Learning from Functionality Outputs: Private Join and Compute in the Real World

Francesca Falzon
ETH Zürich, Switzerland

Tianxin Tang
Eindhoven University of Technology, Netherlands

Abstract

Private Join and Compute (PJC) is a two-party protocol recently proposed by Google for various use-cases, including ad conversion (Asiacrypt 2021) and which generalizes their deployed private set intersection sum (PSI-SUM) protocol (EuroS&P 2020). PJC allows two parties, each holding a key-value database, to privately evaluate the inner product of the values whose keys lie in the intersection. While the functionality output is not typically considered in the security model of the MPC literature, it may pose real-world privacy risks, thus raising concerns about the potential deployment of protocols like PJC.

In this work, we analyze the risks associated with the PJC functionality output. We consider an adversary that is a participating party of PJC and describe four practical attacks that break the other party's input privacy, and which are able to recover both membership of keys in the intersection and their associated values. Our attacks consider the privacy threats associated with deployment and highlight the need to include the functionality output as part of the MPC security model.

1 Introduction

Private Join and Compute (PJC) is an expressive functionality proposed by Google for various real-world use-cases, including ad conversion [29]. PJC enables two parties to privately carry out data analytics on their datasets. In particular, it takes as input two databases, $X = \{(\mathsf{key}_i, \mathsf{val}_i)\}_{i\in[n]}$ and $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in[m]}$, from parties P_1 and P_2, respectively, and outputs the inner product of the values whose keys lie in the intersection i.e., $\sum_{i\in[n],j\in[m]} \mathbb{1}\{\mathsf{key}_i = \mathsf{key}'_j\} \cdot \mathsf{val}_i \cdot \mathsf{val}'_j$, where $\mathbb{1}\{\cdot\}$ denotes the indicator function.

While, historically, the functionality output was not analyzed in the secure multi-party computation (MPC) literature, the potential deployment of protocols like PJC make the threat of such an adversary an imminent reality. For example, the code for the recently deployed Private Set Intersection Sum (PSI-SUM) protocol warns that the intersection-sum *could* reveal something about the intersection [19,20]. Their suggestions for mitigating such attacks include scrubbing inputs to remove "outliers", aborting if the intersection-size is too small, and adding noise to the output.

We go beyond basic functionalities like PSI-SUM, and give a series of attacks against the more complex PJC functionality. As we will see in Section 3, PJC can be viewed as a generalization of PSI-SUM, which outputs the sum of one party's payloads whose keys lie in the intersection. As such, our attacks apply to all protocols that realize both functionalities, including the deployed PSI-SUM protocol of Ion et al. [25]. Our attacks exploit the functionality as it is meant to be used. They do not involve deviating from the protocol or breaking the underlying cryptographic primitives, thus highlighting an intrinsic vulnerability of the output.

Our attacks adapt a wide range of techniques from the combinatorics, statistical inference, and signal processing literature to mount powerful attacks that use the PJC functionality output to recover *not only membership of the keys in the intersection, but also the associated values of those keys*, a recovery goal we coin **key-value recovery (KVR)**. We consider both adaptive and non-adaptive adversarial settings, and show that even an honest-but-curious user carrying out data analytics over a prolonged period of time can achieve KVR with a number of queries linear in the intersection size. Our attacks further show that the aforementioned mitigation techniques are not sufficient and we present attacks in which no outlier values are used and which are robust to noise.

The deployment of these efficient tailor-made MPC solutions brings new challenges regarding the design of end-to-end MPC systems to the forefront. By understanding the extent to which the functionality outputs can be leveraged to learn about the private inputs, we hope to provide intuition for their safe and privacy-preserving deployment. We use the attacks we developed to highlight lessons learned and conclude with recommendations for potential mitigation techniques and future work.

This is the full version of an article to appear at USENIX'25.

	Srch. Tree (§4)	MLE (§5)	DFT (§6.2)	Comp. Sens. (§6.1)
Adaptive	✓			
Non-Adaptive		✓	✓	✓
Exact Recovery	✓		✓	✓
Approximate Recovery		✓		✓
Negative Values		✓	✓	✓
Robust to Noise				✓
Known Value Distribution		✓		

Table 1: Our four attacks (from left to right: the search-tree attack, the maximum-likelihood estimation attack, the discrete fourier transform attack, and the compressed sensing attack) and their assumptions. Adaptive (resp. Non-Adaptive) means that the adversary can (resp. cannot) adaptively chose their input sets. Exact Recovery (resp. Approximate Recovery) means that the adversary can recover the associated values exactly (resp. approximately). A check under Negative Values denotes that the attack can also recover negative values. Known Value Distribution indicates that the adversary must also have auxiallary knowledge of the distribution from which the other party's values are drawn from. All of our attacks assume knowledge of the maximum and minimum possible associated values in the victim's set and that the victim's set is static.

1.1 Our Contributions

We present four attacks, each of which fall into an *adaptive* or *non-adaptive* adversarial setting. In the former, the adversary chooses the associated values of the keys in the target set adaptively. In the latter, the adversarial input doesn't depend on the response. The adversary is a participating member of the PJC protocol (denoted $\tilde{P}_1$) and holds a **target set** of keys, $T = \{\mathsf{key}\}_{i\in[n]}$. The other party, P_2, holds the **recovery database** Y and $\tilde{P}_1$'s goal is to recover the keys in T also contained in Y and their corresponding associated values. We assume that P_2's database is static – a reasonable assumption since data analysis is often carried out on static data or data that is infrequently updated (e.g., annual statistics). A base line attack exists in which the adversary brute force queries each $\mathsf{key} \in T$ using $|T|$ queries. In contrast, many of our attacks succeed with a number of queries logarithmic in $|T|$.

Our first attack is adaptive and takes a binary-search-like approach. This attack is highly scalable, efficient, and works best when the intersection size is small. We show that an adversary requires at most $O(k \log_\ell |T|)$ queries for exact KVR, where k is the intersection size and ℓ is a tunable parameter.

The remaining attacks are non-adaptive and utilize the following observation. Multiple queries to PJC allows one to construct a system of linear equations in which the unknowns correspond to the associated values in Y contained in the intersection. While one could use Gaussian elimination when $|T|$ equations are constructed (thus matching the naive attack), we aim for $< |T|$ queries. One of our attacks uses maximum likelihood estimation (MLE) and auxiallary knowledge of the underlying distribution, to achieve approximate KVR even when the solution space has high dimension.

Our third attack succeeds when the intersection size, k, is small or *sparse*; we adapt compressed-sensing techniques from the signal processing literature to recover the values exactly using only $2k \log_2(n/k)$ queries. This approach is robust to small-scale noise added to the inner product and we also show how it can be be used to infer membership from the outputs of PSI-SUM and PSI Cardinality (PSI-CA).

Our last attack uses the discrete fourier transform (DFT), assumes knowledge of the intersection size, k, and requires $2k$ queries for exact KVR. The DFT measurements identify the indices of the unknowns in the linear system that correspond to keys in the intersection. Once these indices are identified, the unknowns can be recovered.

We support our theoretical work with implementations and demonstrate our attacks' efficiency. We then conclude with suggested countermeasures and future work.

1.2 Related Work

PSI Protocols for Joins. Ion et al. [20] proposed a protocol for Private-Set Intersection Sum, and which was consequently deployed by Google for the purpose of ad conversion [19] Follow-up work proposed a protocol for enabling parties to privately compute the inner product between attribute values associated with items in the intersection [29]. The Apple PSI System [5] is designed to compute threshold PSI with associated data. Yang et al. [36] consider input sets with an associated payload and introduce the Predicate Private Set Intersection primitive for evaluating a predicate over the payloads in the intersection.

The protocol of Mohassel et al. [31] supports SQL-like join/select queries on secret shared data. Meta proposed Private-ID and PS^{3}I which compute a full outer join (union) and an inner join between two databases, respectively [9]. Follow up work extends Private-ID to compute a full outer join of two multi-key datasets [8]. Meta also proposed a protocol which enables the delegation of the join computation to a delegate party [32]. Badrinarayanan et al. [2] gave two protocols for efficiently computing join operations on secret shared database tables with non-unique keys. Recently, Asharov et al. [1] recently proposed the first join and group-by protocols that are also secure against malicious adversaries.

Attacks against PSI-like Functionalities. Guo et al. [24] proposed the first membership-inference attacks against PSI protocols that reveal the intersection cardinality (PSI-CA). They show how a party participating in a PSI-CA protocol and who learns the intersection size can recover the set belonging to the other party. Their attacks improve the naive attack of submitting one element at a time from $O(n)$ to $O(\log n)$, where n is the size of their target set. Jiang et al. [27] further improve the attacks against PSI-CA through a dynamic programming approach and additionally propose an attack against protocols that reveal the sum of the payloads associated with items in the intersection (PSI-SUM). In a related line of work, Zinkus et al. [37] automate the maximiazation of functionality-inherent leakage and derive attacks from this leakage.

Attacks against Encrypted Databases. A number of attacks leveraging the inherent leakage stemming from order-revealing encryption (e.g. [6, 15, 22, 26, 33]) and searchable symmetric encryption (e.g., [17, 21, 23, 28, 34]) have been proposed. Much like our attacks which apply to a broad range of protocols, many of these attacks are generic and only use an inherent leakage common to many ORE/SSE schemes. These attacks, however, are carried out in different threat models and use techniques distinct from those presented here.

2 Private Join and Compute

Notation and Convention. For any $n \in \mathbb{N}$, we let $[n]$ denote the discrete range $[1, n]$. We use $y \leftarrow\$ A$ to indicate y is generated by some algorithm A using A's internal randomness. We use "PPT" as the abbreviation of "probabilistic polynomial time" when describing an algorithm or functionality.

Database Abstraction. We abstract the inputs of the parties as a **database** (i.e., table), DB. A database of size n comprises of n rows (i.e., tuples) such that each row has a unique key (i.e., a primary key) and an associated value. [1] We denote this as $\mathsf{DB} = \{(\mathsf{key}_i, \mathsf{val}_i)\}_{i\in[n]}$. For a key-value pair $(\mathsf{key}, \mathsf{val}) \in \mathsf{DB}$ we write $\mathsf{DB}[\mathsf{key}] = \mathsf{val}$.

Join Logic. In relational databases, a database join provides a way to compute on the joint data contributed by multiple databases. In this work, we consider the **Private Join and Compute (PJC)** functionality [25], which enables two parties – each holding a database – to privately compute the inner product of the associated values whose keys are contained in both databases. Formally, the functionality takes as input two databases (i.e., two sets of key-value pairs) $X = \{(\mathsf{key}_i, \mathsf{val}_i)\}_{i\in[n]}$ and $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in[m]}$ from parties P_1 and P_2, respectively, and outputs the inner product:

$$\boxed{\sum_{i\in[n], j\in[m]} \mathbb{1}\{\mathsf{key}_i = \mathsf{key}'_j\} \cdot \mathsf{val}_i \cdot \mathsf{val}'_j.} \tag{1}$$

to party P_1. See Figure 4 for details.

For completeness, we describe PSI-SUM and PSI-CA in Appendix A.1, which can both be viewed as special cases of PJC. We also formally define the real-ideal world paradigm in Appendix A.2, which is the security notion often used in the MPC literature.

3 Technical Background

We now describe specify the threat models considered and provide an overview for our attacks.

3.1 Recovery Goal and Threat Model

Throughout this work, we assume that P_1 is adversarial (we denote the corrupted party as $\tilde{P}_1$) and that P_2 is the party whose database, Y, the adversary wishes to learn about. We call this database the **recovery database**. We focus on a weak form of adversarial control in which the adversary follows the PJC protocol honestly. We stress that our attacks do not rely on deviating from the protocol or breaking any of the underlying primitives.

$\tilde{P}_1$ holds a target set $T = \{\mathsf{key}_j\}_{j\in[n]}$ and aims to learn information about the recovery database. This information includes the individual membership of each $\mathsf{key} \in T$ in Y and, for each member, its associated value.

Definition 3.1. *Let $T = \{\mathsf{key}_i\}_{i\in[n]}$ be the target set and $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in m}$ be the recovery database. The goal of* **key-value recovery** (**KVR**) *is to compute the secret vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)^T$, where for all $i \in [n]$*

$$s_i = \mathbb{1}\{\mathsf{key}_i \text{ in } Y\} \cdot Y[\mathsf{key}_i]. \tag{2}$$

Note that if s_i evaluates to 0, then one of two cases applies: either key_i not in Y or key_i is in Y and $\mathsf{val}_i = 0$. We do not distinguish between these two cases and later show that this limitation is inherent if the functionality does not output the intersection size. In our second and fourth attacks we also consider a relaxed attack goal, approximate KVR, which we define below.

Definition 3.2. *Let $T = \{\mathsf{key}_i\}_{i\in[n]}$ be the target set and $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in m}$ be the recovery database. The goal of* **approximate-KVR** *for some $\epsilon > 0$, is to compute a vector $\hat{\mathbf{s}} = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n)^T$, where for all $i \in [n]$, $\hat{s}_i \in [s_i - \epsilon, s_i + \epsilon]$, and $s_i = \mathbb{1}\{\mathsf{key}_i \text{ in } Y\} \cdot Y[\mathsf{key}_i]$.*

[1] We use the term "key" following the database literature. Single-key databases only contain a single column for the key attributes, and each key attribute uniquely identifies a row.

We consider (approximate) KVR in the adaptive and non-adaptive adversarial settings.

Adaptive Setting. In the adaptive setting, $\tilde{P}_1$ selects their input adaptively from an allowed range. $\tilde{P}_1$ can adaptively update its input to PJC in between queries; for example, it may choose to query a sequence of distinct databases containing different subsets of T with different associated values assigned to those keys.

Our attack based on a binary-search-like approach (Section 4) is adaptive.

Non-Adaptive Setting. In the non-adaptive setting, the associated values assigned to the target set are selected independently from the query output.

Our maximum likelihood attack (Section 5), discrete fourier attack (Section 6.2), and compressed sensing attack (Section 6.1) all fall into the non-adaptive setting.

Assumptions about Data. In all of our attacks, we assume that the victim's database, Y, remains unchanged across multiple protocol runs. This is a natural assumption in data analytics. For example, one party (P_1) may need to apply different weights to the same static data (held by P_2) for training a data model. This assumption also represents a worst-case scenario and allows us to provide a theoretical guideline on rate limiting for countermeasures.

We additionally assume that the adversary knows the maximum and minimum possible associated values in the victim's database (we denote these as maxval and minval, respectively). This is a reasonable assumption, since in many data analytic scenarios, the possible range of values is public knowledge (e.g., $0-120$ for ages, or $0-219\times10^9$ USD for net worth).

Theses assumptions are summarized in Table 1.

3.2 Attacks Overview

We now turn our attention to modeling the inner product. Recall, that for each protocol invocation, the adversary $\tilde{P}_1$ knows its input database X and resulting inner product b, and can thus write the inner product down as a linear equation with P_2's values as variables which it can solve for. If the adversary invokes the protocol m times, then for $i \in [m]$ we let $X = \{(\mathsf{key}_{i,j}, a_{i,j})\}_{j\in[n]}$ be the i-th input database. Since the target set remains consistent across protocol runs, then we can write $\mathsf{key}_{i,j} = \mathsf{key}_j$ and use $a_{i,j}$ to denote the adversary's associated values and distinguish them from the values in the recovery database. For example, in a data analytics task, $a_{i,j}$ may represent weight information. The inner product b_i can then be formally represented as follows,

$$b_i = \sum_{j=1}^{n} \mathbb{1}\{\mathsf{key}_j \text{ in } Y\} \cdot a_{i,j} \cdot \mathsf{val}_j. \tag{3}$$

For convenience, we model this problem in matrix form. Let $\mathbf{a}_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,n}) \in R^n$ denote the associated values from the adversary on the i-th run. After q runs, the adversary knows $A = (\mathbf{a}_i)_{i\in[q]} \in \mathbb{R}^{q\times n}$ and a vector of inner products $\mathbf{b} = (b_1, b_2, \ldots, b_q)^T \in \mathbb{R}^q$ (cf. Equation (3)). These inner products imply that the secret $\mathbf{s} = (s_1, s_2, \ldots, s_n)^T \in \mathbb{R}^n$(cf. Equation (2)) from the victim, must satisfy $A\mathbf{s} = \mathbf{b}$.

Given n unknowns in $\mathbf{s}$, if we have n linearly independent equations in A containing these unknowns (i.e., $q \geq n$ queries), we can recover $\mathbf{s}$ exactly and efficiently using Gaussian elimination. Gaussian elimination requires $\widetilde{\mathcal{O}}(n^3 \log(\|A\| + \|\mathbf{b}\|))$ computation time. In comparison, a naive attack requiring n queries involves probing s_i for every $i \in [n]$ by setting the associated values of the remaining entries in $\mathbf{s}$ to zero, resulting in $b_i = s_i$ and incurring only constant computation time. Since the naive attack is significantly more efficient, we focus on the **number of queries** as the primary metric for evaluating efficiency.

For $q < n$ queries, this problem becomes one of solving an underdetermined linear system, which has an infinite number of solutions. To overcome this limitation, we adopt statistical and algebraic tools in our attacks and organize them into two categories: non-sparse recovery and sparse recovery. We demonstrate that we can circumvent the obstacle of infinite solutions by relaxing our recovery goal and approximately recovering the associated values for non-sparse data i.e., when the victim's set contains a large intersection with the target set. For sparse secrets, where the victim's set has less overlap, we can, quite surprisingly, achieve exact recovery using techniques from signal processing.

4 A Combinatorial Attack

In this section, we assume that the associated values are positive, i.e., that $\mathsf{minval} = 0$ and $\mathsf{maxval} > 0$.

4.1 The Basic Attack

We revisit the following observation from [24]. In their work, the adversary holds a target set T and P_2 holds a key set $Y = \{\mathsf{key}'_j\}_{j\in[m]}$. The adversary adaptively queries a key-value database $X = \{(\mathsf{key}_j, \mathsf{val}_j)\}_{j\in[n]}$ and its goal is to recover $Y \cap T$. Note that unlike in our setting, [24] considers a stronger adversary against PSI-SUM in which the adversary controls the associated values of the input. Inferring membership of which $\mathsf{key} \in T$ is contained in Y is equivalent to solving the subset-sum problem. While the general subset-sum problem is NP-Hard, when the sequence of associated values is **superincreasing** (i.e., $\mathsf{val}_{j+1} > \sum_{i=1}^{j} \mathsf{val}_i$ for all $1 \leq j <$

n), the attacker can efficiently recover the values [30]. We sketch this attack in the following example.

Example 4.1. *Let* $X = \{(\mathsf{key}_1, 10^0), (\mathsf{key}_2, 10^1), (\mathsf{key}_3, 10^2)\}$ *be the adversary's input and* Y *be* P_2*'s key set. Further suppose that the ouput of* $\mathsf{PSI\text{-}SUM}(X,Y)$ *is* 110*. Since* $(10^0, 10^1, 10^2)$ *is superincreasing and* $110 = 10^2 + 10^1$*,* $\tilde{P}_1$ *learns that* $\mathsf{key}_2, \mathsf{key}_3 \in Y$ *and* $\mathsf{key}_1 \notin Y$*.*

Since PSI-SUM is a special-case of PJC, we can extend the above attack to PJC to achieve KVR. We illustrate this attack with the following example.

Example 4.2. *Let* $\mathsf{minval} = 0$ *and* $\mathsf{maxval} = 100$*, and let* $X = \{(\mathsf{key}_1, 10^0), (\mathsf{key}_2, 10^3), (\mathsf{key}_3, 10^6)\}$ *be the adversary's input and* Y *be* P_2*'s database. Further suppose that the ouput of* $\mathtt{PJC}(X,Y)$ *is* $4{,}070{,}000$*. Since each associated value in* Y *is no more than* 100 *and*

$$4{,}070{,}000 = 4 \cdot 10^6 + 70 \cdot 10^3 + 0 \cdot 10^0,$$

$\tilde{P}_1$ *learns that* $(\mathsf{key}_2, 70), (\mathsf{key}_3, 4) \in Y$ in a single query.

This approach can be generalized to target sets of arbitrary size and we call this attack the **basic attack**. The pseudocode can be found in Figure 5.

4.2 The Search Tree Attack

While the basic attack works remarkably well, the number of (base 10) digits of the adversarial input grows linearly with the size of the target set. This can be impractical, since there is usually an implementation-dependent upper-limit on the size of the integers supported. In C, for example, the `int` data type is represented using 32 bits, however, some compilers have expanded the `long` data type to 64 bits. Even in Python, where the `long` data type can handle arbitrarily long integers, the machine may be limited by memory or compute time. We thus give a more advanced attack that addresses this issue. This attack is additionally parameterized by the maximum supported integer, denoted maxint.

In practice, the key universe may be exponentially large and the majority of the elements in the target set may not actually be in the sender's set. Let $k \leq \min\{|T|, |Y|\}$ denote the **intersection size**, i.e., $|\{(\mathsf{key}, \mathsf{val}) \in Y : \mathsf{key} \in T\}|$.

To reduce the number of queries needed for recovery such that the number only scales with k and not $|T|$ or $|Y|$, while also ensuring that the inner product does not exceed maxint, we propose an attack that takes a generalized binary-search-like approach to quickly eliminate target elements that are not in P_2's set. We motivate our approach with an illustrative example.

Example 4.3. *Let* $\mathsf{minval} = 0$ *and* $\mathsf{maxval} = 100$*. Let* $T = \{\mathsf{key}_i : i \in [8]\}$ *be the target set and* Y *be* P_2*'s database. Further suppose that the adversarial database is*

$$X = \left\{ \begin{array}{c} (\mathsf{key}_1, 1), (\mathsf{key}_2, 1), (\mathsf{key}_3, 1), (\mathsf{key}_4, 1), \\ (\mathsf{key}_5, 10^3), (\mathsf{key}_6, 10^3), (\mathsf{key}_7, 10^3), (\mathsf{key}_8, 10^3) \end{array} \right\}$$

and that the output of $\mathtt{PJC}(X,Y)$ *is* $150{,}000$*. Let* s_j *denote the variable for the associated value of* key_j *in* Y *(if* key_j *is not contained in* Y*, then we can take* $s_j = 0$*). From the output,* $\tilde{P}_1$ *can infer the following 2 equations:*

$$0 = \sum_{j=1}^{4} s_j \quad \text{and} \quad 150 = \sum_{j=5}^{8} s_j.$$

Since the values can only take on positive values, then $s_j = 0$ *for* $j \in [4]$ *and thus* $\mathsf{key}_j \notin Y$ *for* $j \in [4]$*.* $\tilde{P}_1$ *has learned that half of the elements in the target set are not in* Y *with one query and can focus its remaining queries on the other half of the target set.*

The **search tree attack** leverages the above observation and partitions the target set into ℓ sets ($\ell = 2$ results in a binary-search-like approach). The sets are then assigned superincreasing values: the keys within a set are assigned the same associated value, and keys in different sets are assigned distinct values. The difference between the assigned values must be sufficiently large that we obtain a domain separation of the partition and can derive ℓ inner products. Concretely, if $A_1 \cup \cdots \cup A_\ell$ is a partition of $A \subseteq [n]$, then we can construct ℓ inner products

$$b_i = \sum_{j \in A_i} \mathbb{1}\{\mathsf{key}_j \text{ in } Y\} \cdot s_j$$

for each $i \in [\ell]$, where s_j is the secret value of the target key_j in Y if it exists and 0 otherwise.

The smaller the intersection ratio (and consequently fewer keys in T are contained in Y), the more of these equations equal 0 and the fewer queries need to be made. The attack recurses on the partitions that correspond to a non-zero inner product until the individual (non-zero) values are recovered.

The search can be thought of as traversing an ℓ-ary search tree whose root corresponds to the set of all indices; the ℓ children of an inner node correspond to a partition of the parent node's index set and the leaves correspond to individual indices. When we issue a query to PJC using the keys indexed by a node of the search tree and the partition induced by its children, we learn the inner product of the target keys indexed by the set corresponding to the ℓ children. In Figure 2 we depict such a tree for $\ell = 3$ and $n = 27$. Here, the partitions are ordered numerically for ease of representation, but in practice can be sampled randomly.

The pseudocode can be found in Figure 1.

Params: The minimum possible associated value $\mathsf{minval} = 0$;
The maximum associated value maxval in Y;
The maximum supported integer, maxint;
Partition factor ℓ.
Input: Target set $T = \{\mathsf{key}_1, \ldots, t_n\}$
Output: Solution $S = \{(\mathsf{key}, \mathsf{val}) \in Y : \mathsf{key} \in T\}$

```
01 queue ← {[n]}
02 S ← ∅
03
04 while queue ≠ ∅ do
05     A ← dequeue.queue
06     Partition A into equal-sized sets A_1, ..., A_ℓ
07     d ← #digits(⌈|A|/ℓ⌉ · maxval)
08
09     // Define adversarial input set and run PJC.
10     X = {(key_j, 10^{d·(i−1)}) : key_j ∈ T ∧ j ∈ A_i ∧ i ∈ [ℓ]}
11     b ← PJC(X, Y)
12
13     // Extract the ℓ inner products.
14     Prepend "0"s to b until #digits(b) = ℓ · d
15     Parse b into d-digit integers a_1, ..., a_ℓ
16
17     // Derive values and add them to solution.
18     for i ∈ [ℓ] do
19         if |A_i| = 1 and a_i ≠ 0 then
20             // The target key is in Y.
21             j ← A_i[1]
22             S ← S ∪ {(key_j, a_i)}
23         else if |A_i| > 1 and a_i ≠ 0 then
24             // Queue set A_j to explore.
25             queue.enqueue(A_i)
26         end if
27     end for
28 end while
29 return S
```

Figure 1: The Search Tree attack against PJC.

Theorem 4.4. *The Search Tree Attack (Figure 1) achieves associated value recovery, i.e., given a target set T and a secret database Y, it outputs the database $S = \{(\mathsf{key}, \mathsf{val}) \in Y : \mathsf{key} \in T\}$.*

The proof can be found in Appendix B.3. We describe how to select the branching factor (ℓ) in Appendix B.2.

4.3 Query Bounds

Interestingly, the number of queries we must make is independent of $|Y|$ – a property shared by all of our attacks – as well as of $|X|$. We formalize this observation by proving lower and upper bounds on the number of queries the search tree attack must make to achieve KVR.

Theorem 4.5. *Let T be a target set of size n, Y be the secret set, and k be their intersection size. The Search Tree Attack (Figure 1) requires $\Omega(k/\ell + \log_\ell(n/k))$ queries to $\mathtt{PJC}(\cdot, Y)$ for associated value recovery, where ℓ is the partition size.*

The proof can be found in Appendix B.4.

Theorem 4.6. *Let T be a target set of size n, Y be the secret set, and k be their intersection size. The Search Tree Attack (Figure 1) requires $O(k \log_\ell n)$ queries to $\mathtt{PJC}(\cdot, Y)$ for associated value recovery, where ℓ is the partition size.*

The upperbound follows from the fact that, in the worst case, the adversary must make $\log_\ell n$ queries per $\mathsf{key} \in T$ that is also in the recovery set (i.e., it must issue a query for each inner node in the search tree along the path from the root to the leaf corresponding to key).

5 The Maximum Likelihood Attack

Each PJC protocol invocation yields an inner product b_i, computed from secret $\mathbf{s}$ and row vector $\mathbf{a}_i$ in query matrix A. This forms a linear constraint on the secret $\mathbf{s}$. After q invocations, we have q linear constraints, resulting in an underdetermined linear system. In an underdetermined linear system, there are infinite solutions, making exact recovery impossible. Therefore, we aim for approximate recovery, which is sufficient to demonstrate the privacy risks, such as estimating salary or age ranges.

We can achieve approximate recovery using maximum likelihood estimation (MLE), assuming some prior knowledge of real-world distributions. MLE is a standard technique in statistical inference. Unlike the typical usage of estimating distribution parameters, our application of MLE is to output the most likely guess about $\mathbf{s}$, denoted as $\tilde{\mathbf{s}}$, based on prior information about its distribution.

Maximum Likelihood. We provide a quick recap of the maximum likelihood method. Consider a vector of independent random variables $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ over some probability space. Let vector $\mathbf{x}$ denote the observed outcome of $\mathbf{X}$, and let θ denote the parameter vector of the distribution, such as the mean and variance. We now consider the conditional probability $p(\mathbf{x}|\theta)$. By definition, it is the probability of observing the outcome $\mathbf{x}$ given the distribution parameter vector θ. In comparison, MLE considers the reverse: after observing the outcome $\mathbf{x}$, estimate the parameter vector θ. Since defining the inverse of probability is not always possible, a likelihood function $\mathcal{L}$ is introduced. It is a function of θ by treating the outcome $\mathbf{x}$ as invariant. In MLE, it suffices to define $\mathcal{L}(\theta|\mathbf{x}) := p(\mathbf{x}|\theta)$.[2]

[2] In the general case, we define $\mathcal{L}(\theta|\mathbf{x}) \propto p(\mathbf{x}|\theta)$.

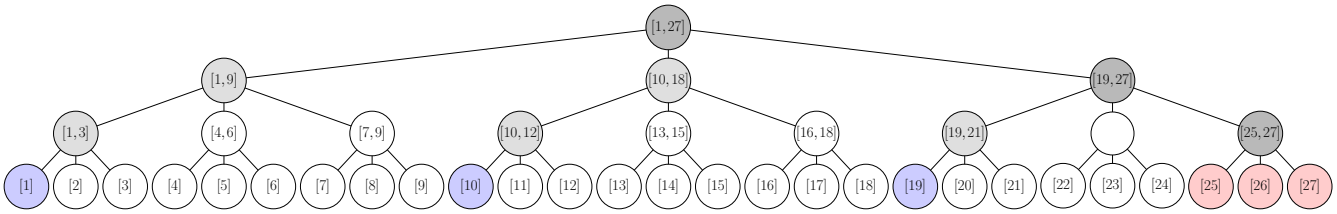


Figure 2: A search tree for a target set size $|T| = 27$, partition size $\ell = 3$, and intersection size $k = 3$. Nodes are labeled with indices of target set keys (for convenience, the partitions comprise consecutive indices). Nodes [1], [10], [19] depict a worst-case arrangement requiring 7 queries. Nodes [25], [26], [27] depict a best-case, requiring only 3 queries.

MLE computes an estimate of θ, denoted by $\hat{\theta}_{\text{MLE}}$, which maximizes the likelihood function,

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\hat{\theta}} \mathcal{L}(\hat{\theta}|\mathbf{x}).$$

Intuitively, this makes sense, as it also maximizes the probability of observing $\mathbf{x}$ given θ.

Because $(X_i)_{i\in[n]}$ are independent random variables, we can expand $\mathcal{L}(\theta|\mathbf{x})$ as follows, $\mathcal{L}(\theta|\mathbf{x}) := p(\mathbf{x}|\theta) = \prod_{i=1}^{n} p(x_i|\theta)$.

Since the likelihood function is monotonically increasing, we can use the log-likelihood function, denoted by $\ell(\theta|\mathbf{x})$. This simplifies the computation by converting the product into a summation,

$$\ell(\theta|\mathbf{x}) = \sum_{i=1}^{n} \ln(p(x_i|\theta)).$$

So far, we have described the standard MLE technique used in statistical inference. We need to adapt it to our setting for recovering the secret $\mathbf{s}$. Unlike the typical application, we are not interested in estimating the distribution parameters θ; we assume they are known as auxiliary information. Instead, we are given a (potentially infinite) set of solutions to the underdetermined linear system, and we aim to output $\tilde{\mathbf{s}}$ that maximizes the likelihood. Hence, for a set of solutions $\{\mathbf{s}_j\}_{j\in\mathbb{N}}$, we define the following likelihood function, with the index of the solution as a parameter to estimate, $\mathcal{L}(j|\mathbf{s}_j,\theta) := p(\mathbf{s}_j|j,\theta)$.

Similarly, we define the log-likelihood function, $\ell(j|\mathbf{s}_j,\theta) := \ln(p(\mathbf{s}_j|j,\theta))$. We output the guess $\tilde{\mathbf{s}} = \mathbf{s}_j$, the j-th solution that maximizes the log likelihood,

$$\begin{aligned} j &= \arg\max_{j} \ell(j|\mathbf{s}_j,\theta) \\ &= \arg\max_{j} \sum_{i=1}^{n} \ln(p(s_j^i|j,\theta)), \quad \mathbf{s}_j = (s_j^1,\ldots,s_j^n)^T. \end{aligned} \tag{4}$$

To help with understanding, let us consider the following example. Given a set of solution $\{\mathbf{s}_j\}_{j\in\mathbb{N}}$, each solution $\mathbf{s}_j$ is a random vector consisting of n independent random variables. Each random variable s_j^i, for all $i \in [n]$, follows the same Gaussian distribution with mean μ and variance σ^2. This is written as $s_j^i \sim \mathcal{N}(\mu,\sigma^2)$. This assumption aligns with the fact that the values of the secret vector describe the same attribute.

Example 5.1 (Secret following a multivariate Gaussian distribution)**.** *Let the secret vector* $\mathbf{s}$ *be drawn from a multivariate Gaussian distribution, with independent random variable* $s_i \sim \mathcal{N}(\mu,\sigma^2)$ *for all* $i \in [n]$*. Let* minval *and* maxval *be the lowerbound and upperbound for each* s_i*, respectively. For convenience, we define* $\boldsymbol{\mu} = (\mu_1,\mu_2,\ldots,\mu_n)^T$*, with* $\mu_1 = \mu_2 = \cdots = \mu_n = \mu$*. After* q *queries, we have a coefficient matrix (i.e., query matrix)* $A \in \mathbb{R}^{q\times n}$ *and an inner product vector* $\mathbf{b} \in \mathbb{R}^q$ *such that* $A\mathbf{s} = \mathbf{b}$*.*

We can formalize the approximate recovery of $\mathbf{s}$*, denoted by* $\tilde{\mathbf{s}}$*, as a linear program,*

$$\begin{aligned} \text{minimize } (\tilde{\mathbf{s}}-\boldsymbol{\mu})^T(\tilde{\mathbf{s}}-\boldsymbol{\mu}) \qquad & \text{subject to} \\ \mathsf{minval} \le \tilde{s_i} \le \mathsf{maxval} \qquad & \textit{and} \\ A\tilde{\mathbf{s}} = \mathbf{b}. \qquad & \end{aligned}$$

The constraints are straightforward. It is left to derive the objective function: $\min\ (\tilde{\mathbf{s}}-\boldsymbol{\mu})^T(\tilde{\mathbf{s}}-\boldsymbol{\mu})$*. This is the maximum likelihood part, which aims to find a solution concentrated around the mean.*

Recall the probability density function (pdf) of $\mathcal{N}(\mu,\sigma^2)$*,*[3] $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$*.*

Plugging in the parameter vector $\theta = (\mu,\sigma^2)$ *into the log-likelihood function in Equation (4), we obtain* $\ell(j|\mathbf{s}_j,\mu,\sigma^2) = \sum_{i=1}^{n} \ln(p(s_j^i|j,\mu,\sigma^2))$*. Expanding its inner term* $\ln(p(s_j^i|j,\mu,\sigma^2))$*,*

$$\ln\left(p(s_j^i|j,\mu,\sigma^2)\right) = \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(s_j^i-\mu)^2}{2\sigma^2}.$$

It follows that, the log-likelihood function is

$$\sum_{i=1}^{n} \ln(p(s_j^i|j,\mu,\sigma^2)) = n\cdot\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{n} \frac{(s_j^i-\mu)^2}{2\sigma^2}.$$

Since the term $n\cdot\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$ *is fixed, to maximize the log-likelihood function, it suffices to maximize*

[3] We use the continuous version for illustration purposes.

$-\sum_{i=1}^{n} \frac{(s_j^i-\mu)^2}{2\sigma^2}$. *Considering* σ^2 *is fixed and the minus sign, we need to minimize* $\sum_{i=1}^{n}(s_j^i-\mu)^2$. *Expressing this as an objective function in matrix form and substituting the indexed* $\mathbf{s}_j$ *with* $\tilde{\mathbf{s}}$ *gives us* $\min\ (\tilde{\mathbf{s}}-\boldsymbol{\mu})^T(\tilde{\mathbf{s}}-\boldsymbol{\mu})$.

6 Sparse Recovery Attacks

Quite surprisingly, in an underdetermined linear system, if we assume the secret or solution is sparse[4] and the coefficient matrix A has a certain property, we can recover the secret exactly! This problem was first rigorously studied by Candès and Tao [11], leading to the development of an area called *compressed sensing*. These techniques are exceptionally powerful because they do not require any prior knowledge about the distribution of the secret $\mathbf{s}$. Additionally, even if the inner products $\mathbf{b}$ are perturbed with noise, these methods remain effective. To distinguish this problem setting from the previous discussion on non-sparse recovery using MLE, we will refer to it as *sparse recovery*.

6.1 Compressed Sensing (CS)

We now discuss how to adapt compressed sensing techniques to both *noiseless* and *noisy* sparse recovery, considering *exact* and *approximate* recovery goals. As this section focuses on sparse recovery, we first define k-sparsity.

Defining k-sparsity. A vector of length n is called k-sparse if it has *at most* k non-zero coordinates, where $k \in [n]$. For simplicity, let us assume for now that we know in advance that the secret is k-sparse. Even if this condition is not met, the techniques we use can still ensure approximate recovery. This is captured by the *stability* property, which we will discuss later.

Uniqueness of k-sparse solutions. Given $A\mathbf{s} = \mathbf{b}$, if $\mathbf{s}$ is k-sparse and A satisfies a certain property, then this k-sparse solution is unique. This property of A was initially termed the *uniform uncertainty principle (UUP)* by Candès and Tao [11], but it is now generally referred to as the *restricted isometry property (RIP)*.

Because the k-sparse solutions are unique, it suffices to find the sparsest solution to the linear system. We can reformulate this as an optimization problem: given $A\mathbf{s} = \mathbf{b}$ as the constraint on $\mathbf{s}$, the objective is to find the sparsest solution. This is equivalent to minimizing the ℓ_0 norm, which represents the number of non-zero coordinates in the solution.

Minimizing the ℓ_1 Norm. Unfortunately, minimizing the ℓ_0 norm is an NP-hard problem that requires a brute-force search. Thankfully, the seminal work by Candès and Tao [11] shows that using the ℓ_1 norm for minimization is sufficient under the restricted isometry property of A. Since the ℓ_1 norm is convex, standard optimization methods can be applied, allowing the problem to be solved using linear programming. Candès and Tao [11] also extend this approach to noisy recovery.

Regarding utilizing compressed sensing techniques in cryptanalysis, the most relevant work is by Bootle et al. [7]. They propose side-channel attacks against the LWE-based signature scheme BLISS by exploiting noisy inner products without modular operations. Their aim is to recover the exact secret. If we also assume that A is sub-Gaussian,[5] as in their work, and aim for exact recovery, their results directly apply. Their approach relies on an estimator called the Dantzig selector. We discuss this approach in detail later.

In addition, we consider a different restriction on A, specifically constraining it to be a binary matrix. This is to derive a non-adaptive attack against PSI-CA and PSI-SUM. Since the adversary no longer has access to the associated values to craft A, we show that we can transform this binary matrix to determine which elements to include in each protocol invocation, ensuring that the compressed sensing techniques still apply.

Due to the binary matrix restriction, we need to resort to other techniques in compressed sensing, specifically those based on the ℓ_1 norm.

Restricted Isometry Property (RIP). We now describe the high-level idea of the restricted isometry property (RIP) and its use in compressed sensing. Suppose we have a matrix A of dimension $q \times n$ and $A\mathbf{s} = \mathbf{b}$ holds. Intuitively, if A is an identity matrix with $q = n$, we can recover $\mathbf{s}$ coordinate by coordinate, as each inner product with a unit vector gives us a coordinate of $\mathbf{s}$. However, for an underdetermined system A with fewer rows than columns (i.e., $q < n$), if the secret $\mathbf{s}$ has only non-zero coordinates, exact recovery is impossible.

In comparison, we can achieve much better results with a k-sparse secret $\mathbf{s}$. If we know which coordinates are non-zero, we can store those values. However, without such information, we need a condition on A to ensure that the aforementioned linear program still applies.

To provide some intuition, consider $A\mathbf{s} = \mathbf{b}$, where A is a $q \times n$-dimensional matrix. We can think of A as a projection matrix that maps $\mathbf{s}$ from a high-dimensional space (n-dimensional) to $\mathbf{b}$ in a lower-dimensional space (q-dimensional). At the same time, this projection preserves the ℓ_p norm, such that $\|\mathbf{b}\|_p \approx \|\mathbf{s}\|_p$. Hence, the matrix A helps with identifying the non-zero positions of $\mathbf{s}$, allowing us to recover k non-zero values using only

[4] By a sparse secret, we mean that there are many zeros and at most k non-zero coordinates in a vector of length n. Here, k is the sparsity.

[5] This type of probability distributions has a strong tail decay, with their tails dominated by those of a Gaussian distribution.

q inner products. These desired properties are captured by *restricted isometry property (RIP)*.

We provide a formal definition (Definition 6.1) under general p norms. In our use cases, $p \in \{1,2\}$ are sufficient; we refer to the RIP with ℓ_1 and ℓ_2 norms as RIP-1 and RIP-2, respectively.

Definition 6.1 (Restricted Isometry Property ($\text{RIP}_{p,k,\delta}$) [4])**.** *A $q \times n$ matrix A is said to satisfy $\text{RIP}_{p,k,\delta}$ if for all k-sparse vector $\mathbf{s}$, we have*

$$\|\mathbf{s}\|_p \leq \|A\mathbf{s}\|_p \leq (1+\delta)\|\mathbf{s}\|_p.$$

As mentioned earlier, Berinde et al. [4] show that the adjacency matrix of an unbalanced expander graph satisfies RIP-1. We provide their theorem here for reference (cf. Theorem 6.2). In particular, for bipartite graph $G = (U,V,E)$, we set $|U| = n$ and $|V| = q$, for $q < n$.

Theorem 6.2 (Constructing A with $\text{RIP}_{1,k,\delta}$ using an unbalanced expander graph [4])**.** *Consider any $q \times n$ matrix A that is the adjacency matrix of an (k,ϵ)-unbalanced expander $G = (U,V,E)$ with left degree d, such that $1/\epsilon, d$ are smaller than n. Then the scaled matrix A/d satisfies the $\text{RIP}_{1,k,\delta}$ property and $\delta = 2\epsilon$.*

Now it is left to describe the approach for generating an unbalanced bipartite graph. We use a simple randomized approach. We provide the pseudocode in Algorithm 1. This algorithm enables us to generate a high-quality unbalanced expander graph with left degree d.

We present the Theorem 6.3 for the RIP-1 property of the generated matrix A using Algorithm 1.

Theorem 6.3 (A is RIP-1 with high probability)**.** *The scaled matrix A/d returned by* GenRIPMatrix(n,q,d,ϵ) *in Algorithm 1 satisfies RIP-1 with a high probability.*

Theorem 6.3 follows from Lemma 6.4 and Theorem 6.2.

Lemma 6.4 ($G = (U,V,E)$ is a (k,ϵ)-unbalanced expander graph with a high probability)**.** *The adjacency matrix A of G returned by* GenRIPMatrix(n,q,d,ϵ) *in Algorithm 1 satisfies (k,ϵ) with a high probability for $q = dke^{1/\epsilon-1}$ and $d \geq \frac{1}{\epsilon}\log_{1-\epsilon}(\frac{k}{10ne})$.*

We provide the proof for Lemma 6.4 in Appendix C.1.

Consider the case for $q = dke^{1/\epsilon-1}$, we can simplify the term as $\frac{ne}{k}(1-\epsilon)^{\epsilon d}$. Ensuring this term $\leq \frac{1}{10}$ implies $d \geq \frac{1}{\epsilon}\log_{1-\epsilon}(\frac{k}{10ne})$.

Since $\frac{x}{1-x}$ is monotonically increasing for $|x| < 1$, we set parameter d to ensure x is within $(0, \frac{1}{10}]$, so that the failure probability is at most $\frac{1}{9}$. For example, when $n = 10^3, k = 10, \epsilon = 0.49$, and $d = 24$, we obtain $q \approx 680$. In the experimental evaluation (Section 7.2), we demonstrate that the parameters can be set more efficiently than the suggested theoretical values. For instance, with $n = 10^3$, $k = 10$, and $d = 66$, setting $q = 133$ is sufficient for exact recovery (see Table 2).

Algorithm 1 Generate $\text{RIP}_{1,k,\delta}$ matrix A using randomized unbalanced expander graph $G = (U,V,E)$.

Input: $q, n, d \in \mathbb{N}$ and $\epsilon \in (0,1)$
Output: $A \in \{0,1\}^{q \times n}$

```
function GenRIPMatrix(n, q, d, ε)
    // G = (U, V, E); adjacency list adj defines E
    U ← {u_1, u_2, ..., u_n}
    V ← {v_1, v_2, ..., v_q}
    // Initialize adjacency list adj.
    adj ← {}
    for all u ∈ U do
        adj(u) ← []
    end for
    for all u ∈ U do
        // Random sample without replacement
        nbrs ← RandomSample(V, d)
        adj(u) ← nbrs
    end for
    // For all i ∈ [q], all j ∈ [n],
    // a_{i,j} = 1 if v_i, u_j are neighbors; 0, otherwise.
    Set A using adj
    return A
end function
```

Algorithm 2 Compressed sensing under the ℓ_1 or ℓ_2 norm for exact recovery from $A\mathbf{s} = \mathbf{b}$.

Input: $A \in \{0,1\}^{q \times n}$, minval, maxval
Output: $\tilde{\mathbf{s}}$

```
Initialize an LP solver with the following,
minimize ∑_{i=1}^{n} u_i   subject to
    −u_i ≤ s̃_i ≤ u_i,
    minval ≤ s̃_i ≤ maxval,   and
    As̃ = b.
LP solver outputs s̃.
return s̃
```

Exact / Approximate Recovery with CS-ℓ_1. We provide the recovery theorem [4] for CS-ℓ_1 (Algorithm 2) in Theorem C.1 for completeness. Note that if we know or can roughly estimate an upper bound for k, exact recovery can be achieved using this theorem. An asymptotic bound on the number of queries needed for CS-ℓ_1 to achieve exact (approximate) recovery [4] is as follows.

Theorem 6.5 (Bound on the number of queries)**.** *With a random (scaled) binary matrix that satisfies RIP-1, using CS-ℓ_1 to recover a k-sparse secret of dimension n exactly requires $q = \Theta(k\log(n/k))$ PJC queries.*

In our experiments (Section 7.2) we use the concrete parameter $q = 2k\log_2(n/k)$, rounded to the nearest integer.

Against PSI-CA / PSI-SUM. Let us now consider how to adapt our CS-ℓ_1 attack to PSI-CA or PSI-SUM.

In this scenario, the adversary controls only the key set and no longer has access to the associated values. The function's output is either the cardinality or the sum. However, we can still convert this problem into a special case of our PJC inner-product problem.

Given a target set $T = \{\mathsf{key}_j\}_{j\in[n]}$ of size n, we can generate the random binary $q \times n$ matrix A that satisfies RIP-1. Scaling is unnecessary, as we can simply divide $\mathbf{b}$ by d to achieve the same effect. For each i-th query, we use the row $\mathbf{a}_i$ to decide which keys in T to include in the input. For example, if $\mathbf{a}_i = (a_{i,j})_{j\in[n]}$, then $a_{i,j} = 1$ indicates that key_j should be included in the set, and 0 indicates exclusion. In this way, the output cardinality or sum behaves the same as the inner product. Therefore, using CS-ℓ_1, we can achieve key recovery (i.e., membership recovery) for PSI-CA and PSI-SUM. We can also achieve KVR with the PSI-SUM. We provide the pseudocode in Algorithm 4.

On Stability. In practice, it is sometimes impossible to know the sparsity or the intersection size beforehand. For example, the intersection size is not revealed by the PJC with inner products [29], meaning that we can only have a rough estimate based on some prior knowledge. We refer to this uncertainty as a *sparsity defect*. In addition, each inner product may be added with noise for leakage mitigation. If the recovery method is robust to both the sparsity defects and noises, we call it *stable recovery*.

Compressed sensing is a stable sparse recovery method. If neither is a concern, then $q = 2k$ protocol runs are sufficient for exact recovery. This is achieved through discrete fourier transform we describe in Section 6.2.

6.2 Discrete Fourier Transform (DFT)

Learning k-sparsity from Intersection Size. When the intersection size k is known, it takes only $2k$ queries to exactly recover the secret vector $\mathbf{s}$. This is achieved using Discrete Fourier Transform (DFT), which also has a practical recovery method. Specifically, we populate the entries in matrix A so that each inner product corresponds to a discrete Fourier coefficient of the secret $\mathbf{s}$. Then for exact recovery, only $2k$ such measurements are required.

Due to space limit, we provide a high-level overview of this approach, followed by a theorem that formally states the main result. For a detailed proof and an in-depth explanation, we refer readers to an excellent textbook on compressed sensing [18]. Here is a brief outline of the approach: the DFT measurements of $\mathbf{s}$ help identify the indices of the secret vector that contain non-zero values, known as the *support*. Once the support is identified, the secret can be recovered by solving a $2k$ overdetermined linear system derived from the DFT coefficients. We refer to Appendix D and book [18] for more details.

6.3 On Robustness to Noise

So far, we have only considered the cases with *noiseless* inner product outputs. In a *noisy* setting, compared with DFT, compressed sensing methods are also robust to noise. Specifically, we can achieve approximate recovery in both CS-ℓ_2 and CS-ℓ_1; under specific restriction on the norm difference, we can achieve exact recovery for CS-ℓ_2 [7]. We refer to the theorems [4], which are provided in the Appendix (Theorems C.1 and C.3) for completeness.

Noisy recovery is achieved by introducing a tunable regularizer λ. We let $\mathbf{r}$ denote the residual of the linear system, where $\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{s}}$. In particular, the regularizer is operated on the infinity norm (i.e., the largest entry) of $A^T\mathbf{r}$.

Algorithm 3 Compressed sensing under the ℓ_1 norm for approximate recovery from $A\mathbf{s} = \mathbf{b} + (noise)$.

Input: $A \in \{0,1\}^{q\times n}$, λ, minval, maxval
Output: $\tilde{\mathbf{s}}$

Initialize an LP solver with the following,
minimize $\sum_{i=1}^{n} u_i$ subject to
  $-u_i \le \tilde{s}_i \le u_i$,
  $\mathsf{minval} \le \tilde{s}_i \le \mathsf{maxval}$, and
  // $[A\tilde{\mathbf{s}} - \mathbf{b}]_i$ denotes the i-th entry of the residual vector
  $-\lambda \le [A\tilde{\mathbf{s}} - \mathbf{b}]_i \le \lambda$.
LP solver outputs $\tilde{\mathbf{s}}$.
return $\tilde{\mathbf{s}}$

Compressed Sensing under ℓ_2 Norm with Dantzig Selector (CS-ℓ_2). In addition to the RIP-1, we can also use other techniques for A satisfying RIP-2. Specifically, we use the Dantzig selector, proposed by Candès and Tao [11]. The core idea is to minimize the ℓ_1 norm for a sparse secret with a regularizer λ on the residual term $\mathbf{r}$. In this case, matrix A is constructed by drawing each entry from a mean-zero Gaussian distribution.[6] We adapt the result from [7], providing pseudocode in Algorithm 5. This also assumes the added noise in the inner products follows a mean-zero Gaussian distribution.

7 Experimental Evaluation

Experimental Setup. We implemented our attacks using Python 3.12.4 and ran our experiments on a computing cluster with 2U Rackmount Chassis, 64 Core AMD EPYC 7742 2.25GHz Processor and 512GB Server Memory. We simulated the output of the PJC functionality on the same compute node as the node we ran

[6] We restrict it to be a mean-zero Gaussian for simplicity. This approach also applies to sub-Gaussian distributions.

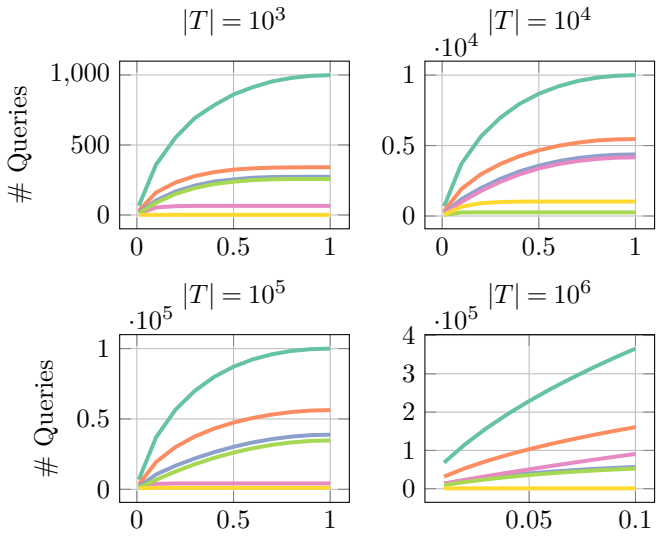


Figure 3: The number of queries required by our Search Tree Attack, plotted against the intersection ratio ρ.

the attack on, as such our attack times do not include network latency. We report the attack times excluding the time for computing the PJC functionality. In all of our experiments, we used synthetic data, as most of the methods, except for MLE, are data-independent.

7.1 Combinatorial Attack

Datasets. For the synthetic data we fixed the size of the secret set Y to be $100,000$ and varied the size of the target set T across 10^3, 10^4, 10^5 and 10^6. We first sampled keys in Y from the universe $[|Y|+|T|]$ and sampled integral associated values from the range $[0,1000]$, i.e., $\mathsf{minval} = 0$ and $\mathsf{maxval} = 1000$. Let k be the intersection size and ρ be the ratio of keys in the intersection with respect to $|T|$, i.e., $\rho n = k$. For $|T| \in \{10^3, 10^4, 10^5\}$, we considered $\rho \in [0.1, 1.0]$ (with a step size of 0.1). For $|T| = 10^6$, we considered $\rho \in [0.01, 0.1]$ (with a step size of 0.01). For each value of ρ and $|T| = n$, we sampled ρn keys from Y_{keys} and then sampled the remaining values from $[|Y|+|T|] \setminus Y_{keys}$, where Y_{keys} denotes the keys of Y.

Results. We ran the attack for partition sizes $\ell \in \{2^1, 2^2, 2^4, 2^6, 2^8, 2^{10}\}$. For each combination of $|T|$, ρ, and ℓ, we ran the attack 10 times and averaged the number of queries needed for KVR and the attack time. The query and timing results can be found in Figures 3 and 7, respectively. For all datasets, increasing the value of ρ resulted in more queries needed for recovery. Correspondingly, the increase in the number of queries that had to be made also resulted in an increase in attack run time. Run time was overall very efficient, ranging from a few milliseconds ($n = 10^3$) to 90s ($n = 10^6$).

In general, increasing the partition size resulted in fewer queries. However, we note some exceptions. For $|T| = 10^3$, $\ell = 2^6$ resulted in fewer queries than $\ell = 2^8$ for all values of ρ; when $\ell = 2^{10}$, we have that $\ell \geq |T|$, and so running the search tree attack collapses to the basic attack, (all values are recovered with exactly one query).

Similar behavior is observed for $|T| = 10^4$ and $|T| = 10^5$. This is because increasing the partition size, may result in a finer partition with *more* sets that must be consequently queried. In Figure 6, we give such an example for $|T| = 10^4$ and (6a) $\ell = 100$ or (6b) $\ell = 1000$. Both parameters result in search trees of height 2; in both cases, since ℓ is larger than the resulting sets at depth 1 of the search tree, we can recover any value by querying at most 2 sets containing its corresponding key in the target set. However, the smaller partition size results in on-average fewer sets with a non-zero inner product that that must be queued and later explored (100 sets versus 1000 sets in the worst-case).

This suggests an alternative strategy in which the partition size is chosen adaptively. We conjecture that choosing the partition size adaptively will not result in a significant improvement.

n	k	Type	q	ℓ_1 loss	Time (ms)
10^2	10	DFT	20	**0.00**	0.94
		CS-ℓ_1	66	0.00	14.10
		CS-ℓ_2	66	0.00	13.98
	20	DFT	40	**0.00**	3.30
		CS-ℓ_1	93	0.00	15.26
		CS-ℓ_2	93	0.00	15.97
	100	MLE	90	101.53	13.21
		MLE	95	**89.17**	13.66
10^3	10	DFT	20	**0.00**	0.94
		CS-ℓ_1	133	0.00	371.89
		CS-ℓ_2	133	0.00	456.20
	100	DFT	200	6.89	757.05
		CS-ℓ_1	664	0.00	5197.40
		CS-ℓ_2	664	**0.00**	3025.00
	200	DFT	400	12.74	13095.62
		CS-ℓ_1	929	0.00	9980.16
		CS-ℓ_2	929	**0.00**	6775.83
	1000	MLE	900	104.27	715.65
		MLE	950	**78.31**	705.58
10^4	1000	DFT	2000	10.03	40018.49
		CS-ℓ_1	6644	0.00	3367613.93
		CS-ℓ_2	6644	**0.00**	2995573.02
	10000	MLE	9500	**73.93**	612962.17

Table 2: Value range $[-1000, 1000]$.

7.2 MLE, DFT and Compressed Sensing

In this section, we present the experimental evaluation of noiseless and noisy recovery using maximum likelihood estimation (MLE), discrete Fourier transform (DFT), and compressed sensing (CS). We implemented these

recovery methods using the Gurobi optimization tool.[7] As discussed in Section 6.1, for compressed sensing under the ℓ_1-norm, we generated the query matrix A using the unbalanced expander graph approach (Algorithm 1) and implemented Algorithm 2 for noiseless recovery and Algorithm 3 for noisy recovery.

For compressed sensing under the ℓ_2-norm, we construct the query matrix A by drawing individual entries from a normal distribution, using the same recovery procedure as in Algorithm 2. For noisy recovery, we implemented the recovery procedure with the Dantzig selector (Algorithm 5).

Datasets. The target set T was selected with sizes $\{10^2, 10^3, 10^4\}$. Since the keys can be selected arbitrarily without affecting our evaluation, we set them to $[|T|]$. To construct the recovery set Y, we did not restrict its size, as our focus is on the elements of Y that lie within the intersection. For simplicity, we defined the key universe as $[|T|]$, and based on the varying intersection size k, we select randomly k elements from $[|T|]$ to server as the keys in Y that lie in the intersection. Their associated values were selected uniformly at random from the integer ranges $[-100, 100]$ and $[-1000, 1000]$.

It is evident that with a uniform random distribution, MLE cannot perform better than a random guess. However, in real-world datasets, a normal distribution (or other non-uniform distributions) is more likely than a uniform distribution. Therefore, to demonstrate the effectiveness of MLE, we selected the associated values from a normal distribution. Specifically, we used $\mathcal{N}(\mu = 0, \sigma = 50)$ for the range $[-100, 100]$ and $\mathcal{N}(\mu = 0, \sigma = 500)$ for $[-1000, 1000]$, both with $\sigma = (\mathsf{maxval} - \mathsf{minval})/4$ to ensure a balanced distribution across the value range.

(Adversarial) Input from $\tilde{P}_1$. In each protocol run, the associated values for the target set are adjusted according to the specific recovery method employed. In particular, the input is artificially crafted using DFT, simulating an active but non-adaptive adversarial attack. For MLE and compressed sensing under the ℓ_1 and ℓ_2 norms, the input is randomly drawn from a distribution. This simulates a non-adaptive adversarial setting, which may be useful for modeling real-world scenarios. For example, training a linear model with random (weighted) feature selection on a data set comprising the associated values of Y that lie in the intersection. In the case of the ℓ_1 norm, $\tilde{P}_1$'s input is a binary selection vector, whereas for the ℓ_2 norm, $\tilde{P}_2$'s input can be drawn from a normal distribution. Our experiments restrict the associated values of T to within a specific range ($[-10, 10]$ for $[-100, 100]$ and $[-100, 100]$ for $[-1000, 1000]$, with the values being floats). It is important to note that the effectiveness of compressed sensing under the ℓ_2 norm with the Dantzig selector requires the mean of the normal distribution to be zero.

[7] https://www.gurobi.com/

n	k	Type	q	ℓ_1 loss	Time (ms)
10^2	10	DFT	20	27.31	0.94
		CS-ℓ_1	66	5.37	17.54
		CS-ℓ_2	66	**0.00**	535.44
	20	DFT	40	10.80	3.26
		CS-ℓ_1	93	9.47	21.81
		CS-ℓ_2	93	**0.00**	726.85
	100	MLE	90	124.70	14.79
		MLE	95	**63.63**	15.21
10^3	10	DFT	20	4.56	0.93
		CS-ℓ_1	133	0.56	301.10
		CS-ℓ_2	133	**0.00**	94232.49
	100	DFT	200	22.38	1855.94
		CS-ℓ_1	664	1.90	1146.95
		CS-ℓ_2	664	**0.00**	515692.35
	200	DFT	400	30.68	11418.77
		CS-ℓ_1	929	2.88	1693.23
		CS-ℓ_2	929	**0.00**	738450.12
	1000	MLE	900	105.21	904.05
		MLE	950	**72.58**	965.71
10^4	1000	DFT	2000	23.73	289838.10
		CS-ℓ_1	6644	**0.59**	543835.42
	10000	MLE	9500	**76.79**	673098.80

Table 3: Value range $[-1000, 1000]$, with noise from $\mathcal{N}(\mu = 0, \sigma = 5)$, clipped at magnitude 100.

Adding Noise. To demonstrate that compressed sensing is robust to noise, we added noise to each inner product, drawn from a normal distribution with mean $\mu_e = 0$ and standard deviations σ_e in $\{2.5, 5\}$. Note that adding noise to the inner products makes the recovery problem significantly harder. In this case, even if the number of queries equals the number of targets, Gaussian elimination cannot produce a solution!

Results and Observations. We present the results from five runs, with statistics for the value range $[-1000, 1000]$ shown in Tables 2 and 3, and for the value range $[-100, 100]$ in Tables 4 and 5 in the Appendix. For each value of n and k, we have highlighted the best performing method to help with interpreting our results. The best performing method was selected based on the following criteria in order: average ℓ_1 loss, q (the number of queries needed) and the recovery time.

We now discuss our results for each method.

MLE. We focus on non-sparse recovery using MLE, by setting the intersection size equal to the target set. While this approach can be extended to accommodate sparse cases by modeling a more complex distribution, our experiments are aimed at evaluating the effectiveness of MLE. Thus, we concentrate on the simpler case described in Example 5.1. We observed that even after $0.95n$ queries, there is still some ℓ_1 loss. Specifically, the ℓ_1 loss is around 7 for the value range $[-100, 100]$ (Ta-

bles 5 and 4). This loss increases significantly when the value range is expanded to $[-1000, 1000]$ (Tables 2 and 3). This increase is expected, as the solution space becomes much larger, making it considerably more challenging to find a solution close to the true values. Nonetheless, this still poses a risk if the value range is small and approximate recovery is a concern, as may be the case for some database entries.

In the noisy setting, we relax the linear constraints and bound the residue term by the noise magnitude, using the same technique as in Line 6 in Algorithm 3. We found that, adding noise does not make a significant difference in the losses. This is likely because MLE is a randomized method that outputs the most likely solution within a large solution space, making it less sensitive to small noise perturbations.

DFT. Our results indicate that DFT recovery performs very well in certain experiments. Since it only requires solving linear equations, it consumes significantly less time compared with the other methods. However, it is unstable in some cases and not robust to noise. Even small perturbations from the noise can result in relatively large losses, which is consistent with the theoretical limitations of the method.

Compressed Sensing. In the noiseless setting, both CS-ℓ_1 and CS-ℓ_2 ensure exact recovery for intersection ratios up to 0.2 with high probability. CS-ℓ_1 is more scalable than CS-ℓ_2; for the latter, we were only able to scale up to $n = 2$K using Gurobi. For noisy recovery, both methods show robustness to noise with $\sigma_e \in \{2.5, 5\}$. While CS-ℓ_1 incurs a small loss, exact recovery remains possible with CS-ℓ_2. However, at larger noise scales (e.g., $\sigma \geq 10$), we observe minor ℓ_1 losses in recovery accuracy for CS-ℓ_2. Compared with DFT, this robustness comes with a tradeoff in the number of queries q, as indicated by the bound $\Theta(k \log(n/k))$ in Theorem 6.5. It is worth noting that in our experiments, we use a multiplicative constant $c = 2$, resulting in $q = 2k \log_2(n/k)$, rounded to the nearest integer. Figure 8 illustrates the number of queries needed with different parameters.

7.3 Discussion

Attack Comparison. From our experimental evaluation, we found that the search tree attack out-performs the other attacks with respect to number of queries, attack time, and scalability. For example, we were able to run it up to $n = 10^6$. The other three attacks utilize solvers and, in particular, MLE and CS rely on solving an optimization problem which inherently limits their scalability. We note, however, that the search tree attack is an adaptive attack that only works in a noiseless setting and when the associated values are non-negative. The remaining attacks address limitations in the tree attack, with all three capable of handling negative values, and CS showing robustness to noise. Moreover, MLE and CS fall under the non-adaptive adversarial setting.

Google PJC Code Repository. Google acknowledges in their PJC code repository [20] potential information leakage from functionality outputs, such as membership leakage due to large associated values. However, these privacy risks remain unclear. Our work takes a crucial step towards analyzing these leakages as suggested and highlighting real-world risks.

All four attacks operate under honest protocol execution, with some requiring malicious inputs, and demonstrate how a weak adversary could infer information about the other party's input. Although the "caveats" listed [20] mention potential risks from malicious inputs, they neither quantify the information that could be inferred nor specify the number of queries or types of inputs required to extract meaningful information.

Our attacks formalize the extent to which this leakage can be exploited. We demonstrate that these attacks are feasible for a more generalized key-value recovery (KVR), rather than being limited to membership recovery. Notably, two attacks (MLE, compressed sensing) do not rely on malicious inputs and align with the security model outlined outlined in [20].

8 Countermeasures and Future Work

In this section, we discuss the PJC deployment, explore techniques for mitigating our attacks, and provide avenues for future work.

PJC Deployment. Although PJC with inner-product may be still in its early stages of deployment, PSI-SUM (a special case of PJC with inner-product) has already been deployed by Google [25] for ad conversion. In this scenario, the advertiser inputs a table of (personal id, spending) pairs, and the ad publisher (e.g., Google) inputs a set of personal ids associated with ad clickers. PSI-SUM allows the advertiser to compute aggregate conversion revenue and determine the number of shared personal ids. This application has been analyzed in prior work [24, 27]. Our compressed-sensing and MLE attacks, when adapted to PSI-SUM, can recover personal ids and compromise the anonymity of the ad clickers, aligning with the adversarial goals in [24, 27]. Moreover, our focus on the generalized PJC with inner-product [29] aims to identify and address potential risks proactively, such as the privacy of associated values, ensuring they are either mitigated or effectively communicated before full-scale deployment.

Input Validation. Adversarially chosen inputs can be detrimental to the other party's privacy and, in fact, "Scrubbing inputs to remove identifiers with outlier values" was already recommended by [20] for their PSI-SUM protocol. One approach for preventing outlier values is to use zero knowledge proofs (ZKPs). ZKPs enable a party to commit to their input and later prove that the committed values satisfy a certain property, e.g., are within a particular range. For example, Bulletproofs [10] enables one to prove that a single committed value is in an n-bit range using a proof of size $O(\log_2 n)$, with the possibile optimization of batching m proofs at the expense of an additive $O(\log_2 m)$ factor.

Bell et al. [3] presented a protocol for secure aggregation with input validation by using [10] to prove in ZK that the input satisfies a given ℓ_∞ bound. One of the challenges of input validation for PSI-like protocols is designing a protocol whose inputs are compatible with either existing ZKP systems or tailor-made ones and this is an interesting line of future work.

Authorized PSI (APSI). Another technique is to authorize the inputs using a trusted third-party prior to computating on the intersection. The goal is to only have approved elements appear in the intersection; elements that are in the interesection of the partys' sets but aren't authorized will not contribute to the output. Authorized PSI (e.g., [13,14,35]) and PSI-CA (e.g., [12]) have already been proposed. Most recently, Falzon and Markatou presented an APSI scheme based on bilinear pairings [16]. APSI is especially useful in settings such as ad-conversion where an auditing firm can act as the trusted third-party and verify the purchases claimed by a company before the intersection is computed.

Extending authorization to PSI-like functionalities such as PJC is a non-trivial problem and seeing if the techniques proposed in [16] can be applied to the Diffie-Hellman based PSI-SUM protocol of Ion et al. [25] is an interesting open problem. Scrubbing inputs via authorization or input validation are useful for defending against our search-tree attack. However they are not sufficient alone, since our non-adaptive attacks can still be carried out. We, thus, recommend that the above approaches be used with other mitigation techniques.

Adding Noise. Our compressed sensing attack (Section 6.1) demonstrated that KVR is still feasible when the noise added to the output is sufficiently small (e.g., normally distributed with $\sigma_e \leq 5$). We conjecture that adding large-scale noise could limit the ability to infer associated values precisely. Therefore, if only exact KVR is a concern, introducing noise may serve as an effective mitigation strategy.

As we have already pointed out, MLE and DFT are susceptible to noise for exact recovery, so adding noise helps mitigate these two attacks. Regarding the search tree attack, which may adapt to noisy setting, and we leave this exploration for future work.

For a provable guarantee, one might consider using differentially-private (DP) noise. However, we conjecture that DP may not necessarily be suitable in this context, as it is most effective with large datasets for aggregate statistics, whereas PJC may operate on single records. Considering the inner-product functionality, the challenge is to add noise in a way that preserves utility while mitigating our attacks. This remains an intersting open problem for future investigation.

Rate Limiting and Query Budgets. Rate limiting is a technique that restricts the frequency of protocol invocations. Similarly, a query budget is a parameter that specifies how many queries can be issued.

Our theoretical results demonstrate that our attacks' success hinges on a certain number of queries being made For example our search tree attack requires a number of queries linear in k (Theorem 4.5), and our compressed sensing attack with the ℓ_1 norm requires $\Theta(k\log(n/k))$ queries (Theorem 6.5). Stronger lower bounds on the queries needed for *any* attack against PJC to succeed could provide a guideline for the maximum allowed number of queries. One interesting future direction would be to investigate the trade-off between number of queries issued and the minimum loss for any KVR attack.

Acknowledgement

Francesca Falzon is supported by Armasuisse Science and Technology. Tianxin Tang is supported by an NWO VIDI grant (Project No. VI.Vidi.193.066). The authors would like to thank the reviewers and shepherd for their helpful comments and Kenneth G. Paterson for his feedback on the introduction.

Ethics Considerations

This work is compliant with the Usenix ethics guidelines. Our work aligns with the intended use of PJC and therefore does not expose any vulnerabilities of the protocol or underlying cryptographic primitives.

Use of Synthetic Data. Our experiments were conducted exclusively using synthetic data generated by sampling numerical values from uniform and normal distributions. The use of synthetic data ensures that no real personal information was utilized or exposed, thereby eliminating any risks of harm to individuals or groups.

Disclosure. We reached out to the authors of [29] (PJC with inner-product) and the PJC repository maintainers [20] on 19.11.2024 and 22.11.2024, respectively, to

inform them about our analysis and suggest including information about our attacks in the README of the code repository [20]; they agreed to discuss the results internally and have updated their README[8] to include links to the prior attack works and additional information about the possible dangers of allowing multiple protocol invocations on the same/similar input set.

Open Science

This work is compliant with the Open Science Policy. We make the source code for the experimental evaluation available at https://zenodo.org/records/14643963. This includes code for all four of our attacks (the tree, MLE, DFT, and compressed sensing attacks) and the script for generating the synthetic data. As attested to in the submission form, we will also undergo the artifact evaluation process. In particular, we will provide an artifact appendix to show how to run our code and reproduce our results.

References

[1] G. Asharov, K. Hamada, R. Kikuchi, A. Nof, B. Pinkas, and J. Tomida. Secure statistical analysis on multiple datasets: Join and group-by. In W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, editors, *ACM CCS 2023*, pages 3298–3312. ACM Press, Nov. 2023. doi:10.1145/3576915.3623119.

[2] S. Badrinarayanan, S. Das, G. Garimella, S. Raghuraman, and P. Rindal. Secret-shared joins with multiplicity from aggregation trees. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *ACM CCS 2022*, pages 209–222. ACM Press, Nov. 2022. doi:10.1145/3548606.3560670.

[3] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. ACORN: Input validation for secure aggregation. In J. A. Calandrino and C. Troncoso, editors, *USENIX Security 2023*, pages 4805–4822. USENIX Association, Aug. 2023.

[4] R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. J. Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 798–805, Sept. 2008. doi:10.1109/ALLERTON.2008.4797639.

[5] A. Bhowmick, D. Boneh, S. Myers, K. Talwar, and K. Tarbe. The Apple PSI System. Technical report, Apple Inc., 2021.

[6] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. The tao of inference in privacy-protected databases. *Proc. VLDB Endow.*, 11(11):1715–1728, jul 2018. doi:10.14778/3236187.3236217.

[7] J. Bootle, C. Delaplace, T. Espitau, P.-A. Fouque, and M. Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 494–524. Springer, Cham, Dec. 2018. doi:10.1007/978-3-030-03326-2_17.

[8] P. Buddhavarapu, B. M. Case, L. Gore, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and M. Xue. Multi-key private matching for compute. *IACR Cryptol. ePrint Arch.*, page 770, 2021. URL: https://eprint.iacr.org/2021/770.

[9] P. Buddhavarapu, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and V. Vlaskin. Private matching for compute. Cryptology ePrint Archive, Report 2020/599, 2020. URL: https://eprint.iacr.org/2020/599.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00020.

[11] E. Candès and T. Tao. The Dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, Dec. 2007. doi:10.1214/009053606000001523.

[12] E. D. Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In J. Pieprzyk, A. Sadeghi, and M. Manulis, editors, *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, volume 7712, pages 218–231. Springer, 2012. doi:10.1007/978-3-642-35404-5_17.

[13] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 213–231, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-17373-8_13.

[8] https://github.com/google/private-join-and-compute/commit/c0cb3f5b5b616caaaef80b7f7e9c335f4fe53ce7

[14] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In R. Sion, editor, *Financial Cryptography and Data Security*, pages 143–159, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-14577-3_13.

[15] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1155–1166. ACM Press, Oct. 2016. doi:10.1145/2976749.2978379.

[16] F. Falzon and E. A. Markatou. Re-visiting authorized private set intersection: A new privacy-preserving variant and two protocols. *Proc. Priv. Enhancing Technol.*, 2025(1):792–807, 2025. URL: https://doi.org/10.56553/popets-2025-0041, doi:10.56553/POPETS-2025-0041.

[17] F. Falzon, E. A. Markatou, Akshima, D. Cash, A. Rivkin, J. Stern, and R. Tamassia. Full database reconstruction in two dimensions. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 443–460. ACM Press, Nov. 2020. doi:10.1145/3372297.3417275.

[18] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Springer New York, New York, NY, 2013. doi:10.1007/978-0-8176-4948-7.

[19] Google. Helping organizations do more without collecting more data. https://security.googleblog.com/2019/06/helping-organizations-do-more-without-collecting-more-data.html. Accessed: 2024-08-30.

[20] Google. private-join-and-compute. https://github.com/google/private-join-and-compute/blob/master/README.md, 2022.

[21] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 315–331. ACM Press, Oct. 2018. doi:10.1145/3243734.3243864.

[22] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy*, pages 655–672. IEEE Computer Society Press, May 2017. doi:10.1109/SP.2017.44.

[23] Z. Gui, K. G. Paterson, and T. Tang. Security analysis of MongoDB queryable encryption. In J. A. Calandrino and C. Troncoso, editors, *USENIX Security 2023*, pages 7445–7462. USENIX Association, Aug. 2023.

[24] X. Guo, Y. Han, Z. Liu, D. Wang, Y. Jia, and J. Li. Birds of a feather flock together: How set bias helps to deanonymize you via revealed intersection sizes. In K. R. B. Butler and K. Thomas, editors, *USENIX Security 2022*, pages 1487–1504. USENIX Association, Aug. 2022.

[25] M. Ion, B. Kreuter, E. Nergiz, S. Patel, M. Raykova, S. Saxena, K. Seth, D. Shanahan, and M. Yung. Private intersection-sum protocols with applications to attributing aggregate ad conversions. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389, 2020. URL: https://eprint.iacr.org/2019/723.pdf.

[26] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, page 235–246, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2557547.2557561.

[27] B. Jiang, J. Du, and Q. Yan. AnonPSI: An anonymity assessment framework for PSI. In *NDSS 2024*. The Internet Society, Feb. / Mar. 2024.

[28] M.-S. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *Proceedings of the IEEE Symposium on Security and Privacy 2018*, S&P 2018, 2018.

[29] T. Lepoint, S. Patel, M. Raykova, K. Seth, and N. Trieu. Private join and compute from PIR with default. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 605–634. Springer, Cham, Dec. 2021. doi:10.1007/978-3-030-92075-3_21.

[30] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978. doi:10.1109/TIT.1978.1055927.

[31] P. Mohassel, P. Rindal, and M. Rosulek. Fast database joins and PSI for secret shared data. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1271–1287. ACM Press, Nov. 2020. doi:10.1145/3372297.3423358.

[32] D. Mouris, D. Masny, N. Trieu, S. Sengupta, P. Buddhavarapu, and B. Case. Delegated private matching for compute. Cryptology ePrint Archive, Report 2023/012, 2023. URL: https://eprint.iacr.org/2023/012.

[33] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 644–655. ACM Press, Oct. 2015. doi:10.1145/2810103.2813651.

[34] S. Oya and F. Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In M. Bailey and R. Greenstadt, editors, *USENIX Security 2021*, pages 127–142. USENIX Association, Aug. 2021.

[35] E. Stefanov, E. Shi, and D. Song. Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *Public Key Cryptography – PKC 2012*, pages 413–430, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-30057-8_25.

[36] Y. Yang, J. Weng, Y. Yi, C. Dong, L. Y. Zhang, and J. Zhou. Predicate private set intersection with linear complexity. In M. Tibouchi and X. Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 143–166. Springer, Cham, June 2023. doi:10.1007/978-3-031-33491-7_6.

[37] M. Zinkus, Y. Cao, and M. D. Green. McFIL: Model counting functionality-inherent leakage. In J. A. Calandrino and C. Troncoso, editors, *USENIX Security 2023*, pages 7001–7018. USENIX Association, Aug. 2023.

A PJC Supplementary Content

We use $\overset{c}{\equiv}$ to denote the relation between two distributions that are computationally indistinguishable. Let f be a PPT functionality computed by parties P_1 and P_2, defined as $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, and we can write $f = (f_1, f_2)$. Each of the parties, P_1 and P_2, has respective input $x, y \in \{0,1\}^*$. P_1 and P_2 compute f on (x,y), and f outputs a random variable $(f_1(x,y), f_2(x,y))$ over the pairs of strings. Let π be a two-party protocol that computes f. Let $\kappa \in \mathbb{N}$ denote the security parameter.

A.1 PSI-SUM and PSI-CA

The **Private Set Intersection Sum (PSI-SUM)** functionality takes as input a set, $X = \{\mathsf{key}_i\}_{i\in[n]}$, from P_1 and a database, $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in[m]}$, from P_2. It outputs the sum of values whose keys lie in the intersection:

$$\boxed{\sum_{i\in[n],j\in[m]} \mathbb{1}\{\mathsf{key}_i = \mathsf{key}'_j\} \cdot \mathsf{val}'_j.} \tag{5}$$

PSI-SUM is equivalent to running PJC when the associated values in X equal to 1.

The **Private Set Intersection Cardinality (PSI-CA)** functionality takes as input two sets of keys, $X = \{\mathsf{key}_i\}_{i\in[n]}$ and $Y = \{\mathsf{key}'_j\}_{j\in[m]}$, from P_1 and P_2, respectively. It outputs the intersection size:

$$\boxed{\sum_{i\in[n],j\in[m]} \mathbb{1}\{\mathsf{key}_i = \mathsf{key}'_j\}.} \tag{6}$$

PSI-CA is equivalent to running PJC when all associated values in the two databases equal 1.

A.2 Security

MPC functionalities such as PJC are often proven secure using simulation based security. This approach considers two worlds: (1) a *real world* in which the actual protocol is executed and the parties interact directly with each other, and (2) an *ideal world* in which the computation is mediated by a trusted third party. An protocol is secure if the two worlds are indistinguishable. In the case of PJC, this means that P_1 should only learn the inner product (Equation 3) and P_2 should learn nothing.

For party P_i, where $i \in \{1,2\}$, let $\mathsf{view}_i^\pi(x,y,\kappa)$ denote the view of P_i during the execution of two-party protocol π computing on (x,y,κ). Specifically, $\mathsf{view}_i^\pi(x,y,\kappa) := (w, r^i; m_1^i, m_2^i, \ldots, m_t^i)$, where $w \in \{x,y\}$ depending on the index i; r^i equals the contents of P_i's internal random tape; $m_1^i, m_2^i, \ldots, m_t^i$ are the messages P_i received. The output, output_i^π, is what P_i can compute based on its own view. The joint output, output^π is defined as $(\mathsf{output}_1^\pi(x,y,\kappa), \mathsf{output}_2^\pi(x,y,\kappa))$.

Definition A.1 (Static Semi-Honest Security)**.** *Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in the presence of static semi-honest adversaries if there exist PPT algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ s.t.*

$$\begin{aligned}
&\{(\mathcal{S}_1(1^\kappa, x, f_1(x,y)), f(x,y))\}_{x,y,\kappa} \\
&\overset{c}{\equiv} \{(\mathsf{view}_1^\pi(x,y,\kappa), \mathsf{output}^\pi(x,y,\kappa))\}_{x,y,\kappa}, \quad \textit{and} \\
&\{(\mathcal{S}_2(1^\kappa, y, f_2(x,y)), f(x,y))\}_{x,y,\kappa} \\
&\overset{c}{\equiv} \{(\mathsf{view}_2^\pi(x,y,\kappa), \mathsf{output}^\pi(x,y,\kappa))\}_{x,y,\kappa},
\end{aligned}$$

where $x, y \in \{0,1\}^*$ such that $|x| = |y|$, and $\kappa \in \mathbb{N}$.

PARAMETERS: Set sizes $n = |X|$ and $m = |Y|$.
FUNCTIONALITY:

- Wait for input $X = \{(\mathsf{key}_i, \mathsf{val}_i)\}_{i\in[n]}$ from receiver P_1.
- Wait for input $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j,)\}_{j\in[m]}$ from sender P_2.
- Give P_1 the inner product $\sum_{\substack{i\in[n],j\in[m]\\ \mathsf{key}_i=\mathsf{key}'_j}} \mathsf{val}_i \cdot \mathsf{val}'_j$.

Figure 4: The PJC ideal functionality.

B Search Tree Attack Appendix

B.1 Basic Attack Pseudocode

The pseudocode for the basic attack can be found in Figure 5.

Params: The Minimum possible associated value $\mathsf{minval} = 0$ and the maximum possible associated value $\mathsf{maxval} > 0$.
Input: Target set $T = \{\mathsf{key}_1, \ldots, \mathsf{key}_n\}$.
Output: Solution $S = \{(\mathsf{key}, \mathsf{val}) \in Y : \mathsf{key} \in T\}$.

```
01 d ← #digits(maxval)
02
03 // Compute adversarial database.
04 X ← {(key_j, 10^{d·(j−1)}) : j ∈ [n]}
05 b ← PJC(X, Y)
06
07 // Recover secret values of keys in the intersection.
08 Prepend "0"s to b until #digits(b)n · d
09 Parse b into d digit integers ans_1, ..., ans_n
10 S ← {(key_j, ans_j) : j ∈ [n] ∧ ans_j ≠ 0}
11 return S
```

Figure 5: The basic attack against PJC.

B.2 Selecting the Partition Factor

The primary criteria we require ℓ to satisfy is that the resulting inner-product does not exceed maxint. This should hold regardless of how the target set is partitioned into ℓ equal-sized sets. For a given partition factor $\hat{\ell} \in [n]$, we would have $\hat{\ell}$ index sets each with $\lceil \frac{n}{\hat{\ell}} \rceil$ items. The maximum possible associated value in Y is maxval and, so, in order to obtain a domain separation and construct the ℓ inner products, the associated values assigned to the sets in the partition must be of the form $10^{d(i-1)}$ for $i \in [\ell]$ and $d = \#\mathsf{digits}(\lceil \frac{n}{\ell} \rceil \cdot \mathsf{maxval})$. If d was any smaller than this, than we would risk not being able to separate the resulting inner product into the ℓ distinct ones.

As a strict upper bound, we thus require that

$$\ell \leq \max \left\{ \hat{\ell} \in [n] \;\middle|\; \begin{array}{l} d = \#\mathsf{digits}(\lceil \frac{n}{\hat{\ell}} \rceil \cdot \mathsf{maxval}) \quad \wedge \\ \\ \mathsf{maxint} \geq \mathsf{maxval} \cdot \lceil \frac{n}{\hat{\ell}} \rceil \cdot \sum_{i=1}^{\hat{\ell}} 10^{d\cdot(i-1)} \end{array} \right\}.$$

In our implementation (Section 7), we demonstrate our attack for a several partition factors ranging from $\ell = 2$ to $\ell = 2^{10}$.

B.3 Proof of Theorem 4.4

We first introduce the following generalization of superincreasing. Let $X = \{(\mathsf{key}_j, \mathsf{val}_j)\}_{j\in[n]}$ and $Y = \{(\mathsf{key}'_j, \mathsf{val}'_j)\}_{j\in[m]}$ be databases. Let $A = [n]$ and $A_1, \ldots, A_\ell$ be a partition of A. **The inner product of X and Y is superincreasing with respect to** $A_1, \ldots, A_\ell$, if for all $1 < i \leq \ell$ such that A_i contains an index of a key in Y, we have

$$\sum_{j\in A_i} \mathbb{1}\{\mathsf{key}_j \text{ in } Y\} \cdot \mathsf{val}_j \cdot Y[\mathsf{key}_j] > \sum_{\substack{j\in A_k \\ 1\leq k<i}} \mathbb{1}\{\mathsf{key}_j \text{ in } Y\} \cdot \mathsf{val}_j \cdot Y[\mathsf{key}_j].$$

Proof. Let $\mathsf{key}_j \in T$ be any key in the intersection and $j \in [n]$ be its index. To show that $(\mathsf{key}_j, \mathsf{val}_j) \in S$ such that $(\mathsf{key}_j, \mathsf{val}_j) \in Y$, we must prove two things: (1) for any enqueued set containing j, either we enqueue a strictly smaller set containing j or we stop and (2) the stopping condition within the **if** clause on line 19 is correct.

The attack starts by enqueuing the set $[n]$ (line 1), and thus $\mathsf{queue} \neq \emptyset$ and the **while** loop is executed.

To see (1), let A be a dequeued set containing j (line 5) and $\mathcal{P} = \{A_1, \ldots, A_\ell\}$ be its partition (line 6). Then there exists some $i^* \in [\ell]$ such that $j \in A_{i^*}$. Let X be defined as in line 10, and $d = \#\mathsf{digits}(\lceil |A|/\ell \rceil \cdot \mathsf{maxval})$, and observe that the inner product of X and Y is superincreasing with respect to $\mathcal{P}$. In particular,

- The output of $\mathtt{PJC}(X', Y)$ is strictly larger than that of $\mathtt{PJC}(X'', Y)$ where

  $$X'\{(\mathsf{key}_j, 10^{d\cdot(i-1)}) : \mathsf{key}_j \in T \wedge j \in A_{i^*}\} \quad \text{and}$$

  $$X'' = \{(\mathsf{key}_j, 10^{d\cdot(i-1)}) : \mathsf{key}_j \in T \wedge j \in A_i \wedge i \in [i^* - 1]\}.$$

  Moreover, the first $d \cdot (i^* - 1)$ least-significant digits of the output of $\mathtt{PJC}(X', Y)$ are 0.

- The output of $\mathtt{PJC}(X', Y)$ is strictly smaller than that of $\mathtt{PJC}(X''', Y)$ where

  $$X''' = \{(\mathsf{key}_j, 10^{d\cdot(i-1)}) : \mathsf{key}_j \in T \wedge j \in A_i \wedge i \in [i^* + 1, \ell]\}$$

  or $\mathtt{PJC}(X''', Y) = 0$. Moreover, if $\mathtt{PJC}(X''', Y) \neq 0$, then the first $d \cdot i^*$ least-significant digits of the inner product are 0.

Thus the value a_{i^*} (defined on line 15) precisely equals the output of $\texttt{PJC}(\widehat{X}, Y)$, where

$$\widehat{X} = \{(\mathsf{key}_j, 1) : \mathsf{key}_j \in T \wedge j \in A_{i^*}\}.$$

By assumption, key_j is in Y and so $a_{i^*} \neq 0$. If $|A_{i^*}| > 1$, then the **else if** clause (line 23) is executed and A_{i^*}, which is strictly smaller than A and contains j is enqueued. If $|A_{i^*}| = 1$, then a key-value pair is added S (we prove the correctness of this key-value pair below).

We now show (2). Since $[n]$ is finite and the invariant (1) holds, then we know that for some dequeued set A and its partition $\mathcal{P} = \{A_1, \ldots, A_\ell\}$ (line 6) there exists some $i^* \in [\ell]$ such that $|A_{i^*}| = 1$ and $j \in A_{i^*}$. By virtue of the fact that A_{i^*} is a singleton and that the inner product of X and Y is superincreasing with respect to $\mathcal{P}$, then a_{i^*} is inner product of $\{(\mathsf{key}_j, 1)\}$ and Y. This implies that $(\mathsf{key}_j, a_{i^*}) \in Y$. By assumption, key_j is in Y and so $a_{i^*} \neq 0$. The **if** clause is executed, thus adding $(\mathsf{key}_j, a_{i^*})$ to S.

Since $[n]$ is finite and, in each iteration, the sets enqueued are strictly smaller than the set whose partition it comprises, there is a finite number of sets enqueued/dequeued. Thus, the **while** loop must be exited and the attack terminates.

Now suppose that $\mathsf{key}_j \in T$ is not in the intersection. Due to invariant (1) above, there exists a set A whose partition comprises of a set A_{i^*} such that for all $i \in A_{i^*}$, key_i is not in Y. So none of its keys are in the intersection, then its corresponding inner product is $a_{i^*} = 0$. A_{i^*} is never added to the queue (none of its descendents in the search tree are explored). Moreover, if $|A_{i^*}| = 1$, then key_j is not added to the solution. □

Corollary B.1. *Let T be the target set, Y be the secret set, and k be their intersection size. Let $\mathcal{T}$ be the ℓ-ary search tree resulting from running the Search Tree Attack (Figure 1). Let v_{key} denote the leaf associated with key. For every $\mathsf{key} \in T$ contained in the intersection, the adversary must issue the query corresponding to each node from the root to the parent of v_{key} in $\mathcal{T}$.*

B.4 Proof of Theorem 4.5

We first prove the following lemma.

Lemma B.2. *Let T be a target set of size n, Y be the secret set, and k be their intersection size. Let $\mathcal{T}$ be the ℓ-ary search tree resulting from running the Search Tree Attack (Figure 1). Let v_{key} denote the leaf associated with key. For every $\mathsf{key} \in T$ contained in the intersection, the adversary must issue the query corresponding to each node from the root to the parent of v_{key} in $\mathcal{T}$.*

Each inner node node in $\mathcal{T}$ corresponds to a subset $A \subseteq [n]$, and the ℓ children of that node correspond to a partition of A. Let $\mathsf{key}_j \in T$ be in the intersection and $j \in [n]$ be its index. Since $\mathcal{T}$ induces a partition on $[n]$, then there exists a path in $\mathcal{T}$ whose nodes correspond to the sequence of sets $A_i^{(j)}$, $i \in [\log_\ell n]$, each containing the index j.

Each set in the queue results in exactly one query. We proceed by induction on i. For $i = 1$, we have $A_i^{(j)} = [n]$, which corresponds to the root of $\mathcal{T}$. $A_i^{(j)}$ is enqueued in line 1 and thus results in a query.

Suppose now that for $i = m < \log_\ell n$, it holds that $A_m^{(j)}$ is enqueued. We want to show that $A_{m+1}^{(j)}$ must also be enqueued. Since $A = A_m^{(j)}$ was enqueued then, by correctness of the attack, A must be dequeued. When dequeued, A is then partitioned into ℓ sets (line 6), one of which contains j, i.e., $A_{m+1}^{(j)} \subsetneq A$. Let $a^{(j)}$ denote the inner product of keys indexed by $A_{m+1}^{(j)}$ (line 15). Since key_j is in Y and has a non-zero associated value, then any database containing key_j that is queried results in a non-zero inner product. In particular, $a^{(j)} \neq 0$. If $|A_{m+1}^{(j)}|$ then the value is recovered. Otherwise, $|A_{m+1}^{(j)}|$ is enqueued (line 23) and thus results in a query. □

Proof. The optimal scenario is that for each PJC query issued, the maximum possible number of sets in the partition (the A_i's line 6) result in an inner product (the a_i's in line 15) of 0. This corresponds to minimizing the number of sets that are enqueued (lines 23–25). This occurs when the keys in the intersection are indexed by the fewest possible sets of the partition, i.e., the leaves of the search tree that index the keys in the intersection lie in the smallest possible subtree.

Let $\mathcal{T}$ denote the ℓ-ary search tree for T. The fewest queries are needed when all k indices of the intersection lie in the smallest possible subtree. Let v denote the lowest common ancestor of the k intersection indices. By Corollary B.2, to achieve KVR, the adversary must issue all queries corresponding to the nodes from the root of the search tree to v and the queries corresponding to the inner-nodes of the sub-tree rooted at v.

Let $\lceil x \rceil_\ell$ denote rounding x to the next power of ℓ. There are $\frac{\ell \cdot \lceil k \rceil_\ell - 1}{\ell - 1}$ nodes in the subtree of $\mathcal{T}$ induced by v and its descendents , and $\frac{\ell \cdot \lceil k \rceil_\ell - 1}{\ell - 1} - \lceil k \rceil_\ell$ inner nodes. There are an additional $\log_\ell n - \log_\ell \lceil k \rceil_\ell = \log_\ell (n / \lceil k \rceil_\ell)$ nodes in the path from the root to v.

In the worst case, when $k = \lceil k \rceil_\ell$, the adversary must issue every query associated with the inner nodes of the subtree induced by v and its descendents. Thus the adversary must issue at least

$$\Omega\left(\frac{k}{\ell} + \log_\ell \frac{n}{k}\right) = \frac{\ell k - 1}{\ell - 1} - k + \log_\ell \frac{n}{k}$$

queries to exactly recover the associated values. □

C Compressed Sensing Appendix

C.1 Proof of Lemma 6.4

Proof. Let X be an arbitrary subset of U, $X \subseteq U$ with size $|X| = s \leq k$. Let M be an arbitrary subset of V of size $|M|$. There are at most $d \cdot s$ vertices in the neighbor set $N(X)$. For the case $N(X) \subseteq M$, if we consider each vertex selection is independent, then we have that $\Pr(N(X) \subseteq M) = \left(\frac{|M|}{m}\right)^{ds}$. Because we require random sampling without replacement, then the vertex selection is no longer independent, and thus gives us an upper bound.

$$\Pr(N(X) \subseteq M) \leq \left(\frac{|M|}{m}\right)^{ds}.$$

We now upperbound the probability that the event G is not a (k,ϵ) expander. Applying the union bound yields

$$\begin{aligned}
&\Pr(G \text{ is not a } (k,\epsilon)\text{-expander}) \\
&\leq \sum_{\substack{X\subseteq U \\ |X|=s\leq k}} \sum_{\substack{M\subseteq V \\ |M|<\alpha s}} \Pr(N(X)\subseteq M) \quad \text{let } \alpha = (1-\epsilon)d \\
&\leq \sum_{\substack{X\subseteq U \\ |X|=s\leq k}} \sum_{\substack{M\subseteq V \\ |M|=\alpha s}} \Pr(N(X)\subseteq M) \\
&= \sum_{s=1}^{k} \sum_{\substack{X\subseteq U \\ |X|=s}} \sum_{\substack{M\subseteq V \\ |M|=\alpha s}} \Pr(N(X)\subseteq M) \\
&\leq \sum_{s=1}^{k} \binom{n}{s}\binom{m}{\alpha s}\left(\frac{\alpha s}{m}\right)^{ds} \\
&\leq \sum_{s=1}^{k} \left(\frac{ne}{s}\right)^{s}\left(\frac{me}{\alpha s}\right)^{\alpha s}\left(\frac{\alpha s}{m}\right)^{ds} \\
&\leq \sum_{s=1}^{k} \left[\left(\frac{ne}{s}\right)\left(\frac{me}{\alpha s}\right)^{\alpha}\left(\frac{\alpha s}{m}\right)^{d}\right]^{s} \\
&\leq \sum_{s=1}^{k} x^{s}, \quad \text{where } x = \left(\frac{ne}{s}\right)\left(\frac{me}{\alpha s}\right)^{\alpha}\left(\frac{\alpha s}{m}\right)^{d} \\
&= \frac{x}{1-x} \text{ for } |x| < 1.
\end{aligned}$$

To bound x,

$$\begin{aligned}
x &= \left(\frac{ne}{s}\right)\left(\frac{me}{\alpha s}\right)^{\alpha}\left(\frac{\alpha s}{m}\right)^{d} \\
&= \left(\frac{ne}{s}\right)\left(\frac{me}{\alpha s}\right)^{(1-\epsilon)d}\left(\frac{\alpha s}{m}\right)^{d} \quad \text{since } \alpha = (1-\epsilon)d \\
&= \left(\frac{ne}{s}\right)\left(\frac{me}{\alpha s}\cdot\frac{\alpha s}{m}\right)^{d}\left(\frac{\alpha s}{me}\right)^{\epsilon d} \\
&= \left(\frac{ne}{s}\right)e^{d}\left(\frac{\alpha s}{me}\right)^{\epsilon d} \\
&= \left(\frac{ne}{s}\right)\left(\frac{\alpha s e^{1/\epsilon-1}}{m}\right)^{\epsilon d} \\
&\leq \left(\frac{ne}{k}\right)\left(\frac{(1-\epsilon)dke^{1/\epsilon-1}}{m}\right)^{\epsilon d} \quad s\leq k, \alpha = (1-\epsilon)d.
\end{aligned}$$

□

Algorithm 4 CS-ℓ_1 attack against PSI-CA or PSI-SUM.

Input: $T = (t_1, t_2, \ldots, t_n)$, type $\in$ {PSI-CA, PSI-SUM}, minval, maxval
Output: $\tilde{\mathbf{s}}$

```
function RunPSI(type, X, Y)
    if PSI = PSI-CA then
        ca ← PSI-CA(X, Y)
        return (ca, ⊥)
    else
        (ca, sum) ← PSI-SUM(X, Y)
        return (ca, sum)
    end if
end function
X ← T
(ca, resp) ← RunPSI(type, X, Y)
k ← ca
Set q, d, ε based on n and k.
A ← GenRIPMatrix(n, q, d, ε) (cf. Algorithm 1)
for i from 1 to q do
    X ← ∅
    for j from 1 to n do
        if a_{i,j} = 1 then
            X ← X ∪ {t_j}
            (ca, resp) ← RunPSI(type, X, Y)
        end if
    end for
end for
Run LP solver on (cf. Algorithm 1)
return s̃
```

The following definition and remark is to demonstrate exact recovery for CS-ℓ_2.

Theorem C.1 ((Approx.) sparse recovery [4])**.** *Let A be a $q \times n$ matrix of an unbalanced $(2k,\epsilon)$-expander and $\alpha(\epsilon) = (2\epsilon)/(1-2\epsilon)$. Let $\mathbf{s}, \tilde{\mathbf{s}}$ be any two vectors such that for $\mathbf{y} = \mathbf{s} - \tilde{\mathbf{s}}$ we have $A\mathbf{y} = 0$, and $\|\tilde{\mathbf{s}}\|_1 \leq \|\mathbf{s}\|_1$. Let S be the set of k largest (in magnitude) coefficients of $\mathbf{s}$, then*

$$\|\tilde{\mathbf{s}} - \mathbf{s}\|_1 \leq 2/(1-2\alpha(\epsilon)) \cdot \|\mathbf{s} - \mathbf{s}_S\|_1.$$

The following definition and remark is to demonstrate exact recovery for CS-ℓ_2.

Definition C.2 (Alternative Definition for Restricted Isometry Property (RIP-2) [11])**.** *A $q \times n$ matrix A is*

said to satisfy RIP-2 if there exists a constant $\delta_k \in (0,1)$ *such that for all* k*-sparse vector* $\mathbf{x}$*,*

$$(1-\delta_k)\|\mathbf{x}\|_2^2 \le \|A\mathbf{x}\|_2^2 \le (1+\delta_k)\|\mathbf{x}\|_2^2.$$

Remark 1 (Exact recovery). Exact recovery is guaranteed provided $\delta_{2k} < 1$.

The following theorem is to demonstrate robustness to noise for CS-ℓ_1.

Theorem C.3 (Approx. sparse recovery [4])**.** *Let* A *be a* $q \times n$ *matrix of an unbalanced* $(2k,\epsilon)$*-expander. Let* $\alpha(\epsilon) = (2\epsilon)/(1-2\epsilon)$*. Consider any two vectors* $\mathbf{s}, \tilde{\mathbf{s}}$*, such that for* $\mathbf{y} = \mathbf{s} - \tilde{\mathbf{s}}$ *we have* $\|A\mathbf{y}\|_1 = \beta \ge 0$*, and* $\|\tilde{\mathbf{s}}\|_1 \le \|\mathbf{s}\|_1$*. Let* S *be the set of* k *largest (in magnitude) coefficients of* $\mathbf{s}$*. Let* S^c *denote the complement of* S*. Then*

$$\|\tilde{\mathbf{s}} - \mathbf{s}_S\|_1 \le 2/(1-2\alpha(\epsilon)) \cdot \|\mathbf{s}_{S^c}\|_1 + \frac{2\beta}{d(1-2\epsilon)(1-2\alpha)}.$$

Algorithm 5 Compressed sensing under the ℓ_2 norm for exact (approximate) recovery from $A\mathbf{s} = \mathbf{b} + (noise)$.

Input: $A \in \{0,1\}^{q \times n}$, minval, maxval, σ_a, σ_e
Output: $\tilde{\mathbf{s}}$

$\lambda \leftarrow \sigma_e \sigma_a \sqrt{2q \log n}$
Initialize an LP solver with the following,
minimize $\sum_{i=1}^{n} u_i$ subject to
  $-u_i \le \tilde{s}_i \le u_i$,
  minval $\le \tilde{s}_i \le$ maxval, and
  $-\lambda \le [A\tilde{\mathbf{s}} - \mathbf{b}]_i \le \lambda$.
LP solver outputs $\tilde{\mathbf{s}}$.
return $\tilde{\mathbf{s}}$

The parameters σ_a and σ_e denote the standard deviation of the distribution determining the entries of A of the noise, respectively.

D DFT

To avoid confusion with $\mathbf{s}$, we introduce x as its function representation, mapping $[n]$ to a complex set $\mathbb{C}$ or integer set

DFT. To avoid confusion with $\mathbf{s}$, we introduce x as its function representation, mapping the index set $[n]$ to a complex set $\mathbb{C}$ or integer set $\mathbb{N}$ (depending on the values). Each entry s_j can thus be represented by $x(j)$. The function x is supported on a set of size k (i.e., the sparsity), which we denote as S.

We now compute the first $2k$ Fourier coefficients of the sparse vector. To determine the support, we use a helper polynomial $p(t)$ of degree k, defined as follows.

$$p(t) := \frac{1}{N} \prod_{s \in S} (1 - e^{-2\pi i s/N} e^{2\pi i t/N}).$$

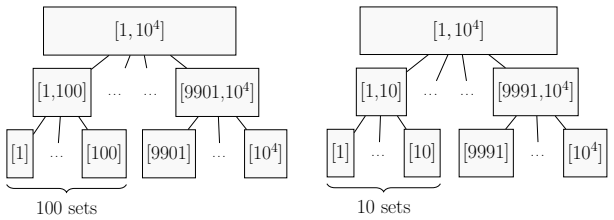
This polynomial has a special property: it evaluates to 0 for all $s \in S$. This allows us to identify the support by computing its roots. We use the first $2k$ Fourier coefficients of x to construct a system of linear equations derived from the inner products. This is complemented by the following observation: the x function evaluates to 0 in the complementary set $\bar{S}$. Therefore, the product $p(t) \cdot x(t) = 0$ for all $t \in [0, n-1]$. We can then solve the roots of polynomial p (i.e., the support) for $t \in [0, n-1]$ using discrete revolution. After identifying the support, since we have $2k$ DFT measurements for k variables, we can recover the secret by solving an overdetermined linear system. For more details, refer to [18] (pages 51-52).

The exact recovery of DFT for k-sparse secret is supported by the following theorem.

Theorem D.1 (Exact Recovery with DFT [18])**.** *For any* $n \ge 2k$*, there exists a practical procedure for the reconstruction of every* k*-sparse vector from its first* $m = 2k$ *discrete Fourier measurements.*

$$\hat{x}(j) := \sum_{s=0}^{n-1} x(u) e^{-2\pi i j s/n}, \quad 0 \le j \le n-1.$$

E More Experimental Results



(a) $\ell = 100$ (b) $\ell = 1000$

Figure 6: An example demonstrating why larger partition sizes (ℓ) may require more queries.

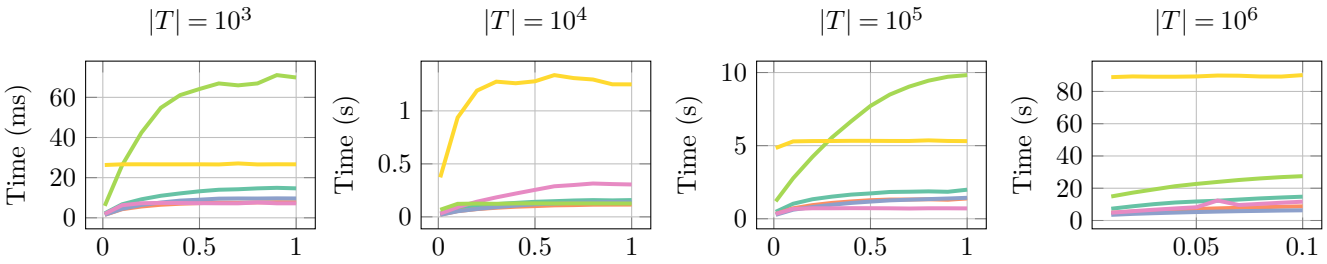


Figure 7: The time required by our Search Tree Attack, plotted against the intersection ratio ρ.

n	k	Type	q	ℓ_1 loss	Time (ms)
10^2	10	DFT	20	**0.00**	1.25
		CS-ℓ_1	66	0.00	14.31
		CS-ℓ_2	66	0.00	14.54
	20	DFT	40	**0.00**	4.02
		CS-ℓ_1	93	0.00	15.30
		CS-ℓ_2	93	0.00	16.76
	100	MLE	90	9.67	20.04
		MLE	95	**6.26**	13.77
10^3	10	DFT	20	**0.00**	0.94
		CS-ℓ_1	133	0.00	447.48
		CS-ℓ_2	133	0.00	424.43
	100	DFT	200	0.58	4855.06
		CS-ℓ_1	664	0.00	5487.19
		CS-ℓ_2	664	**0.00**	2962.98
	200	DFT	400	0.92	16591.96
		CS-ℓ_1	929	0.00	10454.34
		CS-ℓ_2	929	**0.00**	6434.73
	1000	MLE	900	11.12	608.58
		MLE	950	**7.46**	754.37

Table 4: Value range $[-100, 100]$.

n	k	Type	q	ℓ_1 loss	Time (ms)
10^2	10	DFT	20	2.95	1.05
		CS-ℓ_1	66	0.61	17.80
		CS-ℓ_2	66	**0.06**	538.24
	20	DFT	40	3.30	3.60
		CS-ℓ_1	93	0.90	22.85
		CS-ℓ_2	93	**0.13**	741.40
	100	MLE	90	10.27	14.38
		MLE	95	**7.51**	14.81
10^3	10	DFT	20	0.54	0.93
		CS-ℓ_1	133	0.05	269.91
		CS-ℓ_2	133	**0.00**	95802.05
	100	DFT	200	2.76	1537.81
		CS-ℓ_1	664	0.18	1188.24
		CS-ℓ_2	664	**0.00**	520225.09
	200	DFT	400	3.90	9333.13
		CS-ℓ_1	929	0.27	1856.18
		CS-ℓ_2	929	**0.00**	742324.14
	1000	MLE	900	10.89	835.96
		MLE	950	**6.88**	894.05

Table 5: Value range $[-100, 100]$, with noise from $\mathcal{N}(\mu = 0, \sigma = 2.5)$, clipped at magnitude 10.

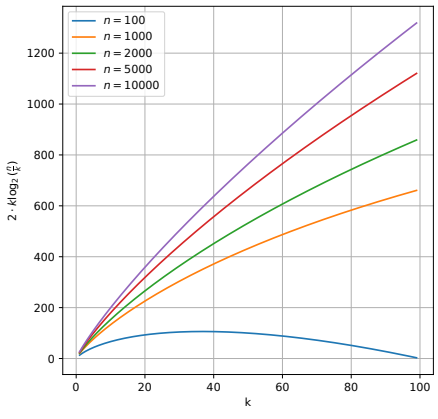


Figure 8: Number of protocol invocations (q) for $c = 2$.