핀테크 산업 응용

엄현상 교수님 강의 실습 5차시
송만섭, 반지훈
2022/05/25

Contents

1. 암호 해독 실습
2. 샤미르 비밀 공유 실습
3. 과제 설명

1) 암호 해독(Cryptanalysis)

> 시스템의 숨겨진 면들을 연구하기 위해 정보 시스템을 분석

> 암호 키가 알려져 있지 않은 경우에도 암호 보안 시스템을 파괴하고 암호화된 메시지의 내용에 접근하기 위해 사용

1) 암호 해독의 종류

- 암호문 단독 공격 (Ciphertext Only Attack)
- 기지 평문 공격 (Known Plaintext Attack)
- 선택 평문 공격 (Chosen Plaintext Attack)
- 선택 암호문 공격 (Chosen Ciphertext Attack)

2) 암호문 단독 공격 (Ciphertest Only Attack)

- > COA라고 부름
- > 가장 적은 정보를 가지고 해독해야 하기 때문에 가장 불리한 공격 방식
- > 단지 암호문만을 가지고 plaintext를 찾는 방식

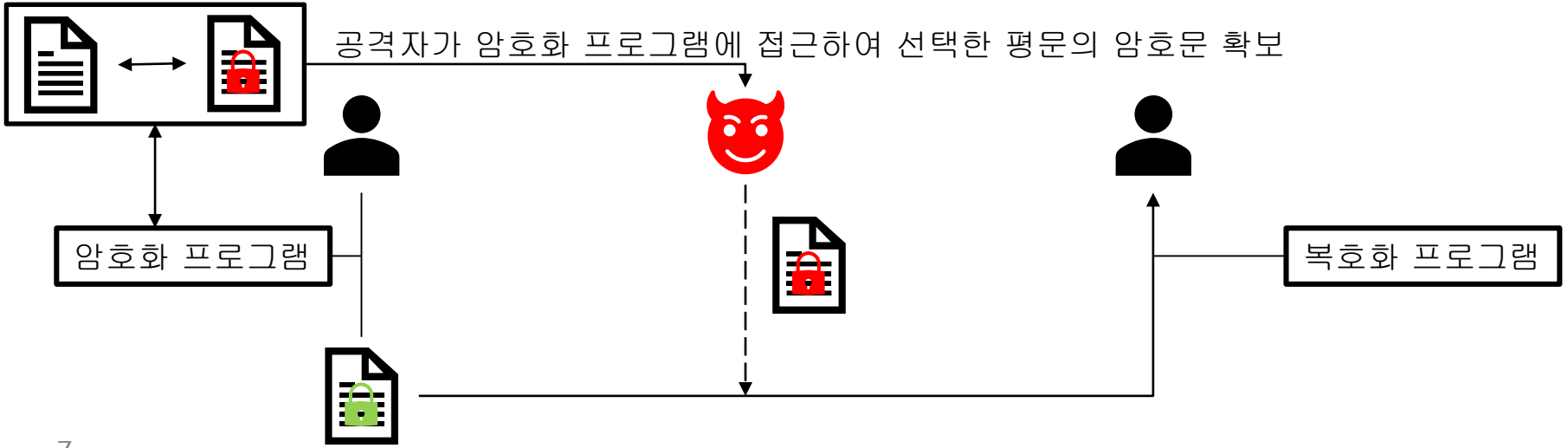
3) 기지 평문 공격 (Known Plaintext Attack)

- KPA라고 부름
- 공격자는 plaintext(일부)와 ciphertext를 모두 알고 있음
- 만약 공격자가 키를 알고 있다면, 같은 키로 암호화한 다른 메시지도 복호화 가능함

4) 선택 평문 공격 (Chosen Plaintext Attack)

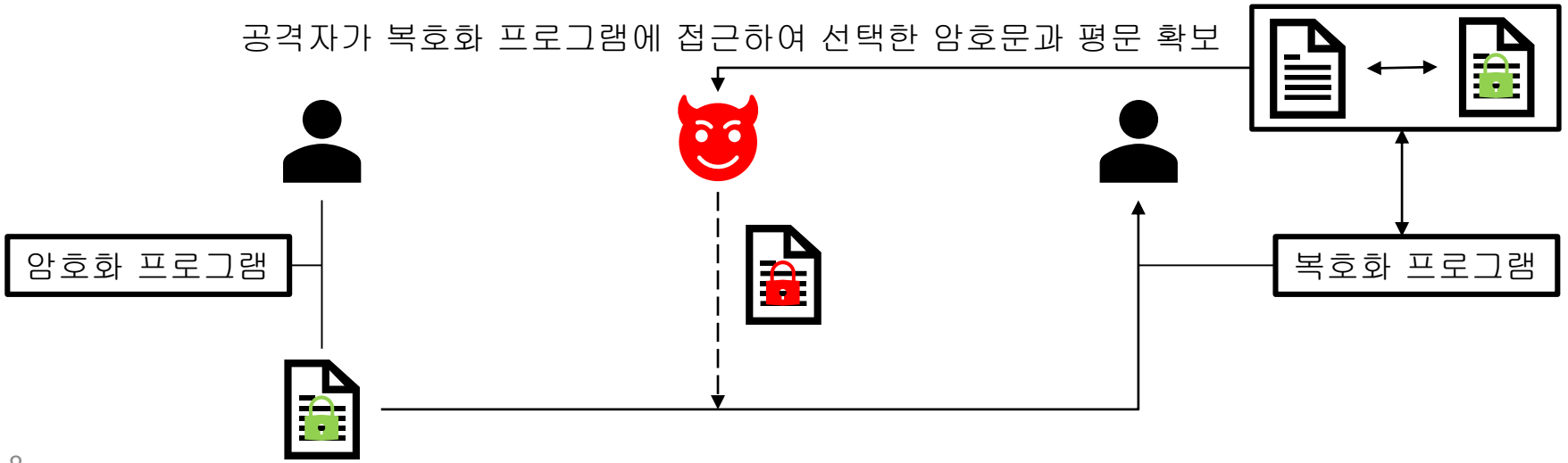
> CPA라고 부름

> 공격자는 암호문 프로그램에서 plaintext를 선택하면 대응하는 ciphertext를 확보함



5) 선택 암호문 공격 (Chosen Ciphertext Attack)

- CCA라고 부름
- 공격자는 복호화 프로그램에 접근하여 특정 ciphertext를 선택하면 대응하는 plaintext를 확보함



CaesarCipher 실습

1) 고대 암호의 일종인 카이사르 암호(CaesarCipher)를 구현해보자

2) 카이사르 암호란?

> 알파벳을 몇번 옮겨 치환하여 바꿔 적는 암호화 방법

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

> Example

```
Enter the string to be encrypted and key:
abc
Enter the key:
1
bcd
```

```go
package main
import "fmt"

func main() {

    var input string
    var key int
    fmt.Println("Enter the string to be encrypted and key:")
    fmt.Scanln(&input)
    fmt.Println("Enter the key:")
    fmt.Scanln(&key)
    for _, char:= range input{
        //rune: 유니코드를 표현하는 타입
        if rune(char)>='A' && rune(char)<='Z'{

            fmt.Printf(string(rotate(rune(char), 'A', key)))
        }else if rune(char)>='a' && rune(char)<='z'{

            fmt.Printf(string(rotate(rune(char), 'a', key)))
        }else{
            fmt.Printf(string(char))
        }
    }

}
func rotate(r rune,base, key int) rune{
    temp:=((int(r)-base)+key)%26
    temp=temp+base
    return rune(temp)
}
```

* 기지 평문 공격 (Known Plaintext Attack) 실습

1) 하나의 단어(plaintext의 일부)를 입력값으로 주어서 전체 ciphertext를 복호화하는 실습
   > 키는 XOR 연산을 사용하여 암호화했다고 가정

2) ./unxor -f xored_file_ABCDEF -g leggings
   > -f 뒤에 나올 것
     - Ciphertext (xored_file_ABCDEF)
   > -g 뒤에 나올 것
     - 일부 Plaintext

* 기지 평문 공격 (Known Plaintext Attack) 실습

> 전체 plaintext

```
1  Lorem ipsum dolor amet tacos leggings readymade truffaut. Organic before they sold out banjo succulents. Tofu sartorial messenger bag, twee wolf tbh deep v aesthetic hot chicken direct
   trade waistcoat schlitz synth. Bicycle rights plaid occupy flannel butcher, aesthetic hell of keytar bespoke chillwave tbh. Banjo quinoa retro af kale chips salvia franzen pitchfork XOXO
   tousled hella aesthetic austin.
```

> 전체 ciphertext

* 기지 평문 공격 (Known Plaintext Attack) 실습 (Cont'd)

3) 드랍박스로 코드 공유

> https://www.dropbox.com/s/yx1a54nleiucj39/%EC%95%94%ED%98%B8%ED%95%B4%EB%8F%85.zip?dl=0

4) go env -w GO111MODULE=off

> GO111MODULE=on이면 패키지 모듈을 사용하겠다는 뜻이므로 gopath를 찾지 않고 goroot에서만 찾게 됨

> Go env -w GO111MODULE=off 명령어를 사용하면 패키지 모듈 대신 GOPATH에 패키지 코드를 저장

- cp -r unxorlib/ $GOPATH/src/

* 기지 평문 공격 (Known Plaintext Attack) 실습 (Cont'd)

5) 실행 결과

```
/tmp/demo $ $GOBIN/unxor -h
Usage of /Users/tomchop/code/go/bin/unxor:
  -f string
        Filename to decrypt
  -g string
        Known plaintext (string)
  -gh string
        Known plaintext (hex encoded)
/tmp/demo $ xxd xored_file_ABCDEF | head
00000000: e7a2 9dce a0cf c2bd 9cde a0cf cfa2 83c4  ................
00000010: bfcf caa0 8adf ed9b caae 80d8 ed83 ceaa  ................
00000020: 88c2 a388 d8ed 9dce ac8b d2a0 8ecf a8cf  ................
00000030: dfbf 9acd ab8e deb9 c18b 829d ccac 81c2  ................
00000040: aecf c9a8 89c4 bf8a 8bb9 87ce b4cf d8a2  ................
00000050: 83cf ed80 deb9 cfc9 ac81 c1a2 cfd8 b88c  ................
00000060: c8b8 83ce a39b d8e3 cfff a289 deed 9cca  ................
00000070: bf9b c4bf 86ca a1cf c6a8 9cd8 a881 cca8  ................
00000080: 9d8b af8e cce1 cfdf ba8a ceed 98c4 a189  ................
00000090: 8bb9 8dc3 ed8b cea8 9f8b bbcf caa8 9cdf  ................
/tmp/demo $ $GOBIN/unxor -f xored_file_ABCDEF -g leggings
Reading from: xored_file_ABCDEF
Know plaintext: leggings
Trying keylen 1:Searching for "\t\x02\x00\x0e\a\t\x14"
Trying keylen 2:Searching for "\t\f\a\a\x13"
Trying keylen 3:Searching for "\x0e\x05\x13"
Found normalized text at 29
Guessing keystream: "\xef\xab\xcd"
Shifting 29 % 3 bytes: 2
Found good decryption for "\xab\xcd\xef"
Dumped 404 decrypted bytes in xored_file_ABCDEF_decrypted
/tmp/demo $ xxd xored_file_ABCDEF_decrypted|head
00000000: 4c6f 7265 6d20 6970 7375 6d20 646f 6c6f  Lorem ipsum dolo
00000010: 7220 616d 6574 2074 6163 6f73 206c 6567  r amet tacos leg
00000020: 6769 6e67 7320 7265 6164 796d 6164 6520  gings readymade
00000030: 7472 7566 6661 7574 2e20 4f72 6761 6e69  truffaut. Organi
00000040: 6320 6265 666f 7265 2074 6865 7920 736f  c before they so
00000050: 6c64 206f 7574 2062 616e 6a6f 2073 7563  ld out banjo suc
00000060: 6375 6c65 6e74 732e 2054 6f66 7520 7361  culents. Tofu sa
00000070: 7274 6f72 6961 6c20 6d65 7373 656e 6765  rtorial messenge
00000080: 7220 6261 672c 2074 7765 6520 776f 6c66  r bag, twee wolf
00000090: 2074 6268 2064 6565 7020 7620 6165 7374   tbh deep v aest
/tmp/demo $
```

2 샤미르 비밀 공유

1) Shamir Secret Sharing

> 의미

- 메시지(S)를 N개의 조각들로 나누고, 그 중 K조각들을 조합하여 메시지(S)를 복원할 수 있는 방식

> 방식

- 메시지(S)를 N개의 share들로 분리
- K개의 share들을 가지고 메시지 (S) 복원

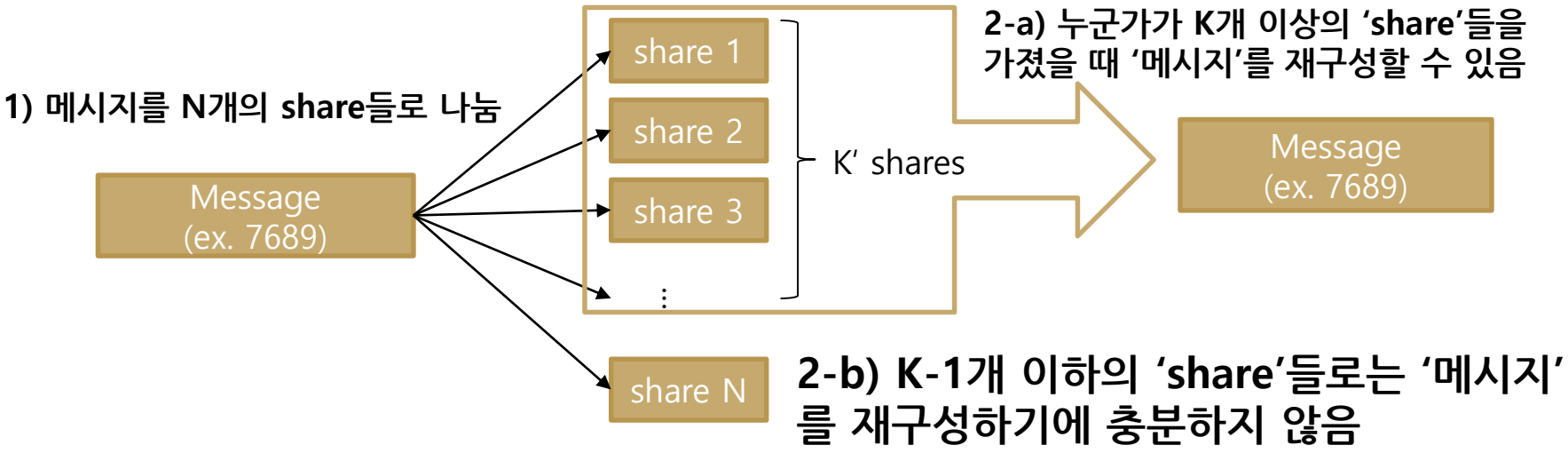
2) 장점

> 여러 개의 키들을 분산시켜 보안성 강화

> 최대 N - K개까지 키들을 분실해도 복원 가능

2 샤미르 비밀 공유 (Cont'd)

3) 동작 과정



2 샤미르 비밀 공유 (Cont'd)

4) 메시지(S)를 N개의 share로 분리

> 2차 다항식에서 계수와 상수를 모두 구해서 2차 다항식을 찾으려면
⇒ 다항식을 지나는 점들의 좌표가 필요함
⇒ N차 다항식의 계수를 모두 알아내려면 N+1의 점들의 좌표가 필요함

$$f(x) = a_1x + a_2x + S$$

구하려는 계수가 3개 이기 때문에 3개의 점들의 좌표가 필요함

$$f(1) = a_1 + a_2 + S$$

$$f(2) = 4a_1 + 2a_2 + S$$

$$f(3) = 9a_1 + 3a_2 + S$$

이를 3개의 좌표를 대입 후, 연립방정식을 활용하여 a_1, a_2,S를 구할 수 있음

2 샤미르 비밀 공유 (Cont'd)

4) 메시지(S)를 N개의 share로 분리

- 메시지(S)의 다항식으로 표현하고 이를 N개의 좌표들로 분리하고, 이 메시지(S)를 가지는 임의의 K-1차 다항식을 생성하면, K개의 점들의 좌표를 이용해서 S를 구할 수 있음
- 상수항을 메시지(S)를 가지는 임의의 K-1차 다항식을 생성하려면, K개의 좌표들 필요 ⇒ 상수 S를 구할 수 있음 (메시지 복원에서 사용됨)

$$f(x) = a_1 x^{k-1} + a_2 x^{k-2} + \cdots a_{k-1} x + S$$

다항식의 상수항은 메시지 S를 나타냄

즉, $f(0) = S$

$$\mathbf{shares} = \{(1, f(1)), (2, f(2)), \ldots, (n, f(n))\}$$

5) K개의 share를 가지고 메시지(S) 복원

> 연립방정식

> 랑그랑주 보간법 (Lagrange interpolation)

- 연립 방정식을 사용하지 않고 다항식을 구하는 방법
  ⇒ 계산 시간이 비교적 짧고 간단하며 자료의 구간에 관계없이 사용 가능함

$$L(x) = \sum_{k=0}^{k-1} y_i l_j(x)$$

여기서 $l_j(x)$는 Lagrange basis polynomial(라그랑주 기초 다항식)이라고 함

$$l_j(x) := \prod_{f=0, f\neq j}^{k-1} \frac{(x - x_f)}{x_j - x_f}$$ 로 정의됨

$$\boxed{L(0) = S}$$ 메시지 (S) 복원

2 샤미르 비밀 공유 (Cont'd)

메시지(S=1)를 n=4조각으로 나누고, k=3조각이 있다면 S를 복원할 수 있도록 함

▼ 1. 메시지(S)를 N개의 share로 분리

1. K=3이므로, 필요한 2(=K-1)차 다항식 생성

$$f(x) = a_1 x^2 + a_2 x + S(\text{메시지})$$

1. a1, a2에 필요한 난수 선택 ⇒ a1 = 2, a2 = 3
2. 2차 다항식 생성

$$f(x) = 2x^2 + 3x + 1$$

3. 다항식으로부터 n = 4개의 점들의 좌표(share) 선택 ⇒ 공유하는 값이됨

$$share = \{(1, 6), (2, 15), (3, 28), (4, 45)\}$$

source: 사이버 보안과 블록체인 개론 및 실습 (서울대학교)

2 샤미르 비밀 공유 (Cont'd)

▼ 2. K개의 share를 가지고 메시지(S) 복원

1. K개의 share 수집

$$K\text{개의 } share = \{(1,6),(2,15),(3,28)\} = \{(x_0,y_0),(x_1,y_1),(x_2,y_2)\}$$

2. 랑그랑주 기초 다항식을 활용하여 L(x) 다항식 계산

$$\text{수식 } L(x) := \sum_{j=0}^{k-1} y_j l_j(x)$$

$$l_j(x) := \prod_{f=0, f=j}^{k-1} \frac{x-x_f}{x_j-x_f} \text{을 활용}$$

$$l_0(x) = \frac{x-x_1}{x_0-x_1}\cdot\frac{x-x_2}{x_0-x_2}$$

$$l_1(x) = \frac{x-x_0}{x_1-x_0}\cdot\frac{x-x_2}{x_1-x_2}$$

$$l_2(x) = \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1}$$

$$l_0(x) = \frac{x-2}{1-2}\cdot\frac{x-3}{1-3} = \frac{x^2-5x+6}{2}$$

$$l_1(x) = \frac{x-1}{2-1}\cdot\frac{x-3}{2-3} = -x^2+4x-3$$

$$l_2(x) = \frac{x-1}{3-1}\cdot\frac{x-2}{3-2} = \frac{x^2-3x+2}{2}$$

$$L(x) = 6(\frac{x^2-5x+6}{2}) + 15(-x^2+4x-3) + 28(x^2-3x+22)$$

$$L(x) = 2x^2+3x+1$$

3. 메시지(S) 복원 == L(0) 계산함 ⇒ L(x)에서 x에 0을 대입하여 메시지(S) 계산

$$L(0) = 1$$

source: 사이버 보안과 블록체인 개론 및 실습 (서울대학교)

* 샤미르 비밀 공유 실습

1) create 함수

> 메시지를 암호화하기 위해 랜덤하게 생성된 방정식의 (x,y) 값을 생성

```
function create(message, K, N) {
  const messageBuffer = new Buffer.from(message) // 전달 받은 메시지를 Buffer로 변경
  const secrets = [...messageBuffer] // Buffer를 Array형태로 변경
  const polynomial = new Array(K).fill(0) // 다항식 기본 형태 생성

  polynomial[0] = secrets[0] // 비밀키를 다항식의 상수에 저장

  for(let i = 1 ; i < K ; i++) {
    polynomial[i] = getRandomInt() // 각 차수에 계수를 랜덤하게 추출
  }

  const shares = new Array(N) // 각 사용자들에게 나눠줄 share(point) 배열 생성

  for(let i = 0 ; i < N ; i++) {
    shares[i] = new Array(2)
    do {
      x = getRandomInt();
    } while(IsExistX(shares, i, x));

    shares[i][0] = x
    shares[i][1] = evaludatePolynomial(polynomial, x) // f(x)를 계산하여 y에 대입
  }

  return shares
}
```

샤미르 비밀 공유 실습 (Cont'd)

2) combine 함수

> create 함수에서 생성한 (x,y) 값을 일정 threshold 이상 모아서 방정식의 상수값(원본 메시지)을 복원

```
function combine(shares) { // 라그랑주 기초 다항식 공식을 활용하여 다항식 계산 및 L(0) 계산
  let secret = 0

  for(let i = 0 ; i < shares.length ; i++) {
    const share = shares[i]
    const x = share[0]
    const y = share[1]

    let numerator = 1
    let denominator = 1

    for(let j = 0 ; j < shares.length ; j++) { // P48. l_j(X) 공식 계산
      if( i != j) {
        numerator = numerator * -shares[j][0]
        numerator = modulo(numerator, primeNumber)

        denominator = denominator * (x - shares[j][0])
        denominator = modulo(denominator, primeNumber)
      }
    }

    inversed = modInverse(denominator, primeNumber)

    secret = secret + y * (numerator * inversed) // P48. L(X) 공식 계산
    secret = modulo(secret, primeNumber)
  }

  var buffer = new Buffer.alloc(1);
  buffer[0] = secret
  return buffer
}
```

```
const shares = create('1', 5, 10)
console.log(combine(shares.slice(1,6)).toString())
```

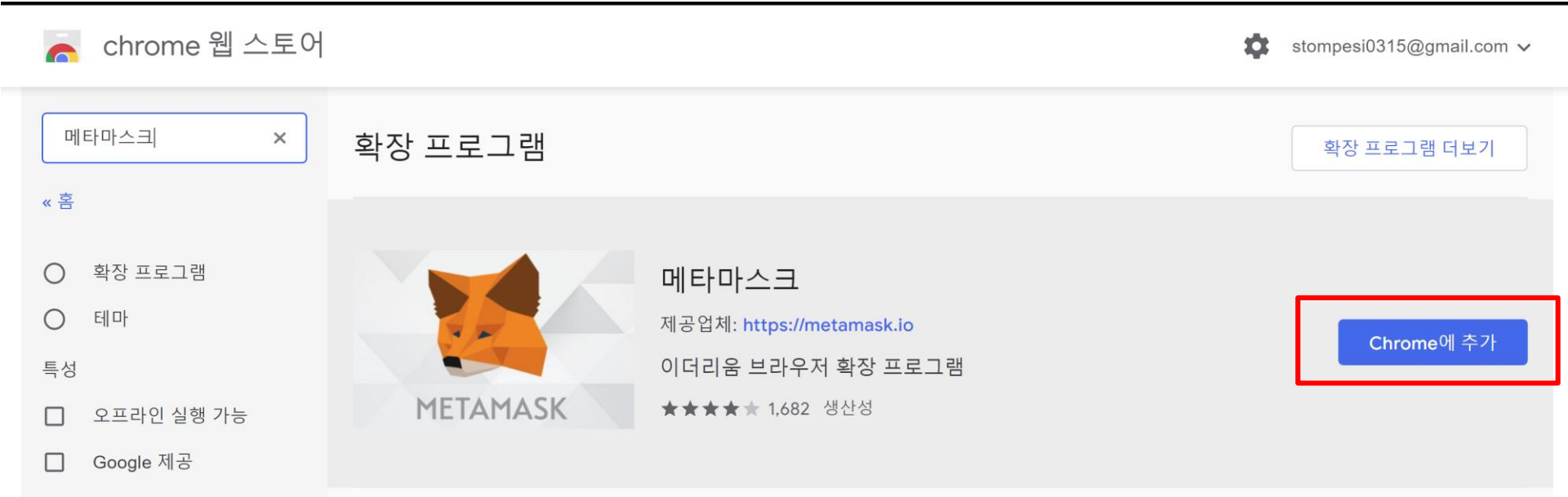
* 과제

과제가 있으니 핀테크 산업 응용 실습 `과제.pdf`를 봐주세요

3 과제 워밍업!

1) 메타마스크 설치

> https://chrome.google.com/webstore/



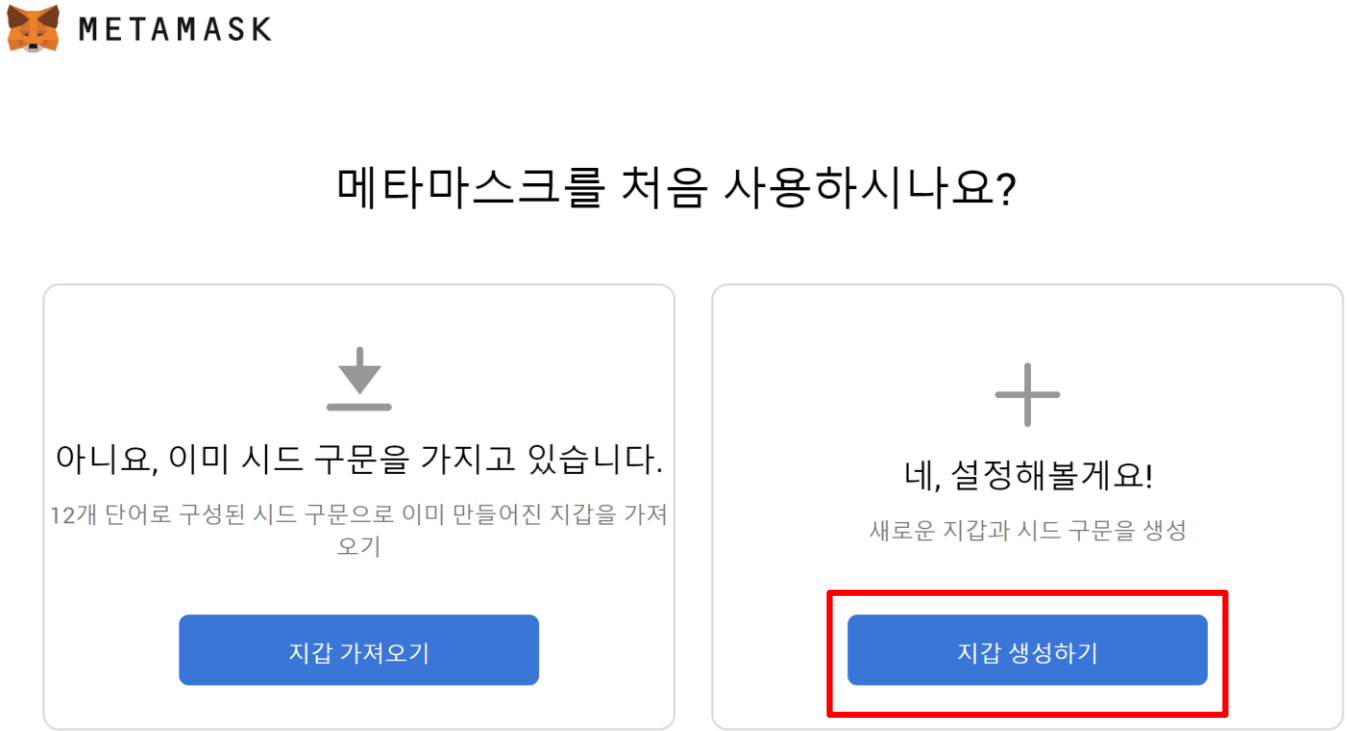
3 과제 워밍업! (Cont'd)

메타마스크 Beta에 오신 것을 환영합니다

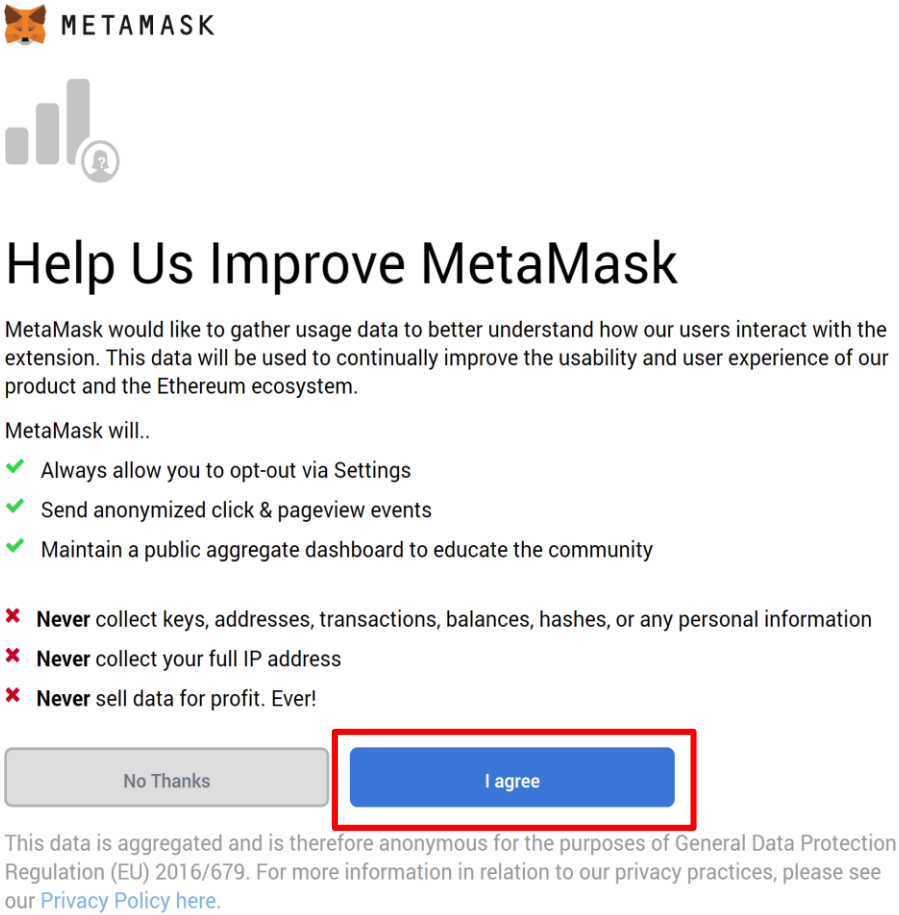
메타마스크는 이더리움을 위한 안전한 저장소입니다.
We're happy to see you.

시작하기

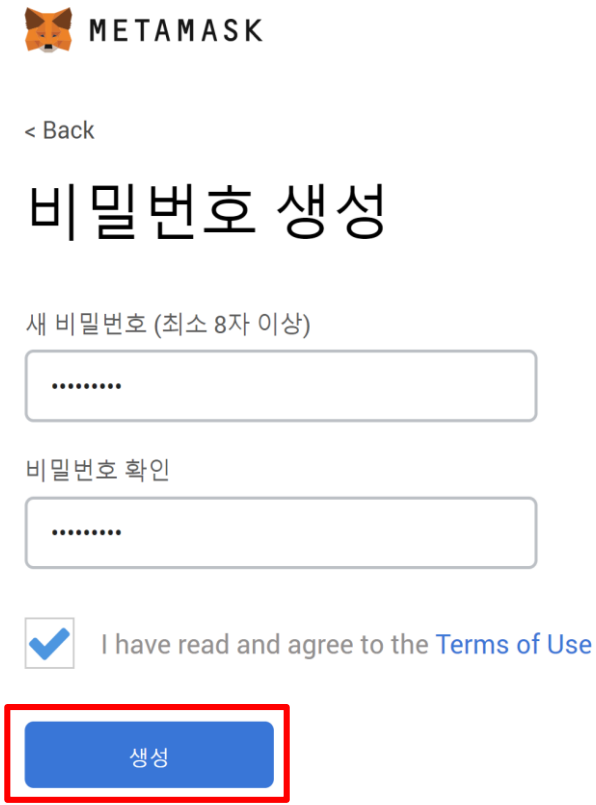
3 과제 워밍업! (Cont'd)



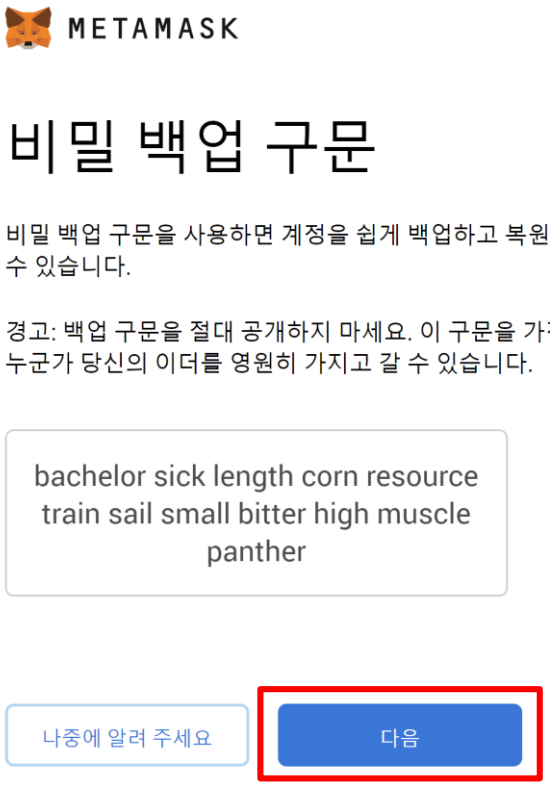
3 과제 워밍업! (Cont'd)



3 과제 워밍업! (Cont'd)

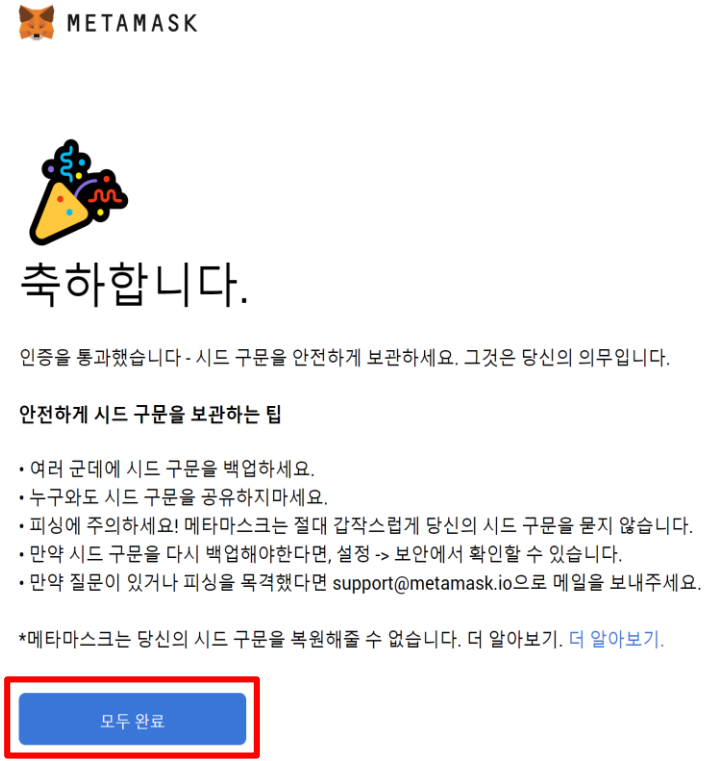


3 과제 워밍업! (Cont'd)



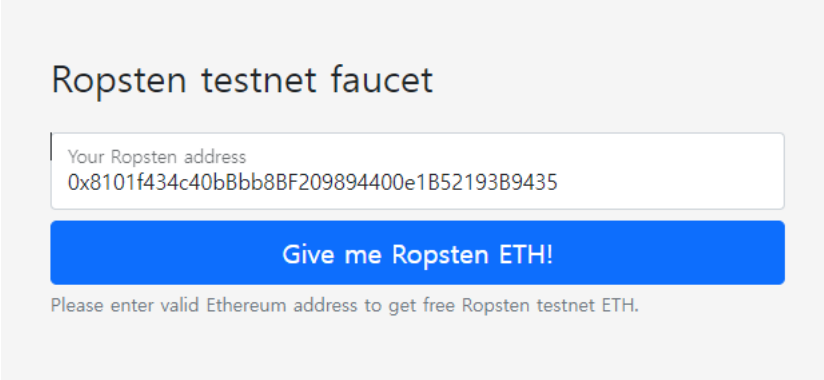
3 과제 워밍업! (Cont'd)

3 과제 워밍업! (Cont'd)

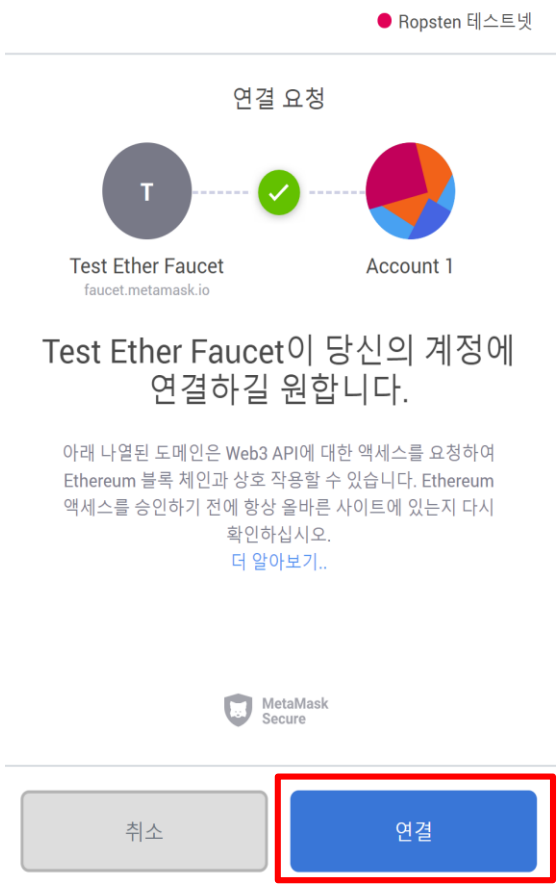


3 과제 워밍업! (Cont'd)

1) https://faucet.egorfine.com/



3 과제 워밍업! (Cont'd)



3 과제 워밍업! (Cont'd)

1 ETH

입금

전송

히스토리

입금

11/25/2019 at 12:10

승인됨

1 ETH

3 과제 워밍업! - javascript 문법

1) 변수의 선언 및 사용

> 변수는 타입에 대한 구분없이 앞에 var 로 선언하고 사용

```
//1. 변수는 앞에 타입에 대한 구분없이 var 로 선언한다
//2. 문장의 끝은 항상 ; (세미콜론)으로 끝낸다
var name;

//3. = 을 사용해서 변수에 값을 입력하는데 문자열 입력시에는
앞뒤로 '(외따옴표) 또는 "(쌍따옴표)를 붙혀준다
name = '홍길동';

//4. 문자나 숫자 및 타입에 관계없이 변수는 var 로 선언한다
var num1;
num1 = 21;

//5. 변수의 선언과 동시에 값을 입력할 수 있다
var num2 = 3;

//6. 두 개의 변수를 더해서 다른 변수에 입력할 수 있다
var sum = num1 + num2;

//7. 숫자와 문자를 더할 경우 결과값은 문자가 된다
아래 연산결과로 sum2 에는 "홍길동21"이 sum3에는 "이순신3"이 입력된다
var sum2 = name + num1;
var sum3 = '이순신' + 3;
```

3 과제 워밍업! - javascript 문법 (Cont'd)

2) 함수의 선언 및 사용

> function 함수이름(파라미터) { 실행코드 } 형태로 선언

> 실행코드에 return 예약어가 없으면 결과값 return 없이 함수가 실행 후에 그대로 종료

```
// 1. 세개의 파라미터를 더한 후 결과값을 리턴하는 함수를 선언
function sum(param1, param2, param3){
    return param1 + param2 + param3;
}

// 2. 함수 실행 후 결과값을 result 에 대입
var result = sum(1,2,3);

// 3. result 에 담긴 결과값을 출력
console.log('result='+result);

// 4. 결과값이 없는 함수의 선언
function print(param1){
    console.log('param1='+param1);
}

// 5. 함수호출 : return 이 없는 함수는 로직을 자체적으로 처리하기 때문에 결과값 대입 불필요
print('출력내용');
```

3) 조건문

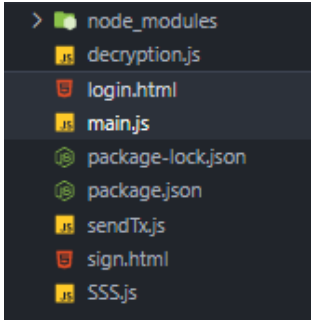
```
var a = 10;

if (a > 11) {
    console.log('a가 11보다 큽니다');
} else if(a == 11) {
    console.log('a가 11과 같습니다');
} else {
    console.log('a가 11보다 작습니다');
}
```

4) 반복문

```
// 0부터 9까지 출력하는 while문
var i=0;
for (i=0;i<10;i++) {
    console.log("for : i의 값은="+i);
    i=i+1;
}
```

3 과제 워밍업! (Cont'd)

1) 과제에 필요한 파일들



- Javascript
  - Main.js: 메인 코드
  - Decryption.js: 암호문 해독
  - sendTx.js: 이더리움 테스트넷 트랜잭션 전송
  - SSS.js: 샤미르 비밀 공유
- HTML
  - Login.html: 로그인 페이지
  - Sign.html: 샤미르 비밀 공유 사인

3 과제 워밍업! (Cont'd)

1) 패키지

> npm init

> npm install --save ethers

```
var querystring = require('querystring');
const {
    create,
    combine
} = require("./SSS")
const {
    sendEthTransaction
} = require("./sendTx")
const {
    decrypt
} = require("./decryption");
const {
    syncBuiltinESMExports
} = require('module');
```

2) 변수

> N: 샤미르 비밀 공유 참가자 수

> Threshold: 샤미르 비밀 공유 복원을 위한 최소 참가자 수

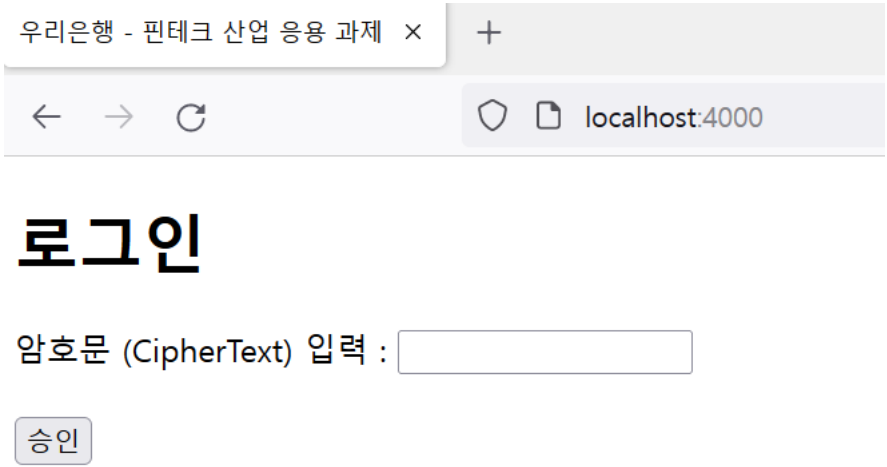
> Shares: 샤미르 비밀 공유 값 〈x,y〉

```
const password = 'Wooribank is good!';

const port1 = 3000;
const port2 = 3001;
const port3 = 3003;
const loginPort = 4000;

const loginTime = 10000
const txTime = 20000 //30seconds
const msg = '@'
const N = 3
const threshold = 2
const shares = new Array(N)
for (var i = 0; i < N; i++) {
    shares[i] = new Array(2)
}
var cipherText = ''
```

3) 로그인 화면

- Ciphertext를 web에 입력하고 복호화 해서 기존 password와 비교하여 로그인

```
function decryptPassword(cipherText) {
    const decryptResult = decrypt(cipherText)
    console.log('decrypt result:', decryptResult)
    if (decryptResult != password) {
        console.log("password is not correct! exit!!")
        process.exit()
    } else {
        console.log("password is correct! Welcome to Wooribank!!!")
    }
}
```

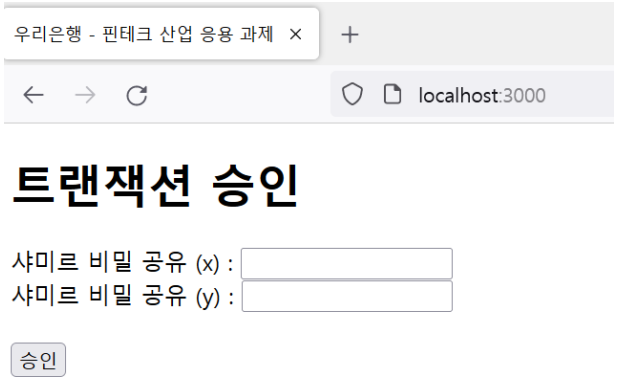


3 과제 워밍업! (Cont'd)

4) 트랜잭션 승인 화면

- 샤미르 비밀 공유에서 create 함수로 만든 <x,y> 값을 웹에 입력 후 승인 버튼 클릭
- 코드에서 정한 일정 threshold 값 이상이 되면 샤미르 비밀 공유의 메시지가 복원되기 때문에 메시지 검증 가능

```
http.createServer(function (req, res) {

    if (req.method == 'GET') {
        console.log("GET")
        fs.readFile('./sign.html', 'utf8', function (error, data) {
            res.writeHead(200, {
                'Content-Type': 'text/html'
            });
            res.end(data);
        });
    } else if (req.method == 'POST') {
        req.on('data', function (chunk) {
            var data = querystring.parse(chunk.toString());
            // console.log(data.x, data.y);
            shares[0][0] = data.x;
            shares[0][1] = data.y;
            res.writeHead(200, {
                'Content-Type': 'text/html'
            });
            res.end('transaction signed');
        });
    }

}).listen(port1, function () {
    console.log('Server1 is running...');
});
```



3 과제 워밍업! (Cont'd)

5) 이더리움 Ropsten 테스트 네트워크에 트랜잭션 전송 코드

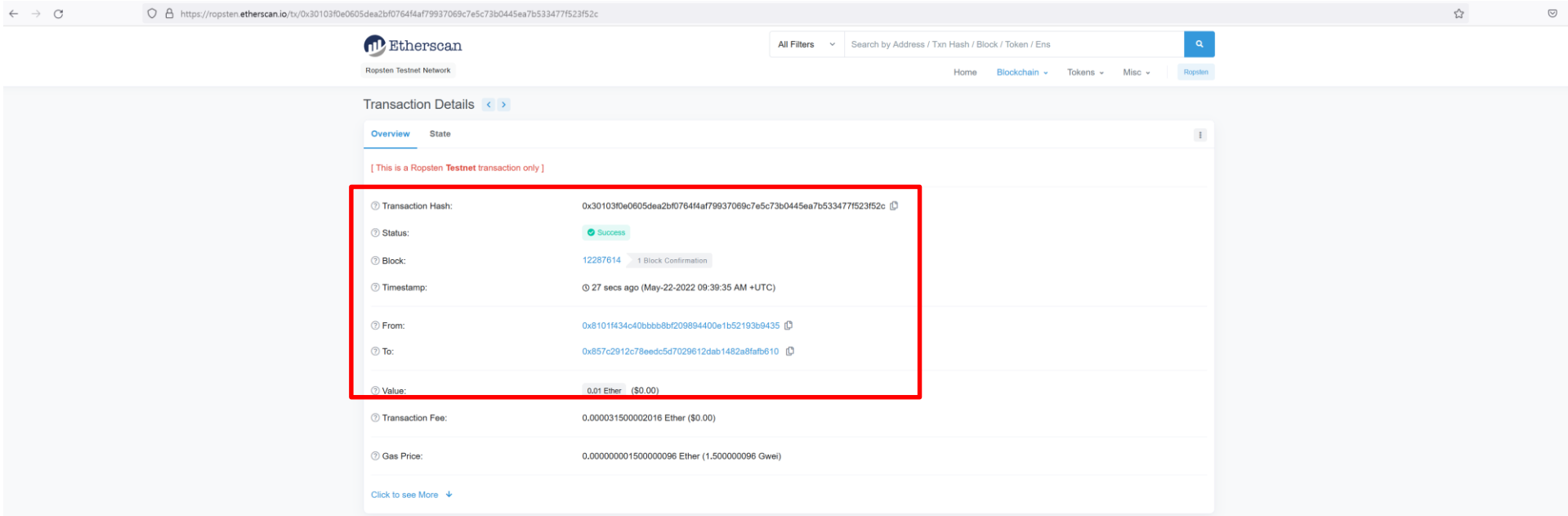
> 내 지갑의 private key, 나의 주소값 (from_address), 상대방의 주소값 (to_address)은 필수

```
function sendEthTransaction() {
  let private_key =
    "0xbd0809435507c                                                    78e9f4"
  let send_token_amount = "0.01"
  let to_address = "0x857c2912C78EeDC5d7029612DaB1482a8fAFb610"
  let from_address = "0x8101f434c40bBbb8BF209894400e1B52193B9435"
  let gas_limit = "0x100000"
  ethersProvider = new ethers.providers.InfuraProvider("ropsten")

  let wallet = new ethers.Wallet(private_key)

  let walletSigner = wallet.connect(ethersProvider)

  ethersProvider.getGasPrice() // gasPrice

  const tx = {
    from: from_address,
    to: to_address,
    value: ethers.utils.parseEther(send_token_amount),
    nonce: ethersProvider.getTransactionCount(from_address, "latest"),
    gasLimit: ethers.utils.hexlify(gas_limit), // 100000
  }

  //프로미스는 주로 서버에서 받아온 데이터를 화면에 표시하기 위해 사용한다.

  walletSigner.sendTransaction(tx).then((transaction) => {
    console.log(transaction)
    console.log("Send finished!")
  })


}
```

6) 트랜잭션 전송 결과 (콘솔)

```
mssong@DESKTOP-FUFOE0E:~/wooriBank/assignment$ node sendTx.js
{
  type: 2,
  chainId: 3,
  nonce: 36,
  maxPriorityFeePerGas: BigNumber { _hex: '0x59682f00', _isBigNumber: true },
  maxFeePerGas: BigNumber { _hex: '0x59682fd4', _isBigNumber: true },
  gasPrice: null,
  gasLimit: BigNumber { _hex: '0x100000', _isBigNumber: true },
  to: '0x857c2912C78EeDC5d7029612DaB1482a8fAFb610',
  value: BigNumber { _hex: '0x2386f26fc10000', _isBigNumber: true },
  data: '0x',
  accessList: [],
  hash: '0x30103f0e0605dea2bf0764f4af79937069c7e5c73b0445ea7b533477f523f52c',
  v: 0,
  r: '0x0f882d931babcd68e1b34da5847ce4059f882df6493f235dd3c1bf4a48bfdb05',
  s: '0x2118688403ec7db937889cd8fad3c7b4c3fb4b252e3f5ba1cc75ae6a8c501362',
  from: '0x8101f434c40bBbb8BF209894400e1B52193B9435',
  confirmations: 0,
  wait: [Function (anonymous)]
}
Send finished!
```

7) 트랜잭션 전송 결과 (Etherscan)



Thank you!
수고하셨습니다!