CSE 406

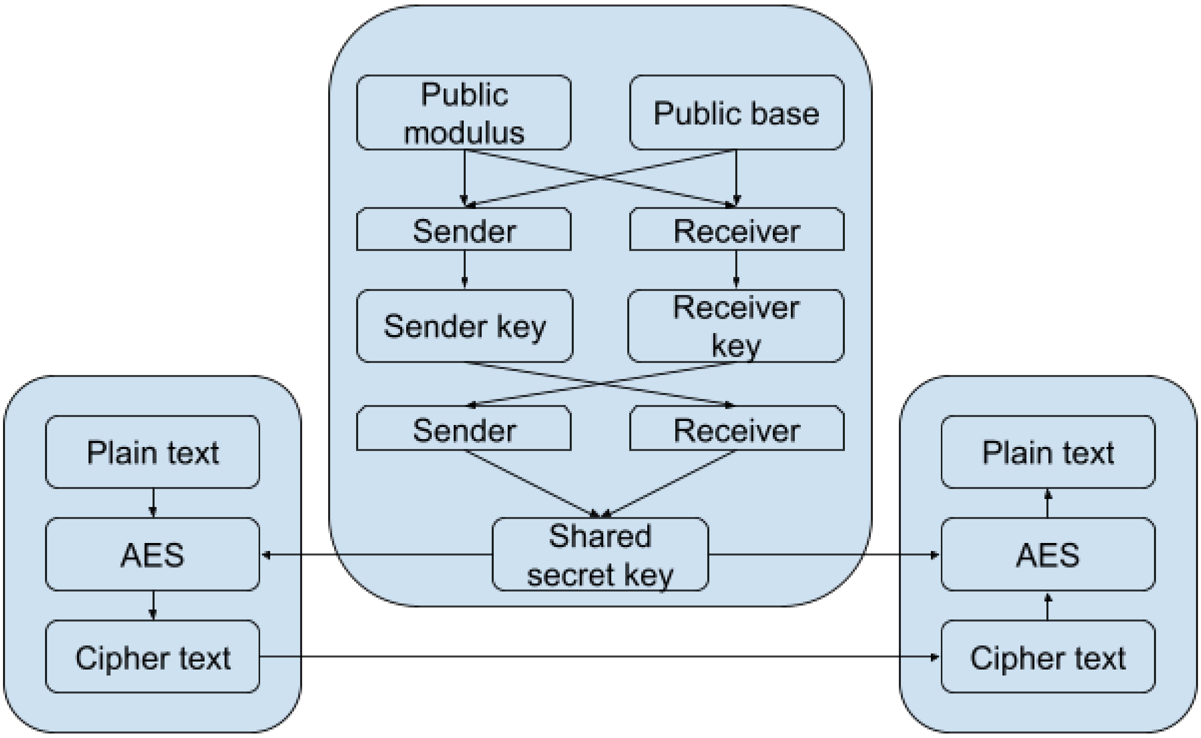
Computer Security Sessional

January 2025

Assignment 01

The goal of this assignment is to implement a cryptosystem that uses a symmetric key for encryption and decryption. The symmetric key is securely shared by the Elliptic Curve Diffie-Hellman key exchange method.

Overview of the cryptosystem



As shown in the figure, both sender and receiver first agree on a shared secret key. Sender uses this key to encrypt the plaintext using AES. The AES ciphertext will then be sent through any communication channel. Finally, the shared key is used by the receiver for the AES decryption of the ciphertext.

Overview of AES

Advanced Encryption Standard (AES) is a popular and widely adopted symmetric key encryption algorithm. AES uses repeat cycles or "rounds". There are 10, 12, or 14 rounds for keys of 128, 192, and 256 bits, respectively.

Each round of the Algorithm consists of four steps:

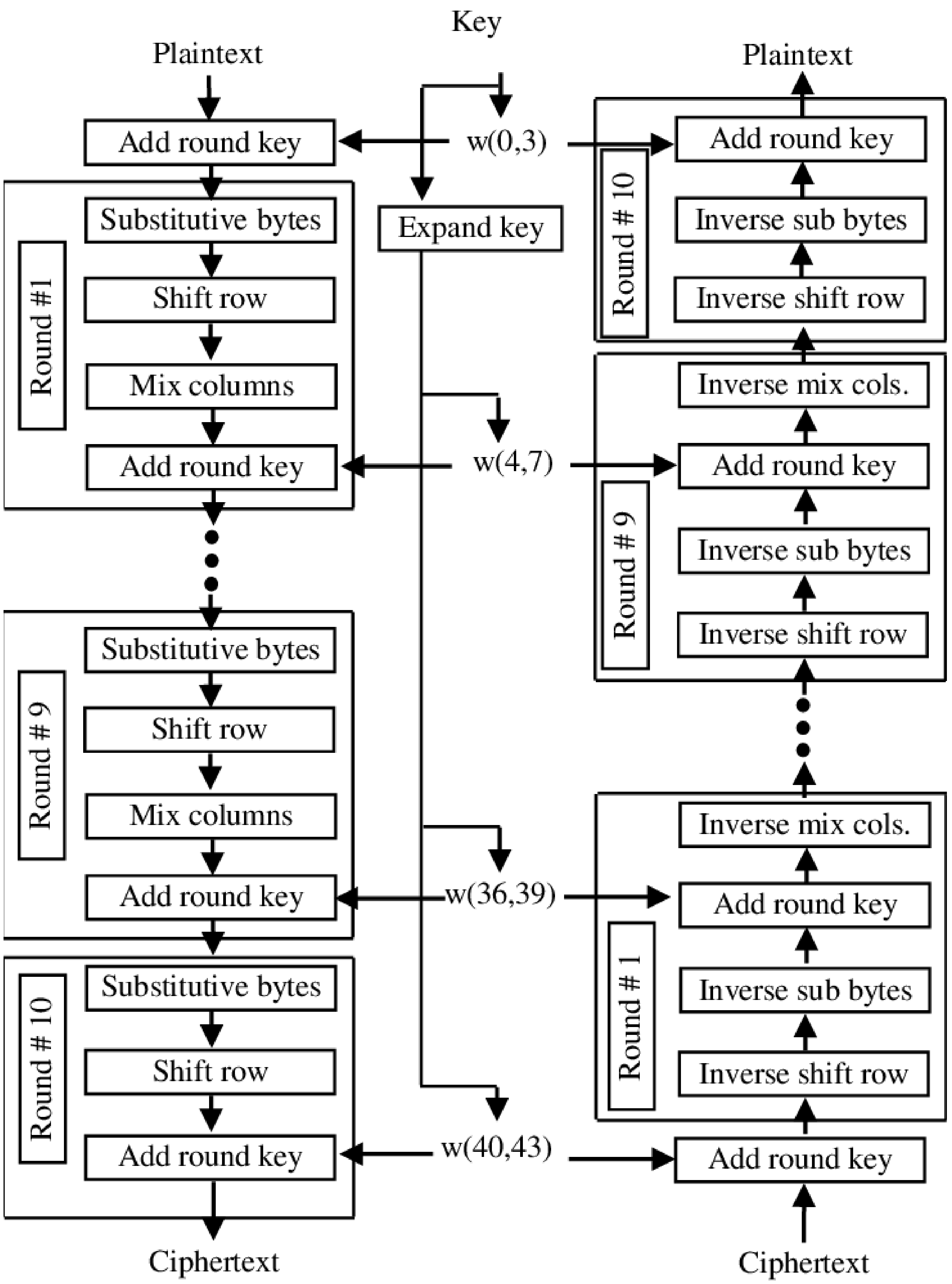
1. **subBytes:** For each byte in the array, use its value as an index into a fixed 256-element lookup table, and replace its value in the state with the byte value stored at that location in the table. You can find the table and the inverse table on this Wikipedia page (also provided in the attached code *bitvector-demo.py*).

2. **shiftRows:** Let R_i denote the i-th row in the state (4x4 matrix). For encryption, shift R_0 left 0 bytes (i.e., no change); shift R_1 left 1 byte; shift R_2 left 2 bytes; shift R_3 left 3 bytes. The shifts are circular. They do not affect the individual byte values themselves. Shift right for decryption.

3. **mixColumns:** For each column of the state, replace the column by its value multiplied by a fixed 4x4 matrix of integers (in a particular Galois Field). This is a complicated step - you can use the BitVector library to make your life easier. A *bitvector-demo.py* file has been provided to help you understand the usage of this library.

4. **addRoundKey:** XOR the state with a 128-bit round key derived from the original key *K* by a recursive process.

The final round has 3 steps, omitting the mixColumns step. The decryption of AES is the inverse of the encryption steps, with a little change in the order of the steps. Please see the figure on the next page for a better understanding.


Key
Plaintext
Plaintext
Add round key
w(0,3)
Add round key
Expand key
Round #1
Substitutive bytes
Shift row
Mix columns
Add round key
Round # 10
Inverse sub bytes
Inverse shift row
w(4,7)
Inverse mix cols.
Add round key
Round # 9
Inverse sub bytes
Inverse shift row
Round # 9
Substitutive bytes
Shift row
Mix columns
Add round key
w(36,39)
Inverse mix cols.
Add round key
Round # 1
Inverse sub bytes
Inverse shift row
Round # 10
Substitutive bytes
Shift row
Add round key
w(40,43)
Add round key
Ciphertext
Ciphertext

Overview of Elliptic Curve Diffie-Hellman

Elliptic Curve Diffie–Hellman key exchange is a mathematical method of securely exchanging cryptographic keys over a public channel and was one of the most secure public-key protocols. It consists of the following steps.

1. The group points G can be defined as an elliptic curve over a finite field. The elliptic curve has the equation $y^2 = x^3 + ax + b$; parameters a and b are agreed upon as the public parameters. This curve function may generate points G = (X_g, Y_g). Also, P, which will be used as the prime modulus and will be shared over a public medium.

2. Alice chooses a secret and random number key K_a, and performs a scalar multiplication with the generated points (K_a * G mod P = A), and sends the resulting point A = (X_A, Y_A) to Bob.

3. Bob chooses a secret and random number key K_b, and performs a scalar multiplication with the generated points (K_b * G mod P = B), and sends the resulting point B = (X_B, Y_B) to Alice.

4. Alice is now able to compute R = K_b * K_a * G mod P

5. Bob is now able to compute R = K_a * K_b * G mod P

We can mathematically show that these two values are identical, and hence, this is the shared secret key.

We will use this shared key in AES. We can only use one coordinate (usually the x coordinate as the AES key). As AES keys need to be 128, 192, or 256 bits long, we will take the bit length as a parameter, say *k*. Then, the modulus must be at least k bits long. We will also keep both K_a and K_b at least (k) bits long.

Tasks

Task 1: Independent Implementation of (128-bit) AES

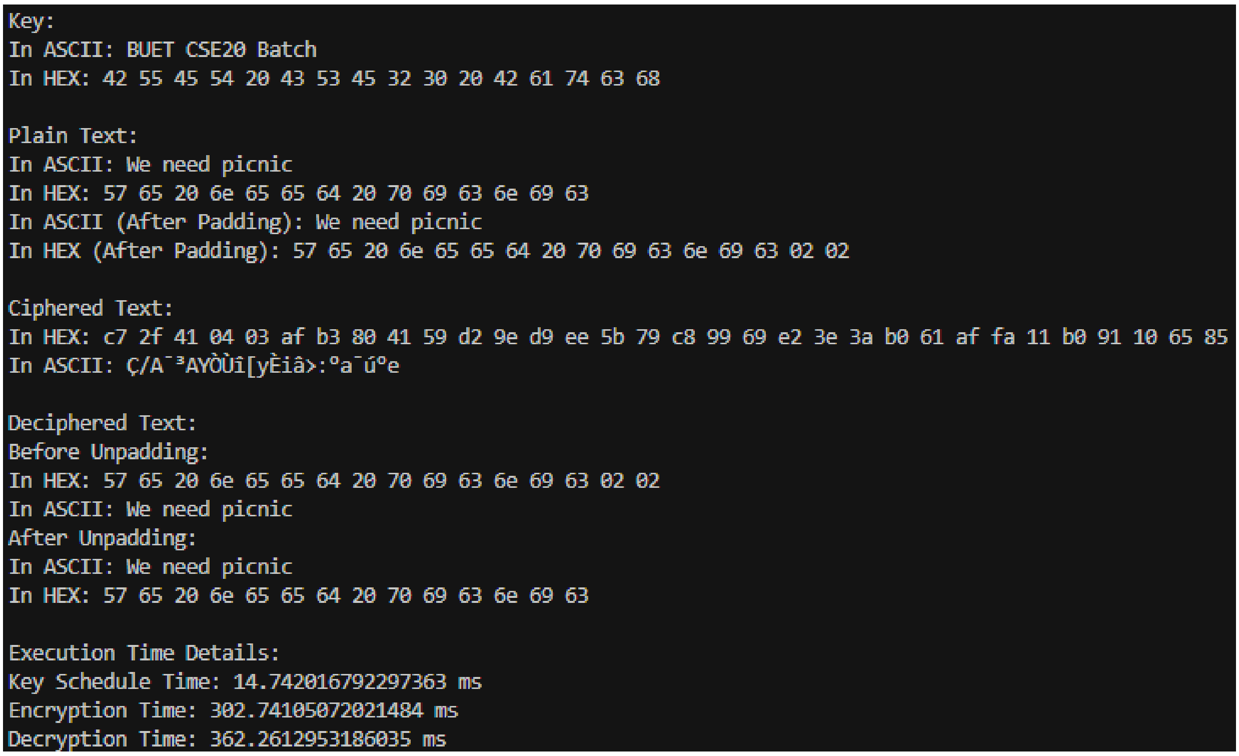
1. The encryption key will be provided by the user as an ASCII string. The key should be 16 characters long, i.e., 128 bits. **Your program should also handle keys of other lengths** (You may follow any approach for this, but you have to justify your approach during evaluation). Implement the key scheduling algorithm as described in this link. In this task, you **must** use CBC with the PKCS#7 padding scheme for text input. You may follow this text for reference.

2. **Encryption:** The program will encrypt a block of text of 128 bits with the keys generated in the previous step. If the text is longer than 128 bits, divide it into 128-bit chunks. If any chunk is smaller than 128 bits, handle it with the mentioned padding scheme.

3. **Decryption:** Decrypt the encrypted text blocks, unpad the decrypted text, and observe if they match the original text.

4. Report time-related performance in the code.

Sample I/O:



```
Key:
In ASCII: BUET CSE20 Batch
In HEX: 42 55 45 54 20 43 53 45 32 30 20 42 61 74 63 68

Plain Text:
In ASCII: We need picnic
In HEX: 57 65 20 6e 65 65 64 20 70 69 63 6e 69 63
In ASCII (After Padding): We need picnic
In HEX (After Padding): 57 65 20 6e 65 65 64 20 70 69 63 6e 69 63 02 02

Ciphered Text:
In HEX: c7 2f 41 04 03 af b3 80 41 59 d2 9e d9 ee 5b 79 c8 99 69 e2 3e 3a b0 61 af fa 11 b0 91 10 65 85
In ASCII: Ç/A¯³AYÒÙî[yÈiâ>:°a¯ú°e

Deciphered Text:
Before Unpadding:
In HEX: 57 65 20 6e 65 65 64 20 70 69 63 6e 69 63 02 02
In ASCII: We need picnic
After Unpadding:
In ASCII: We need picnic
In HEX: 57 65 20 6e 65 65 64 20 70 69 63 6e 69 63

Execution Time Details:
Key Schedule Time: 14.742016792297363 ms
Encryption Time: 302.74105072021484 ms
Decryption Time: 362.2612953186035 ms
```

N.B.

1. The Ciphered Text is the concatenation of the randomly generated Initialization Vector (IV) as the 1st 16 bytes, followed by the actual ciphertext. Your generated IV is very likely not to match the provided sample.
2. Execution time may vary depending on your implementation.
3. The output format shown above **must** be present in your output. You may show additional relevant strings/metrics for better inference if needed, which is entirely up to you. However, you are discouraged from printing too much additional text, which may disrupt the relevance of your task.

Task 2: Independent Implementation of Elliptic Curve Diffie-Hellman

1. Generate the shared parameters *G, a, b*, and *P*.
    a. You **must** generate a random prime *P* based on the key size (128, 192, or 256).
    b. Then randomly pick *a,b* from the range *[0, P)*. You **must** check for non-singularity of the curve.
    c. Then, randomly pick a point *G(x,y)* on the elliptic curve.
    d. You may refer to Euler's criterion and Tonelli–Shanks algorithm for faster computation of G (You don't need to understand the mathematical proof behind it).
    e. You may settle on a fixed initial seed to replicate your output every time.
2. Generate the private and public keys for ALICE and BOB.
    a. Choose a secret and random number key K_a, and perform a scalar multiplication with the generated points $(A = K_a * G \bmod P)$.
    b. Choose a secret and random number key K_b, and perform a scalar multiplication with the generated points $(B = K_b * G \bmod P)$.
    c. Since you are generating the elliptic curve randomly, assume your generated elliptic curve has a large order ≈ *P*.
3. Compute $R = K_a * K_b * G \bmod P$.
4. You may use Python packages for prime and random number generation only.
5. Report time-related performance in the following format. Take an average of at least 5 trials.

k	Computation Time For		
	A	B	shared key R
128			
192			
256			

Task 3: Implementation of the Whole Cryptosystem

To demonstrate the Sender and Receiver, use TCP Socket Programming. To make things easy, please refresh your brain using this guide. Suppose ALICE is the sender and BOB is the receiver. They will first agree on a shared secret key. For this, ALICE will send *a, b, G, P,* and $K_a * G \ (mod\ P)$ to BOB. BOB, after receiving these, will send $K_b * G \ (mod\ P)$ to ALICE. Both will then compute the shared secret key, store it, and inform each other that they are ready for transmission. Now, ALICE will send the AES-encrypted ciphertext (CT) to BOB using the sockets. BOB should be able to decrypt it using the shared secret key.

Bonus Tasks

1. Modify AES to support other types of files with the provision of **proper padding** (image, PDF, etc.) besides text files. In that case, in task 3, you have to transfer these files via socket as well. However, only encrypting/decrypting files will carry partial bonus marks too.
2. Generalize your AES implementation to support 192 and 256-bit keys.
3. Implementation of AES in CTR mode in addition to performance improvement via parallelization (i.e., Thread, GPU, AVX).

Breakdown of Marks

Task	Marks
Independent Implementation of AES with CBC and PKCS#7 Padding	35
Independent Implementation of Elliptic Curve Diffie-Hellman	30
Implementation of the Whole Cryptosystem Using Sockets	20
Viva	10
Correct Submission	5
Bonus: AES with other file types	5
Bonus: AES with 192 and 256-bit keys	5
Bonus: Implementation of AES in CTR mode via parallelization	10

Submission Guidelines

1. Create a directory and name it by your seven-digit student ID <2005XXX>.
2. Rename the source file to your student ID <2005XXX.py>. If you have multiple files, use this format <2005XXX_aes.py>, <2005XXX_ecc.py>, etc. Put the file(s) in the directory created in step 1.
3. Zip the directory and name it by your seven-digit student ID <2005XXX.zip>
4. Submit the zip file only.

N.B. You may need to use Python packages like *importlib* if you need to import files with a numeric prefix.

Deadline

Friday, 2 May, 2025, 10:30 pm

Plagiarism

You can easily find the implementation of AES and Elliptic Curve Diffie-Hellman on the Internet. Do not copy from any web source, friend, or your seniors. The persons involved in such activities will be penalized by **–100%** of the total marks. Also, the persons involved in such activities will be at risk of being **suspended for two terms** as per BUET’s central policy.