**Activity 2 Guide: Data Encrytion Tools**

Table of Contents

[About Encryption 2](#_Toc180494694)

[Part 1 – Symmetric Encryption 2](#_Toc180494695)

[Asymmetric Encryption 3](#_Toc180494696)

[Hashing 3](#_Toc180494697)

[Check for Understanding 4](#_Toc180494698)

[Data Encryption Tools 5](#_Toc180494699)

[GPG Historical Background: 6](#_Toc180494700)

[Install GNU PGP: 6](#_Toc180494701)

[Symmetric Encryption with GPG Using Macintosh 8](#_Toc180494702)

[Windows Instructions 13](#_Toc180494703)

[Symmetric Encryption with GPG 14](#_Toc180494704)

[Encrypt with Right-click 16](#_Toc180494705)

[Command Line Interface (for Windows and Mac) 18](#_Toc180494706)

[Decrypting 23](#_Toc180494707)

[Part 2 – Asymmetric Encryption 25](#_Toc180494708)

[Asymmetric Information 25](#_Toc180494709)

[Web Client and Server Key Exchange 25](#_Toc180494710)

[Enigma Machine is an Example of a Symmetric Key Problem 28](#_Toc180494711)

[An Effective Key Sharing Solution 29](#_Toc180494712)

[The Mathematical Genius of Public and Private Keys 29](#_Toc180494713)

[The RSA Process in Five Steps 32](#_Toc180494714)

[Computer Code for RSA Cryptography. 34](#_Toc180494715)

[Asymmetric Encryption with GPG 37](#_Toc180494716)

[Method 1 – Command Line Interface 37](#_Toc180494717)

[Decryption 39](#_Toc180494718)

[Method 2 – Encryption and Decryption using File Explorer GUI (Windows) 40](#_Toc180494719)

[Exchange Public Keys with a Partner 40](#_Toc180494720)

[Encrypt the Message 42](#_Toc180494721)

[Import a Key 43](#_Toc180494722)

[Encrypt a Message 45](#_Toc180494723)

[Symmetric Encryption with GPG in Macintosh 47](#_Toc180494724)

[Share Your Public Key with a Partner. 47](#_Toc180494725)

[Open Your Partner's Email that He/She Sent to You. 49](#_Toc180494726)

[Part 3 – Comparing Encryption Tools 53](#_Toc180494727)

[Tools Used in Today’s Communication Apps 53](#_Toc180494728)

[Part 4 – Quantum Computing 56](#_Toc180494729)

[A Big Step Forward in Encryption 56](#_Toc180494730)

[Part 5 – Summarize 59](#_Toc180494731)

[Key Takeaways: 60](#_Toc180494732)

[Check for Understanding 61](#_Toc180494733)

[Deliverables: 63](#_Toc180494734)

# About Encryption

#### Introduction to Cryptography

In this activity, you will see two styles of encryption: symmetric and asymmetric encryption. First, you will read some background information and then use the tool GPG to perform both symmetric and asymmetric encryption. In a future activity, you will explore two related topics: hashing and how it is used to protect passwords and HTTPS, used in encrypted web applications.

# Part 1 – Symmetric Encryption

Symmetric encryption uses a single key for both encryption and decryption, as seen in Figure 1.

Plain text doc



#Dd9+c12 9Z-52^$@9@nnN



Plain text doc

Algorithm

Algorithm

Figure Symmetric encryption uses the same pass key to both encrypt as well as decrypt data.

Symmetric encryption algorithms have been in use for millennia. Two important modern encryption algorithms include:

1. **DES (Data Encryption Standard)**:
   * Developed by IBM in 1971.
   * Became insecure due to advancements in computing power.
   * Encrypts data using a 56-bit key.
   * Vulnerable to brute-force attacks due to its relatively short key length.
   * No longer used for serious encryption applications.
2. **AES (Advanced Encryption Standard)**:
   * Replaced DES in the late 1990s.
   * Uses key lengths of 128, 192, or 256 bits. Longer key lengths are more secure.
   * Considered secure due to its longer key lengths and complex encryption process.
   * Quantum computing may eventually render all current encryption algorithms vulnerable.

## Asymmetric Encryption

Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption. A key is like a password used to "lock" and "unlock" an encrypted document. Asymmetric encryption is widely used for secure communications over the internet.

## Hashing

Hashing is a process that converts input data into a fixed-size string of characters, typically appearing as a random sequence. It is a one-way function, meaning it cannot be reversed. Hash functions are used for data integrity and authentication. Common hashing algorithms include MD5, SHA-1, and SHA-256.

How Hashing is Similar to Encryption

1. **Data Transformation**: Both hashing and encryption transform data from its original form to a different form.
2. **Security Purposes**: Both are used to protect data. Encryption protects data in transit or at rest, while hashing ensures data integrity and is used for verifying the authenticity of data.
3. **Algorithms**: Both use mathematical algorithms to transform data.

How Hashing is Different from Encryption

1. **Reversibility**:
   * **Encryption**: Reversible. Encrypted data (ciphertext) can be decrypted back to its original form (plaintext) using the appropriate key.
   * **Hashing**: Irreversible. Hashed data cannot be converted back to its original form. It is a one-way process.
2. **Purpose**:
   * **Encryption**: Used to protect the confidentiality of data, ensuring that only authorized parties can access it.
   * **Hashing**: Used to verify data integrity and authenticity, ensuring that data has not been tampered with.
3. **Output**:
   * **Encryption**: Produces variable-length output depending on the input size.
   * **Hashing**: Produces fixed-length output regardless of the input size.

**Are Passwords Encrypted?**

Typically, passwords are not encrypted but hashed. Encryption is used for data that needs to be recovered in its original form, whereas passwords should not be reversible for security reasons. When users create passwords, the passwords are hashed and stored in a database. When users log in, their entered passwords are hashed, and the result is compared to the stored hash. If they match, the user is authenticated.

**Does Password Cracking Mean Password Decryption?**

No, password cracking does not mean password decryption. Since passwords are typically hashed, cracking involves guessing the password, hashing it, and comparing the hash to the stored hash. In a future lesson, we will explore techniques like brute force, dictionary attacks, and rainbow tables to crack passwords by trying numerous possibilities until a match is found.

## Check for Understanding

Although these questions are not graded, they will prepare you for upcoming assessments.

**1. What is the primary characteristic of symmetric encryption?**

* A) It uses a public key for encryption and a private key for decryption.
* B) It uses different keys for encryption and decryption.
* C) It uses the same key for both encryption and decryption.
* D) It does not require a key for decryption.

**Answer:** C) It uses the same key for both encryption and decryption.

**2. Which of the following encryption algorithms is no longer used for serious encryption due to its vulnerability to brute-force attacks?**

* A) AES (Advanced Encryption Standard)
* B) DES (Data Encryption Standard)
* C) RSA (Rivest-Shamir-Adleman)
* D) SHA-256

**Answer:** B) DES (Data Encryption Standard)

**3. What key length options does AES (Advanced Encryption Standard) offer?**

* A) 56 bits
* B) 64 bits, 128 bits, and 192 bits
* C) 128 bits, 192 bits, and 256 bits
* D) 512 bits and 1024 bits

**Answer:** C) 128 bits, 192 bits, and 256 bits

**4. How does asymmetric encryption differ from symmetric encryption?**

* A) Asymmetric encryption uses the same key for both encryption and decryption.
* B) Asymmetric encryption is reversible, while symmetric encryption is irreversible.
* C) Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption.
* D) Asymmetric encryption does not require keys at all.

**Answer:** C) Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption.

**5. Why are passwords typically hashed instead of encrypted?**

* A) Hashing allows passwords to be decrypted when needed.
* B) Hashing is a reversible process, making it ideal for storing passwords.
* C) Hashing ensures that passwords are not stored in a recoverable form, enhancing security.
* D) Hashing is faster than encryption and requires less computational power.

**Answer:** C) Hashing ensures that passwords are not stored in a recoverable form, enhancing security.

## Data Encryption Tools

In this section, we will demonstrate the tool GnuPG (Gnu Privacy Guard), which will encrypt data both symmetrically and asymmetrically.

**PGP (Pretty Good Privacy)** is an encryption program that provides cryptographic privacy and authentication. PGP is often used for securing emails, files, and data communication. It supports both symmetric and asymmetric encryption.

**GNU Privacy Guard (GnuPG or GPG)** is a free implementation of the OpenPGP standard, developed by the Free Software Foundation (FSF). It was created as an alternative to PGP, offering similar functionalities but under an open-source license, which ensures that users have the freedom to use, study, modify, and distribute the software.

## **GPG Historical Background:**

1. **1991: Release of PGP (Pretty Good Privacy):**  
   In 1991, Phil Zimmermann, an independent programmer and well-known security activist, released Pretty Good Privacy (PGP) to the public for free and provided the code as open source. PGP was revolutionary at the time as it provided uncrackable encryption for free to anyone.

* **Legal Challenges:**  
  At the time, strong encryption technology like PGP was considered a *munition* under U.S. law. The U.S. government generally allowed the export of encryption with key lengths of up to 40 bits (8 bytes) for symmetric algorithms, which was weak and could be broken even by individuals or small organizations. Stronger encryption, using 128-bit or 256-bit keys, was generally not allowed for export without special permission. There is very little difference between software using 40-bit keys and 256-bit keys. The algorithm is the same. The software structure is the same. The only difference is the length of a key value.
* The US Government put Zimmermann under investigation for potentially violating the Arms Export Control Act, which carried severe penalties, including fines of up to $1 million and imprisonment for up to 10 years.
* In 1996, the U.S. government dropped the charges without any indictments. There was growing public support for Zimmermann and increasing recognition of the simplicity and value of encrypting data for personal and business use.

1. **Development of GnuPG (Gnu Privacy Guard):**  
   In 1999, Werner Koch, a German software developer, initiated the development of Gnu Privacy Guard (GnuPG or GPG) as a free and open-source alternative to PGP. Over time, GnuPG became an essential part of the digital privacy in email clients, operating systems, and security tools.

## Install Gnu PGP:

1. Visit [the Gnu Download page](https://gnupg.org/download/index.html), seen in Figure 2, to download the latest version of GPG for your operating system.

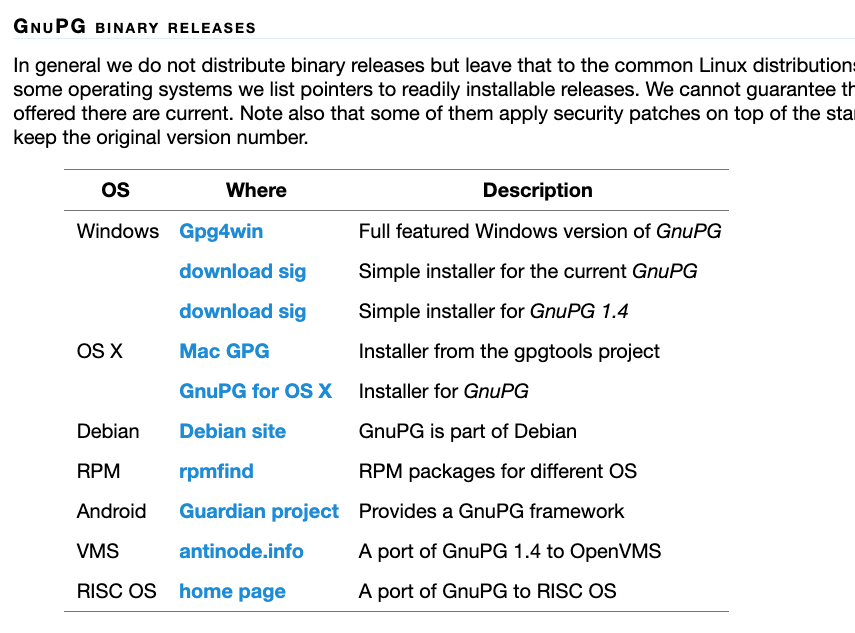


Figure Best download options for GPG Windows and Mac.

=========== Begin Macintosh Section ==================

Skip this section if you are using Windows.

* 1. Choose the Mac version for GPG Suite, shown in Figure 3.

A screenshot of a computer

Description automatically generated

Figure GPG Suite installation screen.

* 1. Launch the program found in the Applications folder as shown in Figure 4.



Figure GPG Keychain is the application icon found in the Macintosh Applications folder.

* 1. By default, the GPG application asks for a "new key pair" value when you first start the application, shown in Figure 5. This is your private key, to be used in the next section of this lesson.
  2. Enter a secret password and remember it as you will use it later in this lesson.

A screenshot of a computer

Description automatically generated

Figure Adding a new private key the first time you run the application.

## Symmetric Encryption with GPG using Macintosh

1. Create a folder to store documents for this assignment, as shown in Figure 6.

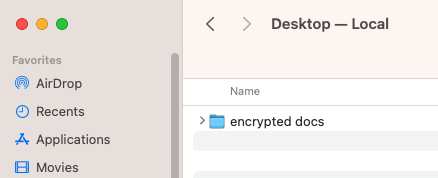


Figure "encrypted docs" is a folder on the desktop.

1. Create a new text document with **TextEdit** or other text editor and add some confidential text, shown in Figures 7 through 9. **Visual Studio Code** is a good alternative to Notepad.

A screenshot of a computer

Description automatically generated

Figure Contents of a text file using TextEdit in MacOS.

A screenshot of a computer

Description automatically generated

Figure TextEdit for MacOS has the Make Plain Text command under the Format menu.

A screenshot of a computer

Description automatically generated

Figure Saving a plain text document using MacOS and TextEdit.

1. For Macintosh, right-click on the text file and choose Services > OpenPGP: Encrypt File, as shown in Figure 10.

A screenshot of a computer

Description automatically generated

Figure Encrypting with MacOS.

1. Uncheck the **recipients**, **sign** or **Add to Recipients** options. These options are for Public/Private key exchanges which we will perform in the next section of this lesson. For now, simply check "Encrypt with Password," as shown in Figure 11.

A screenshot of a computer

Description automatically generated

Figure Choosing "Encrypt with Password" and not choosing sign, or Add to Recipients.

1. The program will generate a new file myplan.txt.gpg which should appear alongside the original document.
2. Open the gpg document with the TextEdit program to see the encrypted contents. Right-click and choose Other, as seen in Figure 12.

A screenshot of a computer

Description automatically generated

Figure Selecting Other.

1. Choose TextEdit from the applications list, as shown in Figure 13.

A screenshot of a computer

Description automatically generated

Figure Selecting TextEdit as the application to open gpg files.

1. You should be able to see the contents are in an unreadable format, as seen in Figure 14.

A screenshot of a computer

Description automatically generated

Figure The encrypted version of my plan is opened with TextEdit. The contents are in a binary, encrypted, format that TextEdit displays as a series of strange characters.



* Take a screenshot of the results at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

Continue to the "Command Line Interface Section" further ahead in the instructions. "Command Line Interface" appears after the Windows Instructions.

**====================== End of Macintosh Instructions ================**

## Windows Instructions

1. Install the application with default settings, as shown in Figures 15 and 16.

A screenshot of a computer

Description automatically generated

Figure Installation options include the core application, Keopatra (Key manager tool), GpgOL (For Microsoft Outlook email encryption), GpgEX (Shell Extension)

A screenshot of a computer

Description automatically generated

Figure Default install folder selected.

## Symmetric Encryption with GPG

1. Create a folder to store documents for this assignment, as shown in Figure 17.

A screenshot of a computer

Description automatically generated

Figure A new folder designated to hold multiple documents that will be created in this lesson.

1. For Windows users, turn off the "Hide extensions for known file types" to see **.txt** extensions on a text file, as shown in Figure 18.

A screenshot of a computer screen

Description automatically generated

Figure Folder options in File Explorer allow you to show or hide the .txt extension for text files.

1. Create a new text document with **Notepad** or other text editor and add some confidential text, as shown in Figure 19. **Visual Studio Code** is a good alternative to Notepad.

A screenshot of a computer

Description automatically generated

Figure Contents of a text file with sensitive information.

## Encrypt with Right-Click

1. For Windows, right-click on the text file and choose "Shore More Options," as shown in Figure 20.

A screenshot of a computer

Description automatically generated

Figure Right-clicking the text file.

1. Choose Encrypt under "More GpgEX options," as shown in Figures 21, 22, and 23.

A screenshot of a computer

Description automatically generated

Figure GPG EX has been integrated with the Windows right-click commands to provide Encryption.

A screenshot of a computer

Description automatically generated

Figure Destination folder is the same as the existing file.

A screenshot of a computer

Description automatically generated

Figure Entering a symmetric password.

The program should generate an encrypted version of the text file with a **gpg** extension.

1. Right-click on the gpg file and open with Notepad to view the encrypted text, as shown in Figures 24 and 25.

A screenshot of a computer

Description automatically generated

Figure Selecting notepad to view the contents of the encrypted file.

A screenshot of a computer

Description automatically generated

Figure The encrypted file is a binary format. The characters used in the encryption do not display well in Notepad, which is designed to work with the ASCII character set.



* Take a screenshot of the results at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

**===================== End Windows Instructions ==================**

# Command Line Interface (for Windows and Mac)

GPG can perform operations in the command line as well as with the GUI application. GPG command line may have more options.

1. Open a command terminal prompt and navigate to the desktop folder where you saved the clear text and encrypted documents.
2. Type **dir** (or **ls** with Mac / Linux) to view the contents of the directory, as seen in Figure 26.

A screenshot of a computer screen

Description automatically generated

Figure View the contents of the directory showing both the plain text (txt) and encrypted text (gpg).

1. Run the following command to encrypt the text file. Use double hyphens when specifying an option. Some students have found that a copy and paste operations introduces text formatting problems. You may have to type the command manually.

gpg --symmetric --cipher-algo AES256 --armour "My diabolical plan.txt"

The "armour" option will **encrypt** the file and **encode** the data to the ASCII character set. Encrypt and encode are distinct concepts.

The *–cipher-algo AES256* option allows you to choose an encryption algorithm. Here are other algorithms that you can try.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cipher** | **Key Length** | **Description** | **Specialization / Speed** |
| **AES128** | 128 bits | AES with a 128-bit key, a widely used encryption standard. | **Speed:** Fast and efficient; **Use Case:** Ideal for general-purpose encryption where a balance of security and performance is needed. |
| **AES192** | 192 bits | AES with a 192-bit key, offering more security than AES128. | **Security:** Offers higher security than AES128; **Use Case:** For applications needing stronger security with minimal performance loss. |
| **AES256** | 256 bits | AES with a 256-bit key, providing the highest security level among the AES options. | **Security:** Highest level of security among AES variants; **Use Case:** Preferred for highly sensitive data requiring maximum security. |
| **3DES** | 168 bits (56 bits x 3) | Applies the DES cipher algorithm three times to each data block, typically using three different keys. | **Security:** Legacy algorithm; **Use Case:** Still used in some financial and legacy systems, but generally slower and less efficient than AES. |
| **CAST5** | 40 to 128 bits | A symmetric-key block cipher with a 64-bit block size and key sizes ranging from 40 to 128 bits. | **Specialization:** Lightweight and flexible; **Use Case:** Suitable for applications requiring smaller key sizes or backward compatibility. |
| **Blowfish** | 32 to 448 bits | A symmetric-key block cipher with a variable key length from 32 bits up to 448 bits. | **Speed:** Fast and efficient for smaller key sizes; **Use Case:** Used in systems where flexibility in key length is advantageous. |
| **Twofish** | Up to 256 bits | A symmetric-key block cipher with a block size of 128 bits and key sizes up to 256 bits. | **Specialization:** Strong and versatile; **Use Case:** Often considered a potential successor to AES, suitable for both software and hardware implementation. |
| **Camellia128** | 128 bits | Camellia cipher with a 128-bit key, similar to AES128 in security and performance. | **Security:** Comparable to AES128; **Use Case:** Alternative to AES, particularly in Japan and Europe, with similar security and efficiency. |
| **Camellia192** | 192 bits | Camellia cipher with a 192-bit key. | **Security:** Comparable to AES192; **Use Case:** Chosen when AES is not preferred, especially in specific regions or industries. |
| **Camellia256** | 256 bits | Camellia cipher with a 256-bit key, similar to AES256 in security and performance. | **Security:** Comparable to AES256; **Use Case:** Used for maximum security in areas where Camellia is preferred over AES. |

1. Use the **dir** command (**ls** in Mac/Linux) to list the conents of the directory again. You should see a new file with an asc extension, seen in Figure 27. **asc** is short for ASCII.

A screenshot of a computer

Description automatically generated

Figure A new file with asc extension is now in the directory. There are now two encrypted files.



* Take a screenshot of the directory results at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

1. Open the resulting **asc** file with notepad (Windows) as shown in Figure 28 or Text Edit (Mac) as shown in Figures 29 and 30.

A screenshot of a computer

Description automatically generated

Figure Using Notepad to view the contents of the encrypted file.

A screenshot of a computer

Description automatically generated

Figure For Macintosh, choose Open With > Other.

A screenshot of a computer

Description automatically generated

Figure Macintosh opening the asc file with TextEdit.

1. You should notice that the file is composed of ASCII characters, which display well with Notepad or other applications. You could copy and paste this text into an email message and send it to a friend as part of the body of a message, as shown in Figures 31 and 32.

A screenshot of a computer

Description automatically generated

Figure Encrypted message in ASCII characters shown in Notepad.

A screenshot of a computer

Description automatically generated

Figure Encrypted message in ASCII characters with TextEdit.



* Take a screenshot of the ascii version of the encrypted document.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

1. Delete the original txt document, as shown in Figure 33. We will recreate it by unencrypting one or both encrypted versions.

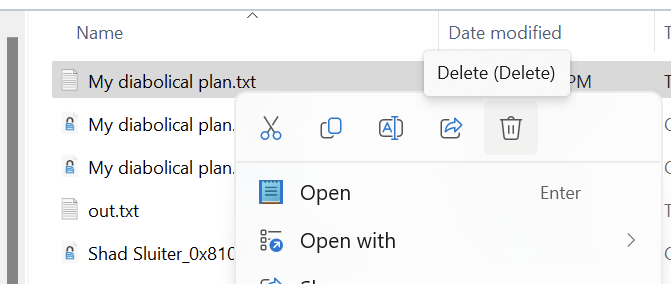


Figure Deleting the "My diabolical plan.txt" document.

## Decrypting

1. There are two ways to decrypt a file, command line and GUI integration. We will decrypt the file two times using each method once. Both methods should result in recreating the original text file.
2. Using the first method, the **command prompt**, type the following command, as seen in Figure 34. The ">" option allows you to direct the output of the operation to a new file name. Use "out.txt" to help distinguish the output file from the original file.

A black screen with white text

Description automatically generated

Figure Decrypting the file using a command prompt command and naming the output file as out.txt. The success message indicates that the AES256 encryption algorithm was used to encrypt the document.

1. For the second approach use integrated commands in the **file explorer**. Right-click on the file and choose "Decrypt," as shown in Figures 35 and 36.

A screenshot of a computer

Description automatically generated

Figure Using the integrated UI commands to decrypt a file.

A screenshot of a computer

Description automatically generated

Figure Decrypting the file using Services > OpenPGP: Decrypt File.

1. The resulting output file should be identical to the original encrypted file. The original file name is used by default when saving the unencrypted file.

**Summary**: Symmetric encryption is easy to understand and use. However, it lacks the ability to securely share data with another user without first sharing the decryption key. For more convenient data sharing, asymmetric encryption was invented.

# Part 2 – Asymmetric Encryption

## Asymmetric Information

Asymmetric encryption uses a pair of encryption keys: a public key for encryption and a private key for decryption, as shown in Figure 37. A key is like a password used to "lock" and "unlock" an encrypted document. Asymmetric encryption is widely used for secure communications over the internet.

* **Public Key**: Can be shared openly and is used to encrypt data.
* **Private Key**: Kept secret and is used to decrypt data.

Plain text doc



#Dd9+c12 9Z-52^$@9@nnN



Plain text doc

Algorithm

Algorithm



Figure Asymmetric encryption uses two keys that are mathematically related. A public key is used to encrypt a document. The document is encrypted differently for each recipient. The recipient uses his/her private key to decrypt the document.

Asymmetric encryption solves the key distribution problem inherent in symmetric encryption, making it useful for secure communications between parties without prior shared secrets.

## Web Client and Server Key Exchange

Consider a common problem: a web server and a browser client need to encrypt data before sending it over the internet. They both wish to share a common encryption key before sending and the same encryption key for decrypting data. But how does the web server share the encryption password with the client while not sharing it with the whole world?

To describe it in another way, imagine that you and a friend share a rented storage unit in Denver, Colorado. The padlock on the storage unit has a key. You live in Phoenix, while your friend lives in Chicago, as shown in Figure 38. Regularly, you and your friend need to visit the storage unit to pick up items. (What kind of suspicious business is this?) This analogy describes the process of sharing a network connection between a server and a client.

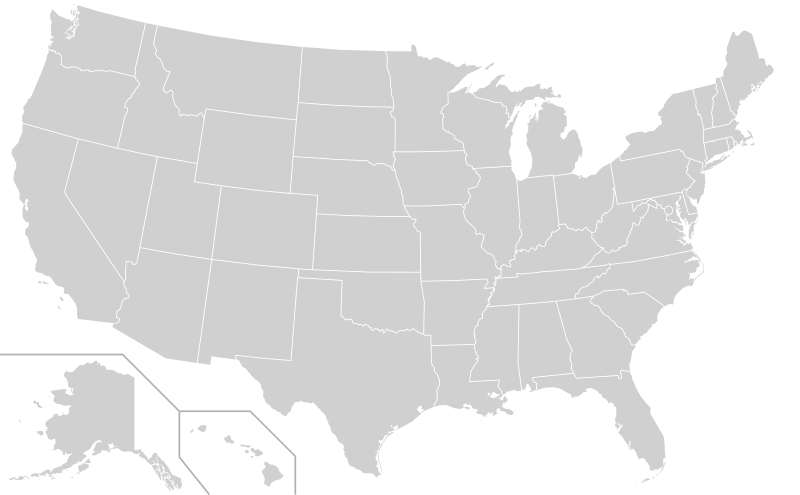


Figure You and your friend share a storage unit located between two cities. This is like a shared network connection with secure traffic.

To start the process, your friend agrees to travel to Denver to open the unit and pick up the keys. The problem now arises of how to safely send a copy of the key to you in Phoenix. Here are some options:

* You could both decide to drive to Denver or you or your friend drive across the country to Phoenix, wasting precious time, gasoline, and money.
* You could also agree to leave key "hidden" somewhere in the weeds outside the storage facility, as shown in Figure 39. This method of "*security through obscurity*" usually loses. Obscurity is like naming a folder or URL with an obscure path name in hopes that a hacker doesn't see it. However, if the URL is discovered, the security is broken.

http://mysite.com/asdfsecreturlpathway



Figure A fake rock is "security through obscurity" which is not a good solution for IT systems.

* Another plan for sharing the key is that you could send the key through the mail, as shown in Figure 40. But the key could be easily intercepted. A paper envelope is like an unencrypted message sent by email. Anyone who finds it can open it. We would need an *unbreakable envelope* to send the key, which is the solution we will eventually come to.

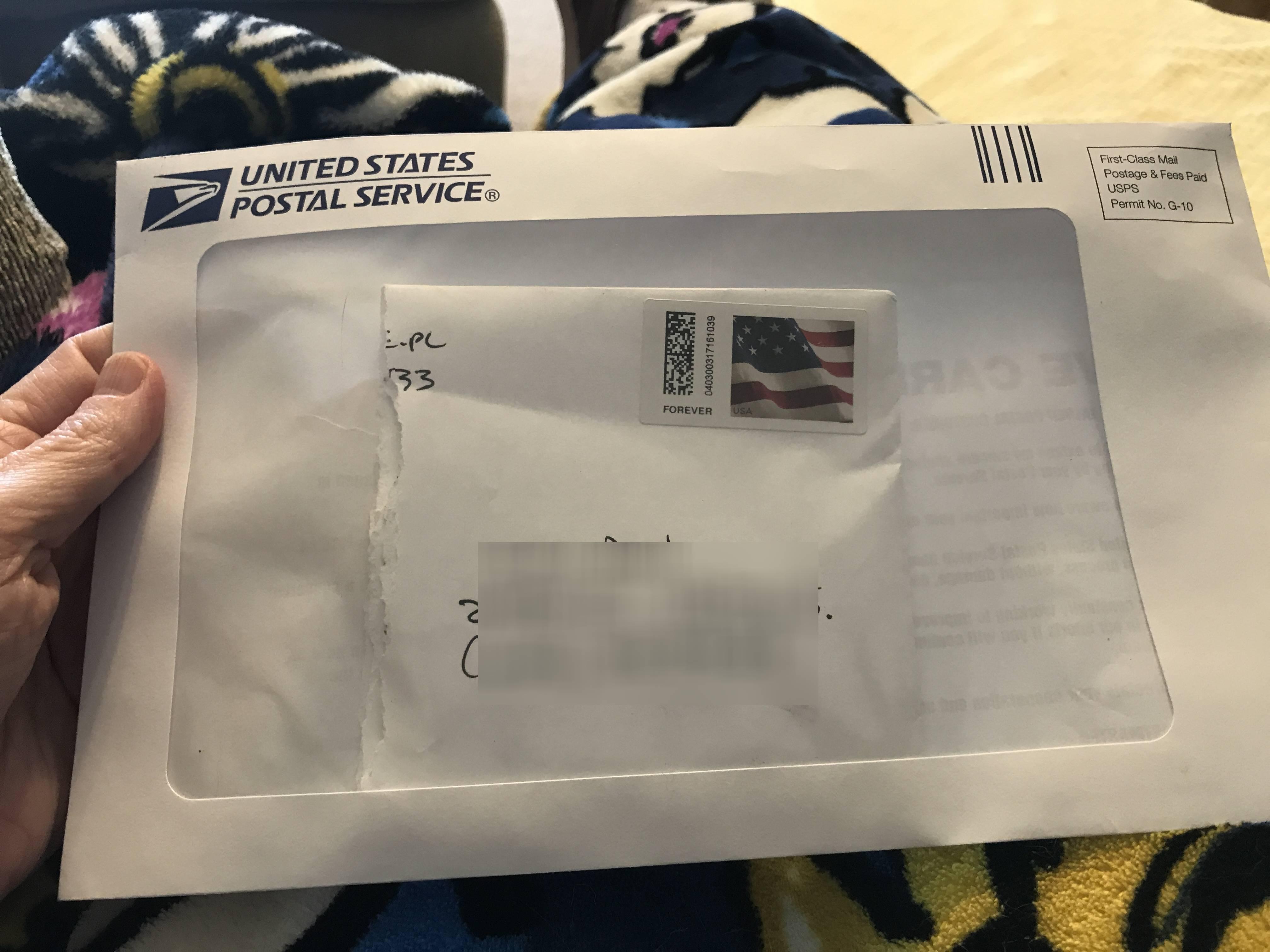


Figure Standard mail envelopes are like unencrypted network traffic, easily intercepted.

## Enigma Machine is an Example of a Symmetric Key Problem

In World War II, the German military used a cypher machine called the Enigma for encrypted communications, seen in Figure 41. The machine relied on symmetric encryption methods, meaning that many copies of code books were published and shared with field commanders on ships, submarines, and military command posts.

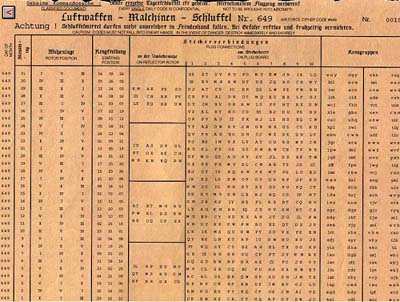


Figure Cypher key sheet for German Enigma machines. Each row represents a new symmetric key for each day of the month.

Sometimes a copy of the cypher key sheet was captured by the enemy and all communication in every location of the German military were compromised for up to 30 days until a new encryption key sheet was distributed. This demonstrates the weakness of a symmetric key in secure communications.

Interestingly, the governments of Poland, United States and Great Britain collaborated to design a machine that was able to break the encryption keys every morning in about 30 minutes. During the war, the Allies knew most of what the Nazi military was planning. Meanwhile, the Nazis believed their machine was secure. The Allies enhanced their own encryption devices based on what they learned about the weaknesses found in the Enigma.

## An Effective Key Sharing Solution

A secure solution for sharing an encryption key would be to send the key through the mail but with an important modification. You could *send the key in an unbreakable lock box* that your friend can close but only you can open, as seen in Figure 42.

* First, you purchase the lock box and set the door code. You are the only person who knows the combination code.
* You *leave the door of the safe open* so that your friend can put the key inside.
* You send the empty safe to your friend using the mail service.
* Your friend puts a key in the safe and shuts the door. He cannot open the door since only you know the combination.
* The safe is returned to you with the key securely inside.

This lock box is like a public key (open door on the lockbox) and a private key (only you know the combination to open it). If someone intercepts the locked box, the key is safe inside the lock box.



Figure A lock box with a code can be closed by another user, but only you know the combination to open it. This is like a private key.

## The Mathematical Genius of Public and Private Keys

Asymmetric encryption is a relatively recent development in the field of cryptography.

**Inventors**

* In 1976, Whitfield Diffie and Martin Hellman from Sandford University, as seen in Figures 43 and 44, jointly invented Public and Private key cryptography, the most important advancement in cryptography in 2,000 years. They cited the work of Ralph Merkle, a graduate student at Berkely, as seen in Figure 45, who was solving a very similar problem. Merkel named his key exchange technique "Merkle Puzzles."



Figure Whitfield Diffie, coinventor of asynchronous encryption.



Figure Martin Hellman, coinventor of asynchronous encryption.



Figure Ralph Merkel, coinventor of asynchronous encryption.

Review their ground-breaking publications here:

* "[New Directions in Cryptography," by Diffie and Hellman, from *IEEE Transactions on* *Information Theory* (1976)](https://ee.stanford.edu/~hellman/publications/24.pdf)
* "[Secure Communications over Insecure Channels](https://dl.acm.org/doi/pdf/10.1145/359460.359473)," by Merkle, from *Communications of the ACM* (1978)

RSA

* The RSA algorithm, named after its inventors **Rivest, Shamir, and Adleman, as seen in Figures 46 through 49**, was developed in 1977, just a year after Diffie and Hellman's work was published. RSA was the first practical implementation of the public key cryptography concept that Diffie and Hellman had theorized.



Figure Ronald Rivest from M.I.T. Coinventor of RSA. Inventor of the MD5 hashing algorithm and encryption algorithms, RC2, RC4 and RC5.



Figure Adi Shamir, coinventor of RSA. Inventor of Differential cryptanalysis, a cryptographic cracking technique.

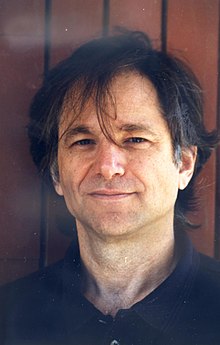


Figure Adleman, coinventor of RSA. Known as the "Father of DNA Computing."

* Before 1976, cryptography was based on symmetric key algorithms, where the same key is used for both encryption and decryption. The major challenge with this approach is that the secure exchange of the key is practically impossible.

**Prime numbers are key to understanding public private keys.**

The algorithm begins by choosing two very large prime numbers at random. These become the factors of a multiplication product that is computationally difficult to factor. Using these two numbers and their product, a public and private key are generated.

## The RSA Process in Five Steps

**Step 1: Choose Two Large Prime Numbers**

* + Select two large prime numbers, typically labeled as **p** and **q**.
  + These primes are chosen randomly and independently of each other. The larger the prime numbers, the more secure the keys will be, but also the more computationally intensive the process will be.
  + For a 2048-bit RSA key, p and q are typically around **1024 bits** each. This means that p and q are both 1024-bit prime numbers. Larger values for p and q are more secure.

**Step 2: Compute the Modulus (n)**

* + Multiply the two prime numbers to get the modulus **n**:

* + The modulus **n** is used as part of both the public and private keys. The security of RSA relies on the difficulty of factoring this large number **n** back into its prime factors **p** and **q**.

**Step 3: Calculate the Euler's Totient Function (φ(n))**

* **Totient Function Calculation:**
  + Compute the totient function **φ(n)**, which is derived from **p** and **q**:
  + This function counts the number of integers up to **n** that are relatively prime to **n** (i.e., they have no common divisors with **n** other than 1).

**Step 4: Choose the Public Exponent (e)**

* **Public Exponent Selection:**
  + Choose an integer **e** that is relatively prime to **φ(n)** and lies between 1 and **φ(n)**. This number **e** will be the public exponent. Common choices for **e** include small prime numbers like 3, 17 for human readability.
  + 65537 is a common choice for e.
  + **65537** is a Fermat prime (a prime of the form )
  + For RSA to work, e must be relatively prime to ϕ(n).
  + Since 65537 is prime, it has a high probability of being relatively prime to ϕ(n) especially when p and q are large primes.

**Step 5: Compute the Private Exponent (d)**

* **Private Exponent Calculation:**
  + Calculate **d** as the modular multiplicative inverse of **e** modulo **φ(n)**, meaning that **d** satisfies the equation:
  + The value **d** is the private exponent, and it is kept secret. This number allows the decryption of data that was encrypted using the public key.

**Step 6: Create Public and Private Keys:**

* **Public Key:**
  + The public key consists of the pair **(n, e)**.
  + This key is shared openly and is used by others to encrypt messages intended for the key owner.
* **Private Key:**
  + The private key consists of the pair **(n, d)**.
  + This key is kept secret and is used by the key owner to decrypt messages that were encrypted with the corresponding public key.

**Step 7: Use the Two Keys**

* **Encryption with the Public Key:**
  + Convert the message string into an array of bytes.
  + Convert the bytes into a large integer.
  + A sender encrypts a message **M** (represented as an integer) using the recipient's public key **(n, e)**:
  + The result **C** is the ciphertext, which can be sent over an insecure channel.
* **Decryption with the Private Key:**
  + The recipient decrypts the ciphertext **C** using their private key **(n, d)**:
  + The result **M** is the original message.
  + Convert the integer M into a string of bytes and then into ASCII characters to get the original message.

**Cracking the System:**

1. **Factoring Challenge:**
   * The security of RSA encryption relies on the difficulty of factoring the large number **n** into its prime factors **p** and **q**. As long as **n** is large enough (typically 2048 bits or more), it can take most computers an enormous amount of time to factor **n** using current technology.
2. **Prime Numbers:**
   * The use of prime numbers is important because the properties of prime numbers and modular arithmetic make it possible to create a one-way function for encryption that is easy to compute in one direction (encryption) but very difficult to reverse without the private key (decryption).

## Computer Code for RSA Cryptography

1. Create a new Java application.
2. Copy this code into the App class.
3. For best results download this document and open it in Microsoft Word. The copy/paste operation in Chrome will remove all line breaks.

package com.shadsluiter;

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class App {

public static void main(String[] args) {

SecureRandom secureRandom = new SecureRandom();

// List of ANSI color codes

List<String> colors = Arrays.asList("\u001B[30m", "\u001B[31m", "\u001B[32m", "\u001B[33m", "\u001B[34m", "\u001B[35m", "\u001B[36m", "\u001B[37m");

// User input

Scanner scanner = new Scanner(System.in);

while(true) {

// black

System.out.println(colors.get(0) + "Welcome to the RSA encryption and decryption program!");

System.out.print("Enter your message: ");

String message = scanner.nextLine();

int bitLength = getBitLength(scanner);

BigInteger exponent = getExponent(scanner);

Date startTime = new Date();

BigInteger e = exponent;

// Generate two 1024-bit prime numbers

BigInteger p = generatePrime(secureRandom, bitLength);

BigInteger q = generatePrime(secureRandom, bitLength);

// Calculate modulus n = p \* q

BigInteger n = p.multiply(q);

// Calculate Euler's Totient Function φ(n) = (p - 1) \* (q - 1)

BigInteger phi = calculatePhi(p, q);

// Ensure that e is coprime with φ(n)

if (!isCoprime(e, phi)) {

System.out.println("The public exponent e is not coprime with φ(n). Please choose a different value for e.");

continue;

}

// Calculate the private exponent d = e^(-1) mod φ(n)

BigInteger d = e.modInverse(phi);

// Print values with colors

printColored("p: " + p, colors.get(0));

printColored("q: " + q, colors.get(1));

printColored("n: " + n, colors.get(2));

printColored("φ(n) = (p - 1) \* (q - 1) = " + phi, colors.get(3));

printColored("e: is the public exponent = " + e, colors.get(4));

printColored("d: is the private key = " + d, colors.get(5));

// Encrypt the message

printColored("Message: " + message, colors.get(0));

BigInteger encryptedMessage = encryptMessage(message, e, n);

printColored("Encrypted message as integer: " + encryptedMessage, colors.get(6));

// Decrypt the message

BigInteger decryptedMessage = encryptedMessage.modPow(d, n);

printDecryptedMessage(decryptedMessage, colors);

Date endTime = new Date();

long timeElapsed = endTime.getTime() - startTime.getTime();

printColored("Time elapsed: " + timeElapsed + " milliseconds", colors.get(7));

// Ask the user if they want to continue

System.out.print("Do you want to encrypt another message? (yes/no): ");

String continueChoice = scanner.nextLine();

if (!continueChoice.equalsIgnoreCase("yes")) {

break;

}

}

}

private static boolean isCoprime(BigInteger e, BigInteger phi) {

return e.gcd(phi).equals(BigInteger.ONE);

}

private static BigInteger getExponent(Scanner scanner) {

// Provide user with multiple choices for the value of e

List<BigInteger> eChoices = new ArrayList<>();

for (int i = 2; i < 9; i += 2) {

// 2 ^ 2 ^ i + 1 =

BigInteger e = new BigInteger("2").pow(2).pow(i).add(BigInteger.ONE);

eChoices.add(e);

}

System.out.println("Choose a value for the public exponent e:");

for (int i = 0; i < eChoices.size(); i++) {

System.out.println((i + 1) + ": " + eChoices.get(i));

}

System.out.print("Enter the number corresponding to your choice: ");

int eChoiceIndex = scanner.nextInt() - 1;

scanner.nextLine(); // Consume the newline character left by nextInt()

return eChoices.get(eChoiceIndex);

}

private static int getBitLength(Scanner scanner) {

int bitLength = 1024;

System.out.println("What bit length would you like to use for the prime numbers? Longer is more secure (40 - 4096)");

while (true) {

System.out.print("Enter a number: ");

bitLength = scanner.nextInt();

scanner.nextLine(); // Consume the newline character left by nextInt()

if (bitLength >= 40 && bitLength <= 4096) {

break;

} else {

System.out.println("Please enter a number between 10 and 4096.");

}

}

return bitLength;

}

private static BigInteger generatePrime(SecureRandom secureRandom, int bitLength) {

return new BigInteger(bitLength, 100, secureRandom);

}

private static BigInteger calculatePhi(BigInteger p, BigInteger q) {

return p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

}

private static void printColored(String message, String color) {

System.out.print(color);

System.out.println(message);

}

private static BigInteger encryptMessage(String message, BigInteger e, BigInteger n) {

BigInteger messageInt = new BigInteger(message.getBytes());

return messageInt.modPow(e, n);

}

private static void printDecryptedMessage(BigInteger decryptedMessage, List<String> colors) {

printColored("Decrypted integer value: " + decryptedMessage, colors.get(0));

printColored("Decrypted message in byte array: " + Arrays.toString(decryptedMessage.toByteArray()), colors.get(1));

printColored("Decrypted message as an ASCII string: " + new String(decryptedMessage.toByteArray()), colors.get(2));

}

}

1. Run the application.
2. Encrypt a phrase using a small prime number value, as shown in Figure 49.
3. Repeat the encryption using a larger key value, as shown in Figure 50. You should see that the amount of time to encrypt/decrypt grows exponentially with the length of the values of p and q.

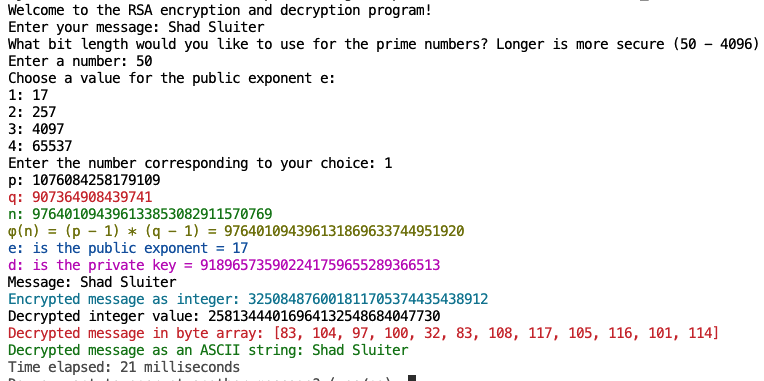


Figure Encrypting a string with a small value for p and q (50 bits).



Figure Encrypting a string with a medium-length value for p and q (1000 bits).



* Take a screenshot of the application running one of the encryption operations at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

## Asymmetric Encryption with GPG

This exercise is best done using a partner or at least two separate email accounts. You will create public/private keys, exchange public keys, encrypt, and share a message and decrypt the message using your private key. We will show the process first using the command line interface and then using the Windows Explorer integration.

## Method 1 – Command Line Interface

1. Create your own private and public key files.

**gpg --gen-key**

1. Follow the steps through the process:



* Take a screenshot of the information about your key.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

*Share your Key with Another User*

1. Use the --armor option to make it an ASCII file. Dump it to a file. Copy and paste from this line may result in errors due to text encoding differences.

**gpg --export --armor youremail@email.com > yourname-public-key.asc.**

(use two hyphens for the export and armor options)

1. For additional information on how to use GPG, use the following command:

**gpg –help**

1. Share your public key via email. Copy the contents of the public-key.asc file into an email or attach the public-key.asci file. Send it to your friend. You could also share your public key using social media or publish it on a website.
2. Your partner also needs to create a public key and share it with you.
3. Import the public key made by your partner.

**gpg --import yourfriend-pub-key.asc**

(two hyphens for import option)

1. View the public keys you have imported into your computer.

**gpg --list-keys**

1. Your list of public keys will grow as you work with more individuals and share public keys.



* Take a screenshot of the list of public keys you have at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

*Encrypt a Message*

With asymmetric encryption, you need to perform the encryption process once per friend. You cannot encrypt a document for a general audience.

1. Encrypt a message in preparation for sharing it with another user. In this example, I am going to share a file (filename.txt) with my friend Shad Sluiter, who has shared his public key with me. Once again, copy and paste may not work with command line commands.

**gpg –-encrypt --armor –-recipient** [**shad.sluiter@gcu.edu**](mailto:shad.sluiter@gcu.edu) **"My diabolical plan.txt"**

1. This will use the public key (sent from shad.sluiter and imported) to encrypt the text file. The armor option encodes the message in ASCII format.
2. Type **dir** to confirm the encrypted file is there. There should be a new, encrypted file in the same directory as "My diaboloical plan.txt"
3. Open the encrypted message. You should see a similar format as shown in Figure 51.

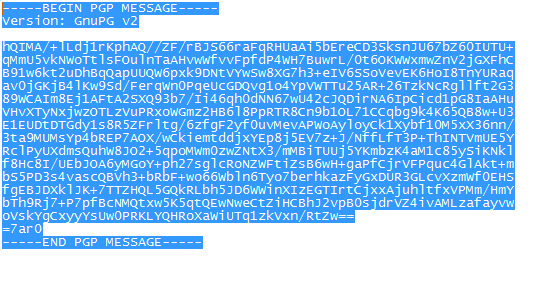


Figure ASCII encoded PGP message sent either as an attached file or as the body of an email.



* Take a screenshot of the encrypted version of your message.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

1. Send the encoded message either as an attachment or copy it into the body of the text.
2. Send it to your friend who will then download it and decrypt it. Even though email itself may be sent as an unencrypted message, the body of the message is in a secure format.

## Decryption

Let's assume you have shared your public key with a friend. Your friend imported the public key into GPG. He/she used your public key to encrypt a message that is intended for you. He/she sent you an email with the ascii version of the encrypted file.

1. Open your friend's email.
2. Save the encrypted message to a file.
3. Run the decrypt command in the command terminal to create an unencrypted text file.

**gpg –-decrypt message-from-friend.txt.asc > output.txt**

1. You must provide your private key to decrypt the message.
2. Open the output file with Notepad. You should be able to see the private message.



* Take a screenshot of the unencrypted message from your friend.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

## Method 2 – Encryption and Decryption Using File Explorer GUI (Windows)

GPG works equally well with the GUI tools as it does with the command line. We will repeat the process using the GUI application, first in Windows and then instructions will be shown for Macintosh.

## Exchange Public Keys with a Partner

1. Run the Kleopatra program which was installed as part of the GPG application, as seen in Figure 52.

A screenshot of a computer

Description automatically generated

Figure Starting screen for Kleopatra.

1. Click New Key Pair.
2. Enter your name and email, as shown in Figures 53 and 54.

A screenshot of a computer

Description automatically generated

Figure Creating a new public key using the Kleopatra program.

A screenshot of a computer

Description automatically generated

Figure Successfully created a public key. The "fingerprint" is a hash value of the public key. You can share this fingerprint with the public as well. It is used to verify that your public key has not been tampered with.

1. Export the Key to a file, as shown in Figure 55. Then select the certificate with your name and choose Export, as shown in Figure 56.

A screenshot of a computer

Description automatically generated

Figure Exporting the public key to a text file.

A screenshot of a computer

Description automatically generated

Figure The public key for Shad Sluiter is saved in the project folder.



* Take a screenshot of the Kleopatra list of keys.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

## Encrypt the Message

1. Right-click on the unencrypted file and choose Sign and Encrypt, as seen in Figure 57.

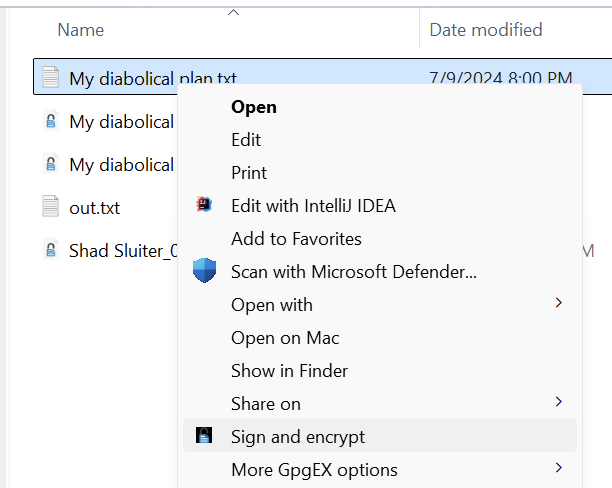


Figure "Sign and encrypt" the file "My diabolical plan.txt"

1. Save the file to the project folder.
2. Send the public key file to your friend as an email attachment. Your partner needs to share his/her public key file with you.

## Import a Key

1. After receiving the public key file from your friend, import the file. Click the import button, as shown in Figure 58.

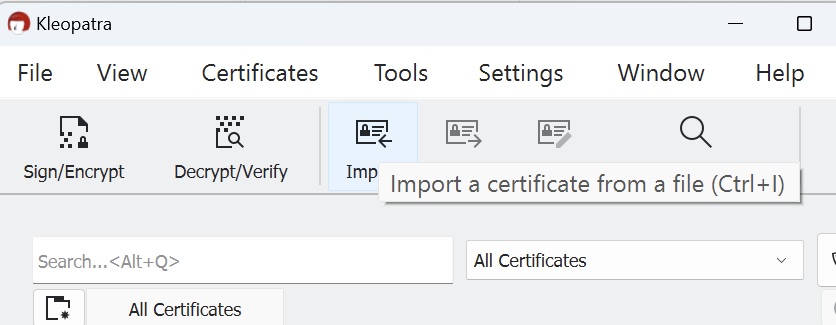


Figure Import a public key from a file sent to you by your partner.

1. Select the file received from your friend, as shown in Figure 59.

A screenshot of a computer

Description automatically generated

Figure The import message recommends that you verify the "fingerprint" value of your partner's public key.

1. You can verify the validity of the certificate using the fingerprint, as seen in Figure 60, or you can implicitly trust your friend.

A screenshot of a computer

Description automatically generated

Figure A new key has been imported into the system from [professorsluiter@gmail.com](mailto:professorsluiter@gmail.com).



* Take a screenshot of the application showing a new certificate has been saved..
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

1. The public key has been imported, as seen in Figure 61.

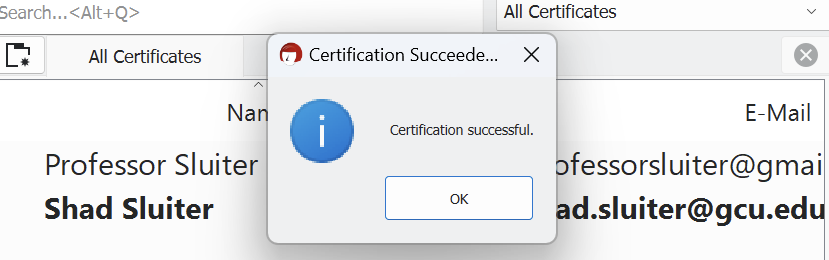


Figure Assuming I checked the fingerprint and the hash values match, I can trust messages sent from this user.

## Encrypt a Message

* 1. Right-click on a text file you wish to encrypt and choose **Sign and Encrypt**, as shown in Figures 62 and 63.

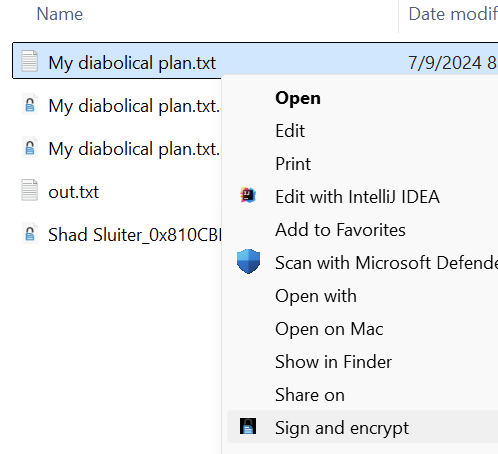


Figure Encrypting a message using a public/private key.

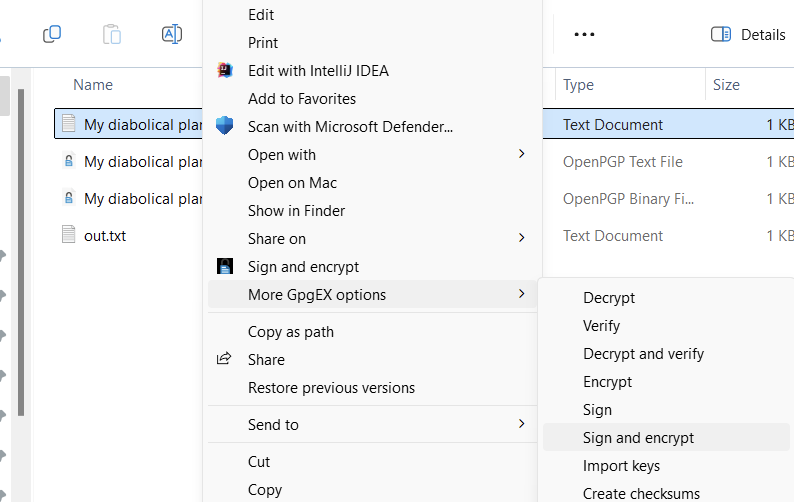


Figure Sign and Encrypt is found in two places in the UI integration.

* 1. Select the source (your name) and the destination (your partner's name), as shown in Figure 64.

A screenshot of a computer

Description automatically generated

Figure This message is being encrypted by shad.sluiter@gcu.edu specifically for [professorsluiter@gmail.com](mailto:professorsluiter@gmail.com). Include yourself if you want to also be able to decrypt the message.

The resulting encrypted message can only be opened by the person who you designate, as shown in Figure 65. His/her private key is required to decrypt the message.

* 1. Click Sign/Encrypt.
  2. Send the resulting encrypted file to your friend.
  3. Your friend should be able to decrypt the file using the Kleopatra GUI application.

A screenshot of a computer

Description automatically generated

Figure Decrypt is complete for two intended recipients, the sender included himself as a recipient.



* Take a screenshot of the successful decryption at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.
  1. Look in the output folder for the decrypted version of the message. You should now be able to see the secret message sent to you by your friend.

**============ End Windows Section =====================**

# Symmetric Encryption with GPG in Macintosh

## Share Your Public Key with a Partner

1. Open the GPG Keychain tool.
2. Right-click on your email and choose Send Public Key via Email, as seen in Figure 66.

A screenshot of a computer

Description automatically generated

Figure Sharing your public key with a partner via email.

1. The application will open your default email client and attach the Public Key file encoded in ASC format, as shown in Figure 67.

A screenshot of a computer

Description automatically generated

Figure Email with your public key asc file attached.

1. Fill in the To: field with the email of your friend.
2. Send the message with the attached public key file.

**Assuming your partner also sends you a public key…**

## Open Your Partner's Email that He/She Sent to You

1. Download the public key sent to you from your friend.
2. Import key button on the PGP app, as shown in Figure 68.

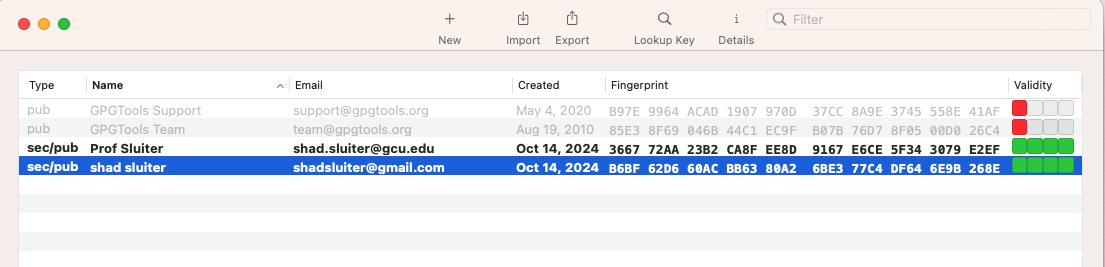


Figure Import button for adding a friend's public key to your computer.

1. Select the asc file you received from your friend.
2. You should see two email addresses in the GPG tool's sec/pub list, as seen in Figure 69. You are now ready to exchange secure encrypted messages with your partner.

A screenshot of a computer

Description automatically generated

Figure Your public key and a friend's public key are registered with the GPG application.

1. In the file Finder, right-click on the secret message text file and choose Services > OpenPG : Encrypte File, as shown in Figure 70.

A screenshot of a computer

Description automatically generated

Figure Encrypting a file with GPG.

1. Check mark the name of your friend in the Recipients List, seen in Figure 71.
2. Check mark "Sign" and "Add to Recipients" boxes.
3. Uncheck "Encrypt with Password."
4. Ensure that Your Key shows your email address.



Figure Selecting your friend as a recipient.

1. You should see a "success" message, as shown in Figure 72.

A screenshot of a computer

Description automatically generated

Figure Success message after decrypting a file destined for your friend.

1. Send the myplan.txt.gpg file to your friend as an attachment in an email.
2. Wait to receive a gpg file from your partner.
3. Save the encrypted message from your partner to a folder on your computer.
4. Right-click the encrypted file and choose Services > OpenGPG : Decrypt, as seen in Figure 73.

A screenshot of a computer

Description automatically generated

Figure Decrypting the file received from your friend.

1. Provide your password.
2. The file should be decrypted and saved to your computer, as shown in Figure 74.

A screenshot of a computer

Description automatically generated

Figure Successfully decrypted a file from your friend using your private key.



* Take a screenshot of the successful decryption at this point.
* Paste the image into a Microsoft Word document.
* Put a caption below the image explaining what is being demonstrated.

# Part 3 – Comparing Encryption Tools

Now that we have explored the mechanics of encryption, let's consider how these technologies are used in real-world communication apps that are a part of our daily lives. The level of protection can vary significantly depending on the app and the encryption method it employs.

## Tools Used in Today’s Communication Apps

|  |  |  |
| --- | --- | --- |
| Application | Encryption Method | Details |
| ****WhatsApp**** | Signal Protocol | End-to-end encryption (E2EE) ensures that only the communicating users can read the messages. Even WhatsApp cannot decrypt them. |
| **Instagram** | HTTPS/TLS for data in transit | Messages and data are encrypted in transit using HTTPS/TLS, but direct messages are not end-to-end encrypted by default. |
| **Facebook Messenger** | Secret Conversations use the Signal Protocol | Standard messages are encrypted in transit using HTTPS/TLS. For end-to-end encryption, users must enable "Secret Conversations," which use the Signal Protocol. |
| **SMS Texting** | None (in standard SMS) | SMS messages are not encrypted, making them vulnerable to interception. Enhanced messaging services like RCS (Rich Communication Services) may offer better security but are not universally encrypted end-to-end. Many Two-Factor Authentication (2FA) systems use texting SMS, making them less secure than 2FA applications on your phone. |
| **Email** | Varies (TLS, PGP, S/MIME) | **TLS:** Encrypts email in transit between servers but not end-to-end. **PGP:** Provides end-to-end encryption if both sender and receiver use PGP. **S/MIME:** Another protocol for end-to-end email encryption, typically used in enterprise settings. |
| **Signal** | Signal Protocol | End-to-end encryption by default for all messages and calls. |
| **Telegram** | MTProto | Messages are encrypted in transit and on servers. "Secret Chats" provide end-to-end encryption. |
| **Apple iMessage** | End-to-end encryption | Uses a proprietary Apple protocol to provide end-to-end encryption for messages between Apple devices. |
| **Google Messages** | Signal Protocol (for RCS messages) | SMS is not encrypted, but RCS messages can be end-to-end encrypted when both parties use compatible devices and carriers. |

# **Part 4 – Quantum Computing**

## **A Big Step Forward in Encryption**

Quantum computing is a huge leap in computing power. It uses ideas from quantum mechanics to perform calculations much faster than regular binary computers. Unlike traditional bits, which are either 0 or 1, quantum bits (qubits) can be both 0 and 1 at the same time because of a property called superposition. This lets quantum computers solve complex problems, like factoring large numbers, much more quickly than classical computers.

**The Threat to Current Encryption Standards**

* **Breaking RSA:**
  + The RSA encryption algorithm is a key part of many secure communications today. Its security relies on how difficult it is to factor large prime numbers. Right now, factoring a large number used in RSA would take an impractical amount of time on regular computers.
  + Quantum computers, using methods like Shor's algorithm, could factor these large numbers much faster, effectively breaking RSA encryption. This would make RSA and similar encryption methods vulnerable to quantum attacks.
* **Will Existing Messages Be Revealed?**
  + If a powerful enough quantum computer is developed, it could potentially decrypt messages that were previously encrypted using RSA or similar algorithms. This means that encrypted data intercepted today could be stored and decrypted in the future when quantum computing becomes available. Sensitive communications, financial transactions, and state secrets could be at risk.
* **The Future of RSA:**
  + While RSA is still secure today, its long-term future is in doubt due to the threat of quantum computing. Security experts are already looking into post-quantum cryptography, which involves creating new encryption methods that can resist quantum attacks.
* **Huge Advantage:**
  + The country that leads in quantum computing will gain a significant advantage both militarily and economically. Quantum computers could break current encryption standards, giving the holder powerful abilities in cyber spying, secure communication, and data manipulation.
  + Militarily, being able to decrypt enemy communications could change the balance of power. Economically, access to decrypted data like financial transactions, corporate secrets, and personal information could provide a major advantage.

**What Will Replace RSA?**

* **Post-Quantum Cryptography:**  
  Researchers are developing new encryption methods believed to be secure against quantum attacks. These are known as post-quantum cryptography.
  + **Lattice-Based Cryptography:** This method relies on tough math problems related to lattice points in high-dimensional spaces. These problems are hard for both classical and quantum computers to solve.
  + **Hash-Based Cryptography:** This approach uses hash functions that are resistant to quantum attacks.
  + **Code-Based Cryptography:** Techniques like the McEliece cryptosystem use the difficulty of decoding random linear codes, offering another quantum-resistant option.
  + **Multivariate Polynomial Cryptography:** This involves solving systems of complex equations, which are believed to be hard even for quantum computers.

**Algorithm Updates:** In September 2024, Google, Microsoft and other technology companies finished a years-long process of replacing RSA encryption with a new quantum-resistant algorithms in anticipation of upcoming quantum developments.

**Quantum Beyond Encryption**

Factoring large numbers isn't the only thing quantum computing can do. Quantum computers have the potential to revolutionize many fields by solving problems that are too hard for classical computers.

1. **Quantum Simulation**
   * **Molecular and Chemical Simulations:**
     + Quantum computers can simulate molecular structures and chemical reactions with high precision, which is extremely challenging for classical computers. This is important in drug discovery, material science, and chemistry, where understanding complex molecular interactions is important.
   * **Material Science:**
     + Quantum computers could help design new materials with specific properties by simulating atomic structures at the quantum level. This could lead to advances in superconductors, catalysts, and other unique materials.
2. **Optimization Problems**
   * **Logistics and Supply Chain Optimization:**
     + Many logistical problems, like optimizing delivery routes and scheduling, involve complex calculations that classical computers struggle with. Quantum computers can potentially solve these problems faster and more accurately.
     + Companies like Volkswagen and DHL are already exploring quantum algorithms to improve traffic flow and delivery routes.
   * **Financial Modeling:**
     + In finance, quantum computers could optimize portfolio management, risk analysis, and pricing of complex financial products. They could model market trends and predict outcomes more effectively than traditional methods.
3. **Cryptography Beyond Factoring**
   * **Quantum Key Distribution (QKD):**
     + While quantum computers threaten current cryptographic methods, they also enable new forms of secure communication, like QKD. This uses quantum mechanics to create encryption keys that are theoretically unbreakable and can detect any attempt at eavesdropping.
     + Protocols like BB84 allow two parties to securely share a cryptographic key. If someone tries to intercept the key, the authorized users will know.
4. **Machine Learning**
   * **Quantum Machine Learning:**
     + Quantum computers could speed up machine learning algorithms, allowing for faster training of models on large datasets. They might discover patterns that classical computers can't detect.
5. **Search Algorithms**
   * **Grover’s Algorithm:**
     + Grover's algorithm is a quantum search method that speeds up unstructured search problems. While classical algorithms check each entry one by one, Grover's algorithm can find a specific item in an unsorted database much faster.
6. **Artificial Intelligence and Complex Decision-Making**
   * **Enhanced Decision-Making:**
     + Quantum computers could improve decision-making processes in AI by enabling more complex simulations and optimizations. This could lead to more advanced AI systems capable of solving harder problems with greater accuracy.
7. **Quantum Communication**
   * **Secure Communication Networks:**
     + Quantum communication protocols can build secure networks that are resistant to eavesdropping. Any attempt to intercept the communication would disturb the quantum state, revealing the presence of an eavesdropper. This is the foundation for developing a quantum internet, which could offer ultra-secure and efficient communication systems.
8. **Drug Discovery and Genomics**
   * **Personalized Medicine:**
     + Quantum computers could analyze huge amounts of genomic data to find patterns leading to personalized medicine. They could also simulate drug interactions at the molecular level, leading to more effective and targeted treatments.
9. **Climate Modeling**
   * **Environmental and Climate Modeling:**
     + Quantum computing could improve the accuracy of climate models by simulating complex atmospheric and oceanic processes. Better models could lead to improved predictions and strategies for addressing climate change.

# Part 5 – Summarize

Write a comprehensive summary of the key concepts demonstrated in this lesson, ensuring each of the following topics is covered in depth. Provide detailed explanations, examples, and citations from both the lesson material and external research.

1. **Compare and Contrast Hashing with Encryption:**
   * Discuss not only the fundamental differences between hashing and encryption, but also how each is used in modern technology (e.g., data integrity, password protection, etc.).
   * Provide examples of real-world scenarios where encryption is necessary, and others where hashing is the best solution.
2. **Compare and Contrast Symmetric and Asymmetric Encryption:**
   * Explain the advantages and disadvantages of each in different use cases (e.g., speed vs. security, ease of key distribution, etc.).
   * Include a comparison how these two approaches are best applied in the real world. Consider situations where encryption is required but one approach is preferred over the other. For example encryption could be used in all of these situations: tape backups, hard drives, web traffic, chat applications, telephone conversations, USB storage, email.
3. **Practical Application in Messaging Apps:**
   * Based on the comparison of various messaging tools, evaluate your current phone applications in terms of encryption strength and security.
   * Explain whether you would change or configure these apps differently and provide reasons based on what you have learned about encryption.
   * Support your plan with examples from the apps discussed.
4. **Research on Quantum Computing and Its Impact on Encryption:**
   * Conduct research on the future of encryption in the quantum age. What encryption methods are considered quantum-resistant?
   * Based on current publications, how soon do experts believe quantum computers will become powerful enough to break current encryption standards?
   * Cite at least two reputable sources for your research.

# **Key Takeaways:**

To summarize the main ideas of this lesson, review the following concepts.

* **Difference Between Encryption and Hashing:**
  + **Encryption:** A reversible process that protects data confidentiality by ensuring only authorized parties can access the original information.
  + **Hashing:** A one-way process used primarily for data integrity, ensuring that data has not been tampered with.
* **Symmetric Encryption:**
  + Uses the same key for both encryption and decryption.
  + Examples include DES (now outdated) and AES, with AES256 being the most secure option.
  + While symmetric encryption is efficient, it poses challenges in securely distributing the key.
* **Asymmetric Encryption:**
  + Involves a pair of keys: a public key for encryption and a private key for decryption.
  + Solves the key distribution problem, making it ideal for secure communication over the internet.
  + Commonly used in HTTPS, where asymmetric encryption helps establish a secure session key.
* **Real-World Applications:**
  + Various communication apps implement encryption differently, with end-to-end encryption (E2EE) being the most secure method, as seen in apps like WhatsApp and Signal.
  + The lesson highlights the importance of choosing the right encryption tools depending on the sensitivity of the communication and the level of security required.
* **Quantum**
  + Quantum computing is an emerging technology that will re-revolutionize the cryptography landscape. New algorithms will break current encryption standards and open new options for secure communications.

## Check for Understanding

Although not graded, these questions will help you prepare for upcoming assessments.

1. **What is the primary difference between symmetric and asymmetric encryption?**

* A) Symmetric encryption uses two keys; asymmetric encryption uses one key.
* B) Symmetric encryption uses a single key for both encryption and decryption; asymmetric encryption uses a public and a private key.
* C) Symmetric encryption is faster; asymmetric encryption is slower.
* D) Symmetric encryption is used for public communication; asymmetric encryption is only used for private communication.

**Correct Answer:** B) Symmetric encryption uses a single key for both encryption and decryption; asymmetric encryption uses a public and a private key.

1. **Which of the following is a major weakness of the DES (data encryption standard) algorithm?**

* A) It uses a 128-bit key which is easily compromised.
* B) It relies on outdated hardware for implementation.
* C) It encrypts data using a 56-bit key, which is vulnerable to brute-force attacks.
* D) It is only compatible with symmetric encryption.

**Correct Answer:** C) It encrypts data using a 56-bit key, which is vulnerable to brute-force attacks.

1. **Why is AES (advanced encryption standard) considered more secure than DES?**

* A) AES uses longer key lengths of 128, 192, or 256 bits.
* B) AES uses asymmetric encryption, while DES uses symmetric encryption.
* C) AES relies on a more complex encryption algorithm with multiple rounds of processing.
* D) Both A and C.

**Correct Answer:** D) Both A and C.

1. **In asymmetric encryption, what is the purpose of the private key?**

* A) To encrypt data so that it can be decrypted by the public key.
* B) To decrypt data that was encrypted with the corresponding public key.
* C) To create a hash value for data verification.
* D) To generate the public key.

**Correct Answer:** B) To decrypt data that was encrypted with the corresponding public key.

1. **Which of the following encryption algorithms is NOT typically used for symmetric encryption?**

* A) AES256
* B) TWOFISH
* C) RSA
* D) CAMELLIA

**Correct Answer:** C) RSA

1. **In the context of HTTPS, what role does asymmetric encryption play?**

* A) It encrypts all data exchanged during the session.
* B) It encrypts the symmetric session key used for the duration of the session.
* C) It generates a digital signature to authenticate the server.
* D) It decrypts the data received from the client.

**Correct Answer:** B) It encrypts the symmetric session key used for the duration of the session.

1. **What does the --armor option in the GPG command line tool do?**

* A) It adds an additional layer of encryption to the file.
* B) It encodes the encrypted data in ASCII characters.
* C) It specifies the use of the AES256 algorithm for encryption.
* D) It prevents the encrypted file from being decrypted.

**Correct Answer:** B) It encodes the encrypted data in ASCII characters.

1. **Why is symmetric encryption considered less secure for sharing data between two parties compared to asymmetric encryption?**

* A) It requires both parties to have the same key, which must be securely shared beforehand.
* B) It uses a weaker encryption algorithm compared to asymmetric encryption.
* C) It is prone to timing attacks.
* D) It can only encrypt small amounts of data at a time.

**Correct Answer:** A) It requires both parties to have the same key, which must be securely shared beforehand.

1. **What is the purpose of hashing in the context of data security?**

* A) To encrypt data so it can be decrypted later.
* B) To verify the integrity and authenticity of data by producing a fixed-size output from variable input.
* C) To generate a key for symmetric encryption.
* D) To compress data for efficient storage.

**Correct Answer:** B) To verify the integrity and authenticity of data by producing a fixed-size output from variable input.

1. **Which statement best describes the relationship between encryption and hashing?**

* A) Both encryption and hashing are reversible processes.
* B) Encryption is reversible, while hashing is irreversible.
* C) Hashing is used to encrypt data during transmission.
* D) Encryption ensures data integrity, while hashing ensures data confidentiality.

**Correct Answer:** B) Encryption is reversible, while hashing is irreversible.

# Deliverables:

1. Submit a Microsoft Word document with screenshots of the applications being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
2. In the same document, include your research questions and summary from Part 5.
3. Convert the Word document to a PDF.
4. Attach the PDF document to the assignment submission.