하이브리드 암호화를 지원하는 멀티 스레드 소켓 통신 프로그램

Multi-threaded Socket Program with Hybrid Crypto

요 약

AES-128과 RSA-2048을 활용한 소켓 통신 프로그램으로 평소에는 채팅을 할 수 있으며 서버가 암호화된 파일 수신자, 클라이언트가 암호화된 파일 송신자로 역할을 하게 된다.
암호화된 파일을 주고받는 과정은 1) 서버가 RSA 공개키를 클라이언트에게 전송, 2) 클라이언트가 암호화된 AES키를 서버에게 전송, 3) 클라이언트가 암호화된 파일을 서버에게 전송의 과정을 거친다.

1. 서 론

Multi-thread를 사용한 TCP 소켓 통신 프로그램은 서버와 클라이언트가 sender와 receiver 함수를 thread로 실행하게 된다. 이때 소켓 통신을 통해 데이터를 주고받으면 bytes의 형태로 데이터를 받게 되는데 이 데이터를 중간에 가로채게 되면 데이터의 내용이 그대로 노출이 되는 것을 방지하기 위해 디자인되었다.

데이터를 받고자 하는 사람이 서버로서 통신을 열어 두고 데이터를 보내고자 하는 사람이 클라이언트로서 통신을 진행하며 서버는 RSA 공개키 / 개인키, 클라이언트는 AES 키를 생성하게 된다. 그 후 서버는 클라이언트에 RSA 공개키를 전달하고, 클라이언트는 AES로 암호화된 데이터와 RSA 공개키로 암호화된 AES 키를 서버에게 보낸다. 서버는 RSA 개인키로 암호화된 AES 키를 복호화 한 후 그 AES 키로 암호화된 데이터를 복호화 한다. 이때 통신을 가로채는 사람이 획득할 수 있는 데이터는 암호화된 데이터, 암호화된 AES 키, RSA 공개키 이므로 이를 통해 원본 데이터를 획득할 수 없게 한다.

이 프로그램은 평시에는 서버와 클라이언트의 채팅을 작동하도록 하며, 서버의 RSA 공개키 전송 명령, 클라이언트의 AES 키 전송 명령이 실행된 이후로는 계속해서 bmp 형식의 파일을 전송할 수 있게 된다. 전송되는 bmp 파일들은 자동으로 클라이언트의 디렉토리에 암호화된 상태로, 서버의 디렉토리에 암호화된 상태화 복호화된 상태로 저장된다.

2. A E S [1]

이 프로그램에서 사용되는 AES는 CBC 모드이다. CBC 모드는 Cipher Block Chaining의 약자로 앞에서 사용한 plaintext가 뒤의 plaintext에 영향을 미치는 방식으로 암호화된 이전 plaintext와 현재의 plaintext를 XOR하여 암호화하는 방식을 말한다. 이렇게 하게 되면 앞의 내용이 뒤에 내용에 영향을 미치는 'avalanche effect'가 발생하여 일반적으로 사용하는 ECB 모드를 사용했을 때 발생하는 bmp 파일 같은 이미지 데이터의 패턴이 보이는 현상(그림 1)을 방지한다. 또한 프로그램이 암호화하는 데이터가 bmp 파일로 데이터의 사이즈가 크기 때문에 속도의 향상을 위해 S-Box, Inverse S-Box, RCon, 2, 3, 9, 11, 13, 14의 Multiplier는 미리 하드코딩되어있다. 또한 연산을 함에 있어 $f(x) = x^8 + x^4 + x^3 + x + 1$ 이라는 irreducible polynomial을 사용하게 되는데 비트연산을 하는 중에 오버 플로우가 발생하면 $f(x)$와 XOR연산을 하는 방식이다.

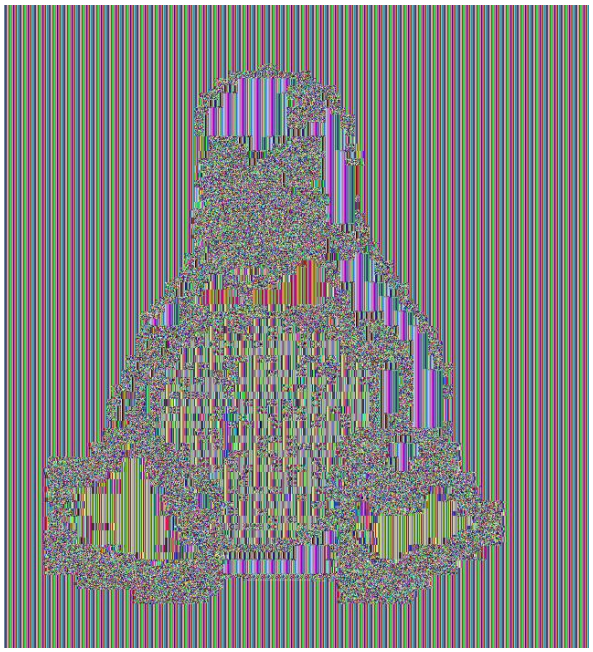

그림 1 ECB 모드로 암호화된 bmp 파일

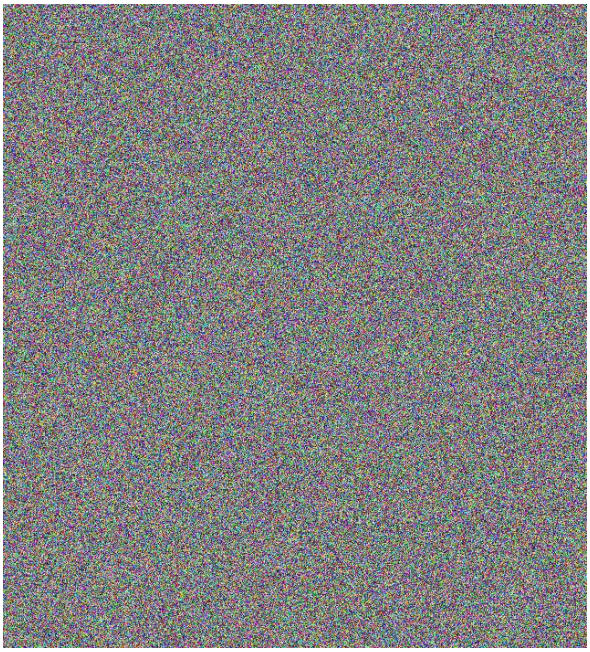

그림 2 CBC 모드로 암호화한 bmp 파일

그림 3 복호화한 bmp 파일

2.1 S-Box / Inverse S-Box

S-Box는 0x00~0xFF의 bytes를 각기 다른 0x00~0xFF로 대응시키는 과정으로 원래의 bytes의 $f(x)$에 대한 역원 구한 후 그 값을 다음 행렬식으로 연산한 결과이다. 여기서 수행되는 더하기 연산 또한 XOR 연산이다.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

반대로 Inverse S-Box는 S-Box에서의 정 반대로 다음 행렬식을 연산한 후 역원을 구하는 방식이다.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2.2 RCon

RCon은 round마다 key를 더욱더 섞어 주는 방식이며 0x01, 0x02, ...으로 계속 2배씩 곱하여 round만큼 만들어준 배열이다. 이때 0x80을 2배하면 역시 오버 플로우가 발생하므로 $f(x)$을 XOR연산하여 오버 플로우를 제거한다. AES-128은 총 10라운드이므로 10개의 RCon을 생성한다.

2.3 KeyExpansion

KeyExpansion은 key를 round 수만큼 추가로 생성하는 단계로 RotWord, SubWord, RCon, Next Round key 단계를 거쳐 진행됩니다. RotWord는 이전 round의 마지막 1 word를 왼쪽으로 1 shift하는 연산이며, SubWord는 shift한 1 word를 S-Box 연산을 하고, RCon은 각 라운드별 RCon을 XOR하며, Next round key은 위 3단계를 거친 이전 round의 마지막 1 word와 이전 round의 첫 word를 XOR한다. 이렇게 만들어진 round의 첫 word와 이전 round의 두번째 word를 XOR하면 두번째 word가 생성되는 방식으로 한 개의 round key를 생

성한다. 따라서 위 과정을 10번 반복하면 AES-128에서 사용되는 모든 round key가 생성된다.

2.4 ShiftRows / Inverse ShiftRows

ShiftRows는 각 Row를 0, 1, 2, 3만큼 왼쪽으로 shift하는 연산이며 Inverse ShiftRows는 각 Row를 0, 1, 2, 3만큼 오른쪽으로 shift하여 복원하는 과정이다.

2.5 MixColumns / Inverse MixColumns

MixColumns은 Column을 섞어주는 역할을 하며 이때 사용되는 행렬은 다음과 같다.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

마찬가지로 Inverse MixColumns은 MixColumns으로 변환된 Columns을 되돌리기 위해 다음 행렬을 곱한다.

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

2.6 AddRoundKey

AddRoundKey는 말그대로 round key를 더하는 단계로 현재 상태에 round key를 XOR한다.

3. R S A [2]

이 프로그램에서 사용되는 RSA는 RSA-2048으로 2048 bits의 길이를 가지는 방식으로 대략 617자리수이다. RSA는 소수 p, q를 선택하고, $n = p \times q$ 를 연산한다. 그 후 Euler totient function인 $\phi(n) = (p-1)(q-1)$ 와 $1 < e < \phi(n)$인 e의 나머지가 1이 되게 하는 정수 e를 구한다. Extended Euclid Algorithm을 통해 $\phi(n)$과 e의 나머지가 1이 되면 자연스럽게 곱셈의 역원인 $d = e^{-1} \pmod{\phi(n)}$를 구할 수 있게 되며 이를 통해 공개키인 $PU = \{ e, n \}$, 개인키인 $PR = \{ d, n \}$이 생성된다. 이렇게 생성된 공개키와 개인키를 통해 만들어지는 plaintext M의 ciphertext는 $C = M^e \bmod n$, ciphertext를 복호화 한 plaintext는 $M = C^d \bmod n$으로 계산하게 된다. 이렇게 계산이 가능한 이유는 $ed = k\phi(n) + 1 \pmod{\phi(n)}$ 이기 때문에 Euler's Theorem에 의해 $a^{\phi(n)} \equiv 1 \pmod{n}$ 이므로 $M^{ed} \bmod n \equiv M^{k\phi(n)+1} \bmod n \equiv M \bmod n$ 이 되기 때문이다.

3.1 Key Generation

Key Generation은 RSA에 필요한 e, d, n을 구하는 함수로 RSA-2048의 규격에 맞추기 위해 p와 q의 범위를 1023 ~ 1024 bit 사이의 숫자로 랜덤하게 뽑게 되며 소수가 맞는지 판별하기 위해 Miller_rabin test를 돌리게 된다. Miller_rabin test를 통과한 p, q는 probable prime으로 소수일 것이라고 생각하고 진행하게 되며 이 프로그램에서는 Extended Euclid algorithm을 사용한다. 만약 d가 음수라면 $d = (d + \phi(n)) \bmod \phi(n)$으로 양수로 만들어 준다.

3.2 Extended Euclid Algorithm

Extended Euclid Algorithm은 e의 $\phi(n)$에 대한 곱셈의 역원을 구하기 위해 사용하며 이 함수를 실행하면 최종 결과가 $k\phi(n) + ed = 1$이 나오게 되는 e를 찾을 때까지 이 함수를 계속 실행하게 된다.

3.3 Modular Exponentiation

Modular Exponentiation은 mod n에 대한 거듭제곱 연산을 하는 함수인데 매우 큰 수에 대한 연산을 사용하므로 거듭제곱을 하면 할수록 메모리를 크게 잡아먹기 때문에 n으로 매번 modular 연산을 하면서 계산하게 된다.

3.4 Miller_rabin [3]

Miller_rabin 함수는 소수를 판별하는 알고리즘 중 하나로 임의의 홀수 n을 소수인지 판별하는 알고리즘이다. 임의의 홀수 n에 대해 $n - 1 = 2^t u$라고 할 수 있으며, n보다 작은 임의의 정수 a에 대해 $x_0 = a^u \bmod n$라

고 하고 $x_i = {x_{i-1}}^2 \ mod \ n$이이 할 때, i가 1부터 t까지 중에 $x_i = 1$ 이고 $x_{i-1} \neq 1$ 이고 $x_{i-1} \neq n - 1$이면 소수가 아니라고 판별하는 NSR test, $x_t \neq 1$이면 소수가 아니라고 판별하는 Fermat test를 하게 된다. 이 두 테스트를 실행할 때 마다 소수가 아닐 확률이 매우 크게 감소하여 s번 실행하게 되면 소수가 아닐 확률이 2^{-s}보다도 훨씬 작게 되므로 s가 크면 클수록 거의 소수라고 판별할 수 있다.

참 고 문 헌

[1] Intel® Advanced Encryption Standard (AES) New Instructions Set, 2010

[2] PKCS #1: RSA Cryptography Specifications Version 2.2, IETF, 2016

[3] Cormen, Leiserson, Rivest, and Stein, Introduction to Algorithms, 3rd ed. MIT Press, 2009