بسم الله الرحمن الرحیم

دانشکده مهندسی کامپیوتر

استاد : آقای دکتر ابوالفضل دیانت

بالاترین درجه دانایی، تشخیص اخلاق از یکدیگر و آشکار کردن اخلاق پسندیده و سرکوب اخلاق ناپسند است. حضرت علی(علیه السلام)

سید مهدی رضوی - امیرحسین مجتهدی

تیر ماه ۱۴۰۲

فهرست مطالب

فهرست تصاویر

۱ Encryption BlowFish

```
def encrypt(data):
        L = data>>32
        R = data & 0xffffffff
        for i in range(0,16):
                L = L^p[i]
                L1 = calculate(L)
                R = R^calculate(L1)
                L,R = R,L
        L,R = R,L
        L = L^p[17]
        R = R^p[16]
        encrypted = (L<<32) ^ R
        return encrypted
```

اولین کد، یک تابع برای رمزنگاری داده استفاده می‌شود. این الگوریتم بر روی بلوک‌های ثابتی از داده‌های ورودی عمل می‌کند. داده‌ی ورودی به دو نیمه ۳۲ بیتی تقسیم شده و سپس ۱۶ بار رمزنگاری انجام می‌شود. در هر بار از ۱۶ بار، مقدار "L" با یک مقدار از آرایه "p" XOR می‌شود، سپس به تابع "calculate" ارسال شده و نتیجه‌ی این تابع با "R" XOR می‌شود. سپس مقادیر "L" و "R" جابه‌جا می‌شوند. پس از انجام ۱۶ بار این عملیات، مقادیر "L" و "R" نهایی با دو مقدار از آرایه "p" XOR می‌شوند، سپس به یکدیگر متصل شده و به عنوان خروجی رمزنگاری شده برگردانده می‌شوند.

```
def calculate(L):
    temp = s[0][L >> 24]
    temp = (temp + s[1][L >> 16 & 0xff]) % (0x1<<32)
    temp = temp ^ s[2][L >> 8 & 0xff]
    temp = (temp + s[3][L & 0xff]) % (0x1<<32)
    return temp
```

در کد دوم، تابع "calculate" پیاده‌سازی شده است. این تابع یک عملیات جایگزینی ساده است که یک عدد ۳۲ بیتی را به عنوان ورودی دریافت می‌کند و یک عدد ۳۲ بیتی را به عنوان خروجی بازگردانده می‌کند. این تابع از چهار جدول جستجو با طول ۲۵۶ ورودی خود را جایگزینی می‌کند. برای اینکار، ورودی به چهار بخش ۸ بیتی تقسیم شده و هر بخش به عنوان اندیس یکی از جداول جستجو انتخاب می‌شود. سپس مقدار ۳۲ بیتی متناظر با هر بخش از جدول برگردانده می‌شود. این چهار مقدار پس از اعمال عملیات XOR و جمع، به عنوان خروجی تابع بازگردانده می‌شوند. این نوع عملیات جایگزینی یک الگوریتم پرکاربرد در بسیاری از رمزنگاری‌های بلوکی است که با ترکیب چندین مرحله از جایگزینی و جابه‌جایی به داده‌های ورودی، سطح امنیت بالایی را فراهم می‌کند.

```
def decrypt(data):
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
        L = p[i]^L
        L1 = calculate(L)
        R = R^calculate(L1)
        L,R = R,L

    L,R = R,L
    L = L^p[0]
    R = R^p[1]
    data_decrypted1 = (L<<32) ^ R
    return data_decrypted
```

داده های عدد صحیح ۶۴ بیتی را می گیرد و یک سری عملیات را برای رمزگشایی آن انجام می دهد.
ابتدا عدد صحیح ۶۴ بیتی را به دو نیمه ۳۲ بیتی L و R تقسیم می کند. سپس ۱۶ بار در یک حلقه تکرار می شود و هر بار عملیات زیر را انجام می دهد:
L XOR با مقدار p[i] نتیجه را از طریق یک تابع محاسبه عبور دهید، R XOR با نتیجه تابع محاسبه اعمال شده به نتیجه عملیات XOR قبلی پس از تکمیل حلقه، مقادیر L و R، L XORs را با مقدار p[۰] تعویض می کند، R XORs با مقدار p[۱]، و سپس آنها را به یک عدد صحیح ۶۴ بیتی ترکیب می کند. عدد صحیح حاصل، نسخه رمزگشایی شده داده های ورودی است.

```
start_time = time.time()
encrypt_data = 110

# if encrypt_data.bit_length() <=63:
#     print("Valid Input!!!")
# else:
#     print("Invalid Input!!")

data_encrypted = encrypt(encrypt_data)
print("Encrypted data is: ",data_encrypted)
print("--- %s encryption seconds ---" % (time.time() - start_time))



print("\n Hex value :  " , hex(data_encrypted) , '\n')

start_time = time.time()
data_decrypted = decrypt(data_encrypted)
print("Data after decryption is : ",data_decrypted)

print("--- %s decryption seconds ---" % (time.time() - start_time))
```

```
Encrypted data is:  15275098562551047922
--- 0.0008032321929931641 encryption seconds ---

 Hex value :   0xd3fc0d3631ef66f2

Data after decryption is :  110
--- 0.0004980564117431641 decryption seconds ---
```

شکل ۱: Algorithm blowFish

۲ Encryption DES Tripple

```
def encrypt(self, data, pad=None, padmode=None):
      ENCRYPT = des.ENCRYPT
      DECRYPT = des.DECRYPT
      data = self._guardAgainstUnicode(data)
      if pad is not None:
         pad = self._guardAgainstUnicode(pad)
      # Pad the data accordingly.
      data = self._padData(data, pad, padmode)
      if self.getMode() == CBC:
         self.__key1.setIV(self.getIV())
         self.__key2.setIV(self.getIV())
         self.__key3.setIV(self.getIV())
         i = 0
         result = []
         while i < len(data):
            block = self.__key1.crypt(data[i:i+8], ENCRYPT)
            block = self.__key2.crypt(block, DECRYPT)
            block = self.__key3.crypt(block, ENCRYPT)
            self.__key1.setIV(block)
            self.__key2.setIV(block)
            self.__key3.setIV(block)
            result.append(block)
            i += 8
         if _pythonMajorVersion < 3:
            return ''.join(result)
         else:
            return bytes.fromhex('').join(result)
      else:
         data = self.__key1.crypt(data, ENCRYPT)
         data = self.__key2.crypt(data, DECRYPT)
         return self.__key3.crypt(data, ENCRYPT)
```

این تابع رمزنگاری محافظتی سه گانه انجام می دهد.
ابتدا اطمینان حاصل می شود که داده ورودی به صورت بایت ها است نه یونیکد.
تابع به صورت اختیاری قابلیت پرکردن داده قبل از رمزنگاری را می پذیرد. کاراکتر پرکردن و الگوریتم پرکردن می تواند مشخص شود.
اگر از روش رمزنگاری CBC استفاده می شود، IV برای سه الگوریتم DES تنظیم می شود.
داده‌ها به بلوک‌هایی با ۸ بایت رمزنگاری می‌شوند.
بلوک های رمزنگاری شده به هم پیوسته می شوند و مقدار برمی گرداند
در کل این تابع رمزنگاری سه گانه DES با اختیار پرکردن انجام می دهد. که مراحلش عبارتند از:
۱- داده ورودی را به بایت تبدیل می کند
۲- اگر لازم بود داده را پر می کند
۳- برای حالت CBC، IV را تنظیم می کند
۴- داده ها را به بلوک های ۸ بایتی تقسیم و با سه الگوریتم DES رمزنگاری می کند
۵- بلوک های رمزنگاری شده را به هم متصل می کند
۶- نهایتا داده رمزنگاری شده بایت را برمی گرداند

```
def decrypt(self, data, pad=None, padmode=None):
      ENCRYPT = des.ENCRYPT
      DECRYPT = des.DECRYPT
      data = self._guardAgainstUnicode(data)
      if pad is not None:
         pad = self._guardAgainstUnicode(pad)
      if self.getMode() == CBC:
         self.__key1.setIV(self.getIV())
         self.__key2.setIV(self.getIV())
         self.__key3.setIV(self.getIV())
         i = 0
         result = []
         while i < len(data):
            iv = data[i:i+8]
            block = self.__key3.crypt(iv, DECRYPT)
            block = self.__key2.crypt(block, ENCRYPT)
            block = self.__key1.crypt(block, DECRYPT)
            self.__key1.setIV(iv)
            self.__key2.setIV(iv)
            self.__key3.setIV(iv)
            result.append(block)
            i += 8
         if _pythonMajorVersion < 3:
            data = ''.join(result)
         else:
            data = bytes.fromhex('').join(result)
      else:
         data = self.__key3.crypt(data, DECRYPT)
         data = self.__key2.crypt(data, ENCRYPT)
         data = self.__key1.crypt(data, DECRYPT)
      return self._unpadData(data, pad, padmode)
```

۳ Encryption TwoFish

```
def encrypt(K, k, S, PT):
    PT = [struct.unpack('>I', struct.pack('<I', x))[0] for x in PT]
    R = [PT[i] ^ K[i] for i in range(4)]

    for r in range(ROUNDS):
        NR = [0, 0, 0, 0]
        FR0, FR1 = F(R[0], R[1], r, K, k, S)
        NR[2] = ROR(R[2] ^ FR0, 1)
        NR[3] = ROL(R[3], 1) ^ FR1
        NR[0] = R[0]
        NR[1] = R[1]
        R = NR
        if r < ROUNDS - 1:
            R[0], R[2] = R[2], R[0]
            R[1], R[3] = R[3], R[1]

    R = [R[2], R[3], R[0], R[1]]
    R = [R[(i+2) % 4] ^ K[i+4] for i in range(4)]
    R = [struct.unpack('>I', struct.pack('<I', x))[0] for x in R]
    return R
```

```
def decrypt(K, k, S, PT):
    PT = [struct.unpack('>I', struct.pack('<I', x))[0] for x in PT]
    R = [PT[i] ^ K[i+4] for i in range(4)]

    for r in range(ROUNDS-1, -1, -1):
        NR = [0, 0, 0, 0]
        FR0, FR1 = F(R[0], R[1], r, K, k, S)
        NR[2] = ROL(R[2], 1) ^ FR0
        NR[3] = ROR(R[3] ^ FR1, 1)
        NR[0] = R[0]
        NR[1] = R[1]
        R = NR
        if r > 0:
            R[0], R[2] = R[2], R[0]
            R[1], R[3] = R[3], R[1]

    R = [R[2], R[3], R[0], R[1]]
    R = [R[(i+2) % 4] ^ K[i] for i in range(4)]
    R = [struct.unpack('>I', struct.pack('<I', x))[0] for x in R]
    return R
```

این کد پایتون یک پیاده سازی از الگوریتم رمزنگاری دو ماهی است. دو ماهی یک رمز بلوکی متقارن است که فقط ۱۶ بایت را در هر بار رمزنگاری می کند. این کد از پیاده سازی C بهینه شده بروس شنایر (https://www.schneier.com/code/twofish-optimized-c.zip) استفاده می کند. این کد شامل توابع زیر است:

to32Char(X):

این تابع یک عدد ۳۲ بیتی را به چهار بایت تبدیل می کند.

bytesTo32Bits(l):

این تابع یک لیست از بایت ها را به یک عدد ۳۲ بیتی تبدیل می کند.

ROR(x, n) , ROL(x, n):

این توابع عملگرهای شیفت راست و چپ روی بایت ها را پیاده سازی می کنند.

ROR4(x, n):

این تابع عملگر شیفت راست روی ۴ بیت را پیاده سازی می کند.

polyMult(a, b) , gfMult(a, b, modulus):

این توابع ضرب دو عدد در حلقه های گالوآ را پیاده سازی می کنند.

gfMod(t, modulus):

این تابع باقی مانده یک عدد در حلقه گالوآ را پیدا می کند.

matrixMultiply(md, sd, modulus):

این تابع ضرب دو ماتریس در حلقه گالوآ را پیدا می کند.

printRoundKeys(K):

این تابع کلیدهای دور رمزنگاری را نمایش می دهد.

keySched(M, N):

این تابع برنامه کلید رمزنگاری را از چکیده کلید M با طول N بایت تولید می کند.

makeKey(Me, Mo, k):

این تابع کلیدهای دور رمزنگاری را با استفاده از تابع h و ضرب MDS تولید می کند.

h(X, L, k):

این تابع چکیده X با استفاده از لغات L و جدول های Q۰ و Q۱ را پیدا می کند.

بخش دوم کد را دریافت کردم. این کد شامل توابع زیر است:

Qpermute(x, Q):

این تابع یک عدد ۴ بیتی را با استفاده از جدول Q معکوس می کند.

F(R0, R1, r, K, k, S):

این تابع تابع F در الگوریتم دو ماهی را پیاده سازی می کند. این تابع دو عدد ۳۲ بیتی R۰ و R۱ را گرفته و با استفاده از تابع g و کلیدهای دور K، دو عدد ۳۲ بیتی F۰ و F۱ را برمی گرداند.

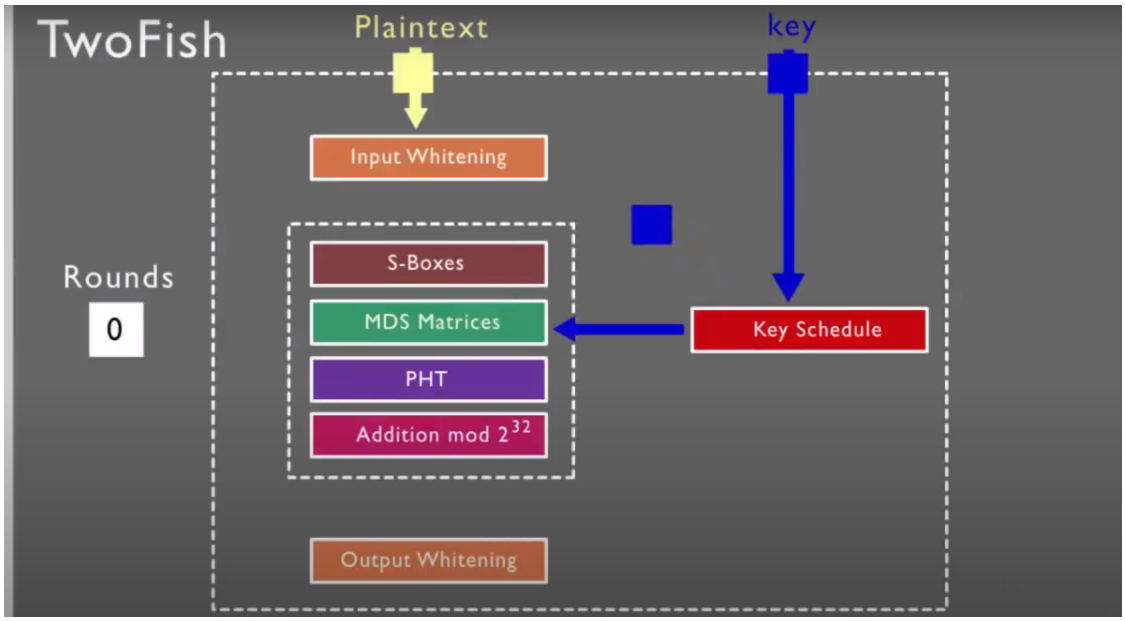
encrypt(K, k, S, PT):

این تابع یک بلوک ۱۶ بایتی PT را با استفاده از کلید K، طول کلید k و لغات S رمزنگاری می کند. این تابع شامل سه مرحله است: سفید کردن ورودی، دور های رمزنگاری و سفید کردن خروجی.

decrypt(K, k, S, PT):

این تابع یک بلوک ۱۶ بایتی PT را با استفاده از کلید K، طول کلید k و لغات S رمزگشایی می کند. این تابع شامل سه مرحله است: سفید کردن ورودی، دور های رمزگشایی و سفید کردن خروجی.

testKey(K, k, S):
این تابع یک آزمون ساده برای الگوریتم دو ماهی را اجرا می کند. این تابع یک بلوک صفر را با استفاده از کلید K، طول کلید k و لغات S رمزنگاری و رمزگشایی می کند و نتایج را نمایش می دهد.
dispLongList(v):
این تابع یک لیست از اعداد ۳۲ بیتی را به صورت ۱۶ رقم هگزادسیمال نمایش می دهد.
Itest128() , Itest192() , Itest256():
این توابع آزمون های مختلف برای الگوریتم دو ماهی با طول های کلید مختلف را اجرا می کنند. این توابع هر بار چکیده خروجی قبل را به عنوان کلید جدید استفاده می کنند.
bench():
این تابع زمان لازم برای رمزنگاری و تولید کلید با الگوریتم دو ماهی را سنجش می کند.
تمامی کدهای بالا در فایل main.py موجود است.
فلوچارت کلی الگوریتم در تصویر زیر مشهود است.



شکل ۲: twoFish Algorithm

۴ IDEA

modAdd(a, b)

این تابع جمع ماژولار دو عدد ۱۶ بیتی را محاسبه می کند.

modMultiply(a, b)

این تابع ضرب ماژولار دو عدد ۱۶ بیتی را محاسبه می کند.

plainSplit(x)

این تابع یک عدد ۶۴ بیتی را به چهار قسمت ۱۶ بیتی تقسیم می کند و آنها را به عنوان خروجی برمی گرداند.

keyGeneration(k)

این تابع یک کلید ۱۲۸ بیتی را گرفته و با استفاده از توابع جانبی، ۵۲ زیرکلید ۱۶ بیتی را تولید می کند و آنها را در یک لیست برمی گرداند.

addInverse(k)

این تابع معکوس جمع یک عدد ۱۶ بیتی را محاسبه می کند.

multiplyInverse(a)

این تابع معکوس ضرب یک عدد ۱۶ بیتی را با استفاده از الگوریتم اقلیدس توانایی و قضایای نظریه اعداد، محاسبه می کند.

power(x, y, m)

این تابع x به توان y را در حلقه ماژولار m با استفاده از الگوریتم توانایی سریع، محاسبه می کند.

gcd(a, b)

این تابع بزرگترین مقسوم علیه مشترک دو عدد را با استفاده از الگوریتم اقلیدس، محاسبه می کند.

invKeyGeneration(k)

این تابع لیست زیرکلید های k را گرفته و با استفاده از توابع جانبی، لیست زیرکلید های معکوس را برای فرآیند رمزگشایی، برمی گرداند.

round(p, k1, k2, k3, k4, k5, k6)

این تابع یک دور از فرآیند رمزنگاری و رمزگشایی الگوریتم IDEA را پیاده سازی می کند. این تابع ۶ زیرکلید و ۶۴ بایت داده را در ورودی می گیرد و با استفاده از عمل های جمع، ضرب، XOR و جابجایی، خروجی ۶۴ بایت داده را برمی گرداند.

finalRound(p, k1, k2, k3, k4)

این تابع دور نهایی فرآیند رمزنگاری و رمزگشایی الگوریتم IDEA را پیدا سازید. این تابع ۴ زیرکلید و ۶۴ بایت داده را در ورودید میرید و با استفاده از عمل های جمع، ضرب و جابجایید خروجید ۶۴ بایت داده را برمیریدانید.

encrypt(p, k)

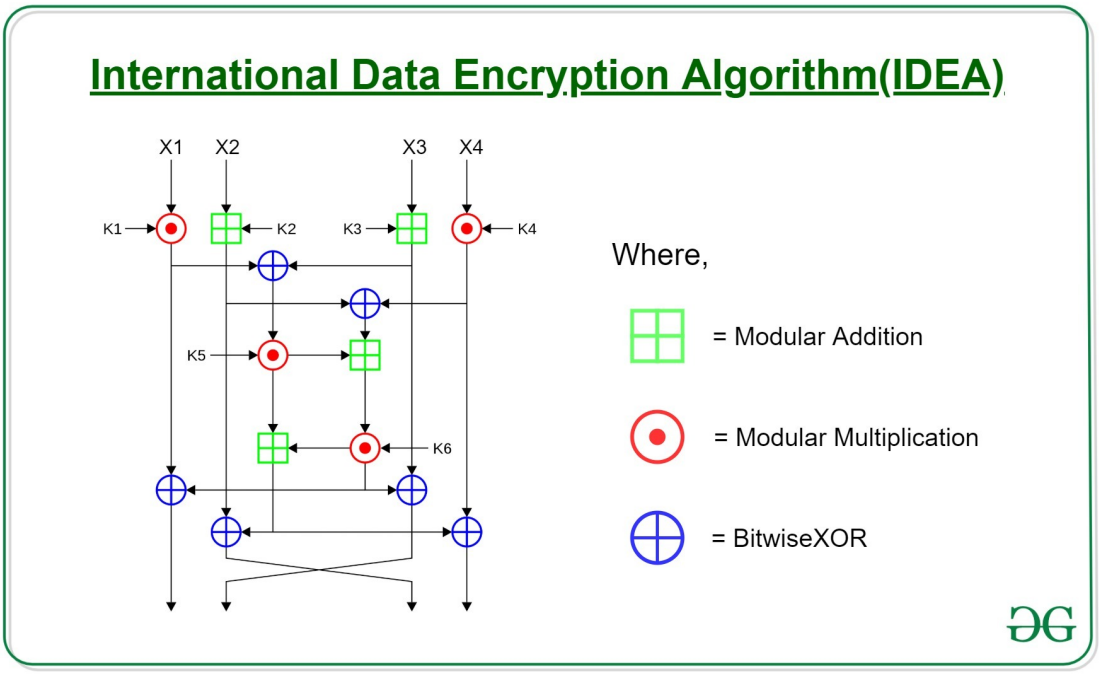
این تابع فرآیند کامل رمزنگارید الگوریتم IDEA را انجام میدهد. این تابع یک کلید ۱۲۸ بیتی و ۶۴ بایت داده را در ورودی می گیرد و با استفاده از توابع جانبی، خروجی ۶۴ بایت داده رمزنگاری شده را برمی گرداند.

decrypt(c, k)

این تابع فرآیند کامل رمزگشایی الگوریتم IDEA را انجام می دهد. این تابع یک کلید ۱۲۸ بیتی و ۶۴ بایت داده رمزنگاری شده را در ورودی می گیرد و با استفاده از توابع جانبی، خروجی ۶۴ بایت داده رمزگشایی شده را برمی گرداند.

```
def encrypt(p, k):
    sk = keyGeneration(k)
    for i in range(0, 8):
        p = round(p, sk[i * 6], sk[i * 6 + 1], sk[i * 6 + 2],
                  sk[i * 6 + 3], sk[i * 6 + 4], sk[i * 6 + 5])
        # print(hex(p))
    p = finalRound(p, sk[48], sk[49], sk[50], sk[51])
    return p
```

```
def decrypt(c, k):
    sk = keyGeneration(k)
    sk = invKeyGeneration(sk)
    for i in range(0, 8):
        c = round(c, sk[i * 6], sk[i * 6 + 1], sk[i * 6 + 2],
                  sk[i * 6 + 3], sk[i * 6 + 4], sk[i * 6 + 5])
        # print(hex(c))
    c = finalRound(c, sk[48], sk[49], sk[50], sk[51])
    return c
```



شکل ۳: Algorithm IDEA

۵ AES

بیایید الگوریتم‌های AES (استاندارد رمزگذاری پیشرفته) و Twofish را بر اساس اندازه کلید، اندازه بلوک و سطوح امنیتی مقایسه کنیم.

۱. اندازه کلید: - AES:

AES اندازه های کلیدی ۱۲۸، ۱۹۲ و ۲۵۶ بیت را پشتیبانی می کند.

- Twofish:

Twofish از اندازه های کلیدی ۱۲۸، ۱۹۲ و ۲۵۶ بیتی پشتیبانی می کند.

۲. اندازه بلوک: - AES: AES دارای اندازه بلوک ثابت ۱۲۸ بیت (۱۶ بایت) است.

- Twofish: Twofish دارای اندازه بلوک متغیر است که می تواند ۱۲۸، ۱۹۲ یا ۲۵۶ بیت (۱۶، ۲۴، یا ۳۲ بایت) باشد.

۳. سطح امنیتی: - AES:

AES به طور گسترده توسط رمزنگاران در سراسر جهان مورد مطالعه و تجزیه و تحلیل قرار گرفته است. به طور گسترده ای امن در نظر گرفته می شود و استاندارد رمزگذاری است که توسط دولت ها و سازمان ها در سطح جهانی استفاده می شود. AES-۱۲۸ از نظر محاسباتی در برابر حملات brute-force ایمن در نظر گرفته می شود.

- Twofish:

Twofish نیز مورد تجزیه و تحلیل کامل قرار گرفته و امن محسوب می شود. با این حال، در مقایسه با AES مورد بررسی گسترده کمتری قرار گرفته است. دو ماهی به طور کلی در نظر گرفته می شود که سطح بالایی از امنیت را ارائه می دهد، قابل مقایسه با AES.

۴. عملکرد: - AES:

AES به دلیل کارایی خود در اجرای نرم افزار و سخت افزار شناخته شده است. این بسیار بهینه شده است و به طور گسترده توسط پلتفرم ها و کتابخانه های مختلف پشتیبانی می شود.

- Twofish:

Twofish نیز کارآمد است اما ممکن است در سناریوهای خاص عملکرد کمی در مقایسه با AES داشته باشد.

۵. ساختار الگوریتم: - AES:

AES یک الگوریتم کلید متقارن مبتنی بر ساختار شبکه جایگزینی-جایگزینی (SPN) است. این شامل چندین دور عملیات تعویض بایت، جابجایی و اختلاط است.

- Twofish:

Twofish یک الگوریتم کلید متقارن بر اساس ساختار شبکه Feistel است. از ترکیبی از جعبه های S وابسته به کلید، جایگشت های وابسته به کلید و عملیات اختلاط استفاده می کند.

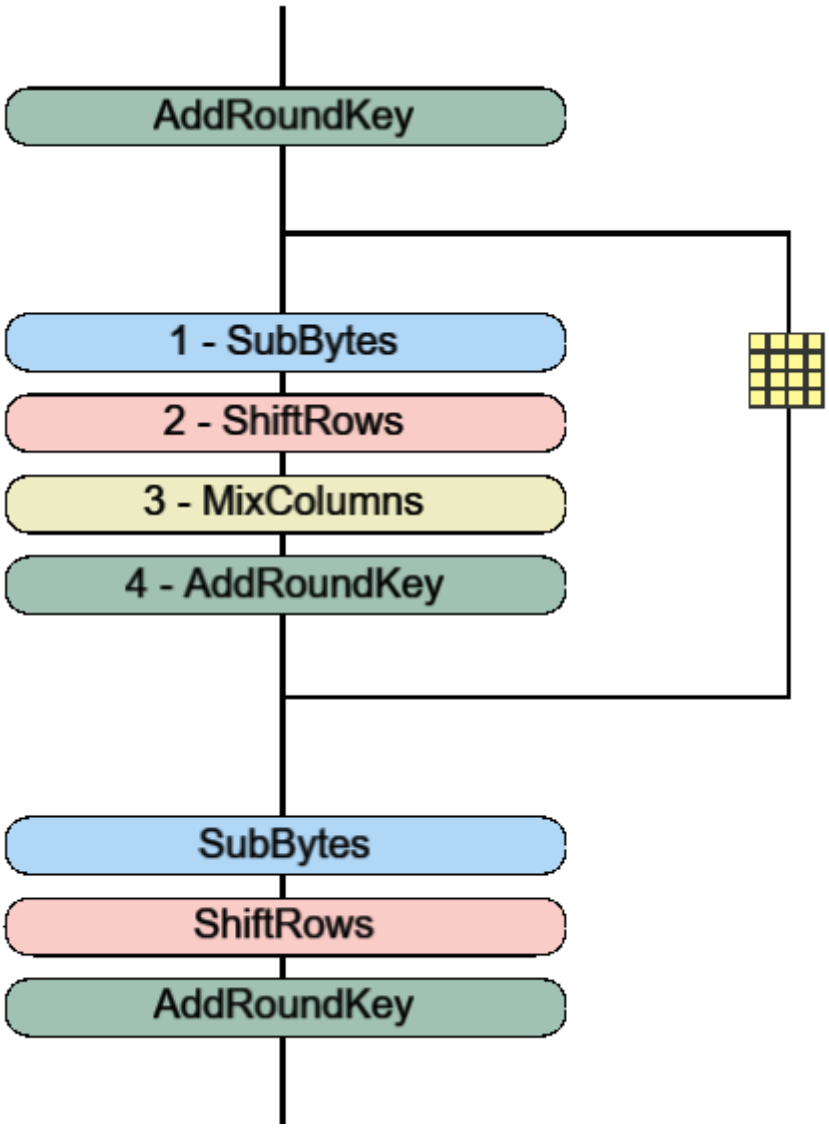
۶. پذیرش الگوریتم: - AES:

AES به عنوان الگوریتم رمزگذاری استاندارد توسط سازمان های مختلف از جمله دولت ایالات متحده پذیرفته شده است. به طور گسترده در برنامه ها و پروتکل هایی که نیاز به رمزگذاری ایمن دارند استفاده می شود.

- Twofish:

Twofish در مرحله انتخاب AES فینالیست شده است اما به عنوان استاندارد انتخاب نشده است. با این حال، هنوز به عنوان یک جایگزین قوی در نظر گرفته می شود و در برنامه های مختلف و محصولات امنیتی مورد استفاده قرار گرفته است.

هر دو AES و Twofish الگوریتم های رمزگذاری ایمن هستند که برای اکثر برنامه ها مناسب هستند. AES به طور گسترده‌تری پذیرفته شده و استاندارد شده است، در حالی که Twofish یک گزینه جایگزین با ویژگی‌های امنیتی مشابه ارائه می‌کند. انتخاب بین این دو به عواملی مانند الزامات سازگاری، در دسترس بودن الگوریتم و نیازهای امنیتی خاص بستگی دارد. هنگام انتخاب یک الگوریتم رمزگذاری، توجه به اکوسیستم رمزنگاری اطراف، پشتیبانی پلت فرم، و الزامات مورد استفاده فردی مهم است.



شکل ۴: Algorithm AES

۶ مقایسه آسیب‌پذیری‌ها

مطمئن! بیایید الگوریتم های رمزگذاری ،AES Twofish و IDEA را با هم مقایسه کنیم:

۱. استاندارد رمزگذاری پیشرفته(AES) : - اندازه کلید: AES از سه اندازه کلید ۱۲۸ بیتی، ۱۹۲ بیتی و ۲۵۶ بیتی پشتیبانی می کند و سطح بالایی از امنیت را ارائه می دهد.

- اندازه بلوک: AES در اندازه بلوک ثابت ۱۲۸ بیتی عمل می کند که ایمن در نظر گرفته می شود.
- امنیت: AES تحت تجزیه و تحلیل گسترده قرار گرفته است و به طور گسترده ای به عنوان یک الگوریتم رمزگذاری امن مورد استفاده قرار گرفته است. در برابر حملات رمزنگاری شناخته شده مقاوم است و سابقه امنیتی قوی دارد.
- کارایی: AES در اجرای سخت افزار و نرم افزار بسیار کارآمد است و برای طیف وسیعی از برنامه ها مناسب است.
- انعطاف پذیری: AES از حالت های مختلف عملکرد پشتیبانی می کند و امکان سفارشی سازی بر اساس الزامات امنیتی خاص را فراهم می کند.
- پذیرش: AES پرکاربردترین الگوریتم رمزگذاری متقارن است و برای رمزگذاری همه منظوره توصیه می شود.

۲. دو ماهی: - اندازه کلید: Twofish از اندازه های کلیدی از ۱۲۸ بیت تا ۲۵۶ بیت پشتیبانی می کند که انعطاف پذیری و مقیاس پذیری را فراهم می کند.

- اندازه بلوک: Twofish در اندازه بلوک ثابت ۱۲۸ بیتی کار می کند.
- امنیت: Twofish به طور گسترده مورد تجزیه و تحلیل قرار گرفته است و یک الگوریتم رمزگذاری امن در نظر گرفته می شود. هیچ آسیب پذیری عملی شناخته شده ای ندارد.
- کارایی: Twofish ویژگی های عملکردی خوبی دارد و می توان آن را به صورت کارآمد هم در نرم افزار و هم در سخت افزار پیاده سازی کرد.
- انعطاف پذیری: Twofish درجه بالایی از انعطاف پذیری را ارائه می دهد و امکان اندازه های کلیدی و اندازه های مختلف بلوک را فراهم می کند. همچنین از حالت های مختلف عملکرد پشتیبانی می کند.
- پذیرش: در حالی که Twofish یک الگوریتم رمزگذاری مورد توجه است، در مقایسه با AES کمتر مورد استفاده قرار می گیرد. با این حال، در برخی از برنامه‌ها استفاده شده است که ویژگی‌های آن با الزامات خاص مطابقت دارد.

۳. الگوریتم رمزگذاری بین المللی داده (IDEA) :

- اندازه کلید: IDEA از یک اندازه کلید ثابت ۱۲۸ بیتی استفاده می کند.
- اندازه بلوک: IDEA در اندازه بلوک ثابت ۶۴ بیت عمل می کند.
- امنیت: IDEA به طور گسترده مورد مطالعه و تحلیل قرار گرفته است. در حالی که هیچ آسیب‌پذیری عملی شناخته‌شده‌ای ندارد، به دلیل اندازه بلوک کوچکتر و اندازه کلید محدود، امنیت کمتری نسبت به AES و Twofish دارد.
- کارایی: IDEA در اجرای نرم افزار و سخت افزار کارآمد است.
- پذیرش: IDEA در گذشته به طور گسترده مورد استفاده قرار می گرفت، اما به نفع الگوریتم های رمزگذاری مدرن تر مانند ،AES پذیرش کاهش یافته است. هنوز هم در برخی از سیستم های قدیمی استفاده می شود.

به طور خلاصه، AES به دلیل امنیت قوی، کارایی و پشتیبانی گسترده، رایج ترین الگوریتم رمزگذاری پذیرفته شده و توصیه شده است. Twofish همچنین یک الگوریتم امن و انعطاف پذیر است که طیف وسیعی از اندازه های کلیدی و اندازه بلوک را ارائه می دهد. ،IDEA در حالی که ایمن تلقی می شود، از نظر اندازه بلوک و اندازه کلید دارای محدودیت هایی است که منجر به کاهش پذیرش آن در سال های اخیر شده است.