IEEE Transactions on Knowledge and Data Engineering (Volume: 28, Issue: 10, Oct. 1 2016), 07 July 2016

Privacy-Preserving Outsourced Media Search

Li Weng, Laurent Amsaleg, and Teddy Furon

Abstract—This work proposes a privacy-protection framework for an important application called *outsourced media search*. This scenario involves a data owner, a client, and an *untrusted* server, where the owner outsources a search service to the server. Due to lack of trust, the privacy of the client and the owner should be protected. The framework relies on multimedia hashing and symmetric encryption. It requires involved parties to participate in a privacy-enhancing protocol. Additional processing steps are carried out by the owner and the client: (i) before outsourcing low-level media features to the server, the owner has to one-way hash them, and partially encrypt each hash-value; (ii) the client completes the similarity search by re-ranking the most similar candidates received from the server. One-way hashing and encryption add ambiguity to data and make it difficult for the server to infer contents from database items and queries, so the privacy of both the owner and the client is enforced. The proposed framework realizes trade-offs among the strength of privacy enforcement, the quality of search, and complexity, because the information loss can be tuned during hashing and encryption. Extensive experiments demonstrate the effectiveness and the flexibility of the framework.

Index Terms—multimedia database, image hashing, indexing, content-based retrieval, data privacy, encryption.

1 Introduction

Multimedia material is nowadays everywhere on Internet. It is massively produced, distributed, and 24×7 consumed by users around the globe. As a consequence, the management of multimedia data, e.g., storage and search, is typically *outsourced* to third parties. Outsourcing offers constant availability, fault tolerance, and gigantic processing power to both data owners and users. For example, it is needed when similarity-based searches are performed on extremely large-scale databases of multimedia content. In practice, outsourcing has become a de facto standard for multimedia repositories, as exemplified by YouTube, Flickr, Picasa, etc.

Outsourcing is however raising potential privacy problems: i) data owners might involuntarily confide sensitive information to third parties; ii) third parties may profile users according to their queries. Caring for privacy suggests that user queries should not be fully known by a third party server, especially when it is not trusted. For example, in a remote diagnosis application, a patient sends medical images to a syndrome database for automatic matching. The privacy concern is that the server should not see the query (which reveals the patient's health status) but still perform the search.

This work focuses on a particular application scenario called *outsourced media search*. In this scenario, a data owner outsources the description of its multimedia data to an external server which provides search service to clients on behalf of the owner. It is typically suited for cloud storage and computing. Here, the untrusted server is a threat to the privacy of both the client and the owner. The challenge is that the server must remain capable of performing the search service and meanwhile know little about the owner's data and the client's interests. This three-party scenario is more difficult than the conventional two-party scenario. So far most existing solutions only address the latter, and cannot be easily extended.

In this work, the outsourced scenario is tackled by a novel privacy-preserving framework based on *robust hashing* and *partial encryption*. In a nutshell, database items and queries are represented by content-based hash values; the hash value of each database item is divided into two parts, one of which is encrypted. The unencrypted part is used by the server for approximate indexing and search. The encrypted part is used by the client for refined candidate ranking. Figure 1 shows a flow chart of the proposal.

Robust hashing is the key element for protecting the privacy of the owner and the client. Conventional low-level features sometimes enable the inferring of content [1], while the one-wayness of hashing makes it hard to recover original content from hash values. This concept has been successfully used in a two-party protocol [2]. In order to cope with the new scenario, another element is incorporated – partial encryption prevents the server from precisely "linking" queries and database items. At the server, a high level of ambiguity is maintained because similarity search can only be performed using unencrypted hash parts. That ambiguity must be such that similarity search remains possible and meaningful. Our framework fulfills this requirement and allows flexible trade-offs among privacy, search quality, and complexity. To summarize, the framework has the following desirable properties:

- **Scalability**: The framework gracefully works with small to extremely large-scale databases because it is compatible with state-of-the-art multimedia indexing strategies.
- **Good retrieval quality**: Search precision is dramatically reduced at the server, but compensated at the client. Enforcing privacy does not compromise retrieval quality.
- **Tunable costs**: The framework offers flexible trade-offs between privacy and computation / communication

This work was supported by the French project SecuLar under Grant ANR-12-CORD-0014.

L. Weng was with Inria Rennes - Bretagne Atlantique, Rennes, 35042 France. He is now with IGN - French Mapping Agency, Saint-Mandé, 94165 France (e-mail: li.weng@ign.fr).
T. Furon is with Inria Rennes - Bretagne Atlantique, Rennes, 35042 France (e-mail: teddy.furon@inria.fr).
L. Amsaleg is with the IRISA lab, CNRS, Rennes, 35042 France (e-mail: laurent.amsaleg@irisa.fr).

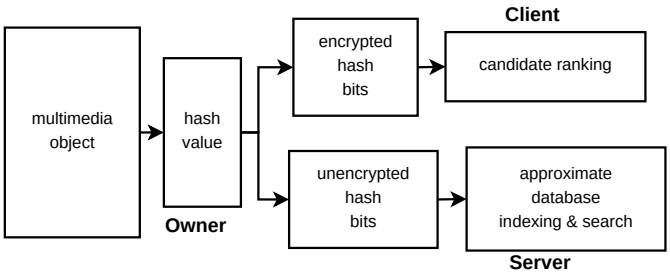

Fig. 1. A flow chart of the proposed solution.

complexity.

- **Generality**: It is generic and versatile enough to fit different applications.

In addition to the framework, we introduce the notion of "privacy gap", a novel way to quantitatively measure privacy for retrieval applications. We also define the condition to achieve the gap, which states that the hash values at the server must be much shorter than in the conventional scenario.

We demonstrate the effectiveness of the proposed framework with synthetic and real datasets containing millions of items. One main lesson is learnt: it is possible to perform efficient privacy-preserving similarity search with high-dimensional data, which is the key for large-scale applications.

The rest of the paper is organized as follows: Section 2 is a brief literature review; Section 3 defines the application scenario, the privacy requirements, and the threat model; Section 4 gives an overview of the proposed framework, while Section 5 describes the system in detail; Section 6 shows experiment results and analysis; Section 7 is about security and privacy analysis; Section 8 compares our proposal with some existing approaches; Section 9 concludes the work.

2 Related Work

Privacy-preserving similarity search focuses on the secure comparison problem: how can values be compared without revealing them [3]. Existing solutions can roughly be divided into two categories. The first category includes solutions from the cryptography community where the problem is considered as a special case of secure multi-party computation (SMC) [4]. These approaches belong to the area of Signal Processing in Encrypted Domain (SPEED) [4], [5]. They typically rely on heavy cryptographic computations, such as homomorphic encryption [6], oblivious transfer [7], and garbled circuit [8]. SPEED approaches are good at preserving privacy, but they are very complicated in terms of computation and communication, thus cannot be used in large-scale applications. They are generally slow because:

- Homomorphic encryption works with very large number representations (e.g. 1024 or 2048 bits) and uses many exponentiation operations;
- It sometimes involves significant interactions between parties and therefore large payloads of communication;
- SPEED approaches do not offer any trade-off.

Nevertheless, some existing SPEED applications deal with the search of biometric data [9], such as face recognition [10], [11], [12] or with multimedia data, such as image search [13]. Scale of their data collections remains small though. An advantage of SPEED approaches is that they can extend to more general threat models, in spite of more complexity.

The second category of solutions adopts an entirely different philosophy, namely Search with Reduced Reference (SRR) [2]. Instead of completely precluding privacy infringements, they try to make infringements computationally difficult to achieve. One possible approach is to significantly raise the ambiguity level of data collection. State-of-the-art solutions adopting such a paradigm typically enforce the *k-anonymity* and/or the *l-diversity* properties [14]. In the case of multimedia data, this can be achieved by e.g. quantizing (or compressing) the low-level descriptions of media items before creating the database. This lossy process reduces the accuracy of information stored in the database, increases collisions in indexing cells, and creates more ties in distance calculations. In turn, ambiguity makes it quite tricky to precisely infer knowledge about database items and queries.

Various contributions have followed this path. A typical approach to reduce the accuracy of information is *randomized embedding*. This dimension reduction approach turns low-level features into very compact signatures, sometimes called hash values. A widely used method is the Johnson-Lindenstrauss (J-L) embedding based on random projections [15], adopted by Lu et al. [16], Voloshynovskiy et al. [17], and Fanti et al. [18]. Another popular method (including our implementation) is the locality-sensitive hashing (LSH) [19], which is typically a quantized J-L embedding. A recently proposed method is the secure embedding [20], which is a privacy-enhanced variant of LSH. There are also deterministic ways of generating signatures. For example, Diephuis et al. use DCT sign bits [21] and Weng et al. use wavelet sign bits [2].

Existing solutions, whether SPEED or SRR, typically assume a two-party (client, server) scenario. The three-party scenario has not been addressed yet. The goal of introducing the server is to significantly reduce the burden of the owner. It is a challenge to both SPEED and SRR approaches. The challenge to SPEED is the complication in protocol design and the potentially increased amount of computation and interaction. Adding an extra party to a secure protocol is a drastic change and may completely alter previously established results. Especially in our case, the added party is an untrusted one, which further complicates the situation. Technically, it might be possible to extend a two-party SPEED solution to a three-party one, but that would inevitably involve the owner in the interaction, defeating the original goal.

SRR approaches can effectively liberate the owner, while the challenge is to control the amount of information (and thus computation) at the server. In particular, we require that the server is not able to infer knowledge from the outsourced database. This restriction does not exist in previous SRR approaches, including ours [2]. This is addressed by hashing and partial encryption. Previously we show that in the two-party scenario privacy can be achieved without sacrificing retrieval performance, but at the expense of increased computation at the server [2]. However, the same trade-off cannot be achieved in the three-party scenario, because the server has already used

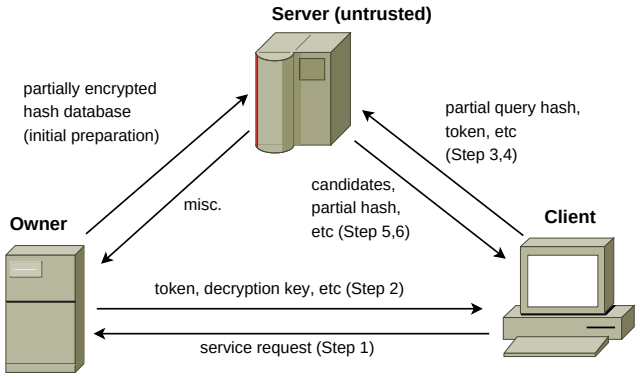

Fig. 2. The application scenario. The steps correspond to Protocol 1.

all available information provided by the owner. Instead, a new concept named "privacy gap" is introduced.

It is interesting to contrast our work with secure cloud search literature, e.g. [22]. Conventionally, secure cloud search deals with text documents, for which similarity is measured by keyword matching. Some existing solutions generate signatures, which are different from hash values. The form of the query is also different - trapdoor information is typically used. There is no tunable privacy. In addition, our two-stage search structure is different from the conventional architecture.

3 Preliminary Definitions

3.1 The Application Scenario

Outsourced media search involves three parties – a data owner, a client, and a server. The owner possesses raw multimedia material (images, videos, audio, ...), whose goal is to offer content-based search within its collection, possibly for content monetization or other profitable purposes. Due to limited resources, searching is outsourced to the third-party server, which typically offers better performance than what the owner could itself provide. After outsourcing, the client sends queries to the server, which runs searches on behalf of the owner. Figure 2 illustrates the application scenario.

The main difference between this scenario and the conventional two-party scenario lies in the role of the server – it is not trusted, thus it can neither directly access the database, nor analyze the database content; on the other hand, it should be able to perform search and reduce the load of the owner.

3.2 The Threat Model

In general, privacy involves system properties such as undetectability, unlinkability, and communication content confidentiality [23]. In our scenario, the owner and the client typically trust each other. In contrast, they do not trust the server, which is thus the main adversary. Privacy is defined for the client and the data owner with respect to the untrusted server.

Enforcing privacy of the data owner means preventing the server from knowing the raw multimedia material [1], or infer some related knowledge, such as categories of the database contents (people, nature, indoor/outdoor, ..., see [24]).

Protocol 1 Privacy-preserving outsourced media search

Parties: Client (C), Server (S), and Owner (O).
Prerequisite: Client and Server register at Owner and share a secure (encrypted) channel with Owner.
Protocol:

1: $C \rightarrow O$: Client requests privacy-preserving search service from Owner.
2: $O \rightarrow C$: Owner sends a token and a decryption key to Client, along with the query format. All the information is encrypted.
3: C : Client obtains the token and other information by decryption, which also authenticates Client's identity.
4: $C \rightarrow S$: Client sends the token and a partial query hash to Server.
5: S : Server verifies the token and performs a search.
6: $S \rightarrow C$: Server sends back the candidate list to Client, along with distance information and encrypted partial hash values.
7: C : Client decrypts partial hash values and find the best match in the candidate list.

Enforcing privacy of the clients refers to preclude the server from discovering clients' interest. This concerns not only the queries of clients, but also the answers from the server. Since queries and answers share high similarity, spying on one or the other might breach the privacy of clients.

There are more privacy threats than the ones caused by this simple query-answer client-server interaction. The server can further threaten the privacy of clients by profiling and correlating queries. By observing the queries from one client, the server might infer some knowledge, such as changes in its interest. It might also discover that several clients tend to pose quite similar queries, hence the server might identify groups of users. This should be precluded as well.

We assume that all parties behave in a curious-but-honest way. They strictly follow the protocol but they can learn from the data in their possession for their own purposes.

4 Overview

This section first gives a global overview of the proposed privacy-preserving framework, then motivates the need for robust hashing and encryption. A simple theoretical model is developed to help the understanding.

4.1 Workflow

The framework for enhancing the privacy of outsourced media search is composed of the following five steps. The first two steps are initial preparation. The rest of the steps constitute the search protocol, which is summarized in Protocol 1.

Preparation at Owner: The owner computes content descriptors from the media data that it owns. Descriptors are one-way hashed into *signatures*. Each signature is in part encrypted and in part left in-the-clear. Signatures are sent to the server.

Indexing at Server: The server indexes in a database the in-the-clear parts of the received signatures.

Querying at Client: The client computes a content descriptor from its own query media data. It is one-way hashed into a signature. Some bits of that signature are sent to the server – they form a *partial query signature*. The positions of these bits correspond to the in-the-clear bits of the signatures generated by the owner.

Searching at Server: The server runs a similarity search to identify the signatures that are most similar to the query. Similarity is computed using the received partial query signature and the in-the-clear parts of database signatures. Once identified, the encrypted parts of the most similar signatures are sent back to the client, along with the distance information computed at the server.

Refining at Client: The client decrypts the received signatures, and completes the similarity search using the received distances, the decrypted signatures, and the query signature.

4.2 Rationale behind the Framework

In traditional setups, e.g. Google Image, the server harvests media contents to create its database, and also receives image queries from clients. In a context where privacy matters, this is no longer adequate. With privacy requirements, the server should have no direct access to the media and that access should solely be restricted to the owner and/or the client. Therefore, the low-level descriptions of media items must be computed at the owner and at the client. But this is not enough; any direct access to high-dimensional feature vectors (e.g. GIST [25], SIFT [26], ...) describing the media should be restricted, as recovering the contents from such descriptors is possible [1]. Content descriptors thus cannot be directly outsourced to the server as this would potentially compromise privacy. In contrast, and in order to protect privacy, the owner and the client must transform content descriptors into *one-way signatures* using robust hashing, also called perceptual hashing [27], [28] or robust fingerprinting [29], [30].

Robust hashing maps multimedia data to compact hash values. Ideally, a hash value is a short string of equally probable and independent bits. Robust hash algorithms typically involve feature transformation, dimension reduction, and quantization. They enforce the *one-way* property that it is computationally difficult to infer the original content from a hash value. Hashing is a quantization function $h(\cdot)$ which strengthens the confidentiality of the data: quantization loss hinders the perfect reconstruction of a vector $\mathbf{d}$ from its signature $\mathbf{x}$. Moreover, function $h(\cdot)$ may also be key-dependent. Furon et al. [31] propose an embedding where an adversary cannot estimate $h(\cdot)$ after observing a limited number of pairs $(\mathbf{d}_i, h(\mathbf{d}_i))$.

On the one hand, with robust hashing, it is impossible to reconstruct a piece of content from its signature. On the other hand, similarity searches can reliably use the signatures as a surrogate for content descriptors because robust hashing results in similar signatures from similar contents. Searching with such surrogates is called privacy amplification [17] and has been utilized to enhance privacy [2], [18].

Despite hashing, the server can infer knowledge of the database by e.g. clustering the signatures received from the owner. Even though the server cannot know the nature of the contents from the signatures, identifying groups of similar signatures might threaten privacy. To make this task more complicated, way less accurate, and hence better shield privacy, the owner partially encrypts each signature before outsourcing them. A signature eventually outsourced to the server therefore contains an encrypted part as well as an in-the-clear part.

The in-the-clear parts of the signatures are used at the server when running similarity searches. For these signatures that are most similar to the partial query signature, the encrypted parts of their signatures are sent back to the client, along with the corresponding distances computed at the server and some meta data. It is up to the client to re-rank the received candidate list using the distances and the decrypted signatures.

The proposed framework forces the owner and the clients to be more proactive than in traditional settings that ignore privacy. Yet, that extra work is small compared to the overhead of any SPEED-based privacy-preserving approach.

4.3 Theoretical Motivation

A more formal motivation of our approach is presented here. Assume that a database is composed of N statistically independent items. An item $\mathbf{X}$ is a string of L (binary) symbols. A query $\mathbf{Q}$ is the result of some distortion applied to one particular item $\mathbf{X}$ in the database. Denote $I(X;Q)$ the mutual information between the symbols of the query and the item. This quantity theoretically measures how noisy is the distortion channel $\mathbf{X} \rightarrow \mathbf{Q}$. It has been proved [32], [33], [34] that no error-free search is possible if

$$\{\Delta \stackrel{def}{=} I(X;Q) - \frac{1}{L}\log_2 N\} < 0 \ . \tag{1}$$

More precisely, whatever the search algorithm, the probabilities of errors (false positives and false negatives) cannot vanish exponentially as $L \rightarrow +\infty$ if $\Delta < 0$. We regard Δ as a theoretical quantity measuring how difficult is the search of the relevant item $\mathbf{X}$ in the database of size N.

The key idea of our approach is to split the search into two rounds: a crude approximate search performed at the server followed by a refinement of the returned list of candidates at the client. The search at the server should remain efficient at scale: it is therefore unfeasible to rely on SPEED solutions. In contrast, simple values (binary strings, ints, floats) should be used at the server, facilitating comparisons, distance calculations, etc. Enforcing privacy, however, suggests that the search at the server should not be too accurate. Privacy is possible when the items relevant to a query are "hidden" in a long list of candidates. Only the client has the ability to re-rank the candidate list, allowing the truly similar elements to emerge. Our aim is thus to artificially lower the search quality at the server but do our best at the refinement stage at the client, i.e., the server should operate at a lower Δ than the client.

There are three ways to enforce a decreased Δ at the server:

- Increase N: The owner inserts dummy items (a.k.a. distractors) in the database. We then need a mechanism to signal dummy items in the list returned to the client.
- Decrease $I(X;Q)$: The client artificially decreases the mutual information by sending a noisy query $\tilde{\mathbf{Q}}$. The Markov chain $\mathbf{X} \rightarrow \mathbf{Q} \rightarrow \tilde{\mathbf{Q}}$ guarantees that $I(X;\tilde{Q}) <$

$I(X;Q)$. The client uses the true query $\mathbf{Q}$ to yield a reliable refinement.

- Decrease L: This amounts to lowering the length L at the server while the client uses the full length.

The first option increases the database size, thus is not favorable unless the database is too small. The other two options are similar in the sense that they both add noise, while the third option essentially adds quantization noise. Our proposal instantiates the third option by means of partial encryption, because it is easy to implement.

Next, we estimate the theoretical hash performance after privacy enhancement, following a similar line as in [35]. When the Hamming distance is used for hash comparison, two n-bit hash values are judged as relevant if their distance d is less than or equal to a threshold t ($0 \leq t \leq n$). The performance of a hash algorithm can be characterized by the true positive rate P_{tp} and the false positive rate P_{fp}:

- P_{tp} = Probability $\{d \leq t|$ The two images are relevant$\}$;
- P_{fp} = Probability $\{d \leq t|$ The two images are irrelevant$\}$.

Assuming the n bits are independent and each bit is a binary classifier with average performance $\{p_{tp}, p_{fp}\}$, the performance of a general scheme can be formulated as:

$$P_{tp} = f(p_{tp}, n, t) \quad (2)$$

$$P_{fp} = f(p_{fp}, n, t) \ , \quad (3)$$

where

$$f(p, n, t) = \sum_{k=n-t}^{n} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \ . \quad (4)$$

In our scenario, the decision making is a two-stage process – first by the server and then by the client. Therefore, the overall hash performance (at the client) is defined as:

$$P_{tp,client} = P_{tp,server} \cdot P_{tp} \quad (5)$$

$$P_{fp,client} = P_{fp,server} \cdot P_{fp} \ , \quad (6)$$

where

$$P_{tp,server} = f(p_{tp}, n', t') \quad (7)$$

$$P_{fp,server} = f(p_{fp}, n', t') \ . \quad (8)$$

The privacy-preserving protocol essentially influences the system performance by Eqn. (5) to (8). In particular, privacy is achieved by setting $n' << n$. The performance also depends on the indexing strategy at the server: for linear scan, $t' = n'$; for hash table look-up, $0 \leq t' < n'$.

5 Architecture of the System

This section details the workflow outlined above.

5.1 Preparation at Owner

The content owner has to perform a series of actions beforehand, off-line, in order to prepare a dataset that will eventually be outsourced to the server.

First, the owner has to compute content descriptors from the media data. Such descriptors are typically high-dimensional vectors. The description $\mathbf{d}$ of one multimedia item is then hashed or quantized into a compact signature $\mathbf{x} = h(\mathbf{d})$. A signature is assumed to be a string of L equally probable and independent symbols. In the sequel we denote by $\mathbf{x}_i$ the signature of the i-th multimedia item. The owner then processes all its hashed signatures and splits each $\mathbf{x}_i$ into a public part $\mathbf{x}_{p,i}$ of L_p bits and a secret part $\mathbf{x}_{s,i}$ of length $L_s = L - L_p$. This latter bit string is encrypted with the owner's secret key: $\mathbf{s}_i = \mathsf{enc}(\mathbf{x}_{s,i})$. The data eventually outsourced to the server is the set of pairs $\{\mathbf{s}_i, \mathbf{x}_{p,i}\}_{i=1}^{N}$.

5.2 Indexing at Server

The server receives a collection of outsourced signatures from the owner. As the role of the server is to efficiently perform similarity searches, it has to index these signatures. The server is of course free to use whatever high-dimensional indexing scheme. Indexing schemes based on the traditional inverted lists using the Bag-of-Words model can be used [36], as well as other schemes such as LSH [19] or the PQ-Code approach [37]. The server might also use a simple exhaustive and sequential search process if the scale is sufficiently small.

The server inserts in its index the parts $\mathbf{x}_{p,i}$ of all the signatures, $\mathbf{s}_i$ being metadata linked to the i-th item.

In our implementation, we consider linear scan and hash table look-up. The latter is suitable for large-scale cases. It works as follows. The L_p bits of $\mathbf{x}_{p,i}$ are divided into n groups of smaller sub-strings, each sub-string being of length ℓ where we assume $\ell = L_p/n$. Then we create n hash tables each having 2^ℓ buckets. Each table acts as an inverted index for one particular group. The buckets of one hash table thus contain the identifiers of the signatures they are associated with.

5.3 Querying at Client

At query time, a client first computes the content descriptor $\mathbf{q}$ from its query media, and then one-way hashes this descriptor into a signature $\mathbf{y} = h(\mathbf{q})$, which is split into two parts $\mathbf{y}_p$ and $\mathbf{y}_s$ in the same way as the owner did for database signatures. The string $\mathbf{y}_p$ forms the querying data sent to the server.

5.4 Searching at Server

The server receives from the client $\mathbf{y}_p$ which is then used to probe the index in search for similar contents. Regardless of the actual indexing and retrieval method, the server eventually builds a list of candidate signatures from the database that have been found similar enough to the query, assuming that meaningful similarity computations are possible between $\mathbf{y}_p$ and $\{\mathbf{x}_{p,i}\}_{i=1}^{N}$.

In our implementation, a temporary candidate list $\mathcal{L}_t$ is first built according to the particular indexing scheme. For linear scan, all database items are in $\mathcal{L}_t$. For hash table look-up, the query $\mathbf{y}_p$ is split into n sub-strings, each probing the relevant hash table; the identifiers stored in the matching buckets are put into $\mathcal{L}_t$. Some variants of this indexing scheme use more than one matching bucket per hash table: multi-probe approaches determine the best mp buckets to probe for each table. Quality of results improve at the cost of extra processing.

The list $\mathcal{L}_t$ is then parsed to retrieve the signatures $\{\mathbf{x}_{p,i}\}_{i\in\mathcal{L}_t}$ and compute Hamming distances:

$$D_{server,i} = d_H(\mathbf{x}_{p,i}, \mathbf{y}_p). \quad (9)$$

The final candidate list $\mathcal{L}$ is composed of the indices in $\mathcal{L}_t$ whose Hamming distance is below a threshold. Another option is to re-rank items in $\mathcal{L}_t$ according to their distance $D_{server,i}$ and include in $\mathcal{L}$ the indices of the top k smallest distances. In the end, the server sends back to the client the list of triplets $\{i, D_{server,i}, \mathbf{s}_i\}_{i\in\mathcal{L}}$.

Since the server evaluates the similarity of the signatures only with the restricted information $\mathbf{x}_{p,i}$, the candidate list might contain a number of false positives. It might also fail to include true positives, because e.g. the indexing method missed relevant signatures due to the harshly estimated similarity.

5.5 Refining at Client

The client receives the list of candidates from the server. It then decrypts all the $\mathbf{s}_i$ parts from this list to obtain $\{\mathbf{x}_{s,i}\}_{i\in\mathcal{L}}$ with the secret key shared by the owner. The client has now access to some extra information about the hash values of the candidate list, allowing it to refine the results from the server:

$$\begin{aligned} D_{client,i} &= D_{server,i} + d_H(\mathbf{x}_{s,i}, \mathbf{y}_s) \quad (10)\\ &= d_H(\mathbf{x}_i, \mathbf{y}). \quad (11)\end{aligned}$$

This refinement is equivalent to using the full information of the signatures. Note that the client never knows the complete hash $\mathbf{x}_i$ because the server only sends back $D_{server,i}$.

It is then easy to run an exhaustive search on the short list. It is likely to filter out a large amount of false positives in the original list $\mathcal{L}$; the false negatives cannot be recovered.

5.6 Discussion

We discuss here several key details that matter for designing the complete system.

5.6.1 Client–Owner Relationships

Our architecture assumes some cooperation between the client and the owner. The client adopts not only the same media description scheme but also the same encryption scheme as the owner. In a preliminary protocol, the back office of the owner registers trusted clients and shares the decryption key via a secure communication channel.

Note that we generally do not consider owner data privacy at the client, because in many retrieval applications the client is able to access the owner's data. The search procedure is just a step to ease the retrieval. Sending back D_{server} is mainly for efficiency. Since hash values reveal a very small amount of information about the data, our solution also has the potential to work with special cases where the owner data is totally confidential. If a higher privacy level is required, the software for the refinement procedure can be enhanced by the owner to further reduce information leak. The goal is to prevent the client from accessing the hash values during the refinement. This can be achieved by code obfuscation [38] and/or trusted computing techniques [39]. In addition, hash values can be periodically updated to alleviate the problem.

5.6.2 Lengths of L_p and L_s

Recall that the one-way hashed description is a signature $\mathbf{x}$ of L symbols. The signature is subsequently divided into $\mathbf{x}_s$ and $\mathbf{x}_p$ whose respective lengths are L_p and $L_s = L - L_p$.

Length L_p is a key element setting the trade-off between privacy and utility at the server. Compared with L, a large value for L_p (many bits are in-the-clear) enables the server to determine a high-quality list of candidates, which is likely to contain most of the true positives and few false positives. But in this case privacy might be endangered: the server may easily infer knowledge by clustering the signatures in its database. In contrast, a small value for L_p might severely alter the quality of the candidate list – a lot of false positives appear in $\mathcal{L}$. False negatives are also likely because true neighbors might be missed more easily. In this case, however, privacy is preserved as higher-level information is more complicated to infer.

The trade-off between the quality of the candidate list built at the server and the privacy threats is central to our architecture. We explore later the impact of the ratio L_p/L in terms of server's and client's retrieval performance.

5.6.3 Candidate List Construction

The size of the candidate list is also a factor impacting the quality of search. Obviously, regardless of L_p, if the server returns to the client all the items in the database, then quality is maximum. But this is not practical. It is costly to send so much information over the communication channels, and it triggers heavy computations at the client.

The candidate list $\mathcal{L}$ must therefore be short enough to remain efficient but long enough to include useful candidates and maintain client privacy. Because hash values are compact, we can afford much longer candidate lists than in conventional systems. The minimum length depends on the privacy requirement. This is further discussed in Section 7.2. As said earlier, the server can either fix a threshold or return the top k similar elements. We face here the traditional division between bounding the similarity or bounding the number of the candidates. In this paper we explore the latter option.

5.6.4 User Management and Quality of Service

The proposed protocol naturally supports multiple clients (users). Since the privacy threat is the server, only one encryption key is used for the hash database. Legitimate clients can obtain the decryption key after entity authentication. In practice, the token based service access can be achieved by e.g. Kerberos [39]. The owner in fact acts as the authentication server and the ticket-granting server.

In order to improve quality of service (QoS), it is possible to encrypt the hash database using multiple keys. Each key corresponds to a different retrieval/privacy trade-off. The owner can either provide several copies of partially encrypted hash databases with different encryption levels, or provide one copy of partially encrypted hash database with different parts encrypted by different keys. The client can request a particular QoS level when asking for a token. The QoS level is flagged in the token and the corresponding key(s) is inserted. The server may search in different databases according to the QoS level. This flexibility facilitates various business models.

6 EXPERIMENTS

This section gives the experimental results. Two families of experiments are proposed here. We first present experiments using synthetic data in order to achieve two goals: (i) thoroughly discuss the ability of the framework to enforce privacy of the owner and the client; (ii) introduce the archetypal shape of the precision/recall graphs when using the system. We then use a real dataset comprising 5 million images and show that privacy can be enforced in reality. We start by describing how signatures are created at the owner and how ambiguity can be imposed at the server.

6.1 Generating Signatures

The LSH framework is used in the experiments for generating the signatures from high-dimensional image descriptors. It is a well known embedding technique. Specifically, our implementation is based on Charikar's algorithm [40]. The embedding from $\mathbb{R}^d \rightarrow \{0,1\}^L$ is computed as follows: the j-th bit of $\mathbf{x} = h(\mathbf{d})$ is given by

$$x_j = \begin{cases} 1 & \text{if } \mathbf{d}^\top \cdot \mathbf{r}_j \geq 0 \\ 0 & \text{if } \mathbf{d}^\top \cdot \mathbf{r}_j < 0 \end{cases}, \tag{12}$$

where $\mathbf{r}_j \in \mathbb{R}^d$ is a random Gaussian vector. The Hamming distance between two signatures estimates their angle:

$$\theta(\mathbf{d}_1, \mathbf{d}_2) \approx \frac{\pi}{L}.d_H(h(\mathbf{d}_1), h(\mathbf{d}_2)). \tag{13}$$

This generates a binary signature of length L for each media item at the owner. Then, each signature is partially encrypted, L_p bits remaining in-the-clear and L_s bits being encrypted.

We apply LSH to both synthetic and real data in our experiments. In particular, LSH is used together with feature segmentation for real data, which means the feature vector is first divided into segments and each segment is hashed separately. This not only improves the discrimination performance, but also enhances privacy protection, as shown in Section 7.2.

6.2 Imposing Database Ambiguity

The owner sends the server N signatures, each made of L_p in-the-clear bits and L_s encrypted bits. Intuitively, the privacy of the owner is enforced when $L_p < \lfloor \log_2(N) \rfloor$. In this case, the number of bits to encode the clear part of the signature is small enough to *impose ambiguity* at the server when creating the database, i.e., it is guaranteed that several signatures have identical L_p bits. In contrast, when $L_p \geqslant \lceil \log_2(N) \rceil$, there might be enough bits to uniquely identify each of the N signatures by only observing the $\{\mathbf{x}_{p,i}\}_{i=1}^N$ parts. It is trivial for the owner to decide the appropriate value for L_p given N, and the consequences of the imposed ambiguity at the server are immediate and follow the well-known $\log_2$ rule. For example, if $N = 10,000,000$ then any value for L_p greater or equal to 24 might raise the risk to give the server a fair amount of unique $\mathbf{x}_p$'s, while values below 23 create confusion. Of course, it is statistically possible that some ambiguity among the signatures remains at the server if the owner leaves in-the-clear more than $\lceil \log_2(N) \rceil$ bits, as observed in the experiments.

6.3 Datasets

6.3.1 Synthetic Data

We first randomly generate $10,000$ vectors of 512 dimensions using i.i.d Gaussian variables. We then hash each vector and create 32-bit signatures, i.e. $L = 32$. These hash values are used as seeds to create the synthetic dataset. To this end, 100 modified copies of each seed are created, and the resulting one million signatures form the data collection of the owner. Modified copies of seeds are generated by randomly modifying each 32-bit hash value up to 50%. The owner then partially encrypts these one million signatures and outsources them to the server. To evaluate the quality/privacy trade-off, a ground truth is created by randomly picking $1,000$ seeds and recording the identifiers of the corresponding $1,000 \times 100$ modified signatures. At search time, precision and recall are computed on the basis of this ground truth, i.e., how many modified signatures can be found when the client queries the collection with one seed. Two comments are in order. First, the seeds are not in the data collection used at search time. Second, the ground truth is not constructed on the basis of distance computation, but on the basis of relevance – which signatures are derived from which seeds. Therefore, *by construction*, it is possible for the distance between one seed and one of its modified signatures to be large since half of the bits can be changed. It is therefore unlikely that a recall of 1 can be reached at the server when searching its dataset for signatures that are most similar to queries. Observing the recall when no encryption is applied thus gives the baseline for the performance when experimenting with this dataset.

6.3.2 Real Data

We also created a second dataset with real images for a *near-duplicate image search* scenario. This real dataset is built upon the validation set of ILSVRC'12 which consists of $50,000$ original images.[1] We expand this image collection by creating a series of near-duplicates. Specifically, we apply to each image 15 different families of incidental distortion, each with 7 levels of strength, as detailed in [2, Table III]. Overall, this results in 5.3 million images. These images are described using 512-dimensional GIST feature vectors [25]. These vectors are reduced to 256 dimensions using PCA in order to increase robustness. The owner then hashes the reduced feature vectors. In particular, feature segmentation is used – each feature vector is divided into 8 segments; each segment is hashed to produce 16 bits. This results in 128-bit signatures ($L = 128$), which are partially encrypted before being outsourced to the server.

The following experiments mainly focus on the encryption parameters. To evaluate the quality/privacy trade-off, a ground truth is created by randomly picking $1,000$ original images and recording the identifiers of the corresponding near-duplicates. The GIST vectors of these $1,000$ images are computed and hashed by the client in the same way as the owner, and partially used as queries. At search time, precision and recall are computed on the basis of this ground truth, i.e., how many near-duplicates can be found when the client queries

1. http://www.image-net.org/challenges/LSVRC/2012/

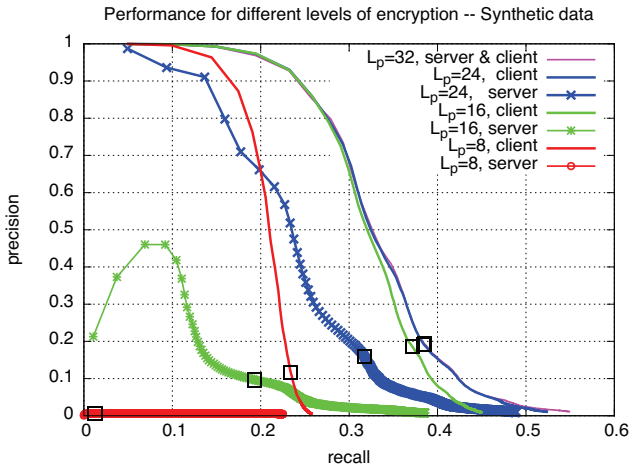

Fig. 3. Retrieval performance vs. encryption (linear scan). There is a "privacy gap" between "client" and "server". The square marks correspond to top 200 ranks, see Fig. 4-5.

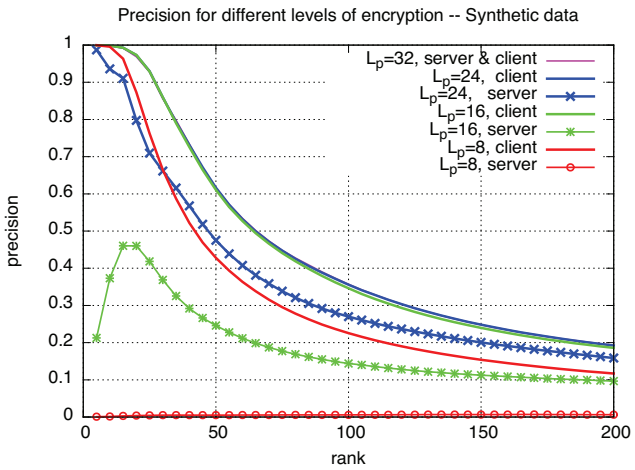

Fig. 4. Precision vs. encryption (linear scan). Encryption decreases precision; "client" always outperforms "server".

the collection with one original image. This dataset is denoted as GIST5M.

6.3.3 *Performance Metrics*

Most experiments produce precision/recall (P-R) curves to show the quality/privacy trade-off at the server as well as at the client. The P-R performance at the server is directly related to the number of bits that are left in the clear by the owner. The fewer bits, the worse performance at the server. In contrast, the client has full information once signatures are decrypted, making re-ranking on distances profitable. Performance at the client is hence typically much better. All the graphs for the experiments therefore show pairs of P-R curves, one plotting the performance at the server, the other at the client. The best answers at the server and at the client are compared with the ground truth. The P-R curves are plotted by varying the number of answers from 5 to $5,000$ with a step of 5. They are best viewed in color.

6.4 Experiments with Synthetic Data

We conducted a first series of experiments using the synthetic dataset. Recall that the signatures outsourced to the server are $L = 32$ bits long and that the database contains $1,000,000$ items. We start with the server performing a linear scan of the data collection. Hash table look-up is discussed later.

6.4.1 *Search with Linear Scan*

These first experiments aim at understanding the impact of L_p and L_s. The resulting P-R curves are shown in Fig. 3. We first set L_p to 32. In this case there is no encryption, allowing to observe baseline results, which are identical at the server and at the client. Note that there are enough bits to fully encode one million signatures. We gradually decrease L_p to 24, 16, and 8. In these cases, the reduced numbers of in-the-clear bits start to create ambiguity at the server. With $L_p = 8$ (75% of the signature is encrypted), the server is highly confused as

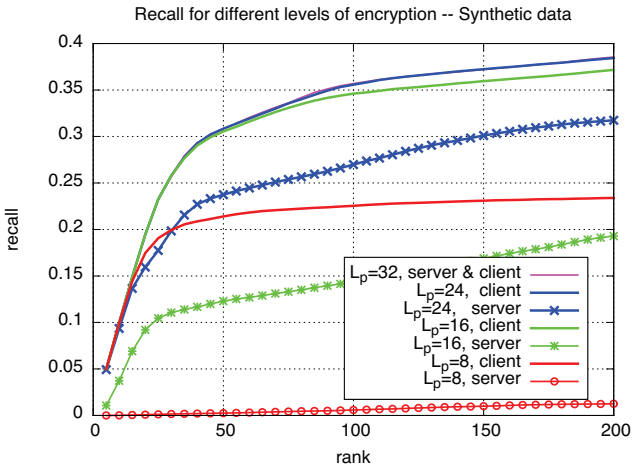

Fig. 5. Recall vs. encryption (linear scan). Encryption decreases recall; "client" always outperforms "server".

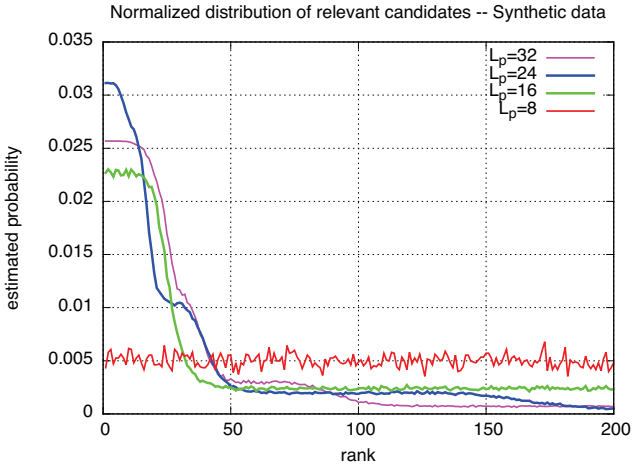

Fig. 6. Normalized distribution of candidates among top ranks at the server (linear scan). Encryption tends to flatten the distribution.

many signatures look identical, which in turn creates numerous distance collisions between items. The situation improves with $L_p = 16$, yet quite some ambiguity exists. With $L_p = 24$, enough bits are in theory left in the clear to encode one million signatures, but ambiguity is still observed in practice. For all curves but the one where $L_p = 32$ (no encryption), the gap between a pair of P-R curves shows the difference in performance between the server (low performance) and the client (high performance). We call it the "*privacy gap*".

Thanks to much more accurate distance information, the client always performs better than the server for a given precision or recall. Due to its poor retrieval performance, the server cannot properly identify items that are relevant to the query, thus client privacy is protected. We observe that the privacy gap increases with the encryption level. When $L_p = 16$, it is interesting that the server's curve is no longer monotonic, contradicting the normal behavior of a retrieval system. Typically the precision should decrease with the number of results, because relevant items tend to concentrate in the front of the candidate list. While in this case, more relevant items are in the middle of the list due to the ambiguity, so the precision slightly improves as the number of results increases (see Fig. 4). This probably makes no sense in another retrieval scenario, but it is indeed what we expect to achieve – to confuse the server. When $L_p = 8$, the precision at the server is close to 0 while the client performs quite well, thanks to its profitable re-ranking. However, note that the performance of the client also degrades with the strength of the encryption. The trade-off between the enforcement of privacy (encryption) and the quality of retrieval is well demonstrated here.

More detailed results are shown in Fig. 4-5, where the precision and the recall are individually plotted for 200 best ranked results (instead of $5,000$). The corresponding range is also marked in Fig. 3 to facilitate correlation. They show that the proposed mechanism can effectively reduce both precision and recall for the server. In general, the privacy gap decreases with the rank. Figure 6 plots the estimated probability that a relevant candidate appears at a particular rank, knowing this rank is below 200. With no or little encryption, i.e. large L_p, correct answers are more likely to appear at top ranks. Thus the plots have a high peak for top ranks. In contrast, when encryption is strong ($L_p = 8$), correct answers are uniformly distributed within the first 200 ranks. In this case, the server is confused. It has small probability to identify the correct answers, showing that privacy is enforced.

6.4.2 Search with Hash Table Look-up

We ran similar experiments when the server uses a hash-based mechanism for indexing, which accelerates retrieval compared with the linear scan used above. Hash-based indexing proceeds as follows: the server splits the L_p bits of the outsourced signatures in groups of $l = 8$ bits. Each group is used to build a hash table. At search time, the server splits the query signature in the same way, and builds $\mathcal{L}_t$ by merging the candidates in the matching bucket(s) of the hash table(s). Then it parses the list and returns all candidates to the client, i.e. $\mathcal{L} = \mathcal{L}_t$.

When signatures are not encrypted ($L_p = 32$), the server uses 4 hash tables. With encryption, the server uses only 3, 2, or 1 hash table(s) corresponding to L_p set to 24, 16, or 8. Which bits should be encrypted does not matter much as hash values are almost equiprobable bits, as demonstrated in [2].

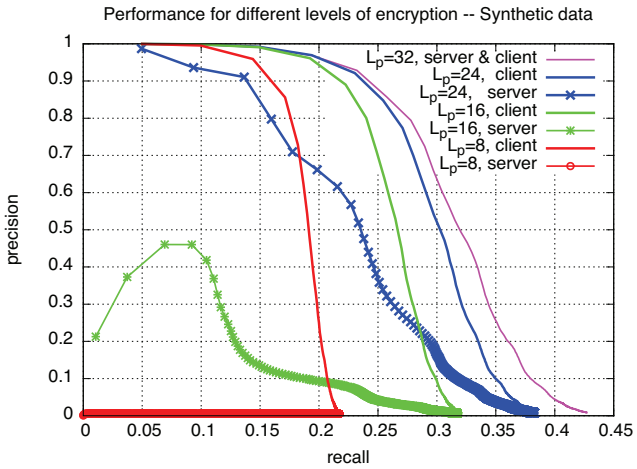

Fig. 7. Retrieval performance vs. encryption level (hash table look-up). The privacy gap still exists.

Figure 7 shows the P-R curves when using hash table look-up. The privacy gap still exists with similar quality/privacy trade-offs. Therefore, the privacy protection mechanism is independent of the server's internal retrieval method. However, note that the recall is lower compared with the linear scan case, as hash-based indexing might fail to identify correct answers due to bucket allocation.

6.5 Experiments with Real Data

We conducted a second series of experiments to observe the behavior of the framework when using signatures generated from real images. Recall that the signatures outsourced to the server are $L = 128$ bits long and that the database contains 5.3 million items. We start with the server running a linear scan. Hash table look-up is discussed later.

6.5.1 Search with Linear Scan

The first experiment with real data shows the performance when $L_p = 128, 40, 32, 24, 16$. The server is set to return the top 0.5% items to the client. There is no encryption when $L_p = 128$, and this gives the baseline performance. Due to the size of the database, severe ambiguity at the server is *guaranteed* when $L_p = 16$. In practice, ambiguity is also likely for higher values. The resulting pairs of P-R curves are shown in Fig. 8, where the privacy gaps exist too. They are directly linked to the value of L_p. The results here are consistent with the ones in the synthetic data experiments, which proves the versatility of the privacy enforcement framework. Compared with Fig. 3, the results in Fig. 8 are better for both the server and the client, because larger L_p values are used; for example, 24 clear bits are sufficient to encode the 5.3 million signatures. Note that the recall is much improved and gets close to one. That is because near-duplicates tend to be close to their original counterpart after robust hashing.

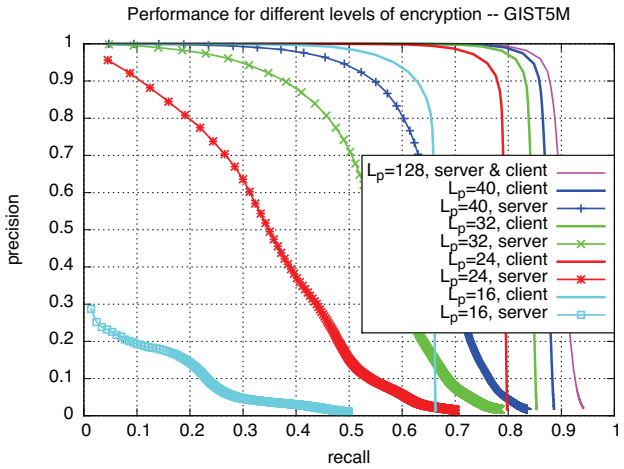

Fig. 8. Comparison of P-R curves with real data (linear scan, L_p bits unencrypted). The privacy gaps are consistent with the synthetic data experiment.

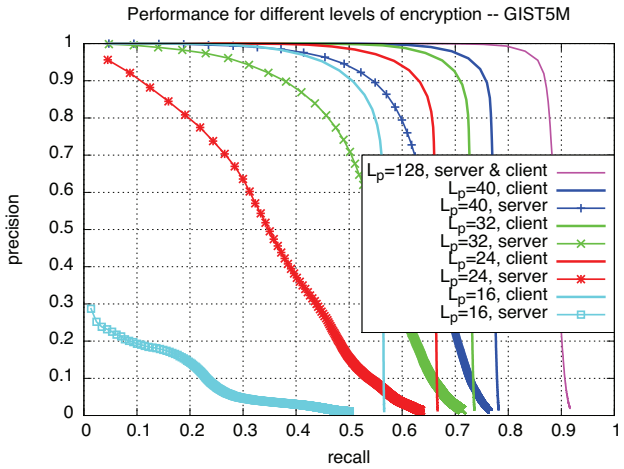

Fig. 9. P-R curves with real data (hash table look-up, 8 bits per table, L_p bits unencrypted). The gaps are smaller than in Fig. 8 – a trade-off between privacy and indexing.

6.5.2 *Search with Hash Table Look-up*

We ran similar experiments when the server uses a hash-based mechanism for indexing. Here again the L_p bits of the outsourced signatures are split into groups of 8 bits. When $L_p = 128$, the server uses 16 hash tables. That number drops to 5 when $L_p = 40$, and goes down to 2 when $L_p = 16$. Figure 9 plots the P-R curves when L_p varies. As before, the performance is controlled by the strength of encryption.

One important remark must be made. With hash-based indexing, the value of L_p directly influences the number of candidates in $\mathcal{L}_t$, hence the contents of $\mathcal{L}$ that is returned to the client. Hash-based indexing creates 2^8 buckets per hash table, so each bucket contains roughly $5.3 \times 10^6/2^8 \approx 21,000$ items. When $L_p = 16$, two hash tables are created at the server for indexing. Probing them at search time creates a list $\mathcal{L}_t$ of about $42,000$ candidates. When $L_p = 128$, there are 16 hash tables to probe, which results in about $336,000$ candidates in $\mathcal{L}_t$. Figure 9 shows the performance when $\mathcal{L} = \mathcal{L}_t$, i.e., all candidates found in the matching buckets are returned to the client. Previous observations still hold, but the gaps look smaller than in Fig. 8. This implies a trade-off between privacy and indexing efficiency. Nevertheless, returning many candidates to the client might cause too much overhead (network, processing, ...). We therefore measured the performance by varying the size of the list $\mathcal{L}$. Figure 10 illustrates the impact of sending the top p_x candidates from $\mathcal{L}_t$, where p_x is set to 100%, 10%, 5%, 1%, and 0.5%. This figure corresponds to $L_p = 24$. When $p_x = 5\%$, the server only sends about $6,000$ candidates to the client, but the resulting performance is close to the original one, for which about $63,000$ candidates are sent. Figure 11 shows a similar experiment, but with $L_p = 16$. There are fewer hash tables and thus fewer candidates compared with the previous case. The performance difference at the client between different settings becomes more noticeable. Selecting top 5% now only returns less than 25% of relevant answers, which is worse than in the previous case. Overall, this shows another trade-off between retrieval performance and communication cost.

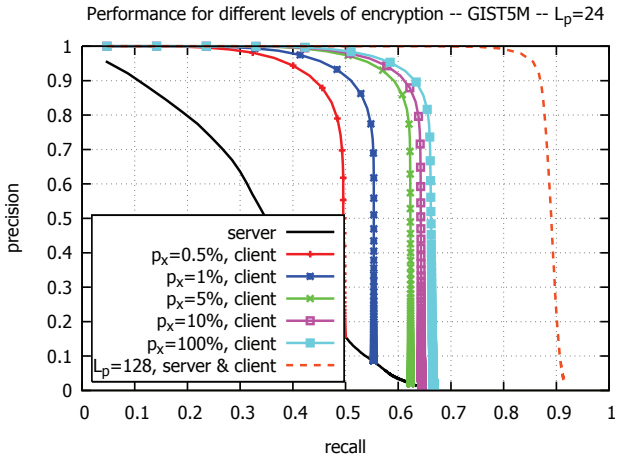

Fig. 10. P-R curves with only top p_x candidates returned to "client" (3 hash tables, $L_p = 24$). A trade-off between retrieval performance and communication cost.

7 Security and Privacy Analysis

The security and privacy provided by our proposal are computational. Without full encryption, the hash values do leak some information. However, our framework can be acceptable, as shown in the following.

7.1 Security analysis

Since standard encryption is used, predicting the encrypted hash bits is computationally difficult for the server. On the other hand, the unencrypted hash bits might reveal some information about the original content. In the following, we analyze the security threat from a reconstruction point of view. We assume that the server's goal is to reconstruct feature vectors from their hash values. Given a feature vector f and

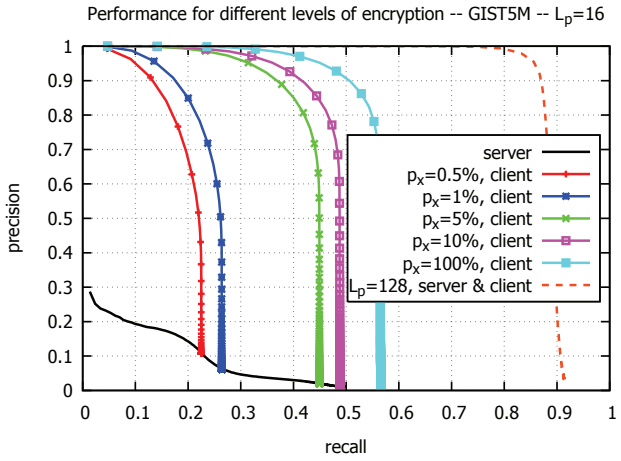

Fig. 11. P-R curves with only top p_x candidates returned to "client" (2 hash tables, $L_p = 16$). A trade-off between retrieval performance and communication cost.

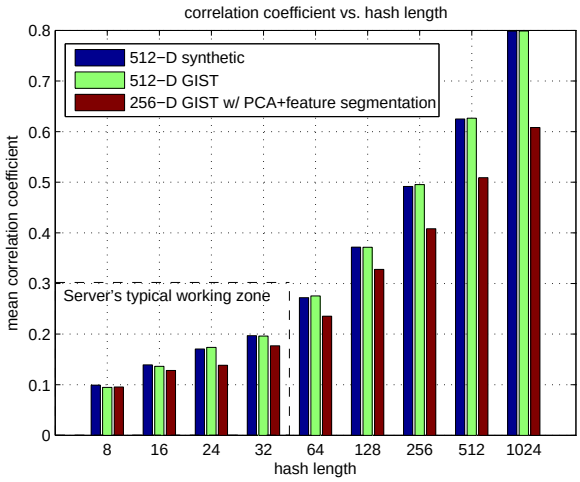

Fig. 12. Correlation coefficient between original and reconstructed vectors.

a random projection matrix $P \in R^{n \times d}$, the hash values are computed by

$$h = \text{sign}(P \cdot f) \ . \tag{14}$$

Assuming $h \in \{1, -1\}^n$, when the hash length is not larger than the feature dimensionality, i.e. $n \leq d$, the reconstructed feature vector can be represented as

$$\hat{f} = P^T \cdot h \ . \tag{15}$$

Otherwise, the reconstruction is achieved by

$$\hat{f} = P^\dagger \cdot h \ , \tag{16}$$

where $P^\dagger = (P^T P)^{-1} P^T$ is the pseudo-inverse of P in a least-square sense. In the following, we distinguish two cases depending on the knowledge of P.

7.1.1 The projection P is known

When the server knows P, Eqn. (15) or (16) can be directly used. However, the server is limited by two factors: 1) The magnitude information is lost due to the sign operation in Eqn. (14); 2) only partial hash bits are accessible. We evaluate the quality of reconstruction by the correlation coefficient between original and reconstructed feature vectors. The average results are shown in Fig. 12. They are obtained using 50 thousand GIST features and synthetic Gaussian data. We notice that the reconstruction quality improves with the hash length. There is no significant difference between synthetic and real data. Since the server can typically access less than 32 bits per hash, at best its correlation coefficient is less than 0.2. That is a satisfactory result considering that privacy is more about ambiguity rather than confidentiality.

7.1.2 The projection P is unknown

In this case, the server needs to find out the projection matrix P first. The original P is a $n \times d$ random (typically Gaussian) matrix, but the server only needs to know part of it corresponding to the public hash length L_p (i.e. $n = L_p$), denoted by P'. The high entropy of P' can typically thwart brute-force attacks. It is more feasible for the server to consider known plaintext attacks or chosen plaintext attacks, which require feature-hash pairs. According to [41], if hash values are not binarized, the number of pairs required to find out P' is on the order of L_p. In practice, this is more difficult and no existing work is found. In our scenario, if the owner and the client do not reveal any feature data, it is difficult for the server to obtain such pairs. Furthermore, the owner might regularly update the encryption key to thwart plaintext attacks.

7.2 Privacy analysis

If a database of N items is indexed by L-bit hash values, on average each hash bucket has $N/2^L$ items. If $L \geq \log_2 N$, the server is likely to find a unique match with the query. When the available hash length is reduced to L_p, the number of candidates in a hash bucket is approximately increased by 2^{L_s} times, which implies that the privacy level is magnified by 2^{L_s} times. In order to maintain a constant privacy level, we might require $N/2^{L_p} \geq k$, which can be linked to the notion of k-anonymity [14].

The privacy gap gives an intuition about the server's incapability in terms of retrieval performance. How does it further influence privacy protection in a long run? In order to gain more insights, we define the following scenario. The feature vectors are partitioned into M clusters. Given a query hash, the server's goal is to find out which cluster the query belongs to. In practice, the clusters can be compared to categories of commercial products. A user profile can be defined as a user's preference over different categories. Targeted recommendations are typically given according to profiles. Thus the server's capability in predicting a query's cluster is directly linked to long-term privacy. We assume that the hash values of the M centroids are public. The server can predict a query's cluster by comparing the unencrypted part of the query hash with the corresponding part of each centroid hash. Figure 13 shows some experiment results with

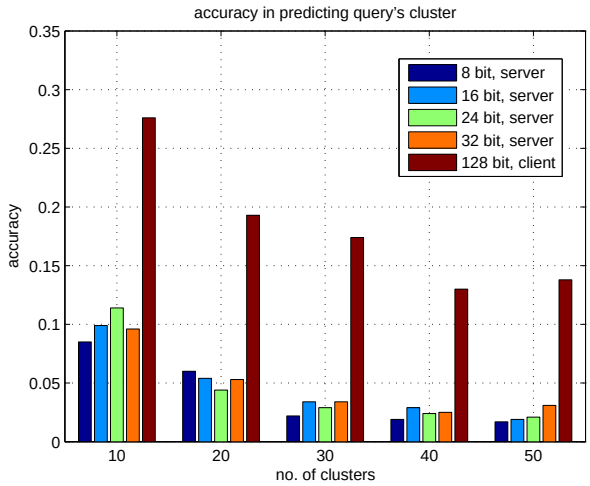

Fig. 13. Accuracy of query cluster prediction by hash comparison with feature segmentation.

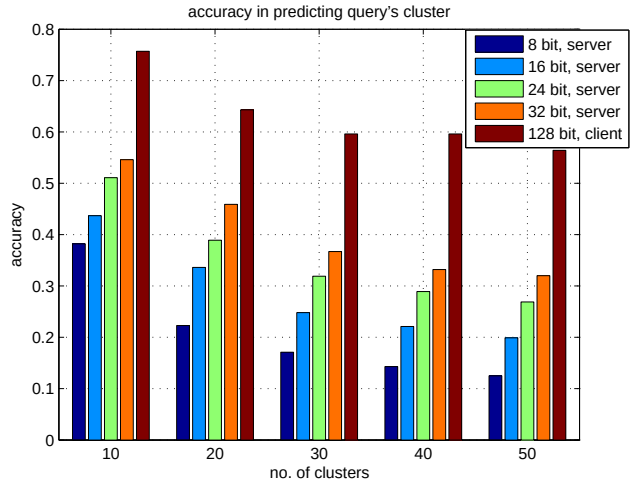

Fig. 14. Accuracy of query cluster prediction by hash comparison without feature segmentation.

50 thousand GIST vectors (256-D after PCA). We set the number of clusters M from 10 to 50. This number could be larger in practice. However, prediction is easier for small M. For the server, at best its accuracy is below 0.15, which is sufficiently low to guarantee poor profiling accuracy. We also show the client's performance in the same figure. In fact, this task is also difficult for the client, whose best accuracy is below 0.3. This is due to the limited hash length and feature segmentation (i.e. divide the feature vector into parts and hash them separately). Figure 14 shows the prediction results without feature segmentation. The accuracy is improved but still poor for the server. We can conclude that hashing with feature segmentation is "privacy friendly", which blurs the feature representation; partial encryption further prevents the server from accurate profiling.

8 Compare with State-of-the-Art

In this section, we give an empirical comparison between our proposal and SPEED approaches in terms of computation and communication complexity. Although this comparison is not exactly accurate due to several practical reasons, it suffices to give an approximate idea about the difference. To the best of our knowledge, no SPEED approach has been proposed for outsourced media search – all existing SPEED approaches assume a two party (client-server) scenario. Furthermore, they do not offer any trade-off.

In general, SPEED approaches consist of three parts: feature transformation, distance computation, and match finding. Existing approaches typically share some similarity in the distance computation part. A typical distance metric is the squared Euclidean distance. Denoting the feature dimensionality by d, this metric roughly requires d squares, d exponentiation operations, $d-1$ multiplications, and one public key encryption for each database item [10]. All these operations work with large numbers represented by 1024 to 2048 bits. Hashing approaches, on the other hand, only require d XOR operations for d-bit hash values. This explains why SPEED approaches are generally slower than their hashing counterparts. Some examples are given below.

We consider the privacy-preserving face recognition by Erkin et. al [10] as a baseline. It is an eigenface-based recognition system implemented by homomorphic encryption. Their experiment involves 320 database records and 80 query items. Specifically, each database item is represented by a 12-dimensional feature vector. During a query, the query image is first projected to 12 eigenface vectors; then the Euclidean distance is computed between the query and each database item, followed by ranking. It typically takes about 40 seconds to compare a query with all database items. An average accuracy of 96% can be achieved.

Our experiment with the same dataset shows that a similar accuracy level can be achieved by a hash-based approach. Specifically, each database item is represented by a 256-bit hash value. During a query, the query image is first projected to 128 eigenface vectors; then a 256-bit hash value is generated by LSH; finally the Hamming distance is computed between the query hash and each database item, followed by ranking. We assume that the server sends all hash values of the database to the client. It only takes about 0.3 ms to compare a query with all database items. However, our simulation does not include symmetric decryption. The speed of software implemented symmetric encryption such as AES is about 100 MB per second in 2009.[2] Taking decryption into account, our solution take approximately 0.5 ms, which is still 80,000 times faster than [10].

The time cost of [10] can be reduced to 18 seconds by pre-computation. Sadeghi et. al [11] further improved the efficiency by replacing part of the protocol with garbled-circuit, which results in 15.5 seconds. Osadchy et. al [12] proposed a different secure face recognition algorithm based on facial feature vocabularies. This algorithm exhibits better recognition performance than the eigenface approach. However, it takes

2. http://www.cryptopp.com/benchmarks.html

0.3 seconds to compare a query with a database item (i.e. 96 seconds for a database of 320 items). Note that these large values do not include the time for offline preprocessing, which can be even more significant. Recently, Sěděnka et. al proposed another secure protocol for biometric comparison [42]. They reported various time costs according to different settings. Regardless of the recognition performance, their protocol at best takes about 23 ms per comparison. Our solution is still about 15,000 times faster.

Table 1 summarizes the comparison between the aforementioned methods. We can observe a trade-off between computation and communication among SPEED methods – the faster, the more communication, and vice versa. Above all, our solution is more efficient than others by several orders of magnitude. Although it is a rough comparison, the significant difference shows that our solution is a better choice for large-scale applications, at least in the multimedia domain.

9 Conclusion and Discussion

Outsourced media search is a challenging three-party scenario, where the privacy of the owner and the clients should be protected against an untrusted server. We address this scenario by a privacy protection framework. The framework requires the owner to hash its multimedia items and partially encrypt the resulting hash values, which are then outsourced to the server. At the client, multimedia items are also hashed and some specific hash bits are used to form the queries to the server. Hashing impedes the reconstruction of multimedia material by its one-wayness, which prevents the server from learning about the queries. On the other hand, encryption leaves only few bits usable at the server, which lowers search precision, and makes clustering-related tasks less accurate. Furthermore, encryption precludes the server from relating queries with candidates. At the client, the coarse results received from the server are filtered. This is essentially a re-ranking process applied to candidate signatures after decryption.

Extensive experiments show that partial encryption generally reduces the retrieval performance of the server, and the performance drops faster when more bits are encrypted. There is always a gap between the client's and the server's performance. This "privacy gap" plays as the guard for privacy and prevents user profiling by introducing uncertainties.

If search performance can be slightly sacrificed, the cost for privacy can be further reduced. We consider reducing the number of candidates by selecting the top matches at the server. Results show that good performance can be maintained with a small proportion of candidates. By tuning the number of returned candidates, flexible trade-offs can be achieved between privacy and communication cost.

Our proposal divides the search into two stages and requires cooperation between the server and the client. This principle generally applies to SRR approaches. The difference lies in how the division is achieved. Our novel combination of hashing and encryption enables flexible trade-offs among privacy, retrieval performance, and complexity. In practice, the privacy level is determined by the available information at the server and the database size; the retrieval performance is mainly determined by the available information at the client. When the performance requirement changes or the database grows, the hash lengths can be adjusted accordingly. Since the number of representable items grows exponentially with the hash length, our solution is suitable for large-scale applications.

In particular, our work does not presume whether owner data is accessible to the client or not, because it works in both cases. Nevertheless, for the more strict scenario (i.e., the database is totally confidential), the privacy guarantee is somehow debatable and application-dependent, because of the information leak. This might be interesting for future study.

References

[1] P. Weinzaepfel, H. Jégou, and P. Perez, "Reconstructing an image from its local descriptors," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2011, pp. 337–344.

[2] L. Weng, L. Amsaleg, A. Morton, and S. Marchand-Maillet, "A privacy-preserving framework for large-scale content-based information retrieval," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 1, pp. 152–167, Jan. 2015.

[3] S. Rane and P. Boufounos, "Privacy-preserving nearest neighbor methods: comparing signals without revealing them," *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 18–28, 2013.

[4] R. Lagendijk, Z. Erkin, and M. Barni, "Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 82–105, 2013.

[5] Z. Erkin, A. Piva, S. Katzenbeisser, R. Lagendijk, J. Shokrollahi, G. Neven, and M. Barni, "Protection and retrieval of encrypted multimedia content: when cryptography meets signal processing," *EURASIP Journal on Information Security*, p. 20, 2007.

[6] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptology–EUROCRYPT'99*. Springer, 1999, pp. 223–238.

[7] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *Proc. of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001, pp. 448–457.

[8] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. of 27th Annual Symposium on Foundations of Computer Science (FOCS)*, Oct. 1986, pp. 162–167.

[9] J. Bringer, H. Chabanne, and A. Patey, "Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends," *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 42–52, 2013.

[10] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Proc. of 9th International Symposium on Privacy Enhancing Technologies (PET)*, 2009, pp. 235–253.

[11] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Proc. of 12th International Conference on Information Security and Cryptology (ICISC)*, 2009, pp. 229–244.

[12] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, "SCiFI - A system for secure face identification," in *Proc. of IEEE Symposium on Security and Privacy (SP)*, 2010, pp. 239–254.

[13] P. Sabbu, U. Ganugula, S. Kannan, and B. Bezawada, "An oblivious image retrieval protocol," in *Proc. of IEEE International Workshop on Advanced Information Networking and Applications*, 2011, pp. 349–354.

[14] M. Gertz and S. Jajodia, Eds., *Handbook of Database Security - Applications and Trends*. Springer, 2008.

[15] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," in *Conference in modern analysis and probability (New Haven, Conn., 1982)*, ser. Contemp. Math. Providence, RI: Amer. Math. Soc., 1984, vol. 26, pp. 189–206.

[16] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling search over encrypted multimedia databases," in *Proc. SPIE, Media Forensics and Security*, vol. 7254, Feb. 2009.

[17] S. Voloshynovskiy, O. Koval, F. Beekhof, and T. Pun, "Conception and limits of robust perceptual hashing: towards side information assisted hash functions," in *Proc. of SPIE*, vol. 7254, Feb. 2009.

[18] G. Fanti, M. Finiasz, and K. Ramchandran, "One-way private media search on public databases: The role of signal processing," *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 53–61, 2013.

TABLE 1
Comparison with SPEED. The measure is based on a single comparison between a query and a database item.

method	building block	computation	communication	No. of feature dimensions	implementation
Erkin 2009 [10]	HE (homomorphic encryption)	125 ms	2.6 MB	12	C++
Sadeghi 2009 [11]	HE, GC (garbled circuit)	48.4 ms	2.5 MB	12	Python
Osadchy 2010 [12]	HE, OT (oblivious transfer)	300 ms	120 KB	30	Java
Sedenka 2015 [42]	HE, GC, OT	23 ms	14.2 MB	9	Java
Our proposal	LSH, symmetric encryption	1.6 μs	256 bits	128	Matlab

[19] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008.

[20] P. Boufounos and S. Rane, "Secure binary embeddings for privacy preserving nearest neighbors," in *Proc. of IEEE International Workshop on Information Forensics and Security (WIFS)*, 2011, pp. 1–6.

[21] M. Diephuis, S. Voloshynovskiy, O. Koval, and F. Beekhof, "Robust message-privacy preserving image copy detection for cloud-based systems," in *Proc. of International Workshop on Content-based Multimedia Indexing (CBMI)*, 2012.

[22] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[23] G. Danezis and S. Gürses, "A critical review of 10 years of privacy technology," in *Proc. of 4th Surveillance & Society Conference*, 2010.

[24] L. Liu and L. Wang, "What has my classifier learned? Visualizing the classification rules of bag-of-feature model by support region detection," in *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[25] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vision*, vol. 42, no. 3, pp. 145–175, May 2001.

[26] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal on Computer Vision*, vol. 60, no. 2, 2004.

[27] F. Khelifi and J. Jiang, "Perceptual image hashing based on virtual watermark detection," *IEEE Transactions on Image Processing*, vol. 19, no. 4, pp. 981–994, April 2010.

[28] H. Özer, B. Sankur, N. Memon, and E. Anarim, "Perceptual audio hashing functions," *EURASIP Journal on Applied Signal Processing*, vol. 12, pp. 1780–1793, 2005.

[29] A. Varna and M. Wu, "Modeling and analysis of correlated binary fingerprints for content identification," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1146–1159, Sep. 2011.

[30] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *Journal of VLSI Signal Processing Systems*, vol. 41, no. 3, pp. 271–284, Nov. 2005.

[31] T. Furon, H. Jégou, L. Amsaleg, and B. Mathon, "Fast and secure similarity search in high dimensional space," in *Proc. of IEEE International Workshop on Information Forensics and Security*, Nov. 2013, pp. 73–78.

[32] F. Willems, T. Kalker, S. Baggen, and J.-P. Linnartz, "On the capacity of a biometrical identification system," in *Proc. of IEEE International Symposium on Information Theory*, 2003.

[33] S. Voloshynovskiy, O. Koval, F. Beekhof, and T. Pun, "Robust perceptual hashing as classification problem: decision-theoretic and practical considerations," in *Proc. of IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2007, pp. 345–348.

[34] P. Moulin, "Statistical modeling and analysis of content identification," in *Information Theory and Applications Workshop*, Feb. 2010, pp. 1–5.

[35] L. Weng, I.-H. Jhuo, M. Shi, M. Sun, W.-H. Cheng, and L. Amsaleg, "Supervised multi-scale locality sensitive hashing," in *Proc. of International Conference on Multimedia Retrieval (ICMR)*, June 2015.

[36] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2003.

[37] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, 2011.

[38] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. of International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, Nov. 2010, pp. 297–300.

[39] W. Stallings, *Cryptography and Network Security*, 4th ed. Prentice Hall, 2005.

[40] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. of 34th annual ACM Symposium on Theory of Computing (STOC)*, 2002, pp. 380–388.

[41] Y. Mao and M. Wu, "Unicity distance of robust image hashing," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 462–467, Sep. 2007.

[42] J. Sedenka, S. Govindarajan, P. Gasti, and K. Balagani, "Secure outsourced biometric authentication with performance evaluation on smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 384–396, Feb. 2015.

Li Weng Li Weng received his PhD in electrical engineering from KU Leuven (Belgium) in 2012. He worked on encryption, authentication, and hash algorithms for multimedia data. He then worked at University of Geneva (Switzerland) and Inria (France) on large-scale CBIR systems with emphasis on privacy protection. He is currently a post-doctoral researcher at IGN - French Mapping Agency. His research interests include multimedia signal processing, machine learning, and information security.

Laurent Amsaleg Laurent Amsaleg received his PhD in June 1995 from the University of Paris 6, France. He worked on relational and object-oriented databases, garbage collection, micro-kernels and single-level stores. He then spent 18 months in the Database group of the University of Maryland (MD, USA), designing flexible database query execution strategies (Query Scrambling). Subsequently, he received a full-time research position at CNRS in France and joined the IRISA lab in Rennes. His research focuses on very large scale high-dimensional indexing as well as on the security and privacy dimensions of multimedia content.

Teddy Furon Teddy Furon received his PhD in signal and image processing from the Ecole Nationale Suprieure des Télécommunications de Paris, Paris, France, in 2002. From 1998 to 2001, he was a Research Engineer with the Security Lab of Thomson, Rennes, France, working on digital watermarking for copy protection. He continued working on digital watermarking as a Postdoctoral Researcher at the TELE Lab of the Université Catholique de Louvain, Louvain-la-Neuve, Belgium. He also worked in the Security Lab of Technicolor. He is at present a Researcher with the LinkMedia team in the Inria Research Center, Rennes, France. Dr. Furon served as Associate Editor of the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and of the Elsevier Digital Signal Processing Journal. He was an elected member of the IEEE Information Forensics and Security Technical Committee.