Government Polytechinc,Pune
(An Autonomous Institute of Government of Maharashtra)

A Micro Project Report on

"Encryption-Decryption &Hashing Application"

Under The Subject

Computer Security [CM4110]
COMPUTER ENGINEERING

Submitted By

2106036 – Ritesh Ganesh Chinchole

2106039 – Rushikesh Pundlik Dawale

2106040 – Shreyas Sunil Deobhankar

2106046- Vishal Sandeepan Devkate

Under The Guidance Of

K.S.Sathwane Mam

Department of Computer Engineering
Government Polytechnic Pune
YEAR 2023-24

CERTIFICATE

This is to certify that the mini-project work entitled

“Encryption-Decryption &Hashing Application”

Is a bonafide work carried out by

2106036 - Ritesh Chinchole
2106039 - Rushikesh Dawale
2106040 -Shreyas Deobhankar
2106046 - Vishal Devkate

Of class 3rd Year in partial fulfillment of the requirement for the completion of course Computer Security (CM4110) - EVEN 2024 of Diploma in Computer Engineering from Government Polytechnic, Pune. The report has been approved as it satisfies the academic requirements in respect of Micro Project work prescribed for the course.

Micro Project Guide	**Head of Department**	**Principal**
(K.S.SATHWANE)	(MRS. J. R. HANGE)	(DR. R. K. PATIL)

ACKNOWLEDGEMENT

First and foremost, we express our heartfelt gratitude to all those who contributed to the success of this endeavor. Our sincere appreciation goes to Government Polytechnic Pune for providing us with guidance and invaluable advice throughout this journey. We extend special thanks and regards to our mentor, K.S.Sathwane Mam, whose unwavering support and encouragement have been instrumental in our progress.

Furthermore, we would like to acknowledge the leadership of our esteemed institution, under the guidance of Principal Rajendra Patil Sir. His visionary leadership and unwavering commitment to excellence have inspired us to strive for greatness. We are also grateful to our Head of Department, J. R. Hange, for her guidance and support in all our endeavors.

It is through the collective efforts of these esteemed individuals that we have been able to develop the skills and confidence necessary to excel as third-year students. Their support and enthusiasm have been a source of motivation for us throughout this journey.

As we reflect on our achievements, we recognize the invaluable role played by each member of our academic community. Together, we have achieved milestones and overcome challenges, and for that, we are truly grateful. We look forward to continuing our journey with the same dedication and enthusiasm, fueled by the unwavering support of our mentors and institution.

ABSTRACT

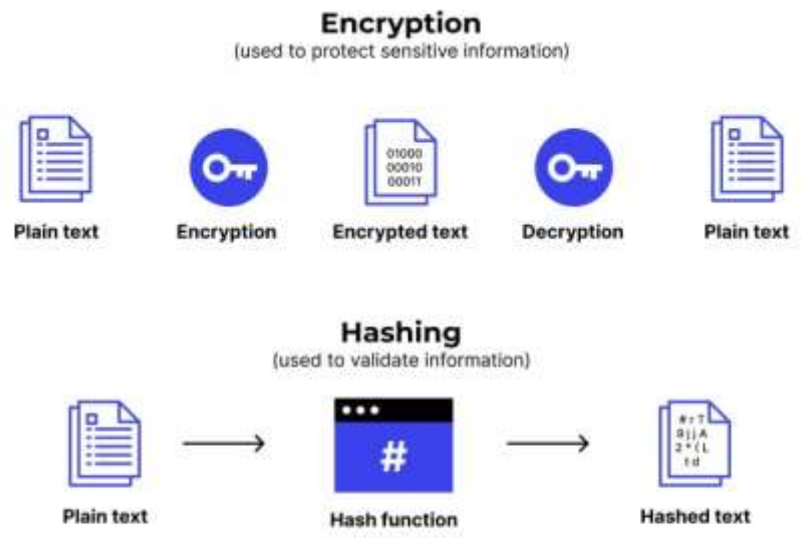
This report outlines the development of an Android application centered on encryption, decryption, and hashing functionalities. With a rising necessity for secure data handling in mobile applications, this project addresses the critical need for encryption, decryption, and hashing techniques to safeguard sensitive information. The application employs contemporary encryption algorithms to ensure robust data security during transmission and storage, offering users a range of encryption methods including symmetric and public-key cryptography. Moreover, the app integrates decryption capabilities, enabling authorized users to access and decrypt encrypted data seamlessly.

Additionally, the application incorporates hashing functionalities for generating unique hash values, facilitating data integrity verification. Utilizing secure hashing algorithms such as SHA-256, the app ensures that even minor alterations to input data yield significantly distinct hash values. The user interface is designed to be intuitive, allowing users to effortlessly encrypt, decrypt, and generate hashes for their data. Furthermore, the application implements best practices for key management and security to enhance the overall robustness of its encryption-decryption and hashing features.

In conclusion, this Android application offers a practical solution for individuals and organizations seeking to secure their sensitive data on mobile devices. By providing a comprehensive suite of encryption, decryption, and hashing functionalities within an intuitive interface, the application empowers users to protect their data from unauthorized access and tampering effectively.

INDEX

Chapter 1 : Introduction

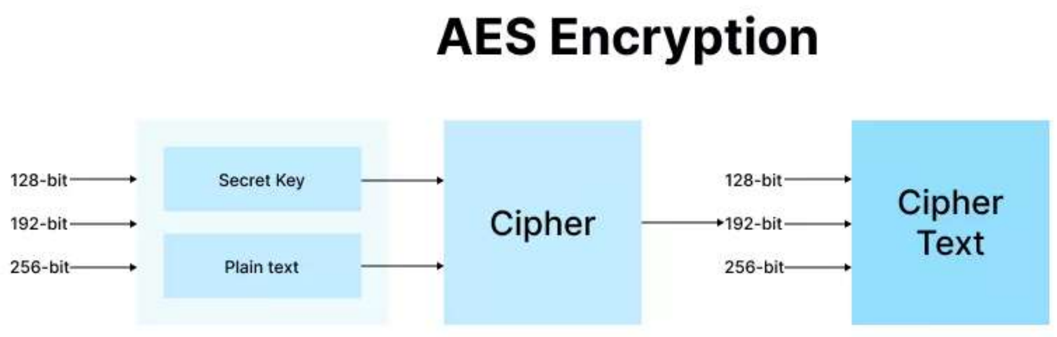


In the realm of information security, encryption, decryption, and hashing are foundational techniques crucial for ensuring the confidentiality, integrity, and authenticity of data. Encryption involves the transformation of plaintext into ciphertext using cryptographic algorithms and keys. This process renders the data unreadable to anyone without the decryption key, effectively protecting it from unauthorized access. For example, in a secure messaging application, messages are encrypted before transmission and decrypted upon receipt, ensuring that only the intended recipient can access the content.

Decryption, conversely, is the reverse process of encryption, where ciphertext is converted back into plaintext using the appropriate decryption key. This allows authorized users to access and interpret encrypted data. For instance, when a user receives an encrypted email, they can use their private decryption key to decrypt the message and read its contents securely.

Hashing, another essential technique, generates a fixed-length hash value from input data using cryptographic hash functions. Unlike encryption and decryption, hashing is a one-way process, meaning that the original data cannot be retrieved from the hash value. Instead, hashing is primarily used for data integrity verification. For instance, in password storage, instead of storing plaintext passwords, systems store hashed representations of passwords. When a user attempts to log in, the system hashes the entered password and compares it to the stored hash value to authenticate the user. This ensures that even if the database is compromised, plaintext passwords remain secure.

Chapter 2:Encryption



AES

Encryption plays a pivotal role in safeguarding sensitive information across digital platforms, and Advanced Encryption Standard (AES) stands out as a cornerstone in modern encryption techniques. AES, adopted by the U.S. government, employs symmetric-key encryption, where the same key is used for both encryption and decryption. This approach ensures efficient and secure communication between parties. AES operates on fixed-length blocks of data, typically 128 bits, and supports key lengths of 128, 192, or 256 bits, offering robust protection against cryptographic attacks.

The AES encryption process involves several steps. First, the plaintext data is divided into blocks, each of which undergoes multiple rounds of substitution, permutation, and mixing operations. These operations, performed according to the encryption key's schedule, obscure the relationship between the plaintext and ciphertext, ensuring that the encrypted data appears random and indistinguishable. The resulting ciphertext is then transmitted securely to the recipient.

For instance, consider a scenario where sensitive financial data is transmitted between a bank and a client. Prior to transmission, the plaintext data, such as account balances or transaction details, undergoes AES encryption using a shared secret key. This process transforms the plaintext into ciphertext, which is then securely transmitted over the network. Upon receipt, the recipient decrypts the ciphertext using the same secret key, retrieving the original plaintext data. Through AES encryption, the confidentiality and integrity of the financial information are preserved, ensuring secure communication between the bank and the client.

Chapter 3:Decryption

Decryption
abcdef
Cipher text
Algorithm
Plain text

Decryption in DES (Data Encryption Standard) follows a similar process to encryption but in reverse. DES is a symmetric-key block cipher that operates on fixed-length blocks of plaintext and ciphertext, typically 64 bits. The decryption process involves using the same secret key that was used for encryption to transform ciphertext back into plaintext. Here's an overview of the decryption process for DES:

Key Schedule: Just like in encryption, the firs

t step in decryption is the key schedule, where the original encryption key is used to generate the round keys. These round keys are used in the decryption rounds.

Initial Permutation: The ciphertext undergoes an initial permutation, which is the inverse of the initial permutation used in encryption. This step rearranges the bits of the ciphertext to prepare it for the decryption rounds.

16 Rounds of Decryption: DES decryption involves 16 rounds of processing, each consisting of the following steps:

Expansion Permutation: Expands the 32-bit right half of the data to 48 bits using a predefined permutation table.

Key Mixing: XORs the expanded right half with the corresponding round key derived from the key schedule.

S-Box Substitution: Applies the S-boxes (substitution boxes) to the result of the key mixing step, replacing each 6-bit block with a 4-bit block.

Permutation: Permutates the output of the S-box substitution using a fixed permutation table.

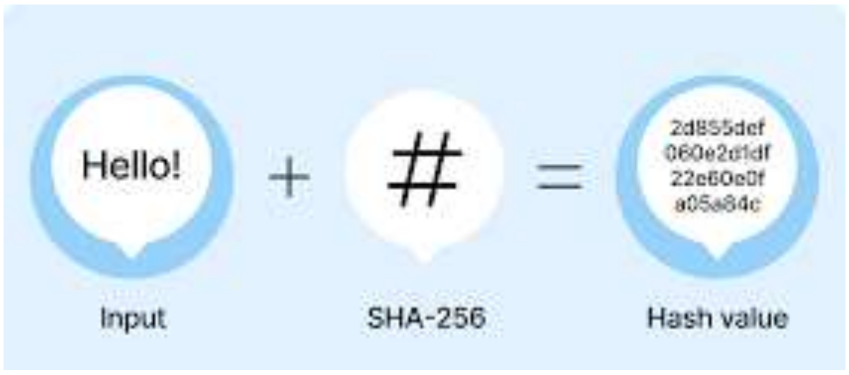
XOR with Left Half: XORs the output of the permutation step with the left half of the data.

Swap: Swaps the left and right halves of the data.

Final Permutation: After completing the 16 decryption rounds, the left and right halves of the data are swapped back, and a final permutation is applied to the entire block. This permutation is the inverse of the initial permutation used in encryption.

Output Transformation: The result of the final permutation is the decrypted plaintext.

Chapter 4 : Hashing



Hashing serves as a fundamental technique in modern cryptography, providing a means to transform data into a fixed-length string of characters, known as a hash value or hash code. SHA-256, a member of the Secure Hash Algorithm (SHA) family, is a widely used cryptographic hash function known for its strong security properties. SHA-256 generates a 256-bit hash value, offering a high level of collision resistance and ensuring that even small changes in the input data result in significantly different hash codes. This makes SHA-256 suitable for a variety of security applications, including digital signatures, data integrity verification, and password hashing.

The process of generating a SHA-256 hash involves several steps. First, the input data is processed in blocks of 512 bits, with padding added to ensure the input is a multiple of this block size. Next, the input block undergoes a series of mathematical operations, including bitwise operations, logical functions, and modular addition. These operations iteratively transform the input data, ultimately producing a 256-bit hash value. The resulting hash code is unique to the input data, providing a digital fingerprint that can be used to verify the integrity of the original data.

For example, in digital signatures, SHA-256 is often used to generate hash values from messages or documents. These hash values are then encrypted using a private key to create a digital signature, which can be verified by decrypting it with the corresponding public key and comparing the decrypted hash code with a freshly computed hash of the original message. If the two hash codes match, it indicates that the message has not been tampered with and was indeed signed by the holder of the private key. This illustrates the critical role of SHA-256 in ensuring data integrity and authenticity in cryptographic applications.

Chapter 5 :Applications of Encryption,Decryption &Hashing

Encryption, decryption, and hashing applications on Android devices serve various purposes across different domains. Here are some common applications:

1. Secure Communication: Android encryption-decryption applications are commonly used for secure communication, especially in messaging apps. These apps encrypt messages before transmission and decrypt them upon receipt, ensuring that only the intended recipients can access the content. This helps safeguard sensitive conversations from eavesdropping and interception, enhancing user privacy and confidentiality.

2. Data Storage Protection: Encryption-decryption applications on Android provide a means to encrypt sensitive data stored on the device, such as files, photos, and documents. By encrypting stored data, users can prevent unauthorized access in case their device is lost, stolen, or compromised. Decryption capabilities allow users to access and decrypt their encrypted data securely when needed.

3. Password Management: Hashing applications on Android are commonly used for password management. These apps store hashed representations of passwords instead of plaintext passwords, reducing the risk of password theft in case of a data breach. When a user logs in, the entered password is hashed and compared to the stored hash value for authentication. This helps protect user accounts from unauthorized access and enhances overall security.

4. Data Integrity Verification: Hashing applications on Android are also used for data integrity verification. Users can generate hash values for files, documents, or messages and compare them against hash values generated at a later time to ensure that the data has not been tampered with. This is particularly useful in verifying the authenticity of downloaded files or detecting unauthorized modifications to sensitive documents.

5. Secure Authentication: Encryption-decryption applications on Android can be used for secure authentication mechanisms, such as two-factor authentication (2FA) or digital signatures. By encrypting authentication tokens or digital signatures using cryptographic keys, these apps ensure the integrity and authenticity of the authentication process, mitigating the risk of identity theft and unauthorized access.

Overall, encryption, decryption, and hashing applications on Android play a crucial role in enhancing data security, privacy, and integrity across various use cases, ranging from secure communication to data storage protection and authentication mechanisms.

Chapter 6: Code and Snippets

XML Code:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:background="@drawable/cssproo"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/inputText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:hint="@string/enter_text"
        android:inputType="text"
        android:padding="14dp"
        android:autofillHints="Enter text only" />

    <Button
        android:id="@+id/encryptButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/inputText"
        android:layout_marginTop="16dp"
        android:text="@string/encrypt" />

    <Button
```

```
        android:id="@+id/decryptButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/encryptButton"
        android:layout_marginTop="16dp"
        android:text="@string/decrypt" />

    <Button
        android:id="@+id/hashButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/decryptButton"
        android:layout_marginTop="16dp"
        android:text="@string/hash" />

    <TextView
        android:id="@+id/resultText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/hashButton"
        android:layout_marginTop="16dp"
        android:text="@string/result"
        android:textSize="18sp" />

</RelativeLayout>
```

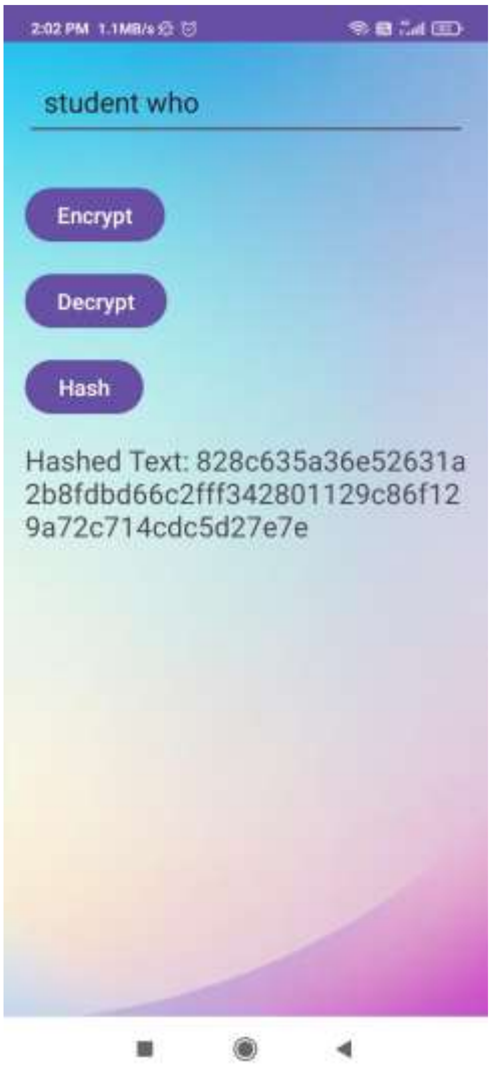
Java Code:

```
package com.example.hash;

import android.os.Bundle;
import android.text.TextUtils;
import android.util.Base64;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

public class MainActivity extends
AppCompatActivity {

    private EditText inputText;
    private TextView resultText;
    private final String AES_KEY =
"1234567890123456"; // Replace this with your
actual secret key

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        inputText = findViewById(R.id.inputText);
        resultText = findViewById(R.id.resultText);

        Button encryptButton =
findViewById(R.id.encryptButton);
        Button decryptButton =
findViewById(R.id.decryptButton);
        Button hashButton =
findViewById(R.id.hashButton);

        encryptButton.setOnClickListener(v ->
encryptText());

        decryptButton.setOnClickListener(v ->
decryptText());

        hashButton.setOnClickListener(v ->
hashText());
    }

    private void encryptText() {
        String input =
inputText.getText().toString().trim();
        if (!TextUtils.isEmpty(input)) {
            try {
                Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
                SecretKeySpec secretKeySpec = new
SecretKeySpec(AES_KEY.getBytes(), "AES");
                cipher.init(Cipher.ENCRYPT_MODE,
secretKeySpec);
                byte[] encryptedBytes =
cipher.doFinal(input.getBytes(StandardCharsets.UTF
_8));
```

```
                String encryptedText =
Base64.encodeToString(encryptedBytes,
Base64.DEFAULT);
                resultText.setText("Encrypted Text: " +
encryptedText);
            } catch (NoSuchAlgorithmException |
NoSuchPaddingException | InvalidKeyException |
                     BadPaddingException |
IllegalBlockSizeException e) {
                e.printStackTrace();
                Toast.makeText(this, "Encryption failed: "
+ e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(this, "Input text is empty",
Toast.LENGTH_SHORT).show();
        }
    }

    private void decryptText() {
        String input =
inputText.getText().toString().trim();
        if (!TextUtils.isEmpty(input)) {
            try {
                Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
                SecretKeySpec secretKeySpec = new
SecretKeySpec(AES_KEY.getBytes(), "AES");
                cipher.init(Cipher.DECRYPT_MODE,
secretKeySpec);
                byte[] decodedBytes =
Base64.decode(input, Base64.DEFAULT);
                byte[] decryptedBytes =
cipher.doFinal(decodedBytes);
                String decryptedText = new
String(decryptedBytes, StandardCharsets.UTF_8);
```

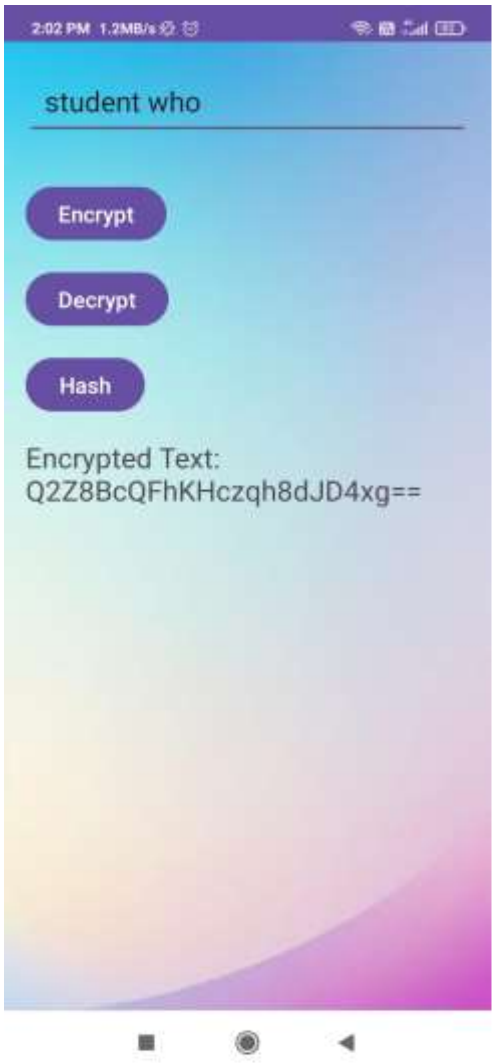
```
                resultText.setText("Decrypted Text: " +
decryptedText);
            } catch (NoSuchAlgorithmException |
NoSuchPaddingException | InvalidKeyException |
                    BadPaddingException |
IllegalBlockSizeException e) {
                e.printStackTrace();
                Toast.makeText(this, "Decryption failed: "
+ e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(this, "Input text is empty",
Toast.LENGTH_SHORT).show();
        }
    }

    private void hashText() {
        String input =
inputText.getText().toString().trim();
        if (!TextUtils.isEmpty(input)) {
            try {
                MessageDigest digest =
MessageDigest.getInstance("SHA-256");
                byte[] hashBytes =
digest.digest(input.getBytes(StandardCharsets.UTF_8
));
                StringBuilder builder = new
StringBuilder();
                for (byte b : hashBytes) {
                    builder.append(String.format("%02x",
b));
                }
                resultText.setText("Hashed Text: " +
builder.toString());
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
                Toast.makeText(this, "Hashing failed: " +
e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(this, "Input text is empty",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

Output:



student who

Hashed Text: 828c635a36e52631a2b8fdbd66c2fff342801129c86f129a72c714cdc5d27e7e



student who

Encrypted Text: Q2Z8BcQFhKHczqh8dJD4xg==

CONCLUSION

In conclusion, encryption, decryption, and hashing applications on Android devices are vital tools for safeguarding sensitive data, enhancing user privacy, and ensuring data integrity. By providing robust encryption, decryption, and hashing functionalities, these applications empower users to secure their communications, protect stored data, and verify data authenticity, ultimately contributing to a safer and more secure digital environment on Android platforms.

REFERENCE

- keycdn.com
- Simplilearn
- Cryptography and Network Security: Principles and Practice