Graduate Project

Securely computing the set intersection between X and Y

Myungsun Kim

Fall, 2020

Information_Security@The Univ. of Suwon

The roadmap

- Motivations
- Phase I
- Phase II
- Final presentation

§1. Motivations

A motivating scenario: S#1

Aviation security

- Client: K 항공사, Server: 질병관리본부 (KCDC)
- Scenario
  - 인류가 Virus-ζ에 의해 고통받고 있는 가까운 미래의 어느날 K 항공사는 인천을 출발하여 이스탄불로 향하는 비행을 준비중
  - K 항공사는 출발전 n명의 탑승객의 신상 정보가 담긴 화일 $X = \{x_1, x_2, \ldots, x_n\}$ 중에서 Virus-ζ 양성반응 승객의 확인이 필요
  - KCDC는 Virus-ζ에 양성 판정을 받은 m 명의 확진자의 화일 $Y = \{y_1, y_2, \ldots, y_m\}$ 보유
- Technical goal:
  - K 항공사는 승객의 신상정보를 KCDC에 제공하지 않고 Virus-ζ 확진자만 확인

A motivating scenario: S#1

A naive solution

- Client: K 항공사, Server: 질병관리본부 (KCDC)
- Naive way #1
  1. K 항공사가 X를 KCDC에 제공
  2. KCDC는 $I \leftarrow X \cap Y$를 계산하고 I를 K 항공사에 전송
  3. K 항공사는 I에 포함된 승객은 탑승거부
- Naive way #2
  1. K 항공사가 KCDC에 Y 제공 요청
  2. KCDC에게 Y 수신 후, $I \leftarrow X \cap Y$를 계산하고 I에 포함된 승객은 탑승거부

☹ What security troubles?

A motivating scenario: S#2

Advertisement security

- Client: $\mathcal{Y}$ 동영상 플랫폼, Server: $\mathcal{Z}$ 마켓
- Scenario
  - $\mathcal{Y}$사는 N명의 user 관련 정보의 집합 $\mathbb{X} = \{x_1, x_2, \ldots, x_N\}$를 가지고 있음. 각 x_i는 user ID δ_i와 해당 user가 $\mathcal{Z}$ 마켓 광고의 모든 노출시간 대비 지속 view 시간 t_i의 쌍 (δ_i, t_i)로 구성
  - $\mathcal{Z}$사는 user 별로 $\mathcal{Y}$사에 제공된 $\mathcal{Z}$의 제품에 대한 광고를 따라 진행된 거래 정보 화일 $\mathbb{Y} = \{y_1, y_2, \ldots, y_M\}$를 가지고 있음. 각 y_j는 user ID σ_j와 거래에 지출한 총금액 a_j의 쌍 (σ_j, a_j)로 구성
- Technical goal:
  - $\mathcal{Z}$사는 자신의 고객 정보 $\mathbb{Y}$를 노출하지 않고 t_i 상위 n명과 a_j 상위 m 명의 관계를 파악하길 원함

A motivating scenario: S#2

A naive solution

- Client: $\mathcal{Y}$ 동영상 플랫폼, Server: $\mathcal{Z}$ 마켓
- A simple way
  - $\mathcal{Y}$사가 $\mathbb{X} = \{x_1, x_2, \ldots, x_N\}$의 집합을 t_i에 따라 정렬
  - $\mathcal{Y}$사는 상위 n개의 x_i로 구성된 집합 $X = \{x_1, \ldots, x_n\}$을 $\mathcal{Z}$ 사에 전송
  - $\mathcal{Z}$사는 자신의 집합 $\mathbb{Y}$를 사용하여 지출 금액별 user의 t_i에 대한 상관관계 도출

☹ What security troubles?

Private set intersection (PSI)

Definition (Private set intersection)

Let C and S be client and server, respectively, and let $X = \{x_1, x_2, \ldots, x_n\}$ be C's private set and $Y = \{y_1, y_2, \ldots, y_m\}$ be S's private set. Then a PSI protocol, $\mathcal{F}_\cap(X, Y)$ has the client C take X as input and S do Y as input and returns to C and S only the intersection $I = X \cap Y$ without revealing any other information such as $x \notin I$ and $y \notin I$. Sometimes, the server may not have the intersection.

$$\mathcal{F}_\cap : (X, Y) \to (X \cap Y, \emptyset)$$

Private set intersection (PSI)

Graduate project goal in 2020

1. **Relational DB, Socket**과 암호 **Library** (예, openssl, gmp, & NTL)을 사용하여
2. Client/Server의 집합 X, Y는 DB에 저장하고
3. 암호 Library를 사용하여 Client/Server가 $\mathcal{F}_{\cap}(X, Y)$를 수행하도록 구현하고
4. 물리적으로 분리된 Client가 Server에게 암호 Library를 사용하여 적절하게 처리된 집합 $\widehat{X}$를 전송하면
5. Server가 자신의 집합 Y를 $\widehat{Y}$로 변형한 후
6. Server가 $I \leftarrow \mathcal{F}_{\cap}(X, Y)$를 계산하고 Client에게 I 전송

§2. Basic Idea

Some simple soutions

- 안전한 Hash 함수 $H(\cdot)$ (예. SHA256)를 사용한 방법
  1. Client가 자신의 Set $X = \{x_1, \ldots, x_n\}$에 H를 적용하여 $H(X) := \{H(x_1), \ldots, H(x_n)\}$를 계산
  2. $H(X)$를 Server에 전송
  3. Server 역시 $H(Y) := \{H(y_1), \ldots, H(y_m)\}$을 계산
  4. Server는 $\mathcal{I} = \{i | \forall i, j : H(x_i) = H(y_j)\}$를 계산하여 Client에 전송
  5. Client는 $I = \{x_i | i \in \mathcal{I}\}$ 계산

Some simple soutions

- $H()$대신 AES와 같은 Block 암호를 사용하는 방법
    1. Client와 Server 사이에 비밀키 κ를 공유
    2. $H()$대신 AES 암호화 함수 $E(\kappa, \cdot)$을 적용하여 $E(\kappa, X) := \{E(\kappa, x_1), \ldots, E(\kappa, x_n)\}$ 계산

    ☹ $\ldots$

Notation

표기	의미
$a \xleftarrow{\$} \mathbb{A}$	집합 $\mathbb{A}$에서 $a \in \mathbb{A}$를 random하게† 선택
$H(\cdot)$	임의의 길이의 입력을 받아 256-bit 길이의 binary string을 출력하는 `SHA256`‡ 함수
$H_1(\cdot)$	임의의 정수를 받아 group $\mathbb{G}$의 원소를 출력하는 함수
$H_\ell(\cdots)$	3개 이상의 Group $\mathbb{G}$의 원소를 입력으로 받아 256-bit 길이의 binary string을 출력하는 함수 (여기서 $\ell = \{2, 3\}$)
X	Client의 집합으로 $X = \{x_1, x_2, \ldots, x_n\}^\star$
Y	Server의 집합으로 $Y = \{y_1, y_2, \ldots, y_m\}$
$a \hookleftarrow b$	$a \leftarrow b \pmod p$
$1 \leq i \leq n$	$x_i \in X$에 대한 index
$1 \leq j \leq m$	$y_j \in Y$에 대한 index

†"Random하게"의 의미는 Random 값을 생성하는 Big Integer Library의 함수를 사용해야함.

‡"`SHA256`"은 표준 SHA 256 함수를 의미함.

$^\star x_i$는 `uint8_t[16]`으로 가정하며 영문자 또는 숫자로 한정함.

A strawman protocol

Algorithm *Setup*(λ)

Input. λ: the bit length of a prime p
Output. (p, g, q)

Choose a safe prime p of λ bits such that $p = 2q + 1$ for a prime q
Find a random generator of a subgroup $\mathbb{G} \subsetneq \mathbb{Z}_p^*$ whose order is q
Send (p, g, q) to the Server
return (p, g, q)

Algorithm $H_1(x)$

Input. x: an integer in $\mathbb{Z}_q$
Output. an integer $h \in \mathbb{G}$

Read the global values (p, g, q) generated by the algorithm *Setup*
Compute $h \leftarrow g^x \pmod{p}$ (or simply, $h \hookleftarrow g^x$)
return h

A strawman protocol

Algorithm *Client*(X) `/* run by the client */`

Input. $X = \{x_1, x_2, \ldots, x_n\}$: Client's private set
Output. The intersection $I = X \cap Y$

Set $I \leftarrow \varnothing$ and compute $A \hookleftarrow \prod_{i=1}^{n} H_1(x_i)$
Compute $B \hookleftarrow A \cdot g^r$ where $r \xleftarrow{\$} \mathbb{Z}_q^*$ /* $\mathbb{Z}_q^* = \{1, 2, \ldots, q-1\}$ */
for $i \leftarrow 1$ to n **do**
    Compute $A_i \leftarrow H_1(x_i)$ and $B_i \hookleftarrow \frac{A}{A_i} = H_1(x_1) \cdots H_1(x_{i-1}) \cdot H_1(x_{i+1}) \cdots H_1(x_n)$
    Compute $\alpha_i \hookleftarrow B_i \cdot g^{r_i}$ where $r_i \xleftarrow{\$} \mathbb{Z}_q^*$
Send $\langle B, (\alpha_1, \alpha_2, \ldots, \alpha_n)\rangle$ to the server
Receive $\langle S, (\beta_1, \beta_2, \ldots, \beta_n)\rangle$ from the server `/* You should wait for a while... */`
for $i \leftarrow 1$ to n **do**
    Compute $C_i \hookleftarrow \beta_i \cdot S^r \cdot S^{-r_i}$
Receive $(U_1, U_2, \ldots, U_m)$ from the server
for $1 \le i \le n$ and $1 \le j \le m$ **do**
    Compute $I \leftarrow I \cup \{x_i\}$ if $C_i = U_j$
return I

A strawman protocol

Algorithm *Server*(Y) `/* run by the server */`

Input. $Y = \{y_1, y_2, \ldots, y_n\}$: Server's private set

Output. $\varnothing$ or The intersection $I = X \cap Y$

1: Set $I \leftarrow \varnothing$

2: Receive $\langle B, (\alpha_1, \alpha_2, \ldots, \alpha_n)\rangle$ from the client `/* You should wait for a while... */`

3: Compute $S \hookleftarrow g^\gamma$ where $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$

4: **for** $i \leftarrow 1$ to n **do**

5: Compute $\beta_i \hookleftarrow (\alpha_i)^\gamma$

6: Send $\langle S, (\beta_1, \beta_2, \ldots, \beta_n)\rangle$ to the client

7: **for** $j \leftarrow 1$ to m **do** `/* Lines 7∼12: optional */`

8: Compute $T_j \leftarrow H_1(y_j)$

9: Compute $U_j \hookleftarrow \left(\frac{B}{T_j}\right)^\gamma$

10: Send $(U_1, \ldots, U_m)$ to the client

11: Receive $(C_1, C_2, \ldots, C_n)$ from the client

12: **for** $1 \le j \le m$ and $1 \le i \le n$ **do**

13: Compute $I \leftarrow I \cup \{y_j\}$ if $U_j = C_i$

14: **return** I

A strawman protocol

Correctness check

- Client's side

$$C_i = \beta_i S^r S^{-r_i} = (\alpha_i)^\gamma (g^\gamma)^r (g^\gamma)^{-r_i}$$
$$= (B_i \cdot g^{r_i})^\gamma \cdot g^{r\gamma} \cdot g^{-r_i\gamma} = \frac{\prod_{i=1}^n H_1(x_i)}{H_1(x_i)} \cdot g^{r_i\gamma} \cdot g^{r\gamma} \cdot g^{-r_i\gamma}$$
$$= \widehat{H_1(x_i)} \cdot g^{r\gamma}$$

- Server's side

$$U_j = \left(\frac{A \cdot g^r}{H_1(y_j)}\right)^\gamma = \left(\frac{\prod_{i=1}^n H_1(x_i) \cdot g^r}{H_1(y_j)}\right)^\gamma$$
$$= \left(\frac{\prod_{i=1}^n H(x_i)}{H_1(y_j)}\right)^\gamma \cdot g^{r\gamma} \underset{\text{if } x_i = y_j}{=} \widehat{H_1(x_i)} \cdot g^{r\gamma}$$

§3. Your available tools

Choice of relational database (RDB)

- 특정 RDB product (예. mysql, firebird, 또는 sequel pro)에는 무관
- Client는 아래의 schema에 따라 table을 만들어 사용

```
create table _PrivateSetX(
    Value           VARCHAR(30) NOT NULL,
    Frequency       INT NOT NULL,
    Amount          DECIMAL(8,2)
);
```

  Server는 `create table _PrivateSetY(...);` 사용
- **PSI** 수행에는 `Value` attribute만 직접 사용하고
- `Frequency`와 `Amount` attribute는 **PSI** 수행 후 이용

Choice of relational database (RDB)

- Query 문을 사용하여 `_PrivateSetX` table에서
- 조건을 만족하는 `Value`의 Subset을 사용
- Example
  1. 특정 조건의 `Value` 검색

```
select  Value
  from  _PrivateSetX
 where  Frequency >= 20
   and  Amount between 10.3 and 209;
```

  2. 검색 결과를 $X = \{x_1, x_2, \ldots, x_n\}$으로 저장하여 사용

Remote communication between client and server

- Client $\mathcal{C}$와 Server $\mathcal{S}$가 양방향으로 통신
  - Client와 Server 모두 Request를 보내기 위한 목적
- 전송되는 모든 데이타는 Payload 형태로 Packetize해서 전송. 예를 들면
  - $X = \{x_1, \ldots, x_n\}$를 전송을 하려하고 모든 $\| x_i \| = 24$ KB 가정

1. $\mathcal{C} \to \mathcal{S}$: $\texttt{REQ:PrivateSetX} \parallel n$
2. $\mathcal{C} \leftarrow \mathcal{S}$: $\texttt{REP:OK}$
3. $\mathcal{C}$: Compute $N = 24 \times n$ and Encode all x_i into binary format $\mathtt{x}_i$. 예. BSON, ASN.1 (BER, PER 등)
4. $\mathcal{C} \to \mathcal{S}$: $\underbrace{N}_{\text{header}} \parallel \underbrace{\mathtt{x}_1 \parallel \cdots \parallel \mathtt{x}_n}_{\text{payload}}$

- Packetization과 Remote Connection 적용

Some background on operations on $\mathbb{Z}_p$

Two most important parameters: p and g

모든 연산은 $\mathbb{Z}_p^*$에서 이루어지며, $\mathbb{Z}_p^*$는 Generator g를 갖는다.

1. $\mathbb{Z}_p^*$: 엄밀하게는 Group $(\mathbb{Z}_p^*, \cdot)$으로 곱셈만 허용
   - $\forall a, b \in \mathbb{Z}_p^* : c \hookleftarrow a \cdot b = b \cdot a \in \mathbb{Z}_p^*$
   - $\forall a, b \in \mathbb{Z}_p^* : d \hookleftarrow a^\ell = \underbrace{a \cdots a}_{\ell \text{ times}} \in \mathbb{Z}_p^*$
   - $\forall a, b \in \mathbb{Z}_p^* : e \hookleftarrow \frac{a}{b} = a \cdot b^{-1} \in \mathbb{Z}_p^* \Leftarrow b$의 Inverse mod p를 찾아서 a와 곱셈한 결과를 e에 저장
2. g가 Generator이므로 $\mathbb{Z}_p^* = \{g^0 = g^{p-1}, g^1, \ldots, g^{p-2}\}$ 여기서 Fermat's Little Theorem에 의해 $g^0 = 1 \hookleftarrow g^{p-1}$

Some background on operations on $\mathbb{Z}_p$

A large and special prime p

- p: a safe prime number of 1024 (or 2048) bits
  1. Big Integer Library를 사용하여 1023 (or 2047) Bits 길이의 Random Prime q 생성
  2. $p = 2 \cdot q + 1$ 계산
  3. Big Integer Library를 사용하여 p가 Prime이면 저장하고 아니면 `goto` Step 1.
- p가 결정되면 $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$가 결정
- "0"은 사용하지 않음 $\Rightarrow$ 실제로는 $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ 사용
- Algorithm에서 $b \hookleftarrow a$:
  - ☹ $a \not\leftarrow b$
  - ☺ $b \leftarrow a \mod p \Rightarrow$ 연산결과는 항상 $b \in \mathbb{Z}_p^*$

Some background on operations on $\mathbb{Z}_p$

- $\mathbb{Z}_p^*$의 Generator 찾기
- $\mathbb{Z}_p^*$의 order $= |\mathbb{Z}_p| = p - 1$이며 $p - 1 = 2q$
- Lagrange Theorem에 의해 $\mathbb{Z}_p^*$는 $\{1, 2, q, 2q\}$인 Subgroup을 4개 소유

Order $= 2q$인 Subgroup $= \mathbb{Z}_p^* \Rightarrow \mathbb{Z}_p^*$를 생성하는 g 찾기

1. Choose an element $1 \neq g \in \mathbb{Z}_p^*$ randomly
2. Check if

$$1 \overset{?}{\neq} g^2 \pmod{p} \text{ and}$$
$$1 \overset{?}{\neq} g^q \pmod{p} \text{ but}$$
$$1 \overset{?}{=} g^{2q} \pmod{p}$$

3. If so, then fix g as a generator; otherwise `goto` Step 1

Some background on operations on $\mathbb{Z}_p$

- $\boxed{\text{Order} = q}$인 Subgroup $\mathbb{G} \subsetneq \mathbb{Z}_p^*$를 생성하는 g 찾기
  1. Choose an element $1 \neq g \in \mathbb{Z}_p^*$ randomly
  2. Check if
     $$1 \overset{?}{\neq} g^2 \pmod{p} \text{ but}$$
     $$1 \overset{?}{=} g^q \pmod{p} \text{ and}$$
     $$1 \overset{?}{=} g^{2q} \pmod{p}$$
  3. If so, then fix g as a generator; otherwise `goto` Step 1

Crypto-library

- 방법 #1
  - openssl 사용
    - Big Integer 연산과 AES, SHA를 모두 사용 가능

```
#include <openssl/sha.h>
#include <openssl/aes.h>
#include <openssl/evp.h>
#include <openssl/bn.h>        /* big number */
...
```

- 방법 #2
  - gmp나 NTL와 같은 Big Integer 연산 전용 Library 사용
  - 다른 Big Integer 연산 지원하는 Library들
    - miracl and flint 등등
  - AES/SHA: Open Source 이용

Phase I

The missions

1. Fix your tools: How to store?, how to communicate?, and How to make secure?
2. Implement two basic algorithms $Client(X)$ and $Server(Y)$

☺ For simplicity, only the client gets the intersection of X and Y

기대 일정

- 기한: 오늘 $\sim$ 10월 31일
- Phase I 구현 내용 이해 및 필수 Library 숙지: $\geq$ 9월 30일

일정 계획: 각자 설계

One critical security concern

Phase I의 PSI는 단순한 공격에 취약

- 임의의 공격자는 Client와 Server의 통신 내용 탈취 가능
    - Client가 전송하는 $\langle B, (\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_n)\rangle$ 고려
    - 여기서 $B = A \cdot g^r$이며 $\alpha_i = B_i \cdot g^{r_i}$
- 공격자는 α_i를 다음과 같이 α_i^*로 대체 가능
    - Choose x_i^* in the universe set and compute $A_i^* \leftarrow H(x_i^*)$
    - Compute $\alpha_i^* \hookleftarrow A_i^* \cdot \alpha_i = \frac{A \cdot A_i^*}{A_i} \cdot g^{r_i}$
    - Replace $\langle B, (\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n)\rangle$ by $\langle B, (\alpha_1, \ldots, \alpha_i^*, \ldots, \alpha_n)\rangle$
- α_i가 Virus-ζ 확진자의 전화번호 x_i에서 계산된 값인 경우, K 항공에 답승한 모든 승객 감염

A fix to this concern

Idea

Client가 모든 α_i를 실제로 계산해서 만들었음을 증명하도록 강제

- Idea의 실현 방법
  1. $\alpha_i = B_i \cdot g^{r_i}$, ($B_i = \frac{A}{A_i}$ 이고 g는 공개값이며 $r_i \in \mathbb{Z}_q^*$인 비밀값)
  2. $A = \prod_{i=1}^{n} A_i$, ($A_i = H_1(x_i)$ 이고 x_i는 비밀값)
  3. Set X가 고정되면 B_i는 고정값 $\Rightarrow$ r_i: 고정값이 매번 바뀌게 하는 값
  4. B_i는 고정된 값으로 재사용 불가하며 g는 공개값
  5. α_i 계산의 핵심 $\Leftrightarrow$ Client가 r_i를 알고 있는냐

A fix to this concern

Idea

Client가 모든 α_i를 실제로 계산해서 만들었음을 증명하도록 강제

- r_i를 α_i의 성실한 계산의 증거로 Server에게 제시
  - ☹ No, No
- r_i가 드러나면
  - $B_i \leftarrow \alpha_i \cdot g^{-r_i} = H_1(x_1) \cdots H_1(x_{i-1}) \cdot H_1(x_{i+1}) \cdots H_1(x_n)$
  - Client 소유로 예상되는 원소 $\hat{x}$를 선택하여 $\widehat{A_i} = H_1(\hat{x})$ 계산한 후
  - Division test를 반복적으로 수행

☞ "r_i"를 Server에게 알려주지 않고, α_i의 계산에 쓰인 r_i를 알고 있다는 것을 증명

§4. Zero-knowledge Proofs (ZKPs)

A concept of ZKPs

Goal: $\boxed{y = g^x \mod p}$를 만족하는 x를 알고 있음을 증명

- Assumptions:
  - (p, g, y)는 모두 공개값
  - (p, g, y)을 공개한 경우에도 현재의 Computing Power로는 x 찾는 멋진 Algorithm 부재
    $\Rightarrow$ Discrete Logarithm Problem Assumption
  - x를 보여주지 않고 $y \leftarrow g^{\faHandRock} \pmod p$의 계산에 사용된 $\faHandRock = x$일 수 밖에 없다는 것을 스스로 인정
- The **Schnorr** technique to solve this problem

A conceptual sketch of Schnorr's approach

설정

1. Prover & Verifier: (p, g, q)를 모두 알고 있음
2. Prover: 비밀값 $x \in \mathbb{Z}_q$은 안전하게 보관; $\underbrace{y \leftarrow g^x \pmod{p}}_{(\star)}$는 공개
3. Prover는 $\boxed{y\text{가 자기만 아는 } x\text{를 이용해 } (\star) \text{ 계산했다}}$를 설득

Prover $\xrightarrow{(p, g, q) \text{ and } y}$ Verifier

1. Choose a random $r \in \mathbb{Z}_q$
2. Compute $\beta \leftarrow g^r \pmod{p}$ $\xrightarrow{\beta}$

$\xleftarrow{e}$ 3. Choose a random $e \in \mathbb{Z}_q$

4. Compute $z \leftarrow r - e \cdot x \pmod{q}$ $\xrightarrow{z}$

5. Check if $\beta \stackrel{?}{=} g^z y^e$
6. If equal, say "ok"; else say "no"

A conceptual sketch of Schnorr's approach

Schnorr 기법의 단점

- Prover와 Verifier가 총 3-Round의 통신을 수행
- 통신 도중에 문제 발생 $\Rightarrow$ 정상 진행 불가

A Solution: Prover가 해시함수 H_2를 이용하여 e를 혼자 계산

Prover $\overset{(p, g, q) \text{ and } y}{\bullet\!\!-\!\!-\!\!-\!\!-\!\!\bullet}$ Verifier

1. Choose a random $r \in \mathbb{Z}_q$
2. Compute $\beta \leftarrow g^r \pmod{p}$
3. Compute $e \leftarrow H_2(p, y, \beta)$
4. Compute $z \leftarrow r - e \cdot x \pmod{q}$ $\xrightarrow{(\beta, z)}$
5. Compute $e \leftarrow H_2(p, y, \beta)$
6. Check if $\beta \stackrel{?}{=} g^z y^e$
7. If equal, say "ok"; else say "no"

The Schnorr algorithm

Algorithm *SchnorrProver*(p, g, q, x, y)

Input. 공개값 (p, g, q, y)와 비밀값 x such that $y = g^x \pmod{p}$
Output. 증거값 $\pi := (\beta, z)$

1: Choose a random $r \in \mathbb{Z}_q$
2: Compute $\beta \hookleftarrow g^r$
3: Compute $e \leftarrow H_2(p, y, \beta) = H(p \parallel y \parallel \beta)$ /* $\parallel$: concatenation */
4: Compute $z \leftarrow r - e \cdot x \pmod{q}$
5: Send $\pi = (\beta, z)$ to the verifier

Algorithm *SchnorrVerifier*(p, g, q, y)

Input. 공개값 (p, g, q, y)
Output. $b = \{0, 1\}$

1: Receive the proof $\pi = (\beta, z)$ and parse π into β and z
2: Compute $e \leftarrow H_2(p, y, \beta) = H(p \parallel y \parallel \beta)$
3: Compute $v \hookleftarrow g^z \cdot y^e$
4: **if** $\beta = v$ **then** **return** 1 /* The proof is good */
5: **else** **return** 0 /* ☹, go home! man */

Goal: to prove that $y_0 \leftarrow g_0^x \pmod{p}$ and $y_1 \leftarrow g_1^x \pmod{p}$

설정

1. Prover & Verifier: (p, g_0, g_1, q)를 모두 알고 있음
2. Prover: 비밀값 $x \in \mathbb{Z}_q$은 안전하게 보관; $y_0 \leftarrow g_0^x, y_1 \leftarrow g_1^x$는 공개

Prover $\overset{(p, g_0, g_1, q) \text{ and } y_0, y_1}{\bullet\!\!\longrightarrow\!\!\bullet}$ **Verifier**

1. Choose a random $r \in \mathbb{Z}_q$
2. Compute $\beta_0 \leftarrow g_0^r \pmod{p}$ &
   $\beta_1 \leftarrow g_1^r \pmod{p}$
3. Compute $e \leftarrow H_3(p, y_0, y_1, \beta_0, \beta_1)$
4. Compute $z \leftarrow r - e \cdot x \pmod{q}$ $\xrightarrow{(\beta_0, \beta_1, z)}$

5. Compute $e \leftarrow H_3(p, y_0, y_1, \beta_0, \beta_1)$
6. Check if $\beta_0 \stackrel{?}{=} g_0^z y_0^e$ & $\beta_1 \stackrel{?}{=} g_1^z y_1^e$
7. If equal, say “ok”; else say “no”

Goal: to prove that $y_0 \leftarrow g_0^x \pmod p$ and $y_1 \leftarrow g_1^x \pmod p$

Algorithm *EqualProver*$(p, g_0, g_1, q, x, y_0, y_1)$

Input. 공개값 (p, g, q, y_0, y_1)와 비밀값 x such that $y_0 = g_0^x \pmod p$ and $y_1 = g_1^x \pmod p$
Output. 증거값 $\pi_{eq} := (\beta_0, \beta_1, z)$

1: Choose a random $r \in \mathbb{Z}_q$
2: Compute $\beta_0 \hookleftarrow g_0^r$ and $\beta_1 \hookleftarrow g_1^r$
3: Compute $e \leftarrow H_3(p, y_0, y_1, \beta_0, \beta_1) = H(p \parallel y_0 \parallel y_1 \parallel \beta_0 \parallel \beta_1)$
4: Compute $z \leftarrow r - e \cdot x \pmod q$
5: Send $\pi_{eq} = (\beta_0, \beta_1, z)$ to the verifier

Goal: to prove that $y_0 \leftarrow g_0^x \pmod{p}$ and $y_1 \leftarrow g_1^x \pmod{p}$

Algorithm *EqualVerifier*$(p, g_0, g_1, q, y_0, y_1)$

Input. 공개값 $(p, g_0, g_1, q, y_0, y_1)$

Output. $b = \{0, 1\}$

Receive the proof π_{eq} and parse into (β_0, β_1, z)

Compute $e \leftarrow H_3(p, y_0, y_1, \beta_0, \beta_1) = H(p \parallel y_0 \parallel y_1 \parallel \beta_0 \parallel \beta_1)$

Compute $v_0 \hookleftarrow g_0^z \cdot y_0^e$ and $v_1 \hookleftarrow g_1^z \cdot y_1^e$

if $\beta_0 = v_0 \wedge \beta_1 = v_1$ **then**

    return 1

else

    return 0

Goal: to prove that $y \leftarrow g_0^{x_0} g_1^{x_1} \pmod{p}$

설정

1. Prover & Verifier: (p, g_0, g_1, q)를 모두 알고 있음
2. Prover: 비밀값 $x_0, x_1 \in \mathbb{Z}_q$은 안전하게 보관; $y \hookleftarrow g_0^{x_0} \cdot g_1^{x_1}$는 공개

$$\text{Prover} \xrightarrow{(p, g_0, g_1, q) \text{ and } y} \text{Verifier}$$

1. Choose a random $r_0, r_1 \in \mathbb{Z}_q$
2. Compute $\beta \leftarrow g_0^{r_0} \cdot g_1^{r_1} \pmod{p}$ &
3. Compute $e \leftarrow H_4(p, y_0, y_1, \beta)$
4. Compute $z_0 \leftarrow r_0 - e \cdot x_0 \pmod{q}$ &

$$z_1 \leftarrow r_1 - e \cdot x_1 \pmod{q} \xrightarrow{(\beta, z_0, z_1)}$$

5. Compute $e \leftarrow H_4(p, y_0, y_1, \beta)$
6. Check if $\beta \stackrel{?}{=} g_0^{z_0} \cdot g_1^{z_1} \cdot y^e$
7. If equal, say “ok”; else say “no”

Goal: to prove that $y \leftarrow g_0^{x_0} g_1^{x_1} \pmod{p}$

Algorithm $\mathit{TwoProver}(p, g_0, g_1, q, x_0, x_1, y)$

Input. 공개값 (p, g, q, y)와 비밀값 x_0, x_1 such that $y = g_0^{x_0} g_1^{x_1} \pmod{p}$

Output. 증거값 $\pi_2 = (\beta, z_0, z_1)$

```
Choose a random r_0, r_1 ∈ Z_q
Compute β ↩ g_0^{r_0} · g_1^{r_1}
Compute e ← H_2(p, y, β) = H(p || y || β)
Compute z_0 ← r_0 − e · x_0 (mod q)
Compute z_1 ← r_1 − e · x_1 (mod q)
Send π_2 = (β, z_0, z_1) to the verifier
```

Choose a random $r_0, r_1 \in \mathbb{Z}_q$

Compute $\beta \hookleftarrow g_0^{r_0} \cdot g_1^{r_1}$

Compute $e \leftarrow H_2(p, y, \beta) = H(p \parallel y \parallel \beta)$

Compute $z_0 \leftarrow r_0 - e \cdot x_0 \pmod{q}$

Compute $z_1 \leftarrow r_1 - e \cdot x_1 \pmod{q}$

Send $\pi_2 = (\beta, z_0, z_1)$ to the verifier

Goal: to prove that $w \leftarrow g_0^{x_0} g_1^{x_1} \pmod{p}$

Algorithm *TwoVerifier*(p, g_0, g_1, q, y)

Input. 공개값 (p, g_0, g_1, q, y)
Output. $b \in \{0, 1\}$

```
Receive the proof π_2 and parse into (β, z_0, z_1)
Compute e ← H_2(p, y, β) = H(p ‖ y ‖ β)
Compute v ⟵ g_0^{z_0} · g_1^{z_1} · y^e
if β = v then
    return 1
else
    return 0
```

§5. The Full-fledged PSI Protocol

The full-fledged protocol

Algorithm *Setup*(λ)

Input. λ: the bit length of a prime p

Output. (p, g_0, g_1, g_2, q)

1: Choose a safe prime p of λ bits such that $p = 2q + 1$ for a prime q
2: Find random generators (g_0, g_1, g_2) of a subgroup $\mathbb{G} \subsetneq \mathbb{Z}_p^*$ whose order is q
3: Send (p, g_0, g_1, g_2, q) to the Server
4: **return** (p, g_0, g_1, g_2, q)

The full-fledged protocol

- Hash functions
  1. $h \leftarrow H_1(a)$: A number-theoretic hash function defined by

$$H_1(a) := g_0^a \pmod p$$
$$\hookleftarrow g_0^a$$

     where $a \in \mathbb{Z}_q$ and $h \in \mathbb{G}$
  2. $h \leftarrow H(a)$: The original SHA256 function
     - $H_2(a, b, c) := H(a \parallel b \parallel c)$
     - $H_3(a, b, c, d) := H(a \parallel b \parallel c \parallel d)$
       where a, b, c, d, e are binary strings of arbitrary length and h is an 256-bit binary string
     - 연접연산 $a \parallel b$: a, b가 binary string일때, (ex. `uint_8 a[16];` `uint_8 b[16];`) $c \leftarrow a \parallel b$의 결과는 a와 b를 모두 포함하는 1개의 binary string에 저장한다는 의미. (ex. `uint_8 c[32];`)

The full-fledged protocol

Algorithm *ZKPClient*(X)

Input. $X = \{x_1, x_2, \ldots, x_n\}$: Client's private set

Output. The intersection $I = X \cap Y$

Set $I \leftarrow \varnothing$ and compute $A \hookleftarrow \prod_{i=1}^{n} H_1(x_i)$

Compute $B \hookleftarrow A \cdot g_0{}^r$ where $r \xleftarrow{\$} \mathbb{Z}_q^*$ /* $\mathbb{Z}_q^* = \{1, 2, \ldots, q-1\}$ */

for $i \leftarrow 1$ to n **do**

    Compute $A_i \leftarrow H_1(x_i)$ and $B_i \hookleftarrow \frac{A}{A_i} = H_1(x_1) \cdots H_1(x_{i-1}) \cdot H_1(x_{i+1}) \cdots H_1(x_n)$

    Compute $\alpha_i \hookleftarrow A_i \cdot g_1{}^{r_i}$ and $\sigma_i \hookleftarrow B_i \cdot g_2^{r_i}$ where $r_i \xleftarrow{\$} \mathbb{Z}_q^*$

Compute $\pi_c \leftarrow TwoProver\left(r, r_1, \ldots, r_n \,\middle|\, \frac{B}{\alpha_1\sigma_1} = \frac{g_0^r}{(g_1g_2)^{r_1}} \wedge \cdots \wedge \frac{B}{\alpha_n\sigma_n} = \frac{g_0^r}{(g_1g_2)^{r_n}}\right)$

Send $\langle B, (\alpha_1, , \ldots, \alpha_n), (\sigma_1, \ldots, \sigma_n), \pi_c \rangle$ to the server

Receive $\langle S, (\beta_1, \beta_2, \ldots, \beta_n), (U_1, \ldots, U_m), \pi_s \rangle$ from the server

Compute $b \leftarrow EqualVerifer(\pi_s; \{\beta_i\}, \{\alpha_i\})$

if $b \neq \texttt{accept}$ **then** abort

for $i \leftarrow 1$ to n **do**

    Compute $\kappa_i \hookleftarrow \beta_i \cdot S^{-r_i}$ and $C_i \leftarrow H_2(\kappa_i, A_i, x_i)$

for $1 \leq i \leq n$ and $1 \leq j \leq m$ **do**

    Compute $I \leftarrow I \cup \{x_i\}$ if $C_i = U_j$

return I

The full-fledged protocol

- Line 6.을 구현할때는 $i = 1$에 대해서만 구현

6.1 Compute $y \hookleftarrow B \cdot \alpha_1^{-1} \cdot \sigma_1^{-1}$

6.2 Compute $h \hookleftarrow g_1 \cdot g_2$ /* because $\frac{g_0^r}{(g_1 g_2)^{r_1}} = g_0^r \cdot h^{-r_1}$ */

6.2 Run $\pi_2 \leftarrow TwoProver(p, g_0, h, q, r, -r_1, y)$

6.2 Copy π_2 to π_c

- Line 9.을 구현할때도 $i = 1$에 대해서만 구현

9. Run $b \leftarrow EqualVerifier(p, g_1, \alpha_1, q, S, \beta_1)$

The full-fledged protocol

Algorithm *ZKPServer*(Y)

Input. $Y = \{y_1, y_2, \ldots, y_n\}$: Server's private set

Output. $\varnothing$

Set $I \leftarrow \varnothing$
Receive $\langle B, (\alpha_1, , \ldots, \alpha_n), (\sigma_1, \ldots, \sigma_n), \pi_c \rangle$ from the client
Compute $b \leftarrow \mathit{TwoVerifier}(\pi_c; B, \{\alpha_i\}, \{\sigma_i\})$
if $b \neq$ `accept` **then** abort
Compute $S \hookleftarrow g_1^\gamma$ where $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$
Compute $\pi_s \leftarrow \mathit{EqualProver}(\gamma | S = g_1^\gamma \wedge \beta_1 = \alpha_1^\gamma \wedge \cdots \wedge \beta_n = \alpha_n^\gamma)$
for $i \leftarrow 1$ to n **do**
    Compute $\beta_i \hookleftarrow (\alpha_i)^\gamma$
for $j \leftarrow 1$ to m **do**
    Compute $S_j \leftarrow H_1(y_j)$ and $\kappa_j \hookleftarrow S_j^\gamma$
    Compute $U_j \leftarrow H_2(\kappa_j, S_j, y_j)$
Send $\langle S, (\beta_1, \beta_2, \ldots, \beta_n), (U_1, \ldots, U_m), \pi_s \rangle$ to the client
return I

The full-fledged protocol

- Line 6.을 구현할때 $i = 1$에 대해서만 구현

  6.2 Run $\pi_{eq} \leftarrow EqualProver(p, g_1, \alpha_1, q, \gamma, S, \beta_1)$

  6.2 Copy π_{eq} to π_s

- Line 3.을 구현할때도 $i = 1$에 대해서만 구현

  6.1 Compute $y \hookleftarrow B \cdot \alpha_1^{-1} \cdot \sigma_1^{-1}$

  6.2 Compute $h \hookleftarrow g_1 \cdot g_2$

  6.2 Run $b \leftarrow TwoVerifier(p, g_0, h, q, y)$

The full-fledged protocol

Correctness check: π_c and π_s

- Client's side
  - Client가 사용한 모든 Random, $(r, r_1, r_2, \ldots, r_n)$에 대한 확인
  - π_c에서 요구하는 조건: $\frac{B}{\alpha_i \cdot \sigma_i} = \frac{g_0^r}{(g_1 \cdot g_2)_i^r}$
  - $B = A \cdot g_0^r$ and $\alpha_i = A_i \cdot g_1^{r_i}$ and $\sigma_i = B_i \cdot g_2^{r_i}$

$$\frac{B}{\alpha_i \cdot \sigma_i} = \frac{A \cdot g_0^r}{A_i \cdot g_1^{r_i} \cdot B_i \cdot g_2^{r_i}} = \frac{A \cdot g_0^r}{A_i \cdot B_i \cdot (g_1 \cdot g_2)^{r_i}}$$
$$= \underbrace{\frac{g_0^r}{(g_1 g_2)^{r_i}}}_{\because B_i = \frac{A}{A_i}}$$

- Server's side
  - π_s에서 요구하는 조건: $S = g_1^\gamma$에 사용된 γ만 확인

The full-fledged protocol

Correctness check: The intersection

- Client's side

$$\kappa_i = \beta_i \cdot S^{r_i} = (\alpha_i)^\gamma \cdot (g_1)^{-\gamma r_i} = \left(A_i g_1^{r_i}\right)^\gamma g_1^{-\gamma r_i} = A_i^\gamma \cdot g_1^{\gamma r_i} \cdot g_1^{-\gamma r_i}$$
$$= A_i^\gamma = (H_1(x_i))^\gamma$$

- Server's side

$$\kappa_j = (S_j)^\gamma = \left(H_1(y_j)\right)^\gamma$$

- Client & Server

$$\begin{aligned} x_i = y_j &\Rightarrow A_i = S_j \\ &\Rightarrow \kappa_i = (H_1(x_i))^\gamma = \kappa_j = \left(H_1(y_j)\right)^\gamma \\ &\Rightarrow H_2(\kappa_i, A_i, x_i) = H_2(\kappa_j, S_j, y_j) \end{aligned}$$

Phase II

The final missions

1. Implement the two algorithms $\mathit{ZKPClient}(X)$ and $\mathit{ZKPServer}(Y)$

☺ For simplicity, only the client gets the intersection of X and Y

2. π_c와 π_s에 대해서 각각 $i = 1$과 $j = 1$에 대한 π_c와 π_s만 추가하여 전송
   - π_c는 $\frac{B}{\alpha_1 \cdot \sigma_1} = \frac{g_0^r}{(g_1 g_2)^{r_1}}$을 보장하는 증거만 생성
   - π_s는 $S = g_1^\gamma \wedge \beta_1 = \alpha_1^\gamma$을 보장하는 증거만 생성

기대 일정

- 기한: 11월 1일 $\sim$ 11월 30일

😄 일정 계획: 12월 7일부터 일정 조정후 Demo

Thank you &

Questions?