Practical Private Set Intersection Protocols with Linear Complexity

Concepts

PSI

PSI consists of two algorithms: {Setup, Interaction}

- **Setup**: a process wherein all global/public parameters are selected.
- **Interaction**: a protocol be tween client and server that results in the client obtaining the intersection of two sets.

APSI

APSI is a tuple of three algorithms: {Setup, Authorize, Interaction}.

- **Setup**: a process wherein all global/public parameters are selected.
- **Authorize** : a protocol between client and CA resulting in client committing to its input set and CA issuing authorizations (signatures), one for each element of the set.
- **Interaction**: a protocol between client and server that results in the client obtaining the intersection of two sets.

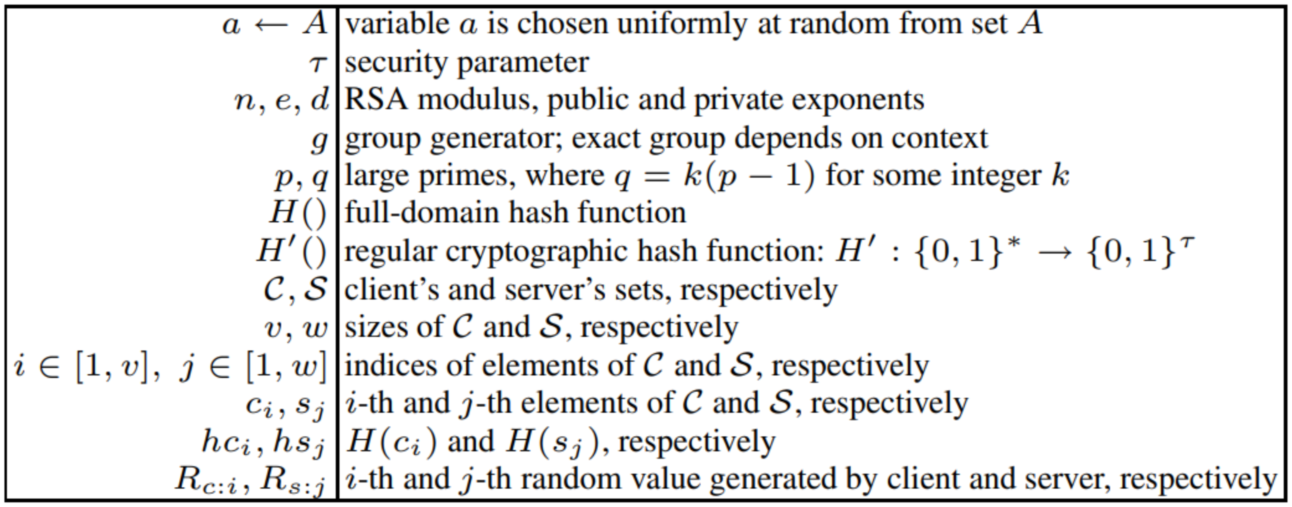
Security Properties

- **Correctness**: A PSI scheme is correct if, at the end of Interaction, client outputs the exact (possibly empty) intersection of the two respective sets.
- **Server Privacy**: Informally, a PSI scheme is server-private if the client learns no information (except the upper bound on size) about the subset of elements on the server that are NOT in the intersection of their respective sets.
- **Client Privacy**: Informally, client privacy (in either PSI or APSI) means that no information is leaked about client's set elements to a malicious server, except the upper bound on the client's set size.
- **Client Unlinkability** (optional): Informally, client unlinkability means that a malicious server cannot tell if any two instances of Interaction are related, i.e., executed on the same inputs by the client.
- **Server Unlinkability** (optional): Informally, server unlinkability means that a malicious client cannot tell if any two instances of Interaction

are related, i.e., executed on the same inputs by the server.

- **Correctness** (APSI): An APSI scheme is correct if, at the end of Interaction, client outputs the exact (possibly empty) intersection of the two respective sets and each element in that intersection has been previously authorized by CA via Authorize.
- **Server Privacy** (APSI): Informally, an APSI scheme is server-private if the client learns no information (except the upper bound on size) about the subset of elements on the server that are NOT in the intersection of their respective sets (where client's set contains only authorizations obtained via Authorize).

Notation



$a \leftarrow A$	variable a is chosen uniformly at random from set A
τ	security parameter
n, e, d	RSA modulus, public and private exponents
g	group generator; exact group depends on context
p, q	large primes, where $q = k(p-1)$ for some integer k
$H()$	full-domain hash function
$H'()$	regular cryptographic hash function: $H' : \{0,1\}^* \rightarrow \{0,1\}^\tau$
$\mathcal{C}, \mathcal{S}$	client's and server's sets, respectively
v, w	sizes of $\mathcal{C}$ and $\mathcal{S}$, respectively
$i \in [1, v],\ j \in [1, w]$	indices of elements of $\mathcal{C}$ and $\mathcal{S}$, respectively
c_i, s_j	i-th and j-th elements of $\mathcal{C}$ and $\mathcal{S}$, respectively
hc_i, hs_j	$H(c_i)$ and $H(s_j)$, respectively
$R_{c:i}, R_{s:j}$	i-th and j-th random value generated by client and server, respectively

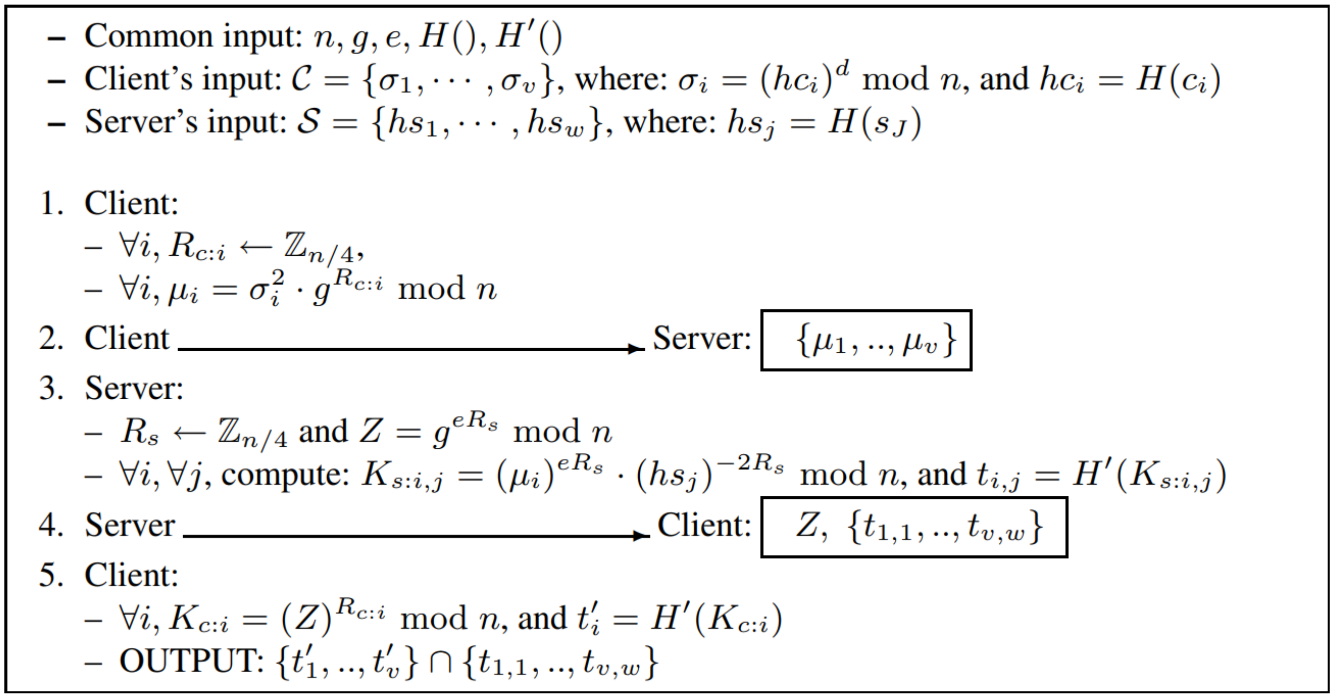
Methods

Baseline: APSI from RSA-PPIT

大致过程：

1. 对于Client端的每个元素，Client计算一个μ，发送给Server
2. Server对于每个μ和Server的每个元素，计算一个t；另外，Server还生成一个数Z，这个数与用于计算t的随机数有关。最后把Z和t都发给Client
3. Client利用Z，对自己的每一个元素计算t'，比较t'和t的交集

具体过程：



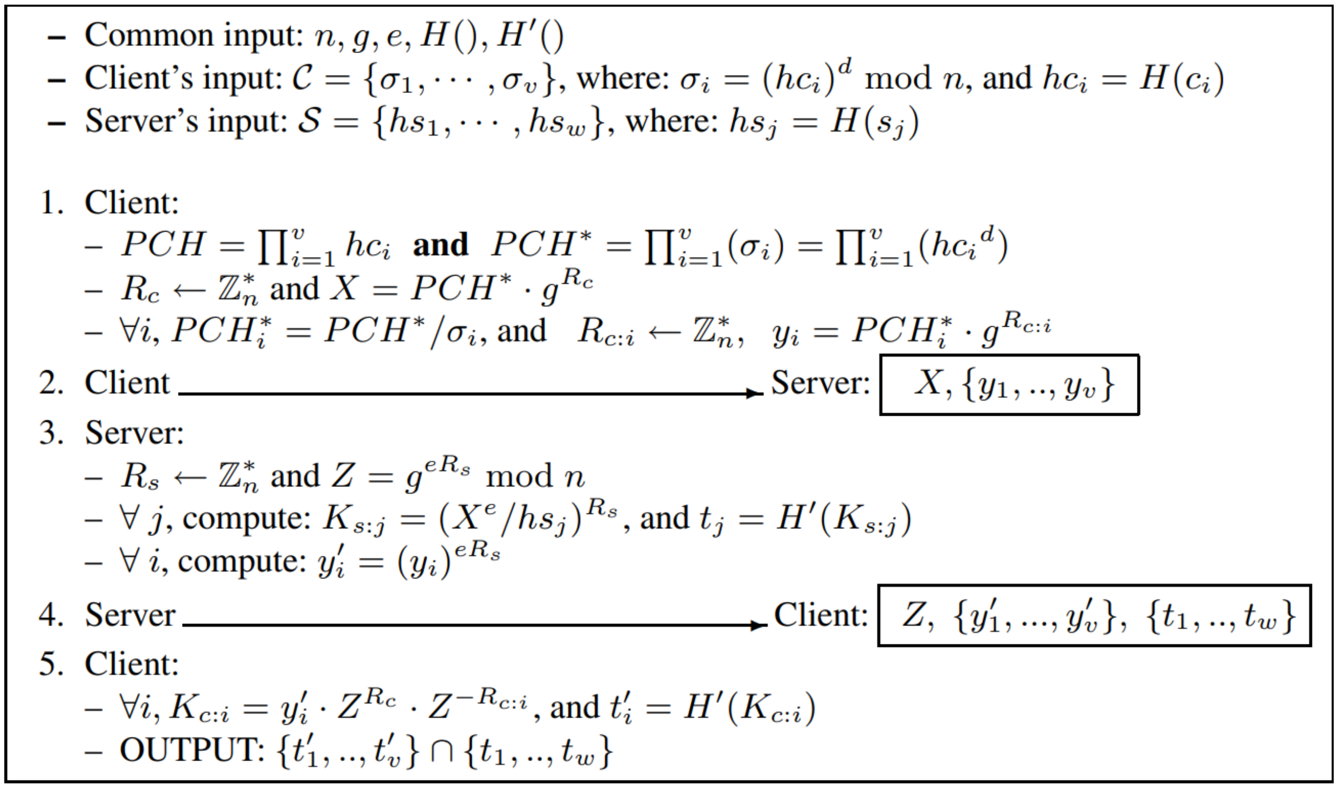
- Common input: $n, g, e, H(), H'()$
- Client's input: $\mathcal{C} = \{\sigma_1, \cdots, \sigma_v\}$, where: $\sigma_i = (hc_i)^d \bmod n$, and $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_J)$

1. Client:
   - $\forall i, R_{c:i} \leftarrow \mathbb{Z}_{n/4}$,
   - $\forall i, \mu_i = \sigma_i^2 \cdot g^{R_{c:i}} \bmod n$
2. Client $\xrightarrow{\hspace{8em}}$ Server: $\boxed{\{\mu_1, .., \mu_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_{n/4}$ and $Z = g^{eR_s} \bmod n$
   - $\forall i, \forall j$, compute: $K_{s:i,j} = (\mu_i)^{eR_s} \cdot (hs_j)^{-2R_s} \bmod n$, and $t_{i,j} = H'(K_{s:i,j})$
4. Server $\xrightarrow{\hspace{8em}}$ Client: $\boxed{Z,\ \{t_{1,1}, .., t_{v,w}\}}$
5. Client:
   - $\forall i, K_{c:i} = (Z)^{R_{c:i}} \bmod n$, and $t'_i = H'(K_{c:i})$
   - OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_{1,1}, .., t_{v,w}\}$

APSI with Linear Costs

This protocol incurs **linear** computation (for both parties) and communication complexity.

具体过程：



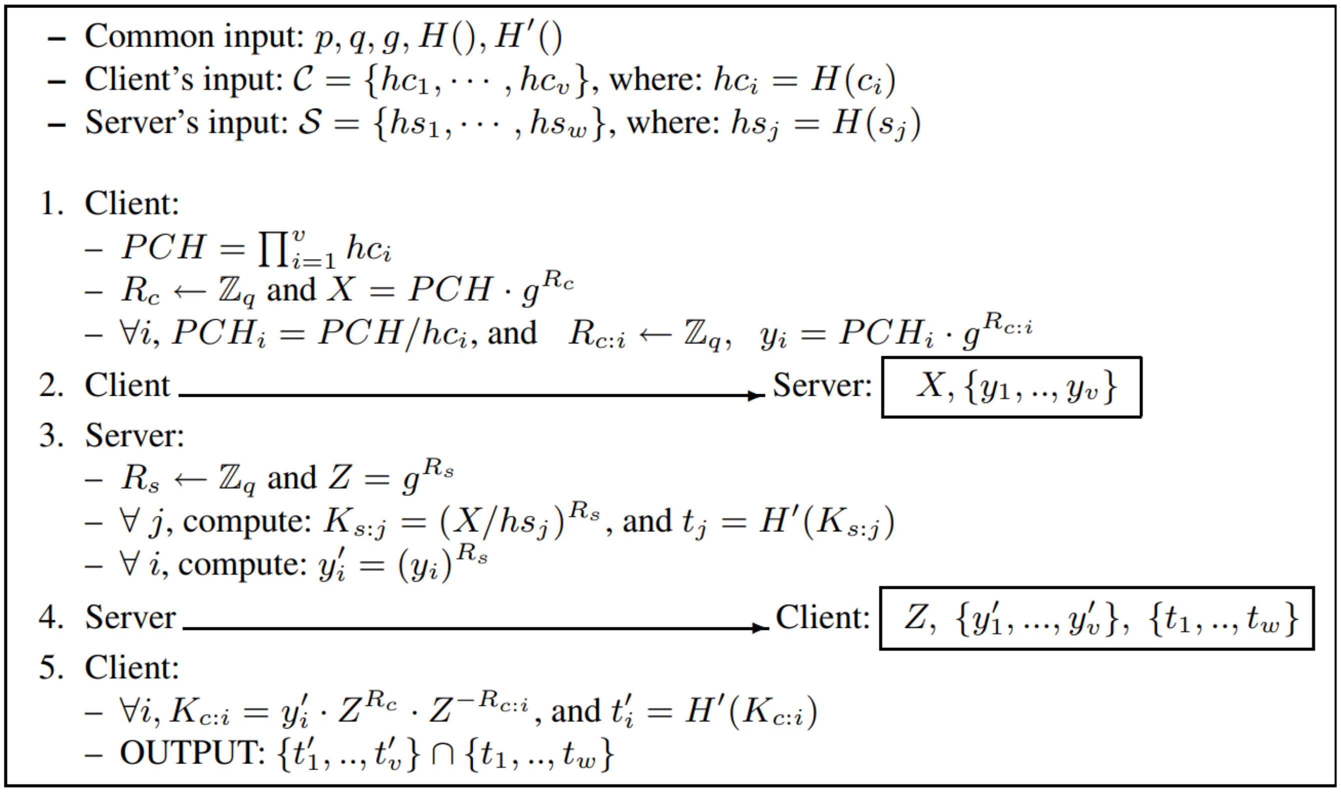
- Common input: $n, g, e, H(), H'()$
- Client's input: $\mathcal{C} = \{\sigma_1, \cdots, \sigma_v\}$, where: $\sigma_i = (hc_i)^d \bmod n$, and $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_j)$

1. Client:
   - $PCH = \prod_{i=1}^{v} hc_i$ **and** $PCH^* = \prod_{i=1}^{v}(\sigma_i) = \prod_{i=1}^{v}(hc_i{}^d)$
   - $R_c \leftarrow \mathbb{Z}_n^*$ and $X = PCH^* \cdot g^{R_c}$
   - $\forall i, PCH_i^* = PCH^*/\sigma_i$, and $R_{c:i} \leftarrow \mathbb{Z}_n^*$, $y_i = PCH_i^* \cdot g^{R_{c:i}}$
2. Client $\xrightarrow{\hspace{8em}}$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_n^*$ and $Z = g^{eR_s} \bmod n$
   - $\forall\ j$, compute: $K_{s:j} = (X^e/hs_j)^{R_s}$, and $t_j = H'(K_{s:j})$
   - $\forall\ i$, compute: $y'_i = (y_i)^{eR_s}$
4. Server $\xrightarrow{\hspace{8em}}$ Client: $\boxed{Z,\ \{y'_1, ..., y'_v\},\ \{t_1, .., t_w\}}$
5. Client:
   - $\forall i, K_{c:i} = y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$, and $t'_i = H'(K_{c:i})$
   - OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

Deriving Efficient PSI

从上一个方法获得对应的PSI版本

具体过程：



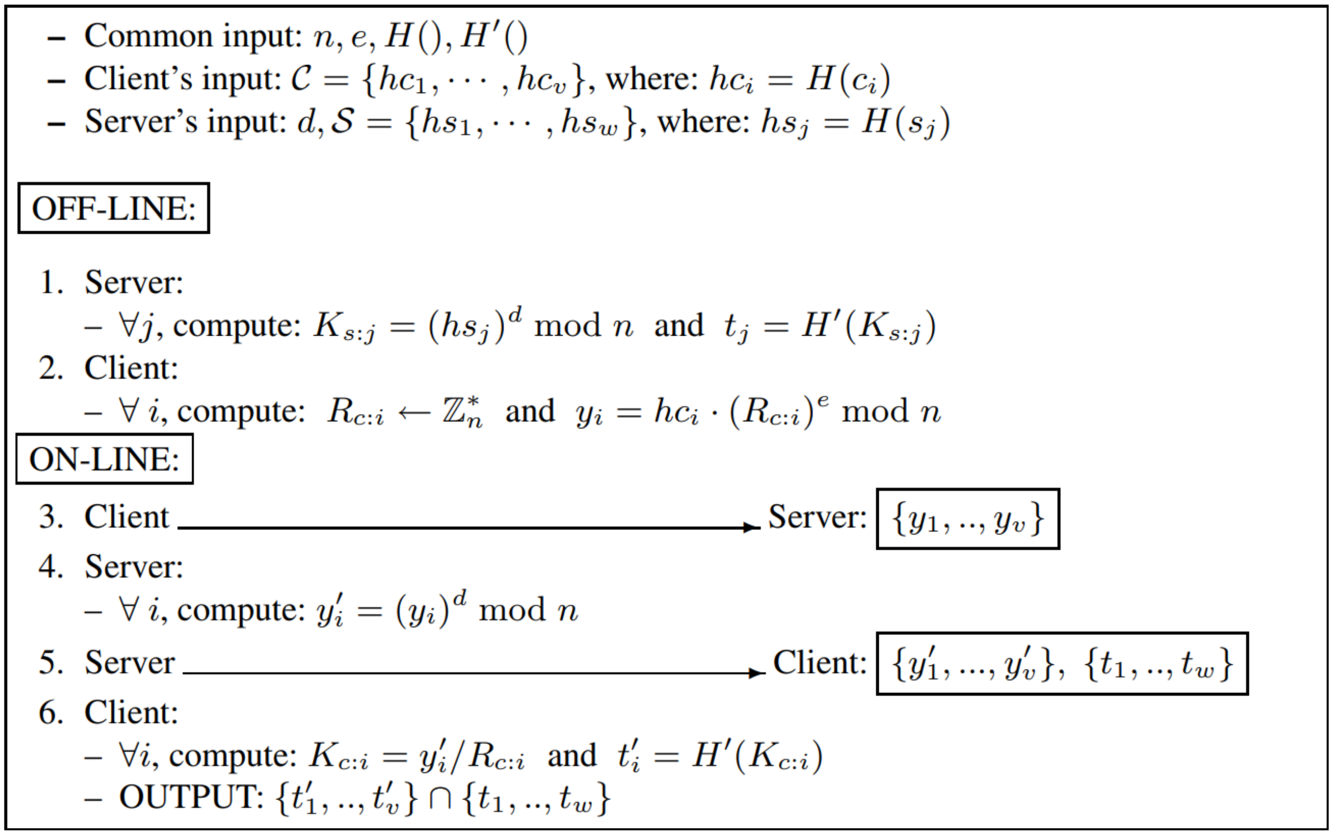
- Common input: $p, q, g, H(), H'()$
- Client's input: $\mathcal{C} = \{hc_1, \cdots, hc_v\}$, where: $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_j)$

1. Client:
   - $PCH = \prod_{i=1}^{v} hc_i$
   - $R_c \leftarrow \mathbb{Z}_q$ and $X = PCH \cdot g^{R_c}$
   - $\forall i$, $PCH_i = PCH/hc_i$, and $R_{c:i} \leftarrow \mathbb{Z}_q$, $y_i = PCH_i \cdot g^{R_{c:i}}$
2. Client $\xrightarrow{\hspace{6em}}$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_q$ and $Z = g^{R_s}$
   - $\forall\, j$, compute: $K_{s:j} = (X/hs_j)^{R_s}$, and $t_j = H'(K_{s:j})$
   - $\forall\, i$, compute: $y'_i = (y_i)^{R_s}$
4. Server $\xrightarrow{\hspace{6em}}$ Client: $\boxed{Z,\ \{y'_1, ..., y'_v\},\ \{t_1, .., t_w\}}$
5. Client:
   - $\forall i$, $K_{c:i} = y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$, and $t'_i = H'(K_{c:i})$
   - OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

最终结果：More Efficient PSI

引入预处理操作，减轻计算负担

具体过程：



- Common input: $n, e, H(), H'()$
- Client's input: $\mathcal{C} = \{hc_1, \cdots, hc_v\}$, where: $hc_i = H(c_i)$
- Server's input: $d, \mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_j)$

$\boxed{\text{OFF-LINE:}}$

1. Server:
   - $\forall j$, compute: $K_{s:j} = (hs_j)^d \bmod n$ and $t_j = H'(K_{s:j})$
2. Client:
   - $\forall\, i$, compute: $R_{c:i} \leftarrow \mathbb{Z}_n^*$ and $y_i = hc_i \cdot (R_{c:i})^e \bmod n$

$\boxed{\text{ON-LINE:}}$

3. Client $\xrightarrow{\hspace{6em}}$ Server: $\boxed{\{y_1, .., y_v\}}$
4. Server:
   - $\forall\, i$, compute: $y'_i = (y_i)^d \bmod n$
5. Server $\xrightarrow{\hspace{6em}}$ Client: $\boxed{\{y'_1, ..., y'_v\},\ \{t_1, .., t_w\}}$
6. Client:
   - $\forall i$, compute: $K_{c:i} = y'_i / R_{c:i}$ and $t'_i = H'(K_{c:i})$
   - OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

缺点:

1. Although very efficient, this PSI protocol has some issues. First, it is unclear how to convert it into an APSI version.
2. If precomputation is somehow impossible, its performance becomes worse than that of the PSI protocol