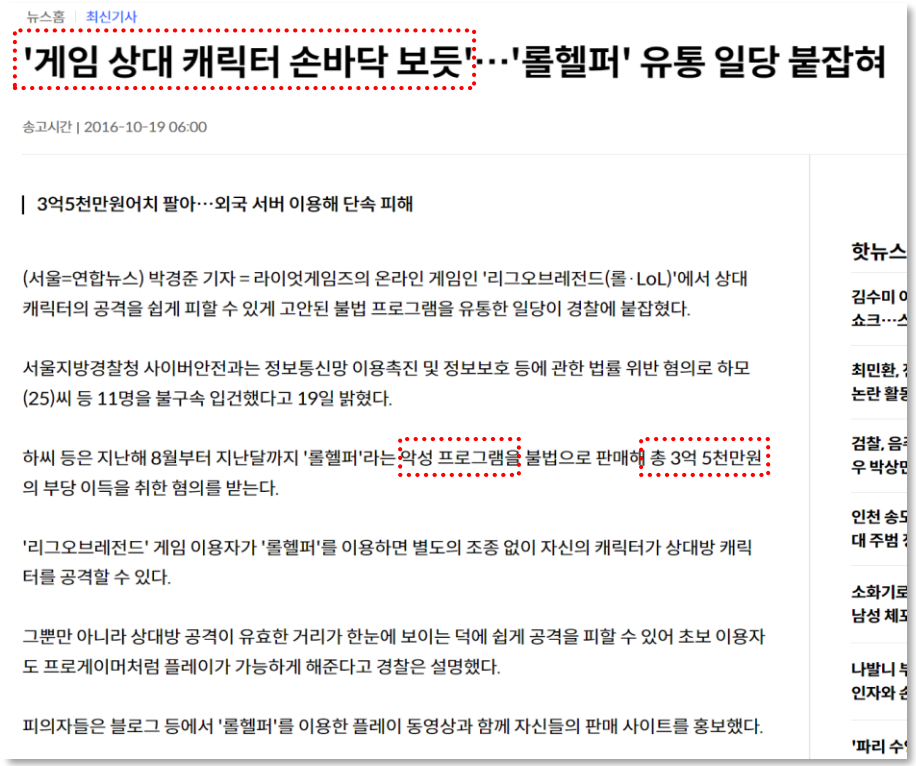
부산대학교 정보컴퓨터공학부 2024 전기 졸업과제

유니티 멀티플레이어 환경에서의 경량 암호 통신 구현

분과 D 47 [민트초코펩시] 팀
지도교수 손준영
조원1 고세화
조원2 김재민
조원3 이강인

연구 배경 및 목적



뉴스홈 | 최신기사

'게임 상대 캐릭터 손바닥 보듯'…'롤헬퍼' 유통 일당 붙잡혀

송고시간 | 2016-10-19 06:00

3억5천만원어치 팔아…외국 서버 이용해 단속 피해

(서울=연합뉴스) 박경준 기자 = 라이엇게임즈의 온라인 게임인 '리그오브레전드(롤·LoL)'에서 상대 캐릭터의 공격을 쉽게 피할 수 있게 고안된 불법 프로그램을 유통한 일당이 경찰에 붙잡혔다.

서울지방경찰청 사이버안전과는 정보통신망 이용촉진 및 정보보호 등에 관한 법률 위반 혐의로 하모(25)씨 등 11명을 불구속 입건했다고 19일 밝혔다.

하씨 등은 지난해 8월부터 지난달까지 '롤헬퍼'라는 악성 프로그램을 불법으로 판매해 총 3억 5천만원의 부당 이득을 취한 혐의를 받는다.

'리그오브레전드' 게임 이용자가 '롤헬퍼'를 이용하면 별도의 조종 없이 자신의 캐릭터가 상대방 캐릭터를 공격할 수 있다.

그뿐만 아니라 상대방 공격이 유효한 거리가 한눈에 보이는 덕에 쉽게 공격을 피할 수 있어 초보 이용자도 프로게이머처럼 플레이가 가능하게 해준다고 경찰은 설명했다.

피의자들은 블로그 등에서 '롤헬퍼'를 이용한 플레이 동영상과 함께 자신들의 판매 사이트를 홍보했다.

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

→ 게임 서버, 클라이언트에 대한 보안 공격 사례 증대

→ 적용 가능한 보안 기술 요구

연구 배경 및 목적

연구 배경 및 목적

출처: https://www.youtube.com/watch?v=TbxY8U0O0bw

연구 배경 및 목적

MMORPG 게임의 보안 기술 개요(네트워크 보안 + 암호화 기술)

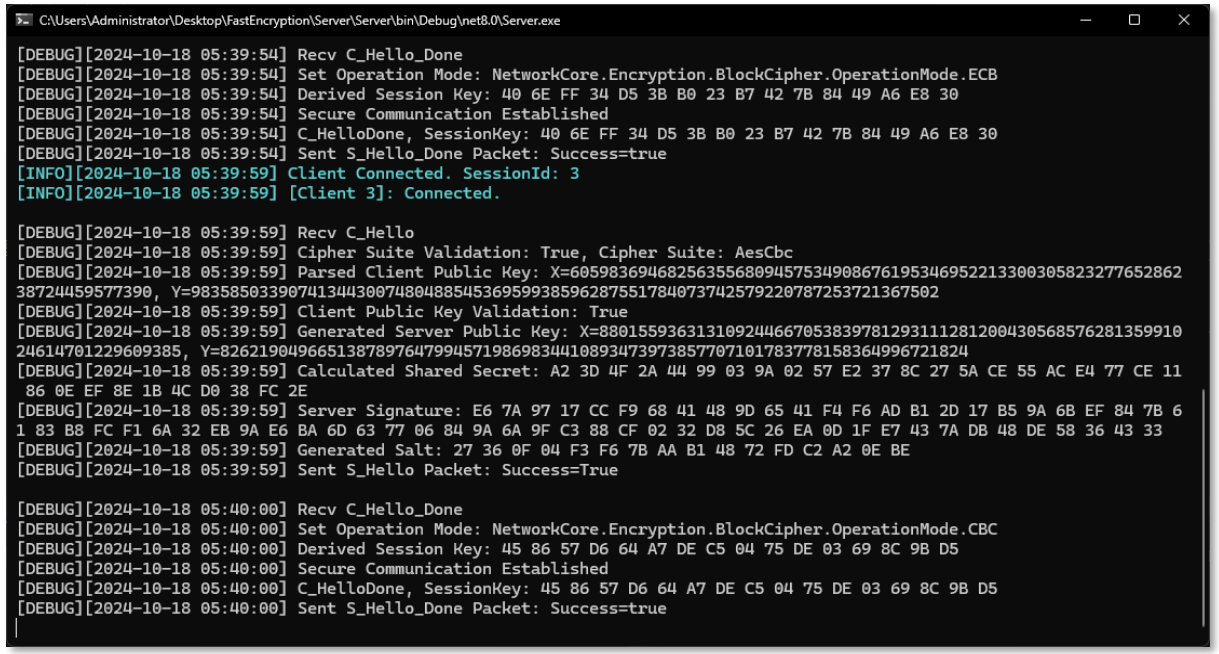
- 멀티플레이어 게임 환경 구성 및 개발
- 다양한 암호 알고리즘 구현
- 구현한 암호 알고리즘 검증 기술 개발
- 안전한 게임 통신 환경 구축 (Handshake Protocol 설계 및 적용)
- 다양한 블록 암호 알고리즘 및 운영모드 적용, 성능 비교
- 멀티플레이어 게임에서 실제 적용 가능한 보안 솔루션!

구현 – 암호 알고리즘

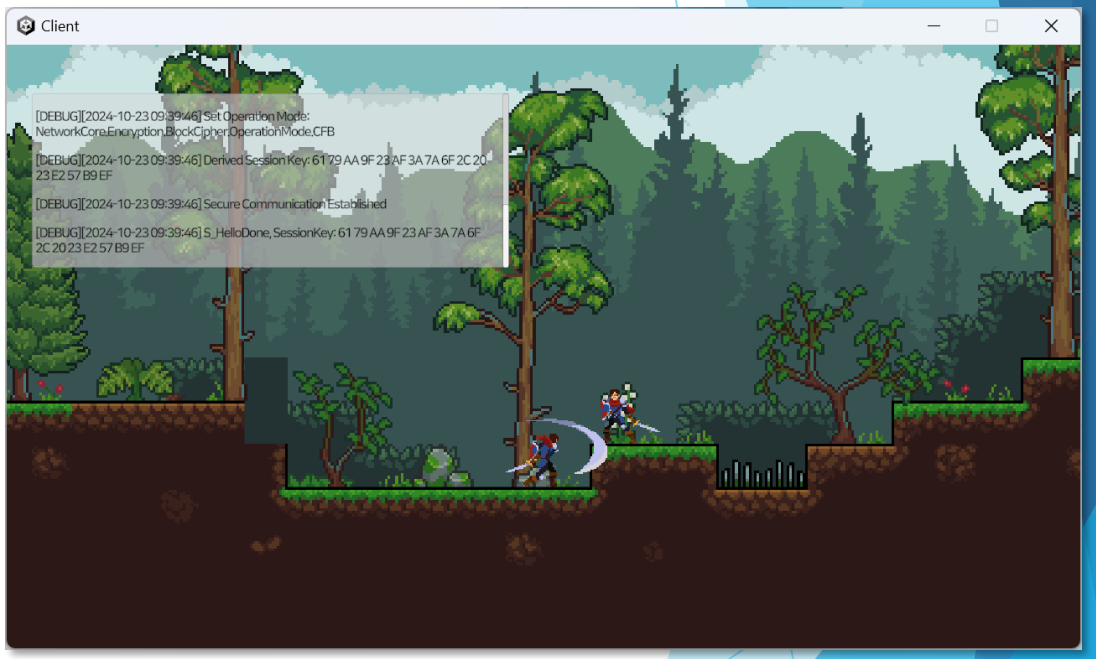
암호 알고리즘	종류
전자 서명 알고리즘	ECDSA (P-256, SHA-256)
키 교환 알고리즘	ECDH (P-256, SHA-256)
키 유도 함수	HKDF (SHA-256)
블록 암호 알고리즘	AES, ARIA, HIGHT, SPECK, TWINE
블록 암호 운영모드	ECB, CBC, CFB, OFB, CTR, GCM

통신에 사용할 암호 알고리즘 구현

구현 – MMORPG 게임



서버

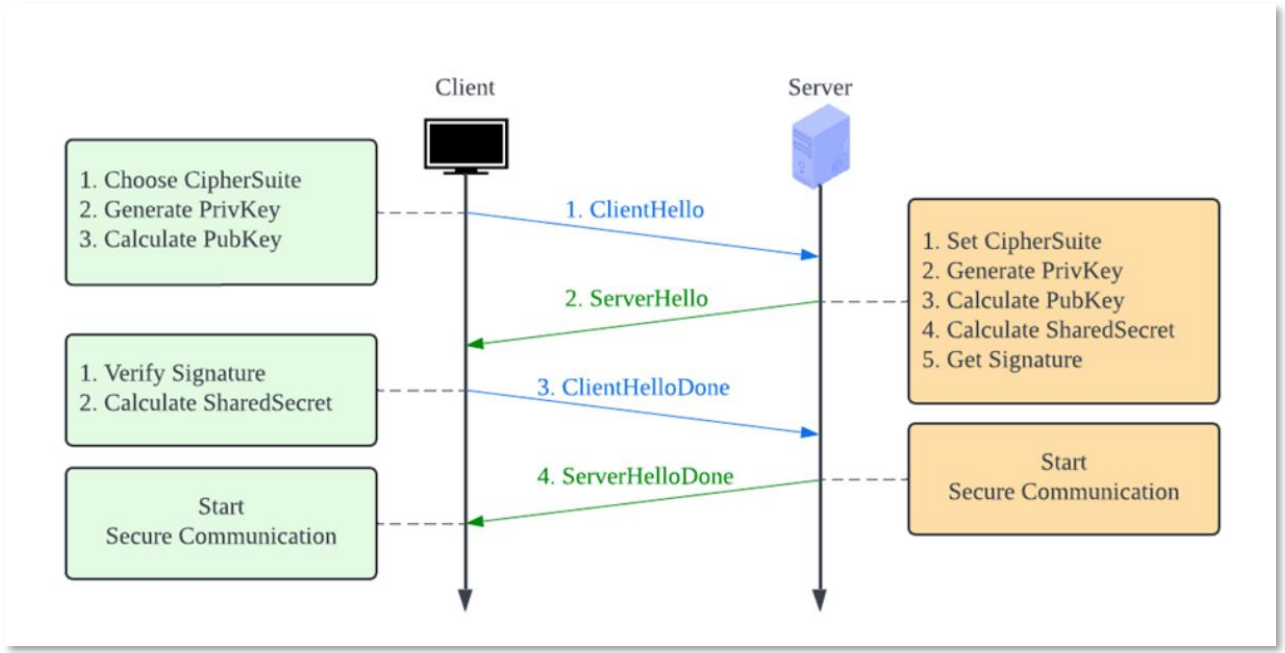


클라이언트

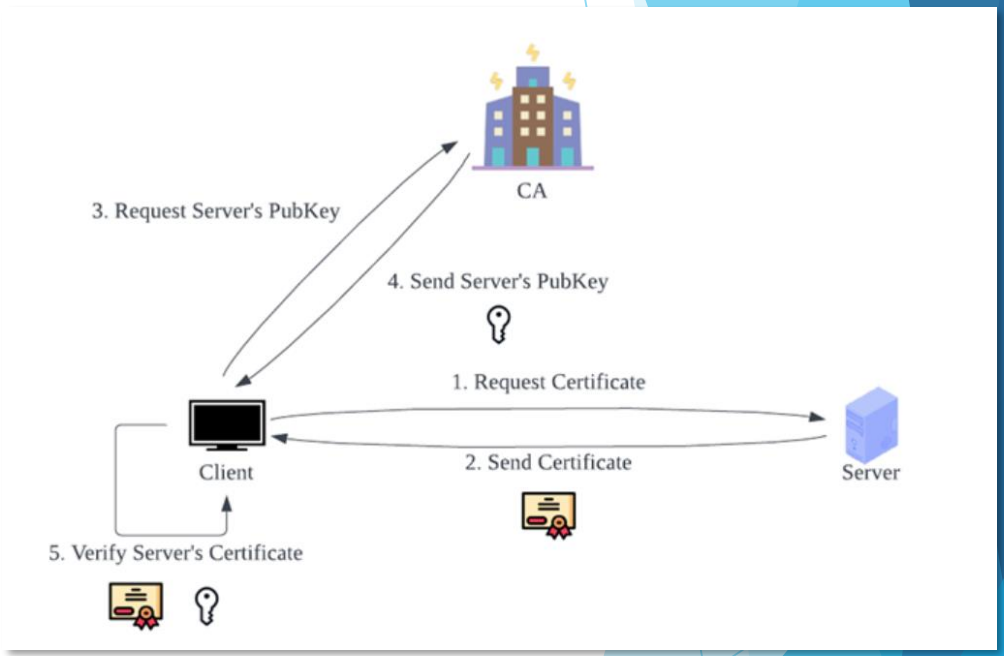
서버: IOCP
클라이언트: Unity

실제 MMORPG 환경 구현

구현 - Handshake Protocol



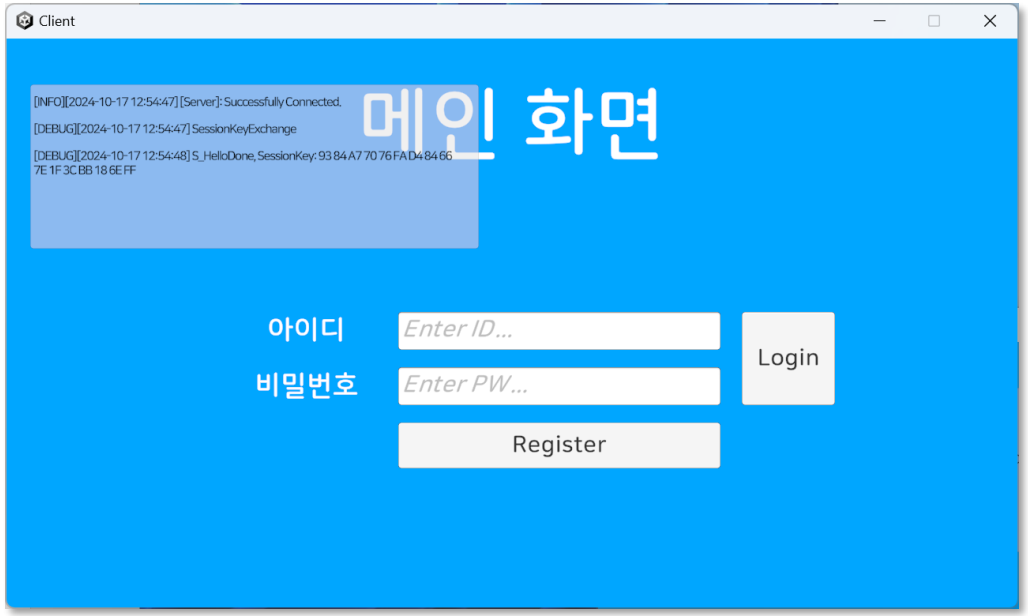
Handshake Protocol



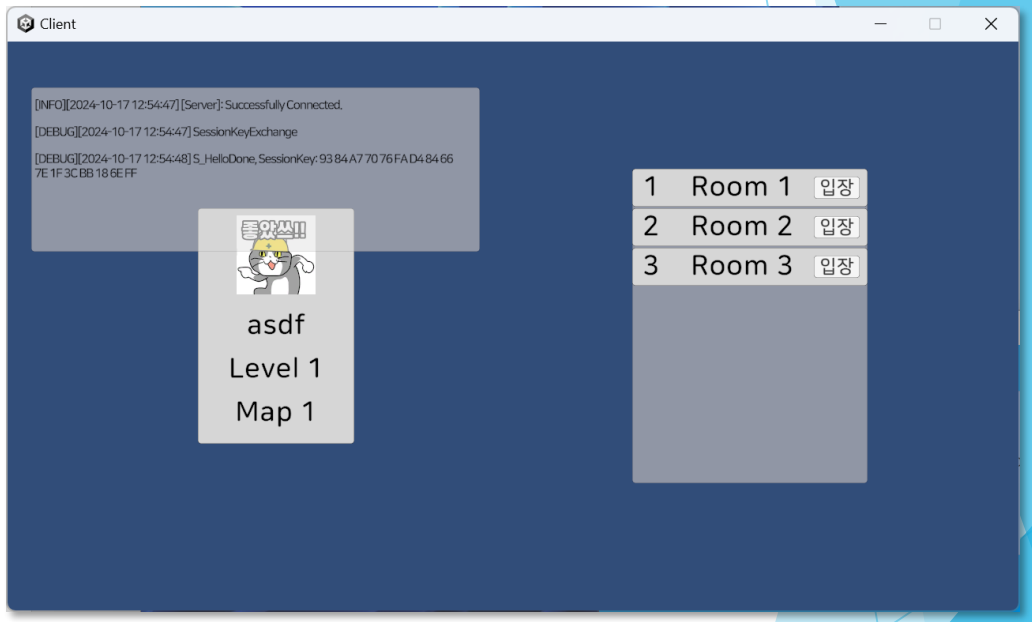
서버 인증 과정

CipherSuite 선택, 서버 인증, 키 교환 및 유도

개발 과정 – 게임 접속 환경



로그인 화면



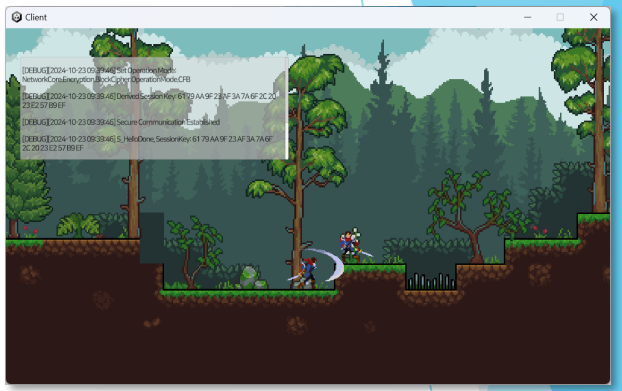
방 입장 화면

게임 클라이언트는 유니티 사용

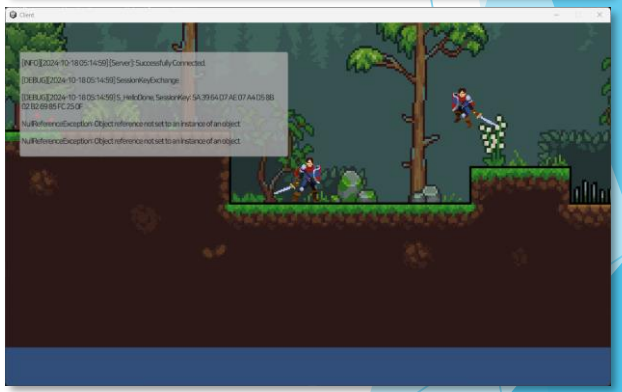
개발 과정 – 멀티플레이어 환경

멀티플레이어 구현(원하는 만큼 플레이어 추가 가능)

공격



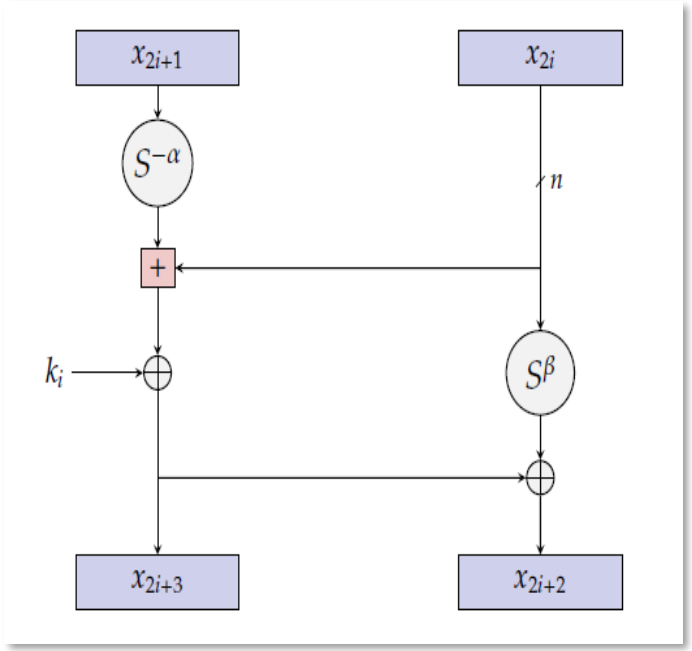
이동



실제 동작 화면 (동기화 되어 있음)

실제 MMORPG와 같은 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 – 구현 검증 기술

```
Msg = c35e2f092553c55772926bdbe87c9796827d17024dbb9233a545366e2e5987dd344deb72df987144b8c6c
d = 0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813
Qx = e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8a
Qy = bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39
k = 6d3e71882c3b83b156bb14e0ab184aa9fb728068d3ae9fac421187ae0b2f34c6
R = 976d3a4e9d23326dc0baa9fa560b7c4e53f42864f508483a6473b6a11079b2db
S = 1b766e9ceb71ba6c01dcd46e0af462cd4cfa652ae5017d4555b8eeefe36e1932

Msg = 3c054e333a94259c36af09ab5b4ff9beb3492f8d5b4282d16801daccb29f70fe61a0b37ffef5c04cd1b70
d = e283871239837e13b95f789e6e1af63bf61c918c992e62bca040d64cad1fc2ef
Qx = 74ccd8a62fba0e667c50929a53f78c21b8ff0c3c737b0b40b1750b2302b0bde8
Qy = 29074e21f3a0ef88b9efdf10d06aa4c295cc1671f758ca0e4cd108803d0f2614
k = ad5e887eb2b380b8d8280ad6e5ff8a60f4d26243e0124c2f31a297b5d0835de2
R = 35fb60f5ca0f3ca08542fb3cc641c8263a2cab7a90ee6a5e1583fac2bb6f6bd1
S = ee59d81bc9db1055cc0ed97b159d8784af04e98511d0a9a407b99bb292572e96
```

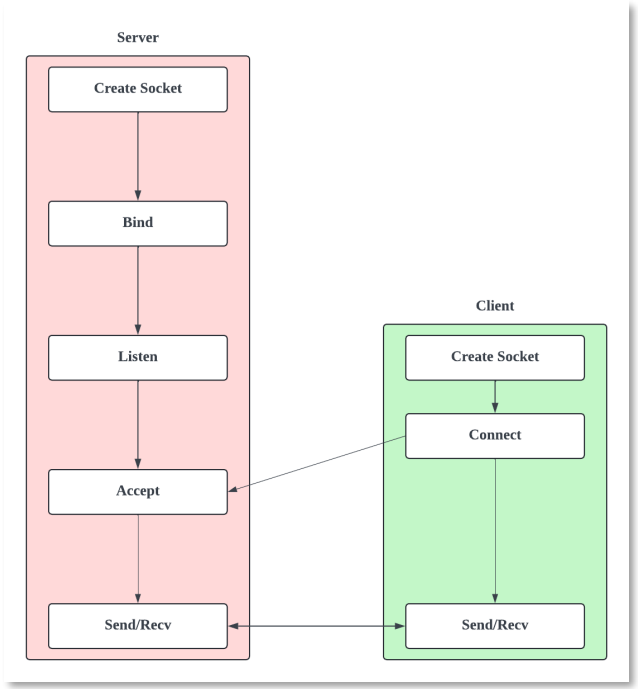
ECDSA 테스트 벡터

```
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.Write("vector.Msg: ");
        printHex(vector.Msg);
        Console.WriteLine();

        Console.Write($"vector.Key: {vector.Key:X}\n");
        Console.Write($"vector.Qx:  {vector.Qx:X}\n");
        Console.Write($"vector.Qy:  {vector.Qy:X}\n");
        Console.Write($"vector.K:   {vector.K:X}\n");
        Console.Write($"vector.R:   {vector.R:X}\n");
        Console.Write($"vector.S:   {vector.S:X}\n");

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

        return false;
    }
}

return true;
```

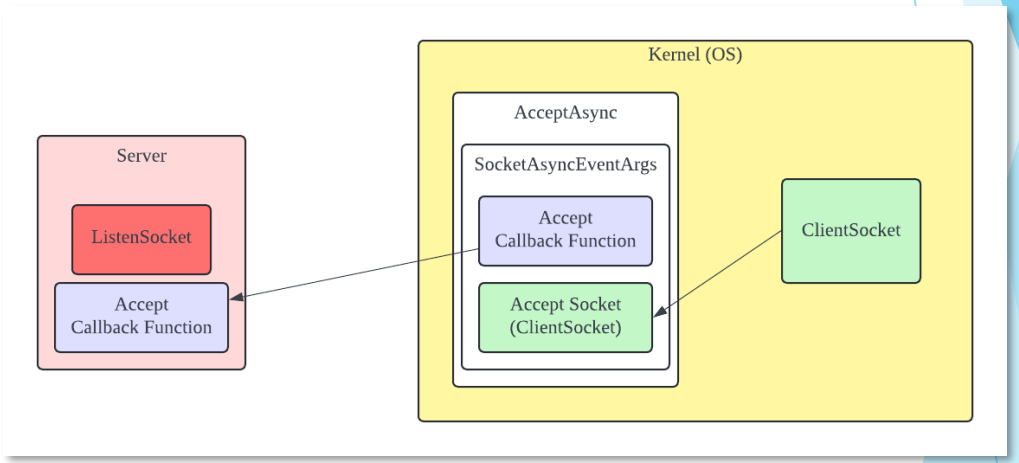
ECDSA 테스트 코드

구현한 암호 알고리즘은 테스트 벡터를 통해 검증 (표준 테스트 벡터)

개발 과정 – 게임 서버



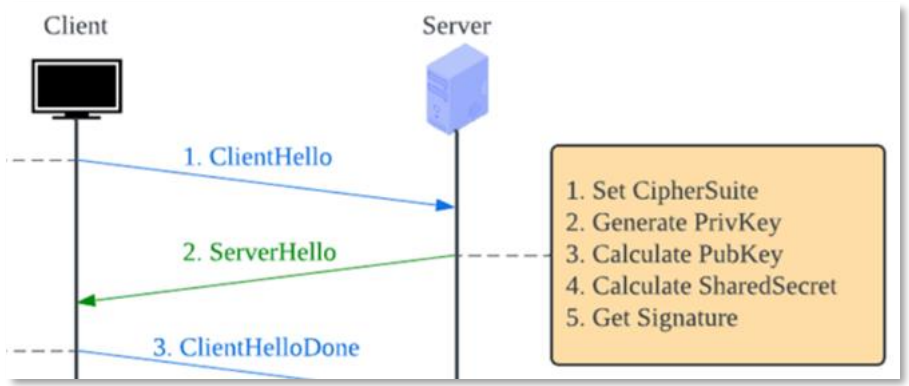
TCP 연결 과정



비동기 I/O 처리 과정

IOCP 모델 채택, 비동기 I/O 사용

개발 과정 – Handshake Protocol



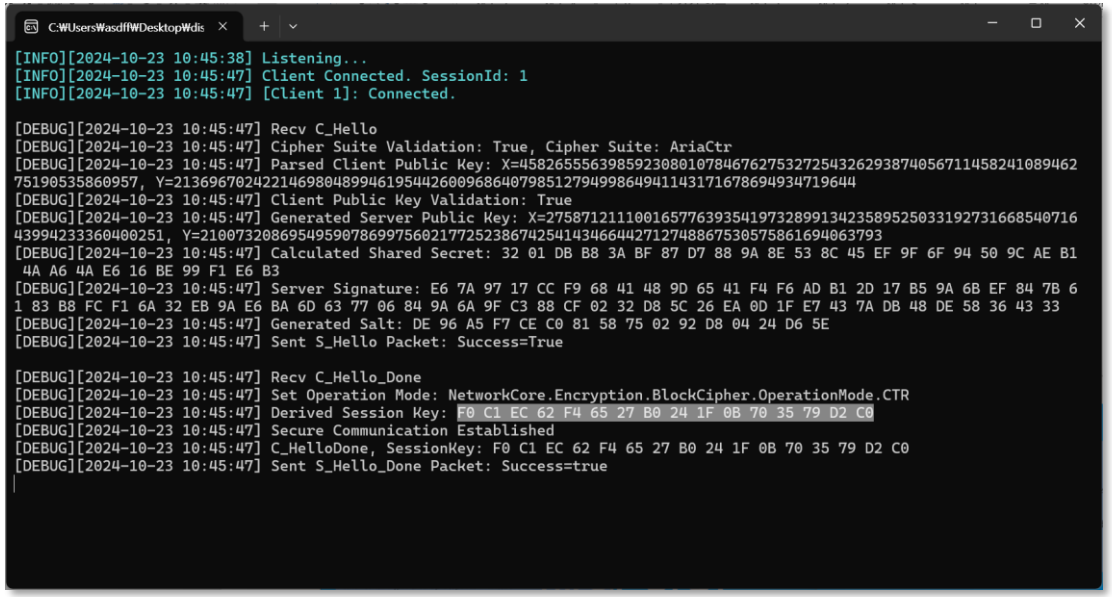
ClientHello 처리

```
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);
```

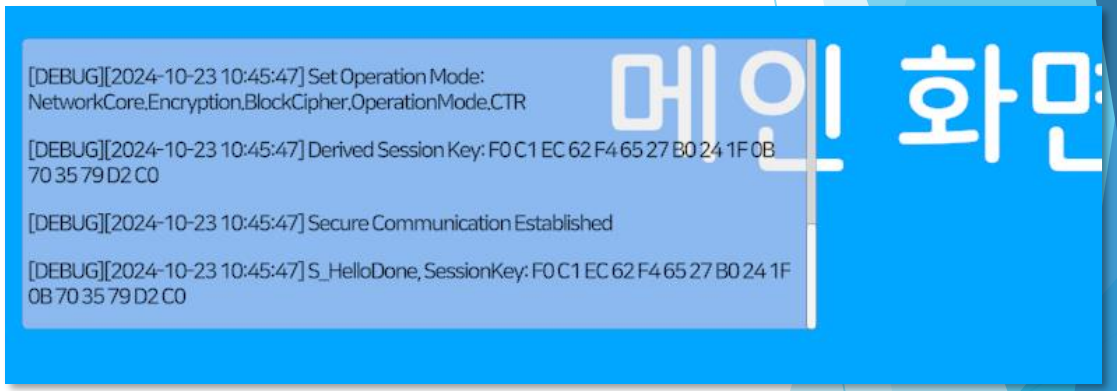
ClientHello 처리 코드

Handshake Protocol 설계, 단계별 작업 구현

개발 결과 – 안전한 게임 통신 환경



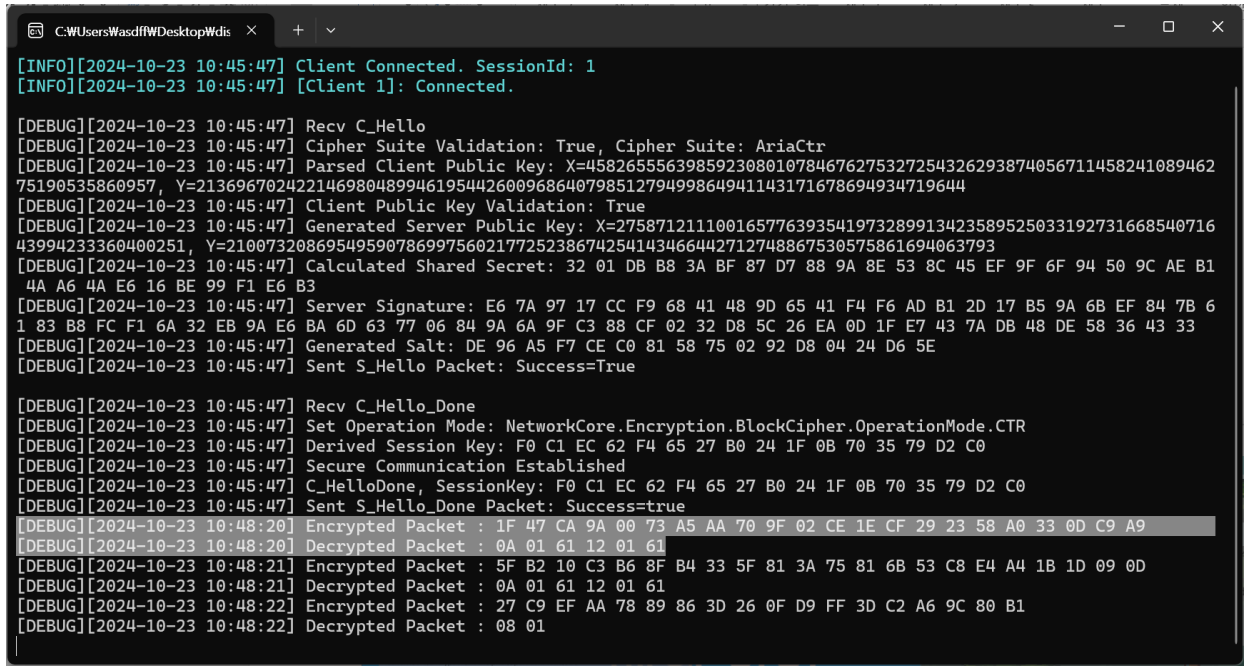
서버의 세션 키 (대칭 키)



클라이언트의 세션 키 (대칭 키)

서버와 클라이언트의 세션 키가 같음 → 대칭 키 유도 성공!

개발 결과 – 안전한 게임 통신 환경



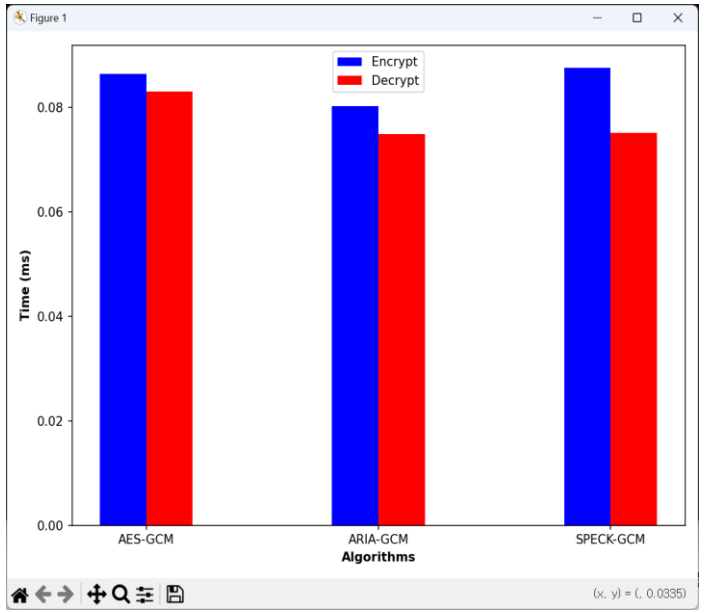
서버의 패킷 복호화

세션 키를 통한 패킷 복호화가 성공적으로 이루어짐

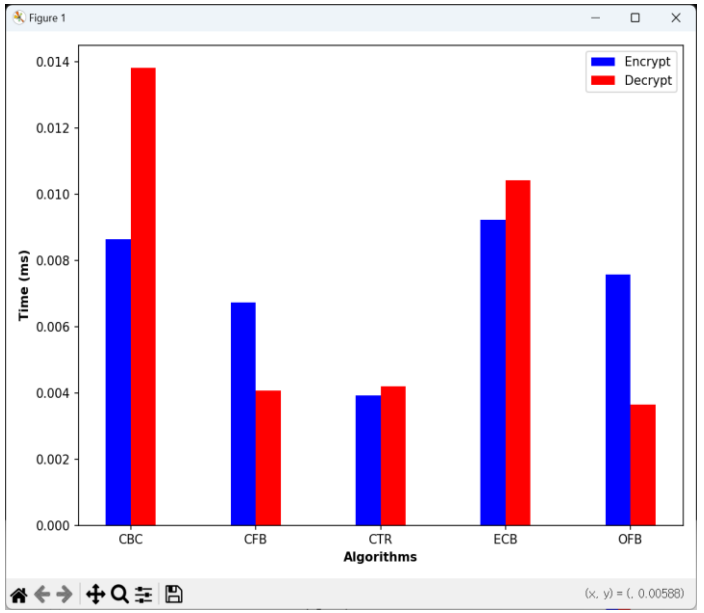
개발 결과 – 블록 암호가 적용된 운영모드별 성능 비교

일반적인 레이턴시

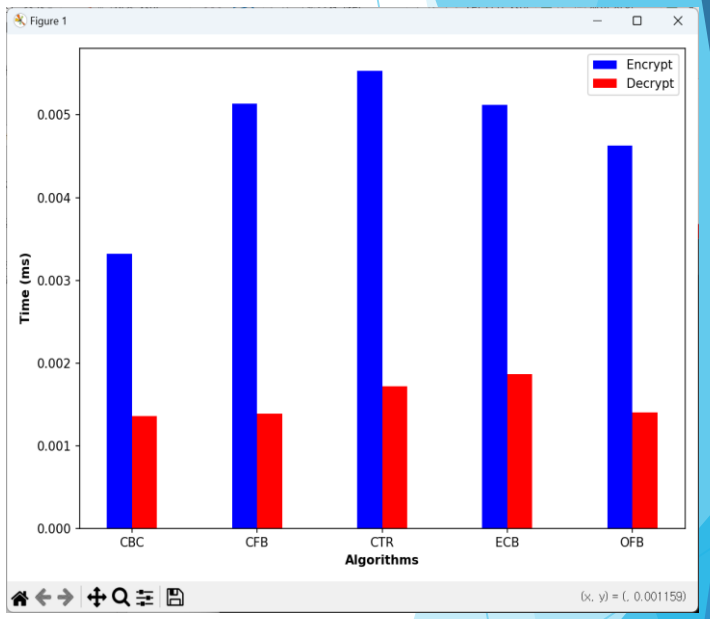
10 ms



GCM 성능 (0.08ms 이하)



AES 성능 (0.014 ms 이하)



SPECK 성능 (0.006 ms 이하)

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

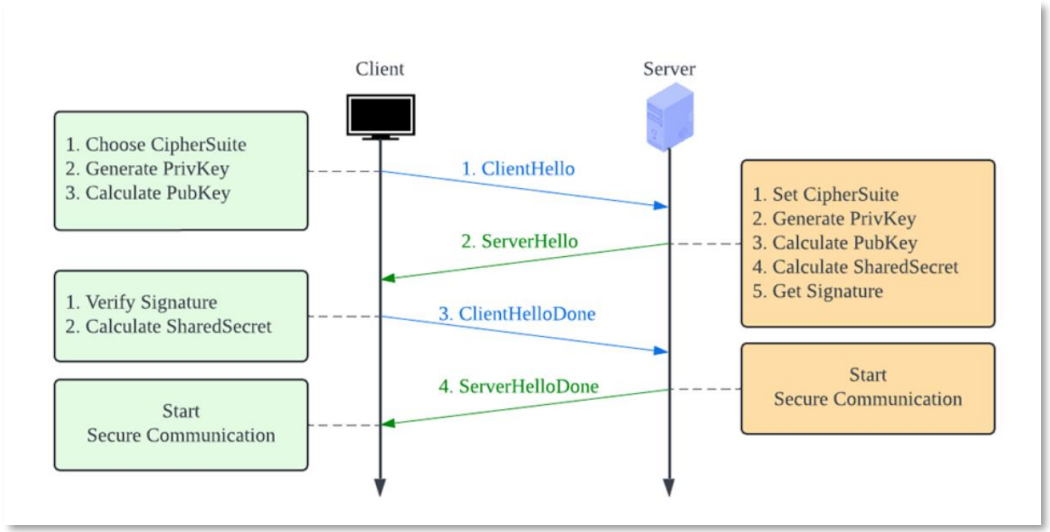
통신에 사용할 암호 알고리즘 구현

성능 비교 결과 플레이에 영향을 주지 않도록 개발함.
구현한 모든 알고리즘을 게임 제조사가 선택하여 적용할 수 있음

결론

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

- 성능 측정 환경이 한정적
  - → 다양한 네트워크 환경에서 좀 더 복잡한 조건으로 성능을 평가할 필요가 있음.

- 적용되는 게임 종류가 한정적임
  - → FPS, AOS 장르 등 다양한 멀티플레이어 게임에서 성능을 평가할 필요가 있음.

감사합니다