters to send by email
encoders.encode_base64(part)

# Add header as key/value pair to attachment part
part.add_header(
    "Content-Disposition",
    f"attachment; filename= {filename}",
)

# Add attachment to message and convert message to string
message.attach(part)
text = message.as_string()

# Log in to server using secure context and send email
context = ssl.create_default_context()
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, text)
```

The MIMEultipart() message accepts parameters in the form of RFC5233-style key/value pairs, which are stored in a dictionary and passed to the .add_header method of the Message base class.
Check out the documentation for Python’s email.mime module to learn more about using MIME classes

Sending Multiple Personalized Emails

Imagine you want to send emails to members of your organization, to remind them to pay their contribution fees. Or maybe you want to send students in your class personalized emails with the grades for their recent assignment. These tasks are a breeze in Python.
Make a CSV File With Relevant Personal Info

An easy starting point for sending multiple personalized emails is to create a CSV (comma-separated values) file that contains all the required personal information. (Make sure not to share other people's private information without their consent.) A CSV file can be thought of as a simple table, where the first line often contains the column headers.
Below are the contents of the file contacts_file.csv, which I saved in the same folder as my Python code. It contains the names, addresses, and grades for a set of fictional people. I
used my+modifier@gmail.com constructions to make sure all emails end up in my own inbox, which in this example is my@gmail.com:

```
name,email,grade
Ron Obvious,my+ovious@gmail.com,B+
Killer Rabbit of Caerbannog,my+rabbit@gmail.com,A
Brian Cohen,my+brian@gmail.com,C
```

When creating a CSV file, make sure to separate your values by a comma, without any surrounding whitespaces.

Loop Over Rows to Send Multiple Emails

The code example below shows you how to open a CSV file and loop over its lines of content (skipping the header row). To make sure that the code works correctly before you send emails to all your contacts, I've
printed Sending email to ... for each contact, which we can later replace with functionality that actually sends out emails:

```
import csv

with open("contacts_file.csv") as file:
    reader = csv.reader(file)
    next(reader)  # Skip header row
    for name, email, grade in reader:
        print(f"Sending email to {name}")
        # Send email here
```

In the example above, using with open(filename) as file:makes sure that your file closes at the end of the code block. csv.reader() makes it easy to read a CSV file line by line and extract its values. The next(reader) line skips the header row, so that the following line for name, email, grade in

reader: splits subsequent rows at each comma, and stores the resulting values in the strings name, email and grade for the current contact.
If the values in your CSV file contain whitespaces on either or both sides, you can remove them using the .strip() method.

Personalized Content

You can put personalized content in a message by using str.format() to fill in curly-bracket placeholders. For example, "hi {name}, you {result} your assignment".format(name="John", result="passed") will give you "hi John, you passed your assignment".
As of Python 3.6, string formatting can be done more elegantly using f-strings, but these require the placeholders to be defined before the f-string itself. In order to define the email message at the beginning of the script, and fill in placeholders for each contact when looping over the CSV file, the older .format() method is used.
With this in mind, you can set up a general message body, with placeholders that can be tailored to individuals.
Code Example
The following code example lets you send personalized emails to multiple contacts. It loops over a CSV file with name,email,grade for each contact, as in the example above.
The general message is defined in the beginning of the script, and for each contact in the CSV file its {name} and {grade} placeholders are filled in, and a personalized email is sent out through a secure connection with the Gmail server, as you saw before:

```
import csv, smtplib, ssl

message = """Subject: Your grade

Hi {name}, your grade is {grade}"""
from_address = "my@gmail.com"
password = input("Type your password and press enter: ")

context = ssl.create_default_context()
```

```
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as
server:
    server.login(from_address, password)
    with open("contacts_file.csv") as file:
        reader = csv.reader(file)
        next(reader)  # Skip header row
        for name, email, grade in reader:
            server.sendmail(
                from_address,
                email,
                message.format(name=name,grade=grade),
            )
```

Yagmail

There are multiple libraries designed to make sending emails easier, such as Envelopes, Flanker and Yagmail. Yagmail is designed to work specifically with Gmail, and it greatly simplifies the process of sending emails through a friendly API, as you can see in the code example below:

```
import yagmail

receiver = "your@gmail.com"
body = "Hello there from Yagmail"
filename = "document.pdf"

yag = yagmail.SMTP("my@gmail.com")
yag.send(
    to=receiver,
    subject="Yagmail test with attachment",
    contents=body,
    attachments=filename,
)
```

This code example sends an email with a PDF attachment in a fraction of the lines needed for our example using email and smtplib.

When setting up Yagmail, you can add your Gmail validations to the keyring of your OS, as described in the documentation. If you don't do this, Yagmail will prompt you to enter your password when required and store it in the keyring automatically.

Transactional Email Services

If you plan to send a large volume of emails, want to see email statistics, and want to ensure reliable delivery, it may be worth looking into transactional email services. Although all of the following services have paid plans for sending large volumes of emails, they also come with a free plan so you can try them out. Some of these free plans are valid indefinitely and may be sufficient for your email needs.

Below is an overview of the free plans for some of the major transactional email services. Clicking on the provider name will take you to the pricing section of their website.

Provider	Free plan
Sendgrid	40,000 emails for your first 30 days, then 100/day
Sendinblue	300 emails/day
Mailgun	First 10,000 emails free
Mailjet	200 emails/day
Amazon SES	62,000 emails/month

You can run a Google search to see which provider best fits your needs, or just try out a few of the free plans to see which API you like working with most.

Sendgrid Code Example

Here's a code example for sending emails with Sendgrid to give you a flavor of how to use a transactional email service with Python:

```
import os
import sendgrid
from sendgrid.helpers.mail import Content, Email, Mail

sg = sendgrid.SendGridAPIClient(
    apikey=os.environ.get("SENDGRID_API_KEY")
)
```

```
from_email = Email("my@gmail.com")
to_email = Email("your@gmail.com")
subject = "A test email from Sendgrid"
content = Content(
   "text/plain", "Here's a test email sent through Python"
)
mail = Mail(from_email, subject, to_email, content)
response = sg.client.mail.send.post(request_body=mail.get())

# The statements below can be included for debugging purposes
print(response.status_code)
print(response.body)
print(response.headers)
```

To run this code, you must first:
Sign up for a (free) Sendgrid account
Request an API key for user validation
Add your API key by typing setx SENDGRID_API_KEY "YOUR_API_KEY" in Command Prompt (to store this API key permanently) or set SENDGRID_API_KEY YOUR_API_KEY to store it only for the current client session
More information on how to set up Sendgrid for Mac and Windows can be found in the repository's README on Github.

Conclusion

You can now start a secure SMTP connection and send multiple personalized emails to the people in your contacts list!
You've learned how to send an HTML email with a plain-text alternative and attach files to your emails. The Yagmail package simplifies all these tasks when you're using a Gmail account. If you plan to send large volumes of email, it is worth looking into transactional email services.

1.Can you email multiple people?
Yes, we can send multiple email. By using smtplib package.

Program:

```
 Import
 smtplib

         from string import Template

         from email.mime.multipart import MIMEMultipart
         from email.mime.text import MIMEText

         MY_ADDRESS = 'my_address@example.comm'
         PASSWORD = 'mypassword'

         def get_contacts(filename):
             """
             Return two lists names, emails containing names and email
         addresses
             read from a file specified by filename.
             """

             names = []
             emails = []
             with open(filename, mode='r', encoding='utf-8') as
         contacts_file:
                 for a_contact in contacts_file:
                     names.append(a_contact.split()[0])
                     emails.append(a_contact.split()[1])
             return names, emails

         def read_template(filename):
             """
             Returns a Template object comprising the contents of the
             file specified by filename.
             """

             with open(filename, 'r', encoding='utf-8') as template_file:
```

```
        template_file_content = template_file.read()
    return Template(template_file_content)

def main():
    names, emails = get_contacts('mycontacts.txt') # read
contacts
    message_template = read_template('message.txt')

    # set up the SMTP server
    s = smtplib.SMTP(host='your_host_address_here',
port=your_port_here)
    s.starttls()
    s.login(MY_ADDRESS, PASSWORD)

    # For each contact, send the email:
    for name, email in zip(names, emails):
        msg = MIMEMultipart()       # create a message

        # add in the actual person name to the message template
        message =
message_template.substitute(PERSON_NAME=name.title())

        # Prints out the message body for our sake
        print(message)

        # setup the parameters of the message
        msg['From']=MY_ADDRESS
        msg['To']=email
        msg['Subject']="This is TEST"

        # add in the message body
        msg.attach(MIMEText(message, 'plain'))

        # send the message via the server set up earlier.
        s.send_message(msg)
```

```
            del msg

        # Terminate the SMTP session and close the connection
        s.quit()

    if __name__ == '__main__':
        main()
```

Could you pull the list of people to email from an external file?

Yes, we can pull the list of people to email from an external file. By using following python program.

Program:

```
import re

fileToRead = 'readText.txt'
fileToWrite = 'emailExtracted.txt'


delimiterInFile = [',', ';']

def validateEmail(strEmail):
    # .* Zero or more characters of any type.
    if re.match("(.*)@(.*).(.*)", strEmail):
        return True
    return False

def writeFile(listData):
    file = open(fileToWrite, 'w+')
    strData = ""
    for item in listData:
        strData = strData+item+'\n'
    file.write(strData)

listEmail = []
```

```
file = open(fileToRead, 'r')
listLine = file.readlines()
for itemLine in listLine:
    item =str(itemLine)
    for delimeter in delimiterInFile:
        item = item.replace(str(delimeter),' ')

    wordList = item.split()
    for word in wordList:
        if(validateEmail(word)):
            listEmail.append(word)

if listEmail:
    uniqEmail = set(listEmail)
    print(len(uniqEmail),"emails collected!")
    writeFile(uniqEmail)
else:
    print("No email found.")
```

How can you personalize the email for the recipient?

Prepare a .csv document: I prepared a .csv document including the information which needs to be modified for each email. I wanted to change the names of the recipients after the word “Dear”: Dear Anton, Dear Alex, etc. So, I had two columns in the .csv file: The Names, The Email addresses.
Example:
Anton, xyz@gmail.com
Alex, abc@gmail.com
Write the content of the email: I wrote the content of my email and saved it in .txt format. The content looks like this:
Dear {},
The rest of the email.
After preparing the .csv document (“email.csv”), and the email content (“message.txt”), I opened up a .py file (Python files have .py extension). For

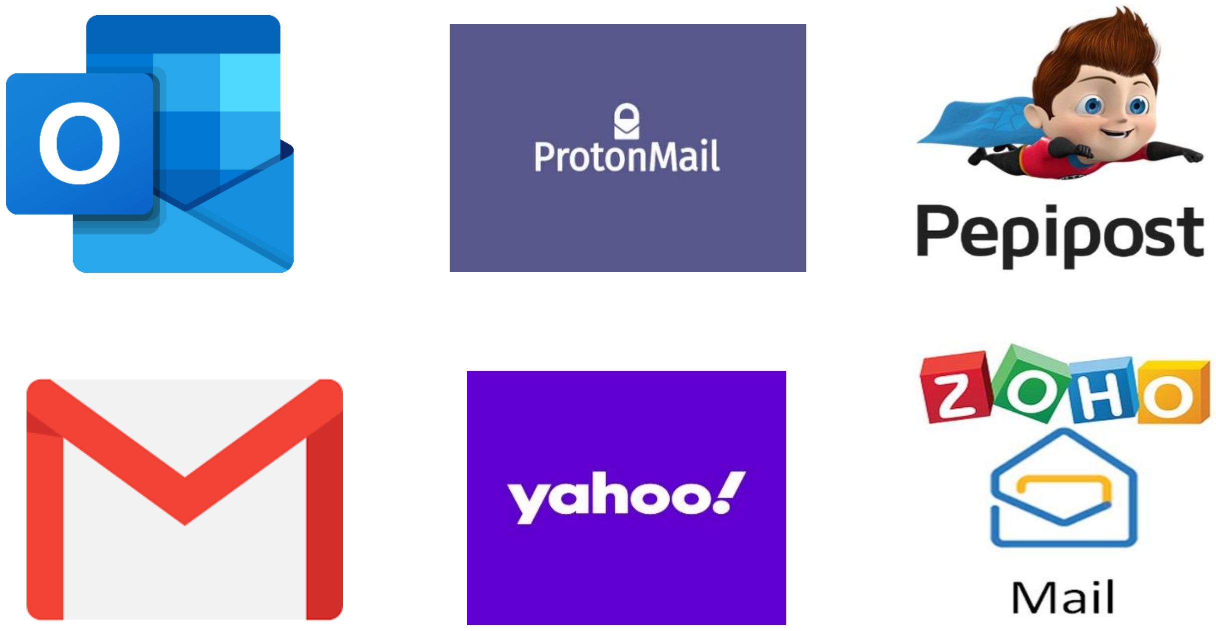
the sake of simplicity, it was essential that all three (.csv, .txt, and .py) files are in the same directory. Then, I needed to install and import two modules from Python: csv and smtp lib. The csv module was necessary to read the .csv files. The smtp lib, [SMTP: “Simple Mail Transfer Protocol“], is used for sending and routing emails between servers. These are the first two lines of the code which you see below. On the fourth line, you will see “MIME” which stands for “Multipurpose Internet Mail Extensions”. With MIME, I was specifying the content type, which in this case is text.

Discussion:

1.What could you do to ensure privacy when sending email?
Answer:
Emails are a vital source of communication across the internet. They have numerous advantages over other sources of communication.

- Emails are delivered extremely fast when compared to traditional post.
- Emails can be sent 24 hours a day, 365 days a year.
- Webmail means emails can be sent and received from any computer, anywhere in the world, that has an internet connection.
- Cheap - when using broadband, each email sent is effectively free. Dial-up users are charged at local call rates but it only takes a few seconds (for conventional email, eg text only) to send an email.
- Email allows for easy referencing. Messages that have been sent and received can stored, and searched through safely and easily. It is a lot easier to go through old email messages rather than old notes written on paper.
- Email allows for mass sending of messages. An effective medium to utilize to get your message out there, you can send one particular message to several recipients all at once.
- Email allows for instant access of information and files. You can opt to send yourself files and keep messages so that you have a paper trail

of conversations and interactions you have online just in case you may need them in the future.



Largest E-mail providers of the world.

Since this a technology that most people are dependent on, there must be certain measures to prevent them from exploitation. The consumers of email services must be ensured with proper privacy and integrity of their data.

- **Use two-factor authentication**

The basic principle of two-factor authentication is simple: combine something you know with something you have. By enabling two-factor authentication (or two-step verification), you aren't putting all of your faith in a password. For Gmail, setting up two-step verification is as simple as clicking a button and entering in your mobile number. Now that you've enabled two-factor authentication, a hacker with your password is out of luck — unless they've also managed to steal your cell phone.

- **Limit forwarding**

When we're sent a message we want to share, we often click "Forward" without thinking about the consequences. Where is the message going? Who will see it? Where will it be stored? If your email is hosted on a corporate server, it is likely there are certain security measures in place to protect any sensitive information contained in your private email. When someone forwards an internal email to a recipient outside of your company, however, you are exposing that data (as well as any other emails in the forwarded chain) to potentially unsecured, unencrypted servers.

- **Set expiration dates on your messages**

While some of us can't stand a messy inbox, the average user doesn't bother cleaning up their private email. Any sensitive information you send to a client could very well be sitting there months later. At that point, you no longer control the fate of your data. Luckily, Virtru lets you set an expiration date on your email, so that after a certain date, it will no longer be readable by the recipient (or anyone else, for that matter).

- **Understand your service provider's TOS**

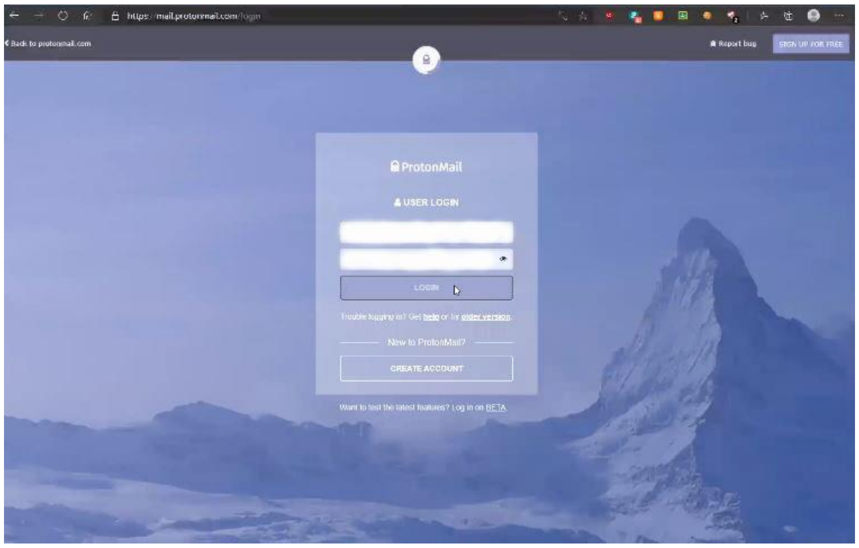
Your email provider's terms of service can tell you a lot more than their media interviews and advertisements can. For starters, it'll let you know what kind of security they are offering you. Are they encrypting messages on their server? Do they have protections against brute-force attacks? Is there any guarantee that your data is being protected? Take Google for example, which openly passes private email through automated scanning. After reading your email provider's TOS, you'll likely realize that keeping your private email secure isn't their first priority — that's entirely up to you.

- **Encrypt your email**

The best way to keep your private email away from prying eyes and hackers is to use encryption. Encryption protects your private email by

jumbling up your messages, making them impossible to decipher unless you explicitly authorize someone to read them. Even if your inbox is compromised, the contents of your message will be unreadable. As an added bonus, if your email ends up getting stored on a server outside of your control, you still have power over who gets to see it — and you can revoke that permission at any time.
Virtru works with the email service you're already using to provide true client-side email encryption for your messages and attachments.
Another application that can be used to encrypt your email is Proton Mail. **ProtonMail** is an end-to-end encrypted email service that uses client-side encryption to protect email content and user data before they are sent to ProtonMail servers, unlike other common email providers such as Gmail and Outlook.com. The service can be accessed through a webmail client, the Tor network, or dedicated iOS and Android apps.

Here is an image that describes how protonmail can be used to encrypt your emails:

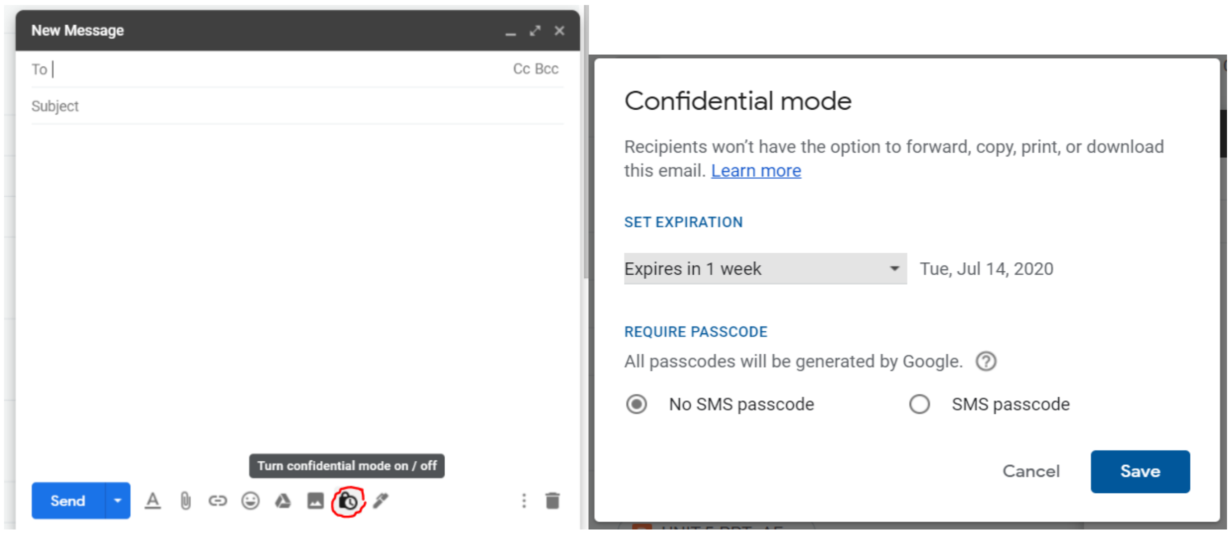

https://mail.protonmail.com/login
Back to protonmail.com
Report bug
SIGN UP FOR FREE
ProtonMail
USER LOGIN
LOGIN
Trouble logging in? Get help or try older version.
New to ProtonMail?
CREATE ACCOUNT
Want to test the latest features? Log in on BETA

2. What expectation of privacy do you have when sending email?
Answer:

- Sensitive data must be protected. Communicating with traditional email is not secure. It only takes one email-related security breach to cause inadvertent disclosure of confidential or sensitive messages and files. With identity theft, hackers and private communication taking place online, it's a risk that shouldn't be take.
- Sensitive email message requires authentication of both parties. Authentication is necessary to ensure that only the intended recipient can take delivery of the message.
- There is no "proof of delivery" with traditional email. What proof, other than your "sent" mailbox records, do we have that we transmitted an important file? Some systems offer a "read receipt," but a recipient can choose to deny the receipt.
- You can't always send large, confidential files. It's important to business to be able to send large files easily. Some resort to saving a file to disk and sending it by courier. It is not safe and we do not know who will retrieve the package.
- You can't control a file once you hit "send" with traditional email. We do not know who our recipient is sharing confidential files and information with once we send it through traditional email. We don't want confidential messages ending up in the wrong hands.

3. If you had a secret message to send, how would you do it?
Answer:

- In Gmail, secret/confidential messages can be sent by turning the confidential mode on. It does not allow the recipients to forward, copy, print, or download that particular email.
  If you choose "No SMS passcode," recipients using the Gmail app will be able to open it directly. If you choose "SMS passcode," recipients will get a passcode by text message.



Confidential mode

Recipients won't have the option to forward, copy, print, or download this email. Learn more

SET EXPIRATION

Expires in 1 week Tue, Jul 14, 2020

REQUIRE PASSCODE

All passcodes will be generated by Google.

[✓ No SMS passcode] SMS passcode

Cancel Save

- There are sites that will send encrypted emails that are more secure than regular email. The recipient will need to know the password for most of them. Lockbin.com is one of the most secure email sites.
- If you want to send an anonymous email that doesn't tell who the sender is, there are several sites you can use. You can use a fake name with Lockbin. These sites will also send emails that won't reveal the sender: silentsender.com, privnote.com, send-email.org

4. How could you automate emailing many people?

Answer:

I) Mail Merge Method:

Mail merges are one of the quickest ways to customize documents like emails, newsletters, and other personalized messages. A mail merge lets you create personalized documents that automatically vary on a recipient-by-recipient basis.

- In a blank Microsoft Word document, click on the **Mailings** tab, and in the **Start Mail Merge** group, click **Start Mail Merge**.
- Click **Step-by-Step Mail Merge Wizard**. Select your document type.
- Select the starting document. Then Select recipients.
- Write the letter and add custom fields.

- Press Enter button on your keyboard and click **Greeting line...** to enter a greeting.

- In the **Insert Greeting Line** dialog box, choose the greeting line format by clicking the drop-down arrows and selecting the options of your choice, and then click **OK**.
- Note that the address block and greeting line are surrounded by chevrons (« »). Write a short letter and click **Next: Preview your letters**.
- Preview your letter and click **Next: Complete the merge**.

II) The BCC Method:

- Open your Gmail account.
- Click on the Compose box to type the email you would like to send to multiple recipients from your Gmail address.
- After writing the email, click on the BCC option besides the CC option.
- Once you select the BCC option, type in the email addresses of the people you want to email or you can select one of the Gmail groups.

Assignment Questions:

1.Why do email services "read" your email? What is their goal?
Answer:
"*If you're not the customer, you're the product.*" This cardinal rule of the Internet has been proven true over and over again, and email is no exception. Luckily, not all providers are targeting you with the same intensity, and some are actually quite private. Some, like Google, mostly scan emails to help with their AI functions, while others, like Yahoo!, are digging for information they can add to your advertising profile. But which email providers are rummaging through your inbox, and which ones can you trust?

Gmail

Google does scan your email for security threats, high priority notifications, calendar events, and some of the other features you might enjoy, like Smart Compose. But you can also go to "Settings -> General" and turn it off.

A bigger concern is that Google allows certain third-party app developers access to users emails, meaning that the contents could be mined for advertising data or even directly read by humans.

Verdict: Surprisingly private, but it's still Google.

Outlook

Microsoft's email service, like Outlook, advertises for revenue, but doesn't use your emails to target those ads. They do scan for security threats, like phishing links and possible malware attachments, but no one is reading your emails at any stage of the process.

Outlook also supports third-party apps, though, and while the developer access policy isn't as open as Google's, some of them may still scan your emails and send it off somewhere. This is most likely with business users who have installed third-party productivity apps.

Verdict: Not reading your emails.

Yahoo!

Yahoo is also one of the only providers that scans your email content. Their algorithm looks at receipts, travel information, and other emails from organizations that it can use to serve you ads. The bright side is that this only applies to emails you get from companies.

You can opt out by going to your privacy controls page and toggling the interest-based ads option. It is turned on by default, however, so it's up to you to turn it off.

Verdict: Reading your emails for aggressive ad targeting.

Nutshell:

Most of the email providers scan your emails for two main purposes:

- To check for any malware or a phishing attack.
- To display advertisement content related to your interest.

But there are certain measures that these email service providers offer you to increase your privacy. Almost all of them provide some steps so that you can minimize the amount of content that they seek through your email.

2. How does PGP secure email differently than GMail?
Answer:
PGP secure email differently than GMail in the following ways:

- Gmail has unlimited access to all of the user's intimate communications but PGP's approach to data privacy provides more security because its **encryption is such that nobody except the sender/receiver can read the messages in their mailbox** .In fact, not even PGP mail service has the ability to read users messages.

- One more benefit of the **end-to-end** and **zero-access encryption** is to ensure that users have complete ownership of their data. And PGP mail service don't have the ability to read it or sell it, even if we wanted to but that is not the case in G-Mail.

- PGP email user uses **zero-access encryption**, which means it is technically impossible to decrypt user messages. Zero-Access Encryption applies to all messages in user's mailbox, even messages which did not come from other PGP users.

i. This provides stronger security compared to G-Mail because even if PGP email service were somehow breached, user messages remain secure because PGP mail service only stores encrypted messages.

ii. The use of Zero-Access Encryption, therefore, adds a strong layer of resiliency against catastrophic data breaches.

iii. Compared to GMail, PGP mail service is a much smaller target, and there is less risk that a vulnerability in another service breaches your email account.

- Google records literally every action done by its users. This includes your IP address, every search that you do, which emails you open, which websites you visit, and much more. PGP mail service takes the opposite approach and by default, does not monitor or record user activity, not even IP addresses.

- In addition to the security of emails at rest, one also needs to consider the security of emails in transit. Both PGP Mail service and Gmail provide extra protection by using TLS encryption whenever possible while communicating with external email providers. However, PGP's goes one step further by also supporting **end-to-end encryption**.

i. In simple terms, end-to-end encryption means that messages are encrypted on the sender's device (before it even leaves their computer or mobile phone), and can only be decrypted by the recipient on their device.

ii. This means that no third party which transmits or intercepts the email between the sender and recipient (i.e. internet service providers, the NSA, or even Pgp mail service as the mail server operator) can decrypt and view the message.

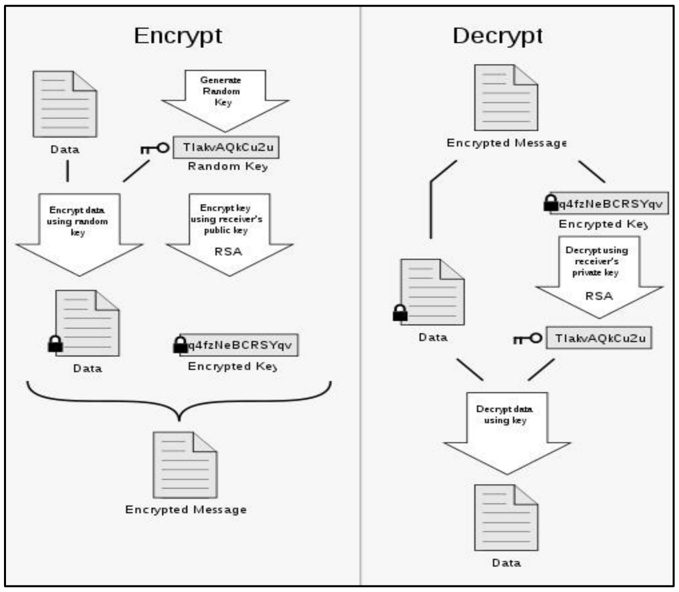
iii. For an **enterprise using PGP Mail service** for their email hosting, it means all communications between employees are automatically protected with end-to-end encryption.

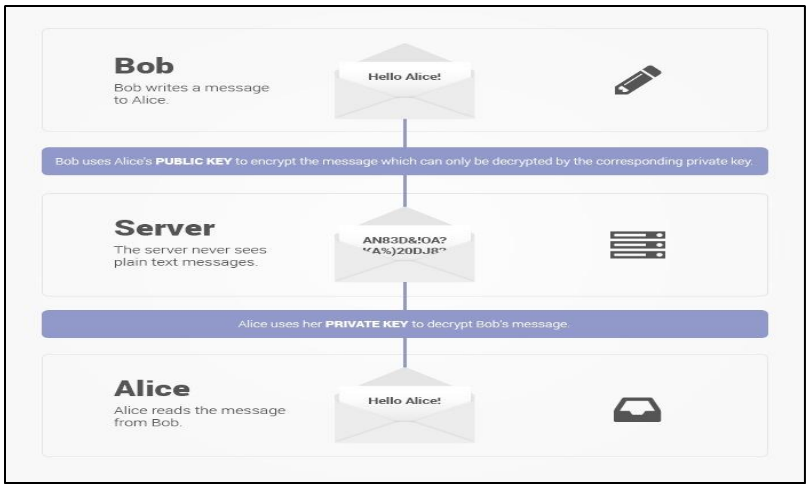
- PGP Mail service uses **Secure Remote Password** in order to protect user credentials. This makes it difficult to conduct a brute force attack to obtain user credentials, even if the attacker has control over the victim's network.

- Both Gmail and PGP mail service support two-factor authentication (2FA), which provides an additional layer of security by requiring that an unique code be entered on each login (the code is usually generated on a separate hardware device). However, PGP Mail service goes a step further by only using strong 2FA methods and disallowing weaker methods such as **2FA over SMS**.

Conclusion

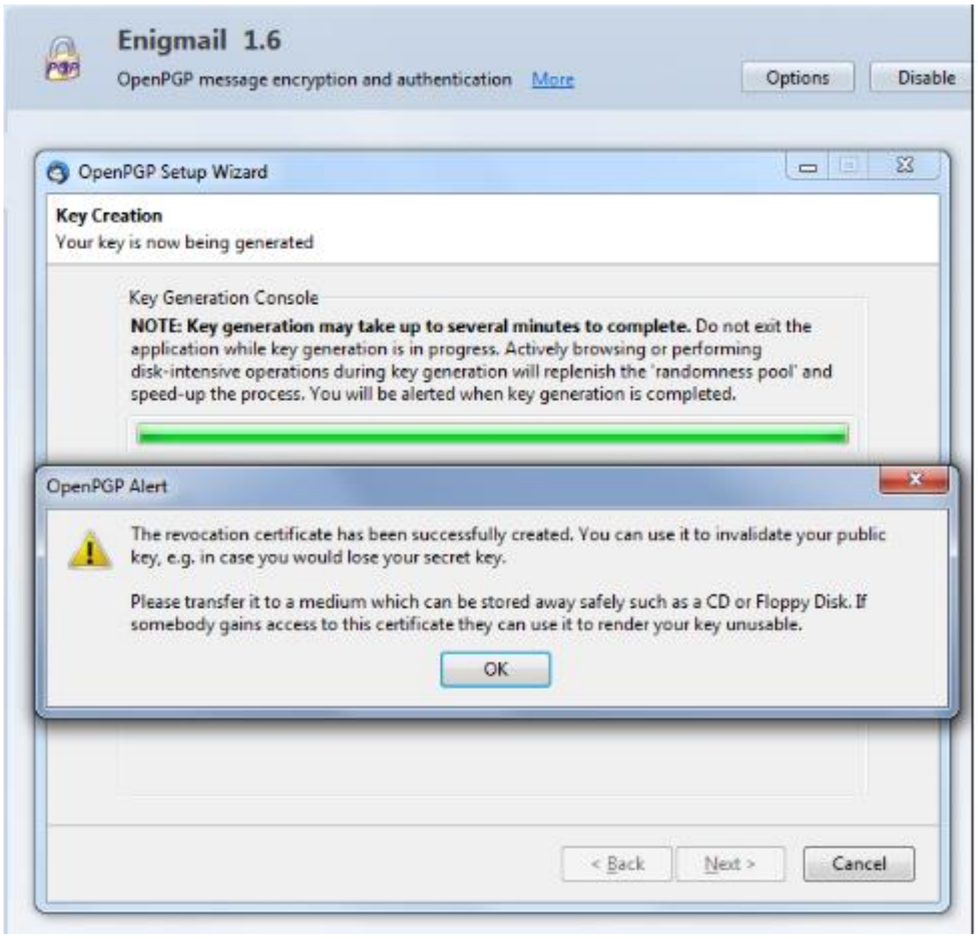
Both Gmail and PGP mail service provide email accounts, but that’s where the similarities end. In terms of technology, legal protection, and position on privacy rights, the two services diverge widely. If you just want an email account, either service will meet your needs. **If email security, and in particular <u>privacy</u> is important to you, then you should consider <u>PGP mail service as a Gmail alternative</u>.**




Bob
Bob writes a message to Alice.
Hello Alice!
Bob uses Alice's PUBLIC KEY to encrypt the message which can only be decrypted by the corresponding private key.
Server
The server never sees plain text messages.
AN83D&!OA?
‘A%)20DJ8
Alice uses her PRIVATE KEY to decrypt Bob's message.
Alice
Alice reads the message from Bob.
Hello Alice!

3. Why don't people use services like PGP more often?
Answer:

- The problems :
  - You need to understand the way public-private key encryption works, generate a key pair, and provide your public key to the person you want to communicate with.
  - Other people you want to communicate with also need to understand and do all these things.
  - Both people need to keep their private keys safe so they don’t become compromised or lost — in which case you’d lose access to the emails. You also need to keep your revocation certificate as it can invalidate your public key if you ever lose your private key.
  - Your private keys must be encrypted with a secure passphrase you have to remember, which is separate from your email account password.
  - You need to ensure you’re both using the same email encryption standard, whether PGP or S/MIME or some other standard.
  - You need to use a third-party solution — either a browser extension, smart phone app, or email client plugin. If you opt for the best-supported option, you’ll need to install an email client, an extension, and an encryption software package separately.
  - You need a mix of different smart phone apps and desktop solutions if you want to access your emails on all your devices.
  - Even if you do all of these things, people will still be able to see who you’re communicating with and what the subjects of your messages are.



Key Creation
Your key is now being generated

Key Generation Console
NOTE: Key generation may take up to several minutes to complete. Do not exit the application while key generation is in progress. Actively browsing or performing disk-intensive operations during key generation will replenish the 'randomness pool' and speed-up the process. You will be alerted when key generation is completed.

OpenPGP Alert

The revocation certificate has been successfully created. You can use it to invalidate your public key, e.g. in case you would lose your secret key.

Please transfer it to a medium which can be stored away safely such as a CD or Floppy Disk. If somebody gains access to this certificate they can use it to render your key unusable.

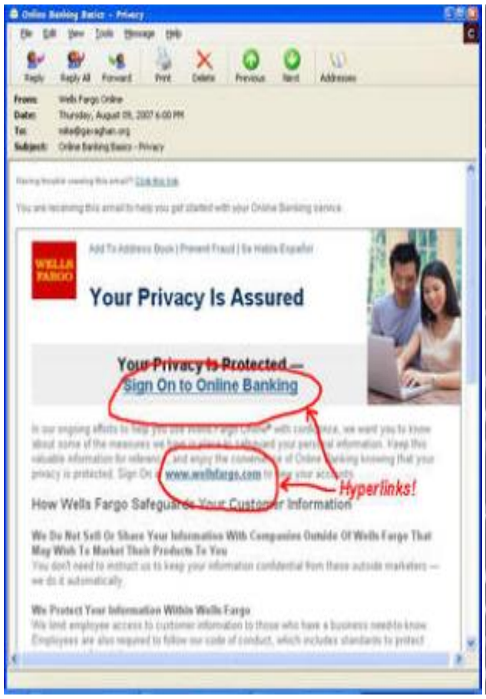
With all this complexity — and so much information leaking out even if you use PGP properly — it's no wonder encrypted email is used so little. It's also no surprise that people choose to use services like Lava bit that seem to be a convenient way of making encryption easy-to-use, but are actually much less reliable than encrypting your own emails.

4. What is phishing?
Answer:

❖ Phishing is a cybercrime in which a target or targets are contacted by email, telephone or text message by someone posing as a legitimate institution to lure individuals into providing sensitive data such as personally identifiable information, banking and credit card details, and passwords.

❖ The information is then used to access important accounts and can result in identity theft and financial loss.

❖ There are a number of different techniques used to obtain personal information from users like: Spear phishing, Session Hijacking , Email / Spam , Content Injection , Web Based Delivery , Link manipulation ,etc.



Online Banking Basics - Privacy

From: Wells Fargo Online
Date: Thursday, August 09, 2007 6:00 PM
To: mike@geraghan.org
Subject: Online Banking Basics - Privacy

You are receiving this email to help you get started with your Online Banking service.

Add To Address Book | Prevent Fraud | En Habla Español

Your Privacy Is Assured

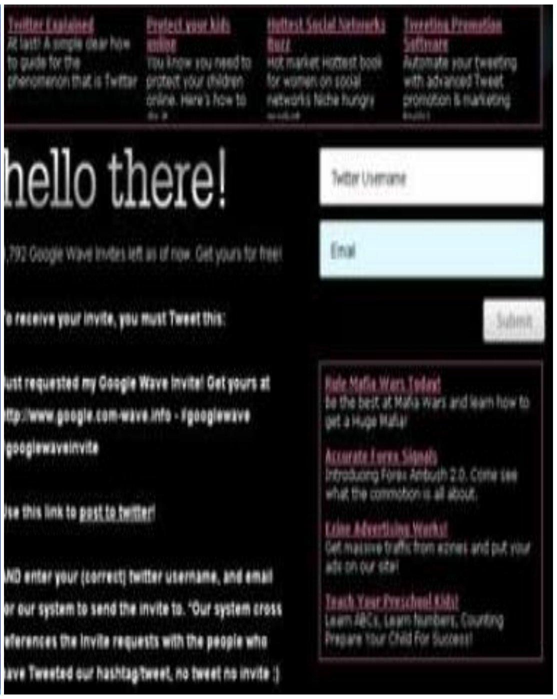
Your Privacy Is Protected — Sign On to Online Banking

[illegible] www.wellsfargo.com [illegible]

How Wells Fargo Safeguards Your Customer Information

We Do Not Sell Or Share Your Information With Companies Outside Of Wells Fargo That May Wish To Market Their Products To You
You don't need to instruct us to keep your information confidential from these outside marketers — we do it automatically.

We Protect Your Information Within Wells Fargo
We limit employee access to customer information to those who have a business need-to-know. Employees are also required to follow our code of conduct, which includes standards to protect



Twitter Explained
At last! A simple clear how to guide for the phenomenon that is Twitter

Protect your kids online
You know you need to protect your children online. Here's how to

Hottest Social Networks Buzz
Hot market Hottest book for women on social networks Niche hungry

Tweeting Promotion Software
Automate your tweeting with advanced Tweet promotion & marketing

hello there!

1,792 Google Wave invites left as of now. Get yours for free!

To receive your invite, you must Tweet this:

Just requested my Google Wave invite! Get yours at http://www.google.com-wave.info - #googlewave #googlewaveinvite

Use this link to post to twitter!

AND enter your (correct) twitter username, and email for our system to send the invite to. "Our system cross references the invite requests with the people who have Tweeted our hashtag/tweet, no tweet no invite :)

Ride Mafia Wars Today!
Be the best at Mafia Wars and learn how to get a Huge Mafia!

Accurate Forex Signals
Introducing Forex Ambush 2.0. Come see what the commotion is all about.

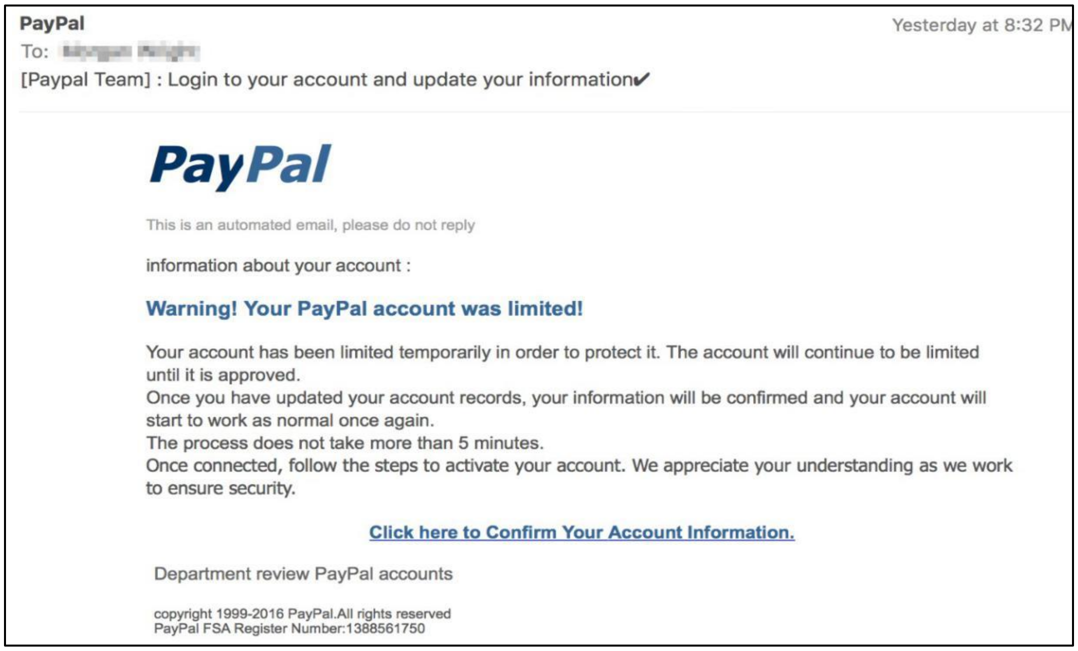
Ezine Advertising Works!
Get massive traffic from ezines and put your ads on our site!

Teach Your Preschool Kids!
Learn ABCs, Learn Numbers, Counting Prepare Your Child For Success!

5.What is spear-phishing?
Answer:
Spear-phishing is a targeted attempt to steal sensitive information such as account credentials or financial information from a specific victim, often for malicious reasons, cyber criminals may also intend to install malware on a targeted user's computer. This is achieved by acquiring personal details on the victim such as their friends, hometown, employer, locations they frequent, and what they have recently bought online. The attackers then disguise themselves as a trustworthy friend or entity to acquire sensitive information, typically through email or other online messaging. This is the most successful form of acquiring confidential information on the internet, accounting for **91% of attacks.**

Spear phishing emails are quite hard to detect because they are so targeted .They look like normal business emails, so it's really hard for spam detection systems to realize it's not a genuine email.This is how it works: An email arrives, apparently from a trustworthy source, but instead it leads the unknowing recipient to a bogus website full of malware. These emails often use clever tactics to get victims' attention.

Spear-phishing attacks target a specific victim, and messages are modified to specifically address that victim, purportedly coming from an entity that they are familiar with and containing personal information. Spear-phishing requires more thought and time to achieve than phishing. Spear-phishing attackers try to obtain as much personal information about their victims as possible to make the emails that they send look legitimate and to increase their chance of fooling recipients. Because of the personal level of these emails, it is more difficult to identify spear-phishing attacks than to identify phishing attacks conducted at a wide scale. This is why spear-phishing attacks are becoming more prevalent.



PayPal Yesterday at 8:32 PM
To: [illegible]
[Paypal Team] : Login to your account and update your information✔

PayPal

This is an automated email, please do not reply

information about your account :

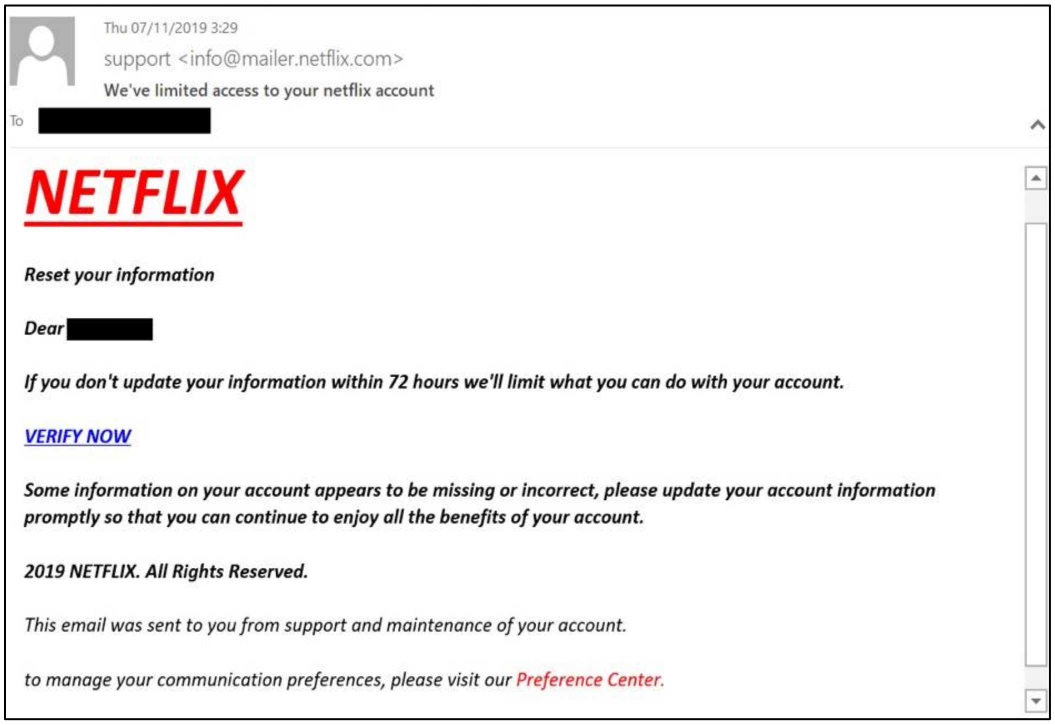
Warning! Your PayPal account was limited!

Your account has been limited temporarily in order to protect it. The account will continue to be limited until it is approved.
Once you have updated your account records, your information will be confirmed and your account will start to work as normal once again.
The process does not take more than 5 minutes.
Once connected, follow the steps to activate your account. We appreciate your understanding as we work to ensure security.

Click here to Confirm Your Account Information.

Department review PayPal accounts

copyright 1999-2016 PayPal.All rights reserved
PayPal FSA Register Number:1388561750


Thu 07/11/2019 3:29

support <info@mailer.netflix.com>

We've limited access to your netflix account

To [redacted]

NETFLIX

Reset your information

Dear [redacted]

If you don't update your information within 72 hours we'll limit what you can do with your account.

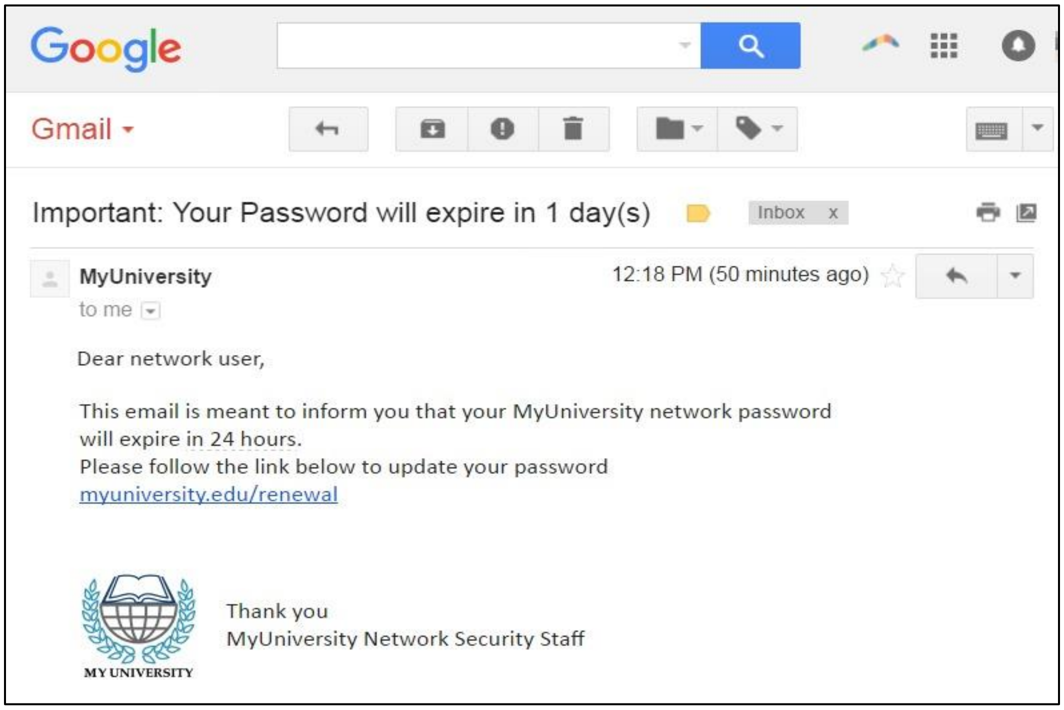
VERIFY NOW

Some information on your account appears to be missing or incorrect, please update your account information promptly so that you can continue to enjoy all the benefits of your account.

2019 NETFLIX. All Rights Reserved.

This email was sent to you from support and maintenance of your account.

to manage your communication preferences, please visit our Preference Center.



Important: Your Password will expire in 1 day(s)   Inbox x

MyUniversity   12:18 PM (50 minutes ago)

to me

Dear network user,

This email is meant to inform you that your MyUniversity network password will expire in 24 hours.
Please follow the link below to update your password
myuniversity.edu/renewal

MY UNIVERSITY

Thank you
MyUniversity Network Security Staff

Conclusion:

Here by,E-mail is one of the most-used forms of communication. We learned How secure are the systems that we use daily and what are the implications of insecure systems .We also looked at alternatives that would ensure security.