Leakage-Abuse Attacks against Order-Revealing Encryption

Paul Grubbs*, Kevin Sekniqi†, Vincent Bindschaedler‡, Muhammad Naveed§, Thomas Ristenpart*
*Cornell Tech †Cornell University ‡UIUC §USC

***Abstract*—Order-preserving encryption and its generalization order-revealing encryption (OPE/ORE) allow sorting, performing range queries, and filtering data — all while only having access to ciphertexts. But OPE and ORE ciphertexts necessarily leak information about plaintexts, and what level of security they provide in practice has been unclear.**

In this work, we introduce new leakage-abuse attacks that recover plaintexts from OPE/ORE-encrypted databases. Underlying our new attacks is a framework in which we cast the adversary's challenge as a non-crossing bipartite matching problem. This allows easy tailoring of attacks to a specific scheme's leakage profile. In a case study of customer records, we show attacks that recover 99% of first names, 97% of last names, and 90% of birthdates held in a database, despite all values being encrypted with the OPE scheme most widely used in practice.

We also show the first attack against the recent frequency-hiding Kerschbaum scheme, to which no prior attacks have been demonstrated. Our attack recovers frequently occurring plaintexts most of the time.

I. Introduction

Due to frequent data breaches and broad interest in outsourcing data to the cloud, companies increasingly want to encrypt sensitive information before storing it in local databases or uploading to services operated by third parties. Standard encryption mechanisms would, however, reduce the value of these databases and services by preventing them from doing useful operations on the data. A seeming solution is to use so-called property-revealing encryption (PRE) schemes that allow limited operations over ciphertexts by making public specific properties of plaintexts. Systems based on PREs are already used in industry [3, 13, 36, 43] and increasingly studied in the academic literature [3,20,24, 28,38,39].

A widely desired property to preserve is order. This allows sorting, as well as range and prefix search queries, which are needed to support other server-side operations. Order-preserving encryption (OPE) [1, 5] ensures that $\mathcal{E}_k(m_1) < \mathcal{E}_k(m_2)$ for $m_1 < m_2$ and $\mathcal{E}_k$ the encryption algorithm. By allowing standard comparison operators directly on ciphertexts one can easily integrate OPE into systems, and many have done so [3, 13, 36, 38, 43]. A generalization of OPE is order-revealing encryption (ORE) [7] which reveals ordering relations by way of a public comparison function that operates on pairs of plaintexts.

What level of security OPE and ORE schemes achieve is poorly understood. Boldyreva et al. [5, 6] explicitly warn about the uncertain security of OPE. When encryption is deterministic, no security is achieved should an encrypted dataset include all possible plaintexts. (Simply sort the ciphertexts and their order reveals the plaintexts.) A recent study by Naveed, Kamara, and Wright (NKW) [34] empirically explores the security of deterministic OPE and ORE schemes when not all possible plaintexts appear in the target dataset. They exploit inference attacks: using some auxiliary information about the distribution of plaintexts, the adversary abuses the order and frequency of plaintexts preserved within a sequence of ciphertexts to recover as many plaintexts as possible. Their best performing attack, called the cumulative attack, worked well to recover data that lies within small domains, such as days of the year (365 possible values).

Intuitively, their attacks perform well when most possible plaintexts appear in a target dataset, seeming to limit their applicability when plaintexts are from large domains. The NKW cumulative attack also scales poorly, requiring time $\mathcal{O}(n^3)$, where n is the greater of the number of unique items in the ciphertexts or auxiliary data. All this leaves open the possibility that OPE and ORE schemes remain secure in practice for plaintext data drawn from larger domains, and practitioners could simply avoid using OPE for small-domain data. Indeed researchers have suggested exactly that approach as a hedge against known attacks [17].

Our contributions. We develop new attacks that target, in aggregate, all suggested OPE/ORE schemes, and show their efficacy against plaintexts drawn from larger domains. Our attacks come in two flavors: generic, like the NKW attack, and scheme-specific. Generic attacks work against any scheme that leaks frequency and order information. We also introduce the first scheme-specific attacks that exploit additional leakage about plaintexts. These latter attacks cover the OPE scheme most widely used in practice, due to Boldyreva, Chenette, Lee, and O'Neill (BCLO) [5]. Finally, we provide an attack against the recent frequency-hiding scheme of Kerschbaum [25].

We use as a running case study an encrypted customer record database, a frequent scenario for these encryption technologies in industry. We utilize a variety of public datasets as stand-ins for the kind of data targeted in an attack, including such customer attributes as first names, last names, ZIP codes, and birthdates. These attribute types come from domains much larger than those considered in the prior work; for example there are hundreds of thousands of unique first names, tens of thousands of birthdates (across

Scheme(s)	First names	Last names
Kerschbaum [25]	30%	7%
Popa et al. [37], Kerschbaum [26]	**84%**	38%
BCLO [5,6]	**99%**	**97%**
CLWW [12]	**98%**	**75%**
BCLO + CLWW [12]	**85%**	44%
Baseline Guessing	5%	1%

Figure 1: Average recovery rate of plaintexts (in percent of all target ciphertexts) across all first and last name datasets for our best attacks against the indicated OPE/ORE scheme(s). The last row indicates the recovery rate given by just guessing the most likely message for each ciphertext. We have bolded those attacks that fully recover more than half the plaintexts, on average.

a given century span), and tens of thousands of unique ZIP codes in the United States. We identify distinct public data sets that can be used as auxiliary information in inference attacks.

Generic attacks and non-crossing matching. We first revisit the NKW attack, which is generic in that it works against any scheme that reveals plaintext frequency and order. Its accuracy in our customer record case study is relatively poor. For example, it recovers on average 44% of encrypted first name records but this accounts for only 5% of the unique first names in the data set, meaning the attack only recovers a small fraction of the high-frequency elements of the distribution. The attack is also relatively slow, requiring six hours for the Fraternal Order of Police dataset on a well-provisioned cluster.

We explore their attack in more detail, and point out some limitations of their approach. Their most accurate attack, called the cumulative attack, casts the adversarial task as solving a linear sum assignment problem. This approach, however, typically yields solutions that violate adversarially known ordering constraints, outputting a guess that ciphertext c_1 encrypts plaintext p_2 and ciphertext c_2 encrypts plaintext p_1, yet $c_1 < c_2$ and $p_2 > p_1$.

We observe that the adversary's goal can be cast as a classic graph problem: a min-weight non-crossing bipartite matching. One set of nodes consists of ciphertexts and the other possible plaintexts, and the edges are weighted using frequency information. Non-crossing refers to the fact that we want the solution to abide by the adversarially known ordering constraints. Unlike the NKW attack, our non-crossing attack takes fuller advantage of frequency and order information. We use a classic algorithm [32] to solve the min-weight non-crossing matching problem, and its runtime is both asymptotically and concretely better than the NKW approach.

We show that our non-crossing attack achieves significantly better accuracy, for example it recovers on average 84% of first names in our target encrypted datasets, about a 2x improvement over NKW. High-frequency plaintexts are particularly at risk: our attack recovers on average 95% of the 20 most frequent first name plaintexts, and 88% of the 20 most frequent last names. The non-crossing attack runs in only a few hours, even for the largest target dataset, where the induced matching problem has over 17 billion edges.

Exploiting further leakage. The attacks so far are generic, and do not take advantage of the additional leakage exhibited by the OPE schemes used in practice. The most widely used scheme is due to Boldyreva, Chenette, Lee, and O'Neill (BCLO) [5], because it is fast, easy to deploy, and has ciphertexts that are only a few bits longer than plaintexts. It's been proven secure in the sense of indistinguishability from a random order-preserving function, but this provides only modest guarantees: prior work has shown that for a uniformly chosen plaintext, the corresponding ciphertext leaks almost the entire first half of the plaintext [6]. Despite widespread deployment of BCLO, the implications of this leakage for real datasets has not been studied.

We rectify this, showing that in practical scenarios using the BCLO scheme to encrypt a set of first names, for example, allows an attacker to recover almost half the data set. The leakage is even worse for last names, with almost 97% of last names trivially recoverable. A key issue making this leakage so damaging is that varying-length plaintexts must be padded to the maximum length plaintext, ensuring that shorter messages completely reside in the leaked first half of the padded plaintext. We emphasize that exploiting this leakage does not require mounting an inference attack, rather an adversary simply can inspect ciphertexts, perform a few elementary calculations, and produce (most) plaintexts.

That said, our graph-based viewpoint on inference attacks allows us to easily combine this leakage with inference to improve recovery rates further. We adapt our non-crossing attack to first compute the leakage implied by [6], use it to exclude matchings that cannot occur, and then solve the resulting, narrower non-crossing matching problem. The resulting attack allows us to recover on average 99% of first names, more than doubling the recovery rate over using the BCLO leakage alone without inference.

We also apply our approach to exploit the leakage of the more recent Chenette, Lewi, Weis, and Wu (CLWW) [12] ORE scheme. The leakage of CLWW is different than that of BCLO, and they argue that their scheme may be more secure: CLWW prove that for uniform randomly distributed messages, their scheme's leakage is asymptotically less than the BCLO leakage. They also propose that security may be further improved by composing an OPE scheme with their ORE scheme. We apply our framework to both the CLWW scheme and the composition of BCLO with CLWW, and show that, unfortunately, CLWW provides an equivalently poor level of security as compared to BCLO in the settings we consider. Interestingly, some plaintext distributions are particularly bad for CLWW leakage: for example, our attack against BCLO does not perform well on ZIP codes (achieving 12% recovery rate), but on CLWW our attack recovers 97% of encrypted ZIP codes. Composition of the two schemes does decrease attack accuracy, but is still far from providing acceptable security.

Exploiting known plaintexts. Our attacks, as well as the NKW attack, assumes a relatively weak adversary that only obtains ciphertexts, but has no information about any specific plaintext values. As we discuss in detail in the body, in real-world deployments many systems relying on OPE/ORE offer attackers the ability to mount known- or chosen-plaintext attacks. We show how to extend our attack framework to take advantage of known plaintexts by simply partitioning the attack problem based on the known values and running an attack against each resulting sub-problem independently. As the BCLO and CLWW attacks already recover most plaintexts, we see how well this partitioning approach would help our generic non-crossing attack. Knowing some small percentage of plaintexts provides a modest improvement for first names and last names, but a huge boost for birthdates and ZIP codes.

Attacking frequency-hiding schemes. The attacks mentioned above work against schemes that at least leak both frequency and order, but Kerschbaum [25] recently introduced a scheme that hides frequency information. Here there is no prior work, as the NKW attacks do not apply to frequency-hiding schemes, and our non-crossing attack framework also does not apply. We propose a new "binomial" attack that performs reasonably well, recovering on average 30% of first names and 7% of last names. Notably, it recovers the majority of high-frequency plaintexts (despite not having frequency information leaked), suggesting these plaintexts are particularly poorly protected by any order-revealing scheme.

Newer ORE schemes. The ORE schemes we consider have been defined as having a *public*, *unkeyed*, *noninteractive* procedure the server can use to reveal the order relationship between the underlying plaintexts of two ciphertexts. Recently, a new line of work [16,30,41] explore schemes at a different point in the design space. These schemes are more similar to searchable symmetric encryption (SSE) in that a user must generate a query-specific trapdoor, or complete multiple rounds of interaction, to perform a range query on ciphertexts. Unfortunately there are currently major impediments barring deployment of these schemes in practice (see Section IX). Our results may encourage systems designers to consider them in greater earnest.

Summary. We are the first to explore the security of OPE and ORE schemes when used with plaintext data types that were, before our work, not known to admit attacks. What's more, our case study of customer data is representative of common industry practice. Underlying our new results is a framework for constructing attacks based on min-weight non-crossing bipartite matching, which allow for easy extensibility in the face of leakage beyond frequency and order. Most importantly for current industry practice, our results show, for the first time, how the leakage of the BCLO scheme would enable recovery of essentially all plaintexts encrypted in typical customer record databases. Suggested practical alternatives such as the CLWW scheme, or the composition of it with BCLO, do not fare much better. See Figure 1 for a high level summary of our quantitative results for first and last names.

Our results offer guidance to practitioners about the security level offered by OPE and ORE schemes. While obviously using property-revealing encryption is better than leaving data in the clear (in some settings the only viable alternative currently), our work indicates that the security benefits of deployed schemes is quite marginal.

In terms of countermeasures, an obvious suggestion is to move towards less leaky schemes, such as those that only reveal order, including Kerschbaum's scheme and the more recent Boneh et al. scheme based on multilinear maps [7, 29]. Unfortunately in most settings there exists inherent challenges to deployment of these schemes. Kerschbaum's scheme is relatively efficient, but requires client-side state which impedes scaling. The Boneh et al. scheme has ciphertexts larger by 10 orders of magnitude than BCLO ciphertexts and requires tens of minutes to compute encryptions. Even so, our attack against such frequency hiding schemes shows that for common use cases the high frequency plaintexts may nevertheless be revealed to attackers.

II. Preliminaries

Basic notation. Let $\mathcal{D}$ be a set and let its size be denoted by $s = |\mathcal{D}|$. We assume some total ordering of elements in $\mathcal{D}$. Let D be a sequence of elements $(d_1, \ldots, d_s)$, where each $d_i \in \mathcal{D}$. For $1 \leq i \leq s$, we define the histogram function for D to be the function $\mathsf{H}_D(i)$ that outputs the number of times the i^{th} element of $\mathcal{D}$ appears in D, divided by s. For $1 \leq j \leq s$, we define the cumulative distribution function (CDF) for D to be the function $\mathsf{F}_D(j) = (\sum_{i=1}^{j} h_i)/s$, where h_i is the number of occurrences of the ith element of D. Observe that one can represent histograms and CDFs linearly in the number of unique elements in D. Below, we will use both "birth date" and "birthdate" to refer to the month, day, and year of an individual's birth.

Order-preserving and revealing encryption. An order-preserving encryption (OPE) [1,5] E allows encrypting data under a secret key k such that $\mathcal{E}_k(m_1) < \mathcal{E}_k(m_2)$ for any $m_1 < m_2$. We focus primarily on deterministic schemes, meaning $\mathcal{E}_k$ defines a function. The benefit of OPE is that existing comparison operations will work transparently on ciphertexts, thus providing a drop-in way to replace plaintexts with ciphertexts while preserving the ability to perform order comparisons, range queries, etc. Order-revealing encryption

(ORE) [7] does not allow an existing comparison operator to work on ciphertexts, but instead comes with a public procedure that can determine the order of two plaintexts given only their two ciphertexts. Again, we will focus primarily on ORE schemes that are deterministic.

PRE-based systems. OPE and ORE are examples of what we refer to as property-revealing encryption (PRE) schemes. PRE schemes encrypt data while allowing limited computation over the revealed plaintext properties. A motivating scenario for PRE schemes is in client-server systems where one wants to perform encryption on the client side but take advantage of the server performing some operations on the client's behalf. In a PRE-based encrypted database, for example, OPE is often used to enable certain kinds of expressive SQL-like queries on encrypted data. The standard database functionality in such a system is extended and modified by a database proxy, which rewrites queries and performs encryption/decryption on behalf of database clients. The clients are often application servers (acting on behalf of human users) which address the database proxy as though it were the database. Example PRE-based databases include IQrypt [22], CryptDB [38,40], and Cipherbase [3].

Another concrete application arises with security-conscious businesses that use network middleware to encrypt and decrypt data as users interact with cloud software-as-a-service (SaaS) applications. Such "encryption proxies" are similar to the database proxy described above, except in most cases it cannot change the way the SaaS application works. PRE schemes including OPE and ORE become useful to ensure stored data does not break (some) useful cloud functionality. Commercial products from Skyhigh Networks [43], CipherCloud [13], and Perspecsys [36] are examples.

In all these deployment settings, a minimal security requirement is the confidentiality of plaintext data in the face of attackers that obtain access to a PRE-encrypted database, e.g., by compromising a server or obtaining insider access to it.

Types of ORE/OPE schemes. OPE and ORE schemes come in several flavors that affect their deployability in the above application scenarios. Unlike the OPE/ORE schemes mentioned above, some schemes require state beyond the secret key to be stored with the client. This state can potentially be held by proxies on behalf of clients, but regardless complicates scaling to large numbers of clients. These schemes are mutable: ciphertexts might change as more values are added to a database. They are also interactive, requiring multiple rounds of communication between the client and server to store or retrieve ciphertexts. Examples of such schemes include [25,26,37].

Statefulness, mutability, and interactivity all hinder deployability. The reason to consider such schemes is that the more deployable stateless schemes (e.g., [5]), which are the only ones currently used in practice, leak more information about plaintexts than just ordering and frequency information. We will only consider these more advanced schemes in Section IV and Section VIII.

III. Overview and Methodology

To experimentally evaluate OPE and ORE security, we fix a methodology in which we empirically evaluate security using public datasets as stand-ins for sensitive plaintexts. Prior work by NKW focused on medical settings, where a database of patient data was outsourced in encrypted form. The attributes (columns) of the databases they considered had plaintext values falling within small domains, the largest being the days of the year (365 possible values). We want to explore security for larger domains where it is not known if effective attacks can be mounted.

Customer records as case study. We therefore fix a running case study of an outsourced database of customer information. In industry currently, OPE is used to encrypt customer records before uploading to cloud services such as Salesforce. A client (sometimes an encryption proxy acting on its behalf) takes as input plaintext records, encrypts their attributes independently, and uploads them to a cloud service using (often) an existing API. While customer data can take on a number of forms, we focus particularly on a subset: (1) first name, (2) last name, (3) US ZIP code, and (4) date of birth. All such attributes benefit from server-side processing that takes advantage of the ability to compare plaintext ordering, and OPE in industry is used to encrypt such data for exactly this reason.

We restrict our attention to male first names, because research on inference attacks has shown that the binary "gender" attribute cannot be hidden by any PRE scheme [34]. Partitioning database records based on gender would be a trivial preprocessing step to remove some uncertainty about the underlying plaintexts of encrypted values. To model real adversaries as closely as possible, we perform this preprocessing as well.

In these contexts, revealing plaintext order to servers enables a variety of server-side operations. Sorting is the most obvious, and for names this allows sorting alphabetically. It also allows range queries, such as finding all names starting with "A", "B", or "C". Perhaps more subtly, OPE enables prefix searches over ciphertexts, so one could search for "Dav*" and retrieve Dave, David, Davik, etc. Date of birth and ZIP codes also benefit from such range and prefix searches because prefixes have structural meaning. For example, querying "606*" gives all ZIP codes associated with the city of Chicago. Together, all this allows preserving functionality on the server-side, in some cases allowing drop-in replacement of plaintext data with encrypted without modifying server implementations at all.

Threat model. We investigate the security of OPE/ORE encrypted attributes when an adversary obtains a one-time snapshot of the encrypted database. Our attacks will target each attribute (column) independently (i.e., we won't use information about the encrypted last name column to help improve recovering first names). Exploring the benefit of such cross-column attacks in this setting remains an interesting topic for future work. So our attacks will consider an adversary that obtains a sequence $C = (c_1, \ldots, c_n)$ of encryptions of one column of data. Here $c_i = \mathcal{E}_k(m_i)$ for some unknown plaintext m_i and encryption scheme $\mathcal{E}$. The key k remains unknown to the attacker, and we assume it is intractable to recover it. The passive adversary can take advantage of *auxiliary information* about the distribution from which plaintexts are drawn. This is the same attack setting as considered by NKW.

We also initiate investigation of stronger adversarial models. In Section VII we consider non-adaptive known-plaintext attacks. In these, a subset of the n plaintexts are known by the adversary. Despite not being considered in prior attacks on OPE/ORE, we believe known-plaintext attacks are likely to be prevalent risks in practice. As we discuss in Section VII, for example, attackers in many contexts can obtain (perhaps indirect) access to an encryption oracle before compromising the encrypted database. We will not examine chosen-plaintext attacks in great detail, though our known-plaintext attacks clearly work in this setting as well.

In all attacks the adversary finishes by outputting a list of n ciphertext, plaintext pairs indicating the attacker's guesses. We measure success in several ways. The raw recovery rate is the fraction of n records that are mapped to the correct plaintext. For deterministic encryption schemes, this means that if we can correctly infer the mapping between the ciphertext for "Michael" and the correct plaintext, and this ciphertext accounts for (say) 4% of the database, then our raw recovery rate will be at least 4%. We therefore also report on unique recovery rate, which is the fraction of unique plaintext values recovered correctly. Finally, neither of these measures indicate how much partial information may arise, since the attacker gets no credit for mapping an encryption of the birthdate "19620105" to "19620106" despite the fact that this leaks a lot of information. We report on prefix recovery rate as the average, over the n data items, of the length of the prefix that matches between the correct plaintext and the one output by the adversary. Note that prefix accuracy for birthdates has an important caveat: all our target birth dates occurred in the twentieth century, so the baseline prefix accuracy is 25% (because the first two characters are always "19").

Data sets. We use a number of datasets to drive our simulated attacks. We have two kinds of datasets, target datasets and, when used, auxiliary data sets. The latter is given to the adversary in the clear to provide it an empirical estimate of the target dataset's distribution.

Dataset	# 1st names	# Last Names	Total Records
FOP (FOP)	3,862	116,677	621,662
California Muni (CALC)	3,777	59,935	255,956
Washington (WA)	3,525	67,206	228,934
Texas Compt. (TXCOM)	2,416	33,802	149,678
Florida (FL)	2,091	32,986	112,566
Maryland (MD)	2,551	36,698	111,183
Connecticut (CT)	2,016	30,623	77,613
New Jersey (NJ)	1,964	29,094	73,119
Iowa (IA)	1,734	22,616	60,035
Ohio (OH)	1,440	21,034	58,792
Texas A&M U. (TXAMU)	1,466	11,437	25,192
North Carolina (NCAR)	696	3,688	6,976
Illinois (IL)	243	1,021	1,259

Figure 2: Unique and overall record counts for our target datasets that include first and last names.

For target first and last name data sets, we used a mixture of municipal, state, and public university government employees, publicly published by the government. We additionally use the database dump from the Fraternal Order of Police (FOP) breach.[1] This includes 623,372 records on police officers for which all rows contain both first and last names, 237,392 rows contain birthdates, and 617,280 contain ZIP codes. There are 22,485 unique birthdates and 26,914 unique ZIP codes. A summary of the datasets is given in Figure 2.

The distributions observed in these datasets are non-uniform. The most common first name appeared in 4.06% of records on average across all the datasets (with variance 0.29% across data sets). The most common last name appeared in 0.8% of records on average across all datasets (with variance of 0.04%). This translates to an empirical min-entropy of 4.63 bits for first names and 7 bits for last names, on average. For the FOP data set, the most common ZIP code appeared in 0.09% of the records (7.6 bits), and the most common birth date in 0.01% of records (11 bits). For reference, password leaks often indicate min-entropies of about 6–7 bits [8]. The maximum accuracy of the baseline guessing attack, in which the most frequent element of the auxiliary data is matched to every ciphertext, was 5.0% for first names from the Connecticut dataset and 1.2% for last names from the North Carolina dataset. Below we will consider first and last name datasets as subsets of the set of all alphabetical strings less than or equal to some fixed length. The effect of this is that our first and last name datasets are quite sparse.

Most of our attack simulations require auxiliary data. We restrict ourselves to publicly available data that would be easy for any attacker to obtain, and do not consider scenarios where an attacker obtains, say, an earlier version of the same database. Our auxiliary data for experiments on first names is statistics for baby names gathered by the US Social Security Administration [46]. For our experiments, we used a year-by-

[1] For privacy reasons we will not include links to these datasets, but they are available from the authors by request.

year tally of the most popular American male first names for the years 1945 to 1993. For experiments on last names, we used statistics gathered by the US Census Bureau during the 2000 census on the exact frequency of last names for every person who filled out the census that year [45]. The census data included 5,023 and 151,672 distinct first and last names, respectively.

Our auxiliary data for birth dates came from the American Community Survey (ACS) 2013, a yearly survey conducted by the US Census Bureau. First, since our target dataset is a database of law enforcement union members, we used the 2013 American Community Survey (ACS) to compute a histogram for the *ages* of all respondents who marked "law enforcement" as their current line of work. Using known statistics on birth month frequency we synthesized an accurate distribution for birth dates with per-day granularity for all days between 1920 and 1999.

Our auxiliary data for ZIP codes is a list of the reported population of each assigned ZIP code according to the 2010 census. The frequency of a ZIP code in the auxiliary data was computed as the proportion of the total US population living there. Intuitively, it seems like the distribution of ZIP codes in a nationwide database like the FOP dump will be similar to the distribution of people into ZIP codes because more populous areas will likely have more police officers, and therefore more FOP members. However, we note that the two distributions are not particularly well-correlated and that our attacks below would probably be more effective with better auxiliary data.

For large, sparse domains like the ones considered here, it is possible for a target datum to be "un-recoverable" according to our accuracy metrics for attacks with auxiliary data, meaning its underlying plaintext does not exist in the auxiliary data. For birth dates and ZIP codes this is not the case, but it is for first and last names. On average for our datasets, above 99% of first name records are recoverable but only 91% of unique first names are. For last names, only 89% of records and 71% of unique values are recoverable on average. Below, we will not correct our results to account for this artificial cap on attack accuracy, since a real adversary would likely face this same problem. This is in line with our very conservative approach to auxiliary data overall. However, in Section V we will evaluate an attack that does not require auxiliary data, so this limitation does not apply there.

IV. The Non-Crossing Attack

In this section we recall the NKW cumulative attack which can work against OPE and ORE schemes that leak both frequency and order. This includes all deterministic OPE schemes, including [5, 12, 26, 31, 37].

We discuss some limitations both in terms of efficiency and accuracy of their attack, and suggest a new attack that performs significantly better. We call this the non-crossing attack. It works for any scheme for which the original NKW cumulative attack works, and it also will be what we build off in later sections when we take advantage of more leakage and stronger adversarial models.

Attack setting. In this section, we follow NKW and consider known-ciphertext attacks with auxiliary information, but no knowledge of any plaintexts. More precisely, an attacker obtains a sequence of ciphertexts $C = (c_1, \ldots, c_n)$ for $c_i = \mathcal{E}_K(m_i)$. Plaintexts may repeat and are drawn according to some (typically unknown) distribution p_m over a message space $\mathcal{M}$. The attacker additionally has *auxiliary information* about p_m, which in practice is simply a sequence $Z = (z_1, \ldots, z_\psi)$ of plaintexts that is believed to be drawn from $\mathcal{M}$ using a distribution close to p_m. For our datasets, it's always the case that $n < \psi$. The attacker outputs a guess of each ciphertexts' plaintexts, and we measure success as per the three recovery rate types defined in the previous section.

The NKW attacks. NKW describes two attacks against schemes which leak order and frequency: the sorting attack and the cumulative attack. The sorting attack simply sorts C and Z and matches the i^{th} largest element of C to the i^{th} largest element of Z. When the target data is sparse, or when $|C| \neq |Z|$ (as is the case for our datasets), this attack performs poorly, so we will not consider it further.

In the cumulative attack, NKW models the inference task as a linear sum assignment problem [9]. We find it conceptually simpler to cast the problem NKW solves in the language of graph algorithms. Let $G = (U, V, E)$ be a bipartite graph where every vertex in U corresponds to a distinct ciphertext and every vertex V corresponds to a plaintext value in the auxiliary data. Thus $n = |U|$ and $\psi = |V|$. For $u \in U$ and $v \in V$, the edge $(u, v) \in E$ is labeled according to a cost (or weight) of mapping ciphertext u to auxiliary datum v. A matching for G is a subset of edges for which no two share a common vertex. For our context, any matching is a potential decryption of some or all of the target ciphertexts. Finding a minimum cost (equivalently, maximum weight) matching should help an attacker obtain a good solution.

NKW explore the following approach to labeling edges. Given a ciphertext sequence C and a side information sequence Z, their attack lets edge $(i, j) \in U \times V$ have label given by

$$w(i,j) = |\mathsf{H}_C(i) - \mathsf{H}_Z(j)|^2 + |\mathsf{F}_C(i) - \mathsf{F}_Z(j)|^2 \qquad (1)$$

where H and F are as in Section II. The attacker seeks a minimal-cost bipartite matching of the graph G, one that contains an edge for each node in U. To do so they use the Hungarian algorithm, padding if necessary U with dummy nodes until $|U| = |V|$. The algorithm runs in time $\mathcal{O}(\psi^3)$.

NKW also briefly discuss labeling edges with only the square distance of CDFs, but they conclude that using CDF and frequency as in Equation 1 performs substantially better. We therefore only use the latter in our experiments.

Limitations of the cumulative attack. NKW show that the cumulative attack can effectively recover plaintexts, albeit only ones from relatively small domains. The largest message space they consider is 365 elements (days of the year). This has left open the question of whether plaintexts drawn from larger domains might still be secure when encrypted under OPE. Unfortunately the performance of the attack is prohibitive for large datasets such as the last name, birthdate, and ZIP code datasets we consider in this paper.

We observe that there exists a classic greedy heuristic that works to find min-cost bipartite matchings efficiently [4,9]. It just takes the minimum cost for each ciphertext individually, and gives a 2-approximate solution in time $\mathcal{O}(n\psi)$. In experiments we have observed that this heuristic often produces fairly accurate approximations. For first names, the average gap between the cumulative attack's recovery rate and the greedy heuristic's recovery rate is 14%. Below we will refer to the attack that finds the matching with the greedy heuristic as "NKW greedy".

The NKW attack, when using either the exact Hungarian algorithm or greedy approximation to find the matching, does not necessarily enforce adversarially-known ordering constraints on plaintexts. The algorithm can obtain solutions from the Hungarian algorithm that include edges that "cross", i.e. ciphertext c is mapped to p' and ciphertext c' is mapped to p but $c < c'$ and $p' > p$. In fact, the NKW attack violated ordering constraints in every one of our experiments on real datasets. One can avoid such crossings by labeling edges with just the square distance of CDFs, but as mentioned this performs poorly. We therefore seek a way to avoid violating ordering constraints, while still taking advantage of the available frequency information.

The non-crossing attack. We introduce what we refer to as the non-crossing attack. An attacker can avoid crossings in their solution by solving a max-weight non-crossing bipartite matching problem on the graph G. Since ordering constraints are encoded into the matching algorithm, we will not include any ordering information in our edge labels. Rather, we will use the L1 distance of frequencies. For ciphertext i and auxiliary datum j, the labeling function is

$$w(i,j) = \alpha - |\mathsf{H}_C(i) - \mathsf{H}_Z(j)|$$

where α is a fixed constant parameter that converts a min-cost problem into a max-weight problem. In our context, any choice of $\alpha > 1$ will not change which matchings are maximum-weight, so the solution to the inference problem will be the same for any $\alpha > 1$. Different choices of α will change what that maximum weight is, but for our purposes this is inconsequential.

This approach also significantly improves computational performance over NKW: there is a well-known dynamic-programming approach that runs in time $\mathcal{O}(n\psi)$ to find the optimal non-crossing matching [32].

Results. We compare the non-crossing and cumulative attacks for our customer record datasets detailed in Section III. We could not scale the NKW's use of the Hungarian algorithm to complete in reasonable time for last name, birthdate, or ZIP code datasets. With the largest last name dataset, we estimated the NKW attack would have taken roughly one hundred days to complete a single experiment on the well-provisioned compute cluster we used for the first name experiments. The Hungarian algorithm is highly nontrivial to parallelize, compounding its scalability issues. For last name, birthdate, and ZIP codes, we will compare our non-crossing attack to the NKW greedy attack. For first names, we will compare the non-crossing attack with the exact Hungarian algorithm.

Figure 3 compares the success of the two attacks for first names (left bars) and last names (right bars), showing raw recovery rates for each dataset. The non-crossing attack always performs strictly better than NKW's cumulative attack, and some times substantially so, nearly doubling the average recovery rate (44% vs. 83%) for first names and septupling (5% vs. 38%) the average recovery rate for last names. When taken as percentage of the "recoverable" names (those plaintexts that also appear in the auxiliary data), on average we recover 84% of first names and 42% of last names.

The unusually low performance on the Illinois dataset is due to its small size — it is only about one-third of the size of the next smallest dataset. Its small size is problematic because it impacts the statistical quality of the sample. For example, the most frequent first name only occurs 34 times and the most frequent last name only occurs 10 times.

Figure 4 shows the average unique recovery rate for high-frequency first and last names. It is cumulative, so the x axis labels refer to the recovery rate for that number of top names. For example, the point $(40, 95)$ means the attack recovered 38 of the top 40 most frequent names, on average. Our attack is especially accurate for these values. From this, we can conclude that for almost all datasets no real security guarantee can be made for high-frequency names. We do not include a separate graph for unique recovery rates for first and last names, but the overall trend is very similar.

Inference on birthdates and ZIP codes is much less accurate for both attacks: exact recovery rates were less than 2% across the board. This occurred for two reasons. First, the auxiliary information available for attackers is not as accurate a reflection of the target data distribution, compared to the names data. For example, our ZIP code data is not really a sample from the same distribution as our auxiliary data — the ZIP code frequencies in our auxiliary data are proportional to the number of people living there, but the frequencies in our target dataset are proportional to FOP membership, which varies state-to-state. Second, the distributions themselves are

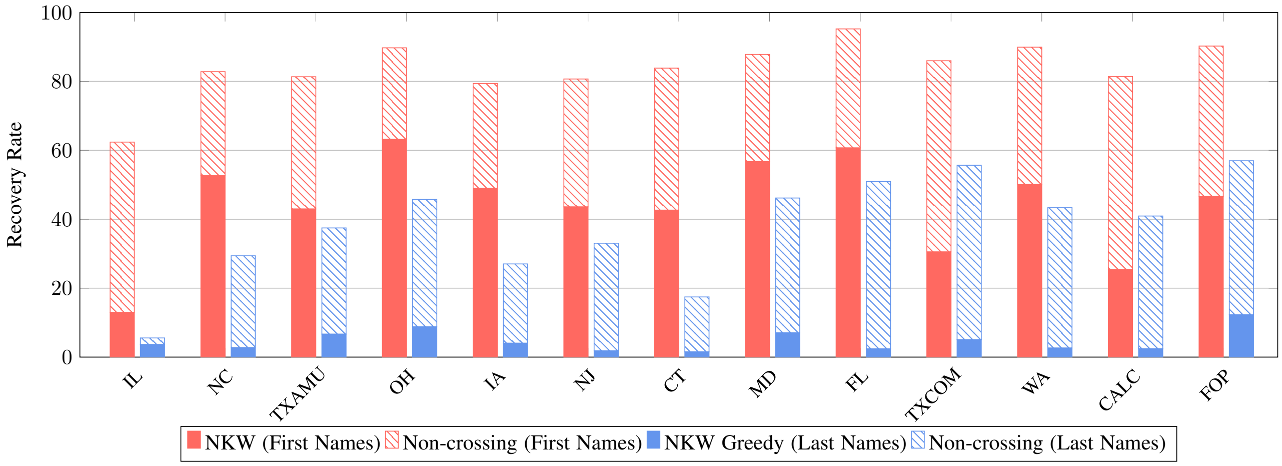

Figure 3: Comparison of raw recovery rates of NKW and the non-crossing attack on first and last names. The non-crossing attack outperforms NKW for all target datasets.

flatter and do not have frequency "peaks" which can be easily recovered by inference attacks.

We do note that partial information is often leaked. The non-crossing attack's prefix recovery rate was 34% for birthdates and 23% for ZIP codes. The corresponding prefix recovery rates for the NKW attack are similar. The non-crossing attack outperforms the baseline by about ten percent in overall prefix recovery for birth dates: we recovered the decade of birth for 75% of the unique birth dates in the database (16,847 out of 22,485). The non-crossing attack also recovered, on average, the first digit of a target ZIP code, giving the adversary the region of residence of the record (e.g. eastern seaboard, midwest, southwest). In all cases, the NKW with greedy heuristic attack performs worse than non-crossing.

In terms of runtime performance, both the NKW with greedy heuristic and the non-crossing attack are reasonably fast. They require at most ten and twelve hours, respectively, to run to completion on the FOP last name dataset, the largest of any we examined. It includes $n \approx 116,000$ unique ciphertexts and $\psi \approx 151,000$ unique values in the auxiliary data. The induced bipartite graph has around 17.4 billion edges. The non-crossing attack is slower than the greedy heuristic due to the need for a backtracking step (similar to the one used in the classical edit distance algorithm) to recover the edges of the max-weight matching. It also writes to two large data structures at each iteration of the inner loop, which can be slow if the memory layout is not tuned. We optimized our implementation by exploiting the fact that the algorithm only needs to examine two rows of the dynamic programming table at a time: the one currently being written to, and the previous one. We reuse two fixed-width buffers for these rows rather than allocating an entire $n \times \psi$ matrix in memory. These two buffers are small enough to fit in the CPU's cache, so the number of slow operations on DRAM is effectively halved. It also reduces the overall memory footprint by about 90%.

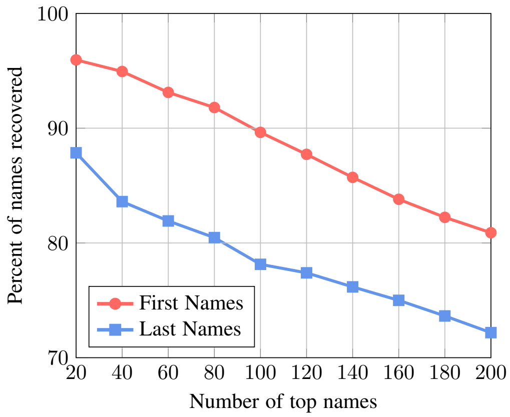

Figure 4: Average unique recovery rates for high-frequency first and last names for non-crossing attack. The red line with circles is first names. The blue line with squares is last names. Note that the y-axis starts from 70.

V. Attacking the BCLO Scheme

The attacks discussed in the previous section only take advantage of frequency and order information. While there exist schemes that leak only this information (or even less), they are not deployed in any real-world systems because all such schemes require either inefficient multilinear maps or client state, mutable ciphertexts (their value changes over time), and multiple rounds of communication to insert a ciphertext or perform a search. Instead, practitioners have widely been deploying more efficient schemes such as the classic one due to Boldyreva, Chenette, Lee, and O'Neill (BCLO) [5]. This scheme is known to leak more than just frequency and order [6], and we now show how to build highly damaging attacks that augment our non-crossing attack to exploit this additional leakage.

The BCLO scheme. The BCLO scheme realizes an OPE scheme by a recursive procedure that samples according to the hypergeometric distribution based on coins pseudorandomly derived from a secret key. The details of the construction are not important to our attacks, and so for brevity we refer readers to [5] for details. It is secure in the sense of being indistinguishable from a random order-preserving function (ROPF).

An ROPF, however, still may leak significant information about the plaintext. Boldyreva, Chenette, and O'Neill [6] analyzed ROPFs relative to a notion of security they call window one-wayness. Let $M = |\mathcal{M}|$ be the size of the domain. They show that an adversary, given the encryption of a uniformly chosen plaintext, can use the ciphertext to immediately infer with high probability that the hidden plaintext falls within a set of size $b \cdot \sqrt{M}$ for some small constant b. When b is small relative to $\sqrt{M}$, this means the attacker learns most of the first half of the plaintext.

We will explore how this leakage affects security for in our running case study, and then show how to build even more damaging inference attacks by augmenting our non-crossing attack to take advantage of the additional leakage. First, however, we discuss one security-critical issue that arises in practical use of the BCLO scheme.

The problem of padding. In previous work on OPE and ORE, the question of variable-length inputs is rarely discussed. Real systems that use OPE for variable-length plaintexts, though, must pad to preserve the functionality of being able to compare strings of different length.

For encryption schemes that reveal or preserve order, variable-length inputs represent a trickier problem than for other types of encryption, because different ways of ordering strings of characters handle variable lengths differently. For example, take the two strings "banana" and "zoo". Viewed as English words, it is clear that "banana" is lexicographically less than "zoo". However, some OPE algorithms (in particular [5]) only accept inputs that are integers in some set. Thus, the question becomes how to convert strings in a specified alphabet to integers *but preserve their alphabetical order*. The naive way to do this for "banana" and "zoo" is to treat them as big-endian numbers base 26, and convert to base 2 (or 10) before encrypting with OPE. It is not hard to see that this approach fails, because the base-2 number represented in base-26 as "banana" is *larger* than the corresponding base-2 number for "zoo". Encrypted with OPE, their relative ordering would be reversed.

The solution is to right-pad all strings to a common input length (i.e., the length of the longest possible input) with the lexicographically smallest character (in our case, a space) before converting the string to an integer for encryption with OPE. This approach was suggested in a technical report by Kolesnikov and Shifka [27], in the context of randomizing OPE ciphertexts while preserving sort order. This is painful when it comes to security of schemes like BCLO that leak high order bits: the padding pushes sensitive data into those bits that are leaked. Figure 5 gives histograms of the lengths of first names and last names across our datasets. As can be seen, a bit more than half of first names and the vast majority of last names fall below half the maximal length. This means that many or, for last names, most plaintexts will be immediately leaked when encrypted under BCLO.

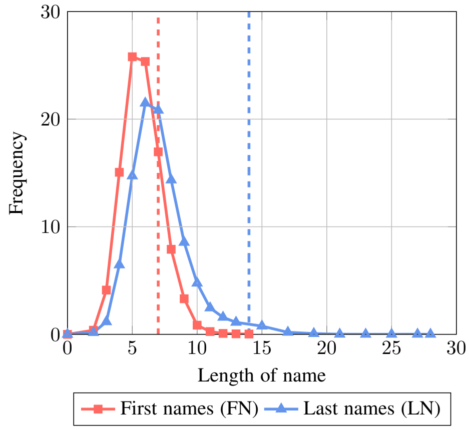

Figure 5: Length distributions of first and last names. The right vertical line is half of the longest last name length, which roughly means every last name to the left of that line will be leaked fully. The left vertical line is the same boundary for first names.

In theory, one could pad to a shorter length and truncate values that are too long. Besides the obvious loss of strict order preservation for all plaintexts, this also removes the ability to decrypt all ciphertexts to the correct plaintext. In practice this truncation would necessitate appending another encryption of the plaintext to the end of the OPE ciphertext so that decryption could occur correctly, which would at least double the storage required for that column in the database. Thus, this method of handling variable-length inputs is at best much more expensive than the alternative, and at worst impossible because of, e.g., unchangeable constraints in SaaS applications.

In our experiments, we encoded first and last names as base-27 integers and padded plaintexts to the length of the longest possible plaintext with the space character, which we defined to be the lexicographically smallest character of our input alphabet. This method of encoding strings as integers may seem strange initially, but observe that the naive way of treating each byte as a base-256 digit will cause the parameters (and, by extension, the ciphertexts) to be much larger than they need to be, since only a small subset of the 256 possibilities are valid characters. As mentioned before, the longest first name appearing in our data sets is 14 characters, and the longest last name is 28 characters.

Birth dates and ZIP codes are fixed-width, so padding is not required.

BCLO's additional leakage. Since the BCLO scheme leaks additional information other than order and frequency, it is logical to ask how much information an adversary learns just by computing the BCLO leakage. This is not an "inference attack" as we have defined it above, because it does not use auxiliary data. It does require knowledge of the input alphabet and padding rules. Below, we will describe how to combine this attack with inference.

We first describe how to compute the leakage given a ciphertext as per [6]. Let $\mathcal{M}$ be the message space with size $M = |\mathcal{M}|$ and let the ciphertext space be $\mathcal{C} = \{1, \ldots, C\}$. Typically in implementations of BCLO one uses message and ciphertext spaces that are of size a power of two, and sets $\log C = 3 + \log M$ as suggested in [6]. Then let

$$m_c = \left\lceil \frac{M \cdot c}{C+1} \right\rceil \qquad \text{and} \qquad \delta = \frac{b}{2\sqrt{M-1}} \ ,$$

where b is a parameter. Roughly speaking, m_c is the ciphertext-specific point around which a window of size up to 2δ can be drawn that contains the plaintext with high probability. The larger the parameter b, the larger the window. Of course we must truncate the window by the endpoints of the message space, meaning that m will be with high probability contained in the range

$$R_c = \left[\max\{1, m_c - \lfloor \delta M \rfloor\} \ , \ \min\{M, m_c + \lfloor \delta M \rfloor\}\right]$$

The analysis of Boldyreva et al. shows that for $b = 3$ the probability of landing in this window is negligibly far from one for uniformly sampled plaintexts [6]. For non-uniform message distributions, it is likely that more careful analysis could shrink the window and obtain more leakage, but we will be conservative and use the larger window for our first and last name attacks. We will use $b = 1$ for our attacks on ZIP codes and birth dates. Making the window smaller slightly increases the leakage and helps reduce the overlap in leaked ranges for nearby values, though it increases the likelihood that the true plaintext falls outside the range. Our birthdate and ZIP code datasets contained a substantial fraction of all the possible plaintexts[2], so there is a great deal of overlap in the leakage which, intuitively, gives the non-crossing attack more chances to match a value incorrectly. Having a smaller window mitigates this.

Abusing just the BCLO leakage. An adversary can attempt to immediately recover the plaintext for each ciphertext in a database using just the leaked range R_c and knowledge of the input alphabet and padding rules. This approach does *not* require any auxiliary data, and requires just a few elementary computations for each ciphertext. In fact, a human attacker can easily just read off partial or even full plaintexts from the m_c values trivially computable from a ciphertext. Some examples are shown in Figure 6.

Plaintext	**Ciphertext**	m_c
michael	cyrzjipnouushzh	michaekypfbkfr
david	aenpse cevvpkmr	david jwbvhec
robert	emlqrnycvblqqnd	robert lwyeorr
john	ccnnczzzpruvjhd	johmzzzysfbunn
james	bzkxrq gzortby	james zyovtq
daniel	aelfspocabjdvjc	daniel jgaginu
richard	ekrzjmjhjxykbba	richardkmfnwwx
jose	ccqrlzzziozokby	josdzzzxvfruqg
mark	cwmlfzzzjxhlklh	marjzzzxzqyduv
christopher	zokwwbrbibyouo	christotnqfolw

Figure 6: The value m_c computed for encryptions of ten first names in the California dataset.

To automate this, we fix a simple heuristic that an attacker can use to guess a message given the ciphertext c. First, compute m_c and the range R_c. Let $[m_l, m_u] = R_c$. Check if any of m_l, m_c, m_u contains two consecutive spaces in the first half of their string representations. Some names contain a single space, so we can only confirm padding after seeing two consecutive spaces. If none do, then forego outputting a guess. Otherwise, for (an arbitrary) one of the strings that does contain two consecutive spaces, simply take the prefix preceding these spaces as the plaintext guess. Observe that this is not *guaranteed* to be the correct plaintext as there could be other validly padded plaintexts in the range R_c.

This heuristic performs quite well, particularly for last names. On average across all our datasets, the raw recovery rate is 45% for first names and 97% for last names. Referring back to Figure 5, our heuristic is able to do almost as well as predicted by simply halving the maximum length and observing the fraction of plaintexts that lie below that length: on average across datasets, 67% of first names are less than 7 characters long, and 99% of last names are less than 14 characters long. While this leakage has been known in the academic literature [6], we believe its severity was not understood for practical scenarios before our work.

We have described above how to directly exploit BCLO's leakage for first and last names, but this approach is readily generalizable to any plaintext distribution containing messages of different lengths. In the case of fixed input length plaintexts, such as ZIP codes and birth dates, there is no need for padding, and so it is impossible to exactly recover any plaintexts using the heuristic above. Nevertheless significant partial information leaks that we will exploit next.

Inference attack with BCLO leakage. The heuristic above recovers less than half of first names on average, and as just mentioned cannot recover full plaintexts for fixed-input-length domains. We can however integrate the BCLO leakage into our non-crossing inference attack.

[2] This fraction is about 56% for ZIP codes and 66% for birthdates, precluding effective use of the sorting attack from NKW, which requires almost all plaintexts to be present.

Let $G = (U, V, E)$ be a bipartite graph where every vertex in U corresponds to a unique ciphertext in C and vertices in V correspond to unique auxiliary data from Z. The sets U and V are sorted, so that the ith largest unique ciphertext is vertex u_i and likewise for v. For each ciphertext $u \in U$ where the ciphertext corresponding to u is c, the adversary computes R_c and then adds an edge (u, v) for each v whose corresponding value p falls within R_c. This excludes edges that fall outside the window for c. Each edge is weighted as before, by $\alpha - |\mathsf{H}_C(i) - \mathsf{H}_Z(j)|$. The adversary outputs the mappings implied by the solution to the max-weight non-crossing bipartite matching problem the graph defines.

For fixed-length inputs such as ZIP codes and birth dates, we run the inference as described above. For variable-length data such as first names and last names, we first run our heuristic attack, and then run the inference as described above only on the ciphertexts for which the heuristic fails to make a guess. The reason to do this is that many plaintexts that are not in the auxiliary data are nevertheless fully recovered by the heuristic.

Inference attack results. Combining the heuristic with inference increases first name recovery rate from 45% to 99% on average across datasets. The smallest recovery rate in any single dataset was 97%. For last names the increase from inference was negligible, as the heuristic alone already obtained 97% recovery on average. The average unique recovery rate was 90% for first names and 94% for last names, with standard deviation less than 2% for both. Our attack here recovers the vast majority of plaintext records, as well as most unique plaintexts. The few unrecovered plaintexts are ones for which the heuristic fails to retrieve them fully and they additionally do not appear in the auxiliary data (i.e., names that are both longer than average and quite rare). Of course, even for these long and rare names, a prefix of the name is nevertheless apparent to attackers.

Figure 7 summarizes the inference attack's recovery rates for birthdates and ZIP codes, for $b = 3$ and $b = 1$. The attacks are much more accurate with $b = 1$. With $b = 3$ we only recover about 1.5% more ZIP codes than with the non-crossing attack. Our accuracy for birthdates with $b = 3$ does not improve compared to the non-crossing attack. With $b = 1$ the attack performed well for birthdates, recovering more than 90% of records and nearly 70% of unique values. This attack performed more modestly on ZIP codes, but we still recovered about 12% of the ZIP code records and 8% of the unique values. The BCLO leakage by itself revealed, on average, 36% of the plaintext for ZIP codes and 31% for birth dates.

That our attack performs well with $b = 1$ is surprising because the probability that the correct plaintext is in the computed window is (roughly) an inverse exponential of b. When $b = 1$ the success probability of their lower-bound attack should ostensibly be only about 0.3 for each ciphertext, but this was clearly not the case in our experiments. The probability of success for the attack in [6] is, however, only analyzed for uniformly-sampled messages, and so they may not be predictive for the non-uniform birthdate and ZIP code distributions.

	Birthdates	**ZIP codes**
Raw, $b = 1$	91	12
Unique, $b = 1$	70	9
Raw, $b = 3$	0	4
Unique, $b = 3$	0	3

Figure 7: Raw and unique recovery rates for birthdates and ZIP codes encrypted with BCLO. The value b refers to the window width discussed above.

VI. Attacking the CLWW Schemes

Our results show that the BCLO scheme's additional leakage represents a significant threat to plaintext confidentiality for real datasets. We now turn to suggestions by Chenette, Lewi, Weis, and Wu (CLWW) to provide ORE and OPE that they prove leak less than the BCLO scheme, yet remains practical [12]. We will test this empirically.

The CLWW ORE scheme. In [12] the authors construct a new ORE scheme. As with BCLO, we will for brevity omit the details of the CLWW scheme; our attacks will only abuse its leakage profile. Towards that profile, for two equal-length bit strings x, y let $\text{ind}_{\text{diff}}(x, y)$ be the index of the first bit that differs between x and y. If $x = y$ then $\text{ind}_{\text{diff}}(x, y)$ outputs $|x| + 1$. Then, a sequence of ciphertexts $C = (c_1, \ldots, c_n)$ for which $c_i = \mathcal{E}_k(m_i)$ leaks order as well as, for every pair $1 \leq i < j \leq n$, the value $\text{ind}_{\text{diff}}(m_i, m_j)$. (Note that frequency of plaintexts is leaked by the latter.)

The CLWW authors argue that their scheme leaks less than BCLO for uniform messages: for only one ciphertext their scheme is semantically secure, and for n revealed ciphertexts of length k, the probability that more than log k bits leak is negligible. The BCLO scheme, in contrast, always reveals approximately one-half of the bits of a plaintext. While this is true, it may not matter in practice, as for certain plaintext distributions the CLWW leakage may be worse (or better) than BCLO. Indeed, we will see that for ZIP codes in particular the CLWW leakage reveals nearly every bit of every plaintext.

The CLWW attack. Because the scheme leaks the index of the first bit that differs between two ciphertexts and the ordering of their underlying plaintexts, it also leaks the *value* of the plaintext bit at that index: the lesser plaintext has a 0 at that position, and the greater plaintext has a 1. We now detail how to take advantage of this leakage. Like the attack against BCLO in the last section, we first compute leakage, modify the bipartite graph to include only viable solutions relative to this leakage, and then apply the non-crossing attack.

		Birthdates	ZIP codes	First names	Last names
CLWW ORE	Raw RR	98	97	98	75
	Unique RR	90	77	79	47
Composed ORE	Raw RR	86	11	85	44
	Unique RR	61	8	48	18

Figure 8: Results of CLWW and decomposition attacks.

In more detail, for each ciphertext, compute the leaked bit relative to every other ciphertext in the database. This gives the values of bits at certain positions in the plaintext, which we can represent as a list of pairs $\mathcal{L}(c) = ((p_0, b_0), \ldots, (p_\ell, b_\ell))$. Then, initialize a bipartite graph $G = (U, V, E)$ as per the BCLO attack. The edge set E is initially empty. An edge (i, j) corresponding to ciphertext c_i and auxiliary datum z_j is added only if for each (p_k, b_k) pair in $\mathcal{L}(c_i)$, the value of the p_k bit position of z_j equals b_k. Such edges are given weight as in the non-crossing attack. Finally, the adversary solves the non-crossing bipartite matching problem and outputs the resulting guesses.

CLWW attack results. Recovery rates for our CLWW are presented in Figure 8. For first and last names, the average raw recovery rate across all datasets for the attack against CLWW is 98% and 75%. When these percentages are taken as a fraction of the "maximum" recovery rate, meaning names that cannot be matched are not included, the recovery rate for first names is 98% on average, and 85% on average for last names. Our attack against CLWW is nearly as accurate as the attack against BCLO, meaning that despite the difference in leakage profiles (and proofs by CLWW that it leaks less for uniform plaintexts), the leakage is, in effect, the same for our non-uniform plaintext distributions. For first and last names, the ORE leakage by itself removes most of the uncertainty from the ciphertexts. We can measure this explicitly by looking at the number of auxiliary data elements that have the same bits in the same positions as those that are leaked from a ciphertext. These are the auxiliary elements that will have edges to this ciphertext in the graph G above. For first names on average, there were only four such elements in the auxiliary data. For last names on average there were nineteen elements. The variance in this number was quite high for last names — there was one dataset whose average was only 1.3 elements, but another dataset whose average was 148 elements.

The recovery rates for birthdates and ZIP codes show that the CLWW attack is quite devastating, recovering almost 100% of values. This is better than the attack against BCLO, suggesting that empirically at least, there are realistic distributions for which CLWW performs poorly. The reason is that the birthdates and ZIP codes target datasets contain a large fraction of the possible plaintexts, for example the ZIP codes dataset has about 56% of the approximately 40,000 possible values. In such a situation, the leakage of CLWW is very bad: 85% of all plaintext bits were immediately leaked by $\mathsf{ind}_{\mathsf{diff}}$. The non-crossing step has little trouble recovering the remaining unknown bits.

Composed OPE and ORE. CLWW also suggest that one might improve security by first encrypting with an OPE scheme and then double-encrypting using an ORE scheme. They argue that this at least inherits the security of the OPE scheme, and may do even better. We refer to this as the composed scheme, and investigate it for the case of applying first the BCLO scheme and then the CLWW scheme.

The decomposition attack. We can combine the CLWW and BCLO attacks to recover a smaller prefix of the message (than would be revealed directly by BCLO ciphertexts). At a high level the attack will simply compute the individual bits leaked by CLWW, which are now intermediate BCLO OPE ciphertext bits. We can nevertheless still use these bits to define a window within which the plaintext falls with high probability. The window now will be larger, because we must conservatively span the windows for all possibilities of the unknown intermediate ciphertext bits. In turn, larger windows mean a smaller prefix being leaked. In more detail the attack works as follows, given target ciphertext $C = \{c_0, \ldots, c_n\}$ and auxiliary data $Z = \{z_0, \ldots, z_\psi\}$.

In the first step, the adversary computes the CLWW leakage for each ciphertext, resulting in a list of pairs for each ciphertext $\mathcal{L}(c_i) = ((p_0, b_0), \ldots, (p_\ell, b_\ell))$. Note that the bit values $b_0, \ldots, b_\ell$ are not plaintext values, but rather the intermediate OPE ciphertext bits. Form a bit string for each ciphertext c_i by setting bit p_i to b_i and all other bits to zero. Call this bit string c_i^{L}. Form another bit string c_i^{R} similarly, except now setting bits whose positions do not appear in the list to one. The true intermediate ciphertext must lie in the range $[c_i^{\mathrm{L}}, c_i^{\mathrm{R}}]$.

Now the attacker computes a BCLO leakage window that contains the leakage windows for both c_i^{L} and c_i^{R}. Do so by computing the leakage for each of those two ciphertexts, and then taking the lower bound on the window for c_i^{L} and the upper bound on the window for c_i^{R}. Call this range R_{c_i}. The probability (for uniformly-sampled messages) that the message lies in this range is at least the same probability for the BCLO leakage function, since the range is computed in the same way but is larger.

Initialize a bipartite graph $G = (U, V, E)$ as before, and add edge (c_i, z_j) only if $z_j \in R_{c_i}$. Weight edges as before, by the distance in frequencies. Finally solve the non-crossing bipartite matching instance and output the associated guesses.

Decomposition attack results. Recovery rates for the decomposition attack are also presented in Figure 8. Composition does reduce attack efficacy, suggesting the CLWW intuition about composition improving security is sound, with average raw recovery rates of 85% and 44% for first and last names. This is about two percent and seven percent better for first and last names than the attacks with just order and frequency leakage discussed in Section IV. When taken as a percentage of the "maximum", the recovery rates are, on average, 86% for first names and 50% for last names. Composition also seems to improve security for birthdates and ZIP codes, reducing the raw recovery rates by 13% and 86% respectively. These are still much better than the corresponding numbers from Section IV. Unfortunately, this reduction in attack efficacy is probably not enough to conclude security, and it may be that one can improve on our composition attack somehow.

VII. Known-Plaintext Attacks

The threat models considered in the OPE literature thus far have excluded known- and chosen-plaintext attacks. In some use-cases of OPE/ORE, untrusted parties may not be able to know or add values. However, this assumption seems unfounded in many practical scenarios, as we now discuss.

Adversarial capabilities in practice. Recall from Section II that ORE and OPE are often deployed to encrypt customer data ultimately stored in customer relationship management systems such as Salesforce. A typical practice of companies is to allow any website visitor to download technical reports if they provide some personal information in a contact form. This technique, sometimes referred to as marketing automation, is widely used. For those companies that use encryption tools, the information entered in these forms is automatically encrypted and then stored in a cloud service used for marketing analytics.

This immediately gives rise to the ability to mount chosen-plaintext attacks. The adversary can fill out the form as desired, and later see the encryptions of this data in the database. It will be apparent to the adversary what encryptions relate to the adversarially chosen inputs, if for no other reason than the timing of submission of data and receipt of new encrypted database items. A similar (albeit artificial) example of a chosen-plaintext oracle on email encrypted with OPE was given in a technical report by Kolesnikov and Shifka [27]. As far as we know, we are the first to report on a chosen-plaintext attack on OPE which arose naturally from a customer use case. A diagram of the attack is shown in Figure 9.

There exists a folklore chosen-plaintext attack against any OPE construction that allows an adversary to learn every bit of a plaintext from a ciphertext given only $\log(M)$ encryption oracle queries, where M is the size of the ciphertext space. Call the ciphertext the adversary wants to decrypt c_t. The attack is a simple binary search, as follows: first the adversary queries the plaintext $\frac{M}{2}$ and gets a ciphertext c_1. If $c_1 < c_t$ it queries $\frac{M}{4}$, else it queries $\frac{3M}{4}$. It repeatedly halves the range until it finds the point p_t whose ciphertext is c_t.

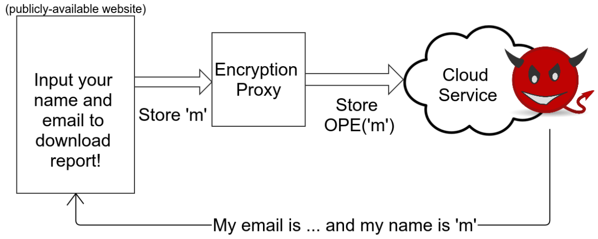
(publicly-available website)
Input your name and email to download report!
Store 'm'
Encryption Proxy
Store OPE('m')
Cloud Service
My email is ... and my name is 'm'

Figure 9: A chosen-plaintext attack against cloud-hosted marketing automation

Since this attack can trivially recover every plaintext from any message distribution, we will not present results for it in this work. We will instead look at the weaker assumption where the plaintexts are known but not chosen, which may make the attack even easier to mount in practice. As we show below, just a handful of known values combined with inference is enough to improve success rates. It may be interesting to explore weaker versions of the chosen-plaintext setting, such as one where the adversary has a limited number of queries or must choose its queries non-adaptively. We expect these attacks to perform even better than the ones presented below, but we leave detailed investigation to future work.

The partitioning attack. We now sketch a simple, generic approach to taking advantage of known or chosen plaintexts. It can be used against any scheme that leaks at least frequency and order, and in conjunction with any of the chosen-ciphertext inference attacks against such schemes.

Assume the attacker is given not only the ciphertext sequence $C = (c_1, \ldots, c_n)$ and auxiliary information $Z = (z_1, \ldots, z_\psi)$, but also the plaintexts for some $q < n$ of these ciphertexts. The adversary knows the positions of these q ciphertexts, let those positions be $p_1, \ldots, p_q$. For notational simplicity assume that $p_1 > 1$ and $p_q < n$.

The adversary can then partition the inference problem for the n ciphertexts into (at most) $q + 1$ sub-problems by splitting at each location p_i. In more detail, for each $1 \leq i \leq q$ and letting $p_0 = 1$ and $p_{q+1} = n$, define the new problem instances to be $C^i = (c_{p_{i-1}+1}, \ldots, c_{p_i - 1})$ and $Z^i = (z_{p_{i-1+1}}, \ldots, z_{p_i - 1})$. Each (C^i, Z^i) pair we can then run independently using whichever ciphertext-only inference attack we prefer. The adversary then takes the union of the resulting guesses, adds the q known plaintexts to this solution appropriately, and outputs the result as its guess.

	Birth dates			ZIP codes		
	0.25%	2.75%	5.25%	0.25%	2.75%	5.25%
Raw RR	90	95	96	22	51	61
Unique RR	68	78	81	14	35	41

Figure 10: Known-plaintext attacks on Birth dates and ZIP codes with order and frequency leakage. Numbers in the second row refer to the percentage of known plaintexts.

Results. We perform experiments using the partitioning attack together with the non-crossing attack using just frequency and order leakage. While the attack increases success for the BCLO, CLWW, and decomposition attacks as well, the relative gain will be more modest as those attacks (without known plaintexts) already perform well. To enable comparison across the various names datasets, we set the number of known plaintexts to be a fraction of the total number of target plaintexts. We experiment with 0.25%, 2.75%, and 5.25% of randomly-sampled plaintexts being known by the adversary. These correspond to somewhere between 5 and 300 plaintexts for 0.25% and 150 and 6,000 plaintexts for 5.25%, depending on the dataset. Recovery rates are averaged over five trials. Because the distribution is uniform over *unique* names and first and last names are long-tailed in terms of frequency, the raw frequency of the randomly-sampled known plaintexts was very low in all experiments. Intuitively, this is because only a small fraction of unique names have high frequency; most occur only once or twice. Birthdates and ZIP codes have a flatter distribution, so the sampled plaintexts had very low frequency there as well.

Figure 10 shows the results of our known-plaintext partitioning attack for 0.25%, 2.75%, and 5.25% of the unique birthdates and ZIP codes. The partitioning attack performs extremely well for birthdates. With only 0.25% of values known, the raw recovery rate jumps from less than one percent (in our attack with no known plaintexts) to nearly 90%. This jump can be attributed to the density of birth months and days for high-frequency years. Since the known plaintexts will, with high probability, reveal the year of most birth dates by upper- and lower-bounding unknown values, the non-crossing attack simply matches the days of the year in sequence. Another way of looking at this is that once the partitioning occurs, the non-crossing attack just performs a kind of sorting attack on the days in each partition. This "density" property is not specific to our datasets — any real birthdate dataset of comparable size to ours will have this density property. The partitioning attack also increases accuracy for ZIP codes substantially.

For first names, the increase in average recovery rates was modest. For 0.25% the average was 84%, only about 0.5% higher than the attack with no known plaintexts. With 5.25% the average was 87%, again a modest gain. The non-crossing attack with no known values already does quite well for first names, so having known plaintexts can only aid us in recovering very low-frequency values. For last names the known plaintexts had a bigger effect. Compared to a 38% average with no known plaintexts, having 0.25% of values known gives a 40% average, and having 5.25% of values known gives a large increase to 49% on average. The standard deviation of attacks on first names was around 7%, and for last names it was between 11% and 14%.

VIII. Attacking Frequency-Hiding Schemes

All our previous attacks are against deterministic OPE and ORE schemes. OPE and ORE are not inherently deterministic but no security notions or constructions of randomized OPE/ORE were known until recently. The first notion of security for randomized ORE was provided by Boneh et al. [7], and they also give a scheme based on multilinear maps that provably meets their definition. A more efficient randomized OPE scheme was proposed by Kerschbaum [25] along with a suitable security notion it was shown to provably meet. The scheme preserves order by storing state (in the form of a binary search tree) on the client. When a value is added, the tree is traversed as it would be in a standard binary tree insertion operation. If the value to be inserted is already present in the tree, randomness is used to choose a new ciphertext for the value, while preserving order.

The Kerschbaum scheme is requires an interactive protocol, and the client must store state whose size is proportional to the number of elements in the database. It also has mutable ciphertexts. All these issues are inherent hurdles to deployment. Nevertheless, there may in the future be settings in which deployment is feasible, or other schemes may be produced that hide frequency while being more practical. We therefore seek to analyze security of schemes that only leak order. To that end, we will describe a simple attack that targets high-frequency elements of a distribution by estimating where the ciphertexts of those elements are relative to the others. We call this attack the "binomial" attack because it uses a simple biased-coin model to estimate the locations of plaintexts.

Plaintext ranges and coin flips. We'll start with some preliminaries. Let $C = (c_1, \ldots, c_n)$ be an ordered list of (randomized) ciphertexts. For simplicity, assume each ciphertext is an encryption of some element of the attacker's auxiliary data Z. As in all our attacks, the basic task we need to perform is a kind of labelling or matching — given a ciphertext, we need to guess what is its underlying plaintext. None of our prior approaches apply here, though, since frequency is not leaked. So, more precisely, we need to find the *range* of ciphertexts which are all encryptions of the same underlying plaintext. Let the plaintext whose range we're trying to find be z_i. The encryptions of z_i are, by correctness, a contiguous sub-list of C, and we need to find the first and

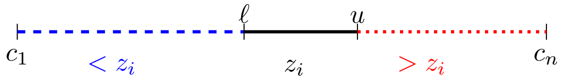

Figure 11: Pictorial aid for explaining the binomial attack. To recover the plaintext z_i, the attacker must locate the range $[\ell, u]$ of ciphertexts whose encryptions are z_i.

last indices of this sub-list. In Figure 11 these two indices are denoted ℓ and u. We can estimate ℓ and u using two simple observations.

The first observation is that if we know ℓ, we can estimate u by estimating the number of times the element z_i occurs in an n-element draw with replacement from Z. If z_i is drawn k times, then $u = k + \ell$. With the auxiliary data Z, estimating k is trivial: if f_{z_i} is the probability of drawing z_i, the distribution of k is the number of times heads occurs in n flips of a biased coin where the probability of a single head is f_{z_i}. Thus, the expected value of k is $n \cdot f_{z_i}$.

We know how to estimate the upper bound given the lower bound, so to finish we only need to show how to estimate the lower bound. Our second observation is that if we can estimate the number of elements of Z *strictly less than* z_i in our sample, call this number j, then we know ℓ right away. Namely, $\ell = j + 1$. In Figure 11 this is the dashed blue region labelled "$< z_i$". Estimating j is, again, another biased coin model: if $f_{<z_i}$ is the total probability of all elements strictly less than z_i, the distribution of j is the number of heads in n flips of a biased coin, where heads occurs with probability $f_{<z_i}$. The expected value of j is $n \cdot f_{<z_i}$.

Confidence intervals. To account for deviations from the expected values of ℓ and u which can happen by chance, we will instead bound the values ℓ and u using Hoeffding's inequality. This inequality says that for n i.i.d. tosses of a coin which returns heads with probability p, for any $\epsilon > 0$, the number of heads $H(n)$ obeys the inequality

$$\Pr\left[\,(p-\epsilon)n \le H(n) \le (p+\epsilon)n\,\right] \ge 1 - 2e^{-2\epsilon^2 n} \ .$$

Clearly, a larger ϵ leads to a larger range of possible values for $H(n)$, which compensates for more uncertainty about the exact plaintext distribution. However, a larger ϵ also lowers the precision (i.e., causes false positives) and can cause ranges to overlap, which requires special handling (as we will describe below). Using the bound, an ϵ can be computed for any desired confidence $d \in [0, 1]$ as

$$\epsilon = \sqrt{\frac{\log \frac{1-d}{2}}{-2n}} \ .$$

To sum up, for plaintext z_i with f_{z_i} and $f_{<z_i}$ defined above, we will estimate the range $[\ell, u]$ for z_i as in Figure 11 as

$$\left[(f_{<z_i} - \epsilon)n \ \ , \ \ (f_{<z_i} + \epsilon)n + (f_{z_i} + \epsilon)n\right] \ .$$

First names

Rank	RR	Rank	RR
1	94	6	100
2	91	7	95
3	85	8	83
4	89	9	94
5	72	10	72

Last names

Rank	RR	Rank	RR
1	83	6	98
2	63	7	82
3	87	8	62
4	100	9	74
5	37	10	63

Figure 12: Average recovery rates (RR) for top ten first and last names in the auxiliary data with our binomial attack. Rank refers to its position in the histogram sorted descending by frequency.

This takes the lower bound of the estimate for ℓ and the upper bound of the estimate for u.

There is one issue of practical importance we have not resolved: if ranges for two different plaintexts overlap, the attacker has some ambiguity about which guess is correct. In our implementation, we resolve overlaps by splitting the range proportional to the probabilities of the two elements. So, for example, if ranges corresponding to plaintexts z_i and z_j overlap, we allot a fraction $f_{z_i}/(f_{z_i} + f_{z_j})$ of the overlap to plaintext z_i and $f_{z_j}/(f_{z_i} + f_{z_j})$ to plaintext z_j. This is a heuristic that seems to work well, but a more principled approach may be possible.

Running the binomial attack. To actually run the binomial attack to recover plaintexts from an ordered list of ciphertexts $(c_1, \ldots, c_n)$ of size n, choose the first k highest frequency plaintext elements $Z' = \{z_1, \ldots, z_k\}$ in the auxiliary data. For each z_i, compute the range $[\ell_{z_i}, u_{z_i}]$ using the method described above and output the mapping $(z_i, \{c_\ell, c_{\ell+1}, \ldots, c_u\})$.

The attacker can target any number k of the highest-frequency elements of the plaintext distribution. There is a point of diminishing returns, though: when an element is too low-frequency we will fail to find any of its ciphertexts because small mismatches between the predicted and actual frequency will cause its ciphertexts to be shifted entirely out of the predicted range.

Results. For all experiments we computed the interval width ϵ using confidence $d = 0.99$. Our recovery rates for birthdates and ZIP codes were low. However, our prefix recovery rate was 37% for birthdates, one full character over the baseline on average. This means our hypothetical attacker can (on average) learn the decade of birth for some records in the database. Our prefix recovery rate for ZIP codes was 12%. Our attack did not perform particularly well for ZIP codes, which is unsurprising — its distribution is closer to uniform than the others.

For first and last names we will discuss two different notions of "recovery rate". Since we are explicitly attacking certain elements of the distribution, one logical way to quantify recovery is as the fraction of ciphertexts of elements we attacked that we correctly matched to their underlying plaintext. We will refer to this as "average recovery rate for the top k names". The other notion of "recovery rate" is the standard one from above — namely, the number of correctly recovered ciphertexts divided by the total number of ciphertexts.

Recovery rates for the top 10 highest-frequency first and last names (i.e., $k = 10$ in the attack description above) are presented in Figure 12. For first names, we recovered the name "michael" with 94% accuracy on average. For last names, we recovered the name "brown" with 100% accuracy in all datasets. The average recovery rate for the top 10 most frequent first names was 86%, and for the top 10 most frequent last names was 76%. Attacking only the top 10 most frequent names, the average whole-dataset recovery rate for first names was 21% and for last names was 4%.

Attacking the top 40 most frequent names, the whole-dataset recovery rates go up (to 30% for first names and 7% for last names) but the per-name recovery rates are lower. For example, when attacking the top 40 first names we only recover 58% of the "michael"s on average. Note that this whole-dataset recovery rate for last names is actually *higher* than the corresponding recovery rate for the NKW greedy attack on last names discussed in Section IV, despite the fact that this attack (unlike NKW greedy) does not use any frequency information.

One would expect that recovery rates might go down for lower-frequency names, but the results suggest that some lower-frequency names have higher recovery rates. This is because of overlaps. For example, the names "james" and "john" are lexicographically close, so fixing the overlaps in their intervals introduces dataset-dependent error. This is also the reason why the recovery rates for some names go down as we attack more names (e.g., "michael" with $k = 10$ vs. $k = 40$ in the previous paragraph).

Attack variants. The attack's use of confidence intervals allows the attacker to tune the precision/recall trade-off of the attack. For example, if the attacker only cares about attacking those people named "michael" but wants to be very sure only "michael"s are attacked, it can lower the confidence value d above to make fewer guesses which have high precision. If it wants to maximize recall, it can instead set the d value very close to 1.

We leave exploring the trade-off between accuracy and the confidence value as an open problem. An interesting setting is one in which the attacker has some model of the discrepancy between its auxiliary data and the true plaintext distribution. With this, an attacker could use different confidence values for different plaintexts.

The bigger picture. Stepping back, we should reflect on the implication of these results for the security of *any* OPE or ORE. Our results show that high-frequency elements encrypted with OPE or ORE can be reliably recovered (with probability many times better than the baseline) *even if their frequency is not leaked.* This raises questions about the fitness of OPE and ORE for the highly non-uniform distributions that arise in practice.

IX. Related Work

Property-revealing encryption schemes. The study of encryption schemes that reveal order was initiated by Agrawal et al. [1], who constructed a scheme that preserved the order property for numeric inputs, but lacked provable guarantees. The first study of OPE with reductionist security guarantees was due to BCLO [5]. Their scheme's window one-wayness security was subsequently analyzed by Boldyreva, Chenette, and O'Neill [6]. We use their analysis in our attacks against the BCLO scheme, as discussed in Section V.

In [6], it was shown that any OPE scheme provably leaking only frequency and order must have exponentially large ciphertexts. Popa et al. [37] extended this negative result to cover stateful, interactive schemes that have immutable ciphertexts. They also introduced an interactive OPE scheme with mutable ciphertexts that leaks at most order and frequency information. Kerschbaum and Schröpfer propose a more efficient interactive OPE scheme [26]. Our attacks in Section IV work against these schemes. Later, Kerschbaum proposed a frequency-hiding, interactive OPE scheme [25], our attack in Section VIII applies to this scheme.

ORE was introduced by Boneh et al. [7] with the hope of providing better security while still allowing order comparison. Their scheme relies on multilinear maps [18], making them currently inefficient [29] and potentially insecure given doubt about the validity of cryptographic hardness assumptions related to multilinear maps [2, 21, 33]. Our attack in Section VIII would affect these schemes. CLWW provide a practical ORE scheme that also leaks frequency [12], our attacks in Section VI affect this scheme, as well as their suggestion of composing OPE with ORE. Lewi and Wu [30] developed a new ORE construction that leaks less than CLWW, but does not achieve ideal leakage. Their construction is elegant but is not yet practical due to large ciphertext sizes. For security parameter $\lambda \approx 128$ and plaintext domain of size n, their ciphertexts require $\mathcal{O}(\lambda n)$ space.

Schemes supporting range queries Recent schemes [16, 30, 41] have been developed that only support range queries. Lewi and Wu's scheme [30] can be modified to support only range queries. Their modified scheme has security similar to the frequency-hiding scheme of Kerschbaum, and is vulnerable to our attacks on that scheme. The scheme of Roche et al. [41] is designed with the assumption that not all values will be queried. This assumption allows them to achieve stronger security at the cost of applicability to most applications of ORE. Faber et al. [16] adapt the OXT searchable encryption protocol of Cash et al. [11] to support range queries by transforming them into disjunctive queries, but at the cost of a very large increase in database size.

To enable range queries on a column containing M n-bit values with security parameter $\lambda \approx 128$, they require size $\mathcal{O}(\lambda M \log n)$. Their scheme does resist offline inference attacks, but the large increase in space complexity greatly diminishes its practical deployability. Since these are not ORE schemes they are outside the scope of this work.

Systems using ORE/OPE. Several companies and services advertise encryption that preserves functionality, including CipherCloud and Skyhigh Networks [13, 22, 36, 43]. CryptDB [38, 40] was the first system in the academic literature that introduced a scheme for running a large subset of the SQL language on the server side, given an encrypted database (EDB). It used deterministic encryption and OPE, as well as standard encryption. Since then, the popularity of EDBs has increased. Many companies such as IQrypt [22] and SAP [42] are producing their own CryptDB-inspired solutions. Some more recent academic systems use OPE or ORE as well, such as Seabed [35] and Minicrypt [48].

Attacks against property-revealing encryption. Inference attacks were first considered against searchable symmetric encryption [14, 44] by Islam, Kuzu, and Kantarcioglu [23] with subsequent improvements and investigation of active attacks by Cash et al. [10] and Zhang et al. [47]. Grubbs et al. [19] presented an active attack against the multi-user searchable encryption scheme used in Mylar [39]. Naveed et al. [34] were the first to consider inference attacks against CryptDB-style EDBs, as discussed in Section IV.

Concurrent work. In a concurrent and independent work, Durak, DuBuisson, and Cash [15] show how the sorting attack from NKW [34] can be extended to the multi-column case, when two or more columns contain correlated data. They demonstrate that multi-column attacks are more devastating than separate attacks on individual columns. They also show attacks on non-ideal OPE/ORE schemes that leak more than the order and frequency of the ciphertexts, including the BCLO scheme. Their primary focus is sparse datasets such as GPS coordinates, and do not look at names, birthdates, or ZIP codes as we consider in our running case study.

Durak et al. do not exploit auxiliary information, and instead rely only on leakage, whereas our attacks show how to both exploit leakage by itself and to augment an auxiliary-information-using inference attack. Their attacks therefore only provide approximate recovery of plaintexts (e.g., determining that a plaintext lies within some 10km radius), whereas our attacks recover entire plaintexts, and in many cases fully recover most records in a database.

X. Conclusion and Future Work

In this work we have studied the security of OPE and ORE as they are used in real systems. We developed new cryptanalytic techniques for several extant OPE and ORE schemes and evaluated them experimentally by performing plaintext recovery attacks against first and last names, birthdates, and ZIP codes from several real datasets. Our attacks are effective in fully recovering plaintexts from OPE and ORE ciphertexts.

Our work here has been empirical, but we believe our attacks will prove effective against many other kinds of data sets used in practice. We also leave as an open question providing a more formal analysis of inference attacks. Future work could also develop *adaptive* inference attacks, where an attacker can make a limited number of adaptive chosen plaintext queries while running an inference attack.

Our results suggest that OPE/ORE often provides only marginal security for an important use case. We believe the results will generalize to many other settings. We also do not know if these attacks are optimal, so future work may surface even more damaging attacks. Given all this, we recommend practitioners avoid using OPE/ORE if possible. In some deployment scenarios the only practical alternative in the short term is leaving data in the clear, and here OPE/ORE is clearly better than no encryption. For such cases, we hope our methodologies can be used to help evaluate the security achieved.

Acknowledgements

Grubbs and Ristenpart have large financial stakes in Skyhigh Networks. This work was supported in part by NSF grants CNS-1330308, CNS-1514163, and a generous gift from Microsoft. Naveed was partially supported by a Google PhD fellowship and the Sohaib and Sara Abbasi fellowship.

The authors wish to thank David Cash, whose insightful observation about non-crossing bipartite matchings led to the non-crossing attack in Section IV. The authors also thank Elaine Shi and Ethan Ceccheti for helpful comments on an early version of this work.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *ACM SIGMOD*, 2004.

[2] M. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched NTRU assumptions. In *CRYPTO*, 2016.

[3] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with Cipherbase. In *CIDR*, 2013.

[4] D. Avis. A survey of heuristics for the weighted matching problem. *Networks: An International Journal*, 1983.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.

[6] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[7] 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*, 2015.

[8] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *IEEE Symposium on Security and Privacy*, 2012.

[9] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012.

[10] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM CCS*, 2015.

[11] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.

[12] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *FSE*, 2016.

[13] Ciphercloud. http://www.ciphercloud.com/.

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 2011.

[15] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *ACM CCS*, 2016.

[16] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security*, 2015.

[17] S. Gallagher. MS researchers claim to crack encrypted database with old simple trick. https://tinyurl.com/qcpzcvl.

[18] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[19] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov. Breaking web applications built on top of encrypted data. In *ACM CCS*, 2016.

[20] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song. Shadowcrypt: Encrypted web applications for everyone. In *ACM CCS*, 2014.

[21] Y. Hu and H. Jia. Cryptanalysis of GGH map. Cryptology ePrint Archive, Report 2015/301, 2015. http://eprint.iacr.org/2015/301.

[22] Iqrypt: encrypt and query your database. http://www.iqrypt.com/.

[23] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. *NDSS*, 2012.

[24] S. Keelveedhi, M. Bellare, and T. Ristenpart. DupLESS: server-aided encryption for deduplicated storage. In *USENIX Security*, 2013.

[25] F. Kerschbaum. Frequency-hiding order-preserving encryption. In *ACM CCS*, 2015.

[26] F. Kerschbaum and A. Schröpfer. Optimal average-complexity ideal-security order-preserving encryption. In *ACM CCS*, 2014.

[27] V. Kolesnikov and A. Shikfa. On the limits of privacy provided by order-preserving encryption. *Bell Labs Technical Journal*, 2012.

[28] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. Mimesis Aegis: A mimicry privacy shield–a systems approach to data privacy on public cloud. In *USENIX Security*, 2014.

[29] K. Lewi, A. J. Malozemoff, D. Apon, B. Carmer, A. Foltzer, D. Wagner, D. W. Archer, D. Boneh, J. Katz, and M. Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In *ACM CCS*, 2016.

[30] K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM CCS*, 2016.

[31] T. Malkin, I. Teranishi, and M. Yung. Order-preserving encryption secure beyond one-wayness. *ASIACRYPT*, 2014.

[32] F. Malucelli, T. Ottmann, and D. Pretolani. Efficient labelling algorithms for the maximum noncrossing matching problem. *Discrete Applied Mathematics*, 1993.

[33] E. Miles, A. Sahai, and M. Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *CRYPTO*, 2016.

[34] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM CCS*, 2015.

[35] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big data analytics over encrypted datasets with Seabed. In *OSDI*, 2016.

[36] Perspecsys: A blue coat company. http://perspecsys.com/.

[37] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE Security and Privacy*, 2013.

[38] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *ACM SOSP*, 2011.

[39] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan. Building web applications on top of encrypted data using Mylar. In *NSDI*, 2014.

[40] R. A. Popa, N. Zeldovich, and H. Balakrishnan. Guidelines for using the cryptdb system securely. Cryptology ePrint Archive, Report 2015/979, 2015. http://eprint.iacr.org/2015/979.

[41] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich. POPE: Partial order preserving encoding. In *ACM CCS*, 2016.

[42] Sap seeed project. https://www.sics.se/sites/default/files/pub/andreasschaad.pdf.

[43] Skyhigh Networks. https://www.skyhighnetworks.com/.

[44] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Security and Privacy*, 2000.

[45] US census bureau name statistics. https://www.ssa.gov/OACT/babynames/.

[46] US social security name statistics. https://www.ssa.gov/OACT/babynames/.

[47] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security*, 2016.

[48] W. Zheng, F. Li, R. Agarwal, R. A. Popa, and I. Stoica. Minicrypt: Reconciling encryption and compression for big data stores. In *EuroSys*, 2017.