Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search

Erkam Uzun, Simon P. Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity21/presentation/uzun

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search

Erkam Uzun
Georgia Institute of Technology

Simon P. Chung
Georgia Institute of Technology

Vladimir Kolesnikov
Georgia Institute of Technology

Alexandra Boldyreva
Georgia Institute of Technology

Wenke Lee
Georgia Institute of Technology

Abstract

The explosive growth of biometrics use (e.g., in surveillance) poses a persistent challenge to keep biometric data private without sacrificing the apps' functionality.

We consider private querying of a real-life biometric scan (e.g., a person's face) against a private biometric database. The querier learns only the label(s) of a matching scan(s) (e.g. a person's name), and the database server learns nothing.

We formally define *Fuzzy* Labeled Private Set Intersection (FLPSI), a primitive computing the intersection of noisy input sets by considering closeness/similarity instead of equality.

Our FLPSI protocol's communication is *sublinear* in database size and is concretely efficient. We implement it and apply it to facial search by integrating with our fine-tuned toolchain that maps face images into Hamming space.

We have implemented and extensively tested our system, achieving high performance with concretely small network usage: for a 10K-row database, the query response time over WAN (resp. fast LAN) is 146ms (resp. 47ms), transferring 12.1MB; offline precomputation (with no communication) time is 0.94s. FLPSI scales well: for a 1M-row database, online time is 1.66s (WAN) and 1.46s (fast LAN) with 40.8MB of data transfer in online phase and 37.5s in offline precomputation. This improves the state-of-the-art work (SANNS) by $9-25\times$ (on WAN) and $1.2-4\times$ (on fast LAN).

Our false non-matching rate is 0.75% for at most 10 false matches over 1M-row DB, which is comparable to underlying plaintext matching algorithm.

1 Introduction

Recent advances in deep learning (DL)-based biometric identification have made possible real-time identification of persons in footage collected by surveillance equipment. The trend toward real-time surveillance in public and private places (e.g., streets, city halls, airports, retail stores, pharmacies, gas stations etc.) has immense benefits for public safety or customer convenience. However, adoption of these technologies come at a significant privacy cost, which raises serious objections.

To our knowledge, existing biometrics surveillance systems have the following major challenges. First, vendors store and process the collected biometric data on their server in plaintext for the ease of deployment and practicality. Second, people cannot opt-out of these systems, since video footage (or any captured faces) are directly uploaded to a remote server.

Identifying "persons of interest" may be warranted [65], but tracking *everybody else* in the process may open the doors to illegitimate surveillance and certain human right abuses [67]. In response, privacy stakeholders are pressing for a moratorium on permanent adoption of these systems, and in fact they have already succeeded in banning facial surveillance in several countries and U.S. states [10, 64, 66].

In this paper, we propose a middle ground solution, *privacy-preserving biometric search*. Here the server $\mathcal{S}$ holding a large biometric database with corresponding labels (e.g., identities) should learn nothing about the query or the result, while the querier (client $\mathcal{C}$) should learn nothing about the database besides the label(s) of the query's match(es).

A similar problem of exact private match is extensively studied in a variety of scenarios (e.g., contact list discovery and online dating services), and can be achieved via (labeled) private set intersection (LPSI), a standard primitive [16, 17, 41, 49]. Even though the state-of-the-art CHLR18 [16] achieves a practical efficiency with communication costs sublinear to DB size, LPSI cannot directly be applied to our problem because it targets *exact* matches, while biometrics are noisy (e.g., due to lighting, imprecise scans, etc.).

We introduce FLPSI: a *fuzzy LPSI* protocol for fast privacy-preserving biometric search. We address a number of technical challenges in protocol/definition design and formal proofs.

To our knowledge, none of the prior work related to fuzzy matching achieves practical efficiency for real-time surveillance, mainly because of communication growing (at least) linearly with database size (see Sect. 2 and Sect. 11.5 for the related work). For example, two protocols of the state-of-the-art (SANNS [15]) require 1.7-5.4 GB communication and 15.1-41.7 sec. online response times over WAN per query over a million-row database.

We follow a much more scalable approach that reduces our fuzzy matching problem to an easier exact-matching subproblems that could be solved with communication cost sublinear in DB size, by leveraging optimizations of the state-of-the-art (L)PSI techniques [16, 17]. The crux of our solution is twofold. First, we translate the closeness (e.g., in Euclidean space) of two biometrics into a *t-out-of-T* set-based matching without sacrificing accuracy. That is, we encode a given biometric input into a set of T items, s.t. the two sets will likely have at least *t exactly* common items iff the biometrics are of the same person. Second, we build an efficient threshold set-matching protocol from fully homomorphic encryption (FHE), garbled circuits (GC) and *t-out-of-T* secret sharing, and solve several challenges in definitional approach.

1.1 Summary of Our Contributions

- We describe and formally define the functionality and security of *Fuzzy Labeled Private Set Intersection* (FLPSI). We build a FLPSI protocol using the AES blockcipher, homomorphic encryption, garbled circuits and *t-out-of-T* secret sharing. We prove the security in the semi-honest model.
- We show how to interpret closeness (e.g., in Euclidean space) between biometric inputs as *t-out-of-T* exact set-item matchings without sacrificing the accuracy.
- We give simulation-based FLPSI security definition (prior definitions of fuzzy primitives are game-based).
- We introduce a number of optimizations, in addition to the prior (L)PSI techniques we use.
- We extensively evaluate our protocol in different settings. We achieve 1.66s online running time over WAN with 40.8MB transfer per query over a million-row database.
- We systematically compare our design with prior art, and outperform all of them in their best settings, often by several orders of magnitude both in run time and communication. For example, on the largest dataset (of 10M records), we speed up by a factor of 3-33× and decrease the overall data communication by a factor of up to 48-452× compared to the two protocols of the state-of-the-art, SANNS [15].
- We highlight sublinear and concretely very small network use of our solution. In contrast with most other related work, our solution will scale on *very* small-bandwidth networks.

2 Related Work

As noted above, (L)PSI protocols [16, 17, 41, 49] and other exact-match protocols are inapplicable in our setting.

Freedman et al. [29] informally introduced the problem of *private fuzzy search* as an application of their private matching protocol. They proposed a basic construction, and left improving its efficiency as future work.

We now discuss state of the art techniques in fuzzy search.

Threshold Matching. The works [11, 18, 76] are based on threshold *t-out-of-T* matching outlined in [29]. These constructions incur (at least) linear in DB size and concretely inefficient communication and computation. We compare them in detail in Sect. 11.5 and Fig. 12 and show that they do not scale to a million-row DB.

A related line of work uses threshold over Euclidean or Hamming distance or cosine similarity between players' vectors [4, 5, 23, 34, 47, 53, 75]. While these works are generally faster than [11, 18, 76], our solution is still orders of magnitude more efficient. We provide detailed performance comparison in Sect. 11.5 and Fig. 11.

Our solution, also based on threshold *t-out-of-T* matching, must overcome the technical difficulties of i) high variability (and hence high distance) of feature vectors of the same person, and ii) large size of extracted feature vectors (hence high costs). We resolve both by carefully integrating fine-tuned random private subsampling of the feature vectors prior to computing threshold match (see Sect. 5).

Nearest Neighbor Search (NNS). A related line of work, albeit solving a *different* problem from the privacy perspective, is secure NNS. Indeed, NNS may (and is expected to) return match(es) for a person who is not present in DB; hence NNS does not meet our security requirements. However, we compare to NNS solutions as well, as they are *close enough in spirit* to our application scenario, and they are *faster* than prior work on private fuzzy search discussed above. Note, we do not consider outsourcing-based NNS [72, 78, 79, 81, 82] as they require a third party who learns the query ([52] requires two non-colluding servers). State-of-the-art optimized NNS [15] (SANNS) relies on a *fast* network connection (up to 56 gigabit/s) for efficiency as they transfer 1.7-5.4 GB to run a 10-NNS over a database of a million entries (6.1-57.7 GB over a database of 10 million entries). Hence, SANNS is not practical enough for real-time tasks at scale. We give a detailed comparison to SANNS in Sect. 11.5.

Finally, we mention, but do not discuss in detail, work on fuzzy searchable encryption [6, 43], as they address a different setting where the querier owns the data stored on an untrusted server (i.e. non-private DB).

3 Overview and Technical Details

Here we review existing non-private (plaintext) fuzzy matching algorithms and building privacy protection into them.

3.1 Plaintext Fuzzy Matching

Existing facial surveillance systems, informally, work as follows. A client $\mathcal{C}$ captures facial images of people from a surveillance video footage, then transmits the biometric data to a server $\mathcal{S}$ with transport encryption, and $\mathcal{S}$ receives the data in plaintext. Then, the server uses a DL system to turn raw

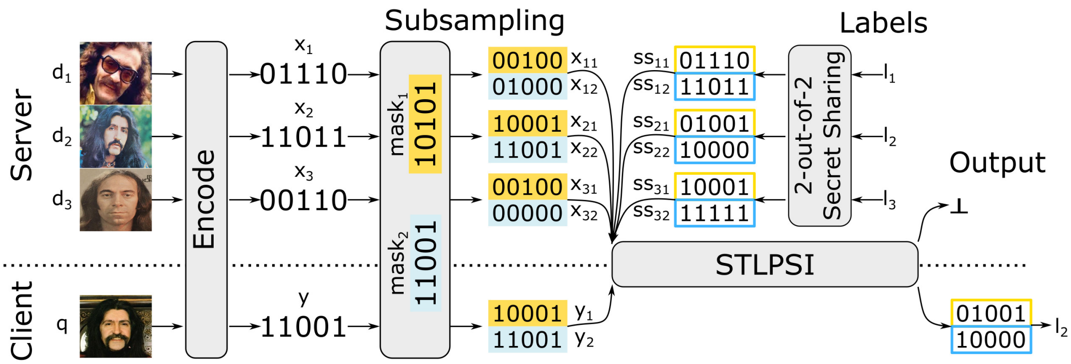

Figure 1: Overview of FLPSI. For clarity, subsampling is depicted without AES encryption and 2PC.

biometric readings into embedding vectors with a (probabilistic) guarantee that two such vectors will be close in Euclidean distance iff they are from the same person. If the server finds such a close biometric entry in its database, it returns the label (e.g, identity) of the entry to the client. Otherwise, it returns "*no match*" result to the client. In our evaluation, we used FaceNet [55], which is the state-of-the-art DL system achieving at most 0.67% for 10 false matches per query over 1M-row DB. See Fig. 7.

Privacy concerns. Clearly, since the data is typically processed in plaintext by the server, it achieves maximal privacy, while the client achieves none. Next, we discuss achieving maximal client privacy as well.

3.2 Private Fuzzy Matching

Our goal is to build a protocol that reveals labels of query matches only to $\mathcal{C}$, while maintaining confidentiality of $\mathcal{C}$'s query and $\mathcal{S}$'s database. To achieve this, $\mathcal{C}$ and $\mathcal{S}$ can locally compute DL embeddings from their biometric data, then apply standard MPC tools to compute Euclidean (or cosine similarity) distance between the $\mathcal{C}$'s query and each of the $\mathcal{S}$'s database items [4, 5, 23, 34, 53]. However, this does not scale (cf Fig. 11). Our much more scalable approach is based on a *t-out-of-T matching* scheme, described in detail next. Fig. 1 shows a high-level overview of our FLPSI protocol.

Binary encoding. To accommodate *t-out-of-T* matching, we first address the incompatibility between DL embeddings (operating in Euclidean space) and the crypto components (operating in Hamming space) of our protocol. (Operating in Hamming space, e.g., computing closeness is *much* cheaper in MPC). To do this, parties additionally apply a space mapping function, which is based on locality-sensitive hashes [13, 36, 38, 51], to convert the embedding vectors into bio-bit vectors (x_i and y) with the desired property (they are Hamming-close if they originate from the biometrics of the same person). This is also used as a dimension reduction process in scalable image search applications [42, 46, 77]. We note that there are different DL-based algorithms that generate binary representations directly from the raw data [21, 39, 70]. We omit exploring the best algorithm, and refer to [71] for a comprehensive survey. For lack of space, we present our space mapping technique from [68] in Appx. A. We will refer to the set of functions converting biometric data into bio-bit vectors, as "$Encode(.)$".

$\mathcal{C}$ and $\mathcal{S}$ proceed as follows after encoding their biometric data into bio-bit vectors y (held by $\mathcal{C}$) and $x_i \in \mathcal{X}$ (held by $\mathcal{S}$).

- **Subsampling:** generate T random subsamples of y and each x_i bio-bit vectors (in the same way, e.g., $x_{21} = x_2 \wedge mask_1$), s.t. at least t of them will be the same iff y and a x_i belong to the same person (if bio-bit vectors are Hamming-close, this can be achieved whp);
- **Secret sharing:** generate *t-out-of-T* secret shares of the label l_i (e.g., identity) of each $x_i \in \mathcal{X}$ (each share is associated with a subsample of x_i), s.t. any t shares can reconstruct l_i;
- **STLPSI:** interactively execute a private *t-out-of-T* matching protocol (*Set Threshold LPSI, or STLPSI*) on the $\mathcal{C}$'s subsample set and the $\mathcal{S}$'s sets of (subsample, secret share) pairs[1]: the label l_i of an $x_i \in \mathcal{X}$ is revealed to $\mathcal{C}$ iff at least t of the subsamples of y and x_i are equal (which means $\mathcal{C}$ obtains shares of matching subsamples of x_i).

Our private matching achieves false positive and negative rates equal to the state-of-the-art plaintext algorithms.

3.2.1 Our Solutions to Technical Challenges

Now we discuss the most interesting technical challenges.

Subsample confidentiality As described above, $\mathcal{C}$ learns the subsamples (and respective subsampling masks), which may help $\mathcal{C}$ learn something additional about database. For example, in case of a false-positive match, the semi-honest $\mathcal{C}$ will now learn with confidence positions in bio-bit vector, thereby learning the original biometric, which may not be included in the result set. Further, it may be the case (and publicly known) that faces in $\mathcal{S}$'s database are similar (e.g. manifested by certain positions of the bio-bit vectors being equal). A match from $\mathcal{C}$'s query will inform the malicious $\mathcal{C}$ how to set the bits of his next query so as to improve his

[1] Note that the secret shares are now treated as *labels* in the STLPSI.

chance of "hitting" a face in database (a false match).

We can resolve this by operating over encrypted subsamples only. For this, $\mathcal{S}$ chooses the random subsampling/projection masks and an AES encryption key $k_{\mathcal{S}}$. Then $\mathcal{S}$ via MPC allows $\mathcal{C}$ to compute the AES-encryptions of masked projections on the $\mathcal{C}$'s query bio-bit vector y, while keeping the projection masks and $k_{\mathcal{S}}$ secret from $\mathcal{C}$. The server efficiently computes AES-encryptions of masked projections on its large database non-interactively in $O(|\mathcal{X}|)$. Note that $\mathcal{S}$ has to refresh these masks and keys for each execution.

Concealing partial matches in single execution. (L)PSI protocols (e.g., [16, 17, 41, 49]) do not directly implement the above STLPSI functionality since they, by design, reveal partial (below-threshold t) matches.We resolve this by building effient STLPSI from *t-out-of-T* secret sharing and FHE, based on prior (L)PSI works (e.g., CHLR18 [16]).

Concealing partial matches in repeated executions. This subtle issue arises when generated shares are not refreshed between queries, and $\mathcal{C}$ may collect threshold t shares *across* queries. We resolve this by carefully resetting secret shares, subsampling masks and keys in each execution.

Novel definitional approach. In MPC, the preferred simulation-based security definitions offer clean and composable guarantees. At the same time, they require precise specification of ideal-world behavior, which we (as a community) do not know how to achieve for biometric functions. Because of this, biometric authentication definitions are usually game-based and not composable, but which allow to *bound*, rather than *precisely specify* adversary success.

One of our contributions is a *novel definitional approach* (see Section 7.1), which allows the best of both worlds: our definition is indeed simulation-based, and yet we bound adversary success rather than exactly specifying it. Our definition is generic and incorporates optional leakage, which is often needed for efficient sublinear protocols. We believe this definitional approach can serve as a template in defining primitives in the biometric space.

3.2.2 Trust Assumptions and Threat Model

- $\mathcal{C}$ and $\mathcal{S}$ locally apply binary encoding to their biometric data. We assume they own the same DL model that is trained on a public dataset and not tailored to any particular user from either party. Hence, we consider membership inference [58] or model inversion [27] attacks to be out of our scope.
- Considering our motivating application, we do not discuss here how the query biometric is obtained; we note that face detection in video footage is an easy and solved problem [69].
- We prove our 2PC (one $\mathcal{C}$ and one $\mathcal{S}$) in the semi-honest model (parties follow the protocol specification). In particular, parties do not corrupt their inputs (e.g., via a pixel perturbing attack [62]). This models natural scenarios in practice, as well as serves as a stepping stone to stronger models, such as handling malicious adversaries. Of course, many practical applications require protection against active cheating. Indeed, the biometric information served by $\mathcal{S}$ may be highly sensitive, and hence a possibility of data exfiltration by a malicious $\mathcal{C}$ would preclude offering the search service to a broader audience. We leave malicious security as important future work.

Resilience Against Certain Malicious Behaviour. While our protocol is in the SH model, we informally discuss several natural malicious attacks, their impact and mitigation.

Firstly, $\mathcal{S}$ can exclude its DB entries from search results simply by sending encryptions of random values. This can be *also* achieved by appropriate input substitution, and therefore is not an attack in a standalone execution setting. In general, efficiently ensuring that a malicious $\mathcal{S}$ is unable to omit entries in its DB is a hard technical problem.

Further, $\mathcal{C}$ can try to learn DB by querying random bit-vectors or brute-forcing arbitrary biometric inputs. Brute-forcing is a well-understood attack, which is mitigated by rate-limiting. Querying bit-vectors is not helping $\mathcal{C}$, since the bit-vector search space is larger than the space of faces.

4 Definition of FLPSI

In this section, we define a general syntax for a fuzzy labeled PSI. We start with the notion of closeness (fuzzy matching), adopted from [6].

Definition 4.1. Closeness Domain. *We say that* $\Lambda = (\mathcal{D}, \mathrm{Cl})$ *is a* closeness domain *if* $\mathcal{D}$ *is a set, and* Cl *is the (partial) closeness function that takes any* $x, y \in \mathcal{D}$ *and outputs a member of* $\{\mathtt{close}, \mathtt{far}\}$*, so that* Cl *is symmetric (i.e.,* $\mathrm{Cl}(x,y) = \mathrm{Cl}(y,x)$*).*

There are no requirements on the output of `close` for pairs that are "near" (i.e. points that neither close nor far).

Definition 4.2. Fuzzy Labeled PSI (FLPSI). *FLPSI protocol is defined for a closeness domain* $(\mathcal{D}, \mathrm{Cl})$ *and a label space* $\mathcal{LS}$ *by the interactive algorithm* $(\mathcal{C}, \mathcal{S})$*, where the client* $\mathcal{C}$ *inputs a query* $q \in \mathcal{D}$*, and the server* $\mathcal{S}$ *inputs a database* $Db = \{(d_1, l_1)..,(d_N, l_N)\}$*, where items* $d_i \in \mathcal{D}$ *and labels* $l_i \in \mathcal{LS}$*. At the end,* $\mathcal{C}$ *outputs a set* R*, and* $\mathcal{S}$ *outputs* $\perp$*. FLPSI must satisfy the following correctness and security properties:*

Correctness. We use ε-correctness instead of a perfect correctness, as it is common for biometrics-matching systems to have errors, e.g., false matches (ε_1) and non-matches (ε_2). Then, it requires that, for q and Db, we have an output set R consisting *only* of pairs (q, l_i) such that for each $i \in [N]$[2];

if $\mathrm{Cl}(q, d_i) = \mathtt{far}$, then $\Pr[(q, l_i) \notin R] \geq 1 - \varepsilon_1$;

if $\mathrm{Cl}(q, d_i) = \mathtt{close}$, then $\Pr[(q, l_i) \in R] \geq 1 - \varepsilon_2$, where $(d_i, l_i) \in Db$.

[2] $[N]$ is a shorthand for $\{1, 2, \ldots, N\}$.

In our construction the domain $\mathcal{D}$ refers to facial biometrics in a surveillance scenario. Hence, $\mathrm{Cl}(q,d_i) = \texttt{far}$ refers to each of q,d_i coming from different people, while $\mathrm{Cl}(q,d_i) = \texttt{close}$ refers to both of them belonging to the same person. The label space $\mathcal{LS}$ refers to people's identities or other info. Hence, the client learns the information corresponding to the person in its query, if the photo(s) of this person is in the database.

Security of FLPSI is formally defined in Sect. 7.1 in the simulation paradigm by specifying the ideal functionality. We stress that we separately require $\Pi_{\texttt{FLPSI}}$ to satisfy the above correctness requirement. We present this low-level technical definition discussion together with the proofs in Sect. 7.1, and focus on the protocol description next.

5 Building Blocks of FLPSI

In this section, we discuss the ideas behind our protocol and its building blocks (presented formally in Sect. 6).

5.1 Binary Encoding

Our construction starts with a binary encoding step, where the client and server locally turn each of their raw biometric inputs (e.g., facial photos) $q,d_i \in \mathcal{D}$, for $i \in [N]$, into bio-bit vectors $y = Encode(q)$ and $x_i = Encode(d_i)$, respectively, so that if there is a q,d_i pair of the same person, then y,x_i are likely close wrt Hamming distance.

5.2 Subsampling and 2PC

Now the client and server could apply random projections for each bit vector y and x_i, respectively. This outputs a set of subsampled bit vectors, $\mathcal{Y} = \{y_1,\ldots,y_T\}$ and $\mathcal{X}_i = \{x_{i1},\ldots,x_{iT}\}$, with the property that if y and x_i are close, then some subsamples of them will likely be the same [12, 20, 68].

$\mathcal{S}$ hides the subsampling projections from $\mathcal{C}$ to avoid it reconstructing the inputs in the database (see Sect. 3.2.1). To do this, $\mathcal{S}$ chooses a 128-bit AES blockcipher key $k_{\mathcal{S}}$ and generates the projection masks $\{mask_1,\ldots,mask_T\}$. Note that $\mathcal{S}$ can locally compute its subsample set $\mathcal{X}_i$ for each of its bio-bit vector x_i s.t. $x_{ij} = AES_{k_{\mathcal{S}}}(x_i \wedge mask_j)$, where $j \in [T]$.

Next both parties execute a 2PC protocol $(\mathcal{C}_{AES},\mathcal{S}_{AES})$. This protocol privately computes each $y_j \in \mathcal{Y}$ s.t. $y_j = AES_{k_{\mathcal{S}}}(y \wedge mask_j)$ for $\mathcal{C}$, s.t. it can learn matched encrypted subsamples without learning the projection masks and $k_{\mathcal{S}}$.

We implement this 2PC protocol as Yao's Garbled Circuits (GC)[3] using the EMP toolkit [73]. We use *subsamples* and *encrypted subsamples* interchangeably, by referring $\mathcal{Y}$, throughout the paper.

5.3 Secret Sharing

Our construction will use STLPSI (introduced in the next section) along with a *t-out-of-T* secret sharing scheme, whose syntax, correctness and security we now define.

Definition 5.1. t-out-of-T Secret Sharing. *A t-out-of-T secret sharing scheme is defined by algorithms* (KS,KR) *for sharing and reconstructing a secret. The domain of secrets is* $\{0,1\}^{\mathcal{K}}$*, where* $\mathcal{K}$ *is some parameter.* KS *takes a secret* s *and outputs a set* $\{ss_1,\ldots,ss_T\}$ *of shares.* KR *takes as input a set of shares* $\{ss_1,\ldots,ss_d\}$ *and returns an integer* $s \in \{0,1\}^{\mathcal{K}}$ *if* $d \geq t$*, or* $\perp$ *if* $d < t$.

Correctness. For correctness we demand that for any $s \in \{0,1\}^{\mathcal{K}}$, any set $SS \in [\mathsf{KS}(s)]$, and $SS_i \subseteq SS$, where $|SS_i| \geq t$, it holds that $\mathsf{KR}(SS_i) = s$ with probability 1.

Privacy. For privacy we demand that for any $s \in \{0,1\}^{\mathcal{K}}$ and set $SS \in [\mathsf{KS}(s)]$ it holds that any subset $SS_i \subseteq SS$ of size $|SS_i| < t$ does not give any information about s, i.e., its probability distribution is independent of s.

In our protocol, $\mathcal{S}$ generates *t-out-of-T* secret shares for the label $l_i \in \mathcal{LS}$ associated with the i^{th} entry in its database. Note that $\mathcal{S}$ attaches an agreed-upon token 0^{λ}, where λ is a security parameter, to each label l_i before secret sharing it. Then, $\mathcal{C}$ can verify if any set of t shares (obtained via a single STLPSI execution) correctly reconstruct a valid label. Overall, for a given label $l_i \in \mathcal{LS}$, $\mathcal{S}$ generates $\{ss_{i1},\ldots,ss_{iT}\} \overset{\$}{\leftarrow} \mathsf{KS}(0^{\lambda} \mid\mid l_i)$.

5.4 Set Threshold LPSI (STLPSI)

Though prior steps prepare the inputs to accommodate a *t-out-of-T* matching, existing private *t-out-of-T* matching schemes are not practical for a real-time surveillance (see Sect. 11.5.1). We require small communication (e.g., under 128 MB), to support server with a large (1M rows) database and a WAN or less channel to the client.

A closely related LPSI primitive does achieve above performance [16], but cannot be plugged in *directly* as a building block to FLPSI, since it does not implement *t-out-of-T* matching (LPSI reveals below-threshold t matching to the client). It is, however, possible to combine with an appropriate carefully designed secret sharing scheme to achieve this feature as well.

For modularity and because STLPSI is a useful primitive, we formalize it and implement it based on prior techniques, mostly drawn from CHLR18. We integrate a number of optimizations specific to our setting, such as different bucketing, removing now redundant preprocessing steps and the use of cuckoo hashing in CHLR18. We prove security of the resulting protocol (Theorem 5.1).

[3] Note, we could have used a generic oblivious PRF (OPRF) to implement encrypted subsampling; however, known efficient OPRF are public-key based, even when both key and input are known to the evaluator. This would result in a more expensive solution, since (expensive public-key) OPRF must be applied by $\mathcal{S}$ to each DB entry for each query. In contrast, while we pay slightly more to evaluate AES inside MPC on the $\mathcal{C}$'s query, we pay *much* less to encrypt DB entries.

Functionality $\mathcal{F}_{\mathtt{STLPSI}}$

Given inputs $\mathcal{Y} \subset \mathcal{MS}$ of $\mathcal{C}$, and $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$ and $SS = \{SS_1, \ldots, SS_N\}$ of $\mathcal{S}$, where $\mathcal{X}_i \subset \mathcal{MS}$ and $SS_i \subset \mathcal{SS}$, the functionality sends the output result set R (cf Def. 5.2) to the client, and nothing to the server.

Figure 2: The Ideal Functionality $\mathcal{F}_{\mathtt{STLPSI}}$

5.4.1 Formal Definition of STLPSI

In this section, we formally define a general syntax and correctness for a private *t-out-of-T* matching protocol.

Definition 5.2. Set Threshold Labeled Private Set Intersection (STLPSI). *STLPSI protocol is defined by the input space $\mathcal{MS}$, label space $\mathcal{SS}$*[4]*, threshold values $t, T \in \mathbb{N}$ and the interactive algorithm $(\mathcal{C}, \mathcal{S})$. $\mathcal{C}$ inputs a query set $\mathcal{Y} = \{y_1, \ldots, y_T\} \subset \mathcal{MS}$, and $\mathcal{S}$ inputs database sets $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$, where $\mathcal{X}_i = \{x_{i1}, \ldots, x_{iT}\} \subset \mathcal{MS}$, and $SS = \{SS_1, \ldots, SS_N\}$, where $SS_i = \{ss_{i1}, \ldots, ss_{iT}\} \subset \mathcal{SS}$. At the end, $\mathcal{C}$ outputs a set R, while $\mathcal{S}$ outputs $\perp$.*

Correctness. We require that $R = \{r_1, \ldots, r_N\}$ s.t. for each $i \in [N]$, let $r'_i = \{(x_{ij}, ss_{ij}) : x_{ij} = y_j \in \mathcal{Y}, ss_{ij} \in SS_i\}_{j \in [T]}$, then we have that $r_i = (r'_i, l_i)$ iff $|r'_i| \geq t$, otherwise $r_i = \emptyset$. That is, through the set R, $\mathcal{C}$ learns such tuples (x_{ij}, ss_{ij}) and the label l_i associated with them iff it gets at least t exactly matching items between sets $\mathcal{Y}$ and $\mathcal{X}_i$.

Now we define the security of STLPSI. Let Π be an STLPSI protocol, defined according to Def. 5.2. The ideal functionality $\mathcal{F}_{\mathtt{STLPSI}}$ is defined in Fig. 2. In Appx. B, we recall the standard definition of securely realizing ideal functionality in the semi-honest model [24], formulated as Def. B.1 for the 2PC case. Now we can put these together to define security of STLPSI.

Definition 5.3. STLPSI Security. *We say that a protocol $\Pi_{\mathtt{STLPSI}} = (C, S)$, defined w.r.t. input space $\mathcal{MS}$ and label space $\mathcal{SS}$, is a secure STLPSI protocol (in the semi-honest model) with no leakage if $\Pi_{\mathtt{STLPSI}}$ securely realizes (cf. Def. B.1) functionality $\mathcal{F}_{\mathtt{STLPSI}}$ of Fig. 2.*

5.4.2 Constructing STLPSI Protocol

For clarity, we first explain how a private *t-out-of-T* matching works on two sets, $\mathcal{Y}$ from the client and $\mathcal{X}_i \in \mathcal{X}$ from the server (each with T items) in a strawman design. Then, we introduce our optimizations for an efficient construction.

Strawman design. First, $\mathcal{C}$ and $\mathcal{S}$ agree on an FHE scheme, where $\mathcal{C}$ generates (public, secret) keys (pk, sk) and sends pk to $\mathcal{S}$. $\mathcal{C}$ also homomorphically encrypts each set item $y_j \in \mathcal{Y}$ into a ciphertext $[\![y_j]\!]$ and sends to $\mathcal{S}$. Then, $\mathcal{S}$ computes $[\![z_{ij}]\!] = r \times ([\![y_j]\!] - x_{ij}) + ss_{ij}$ under encryption[5], where $r \in_R \mathcal{P}$ and is refreshed for each computation, and ss_{ij} is the secret share (of label l_i) associated with x_{ij}. $\mathcal{S}$ sends the ciphertext $[\![z_{ij}]\!]$ to $\mathcal{C}$. Recall that secret shares are uniformly sampled items in $\mathcal{SS}$ (equal to $\mathcal{MS}$). Notice, $z_{ij} = s_{ij}$ iff $y_j = x_{ij}$, Otherwise, z_{ij} is random on $\mathcal{SS}$. Now, it is guaranteed that $\mathcal{C}$ can reconstruct the label l_i iff it gets at least t shares of l_i (see Def. 5.1). Otherwise, $\mathcal{C}$ learns nothing and cannot distinguish possible below-threshold t matches.

Optimizations. Applying above evaluation for each DB item does not scale to large DBs (e.g., of a million sets). We adopt the following optimizations from the (L)PSI literature for compressing DB items and reducing the circuit depth in FHE evaluations [9, 16, 17, 28, 29, 48, 49, 60]. With the exception of *bucketing*, we closely follow CHLR18 [16] in applying the following optimizations:

Polynomial interpolation is used instead of the above strawman to generate $[\![z_{ij}]\!]$. To do this, $\mathcal{S}$ interpolates an N-degree polynomial P_j, by using $(\langle\text{item}\rangle, \langle\text{secret share}\rangle)$ pairs (x_{ij}, ss_{ij}) s.t. $P_j(x_{ij}) = ss_{ij}$. Since $P_j(y) = \alpha_N y^N + \ldots + \alpha_1 y + \alpha_0$, where the α_i could be pre-computed by $\mathcal{S}$ in the offline phase, P_j is homomorphically evaluated in $O(\log N)$ multiplicative depth given $[\![y]\!]$. Further, a single $[\![z_{ij}]\!]$ now encodes a secret share corresponding to any of the matching x_{ij}.

Bucketing. Prior exact matching works use different methods (e.g., cuckoo hashing, bloom filters) to bucketize the DB items, so that the query item needs to be compared only with DB items in the same bucket. In our protocol, since each of T set items are generated through different LSH projections, each projection is interpreted as a bucket (with N items). Note that bucketing is a standard PSI technique [50], also used in CHLR18. It improves asymptotic performance, but concretely is costly, as buckets must be padded with dummy element for security. Crucially, this additional bucketing is *not needed* in our application. As noted above, projections already define the buckets within which the search is performed, and they need not be padded.

We combine bucketing with windowing, described next.

Windowing. Interpolating polynomials over buckets doesn't scale to large N values (e.g., a million). If $\mathcal{C}$ sends the encryptions of $y^{2^0}, y^{2^1}, y^{2^2}, \ldots, y^{2^{\log N}}$, $\mathcal{S}$ can homomorphically compute all necessary powers of y in $O(\log\log N)$ multiplicative depth. This technique decreases $\mathcal{C} \leftarrow \mathcal{S}$ communication cost by a factor of N, while increasing the $\mathcal{C} \rightarrow \mathcal{S}$ cost by a factor of $\log N$, which has a small impact on overall communication since $\mathcal{C}$ only holds a set of T items.

Splitting. To speed-up homomorphic evaluations, $\mathcal{S}$ splits each bucket into a partitions, s.t. it interpolates a $\frac{N}{a}$-degree polynomial per partition. This reduces the multiplicative depth to $O(\log\log\frac{N}{a})$, and the number of y powers ($\mathcal{C}$ sends to $\mathcal{S}$) to $\log\frac{N}{a}$, but increases the $\mathcal{C} \leftarrow \mathcal{S}$ communication by a factor of a as $\mathcal{S}$ sends results for all partitions to $\mathcal{C}$.

Batching. This is a well-known technique in FHE to enable Single Instruction Multiple Data (SIMD) on ciphertexts. For more details and example applications, we refer

[4] Since the labels are secret shares in STLPSI, we use $\mathcal{SS}$ for the label space to avoid confusion with the label space $\mathcal{LS}$ of the main protocol.

[5] Following [16], we slightly abuse notation to emphasize FHE operations with known values. Formally, we are computing $[\![z_{ij}]\!] = [\![r \times (y_j - x_{ij}) + ss_{ij}]\!]$.

Inputs: $\mathcal{C}$ inputs set $\mathcal{Y} = \{y_1, \dots, y_T\} \subset \mathcal{MS}$; $\mathcal{S}$ inputs $\mathcal{X} = \{\mathcal{X}_1, \dots, \mathcal{X}_N\}$, where $\mathcal{X}_i = \{x_{i1}, ..., x_{iT}\} \subset \mathcal{MS}$, and $SS = \{SS_1, \dots, SS_N\}$, where $SS_i = \{ss_{i1}, \dots, ss_{iT}\} \subset \mathcal{SS}$ is the secret shares of $0^\lambda \,||\, l_i$ s.t. λ is a param., $l_i \in \mathcal{LS}$.

1. **[FHE parameters]** Parties agree on FHE parameters $(m, m_p, m_{ct}, \mathcal{P})$ for an IND-CPA secure FHE scheme, and on threshold (t), split $(B = \frac{NT}{ma})$ and windowing $(w \in \{2^1, 2^2, \dots, 2^{\log B}\})$ parameters. Then, $\mathcal{C}$ generates (public, secret) FHE keys (pk, sk), then sends pk to $\mathcal{S}$.
2. **[Pre-process $\mathcal{X}$ and SS]** $\mathcal{S}$ bucketizes $\mathcal{X}$ into a table and splits each bucket (column) into a partitions, then interpolates a B-degree polynomial P_k for each partition, s.t. $P_k(x_{ij}) = ss_{ij}$, where ss_{ij} is the secret share associated with x_{ij}. Then, $\mathcal{S}$ batches coefficients at corresponding row (of lenght m) of the table into a plaintext polynomial p_k^ℓ, where $\ell \in B$, $k \in a$.
3. **[Compute encrypted query from $\mathcal{Y}$]** $\mathcal{C}$ concatenates its input set $\frac{m}{T}$ times and batches into a plaintext polynomial. Then, it homomorphically encrypts (by using pk) each windowing power (w) of this plaintext polynomial into the ciphertexts $[\![y_w]\!]$, then sends them to $\mathcal{S}$.
4. **[Homomorphic intersection]** $\mathcal{S}$, under encryption, i) expands the received $[\![y_w]\!]$ to $\{[\![y_0]\!], [\![y_1]\!], \dots, [\![y_B]\!]\}$; ii) evaluates $[\![z_k]\!] = \sum_{\ell=0}^{B} [\![y_\ell]\!] . p_k^\ell$ for each $k \in a$, and sends all ciphertexts $[\![z_k]\!]$ to $\mathcal{C}$ after applying noise flooding and modulus switching on them.
5. **[Decrypt and get result]** $\mathcal{C}$ decrypts (by using sk) ciphertexts $[\![z_k]\!]$ to obtain result vector z_k (of length m) for each partition $k \in a$. Now, each item of z_k will be the evaluation of $P_k(y_j) = ss_{ij}$ iff there is any $x_{ij} = y_j$ in partition k of corresponding bucket, otherwise the item will be a random element in $\mathbb{F}_{\mathcal{P}}$.
6. **[Reconstruct label]** $\mathcal{C}$ runs KR algorithm on each $\binom{T}{t}$ combination of consecutive T items of z_k, and gets a reconstruction result s_i. We have $r'_i = \{(x_{ij}, ss_{ij}) : x_{ij} = y_j \in \mathcal{Y}, ss_{ij} \in SS_i\}_{j \in [T]}$. Then $s_i = 0^\lambda \,||\, l_i$ and $r_i = (r'_i, l_i)$ iff $|r'_i| \geq t$, otherwise s_i is a random element from $\mathbb{F}_{\mathcal{P}}$ (see Def. 5.1) and $r_i = \emptyset$. 0^λ validates a label l_i in DB.

Output: $\mathcal{C}$ outputs a result set $R = \{r_1, ..., r_N\}$, where each $r_i \neq \emptyset$ is recovered in Step 6. $\mathcal{S}$ outputs $\perp$.

Figure 3: STLPSI protocol $\Pi_{\texttt{STLPSI}}$

[8, 17, 30, 31, 60]. In this work, we specifically use SIMD batching from FHE library SEAL [56]. To accommodate it, $\mathcal{S}$ groups coefficients, associated with the same powers of $y_1, y_2, \dots, y_T$ from different buckets, into vectors of length m. Since m is parameterized as $m \gg T$, $\mathcal{S}$ also concatenates coefficients from $\frac{m}{T}$ partitions. This means batching $\frac{m}{T}$ sets into a single vector, that decreases each partition size to $\frac{NT}{ma}$. Finally, $\mathcal{C}$ concatenates its set $\frac{m}{T}$ times and batches into a plaintext polynomial, then it computes all windowing powers of it and sends encryptions of them to $\mathcal{S}$. Overall, batching decreases i) the FHE multiplicative depth to $O(\log\log \frac{NT}{ma})$; ii) the number of y powers ($\mathcal{C}$ sends to $\mathcal{S}$) to $\log \frac{NT}{ma}$; and iii) $\mathcal{C} \leftarrow \mathcal{S}$ communication by a factor of $\frac{m}{T}$.

Noise flooding. $\mathcal{S}$ re-randomizes the returned ciphertexts by adding fresh encryptions of zero with large noise [22]. This results in increased FHE parameters. See Sect. 11.2.

Modulus switching. This is a technique that FHE scheme allows to reduce the complexity of a ciphertext at some degrees [9]. In our design, $\mathcal{S}$ performs SEAL's modulus switching on encrypted results before sending them to $\mathcal{C}$.

After receiving the evaluation results, the client decrypts each of them to $\frac{m}{T}$ sets (each with T results). Then, it runs the reconstruction algorithm KR from Def. 5.1 on $\binom{T}{t}$ combinations of each set and obtains a label l_i iff at least t query subsamples match with the ones from i^{th} database set.

5.4.3 Full Protocol and Security Proof of STLPSI

Our STLPSI protocol is formally presented in Fig. 3.

Theorem 5.1. *In the presence of a semantically secure fully homomorphic encryption and t-out-of-T secret sharing schemes, the $\Pi_{\texttt{STLPSI}}$ protocol of Fig. 3 is a secure (in the semi-honest model) STLPSI protocol with no leakage if each of the server's input sets of labels $SS_i \in SS$ for $i \in [N]$ are:*

- *randomly sampled (and unknown to $\mathcal{C}$) t-out-of-T Shamir's secret shares of $0^\lambda \,||\, l_i$, where λ is a security parameter and $l_i \in \mathcal{LS}$ is the label associated with i^{th} database record;*
- *the domain of secret shares $\mathcal{SS}$ is equal to the domain $\mathbb{F}_{\mathcal{P}}$ of the underlying fully homomorphic encryption scheme.*

Proof. The intuition for the protocol security is presented above in Sect. 5.4.2. For space, we formally prove security of our protocol $\Pi_{\texttt{STLPSI}}$ w.r.t. Def. 5.3 in Apx. C.1. □

6 FLPSI Protocol

In Sect. 3.2 we present the technical intuition of our $\Pi_{\texttt{FLPSI}}$ protocol. Fig. 4 formalizes that discussion and presents $\Pi_{\texttt{FLPSI}}$ for the closeness domain $(\mathcal{D}, \mathsf{Cl})$ and label space $\mathcal{LS}$. The protocol uses the building blocks AES blockcipher, *t-out-of-T* secret sharing scheme $(\mathsf{KS}, \mathsf{KR})$, 2PC protocol $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$, and STLPSI protocol $(\mathcal{C}_{STLPSI}, \mathcal{S}_{STLPSI})$.

In the protocol, *Encode* is an algorithm that generates $\mathcal{L}$-bit bit vector for an input from $\mathcal{D}$; $k_\mathcal{S}$ is 128-bit AES blockcipher key; T is number of subsamples; t is matching threshold; λ is a security parameter; and $\wedge$ is "*logical and*" operation, used in subsampling function, i.e., $AES_{k_\mathcal{S}}(y \wedge mask_j)$.

The outputs of both $\mathcal{S}$'s subsampling in Step 3 and $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ (Step 5), and the input items of $\mathcal{C}_{STLPSI}$ and $\mathcal{S}_{STLPSI}$ should be in the same domain $\mathcal{MS}$. Moreover, the output of secret sharing KS (Step 4) and the input labels of $\mathcal{S}_{STLPSI}$ should be from the same domain $\mathcal{SS}$.

6.1 Instantiating FLPSI Protocol

We now discuss specific instantiations of $\Pi_{\texttt{FLPSI}}$ building blocks, tailored for our use case. Discussion of low-level protocol and subprotocol parameters is presented in Sect. 11.2.

Inputs: $\mathcal{C}$ inputs query $q \in \mathcal{D}$; $\mathcal{S}$ inputs a database $Db = \{(d_1, l_1), ..., (d_N, l_N)\}$, where each $d_i \in \mathcal{D}$ and label $l_i \in \mathcal{LS}$.

1. **[Encode]** Parties agree on function $Encode : \mathcal{D} \rightarrow \{0,1\}^{\mathcal{L}}$. $\mathcal{C}$ computes $y = Encode(q)$, and $\mathcal{S}$ computes $x_i = Encode(d_i)$ for each $i \in [N]$.
2. **[Init]** The server $\mathcal{S}$ samples an AES key $k_{\mathcal{S}} \xleftarrow{\$} \{0,1\}^{128}$ and T projection masks $\{mask_1, \ldots, mask_T\}$.
3. **[Server's local subsampling]** The server generates subsample set $\mathcal{X}_i = \{x_{i1}, ..., x_{iT}\}$, where $x_{ij} \in \mathcal{MS}$ such that $x_{ij} = AES_{k_{\mathcal{S}}}(x_i \wedge mask_j)$ for each $i \in [N]$ and $j \in [T]$.
4. **[Secret sharing]** $\mathcal{S}$ generates a secret share set $SS_i = \{ss_{i1}, \ldots, ss_{iT}\} \xleftarrow{\$} \mathsf{KS}(0^\lambda \,||\, l_i)$ for each $i \in [N]$, where $ss_{ij} \in \mathcal{SS}$, 0^λ is an agreed token, and l_i is the i^{th} label.
5. **[Client's 2PC oblivious subsampling]** $\mathcal{C}$ and $\mathcal{S}$ run $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$, where $\mathcal{C}_{AES}$ inputs y, $\mathcal{S}_{AES}$ inputs $k_{\mathcal{S}}$ and $\{mask_1, \ldots, mask_T\}$. Then, $\mathcal{C}_{AES}$ learns the subsample set $\mathcal{Y} = \{y_1, ..., y_T\}$, where $y_j \in \mathcal{MS}$ s.t. $y_j = AES_{k_{\mathcal{S}}}(y \wedge mask_j)$ for each $j \in [T]$, and $\mathcal{S}_{AES}$ learns $\perp$.
6. **[STLPSI execution]** $\mathcal{C}$ and $\mathcal{S}$ run $(\mathcal{C}_{STLPSI}, \mathcal{S}_{STLPSI})$, where $\mathcal{C}_{STLPSI}$ inputs $\mathcal{Y}$, and $\mathcal{S}_{STLPSI}$ inputs $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$, and $SS = \{SS_1, \ldots, SS_N\}$. At the end, $\mathcal{S}$ learns $\perp$, and $\mathcal{C}$ learns a set R as per Definition 5.2. I.e., we have $r'_i = \{(x_{ij}, ss_{ij}) : x_{ij} = y_j \in \mathcal{Y}, ss_{ij} \in SS_i\}_{j \in [T]}$ for each $i \in [N]$, and $r_i = (r'_i, l_i)$ iff $|r'_i| \geq t$, otherwise $r_i = \mathbb{0}$. Then, $R = \{r_1, ..., r_N\}$.

Output: For each $r_i \in R$, $r_i \neq \mathbb{0}$, $\mathcal{C}$ outputs l_i label recovered in Step 6. $\mathcal{S}$ outputs $\perp$.

Figure 4: FLPSI protocol $\Pi_{\texttt{FLPSI}}$

In the binary encoding step, $\mathcal{C}$ and $\mathcal{S}$ locally i) encode their raw biometrics (q and d_i, resp.) into embedding vectors, by using the state-of-the-art DL model (e.g., FaceNet [55]); and ii) translate the Euclidean space into Hamming, by using Super-Bit Locality Sensitive Hash (SBLSH) [38] (see Appx. A), that converts DL embeddings into bio-bit vectors (y and x_i, resp.).

Next, following Sect. 5.2, $\mathcal{C}$ and $\mathcal{S}$ generate their subsample sets ($\mathcal{Y}$ and $\mathcal{X}_i$, resp.). Recall that, $\mathcal{S}$ generates each projection mask, by randomly choosing $\mathcal{N}_{sb}$ ones, which essentially turns all other bits into a constant zero.

Before each protocol execution, $\mathcal{S}$ regenerates the projection masks, AES encryption key $k_{\mathcal{S}}$, and secret shares of each label l_i for each DB record (by using *t-out-of-T* Shamir's secret sharing scheme [57]), so that $\mathcal{S}$ ensures that $\mathcal{C}$ is not able to use any information seen in a prior execution.

Finally, parties run STLPSI protocol in Fig. 3, where $\mathcal{C}$ inputs its subsamples, and $\mathcal{S}$ inputs subsamples and their associated secret shares for each DB record. At the end, $\mathcal{S}$ learns nothing and $\mathcal{C}$ learns the label l_i (and matching items) of i^{th} database record iff the i^{th} record has at least t matching subsamples with the $\mathcal{C}$'s set.

Correctness. The protocol works correctly with small false matching (ε_1) and non-matching (ε_2) error probabilities. Since we can only empirically estimate these errors, we defer this analysis to Sect. 11.2. In summary, we target to get maximum error rates of $\varepsilon_1 = 0.001$ and $\varepsilon_2 = 0.01$ for the smallest DB.

7 Security Analysis of FLPSI Protocol

We start our analysis by formally defining security of FLPSI. We then state and prove security in Sect. 7.2.

7.1 Security Definition of FLPSI

SECURITY DEFINITION DISCUSSION. Following the common approach to modeling security of 2PC, we use the ideal-real paradigm and consider static security against a semi-honest adversary that can corrupt at most one participant.

We need to define an ideal functionality for FLPSI and what it means for a protocol to securely realize it. An ideal functionality takes inputs from the parties, computes the desired parties' outputs, and returns them to the parties, along with the corresponding leakage (if any). We require that the view of each party in real protocol's execution is indistinguishable from the view produced by the ideal-world simulator.

This is a common general approach, which, unfortunately, does not fit FLPSI and many natural biometric functionalities. The difficulty we are facing is that the ideal functionality must precisely match what happens in the real world. In particular, the parties' outputs should have the same distribution in both worlds on all inputs. In our case this would mean that the ideal functionality specifies the *exact* probability of any two close elements correctly identifying as close by the protocol, as well as the probability of far elements correctly identifying as far. This is *unrealistic* to achieve for real-world settings, such as facial biometrics we focus on.

This complication is *avoided* in game-based definitions, where no ideal functionality is defined, and hence there is no need to explicitly specify it. Indeed, security there can be defined as an adversary unable to succeed (e.g., learn something forbidden) with probability above a certain threshold. However, they will not guarantee that absolutely nothing additional is revealed, and such protocols are not freely composable – these are features of ideal-real style definitions.

Our approach. We reconcile the yin and yang and achieve the best of both by defining the ideal FLPSI functionality via a reference to a real FLPSI protocol $\Pi_{\texttt{FLPSI}}$. Namely, we will say that ideal functionality outputs whatever the real $\Pi_{\texttt{FLPSI}}$ formally outputs. Recall, in our case (cf Def. 4.2), it is the set R or pairs (q, l_i). While at the first glance this may seem a tautology, this approach does provide a guarantee that nothing beyond the explicit protocol output is revealed. Now we are in a good place, since we can easily control explicit protocol output by specifying the *correctness* property. Indeed, Def. 4.2 requires (modulo errors bounded by ε) that the $\mathcal{C}$ outputs `close` labels, and in particular does not output `far` labels or any other information it may have learned from the view of execution.

In sum, the correctness requirement of Def. 4.2 guarantees that $\Pi_{\texttt{FLPSI}}$'s syntactic output only contains allowed records; the simulation-based component guarantees that no additional

Functionality $\mathcal{F}_{\mathtt{FLPSI}}^{\mathcal{L},\Pi}$

Given inputs $q \in \mathcal{D}$ of C, and $Db = \{(d_1, l_1) \ldots, (d_N, l_N)\}$ of $\mathcal{S}$, where each $d_i \in \mathcal{D}, l_i \in \mathcal{LS}$,

- trusted party runs an honest execution of the protocol Π on the players' inputs and obtains the output result set R;
- return R and $\mathcal{L}_C$ to the client;
- return $\mathcal{L}_S$ to the server.

Figure 5: The Ideal Functionality $\mathcal{F}_{\mathtt{FLPSI}}^{\mathcal{L},\Pi}$

information (beyond the above output) is revealed. Put together, this makes a composable and usable security definition. We are not aware of this definitional approach being used in prior work, and believe it can be broadly useful, especially working with biometrics.

We caution the reader that the correctness requirement – and hence our definition – are not perfect. A "bad" protocol may exploit the allowed small manipulations of probability of returning each particular record and leak unauthorized information to $\mathcal{C}$. However, in FLPSI (in contrast, e.g., to MPC of approximations [26]), correctness condition is restrictive and severely limits possible leakage: we return input DB records which cannot be modified to leak. Moreover, our definition can be further tightened, e.g. by correctness requiring that a record return probability does not change if some other DB record changes. Formal exploration of this new definitional style for fuzzy MPC is an interesting future research direction.

We parameterize the security definition with a leakage profile describing leakage of information to the parties. This is common in the searchable encryption but is somewhat novel for MPC-style definitions. Our construction will have very limited leakage to $\mathcal{C}$: a measure of closeness of $\mathcal{C}$'s input with $\mathcal{S}$'s entry it matched; no leakage in case of no match.

FORMAL FLPSI SECURITY DEFINITION. Let Π be an FLPSI protocol for a closeness domain $(\mathcal{D}, \mathrm{Cl})$ and label space $\mathcal{LS}$, defined according to Def. 4.2. Let $\mathcal{L} = \{\mathcal{L}_C, \mathcal{L}_S\}$ be the leakage profile describing leakage to $\mathcal{C}$ and $\mathcal{S}$. The ideal functionality $\mathcal{F}_{\mathtt{FLPSI}}$ is defined in Figure 5. Then, the security of FLPSI is formally defined as follows.

Definition 7.1. FLPSI Security. *We say that a protocol $\Pi_{\mathtt{FLPSI}} = (C, S)$ defined w.r.t. the closeness domain $(\mathcal{D}, \mathrm{Cl})$ is a secure FLPSI protocol (in the semi-honest model) with leakage profile $\mathcal{L} = \{\mathcal{L}_C, \mathcal{L}_S\}$, if $\Pi_{\mathtt{FLPSI}}$ securely realizes (cf. Def. B.1) functionality $\mathcal{F}_{\mathtt{FLPSI}}^{\mathcal{L},\Pi_{\mathtt{FLPSI}}}$ of Fig. 5.*

7.2 Security Theorem and Proof of FLPSI

We now formally state the security theorem of FLPSI construction, instantiated with secure 2PC protocols $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ and $(\mathcal{C}_{STLPSI}, \mathcal{S}_{STLPSI})$, and Shamir's *t-out-of-T* secret sharing scheme. We also state the leakage profile of $\Pi_{\mathtt{FLPSI}}$, then prove the security theorem of it in the semi-honest model.

Theorem 7.1. *Assume AES is a pseudorandom function (PRF). In the presence of secure (in the semi-honest model) 2PC protocols $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ and $(\mathcal{C}_{STLPSI}, \mathcal{S}_{STLPSI})$, and a* t-out-of-T *secret sharing scheme, the $\Pi_{\mathtt{FLPSI}}$ protocol of Fig. 4 is a secure (in the semi-honest model) Fuzzy LPSI protocol with leakage profile $\mathcal{L} = \{\mathcal{L}_C, \mathcal{L}_S = \perp\}$.*

Leakage $\mathcal{L}_S$ to $\mathcal{S}$ in $\Pi_{\mathtt{FLPSI}}$. There is no leakage to $\mathcal{S}$.

Leakage $\mathcal{L}_C$ to $\mathcal{C}$ in $\Pi_{\mathtt{FLPSI}}$. $\Pi_{\mathtt{FLPSI}}$ reveals to the client a measure of quality of the match with the database entry (i.e., the number of (obliviously) encrypted matching subsamples). In case of multiple matches, the client also learns which (obliviously) encrypted subsamples matched (i.e. were common) across the different matched database entries.

Recall that our initial privacy goal is to achieve *client privacy*, which is satisfied by revealing and leaking nothing to the server. Furthermore, we emphasize the leakage to the client is strictly less (and in fact much less) than the client learning the matching database entry(ies) (or its bit vector) of the server. It is easy to see that this leakage is inferred from the matching entry(ies) held by the server. Inspecting the relevant portion of the proof, it is easy to see that the Sim_C actions informed by leakage can be easily performed without $\mathcal{L}_C$, and with the knowledge of the matching database entries of the server.

In a typical scenario, where parties share all the information/data about matches (e.g., photos, name, age, etc. of a person of interest) with each other, this leakage does not have any security impact on the desired system.

Proof. For lack of space, we formally prove the security of our main protocol $\Pi_{\mathtt{FLPSI}}$ w.r.t. Def. 7.1 in Apx. C.2. □

8 Complexity Analysis of FLPSI

In Apx. D, we explain our communication and computation complexity in detail. In summary, FLPSI has $\mathrm{O}(\frac{\mathrm{NT}}{\mathrm{mB}}\ell)$ and $\mathrm{O}(\frac{\mathrm{NT}}{m})$ communication and computation complexities, respectively. In the notations, N is database size, T is number of subsamples, m is SIMD vector size, B is the size of each DB split and ℓ is the length of an item in $\mathcal{MS}$ and $\mathcal{SS}$. Note that mB could be parameterized to be (almost) equal to N (see Sect. 11.2) if we are not considering a database of, e.g., hundreds of millions of people. Then, our communication complexity would approximate to $\mathrm{O}(T\ell)$ in practice.

9 Environment and Implementation Details

We use an Azure F72s_v2 instance, which has 72 virtual cores equivalent to that of 2.7 GHz Intel Xeon Platinum 8168 CPU and 144 GB of RAM each. We also have two sets of experiments: for *fast* and *slow* network connections between $\mathcal{C}$ and $\mathcal{S}$. While the former has 500 MB/s connection with 0.5 ms latency, the latter is having 40 MB/s with 34 ms latency. We use Ubuntu 18.04 in this instance. *Note that, even though,* *our design does not require a fast network connection or high number of threads, we use above environment for creating an identical comparison setting with the state-of-the-art* [15].

We implement our protocol on top of the homomorphic encryption library SEAL v3.5 [56], through Brakerski/Fan-Vercauteren (BFV) scheme [25]. To extract embedding vectors from facial images, we use the Python implementation of FaceNet[6] (with the Inception-Resnet-V1 architecture [63]) after aligning faces, as recommended in [80].

10 Optimizing FLPSI Implementation

In addition to applying optimization tricks to compress the database and reduce the homomorphic multiplication depth in STLPSI, as explained in Sect. 5.4.2, we further optimize our protocol for better performance and accuracy as follows.

Noise reduction (NR) in binary encoding. Inspired by [7, 59, 68], the client can extract multiple face samples from a short surveillance video in order to perform noise removal. This can be done very seamlessly at some specific application scenarios. Since people cannot be completely in the same pose throughout a video recording, $\mathcal{C}$ can treat each individual frame in a video as a different sample. On the other hand, $\mathcal{S}$ can capture multiple samples per person more conveniently since it may have a controlled environment unlike $\mathcal{C}$.

In this optimization, both parties take bit vectors, generated in the binary encoding step (from Sect. 5.1) through multiple biometric readings, and majority vote over each bit. If a certain amount of them agree (e.g., at least 90 percent), they keep it. Otherwise, they cancel (zero-out) it. After eliminating noisy bits, the residual bit vector is given to the subsampling layer.

Subsample compression. Since we use AES blockcipher with 128-bit key $k_{\mathcal{S}}$, we can compress its inputs to 128 bits to avoid multiple rounds of block-ciphering. This will reduce the online communication and computation costs of the 2PC subsampling protocol from Sect. 5.2. To do this, we can effectively compress a subsample as it mostly contains zero bits, e.g., only 14-out-of-256 bits are ones in our setting, as follows. We split the bio-bit vector into 128-bit of chunks, and evenly subsample each chunk (e.g., 7-out-of-128) without colliding subsampled bits across the chunks. For instance, if 8^{th} bit is subsampled in the first chunk, we do not subsample 8^{th} bit of the second chunk. Finally, we compute the bit-wise XOR of all chunks as the compressed output.

Optimizing STLPSI (load balancing). We introduce a new optimization to balance the loads across the buckets in $\mathcal{S}$'s reconstructed database (see Sect. 5.4.2). We decrease the number of partitions, as argued next. Note that a certain subsample(s) may be the same for too many DB entries, while the rest are shared by less of them. Also notice, it is not mathematically possible to build a (Lagrange or Newton) interpolation polynomial over such *(item, secret share)* pairs, where any two *items* are the same [44]. That is why $\mathcal{S}$ has to put each of the colliding subsamples into distinct partitions, and thus there is an unavoidable lower bound on the number of partitions, and accordingly, on the computation and communication costs. In STLPSI, before building the database, $\mathcal{S}$ truncates such (subsample, secret share) pairs after reaching a certain collision threshold, which balances the load of the each bucket of its constructed coefficient table. In Sect. 11.4, we empirically show the impact of this optimization on the overall costs.

Par.	Description	Value
t	matching threshold	2
T	number of subsamples	64
$\mathcal{L}$	length of bio-bit-vectors	256
τ_{rb}	consistency threshold ratio	0.9
$\mathcal{N}_{sb}$	number of subsampled bits	14
$k_{\mathcal{S}}$	$\mathcal{S}$'s key for a AES blockcipher	$\{0,1\}^{128}$
$\mathcal{P}$	prime mod. of domain $\mathbb{F}_{\mathcal{P}}$	8519681
λ	security param. for token 0^{λ}	$\lfloor \log \mathcal{P} \rfloor$=23
N	number of database entry	[10K-10M]

Figure 6: List of parameters and their fixed values.

11 Evaluation

In this section, we extensively evaluate our protocol, and then systematically compare it to the prior art. Note that we achieve our results by applying all optimizations.

11.1 Datasets

Evaluation datasets. We use a DL model that is pre-trained on the MSCeleb1M dataset, including over 8 million unconstrained facial images of around 100 thousand identities [33].

Query set. We use the YouTube Faces (YTF) benchmark dataset, that contains noisy collections of unconstrained facial videos from 1,595 public figures [74]. Since the preprocessing may use multiple biometric scans per person to generate reliable bio-bit vectors, we randomly pick (at most) ten frames each for $\mathcal{C}$ and $\mathcal{S}$ to test ε errors. We assume $\mathcal{C}$ always queries these 1,595 people over any size of DB in our experiments.

Database set. We generate photo-realistic synthetic faces to create large-scale databases since there is no such big public datasets. We use StyleGAN [40] to create databases of 10 thousand (Face-10K), 100 thousand (Face-100K) and one million (Face-1M) identities along with the YTF identities (with isolated samples from the query set).

Comparison datasets. For our comparative analysis, we use AT&T [54] and Deep1B [3] datasets, which are used in prior art. Note that we use these datasets in the same way as they are used in the prior art. AT&T[7] includes 400 facial images from 40 people, where 8 faces of each (320 in total)

[6] https://github.com/davidsandberg/facenet

[7] https://www.kaggle.com/kasikrit/att-database-of-faces

# of false matches	FRR (%) for Plaintext / FLPSI		
	Face-10K	Face-100K	Face-1M
1	2.89/2.95	2.93/2.97	2.99/3.01
2	1.62/1.65	1.86/1.95	2.13/2.18
3	1.26/1.32	1.64/1.66	1.97/2.01
4	1.06/1.14	1.39/1.42	1.55/1.56
5	0.92/1.01	1.14/1.18	1.18/1.25
6	0.81/0.85	0.94/0.97	1.06/1.12
7	0.72/0.77	0.83/0.86	0.92/0.94
8	0.56/0.59	0.74/0.79	0.87/0.92
9	0.53/0.58	0.69/0.74	0.73/0.79
10	0.51/0.56	0.58/0.63	0.67/0.75

Figure 7: FRRs of underlying plaintext matching system and FLPSI protocol for at most 10 false matches per query errors.

are kept as database items and 2 faces of each are queried. Deep1B contains a billion image descriptors (each 96 dimension vector), which is generated by passing images through a deep neural network [3]. We use the original query set, which includes 10 thousand data points, published by the authors[8]. And, we conduct queries over two subsets of Deep1B that consist of randomly selected one million and 10 million entries (labeled as Deep1B-1M and Deep1B-10M, respectively). We treat Deep1B descriptors as embedding vectors in our pipeline since it is not a facial dataset.

11.2 Parameters

In the following, we introduce the parameters and our parameter selection process. *Note that once we fix our parameters, we use them without changing across different experiments.*

ε-correctness errors. These refer to the errors in the ε-correctness of FLPSI. Recall from the Sect. 4 that, ε_1 infers the false matches, and ε_2 infers the false non-matches (or, false rejection rate – FRR). Interpreting in our context, false matches denotes the number of different identities obtained other than the queried one, while false non-matches standing for the number of "not exist" results in response to querying existing people in the database.

In our experiments, we target to get at most 10 false matches and 1% false non-match rate for any of the database sizes, which meets accuracy requirement of the commercial systems [2, 32, 45].

Parameter choices for the targeted errors. In the following, we summarize our parameter searching method to find the ones achieving the targeted errors.

In Fig. 6, in addition to t and T, we enumerate and describe all parameters $(\mathcal{L}, \tau_{rb}, \mathcal{N}_{sb})$ required in DL, SBLSH and NR steps, which affect the errors. We first search the parameters for the plaintext baseline to see if we can obtain the targeted errors without enabling privacy-preserving blocks. We search (following Apx. E) and fix our parameters to the values in Fig. 6, then use them for all the experiments below.

Parameter choices for privacy-preserving blocks. The parameters of BFV scheme are three integers $(m_p, m_{ct}, \mathcal{P})$, where m_p is the polynomial modulus degree , m_{ct} is the ciphertext modulus and $\mathcal{P}$ is the plaintext modulus [14]. We initialize $m_p = 2^{13}$, $m_{ct} = 218$ bits and $\mathcal{P} = 8519681$ to always achieve at least a 128-bit security level as recommended in [14]. These parameters allow us to perform a standard noise flooding operation as part of our STLPSI protocol (see Sect. 5.4.2). The LWE estimator[9] by Albrecht et al. [1] suggests $128-131$ bits security level for this setting. We switch the ciphertext modulus from 218 to 55 bits in the modulus-switching step to decrease the communication size from $\mathcal{S}$ to $\mathcal{C}$. For the parameters such as standard deviation error and secret key distribution we use the default values of SEAL. We set the SIMD vector size to $m = 8192$, and the size of the token 0^{λ} to 23 bits (at most), which is the same length of the labels of database records.

Achieved errors for the fixed parameters. After fixing the parameters, we measure the errors of end-to-end FLPSI protocol to see if it holds our ε-correctness requirement. Fig. 7 shows the FRRs per query for the targeted false-matches (at most 10 per query for any DB size). Note that these error rates have implications on the confidentiality of DB, and nothing relevant to the query data, which is the first privacy goal of our protocol. As mentioned before, revealing false matches (e.g., within industrial standards [2, 32, 45]) to the client is allowed since it is unavoidable in desired application. Having said that, though FLPSI slightly increases the FRR errors compared to underlying plaintext system (due to the reason explained in Sect. 10), it still holds the correctness for all settings.

11.3 Costs of FLPSI

Fig. 8 shows experimental results of FLPSI protocol. For each database size N, it presents the storage needs and preprocessing times for the offline phase, total online communication overhead, and end-to-end online computation times for different number of threads (Th). We report total response times for the fast and slow network configurations, introduced in Sect. 9. For clarity, we discuss the results of **a single query over Face-1M** dataset in the following. We average over 100 queries for the FLPSI results.

11.3.1 Offline Preprocessing Cost of FLPSI

We run a one-time initialization phase to compile the DB from facial images. We do not include this cost in our summary tables. Our protocols refresh *t-out-of-T* secret sharings and AES blockcipher key $k_{\mathcal{S}}$ (both held by $\mathcal{S}$) per query. This is performed solely by $\mathcal{S}$ in expectation of the query. This

[8] http://sites.skoltech.ru/compvision/noimi/

[9] We use the commit fb7deba from https://bitbucket.org/malb/lwe-estimator/src/master/

Database	Offline		Online	Online response time (milliseconds)								
	Storage	Preprocess	comm.	Computation time with different number of threads							Best query	
	(MB)	time (s.)	(MB)	Th=1	8	16	32	64	72	Sp-up	fast	slow
Face-10K	5	0.94	12.1	523	93	68	**46**	57	56	11.4×	47	146
Face-100K	51	4.07	20.4	4457	635	376	257	241	**186**	24.0×	187	386
Face-1M	501	37.5	40.8	43956	5944	3058	1828	1647	**1355**	32.4×	1455	1655

Figure 8: FLPSI results (per query). The best computation times are in bold-face, and the best computation speed-ups are measured against the single-threaded results. Total response times are reported under the last two columns for fast/slow networks.

Step	Party	Run time percent
Building encrypted query	$\mathcal{C}$	3.66%
Homomorphic evaluation	$\mathcal{S}$	91.6%
Decrypting query results	$\mathcal{C}$	3.79%
Extracting matches	$\mathcal{C}$	0.95%

Figure 9: Run time percent. of steps in a query over Face-1M.

cost is easily amortized (run concurrently) with an actively executing query, and we report it as an offline cost. In our experiments, $\mathcal{S}$ needs at most 501 MB of storage and 37.5 sec. to pre-compute and buffer a copy of constructed database of 1M entries. We include buffer reading time in the following online evaluations.

11.3.2 Online Communication Cost

We have a fixed (8.5 MB per query) communication cost from obliviously extracting the subsamples of a single bio-bit-vector of the client through the 2PC $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ protocol. Hence, this cost is independent from the database size. FLPSI achieves at most **40.8 MB per query** communication cost, which shows that we are not relying on the *fast* network connection for efficiency. The last two columns of Fig. 8 show that data communications increase from 100 to 300 ms (at most) even if we switch from the fast to slow network connection. This is our major advantage compared to prior art (see Sect. 11.5). Hence, we can conclude that FLPSI is compatible with the existing network infrastructures of potential clients in the desired surveillance scenario.

11.3.3 Online Computation Cost

Even in the single-threaded execution scenario, FLPSI achieves promising performance (at most 44 seconds). Given that, since we spend most of the time for homomorphically evaluating the polynomials on the server side, as presented in Fig. 9, we can use multi-threading to speed up this computation. Note that setting up a powerful server could be more applicable than providing fast network connections (e.g., in gigabit scale) for every client. Using 72 threads achieves 32.4× faster computation compared to using a single thread. Moreover, since $\mathcal{S}$ concurrently evaluates partitions, which could be less than the number of threads for small databases, computation time does not decrease linearly (or increases) as $\mathcal{S}$ uses more threads.

Database	Communication		Response time (fast/slow)	
	(MB)	Saving	(seconds)	Speed up
Face-10K	72	6×	2.12/2.33	4.1×/3.7×
Face-100K	528	26×	17.8/21.7	4.0×/4.7×
Face-1M	2124	52×	189/199	4.3×/4.5×

Figure 10: FLPSI per query results taken *without* load balancing the server's buckets. Data communications are reduced by `saving` factors, and response times are improved by `speed up` factors with optimizations.

Best end-to-end timing: In Fig. 8, we show the best achievable response times for each of the database sizes at the last two columns. Overall, by using multi-threading, FLPSI can privately search a single person over a database of a million people in **1.46 sec.** and **1.66 sec.** with fast and slow network connections, respectively. To the best of our knowledge, this is the fastest response time compared to prior art, with similar functionality, in a desired application scenario.

11.4 Impact of Load Balancing Optimization

In the following, we explain how we decrease the overall communication and computation costs, through the optimization from the Sect. 10. To do this, we repeat the experiments *without* applying this optimization, whose results are presented in Fig. 10. Then, we compare them with those in Fig. 8. For clarity, we only report total communication overheads and single threaded response times. To show the impact of our optimization, we also report the achieved saving factors in communication and speed ups in computation costs, by comparing optimized and non-optimized results. *Overall, we reduce the communication overheads up to 52× and speed up the response times up to 4.3/4.5× on fast/slow networks.*

11.5 End-to-end Comparison with Prior Art

In this section, we systematically compare FLPSI with previous *private fuzzy matching* protocols. Considering their functionality and security guarantees for our application scenario, we group prior art in two categories: i) *threshold matching* and ii) *k-nearest neighbor search*. In (i), as in our work, $\mathcal{S}$

Protocol	Communication		Resp. time (fast)	
	(MB)	Saving	(sec.)	Speed up
FLPSI	0.39	-	0.014	-
Yasuda et al. [75]†	9.92	25.5×	1.70	121×
Huang et al. [34]†	17.9	46.0×	6.08	434×
Osadchy et al. [47]†	35.2	90.3×	99.2	7086×
Blanton et al. [5]	2.8	7.18×	9.37	669×
Barni et al. [4]†	9.11	23.4×	16.0	1110×
Sadeghi et al. [53]	2.8	7.18×	15.5	1286×
Erkin et al. [23]	7.3	18.7×	18.0	1143×

Figure 11: Comparing FLPSI with existing *distance thresholding* protocols. Communication costs and response times per query over AT&T database. †Costs are scaled for AT&T database based on reported results in cited works.

may return empty result (depending on the ε_1 error) to $\mathcal{C}$ if no close entry exists in the database. In (ii), $\mathcal{S}$ always guarantees to return k database entries to $\mathcal{C}$ regardless of the query. While (ii) is a different functionality, we compare our work with protocols in both categories, as the state-of-the-art (SANNS [15]) in (ii) is also faster than protocols in (i), and is the fastest among protocols "close enough in spirit".

As discussed earlier, we do not compare with *exact matching protocols* (e.g., (L)PSI protocols from [16,17,41,49]), as they do not support fuzzy matches. We solve a *much* harder problem than exact matching.

11.5.1 Comparison to Threshold Matching Approaches

As discussed in Sect. 2, prior art either a) applies thresholding to computed Euclidean (or Hamming and cosine similarity) distance [4,5,23,34,47,53,75], or b) runs *t-out-of-T* matching [11,18,29,76] between query and database (feature) vectors. Though they satisfy the functionality requirement and security guarantees for our application scenario, none of them propose a practically applicable system for a real-time surveillance task.

Distance thresholding approaches. Fig. 11 compares concrete costs of FLPSI to prior work [4,5,23,34,47,53,75]. Note that the cited works report communication and computation costs linear in the database size. They achieve between 1.7-99.2 sec. response times and 2.8-35.2 MB network overheads per query over AT&T database.

Further, majority of them do not satisfy our ε-correctness requirements. We achieve **121-7086**× faster response time (14 ms. per query) and **7.18-90.3**×less communication for the *same* database, while meeting our ε-correctness requirements. Note that we consider *single* threaded execution for all works, but could not execute them in the exact same environment. However, since all run on similar clock speeds, our achieved speed-ups would slightly vary on the same environment.

Protocol	Communication	Computation
FLPSI	$O(\frac{NT}{mB}\ell)\approx O(T\ell)$	$O(\frac{NT}{m})$
CEC [11]	$O(N\lvert\mathbb{F}_{\mathcal{P}}\rvert\ell)$	$O(N(\lvert\mathbb{F}_{\mathcal{P}}\rvert+T)T'_{\varepsilon})$
YSPW [76]	$O(NT^2\ell)$	$O(N(poly(T)+T^2T'_{\varepsilon}))$
CH_1 [18]	$O(NT\ell)$	$O(N(\binom{T}{t}poly(T)+TT'_{\varepsilon}))$

Figure 12: Comparing FLPSI with existing *t-out-of-T* protocols that are still considered secure. Only the dominant terms are kept for all protocols. ℓ is the size of a ciphertext in the chosen encryption scheme. T'_{ε} is the time needed for all homomorphic operations in a single cycle.

t-out-of-T matching approaches. Systems [11,18,76] (referred as CH_1[10], YSPW, CEC, resp.) are existing, secure, *t-out-of-T* protocols. Fig. 12 compares asymptotic communication and computation complexity of [11, 18, 76] to our system. FLPSI behaves better both in computation and communication than CH_1, YSPW, and CEC protocols, as both of their communication and computation complexities are linear in database size. Further, computation and communication of CEC [11] are linear also with the domain size. In concrete terms, CEC reports 3GB communication for a database of 100 T-dimension vectors, where each vector item could be one of 4 distinct letters. Thus, CEC does not scale for our case (FLPSI operates in a domain with over 2^{23} integers). CH_1 [18] and YSPW [76] do not report concrete costs.

11.5.2 Comparison to kNNS Approaches

We emphasize that "k-nearest neighbor search" protocols solve a somewhat related, yet different problem, and do not meet the security guarantees we consider. Nevertheless, we compare them to FLPSI because we wish to present a broader perspective and to illustrate that our work is more efficient not only than protocols for our exact problem, but than any prior work "close enough in spirit."

kNNS is related to FLPSI. Before discussing performance, we briefly explain the relevance of kNNS to our setting. Indeed, a protocol returning a nearest neighbor could be used to construc a (leaky) FLPSI, e.g. as follows: $\mathcal{C}$ and $\mathcal{S}$ run 1NNS. $\mathcal{C}$ obtains the output and checks if it meets the threshold of FLPSI before returning it (causing leakage to $\mathcal{C}$ if it does not). To search and return multiple matches, $\mathcal{C}$ and $\mathcal{S}$ could either proceed iteratively, increasing k by a small amount, or guess a larger k and risk higher leakage.

Performance comparison. We compare our design with Chen et al. [15]'s two protocols since, to our knowledge, they are the fastest protocols compared to all other kNNS approaches [19,35,37,61], which do not use a trusted third-party in their pipelines.

[15] show (at least) 8-31× faster response times compared to optimally implemented prior art. They propose an optimized linear scan (SANNS-linear) and an approximate search

[10] Ye et al. [76] break the security of the second protocol from [18].

Protocol	Deep1B-1M				Deep1B-10M			
	Communication		Response time (fast/slow)		Communication		Response time (fast/slow)	
	Total	Saving	(seconds)	Speed up	Total	Saving	(seconds)	Speed up
FLPSI	40.8 MB	-	1.46/1.66	-	128 MB	-	12.7/13.5	-
SANNS-linear	5.39 GB	132×	5.79/41.7	3.97/25.1×	57.7 GB	452×	73.1/446	5.76/33.0×
SANNS-approx	1.72 GB	42×	1.70/15.1	1.16/9.09×	6.07 GB	48×	5.27/41.8	0.41/3.10×

Figure 13: Comparing FLPSI to two protocols of SANNS [15]. Best achieved response times are reported for fast/slow networks.

(SANNS-approx) protocols, which are built upon additive homomorphic encryption, garbled circuits and oblivious read only memory, to conduct secure kNNS over large databases.

To conduct an almost identical comparison, we evaluate FLPSI on the same Azure instances with the same *fast/slow* network connections, as introduced in Sect. 9, and over the same image datasets: Deep1B-1M and Deep1B-10M.

Communication and computation costs. Fig. 13 compares total communication overheads and the best achieved response times through the fast/slow networks for the both database sizes. Due to our sublinear communication, FLPSI decreases required bandwidth by **132-452**× and **42-48**× (depending on the database size) compared to SANNS's linear and approximate protocols, respectively. This implies significant improvement in wall-clock time, especially on slower networks. In fact, SANNS outperforms FLPSI only on Deep1B-10M dataset, with *fast* network connection, and via its approximate algorithm. For instance, the best response time of SANNS-approx protocol increases from 1.7 to 15.1 sec. as we switch the network from fast to slow connection. Similarly, SANNS-linear's performance decreases even more in the same situation, as it has more data overhead than their approximate protocol. On the other hand, FLPSI preserves its performance regardless of the network connection, as it has 128 MB of communication overhead even for a database of 10 million entries. Overall, we achieve up to **5.8/33**× and **1.2/9.1**× faster response times compared to SANNS's linear and approximate protocols, respectively, on the fast/slow networks.

12 Conclusions

We define FLPSI, fuzzy labeled private set intersection, and propose an efficient construction. In FLPSI, client $\mathcal{C}$ holds a biometric query and server $\mathcal{S}$ holds a labeled biometric database, where labels may be, e.g., persons' identities. In FLPSI, $\mathcal{C}$ learns the label *iff* the query is in the database, and $\mathcal{S}$ will learn nothing. Our definitional approach uniquely combines the properties of game-based and simulation-based definitions, and can be useful in other settings.

Designing an efficient protocol for FLPSI is challenging mainly due to the need to manage the noisiness of biometric data. We realize FLPSI in the semi-honest model from a blockcipher, garbled circuits, secret sharing, and fully homomorphic encryption.

FLPSI achieves *sublinear* communication cost relative to the database. Our experiments show that our solution scales well to massive datasets including up to 10 million entries. Additionally, our comparative results show that i) FLPSI achieves up to 48-452× less communication cost and ii) up to 3.1/33× faster response times compared to protocols from the state-of-the-art on a database of 10 million entries. Notably, FLPSI has a major advantage over prior art by not relying on high speed network connection for efficiency.

References

[1] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[2] Android Open Source Project. Biometric security, 2020. https://source.android.com/security/biometric/measure.

[3] A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *IEEE CVPR*, 2016.

[4] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, et al. Privacy-preserving fingercode authentication. In *MM&Sec*, 2010.

[5] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*. Springer, 2011.

[6] A. Boldyreva and N. Chenette. Efficient fuzzy search on encrypted data. In *International Workshop on FSE*. Springer, 2014.

[7] K. W. Bowyer, K. Hollingsworth, and P. J. Flynn. Image understanding for iris biometrics: A survey. *CVIU*, 110(2):281–307, 2008.

[8] Z. Brakerski, C. Gentry, and S. Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *International Workshop on Public Key Cryptography*, pages 1–13. Springer, 2013.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT*, 6(3):1–36, 2014.

[10] Business Insider. https://www.businessinsider.com/senate-bill-sanders-merkley-ban-corporate-facial-recognition-without-consent-2020-8.

[11] I. Calapodescu, S. Estehghari, and J. Clier. Compact fuzzy private matching using a fully-homomorphic encryption scheme, Aug. 29 2017. US Patent 9,749,128.

[12] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. Smith. Reusable fuzzy extractors for low-entropy distributions. In *EUROCRYPT*, 2016.

[13] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

[14] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, et al. Security of homomorphic encryption. *HomomorphicEncryption.org, Tech. Rep*, 2017.

[15] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi. SANNS: Scaling up secure approximate k-nearest neighbors search. In *USENIX Security*, 2020.

[16] H. Chen, Z. Huang, K. Laine, and P. Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *CCS*, 2018.

[17] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *CCS*, 2017.

[18] L. Chmielewski and J.-H. Hoepman. Fuzzy private matching. In *ARES*, 2008.

[19] D. Demmler, T. Schneider, and M. Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[20] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, 2004.

[21] Z. Dong, C. Jing, M. Pei, and Y. Jia. Deep cnn based binary hash video representations for face retrieval. *Pattern Recognition*, 81, 2018.

[22] L. Ducas and D. Stehlé. Sanitization of fhe ciphertexts. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–310. Springer, 2016.

[23] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, 2009.

[24] D. Evans, V. Kolesnikov, and M. Rosulek. A pragmatic introduction to secure multi-party computation. *FnT Privacy and Security*, 2, 2018.

[25] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[26] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, July 2006.

[27] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, pages 1322–1333, 2015.

[28] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.

[29] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[30] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the aes circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

[31] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.

[32] P. Grother, P. Grother, M. Ngan, and K. Hanaoka. *Face recognition vendor test (frvt) part 2: Identification*. NIST, 2019.

[33] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *ECCV*, 2016.

[34] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX*, pages 331–335, 2011.

[35] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *NDSS*, 2011.

[36] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[37] P. Indyk and D. Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC*, 2006.

[38] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, pages 108–116, 2012.

[39] R. Ji, H. Liu, L. Cao, D. Liu, Y. Wu, and F. Huang. Toward optimal manifold hashing via discrete locally linear embedding. *IEEE Transactions on Image Processing*, 26(11):5411–5420, 2017.

[40] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, pages 4401–4410, 2019.

[41] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious prf with applications to private set intersection. In *CCS*, 2016.

[42] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE ICCV*, pages 2130–2137, 2009.

[43] M. Kuzu, S. Islam, and M. Kantarcioglu. Efficient similarity search over encrypted data. In *IEEE ICDE*, 2012.

[44] E. Meijering. A chronology of interpolation: from ancient astronomy to modern signal and image processing. *Proc. IEEE*, 2002.

[45] Microsoft. Biometric requirements, 2020. https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/windows-hello-biometric-requirements.

[46] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[47] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. Scifi-a system for secure face identification. In *IEEE S&P*, 2010.

[48] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, 2015.

[49] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based psi via cuckoo hashing. In *EUROCRYPT*, 2018.

[50] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on {OT} extension. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 797–812, 2014.

[51] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *Advances in neural information processing systems*, 22:1509–1517, 2009.

[52] M. S. Riazi, B. Chen, A. Shrivastava, D. S. Wallach, and F. Koushanfar. Sub-linear privacy-preserving search with untrusted server and semi-honest parties. *CoRR*, 2016.

[53] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, 2009.

[54] F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *IEEE WACV*, 1994.

[55] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE CVPR*, 2015.

[56] Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, Aug. 2020. Microsoft Research, Redmond, WA.

[57] A. Shamir. How to share a secret. *Commun. ACM*, 1979.

[58] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE S&P*, 2017.

[59] S. Simhadri, J. Steel, and B. Fuller. Cryptographic authentication from the iris. In *ISC*, pages 465–485. Springer, 2019.

[60] N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

[61] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *IEEE DAC*, 2015.

[62] J. Su, D. V. Vargas, and K. Sakurai. One pixel attack for fooling deep neural networks. *IEEE TEVC*, 2019.

[63] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[64] The Guardian. https://www.theguardian.com/technology/2020/aug/11/south-wales-police-lose-landmark-facial-recognition-case, June 2020.

[65] The Intercept. https://theintercept.com/2018/05/30/face-recognition-schools-school-shootings/, Dec. 2020.

[66] The NYT. https://www.nytimes.com/2020/01/18/technology/clearview-privacy-facial-recognition.html, June 2020.

[67] The Verge. Moscow's facial recognition system can be hijacked. https://www.theverge.com/2020/11/11/21561018/moscows-facial-recognition-system-crime-bribe-stalking, Dec. 2020.

[68] E. Uzun, C. Yagemann, S. Chung, V. Kolesnikov, and W. Lee. Cryptographic key derivation from biometric inferences for remote authentication. In *ASIACCS*, 2021.

[69] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[70] Y. Vizilter, V. Gorbatsevich, A. Vorotnikov, and N. Kostromov. Real-time face identification via cnn and boosted hashing forest. In *IEEE CVPR Workshops*, pages 78–86, 2016.

[71] J. Wang, T. Zhang, N. Sebe, H. T. Shen, et al. A survey on learning to hash. *IEEE TPAMI*, 40(4):769–790, 2017.

[72] Q. Wang, S. Hu, K. Ren, M. He, M. Du, and Z. Wang. Cloudbi: Practical privacy-preserving outsourcing of biometric identification in the cloud. In *ESORICS*, 2015.

[73] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit. `https://github.com/emp-toolkit`, 2016.

[74] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *IEEE CVPR*, 2011.

[75] M. Yasuda. Secure hamming distance computation for biometrics using ideal-lattice and ring-lwe homomorphic encryption. *Information Security Journal: A Global Perspective*, 26(2):85–103, 2017.

[76] Q. Ye, R. Steinfeld, J. Pieprzyk, and H. Wang. Efficient fuzzy matching and intersection on private datasets. In *ISISC*, 2009.

[77] X. Yi, C. Caramanis, and E. Price. Binary embedding: Fundamental limits and fast algorithm. In *ICML*, pages 2162–2170, 2015.

[78] J. Yuan and S. Yu. Efficient privacy-preserving biometric identification in cloud computing. In *IEEE INFOCOM*, 2013.

[79] C. Zhang, L. Zhu, and C. Xu. Ptbi: An efficient privacy-preserving biometric identification based on perturbed term in the cloud. *IS*, 2017.

[80] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[81] L. Zhu, C. Zhang, C. Xu, X. Liu, and C. Huang. An efficient and privacy-preserving biometric identification scheme in cloud computing. *IEEE Access*, 6:19025–19033, 2018.

[82] Y. Zhu, Z. Wang, and J. Wang. Collusion-resisting secure nearest neighbor query over encrypted data in cloud, revisited. In *IEEE/ACM IWQoS*, 2016.

A Super-Bit Locality Sensitive Hashing

Though the Euclidean space of DL accurately captures the statistical properties of the raw input data, unfortunately, even the two consequent biometric scans of a person will not result the same embeddings due to fuzzy/noisy nature of biometrics. In order to accommodate *t-out-of-T* matching, both parties translate the Euclidean space into Hamming, by using Super-Bit Locality Sensitive Hash (SBLSH) [38]. SBLSH is built on top of Sign-Random-Projection LSH (SRP-LSH) [13] , which turns input vectors into *one-bit hash* s.t. if two input vectors are close in angular distance, it is likely that their SRP-LSH will be the same. In particular, SRP-LSH is defined as $h_v(x) = sgn(v^T x)$, where x and v are d-dimensional vectors, and $sgn(.)$ is the sign function (i.e. 1 if the input is greater than or equal to 0, otherwise 0). Note, x is the input (e.g., embedding vector), and v is sampled with normal distribution.

SBLSH demonstrated that SRP-LSH can be used to turn x into $\mathcal{L}$-bit codes. It independently samples $\{v_1, ..., v_{\mathcal{L}}\}$ vectors, then for each $i \in [\mathcal{L}]$, it calls $h_{v_i}(x)$, and thus generates $\mathcal{L}$-bit codes. For details, we refer readers to [13, 38].

B Securely Realizing Ideal Functionality

In this section, we recall the standard definition of securely realizing ideal functionality in the semi-honest model [24], formulated for the 2PC case in Def. B.1.

Definition B.1. Securely Realizing Ideal Functionality. *We say that a real-world protocol* Π *securely realizes an ideal-world functionality* $\mathcal{F}$ *in the presence of static semi-honest adversaries if there exists a simulator* Sim *such that, for every corrupt party* $P_i, i \in \{1,2\}$ *and all valid inputs* x_1, x_2*, the distributions* $\mathrm{Real}_\Pi(\kappa, P_i; x_1, x_2)$ *and* $\mathrm{Ideal}_{\mathcal{F},\mathrm{Sim}}(\kappa, P_i; x_1, x_2)$ *are computationally indistinguishable (in* κ*).* Real *and* Ideal *ensembles are defined as follows:*

$\mathrm{Real}_\Pi(\kappa, P_i; x_1, x_2)$*: run* Π *with security parameter* κ*, where each party* P_i *runs honestly using private input* x_i*. Let* V_i *denote the final view of party* P_i*, and let* y_1, y_2 *be the final outputs of the two parties. Output* $\{V_i, (y_1, y_2)\}$*.*

$\mathrm{Ideal}_{F,\mathrm{Sim}}(\kappa, P_i; x_1, x_2)$*: Let* $(y_1, y_2) \leftarrow \mathcal{F}(x_1, x_2)$*. Output* $\{\mathrm{Sim}(P_i, (x_i, y_i)), (y_1, y_2)\}$*.*

C Proving the Security of STLPSI and FLPSI

In this section, we formally prove the Theorem 5.1 and Theorem 7.1. We specifically describe ideal world simulators $\mathrm{Sim}_{\mathcal{C}}$ and $\mathrm{Sim}_{\mathcal{S}}$ emulating the views $\mathrm{VIEW}_{\mathcal{C}}$ and $\mathrm{VIEW}_{\mathcal{S}}$ of $\mathcal{C}$ and $\mathcal{S}$ in the real execution for both protocols. Recall, the player's view includes its input, output, randomness, and the messages it received. The (challenging part of the) task of the simulators is to emulate the received messages in a consistent manner. Recall, a simulator Sim takes as input the simulated player's input, output and leakage (if there is any). In the following, we formally present the required simulators and the proofs of the indistinguishability of the simulated and real views.

C.1 Proving the Security Theorem of $\Pi_{\mathtt{STLPSI}}$

It is immediate that $\Pi_{\mathtt{STLPSI}}$ correctly computes the set intersection and the associated labels if the underlying Shamir's secret reconstruction succeeds, i.e. when there are at least t intersecting items between a query and database set (cf. Def. 5.1). Now, we formally prove the Theorem 5.1.

For the ease of exposition, we assume that the simulator/protocol is parameterized by $(t, T, \lambda, m, m_p, m_{ct}, \mathcal{P}, a, B)$, which are fixed and public (see Sect. 11.2), and that *t-out-of-T* secret sharing scheme (Sect. 5.3) is used.

C.1.1 Simulating the Client

Recall, $\mathrm{Sim}_{\mathcal{C}}$ takes the client's query set $\mathcal{Y} = \{y_1, \ldots, y_T\}$ and the output of the real execution (labels of the matching database sets X_i and corresponding (item, share) pairs, if at least t of them matched). To construct $\mathrm{Sim}_{\mathcal{C}}$, we first describe the real view that needs to be simulated.

Real view of $\mathcal{C}$. In Steps 1-3 and 5-6, $\mathcal{C}$ receives no messages, and thus $\mathrm{Sim}_{\mathcal{C}}$ does nothing to simulate them.

In Step 6, $\mathcal{C}$ attempts to reconstruct a label l_i (and succeeds if there is a matching one. As required by the security of STLPSI, the client should not learn any below-threshold t matches. STLPSI achieves this since the server takes a set of secret shares (for each label) as inputs, and again, Shamir's secret sharing scheme guarantees the indistinguishability of each individual share (or any below-threshold t combinations of them) from a random item in the share domain $\mathbb{SS}$, which is the same with the agreed FHE scheme's domain $\mathbb{F}_{\mathcal{P}}$.

Also note that we assume the server randomly re-generates different set of secret shares for each label before each execution of STLPSI protocol. This prevents a serious leakage, an adversary combining (possible) below-threshold t shares, which are obtained across distinct executions, to sum up enough shares reconstructing a label.

In Step 3, $\mathcal{C}$ computes and sends $\log B$ homomorphic ciphertexts to $\mathcal{S}$. In Step 4, $\mathcal{C}$ receives back a homomorphic ciphertexts, each of which is an encryption of a degree-m FHE plaintext polynomial. Crucially, the ciphertexts sent to $\mathcal{C}$ by $\mathcal{S}$ are rerandomized with high noise (noise flooding), to hide the history of ciphertext construction.

Constructing the client's simulator. We need to simulate the output of Step 4, homomorphic evaluations of intersection functions. Hence, $\text{Sim}_{\mathcal{C}}$ is defined as follows.

Recall the $\text{Sim}_{\mathcal{C}}$ has the output of the real execution. Hence, if the output is empty (there is no match), $\text{Sim}_{\mathcal{C}}$ generates a vectors, where each of them includes m random items from the agreed FHE scheme's domain $\mathbb{F}_{\mathcal{P}}$. And, if the output has a matching label l_i, $\text{Sim}_{\mathcal{C}}$ inserts its associated shares into the corresponding vector indices (which are also obtained from the output) instead of random items from $\mathbb{F}_{\mathcal{P}}$. Then, $\text{Sim}_{\mathcal{C}}$ batches each of these a vectors into a FHE plaintext polynomial and homomorphically encrypts it into a ciphertext. The ciphertext is then noise-flooded with the same noise distribution as used in the protocol. This ensures that the noise distribution in the simulated ciphertext is indistinguishable from that of the real execution. $\text{Sim}_{\mathcal{C}}$ then applies modulus switching with the same parameters as in the real execution. The resulting a ciphertexts serve as a simulation of the client's view. By the IND-CPA security assumption on the agreed FHE scheme, this view is indistinguishable from the client's view $\text{VIEW}_{\mathcal{C}}$ in the real execution of $\Pi_{\texttt{STLPSI}}$.

C.1.2 Simulating the Server

Simulating the server is straightforward. In Step 4, $\mathcal{S}$ receives $\log B$ ciphertexts, where each of them is an encryption of a degree-m FHE plaintext polynomial. $\text{Sim}_{\mathcal{S}}$ generates new encryptions of zero in place of the encryptions in this step. By the IND-CPA security of the agreed FHE scheme, this view is indistinguishable from the server's view $\text{VIEW}_{\mathcal{S}}$ in the real execution of $\Pi_{\texttt{STLPSI}}$.

C.2 Proving the Security Theorem of $\Pi_{\texttt{FLPSI}}$

In this section, we prove Theorem 7.1 of our main protocol.

C.2.1 Simulating the Client

Describing the client's view. After describing $\text{VIEW}_{\mathcal{C}}$, we explain what the simulator does to simulate the view and why this works. The simulator $\text{Sim}_{\mathcal{C}}$'s inputs are a query $q \in \mathcal{D}$, the leakage $\mathcal{L}_{\mathcal{C}}$, and the output of the real execution (label(s) of the matched biometric(s), if any match occurred).

Let q be the client's query biometric data, and y be the output bio-bit vector, computed through DL, SBLSH and NR in the preprocessing. Only y is used in the rest of the protocol.

The preprocessing stage (Step 1) and Steps 2-4 are non-interactive and the client receives no messages. Hence, $\text{Sim}_{\mathcal{C}}$ does nothing to simulate these steps.

In Step 5, $\mathcal{C}$ and $\mathcal{S}$ run MPC, where $\mathcal{C}$ inputs y, and $\mathcal{S}$ inputs $k_{\mathcal{S}}$ and $\{mask_1, \ldots, mask_T\}$. Then, $\mathcal{C}$ gets subsample set $\mathcal{Y} = \{y_1, \ldots, y_T\}$ s.t. $y_j = AES_{k_{\mathcal{S}}}(y \wedge mask_j)$. $\mathcal{C}$ receives MPC messages here, which (by the security of the underlying MPC protocol) carry no information and are simulated by the simulator guaranteed by the MPC protocol. However, the output of the MPC is something that $\mathcal{C}$ obtains in its view, and we need to simulate it.

In Step 6, the client submits the encrypted subsamples $\mathcal{Y} = \{y_1, \ldots, y_T\}$, received from Step 5, to the STLPSI protocol and gets the set of shares (and identities of the corresponding matching encrypted subsamples) as output, if there is a match (which means there are at least t matches). If there was no match, $\mathcal{C}$ receives the empty set from the STLPSI protocol. Because $\text{Sim}_{\mathcal{C}}$ is given the output, it will know whether STLPSI returns empty. However, in case a match is returned in the FLPSI protocol, we do not know how many subsamples matched. We cannot simulate this (without leakage), as it depends, e.g., on how close $\mathcal{C}$'s and $\mathcal{S}$'s matching bio-bit vectors are. Thus, this information (the number of matched subsamples in case of a match) constitutes leakage $\mathcal{L}_{\mathcal{C}}$, and $\text{Sim}_{\mathcal{C}}$ will use it for the simulation. We again emphasize that this leakage is strictly less (and in fact much less) than $\mathcal{C}$ learning the matching bio-bit vector (or the original biometric) of $\mathcal{S}$.

Constructing the client's simulator. We need to simulate the output and input of the STLPSI call in Step 6. STLPSI inputs from the client is the set of elements (simulated by random elements in the range of the AES function and which are further used in the simulation of the AES step, described next). STLPSI output to the client is a set of labels and allows to reconstruct the output of $\Pi_{\texttt{FLPSI}}$, together with the corresponding matched set elements. The $\Pi_{\texttt{FLPSI}}$ output (which is given to $\text{Sim}_{\mathcal{C}}$ as input) indicates if there was a match (or matches) and specifies the corresponding label(s).

If no match was achieved, $\text{Sim}_{\mathcal{C}}$ sets the simulated output of STLPSI to be empty.

If there was a single match over the database, $\text{Sim}_{\mathcal{C}}$ knows the label to be returned in STLPSI. It also uses leakage $\mathcal{L}_{\mathcal{C}}$ to determine how many subsamples should be returned in STLPSI. It then uses the received label l_i to generate the simulated secret shares input into STLPSI and obtained in the matched subsamples. $\text{Sim}_{\mathcal{C}}$ then randomly chooses the set elements (from the AES outputs it simulated) to be the ones resulting in matches.

The case of multiple matches is handled similarly. The only interesting difference is in simulating how many subsamples should be returned for each label l_i. This is established with the help of the leakage $\mathcal{L}_{\mathcal{C}}$.

Having constructed the simulated input and output of $\Pi_{\mathtt{STLPSI}}$, $\mathrm{Sim}_{\mathcal{C}}$ uses the client-side simulator guaranteed by the security of $\Pi_{\mathtt{STLPSI}}$, to simulate the messages exchanged as part of the Step 6. Note that the input of the protocol is distributed according to the requirements of Theorem 5.1, and hence simulation goes through.

We need to simulate messages received in the MPC call of Step 5. The output of the MPC call is the T random elements chosen by $\mathrm{Sim}_{\mathcal{C}}$ as described above. The input to the MPC call is the client's input y, which is also given to $\mathrm{Sim}_{\mathcal{C}}$. Thus, the real-world messages generated by the MPC subprotocols called in Step 5 are simulated by running the client-side simulator provided by the MPC protocol.

This completes the description of the simulator $\mathrm{Sim}_{\mathcal{C}}$. As noted above, the discussion included in the view description and the simulator construction is a direct argument of the indistinguishability of the simulated and real views.

C.2.2 Simulating the Server

Simulating $\mathcal{S}$ is significantly easier as it does not learn anything or receive any leakage in the protocol execution. The only protocol messages received by $\mathcal{S}$ are those of the calls to MPC and STLPSI in Steps 5 and 6. $\mathrm{Sim}_{\mathcal{S}}$ simulates inputs to both calls simply by following the protocol on its input, and there are no outputs to $\mathcal{S}$ in these steps. Thus, the messages received by $\mathcal{S}$ in these steps are simulated by the corresponding server-side simulators of the MPC and STLPSI.

This completes the description of the simulator $\mathrm{Sim}_{\mathcal{S}}$. The argument of the indistinguishability of the simulated and real views is immediate.

D Complexity Analysis of FLPSI

In this section, we present the computation and communication complexities wrt the database and query sizes. $\mathcal{C}$ holds a set of T subsamples for a single query, and $\mathcal{S}$ holds a database of N records with associated labels, each with T (subsample, share) pairs. Let a, B, m be the number of partitions, size of each partition and size of SIMD batching vector, respectively.

Communication complexity. FLPSI includes two interactive protocols: 2PC subsampling $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ and STLPSI $(\mathcal{C}_{STLPSI}, \mathcal{S}_{STLPSI})$. Let β be the data transmission cost for a single $(\mathcal{C}_{AES}, \mathcal{S}_{AES})$ call, then the communication complexity for the former is $O(T\beta)$, which does not depend on the database size. Let ℓ be the length of an item in $\mathcal{MS}$ and $\mathcal{SS}$, which is equal to domain of FHE scheme $\mathbb{F}_{\mathcal{P}}$, where $\ell = \log \mathcal{P}$. Then, STLPSI has $O(a\ell + T\ell) = O(T\ell(\frac{N}{mB} + 1))$ communication complexity. Since mB could be parameterized to be (almost) equal to N (see Sect. 11.2), the total communication complexity is $O(T(\ell + \beta))$ (or $O(T\ell)$ considering the dominant term) in practice. This is *sublinear* relative to the database, but linear relative to the number of subsamples.

Computation complexity. *In the offline phase*, $\mathcal{S}$ needs to interpolate $m \times a$ polynomials, each in the degree of $B = \frac{NT}{ma}$. Given that the interpolation has a $O(B^2)$ complexity, then the offline complexity is $O(\frac{(NT)^2}{ma})$ [16]. *In the online phase*, $\mathcal{S}$ homomorphically evaluates a B-degree interpolation polynomial for all partitions, which has a $O(\frac{NT^2}{m^2})$ complexity. Since $T \ll m$, we have $O(\frac{NT}{m})$ FHE operations. Moreover, $\mathcal{C}$ tries $\binom{T}{t}$ combinations among plaintext results of each partition, which brings an additional $O(\binom{T}{t}\frac{ma}{T})$ share recovery cost through *plaintext* data. Note that we fix t to a small value for all of the evaluated datasets, thus the share recovery cost does not become a bottleneck in our pipeline (e.g., only 0.95% of the query time, as reported in Fig. 9).

E Parameter Selection Process

Tuning all parameters together has its own challenges because this is a big search space to explore. Since t and T values (especially t) is also critical for the complexity of our protocol, we set $t = 2$ and search for the minimum possible T value. To achieve this, we first tune the length of bio-bit vectors. Then, we brute force the $\mathcal{N}_{sb}$ and T by targeting to the minimum errors. We also consider the threshold τ_{rb} for the ratio of reliable bits along with these parameters. Instead of brute forcing, we follow a more probabilistic approach to find its optimal value. That is, we have to guarantee that enough bits are retained at the end of the NR layer to pick T distinct subsampling functions (each has $\mathcal{N}_{sb}$ ones). Hence, 1) the number of the remaining reliable bits ($\mathcal{N}_{rb}$) should be more than the number of subsampled bits in each subsampling function ($\mathcal{N}_{rb} > \mathcal{N}_{sb}$) and 2) $\binom{N_{rb}}{\mathcal{N}_{sb}} \geq T$ inequality should to be guaranteed. Finally, we fix our parameters to the values presented in Fig. 6.