Blockchain

Nepal

Advanced Encryption Standard (AES)

> Algorithm : Rijndael (Authors: Vincent Rijmen + Joan Daemen)
> Selected by NIST, November 2001

Properties

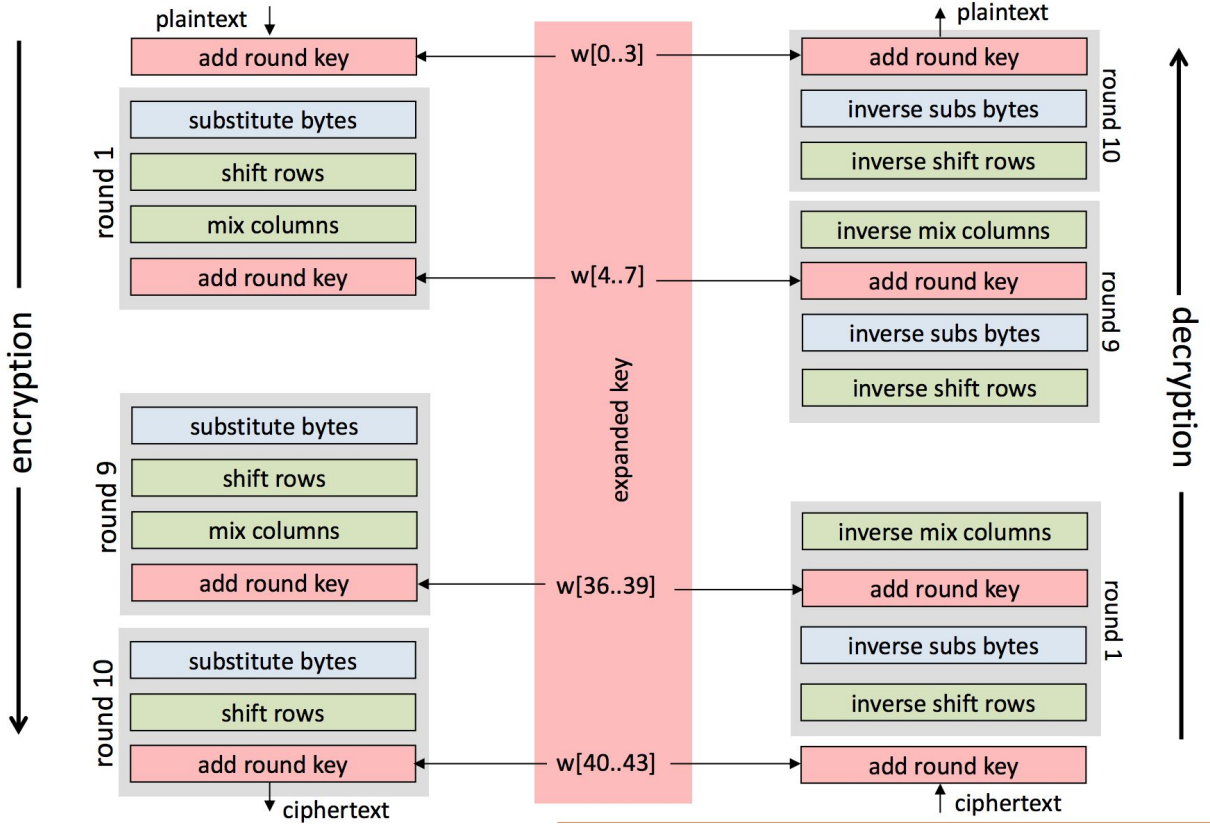
Key Size (bits)	128	192	256
I/O Size (bits)	128	128	128
No. of rounds (bits)	10	12	14
Round key size (bits)	128	128	128

> Decryption algorithm is different from encryption

> Not a Feistel Structure

> Substitution Box is 8 bit to 8 bit substitution

AES Structure



Operations

> Add Round Key
> Substitute Bytes
> Shift Rows
> Mix Columns

> Key Expansion

Refer: AES (Wikipedia)

AES in Action - Key

```
// 1. Create/generate a secret key
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
SecretKey aesKey = keyGenerator.generateKey();

// OR

// Key should be 16 bytes
SecretKeySpec aesKey = new SecretKeySpec("0123456789abcdef".getBytes(), "AES");

// 2. Message to encrypt
byte[] message = "Hello world!".getBytes();
```

AES in Action - Encryption

```
// Get cipher instance

Cipher aesCipher = Cipher.getInstance("AES");

// Encrypt the message

aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);

byte[] encryptedMessage = aesCipher.doFinal(message);

System.out.println("Encrypted message:"+new
sun.misc.BASE64Encoder().encode(encryptedMessage));
```

AES in Action - Decryption

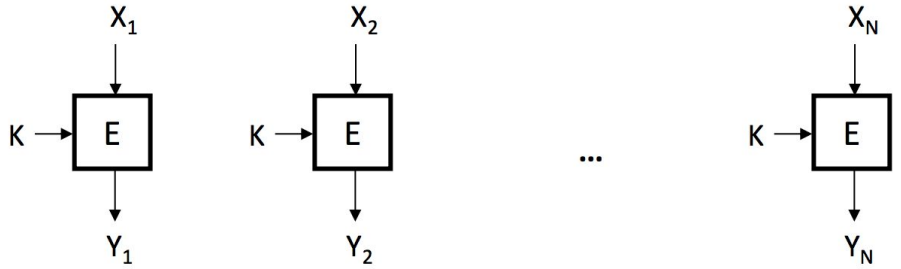
```
// Get cipher instance

Cipher aesCipher = Cipher.getInstance("AES");

// Decrypt the message

aesCipher.init(Cipher.DECRYPT_MODE, aesKey);

byte[] origMessage = aesCipher.doFinal(encryptedMessage);

System.out.println("Plain message:"+new String(origMessage));
```

Block Cipher Modes

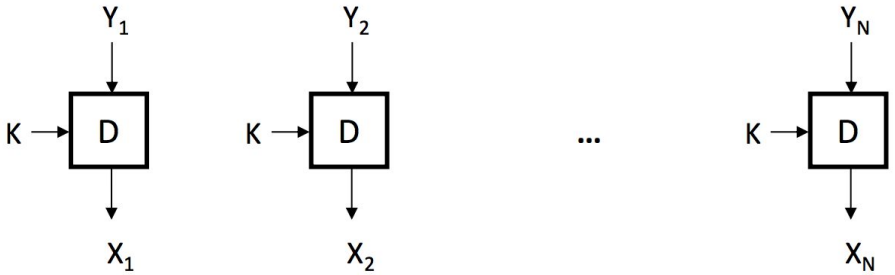
- Electronic Codebook (ECB) mode
- Cipher Block Chaining (CBC) mode
- Cipher Feedback (CFB) mode
- Output Feedback (OFB) mode
- Counter (CTR) mode

ECB Mode

- encrypt



- decrypt



ECB Mode

- Each block encrypted independently of each other
- No randomization (for same key, identical plaintext produce identical cipher text ➔ patterns in plaintext are not hidden (not good))

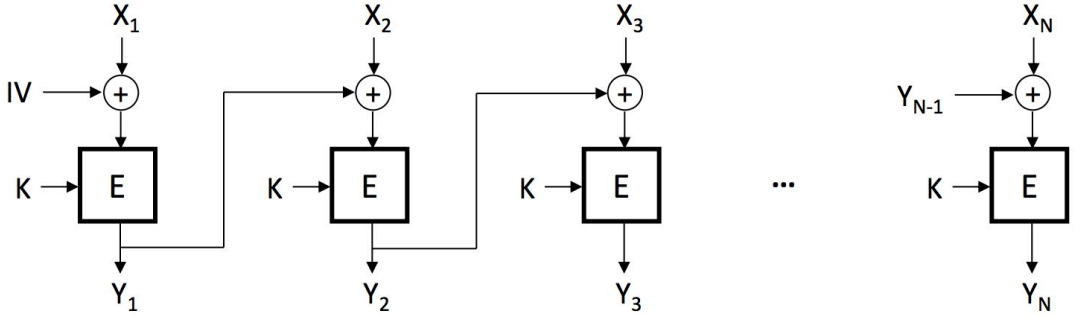
- Error Propagation: one bit error in a ciphertext block affects only the corresponding plaintext block

- Reordering of ciphertext ➔ reorders the plaintext
- Cut-Paste attack is possible

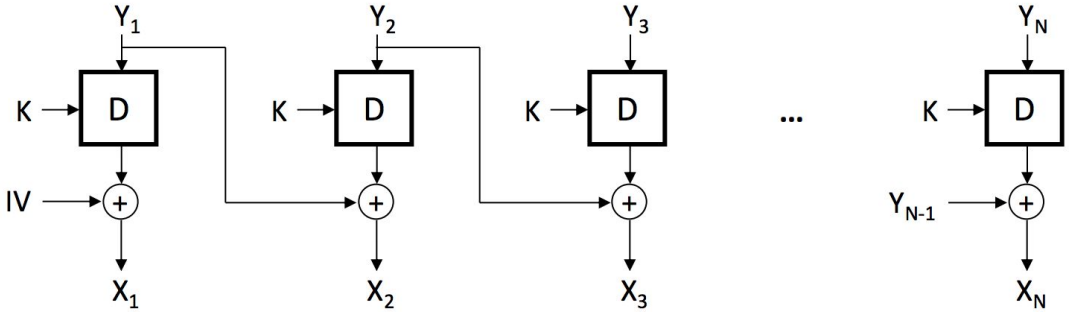
- Overall : Don’t use for messages longer than 1 block or if key is reused

CBC Mode

- encrypt



- decrypt

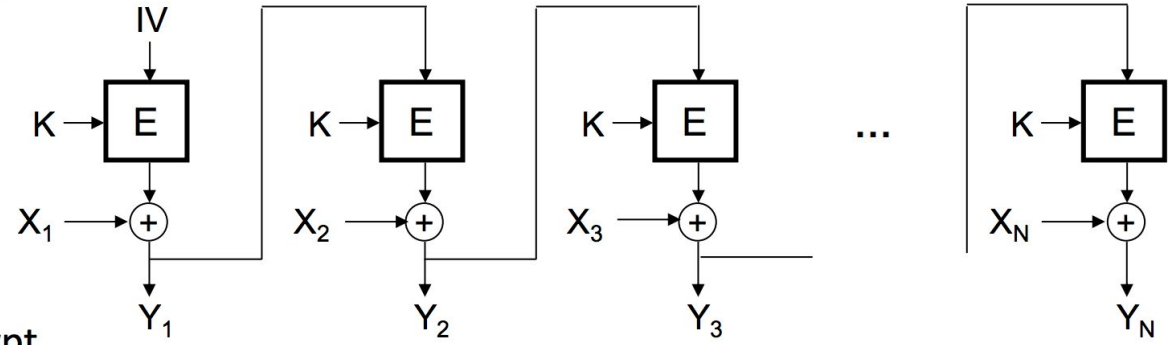


CBC Mode

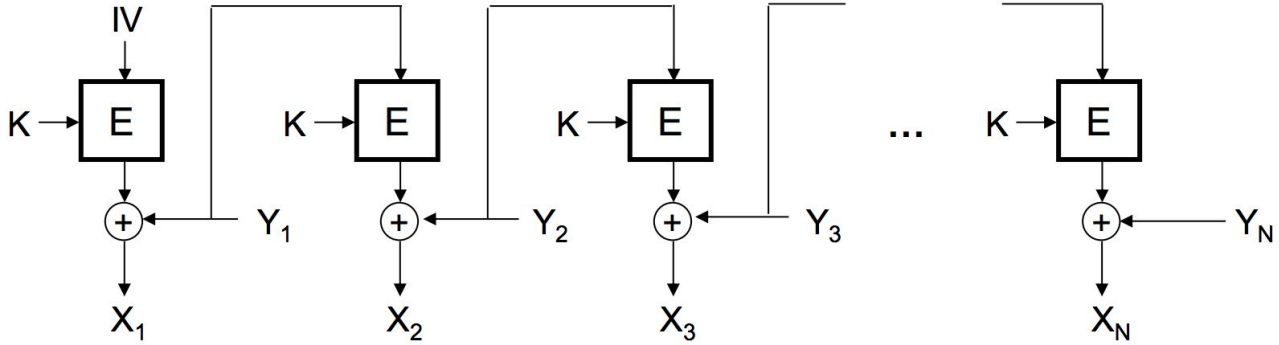
- For same key, plaintext results to different ciphertext for different IV
- Ciphertext block Yi depends on plaintext block Xi and all preceding plaintext blocks
  - Effect of reordering ➔ ??
  - Cut-Paste attack ➔ ??
- Error Propagation: one bit error in a ciphertext block Yi has an effect on the ith and (i+1)th plaintext block
- Self - Synchronizing Property
- Random access, parallel computation, no pre-computation
- IV need not be secret but should be unpredictable and non-manipulable by the attacker
  Eg. IV = $E_k(N)$, where N=nonce (non-repeating value)

CFB Mode

- encrypt



- decrypt

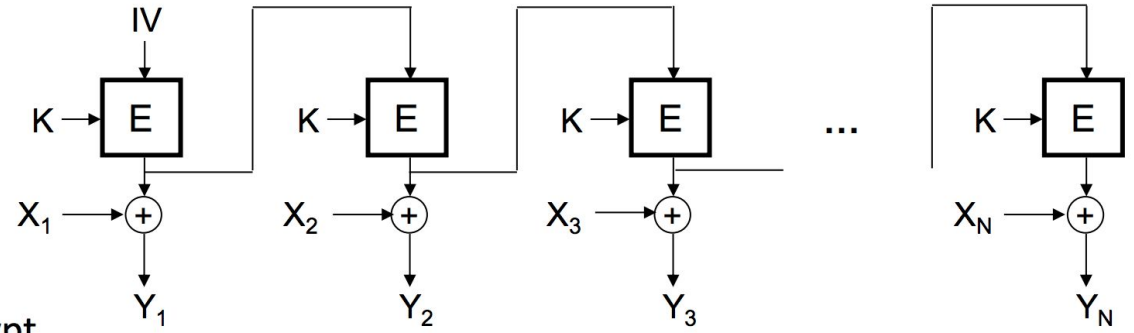


CFB Mode

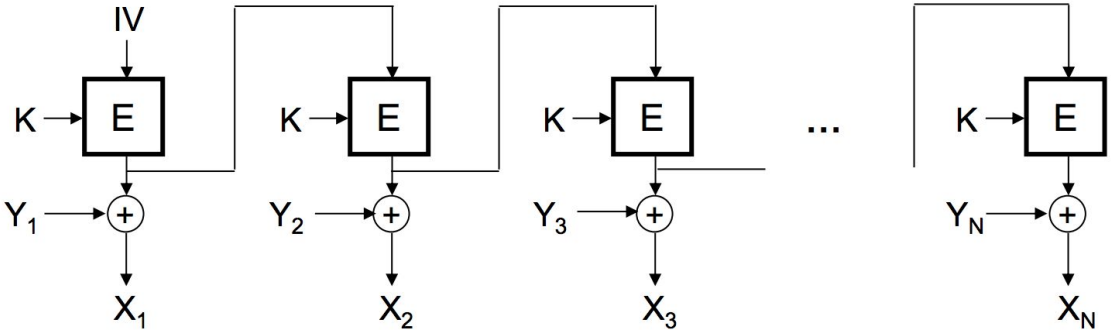
- For same key, plaintext results to different ciphertext for different IV
- Ciphertext character cj depends on plaintext character mj and all preceding characters
  - Effect of reordering ➔ ??
- Error Propagation : 1 bit error in cj ➔ incorrect mj + n next plaintext characters
- Self-Synchronizing property
- Random access, parallel computation, no pre-computation required
- IV need not be secret, can be sent in clear

OFB Mode

- encrypt



- decrypt

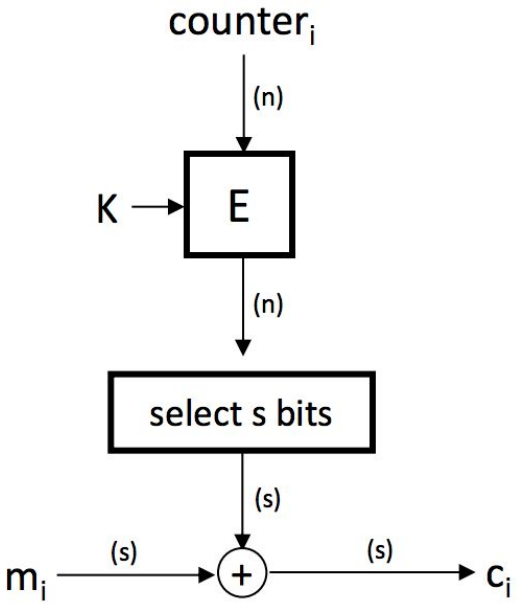


OFB

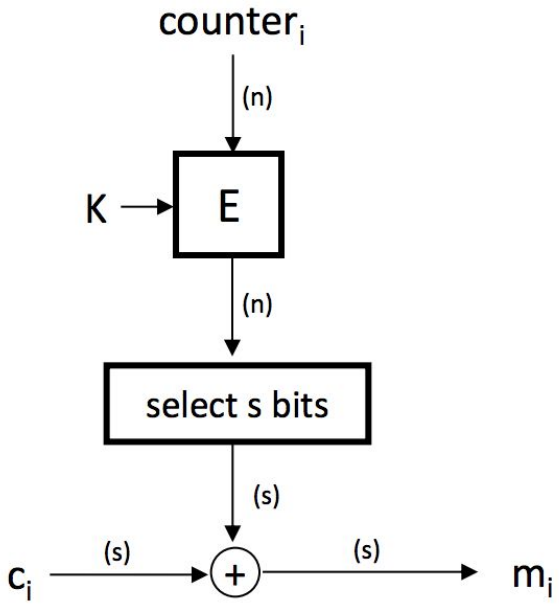
- Should use a different IV for each new message otherwise message will be encrypted with same key stream
- ciphertext character cj depends on mj only
  - Effect of re-arranging ciphertext ➜ ??
- Error Propagation : 1 bit error in cj ➜ only incorrect mj ; no other characters affected
- IV can be sent in clear, but should not be manipulable by attacker
- Need synchronization
- No random access, sequential computation only, pre-computation possible

CTR Mode

– encrypt



– decrypt



CTR Mode

- Similar to OFB, but no IV, instead counters used
- the i-th character/block can be decrypted independently of the others
  - random access (unlike OFB)
  - parallelizable (unlike OFB)
- the values to be XORed with the plaintext can be pre-computed (unlike CFB)
- it is crucial that counter values do not repeat, otherwise…
  - given $Y = E_K(ctr)+X$ and $Y' = E_K(ctr)+X'$
  - $Y + Y' = X + X'$

Summary (1/2)

- ECB ➜ only encrypt a single block, for eg. AES Key or IV
- CBC ➜ for long messages, but ...
  - Change IV (unpredictable & non-manipulable) for every message
  - Only the decryption can be parallelized, random access, no pre-computation
  - Limited error propagation, self-synchronizing property
- CFB
  - IV should be changed for every message
  - Only the decryption can be parallelized, random access, no pre-computation
  - Extended error propagation, self-synchronizing property
- OFB
  - Changing the IV for every message is very important
  - Cannot be parallelized, no random access, pre-computation is possible
  - No error propagation, needs synchronization

Summary (2/2)

- CTR ➜ best for most cases
  - Non-repeating counters are very important
  - parallelizable, random access, pre-computation
  - No error propagation, needs synchronization

- Note:
  - CFB, OFB, CTR ➜ only the encryption algorithm is used so block ciphers operating in these modes are optimized for encryption (eg. AES)
  - Encrypted message is longer than clear message due to padding
  - None of these modes provide integrity protection

Java Examples - Generating IV

```
// 16 byte IV
int ivSize = 16;
byte[] iv = new byte[ivSize];
SecureRandom random = new SecureRandom();
random.nextBytes(iv);
IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
```

Key Generation

```
// 16 Bytes key generation using hash function

MessageDigest digest = MessageDigest.getInstance("SHA-256");
digest.update(key.getBytes("UTF-8"));
byte[] keyBytes = new byte[16];
System.arraycopy(digest.digest(), 0, keyBytes, 0, keyBytes.length);
SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
```

Encryption

```
// Encrypt
byte[] msgBytes = plainText.getBytes();
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivParameterSpec);
byte[] encrypted = cipher.doFinal(msgBytes);
```

Decryption

```
// Decrypt
Cipher cipherDecrypt = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipherDecrypt.init(Cipher.DECRYPT_MODE, secretKeySpec, ivParameterSpec);
byte[] decrypted = cipherDecrypt.doFinal(encryptedBytes);
```

Exercise

1. Write a program that encrypts and decrypt a message with the provided key and write proper tests for it. Experiment with different block modes.
2. Write a program to encrypt & decrypt files present in a folder. User input folder path and the key to encrypt or decrypt the folder. Handle exceptions gracefully.