The Strength of Weak Randomization: Easily Deployable, Efficiently Searchable Encryption with Minimal Leakage

David Pouliot
Portland State University
dpouliot@pdx.edu

Scott Griffy
Portland State University
scog@pdx.edu

Charles V. Wright
Portland State University
cvw@pdx.edu

***Abstract*—Efficiently searchable and easily deployable encryption schemes enable an untrusted, legacy service such as a relational database engine to perform searches over encrypted data. The ease with which such schemes can be deployed on top of existing services makes them especially appealing in operational environments where encryption is needed but it is not feasible to replace large infrastructure components like databases or document management systems. Unfortunately all previously known approaches for efficiently searchable and easily deployable encryption are vulnerable to inference attacks where an adversary can use knowledge of the distribution of the data to recover the plaintext with high probability.**

We present a new efficiently searchable, easily deployable database encryption scheme that is provably secure against inference attacks even when used with real, low-entropy data. We implemented our constructions in Haskell and tested databases up to 10 million records showing our construction properly balances security, deployability and performance.

I. Introduction

Many organizations today are moving to the cloud, shipping their critical data to servers over which they have little control. Encrypting data before uploading it into the cloud protects against theft or accidental disclosure, but standard encryption mechanisms also prevent the cloud service from performing any useful computation on the client's behalf. One of the most desirable operations on encrypted data is search.

The problem of searching on encrypted data involves inherent trade-offs between security, performance, and utility. One part of a cryptographic construction's utility is the expressiveness of the queries that it supports. For example, a construction that supports a large subset of Structured Query Language (SQL) has more utility than one that supports only exact-match queries. Another aspect of utility more relevant to this paper is the ease or difficulty of deploying a construction for use in the real world.

A. The Importance of Deployability

Searchable Symmetric Encryption (SSE) encompasses a class of efficient constructions for encrypted databases that provide security for data and queries [12], [15], [22], [46]. One practical limitation of SSE schemes is that they can be very difficult to retrofit into existing production systems. Grubbs [26] recounts one organization's experience, where a team of applied cryptographers invested 12 person-months of effort to implement and deploy a recent, high-performance SSE scheme [12]. In the end, the new construction proved too difficult to integrate with the businesses' existing systems, and the prototype was scrapped.

In the real world, there is demand for protections that can be deployed *immediately*. To meet this need, another parallel line of work has investigated "efficiently searchable" and "efficiently deployable" schemes that enable an untrusted server to efficiently index and search on encrypted data [4], [6], [8], [31], [39]. An *efficiently searchable encryption* (ESE) scheme is one that reveals some function of the plaintext in order to allow logarithmic search time using a legacy database or information retrieval system. This might be done through a query proxy rather than an complex database construction. An "easily deployable" efficiently searchable encryption scheme is one that retains (enough of) the expected format of the plaintext to enable search with an existing application or cloud service [31] [39].

Breaking from SSE, ESE gives up some security in exchange for the ability to easily deploy encryption on existing cloud services without the long delays and high cost of developing entirely new cryptographic systems (for example, see Grubbs [26]). This approach also frees the encryption developer from needing to worry about a whole host of practical systems issues including: availability, redundancy, performance, scaling, backups,

system monitoring, user accounts and access control, etc. The underlying cloud service already takes care of these.

The CryptDB system of Popa et al [44] first demonstrated the potential of this approach for protecting outsourced relational databases. By layering a collection of encryption techniques, including efficiently searchable and order-preserving encryption, CryptDB supports most SQL queries used by real applications. They achieved near-native performance on queries from a standard database benchmark, while storing only ciphertext in a standard MySQL database. Similar efforts from industry include SAP's SEEED project [25] and Microsoft's *Always Encrypted* feature in SQL Server 2016 [1], which is available commercially today.

Unfortunately, efficiently searchable constructions like deterministic and order-preserving encryption trade off more than just a little security. Inference attacks have recently been demonstrated that enable an adversary to recover some or all of the plaintext records if it has some auxiliary information about their statistical distribution [11], [18], [27], [41], [45], [50], [51]. In practice, these attacks allow the adversary to recover almost the entire database. Given the power of these attacks, it is reasonable to ask: *Is security feasible at all for efficiently searchable and easily deployable encryption in the real world?* Much to our surprise, it turns out that the answer is a qualified "yes."

B. Contributions

In this paper, we focus on maximizing security in operational scenarios where deployability is a hard requirement. As a consequence of this deployability requirement, the security of our schemes will necessarily be somewhat less compared to other approaches where easy real-world deployment is of no concern. Still, we aim to provide provable security against the most common adversaries, for applications that otherwise could afford no security at all.

We present a new, efficiently searchable, easily deployable database encryption scheme that is provably secure against inference attacks even when used with low-entropy data from the real world. The security of our schemes is tunable with a single parameter, allowing database owners to choose the most appropriate balance of security versus runtime performance and space overhead for the demands of their individual applications. We also achieve even stronger levels of security by bucketizing a small portion of the data.

Weakly Randomized Encryption. Our core technique is a generalization of a "folklore" encryption technique that we call *weakly randomized encryption* (WRE). WRE is a middle ground between deterministic encryption (DET) and conventional, strongly randomized encryption. DET enables efficient, logarithmic-time search because it allows a legacy server to create an index from only ciphertexts, but on the other hand, it provides very little security for real data [41]. Conventional (strongly) randomized encryption prevents the adversary from learning even a single bit about the plaintext [24], but in doing so, it also precludes the possibility of efficient search.

In a weakly randomized encryption, only a few bits of randomness sampled from a low-entropy distribution are used in each encryption. Our analysis shows that this is sufficient to protect against inference attacks if we choose the distribution carefully. In order to perform our WRE schemes, one must know the probability distribution of the plaintexts. We believe it is not unreasonable to ask that the data owner must know his data at least as well as the attacker does. The distribution can also be calculated during database initialization.

Deployability. Our constructions are compatible with standard SQL relational databases. They can be deployed immediately on popular cloud service platforms including Google Cloud SQL[1] and Amazon Relational Database Service[2]. They are efficiently scalable up to databases containing millions of records. We performed queries returning up to 10,000,000 records. Our encrypted database, including its server-generated indices, requires less than twice the space required for the plaintext DB. Query response time with our Poisson Random Frequency construction achieves response times within 27% of those of the plaintext database.

Security. We show that our construction is secure against a passive "snapshot" attacker. We give the adversary access to only the encrypted data and a source of auxiliary information. We assume he does not have access to the encrypted queries, the access patterns or return results. This model includes important real-world threats, including attackers who can obtain offline access to the encrypted database by SQL injection or by stealing a backup hard drive. In contrast, previous easily deployable, efficiently searchable schemes fail to achieve even this modest level of security [41].

We acknowledge that the adversary in this model is weaker than the more powerful adversaries that are typically considered for SSE or oblivious RAM. For new applications that do not require deployability on legacy

[1] https://cloud.google.com/sql/
[2] https://aws.amazon.com/rds/

infrastructure, we recommend that system builders use those stronger but less-deployable constructions.

Outline. The paper is organized as follows. We review related work in Section II. We introduce our notion of security for WRE against inference attacks in Section III and in Section IV we present the generic template for a weakly randomized encryption. Then, in Section V we give sequentially stronger variations on this idea, leading up to our most secure construction, WRE with bucketized Poisson salt allocation. We evaluate the performance of our new constructions experimentally with real databases in Section VI.

II. Related Work

Bellare, Boldyreva, and O'Neill [6] and Amanatidis, Boldyreva and O'Neill [4] proposed and analyzed early ESE schemes using deterministic encryption (DET). Those schemes are provably secure only when the plaintext database has high min-entropy [6].

Order-preserving encryption was first described in 2004 by Agrawal et al., who proposed a method to encrypt data so that the resulting ciphertexts retain the same ordering as the plaintext [3]. OPE was first studied formally by Boldyreva, Chenette, Lee, and O'Neill [8]. Boldyreva, Chenette, and O'Neill introduced the related notion of efficiently orderable encryption [9], in which a public, efficient function can be used to compare the ciphertexts. The similar notion of order-revealing encryption was proposed by Boneh et al. [10], and recent works give efficient symmetric schemes for ORE [14], [40]. The term "property-preserving encryption" is sometimes used to encapsulate both OPE/ORE and ESE [42].

Homomorphic encryption [21] and oblivious RAM [23], [47] offer very strong guarantees of security, but their practical performance is limited compared to other, more efficient techniques like SSE [12], [15], [22], [46]. Recent SSE schemes support rich queries [7], [16], [19], [30], [34], [36] and achieve fast performance even on very large data sets [12] by exploiting spatial locality [5], [17] and/or secure hardware extensions [20].

Deterministic ESE constructions were shown to be insecure against inference attacks [41], [11], [45]. Even more powerful attacks have been demonstrated against OPE and ORE [41] [18] [29]. In some passive attack scenarios, e.g. with a stolen hard drive, if the attacker has access to server logs and other system artifacts, then this may enable more powerful attacks [28]. Our techniques provide a necessary but not sufficient level of protection against such attacks. Protecting other system-level artifacts against inference attack remains an interesting open problem, and is beyond the scope of this work.

Other recent attacks assume an online adversary who can observe queries and their results in order to mount inference attacks [38] [35]. Countering such attacks remains an important open problem for all encrypted database constructions.

Techniques for "bucketizing" search tokens, for example with hash collisions, to reduce information leakage have been proposed in encrypted search schemes since their inception [6], [22], [39], [46]. Another line of work also uses bucketization to enable range queries over encrypted data without the use of ORE [32], [33], [49].

The paper most closely related to ours is a technical report by Lacharité and Paterson on *Frequency Smoothing Encryption* [37]. Their frequency-based homophonic encoding is equivalent to our proportional salt allocation method (see Section V-B).

III. Security Definitions

Our security definitions are closely modeled after the standard notion of security against a *chosen plaintext attack*. Like all previous efficiently searchable constructions, our scheme does not meet the standard definition of Indistinguishable Under Chosen Plaintext Attack (IND-CPA) security, as we must reveal the equality of some plaintexts in order to allow efficient searching. We extend the standard IND-CPA definition as follows.

Where the CPA adversary submits pairs of plaintext messages to its challenger, our adversary submits pairs of *lists* of messages. In the real world, a snapshot adversary does not know the order in which plaintext messages were added to the database. To capture this limitation on the adversary, in our game after the challenger randomly selects one list of messages, it randomly shuffles the selected list to prevent the adversary using any information about the original order. Finally, the challenger encrypts all messages in the shuffled list and provides the encrypted list back to the adversary. The adversary's task is then to determine which list was selected.

The two lists of messages are required to contain the same number of messages, and the messages (across both lists) must all be of the same size. Otherwise the adversary could use the size of the ciphertexts to distinguish between the lists of messages. The other requirement is that the order of messages in the lists must be random. Otherwise the ordering of the search tags could be used by the adversary to distinguish.

We call our security game Indistinguishability under chosen unordered database attack (IND-CUDA).

Definition 1 (Negligible Function)**.** *A function $: N \rightarrow N$ is negligible in k if for every positive polynomial $p()$ and sufficiently large $k, (k) < 1/p(k)$. Let $poly(k)$ and $negl(k)$ denote unspecified polynomial and negligible functions in k, respectively.*

Definition 2 (Pseudo-Random Function (PRF))**.** *Let $F : \{0,1\}^k x \{0,1\}^n \rightarrow \{0,1\}^m$ be an efficient keyed function. F is a pseudo-random function if for all probabilistic polynomial time distinguishers $\mathcal{D}$ there is a negligible function $negl$ such that*

$$\left| Pr[\mathcal{D}^{F_k(\cdot)} = 1] - Pr[\mathcal{D}^{f(\cdot)} = 1] \leq negl(k) \right|$$

where the first probability is taken over the uniform random choice of k and the randomness of $\mathcal{D}$ and the second probability it taken over the uniform choice of $f \in \textbf{\textit{Func}}_n$ and the randomness of $\mathcal{D}$.

Informally, a pseudo-random function is a polynomial time function that is indistinguishable from a truly random function by any polynomial time adversary.

Definition 3 (Statistical Distance)**.** *The* statistical distance Δ *between two random variables X, Y over a common domain ω is defined as:*

$$\Delta(X, Y) = \frac{1}{2} \sum_{\alpha \in \omega} \left| Pr(X = \alpha) - Pr(Y = \alpha) \right|$$

Two random variables X, Y are said to be ϵ-close if the statistical distance between them is at most ϵ. Variables X, Y are called *statistically indistinguishable* if $\epsilon = negl(\alpha)$ with security parameter α.

An important application of Definition 3 is its use in the probability of distinguishing between two random variables or two distributions [2]. This probability is bounded by the statistical distance between the distributions.

Definition 4 (Distinguishing Two Distributions)**.** *Let P_0 and P_1 be probability distributions on a finite set R. Then for every adversary $\mathcal{A}$, we have the distinguishing advantage of $\mathcal{A}$ between P_0 and P_1,*

$$Pr[\textsf{\textit{Dist}}_{\mathcal{A}}(P_0, P_1)] \leq \Delta(P_0, P_1)$$

Definition 5 (**SHUFFLE**)**.** *Let S be a set containing n distinct objects. A* SHUFFLE *of S is an ordered list of the objects in S. A* SHUFFLE *of the set $\{1, 2, ..., n\}$ is called a* SHUFFLE *of n.*

To shuffle a list, the set S is the indexes into the list. The shuffle of a list re-orders the indexes. Informally this is a permutation of a list.

If S contains n distinct objects, then there are exactly $n!$ SHUFFLES of n.

Definition 6 (**PSEUDO RANDOM SHUFFLE (PRS)**)**.** *Let* PRS *be a deterministic polynomial time function, on input a key $k \in \{0,1\}^n$, message $m \in \{0,1\}^*$, and set of messages $\{l_0, l_1, ...l_i\}$ where $l_i \in \{0,1\}^*$, outputs a* SHUFFLE *of $\{l_0, l_1, ...l_i\}$. We say* PRS *is a Pseudo Random Shuffle if:*

- *(Pseudorandomness:) For any probabilistic polynomial time algorithm $\mathcal{D}$ there is a negligible function* NEGL *such that*

  $$\left| Pr[\mathcal{D}(\text{PRS}(k, m, l))] - Pr[\mathcal{D}(\text{R}(l))] \right| \leq \text{NEGL}(n)$$

  where the first probability is taken over the uniform choice of $k \in \{0,1\}^n$, $m \in \{0,1\}^*$ and the randomness of $\mathcal{D}$, and the second probability is taken over $\text{R}(l)$*, where* R *is a uniformly random shuffle algorithm.*

Definition 7 (**The IND-CUDA Indistinguishability Experiment**)**.** *Let Π = (*Gen*,* Enc*,* Dec*) be a WRE searchable encryption scheme with message space $\mathcal{M}$ and key space $\mathcal{K}$. Let $\mathcal{X}$ be the security parameter for Π. Let $\mathcal{A}$ be an poly-bounded adversary.*

IND-CUDA$_{\Pi,\mathcal{A}}(n, \mathcal{X})$*:*

- $(k_0, k_1) \leftarrow \textsf{Gen}(1^n)$.
- $\mathcal{A}(\mathcal{X}, n_0, n_1)$ *chooses a pair of lists of messages M_0, M_1 where $|M_0| = |M_1|$ and for all $m_i, m_j \in M_0, m_k, m_l \in M_1, |m_i| = |m_j| = |m_k| = |m_l|$*
- *A uniform bit $b \in 0, 1$ is chosen.*
- $edb \leftarrow \textsf{Enc}((k_0, k_1, \mathcal{X}), PRS(M_b))$.
- $b' \leftarrow \mathcal{A}(edb)$
- *The output of the experiment is 1 if $b' = b$, and 0 otherwise. We write* ***IND-CUDA***$_{\mathcal{A},\Pi}$ *= 1 if the output of the experiment is 1, and in this case we say that $\mathcal{A}$ succeeds.*

Definition 8 (IND-CUDA Indistinguishability)**.** *We say that the encryption scheme Π with security parameters λ and n_0, n_1 has IND-CUDA security if, for all probabilistic polynomial time adversaries $\mathcal{A}$,*

$$Pr[\textbf{\textit{IND-CUDA}}_{\mathcal{A}(n_0,n_1,\mathcal{X})} = 1] \leq \frac{1}{2} + negl(\mathcal{X}, n_0, n_1)$$

where $(\mathcal{X}, n_0, n_1)$ are the security parameters of our scheme.

In Section V, we introduce our constructions and in Section V-C we use our security definitions to evaluate

the single-column security of one of our constructions (which uses a Poisson distribution). Our schemes are tune-able to trade-off performance for security, and have acceptable performance and security for sizable databases as shown in Section VI.

IV. Weakly Randomized Encryption

In this section we formalize and extend a "folklore" technique that we call *weakly randomized encryption* (WRE) in text and in Figure 1. This is the basis for all variants described in Section V.

Weakly Randomized Encryption

Let F be a pseudorandom function with key length n_1. Let $\Pi' = (Gen', Enc', Dec')$ be an IND-CPA secure private key encryption scheme with message space $m \in \{0,1\}^*$ and key length n_0. Let $getSalts$ be a function that on input message $m \in \{0,1\}^*$ and message probability distribution function P_m and a security parameter $\mathcal{X}$, outputs S, a list of integers representing the salts and P_S, a probability distribution over the salts. Define a private-key weakly randomized encryption scheme Π as follows:

- Gen: on input $1^{n_0}, 1^{n_1}$ run $Gen'(1^{n_0})$ receiving key k_0 and choose uniform $k_1 \in \{0,1\}^{n_1}$. Choose security parameter $\mathcal{X}$.
- Enc: on input keys k_0, k_1, security parameter $\mathcal{X}$, and a message m, choose a random salt
  $(S, P_S) \leftarrow getSalts(m, P_m, \mathcal{X})$
  $s \xleftarrow{\$} sample(S, P_S)$
  Output the (search tag, ciphertext):
  $$(t, c) \leftarrow \big(F_{k_1}(s||m),\ Enc'_{k_0}(m)\big)$$
- Dec: on input key k_0, and ciphertext (t, c) output plaintext message
  $$m \leftarrow Dec'_{k_0}(c)$$
- Search: on input keys k_0, k_1, parameter $\mathcal{X}$, and a message m, $(S, P_S) \leftarrow getSalts(m, P_m, \mathcal{X})$
  Output query ($query$) on table (T) containing search tag column (T_t) as shown below:
  $$\begin{aligned} query \leftarrow & \big(T_t = F_{k_1}(s_1||m)\big) \vee \\ & \big(T_t = F_{k_1}(s_2||m)\big) \vee ... \vee \\ & \big(T_t = F_{k_1}(s_{|s|}||m)\big) \end{aligned}$$

Fig. 1: Weakly Randomized Encryption, Decryption and Search

Previous efficiently searchable encryption constructions either have specific requirements on the plaintext data, like high min-entropy [6], or place limitations on the adversary, such like limiting oracle queries [4] to distinct plaintexts.

We can reduce the vulnerability of deterministic encryption due to frequency analysis and other leakage inference attacks by adding a small amount of randomness to the encryption.

We show how a weakly randomized encryption scheme can be constructed as the composition of (i) any efficiently-searchable encryption scheme that satisfies the security definitions from [4] and (ii) a weak randomization, or "salting," function. In this work, we construct our schemes using a variation of the Amanatidis, Boldyreva, and O'Neill [4] ESE, which is itself composed of a randomized encryption scheme (RE) that leaks nothing about the plaintext and a pseudo-random function (PRF) that leaks nothing except equality.

Encryption. The WRE encryption takes as input: symmetric keys k_0, k_1; a plaintext m; and the probability distribution P_M of the plaintexts. The encryption algorithm begins by calling the $getSalts$ subroutine to pseudorandomly generate a probability distribution P_S over a set S of salts for the message m. The $getSalts$ subroutine uses the plaintext distribution P_M to choose a distribution for the salts that makes the frequencies of the ciphertexts (nearly) independent of the plaintext. We give a handful of candidate algorithms for $getSalts$, and evaluate their security, in the following sections. A salt $s \in S$ is chosen at random according to P_S and is prepended to the message. The encoding of the pre-pended salts must ensure that no pairs of salts and messages of different lengths results in the same search tag. Finally, the salt and plaintext are input into the PRF to create the search tag and the plaintext is encrypted with the randomized encryption algorithm.

Search. To search the encrypted database for all records with plaintext equal to m, the client first computes all possible search tags $t_1, t_2, \ldots, t_n$ for m and then requests all records having tags equal to t_1 or t_2 ... or t_n. Because the number of unique search tags for each plaintext is small, WRE allows the server to build useful indexes on the encrypted data, just as with DET. To perform the search for each t_i, the server can use built-in indexing techniques to return the list of matching records on columns added by our scheme. Because no custom indexing scheme needs be used, this allows it to be deployed on unmodified DBMS services.

Decryption. Given a search tag and a randomized ciphertext, the WRE decryption routine discards the tag and uses the randomized encryption scheme's decryption function on the ciphertext to obtain the plaintext.

Updates. One advantage of WRE versus stronger searchable encryption schemes like SSE is updates are simple with WRE. To insert a new record in the encrypted database, we use the encryption function to obtain its weakly randomized search tag and its (strongly randomized) ciphertext. Then we simply append the tag and ciphertext to the database. If we assume new records inserted are drawn from the same plaintext distribution, then adding new records will not affect the WRE tag frequencies. Thus it is secure under the snapshot adversary model. The challenge with SSE updates comes from a different security model that allows the adversary to query the database while providing forward security. Because of security model and the encrypted indexes used by SSE, SSE typically performs updates in batches using new keys resulting in multiple indexes.

Future work will address security when the distribution changes from updates or if the adversary has specific knowledge of the updated records.

The improvement in security, if any, of WRE over deterministic encryption is not immediately clear. Surprisingly, our analysis also shows that, with a carefully chosen $getSalts$ algorithm, we can construct a weakly randomized encryption that leaks virtually no information about the plaintext to a snapshot adversary who knows the distribution P_M.

V. WRE Schemes

In this section, we present our variants that each complete the WRE construction described in Section IV. We first present simpler/weaker constructions to give the reader an understanding of our motivations for our later, more secure schemes in Sections V-C and V-C1. We do not fully analyze the security of these weaker schemes because we believe they are inferior to later schemes.

A. Fixed Salts Method

We refer to the "folklore" version of weakly randomized encryption as the "fixed salts" method, because it always uses a constant number of salts for every plaintext, regardless of the frequency of the plaintext. We label the security parameter of this scheme as N, the number of unique salts per plaintext.

Notion of Security. If a plaintext m occurs in the unencrypted database with frequency p, then with fixed salts, each of m's N ciphertexts will occur in the EDB with frequency $\frac{p}{N}$. Intuitively, the fixed salt method improves on the security of deterministic encryption because it reduces the differences in the plaintext frequencies.

Limitations. First, the overall improvement to security is small. For large databases, the adversary can still guess the plaintext with very high accuracy. Second, the fixed salt WRE is not very efficient. In order to achieve any security for a database of moderate size, it needs a large number of salts, making query processing unnecessarily intensive, especially for low-frequency plaintexts. We could potentially improve both of these aspects if we modified the chance of picking each salt with the frequency of its respective plaintext. We formalize this idea in the next section.

B. Proportional Salts Method

The fixed salts method can be improved by taking into account the frequencies of the plaintexts in the database. Intuitively, we would like each search tag to occur with roughly the same frequency, regardless of the plaintext. In the proportional salts method, we allocate a different number of salts to each plaintext, in proportion to its frequency in the plaintext data. Let the security parameter be the total number of unique ciphertexts be N_T. Then for a plaintext m with frequency $P_M(m)$, we use $N_m \approx P_M(m) \cdot N_T$ salts. Therefore, for any two plaintexts $m_0, m_1 \in \mathcal{M}$, their search tags will appear in the EDB with approximately the same frequency.

Limitations. Unlike the fixed salts method, proportional salt allocation requires that the data owner must know the plaintext distribution P_M in order to encrypt a message.

Another limitation of proportional salts stems from the fact that we must allocate an integer number of salts for each plaintext. This gives rise to an aliasing problem, where in certain situations using more salts can actually reduce the security. For example, consider an example database column with $P_M(m_1) = 0.7$ and $P_M(m_2) = 0.3$. For $N_T = 10$, this works out nicely, but if we encrypt this database with $N_T = 12$, then we will round our number of search tags to 8 for plaintext m_1, each with frequency 0.0875, and 4 for plaintext m_2, each with frequency 0.075. Given sufficiently many encrypted records, the adversary will be able to distinguish the plaintexts using this frequency disparity.

In the following sections, we address the aliasing problem of proportional salts by randomizing the frequencies of salts.

C. Poisson Random Frequencies

A Poisson process is a simple stochastic process often used to model the arrival of events in a system, for example the occurrence of earthquakes in a geographical region, or the arrival of buses at a bus stop. In a Poisson

process with rate parameter λ, the times between arrival events, called the "interarrival times," are independent and identically distributed, and they follow an Exponential distribution with parameter λ. The number of arrivals in an interval of length t is independent of the events in all intervals before and after, and it is Poisson distributed with expected value λt.

In the Poisson variant of WRE, the security parameter is the Poisson rate parameter λ. On expectation, this method will generate about $\lambda + |\mathcal{M}|$ search tags in total across all plaintexts. To allocate salts for plaintext $m \in \mathcal{M}$ and to assign their relative weights, we sample arrivals in the interval $[0, P_M(m)]$ from a Poisson process with rate λ. Let the number of arrival events in the interval be N, and let their times be denoted $a_1, \ldots, a_N$. Additionally, we define $a_0 = 0$ and $a_{N+1} = P_M(m)$. The interarrival times are $x_i = a_i - a_{i-1}$ for $i \in 1, \ldots, N+1$.

Based on the outcome of this experiment, we allocate $N+1$ salts to plaintext m, and when we encrypt m, we choose salt i with probability $\frac{x_i}{P_M(m)}$. The resulting search tag will then have frequency equal to x_i in the encrypted database. Also note that N has a Poisson distribution, thus on average we will allocate about $\lambda \cdot P_M(m) + 1$ salts to plaintext m.

The pseudocode for our Poisson method's algorithm is shown below in Algorithm 1.

Algorithm 1 Poisson Salt Distributions

```
1:  function GetSalts-Poisson(P_M, m, k, λ)
2:      s = 0
3:      E = Exponential(λ)
4:      total = 0
5:      while total < P_M(m) do
6:          s = s + 1
7:          weight[s] ← Sample(E)
8:          total = total + weight[s]
9:      weight[s] ← P_M(m) − (total − weight[s])
10:     S = [1, s]
11:     for s ∈ S do
12:         P_S(s) ← weight[s] / P_M(m)
13:     return S, P_S
```

Security. Our analysis shows how the unique properties of the Poisson process make it ideally suited for use in weakly randomized encryption. Most critically, the Poisson process guarantees that, subject to one constraint on λ, all search tag frequencies for all plaintexts are pseudorandom samples from indistinguishable Exponential distributions. Therefore a computationally bounded adversary learns nothing about the plaintext from the frequencies of the ciphertexts.

Proof sketch. In the Poisson approach, the frequency of the first salt for each plaintext is not drawn from the same Exponential distribution as the others. To see why this is so, notice that the Poisson process may generate zero arrival events in the interval $[0, P_M(m)]$. This occurs whenever the first arrival time from the Poisson process occurs after the end of the interval; in other words, when the first interarrival time is greater than $P_M(m)$. Then we have only a single salt, and hence a single ciphertext that appears in the encrypted database with the same probability as the plaintext, $P_M(m)$. Therefore the distribution of the first search tag frequency is not in fact an Exponential; all the probability mass that the Exponential would assign to values greater than $P_M(m)$ is instead lumped onto the point $P_M(m)$. We call this distribution a "capped Exponential" with parameters λ and $\tau = P_M(m)$. Figure 2 illustrates the difference between the capped and regular Exponential distributions.

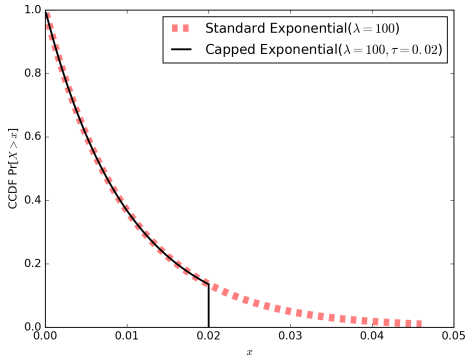

Fig. 2: Complementary cumulative distribution for capped versus standard Exponentials

The adversary could attempt to exploit this difference to his advantage in the IND-CUDA game. In the extreme case, the adversary would choose M_0 with all unique plaintexts, and choose M_1 where all $m \in M_1$ are the same plaintext. With a low value of λ, the messages from M_0 in this example would all be drawn from the capped exponential while all of the messages from M_1 would not. However, with a high enough λ in relation to the number of messages in M_0, all of the messages from M_0 would be drawn from the non-capped exponential with very high probability. The important point is to choose an appropriate λ parameter based on the distribution you are encrypting.

The statistical distance between the standard

Exponential(λ) and the capped Exponential with λ and τ is defined in Definition 3 as one half of the total variation distance between the two distributions. Notice that the two distributions are identical to the left of τ. Therefore all of the difference in distribution comes from the upper tail of the standard Exponential, where the capped Exponential assigns zero probability. From the definition of the Exponential distribution, this quantity is

$$\begin{aligned}&\Delta(Exp(\lambda), CappedExp(\lambda,\tau))\\ =&Pr(X>\tau|X\sim Exp(\lambda))\\ =&e^{-\lambda\tau}\end{aligned}$$

Thus the probability of the adversary distinquishing between the capped exponential and exponential distributions is negligible in λ, which is the goal of our security definition.

If we let $\tau = \max_m P_M(m)$ be the smallest plaintext frequency, then by increasing λ relative to τ, we can make this probability arbitrarily small. Furthermore, we can calculate the Poisson rate parameter λ that is required to achieve a desired security parameter ω, where $\omega \leq Pr(X > \tau))$.

$$\lambda \geq \frac{\log \omega}{\tau}$$

Limitations. When the adversary has the frequencies of all search tags and knows P_M, Lacharite and Paterson [43] pointed out another possible attack, wherein the adversary finds a set of search tags whose counts sum up to the expected count for a (set of) target plaintext(s). The adversary might then reasonably conclude that those search tags all represent encryptions of the given plaintext(s).

When the adversary targets a single plaintext, it must solve an instance of the subset sum problem (SSP). When targeting multiple plaintexts simultaneously, the adversary must solve an instance of the multiple knapsack problem (MKP). While both problems are NP, there exist efficient approximation algorithms, for example [13].

However, even if the adversary can find a solution to the computational problem, this does not guarantee that the matching that it finds will be correct. To see that this is true, consider the case where each search tag occurs exactly once. Then all possible plaintext-to-search-tag matchings give equally valid solutions to the problem, and the adversary can do no better than random guessing. We leave a more detailed exploration of the efficacy of such attacks for future work. Instead, in the following section, we present an improved WRE construction using bucketization to eliminate the attack entirely.

1) Bucketized Poisson Random Frequencies: The Poisson WRE approach above generated randomized search tags for each plaintext. Doing so makes any one search tag equally likely under all possible plaintexts, so the adversary learns nothing by examining a single tag. Unfortunately, that does not guarantee security against an adversary who considers the combined frequencies of several tags at once. In this section, we show how a simple extension of the Poisson WRE approach, using bucketization, can protect against the matching attacks described in the previous section.

Figure 3 illustrates the difference in the two schemes. In the (non-bucketized) Poisson WRE, we sample a set of points from a Poisson process for each plaintext m, over the interval $[0, P_M(m)]$. We then use the inter-arrivals between the points to determine the frequencies of the search tags for m. This is equivalent to starting with the set of points $\{F_M(m) : m \in \mathcal{M}\}$ and then sampling more points from the Poisson process over the interval $[0, 1]$.

In the Bucketized Poisson approach, we omit the points from $F_M(m)$, and we simply sample from the Poisson process, independent of the plaintext frequencies. As a result, some inter-arrival intervals will overlap with the intervals for more than one plaintext. Notice that in Figure 3 with the bucketized construction, the tag t_3 can represent either plaintext m_1 or m_2.

The Bucketized Poisson also makes a slight modification to the encryption and search algorithms from Figure 1. Instead of inputting the plaintext appended to a salt to the PRF, just the salt is given.

- Enc: on input keys k_0, k_1, security parameter $\mathcal{X}$, plaintext distribution P_M and a message m, choose a uniform salt
  $(S, S_M) \leftarrow getSalts(P_M, \mathcal{X})\}$
  $s \xleftarrow{\$} sample(S(m), S_M(m))$
  Output the (search tag, ciphertext):
  $$(t, c) \leftarrow \ \big(F_{k_1}(s),\ Enc'_{k_0}(m)\big)$$
- Search: on input keys k_0, k_1, parameter $\mathcal{X}$, and a message m, let $(S, S_M) \leftarrow getSalts(P_M, \mathcal{X})\}$ $s = S(m)$. Output search query for
  $$q \leftarrow T_t = F_{k_1}(s_1) \vee T_t = F_{k_1}(s_2) \vee ... \vee T_t = F_{k_1}(s_{|s|})$$

This additional ambiguity completely removes the small advantage that an adversary might obtain from the imperfection of the capped exponential distribution. It also negates any kind of frequency-based matching

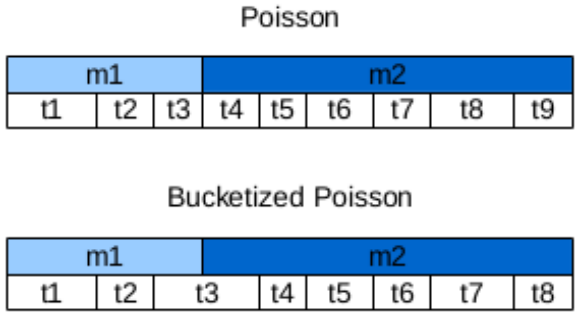

Fig. 3: Poisson Search Tag Frequency Example

attack, because the tag frequencies and the plaintext frequencies are independent. The downside is that with the bucketized WRE, query results will contain a small number of false positives. The false positive rate is controlled by parameter λ: increasing λ (thus decreasing the expected frequency of each tag) decreases the expected number of false positives.

Theorem V.1 (Single-Column Security for Bucketized Poisson WRE)**.** *A Bucketized Poisson WRE scheme with parameters (λ, n_0, n_1), is IND-CUDA secure.*

Proof sketch. With the Bucketized Poisson algorithm, the actual ciphertext search tags will have exactly the same values and the same frequency, no matter which M_0 or M_1 is encrypted. The only difference will be the ordering of these search tags. Since the ordering is determined by the output of a pseudo-random shuffle, the adversary cannot learn anything from this ordering either. The security of this construction also does not depend on λ like the first Poisson approach. The λ value will affect the false positive rate and performance, but not the security.

VI. Performance Evaluation

We implemented several flavors of weakly randomized encryption, including the fixed salts method and Poisson salt allocation, in the Haskell programming language. To evaluate the performance of our prototype on realistic data and queries at a variety of scales, we used the SPARTA [48] framework from MIT-LL.

The SPARTA test framework includes a data generator and a query generator. The data generator builds artificial data sets with realistic statistics based on real data from the US Census and Project Gutenberg. The query generator creates queries for this test database based on the desired query types and number of return results.

Algorithm 2 Bucketized Poisson

function GetSalts-Poisson(P_M,M,m, k, λ)
  $s = 0$
  $wordFr = [], salts = []$
  $E = Exponential(\lambda)$
  $total = 0$
  while $total < 1.0$ **do**
    $s = s + 1$
    $weight[s] \xleftarrow{\$} Sample(E)$
    $total = total + weight[s]$
  $weight[s] \leftarrow 1.0 - (total - weight[s])$
  $M' \leftarrow PRS(M)$
  $fr = P_M(m'_1) + ...P_M(m'_{x-1})$ where $m = m'_x$
  $i = 0, cdf = 0$
  while $cdf < fr$ **do**
    $cdf = cdf + weights[i]$
    $i = i + 1$
  $weights[i] = cdf - fr$
  $cdf = fr$
  while $cdf < (fr + P_M(m))$ **do**
    $wordFr.append\left(\frac{weights[i]}{fr}\right)$
    $salts.append(i)$
    $i = i + 1$
    $cdf \leftarrow cdf + weights[i]$
  if $cdf > (fr + P_M(m))$ **then**
    $dif \leftarrow (fr + P_M(m)) - cdf$
    $wordFr.append\left(\frac{weights[i]-dif}{fr}\right)$
    $salts.append(i)$
  return $(salts, wordFr)$

A. Experimental Setup

We used the database generator to generate databases with 100,000 records, 1 million records and 10 million records. We generated over 1,000 queries for each database, consisting of a mix of queries that returned result sizes between 1 and 10,000 records.

We encrypted the columns `fname`, `lname`, `ssn`, `city`, and `zip` with WRE. The rest of the SPARTA columns were inserted into the test database in plaintext.

Each encrypted column is expanded into two columns: one 64 bit Integer column for the WRE search tag and another column to hold the (strongly randomized) AES-encrypted data. The plaintext table contains 23 columns. Therefore the ciphertext table contains the 23 encrypted data columns, plus the 5 additional search tag columns.

We tested the performance of the fixed salt method with 100 and 1,000 salts, and we tested Poisson salt allocation using λ of 100, 1,000, and 10,000.

We performed the tests with the client and the database server located on the same local network via a 1 Gbps Ethernet switch. The server has 12 CPU cores (dual Xeon E5645), 64GB of RAM, and an array of 10k RPM hard drives. It runs the Ubuntu Server 14.04 operating system and Postgres 9.6 as the DBMS.

B. Experimental Results

Ciphertext Expansion. Table I shows the overall ciphertext expansion, including the ciphertext expansion from the AES encrypted data, the additional search tag columns and the additional indexes on the search columns. Note that the number of salts used and whether a fixed salt or a Poisson Salt Distribution do not affect the database size. The database ciphertext expansion is directly related to the number and type of columns encrypted

Encryption Type	DB Size	DB + Indexes Size
100k Plaintext	112 MB	136 MB
100k Encrypted	156 MB	244 MB
1M Plaintext	1116 MB	1365 MB
1M Encrypted	1558 MB	2447 MB
10M Plaintext	11 GB	13 GB
10M Encrypted	15 GB	24 GB

TABLE I: Ciphertext Expansion

Database Creation. Inserting 10 million plaintext records took a total 6,356 seconds on average. Inserting 10 million ciphertext records took 58,604 seconds on average. Because the database must only be initialized once, the practical impact of this 9x slowdown is not especially significant for most applications. Also, we believe that with a little effort, the performance of our un-optimized implementation could be improved substantially.

Query Runtime. We performed two variations of each SPARTA-generated query. The first variation takes the form *SELECT ID from main where ....* Since column `ID` is the primary key, these queries only require that the DBMS scan the indexes to find the list of matching records. The second variation takes the form *SELECT * from main where ....* This selects the entire record, and thus requires retrieving the encrypted records from storage and transferring them across the network. The time shown for each query includes the time to compute the encrypted query.

Since caching can have a big impact on database performance, we ran each set of queries under two scenarios. In the first scenario we cleared the caches in the OS and in the Postgres database before running each query. To clear the Postgres cache, we restarted Postgres. To clear the OS cache, we ran the following:

```
echo 3 > /proc/sys/vm/drop_caches
```

In the other scenario, the cache was left alone.

Figures 4 and 5 display tests run with a cold cache. Figures 6 and 7 have the results of the warm cache tests. The results of these experiments show that the WRE schemes achieve query response times with our Poisson Random Frequency construction within 27% of plaintext database response times on equality queries. As expected, as the number of unique search tags increases, so does the query response time. Across all of our experimental configurations, the Fixed Salt scheme with 1000 salts is slower than the Poisson construction with $\lambda = 1000$, and similarly, the Poisson with $\lambda = 1000$ performs slightly slower than Poisson with $\lambda = 100$. This result is not surprising, since the Fixed Salt technique generates 1000 tags for each plaintext, while the $\lambda = 1000$ results in $\lambda + |M|$ tags for the entire column.

Bucketized Poisson False Positives. In section V-C1, we mentioned that the Bucketized Poisson algorithm may result in false positives in the search result. Figures 8 and 9 show the false positives introduced on the SPARTA queries used in our performance evaluation. The X axis shows the number of records returned for each query with Poisson salt allocation, which does not introduce false positives. The Y axis shows the number of records returned for the same queries with the bucketized version of the algorithm.

With lower values of λ, the Bucketized Poisson algorithm appears to mask the true number of return results. In Figure 9, with $\lambda = 10,000$, we see some correlation between the number of matching records in the database and the number of ciphertext records that match the bucketized query. However, in Figure 8 with $\lambda = 1000$, the relationship is much weaker. In the future, this masking might be leveraged to prevent reconstruction attacks [35], [38], where an adversary uses access pattern leakage to recover the contents of the database.

VII. Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-15-C-4070. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SSC Pacific.

Reset Cache Before Each Query

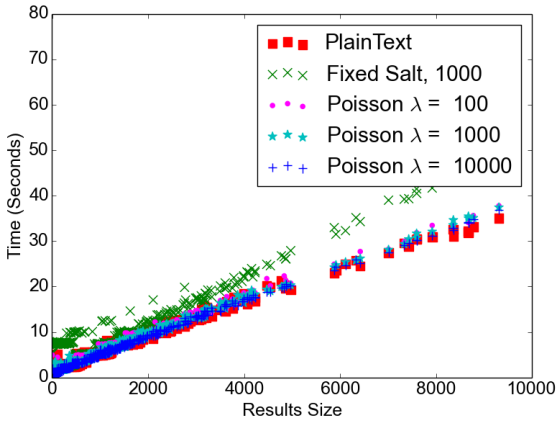

Fig. 4: "SELECT ID" Equality

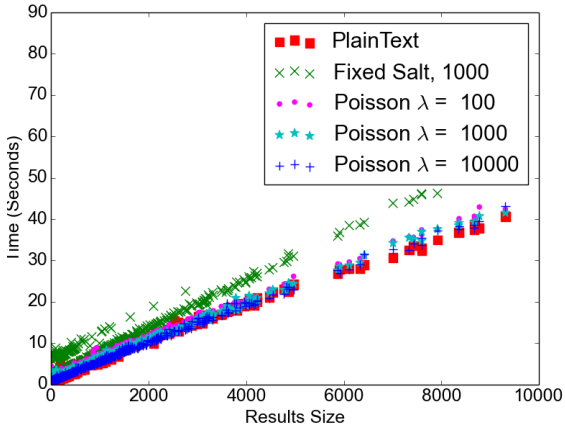

Fig. 5: "SELECT *" Equality

Warm Cache - No Cache Reset

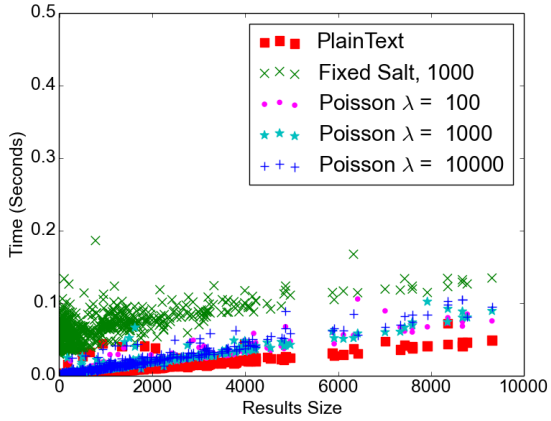

Fig. 6: "SELECT ID" Equality

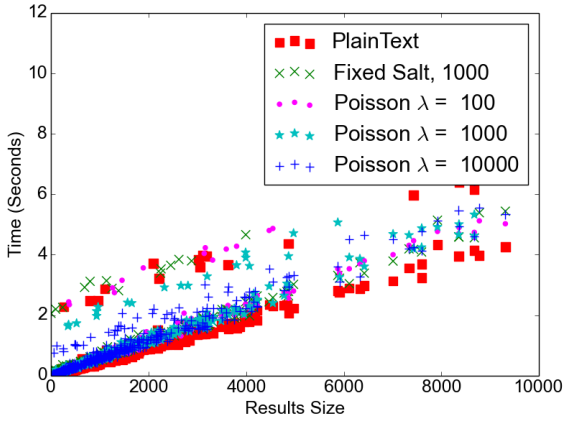

Fig. 7: "SELECT *" Equality

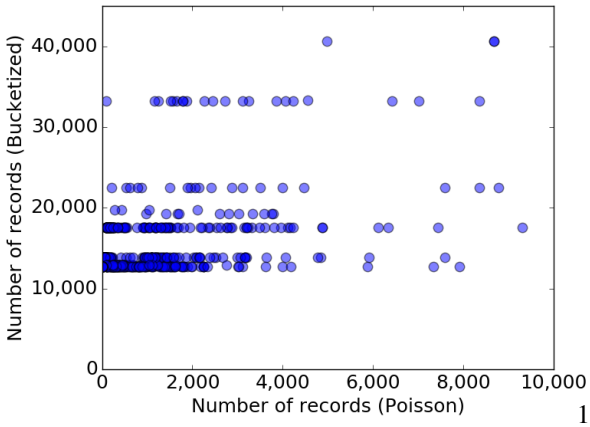

Fig. 8: Bucketized Poisson False Positive ($\lambda = 1000$)

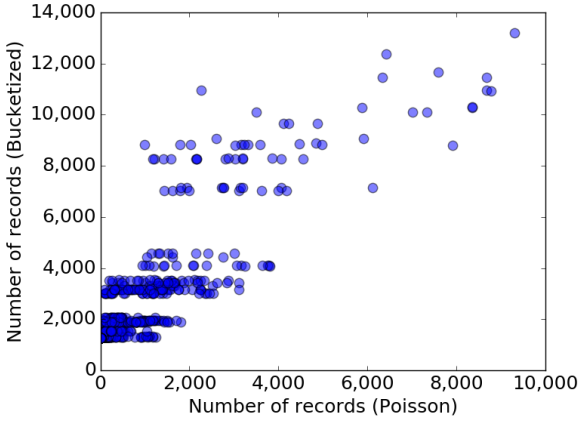

Fig. 9: Bucketized Poisson False Positive ($\lambda = 10,000$)

REFERENCES

[1] "Always encrypted (database engine)," https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine, accessed: 2017-10-31.

[2] "A graduate course in applied cryptography," http://toc.cryptobook.us/, accessed: 2018-5-1.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *SIGMOD*, 2004, pp. 563–574.

[4] G. Amanatidis, A. Boldyreva, and A. O'Neill, "Provably-Secure Schemes for Basic Query Support in Outsourced Databases." in *DBSec*, ser. Lecture Notes in Computer Science, S. Barker and G.-J. Ahn, Eds., vol. 4602. Springer, 2007, pp. 14–30. [Online]. Available: http://dblp.uni-trier.de/db/conf/dbsec/dbsec2007.html#AmanatidisBO07;http://dx.doi.org/10.1007/978-3-540-73538-0_2;http://www.bibsonomy.org/bibtex/294c62ccd6e81c58c00817ce4cb858e60/dblp

[5] G. Asharov, M. Naor, G. Segev, and I. Shahaf, "Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 1101–1114.

[6] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and Efficiently Searchable Encryption," in *CRYPTO*, 2007, pp. 535–552.

[7] T. Boelter, R. Poddar, and R. A. Popa, "A secure one-roundtrip index for range queries," Cryptology ePrint Archive, Report 2016/568, 2016, https://eprint.iacr.org/2016/568.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill, "Order-preserving symmetric encryption," in *EUROCRYPT*, 2009, pp. 224–241.

[9] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: improved security analysis and alternative solutions," in *CRYPTO*, 2011, pp. 578–595.

[10] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation." in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015, pp. 563–594. [Online]. Available: http://dblp.uni-trier.de/db/conf/eurocrypt/eurocrypt2015-2.html#BonehL0SZZ15

[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-Abuse Attacks Against Searchable Encryption." in *ACM Conference on Computer and Communications Security*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 668–679. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2015.html#CashGPR15

[12] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation." in *NDSS*. The Internet Society, 2014. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2014.html#CashJJJKRS14

[13] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[14] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *International Conference on Fast Software Encryption*. Springer, 2016, pp. 474–493.

[15] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in *CCS*, 2006, pp. 79–88.

[16] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 185–198.

[17] I. Demertzis and C. Papamanthou, "Fast searchable encryption with tunable locality," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1053–1067.

[18] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1155–1166. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978379

[19] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M.-C. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," 2015, pp. 123–145.

[20] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi, "Hardidx: practical and secure index with sgx," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2017, pp. 386–408.

[21] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–169.

[22] E.-J. Goh, "Secure Indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003. [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2003.html#Goh03;http://eprint.iacr.org/2003/216;http://www.bibsonomy.org/bibtex/235b9c14f05f89b78a4e03253222c786b/dblp

[23] O. Goldreich, "Towards a theory of software protection and simulation by oblivious rams," in *STOC*, A. V. Aho, Ed. ACM, 1987, pp. 182–194. [Online]. Available: http://dblp.uni-trier.de/db/conf/stoc/stoc87.html#Goldreich87

[24] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[25] P. Grofig, M. Hrterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schrpfer, and W. Tighzert, "Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data." in *Sicherheit*, ser. LNI, S. Katzenbeisser, V. Lotz, and E. R. Weippl, Eds., vol. 228. GI, 2014, pp. 115–125. [Online]. Available: http://dblp.uni-trier.de/db/conf/sicherheit/sicherheit2014.html#GrofigHHKKSST14

[26] P. Grubbs, "On deploying property-preserving encryption," 2016, real World Cryptography. [Online]. Available: https://drive.google.com/file/d/0Bzm_4XrWnl5zWndfZTVsRkpyWm8/view

[27] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov, "Breaking web applications built on top of encrypted data." *IACR Cryptology ePrint Archive*, vol. 2016, p. 920, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2016.html#GrubbsMNRS16

[28] P. Grubbs, T. Ristenpart, and V. Shmatikov, "Why your encrypted database is not secure," Cryptology ePrint Archive, Report 2017/468, 2017, http://eprint.iacr.org/2017/468.

[29] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 655–672. [Online]. Available: https://doi.org/10.1109/SP.2017.44

[30] F. Hahn and F. Kerschbaum, "Poly-logarithmic range queries on encrypted data with small leakage," in *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*. ACM, 2016, pp. 23–34.

[31] W. He, D. Akhawe, S. Jain, E. Shi, and D. X. Song, "ShadowCrypt: Encrypted Web Applications for Everyone." in *ACM Conference on Computer and Communications Security*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 1028–1039. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2014.html#HeAJSS14

[32] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 21, no. 3, pp. 333–358, 2012.

[33] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 720–731.

[34] S. Kamara, "Encrypted search." *ACM Crossroads*, vol. 21, no. 3, pp. 30–34, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/crossroads/crossroads21.html#Kamara15

[35] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic Attacks on Secure Outsourced Databases." in *ACM Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1329–1340. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2016.html#KellarisKNO16

[36] F. Kerschbaum and A. Tueno, "An efficiently searchable encrypted data structure for range queries," *arXiv preprint arXiv:1709.09314*, 2017.

[37] M.-S. Lacharite and K. G. Paterson, "Frequency-smoothing encryption: preventing snapshot attacks on deterministically-encrypted data," Cryptology ePrint Archive, Report 2017/1068, 2017, https://eprint.iacr.org/2017/1068.

[38] M. Lacharit, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 297–314.

[39] B. Lau, S. P. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva, "Mimesis Aegis: A Mimicry Privacy Shield-A System's Approach to Data Privacy on Public Cloud." in *USENIX Security Symposium*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 33–48. [Online]. Available: http://dblp.uni-trier.de/db/conf/uss/uss2014.html#LauCSJLB14

[40] K. Lewi and D. J. Wu, "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds." in *ACM Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1167–1178. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2016.html#LewiW16

[41] M. Naveed, S. Kamara, and C. V. Wright, "Inference Attacks on Property-Preserving Encrypted Databases." in *ACM Conference on Computer and Communications Security*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 644–655. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2015.html#NaveedKW15

[42] O. Pandey and Y. Rouselakis, "Property preserving symmetric encryption." in *EUROCRYPT*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, 2012, pp. 375–391. [Online]. Available: http://dblp.uni-trier.de/db/conf/eurocrypt/eurocrypt2012.html#PandeyR12

[43] K. Paterson and M.-S. Lacharité, Personal communication, December 2017.

[44] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing." in *SOSP*, T. Wobber and P. Druschel, Eds. ACM, 2011, pp. 85–100. [Online]. Available: http://dblp.uni-trier.de/db/conf/sosp/sosp2011.html#PopaRZB11

[45] D. Pouliot and C. V. Wright, "The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption." in *ACM Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1341–1352. [Online]. Available: http://dblp.uni-trier.de/db/conf/ccs/ccs2016.html#PouliotW16

[46] D. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searching on Encrypted Data," in *S&P*, 2000, pp. 44–55.

[47] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage." in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013, pp. 253–267. [Online]. Available: http://dblp.uni-trier.de/db/conf/sp/sp2013.html#StefanovS13

[48] M. Varia, B. Price, N. Hwang, A. Hamlin, J. Herzog, J. Poland, M. Reschly, S. Yakoubov, and R. K. Cunningham, "Automated Assessment of Secure Search Systems," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 22–30, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2723872.2723877

[49] J. Wang and X. Du, "A secure multi-dimensional partition based index in das." in *APWeb*, ser. Lecture Notes in Computer Science, Y. Zhang, G. Yu, E. Bertino, and G. Xu, Eds., vol. 4976. Springer, 2008, pp. 319–330. [Online]. Available: http://dblp.uni-trier.de/db/conf/apweb/apweb2008.html#WangD08

[50] L. Wang, P. Grubbs, J. Lu, V. Bindschaedler, D. Cash, and T. Ristenpart, "Side-channel attacks on shared search indexes." in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 673–692. [Online]. Available: http://dblp.uni-trier.de/db/conf/sp/sp2017.html#WangGLBCR17

[51] C. V. Wright and D. Pouliot, "Early detection and analysis of leakage abuse vulnerabilities," Cryptology ePrint Archive, Report 2017/1052, 2017, http://eprint.iacr.org/2017/1052.