CAMsure: Secure Content-Addressable Memory for Approximate Search

M. SADEGH RIAZI, MOHAMMAD SAMRAGH, and FARINAZ KOUSHANFAR,
University of California San Diego

We introduce CAMsure, the first realization of secure Content Addressable Memory (CAM) in the context of approximate search using near-neighbor algorithms. CAMsure provides a lightweight solution for practical secure (approximate) search with a minimal drop in the accuracy of the search results. CAM has traditionally been used as a hardware search engine that explores the entire memory in a single clock cycle. However, there has been little attention to the security of the data stored in CAM. Our approach stores distance-preserving hash embeddings within CAM to ensure data privacy. The hashing method provides data confidentiality while preserving similarity in the sense that a high resemblance in the data domain is translated to a small Hamming distance in the hash domain. Consequently, the objective of near-neighbor search is converted to approximate lookup table search which is compatible with the realizations of emerging content addressable memories. Our methodology delivers on average two orders of magnitude faster response time compared to RAM-based solutions that preserve the privacy of data owners.

CCS Concepts: • **Security and privacy** → **Database and storage security**; *Security in hardware*; • **Information systems** → *Information storage systems*; • **Hardware** → Power and energy;

Additional Key Words and Phrases: Privacy-Preserving Computing, Approximate Search, Content-Addressable Memory, In-Memory Computation, Near-Neighbor Search

ACM Reference format:
M. Sadegh Riazi, Mohammad Samragh, and Farinaz Koushanfar. 2017. CAMsure: Secure Content-Addressable Memory for Approximate Search. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 136 (September 2017), 20 pages.
https://doi.org/10.1145/3126547

1 INTRODUCTION

The ongoing development of technology has led to the generation of a massive volume of data. Today's cloud servers contain a database of information that belongs to users (clients) across the world. Such central storage of information enables clients to search for their content of interest in the database. This user-cloud search model appears in a broad variety of applications such as online recommender systems [1], face recognition [2], secure biometric authentication [3], to name a few. For example, in online dating websites, the server maintains a database of all clients' profiles. The

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2017 and appears as part of the ESWEEK-TECS special issue. We thank reviewers for their insightful comments. We would also like to show our gratitude to Mohsen Imani for his valuable suggestions in this work. This work is supported in parts by ONR (N00014-11-1-0885) and MURI (FA9550-14-1-0351) grants.

Authors' addresses: M. S. Riazi, M. Samragh, and F. Koushanfar, 9500 Gilman Dr, La Jolla, CA 92093; emails: {mriazi, msamragh, fkoushanfar}@ucsd.edu.

search algorithm aims to find the most similar profile in the database that best matches a specific query. The output of the search is the ID of the most similar profile. In this context, the search algorithm is approximate by nature since the similarity metric is defined heuristically.

The database might contain private and sensitive information, hence, the data has to be handled securely. For instance, in the case of online dating websites, users may prefer not to disclose their private attributes to the cloud server. As another example, consider biometric authentication where the fingerprint of a user is captured and matched against the valid profiles on the server. If an attacker can hack the server and get access to the database, the security of the system and the privacy of users are both diminished. It is worth mentioning that, once the sensitive data is compromised, the attacker can use this information to fool any other fingerprint-based authentication system. Recent data breaches of giant Internet companies such as Yahoo [4] and Google [5] have demonstrated that cloud servers are vulnerable to internal and external attacks. Therefore, scalable methodologies should be developed, both in software and hardware, to guarantee the security of the query, search result(s), and the data stored on the server.

Content Addressable Memories (CAMs) have conventionally been used for fast and efficient packet forwarding in Internet routers [6]. CAM can be viewed as hardware engine for efficient lookup table search. Unlike traditional algorithmic solutions that require sequential access to memory elements, CAM provides a hard-wired architecture that searches the entire memory in a single clock cycle. Asides from their usage in networks, emerging methodologies have been proposed to use CAMs for the purpose of in-memory computing, aiming to improve the energy efficiency and performance of conventional processors [7–10]. This means that CAMs will be tightly coupled with the main processing elements of emerging computers, hence, their security is of great importance. Nevertheless, to the best of our knowledge, there has been no prior work to address the security of CAM. A naive solution for security is storing encrypted data in CAM. This approach is not feasible since the nature of semantically secure encryption mechanisms (e.g., AES) implies that the similarity of two elements is not preserved in the ciphertext domain. Hence, one cannot use the encrypted data to perform the approximate search using CAMs.

The existing solutions for secure approximate search mostly involve computationally expensive cryptographic operations. The underlying cryptographic protocols of these schemes are either Homomorphic Encryption [11] or Yao's Garbled Circuits [12]. The high computation and communication burden of these methods hinder their practical implementation for data-intensive applications. Order-Preserving Encryption (OPE) [13] allows comparison over encrypted. However, NNS solutions based on OPE and Deterministic Encryption (DTE) are proven to be inseucre by Naveed et al. [14]. Protocols based on Asymmetric Scalar-Product-preserving Encryption (ASPE) [15] have also been proven to be insecure against the Chosen Plaintext Attack introduced by Yao et al. [16]. Another line of research focuses on encryption-free lightweight randomized embeddings that provide security at the cost of low search quality [17]. This solution adds specific noise to the raw data in order to reduce the information leakage. Although this approach is computationally less intensive, the added randomness significantly reduces the search accuracy.

This paper proposes CAMsure, a lightweight solution for securing emerging CAMs in the context of approximate search. Instead of writing the raw data on CAM, our approach stores distance-preserving hash embeddings to provide data privacy. The hashing scheme that is utilized in CAMsure is called Locality Sensitive Hashing (LSH) [18]. This family of hashing methods creates a randomized embedding of data while preserving the pairwise similarity. LSH was initially proposed to reduce the computational complexity of search by converting high-dimensional data into a compact binary string (hash value). However, it is shown that original LSH schemes are vulnerable to certain attacks as a malicious party can infer some information about the secret data based on its hash value [19]. The vulnerability originates from the fact that LSH schemes preserve the pairwise similarity of *any* two input values. Recently, a new secure LSH has been proposed to mitigate the information leakage [19]. The hash values generated from this new LSH have a low Hamming distance only if the original elements are very similar (more than a specific threshold T_S). As a result, they are secure against known attacks.

LSH translates the problem of approximate search to finding a hash within the database that has the smallest *Hamming Distance* (HD) to the query. The theory of LSH guarantees that similar elements in the data domain will have a small Hamming distance in their corresponding hash embeddings (with a high probability). The approximate lookup nature of emerging CAM technologies can enable approximate search on the hash values. As a result, CAMsure can be realized with a minimal alterations in CAM [8], i.e., voltage overscaling. Our explicit contributions are as follows:

- Introducing CAMsure, the first realization of secure in-memory computation using CAM in the context of approximate search. Our proposed secure CAM supports dynamic insertion and deletion of records without compromising system efficiency and privacy of data owners. CAMsure ensures data confidentiality even if the database is compromised.
- Proposing an end-to-end system that connects state-of-the-art hashing schemes to content-based memory architectures. Our methodology avoids costly computations of previously proposed secure search solutions by leveraging in-memory computation techniques. Another feature of this approach is that it allows lightweight deployment of secure search over distributed networks.
- Providing comprehensive analysis on search latency, power consumption, precision, and privacy guarantees of CAMsure and illustrating the practicality of our approach on a real-world dataset. As opposed to conventional RAM-based solutions, CAMsure has single cycle search latency, resulting in up to two orders of magnitude faster search while preserving the privacy of data owners.

2 PRELIMINARIES AND DEFINITIONS

2.1 Content-Addressable Memory

Conventional Random Access Memories (RAMs) offer an address-based access to the data. During a read operation, a RAM takes an address and outputs the corresponding content. In RAM-based architectures, the data is physically separated from the processor. Current processors need to fetch the data from the memory, thus, they suffer from the overhead of data movement in different levels of the memory hierarchy.

Another trend of memory architectures offers content-based access to the data [20]. Unlike conventional RAMs, CAMs store a table of contents and identify the index of a row in the table that matches the query. In other words, CAMs take the content as the input and output the address in which the content is stored. This memory architecture searches the entire table in a single clock cycle, hence, it provides several benefits in terms of search speed and energy consumption.

Figure 1 presents a schematic view of the CAM architecture, which comprises an input buffer, a set of stored words, and an address encoder. Each stored word, called a match-line, is composed of multiple CAM cells that are responsible for holding bit values. In a search operation, the buffer distributes the input (search key) among all the words (rows) in the CAM. The output voltage of a cell is discharged if the stored value is not equal to the corresponding bit in the input query. If all cells within a row match the input query, the output voltage of the whole match-line remains high and triggers the address encoder. The address encoder then generates the matching index based on the triggered match line.

The focus of this paper is to provide the security of CAM's contents. The functionality of CAM is to perform search operations; thus, we employ the CAM architecture to implement efficient

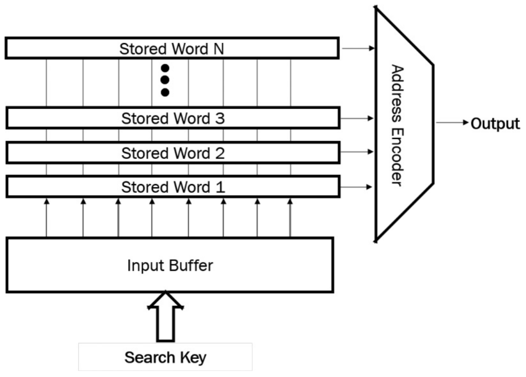


Fig. 1. Schematic view of Content Addressable Memory (CAM).

lookup tables for secure near-neighbor search. Note that any lookup table can be implemented using conventional RAM architectures. Nevertheless, CAMs are superior compared to RAMs due to the following reasons:

- CAMs can search the entire table in a *single clock cycle* while RAMs perform *sequential* fetch and comparison operations.
- CAMs enable in-memory computation while RAMs require a processing unit which incurs additional delay and energy cost of data movement.
- CAMs reduce the response delay, improving the runtime of data-intensive applications.

CMOS-based CAMs [21] and resistive CAMs [22] have been proposed in the literature. CMOS-based CAMs are area-intensive and power consuming. For instance, a CMOS-based CAM consumes about 20× more power and requires 3.8× larger area compared to a RAM of the same storage capacity [23]. Alternatively, resistive cells deliver high-density and low-power CAMs [24, 25].

2.2 Approximate Search

In approximate search, the goal is to find all entries in a database that are *similar* to a given query q. Throughout this paper, we use the terms *approximate search* and *near-neighbor search* interchangeably. We refer to each entry of the database as a *word*. The user's request is referred to as a *query*. Each data is represented as a D-dimensional vector whose elements are binary, integer, or real values depending on the underlying application. To quantify the similarity, a metric (e.g., Jaccard, Cosine, or Euclidean) which is application-specific should be defined. For example, one of the most popular similarity metrics for web documents is the Jaccard similarity. In this case, each document is represented as a set. The Jaccard similarity for two sets $x, y \subseteq \Omega = \{1, 2, \ldots, |\Omega|\}$ is defined as

$$\mathcal{R} = \frac{|x \cap y|}{|x \cup y|}. \tag{1}$$

Another popular metric is the Cosine similarity. For two vectors $x, y \in \mathbb{R}^D$, the Cosine similarity can be computed as

$$C = \frac{x^T y}{||x||_2 \cdot ||y||_2}, \tag{2}$$

where $||.||_2$ denotes norm-2 of a vector.

The output of the search is the indices of words in the database whose similarities are more than a predefined threshold (T_S). More precisely, we are looking for

$$Search(q) = \{i \mid Similarity(d_i, q) > t\},$$

where d_i is the i^{th} word in the database. Without loss of generality, the similarity measurement can always be normalized to a value between zero (no similarity) and one (identical). The approximate search is called *secure* if the database holder outputs indices i without inferring any information about the query q and words d_i.

LSH is a popular approximate search method that creates a probabilistic embedding (hash) of a data with the following property: if two inputs are similar in the input domain, their hashes have a higher probability of collision. More precisely,

$$Probability\{h(x) = h(y)\} \;=\; f(Similarity(x, y)) \tag{3}$$

where $h(.)$ is the hash function and $f(.)$ is a monotonically increasing function. There exist variants of LSH that preserve different similarity metrics. For instance, MinHash preserves Jaccard similarity and SimHash preserves Cosine similarity. We briefly describe each of these hash functions while the reader can refer to [19] for more detailed explanation.

2.2.1 Jaccard Similarity and MinHash. As we discussed in Section 2.2, Jaccard similarity is defined over sets. To compute the hash value, a random permutation $\pi : \Omega \rightarrow \Omega$ is applied on the input set. The hash is the minimum value of the permuted set; more precisely, $h_{min}(x) = min(\pi(x))$. For instance, consider the set $x = \{2, 4, 5\} \subset \Omega = \{1, 2, 3, 4, 5\}$ and the random permutation

$$\pi : 1 \longrightarrow 4, 2 \longrightarrow 1, 3 \longrightarrow 5, 4 \longrightarrow 2, 5 \longrightarrow 3$$

that maps the set x to $\pi(x) = \{1, 2, 3\}$, hence, $h_{min}(x) = 1$. It has been shown [19] that MinHash is a valid LSH since

$$Probability\{h_{min}(x) = h_{min}(y)\} \;=\; \mathcal{R},$$

where $\mathcal{R}$ is the Jaccard similarity defined in Equation (1) and the function $f(.)$ in Equation (3) is equal to $f(\alpha) = \alpha$ which is a monotonically increasing function.

Alternatively, each set can be represented as a binary vector of length $|\Omega|$ where the j^{th} binary value indicates whether j is a member of the set (value one) or not (value zero). Therefore, MinHash can be viewed as a function over binary vectors of length $D = |\Omega|$, $h_{min} : \{0, 1\}^D \rightarrow \mathbb{N}$. In order to consume less storage space, it is preferable to generate 1-bit LSH ($h_{min}^{1-bit} : \{0, 1\}^D \rightarrow \{0, 1\}$) by applying a universal hash function to the output of MinHash (see [19] for more information about the universal hash function). The collision probability of $h_{min}^{1-bit}(x)$ with $h_{min}^{1-bit}(y)$ is shown to be $1 + \frac{\mathcal{R}}{2}$ [19]. One needs to perform the hash evaluation l times with l different random permutations to generate an l-bit LSH embedding.

2.2.2 Cosine Similarity and SimHash. SimHash is a popular LSH method that preserves Cosine similarity and is based on Signed Random Projections (SRP) [26] which can be computed as follows: first, a random D-dimensional row vector w is generated where vector components are drawn from i.i.d normal distributions, $w_i \sim N(0, 1)$. The 1-bit SimHash is generated by computing the sign of the projection of input vector to the randomly generated vector w, $h_{sim}(x) = sign(w \cdot x^T)$. In fact, we have $h_{sim} : \mathbb{R}^D \rightarrow \{0, 1\}$. It is not difficult to show the following

$$Probability\{h_{sim}(x) = h_{sim}(y)\} \;=\; 1 - \frac{\theta}{\pi},$$

where $\theta = \cos^{-1}(\frac{x^T y}{||x||_2 \cdot ||y||_2})$. Comparing this with Equation (3), the function $f(.)$ can be formulated as $f(\alpha) = 1 - \frac{\cos^{-1}(\alpha)}{\pi}$. In order to generate an l-bit LSH embedding, we need to compute 1-bit SimHashes l times with l different randomly sampled w vectors.

3 METHODOLOGY

3.1 Scenario and Privacy Concerns

We assume that the CAM architecture is employed in a cloud server to store the database. This database holds secret data from certain parties that we call *data owners*. We refer to each record of the database as a *word*. A user wants to search the database for words that are most similar to her query. Traditionally, the client's query and the words are revealed to any party that has access to the database. Storing the raw sensitive data on CAM memories in a centralized fashion makes users susceptible to internal attacks (i.e. server acting as a malicious party) or external threats (i.e. server compromised by a hacker).

In our computational model, the sensitive data is stored in the secure CAM (database) while the meta-data and user's ID are stored in plaintext by the cloud server in a regular memory. For example, in an online dating application, the sensitive data includes user's age, ZIP code, physical attributes, and personal preferences of the person that he/she prefers to be matched to. The metadata includes the username, hash of the user's password, and user's ID in the database. The metadata is used to connect two persons whose profiles are matched. Note that this kind of metadata is the least possible information that has to be kept by the cloud server in order to operate such centralized web-based applications. The focus of this paper is to create the first of its kind secure CAM.

Threat model: This paper assumes that the owner of the CAM (the server) is *untrusted*. More precisely, CAMsure is secure against honest-but-curious (semi-honest) server. In this attack model, server may attempt to infer information about the data that she stores, sends, and receives but it is assumed that the server will follow the protocol. This threat model strongly satisfies today's privacy concerns where users wish to hide the content of their query from the cloud server. Since we do not trust the server, the privacy of CAM words, the query, and also the search result should be preserved. This threat model also covers the situation where the entire database is compromised by an attacker.

To provide security for the CAM architecture, the users employ a hashing technique that hides the content of their data. The hashing method preserves similarity, meaning that data points that are close in the data domain are also similar in the hash domain, and those that are not close in the data domain are dissimilar in the hash domain. Section 3.2 describes, in high-level, the methodology of CAMsure for preserving the confidentiality of CAM data.

3.2 CAMsure Overview

CAMsure delivers scalable and lightweight privacy-preserving approximate search. As depicted in Figure 2, the scenario of CAMsure comprises two main ingredients: (i) a hashing scheme called LSH and (ii) a database consisting of one (or several) CAM blocks.

On the client side, the input is hashed using LSH to create a query which hides the actual content of the input vector. LSH takes as input a vector of real-valued elements and maps it into a binary string. The amount of information that the hash reveals about the actual input can be controlled as we discuss in Section 4.3.

On the server side, the hash values provided by data owners are stored in the database. The database is realized by a certain number of CAMs. Due to the resemblance-preserving property of LSH, server can search for similar hash words within the database without actually knowing

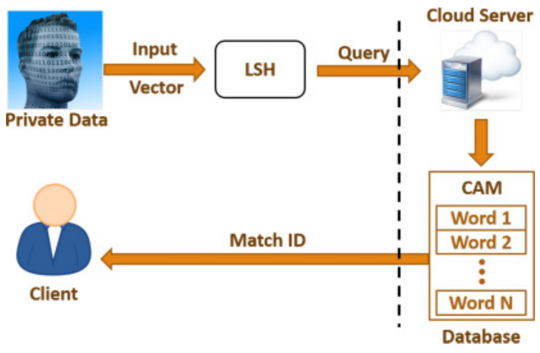


Fig. 2. Overview of the secure approximate search scenario using CAMsure.

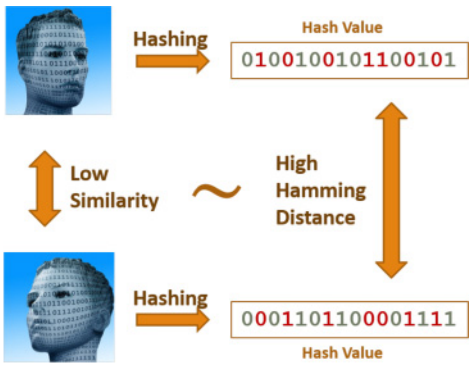


Fig. 3. Relationship between the data and hash domains. The hashing scheme preserves the proximity of inputs only for similar data.

what the query or the words correspond to; therefore, CAMsure provides the security of CAM contents against malicious parties. In addition to providing security, CAMsure reduces the computational complexity of approximate search by converting high-dimensional vectors into hash values. It supports dynamic insertion and deletion of database words since the hash embeddings are generated independently. To the best of our knowledge, CAMsure provides the first secure CAM for the purpose of approximate search. We elaborate on the concept of LSH in Section 3.3. In Section 3.4, we discuss the additional modifications required to support approximate search in CAMs.

3.3 Hashing Methodology

In this section, we illustrate the hashing method that is utilized in CAMsure. Figure 3 depicts the concept of LSH for secure approximate search. The left-hand side belongs to the data domain where each input is represented as a feature vector of real values, e.g., features extracted from an image of the person's face. The right-hand side represents the hash domain where each word is a binary string computed from the original data. The two hash strings in Figure 3 are different since the original feature vectors (the faces) are dissimilar. Any LSH scheme guarantees that the probability of two LSH bits colliding (being equal) is a monotonic function of the similarity in the original domain. For example, assume that LSH guarantees a collision probability of 95% in the hash domain for a similarity threshold of $T_S = 0.9$ in the data domain. This implies that, on average,

$0.95 \times l$ bits should match for l-bit LSH embeddings of two inputs with a similarity of $t = 0.9$. Therefore, utilizing LSH enables us to translate the near-neighbor search to finding the words in the database that have Hamming Distance of $(1 - 0.95) \times l$ or less. Although this approach introduces a certain degree of inaccuracy, the added error rate is negligible in practice [27]. We elaborate on the accuracy of CAMsure in Section 4.2 comprehensively.

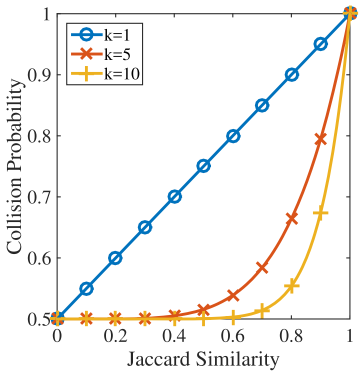
The key achievement of using LSH is that the server no longer needs to compute pairwise similarities using a processor. The near-neighbor search is performed by just checking if the incoming query has small Hamming distance (given certain threshold) to any of the database words. This check can be efficiently done in only *one* clock cycle by minimal modification in the CAM [8].

As we discussed, MinHash and SimHash require a set of randomly generated numbers, i.e. MinHash requires random permutation and SimHash requires random projection vector, which we refer to as *random seeds*. Although these seeds are generated randomly for each LSH bit, they have to be consistent for generating the hash of all the data in the database and also for the incoming query. To satisfy this requirement, the server generates these seeds and announces them publicly. Whenever a new user wants to search the database, she needs to download these seeds, compute the hash of her input, and send the hash to the server. The size of these seeds is very small, usually in the order of KB and is especially suited for Internet settings. This is in contrast to secure multiparty computation methods which require very high communication bandwidth (in the order of MB or GB) [12]. It is worth mentioning that these seeds have to be downloaded only once at the time when the user registers to the cloud server.

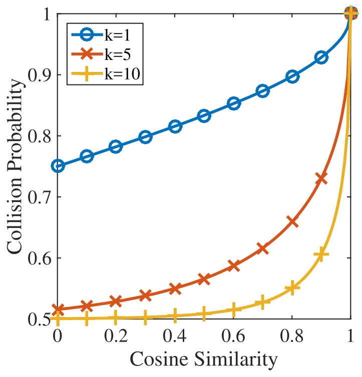
3.3.1 Data Leakage of Traditional LSH Schemes. The amount of information that one can infer from a single LSH embedding and the publicly available random seeds is limited and is discussed in Section 4.3. However, an attacker can generate multiple fake input vectors and perform the *triangulation* attack as introduced in [19]. The susceptibility comes from the fact that traditional LSH schemes allow an estimation of the similarity between original vectors corresponding to any pair of hash values. Assume that an attacker wants to reconstruct the original vector v hashed into $hash(v)$. He can create random vectors v_i, compute their hashes $hash(v_i)$, and compare them to $hash(v)$. Since traditional LSH methods preserve the pairwise similarity for *any* two input vectors, the attacker can identify vectors v_i to which v is most similar according to the Hamming distances of $hash(v)$ and $hash(v_i)$. Consequently, the attacker reduces the subspace in which v can reside and estimates the elements of v up to a certain precision. Performing the same procedure iteratively results in a very good estimation of v. Therefore, if the server is malicious or an attacker can get access to the database, the privacy of all users is compromised.

Recently, a novel hashing technique is proposed to mitigate the triangulation attack [19]. The idea behind their method is to make sure that the collision probability of LSH bits is high only when the two inputs are *very similar* and is as low as 50% for non-neighbors (same as collision probability of two completely random bits). As a result, if an attacker repeats the same triangulation attack, he cannot estimate the original attributes by brute-forcing the possible input space. In the following section, we illustrate how the hashing method of [19] works and how it achieves security against the triangulation attack.

3.3.2 LSH Transformation. Authors of [19] have proposed an LSH *transformation* that eliminates unnecessary information leakage of traditional LSH methods. To perform this task, they suggest that the transformed hashing scheme should have the following property: for any two input vectors that have similarity lower than a threshold (T_S), their corresponding hashes must have very high Hamming distance. As a direct consequence of this condition, for any two dissimilar inputs, the collision probability of any single LSH bit must be close to 0.5, making them



(a) MinHash



(b) SimHash

Fig. 4. Collision probability vs. similarity for MinHash and SimHash functions for three different security parameters $k \in \{1, 5, 10\}$. $k = 1$ represents traditional LSH methods.

indistinguishable from two randomly generated bits. In other words, the correlation between two hashes must be very close to zero in order to avoid information leakage.

The proposed transformation which satisfies this condition is

$$h_{secure}^{1-bit}(x) = h_{univ}(h_1(x), h_2(x), \ldots, h_k(x)), \tag{4}$$

where h_i, $i \in \{1, 2, \ldots, k\}$ are k independent hash functions (with independent initial random seeds) and $h_{univ}(.)$ is defined as

$$h_{univ}(\alpha_1, \alpha_2, \ldots, \alpha_k) = \left(r_{k+1} + \sum_{i=1}^{k} r_i\alpha_i\right) mod\ p,\ mod\ 2, \tag{5}$$

where r_i, $i \in \{1, 2, \ldots, k\}$ are randomly generated integers and p is a prime number. Note that r_i should remain fixed for all of the hash computations once they have been derived. Parameter k is called the *security parameter.* h_i can be implemented as any traditional LSH function, e.g., MinHash or SimHash.

The effect of this transformation is that the collision probability of two 1-bit hashes, $h_{secure}^{1-bit}(x)$ and $h_{secure}^{1-bit}(y)$, drops rapidly as the similarity of x and y decreases. For instance, the collision probability of secure MinHash is

$$Probability\ \{h_{secure}^{1-bit}(x) = h_{secure}^{1-bit}(y)\} = \frac{\mathcal{R}^k + 1}{2} \tag{6}$$

where $\mathcal{R}$ is the Jaccard similarity between x and y. Similarly, the collision probability of secure SimHash is

$$Probability\ \{h_{secure}^{1-bit}(x) = h_{secure}^{1-bit}(y)\} = \frac{\left(1 - \frac{\cos^{-1}(C)}{\pi}\right)^k + 1}{2} \tag{7}$$

where C denotes the Cosine similarity of x and y. The formal proofs of Equations (6) and (7) are provided in [19].

We have depicted the collision probability of secure MinHash and SimHash for different security parameters, $k \in \{1, 5, 10\}$, in Figure 4. Note that $k = 1$ is identical to traditional LSH functions (please see Equation (4)) and represents the baseline for comparison with traditional LSH schemes.

As mentioned in Section 3.3.1, the triangulation attack is possible for traditional LSH schemes since they allow a similarity estimation for *any* two input vectors whose similarity ranges from 0 to 1. This is shown as the curves corresponding to $k = 1$ in Figure 4. As can be seen, an attacker can estimate how similar vectors v_i are to the secret vector v by comparing the HD between $hash(v_i)$ and $hash(v)$. v is most similar to the vector v_j whose Hamming distance to $hash(v)$ is minimum. However, the same property does not hold for $k = 5$ or $k = 10$ since the collision probability drops drastically as the similarity goes to zero. For example, assume v_j has a similarity of 0.8 to v, therefore, the collision probabilities for 1-bit secure SimHashes are 0.9, 0.66, and 0.55 for $k = 1, 5, 10$, respectively (see Figure 4). Please note that the collision probability of any two random bits chosen from the Uniform distribution is 0.5. The collision probability of hash bits directly translates to the expected number of bit-matches of the hashes.

Assuming l-bit hash embeddings, the number of bit matches between $hash(v_j)$ and $hash(v)$ are on average $0.9l$, $0.66l$, and $0.55l$ for $k = 1, 5, 10$, respectively. Since $0.9l$ is an outlier, the attacker gets to know that v_j must be very similar to v and he can iteratively continue his search in a smaller subspace. However, when the secure hashing function with parameter $k = 10$ is used, the attacker observers $0.55l$ bit matches (with a high probability). Since this number is very close to the number of bit matches between any two random l-bit strings, the attacker cannot infer any information. In CAMsure, the privacy of data owners and data users is preserved since the server does not have access to the plaintext of users' data and the query. This means that even if the entire database is compromised or the server is malicious, users' data is not revealed. The comprehensive security analysis is provided in Section 4.3.

As we have illustrated in this section, in order to find near-neighbors of the query, one needs to find database words whose Hamming distance to the query is lower than a certain threshold. For example, if the threshold is 2, any word in the database with Hamming distance of 0, 1, or 2 to the query should be reported as a near-neighbor. In Section 3.4, we explain how one can perform the approximate search with HD tolerance in CAMs.

3.4 Approximate Search in CAMs

The probabilistic nature of LSH implies that even the most similar raw records might have some Hamming distance in their hash embeddings. Consequently, the lookup table search is no longer a search with exact matching. Any stored word whose Hamming distance to the input query is below some pre-defined threshold is considered a match (near-neighbor) and its corresponding match-line should trigger a high voltage. Therefore, the proposed CAM should be capable of performing search with certain HD tolerance.

The earliest implementation of CAM can only perform exact matching. Ternary CAMs (TCAMs) provide the additional functionality of having "don't care" bits in the seach key [28]. These bits are masked during the search, i.e., the masked CAM cells output a high voltage regardless of the corresponding bits in the input query. The "don't care" bits in TCAMs are not permuted, i.e. their location in the input string is fixed; thus, TCAMs cannot be readily used for evaluating the Hamming distance. Therefore, throughout this paper, we do not focus on TCAMs.

Authors of [8] present a modified CAM that allows approximate search based on Hamming distance. Figure 5 illustrates the idea behind their methodology for approximate matching. The solid red line corresponds to the match-line's transient voltage when the query matches the stored word with zero Hamming distance. The dashed lines correspond to match-lines with different Hamming distances. The HD tolerance can be tuned using two techniques:

- **Changing the sampling time:** It can be observed that, as the number of mismatched cells is increased, the output voltage drops more abruptly. This change in the decay time allows us to infer a partially mismatched data as a match by simply changing the sampling time; the earlier the output is sampled, the more Hamming distance is tolerated.

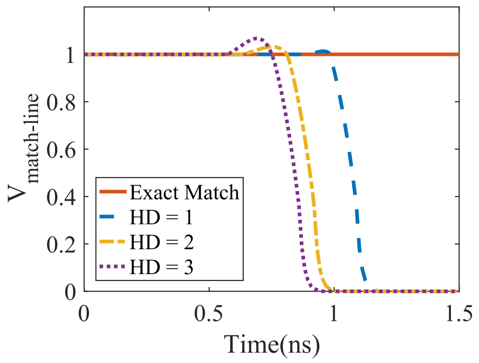


Fig. 5. CAM match-line transient output voltage for exact and approximate matching.

Table 1. Cache Design for the RAM Baseline

Cache Size	Line Size (bytes)	Associativity	No. Banks	Technology
Variable (based on lookup table size)	32	4	1	45 nm

- **Voltage over-scaling:** Another method to tune the Hamming distance tolerance is to fix the sampling time and change the supply voltage applied over the cells. Note that decreasing the supply voltage also reduces the power consumption of the CAM.

Authors of [8] mention two downsides for the proposed approximate CAM: (i) possibility of false match and (ii) having multiple matches. Possibility of mismatch simply refers to having approximate matches, which in fact, is a design goal for CAMsure. Having multiple matches is also desired as the goal of near-neighbor search is not to identify a single record but to find all data points that are similar to the query. In addition to the aforementioned issues, their design is not capable of distinguishing Hamming distances that are higher than 2. This issue might be partially mitigated by increasing the capacitance of the CAM cells which can make higher Hamming distances distinguishable.

4 ANALYSES

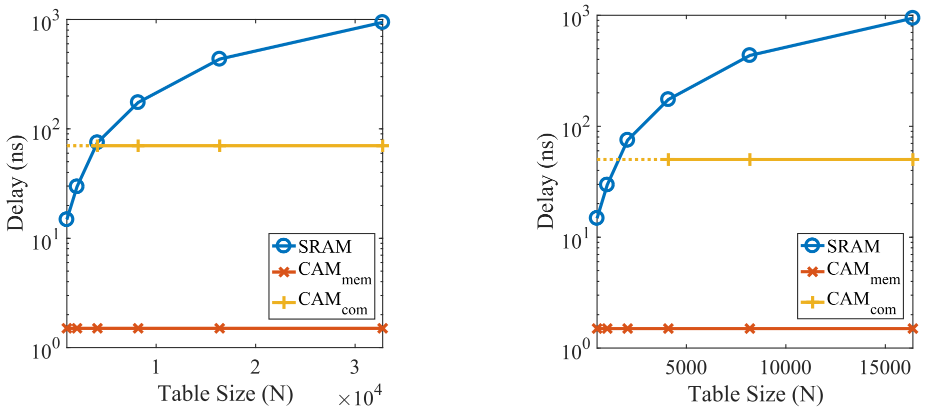
In this section, we first demonstrate the superiority of CAM over RAM for the purpose of near-neighbor search. Next, we analyze the accuracy and security aspects of CAMsure.

4.1 Delay and Energy Analysis

We compare the proposed search method over hash embeddings stored in Static RAM (SRAM) and CAM architectures. Specifically, we compare the two architectures for different hash embedding widths (i.e. 32-bit and 64-bit) denoted by l and different sizes of the lookup table which we denote by N. For the SRAM architecture, we simulate a cache using the CACTI 5.3 tool [29] which is an open-source software from HP Co. to estimate the read-energy and search delay of different cache designs. Table 1 outlines the cache used in our reports. For the CAM-based implementation, we incorporate the energy and delay reports of [8] to analyze the proposed methodology for memristive CAMs that are capable of approximate search. Specifically, we assume that multiple CAM blocks from [8] with the corresponding bit-width are instantiated to accommodate the database; the search delay would be the same as reported in [8], while the energy consumption

Table 2. Specifications of CAM Baselines

				Search Energy (pJ)		
CAM Architecture	**Delay (ns)**	**No. Words**	**Technology (nm)**	Exact	HD = 1	HD = 2
32-bit resistive [8]	1.5	64	45	3.901	1.738	1.332
64-bit resistive [8]				4.568	1.953	1.479
32-bit CMOS [30]	70	4096	NA	1617	-	-
64-bit CMOS [31]	50	4096	NA	2475	-	-



(a) 32-bit hash embeddings (b) 64-bit hash embeddings

Fig. 6. Delay analysis for hash embeddings of size 32 and 64-bit. CAM_{mem} and CAM_{com} denote memristive and commercial CMOS-based CAMs, respectively. The dotted line for CAM_{com} shows interpolated data.

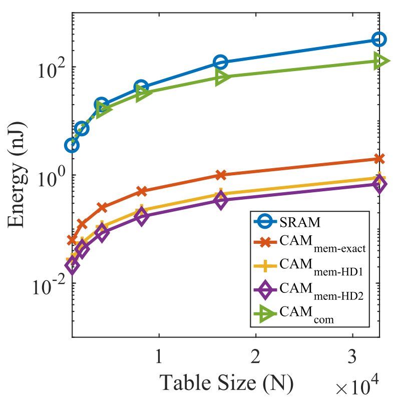
increases linearly with respect to the number of instantiated blocks. We also compare commercialized CMOS-based CAMs from the LANCAM family [30, 31]. Table 2 outlines the specifications of the CAM blocks.

Note that both SRAM and CAM are used to implement the same search algorithm, therefore, their search accuracy is the same; the only difference between SRAM and CAM is in the way they perform the search. The following sections compare the delay and energy of a single search for SRAM and CAM.

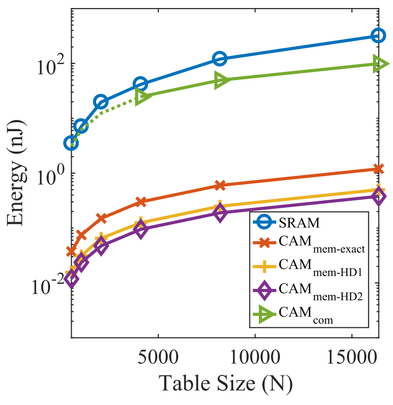
4.1.1 Delay Analysis. We compare the search delay of different architectures in Figure 6. For the cache-based implementation, we change the cache size of the CACTI memory to accommodate the number of words within the table. We give an advantage to the baseline SRAM by assuming that the processing unit is fully pipelined and the memory can operate in maximum throughput. For this purpose, we use the "interleave cycle time" metric reported by the CACTI tool which we denote by T. We also assume that the cache hit rate is 100%. The number of sequential read operations to perform a single search is $\frac{N}{m}$ where N is the number of words within the database and m is the number of words fetched by a single read operation. More specifically, m is the number of words within one line of the cache

$$m = \frac{line\ size}{bytes\ per\ embedding}.$$

A 64-bit (32-bit) embedding requires 8 bytes (4 bytes) to be stored and assuming a line size of 32 bytes in Table 1, each interleaved access provides $m = \frac{32}{8} = 4$ ($m = \frac{32}{4} = 8$) words. The search



(a) 32-bit hash embeddings



(b) 64-bit hash embeddings

Fig. 7. Energy consumption analysis for hash embeddings of size 32 and 64-bit and different Hamming distances. CAM_{mem} and CAM_{com} denote memristive and commercial CMOS-based CAMs, respectively. The dotted line for CAM_{com} shows interpolated data.

delay reported in Figure 6 is then computed as

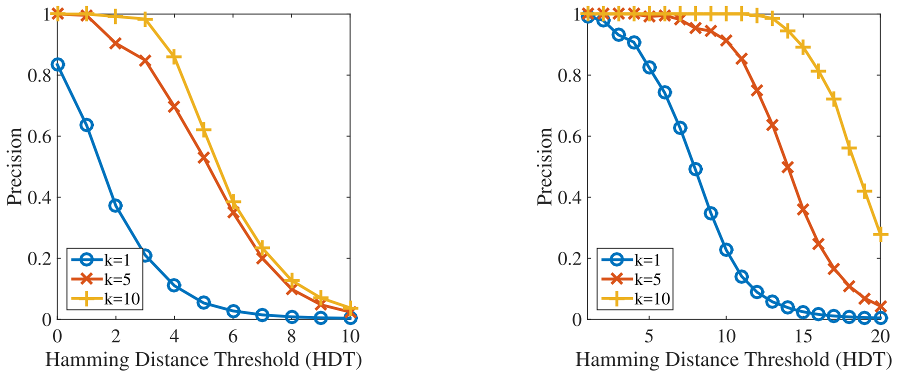
$$Delay = T\frac{N}{m}.$$

For the CAM-based table, we assume that multiple banks of the memristive and the CMOS-based blocks (Table 2) are instantiated to perform the search in parallel, thus, the CAM search time is fixed for different database sizes. In contrast, the SRAM search time is increased for a higher number of words in the lookup table since the words are processed sequentially. The results show that, compared to SRAMs, CAM architectures reduce the search delay by *two orders of magnitude*.

4.1.2 Energy Analysis. Figure 7 compares the energy consumption of different memory architectures. We take the "dynamic energy per read port" metric provided by the CACTI tool and multiply it by the number of sequential read operations to compute the search energy for SRAM. For CAMs, we assume that the search energy is linearly increased with respect to the number of CAM blocks. It is observed that the search energy of CAM is significantly lower than that of SRAM. Memristive CAMs offer a smaller energy consumption than CMOS-based CAMs. It is noteworthy that the power consumption is decreased as we increase the Hamming distance. A high HD tolerance can be achieved by lowering the supply voltage which in-turn reduces the search energy.

4.2 Accuracy Analysis

The performance and power analyses in Section 4.1 are general and account for any dataset regardless of the application. However, the search accuracy analysis which we discuss in this section is inherently dependent on the underlying dataset. For this purpose, we focus on the SpeedDating [32] dataset which contains 8378 text survey samples (profiles). Each profile is represented as a 190-dimensional vector. Each element of the vector is a feature that contains sensitive information about the individuals, e.g., gender, race, age, the field of study, zipcode, income, etc. We report the search accuracy by comparing the result from CAMsure with the most similar elements in the original domain (before hashing) that pass similarity threshold of 0.95. Therefore, both the ideal search result and the result from CAMsure can be viewed as a bag of indices (of the



(a) 32-bit hash embeddings (b) 64-bit hash embeddings

Fig. 8. Precision analysis for hash embeddings of size 32 and 64-bit for different security parameters (k).

database) which are claimed to be similar. CAMsure outputs the index of any word in the database that passes the Hamming Distance Threshold (HDT). In other words, any word that has Hamming distance of HDT or less to the query is reported as a near-neighbor.

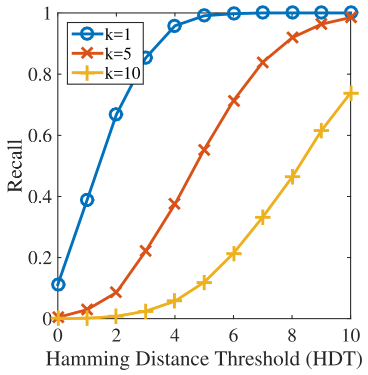
We report the accuracy results based on two metrics: *precision* and *recall*. *Precision* is defined as $\frac{n_c}{n_r}$, where n_r denotes the number of total near-neighbor indices reported by CAMsure and n_c denotes the number of indices reported by CAMsure that are correct (truly similar). Here, we consider two points with similarity measure of 0.95 or higher to be called near-neighbors. *Recall* is defined as $\frac{n_c}{n_t}$ where n_t is the total number of words within the database that are considered as near-neighbors of the query. An ideal system should report *all and only all* of the true near-neighbors which means both precision and recall should ideally be equal to one. A trivial solution that outputs all entries in the database has high recall (=1) but unacceptable precision (~0). We generate 32-bit and 64-bit LSH embeddings for each profile in the dataset. We randomly select the query as one of the profiles from the dataset and repeated the experiment for 100 different queries for each set of parameters.

Figure 8 illustrates the precision results as a function of the HDT for different security levels (k), where two plots are depicted for 32-bit and 64-bit LSH embeddings, respectively. The number of bits in the LSH embedding is equal to the bit-width of each word stored in CAM. The search precision is higher for low values of HDT. This is persistent with our intuition since the HDT defines the measure of search accuracy: by choosing higher values for the HDT, more words are selected from the database that may not correspond to a true near-neighbor, reducing the overall precision.

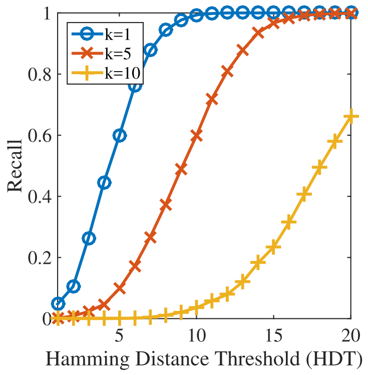
Figure 9 provides the recall results as a function of the HDT for different security levels (k) and different LSH embeddings (32-bit and 64-bit). In contrast to the precision reports, increasing the HDT results in a higher recall. Higher values of HDT allow for more database words to be selected. Therefore, there is a trade-off between precision and recall which is a known concept in near-neighbor search algorithms.

It is beneficial to plot the precision as a function of the recall in order to illustrate the trade-off (Figure 10). As the analysis suggests, the hashing scheme utilized in CAMsure achieves a reasonably high precision and recall while providing data confidentiality.

Two observations are worth mentioning: first, 64-bit LSH embeddings deliver better precision/recall trade-off with the cost of higher computational complexity (see Figure 10). Second,

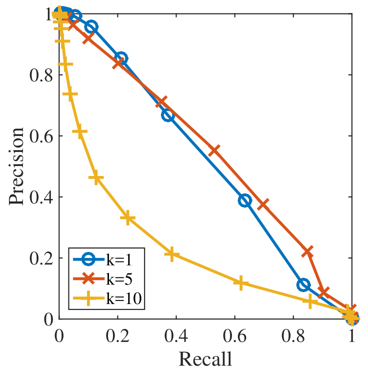


(a) 32-bit hash embeddings

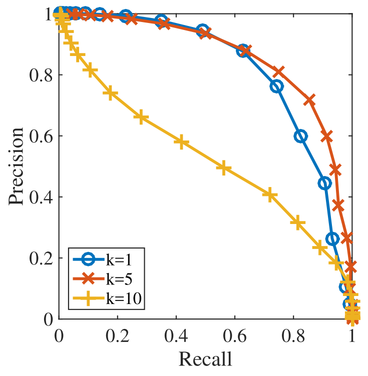


(b) 64-bit hash embeddings

Fig. 9. Recall analysis for hash embeddings of size 32 and 64-bit for different security parameters (k).



(a) 32-bit hash embeddings



(b) 64-bit hash embeddings

Fig. 10. Precision vs. Recall analysis of 32 and 64-bit hash embeddings for different security parameters (k).

increasing the security parameter k comes with the cost of lower precision and recall. Therefore, the number of bits in the hash embedding (l) and the security parameter (k) are the two factors that should be tuned to utilize CAMsure for different datasets.

4.3 Security Analysis

We provide a comprehensive security analysis of CAMsure in the event where the cloud server is malicious or the database is compromised by an attacker. We analyze, both theoretically and experimentally, the scenario where an attacker wants to infer as much information as possible about one or multiple words in the database. Assuming that the attacker has complete access to the database, we are interested in knowing what information can be inferred from the original data given the hash embeddings. In Section 4.3.1, we explain the state-of-the-art method, called *compressed sensing*, that aims to reconstruct the original data given its hash. In the second analysis, we consider the idea of the *brute-force* attack on CAMsure. We provide theoretical and experimental analysis for this attack. The third analysis focuses on validating the theory of LSH transformation. Finally, we focus on a more sophisticated attack, called *Triangulation* [19], which aims to extract information by leveraging the correlation between hash embeddings of multiple data points scattered in different spans of the space.

4.3.1 Compressed Sensing. The theory of compressed sensing aims to extract information from randomized embeddings. However, there is currently no practical approach that can extract meaningful information from the LSH embeddings utilized in this paper. In general, compressed sensing requires more than $\theta(s \log_2 D)$ measurements to provide reasonable accuracy [33] where s and D are the number of non-zero elements and the dimensionality of the input vector, respectively. However, in CAMsure, the bit-width of the hash embedding (32 or 64) is far smaller than the aforementioned lower bound which is at least $190 \times log_2(190) = 1438$ bits for the Speed-Dating dataset. Compressed sensing algorithms are similar to the idea of triangulation attack which we discuss in Section 4.3.4.

4.3.2 Brute-force Attack. The attacker might attempt to guess the original value of a vector and validate his choice by comparing the hash value of his guess with the database entries. If he finds a hash value identical to what he has computed, the original data vector is revealed to him. However, this attack is computationally impossible as we describe next. If we represent each real number as a fixed point 32-bit, a D dimensional vector input has

$$N_{possible} = (2^{32})^D$$

possible different values. According to NIST standard [34], any brute-force attack that requires 2^{128} or higher operations is considered infeasible. Since $N_{possible}$ grows *exponentially* with D, even for small dimensionality ($D = 4$ or higher), the attack is infeasible.

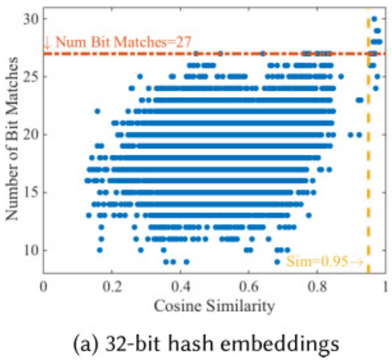
4.3.3 Validating the LSH Transformation. The theory behind the LSH transformation of [19] suggests that mutual information between the bits of two hashes drops rapidly as the similarity of the original vectors is decreased. For example, in the case of SimHash,

$$I(h^{1-bit}_{secure}(x); h^{1-bit}_{secure}(y)|\theta) < \left(1 - \frac{\theta}{\pi}\right)^k log\left(\frac{1 + \left(1 - \frac{\theta}{\pi}\right)^k}{1 - \left(1 - \frac{\theta}{\pi}\right)^k}\right), \tag{8}$$

where $I(.)$ denotes the mutual information [35] between two SimHash bits and $\theta = \cos^{-1}(C)$ with C being the Cosine similarity, defined in Equation (2). In this section, we validate this theory experimentally. We choose a random profile from the Speed-Dating dataset as our query and consider its similarity to all other profiles in the database. In particular, we compute the true similarity measure in the data domain $s_{original}$ and the number of matched bits in the hash domain s_{hashed} using a security parameter of $k = 5$ for 32-bit and 64-bit embeddings. Figure 11 depicts s_{hashed} as a function of $s_{original}$ between the chosen profile and all other profiles in the database.

The vertical yellow lines in Figure 11 show the similarity of 0.95. The horizontal red lines depict 27 and 53 number of bit matches for 32 and 64-bit LSH embeddings, respectively. As our experiments show, the results closely follow Equation (8). If two profiles are not similar in the original vector representation (x-axis), the corresponding tuple ($s_{original}$, s_{hashed}) resides on the left-hand side of the yellow line. It can be seen that the corresponding hashes have a random number of bit matches anywhere between zero to $HDT = 27$ ($HDT = 53$) for such profiles. Meanwhile, two profiles with a similarity of 0.95 or higher reside on the right-hand side of the yellow line and the corresponding number of bit matches is higher than the threshold $HDT = 27$ ($HDT = 53$). This is consistent with our expectation from the characteristic of the LSH transformation.

4.3.4 Triangulation Attack. A more sophisticated attack is to create multiple random vectors, compute their hashes, and triangulate the secret vector by comparing the HD of the corresponding



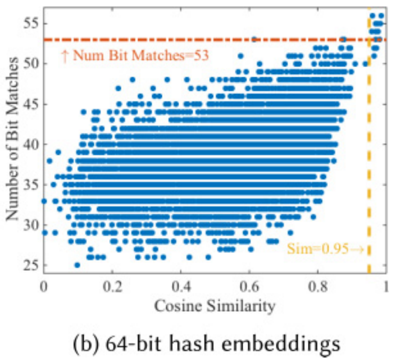


Fig. 11. Validating LSH transformation in practice on Speed-Dating dataset for security parameter $k = 5$.

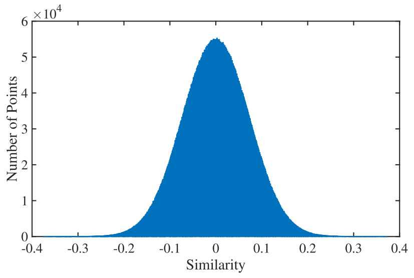


Fig. 12. Histogram of pairwise similarities between p and 10^7 randomly selected points in 190-dimensional space.

hashes (see Section 3.3.1). However, the attack is not effective for the LSH transformation that we have utilized as we explain here.

Each D-dimensional vector can be viewed as a point in the D-dimensional space. As the dimensionality increases, the volume of space grows exponentially. This phenomenon is known as *curse of dimensionality* [36]. The distance of two randomly chosen points in a high-dimensional space is much higher than that of a low-dimensional space. To Illustrate this phenomenon in practice, we randomly choose a point p in the 190-dimensional space (the same dimensionality as the Speed-Dating dataset). After that, 10^7 random points have been created and the mutual Cosine similarities between each one of these and p have been computed. Figure 12 shows the statistical distribution of mutual similarities. We have repeated the experiment for 10 different random choices of p. Based on experimental results on 10^7 different points, the values for similarities range from -0.37 to 0.38 and not even one point has a similarity higher than 0.4. Please note that the attacker needs to obtain the mutual similarity of 0.9 or higher (see Section 3.3.2) to successfully estimate the true vector corresponding to a given hash. However, as is illustrated in Figure 12, an attacker simply cannot find the starting vectors and as a result, the attack is not effective.

5 RELATED WORK

To the best of our knowledge, there has been no attempt to secure the search operation in content addressable memories. Here, we briefly describe previous solutions to secure near-neighbor search problem that use the convectional Von Neumann processing architecture. All of these methods require a processing unit at the server side and incur high computation and communication while CAMsure can process the query in *real time*. We categorize previous works into three main groups:

Differential Privacy (DP): This line of research provides certain privacy guarantees by adding specific kind of noise to the data that is stored on the server. The noise is added such that the statistical properties (e.g. average, variance, etc.) of the whole database are preserved while individual words are altered [37]. This privacy enhancing approach is useful in scenarios where the clients are interested in statistical properties of the database not the individual records within the database. In addition, the security model of DP is different; the assumption is that the server which holds all users' data is *trusted* and an attacker only has the ability to query the database and infer as much information as possible based on the results that he receives. In contrast, our assumption is that the server is not trusted. Therefore, CAMsure provides stronger security and privacy guarantees.

Property-Preserving Encryptions: As we discussed in Section 1, secure NNS solutions based on Order-Preserving Encryption (OPE) [13], Deterministic Encryption (DTE), and Asymmetric Scalar-Product-preserving Encryption (ASPE) [15] are proven to be insecure by [14] and [16], respectively. While Order Revealing Encryption (ORE), Inner Product Encryption (IPE), and Hidden Vector Encryption (HVE) are appropriate candidates for securing conventional RAM-based databases, they are not compatible with the CAM architecture. Another line of research is based on the searchable encryption [38]. However, this type of solutions can only answer exact query matches and not the near neighbors.

Noise-Addition Techniques: Another idea is to add noise to the query *before* sending it to the server in order to protect the privacy of data owners. Authors in [17] have looked into the possibility of such idea. They observe that adding noise introduces a privacy/utility trade-off. In other words, if the user adds stronger noise, an attacker will infer less information about the original data but as a consequence, the precision of search is sacrificed. This approach has very limited practical usage as a reasonable privacy guarantee results in a very poor precision for the near-neighbor search and sabotages one of the main goals which is the search quality. In contrast, we have shown in Section 4.2 that our approach delivers high precision/recall for secure approximate search.

Secure Function Evaluation (SFE) protocols: SFE protocols allow two or multiple parties to evaluate any function on their private inputs. There are two possible ways to deploy SFE protocols for secure search: (i) the server possess the entire database in plaintext and acts as the first party and the user who wants to query the database acts as the second party [39, 40]. In this scenario, the server and the user engage in a secure *two-party* computation where the database and the user's query remain private. However, it is assumed that the server already knows all the data in plaintext which is in contrast to our security model. (ii) All of the data is not centralized and it is kept by the data owners only. To process each query, all parties need to engage in a secure *multiparty* computation. In this case, the privacy of all data owners is guaranteed but it is necessary for all data owners to be online and have a peer-to-peer pre-established communication channel which is not a realistic assumption.

SFE protocols preserve the privacy of engaging parties completely and they do not leak any information about the inputs. Unfortunately, all of these protocols require massive computation time, communication bandwidth, and several rounds of communication between the engaging parties. The aforementioned drawbacks make SFE protocols to have very limited practical usage.

Recently, authors of [12] have proposed to deploy Yao's Garbled Circuits protocol (one of the most efficient SFE protocols) for the privacy-preserving k-nearest neighbors problem. They consider the two-party computation model (scenario i). They report 2.54 GB communication between two parties and the execution time of 6.7s for 1 Gbps communication bandwidth when the database contains 128,000 records. CAMsure only requires the user to locally compute the hash of her data (negligible time) and send this data to the cloud server (transmitting less than a KB of data) while the query can be answered in one clock cycle on the server.

6 CONCLUSION

This paper proposes CAMsure, the first realization of secure content addressable memory in the context of approximate near-neighbor search. CAMsure utilizes state-of-the-art hashing methods to translate the near-neighbor search algorithm to approximate table lookup. Our comprehensive security analysis shows that CAMsure preserves the confidentiality of data even when the memory is compromised by a malicious party. The proposed methodology is compatible with emerging CAM technologies. The security of content addressable memory is critically important since CAMs are tightly coupled with the main processing units for in-memory computing, aiming to enhance the efficiency of modern computers. Our analysis shows that CAMsure can improve the runtime of RAM-based implementations by up to two orders of magnitude while providing data security. Expansion of the proposed method for applications other than near-neighbor search is another direction to follow in future.

REFERENCES

[1] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L. Lagendijk. 2011. Efficiently computing private recommendations. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).*

[2] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. 2009. Efficient privacy-preserving face recognition. In *International Conference on Information Security and Cryptology*. Springer.

[3] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Fabio Scotti, et al. 2010. Privacy-preserving fingercode authentication. In *Proceedings of the ACM Workshop on Multimedia and Security.*

[4] https://help.yahoo.com/kb/account/SLN27925.html. 2017. Yahoo Security Notice. (2017).

[5] https://www.cnet.com/news/google-fired-engineer-for-privacy-breach/. 2017. Google fires engineer for privacy breach. (2017).

[6] Midas Peng and Sherri Azgomi. 2001. Content-Addressable memory (CAM) and its network applications. In *International IC-Taipei Conference Proceedings.*

[7] Mohsen Imani, Yeseong Kim, Abbas Rahimi, and Tajana Rosing. 2016. ACAM: Approximate computing based on adaptive associative memory with online learning. In *International Symposium on Low Power Electronics and Design (ISLPED).*

[8] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K. Gupta. 2015. Approximate associative memristive memory for energy-efficient GPUs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.*

[9] Mohsen Imani, Daniel Peroni, Abbas Rahimi, and Tajana Rosing. 2016. Resistive CAM Acceleration for Tunable Approximate Computing. *IEEE Transactions on Emerging Topics in Computing* (2016).

[10] Mohammad Samragh Razlighi, Mohsen Imani, Farinaz Koushanfar, and Tajana Rosing. 2017. Looknn: Neural network with no multiplication. In *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE.

[11] Yinian Qi and Mikhail J. Atallah. 2008. Efficient privacy-preserving k-nearest neighbor search. In *IEEE International Conference on Distributed Computing Systems (ICDCS).*

[12] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2015. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *Proceedings of the Design Automation Conference (DAC).*

[13] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. 2009. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.*

[14] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference attacks on property-preserving encrypted databases. In *ACM SIGSAC Conference on Computer and Communications Security.*

[15] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure knn computation on encrypted databases. In *Proceedings of the ACM SIGMOD International Conference on Management of data.*

[16] Bin Yao, Feifei Li, and Xiaokui Xiao. 2013. Secure nearest neighbor revisited. In *IEEE International Conference on Data Engineering (ICDE).*

[17] Petros Boufounos and Shantanu Rane. 2011. Secure binary embeddings for privacy preserving nearest neighbors. In *IEEE International Workshop on Information Forensics and Security (WIFS).*

[18] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*.

[19] M. Sadegh Riazi, Beidi Chen, Anshumali Shrivastava, Dan Wallach, and Farinaz Koushanfar. 2016. Sub-linear privacy-preserving search with untrusted server and semi-honest parties. *arXiv preprint arXiv:1612.01835* (2016).

[20] Gen Kasai, Yukihiro Takarabe, Koji Furumi, and Masato Yoneda. 2003. 200MHz/200MSPS 3.2 W at 1.5 V Vdd, 9.4 Mbits ternary CAM with new charge injection match detect circuits and bank selection scheme. In *Proceedings of the IEEE Custom Integrated Circuits Conference*.

[21] Igor Arsovski, Trevis Chandler, and Ali Sheikholeslami. 2003. A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme. *IEEE Journal of Solid-State Circuits* (2003).

[22] Shoun Matsunaga, Kimiyuki Hiyama, Atsushi Matsumoto, Shoji Ikeda, Haruhiro Hasegawa, Katsuya Miura, Jun Hayakawa, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. 2009. Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices. *Applied Physics Express* (2009).

[23] Ashish Goel and Pankaj Gupta. 2010. Small subset queries and bloom filters using ternary associative memories, with applications. *ACM SIGMETRICS Performance Evaluation Review* (2010).

[24] Jing Li, Robert K. Montoye, Masatoshi Ishii, and Leland Chang. 2014. 1 Mb 0.41 μm^2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. *IEEE Journal of Solid-State Circuits* (2014).

[25] Shoun Matsunaga, Akira Katsumata, Masanori Natsui, Shunsuke Fukami, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. 2011. Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control. In *Symposium on VLSI Circuits (VLSIC)*.

[26] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of massive datasets*. Cambridge University Press.

[27] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing*.

[28] Anthony J. McAuley and Paul Francis. 1993. Fast routing table lookup using CAMs. In *Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*.

[29] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P. Jouppi. CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. *Technical Report HPL-2008-20*.

[30] http://pdf1.alldatasheet.com/datasheet-pdf/view/150649/MUSIC/MU9C4320L.html. 2017. MU9C4320L datasheet. (2017).

[31] http://www.datasheetcatalog.com/datasheets_pdf/M/U/9/C/MU9C1480B-50TAC.shtml. 2017. MU9C1480B datasheet. (2017).

[32] Raymond Fisman, Sheena S. Iyengar, Emir Kamenica, and Itamar Simonson. 2006. Gender differences in mate selection: Evidence from a speed dating experiment. *The Quarterly Journal of Economics* (2006).

[33] Emmanuel J. Candès and Michael B. Wakin. 2008. An introduction to compressive sampling. *IEEE Signal Processing Magazine* (2008).

[34] http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf. 2017. National institute of standards and technology. (2017).

[35] Thomas M. Cover and Joy A. Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

[36] Eamonn Keogh and Abdullah Mueen. 2011. Curse of dimensionality. In *Encyclopedia of Machine Learning*. Springer.

[37] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*. Springer.

[38] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. 2014. Blind seer: A scalable private dbms. In *IEEE Symposium on Security and Privacy (S&P)*.

[39] M. Sadegh Riazi, Ebrahim M. Songhori, and Farinaz Koushanfar. 2017. PriSearch: Efficient search on private data. In *Proceedings of the 54th Annual Design Automation Conference*. ACM.

[40] M. Sadegh Riazi, Neeraj K. R. Dantu, L. N. Vinay Gattu, and Farinaz Koushanfar. 2016. GenMatch: Secure DNA compatibility testing. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*.

Received April 2017; revised June 2017; accepted June 2017