

부산대학교 정보컴퓨터공학부 2024 전기 졸업과제

유니티 멀티플레이어 환경에서의 경량 암호 통신 구현

분과 D 47 [민트초코펍시] 팀

지도교수

손준영

조원1

고세화

조원2

김재민

조원3

어강인



연구 배경 및 목적



연구 배경 및 목적



출처: <https://www.youtube.com/watch?v=DAQlyrh8cV8>



연구 배경 및 목적



출처: <https://www.youtube.com/watch?v=TbxY8U000bw>



연구 배경 및 목적

메이플스토리 1320만명 개인정보 유출

김민권 | 승인 2011.11.26 01:40

이름, 계정, 암호화된 비밀번호, 암호화된 주민등록번호 유출
개인정보보호법 시행 이후 첫 유출 사건으로 기록

우려했던 일이 또 발생했다. 넥슨은 자사에서 운영하는 메이플스토리 이용자 정보가 해킹에 의해 지난 11월 18일 유출된 것을 11월 24일 확인했다고 밝혔다. 이번 해킹으로 메이플스토리 전체 이용자 1800만명 중 1320만명의 개인정보가 유출된 것으로 드러났다.



넥슨은 메이플스토리 공지사항을 통해 “이번 일로 고객 여러분에게 발생할 수 있는 피해를 예방하고 신속히 범인을 검거하기 위해 25일 관련법령에 따라 수사기관과 관계기관에 즉시 조사를 의뢰했다”며 “이번에 유출된 개인정보는 이름과 계정, 암호화된 비밀번호, 암호화된 주민등록번호 등이 일부 유출된 것으로 파악하고 있다. 만일의 경우에 대비해 즉시 메이플스토리 이용자 계정의 비밀번호를 변경해 주길 당부한다”고 공지했다.

또 “경찰청 수사와 관계기관의 조사 결과에 따라 추가로 확인되는 내용에 대해 즉시 알리도록 하겠다”며 “이번 사건은 메이플스토리에만 해당되는 것으로 넥슨이 서비스하는 다른 게임들과는 전혀 무관하다”고 강조했다.

방통위는 25일 “넥슨의 메이플스토리 회원 1800만명 중 1320만명의 개인정보가 지난

로그인 성공내역

고객님의 최근 90일 동안의 로그인 성공한 내역을 확인할 수 있습니다.

| 일시 | 서비스 | IP | 접속지역 | 운영체제 | 브라우저 |
|------------------------|-------------|-----------------|------|------------|--------|
| 2023.10.17 오전 12:05:17 | 내정보 | 121.172.55.25 | 대한민국 | Windows 10 | Chrome |
| 2023.10.17 오전 12:03:35 | 던전앤파이터 | 112.149.240.237 | 대한민국 | Windows 10 | Chrome |
| 2023.10.16 오전 08:07:37 | 던전앤파이터 | 59.26.106.234 | 대한민국 | Windows 10 | Chrome |
| 2023.10.13 오전 05:13:43 | 서든라디오 (모바일) | 121.168.117.252 | 대한민국 | Android | - |
| 2023.10.12 오후 11:37:36 | 기타 | 124.198.98.152 | 대한민국 | - | - |
| 2023.10.09 오전 11:41:50 | 서든라디오 (모바일) | 222.118.181.242 | 대한민국 | Android | - |
| 2023.10.09 오전 05:28:54 | 서든라디오 (모바일) | 27.113.74.43 | 대한민국 | Android | - |
| 2023.10.09 오전 05:27:48 | 서든라디오 (모바일) | 124.111.117.31 | 대한민국 | Android | - |
| 2023.10.09 오전 05:27:14 | 서든라디오 (모바일) | 59.14.45.119 | 대한민국 | Android | - |
| 2023.10.01 오후 04:41:12 | 넥슨 홈페이지 | 210.123.103.118 | 대한민국 | Windows 10 | Chrome |

출처: <https://www.dailysecu.com/news/articleView.html?idxno=1154>

출처: <https://www.inven.co.kr/board/maple/5974/2613666>



연구 배경 및 목적

MMORPG 게임의 네트워크 보안



- ◆ 안전한 통신 환경 구축
- ◆ 블록 암호 알고리즘의 운영모드별 성능 비교



실제 MMORPG 게임에서 적용 가능한 보안 솔루션!



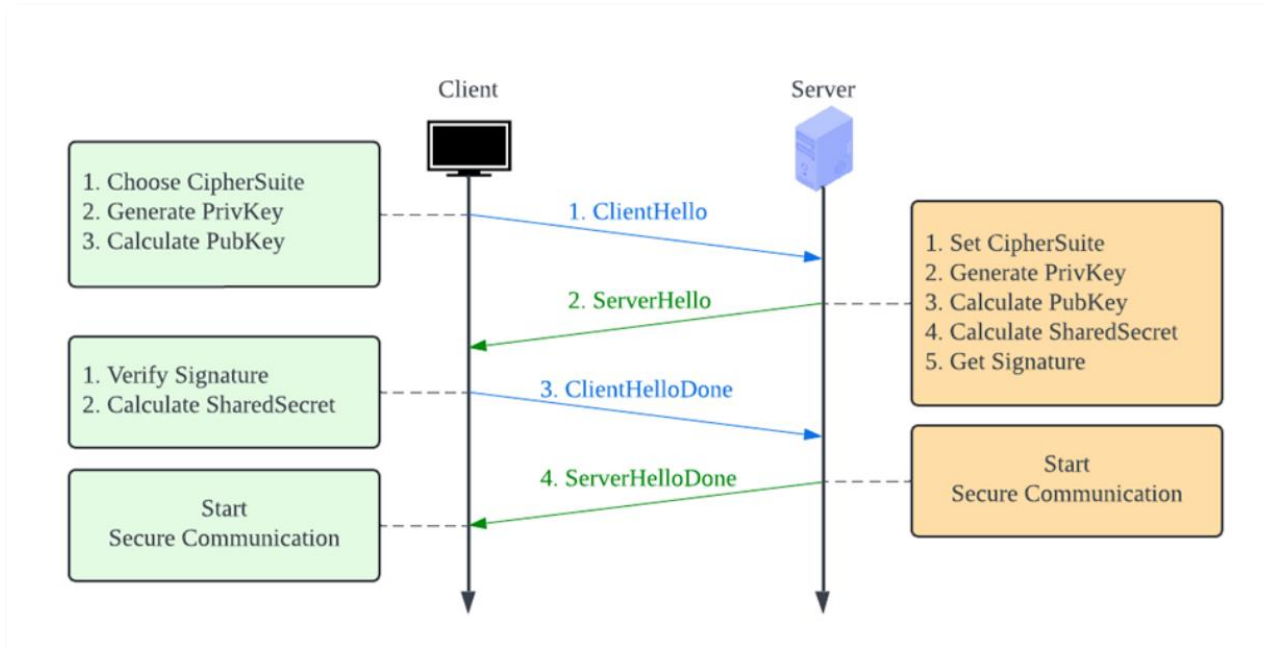
구현 - 암호 알고리즘

| 암호 알고리즘 | 종류 |
|------------|--------------------------------|
| 전자 서명 알고리즘 | ECDSA (P-256, SHA-256) |
| 키 교환 알고리즘 | ECDH (P-256, SHA-256) |
| 키 유도 함수 | HKDF (SHA-256) |
| 블록 암호 알고리즘 | AES, ARIA, HIGHT, SPECK, TWINE |
| 블록 암호 운영모드 | ECB, CBC, CFB, OFB, CTR, GCM |

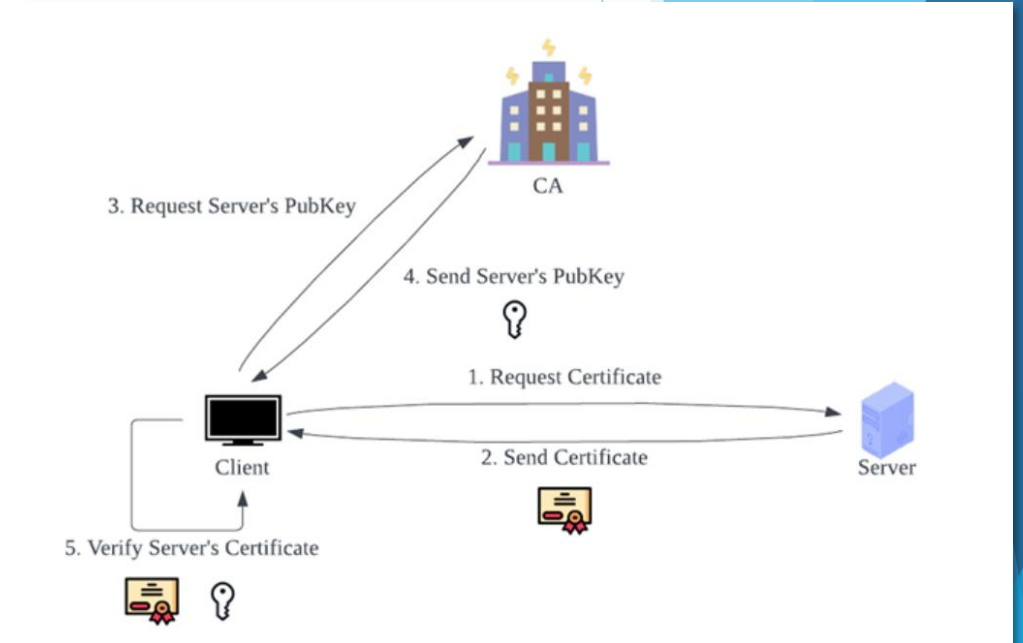
통신에 사용할 암호 알고리즘 구현



구현 - Handshake Protocol



Handshake Protocol



서버 인증 과정

CipherSuite 선택, 서버 인증, 키 교환 및 유도



구현 - MMORPG

```
C:\Users\Administrator\Desktop\FastEncryption\Server\Server\bin\Debug\net8.0\Server.exe

[DEBUG][2024-10-18 05:39:54] Recv C_Hello_Done
[DEBUG][2024-10-18 05:39:54] Set Operation Mode: NetworkCore.Encryption.BlockCipher.OperationMode.ECB
[DEBUG][2024-10-18 05:39:54] Derived Session Key: 40 6E FF 34 D5 3B B0 23 B7 42 7B 84 49 A6 E8 30
[DEBUG][2024-10-18 05:39:54] Secure Communication Established
[DEBUG][2024-10-18 05:39:54] C_HelloDone, SessionKey: 40 6E FF 34 D5 3B B0 23 B7 42 7B 84 49 A6 E8 30
[DEBUG][2024-10-18 05:39:54] Sent S_Hello_Done Packet: Success=true
[INFO][2024-10-18 05:39:59] Client Connected. SessionId: 3
[INFO][2024-10-18 05:39:59] [Client 3]: Connected.

[DEBUG][2024-10-18 05:39:59] Recv C_Hello
[DEBUG][2024-10-18 05:39:59] Cipher Suite Validation: True, Cipher Suite: AesCbc
[DEBUG][2024-10-18 05:39:59] Parsed Client Public Key: X=605983694682563556809457534908676195346952213300305823277652862
38724459577390, Y=98358503390741344300748048854536959938596287551784073742579220787253721367502
[DEBUG][2024-10-18 05:39:59] Client Public Key Validation: True
[DEBUG][2024-10-18 05:39:59] Generated Server Public Key: X=880155936313109244667053839781293111281200430568576281359910
24614701229609385, Y=82621904966513878976479945719869834410893473973857707101783778158364996721824
[DEBUG][2024-10-18 05:39:59] Calculated Shared Secret: A2 3D 4F 2A 44 99 03 9A 02 57 E2 37 8C 27 5A CE 55 AC E4 77 CE 11
86 0E EF 8E 1B 4C D0 38 FC 2E
[DEBUG][2024-10-18 05:39:59] Server Signature: E6 7A 97 17 CC F9 68 41 48 9D 65 41 F4 F6 AD B1 2D 17 B5 9A 6B EF 84 7B 6
1 83 B8 FC F1 6A 32 EB 9A E6 BA 6D 63 77 06 84 9A 6A 9F C3 88 CF 02 32 D8 5C 26 EA 0D 1F E7 43 7A DB 48 DE 58 36 43 33
[DEBUG][2024-10-18 05:39:59] Generated Salt: 27 36 0F 04 F3 F6 7B AA B1 48 72 FD C2 A2 0E BE
[DEBUG][2024-10-18 05:39:59] Sent S_Hello Packet: Success=True

[DEBUG][2024-10-18 05:40:00] Recv C_Hello_Done
[DEBUG][2024-10-18 05:40:00] Set Operation Mode: NetworkCore.Encryption.BlockCipher.OperationMode.CBC
[DEBUG][2024-10-18 05:40:00] Derived Session Key: 45 86 57 D6 64 A7 DE C5 04 75 DE 03 69 8C 9B D5
[DEBUG][2024-10-18 05:40:00] Secure Communication Established
[DEBUG][2024-10-18 05:40:00] C_HelloDone, SessionKey: 45 86 57 D6 64 A7 DE C5 04 75 DE 03 69 8C 9B D5
[DEBUG][2024-10-18 05:40:00] Sent S_Hello_Done Packet: Success=true
```

서버

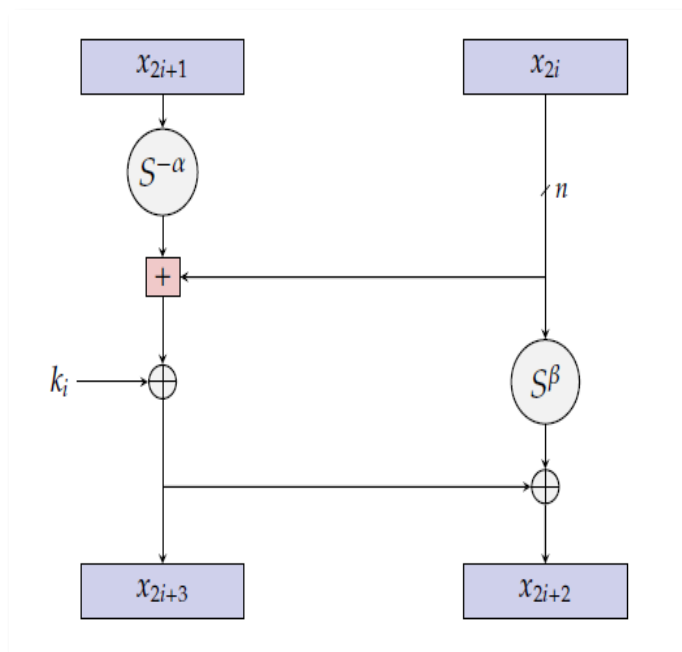


클라이언트

서버: IOCP, 클라이언트: Unity, 실제 MMORPG 환경 구현



개발 과정 - 암호 알고리즘



SPECK 암호화 (그림)

```
public override byte[] Encrypt(byte[] plainText)
{
    if (plainText.Length != 16)
        throw new ArgumentException("plainText must be exactly 16 bytes long.");

    ulong[] Pt = new ulong[2];
    ulong[] Ct = new ulong[2];

    Pt[0] = BitConverter.ToUInt64(plainText, 0);
    Pt[1] = BitConverter.ToUInt64(plainText, 8);

    Ct[0] = Pt[0]; Ct[1] = Pt[1];
    for (int i = 0; i < 32; i++)
    {
        Ct[1] = ROTR64(Ct[1], 8);
        Ct[1] += Ct[0];
        Ct[1] ^= rk[i];
        Ct[0] = ROTL64(Ct[0], 3);
        Ct[0] ^= Ct[1];
    }

    byte[] cipherText = new byte[16];
    System.Buffer.BlockCopy(BitConverter.GetBytes(Ct[0]), 0, cipherText, 0, 8);
    System.Buffer.BlockCopy(BitConverter.GetBytes(Ct[1]), 0, cipherText, 8, 8);

    return cipherText;
}
```

SPECK 암호화 (코드)

암호 알고리즘은 직접 구현



개발 과정 - 암호 알고리즘

```
Msg = c35e2f092553c55772926bde87c9796827d17024dbb9233a545366e2e5987dd344deb72df987144b8c6cd
d = 0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813
Qx = e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aef fa8a
Qy = bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39
k = 6d3e71882c3b83b156bb14e0ab184aa9fb728068d3ae9fac421187ae0b2f34c6
R = 976d3a4e9d23326dc0baa9fa560b7c4e53f42864f508483a6473b6a11079b2db
S = 1b766e9ceb71ba6c01dcd46e0af462cd4cfa652ae5017d4555b8eeef36e1932

Msg = 3c054e333a94259c36af09ab5b4ff9beb3492f8d5b4282d16801daccb29f70fe61a0b37fef5c04cd1b70
d = e283871239837e13b95f789e6e1af63bf61c918c992e62bca040d64cad1fc2ef
Qx = 74ccd8a62fba0e667c50929a53f78c21b8ff0c3c737b0b40b1750b2302b0bde8
Qy = 29074e21f3a0ef88b9efdf10d06aa4c295cc1671f758ca0e4cd108803d0f2614
k = ad5e887eb2b380b8d8280ad6e5ff8a60f4d26243e0124c2f31a297b5d0835de2
R = 35fb60f5ca0f3ca08542fb3cc641c8263a2cab7a90ee6a5e1583fac2bb6f6bd1
S = ee59d81bc9db1055cc0ed97b159d8784af04e98511d0a9a407b99bb292572e96
```

ECDSA 테스트 벡터

```
foreach (ECDSATestVector vector in vectors)
{
    byte[] rBytes = vector.R.ToByteArray(isUnsigned: true, isBigEndian: true);
    byte[] sBytes = vector.S.ToByteArray(isUnsigned: true, isBigEndian: true);
    byte[] TestSignature = new byte[64];
    Array.Copy(rBytes, 0, TestSignature, 0, rBytes.Length);
    Array.Copy(sBytes, 0, TestSignature, rBytes.Length, rBytes.Length);

    byte[] signature = ecdsa.Sign(vector.Msg, vector.Key, vector.K);
    if (!signature.SequenceEqual(TestSignature))
    {
        Console.WriteLine($"vector.Msg: ");
        printHex(vector.Msg);
        Console.WriteLine();

        Console.WriteLine($"vector.Key: {vector.Key:X}Wn");
        Console.WriteLine($"vector.Qx: {vector.Qx:X}Wn");
        Console.WriteLine($"vector.Qy: {vector.Qy:X}Wn");
        Console.WriteLine($"vector.K: {vector.K:X}Wn");
        Console.WriteLine($"vector.R: {vector.R:X}Wn");
        Console.WriteLine($"vector.S: {vector.S:X}Wn");

        Console.WriteLine("Calculated Signature: ");
        printHex(signature);
        Console.WriteLine();

        return false;
    }
}

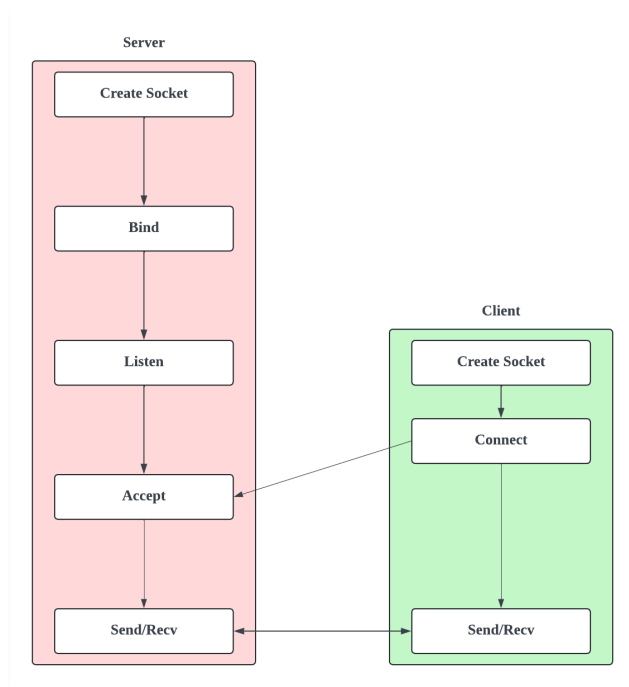
return true;
```

ECDSA 테스트 코드

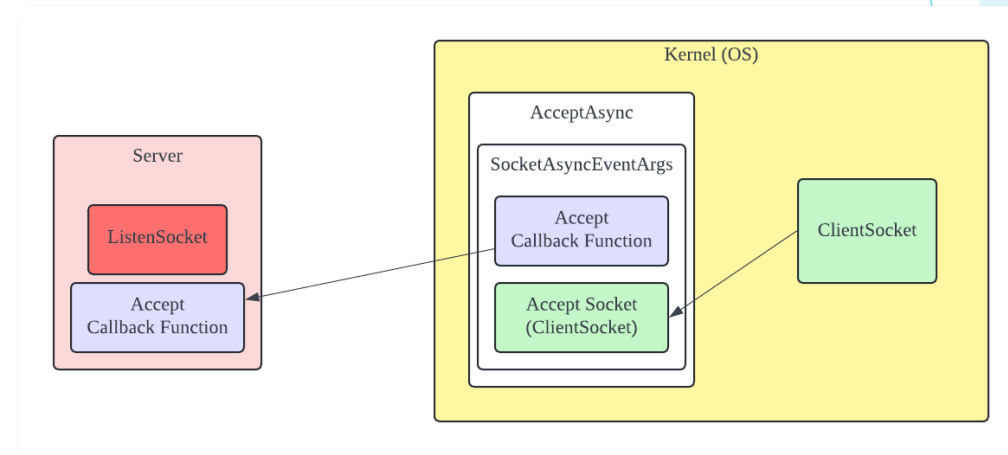
구현한 암호 알고리즘은 테스트 벡터를 통해 검증



개발 과정 - 게임 서버



TCP 연결 과정

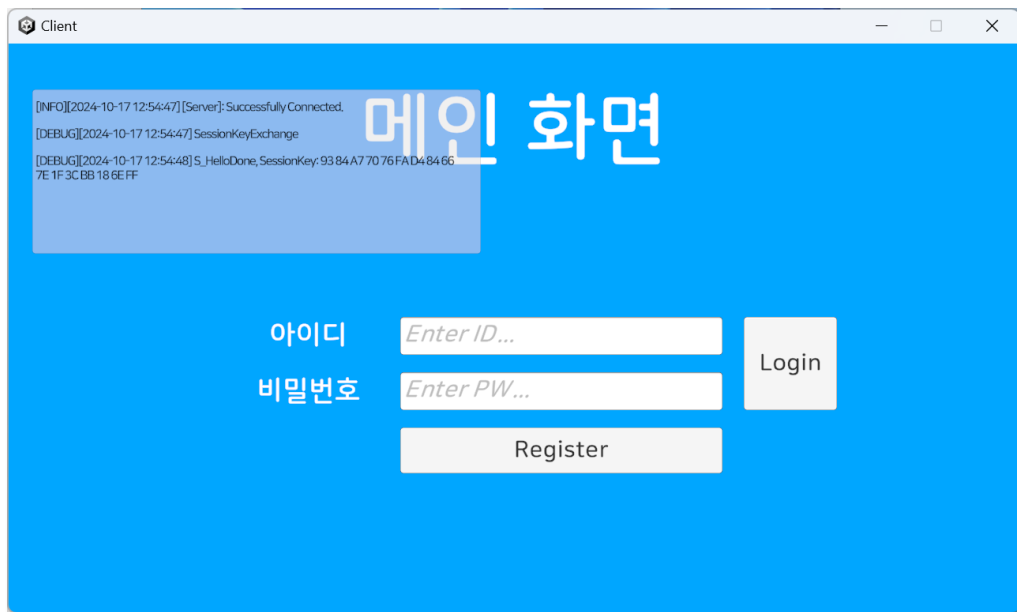


비동기 I/O 처리 과정

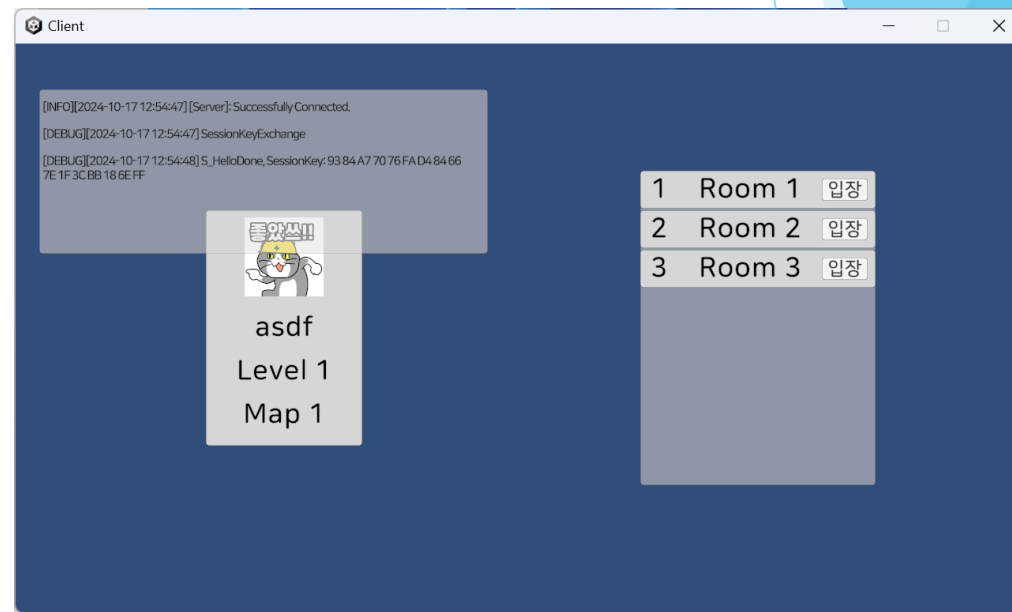
IOCP 모델 채택, 비동기 I/O 사용



개발 과정 - 게임 클라이언트



로그인 화면

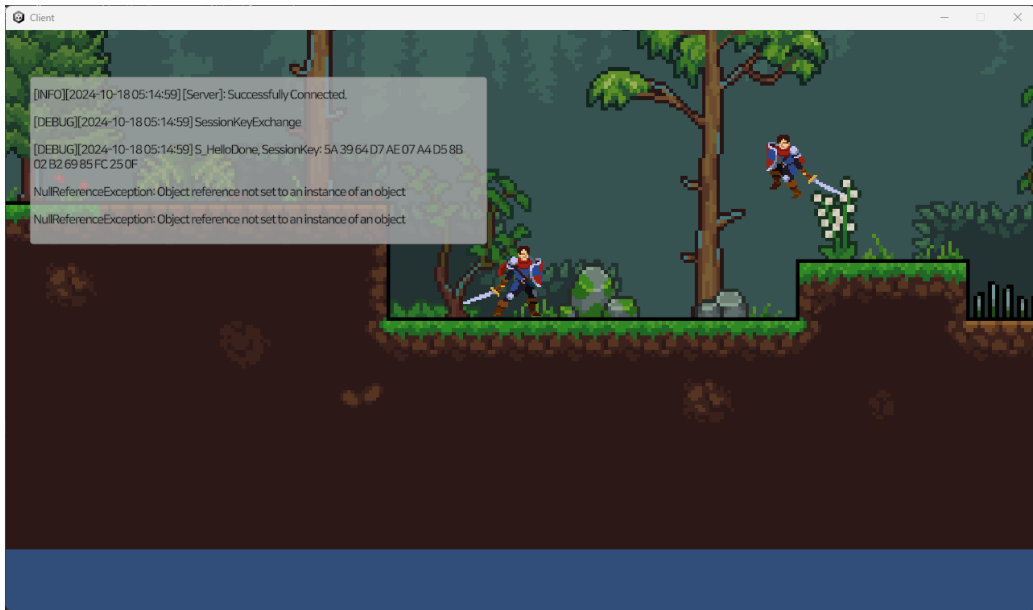


방 입장 화면

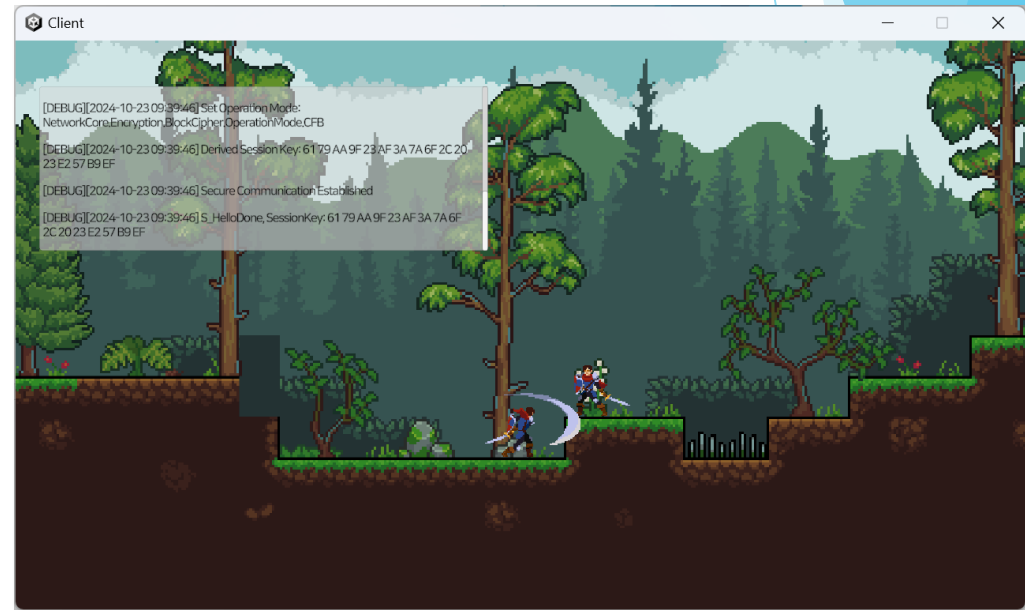
게임 클라이언트는 유니티 사용



개발 과정 - 게임 클라이언트



이동 동기화 화면

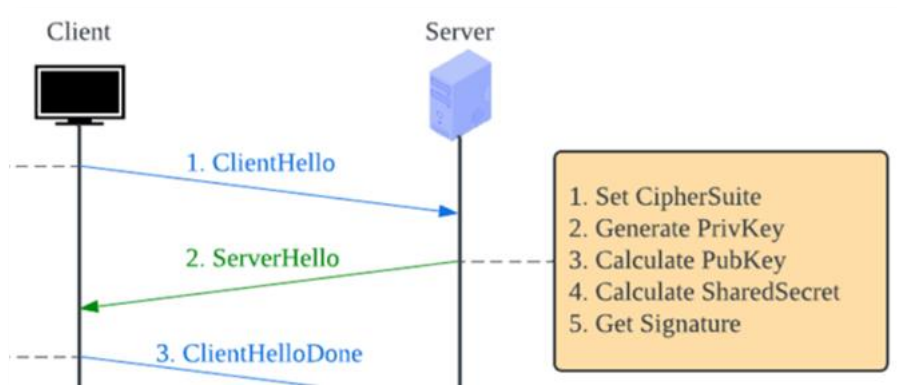


공격 동기화 화면

게임 클라이언트는 유니티 사용



개발 과정 - Handshake Protocol



ClientHello 처리

```
public static void C_HelloHandler(PacketSession session, IMessage packet)
{
    Console.WriteLine("");
    Logger.DebugLog("Recv C_Hello");

    C_Hello helloPacket = packet as C_Hello;

    bool isSuccess = true;

    // Cipher Suite Validation Check
    isSuccess &= IsValidCipherSuite(helloPacket.CipherSuite);
    Logger.DebugLog($"Cipher Suite Validation: {isSuccess}, Cipher Suite: {helloPacket.CipherSuite}");

    // Packet Parsing
    CipherSuite cipherSuite = helloPacket.CipherSuite;
    byte[] clientPubKeyXArray = helloPacket.PubKeyX.ToByteArray();
    byte[] clientPubKeyYArray = helloPacket.PubKeyY.ToByteArray();
    BigInteger clientPubKeyX = new BigInteger(clientPubKeyXArray, isUnsigned: true, isBigEndian: true);
    BigInteger clientPubKeyY = new BigInteger(clientPubKeyYArray, isUnsigned: true, isBigEndian: true);
    ECPoint clientPubKey = new ECPoint(clientPubKeyX, clientPubKeyY);
    Logger.DebugLog($"Parsed Client Public Key: X={clientPubKeyX}, Y={clientPubKeyY}");

    // Client Public Key Validation Check
    isSuccess &= (session.ECDSA.IsValidPoint(clientPubKey) == 0);
    Logger.DebugLog($"Client Public Key Validation: {isSuccess}");

    // Create PrivKey
    BigInteger privKey = session.ECDH.GeneratePrivKey();

    // Calculate PubKey
    ECPoint pubKey = session.ECDH.GetPublicKey(privKey);
    byte[] pubKeyX = pubKey.X.ToByteArray(isUnsigned: true, isBigEndian: true);
    byte[] pubKeyY = pubKey.Y.ToByteArray(isUnsigned: true, isBigEndian: true);
    Logger.DebugLog($"Generated Server Public Key: X={pubKeyX}, Y={pubKeyY}");
}
```

ClientHello 처리 코드

Handshake Protocol의 단계별 작업을 코드로 구현



개발 결과 - 안전한 게임 통신 환경

```
C:\Users\Wasdf\W\Desktop\Wdis >
[INFO][2024-10-23 10:45:38] Listening...
[INFO][2024-10-23 10:45:47] Client Connected. SessionId: 1
[INFO][2024-10-23 10:45:47] [Client 1]: Connected.

[DEBUG][2024-10-23 10:45:47] Recv C_Hello
[DEBUG][2024-10-23 10:45:47] Cipher Suite Validation: True, Cipher Suite: AriaCtr
[DEBUG][2024-10-23 10:45:47] Parsed Client Public Key: X=458265556398592308010784676275327254326293874056711458241089462
75190535860957, Y=21369670242214698048994619544260096864079851279499864941143171678694934719644
[DEBUG][2024-10-23 10:45:47] Client Public Key Validation: True
[DEBUG][2024-10-23 10:45:47] Generated Server Public Key: X=275871211100165776393541973289913423589525033192731668540716
43994233360400251, Y=21007320869549590786997560217725238674254143466442712748867530575861694063793
[DEBUG][2024-10-23 10:45:47] Calculated Shared Secret: 32 01 DB B8 3A BF 87 D7 88 9A 8E 53 8C 45 EF 9F 6F 94 50 9C AE B1
4A A6 4A E6 16 BE 99 F1 E6 B3
[DEBUG][2024-10-23 10:45:47] Server Signature: E6 7A 97 17 CC F9 68 41 48 9D 65 41 F4 F6 AD B1 2D 17 B5 9A 6B EF 84 7B 6
1 83 B8 FC F1 6A 32 EB 9A E6 BA 6D 63 77 06 84 9A 6A 9F C3 88 CF 02 32 D8 5C 26 EA 0D 1F E7 43 7A DB 48 DE 58 36 43 33
[DEBUG][2024-10-23 10:45:47] Generated Salt: DE 96 A5 F7 CE C0 81 58 75 02 92 D8 04 24 D6 5E
[DEBUG][2024-10-23 10:45:47] Sent S_Hello Packet: Success=True

[DEBUG][2024-10-23 10:45:47] Recv C_Hello_Done
[DEBUG][2024-10-23 10:45:47] Set Operation Mode: NetworkCore.Encryption.BlockCipher.OperationMode.CTR
[DEBUG][2024-10-23 10:45:47] Derived Session Key: F0 C1 EC 62 F4 65 27 B0 24 1F 0B 70 35 79 D2 C0
[DEBUG][2024-10-23 10:45:47] Secure Communication Established
[DEBUG][2024-10-23 10:45:47] C_HelloDone, SessionKey: F0 C1 EC 62 F4 65 27 B0 24 1F 0B 70 35 79 D2 C0
[DEBUG][2024-10-23 10:45:47] Sent S_Hello_Done Packet: Success=true
```

서버의 세션 키 (대칭 키)

```
[DEBUG][2024-10-23 10:45:47] Set Operation Mode:
NetworkCore.Encryption.BlockCipher.OperationMode.CTR

[DEBUG][2024-10-23 10:45:47] Derived Session Key: F0 C1 EC 62 F4 65 27 B0 24 1F 0B
70 35 79 D2 C0

[DEBUG][2024-10-23 10:45:47] Secure Communication Established

[DEBUG][2024-10-23 10:45:47] S_HelloDone, SessionKey: F0 C1 EC 62 F4 65 27 B0 24 1F
0B 70 35 79 D2 C0
```

클라이언트의 세션 키 (대칭 키)

서버와 클라이언트의 세션 키가 같음 → 대칭 키 유도 성공!



개발 결과 - 안전한 게임 통신 환경

```
C:\Users\Wasdf\W\Desktop\Wdis x + v
[INFO][2024-10-23 10:45:47] Client Connected. SessionId: 1
[INFO][2024-10-23 10:45:47] [Client 1]: Connected.

[DEBUG][2024-10-23 10:45:47] Recv C_Hello
[DEBUG][2024-10-23 10:45:47] Cipher Suite Validation: True, Cipher Suite: AriaCtr
[DEBUG][2024-10-23 10:45:47] Parsed Client Public Key: X=458265556398592308010784676275327254326293874056711458241089462
75190535860957, Y=21369670242214698048994619544260096864079851279499864941143171678694934719644
[DEBUG][2024-10-23 10:45:47] Client Public Key Validation: True
[DEBUG][2024-10-23 10:45:47] Generated Server Public Key: X=275871211100165776393541973289913423589525033192731668540716
43994233360400251, Y=21007320869549590786997560217725238674254143466442712748867530575861694063793
[DEBUG][2024-10-23 10:45:47] Calculated Shared Secret: 32 01 DB B8 3A BF 87 D7 88 9A 8E 53 8C 45 EF 9F 6F 94 50 9C AE B1
4A A6 4A E6 16 BE 99 F1 E6 B3
[DEBUG][2024-10-23 10:45:47] Server Signature: E6 7A 97 17 CC F9 68 41 48 9D 65 41 F4 F6 AD B1 2D 17 B5 9A 6B EF 84 7B 6
1 83 B8 FC F1 6A 32 EB 9A E6 BA 6D 63 77 06 84 9A 6A 9F C3 88 CF 02 32 D8 5C 26 EA 0D 1F E7 43 7A DB 48 DE 58 36 43 33
[DEBUG][2024-10-23 10:45:47] Generated Salt: DE 96 A5 F7 CE C0 81 58 75 02 92 D8 04 24 D6 5E
[DEBUG][2024-10-23 10:45:47] Sent S_Hello Packet: Success=True

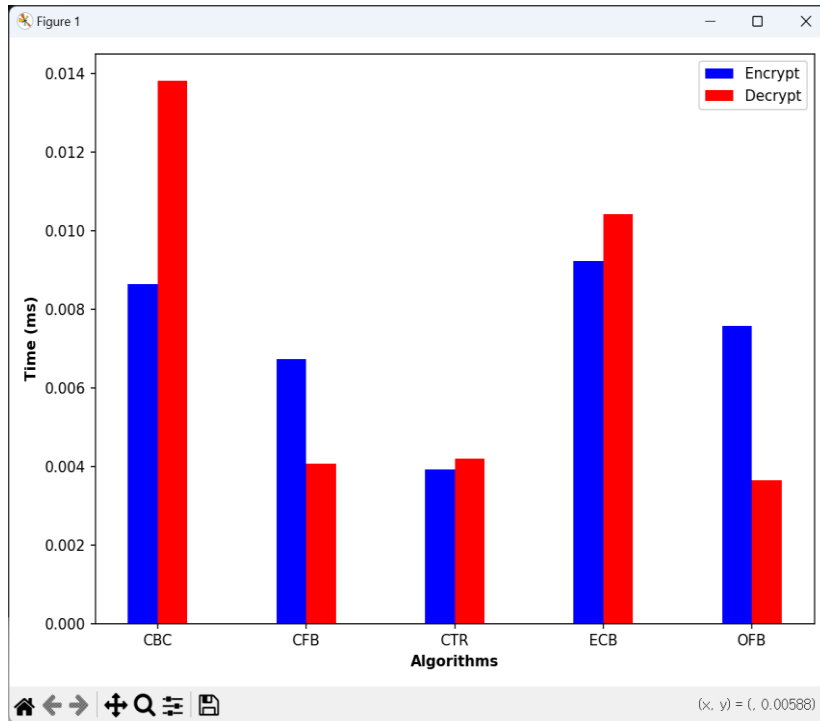
[DEBUG][2024-10-23 10:45:47] Recv C_Hello_Done
[DEBUG][2024-10-23 10:45:47] Set Operation Mode: NetworkCore.Encryption.BlockCipher.OperationMode.CTR
[DEBUG][2024-10-23 10:45:47] Derived Session Key: F0 C1 EC 62 F4 65 27 B0 24 1F 0B 70 35 79 D2 C0
[DEBUG][2024-10-23 10:45:47] Secure Communication Established
[DEBUG][2024-10-23 10:45:47] C_HelloDone, SessionKey: F0 C1 EC 62 F4 65 27 B0 24 1F 0B 70 35 79 D2 C0
[DEBUG][2024-10-23 10:45:47] Sent S_Hello_Done Packet: Success=true
[DEBUG][2024-10-23 10:48:20] Encrypted Packet : 1F 47 CA 9A 00 73 A5 AA 70 9F 02 CE 1E CF 29 23 58 A0 33 0D C9 A9
[DEBUG][2024-10-23 10:48:20] Decrypted Packet : 0A 01 61 12 01 61
[DEBUG][2024-10-23 10:48:21] Encrypted Packet : 5F B2 10 C3 B6 8F B4 33 5F 81 3A 75 81 6B 53 C8 E4 A4 1B 1D 09 0D
[DEBUG][2024-10-23 10:48:21] Decrypted Packet : 0A 01 61 12 01 61
[DEBUG][2024-10-23 10:48:22] Encrypted Packet : 27 C9 EF AA 78 89 86 3D 26 0F D9 FF 3D C2 A6 9C 80 B1
[DEBUG][2024-10-23 10:48:22] Decrypted Packet : 08 01
```

서버의 패킷 복호화

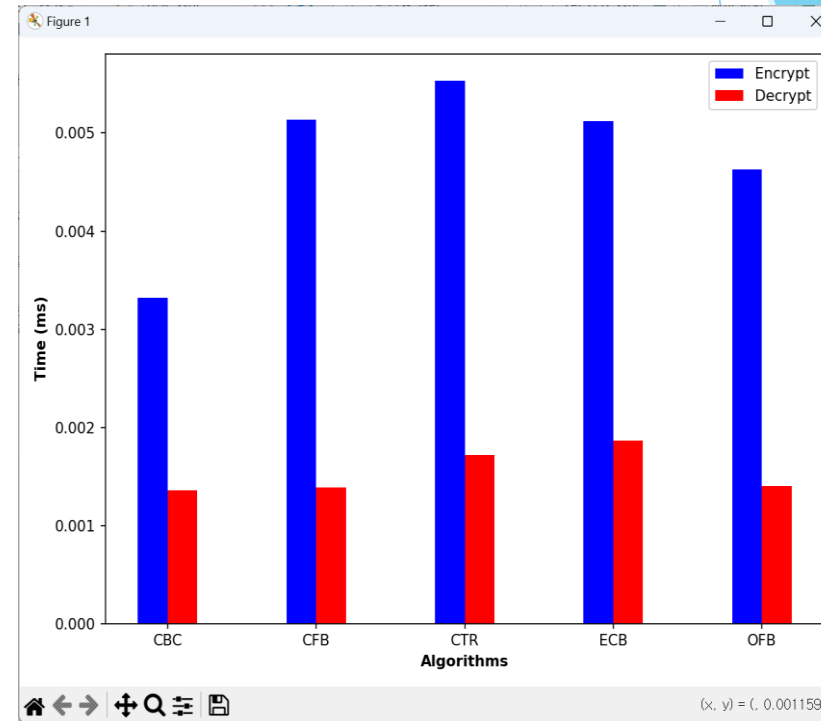
세션 키를 통한 패킷 복호화가 성공적으로 이루어짐



개발 결과 - 블록 암호 성능 비교



AES 성능

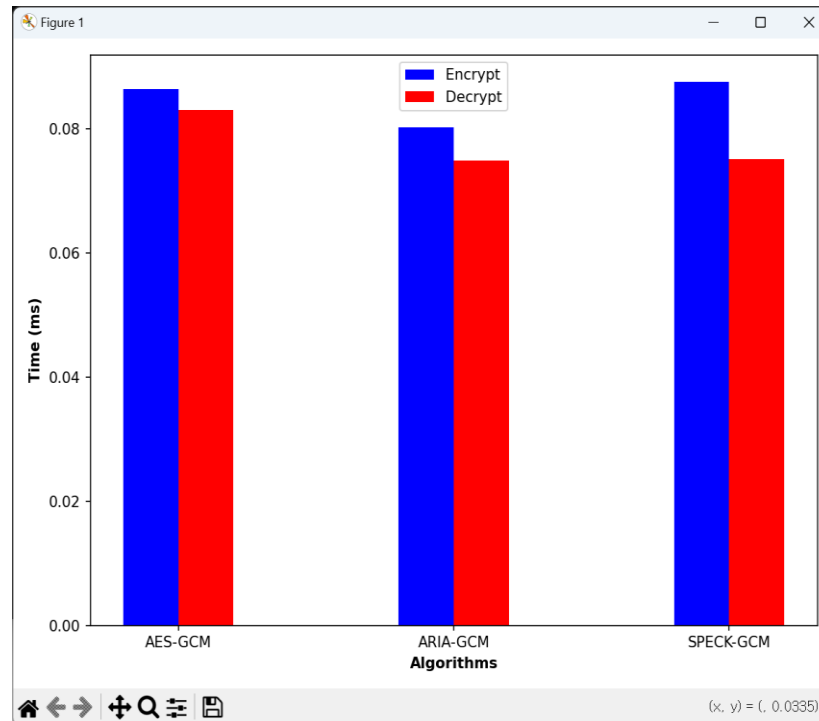


SPECK 성능

SPECK의 성능이 상당히 뛰어남



개발 결과 - 블록 암호 성능 비교



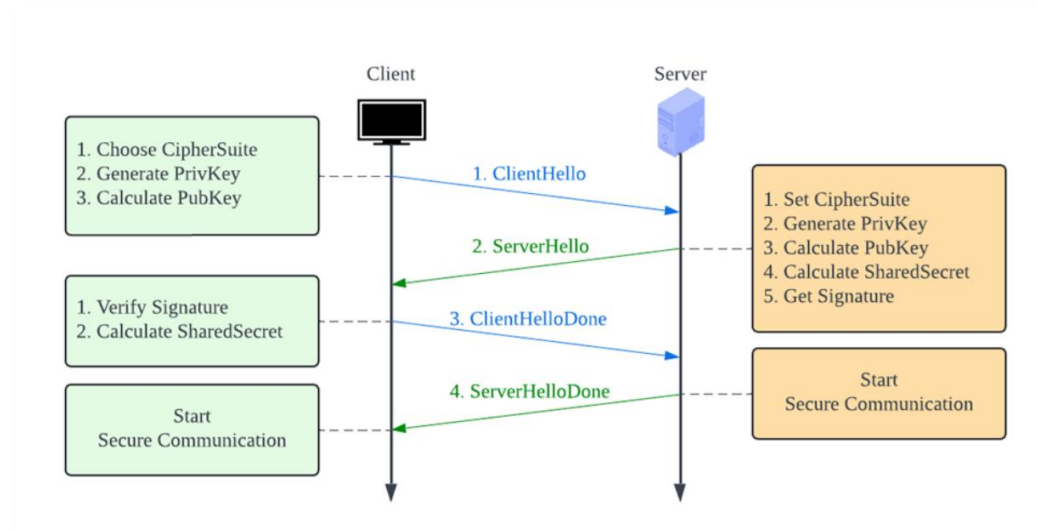
블록 암호별 GCM 성능

상대적으로 느린 GCM, 느려도 0.1ms 이하



안전한 게임 통신 환경

Handshake Protocol



결론

어떤 블록 암호 알고리즘 사용?

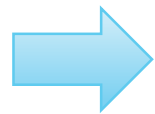
AES-GCM 암호화

0.08 ms



일반적인 레이턴시

10 ms



어떤 암호 알고리즘을 적용하더라도 OK!



연구의 한계 및 향후 연구 방향

❖ 테스트 환경이 한정적



다양한 네트워크 환경에서 좀 더 복잡한 조건으로 성능을 평가할 필요가 있음.

❖ 성능 측정 지표로 시간만 사용함



보안성 평가가 추가적으로 포함될 필요가 있음.



감사합니다

