The Apple PSI System

Abhishek Bhowmick
Apple Inc.

Dan Boneh
Stanford University

Steve Myers
Apple Inc.

Kunal Talwar
Apple Inc.

Karl Tarbe
Apple Inc.

July 29, 2021

Abstract

This document describes the constraints that drove the design of the Apple *private set intersection* (PSI) protocol. Apple PSI makes use of a variant of PSI we call *private set intersection with associated data* (PSI-AD), and an extension called *threshold private set intersection with associated data* (tPSI-AD). We describe a protocol that satisfies the constraints, and analyze its security. The context and motivation for the Apple PSI system are described on the main project site.

1 Introduction

Apple requires a private set intersection (PSI) protocol that must satisfy a rigid set of constraints. We first describe the abstract problem that the protocol aims to solve and then describe the set of required constraints. The remainder of the document describes a specific protocol that meets the constraints. We describe the protocol in Section 4 and analyze its security in Section 4.4. Section 5 discusses a number of real world considerations, and Section 6 surveys existing PSI techniques in the literature and how they relate to the problem at hand. The goal of this writeup is primarily to describe the Apple PSI system and to inform the public how the protocol works.

2 Streaming threshold PSI with associated data

We begin by describing a warm-up problem we call **PSI with associated data** or **PSI-AD**. In [CT10], a variant of this problem is called *PSI with data transfer*.

Let $\mathcal{U}$ be the universe from which hash values are selected. In the Apple PSI system, the set $\mathcal{U}$ is the universe of all possible image hashes. The PSI-AD problem involves two parties, a server and a client, with the following setup:

- The server has a set of hash values $X \subseteq \mathcal{U}$ of size n. As usual for a set, the hashes in X must be distinct: every hash value can appear at most once in X.
- The client has an ordered list of m triples

  $$\bar{Y} = \Big((y_1, id_1, ad_1), \ldots, (y_m, id_m, ad_m)\Big) \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m.$$

  Each triple (y, id, ad) contains a hash value $y \in \mathcal{U}$, an identifier $id \in \mathcal{ID}$, and associated data $ad \in \mathcal{D}$. Every triple has a unique identifier id: if two triples in $\bar{Y}$ contain the same id, then both triples will also contain the same hash value and the same associated data. In other words, two triples that contain the same identifier are identical. We note that it is possible for two triples to contain the same hash value y, but different identifiers.

The triples in $\bar{Y}$ are provided to the client in a streaming fashion, one triple at a time, and must be "processed" as they arrive at the client. This is the reason why $\bar{Y}$ is defined as an *ordered* list of triples.

Identifiers are not secret. We stress that the identifiers $id \in \mathcal{ID}$ in the triples $\bar{Y}$ are not secret. They are sampled as fresh independent random bit strings. The intent is for the server to learn the entire list of identifiers in $\bar{Y}$. In practice they are used by the client to address objects stored on the server.

Notation: We will use the following notation to define the problem.
For $\bar{Y} \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m$ define

- $id(\bar{Y})$ is the set of identifiers of triples in $\bar{Y}$. For example, if

$$\bar{Y} = \big((y_1, id_1, ad_1),\ (y_2, id_2, ad_2),\ (y_3, id_3, ad_3),\ (y_1, id_1, ad_1)\big)$$

  then $id(\bar{Y}) = \{id_1, id_2, id_3\}$.

- $id(\bar{Y} \cap X)$ is the set of identifiers of triples in $\bar{Y}$ whose first component is also in X. For example, if $X = \{y_1, y_2\}$ and $\bar{Y}$ is as in the first bullet, then $id(\bar{Y} \cap X) = \{id_1, id_2\}$.

- $\bar{Y}_{id} \in \mathcal{ID}^m$ is the list of identifiers in the triples in $\bar{Y}$. If $\bar{Y}$ is as in the first bullet then $\bar{Y}_{id} = (id_1, id_2, id_3, id_1) \in \mathcal{ID}^4$.

- Projection: we write $\bar{Y}_{\{id,ad\}} \subseteq (\mathcal{ID} \times \mathcal{D})$ for the set of identifiers and associated data in the triples in $\bar{Y}$. For example, if $\bar{Y}$ is as in the first bullet then $\bar{Y}_{\{id,ad\}} = \big\{(id_1, ad_1),\ (id_2, ad_2),\ (id_3, ad_3)\big\}$.

- Selection: for a set of identifiers $T \subseteq \mathcal{ID}$ we use $\bar{Y}[T] \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^{\leq m}$ to denote the list of triples in $\bar{Y}$ whose identifiers are in T. For example, if $\bar{Y}$ is as in the first bullet and $T = \{id_1, id_2\}$, then $\bar{Y}[T] = \big((y_1, id_1, ad_1),\ (y_2, id_2, ad_2),\ (y_1, id_1, ad_1)\big)$.

- We write $x \leftarrow d$ to denote an assignment of a value d to the variable x. For a finite set $\mathcal{X}$ we write $x \overset{\$}{\leftarrow} \mathcal{X}$ to indicate that x is a random variable sampled uniformly over $\mathcal{X}$. For a randomized algorithm A we write $x \overset{\$}{\leftarrow} A(\cdot)$ to denote the random variable that is the output of $A(\cdot)$.

The PSI-AD problem is to design a protocol Π for the following functionality:

$$\mathcal{F}\big(X\ ;\ \bar{Y}\big) := \Big(\ \bar{Y}_{id},\ \bar{Y}[id(\bar{Y} \cap X)]_{\{id,ad\}}\ \ ;\ \ \perp\ \Big).$$

This terminology means that at the beginning of the protocol the server has X and the client has $\bar{Y}$. Once the protocol Π terminates, the server will learn $\bar{Y}_{id}$ and $\bar{Y}[id(\bar{Y} \cap X)]_{\{id,ad\}}$, and nothing else. That is, the server learns the list of all identifiers $\bar{Y}_{id}$, and the set of identifiers and associated data for all the triples in $\bar{Y}$ whose first component is in X. The server should learn nothing else. The client should learn nothing, although we usually relax this a bit and allow the client to learn the size of X. We will come back to the precise security requirements for the protocol Π in Section 4.4.

Remark 1 (Associated data). A PSI-AD protocol reveals to the server the associated data for triples in $\bar{Y}[id(\bar{Y} \cap X)]$. However, the hash values $y \in \mathcal{U}$ in those triples should remain hidden from the server. This gives the Apple PSI system flexibility in choosing what to embed in the associated data. The system can choose to make the associated data independent of y, in which case y remains hidden from the server. Or the system can choose to explicitly embed y in the associated data so that the complete contents of the intersection is revealed. Either way, a PSI-AD protocol is designed to reveal only the associated data of the intersection, but reveal nothing else about the hash values.

Remark 2 (PSI-CA and PSI-AD). **PSI cardinality**, or **PSI-CA**, is a variant of PSI-AD where the server learns the intersection cardinality (i.e., the size of $id(\bar{Y} \cap X)$) and nothing else. PSI-CA is a special case of PSI-AD. To see why, consider an instance of PSI-AD where the client selects all the identifiers and associated data independently at random from $(\mathcal{ID} \times \mathcal{D})$, and where $\mathcal{ID}$ is sufficiently large to that all the selected identifiers are distinct with high probability. Then the server learns the cardinality of $id(\bar{Y} \cap X)$ and nothing else. Indeed, all the identifiers and associated data in the server's PSI-AD output can be efficiently simulated just given the intersection cardinality. For this reason, designing a protocol for PSI-AD is at least as hard as designing a protocol for PSI-CA.

2.1 Threshold PSI-AD

The Apple PSI system is designed to solve a slightly more complicated problem called **threshold PSI with associated data** or **tPSI-AD**. The setup is as follows:

- As in PSI-AD, The client has an ordered list $\bar{Y} \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m$ of m triples.
- As in PSI-AD, the server has a set of hash values $X \subseteq \mathcal{U}$ of size n.
- In addition, there is a threshold parameter t known to both the server and the client.

The tPSI-AD problem is to design a protocol Π so that at the end of the protocol: (i) the client only learns $|X|$, and (ii) the server only learns $\bar{Y}_{id}$ and

- if $\left|id(\bar{Y} \cap X)\right| \leq t$ then the server learns $id(\bar{Y} \cap X) \subseteq \mathcal{ID}$,
- if $\left|id(\bar{Y} \cap X)\right| > t$ then the server learns $\bar{Y}[id(\bar{Y} \cap X)]_{\{id,ad\}} \subseteq (\mathcal{ID} \times \mathcal{D})$.

In other words, if $\left|id(\bar{Y} \cap X)\right| \leq t$ then the server learns the identifiers in $id(\bar{Y} \cap X)$, but learns no associated data. However, when $\left|id(\bar{Y} \cap X)\right| > t$, the server learns $\bar{Y}[id(\bar{Y} \cap X)]_{\{id,ad\}}$ which contains the associated data for all the identifiers in the intersection. Either way, the server learns nothing about the triples in $\bar{Y}$ that are outside the intersection (other than their identifiers).

2.2 Fuzzy threshold PSI-AD

While a protocol for threshold PSI-AD is sufficient for the Apple PSI system, we would like the system to satisfy one additional property. When the intersection cardinality is below the threshold, it is desirable that the server have some uncertainty as to the exact size of the intersection and its contents. The reason why this is needed is discussed in the technical summary document [App21].

To do so, we allow the client to introduce synthetic matches so that the server is uncertain which identifiers are in the intersection. We refer to this problem as **fuzzy threshold PSI with associated data** or **ftPSI-AD**. The setup is as follows:

- As in tPSI-AD, The client has an ordered list $\bar{Y} \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m$ of m triples. The server has a set of hash values $X \subseteq \mathcal{U}$ of size n. Both parties know the threshold parameter t.

- In addition, the client has a small secret set $S \subseteq id(\bar{Y})$ of identifiers that it designates as **synthetic matches**. The triples whose identifiers are in S will always appear to intersect with X, but will not count towards the threshold, nor have any associated data paired with them. We will explain the purpose of synthetic matches in a moment.

The ftPSI-AD problem is to design a protocol Π so that at the end of the protocol : (i) the client only learns $\left|X\right|$, and (ii) the server only learns $\bar{Y}_{id}$ and

- if $\left|id(\bar{Y} \cap X) \setminus S\right| \leq t$ then the server learns $id(\bar{Y} \cap X) \cup S \subseteq \mathcal{ID}$,
- if $\left|id(\bar{Y} \cap X) \setminus S\right| > t$ then the server learns $\bar{Y}[id(\bar{Y} \cap X) \setminus S]_{\{id,ad\}} \subseteq (\mathcal{ID} \times \mathcal{D})$ and the set $S \subseteq \mathcal{ID}$.

Note that when the set S of synthetics is empty, this functionality reduces to tPSI-AD from the previous section.

Now, suppose the client designates up to $k \cdot t$ elements in $id(\bar{Y})$ as synthetics, for some small constant k. Then, when the intersection cardinality is at most t, the server learns the set of identifiers $[id(\bar{Y} \cap X) \cup S]$. Because the protocol reveals nothing else about S, the server cannot tell which identifiers are in the intersection and which are synthetics. Thus, when the intersection is small, the set S introduces some uncertainty as to the exact size of the intersection as well as some uncertainty as to the set of identifiers in the intersection. We refer to the technical summary document [App21] for a further discussion of the purpose of synthetic matches and how the client selects the set S.

We note that there are protocols for tPSI-AD that fully hide the identifiers in the intersection when $\left|id(\bar{Y} \cap X)\right| \leq t$. Examples include protocols derived from [HOS17, GN17, ZC18, GS19, BDP20]. However, these protocols do not satisfy the rigid communication and performance requirements in Section 2.3.

2.3 Protocol requirements and constraints

Due to the specific problem settings, there are rigid constraints on the protocol's communication pattern and security requirements.

The required communication pattern. The protocol is only allowed to send the following messages:

- **Setup.** The server uses its set X to compute some public data, denoted pdata. The same pdata is then sent to all clients in the world (as part of an OS software update).

  The size of pdata can be proportional to the size of X, but must be independent of the number of clients. During setup the server can generate and store a short secret key, but no other state is stored on the server. Moreover, it should be possible for a trusted third party who knows both X and pdata to certify that pdata was constructed correctly.

- **Client vouchers.** Whenever the client receives a new triple $tr := (y, id, ad)$ it uses pdata to construct a short voucher V_{tr}, and sends V_{tr} to the server. No other communication is allowed between the client and the server. The work to create V_{tr}

and the length of V_{tr} should be at most linear in the threshold (and at most linear in the length of a hash value, the size of an identifier, and the length of the associated data). Importantly, the client's work to create V_{tr} and its length should be independent of $\lvert X \rvert$ and $\lvert \bar{Y} \rvert$. All vouchers should have the same length.

All client-side processing of a triple tr must be done as soon as the client receives this triple. A protocol that delays the processing of client triples, so that triples are processed as a batch, cannot be used. This is due to the logic for processing data on the client. Moreover, if there is a gap between the time that the triple is received and the time that the triple is processed by the client, then a device reset during the delay period would cause the triple to never be processed.

- **Processing at the server.** The server uses its short secret key, generated during setup, to process incoming vouchers from the client. No other state is required at the server for this. When a voucher is first received at the server, the server processes it and marks it as non-matching, if that voucher was computed from a non matching hash value. Once every time period (say, once a week) the server processes all the remaining vouchers, namely ones computed from real or synthetic matches, to reveal the ftPSI-AD output based on all the data sent from the client so far.

- **Client abort.** If the client aborts the protocol, the server should be able to process the set of vouchers it received prior to the abort and obtain the ftPSI-AD output from the vouchers sent prior to the client abort.

- **Updates.** If needed, the server should have the option to push an updated pdata to the client (as part of an OS software update), and the vouchers previously computed by the client using the old pdata should be "compatible" with vouchers computed using this new pdata. That is, vouchers pre- and post- update should jointly count towards the threshold.

In addition, we note that the contents and the size of the set of synthetics S is not known to the client at setup time. The set is constructed incrementally over time.

High level security requirements. We define the precise security model in Section 4.4. Here we explain the security requirements at a high level.

- *Privacy for the server:* A malicious client should learn nothing about the server's dataset $X \subseteq \mathcal{U}$ other than its size. In particular, it is important that the client learn nothing about the intersection size $\lvert id(\bar{Y} \cap X) \rvert$. Otherwise, the client can use that to extract information about X by adding test items to its list $\bar{Y}$, and checking if the intersection size changes.

- *Privacy for the client:* Let X be the server's input from which pdata is derived. A malicious server must learn nothing about the client's $\bar{Y}$ beyond the output of ftPSI-AD with respect to this set X. This will be defined precisely in Section 4.4.

- The protocol should have no false positives: for a client triple $(y, id, ad) \in \bar{Y}$, if the client is behaving honestly and $y \notin X$, then the server should learn nothing about y or ad. This rules out some constructions that rely on Bloom filters.

- The protocol need not provide correct output against a malicious client. That is, the protocol need not prevent a malicious client from causing the server to obtain an incorrect ftPSI-AD output when the protocol terminates. The reason for this is that a malicious client can always choose to hide some of its data from the PSI system in order to cause an undercount of the intersection. Moreover, a malicious client that attempts to cause an overcount of the intersection will be detected by mechanisms outside of the cryptographic protocol.

Remark 3 (false negatives). In what follows we will slightly relax the correctness requirement and allow the protocol to miss a small number of random real matches. That is, let X' be a random subset of X where $|X'| \geq (1-\delta)|X|$ for some small $\delta > 0$. We allow the protocol to run with the server contributing X' rather than X as its input. Consequently, the server will only learn the associated data for elements that match X', not X, when $|id(\bar{Y} \cap X') \setminus S| > t$. This improves performance, and has little impact on the effectiveness of the system since X is updated periodically and after each update a new random X' is computed, as discussed in Section 5. If needed, these false negatives can be eliminated with a tweak to the data structure used.

3 Building blocks

The protocols described in the next section make use of a number of cryptographic primitives:

- (Enc, Dec) is a symmetric encryption scheme with key space $\mathcal{K}'$. We will need (Enc, Dec) to satisfy two security properties. First, we need the scheme to satisfy a standard security notion called IND\$-CPA security (see, e.g., [Rog04, §3] for a definition). Second we need the scheme to satisfy a weak form of robustness [ABN18, FLPQ13, FOR17] we call *random key robustness*. This property says that if a message m is encrypted under one random key k and then decrypted under another independent random key k', then decryption should fail with high probability. More precisely, for all efficient adversaries $\mathcal{A}$

$$\Pr\left[m \xleftarrow{\$} \mathcal{A}(); \quad k, k' \xleftarrow{\$} \mathcal{K}'; \quad c \xleftarrow{\$} \mathrm{Enc}(k, m) \ : \ \mathrm{Dec}(k', c) \neq \mathit{reject}\right] \leq \text{negl.} \qquad (1)$$

  A symmetric encryption scheme that provides authenticated encryption [BN08] (meaning that it is both IND\$-CPA secure and has ciphertext integrity), satisfies both properties above. The implementation uses AES128-GCM with a random 96-bit nonce.

- $E(\mathbb{F}_p)$ is an elliptic curve of prime order q. Let G be a fixed generator of $E(\mathbb{F}_p)$. We will need the Decision Diffie-Hellman (DDH) assumption to hold in $E(\mathbb{F}_p)$. The implementation uses the curve NIST P256 with the generator G specified in the NIST standard. We refer to [BS20, ch. 15] as a reference for these concepts.

- $H : \mathcal{U} \to E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ is a hash function that we will model as a random oracle. The implementation uses a construction based on the evolving Hash to Elliptic Curves standard in the Internet Engineering Task Force's Crypto Forum Research Group's draft standard [FHSS+20].

- $H' : E(\mathbb{F}_p) \to \mathcal{K}'$ is a secure key derivation function, namely, the uniform distribution on $E(\mathbb{F}_p)$ is mapped to an almost uniform distribution on $\mathcal{K}'$. One can use HKDF [KE10].
- We will use Shamir secret sharing on an element of $\mathcal{K}'$ to obtain shares in $\mathbb{F}_{\mathrm{Sh}}^2$ for some field $\mathbb{F}_{\mathrm{Sh}}$. The field $\mathbb{F}_{\mathrm{Sh}}$ needs to be sufficiently large so that when choosing $t+1$ random elements from $\mathbb{F}_{\mathrm{Sh}}$, the probability of a collision is low.
- A pseudorandom function (PRF) $F : \mathcal{K}'' \times \mathcal{ID} \to \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$, where the sets $\mathcal{X}$ and $\mathcal{R}$ are the domain and range of a detectable hash function, respectively, as defined in Section 3.2 below. This PRF can be constructed from HMAC [KBC97].

3.1 The Diffie-Hellman random self reduction

Let $\mathbb{G}$ be a group of prime order q where the group operation is written additively. Fix a generator $G \in \mathbb{G}$. We say that a triple (L, U, V) in $\mathbb{G}^3$ is a **Diffie-Hellman tuple**, or **DH tuple**, if there exists an $\alpha \in \mathbb{F}_q$ such that $L = \alpha \cdot G$ and $V = \alpha \cdot U$.

Naor and Reingold [NR97] describe a partial random self reduction for DH tuples. Given a triple (L, T, P) in $\mathbb{G}^3$ as input, the self reduction does:

- choose a random β, γ in $\mathbb{F}_q$,
- compute $Q \leftarrow \beta \cdot T + \gamma \cdot G$ and $S \leftarrow \beta \cdot P + \gamma \cdot L$,
- output (L, Q, S).

Naor and Reingold show that this transformation $(L, T, P) \to (L, Q, S)$ has the following properties:

- if the provided triple (L, T, P) is a DH tuple, where $L = \alpha \cdot G$, then Q is a fresh uniformly sampled element in $\mathbb{G}$, and $S = \alpha \cdot Q$.
- if the provided triple (L, T, P) is not a DH tuple, then (Q, S) is a fresh uniformly sampled pair in $\mathbb{G}^2$.

These properties will be used in the next section.

3.2 Detectable hash functions

We will also need a new primitive which we call an (s, t)-detectable hash function, or (s, t)-DHF. In what follows we say that a hash function $\mathrm{DHF} : \mathcal{K} \times \mathcal{X} \to \mathcal{R}$ is **weak t-wise independent** if the distribution

$$\left\{ k \overset{\$}{\leftarrow} \mathcal{K}, \quad x_1, \ldots, x_t \overset{\$}{\leftarrow} \mathcal{X}, \quad \text{output } \big(\mathrm{DHF}(k, x_1), \ldots, \mathrm{DHF}(k, x_t)\big) \in \mathcal{R}^t \right\}$$

is identical to the uniform distribution on $\mathcal{R}^t$. We use the "weak" qualifier to indicate that the t-wise independence property need only hold for *random* elements in the domain $\mathcal{X}$.

We say that a weak t-wise independent hash is s-detectable if there is an efficient deterministic **detection algorithm** D that is invoked as $D(\mathbf{v})$, where $\mathbf{v}$ is a vector $\mathbf{v} = (v_1, \ldots, v_m) \in \mathcal{R}^m$. The algorithm outputs a set $\hat{T} \subseteq [m]$ or outputs *fail*. The detection algorithm D must satisfy the following property:

For a subset $T \subseteq [m]$ define the following distribution $\mathcal{D}_T$ on $\mathcal{R}^m$:

- choose k uniformly in $\mathcal{K}$;
- construct a vector $\mathbf{v} \leftarrow (v_1, \dots, v_m) \in \mathcal{R}^m$ as follows: for $i = 1, \dots, m$:
  if $i \in T$ then $x_i \xleftarrow{\$} \mathcal{X}$, $v_i \leftarrow \mathrm{DHF}(k, x_i)$, else $v_i \xleftarrow{\$} \mathcal{R}$;
- output $\mathbf{v}$.

We say that a weak t-wise independent DHF is **s-detectable** if for all bounded m and all $T \subseteq [m]$, where $|T| \geq \max(t+1, m-s)$, the detection algorithm D satisfies:

$$\Pr\big[\ \mathbf{v} \xleftarrow{\$} \mathcal{D}_T\ :\ D(\mathbf{v}) = T\ \big] \geq 1 - \epsilon$$

for some small ϵ (e.g., ϵ less than 2^{-60}).

In other words, when at least $t+1$ entries in $\mathbf{v} \in \mathcal{R}^m$ are generated by DHF, and at most s entries of $\mathbf{v}$ are random in $\mathcal{R}$, the detection algorithm correctly identifies all the entries in $\mathbf{v}$ that were generated using DHF. However, if only t entries in $\mathbf{v}$ are generated by DHF, then by the weak t-wise independence property, nothing is revealed about which entries of $\mathbf{v}$ were generated by DHF.

Definition 1. A hash function $\mathrm{DHF} : \mathcal{K} \times \mathcal{X} \to \mathcal{R}$ is a **(s,t)-DHF** if (i) the function DHF is weak t-wise independent and s-detectable, (ii) the set $\mathcal{X}$ is sufficiently large so that choosing $t+1$ random elements from $\mathcal{X}$ results in distinct elements with high probability, and (iii) all the elements in $\mathcal{R}$ have equal length as binary strings.

A voucher in our protocol will contain an element in $\mathcal{R}$, and therefore the set $\mathcal{R}$ should be as small as possible. We will construct a suitable (s,t)-DHF in Section 4.3.

4 Threshold PSI-AD using the DH random self reduction

In this section describe an approach to ftPSI-AD that meets the requirements from Section 2.3. We assume that all data exchanged between the client and the server is sent over a secure and mutually authenticated channel.

4.1 A protocol for tPSI-AD

As a warm-up, we first describe a protocol for threshold PSI-AD (tPSI-AD). In the next section we will extend the protocol to add support for synthetic matches and obtain a protocol for ftPSI-AD. Throughout the section we use t for the threshold, and m for an upper bound on the number of triples that the client will process.

Server setup. The server constructs the public data pdata by processing its set $X \subseteq \mathcal{U}$ as follows:

- *step 0:* Process X to remove any duplicates. Let $n := |X|$.
- *step 1:* Construct a Cuckoo table T:

  - Let $n' \leftarrow (1+\epsilon') \cdot n$, where the choice of ϵ' will be explained in a moment.
  - Choose random hash functions $h_1, h_2 : \mathcal{U} \rightarrow \{1, \ldots, n'\}$ and a random hash function $H : \mathcal{U} \rightarrow E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$. These hash functions may depend on $|X|$, but not on X.
  - Insert the elements of X into a Cuckoo table T of size $n' = (1+\epsilon') \cdot n$ using the two hash functions $h_1, h_2 : \mathcal{U} \rightarrow \{1, \ldots, n'\}$, where there is at most one element of X in each cell of T.

    Elements in X that cannot be inserted into the Cuckoo table are dropped. The value ϵ' is set so that at most a small fraction of elements in X will be dropped (see Remark 3). If needed, the fraction of dropped elements can be reduced by using three hash functions h_1, h_2, h_3. This has little impact on the protocol. See also Remark 7 below regarding an improvement to the Cuckoo data structure.
  - **Caution:** it is important to ensure that there are no collisions among the Cuckoo hash functions h_1, h_2, namely, there is no $y \in \mathcal{U}$ such that $h_1(y) = h_2(y)$. An easy way to ensure that there are no collisions is to tweak the hash functions whenever a collision occurs. That is, if $h_1(y) = h_2(y)$ then define $h_2(y) \leftarrow h_1(y) + 1$. [if $h_1(y) = h_2(y) = n'$ then set $h_2(y) \leftarrow 1$]

- *step 2:* The server chooses a random $\alpha \neq 0$ in $\mathbb{F}_q$ and computes $L \leftarrow \alpha G \in E(\mathbb{F}_p)$. This α will be kept secret by the server.

- *step 3:* For $i = 1$ to n' compute using the Cuckoo table T:

  - if $T[i]$ is not empty, set $P_i \leftarrow \alpha \cdot H(T[i]) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$, where $T[i] \in \mathcal{X} \subseteq \mathcal{U}$.
  - if $T[i]$ is empty, choose P_i at random from $E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$.
  - Among the resulting n' points, it is important that points that were generated using the first bullet are indistinguishable from points that were generated using the second bullet. In the proof of security we will use the Decision Diffie-Hellman assumption to argue that this is the case.

- *step 4:* Set $\mathsf{pdata} := (L, P_1, \ldots, P_{n'})$ along with a description of the hash functions H, h_1, h_2. The description of each hash function is a random 128-bit domain separation nonce chosen at random at the beginning of the Cuckoo table construction.

Client setup. The client performs the following steps:

- Obtain pdata. Ensure that L and $P_1, \ldots, P_{n'}$ are non-zero points in $E(\mathbb{F}_p)$ and are all distinct. If not, pdata is rejected and the client aborts.

- Select fresh random keys:

  - Choose a random secret key $adkey \xleftarrow{\$} \mathcal{K}'$ for the encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$.
  - Choose a random secret key $fkey \xleftarrow{\$} \mathcal{K}''$ for the PRF $F : \mathcal{K}'' \times \mathcal{ID} \rightarrow \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$.
  - Initialize a threshold Shamir secret sharing for $adkey$, so that $t+1$ or more shares are needed to reconstruct $adkey$.

This completes the description of the setup procedure.

Client generates a voucher for a triple $(y, id, ad) \in \mathcal{U} \times \mathcal{ID} \times \mathcal{D}$**.**

- *step 1:* compute
$$adct \xleftarrow{\$} \mathrm{Enc}\big(adkey,\ ad\big).$$
That is, we encrypt ad using the key $adkey$. The system must ensure that all ciphertexts $adct$ are the same length.

- *step 2:* compute $(x, z, x', r') \leftarrow F(fkey, id) \in \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$. In this section we only use the x output of F. We will need z, x', r' in the next section.

- *step 3:* generate a new threshold Shamir secret share $sh \in \mathbb{F}_{\mathrm{Sh}}^2$ of $adkey$, so that $t+1$ shares are needed to reconstruct $adkey$. The x-coordinate of sh is set to $x \in \mathbb{F}_{\mathrm{Sh}}$ computed in step 2.

  Note: $\mathbb{F}_{\mathrm{Sh}}$ needs to be large enough so that when choosing $t+1$ pseudorandom x-coordinates from $\mathbb{F}_{\mathrm{Sh}}$, the probability of a collision is small. The reason to use a pseudorandom x-coordinate derived from id is to ensure that duplicate triples with the same id always result in the same Shamir share sh.

- *step 4:* choose a random key $rkey \xleftarrow{\$} \mathcal{K}'$ and compute
$$rct \xleftarrow{\$} \mathrm{Enc}\big(rkey, (adct, sh)\big).$$
That is, rct is an encryption of $(adct, sh)$ using the key $rkey$.

- *step 5:* for $j = 1, 2$ do:
  - *step 5.1:* compute $w \leftarrow h_j(y) \in \{1, \dots, n'\}$
    (recall that h_1, h_2 are the Cuckoo hash functions).
  - *step 5.2:* choose random β_j and γ_j in $\mathbb{F}_q$ and use P_w, L from `pdata` to compute:
$$Q_j \leftarrow \beta_j \cdot H(y) + \gamma_j \cdot G \quad \text{and} \quad S_j \leftarrow \beta_j \cdot P_w + \gamma_j \cdot L.$$
    Intuition: the client is applying the DH random self reduction from Section 3.1 to the triple $(L, H(y), P_w)$. If $y = T[w]$ then $P_w = \alpha \cdot H(y)$ and then (Q_j, S_j) satisfies $\alpha \cdot Q_j = S_j$. Otherwise, (Q_j, S_j) is a pair of random independent points in $E(\mathbb{F}_p)$.
  - *step 5.3:* set $ct_j \xleftarrow{\$} \mathrm{Enc}\big(H'(S_j),\ rkey\big)$.
    That is, ct_j is an encryption of $rkey$ using the key $H'(S_j) \in \mathcal{K}'$.

- *step 6:* choose a random bit b in $\{1, 2\}$ and set
$$voucher \leftarrow (id,\ Q_b, ct_b,\ Q_{3-b}, ct_{3-b},\ rct).$$
Send this voucher to the server.

  As we will see, if y is a match (i.e. y is in X), then exactly one of ct_1 or ct_2 will be successfully decrypted on the server. We also note that the ciphertext rct, is the longest element in the voucher; the other fields in the voucher are short.

The server processes the set of received vouchers. At any time, the server can process the set of received vouchers to obtain the tPSI-AD output based on the data processed so far by the client. Recall that $\alpha \in \mathbb{F}_q$ is the server's secret generated during server setup.

- *step 0:* initialize an empty set SHARES and an empty list IDLIST.
- *step 1:* for each received voucher $(id, Q_1, ct_1, Q_2, ct_2, rct)$ do:
  - append id to the list IDLIST.
  - for $j = 1, 2$ compute
    - set $\hat{S}_j \leftarrow \alpha \cdot Q_j \in E(\mathbb{F}_p)$
    - set $rkey_j \leftarrow \mathrm{Dec}(H'(\hat{S}_j), ct_j)$;
      if decryption fails then set $goodkey_j \leftarrow false$ and skip the next step.
    - set $(adct_j, sh_j) \leftarrow \mathrm{Dec}(rkey_j, rct)$;
      if decryption fails then set $goodkey_j \leftarrow false$, otherwise set $goodkey_j \leftarrow true$.
  - if $goodkey_1 = goodkey_2 = false$ then this is a non-match and the voucher can be ignored.
  - if for $b \in \{1, 2\}$ we have $goodkey_b = true$ but $goodkey_{3-b} = false$ then add the triple $(id, adct_b, sh_b)$ to the set SHARES. This is a match. Recall that sh_b is a Shamir share of $adkey$.
  - if $goodkey_1 = goodkey_2 = true$, namely both ct_1 and ct_2 were successfully decrypted, then with high probability this client voucher is invalid and can be ignored.
- Main point: if the voucher corresponds to a triple (y, id, ad) where y is a match (i.e., y is in X) then exactly one of ct_1 or ct_2 will decrypt correctly to $rkey$, from which the server obtains the pair $(adct, sh)$ that the client computed in Steps 1 and 3. For all other ciphertexts ct_j, the point $\hat{S}_j = \alpha \cdot Q_j$ that the server computes is independent of the point S_j used by the client in Step 5.3, and then decryption of ct_j should fail by the random key robustness property of $(\mathrm{Enc}, \mathrm{Dec})$.
- *step 2:* let t' be the number of distinct Shamir shares in the set SHARES. If the client is behaving honestly, then t' should be equal to the size of $id(\bar{Y} \cap X)$.
  - if $t' \leq t$ then let OUTSET be the set of identifiers in SHARES.
  - if $t' > t$ then do:
    - use $(t + 1)$ distinct Shamir shares in SHARES to reconstruct $adkey \in \mathcal{K}'$.
    - initialize an empty set OUTSET.
    - for each triple $(id, adct, sh)$ in SHARES use $adkey$ to decrypt the ciphertext $adct$ to obtain
      $$ad \leftarrow \mathrm{Dec}(adkey,\ adct).$$
      If decryption fails, then the corresponding voucher is invalid. If decryption succeeds, add the pair (id, ad) to the set OUTSET.
  - output the list IDLIST and the set OUTSET.

This completes the description of the protocol.

Correctness. Let us briefly argue that when the client and the server honestly follow the protocol, the server learns the required tPSI-AD output.

Theorem 1 (correctness). *Suppose that the client and the server honestly follow the protocol, and that (i) $H' : E(\mathbb{F}_p) \to \mathcal{K}'$ is a secure key derivation function, (ii) $(\mathrm{Enc}, \mathrm{Dec})$ is random key robust as defined in (1) on page 6, and (iii) F is a secure PRF. Then the server learns the required tPSI-AD output with high probability.*

Proof Sketch. First, by construction, the output IDLIST is equal to $\bar{Y}_{id}$. Second, let us show that the server learns the set $id(\bar{Y} \cap X)$, as required in tPSI-AD.

Let $(y, id, ad) \in \bar{Y}$ be a client triple, and let $(id,\ Q_1, ct_1,\ Q_2, ct_2,\ rct)$ be the corresponding voucher sent to the server for this triple.

Suppose that id is not in $id(\bar{Y} \cap X)$. We know that with high probability, the point $\alpha H(y) \in E(\mathbb{F}_p)$ is not equal to either points $P_{h_1(y)}$ or $P_{h_2(y)}$ in pdata. In other words, neither $(L, H(y), P_{h_1(y)})$ nor $(L, H(y), P_{h_2(y)})$ are Diffie-Hellman tuples. Now, for $j = 1, 2$, the client creates the ciphertext ct_j using a key computed as $H'(S_j)$. The server, however, attempts to decrypt ct_j using the key $H'(\alpha Q_j)$. We know that Q_j and S_j are two random independent points in $E(\mathbb{F}_p)$, by the properties of the Diffie-Hellman random self reduction from Section 3.1. Therefore, the keys $H'(S_j)$ and $H'(\alpha Q_j)$ are sampled (almost) uniformly and independently in $\mathcal{K}'$. Hence the random key robustness property of $(\mathrm{Enc}, \mathrm{Dec})$ implies that, with high probability, the server will fail to decrypt both ct_1 and ct_2.

For id that is in $id(\bar{Y} \cap X)$, we know that exactly one of the tuples $(L, H(y), P_{h_1(y)})$ or $(L, H(y), P_{h_2(y)})$ is a Diffie-Hellman tuple. Here we are using the fact that for all y, the points $P_{h_1(y)}$ and $P_{h_2(y)}$ in pdata are distinct points in $E(\mathbb{F}_p)$, and that the Cuckoo hash functions h_1, h_2 are collision free so that $h_1(y) \neq h_2(y)$. Therefore, the server succeeds in decrypting exactly one of the ciphertexts ct_1 or ct_2 in the voucher $(id,\ Q_1, ct_1,\ Q_2, ct_2,\ rct)$, and learns $rkey$ for that voucher. It will fail to decrypt the other ciphertext for the same reason as in the previous paragraph.

These two observations show that when the protocol terminates, the server obtains the the set of identifiers $id(\bar{Y} \cap X)$, as required in tPSI-AD. This is the set of identifiers from vouchers for which the server succeeds in decrypting one of ct_1 or ct_2.

Next, suppose that $id(\bar{Y} \cap X)$ contains more than t elements. Then the server can reconstruct $adkey$. To see why, observe that as we explained above, the server learns $rkey$ for all vouchers $(id,\ Q_1, ct_1,\ Q_2, ct_2,\ rct)$ where id is in $id(\bar{Y} \cap X)$. This $rkey$ can be used to decrypt rct from the voucher to reveal the pair $(adct, sh)$, where the second element is a Shamir share of $adkey$. Because the field $\mathbb{F}_{\mathrm{Sh}}$ is sufficiently large, and F is a secure PRF, this reveals more than t distinct Shamir shares of $adkey$ with high probability. From these, the server can reconstruct $adkey$. Once the server has $adkey$ it can decrypt all the ciphertexts $adct$ that correspond to identifiers in $id(\bar{Y} \cap X)$. This reveals $\bar{Y}[id(\bar{Y} \cap X)]_{\{id,ad\}}$ as required. $\square$

Remarks. Before we continue, we first make a few remarks about the protocol.

Remark 4 (No client output). The client should learn nothing about the server's output from the protocol. Otherwise, the client could use the server's output to compromise the server's dataset X, by adding test items to its data $\bar{Y}$, and checking if $\left|id(\bar{Y} \cap X)\right|$ changes. In particular, all communications from the server to the client should be independent of the server's output from the protocol.

Remark 5 (Validating the set X). Recall that during setup, the server builds a cuckoo table for the provided set of values $X \subseteq \mathcal{U}$. It fills the empty slots in the cuckoo table with random points in $E(\mathbb{F}_p)$. A malicious server could try to choose those entries non-randomly so as to match spurious hash values in $\mathcal{U}$, thereby secretly increasing the size of the set X. This is called *overfitting* in [PRTY20, GPR+21]. We treat this the same way as one would treat a malicious server who is attempting to modify the provided set X. In principle, one could mitigate tampering with the set X by relying on a third party, who knows both pdata and X, to certify that pdata is constructed correctly for X. The Apple PSI system addresses this issue differently, by using measures implemented outside of the cryptographic protocol.

Remark 6 (Choosing Shamir shares). To reduce the size of the vouchers, and thus bandwidth used by the client, the client can use a subset of $\mathbb{F}_{\mathsf{Sh}}$ for the x-coordinate in Shamir Secret Sharing. The only requirement is that when choosing $t+1$ pseudorandom elements from the subset, the probability of a collision is sufficiently small. If we limit the size of the set from which we draw the x-coordinates in real vouchers, we must do the same for synthetics in the protocol described in the next section.

Remark 7 (An improvement to the Cuckoo data structure). The work of [PRTY20, GPR+21] proposes an improvement to the Cuckoo data structure that enables us to further shrink the voucher by including only a single point from $E(\mathbb{F}_p)$ in the voucher.

4.2 A protocol for ftPSI-AD

In this section we extend the tPSI-AD protocol from the previous section to add support for synthetic matches. Recall that synthetic matches were discussed in Section 2.2. As before, we use t for the threshold, and m for the maximum number of triples that the client will process. We use $s_{\max}$ to denote an upper bound on the size of the set S, namely an upper bound on the number of synthetic matches.

Server setup. The server setup is identical to the server setup in the previous section.

Client setup. The client performs the following steps:

- Obtain pdata. Ensure that L and $P_1, \ldots, P_{n'}$ are non-zero points in $E(\mathbb{F}_p)$ and are all distinct. If not, pdata is rejected and the client aborts.
- Select fresh random keys:
  - Let DHF : $\mathcal{K} \times \mathcal{X} \to \mathcal{R}$ be a $(s_{\max}, t)$-DHF, namely a weak t-wise independent $s_{\max}$-detectable hash function. Recall that t is the ftPSI-AD threshold and $s_{\max}$ is an upper bound on the size of the set S. Choose a random secret key $hkey \xleftarrow{\$} \mathcal{K}$.
  - Choose a random secret key $adkey \xleftarrow{\$} \mathcal{K}'$ for the encryption scheme (Enc, Dec).
  - Choose a random secret key $fkey \xleftarrow{\$} \mathcal{K}''$ for the PRF $F : \mathcal{K}'' \times \mathcal{ID} \to \mathbb{F}_{\mathsf{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$.
  - Initialize a threshold Shamir secret sharing for $adkey$, so that $t+1$ or more shares are needed to reconstruct $adkey$.

This completes the description of the setup procedure.

Client generates a voucher for a triple $(y, id, ad) \in \mathcal{U} \times \mathcal{ID} \times \mathcal{D}$**.** There are two cases: either $id \in S$ or $id \notin S$. We describe each in turn.

Case 1: $id \notin S$ (i.e., the triple is not synthetic)

- *step 1:* compute
$$adct \xleftarrow{\$} \text{Enc}\big(adkey,\ ad\big).$$
That is, we encrypt ad using the key $adkey$. The system must ensure that all ciphertexts $adct$ are the same length.

- *step 2:* compute $(x, z, x', r') \leftarrow F(fkey, id) \in \mathbb{F}_{\text{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$ and
$$r \leftarrow \text{DHF}(hkey, x') \in \mathcal{R}.$$
We use x and r in the next two steps, but will not use z and r' in this part.

  Note: the domain $\mathcal{X}$ of the DHF needs to be large enough so that when choosing $t+1$ pseudorandom elements in the domain, the probability of a collision is small.

- *step 3:* generate a new threshold Shamir secret share $sh \in \mathbb{F}_{\text{Sh}}^2$ of $adkey$, so that $t+1$ shares are needed to reconstruct $adkey$. The x-coordinate of sh is set to $x \in \mathbb{F}_{\text{Sh}}$ computed in step 2.

  Note: $\mathbb{F}_{\text{Sh}}$ needs to be large enough so that when choosing $t+1$ pseudorandom x-coordinates from $\mathbb{F}_{\text{Sh}}$, the probability of a collision is small.

- *step 4:* choose a random key $rkey \xleftarrow{\$} \mathcal{K}'$ and compute
$$rct \xleftarrow{\$} \text{Enc}\big(rkey, (r, adct, sh)\big).$$
That is, rct is an encryption of $(r, adct, sh)$ using the key $rkey$. This is similar to rct in the tPSI-AD protocol with the primary difference being the inclusion of r.

- *step 5:* for $j = 1, 2$ do:
  - *step 5.1:* compute $w \leftarrow h_j(y) \in \{1, \dots, n'\}$
    (recall that h_1, h_2 are the Cuckoo hash functions).
  - *step 5.2:* choose random β_j and γ_j in $\mathbb{F}_q$ and use P_w, L from `pdata` to compute:
  $$Q_j \leftarrow \beta_j \cdot H(y) + \gamma_j \cdot G \quad \text{and} \quad S_j \leftarrow \beta_j \cdot P_w + \gamma_j \cdot L.$$
  Intuition: if $y = T[w]$ then $P_w = \alpha \cdot H(y)$ and then (Q_j, S_j) satisfies $\alpha \cdot Q_j = S_j$. Otherwise, (Q_j, S_j) is a pair of random independent points in $E(\mathbb{F}_p)$.
  - *step 5.3:* set $ct_j \xleftarrow{\$} \text{Enc}\big(H'(S_j),\ rkey\big)$.
    That is, ct_j is an encryption of $rkey$ using the key $H'(S_j) \in \mathcal{K}'$.

- *step 6:* choose a random bit b in $\{1, 2\}$ and set
$$voucher \leftarrow (id,\ Q_b, ct_b,\ Q_{3-b}, ct_{3-b},\ rct).$$
Send this voucher to the server. If y is a match (i.e. y is in X), then exactly one of ct_1 or ct_2 will be successfully decrypted on the server.

Case 2: $id \in S$ (i.e., the triple (y, id, ad) is designated as synthetic)

- *step 1:* choose a random key *ckey* in $\mathcal{K}'$ and compute

  $$adct \xleftarrow{\$} \mathrm{Enc}(ckey,\ 000\ldots00).$$

  That is, encrypt a sequence of 0's using the key *ckey* in $\mathcal{K}'$. All *adct*, real and synthetic, should be the same length and indistinguishable. As we will see, the server will never decrypt this *adct*.

- *step 2:* compute $(x, z, x', r) \leftarrow F(fkey, id) \in \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$. We use x, z, r in the next two steps, but will not use x' in this part.

- *step 3:* set $dummy \leftarrow (x, z) \in \mathbb{F}_{\mathrm{Sh}}^2$, using the x, z computed in step 2 (a dummy Shamir secret share),

- *step 4:* choose a random key $rkey \xleftarrow{\$} \mathcal{K}'$ and compute $rct \xleftarrow{\$} \mathrm{Enc}\big(rkey, (r, adct, dummy)\big)$ using the pseudorandom $r \in \mathcal{R}$ computed in step 2.

- step 5: do:

  - *step 5.1:* choose a random β in $\mathbb{F}_q$; set $Q_1 \leftarrow \beta \cdot G$ and $S_1 \leftarrow \beta \cdot L$;
  - *step 5.2:* choose Q_2 and S_2 independently at random in $E(\mathbb{F}_p)$;

    intuition: (Q_1, S_1) satisfies $S_1 = \alpha \cdot Q_1$, but (Q_2, S_2) is a random pair of points.
  - *step 5.3:* set $ct_1 \xleftarrow{\$} \mathrm{Enc}\big(H'(S_1),\ rkey\big)$ and $ct_2 \xleftarrow{\$} \mathrm{Enc}\big(H'(S_2),\ rkey\big)$.
    Here ct_1 and ct_2 are an encryption of *rkey* using the keys $H'(S_1)$ and $H'(S_2)$ respectively.

- *step 6:* choose a random bit b in $\{1, 2\}$ and set

  $$voucher \leftarrow (id,\ Q_b, ct_b,\ Q_{3-b}, ct_{3-b},\ rct).$$

  Send this voucher to the server.

  Intuition: exactly one of ct_1 or ct_2 will be successfully decrypted on the server, as with a real match. As long the intersection size is t or less, the resulting $(rkey, r, adct, dummy)$ will be indistinguishable from a real match.

The server processes the set of received vouchers. At any time, the server can process the set of received vouchers to obtain the ftPSI-AD output based on the data processed so far by the client.

- *step 0:* initialize an empty set SHARES and an empty list IDLIST.

- *step 1:* for each received voucher $(id, Q_1, ct_1, Q_2, ct_2, rct)$ do:

  - append *id* to the list IDLIST.
  - for $j = 1, 2$ compute
    - set $\hat{S}_j \leftarrow \alpha \cdot Q_j \in E(\mathbb{F}_p)$

        ∗ set $rkey_j \leftarrow \mathrm{Dec}(H'(\hat{S}_j), ct_j)$;
          if decryption fails then set $goodkey_j \leftarrow false$ and skip the next step.
        ∗ set $(r_j, adct_j, sh_j) \leftarrow \mathrm{Dec}(rkey_j, rct)$;
          if decryption fails then set $goodkey_j \leftarrow false$, otherwise set $goodkey_j \leftarrow true$.

  – if $goodkey_1 = goodkey_2 = false$ then this is a non-match and the voucher can be ignored.
  – if for $b \in \{1, 2\}$ we have $goodkey_b = true$ but $goodkey_{3-b} = false$ then

    add the tuple $(id, adct_b, sh_b, r_b)$ to the set SHARES.

    This is a match. Recall that sh_b is a Shamir share of *adkey*, and r_b is in the range $\mathcal{R}$ of the detectable hash function.
  – if $goodkey_1 = goodkey_2 = true$ then with high probability the client voucher is invalid and can be ignored.

- *step 2:* let t' be the number of distinct Shamir shares in the set SHARES. If the client is behaving honestly, then t' should be equal to $\left|id(\bar{Y} \cap X) \cup S\right|$. Some of the Shamir shares in SHARES are "real", obtained from a real match, while others are "dummy" due to a synthetic match. The server does not know which is which. When the intersection size is above the threshold, the detectable hash function will be used to identify the real Shamir shares and filter out the dummy ones.
  – let OUTSET be the set of identifiers in SHARES.
    This set will be equal to $id(\bar{Y} \cap X) \cup S \subseteq \mathcal{ID}$.
  – if $t' \leq t$ then output IDLIST and OUTSET and stop.
  – otherwise, $t' > t$. Let $\mathsf{RLIST} \in \mathcal{R}^{\leq t'}$ be a list whose elements are all the distinct detectable hash function values in SHARES (i.e., a list containing the last element from every tuple in SHARES, while eliminating duplicates).
  – run the detection algorithm of the $(s_{\max}, t)$-DHF giving it the list $\mathsf{RLIST} \in \mathcal{R}^{\leq t'}$ as input.
  – if the detection algorithm outputs *fail* then with high probability there are not enough real Shamir shares in SHARES, namely $\left|id(\bar{Y} \cap X) \setminus S\right| \leq t$; output IDLIST and OUTSET and stop.
  – otherwise, the detection algorithm outputs a set $\hat{T} \subseteq [t']$ of at least $t + 1$ indices into RLIST. Let $\mathsf{RLIST}[\hat{T}] \subseteq \mathcal{R}$ be the set of DHF values in RLIST at positions indicated by $\hat{T}$.
  – let SHARES′ be the subset of SHARES that contains all tuples whose last coordinate is in $\mathsf{RLIST}[\hat{T}]$. Then SHARES′ contains at least $t+1$ distinct "real" Shamir shares.
  – use $(t + 1)$ distinct Shamir shares in SHARES′ to reconstruct an $adkey \in \mathcal{K}'$.
  – initialize an empty set OUTSET.
  – for each tuple $(id, adct, sh, r)$ in SHARES′ use *adkey* to decrypt the ciphertext *adct* to obtain

$$ad \leftarrow \mathrm{Dec}(adkey,\ adct).$$

    If decryption fails, then the corresponding voucher is invalid. If decryption succeeds, add the pair (id, ad) to the set OUTSET.

– Let S be the of identifiers in SHARES that are not in SHARES'.

– output the list IDLIST and the sets OUTSET and S.

This completes the description of the protocol. If the algorithm outputs a set OUTSET of pairs (id, ad), but the set contains fewer than $t + 1$ pairs, then either the DHF detection algorithm failed (low probability) or the client sent invalid vouchers.

Correctness. The following theorem shows that when the client and the server honestly follow the protocol, the server learns the required ftPSI-AD output.

Theorem 2 (correctness)**.** *Suppose that the client and the server honestly follow the protocol, and that (i) $H' : E(\mathbb{F}_p) \to \mathcal{K}'$ is a secure key derivation function, (ii)* (Enc, Dec) *is random key robust as defined in* (1) *on page 6, (iii) F is a secure PRF, and (iv)* DHF *is an (s_{max}, t)-DHF. Then the server learns the required ftPSI-AD output with high probability.*

The proof is essentially the same as the proof of Theorem 1. The main difference is that when $\left|id(\bar{Y} \cap X) \setminus S\right| \leq t$, the server should output the list of identifiers $id(\bar{Y} \cap X) \cup S$. Indeed, in this case, by construction, the identifiers in the set S are indistinguishable from the identifiers in $id(\bar{Y} \cap X)$, and therefore will be included in the list of identifiers OUTSET that the server outputs, as required. Moreover, when $\left|id(\bar{Y} \cap X) \setminus S\right| > t$ and $\left|S\right| \leq s_{\max}$, the detection algorithm of the DHF will identify the real Shamir shares in SHARES, and consequently the server will output the required $\mathsf{OUTSET} = \bar{Y}[id(\bar{Y} \cap X) \setminus S]_{\{id,ad\}}$.

Remarks. Remarks 4-7 from the previous section apply to this protocol as well. In addition, we make the following remark about maliciously injected synthetics.

Remark 8 (Synthetic injection)*.* Synthetic matches added by the client provide a fuzzy intersection size to the server when the intersection size is below the threshold. However, consider a client that deliberately chooses the set S of synthetics to be larger than $s_{\max}$, thereby violating the upper bound on S. Then the DHF detection algorithm may fail even when the number of "real" Shamir shares obtained by the server is above the threshold. The server will then obtain an incorrect ftPSI-AD output. As explained at the beginning of the document, the system is not required to ensure *correctness* against a misbehaving client because there are many ways in which a malicious client can refuse to participate. Nevertheless, we note that the server can detect an excess number of synthetics by the client and flag this type of failure. Moreover, for the DHF described in the next section, it is possible to locate the real Shamir shares even if there are more than $s_{\max}$ dummy shares, albeit using a different and more costly detection algorithm. See Remark 9.

4.3 Constructing a detectable hash function

To complete the description of the protocol we need an s-detectable t-wise independent hash function $\text{DHF} : \mathcal{K} \times \mathcal{X} \to \mathcal{R}$, or an (s, t)-DHF. Recall that

- s is an upper bound on the size of the set S, and
- $t + 1$ is the threshold number of matches before the server learns any associated data.

For the rest of this section let ℓ be a 64-bit prime.

Construction. The function DHF : $\mathcal{K} \times \mathcal{X} \to \mathcal{R}$ is defined as follows:

- $\mathcal{K} := \mathbb{F}_\ell^{s\times t}, \quad \mathcal{X} := \mathbb{F}_\ell, \quad \mathcal{R} := \mathbb{F}_\ell^{s+1}$.
- We treat a key $k \in \mathcal{K}$ as a sequence of s polynomials $p_1, \dots, p_s \in \mathbb{F}_\ell[X]$ each of degree at most $t-1$. Each row of the matrix $k \in \mathcal{K}$ is one polynomial.
- For $k \in \mathcal{K}$ and $x_0 \in \mathcal{X}$ define

  $$\mathrm{DHF}(k, x_0) := \big(x_0, p_1(x_0), \dots, p_s(x_0)\big) \in \mathbb{F}_\ell^{s+1}$$

  We will treat the output $\mathrm{DHF}(k, x_0)$ as a column vector in $\mathbb{F}_\ell^{s+1}$.

A standard argument shows that the function DHF is weak t-wise independent: for random $k \xleftarrow{\$} \mathcal{K}$ and random $x_1, \dots, x_t \xleftarrow{\$} \mathbb{F}_\ell$, the distribution $\big(\mathrm{DHF}(k, x_1), \dots, \mathrm{DHF}(k, x_t)\big)$ is uniform in $\mathcal{R}^t$.

The detection algorithm. The detection algorithm is given a matrix of m columns, where each column is an element of $\mathcal{R} = \mathbb{F}_\ell^{s+1}$. It needs to detect those columns that are an output of the DHF, assuming there are at least $\max(t+1, m-s)$ such columns. Recall that this means that there are at least $t+1$ DHF columns and at most s random columns. This problem is related to decoding the interleaved Reed-Solomon code under random noise. Our decoder uses a technique first described by Coppersmith and Sudan [CS03].

- *step 1:* expand each of the m given column vectors $\mathbf{r} \leftarrow (x_0, r_1, \dots, r_s) \in \mathbb{F}_\ell^{s+1}$ into a column vector $\mathbf{r}' \in \mathbb{F}_\ell^{s+t}$ by setting

  $$\mathbf{r}' \leftarrow (1, x_0, x_0^2, \dots, x_0^{t-1}, r_1, \ \dots, r_s) \in \mathbb{F}_\ell^{s+t}.$$

- *step 2:* let $M \in \mathbb{F}_\ell^{(s+t)\times m}$ be the resulting matrix after expansion. Every row of M contains at least $t+1$ evaluations of some polynomial f of degree at most $t-1$. Hence, the $t+1$ correct DHF columns of M must be linearly dependent.
- *step 3:* if $\dim(\mathit{kernel}(M)) = 0$ then output *fail* and stop (here $\mathit{kernel}(M)$ is the right kernel of M); otherwise:
  - let $\mathbf{w} \in \mathbb{F}_\ell^m$ be a vector in the kernel of M, namely $M \cdot \mathbf{w} = 0$.
  - let $Z \subseteq [m]$ be the set of indices where $\mathbf{w}$ is non-zero.
  - let $M[Z] \in \mathbb{F}_\ell^{(s+t)\times|Z|}$ be the sub-matrix of M restricted to the columns in Z.
  - output the set of indices $\hat{Z} \subseteq [m]$ where $i \in \hat{Z}$ iff column number i of M is in the linear span of the columns in $M[Z]$. Note that $\hat{Z}$ is a superset of Z.

This completes the description of the detection algorithm.

To show that the detection algorithm works correctly, consider the case where M contains at least $t+1$ DHF columns. Let $\hat{M}$ be the sub-matrix of M containing all the expanded random columns and exactly t of the expanded DHF columns. This is a matrix containing at most $s+t$ columns and exactly $s+t$ rows. Moreover, the top t rows are linearly independent, and the bottom s rows are truly random. The columns of such a matrix are linearly independent with probability about $1-(1/\ell)$, assuming $\ell \gg s+t$. The remaining columns of M are in the linear span of the t DHF columns of $\hat{M}$. Therefore, the vector $\mathbf{w}$ can only be non-zero at positions that correspond to DHF columns. Moreover, with high probability, these columns will span the linear space of DHF columns. Hence, with high probability, the algorithm outputs exactly the indices of the columns in M that are DHF columns, as required.

Remark 9. The set S of synthetics grows in size over time. In the unlikely event that its size exceeds s, so that we end up with more than s synthetics, we can use an extended detection procedure that follows the method of Coppersmith and Sudan [CS03]. This method applies when the number of DHF columns is at least $t+d$, for a suitable choice of $d \geq 1$.

4.4 Security

We now turn to analyzing the security of the protocol from Section 4.2. We show that the scheme provides the following security properties:

- *Privacy for the server:* informally, a malicious client learns $\lvert X \rvert$, but nothing else about the server's dataset X. The proof relies on the Decision Diffie-Hellman (DDH) assumption in $E(\mathbb{F}_p)$ where $H : \mathcal{U} \to E(\mathbb{F}_p) \smallsetminus \{\mathcal{O}\}$ is modeled as a random oracle. Recall that the server is not guaranteed to obtain the correct ftPSI-AD output, and this is fine as discussed in Section 2.3. Indeed, a malicious client can withhold elements from its input $\bar{Y}$, or report fictitious matches, or even choose not to participate in the first place.

- *Privacy for the client:* informally, a malicious server who misbehaves learns nothing about the client's dataset, beyond the required output of the ftPSI-AD functionality.

In this section we define these properties more precisely and sketch the security proofs. Generally speaking, there are two approaches to defining security for a cryptographic protocol:

- *A simulation based definition:* This style of definition makes it possible to rigorously state the two informal requirements above. Moreover, if one follows a universal composability (UC) framework [Can00] then it is possible to deduce certain security properties when the protocol is composed, or runs concurrently, with other UC protocols as part of a larger system. We refer to [Lin16] for a survey of this definition style.

- *A game based definition:* This style of definition shows very concretely that the adversary cannot mount certain attacks. The resulting security statements can be used to set the security parameter to ensure real world security. However, for a complex cryptographic task, one has to separately justify why the security games capture the desired security properties. We refer to [BR06] for a detailed explanation of this definition style.

In some cases it is possible to prove that a game based definition is equivalent to a simulation based definition, after which one typically only uses the game based definition. In other cases, however, establishing equivalence may be more difficult, in which case one might opt to use a simulation based definition.

In this section we give a simulation based definition to formalize the two informal properties listed at the beginning of the section. We then state the security theorems in terms of these definitions. To keep the discussion accessible, we will not use a formal logical framework to give the definition, but instead write the definition in plain english. Future work can translate the text into a logical framework suitable for a mechanized UC proof system such as EasyUC [CSV19].

For readers who are more comfortable with game based definitions, we point to an excellent security analysis by Bellare [Bel21] that provides a game based proof of security for the core tPSI-AD protocol in Section 4.1. We are fortunate to have two types of analysis for the protocol. Both analyses reach the same conclusions.

In what follows we adapt a simplified universal composability (UC) framework due to Canetti, Cohen, and Lindel [CCL15]. We give a high level description of the framework and refer to [CCL15] for the low level details. Because we are only concerned with privacy of the inputs, we slightly modify the framework to remove the correctness guarantee for the outputs. One complication is that our security analysis requires a programmable random oracle, and this raises a number of definitional challenges in the UC framework, as discussed in [CJS14, CDG$^+$18]. We refer to [Sho20, §1.2] for a recent discussion of the implications of these issues and their impact on composability. We model the random oracle as done in [Sho20] and in several other papers. We are primarily using this setup to analyze the standalone protocol.

The ideal functionality. The first component of a simulation based definition is an ideal functionality $\mathcal{F}$. This $\mathcal{F}$ defines what each party in the system is meant to learn when the protocol completes. Figure 1 gives the ideal functionality for fuzzy threshold private set intersection with associated data (ftPSI-AD). Note that we deliberately ignore the fact that a small number of elements from X may be dropped from the server's input as discussed in Remark 3.

The ideal world/real world paradigm. A simulation based definition for a protocol Π defines two worlds: a real world and an ideal world.

- In the real world, honest parties follow the specified protocol Π and interact with corrupt parties who are controlled by an adversary $\mathcal{A}$.
- In the ideal world, an adversary called a **simulator**, denoted by Sim, interacts with the ideal functionality, and controls the same corrupt parties as in the real world. The ideal world is set up to ensure that Sim learns nothing beyond what the ideal functionality reveals to the corrupt parties.

The protocol Π is said to be a secure emulation of the ideal functionality if the ideal world is "indistinguishable" from the real world. This implies that all the security properties that hold in the ideal world must also hold in the real world. In particular, the real world adversary can learn nothing beyond what is revealed by the ideal functionality.

To make this indistinguishability concept precise we introduce yet another adversary called the **environment**, denoted by $\mathcal{Z}$. The environment interacts with adversary $\mathcal{A}$ in the real world, and interacts with adversary Sim in the ideal world. Its goal is to distinguish these two interactions. As we will see, if $\mathcal{Z}$ cannot tell which world it is in, then we say

Parameters known to all parties:

- two parties: server and client,
- B is the maximum set size for the server and client,
- t is the threshold,
- $s_{\max}$ is the maximum size of the set S of synthetics,
- all associated data values ad in $\mathcal{D}$ have the same public length.

The functionality $\mathcal{F}$:

- Wait for input $X = \{x_1, x_2, \ldots\}$ from the server;
  abort if the server is corrupt and $|X| > B$.
- Send $|X|$ to the client; abort if the client is corrupt and aborts.
- Wait for input $\bar{Y} \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m$ and $S \subseteq id(\bar{Y})$ from the client;
  abort if the client is corrupt and ($m > B$ or $|S| > s_{\max}$).
- Send $\bar{Y}_{id}$ to the server.
- If $|id(\bar{Y} \cap X) \smallsetminus S| > t$:
  send $\bar{Y}[id(\bar{Y} \cap X) \smallsetminus S]_{\{id,ad\}}$ and S to the server,
  otherwise:
  send $id(\bar{Y} \cap X) \cup S$ to the server.

Figure 1: The ideal functionality $\mathcal{F}$ for ftPSI-AD

that the ideal world and the real world are indistinguishable. We then say that protocol Π is a *secure emulation* of the ideal functionality. We note that in the definition given below, the adversary $\mathcal{A}$ is working together with $\mathcal{Z}$ to help $\mathcal{Z}$ distinguish the ideal world from the real world, while the simulator Sim is trying to make the two worlds look the same.

Let us define the parties at work in the real and ideal worlds in more detail, as applied to our settings. In our description we consider a situation where the client is honest and the server is malicious. Figure 3 shows the parties at work in this case and how they interact. The symmetric situation, where the server is honest and the client is malicious, is defined analogously and shown in Figure 4. For context, Figure 2 gives a schematic of the protocol Π for ftPSI-AD from Section 4.2.

- The parties at work in the real world (honest client and malicious server, Figure 3a):
  - $\mathsf{C}^H_{\text{real}}$: an honest real world client. $\mathsf{C}^H_{\text{real}}$ is given its input by the environment $\mathcal{Z}$, and honestly follows the protocol Π by sending messages to the malicious server. In addition, $\mathsf{C}^H_{\text{real}}$ can query the random oracle H.
  - $\mathcal{A}^H_{\text{s}}$: a real world server adversary. The adversary interacts with $\mathsf{C}^H_{\text{real}}$ and with the environment $\mathcal{Z}$ by sending messages to and receiving messages from both parties. In addition, $\mathcal{A}^H_{\text{s}}$ can query the random oracle H.
  - $\mathcal{Z}$: the environment. $\mathcal{Z}$ provides $\mathsf{C}^H_{\text{real}}$ with its input and interacts with $\mathcal{A}^H_{\text{s}}$ by sending messages to $\mathcal{A}^H_{\text{s}}$ and receiving messages from $\mathcal{A}^H_{\text{s}}$. Eventually $\mathcal{Z}$ outputs 0 or 1. Note that $\mathcal{Z}$ can query the random oracle H by asking $\mathcal{A}^H_{\text{s}}$ to issue the query on its behalf and send back the response.

Parameters known to all parties:
- B is the maximum set size for the server and client,
- t is the threshold,
- $s_{\max}$ is the maximum size of the set S of synthetics.

inputs: the server S has $X \subseteq \mathcal{U}$ where $\left|X\right| \leq B$,
the client C has $\bar{Y} \in (\mathcal{U} \times \mathcal{ID} \times \mathcal{D})^m$ and $S \subseteq id(\bar{Y})$,
where $m \leq B$ and $\left|S\right| \leq s_{\max}$.

- *server → client:* The server moves first by preparing $\mathsf{pdata} \leftarrow (L, P_1, \dots, P_{n'})$ described in Step 4 of the server setup procedure, and sends pdata to the client.
- *client → server:* the client responds by sending m vouchers back to the server, as described in the voucher generation procedure.
- *server output:* the server outputs $\bar{Y}_{id}$ and either $(\bar{Y}[id(\bar{Y} \cap X) \smallsetminus S]_{\{id,ad\}}$ and $S)$ or $id(\bar{Y} \cap X) \cup S$, as described in the voucher processing procedure.

Figure 2: A schematic of protocol Π for ftPSI-AD

Both $\mathsf{C}^H_{\text{real}}$ and $\mathcal{A}^H_{\text{s}}$ have access to the same random oracle H. (Technically, H is made available to them as a so called ideal functionality.)

- The parties at work in the ideal world (honest client and malicious server, Figure 3b):
  - $\mathsf{C}_{\text{ideal}}$: an honest ideal world client. The client is given its input by the environment $\mathcal{Z}$, and simply forwards its input to the ideal functionality $\mathcal{F}$.
  - Sim_{s}: a server simulator. This Sim_{s} sends some set X to the ideal functionality $\mathcal{F}$ and receives back its output from $\mathcal{F}$. Sim_{s} also interacts with $\mathcal{Z}$. Its goal is to make its interaction with $\mathcal{Z}$ look the same as $\mathcal{Z}$'s interaction with $\mathcal{A}^H_{\text{s}}$ in the real world.
  - $\mathcal{F}$: the ideal functionality. It accepts inputs from $\mathsf{C}_{\text{ideal}}$ and Sim_{s}, and sends the resulting server output to Sim_{s}.
  - $\mathcal{Z}$: the environment. $\mathcal{Z}$ sends an input to $\mathsf{C}_{\text{ideal}}$ and interacts with Sim_{s} as in the real world. Eventually $\mathcal{Z}$ outputs 0 or 1. $\mathcal{Z}$ can query the random oracle H by sending the query to Sim_{s} as it would in the real world.

In the real world execution the adversary $\mathcal{A}^H_{\text{s}}$ also controls the order in which transmitted messages are delivered to parties. However, the communication pattern in the ftPSI-AD protocol Π is sufficiently simple that this power does not help the adversary. Usually $\mathcal{Z}$ is also given the output of the honest parties, but since we do not require output correctness, we do not include that here.

It should be clear that the ideal world simulators Sim_{s} and Sim_{c}, shown in Figures 3 and 4, learn nothing beyond what is revealed by the ideal functionality. Therefore, if the environment $\mathcal{Z}$ cannot distinguish between the ideal and real worlds, then the same must hold for the real world adversaries $\mathcal{A}_{\text{s}}$ and $\mathcal{A}_{\text{c}}$, respectively. We use this framework to define

$$\begin{tikzcd}
\mathsf{C}^H_{\text{real}} \arrow[r, shift right, "\text{vouchers}"'] & \mathcal{A}^H_{\text{s}} \arrow[l, shift right, "\textsf{pdata}"'] \arrow[r, leftrightarrow] & \mathcal{Z} \arrow[ll, dashed, bend right, "\text{client input: } \bar{Y}, S"'] \arrow[r, "0/1"] & {}
\end{tikzcd}$$

(a) The real world with an honest client C_{real} and a malicious server $\mathcal{A}^H_{\text{s}}$

$$\begin{tikzcd}
\mathsf{C}_{\text{ideal}} \arrow[r, "\bar{Y}, S"] & \mathcal{F} \arrow[r, shift right, "\text{server output}"'] & \mathrm{Sim}_{\text{s}} \arrow[l, shift right, "X"'] \arrow[r, leftrightarrow] & \mathcal{Z} \arrow[lll, dashed, bend right, "\text{client input: } \bar{Y}, S"'] \arrow[r, "0/1"] & {}
\end{tikzcd}$$

(b) The ideal world with an honest client $\mathsf{C}_{\text{ideal}}$ and simulator Sim_{s}

Figure 3: Communication in the ideal and real worlds: honest client, malicious server

$$\begin{tikzcd}
\mathsf{S}^H_{\text{real}} \arrow[r, "\textsf{pdata}"] & \mathcal{A}^H_{\text{c}} \arrow[r, leftrightarrow] & \mathcal{Z} \arrow[ll, dashed, bend right, "\text{server input: } X"'] \arrow[r, "0/1"] & {}
\end{tikzcd}$$

(a) The real world with an honest server S_{real} and a malicious client $\mathcal{A}^H_{\text{c}}$

$$\begin{tikzcd}
\mathsf{S}_{\text{ideal}} \arrow[r, "X"] & \mathcal{F} \arrow[r, "|X|"] & \mathrm{Sim}_{\text{c}} \arrow[r, leftrightarrow] & \mathcal{Z} \arrow[lll, dashed, bend right, "\text{server input: } X"'] \arrow[r, "0/1"] & {}
\end{tikzcd}$$

(b) The ideal world with an honest server $\mathsf{S}_{\text{ideal}}$ and simulator Sim_{c}

Figure 4: Communication in the ideal and real worlds: honest server, malicious client

and prove privacy for an honest server against a malicious client, and privacy for an honest client against a malicious server.

4.4.1 Privacy for the server's dataset X against a malicious client

First, let's show that a malicious client $\mathcal{A}_{\mathrm{c}}^{H}$ that interacts with an honest server with input X, learns nothing about X other than its size.

For an adversary $\mathcal{A}_{\mathrm{c}}^{H}$, a simulator $\mathrm{Sim}_{\mathrm{c}}$, and an environment $\mathcal{Z}$, define the following two random variables:

- $\mathrm{REAL}_{\Pi,\mathcal{A}_{\mathrm{c}}^{H},\mathcal{Z}}$ is the random variable defined as the output of $\mathcal{Z}$ in a real world execution after interacting with the malicious client $\mathcal{A}_{\mathrm{c}}^{H}$ as in Figure 4a.
- $\mathrm{IDEAL}_{\mathcal{F},\mathrm{Sim}_{\mathrm{c}},\mathcal{Z}}$ is the random variable defined as the output of $\mathcal{Z}$ in an ideal world execution after interacting with the simulator $\mathrm{Sim}_{\mathrm{c}}$ as in Figure 4b.

Definition 2. We say that a protocol Π for $\mathcal{F}$ is **server private** if for every efficient adversary $\mathcal{A}_{\mathrm{c}}$, there exists an efficient adversary $\mathrm{Sim}_{\mathrm{c}}$, such that for every efficient environment $\mathcal{Z}$,

$$\left|\Pr\left[\mathrm{REAL}_{\Pi,\mathcal{A}_{\mathrm{c}}^{H},\mathcal{Z}} = 1\right] - \Pr\left[\mathrm{IDEAL}_{\mathcal{F},\mathrm{Sim}_{\mathrm{c}},\mathcal{Z}} = 1\right]\right| \le \epsilon$$

for some negligible ϵ.

The terms "efficient" and "negligible" are usually interpreted asymptotically with respect to a security parameter. Canetti [Can00, Def. 16] suggests a way to interpret these terms as concrete values, but we will not do that here.

Theorem 3. *Protocol Π in Figure 2 is server private, assuming $H : \mathcal{U} \to E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ is modeled as a random oracle, and Decision Diffie-Hellman (DDH) holds in $E(\mathbb{F}_p)$.*

Proof Sketch. Let $\mathcal{A}_{\mathrm{c}}^{H}$ be an efficient adversary as in Figure 4a. Let us define the following adversary $\mathrm{Sim}_{\mathrm{c}}$ that interacts with an external environment $\mathcal{Z}$ as in Figure 4b. Adversary $\mathrm{Sim}_{\mathrm{c}}$ works as follows:

1. $\mathrm{Sim}_{\mathrm{c}}$ runs adversary $\mathcal{A}_{\mathrm{c}}^{H}$ as follows:
   - When $\mathrm{Sim}_{\mathrm{c}}$ receives a message from $\mathcal{Z}$ it forwards the message to $\mathcal{A}_{\mathrm{c}}^{H}$. When $\mathrm{Sim}_{\mathrm{c}}$ receives a message from $\mathcal{A}_{\mathrm{c}}^{H}$ intended for $\mathcal{Z}$ it forwards the message to $\mathcal{Z}$. If $\mathcal{Z}$ or $\mathcal{A}_{\mathrm{c}}^{H}$ abort then so does $\mathrm{Sim}_{\mathrm{c}}$.
   - $\mathrm{Sim}_{\mathrm{c}}$ emulates a random oracle $H : \mathcal{U} \to E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ for $\mathcal{A}_{\mathrm{c}}^{H}$. When $\mathcal{A}_{\mathrm{c}}^{H}$ queries H at a point $x \in \mathcal{U}$, adversary $\mathrm{Sim}_{\mathrm{c}}$ responds consistently: it samples $R \xleftarrow{\$} E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ and sends R to $\mathcal{A}_{\mathrm{c}}^{H}$. Consistently means that if $\mathcal{A}_{\mathrm{c}}^{H}$ issues the same query $H(x)$ again, then $\mathrm{Sim}_{\mathrm{c}}$ responds with the same R as before.
2. If $\mathcal{Z}$ does not abort then eventually $\mathrm{Sim}_{\mathrm{c}}$ receives from the functionality $\mathcal{F}$ the cardinality n of the honest server's set X.
3. $\mathrm{Sim}_{\mathrm{c}}$ sets $n' \leftarrow (1+\epsilon')\cdot n$ and chooses $n'+1$ random points $L, P_1, \ldots, P_{n'}$ in $E(\mathbb{F}_p)\setminus\{\mathcal{O}\}$. It also chooses random nonces to define the hash functions $h_1, h_2 : \mathcal{U} \to [n']$.

4. $\mathrm{Sim_c}$ sends $\mathsf{pdata} \leftarrow (L, P_1, \ldots, P_{n'})$ and the nonces to $\mathcal{A}_\mathrm{c}^H$.

This completes the description of $\mathrm{Sim_c}$.

We claim that $\Pr\big[\textsc{real}_{\Pi,\mathcal{A}_\mathrm{c}^H,\mathcal{Z}} = 1\big]$, the probability that $\mathcal{Z}$ outputs 1 in the ideal world, is close to $\Pr\big[\textsc{ideal}_{\mathcal{F},\mathrm{Sim_c},\mathcal{Z}} = 1\big]$. To see why, we argue that the input that $\mathrm{Sim_c}$ gives to $\mathcal{A}_\mathrm{c}^H$ in Step 4 is indistinguishable from the input that $\mathcal{A}_\mathrm{c}^H$ receives from $\mathsf{S}_\mathrm{real}^H$ in a real world execution. It follows that the resulting messages to $\mathcal{Z}$ in both worlds are indistinguishable, and therefore $\mathcal{Z}$ will behave the same in both worlds.

Let us argue that the input given to $\mathcal{A}_\mathrm{c}^H$ from $\mathrm{Sim_c}$ is indistinguishable from the input given to $\mathcal{A}_\mathrm{c}^H$ by $\mathsf{S}_\mathrm{real}^H$. Considering how the input to $\mathcal{A}_\mathrm{c}^H$ is formed in both worlds we see that a distinguisher $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ must be able to distinguish between the following two experiments defined with respect to a random oracle H:

real world experiment	ideal world experiment
$(X, \mathit{state}) \xleftarrow{\$} \mathcal{B}_1^H()$	$(X, \mathit{state}) \xleftarrow{\$} \mathcal{B}_1^H()$
where $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{U}$	where $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{U}$
$\alpha \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}, \quad L \leftarrow \alpha G$	$L, P_1, \ldots, P_n \xleftarrow{\$} E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$
$\mathsf{pdata} \leftarrow \big(L,\ \alpha H(x_1), \ldots, \alpha H(x_n)\big)$	$\mathsf{pdata} \leftarrow \big(L,\ P_1, \ldots, P_n\big)$
output $\mathcal{B}_2^H(\mathsf{pdata}, \mathit{state}) \in \{0,1\}$	output $\mathcal{B}_2^H(\mathsf{pdata}, \mathit{state}) \in \{0,1\}$

The first line in each experiment corresponds to the environment $\mathcal{Z}$ generating a set X for the honest server. The last line in each experiment corresponds to $\mathcal{A}_\mathrm{c}^H$ being given the input pdata in each world followed by $\mathcal{Z}$ outputting 0 or 1. Note that $\mathcal{B}_2$ knows the set X, since it can be communicated to $\mathcal{B}_2$ through the *state* variable from $\mathcal{B}_1$.

Let W_{real} be the event that $\mathcal{B}_2$ outputs 1 in the left experiment, and W_{ideal} be the event that $\mathcal{B}_2$ outputs 1 in the right experiment. A standard argument shows that if DDH holds in $E(\mathbb{F}_p)$, and H is a programmable random oracle, then the quantity

$$\Big|\Pr[W_{real}] - \Pr[W_{ideal}]\Big|$$

is negligible for all efficient distinguishers $\mathcal{B}$. The same remains true even if pdata is expanded to include additional uniformly sampled points in $E(\mathbb{F}_p)$ at a number of set locations. We conclude that the input given to $\mathcal{A}_\mathrm{c}^H$ by $\mathrm{Sim_c}$ is indistinguishable from the input given to $\mathcal{A}_\mathrm{c}^H$ in a real world interaction, as required. □

4.4.2 Privacy for the client against a malicious server

Next, let's show that a malicious server $\mathcal{A}_\mathrm{s}^H$ that interacts with an honest client with input $(\bar{Y}, S)$, learns nothing other than what is revealed by the ideal functionality.

For an adversary $\mathcal{A}_\mathrm{s}^H$, a simulator $\mathrm{Sim_s}$, and an environment $\mathcal{Z}$, define the following two random variables:

- $\textsc{real}_{\Pi,\mathcal{A}_\mathrm{s}^H,\mathcal{Z}}$ is the random variable defined as the output of $\mathcal{Z}$ in a real world execution after interacting with the malicious server $\mathcal{A}_\mathrm{s}^H$ as in Figure 3a.
- $\textsc{ideal}_{\mathcal{F},\mathrm{Sim_s},\mathcal{Z}}$ is the random variable defined as the output of $\mathcal{Z}$ in an ideal world execution after interacting with the simulator $\mathrm{Sim_s}$ as in Figure 3b.

Definition 3. We say that a protocol Π for $\mathcal{F}$ is **client private** if for every efficient adversary $\mathcal{A}_\mathrm{s}$, there exists an efficient adversary Sim_s, such that for every efficient environment $\mathcal{Z}$,

$$\left|\Pr\left[\mathrm{REAL}_{\Pi,\mathcal{A}_\mathrm{s}^H,\mathcal{Z}} = 1\right] - \Pr\left[\mathrm{IDEAL}_{\mathcal{F},\mathrm{Sim}_\mathrm{s},\mathcal{Z}} = 1\right]\right| \le \epsilon$$

for some negligible ϵ.

As in the previous section, the terms "efficient" and "negligible" are interpreted asymptotically with respect to a security parameter.

In what follows, we use the following notation (i) B is an upper bound on the size of the inputs X and $\bar{Y}$, (ii) q is the size of $E(\mathbb{F}_p)$, and (iii) Q_{ro} is an upper bound on the number of random oracle calls made by $\mathcal{A}_\mathrm{s}^H$.

Theorem 4. *Protocol* Π *in Figure 2 is client private, assuming* $H : \mathcal{U} \to E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ *is modeled as a random oracle,* $H' : E(\mathbb{F}_p) \to \mathcal{K}'$ *is a secure key derivation function,* (Enc, Dec) *is IND\$-CPA secure,* DHF *is weak t-wise independent, F is a secure PRF, and* $2(B + Q_{ro})/(q-1)$ *is negligible.*

Proof Sketch. Let $\mathcal{A}_\mathrm{s}^H$ be an efficient adversary as in Figure 3a. Let us define the following adversary Sim_s that interacts with an external environment $\mathcal{Z}$ as in Figure 3b. Adversary Sim_s runs $\mathcal{A}_\mathrm{s}^H$ and interacts with it as follows:

1. When Sim_s receives a message from $\mathcal{Z}$ it forwards the message to $\mathcal{A}_\mathrm{s}^H$. When Sim_s receives a message from $\mathcal{A}_\mathrm{s}^H$ intended for $\mathcal{Z}$ it forwards the message to $\mathcal{Z}$.

2. Sim_s emulates a random oracle $H : \mathcal{U} \to E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ for $\mathcal{A}_\mathrm{s}^H$ using the following procedure. First, initialize an empty list L_H. Then, for $j = 1, \ldots, Q_{ro}$, when $\mathcal{A}_\mathrm{s}^H$ issues a query for $H(x_j)$, where $x_j \in \mathcal{U}$, do:

   - if there is a triple (x_j, β, R) in L_H then send R to $\mathcal{A}_\mathrm{s}^H$ (namely, $H(x_j) = R$);
   - otherwise, do: $\beta_j \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$, $R_j \leftarrow \beta_i \cdot G \in E(\mathbb{F}_p)$,
     send R_j to $\mathcal{A}_\mathrm{s}^H$, and append (x_j, β_j, R_j) to the list L_H.

3. Eventually, $\mathcal{A}_\mathrm{s}^H$ either aborts or outputs $\mathsf{pdata} \leftarrow (L, P_1, \ldots, P_{n'})$ along with two hash functions $h_1, h_2 : \mathcal{U} \to [n']$. If $\mathcal{A}_\mathrm{s}^H$ aborts, then Sim_s aborts and terminates.

4. Sim_s checks that $(L, P_1, \ldots, P_{n'})$ are non-zero points in $E(\mathbb{F}_p)$ and all are distinct. If not, then Sim_s runs $\mathcal{A}_\mathrm{s}^H$ until it terminates, and terminates.

5. *Input extraction:* Sim_s extracts a set $X \subseteq \mathcal{U}$ from the pdata output by $\mathcal{A}_\mathrm{s}^H$.
   Sim_s sets $X \leftarrow \emptyset$ and does:
   For each triple (x, β, R) in L_H:
   if $P_{h_1(x)} = \beta \cdot L$ or $P_{h_2(x)} = \beta \cdot L$ then add $x \in \mathcal{U}$ to the set X.

   To give some intuition as to why an $x \in \mathcal{U}$ that satisfies the condition should be added to X, recall that every triple (x, β, R) in L_H satisfies $R = \beta \cdot G$ and defines $H(x) = R$. Then, if $P_{h_1(x)} = \beta \cdot L$ and $L = \alpha \cdot G$, then $P_{h_1(x)} = \beta\alpha \cdot G = \alpha \cdot R$. Therefore, for this $x \in \mathcal{U}$ we have $P_{h_1(x)} = \alpha \cdot H(x)$. This implies that $x \in \mathcal{U}$ is part of the server's set X. The same applies if $P_{h_2(x)} = \beta \cdot L$.

6. $\mathrm{Sim}_{\mathrm{s}}$ sends X to the functionality $\mathcal{F}$.

7. $\mathcal{A}_{\mathrm{s}}^{H}$ may continue to query H and interact with $\mathcal{Z}$. Eventually, $\mathcal{Z}$ sends $(\bar{Y}, S)$ to $\mathsf{C}_{\mathrm{ideal}}$ who forwards it to $\mathcal{F}$ as in Figure 3b. Subsequently $\mathcal{F}$ sends to $\mathrm{Sim}_{\mathrm{s}}$ the list $\bar{Y}_{id}$ and one of two things:

    - *type (i) output:* $\bar{Y}[id(\bar{Y} \cap X) \setminus S]_{\{id,ad\}} \subseteq (\mathcal{ID} \times \mathcal{D})$ and $S \subseteq id(\bar{Y})$, or
    - *type (ii) output:* $id(\bar{Y} \cap X) \cup S \subseteq id(\bar{Y})$.

    Let $m := \left|\bar{Y}_{id}\right|$. $\mathrm{Sim}_{\mathrm{s}}$ now needs to generate m vouchers to give to $\mathcal{A}_{\mathrm{s}}$. This mechanically follows the steps of the protocol.

8. First, $\mathrm{Sim}_{\mathrm{s}}$ does the following:

    - choose a random key $hkey \xleftarrow{\$} \mathcal{K}$ for a $(s_{\max}, t)$ detectable hash function DHF : $\mathcal{K} \times \mathcal{X} \to \mathcal{R}$. Here t is the ftPSI-AD threshold and $s_{\max}$ is an upper bound on the size of the synthetic set S.
    - choose a random key $adkey \xleftarrow{\$} \mathcal{K}'$ for the encryption scheme (Enc, Dec).
    - choose a random key $fkey \xleftarrow{\$} \mathcal{K}''$ for the PRF $F : \mathcal{K}'' \times \mathcal{ID} \to \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$.
    - Initialize a threshold Shamir secret sharing for $adkey$, so that $t+1$ or more shares are needed to reconstruct $adkey$.

9. Next, for each id in the list $\bar{Y}_{id}$, the simulator $\mathrm{Sim}_{\mathrm{s}}$ generates a voucher as follows:

    - *Preparation:*
        - Choose two random keys $ckey, rkey$ in $\mathcal{K}'$.
        - compute $(x, z, x', r_1) \leftarrow F(fkey, id) \in \mathbb{F}_{\mathrm{Sh}}^2 \times \mathcal{X} \times \mathcal{R}$.
        - Generate a threshold Shamir secret share $sh \in \mathbb{F}_{\mathrm{Sh}}^2$ of $adkey$, where $t+1$ shares are needed to reconstruct $adkey$. The x-coordinate of sh is set to $x \in \mathbb{F}_{\mathrm{Sh}}$ from the previous step.
        - Set $dummy \leftarrow (x, z) \in \mathbb{F}_{\mathrm{Sh}}^2$, a dummy Shamir secret share.
        - Compute $r_2 \leftarrow \mathrm{DHF}(hkey, x') \in \mathcal{R}$.
        - set

        $$adct_1 \xleftarrow{\$} \mathrm{Enc}(ckey, 000\ldots00) \quad \text{and} \quad rct_1 \xleftarrow{\$} \mathrm{Enc}\big(rkey, (r_1, adct_1, dummy)\big).$$

        Note: $adct_1$ should be the same length as $adct$ in all vouchers.
    - case (i): $id \notin id(\bar{Y} \cap X) \cup S$. Choose random Q_1, S_1, Q_2, S_2 in $E(\mathbb{F}_p)$ and set $rct \leftarrow rct_1$.
    - case (ii): $id \in id(\bar{Y} \cap X) \cup S$.
        - choose a random β in $\mathbb{F}_q$, and compute $Q_1 \leftarrow \beta \cdot G$ and $S_1 \leftarrow \beta \cdot L$,
        - choose Q_2 and S_2 independently at random in $E(\mathbb{F}_p)$,
        - if $\mathrm{Sim}_{\mathrm{s}}$ received type (i) output in Step 7 and $id \notin S$:
            then $\mathrm{Sim}_{\mathrm{s}}$ received a pair (id, ad) in Step 7, set

            $$adct \xleftarrow{\$} \mathrm{Enc}\big(adkey, ad\big) \quad \text{and} \quad rct \xleftarrow{\$} \mathrm{Enc}\big(rkey, (r_2, adct, sh)\big),$$

        otherwise: set $rct \leftarrow rct_1$.

- Set $ct_1 \xleftarrow{\$} \text{Enc}(H'(S_1), rkey)$ and $ct_2 \xleftarrow{\$} \text{Enc}(H'(S_2), rkey)$.
  Here ct_1 and ct_2 are an encryption of $rkey$ using the keys $H'(S_1)$ and $H'(S_2)$, respectively.
- Choose a random bit b in $\{1, 2\}$ and set

$$voucher_{id} \leftarrow (id,\ Q_b, ct_b,\ Q_{3-b}, ct_{3-b},\ rct). \tag{2}$$

10. Sim_s sends the list of m vouchers to $\mathcal{A}_\text{s}$ in the order that id's are listed in $\bar{Y}_{id}$.

This completes our description of the simulator Sim_s.

We claim that $\Pr\big[\text{REAL}_{\Pi,\mathcal{A}_\text{s}^H,\mathcal{Z}} = 1\big]$, the probability that $\mathcal{Z}$ outputs 1 in the ideal world, is close to $\Pr\big[\text{IDEAL}_{\mathcal{F},\text{Sim}_\text{s},\mathcal{Z}} = 1\big]$. To see why, we argue that the list of m vouchers that Sim_s gives to $\mathcal{A}_\text{s}^H$ in Step 10 is indistinguishable from the vouchers that $\mathcal{A}_\text{s}^H$ receives from C_real^H in a real world execution. It follows that the resulting messages to $\mathcal{Z}$ in both worlds are indistinguishable, and therefore $\mathcal{Z}$ will behave the same in both worlds.

First, let $L = \alpha \cdot G$ for some (unknown) $0 \neq \alpha \in \mathbb{F}_q$. Second, if $\mathcal{A}_\text{s}^H$ did not query for $H(y)$ then $H(y)$ is (implicitly) defined as a fresh random element in $E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$. Otherwise, $H(y) = R_j = \beta_j G$ for some $(y, \beta_j, R_j) \in L_H$.

The failure event $\mathcal{E}$: Let us define a negligible probability event $\mathcal{E}$ that would cause the simulation to fail. During the ideal world execution with Sim_s, let $\bar{Y}$ be a list of triples sent by $\mathcal{Z}$ to $\mathsf{C}_\text{ideal}^H$, and let pdata be the data output by $\mathcal{A}_\text{s}^H$. Let X be the result of input extraction in Step 5. We say that event $\mathcal{E}$ happened if there exists a triple (y, id, ad) in $\bar{Y}$ such that

$$y \notin X \qquad \text{but} \qquad \alpha H(y) = P_{h_1(y)} \quad \text{or} \quad \alpha H(y) = P_{h_2(y)}. \tag{3}$$

We claim that

$$\Pr[\mathcal{E}] \leq 2(B + Q_{ro})/(q-1) \tag{4}$$

which is negligible by assumption.

To see why (4) holds, consider a triple $tr := (y, id, ad)$ in $\bar{Y}$ that satisfies (3). If $\mathcal{A}_\text{s}^H$ queried for $H(y)$ before the extraction in Step 5, then this triple cannot satisfy (3) because $\alpha H(y) \in \{P_{h_1(y)}, P_{h_2(y)}\}$ implies $y \in X$. If $\mathcal{A}_\text{s}^H$ queried for $H(y)$ in Step 7 then $\mathcal{A}_\text{s}^H$ has already output pdata, so $H(y)$ will satisfy $\alpha H(y) \in \{P_{h_1(y)}, P_{h_2(y)}\}$ with probability at most $2/(q-1)$. Since $\mathcal{A}_\text{s}^H$ makes at most Q_{ro} queries to H, the probability that a y queried in Step 7 causes (3) to hold is at most $2Q_{ro}/(q-1)$. Finally, if $\mathcal{A}_\text{s}^H$ never queried for $H(y)$, then $H(y)$ is independent of $\mathcal{A}_\text{s}^H$ view. Since there are at most B triples in $\bar{Y}$, the probability that such a y causes (3) to hold is at most $2B/(q-1)$. Combining the two bounds gives (4).

To complete the proof we argue that when the failure event $\mathcal{E}$ does not happen, Sim_s generates correct vouchers. Consider a triple (y, id, ad) in $\bar{Y}$. The real world client C_real^H generates a voucher

$$voucher_{id} \leftarrow (id,\ Q_1, ct_1,\ Q_2, ct_2,\ rct)$$

for this triple. We sketch the argument as to why Sim_s generates a voucher sampled from an indistinguishable distribution as this $voucher_{id}$. A complete proof would carry out this argument in a sequence of hybrid steps. Here we give the outline for the argument.

There are three cases:

- case (i): $\alpha \cdot H(y)$ is equal to exactly one of $P_{h_1(y)}$ or $P_{h_2(y)}$ and $id \notin S$. By (4), we know that $y \in X$ with overwhelming probability. Therefore, $\mathrm{Sim_s}$ generates $voucher_{id}$ using Step 9 case (ii).

  We know that in this case the real world client $\mathsf{C}^{H}_{\text{real}}$ generates a voucher containing ct_1 and ct_2 that are an encryption of a random $rkey \in \mathcal{K}'$. Adversary $\mathcal{A}_{\mathrm{s}}$ can compute the decryption key for exactly one of ct_1 or ct_2, and has no other information about the decryption key for the other one. Which of ct_1 and ct_2 is which is chosen at random by the client. When $\mathrm{Sim_s}$ generates a simulated voucher in Step 9 case (ii) it does the same: it ensures that ct_1 and ct_2 are encryptions of a random $rkey \in \mathcal{K}'$, and that the adversary knows the decryption key for exactly one of them, exactly as in the real voucher.

  It remains to argue that the simulated rct is distributed correctly. In the real world, when ct_j for $j \in \{1, 2\}$ is decrypted to reveal $rkey$, this $rkey$ can be used to decrypt rct to reveal $(r, adct, sh)$. Here $r \in \mathcal{R}$ is in the image of the DHF, and $sh \in \mathbb{F}^2_{\mathrm{Sh}}$ is a Shamir share of $adkey$. To argue that the simulated rct generated by $\mathrm{Sim_s}$ is sampled from an indistinguishable distribution, there are two cases:

  Type (i) output in Step 7: in this case the plaintext in the decrypted rct from the real world client is distributed as the decrypted rct from $\mathrm{Sim_s}$. The same applies to $adct$.

  Type (ii) output in Step 7: in this case we know that $\left|id(\bar{Y} \cap X) \setminus S\right| \le t$. For the real world client, the plaintext in the decrypted rct contains $(r, adct, sh)$, where $r = \mathrm{DHF}(hkey, x')$ and $sh \in \mathbb{F}^2_{\mathrm{Sh}}$ is a Shamir share with x-coordinate x. Both x' and x are generated pseudorandomly by $F(fkey, id)$. Now, in the voucher from $\mathrm{Sim_s}$, the plaintext in the decrypted $rct = rct_1$ contains $(r, adct_1, dummy)$, where both r and $dummy \in \mathbb{F}^2_{\mathrm{Sh}}$ are generated pseudorandomly by $F(fkey, id)$. Because (i) DHF is weak t-wise independent, (ii) t Shamir shares are uniform in $\mathbb{F}^2_{\mathrm{Sh}}$, and (iii) F is a secure PRF, the pair $(r, dummy)$ in this decrypted rct_1 is indistinguishable from (r, sh) in the real world client voucher. Moreover, because in the real world the adversary sees at most t distinct Shamir shares of $adkey$, these shares reveal nothing about $adkey$. Recall that $adkey$ is the key used to create the ciphertext $adct$ in the real world voucher. Similarly, in the ideal world, the adversary sees $adct_1$, but has no other information about the key $ckey$ that $\mathrm{Sim_s}$ uses to create this ciphertext. Hence, by the IND\$-CPA property of $(\mathrm{Enc}, \mathrm{Dec})$, the $adct$ generated by the real world client is indistinguishable from $adct_1$ generated by $\mathrm{Sim_s}$.

- case (ii): $\alpha \cdot H(y)$ is equal to none of $P_{h_1(y)}$ or $P_{h_2(y)}$ and $id \notin S$. Then $y \notin X$ and therefore $\mathrm{Sim_s}$ generates $voucher_{id}$ using Step 9 case (i).

  In the real world the adversary receives ct_1 and ct_2 in the voucher generated by the real world client, but has no other information about the keys used to create these ciphertexts. The same holds for the voucher generated by $\mathrm{Sim_s}$ in the ideal world. Therefore, in both worlds the adversary sees rct, but has no other information about the key $rkey$ used to create this ciphertext. Hence, by the IND\$-CPA property of $(\mathrm{Enc}, \mathrm{Dec})$, the rct generated by the real world client is indistinguishable from rct generated by $\mathrm{Sim_s}$.

- case (iii): $id \in S$. In this case, $\mathcal{A}_{\mathrm{s}}$ can decrypt exactly one of ct_1 or ct_2 in the voucher generated by the real world client and recover $rkey$. It can then decrypt rct

to obtain $(r, adct_1, dummy)$, where both r and $dummy$ are generated by $F(fkey, id)$. The voucher generated by $\mathrm{Sim_s}$ in (2) is generated exactly the same way.

Note that crucially, $\alpha \cdot H(y)$ can never be equal to both $P_{h_1(y)}$ and $P_{h_2(y)}$. This is because all of $P_1, \ldots, P_{n'}$ are distinct and $h_1(y) \neq h_2(y)$ by construction. Hence, the three cases above cover all the triples in $\bar{Y}$.

In summary, if event $\mathcal{E}$ does not happen, then in all three cases the vouchers generated by $\mathrm{Sim_s}$ in (2) are sampled from a distribution that is indistinguishable from the distribution from which the real world $\mathsf{C}^H_{\mathrm{real}}$ samples the vouchers. Hence, the data given to $\mathcal{A}^H_{\mathrm{s}}$ in the ideal world is indistinguishable from the data given to it in the real world, and therefore $\mathcal{Z}$ behaves the same in both worlds. $\square$

5 Real world considerations

Multiple client devices. So far we described the client as a single device. In practice a single user may own multiple devices. Client devices may contain an overlapping sequence of triples: the same triple (y, id, ad) may reside on multiple devices, resulting in duplicate triples in the overall client tuple $\bar{Y}$. Every device will send a voucher for all its triples, causing multiple vouchers to be received at the server for a single identifier id. Nevertheless, an id should only count once towards the tPSI-AD threshold, even if the server receives multiple vouchers for this id.

To address this, all the devices belonging to a single user share a small amount of secret state. Specifically, this state consists of the keys $(hkey, adkey, fkey)$ and the Shamir secret sharing polynomial generated during client setup. This way all the vouchers contain Shamir shares of the same $adkey$, even if the vouchers are generated on different client devices. Moreover, in Step 2 of voucher generation, the client generates the x-coordinate of every Shamir share as $F(fkey, id)$. Therefore, duplicate triples will result in identical Shamir shares sent to the server. This is the reason for generating the Shamir x-coordinate pseudorandomly from id. It ensures that every identifier counts once towards the tPSI-AD threshold, even if multiple vouchers are sent to the server for this identifier.

Duplicate images. A user might store multiple variants or near-duplicates of the same image on their client. In our language, this means that a single client could hold two triples (y, id, ad) and (y, id', ad) that have same hash y, but different identifiers. This causes an issue that is addressed outside of the cryptographic protocol. Suppose a user copies a single image from a USB drive onto his or her device. The image will be assigned an identifier id. Later the user copies the same image from the USB drive onto a different client device. The new copy of the image will be assigned a new identifier id' which is likely to be different from id. Because the two copies have different identifiers they will count twice towards the tPSI-AD threshold. In particular, the two triples will cause two distinct Shamir shares to be sent to the sever, even though they correspond to the same semantic image. Several solutions to this were considered, but ultimately, this issue is addressed by a mechanism outside of the cryptographic protocol.

6 A brief review of the literature

We briefly review the recent literature on private set intersection (PSI), and work related to the PSI-AD and tPSI-AD variants that we are interested in.

OPRF Methods An important category of PSI protocols is based on an oblivious PRF (OPRF). The OPRF can be implemented using the Diffie-Hellman PRF, using RSA, or using OT extension. The protocols based on OPRF from OT extension are the most efficient in computation. In particular, [KKRT16, PSZ18] make use of a one-query OPRF from OT extension along with Cuckoo hashing, while [PRTY19, CM20] build an elegant many-query OPRF from OT extension and do not need a data structure. However, these protocols do not satisfy the strict communication requirements in our settings.

The work of [CGT12] uses a DH-based OPRF to only reveal the cardinality of the intersection by secretly permuting the PRF evaluations of the server's dataset, so that the server does not know which of its PRF values corresponds to which element of X. The protocols in Section 4 can be viewed as an adaptation of this construction to our required constraints.

The recent work of [IKN+20] supports computation of set intersection cardinality. The proposed Diffie-Hellman method is related to [CGT12]. They also propose a method using a Bloom filter that can have false positives, which is undesirable in our settings.

The work of [GMR+21] also supports computation of set intersection cardinality, but requires more interaction between the client and server than is allowed in our settings.

Two party computation. [PSWW18, PSTY19, FNO19] provide efficient circuits for computing set intersection and cardinality. They then evaluate the circuit using 2PC techniques, either using Yao Garbled circuits or using GMW, so that only one party learns the intersection cardinality. Another technique in [CO18] shows how to combine the best non-MPC techniques with MPC to compute any function of the intersection. These methods do not quite fit the strict communication requirements in our settings.

[CLR17] gives a PSI protocol using fully homomorphic encryption (FHE) designed for the case where the sender's set is much larger than the device's set. This protocol applies in the settings where the server creates global public data pdata, but relies on FHE. A dual of their scheme only needs linearly homomorphic encryption.

Bloom filter methods. [EFG+15, KLS+17, DC17] construct PSI schemes using Bloom filters and additively homomorphic encryption. The scheme of [DC17] can be used to compute intersection cardinality, but can have false positives.

Threshold PSI. [HOS17] considers a variant of the PSI problem, called *threshold PSI*, where one of the parties learns the intersection contents only when then intersection cardinality exceeds a specified threshold. This problem is also studied in [GN17, ZC18, GS19, BDP20]. This variant of threshold PSI is quite different from our tPSI-AD setting. In our setting the requirement is that the protocol not reveal the intersection contents to either party, even when the threshold is exceeded. Only the associated data for the intersection is revealed. The protocol of [ZC18] can be adapted to reveal associated data when the intersection cardinality exceeds a threshold, as in our settings, but the protocol requires multiple round trips.

Acknowledgments

We thank Mihir Bellare and Benny Pinkas for helpful comments and discussions about this writeup.

References

[ABN18] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. *J. Cryptology*, 31(2):307–350, 2018. Early version in proc. of TCC 2010; Cryptology ePrint Archive, Report 2008/440.

[App21] Apple Inc. CSAM detection: technical summary, 2021. `https://apple.com`.

[BDP20] Pedro Branco, Nico Döttling, and Sihang Pu. Multiparty cardinality testing for threshold private set intersection. Cryptology ePrint Archive, Report 2020/1307, 2020. Cryptology ePrint Archive, Report 2020/130.

[Bel21] Mihir Bellare. A concrete-security analysis of the apple PSI protocol, 2021. public report.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, 2008. Early version in proc. of Asaicrypt 2000; Cryptology ePrint Archive, Report 2000/025.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, volume 4004 of *LNCS*, pages 409–426, 2006. Cryptology ePrint Archive, Report 2004/331.

[BS20] Dan Boneh and Victor Shoup. *A graduate course in applied cryptography*. 2020. `https://cryptobook.us`.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptol. ePrint Arch.*, 2000:67, 2000.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *CRYPTO*, volume 9216 of *LNCS*, pages 3–22. Springer, 2015. Cryptology ePrint Archive, Report 2014/553.

[CDG$^+$18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *EUROCRYPT*, volume 10820 of *LNCS*, pages 280–312. Springer, 2018. Cryptology ePrint Archive, Report 2018/165.

[CGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS 2012*, 2012. Cryptology ePrint Archive, Report 2011/141.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *ACM CCS*, pages 597–608. ACM, 2014. Cryptology ePrint Archive, Report 2014/908.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *CCS 2017*, pages 1243–1255. ACM, 2017. Cryptology ePrint Archive, Report 2017/299.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO 2020*, volume 12172 of *LNCS*, pages 34–63. Springer, 2020. Cryptology ePrint Archive, Report 2020/729.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN 2018*, 2018. Cryptology ePrint Archive, Report 2018/105.

[CS03] Don Coppersmith and Madhu Sudan. Reconstructing curves in three (and higher) dimensional space from noisy data. In *STOC*, pages 136–142. ACM, 2003.

[CSV19] Ran Canetti, Alley Stoughton, and Mayank Varia. EasyUC: Using easycrypt to mechanize proofs of universally composable security. In *Computer Security Foundations Symposium*, pages 167–183. IEEE, 2019. Cryptology ePrint Archive, Report 2019/582.

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *FC 2010*, pages 143–159, 2010. Cryptology ePrint Archive, Report 2009/491.

[DC17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, volume 10343 of *LNCS*, pages 261–278. Springer, 2017. Cryptology ePrint Archive, Report 2016/108.

[EFG+15] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *ACISP 2015*, volume 9144 of *LNCS*, page 413–430. Springer, 2015. online version.

[FHSS+20] A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves, 2020. `https://tools.ietf.org/id/draft-irtf-cfrg-hash-to-curve-06.html`.

[FLPQ13] Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In *Proc. of PKC '13*, volume 7778 of *LNCS*, pages 352–368, 2013. Cryptology ePrint Archive, Report 2012/673.

[FNO19] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *WPES 2019*, pages 14–25. ACM, 2019. Cryptology ePrint Archive, Report 2018/238.

[FOR17] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symmetric Cryptology*, 2017(1):449–473, 2017. Cryptology ePrint Archive, Report 2017/288.

[GMR$^+$21] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *PKC 2021*, 2021. Cryptology ePrint Archive, Report 2021/243.

[GN17] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. Cryptology ePrint Archive, Report 2017/1064, 2017. Cryptology ePrint Archive, Report 2017/1064.

[GPR$^+$21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanay. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO*, 2021. Cryptology ePrint Archive, Report 2021/883.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *CRYPTO 2019*, volume 11693 of *LNCS*, pages 3–29. Springer, 2019. Cryptology ePrint Archive, Report 2019/175.

[HOS17] Per Hallgren, Claudio Orlandi, and Andrei Sabelfeld. PrivatePool: Privacy-preserving ridesharing. In *CSF 2017*, 2017. online version.

[IKN$^+$20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P 2020*, pages 370–389. IEEE, 2020. Cryptology ePrint Archive, Report 2020/723.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: keyed-hashing for message authentication. *RFC*, 2104:1–11, 1997.

[KE10] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). *RFC*, 5869:1–14, 2010.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of CCS 2016*, pages 818–829. ACM, 2016. Cryptology ePrint Archive, Report 2016/799.

[KLS$^+$17] Agnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. In *PoPETs 2017*, 2017. Cryptology ePrint Archive, Report 2017/670.

[Lin16] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. Cryptology ePrint Archive, Report 2016/046.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS '97*, pages 458–467, 1997.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO 2019*, volume 11694 of *LNCS*, pages 401–431. Springer, 2019. Cryptology ePrint Archive, Report 2019/634.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT 2020*, volume 12106 of *LNCS*, pages 739–767. Springer, 2020. Cryptology ePrint Archive, Report 2020/193.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT 2019*, volume 11478 of *LNCS*, pages 122–153. Springer, 2019. Cryptology ePrint Archive, Report 2019/241.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT 2018*, volume 10822 of *LNCS*, pages 125–157. Springer, 2018. Cryptology ePrint Archive, Report 2018/120.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018. Cryptology ePrint Archive, Report 2016/930.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In *FSE'04*, volume 3017 of *LNCS*, pages 348–359. Springer, 2004.

[Sho20] Victor Shoup. Security analysis of SPAKE2+. In *TCC*, volume 12552 of *LNCS*, pages 31–60. Springer, 2020. Cryptology ePrint Archive, Report 2020/313.

[ZC18] Yongjun Zhao and Sherman S. M. Chow. Can you find the one for me? Privacy-preserving matchmaking via threshold PSI. In *WPES 2018*, 2018. Cryptology ePrint Archive, Report 2018/184.