ABSTRACT

Title of Dissertation:	ALGORITHMS FOR RECONSTRUCTING DATABASES AND CRYPTOGRAPHIC SECRET KEYS IN ENTROPIC SETTINGS
	Aria Shahverdi Doctor of Philosophy, 2022
Dissertation Directed by:	Professor Dana Dachman-Soled Department of Electrical and Computer Engineering

A small amount of information leakage can undermine the security of a design that is otherwise considered secure. Many studies demonstrate how common leakages such as power consumption, electromagnetic emission, and the time required to perform certain operations can reveal information, such as the secret key of a cryptosystem. As a first contribution, in this work, we explore the possibility of cache attacks, a type of timing side-channel attack, in a new setting, namely, data processing. Later we show an improved attack on Learning Parity with Noise problems with a sparse secret. We propose two algorithms that are asymptotically faster than state-of-the-art. Finally, we show that the structure presented in RLWE constructions, in contrast to LWE constructions, opens up new attacks. Constructions based on LWE can be proven secure as long as the secret retains enough entropy. We show, however, that constructions based on RLWE can be completely broken even if the secret key retains 3/4 of its entropy.

ALGORITHMS FOR RECONSTRUCTING DATABASES AND CRYPTOGRAPHIC SECRET KEYS IN ENTROPIC SETTINGS

by

Aria Shahverdi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Dedication

I would like to dedicate this dissertation to my beloved family.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Dachman-Soled for her invaluable advice, and continuous support during my Ph.D. study. Her immense knowledge and experience have encouraged me to pursue my academic research. I am also profoundly grateful for her patience throughout my studies and research, which helped me focus on the research. I was also lucky to have her insightful feedback, and I appreciate her flexibility in arranging meetings. I always felt that I could rely on her guidance when research did not go according to my expectation.

I would also like to thank my committee members, Prof. Srivastava, Prof. Katz, Prof. Dumitraş, and Dr. Aalgic, for agreeing to serve on my thesis committee. Their feedback and questions on the dissertation and during the defense helped me hugely improve the quality of the write-up.

I was also fortunate to have a chance to collaborate with Mukul Kulkarni, Huijing Gong, Hunter Kippen, and Mahammad Shirinov. We had many constructive conversations during meetings and other occasions, which helped me vastly deepen my understanding.

In addition, I would like to express my gratitude to Melanie Prange, William (Bill) Churma, Emily Irwin, and Dana Purcell. They always quickly responded to my questions, and their support made me focus on research without worrying about administrative work.

Finally, I wish to express my deepest gratitude to my parents, Homa and Farhad, who always encouraged me and sacrificed more than I could imagine to help me be the person I am today. I will forever remain in their debt for all they have done for me. I am most indebted to Negin, my loving and caring wife. She has always supported me unconditionally during my darkest moments as I pursued my Ph.D. The best thing that happened during my graduate program was that I met Negin. I especially appreciate her unconditional love and companionship. Thanks to my sister Bahar, who shows me endless love, support, and belief, I have a lot to be thankful for. In addition, I would like to thank my sister-in-law Nazanin and my brother-in-law Arsalan for their unending support and encouragement. Additionally, I would like to thank all my friends at College Park, whom I am fortunate to have.

I apologize to those I have inadvertently forgotten.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AES	Advanced Encryption Standard
AMD	Advanced Micro Devices
BLISS	Bimodal Lattice Signature Scheme
BKW	Blum-Kalai-Wasserman
BDD	Bounded Distance Decoding
CPU	Central Processing Unit
CVP	Closest Vector Problem
DNN	Deep Neural Network
DFT	Discrete Fourier Transform
DNF	Disjunctive Normal Form
ECDSA	Elliptic Curve Digital Signature Algorithm
HW	Hamming Weight
HCUP	Healthcare Cost and Utilization Project
k-NN	k-Nearest Neighbours
LLC	Last Level Cache
LPN	Learning Parity with Noise
LWE	Learning with Errors
LRU	Least-Recently Used
LCS	Longest Common Substring
NIST	National Institute of Standards and Technology
NIS	Nationwide Inpatient Sample
NP	Non-Deterministic Polynomial-Time
NTT	Number Theoretic Transform
PAC	Probably Approximately Correct
RLK	Restricted Left Kernel
R-DLWE	Ring - Decision Learning with Errors
R-SLWE	Ring - Search Learning with Errors
RLWE	Ring Learning with Errors
RSA	Rivest–Shamir–Adleman
SGX	Software Guard Extensions
SQL	Structured Query Language
TLS	Transport Layer Security
VM	Virtual Machine

Chapter 1: Introduction

As third-party computing on data in the cloud becomes more prevalent, clients are becoming increasingly concerned about protecting their sensitive data. The cloud computing paradigm has been hugely successful due to its affordability. This affordability is often achieved by reducing costs for the cloud, e.g., by sharing resources across clients. Shared resources have opened a wide range of new attack vectors for malicious users. Multiple studies have shown that malicious users can use shared resources to extract information that is otherwise considered private [68, 81, 99, 110, 112]. A majority of the prior work focused on the cache as a shared resource that the malicious user can monitor. There have been attacks on cryptographic construction such as encryption, e.g., AES [62, 103], RSA [110], ElGamal [68, 81, 112] signature scheme, such as Elliptic Curve Digital Signature Algorithm (ECDSA) [15, 20].

Another line of research assumes the server itself is entirely untrusted or malicious. These assumptions have motivated the design of new techniques for computing on encrypted data, such as Order Preserving Encryption by Agrawal et al. [7] which allow the server to directly run a comparison on encrypted data. These constructions prompted a line of research that shows attacks on an encrypted database such as the work of Kellaris et al. [72] which shows that leakage on access patterns or the volume of the query responses can lead to a construction of the database. These attacks are mainly theoretical since most of these advanced encryption systems have yet to be deployed.

Traditional cryptographic protocols are based on well-studied assumption factoring and discrete logarithm. However, with the introduction of Shor's factoring algorithm [101] which showed that factoring is in quantum polynomial time, it has become clear that it is essential to develop new types of cryptographic constructions based on new assumptions. As a result of this discovery, the problems which can withstand powerful quantum computers attract attention. One such a problem is Learning Parity with Noise. The *(search) LPN* problem with dimension n and noise rate η, asks to recover the secret parity $\mathbf{s}$, given samples $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e)$, where $\mathbf{x} \in \{0,1\}^n$ is chosen uniformly at random, $\mathbf{s} \in \{0,1\}^n$, error $e \in \{0,1\}$ is set to 1 with probability η and 0 with probability $1-\eta$, and the dot product is taken modulo 2. Solving a linear system of n equations over $\mathbb{F}_2$ to recover a secret of dimension n can be done in polynomial time however, adding a small amount of noise e makes the problem (believed to be) quantum-hard. Variants of the LPN problem have also been considered in the literature: Sparse LPN [25], where the $\mathbf{x}$ vectors in the LPN problem statement are sparse, LPN with structured noise, where the noise across multiple samples is guaranteed to satisfy some constraint [18], and Ring LPN [64]. While there have been some studies on the best algorithm to run to solve the secret, the best-known algorithm remains to be by the early work of Blum, Kalai, and Wasserman [24], and it seems there is a lack of algorithms in certain regimes.

Numerous other problems have also been studied which are believed to be suitable for post-quantum cryptography, such as Lattice-based [19, 49, 52] and code-based [21]. The National Institute of Standards and Technology (NIST) has also launched a competition [36] to find the most efficient and secure construction, and constructions based on lattices are found to be among the promising directions. The proposed work in lattice-based constructions is based on the Learning With Error (LWE) assumptions [98]. The LWE problem is defined as the problem of distinguishing the two distributions $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$ and $(\mathbf{a}, u)$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is a secret, $\mathbf{a} \in \mathbb{Z}_q^n$ and $u \in \mathbb{Z}_q$ are uniform, and e has small norm and q is a prime. A more efficient version of the LWE problem is based on the Ring structure, called RLWE [84, 86]. The RLWE problem is defined as the problem of distinguishing the two distributions $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$ and $(\mathbf{a}, \mathbf{u})$, where $\mathbf{s} \in R_q$ is a secret, $\mathbf{a} \in R_q$ and $\mathbf{u} \in R_q$ are uniform, and $e \in R$ has small norm and $R_q := \mathbb{Z}_q[x]/(x^n + 1)$. Since the introduction of the RLWE, there have been studies on whether the RLWE is vulnerable to new types of attacks. Due to the extra structure present in the ring structure, a new type of attack may be possible.

1.1 Our Contribution

This dissertation will try to address some of the open problems which are addressed in the previous section. Specifically, this work focuses on cache attacks on databases, proposes an improved algorithm for the LPN problem in some parameter settings, and finally presents an attack on RLWE constructions given certain leakage patterns. Each direction is explained in the following.

1.1.1 Cache Side-Channel Attack on Database

In Chapter 3, we show a cache side-channel attack on SQLite. There have not been many side-channel attacks on a real application other than cryptosystems. This chapter demonstrates how side-channel attacks can be applied to database management systems. We showed that using the information leaked through the cache, the attacker can recover the volume of the responses. However, the recovered volumes are not exactly equal to the correct volumes. In contrast to the previous works, which assumed the server was malicious, and as a result, the server was allowed to observe the exact volume. We show how to extend the previous work to noisy volumes and explore new settings, such as the cases where some queries are never made. This work was originally published in USENIX 2021 [100], in collaboration with Dana Dachman-Soled and Mahammad Shirinov.

1.1.2 New Algorithms for LPN with Sparse Parities

Later in Chapter 4, we consider LPN with sparse parities. It is worth mentioning that it can be assumed that the secret is drawn from the same distribution as the noise, as there is a reduction from LPN with secret $\mathbf{s}$ to LPN with secret $\mathbf{e}$, where $\mathbf{e}$ is the error vector obtained after n samples are drawn [17]. We assume that the “sparsity” or Hamming weight of the secret vector represented by k is significantly less than $\eta \cdot n$, where η is the error rate. Motivation for considering this variant is twofold. Firstly, Sparse secret might be used in some applications such as fully homomorphic encryption for efficiency purposes [37]. Secondly, we can consider an attack where some coordinates of the secret key $\mathbf{s}'$ is recovered (using some type of attack, e.g., side-channels) but we don’t have high confidence in each coordinate. We can turn our focus on $\mathbf{xs} - \mathbf{xs}' = \mathbf{x}(\mathbf{s} - \mathbf{s}')$ which is a sparse secret. In this work, our focus is to achieve an algorithm that, for certain regimes of k, beats the prior best algorithms asymptotically in the exponent. This work was originally published in TCC 2021 [40], in collaboration with Dana Dachman-Soled, Huijing Gong, and Hunter Kippen.

1.1.3 (In)Security of Ring-LWE Under Partial Key Exposure

Finally, in Chapter 5, we demonstrate a key recovery attack on the RLWE scheme. We showed that leaking a certain pattern of key coordinates will result in the full recovery of the secret key. In particular, we demonstrated that by leaking only 1/4-fraction of the secret key, the attacker can obtain a system of equations that by solving yields the secret key. The attacker can recover the secret key block by block. We show that in some cases, by observing only a small number of samples, the full recovery is feasible. This work was originally published in MathCrypt 2019 [43], in collaboration with Dana Dachman-Soled, Huijing Gong, and Mukul Kulkarni.

Chapter 2: Preliminary

2.1 Background for Chapter 3

In this section we remind the reader some of the preliminary notation and results used throughout Chapter 3.

2.1.1 Notation

The number of possible values in a certain column of database is denoted by N, and we also assume without loss of generality that the set of possible values are between 1 and N. For any pair $(x, y) : x, y \in [N], x \leq y$, $[x, y]$ defines a range (inclusive in both side) and the same notation is used to to represent a range query. The volume (cardinality) of each range query is represented by $|[x, y]|$. There are $N' = \binom{N}{2} + N$ possible ranges.

2.1.2 Computer Architecture

Cache Architecture In order to reduce the access time to main memory, modern CPUs are equipped with multiple levels of cache. They form a hierarchy such that the Level 1 cache is the fastest and smallest, whereas the Level 3 cache is the slowest and largest.

The Level 1 cache is divided into two separate caches, one holds the data and the other holds the instructions. In the higher level caches data and instructions are held in the same cache. Level 3 is a shared-memory space and is the Last Level Cache (LLC). The LLC is all-inclusive of the lower levels of the architecture, meaning that any data present in L1 and L2 is also present in the LLC.

Each cache comprises multiple sets and each set contains multiple cache lines. Each line of main memory is mapped to a unique cache set. Within this set, however, a memory line can be mapped to any of the cache lines. Typically, each line of cache holds 64 Bytes of data. Upon writing a line to a set that is already full, a decision regarding which memory line to evict must be made. This decision is called a *Replacement Policy* and depends on the cache architecture. A popular replacement policy is *least-recently used (LRU)*, which replaces the least recently used entry with the new one.

Flush+Reload Attack Caches are vulnerable to information leakage since an adversary who is co-located with the victim on the same processor can retrieve useful information about a victim's activities. Specifically, the adversary can monitor its own access time to the cache and use deviations in access time to deduce information about whether or not the victim has accessed a certain memory line or not. The reason that such an attack is feasible is that the adversary and victim share the same resource i.e. the cache. A victim and an attacker on the same physical CPU core share all levels of the cache, whereas if they are not on the same physical core, they at least share the L3 cache. Moreover, in a setting where the adversary and victim share a library, they will both have access to the physical memory locations in which the *single copy* of the library is stored. The attacker can now explicitly remove a line corresponding to the shared physical memory from the cache. To exploit the shared physical memory in a useful way, Yarom and Falkner introduced an attack called *Flush+Reload* [110]. The attacker flushes a monitored line from the cache using a special command called `clflush`. This command causes the monitored line to be removed from the L1, L2 and L3 caches. As mentioned before, L3 is inclusive and as a result the removed line will be removed from all the other caches, even if the attacker and the victim are not on the same physical core. The attacker then lets the victim continue to run its program. After some time has elapsed, the attacker regains control and measures memory access time to determine whether or not the monitored line is present in the cache. If the monitored line is present in the cache (reloading runs fast), the attacker deduces that the same line was accessed by the victim during its run. If the monitored line is not present in the cache (reloading runs slow), the attacker deduces that the victim did not access the line during its run. Hence the attacker knows whether the victim accessed a specific line or not. In order to perform the Flush+Reload attack we used the package provided in the Mastik framework. Mastik [108] is a toolkit with various implementations of published micro-architectural side-channel attacks. It provides an interface that can be used to set the monitored lines. For our work we used `fr-trace` to monitor various cache lines.

Cache Prefetching When an instruction or data is needed from memory, it is fetched and brought into the cache. To reduce execution time further, *Cache Prefetching* is implemented to bring a memory line into the cache *before* it is needed. The prefetching algorithm decides what and when to bring data and instruction to the cache. Hence, when the program needs the data or instruction in the future, it will be loaded from the cache instead of memory. This is based on the past access patterns or on the compiler's knowledge.

2.1.3 Miscellaneous

Range Queries A range query is an operation on a database in which records with column values between a certain lower and higher bound are returned. Assuming there exists a column c in a database with values between 1 and N, the command $\texttt{range}[a, b]$ for $1 \leq a \leq b \leq N$ returns all the entries in the database which have a value in column c in the range $[a, b]$ (inclusive for both a and b).

Clique Finding Problem The Clique problem is the problem of finding a clique–a set of fully connected nodes–in a graph. We utilize the clique finding algorithm in the NetworkX Package. NetworkX is a Python library for studying graphs and networks. The NetworkX package can be used to find the clique number (size of the largest clique in the graph) as well as all cliques of different sizes in the graph.

2.2 Background for Chapter 4

In this section we remind the reader some of the preliminary notation and results used throughout Chapter 4.

2.2.1 Notations

We use $:=$ as deterministic assignment and $\leftarrow$ as uniformly randomized assignment. We also use bold lowercase, e.g. $\mathbf{x}$, to denote vectors and bold uppercase, e.g. $\mathbf{A}$, to denote matrix. The set $\{1, 2, \ldots, n\}$ is often denoted by $[n]$. The i-th coordinate of vector $\mathbf{x}$ is denoted by $\mathbf{x}[i]$. For the vector $\mathbf{x}$ of dimension n and a set R that is a subset of $[n]$, we denote $\mathbf{x}|_R$ to be the restriction of $\mathbf{x}$ to the coordinates in R, namely $\mathbf{x}|_R = \mathbf{x}[i_1] \mid\mid \mathbf{x}[i_2] \mid\mid \ldots \mid\mid \mathbf{x}[i_{|R|}], \forall i \in R$, where $||$ denotes concatenation. The indices in $\mathbf{x}$ are from 1 to n. For simplicity, we reset the indices in $\mathbf{x}|_R$ and have the indices from 1 to $|R|$.

2.2.2 Probability Bounds

The following inequality is used to bound the magnitude of an observed random variable with respect to the true expected value of that random variable. The Chernoff-Hoeffding bound extends the Chernoff bound to random variables with a bounded range. Another important fact is that Chernoff-Hoeffding bound assumes the random variables are independent whereas Chebyshev's bound applies to arbitrary random variables. The reader in encouraged to refer to [89] for more in depth reading.

Theorem 1 (Multiplicative Chernoff Bounds)**.** *Let $X_1, X_2, \ldots, X_n$ be n mutually independent random variables. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$,*

$$\Pr[X \leq (1-\beta)\mu] \leq \exp\left(\frac{-\beta^2\mu}{2}\right) \textit{ for all } 0 < \beta \leq 1$$

$$\Pr[X \geq (1+\beta)\mu] \leq \exp\left(\frac{-\beta^2\mu}{3}\right) \textit{ for all } 0 < \beta \leq 1$$

Theorem 2 (Chernoff-Hoeffding)**.** *Consider a set of n independent random variables $X_1, X_2, \ldots, X_n$. If we know $a_i \leq X_i \leq b_i$, then let $\Delta_i = b_i - a_i$. Let $X = \sum_{i=1}^n X_i$.*

Then for any $\alpha \in (0, 1/2)$

$$\Pr\left(\left|X - \mathbb{E}[X]\right| > \alpha\right) \leq 2\exp\left(\frac{-2\alpha^2}{\sum_{i=1}^{n} \Delta_i^2}\right).$$

Theorem 3 (Chebyshev's)**.** *Consider a set of n arbitrary random variable $X_1, X_2, \ldots, X_n$. Let $X = \sum_{i=1}^{n} X_i$. Then for any $\alpha > 0$,*

$$\Pr\left(\left|X - \mathbb{E}[X]\right| \geq \alpha\right) \leq \frac{\mathrm{Var}\left[X\right]}{\alpha^2}.$$

The following lemma is being used to further simplify the $\mathrm{Var}[X]$ in Theorem 3.

Lemma 4. *Let $X_1, X_2, \ldots, X_n$ be n arbitrary random variables. Then*

$$\mathrm{Var}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{Var}\left[X_i\right] + 2\sum_{i=1}^{n}\sum_{j>i} \mathrm{Cov}\left[X_i, X_j\right].$$

2.2.3 Learning Parities

In this section, we define three Oracles . The first is the *standard* LPN Oracle, that samples $\mathbf{x}$ uniformly. The second is the noise Oracle, which sets $\mathbf{x}$ to the zero vector. The purpose of this Oracle is to return additional noise sampled identically to the noise found in a normal LPN sample. The third Oracle is the p-biased LPN Oracle, which samples $\mathbf{x}$ according to a p-biased Bernoulli distribution, as defined later.

Definition 5 (Bernoulli Distribution)**.** *Let $p \in [0, 1]$. The discrete probability distribution of a random variable which takes the value 1 with probability η and the value 0 with probability $1 - \eta$ is called Bernoulli Distribution and it is denoted by Ber_η.*

Definition 6 (LPN Oracle)**.** *Let secret value* $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ *and let* $\eta < 1/2$ *be a constant noise parameter. Let* Ber_η *be a Bernoulli distribution with parameter* η*. Define the following distribution* $\mathcal{L}^{(1)}_{\mathbf{s},\eta}$ *as follows*

$$\left\{\, (\mathbf{x}, b) \mid \mathbf{x} \leftarrow \mathbb{Z}_2^n,\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done module 2. Upon calling the LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$*, a new sample* $\mathsf{s} = (\mathbf{x}, b)$ *from the distribution* $\mathcal{L}^{(1)}_{\mathbf{s},\eta}$ *is returned.*

Definition 7 (Noise Oracle)**.** *Let secret value* $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ *and let* $\eta < 1/2$ *be a constant noise parameter. Let* Ber_η *be a Bernoulli distribution with parameter* η*. Define the following distribution* $\mathcal{L}^{(2)}_{\mathbf{s},\eta}$ *as follows*

$$\left\{\, (\mathbf{x}, b) \mid \mathbf{x} := 0^n,\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done module 2. Upon calling the Noise Oracle $\tilde{\mathcal{O}}_\eta$ *a new sample* $\mathsf{s} = (\mathbf{x}, b)$ *from the distribution* $\mathcal{L}^{(2)}_{\mathbf{s},\eta}$ *is returned.*

Definition 8 (p-biased LPN Oracle)**.** *Let secret value* $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ *and let* $\eta < 1/2$ *be a constant noise parameter. Let* Ber_η *be a Bernoulli distribution with parameter* η *and* $\mathsf{Ber}^n_{(1-p)/2}$ *be Bernoulli distribution with parameter* $(1-p)/2$ *over* n *coordinates. Define the following distribution* $\mathcal{L}^{(3)}_{\mathbf{s},\eta,p}$ *as follows*

$$\left\{\, (\mathbf{x}, b) \mid \mathbf{x} \leftarrow \mathsf{Ber}^n_{(1-p)/2},\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done modulo 2. Upon calling the p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta}(\mathbf{s})$ *a new sample* $\mathsf{s}^p = (\mathbf{x}, b)$ *from the distribution* $\mathcal{L}^{(3)}_{\mathbf{s},\eta,p}$ *is returned.*

As our algorithms require linear combinations of LPN samples, we present the following lemma that describes the noise growth associated with the linear combination.

Lemma 9 (New Sample Error [24])**.** *Given a set of* ℓ *samples* $(\mathbf{x}_1, b_1), \ldots, (\mathbf{x}_\ell, b_\ell)$ *from an LPN Oracle* $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ *with secret* $\mathbf{s}$*, where the choice of samples may depend on the values of* $\mathbf{x}_i$ *but not on the values of* b_i*, then the new sample* $\mathsf{s}_{\ell+1}$ *can be formed as follows* $\mathsf{s}_{\ell+1} = \sum_{i=1}^{\ell} \mathsf{s}_i$ *which has the property that* $b_{\ell+1}$ *is independent of* $\mathbf{x}_{\ell+1}$ *and the probability that the label of the constructed sample is not correct is as follows:* $\eta' = \Pr[b' \neq \langle \mathbf{x}_{\ell+1}, s\rangle] = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\ell}$.

For reference we additionally provide the runtime of the original BKW algorithm:

Theorem 10 (BKW [24])**.** *The length-n parity problem, for noise rate* η *for any constant less than* $1/2$*, can be solved with number of samples and total computation time of* $2^{O(n/\log n)}$*.*

Sometimes we denote the "corrupted" label b in sample $\mathsf{s} = (\mathbf{x}, b)$ by $f(\mathbf{x})$. The function f is called the parity function. So we use the phrase that "the label is $f(\mathbf{x})$" to mean that the label is $\langle \mathbf{x}, \mathbf{s}\rangle$ with probability $1-\eta$ and $\langle \mathbf{x}, \mathbf{s}\rangle + 1$ with probability η. For sample i, the j-th coordinate of $\mathbf{x}$ is denoted by $\mathsf{s}_i.\mathbf{x}[j]$ and the j-th coordinate of $\mathbf{s}$ is denoted by $\mathsf{s}_i.\mathbf{s}[j]$, and we sometimes drop s_i when the context is clear. For simplicity, given two sample pairs $\mathsf{s}_1 = (\mathbf{x}_1, b_1)$ and $\mathsf{s}_2 = (\mathbf{x}_2, b_2)$ a new sample $\mathsf{s}_3 = \mathsf{s}_1 + \mathsf{s}_2$ can be formed by $\mathsf{s}_3 = (\mathbf{x}_1 + \mathbf{x}_2, b_1 + b_2)$ with the additions being done mod 2. Lemma 9 shows the error rate of a sample formed by additions of some number of LPN samples.

2.2.4 Fourier Analysis

The boolean Fourier transform is defined for boolean functions defined over the domain $\{-1, 1\}$. Throughout the rest of the chapter, when we discuss boolean functions, we will use this representation. To map a boolean function from $\{0, 1\} \in \mathbb{F}_2$ to $\{-1, 1\}$, we set $-1 := 1_{\mathbb{F}_2}$ and $1 := 0_{\mathbb{F}_2}$. We now present some additional notation regarding the representation of the LPN problem in the $\{-1, 1\}$ domain.

Notation. *Assuming the LPN secret* $\mathbf{s}$ *is represented in* $\mathbb{F}_2^n$*, the following represent the boolean inner product of input* $\mathbf{x}$ *with* $\mathbf{s}$ *in different notation.*

$$f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle \in \mathbb{F}_2 \textit{ for } \mathbf{x}, \mathbf{s} \in \mathbb{F}_2^n$$

$$f_{\mathbf{s}}(\mathbf{x}) = \prod_{i=1}^{n} \mathbf{x}[i]^{\mathbf{s}[i]} \in \{-1, 1\} \textit{ for } \mathbf{x} \in \{-1, 1\}^n \textit{ and } \mathbf{s} \in \mathbb{F}_2^n$$

hence to represent a sample $(\mathbf{x}, b)$ *from LPN oracle* $\mathcal{O}_{0,\eta}^{\mathsf{LPN}}(\mathbf{s})$ *we have the following two notations*

$$b = f_{\mathbf{s}}(\mathbf{x}) + e \textit{ for } \mathbf{x}, \mathbf{s} \in \mathbb{F}_2^n \textit{ and } e \in \mathbb{F}_2$$

$$b = f_{\mathbf{s}}(\mathbf{x}) \cdot e \textit{ for } \mathbf{x} \in \{-1, 1\}^n,\ \mathbf{s} \in \mathbb{F}_2^n \textit{ and } e \in \{-1, 1\}$$

Consider a vector $\mathbf{x} \in \{-1, 1\}^n$. We denote by $\mathcal{D}_p$ the product distribution over $\{-1, 1\}^n$, where each bit of the vector is independent and has mean p.

Definition 11 (Fourier Expansion)**.** *For a product distribution* $\mathcal{D}_p$ *as above, every function*

$f : \{-1,1\}^n \to \mathbb{R}$ can be uniquely expressed as the multilinear polynomial

$$f(\mathbf{x}) = \sum_S \hat{f}_p(S)\chi_{S,p}(\mathbf{x}), \text{ where } \chi_{S,p}(\mathbf{x}) = \prod_{i \in S} \frac{\mathbf{x}[i] - p}{\sqrt{1-p^2}}.$$

This expression is called the Fourier expansion of f with respect to $\mathcal{D}_p$, and the real numbers $\hat{f}(S)$ are called the Fourier coefficients of f on S.

The Fourier transform defines an inner product between two boolean functions f and g: $\langle f, g \rangle_p = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[f(\mathbf{x}) \cdot g(\mathbf{x})]$. The Fourier coefficient for any $S \subset N$ over product distribution $\mathcal{D}_p$ is defined as follows:

$$\hat{f}_p(S) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[f(\mathbf{x}) \cdot \chi_S(\mathbf{x})].$$

Claim 1. *Let $\mathsf{s}^p = (\mathbf{x}, b)$ be a p-biased sample and let $b = f_{\mathbf{s}}(\mathbf{x}) \cdot e$, where $e \in \{-1,1\}$ is independent of $\mathbf{x}$ and $\mathbb{E}[e] = 1 - 2\eta'$. Define $\hat{b}_p(\{j\}) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[b \cdot \chi_{\{j\},p}(\mathbf{x}))]$. If $\mathsf{s}^p.\mathbf{s}[j] = 0$, then $\hat{b}_p(\{j\}) = 0$. Whereas if $\mathsf{s}^p.\mathbf{s}[j] = 1$, then $\hat{b}_p(\{j\}) = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$.*

Proof. In the first part of the proof we assume the p-biased sample is absent of noise, i.e. $b = f_{\mathbf{s}}(\mathbf{x})$ and we compute the Fourier coefficient of the singleton set $\{j\}$ in two cases where **(Case 1)** $\mathsf{s}^p.\mathbf{s}[j] = 0$ and **(Case 2)** $\mathsf{s}^p.\mathbf{s}[j] = 1$. In the following we simplify $\mathsf{s}^p.\mathbf{x}[j]$ by $\mathbf{x}[j]$ and $\mathsf{s}^p.\mathbf{s}[j]$ by $\mathbf{s}[j]$.

We compute the Fourier coefficients of $f_{\mathbf{s}}(\mathbf{x})$ denoted by $\hat{f}_p$ as follows.

$$\begin{aligned}\hat{f}_p(S) = \langle f(\mathbf{x}), \chi_{S,p}\rangle_p &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[f_{\mathbf{s}}(\mathbf{x}) \cdot \chi_{S,p}(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[f_{\mathbf{s}}(\mathbf{x}) \cdot \prod_{i\in S}\frac{\mathbf{x}[i]-p}{\sqrt{1-p^2}}\right]\end{aligned}$$

- **Case 1 :** Let's focus on the set $S = \{j\}$ such that $\mathbf{s}[j] = 0$.

$$\begin{aligned}
\hat{f}_p(\{j\}) &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[f_{\mathbf{s}}(\mathbf{x}) \cdot \frac{\mathbf{x}[j]-p}{\sqrt{1-p^2}}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\left(\prod_{i:\mathbf{s}[i]=1} \mathbf{x}[i]\right) \cdot \frac{\mathbf{x}[j]-p}{\sqrt{1-p^2}}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\left(\prod_{i:\mathbf{s}[i]=1} \mathbf{x}[i]\right)\right] \cdot \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\frac{\mathbf{x}[j]-p}{\sqrt{1-p^2}}\right] \qquad (2.1) \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\left(\prod_{i:\mathbf{s}[i]=1} \mathbf{x}[i]\right)\right] \cdot 0 \qquad (2.2) \\
&= 0
\end{aligned}$$

Where equation (2.1) follows from independence and equation (2.2) follows since $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\mathbf{x}[i] = p$.

- **Case 2 :** Now let's focus on the set $S = \{j\}$ such that $\mathbf{s}[j] = 1$.

$$\begin{aligned}
\hat{f}_p(\{j\}) &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[f_{\mathbf{s}}(\mathbf{x}) \cdot \frac{\mathbf{x}[j]-p}{\sqrt{1-p^2}}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\left(\prod_{i:\mathbf{s}[i]=1} \mathbf{x}[i]\right) \cdot \frac{\mathbf{x}[j]-p}{\sqrt{1-p^2}}\right] \\
&= \frac{1}{\sqrt{1-p^2}}\left(\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\prod_{i:\mathbf{s}[i]=1\wedge i\neq j} \mathbf{x}[i]\right] - p\cdot\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[\prod_{i:\mathbf{s}[i]=1} \mathbf{x}[i]\right]\right) \qquad (2.3) \\
&= \frac{1}{\sqrt{1-p^2}}\left(\prod_{i:\mathbf{s}[i]=1\wedge i\neq j} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[\mathbf{x}[i]] - p\cdot\prod_{i:\mathbf{s}[i]=1} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[\mathbf{x}[i]]\right) \qquad (2.4) \\
&= \frac{1}{\sqrt{1-p^2}}\left(p^{k-1} - p\cdot p^k\right) \\
&= p^{k-1}\sqrt{1-p^2}
\end{aligned}$$

Where equation (2.3) follows since $\mathbf{x}[j]^2 = 1$ in the first term and eqaution (2.4) follows from independence.

Now we compute the Fourier Coefficient for the case where the example label is noisy, i.e. $b = f_{\mathbf{s}}(\mathbf{x}) \cdot e$. The distribution of the noise e is represented as follows.

$$e = \begin{cases} 1 & \text{with probability } 1 - \eta' \\ -1 & \text{with probability } \eta' \end{cases}$$

The Fourier coefficient of the labels can be computed as follows.

$$\begin{aligned} \left|\hat{b}_p(S)\right| &= \left|\langle f_{\mathbf{s}}(\mathbf{x}) \cdot e, \chi_{S,p}\rangle_p\right| \\ &= \left|\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[f_{\mathbf{s}}(\mathbf{x}) \cdot e \cdot \chi_{S,p}(\mathbf{x})]\right| \\ &= \left|\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}(e) \cdot \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}[f_{\mathbf{s}}(\mathbf{x})\chi_{S,p}(\mathbf{x})]\right| \\ &= (1 - 2\eta') \cdot \left|\hat{f}_p(S)\right| \end{aligned}$$

- **Case 1 :** For j such that $\mathbf{s}[j] = 0$

$$\hat{b}_p(\{j\}) = 0$$

- **Case 2 :** For j such that $\mathbf{s}[j] = 1$

$$\hat{b}_p(\{j\}) = (1 - 2\eta') \cdot p^{k-1}\sqrt{1 - p^2}$$

□

2.2.5 Miscellaneous

Definition 12 (Restricted Left Kernel)**.** *Given a matrix* $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ *for* $m \leq n$ *and set* $R \subset [n]$ *such that* $|R| < m$*,* RLK *first finds a vector* $\mathbf{u} \in \mathbb{Z}_2^m$ *such that* $\mathbf{v} = \mathbf{u} \cdot \mathbf{A}$ *and* $\mathbf{v}|_R = 0^{|R|}$*. The* RLK *algorithm returns* $(\mathbf{v}, \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R)$*.*

Note that the RLK algorithm mentioned above can be implemented by simply modifying matrix $\mathbf{A}$ and only takes the columns pointed by set R, i.e. restriction of $\mathbf{A}$ to only columns pointed by R. Let's denote the new matrix by $\mathbf{A}'$, find a vector in left kernel of $\mathbf{A}'$ and call it $\mathbf{u}$. Then $\mathbf{v}$ can simply be computed as $\mathbf{v} = \mathbf{u} \cdot \mathbf{A}$.

Definition 13 (Hamming Weight)**.** *Given a vector* $\mathbf{u} \in \mathbb{Z}_2^m$*,* $\mathsf{weight}(\mathbf{u})$ *returns the number of 1's in vector* $\mathbf{u}$*, i.e. the Hamming weight of* $\mathbf{u}$.

2.3 Background for Chapter 5

For a positive integer n, we denote by $[n]$ the set $\{0, \ldots, n-1\}$. We denote vector x using the notation $\vec{x}$ and matrices using capital letters $\vec{A}$. For vector $\vec{x}$ over $\mathbb{R}^n$ or $\mathbb{C}^n$, define the ℓ_2 norm as $\|\vec{x}\|_2 = \left(\sum_i |x_i|^2\right)^{1/2}$. We write as $\|\vec{x}\|$ for simplicity. We use the notation $\approx_{t(n),p(n)}$ to indicate that adversaries running in time $t(n)$ can distinguish two distributions with probability at most $p(n)$. We present the background and standard definitions related to lattices, algebraic number theory, RLWE, and NTT transform in the followings.

2.3.1 Background on Lattices

An n-dimensional *lattice* $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$. We exclusively consider the full-rank lattices, which are generated as the set of all linear integer combinations of some set of n linearly independent *basis* vectors $B = \{\vec{b}_j\} \subset \mathbb{R}^n$:

$$\Lambda = \mathcal{L}(B) = \left\{ \sum_j z_j \vec{b}_j : z_j \in \mathbb{Z} \right\}.$$

The *determinant* of a lattice $\mathcal{L}(B)$ is defined as $|\det(B)|$, which is independent of the choice of basis B. The *minimum distance* $\lambda_1(\Lambda)$ of a lattice Λ (in the Euclidean norm) is the length of a shortest nonzero lattice vector.

The *dual lattice* of $\Lambda \subset \mathbb{R}^n$ is defined as following, where $\langle \cdot, \cdot \rangle$ denotes the inner product.

$$\Lambda^\vee = \{\vec{y} \in \mathrm{Span}(B) : \forall \vec{x} \in \Lambda, \langle \vec{x}, \vec{y} \rangle = \sum_i x_i y_i \in \mathbb{Z}\}.$$

Note that, $(\Lambda^\vee)^\vee = \Lambda$, and $\det(\Lambda^\vee) = 1/\det(\Lambda)$.

2.3.2 Algebraic Number Theory

For a positive integer m, the m^{th} *cyclotomic number field* is a field extension $K = \mathbb{Q}(\zeta_m)$ obtained by adjoining an element ζ_m of order m (i.e. a primitive m^{th} root of unity) to the field of rationals, where ζ_m satisfies the relation $f(\zeta_m) = 0$ for some irreducible polynomial $f(x) \in \mathbb{Q}[x]$, which is monic without loss of generality. The polynomial f is called the minimal polynomial of ζ_m, and the degree n of the number field is the degree of

f. The minimal polynomial of ζ_m is the m^{th} *cyclotomic polynomial*

$$\Phi_m(X) = \prod_{i \in \mathbb{Z}_m^*} (X - \omega_m^i) \in \mathbb{Z}[X],$$

where $\omega_m \in \mathbb{C}$ is any primitive m^{th} root of unity in $\mathbb{C}$.

2.3.3 Ring of Integers and Its Ideals

An *algebraic integer* is an element whose minimal polynomial over the rationals has integer coefficients. For a number field K, let $R \subset K$ denote the set of all algebraic integers in K. This set forms a ring (under the usual addition and multiplication operations in K), called the *ring of integers* of the number field. Ring of integers in K is written as $R = \mathbb{Z}[\zeta_m]$. An (*integral*) *ideal* $\mathcal{I} \subseteq R$ is a non-trivial (i.e. $\mathcal{I} \neq \varnothing$ and $\mathcal{I} \neq \{0\}$) additive subgroup that is closed under multiplication by R, i,e., $r \cdot a \in \mathcal{I}$ for any $r \in R$ and $a \in \mathcal{I}$.

Definition 14. *For $R = \mathbb{Z}[\zeta_m]$, define $g = \prod_p (1 - \zeta_p) \in R$, where p runs over all* odd *primes dividing m. Also, define $t = \frac{\hat{m}}{g} \in R$, where $\hat{m} = \frac{m}{2}$ if m is even, otherwise $\hat{m} = m$.*

The *dual ideal* $R^\vee$ of R is defined as $R^\vee = \langle t^{-1} \rangle$, satisfying $R \subseteq R^\vee \subseteq \hat{m}^{-1}R$. The quotient $R_q^\vee$ is defined as $R_q^\vee = R^\vee / qR^\vee$.

2.3.4 Ring Learning With Errors

We next present the formal definition of the Ring-LWE problem as given in [87].

Definition 15 (**Ring-LWE Distribution**)**.** *For a "secret" $s \in R_q^\vee$ (or just $R^\vee$) and a* *distribution χ over $K_{\mathbb{R}}$, a sample from the ring-*LWE *distribution $A_{s,\chi}$ over $R_q \times (K_{\mathbb{R}}/qR^{\vee})$ is generated by choosing $a \leftarrow R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = a \cdot s + e \bmod qR^{\vee})$.*

Definition 16 (Ring-LWE, Average-Case Decision). *The* average-case decision *version of the ring-*LWE *problem, denoted R-$\mathsf{DLWE}_{q,\chi}$, is to distinguish with non-negligible advantage between independent samples from $A_{s,\chi}$, where $s \leftarrow R_q^{\vee}$ is sampled uniformly at random, and the same number of uniformly random and independent samples from $R_q \times (K_{\mathbb{R}}/qR^{\vee})$.*

A Note on the Tweak Alperin-Sheriff and Peikert [16] show that an equivalent "tweaked" form of the Ring-LWE problem can be used in cryptographic applications without loss in security or efficiency. This is convenient since the "tweaked" version does not involve $R^{\vee}$. The "tweaked" Ring-LWE problem can be obtained by implicitly multiplying the noisy products b by the "tweak" factor t, and, as it is explained in [16], $t \cdot R^{\vee} = R$. This yields new values

$$b' = t \cdot b = (t \cdot s) \cdot a + (t \cdot e) = s' \cdot a + e' \bmod qR,$$

where $a, s' = t \cdot s \in R_q$, and the errors $e' = t \cdot e$ come from the "tweaked" error distribution $t \cdot \chi$.

2.3.5 Number Theoretic Transform (NTT)

Let $R_q := \mathbb{Z}_q[x]/(x^n + 1)$ be the ring of polynomials, with $n = 2^d$ for any positive integer d. Also, let $m = 2n$ and $q = 1 \bmod m$. For, ω a m^{th} root of unity in $\mathbb{Z}_q$ the NTT of polynomial $\vec{p} = \sum_{i=0}^{n-1} p_i x^i \in R_q$ is defined as,

$$\widehat{\vec{p}} = \mathsf{NTT}(\vec{p}) := \sum_{i=0}^{n-1} \widehat{p}_i x^i$$

where the NTT coefficients $\widehat{p}_i$ are defined as: $\widehat{p}_i = \sum_{j=0}^{n-1} p_j \omega^{j(2i+1)} = \vec{p}\left(\omega^{2i+1}\right)$.

The function NTT^{-1} is the inverse of function NTT, defined as

$$\vec{p} = \mathsf{NTT}^{-1}(\widehat{\vec{p}}) := \sum_{i=0}^{n-1} p_i x^i$$

where the NTT inverse coefficients p_i are defined as: $p_i = n^{-1} \sum_{j=0}^{n-1} \widehat{p}_j \omega^{i(2j+1)}$.

Chapter 3: Cache Side-Channel Attack on Database

3.1 Introduction

Data processing in the cloud is becoming continually more pervasive and cloud computing is intrinsic to the business model of various popular services such as Microsoft's Office 365, Google's G Suite, Adobe Creative Cloud or financial services such as intuit [1]. Besides for cloud usage by industry, federal agencies are now utilizing cloud services, even for storage and analytics of sensitive data. For example, Microsoft recently won a $10 billion government contract from the Department of Defense (DoD) to create a "secure cloud" for the Pentagon [2]. While providing important functionality, processing of sensitive information in the cloud raises important security challenges. In the extreme case, one may not trust the cloud server itself to handle the sensitive data, corresponding to a threat model in which the cloud server is assumed to be malicious. In this case, data must be encrypted, which raises the challenging task of computation over encrypted data. Techniques and tools for computation over encrypted data have been addressed in a myriad of papers [29, 35, 77, 97] and various privacy attacks have also been exhibited [72, 92]. A weaker threat model, considered in this work, assumes that the server may be trusted to handle the sensitive data (e.g. a privacy agreement has been signed with the cloud service), but that a spy process is running on the same public server. If a spy process is co-located with the victim on the same physical machine they will share hardware such as a cache, which serves as a side channel.

Our goal is to explore the effect of side-channels on open-source database engines. We present an attack on SQLite, a C-language library that implements a small and fast SQL database engine and is among the top ten databases in the ranking released by db-engines.com. Our threat model assumes that an external user queries a private database stored on a victim VM, upon which the victim VM processes the query using SQLite and returns the result to the external user. The attacker is disallowed from directly querying the database or observing the outputs of a query. Since the attacker is running a spy VM co-located with the victim VM in the cloud, it can monitor the shared cache to obtain side-channel leakage. The goal of the attacker is to reconstruct the column upon which the victim is making range queries.

3.1.1 Relationship to Attacks on Searchable Encryption

Our work is inspired by the line of works of Kellaris et al. [72], Grubbs et al. [58], Lacharité et al. [75] and Grubbs et al. [59]. These works exhibited database reconstruction attacks in scenarios where range queries are made to an *encrypted* database and the access pattern (i.e. which records are returned) [72, 75] or communication volume (i.e. the number of records returned) [58, 72] is observed by the malicious server. However, recall that in our threat model, an attacker cannot simply observe the access pattern or communication volume, and must instead resort to side channels (such as a shared cache) to learn information. Indeed, our attack will utilize the cache side-channel to learn information about the communication volume of the range queries. Briefly, this is done by finding a line of code that is executed once for each record returned in a response to a range query, and tracking how many times that line of code is executed.

Since cache side-channels are inherently noisy, we are only able to measure the *approximate* or *noisy* volumes of the range queries. We emphasize that even adding a small amount of noise to the volume of each range foils the reconstruction attacks from prior work. We assessed the effects of noise on brute-force reconstruction (an analogue of the brute-force algorithm suggested by [72] for the dense database setting), and on the clique-finding approach developed by [58]. As will be discussed in depth in Section 3.1.4, we conclude that both of these approaches fail in the noisy setting.

3.1.2 Our Approach

We develop a new algorithmic approach that reduces our noisy problem to other computational problems that are well-studied in the literature and for which highly optimized solvers have been developed. Specifically, we will leverage both a noise-tolerant clique-finding algorithm (similar to [58], but with some crucial modifications) as well as a closest vector problem (CVP) solver. In more detail, we first use the noisy cache data to craft an instance of the clique-finding problem that is noise-tolerant. Recovered cliques will then be used to obtain candidate databases that are "close" to the original database. To extrapolate volumes that may be entirely missing from the recovered cliques, we develop a "Match & Extend" algorithm. After the Match & Extend step, we expect to have reconstructed approximate volumes for all ranges. We then apply a "Noise Reduction Step" that takes the "close" solution outputted by the previous step, consisting of approximate volumes for each of the ranges $[i, i]$ for $i \in N$, and uses it to craft an instance of the CVP problem. Solutions to the CVP problem correspond to reconstructed databases in which the overall noise is further reduced.

We note that since our side-channel attack proceeds by measuring (approximate) range query volumes, it is agnostic to whether the victim's database is encrypted. As long as the spy can monitor a line of code that is executed by the database engine for each record returned by a range query, our attack is feasible. Searchable encryption schemes that have this property would still be susceptible to this side-channel attack. For example searchable encryption schemes that can be integrated with standard database engines, such as order preserving encryption [8, 28] and order revealing encryption [30].

A limitation of our work is that our approach uses solvers for NP-hard problems as subroutines. The complexity of these NP-hard problems grows quickly with the size of the range, and therefore will work well in practice for ranges up to size 15. This is in contrast to the recent work of Grubbs et al. [59], which showed how to do "approximate reconstruction" in a way that scales only with the desired accuracy level and not the range size. However, the work of Grubbs et al. [59] assumes the adversary gets to perfectly observe the *access pattern*—i.e. which records are returned for each query—which provides far more information than simply observing the volumes. It seems difficult to extract the *access pattern* for a response to a database query from a cache side-channel attack. Specifically, to extract access pattern from the cache, Prime & Probe must be used to monitor the data cache, and because a single record from the database can fill a large portion of the cache, it is difficult to distinguish which records were accessed by observing only the cache. Additionally, the mapping from the memory location to cache line is not one to one and hence, a large number of records will map to the same cache locations, making it difficult to distinguish which records were accessed.

We extensively test our attack in various scenarios, using real databases (with data distribution close to uniform), as well as synthetic databases with Gaussian data and various settings of the standard deviation. We also experiment with uniform queries (each possible range query is made with equal probability) and non-uniform queries (different range queries are made with different probabilities). We also extend our analysis to study the effect of extra load on the system. Furthermore, we extend the Match & Extend algorithm by studying what will happen if not all the possible ranges are queried.

3.1.3 Formal Setting

We consider a database of size n and an attribute with range size N for which range queries (i.e. SQL queries that return all records corresponding to values between $[a, b]$) can be made. The size of query response corresponding to range query $[a, b]$ is denoted by $\big|[a, b]\big|$. In this work each volume is represented by a node in a graph, and we briefly explain how the Graphs are constructed.

Node Each node in the graph is associated with a range $[x, y]$. When we refer to a node $[x, y]$ in the graph, we mean the node with label v_i associated with range $[x, y]$. In this work we refer to each node of the graph by its label. Each node $[x, y]$ is associated with the value $|[x, y]|$, which represents the volume of the range query $[x, y]$.

Edge There is an edge between nodes $[x, y]$ and $[w, z]$ iff there exists a node $[u, v]$ such that $|[x, y]| + |[u, v]| = |[w, z]|$.

The goal of our attack is to reconstruct the entire column corresponding to the field with range size N. Specifically, for each $i \in [N]$, we would like to recover the number of records n_i that take value i in the attribute under inspection. Similar to Grubbs et al. [58], the volumes in the form of $[1, i]$ for $1 \leq i \leq N$ are called "elementary volumes". Note that to fully recover all the ranges in the form $[i, i]$ it is enough to recover "elementary volumes". We use clique finding algorithms to find these "elementary volumes" and the following claim establishes that the "elementary volumes" forms a clique.

Claim 2. *The set of nodes $S := \{[1, i] : i \in [N]\}$ form a clique of size N in the graph defined above.*

Proof. To prove existence of the clique we need to show that for any pair of distinct nodes in the above set S, there is an edge. Let $[1, i]$, $[1, j]$ be a pair of distinct nodes in the set S. Then either $i < j$ or $i > j$. Assume without loss of generality that $i < j$. Then $|[1, j]| = |[1, i]| + |[i+1, j]|$. Since any database record that has value between $[1, j]$ (inclusive), must either have value between $[1, i]$ (inclusive), or have value between $[i+1, j]$ (inclusive). Then by definition of the graph, there is an edge connecting $[1, i]$ and $[1, j]$. Specifically, to see that this indeed follows from the definition of the graph, let $[w, z] = [1, j]$, $[x, y] = [1, i]$ and $[u, v] = [i + 1, j]$. □

To better establish the database reconstruction we continue by presenting an example. Assume we have a database of students with their grade which is according to Table 3.1.

For the sample database presented here, we have $N = 4$, $N' = 10$ and we have

Name	Grade
A1	1
A2	1
A3	1
A4	1
A5	1
B1	2
B2	2
B3	2
B4	2
B5	2
B6	2
B7	2
C1	3
C2	3
C3	3
C4	3
D1	4
D2	4

Table 3.1: Sample Database

the following values for the volumes for each range $|[1,1]| = 5$, $|[2,2]| = 7$, $|[3,3]| = 4$, $|[4,4]| = 2$, $|[1,2]| = 12$, $|[2,3]| = 11$, $|[3,4]| = 6$, $|[1,3]| = 16$, $|[2,4]| = 13$, and $|[1,4]| = 18$.

Figure 3.1 shows the graph constructed for the sample database and each edge is labeled with a range that causes the edge to be formed. Finally if we assume we observe all the volumes, we have the Figure 3.2. The clique is of the form $\{5, 12, 16, 18\}$ and the recovered database is $\langle 5, 7, 4, 2\rangle$. While it is true that in our toy example there are other cliques that has formed such as $\{5, 7, 12, 18\}$ with recovered database of $\langle 5, 2, 5, 6\rangle$ this situation will most likely not going to happen for larger examples. Even in this toy example, if we compute all the ranges for the latter database it is missing a node, namely 16, which means that this recovered database does not correspond to the graph presented in Figure 3.2.

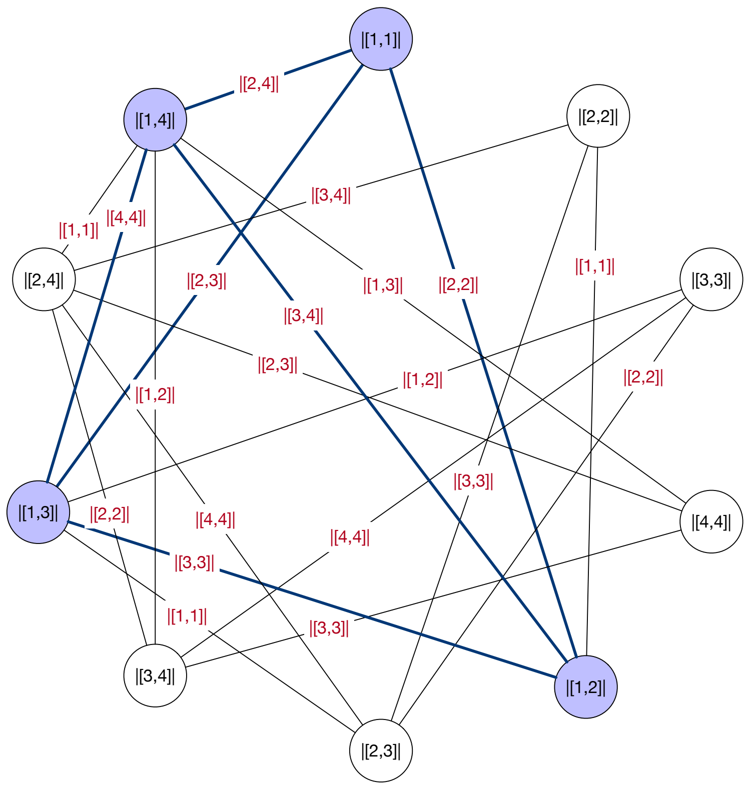

Figure 3.1: Graph constructed for the ranges of sample database

Furthermore, the focus of this work is on "dense" databases, meaning that every possible value from 1 to N is taken by some records in the database. We note that in the searchable encryption setting this is not the typical case since ciphertexts encrypting values between 1 and N are typically sampled from a larger space. However, in this work, our main focus is on cleartext databases and attackers who learn information about them via the cache side-channel. For simplicity, we assume that ranges are always from $1 - N$. However, the result generalizes to any range $a - b$, where database records can take on at most N discrete values within the range. Our attack model assumes that a malicious party can only launch side-channel attacks to reconstruct the database. In particular, we assume that the attacker monitors its read timing from a cache line to deduce useful information about the victim. As discussed, the noise introduced by the cache side-channel makes our

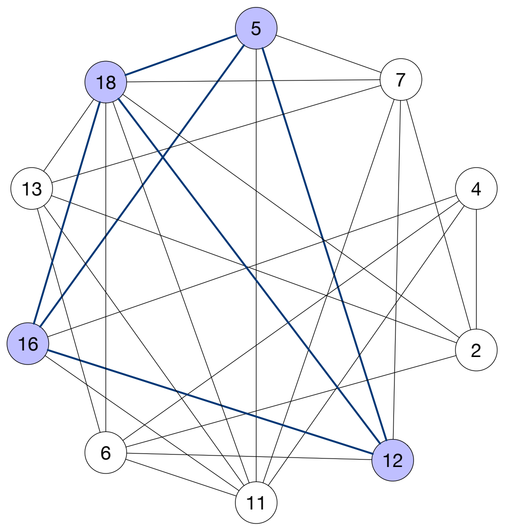

Figure 3.2: Graph constructed from the observed volumes

setting more challenging. We note that Grubbs et al. [58] mentioned a type of side-channel where an attacker intercepts the connection between user and server and counts the TLS packets in order to obtain volumes of range queries, but they did not consider the difficulties that arise when the measurement channel introduces noise into the computed volumes.

3.1.4 Our Contributions

We next summarize the main contributions of this work.

Weaker threat model: Side-channels Prior work considers a threat model of a malicious server that is computing on an encrypted database. We consider an honest server computing on a cleartext (or encrypted) database and a malicious third-party that is co-located with the honest server in the cloud, sharing a cache, and cannot issue queries to the database. The malicious third-party can only obtain information by monitoring the shared cache. In particular, this means that the third-party cannot learn the *exact* volumes of range queries and only obtains *approximate* or *noisy* volumes.

Assessing effectiveness of previous algorithms in the noisy setting We first analyzed the effectiveness of a brute-force attack, similar to the one suggested in the work of Kellaris et al. [72], but adapted to the *noisy* and *dense* database setting. When we ran this version of the brute-force search algorithm, it failed to return a result, even after a day of running. We expected this to be the case, since when the volumes are noisy, there are far more choices that need to be checked in each step of the brute-force search.

We next analyzed the effectiveness of an attack based on clique-finding, as in the work of Grubbs et al. [58]. A graph is constructed based on the observed volumes of the range queries. To construct the graph from *exact* volumes, one first creates nodes with labels corresponding to their volume, i.e. the node with label v_i has volume v_i. There is a connection between node v_i to v_j if there exists a node v_k such that $v_i = v_j + v_k$. Note that by this construction the nodes corresponding to elementary volumes form a clique of size N which can be recovered by clique finding algorithm. The ranges $[1, 1], [1, 2], \dots, [1, N]$ and the full database can then be recovered from this information.

In the noiseless setting we always expect to get a clique of size N; however, in the noisy setting there are multiple edges missing in the constructed graph and so a clique of size N will typically not exist. For example, when we ran the algorithm on our noisy data with $N = 12$, the size of the cliques returned was at most 3. Further, the clique of size 3 no longer corresponds to the volumes of the elementary volumes, and therefore is not useful for (even partial) reconstruction of the database.

Developing algorithms for the noisy setting Whereas Grubbs et al. [58] used exact volumes to reduce the database reconstruction to a clique-finding problem, we begin by reducing the reconstruction problem given with volumes to a **Noise-Tolerant Clique Finding Problem** by introducing a notion called a *noise budget.* Remember in the *exact* volume case, there is a connection between node v_i to v_j if there exists a node v_k such that $v_i = v_j + v_k$. Here, to construct the graph from *noisy* volumes, we create a window, $w(v_k)$, of acceptable values around each leaked volume v_k, where the width of the windows is determined by the noise budget. We place an edge between node v_i and v_j if there exists a node v_k such that $|v_i - v_j| \in w(v_k)$. The clique finding algorithm will return a clique that allows one to recover the volumes, and the full database can then be recovered from this information. An attacker can determine a good setting of the *noise budget* by mounting an attack in a preprocessing stage on a different, known database under same or similar conditions. Specifically, the attacker can first create its own known database (unrelated to the unknown private database). The attacker can then simulate the side-channel attack on the known database on a similar system and compare the recovered approximate/noisy volumes with the correct volumes, and observe by how much they are off, to determine an appropriate *noise budget.* In some cases, incorporating the *noise budget* into the construction of the graph and running the clique-finding algorithm already allows us to successfully reconstruct a fairly accurate database. However, there are some cases where, even after increasing the *noise budget*, the algorithm fails to recover a candidate database (i.e. a clique of size N does not exist). Further, even in cases where increasing the *noise budget* does allow for reconstruction of some candidate database, the accuracy of the candidate database suffers and the run-time increases. We therefore introduce an additional algorithm called **Match & Extend**, which allows successful reconstruction of candidate databases with improved accuracy.

The **Match & Extend** algorithm starts by obtaining a candidate clique from the graph. If the size of the clique is equal to N (the maximum range) we are done. Otherwise, the algorithm looks at all the other cliques present in the graph starting from the largest to the smallest. For each clique, a potential database is recovered. We then pick one of the databases as our base solution and compare it with the other recovered databases. In the **Match** phase, the algorithm looks for the "approximate longest common substring" between two databases. The "approximate" version of the longest common substring considers two substrings equal if their corresponding values are within an acceptable range dictated by the *noise budget*. Two values a and b are "approximately" equal if $|a - b| \leq \min(a, b) \cdot 2 \cdot \texttt{noise budget}$. Then for the databases which have enough overlap with the base solution, the **Extend** phase will compare the non-matching parts of the two solutions and will try to reconcile the volumes in them into one "combined" database.

Finally, in the **Noise Reduction Step**, we use the results of the previous steps along with a closest vector problem (CVP) solver to reconstruct nearly the *exact* original database, despite the noisy measurements. The recovered database of the previous step returns the ranges of the format $[1, 1], [2, 2], \ldots, [N, N]$. We can reconstruct potential volumes for each range with these recovered volumes and for each computed volume we select the closest volume from the initial noisy volume set obtained from the side-channel data. We construct a lattice basis using the known linear dependencies between the volumes of different ranges. The volumes obtained from the side-channel data correspond to the target point for the CVP problem. Using the CVP solver, we find a set of volumes contained in the lattice (so they satisfy the linear dependencies) that are closest to the target point. This "self-correction" technique allows us to recover a better candidate solution for the database.

Launching the side-channel attack We adapt the Flush+Reload technique for obtaining the (approximate) volumes of responses to range queries in SQLite. This allows us to learn a set of *noisy* volumes corresponding to the range queries made by external parties to the database stored by the victim. The monitoring process starts as soon as an activity is detected and continues for the duration of the SQLite query processing. Since the databases we attack are large, the processing takes an extended amount of time, meaning that there are many opportunities for noise to be introduced into a trace. On the other hand, we require accurate measurements for our attack to succeed which is similar to keystroke timing attacks of Gruss et al. [61]. We contrast our setting to other side-channel settings, which typically require *accurate* measurements over a *short* period or, can tolerate *inaccurate* measurements over a *longer* period. For example, side-channel attacks on cryptographic schemes require accurate information to reconstruct the high-entropy keys, but typically take a short period of time, since the keys themselves are short. On the other hand, side-channel attacks for profiling purposes typically monitor an application for longer periods of time, but can tolerate noise well since their goal is just to distinguish between several distinct scenarios.

To achieve high accuracy over a long period of time, we must handle interrupts as well as false positives and false negatives. For interrupts, we must mitigate their effects by detecting and dropping those traces in which an interrupt occurs. There can also be false positives as a result of CPU prefetching, which we show how to detect. False negatives occur if the victim process accesses the monitored line of code after the spy “Reloads” the line, and before the spy "Flushes" the line. We do not directly detect false negatives, but instead show how to deal with them algorithmically.

3.1.5 Overview of Experimental Results

We ran our attacks in five different experimental settings including uniform and non-uniform queries on real databases and synthetic databases which were sampled from Gaussian (Normal) distributions with different standard deviations as well as in two sets of experiments where the system is under heavy load and other cases where some of the range queries are missing. The databases all contained $100,000$ rows with 135 attributes. The synthetic database from the Gaussian distribution has the same number of entries and attributes, but the column on which the range queries are made is sampled from Gaussians with standard deviation of 3 and 4, which represent narrow and wide Gaussians, respectively. The Match & Extend algorithm recovered the database in 100% of the cases within 190 seconds with maximum error percentage of 0.11%.

3.1.6 Related Work

Cache Attacks The first work on cache attack were introduced by Tsunoo et al. [104] that shows a timing attack on MISTY1 block cipher. Later, Osvik et al. [94] presented an attack that allowed the extraction of AES keys. In another early work, Acıiçmez [6] showed an attack that targets instruction cache. Ristenpart et al. [99] demonstrated the possibility of launching cache side-channel attacks in the cloud (as opposed to on a local machine) and they pointed out that such vulnerabilities leak information about the victim VM.

Subsequent work showed how the cache side-channel can be used to extract cryptographic keys for ElGamal [111], AES [69], RSA [110] and recently BLISS [57] (a lattice-based signature scheme). In more recent work, Yarom and Falkner [110] presented a powerful attack using Flush+Reload on the Level 3 cache of a modern processor. They tested their attack in two main scenarios, (a) victim and spy running on two unrelated processes in a single operating system and (b) victim and spy running on separate virtual machines. Another attack of note by Yarom and Benger [109] on ECDSA leaks the nonce which results in signature forgery. A recent work by Moghimi et al. [90] showed the vulnerability of AES encryption in an SGX environment which, prior to this attack, was broadly believed to be secure. Ge et al. [53] surveyed recent attacks and classified them according to which shared hardware device they target. Yan et al. [107] shows the effectiveness of Flush+Reload and Prime & Probe to reduce the search space of DNN architectures. In a more recent type of attack, Hong et al. [65] shows how to perform Deep Neural Network fingerprinting by just observing the victim's cache behavior. In another work by Hong et al. [66], it is shown how to use cache attack to construct the main components of the Neural Network on the cloud.

Database Reconstruction Kellaris et al. [72], motivated by practical implementations of searchable symmetric encryption or order-preserving encryption, studied the effect of auxiliary information on the overall security of the scheme. They identified two sources of leakage (a) access pattern (b) communication volume. They developed a reconstruction attack in which the server only needs to know the distribution of range query. They presented an attack using N^4 queries, where N is the ranges of the value. Lacharité et al. [75] presents various types of attacks: full reconstruction, approximate reconstruction as well as a highly effective attack in which adversary has access to a distribution for the target dataset. Their attacks are based on the leakage of access pattern as well as leakage from the rank of an element. Grubbs et al. [58] present an attack that reconstructs the database given the volumes of the response of range queries. They showed an attack using a graph-theoretic approach and specifically clique finding. Each volume is presented with a node in the graph. They demonstrated properties that hold in practice for typical databases and based on these properties they developed an algorithm which runs in multiple iterations of adding/deleting nodes. Once there is no more addition and deletion to be performed they announce that as the candidate database. They showed that this approach is indeed successful in recovering most of the columns of their example database. In cases where this algorithm could not find any possible result they used a clique algorithm to reconstruct the database, and they showed that clique could help to reconstruct even more instances.

In another line of work regarding searchable encryption, Cash et al. [34] presented leakage models for searchable encryption schemes and presented attacks. Specifically using this leakage they could recover queries as well as the plaintext. Naveed et al. [92] presented a series of attacks on Property-preserving Encrypted Databases. Their attack only used the encrypted column and used publicly known information. They showed an attack which could recover up to a certain attribute for up to 80% of users. Grubbs et al. [60] presented an attack on order-preserving encryption and order-revealing encryption and showed they can reveal up to 99% of encrypted values. Kornaropoulos et al. [73] studied the database reconstruction given leakage from the k-nearest neighbours (k-NN) query. In a follow up work by the same authors, Kornaropoulos et al. [74] extended their previous work by presenting an attack on encrypted database without the knowledge of the data or query distribution. All these attacks are in the encrypted database setting in which each value is encrypted whereas the focus of this work is on databases where the value of each entry is saved in clear text, and an attacker who may only obtain information about the database via side channels.

3.2 Our Attack

In Section 3.2.1 we describe how to recover approximate volumes via the cache side-channel. In Section 3.2.2 we describe how the clique-finding algorithm was used in the prior work of Grubbs et al. [58] to recover a database from noiseless volumes. In Section 3.2.3 we explain our noise-tolerant clique-finding algorithm for our setting, where volumes are noisy. In Section 3.2.4 we present the details of the Match & Extend algorithm which is used for extrapolating volumes that are omitted from the clique. Finally in Appendix A.2 we describe how to use closest vector problem (CVP) solvers to further reduce the noise and improve the overall accuracy of the recovered databases.

3.2.1 Recovering Approximate Volumes

In this section, we explain how to find the lines of code in the SQLite library to monitor in the Flush+Reload attack and how to reduce the noise in our measurements. We will then explain how to recover the approximate volumes.

Victim's Query The victim issues a range query to SQLite database. SQLite returns the relevant entries as it processes the query. These entries are simply saved in a linked list and once SQLite is finished with processing the query, the linked list is returned to the victim.

Detecting Lines to Monitor SQLite stores columns using the BTree data structure. We examined the SQLite program, and by using the `gcov` command we detected lines that are called once in each iteration of a range query. Monitoring the number of times these lines are called allowed us to determine the volume of a query response. It is important to notice that the duration of each query can also be measured and that can also be used as an indicator for the volume. However, this resulted in far greater noise since there was no reliable way to translate time to volume (time to iterate over rows was inconsistent). Hence we decided to explicitly count. To obtain the number of times each line is executed we compiled our library using `-fprofile-arcs` and `-ftest-coverage` flags. We ran the range query command and by using the `gcov` command we counted the number of times each line is executed in files sqlite3.c and main.c, respectively.

We looked for more lines throughout the SQLite program and we chose to simultaneously monitor *two* lines to increase the measurement accuracy. However, note that the attack presented in this work can be extended to other database management system, as long as the volume of the returned query can be obtained through monitoring the I-cache. As also observed by Allan et al. [15], monitoring two lines has the benefit that in case the attack code fails to detect an activity in one of the lines due to overlap between attacker reload and victim access there is still a high probability of seeing activity in the second line. There might be some excessive false positives due to the mismatch of hits for both of the lines and we mitigate for that by considering close hits to be from the same activity.

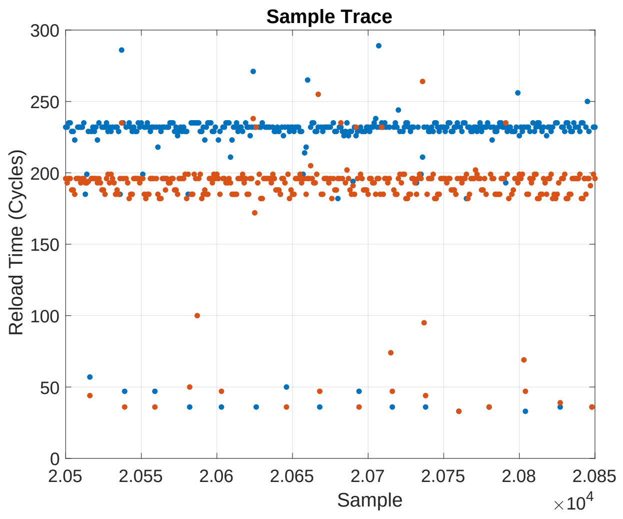

Figure 3.3: The result of running Flush+Reload attack on SQLite. There are two lines being monitored by attacker. The x-axis shows the sample point in which reload occurs and y-axis depicts the amount of time needed to reload the monitored line from memory at that time instance. Since two lines are being monitored, we have two sets of measurement at each time instance. Notice there are some orange lines appearing close to each other, those are because of speculative execution.

Using the Mastik Toolkit Once we detect the lines that leak the volume of the queries, we use the Mastik Toolkit to monitor those lines while SQLite is processing a range query.

Figure 3.3 shows one sample measurement. Two monitored lines are represented by blue and orange color. Mastik-`FR-trace` will automatically start measuring once it detects a hit in either of the monitored lines in the SQLite program. Once there is no more activity detected by Mastik for a while (as set in the `IDLE` flag), it will automatically end the measurement. During the interval where range query execution occurs, there are samples with reload time less than 100 cycles. Those are the samples points in which SQLite accessed the line the attacker is monitoring and hence a small reload time is seen by the attacker. We then count the number of times there is a hit in either a blue or orange measurement. The hit count corresponds to the volume of the query.

Monitoring the relevant cache lines is also crucial to detect when/if range queries

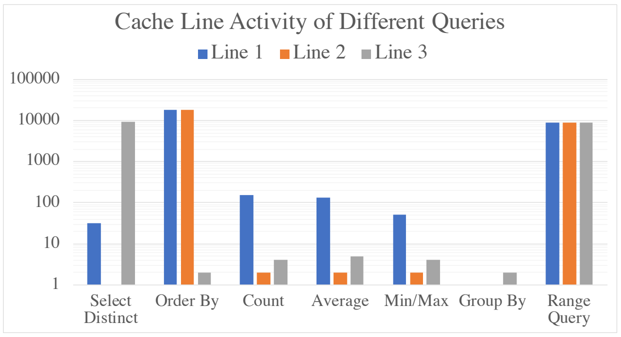

Figure 3.4: Cache line activity when different queries are issued. The cache line activity represents the number of hits detected by the attacker. The figure is in log scale. It can be seen that by looking at the cache activity of these three lines, different queries can be distinguished.

are issued. Figure 3.4 shows the cache activity of three cache lines in the first couple of thousands of samples. The cache line activity represents the number of hits detected by the attacker. Counting the number of hits represents whether or not the victim is using a specific line. We expect to see multiple cache activities in lines related to range queries and for the queries that are not relevant to range queries there is not much activity going on in at least one of the cache lines.

Noise in the Traces The number of hits that we count might be different than the actual value of the volume since measurements are not noiseless. Here we explain some of the sources of noise.

- False Positive: Speculative execution of an instruction causes the memory line to be brought into the cache *before* it is executed. In terms of the Flush+Reload attack, it will still *look like* this instruction was executed, since there will be a fast access. Generally the true hits happen at fixed time intervals. If we see a hit which happens much sooner than the expected time for a hit it is most likely a false positive and we assume it occurred due to speculative execution and do not count it as a hit.

- False Negative: These occur if the victim process accesses the monitored line of code after the spy *Reloads* the line, and before the spy *Flushes* the line. We do not attempt to detect false negatives experimentally, but rather deal with them algorithmically: as will be discussed in Section 3.2.3, we use an asymmetric window around each observed volume to compensate for the fact that true volumes are typically greater than the observed volume. In our experiments we allocate 90% of the window width to the values greater than the observed volume.

Running the Experiment We randomly select and execute range query $[a, b]$ while concurrently monitoring lines using Mastik-`FR-trace` to gather a single trace. We repeat this experiment a number of times in order to gather enough traces. For each trace we count the number of times that either of the lines shows a hit and after mitigating the False Positive issue, we report the number of hits as the volume of the range query for that trace.

Figure 3.5 shows the result of aggregating the volumes reported by the traces. Some volumes are observed far more frequently than others, and those values are saved as an approximation to the expected volumes. In a noiseless setting we expect to see at most $\binom{N}{2} + N$ values (there might be some volumes which correspond to more than one range query). In the noisy setting, there are cases where the trace is "good enough" but the volume is not correct. By aggregating all the traces the effect of those instances will be insignificant and the approximation of correct volumes will stand out. However, the volumes we recover are not exactly the correct volumes from the database. Figure 3.6 shows a closer snapshot of Figure 3.5 for volumes in the range $7700 - 8800$. The red dotted bars represent

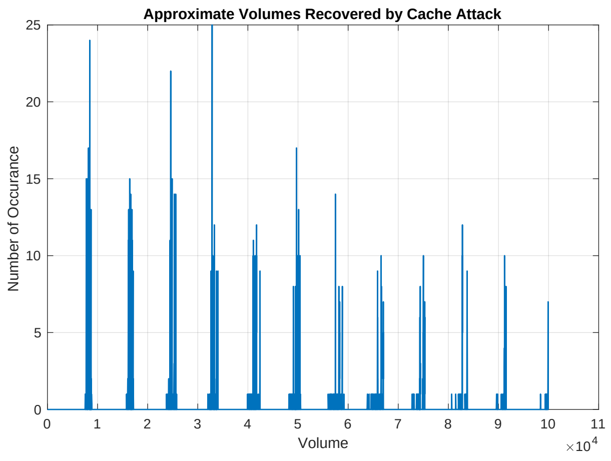

Figure 3.5: Sample noisy volumes recovered by cache attack. The x-axis is the volume and y-axis shows the number of occurrence of that volume. For a sample database, we ran the range query multiple times and for each range query we monitored the cache activity to recover the volume of the range for that query. We repeated this process multiple times and counted how many times a volume occurred.

the actual volume of the range query response, while the blue line shows the approximate volumes recovered by the cache attack. For every correct volume (red line), there is a blue line with some high value close to it.

3.2.2 Clique Finding–Noiseless Volumes

To construct the graph we first explain the clique finding algorithm of Grubbs et. al. [58] and then extend their technique to cover the noisy case. There are two main parts to the Algorithm.

- **Creating Nodes** Given the set of recovered volumes V we create a node for representing each volume and label the node by its corresponding volume, meaning the node v_i has volume v_i.
- **Creating Edges** We create an undirected edge between two nodes $v_i, v_j \in V$ if there exists a node $v_k \in V$ such that $v_i = v_j + v_k$.

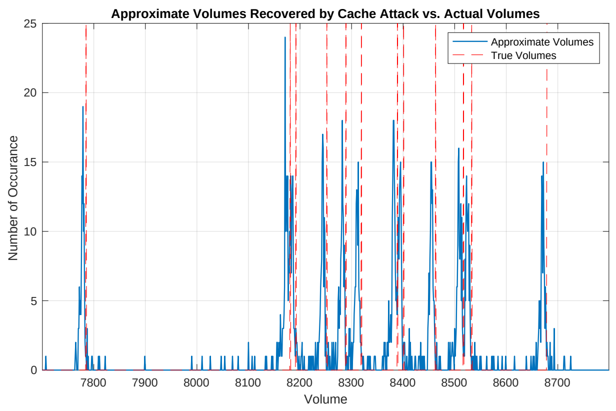

Figure 3.6: A closer look at the recovered volumes. The blue figure is the actual measurement from processing traces from the cache attack. The red bar is the actual volume expected to be observed. It can be seen that the recovered volumes (blue) is approximate version of actual volumes (red).

By running the clique finding algorithm on the constructed graph, one can recover the volumes. Assuming the range of values are from 1 to N there are $\binom{N}{2}+N = \frac{N(N+1)}{2}$ possible ranges, and therefore $\frac{N(N+1)}{2}$ nodes in the graph. Each range $[i,j]$ for $1 \leq i \leq j \leq N$ is represented by a node. The nodes that correspond to ranges of the format $[1,i]$ for $1 \leq i \leq N$, i.e. elementary volumes, form a clique since for each pair of ranges of the form $[1,i]$ and $[1,j]$ for $1 \leq i < j \leq N$ there is another range of the form $[i+1,j]$ for $1 \leq i < j \leq N$, which implies, due to how the graph is constructed, that there is an edge between $[1,i]$ and $[1,j]$, and Claim 2 formally proves it. The clique finding algorithm finds the nodes $[1,i]$ for $1 \leq i \leq N$. To recover the original ranges which are of the form $[i,i]$ for $1 \leq i \leq N$, all that is needed is to sort the nodes based on their labels, which corresponds to their volumes, and subtract them sequentially since $\big|[i,i]\big| = \big|[1,i]\big| - \big|[1,i-1]\big|$ for $1 < i \leq N$.

3.2.3 Clique Finding–Noisy Volumes

In the noisy case considered here, all recovered volumes are close to the correct volumes, but the exact volumes may not have been recovered. Hence, the procedure for noiseless case fails to find the cliques of large enough size. This is because the condition to connect nodes v_i, v_j will almost always fail (even when there should be an edge) since there will not be a third volume v_k such that the equation $v_i = v_j + v_k$ is exactly satisfied. This means that the constructed graph is missing too many edges and the large cliques are not formed. To mitigate the effect of the noise, we modify the second step of the graph generation algorithm i.e. *Creating Edges*.

While the recovered volumes are close to the correct ones, as explained in Section 3.2.1, since the traces are noisy we do not expect to get the exact volumes and we often under-count. We call the ratio of the recovered volume to the correct volume the "noise ratio." In the first step the attacker performs a preprocessing step which involves mounting the attack on a database known to the attacker. The attacker then assesses the quality of the traces to find the approximate value of the "noise ratio." To find it the attacker heuristically looks at the recovered volumes and compares them to the correct volumes they are expecting to compute. Then based on all the noise ratios, the attacker sets a value for "noise budget" which is the mean of the "noise ratio" he observed over all volumes. Once the noise budget is fixed, for each recovered volume the attacker creates a window of acceptable values around it. Assuming the recovered volume is v_i, the attacker creates an asymmetric window around v_i with lower bound and upper bound of $v_i \times (1 - 0.1 \cdot \texttt{noise budget})$ and $v_i \times (1 + 0.9 \cdot \texttt{noise budget})$, respectively. As also mentioned in Section 3.2.1, the window is asymmetric with 90% of its width on the right hand side of it, as the noisy volumes are typically less than the true volumes. For a volume v_i we denote by $w(v_i)$ the window around it. To construct the graph in the noisy case we modify the second step of the algorithm explained in Section 3.2.2 as follows:

- **(Modified) Creating Edges** We create an undirected edge between two nodes $v_i \in V$ and $v_j \in V$ if there exists a node $v_k \in V$ such that $|v_i - v_j| \in w(v_k)$.

In particular, we will say that candidate volumes u and v are "approximately equal" if $\frac{|u-v|}{min(u,v)} \leq$ `noise budget`. As we will show in Section 3.3, using the above algorithm with the just-mentioned modification is in some cases sufficient to approximately reconstruct the database.

3.2.4 Match & Extend

In this section we describe an improvement on the noisy clique-finding algorithm that is used in cases where the noisy clique-finding algorithm fails to find a maximal clique of size N, even with appropriate adjustment of the noise budget.

First, recall that the idea behind the clique finding algorithm is that if we have the volumes of all ranges present in our data, then there must exist a clique in the graph corresponding to the volumes of the ranges $[1, i]$ for $1 \leq i \leq N$. Now, let us assume there is a missing (approximate) volume corresponding to range $[i, j]$. This will result in the missing connection from the node $[1, j]$ to node $[1, i-1]$ as the reason that there had to be a connection was because $\big|[1, j]\big| \approx \big|[1, i-1]\big| + \big|[i, j]\big|$. As a result of this missing volume, the maximal clique of size N will not form. If we run the clique finding algorithm on the data with the missing volume, it will return cliques of size smaller than N and for each of them recover a candidate database. Then the algorithm will merge the information in these smaller databases to form larger ones. In the following we explain the idea of the algorithm with an example. Consider a database with 5 possible values in the range, i.e. $N = 5$, assume the database is $\langle 30, 100, 80, 30, 60\rangle$ (i.e. the database contains 30 records with value 1, 100 records with value 2, etc.). The set of possible values for the volume of a range query is $V = \{30, 60, 80, 90, 100, 110, 130, 170, 180, 210, 240, 270, 300\}$, i.e. $\big|[1, 1]\big| = 30$, $\big|[1, 2]\big| = 130$ and so on. The graph constructed from the these volumes is shown in Figure 3.7 and the maximal clique found by the clique finder algorithm is shown by bold connections. The returned nodes are $\{30, 130, 210, 240, 300\}$ and the reconstructed database is $\langle 30, 100, 80, 30, 60\rangle$. Assume the recovered volumes are noisy and the set of possible values for the volume of a range query is $V = \{29, 58, 79, 89, 98, 108, 128, 160, 178, 209, 239, 268, 299\}$. In Figure 3.8 all the noisy volumes are rather close to their actual values except the volume 160 which is far from the correct one — 170. To construct the graph in this setting we use the algorithm mentioned in Section 3.2.3 and only for the sake of this example we take the window around volume v_i to have lower and upper bound of v_i-1 and v_i+3, respectively. Some connections will be missing as a result of the error in the measurement. For example the connection from node 299 to node 128 is not going to be formed since there is no longer a window which contains 171. If we run the clique finding algorithm on the new graph the result is going to be a clique of size smaller than $N = 5$. As seen in Figure 3.9, the clique finding algorithm returns a clique of size 4 with values $\{29, 128, 209, 239\}$ that results in database $\langle 29, 99, 81, 30\rangle$. Figure 3.10 shows another clique of size 4 with values $\{29, 209, 239, 299\}$ that results in database $\langle 29, 180, 30, 60\rangle$. It can be observed the two databases *approxi-*

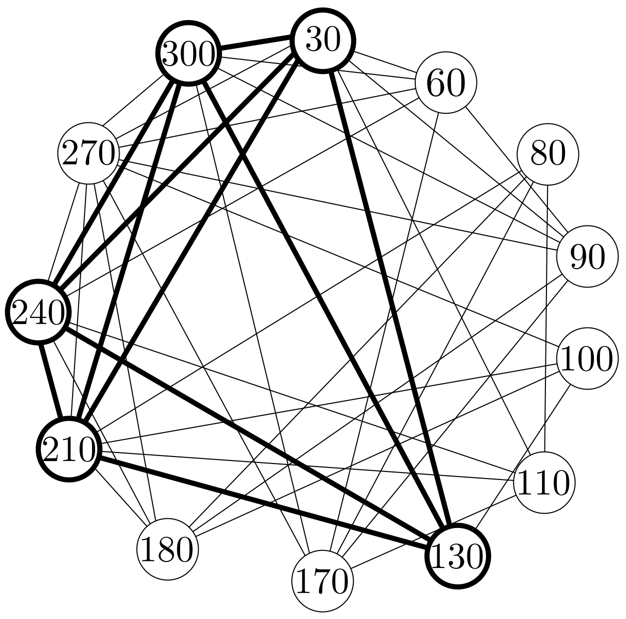

Figure 3.7: The graph constructed from the *exact* volumes of the database $\langle 30, 100, 80, 30, 60\rangle$ and the maximal clique corresponding to that.

mately "match" in some locations, i.e. $\langle 29, 180(99 + 81), 30\rangle$. It is important to note that although in this example 180 is exactly equal to $99 + 81$, this need not hold in general, thus we consider two sequences a match if their corresponding values are *approximately* equal. Having established this long match, we can deduce that value 60 also belongs to the database and we can "extend" the initial candidate to include 60 and return the database $\langle 29, 99, 81, 30, 60\rangle$. In another scenario assume we first detect the database of Figure 3.10 and then we discover the database in Figure 3.9. In that case we can see that we can rewrite the initial candidate, i.e. $\langle 29, 180, 30, 60\rangle$ as $\langle 29, 180 = (99 + 81), 30, 60\rangle$ using the second candidate.

We next describe in detail the main steps taken in the Match & Extend algorithm. The high level steps of the Match & Extend algorithm can be found in Figure 3.11. We note that, in some cases, simply increasing the noise budget allows us to successfully find a clique of size N. However, as we will discuss in Section 3.3, by not increasing the noise

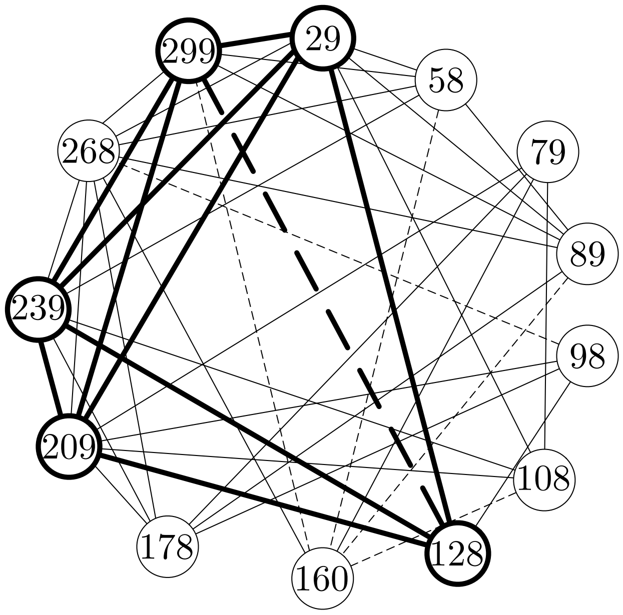

Figure 3.8: The graph constructed from the *approximate/noisy* volumes of the database $\langle 30, 100, 80, 30, 60 \rangle$. An edge in the maximal clique is missing.

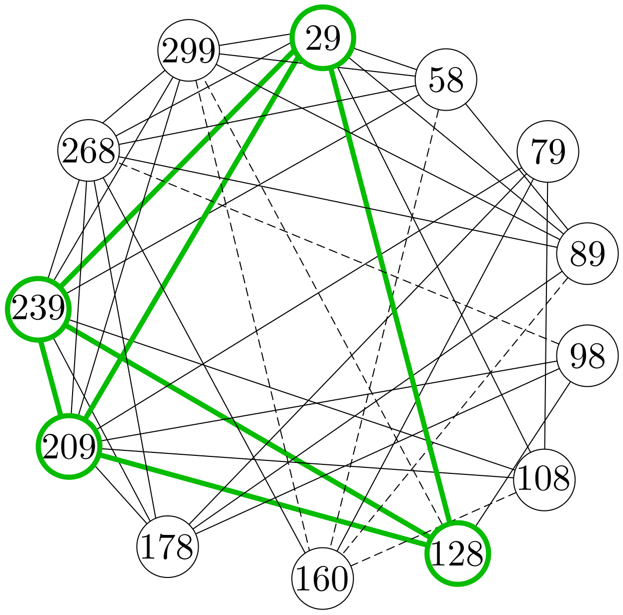

Figure 3.9: A maximal Clique in a graph with *approximate/noisy* volumes.

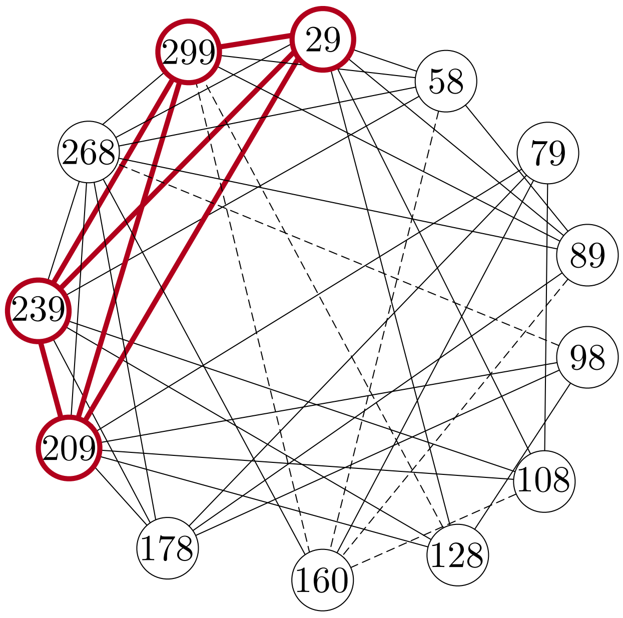

Figure 3.10: Another maximal Clique in a graph with *approximate/noisy* volumes.

budget and instead running the Match & Extend algorithm, we can recover a database that is closer to the true database.

```
1: baseSolution = FindMaximalClique()
2: allCliques = FindRemainingCliques(K, $\ell$)
3: while length(baseSolution) < $N$ do
4:     candidateSolution= FindBestCandidate(allCliques)
5:     baseSolution= Merge(baseSolution, candidateSolution)
6: end while
7: return baseSolution
```

Figure 3.11: Match & Extend Algorithm

FindMaximalClique The first step in the Match & Extend algorithm is to find a maximal clique in the constructed graph. Let K denote the size of a maximal clique recovered in this step. If there is more than one clique with the same maximal size select one of them arbitrarily. Once the clique is found, the corresponding database is computed. We call this database `baseSolution` and the rest of the algorithm will expand this database. If the size of maximal clique found in this step is N we are done, otherwise the Match & Extend algorithm expands the `baseSolution`.

FindRemainingCliques Recover all cliques of size $K, K-1, K-2, \ldots, K-\ell$ and sort them from the largest clique size to the smallest. For each clique, the corresponding database is found and is called `candidateSolution`. The `candidateSolution` is in the form of an ordered list of volumes that correspond to neighbouring ranges of the database. Note that the cliques to be found in this step are not restricted to be from the ranges in the form $[1,1],[1,2],...,[1,K]$ for some K. In fact, and this holds in the noiseless setting too, any set of volumes corresponding to ranges of the form $[i,i_1],[i,i_2],\ldots,[i,i_k]$ where $i \leq i_1 < i_2 < \cdots < i_k$ will form a clique of size k, provided that all the differences of the volumes corresponding to these ranges are present in our data. This fact will enable our algorithm to discover the volumes of different parts of the true database and "merge" those parts to recover the original database.

ApproximateLCSubstring This is a subroutine that is invoked as a part of **Merge** function. Given a `baseSolution` and a `candidateSolution` in form of lists of volumes of neighboring ranges, it finds the longest common substring of these solutions, i.e. the longest contiguous list of volumes where both solutions agree. We call this substring the `commonSub-Solution`. To find it, we use a standard *longest common substring* algorithm with a modification that the elements of the substring need to be only approximately equal (as defined in Section 3.2.3) to the corresponding elements of `baseSolution` and `candidateSolution`. At termination this will return the `commonSub-Solution` and the starting and ending indices of `commonSub-Solution` in the two given solutions.

Merge Given the `baseSolution` and a `candidateSolution` in the form of lists of volumes, attempt to combine the information in them into one larger solution. We refer to this as "merging" the two solutions. The **Merge** function first invokes **ApproximateLCSubstring** to find the approximate longest common substring of the two solutions. After the longest common substring of the two solutions and the locations of this substring in the two solutions are found, there can still be volumes where the `baseSolution` and `candidateSolution` agree, which are not recognized by the **ApproximateLCSubstring**. For example, if the `baseSolution` is $\langle \mathbf{29}, 99, 81, 30\rangle$ and `candidateSolution` is $\langle \mathbf{29}, 180, 30, 60\rangle$, the `commonSub-Solution` may be found as $\langle \mathbf{29}\rangle$, however one can see that the two solutions agree at $\langle 99, 81\rangle$ as well, only in the `candidateSolution` this information appears as the volume of one range $\langle 180\rangle$ which is the union of those two neighboring ranges in `baseSolution`. The merging algorithm identifies such cases and extends the `commonSub-Solution` accordingly. The algorithm searches for occurrences where a volume v_i next to the end of the `commonSub-Solution` in one of the solutions (say in `baseSolution`) is approximately equal to the sum of volumes $u_j, u_{j+1}, \ldots, u_{j+r}$ for $r \geq 0$ next to the same end of the `commonSub-Solution` in the other solution (say `candidateSolution`). In such a case, it extends the `commonSub-Solution` by appending to it $\langle u_j, u_{j+1}, \ldots, u_{j+r}\rangle$, and changing endpoints of the `commonSub-Solution` in `baseSolution` and `candidateSolution`. So in the database given above, the algorithm will look at the neighbors of $\langle \mathbf{29}\rangle$ and discover that $180 \approx 99 + 81$, and extend the `commonSub-Solution` to $\langle \mathbf{29}, \mathbf{99}, \mathbf{81}\rangle$. Then, the algorithm will look at the neighbors of $\langle \mathbf{29}, \mathbf{99}, \mathbf{81}\rangle$ and discover that $30 \approx 30$, extending the `commonSolution` further to $\langle \mathbf{29}, \mathbf{99}, \mathbf{81}, \mathbf{30}\rangle$. It is important to mention that while the values in the example were exactly equal, the algorithm accepts values which are approximately equal as well, meaning that we look for whether $180 \overset{?}{\approx} 99{+}81$ or whether $30 \overset{?}{\approx} 30$. After the `commonSub-Solution` is maximally extended, our two solutions will have the following form: `baseSolution` $= \langle$`pref1`, `comm`, `suff1`$\rangle$ and `candidateSolution` $= \langle$`pref2`, `comm`, `suff2`$\rangle$, where `comm` is the `commonSub-Solution` found as previously explained, and any of the prefixes and suffixes may be empty. The algorithm then will do one of four things: (a) if `pref1` (similarly, `suff1`) is empty, it will extend the `commonSub-Solution` to `comm` $=$ `pref2`$||$`comm` (similarly, `comm` $||$`suff2`), (b) if `pref2` (similarly, `suff2`) is empty, it will extend the `commonSub-Solution` to `comm` $=$ `pref1`$||$`comm` (similarly, `comm` $||$`suff1`), (c) if both `pref1` and `pref2` (similarly, `suff1` and `suff2`) are of length 1, meaning they both contain one volume (say, a and b, with $a < b$), and if the absolute value of the difference of these volumes appears in our volume measurements, then `comm` $= \langle b-a, a\rangle||$`comm` (similarly `comm` $=$ `comm` $||\langle a, b-a\rangle$), (d) if none of the above conditions are satisfied, the algorithm will abort the merge and repeat its steps for another `candidateSolution`. The condition (c) above is for identifying the cases where the volume in, say, `suff1` corresponds to a range $[i, j]$, which includes in itself the range of the volume in `suff2`, which can be $[i, k]$ for $k < j$. If the difference of these two volumes appears in the measured volumes, that difference likely corresponds to the range $[k+1, j]$, so we replace $\langle b\rangle$ ($[i, j]$) by $\langle a, b-a\rangle$ (which is the range $[i, k]$ and range $[k+1, j]$, respectively).

Back to our example database of $\langle 30, 100, 80, 30, 60\rangle$, we had last found the `commonSub-Solution` to be $\langle$**29, 99, 81, 30**$\rangle$. In the merge step, we are going to have `baseSolution` $= \langle$`comm`$\rangle$ and `candidateSolution` $= \langle$`comm`$, 60\rangle$. This falls under the case (a) where `suff1` is empty, so the algorithm appends `suff2` $= \langle 60\rangle$ to the `commonSub-Solution` and returns the solution as `baseSolution` = ⟨**29, 99, 81, 30, 60**⟩.

FindBestCandidate Any time a merge is successful, two solutions are combined into one to create a larger solution. The reason why this larger solution was not initially found by the clique finder is that some volumes or connections in the graph were missing, and so a potential clique corresponding to this solution could not be formed. Every merge of two solutions identifies the number of missing volumes that prevented the combined solution from being found in the first place; in fact, if we were to add those missing volumes to the graph and start the algorithm again, the combined "merged" solution would show up among all listed solutions. Therefore we use the number of missing volumes as a metric for assessing the *goodness* of a candidate solution; if there are few missing volumes, it suggests that the `baseSolution` and `candidateSolution` agree in many volumes of the database, and are thus compatible, whereas if there are many missing volumes, the two solutions likely have different information about the volumes. The **FindBestCandidate** finds the candidate solution among all cliques that has the least number of such missing volumes with respect to being merged with the `baseSolution`.

3.3 Experimental Results

We performed five sets of experiments. The first two experiments (I and II) are for the cases where there is no additional noise during the measurement and the query distribution over all the possible queries are uniform. The third set of experiments (III) in Section 3.3.1 studies the effect of extra load on the system while taking measurements. The fourth set of experiments (IV) in Section 3.3.2 looks at different query distributions.

In the last set of experiments (V) in Section 3.3.3, we look at the effect of missing some volumes due to the fact that some range queries may have never been issued, or due to noise causing a query to be missed entirely. In Experiment I, we first prepare 10 databases from the NIS2008 database, by randomly selecting $100,000$ records. Nationwide Inpatient Sample (NIS) is part of the Healthcare Cost and Utilization Project (HCUP) which is used to analyze national trends in healthcare [5]. The NIS is collected annually and it gathers approximately 5 to 8 million records of inpatient stays. We selected NIS from the year 2008; the full description of each attribute of the database is reported in [5, Table1]. In the first set of experiments we performed uniform range queries on the AMONTH attribute which corresponds to admission month coded from (1) January to (12) December (i.e., each of the possible ranges were queried with equal probability). In the second set of experiments (Experiment II) we sampled the database as follows: For each of the 10 databases from Experiment I, for each record in the database, instead of using the real value for the AMONTH column, we generated synthetic data by sampling a value from a Gaussian (Normal) distribution with mean $\frac{1+N}{2} = 6.5$ and standard deviation of 3 and 4, respectively. So within Experiment II, we considered two data distributions, a "narrow" Gaussian with standard deviation 3 and a "wide" Gaussian with standard deviation 4.

We ran the experiments on a Lenovo W540 Laptop with Intel Core i7-4600M CPU clocking at 2.9 GHz running Ubuntu 16.04. The L1, L2 and L3 caches have capacities 32KB, 256KB and 4MB, respectively. For the SQLite, we use the amalgamation of SQLite in C, version 3.20.1. We heuristically observed that if we gather around 120 measurements for any one range query, the aggregated side-channel measurements will result in a peak corresponding to the approximate volume. Since there are at most 78 different range queries for $N = 12$ we decided to gather around $10,000$ traces to make sure there are enough traces for each range to be able to see a peak for each approximate volume.

We gathered $10,000$ traces corresponding to $10,000$ uniformly chosen range queries for Experiments I and II, i.e. 1 trace for each query. We processed all those traces to obtain the approximate/noisy volumes. On average, gathering $10,000$ traces takes around 8 hours and processing them takes another 3 hours. The experiments and the code to run the Clique-finding algorithm, Match & Extend and noise reduction step can be found in the following GitHub Repository [3].

After processing the measurements, we obtained a set of approximate volumes, on which we then ran noisy clique-finding and Match & Extend, which in turn output reconstructed databases. Figure 3.12 illustrates the quality of the recovered values for the noisy clique-finding and Match & Extend algorithms. The noisy clique-finding algorithm is run with several values for the noise budget while the Match & Extend is run with a fixed noise budget of 0.002. For each of the N values $1, 2, \ldots, N$, we expect to recover a candidate volume, corresponding to the number of records in the database that take that value. For a database with range of size N, we define the success rate as the number of candidate volumes recovered divided by N. For example in our experiments if we recover only 11 candidate values for a database of range of size $N = 12$, then we have a success rate of $11/12$. It is also worth mentioning that the attacker can distinguish a successful attack from the failed one, since N, i.e. the size of the range, is known to the attacker. We define the error rate of a recovered volume as its percentage of deviation from the original volume that it corresponds to. We look at the recovered database and compare it to the original one. For each candidate volume v' that is recovered, we compare it to the corresponding value in the real database, v and report the error rate as $\left(\frac{|v'-v|}{v}\right) \times 100$. So for example, if a recovered volume is 7990 and the actual volume was 8000 we report an error percentage of 0.12%. If the algorithm only recovered 11 values for a database of size 12, we will report the percentage error for the 11 recovered values.

Figure 3.12 and 3.13 show both the success rate and the error percentage for Experiment I and II. For *success rate* (orange line), it can be seen that for the noisy clique-finding algorithm, increasing the noise budget helps to recover more volumes in both experiments. The Match & Extend algorithm, used with a fixed noise budget of 0.002 could recover all the volumes in both of the experiments. For *error percentage* the average percentage of error is marked with a blue dot. The 90% confidence interval is marked with the black marker. The confidence interval indicates that for a new set of experiments with the same setting, we are 90% confident that the average error rate will fall within that interval. For the noisy clique-finding algorithm, increasing the noise budget causes the average error percentage to increase and the confidence interval to grow. In some cases with noise budget 0.005 and 0.006, some of the recovered databases in Experiment I were very far off from the actual databases, causing the error interval in these settings to be much larger than in other settings.

In a nutshell, although it seems that increasing the noise budget helped to achieve higher success rates, since the error percentage grows, the quality of the recovered databases is lower. For the Match & Extend algorithm the average amount of error and the width of error interval is comparable to the noisy clique-finding algorithm with small noise budget but the success rate is much higher. Figure 3.14 shows the average run time as well as

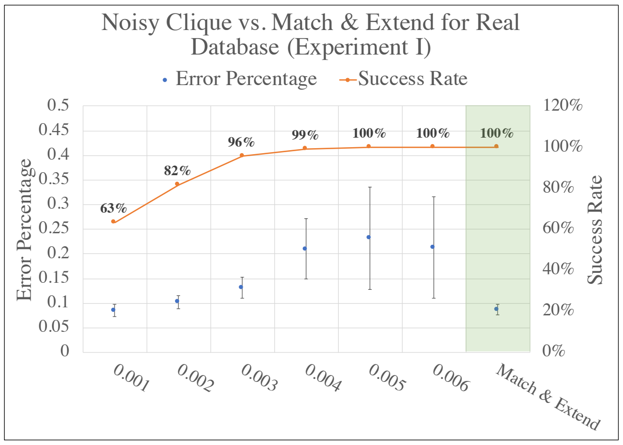

Figure 3.12: Success Rate and Error Percentage for Experiment I

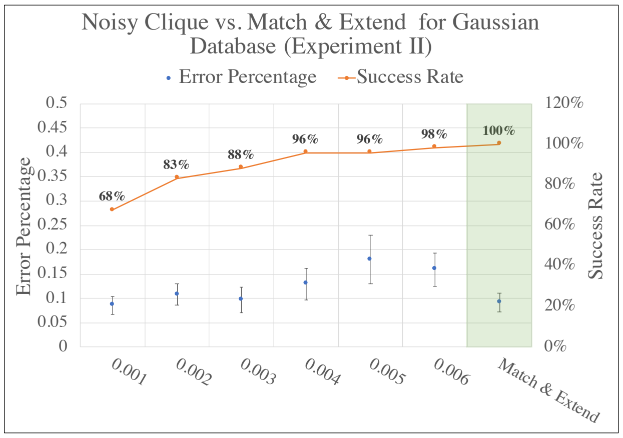

Figure 3.13: Success Rate and Error Percentage for Experiment II

90% confidence interval of the successful database recovery in seconds. It can be seen that the average run time of noisy clique grows with the value of noise budget. The Match & Extend algorithm, however, always uses noise budget of 0.002 and so its average running time remains low.

Table 3.2 compares the performance of the Match & Extend algorithm and the noisy

	Noisy Clique (0.006)	Match & Extend
Error Percentage	0.10 % - 0.27 %	0.07 % - 0.11 %
Run Time (s)	0 - 1900	38 - 190

Table 3.2: Performance Comparison of Noisy Clique with Noise Budget 0.006 vs. Match & Extend Algorithm, with 99% Confidence interval

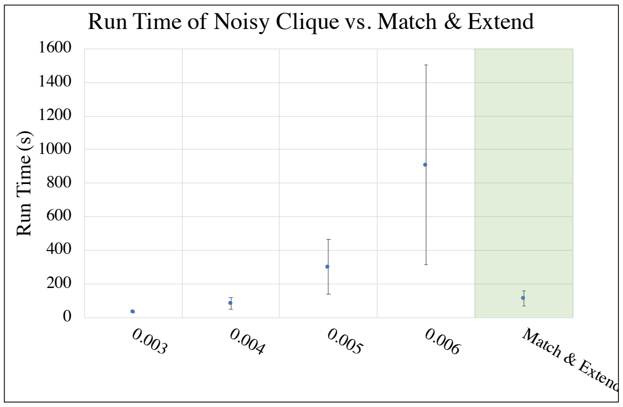

Figure 3.14: Average running time of Noisy Clique for different Noise Budget vs. Match & Extend Algorithm

clique algorithm on successful instances (meaning, the performance of the noisy-clique is taken only over the instances for which the recovered database had the correct size N). The noisy clique-finding algorithm with noise budget 0.006 performs better in terms of success rate than noisy clique-finding with smaller noise budgets, and we select it as a comparable algorithm to Match & Extend algorithm. We are 99% certain that in a new set of experiments with the same setting as presented here, the Match & Extend algorithm would output a result in at most 190 seconds with at most 0.11% error. The noisy clique finding algorithm, on the other hand, would output a result in at most 1900 seconds with at most 0.27% error. We have also analyzed the effect noise reduction step to further reduce the noise, which can be found in Appendix A.3.

3.3.1 Additional Noisy Process Running

For Experiment III, we extend our analysis to the case where there is extra load on the system—i.e. extra processes running—during the time of the attack. To run the noisy processes we use the command `stress -m i`, where i represents the number of parallel

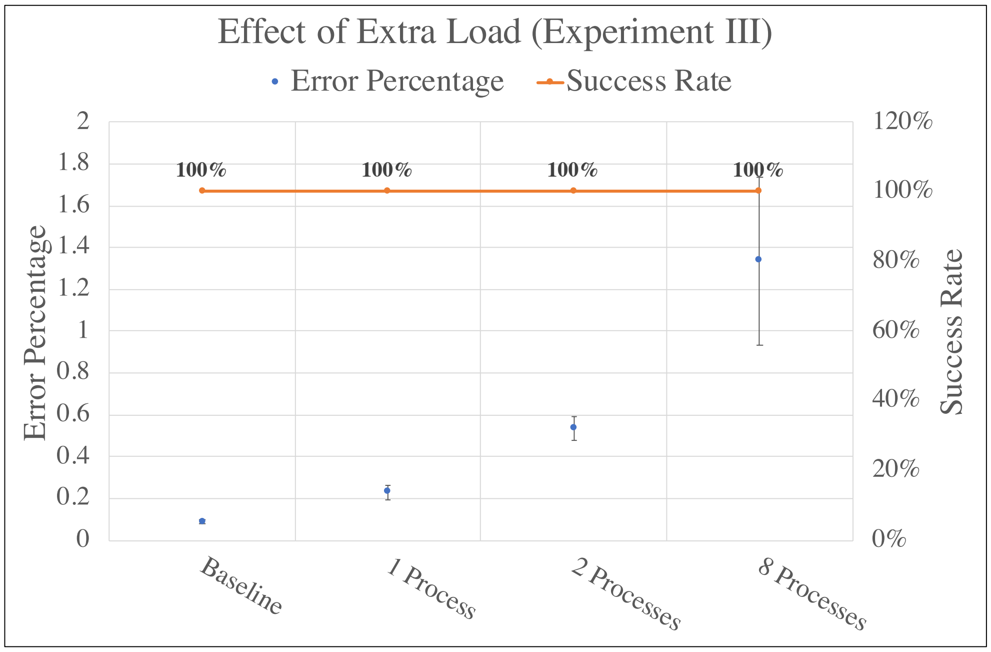

Figure 3.15: Success Rate and error percentage for Experiment III. The system is analyzed under different varying load. The load is increased by adding extra 1,2,8 processes.

threads running with CPU load of 100%. We repeat the experiment for $i \in \{1, 2, 8\}$ and present the results in Figure 3.15. We took 5 databases from Experiment I and obtained new sets of traces while the system had noisy processes running at the background. We recovered all the coordinates of the databases for all cases, so the success rate for all cases remains at 100%. The quality of the recovered databases worsened with heavier loads, however even with 8 noisy processes the average error remains less than 2%.

3.3.2 Non-uniform Query Distribution

As mentioned, in order to be able to detect an approximate volume in our side-channel measurements, we need to observe at least 120 measurements per range query. While we require that each query must be made some minimum number of times, we do not impose that the query distribution must be uniform. This is in contrast to previous work by Kellaris

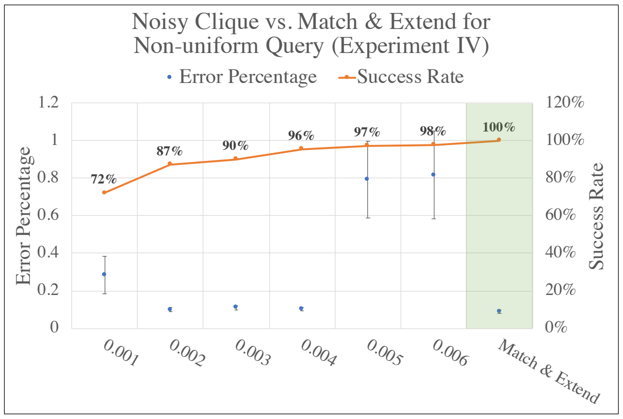

Figure 3.16: Success Rate and Error Percentage for Experiment IV

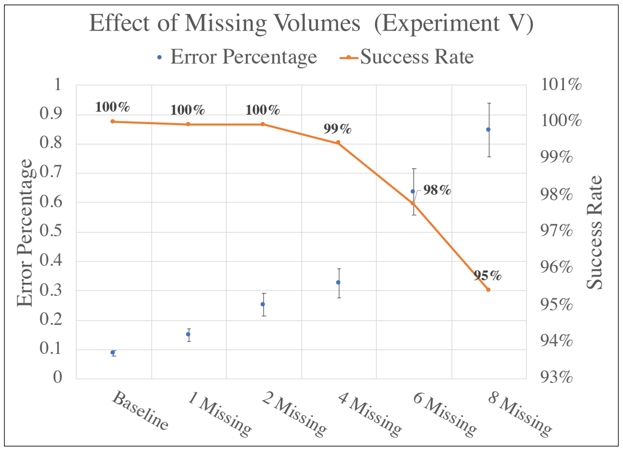

Figure 3.17: Success Rate and Error Percentage for Experiment V

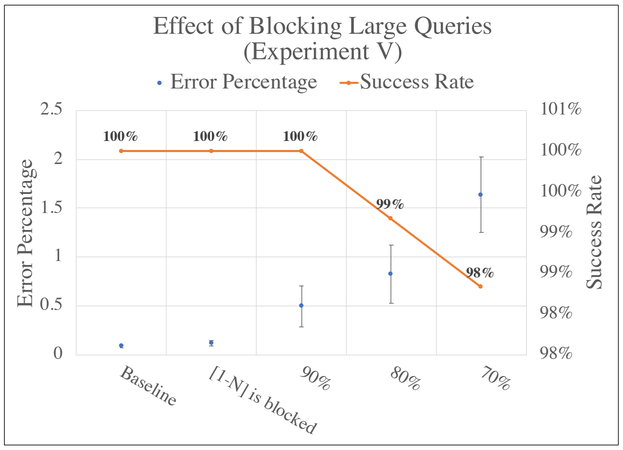

Figure 3.18: Success Rate and Error Percentage for Experiment V

et al. [72] who crucially required uniform query distribution. Our requirement can be seen as the noisy analogue of Grubbs et al. [58], who required each volume to be observed at least once in the *noiseless* setting. To tolerate the noise present in the measurement in our setting, we require each query to be observed at least 120 times. Later, in Section 3.3.3, we will further relax this requirement by showing that the recovery can be done even if some of the range queries are entirely omitted. This can be viewed as a weakening of even the requirements of Grubbs et al. [58].

For Experiment IV, we used the same databases as in Experiment I, but performed non-uniform range queries. We picked 5 database from Experiment I and tested them with 3 sets of non-uniform query distributions which results in 15 scenarios in total. The first query distribution is chosen based on the assumption that queries of the form $[i, i+1]$ are made twice as often as the other queries. The second query distribution assumed that queries in the form of $[i, i+1]$ and $[i, i+2]$ are made twice as often as the other queries. For the last query distribution, we tested the hypothesis that the determining factor for our attack seems to be the ability to identify "peaks" that roughly correspond to the volumes of the range queries (e.g. see Figure 3.5 and Figure 3.6). A challenging query distribution is therefore one which causes one of the peaks to disappear as the peak adjacent to it dominates it. To test our hypothesis, we chose a distribution in which range $[a, b]$ was queried twice as often as $[c, d]$ when ranges $[a, b]$ and $[c, d]$ had close volumes (and therefore close peaks).

Figure 3.16 shows both the success rate and the error percentage for Experiment IV. It can be seen that the noisy clique-finding algorithm exhibits better success rate as the noise budget increases, while the error percentage grows as well. Match & Extend algorithm, however has 100% success rate with low error percentage. We have also analyzed the effect of the noise reduction step to further reduce the noise, which can be found in Appendix A.3.

3.3.3 Missing Queries

For Experiment V, we study the performance of our algorithm when some of the ranges are never queried. We consider two cases. In the first case, we look at a setting in which certain randomly chosen ranges are never queried and in the second case, we consider a setting in which the queries corresponding to the largest volumes are never made. For the first case, we randomly drop $\{1, 2, 4, 6, 8\}$ volumes from the measurements. The performance of Match & Extend algorithm can be seen in Figure 3.17. As more and more volumes are dropped, the success rate of the algorithm decreases. For example, in the case where 8 volumes are missing we recovered around 95% of the databases entries. However, the error percentage grows as more and more volumes are missing, although it remains below 1% error even for the case where 8 volumes are missing. For the second case, we first look at the case where the query $[1, N]$ is blocked, i.e. the attacker is not allowed to query the whole database. In other cases the queries which ask for more than $90\%, 80\%$ and 70% of the database is blocked, respectively. It can be seen in Figure 3.18 that in these cases the success rate remains around 98% and the error percentage of the recovered coordinates stays below 2%.

Chapter 4: New Algorithms for LPN with Sparse Parities

4.1 Introduction

The *(search) Learning Parity with Noise (LPN)* problem with dimension n and noise rate η, asks to recover the secret parity $\mathbf{s}$, given samples $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e)$, where $\mathbf{x} \in \{0,1\}^n$ is chosen uniformly at random, $\mathbf{s} \in \{0,1\}^n$, error $e \in \{0,1\}$ is set to 1 with probability η and 0 with probability $1 - \eta$, and the dot product is taken modulo 2.

While solving a linear system of n equations over $\mathbb{F}_2$ to recover a secret of dimension n can be done in polynomial time via Gaussian elimination, even adding a small amount of noise e renders the above a seemingly hard learning problem, even given a large number of samples. Specifically, the search LPN problem, which typically assumes the noise rate is a small constant, is believed to be hard, with the asymptotically best algorithm (known as BKW) requiring runtime $2^{\Theta(n/\log(n))}$ and $2^{\Theta(n/\log(n))}$ number of samples to recover $\mathbf{s}$ of dimension n. Some evidence of its hardness comes from the fact that it provably cannot be learned efficiently in the so called *statistical query (SQ)* model under the uniform distribution [22, 24].

Though originally arising in the fields of computational learning theory and coding theory, the LPN problem has found numerous applications in cryptography (see e.g. [23, 54, 67, 70] for a partial list of applications) due to the fact that (1) there is a search-to-decision reduction, meaning that the decision version – which is more amenable to cryptographic applications and asks to distinguish $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e)$ from $(\mathbf{x}, b)$, where b is random – is as hard as the search version (which asks to recover $\mathbf{s}$) and (2) the LPN problem is believed to be *quantum-hard*, as opposed to other standard cryptographic assumptions such as discrete logarithm and factoring which are known to have polynomial time quantum algorithms [101].

Variants of the LPN problem have also been considered in the literature: Sparse LPN [25], where the $\mathbf{x}$ vectors in the LPN problem statement are sparse, LPN with structured noise, where the noise across multiple samples is guaranteed to satisfy some constraint [18], and Ring LPN [64]. While typically the error rate is assumed to be constant, LPN with low noise rate has also been considered with applications to cryptography [33]. Indeed, LPN with noise rate even as low as $\Omega(\log^2(n)/n)$ is considered a hard problem [33]. We further note that without loss of generality can assume that the secret is drawn from the same distribution as the noise, as there is a reduction from LPN with secret $\mathbf{s}$ to LPN with secret $\mathbf{e}$, where $\mathbf{e}$ is the error vector obtained after n samples are drawn [17].

In this work we consider LPN with sparse parities (i.e. the "sparsity" or Hamming weight k of the secret vector is significantly less than $\eta \cdot n$, where η is the error rate). We consider both the constant noise and the low noise setting (where the error rate is subconstant). Motivations for considering this variant of LPN include the fact that sparse secrets may be used in practical cryptosystems for efficiency purposes (as is the case for some fully homomorphic encryption implementations [37]), or some bits of the secret may be leaked via a side-channel attack. More generally, analyzing the security of LPN with sparse parities tests the robustness of the standard LPN assumption, since a lack of polynomial-time algorithms in the sparse parities setting (when k is super-constant) would then raise our confidence in the security of the standard setting. We also consider applications of our results to other learning problems, such as learning DNFs and Juntas. Prior work on LPN with sparse parities, has mainly considered obtaining algorithms with runtime $n^{c \cdot k}$ for constant $c < 1$ [56, 105]. This beats the trivial brute-force search with runtime $\binom{n}{k}$ in the regime where $k \ll n$. In this work, our focus is to achieve an algorithm which, for certain regimes of k, beats the prior best algorithms asymptotically *in the exponent*. Since our goal is to achieve asymptotic improvement in the exponent, we will compare our algorithm's runtime against brute-force search and not the prior work of [56, 105], since the latter algorithms are equivalent to brute-force search in terms of asymptotics in the exponent.

4.1.1 Our Results

We obtain new LPN algorithms for sparse parities that improve upon the state-of-the-art in certain regimes, which will be discussed below.

Our first result pertains to the constant noise setting, where the noise rate $\eta \in \Theta(1)$. In the theorem below, $p \in (0, 1)$ is a free parameter that we set later to optimize our runtime.

Theorem 17. *For $\delta \in [0, 1]$, $p \in (0, 1)$, LPN for parities of sparsity k out of n variables and constant noise rate can be learned with total number of samples and total computation time of*

$$\mathsf{poly}\left(\frac{1}{(1-2\eta)^{\sqrt{np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{n}{\delta}) \cdot \left(2^{\frac{np}{\log(np)}} \cdot \log(np)\right)\right),$$

and success probability of $1 - \delta - \left(\frac{16}{(1-2\eta)^{\sqrt{8np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{2n}{\delta}) \cdot \exp(\frac{-pn}{8})\right)$.

By setting the parameter p appropriately, we obtain the following:

Corollary 18. *For sparisty* $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, *where* $c \in o(\log\log(n))$ *and* $c \in \omega(1)$, *the runtime of our new learning algorithm presented in Figure 4.2 is contained in both* $\log(n)^{o(k)}$ *and* $2^{o(n/\log(n))}$, *and it succeeds with constant probability. For this range of* k, *brute-force search requires runtime* $\log(n)^{\Omega(k)}$ *and* BKW *requires runtime of* $2^{\Omega(n/\log(n))}$.

Our second result pertains to the low noise setting, where the noise rate $\eta \in o(1)$. Again, $p \in (0, 1)$ is a free parameter that we set later to optimize our runtime.

Theorem 19. *Assuming parameters are set such that*

$$\log\left(\frac{1}{(1-2\eta)^{2np+2} \cdot p^{2(k-1)}(1-p^2)}\right) \in o(1/\eta \cdot \log(np)),$$

and that $\delta \in [0, 1]$, $p \in (0, 1)$. *LPN for parities of sparsity* k *out of* n *variables and noise rate* $\eta \in o(1)$ *can be learned using* $(2np+1)^2 \cdot \log(n)$ *number of samples, total computation time of* $N :=$ poly $\left(\frac{1}{(1-2\eta)^{2np+2} \cdot p^{2(k-1)}(1-p^2)}\right)$ *and achieves success probability of*

$$1 - \delta - \left(N \cdot \left(2 \cdot \exp(-p \cdot n/8) + \exp(-n/48) + 1/2^{np/4}\right)\right)$$

By setting the parameter p appropriately, we obtain the following:

Corollary 20. *For sparsity* $k(n)$ *such that* $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$, *noise rate* $\eta \neq 1/2$ *such that* $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$, *for* $f(n) \in \omega(1) \cap n^{o(1)}$, *the Learning Algorithm of Figure 4.4 runs in time* $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2k}} \cdot \log(n) \cdot (np)^3\right) \in \left(\frac{n}{k}\right)^{o(k)}$ *with constant success probability. In* *this setting, the running time of brute-force is* $\binom{n}{k} \geq (\frac{n}{k})^k$ *and the running time of lucky brute-force is* $e^{\eta n} \in \left(\frac{n}{k}\right)^{\omega(k)}$.

Finally, applying known reductions to LPN [51] and solving LPN using our algorithm, we also obtain applications to learning other classes of functions such as DNF and Juntas:

- Our algorithm can be applied to learn DNFs of size s and approximation factor ϵ, with asymptotic improvements over Verbeurgt's bound [106] of $O\left(n^{\log \frac{s}{\epsilon}}\right)$, and with negligible failure probability when $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.

- Our algorithm can be applied to learn Juntas of size k with a runtime of $n^{o(k)}$ and a negligible failure probability when $k \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $k \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.

4.1.2 Technical Overview

Fourier Analysis of Boolean Functions Every Boolean function, $f : \{0,1\}^n \to \{0,1\}$—equivalently $f : \{-1,1\}^n \to \{-1,1\}$—can be represented as a linear combination $f(\mathbf{x}) = \sum_{S\subseteq[n]} \hat{f}(S) \cdot \chi_{S,p}(\mathbf{x})$, known as the Fourier representation of f. Typically, we consider the uniform distribution over examples $\mathbf{x}$, in which case $\chi_{S,p}(\mathbf{x})$ is defined as $\prod_{j\in S} \mathbf{x}[j]$ and $\hat{f}(S) = \mathbb{E}_{\mathbf{x}\sim\{-1,1\}^n}\left[f(\mathbf{x}) \cdot \chi_{S,p}(\mathbf{x})\right]$. However, for any product distribution $[p_1, \ldots, p_n]$, where $\mathbb{E}[\mathbf{x}[j]] = p_j$, we can also define $\chi_{S,p}(\mathbf{x}) := \prod_{j\in S} \frac{\mathbf{x}[j]-p_j}{\sqrt{1-p_j^2}}$ and $\hat{f}(S) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_p}\left[f(\mathbf{x}) \cdot \chi_{S,p}(\mathbf{x})\right]$, where $\mathcal{D}_p$ is a product distribution defined over $\{-1,1\}^n$ and is parameterized by its mean vector $[p_1, \ldots, p_n]$. Fourier analysis is a strong tool in computational learning theory for learning under the uniform distribution (and can be extended to product distributions as well). Specifically, the Low Degree Algorithm of [79] guarantees that if most of the Fourier weight of a Boolean function is concentrated on low degree parities (i.e. $\chi_{S,p}$ with small $|S|$), then an approximate version of the function can be reconstructed, *even in the presence of noise.* However, for learning large parities under the uniform distribution Fourier analysis is not useful since for a parity corresponding to secret $\mathbf{s}$ of Hamming weight k, all of the Fourier weight is on a single Fourier coefficient of degree k and searching for this Fourier coefficient would require a brute-force search that enumerates over all possible parities of size at most k. If the distribution is p-biased instead of uniform, however, then the above is no longer the case. Specifically, if we consider a product distributions where the example $\mathbf{x}$ is no longer uniformly random, but each coordinate of $\mathbf{x}$ is set to 0 with probability $1/2 + p/2$ and 1 with probability $1/2 - p/2$ (so the expectation $\mathbb{E}[x[j]] = p$ for each coordinate of $\mathbf{x}$, once it is represented in ± 1 notation), then the Fourier weight is now spread over all parities S such that $\forall j \in S,\ \mathbf{s}[j] = 1$. In particular, this means that by approximately computing the Fourier coefficient of all subsets consisting of a single element $S = \{\mathbf{s}[1]\}, \ldots, S = \{\mathbf{s}[n]\}$, we can distinguish the subsets of size 1 with non-zero versus zero Fourier weight and thus determine all i such that $\mathbf{s}[j] = 1$. We note that when the distribution is p-biased, the magnitude of the Fourier coefficients that we must approximate is of the order p^k, and we will therefore require $\mathsf{poly}((1/p)^k)$ samples to approximate the quantity (even without considering noise). We will see in the following that in order for our approach to improve upon known algorithms, we must consider *sparse parities* with $k \in o(n)$.

Attack Overview Given the above discussion, the main idea of our attack is to convert samples drawn from the uniform distribution to samples drawn from a p-biased distribution and then use Fourier analysis techniques to learn the elements of the parity one by one.

In order for this approach to succeed, our algorithm first needs to generate a sufficient number of p-biased LPN samples, given uniformly random LPN samples. Specifically, the attacker has access to unbiased LPN oracle which outputs samples $\mathbf{x}_i$ and corresponding label b_i such that $b_i = \langle \mathbf{x}_i, \mathbf{s} \rangle + e_i$, noise e_i has rate η meaning that error e_i is 1 with probability η and 0 with probability $1 - \eta$. The attacker will generate new samples $\mathbf{x}'_i$, which are p-biased, and a corresponding label b'_i, with a higher error rate η'. We then approximate the Fourier coefficient of coordinate j, constructed as above, by $\hat{b}_p(\{j\}) := \mathbb{E}_{\mathbf{x}' \sim \mathcal{D}_p}[b' \cdot \chi_{\{j\},p}(\mathbf{x}')]$. The main observation is that for the secret key coordinate j such that $\mathbf{s}[j] = 0$ we have $\hat{b}_p(\{j\}) = 0$ and for the coordinates j such that $\mathbf{s}[j] = 1$ we have $\hat{b}(\{j\}) = (1 - 2\eta') \cdot p^{k-1}\sqrt{1 - p^2}$. The value of $\hat{b}_p(\{j\})$ is estimated by using a sample mean with a sufficient number of generated p-biased samples to approximate the expectation.

We present two algorithms for generating the p-biased samples, each algorithm is appropriate for a different scenario. Specifically, our first algorithm is appropriate for the standard case where the noise rate is constant, while our second algorithm is appropriate for the *low noise* case where the noise rate is sub-constant. After generating the p-biased samples, the Fourier estimation step is similar in both settings. We next elaborate on our algorithm for each of the two settings.

Constant Noise In the case where the noise rate is constant, to generate the p-biased samples, we apply a variant of the BKW algorithm. The BKW algorithm gives an

$2^{O(n/\log(n))}$-time algorithm for the LPN problem that also requires $2^{O(n/\log(n))}$ number of samples. An intermediate step of the BKW algorithm uses access to its LPN oracle to generates samples $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s}\rangle \oplus e')$, where $\mathbf{x}$ is a vector that has all 0's except in a single position, and e' is an error term with higher noise rate than the original error. The key idea of our algorithm is that in order to create p-biased samples, we can choose a random set of coordinates, $R \subseteq [n]$, by including each $i \in [n]$ in the set R independently with probability p, and then run the subroutine of the BKW algorithm on the *smaller set* R, of expected size pn, in order to create a sample $\mathbf{x}$ that is set to 0 for all $i \in R$. Such a sample $\mathbf{x}$ is now distributed identically to a p-biased sample. The error rate increases, but since Fourier analysis is robust against noise, these p-biased samples can still be used to estimate the Fourier coefficients corresponding to singleton sets of the form $S = \{\mathbf{s}[1]\}, \ldots, S = \{\mathbf{s}[n]\}$ to determine the secret $\mathbf{s}$. Crucially, our algorithm gains over simply running BKW on the entire instance because the set of coordinates we run BKW on is of size $O(pn)$ instead of size n. Thus, generating the biased samples runs in time $2^{O(pn/\log(pn))}$ instead of time $2^{O(n/\log(n))}$. When p is subconstant, we achieve an asymptotic gain in the exponent. In contrast, the Fourier estimation step runs in time $\mathsf{poly}((1/p)^k)$, so we must also set p large enough so that this step achieves asymptotic gain in the exponent beyond the brute-force search time of $\binom{n}{k}$. We discuss at the end of the section the regime in which it is possible to set the parameter p so that our algorithm improves asymptotically in the exponent beyond the best known algorithms.

Low Noise When the noise rate is sufficiently low, we can generate p-biased samples using a simpler approach. As before, we randomly select a set $R \subseteq [n]$, by including each

$i \in [n]$ in the set R independently with probability p. Now, instead of running BKW on the coordinates in the set R, we simply choose $O(np)$ number samples (since R has expected size np) from the non-biased oracle and find a linear combination (guaranteed to exist) that sets all the coordinates in R to 0. Again, the noise increases in the generated sample. Nevertheless, we gain over the trivial approach (which instead of p-biasing the oracle simply creates linear combinations that have $\mathbf{x}$ set to all 0 except for in a single coordinate) because the linear combination we generate is over at most $O(np)$ versus $O(n)$ vectors, which in turn guarantees that the noise rate will be lower. However, we note that the above description is a bit inaccurate, since we must include an additional step to ensure that the added noise is independent of the set of samples. We elaborate on these details in Section 4.3.1, Figure 4.3 and Lemma 26. We gain from this technique by choosing p small enough to lower the noise rate but large enough to ensure that the $(1/p)^k$ necessary to estimate the Fourier coefficient still beats brute-force search asymptotically in the exponent.

In the low noise case we further show that we can generate the large number of samples needed for the Fourier analysis using only a *polynomial size* set of examples from the original LPN oracle. In this case, the generated samples will not be i.i.d., but we will use a construction inspired by the *designs* of Nisan and Wigderson to generate an exponentially large set of samples, where each pair of samples from the generated set has low covariance. The details of this design are presented in Section 4.3.1. Further we note that, it is also possible to use a random choice of subsets in place of this design. However, the deterministic procedure allows for bounding the covariance of the newly generated samples which is crucial in our analysis as seen later. This will be enough to then run the Fourier analysis, which requires that one can use random sampling to estimate the mean of a random variable. We can bound the deviation from the mean using Chebyshev's inequality since we guarantee that the covariance between any two distinct samples is small.

Parameters We now discuss the regime of k and η in which we improve on prior algorithms, and how to set the parameter p to achieve the optimal run time. For the constant noise setting, with secret $\mathbf{s}$ with sparsity in the form $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, where $c \in o(\log\log(n))$ and $c \in \omega(1)$, we set $p = 1/\log^{1/(c)}(n)$ to obtain an algorithm that improves upon both brute-force and BKW asymptotically in the exponent. Recall that prior work on LPN with parities of sparsity k reduced the constant in the exponent beyond brute-force, but did not achieve asymptotic improvement in the exponent. In our work we care about asymptotic improvement in the exponent and therefore do not compare against those algorithms. For the low noise setting we show that for sparsity $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$ and the noise rate of $\eta \neq 1/2$ and $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$, for $f(n) \in \omega(1) \cap n^{o(1)}$, by setting $p = \frac{1}{f(n)}$ and $\frac{1}{p} \in \left(\frac{n}{k}\right)^{o(1)}$, our algorithm improves upon both brute-force and "lucky brute-force"–i.e. an algorithm which gathers m samples until it has n noiseless samples with high confidence (where m depends on the noiserate) and then attempts Gaussian elimination with every possible subset of size n, giving runtime $\mathsf{poly}(\binom{m}{n})$–asymptotically in the exponent. To our knowledge, these are the best algorithms when considering asymptotics in the exponent.

Application to DNF and Juntas In addition to parities, the reductions by Feldman et al. [51] provide a way to translate improvements in solving LPN to learning Juntas and DNFs. As such, we present a formulation of our constant noise algorithm that is parameterized according to these reductions, and provide parameter settings such that our algorithm, when applied to learning DNFs or Juntas, yields asymptotic improvements in the exponent. For DNFs, we present an asymptotic result similar to that of [56] in that we improve on Verbeurgt's bound of $O(n^{\log \frac{s}{\epsilon}})$ for learning DNFs of size s with approximation factor ϵ for a different regime of $\frac{s}{\epsilon}$, where $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log \log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, for $c \in n^{1-o(1)}$. Note that for Juntas, we present an algorithm that learns Juntas of k variables in $n^{o(k)}$ time for $k \in \omega\left(\frac{c}{\log n \log \log c}\right)$, and $k \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.

4.1.3 Related Work

Learnig Parity with Noise Blum, Kalai and Wasserman [24] presented the first algorithm that improved upon the trivial $2^{\Omega(n)}$ time algorithm for LPN. They showed that LPN with constant error rate can be learned in slightly subexponential time $2^{O(n/\log n)}$ with the same amount of samples. To date, their algorithm remains the state-of-the-art in terms of asymptotics in the exponent in the constant error rate regime.

Lyubashevsky [82] extended the previous algorithm by Blum et al. [24] and reduced the overall sample complexity. Lyubashevsky developed an algorithm for creating a super-polynomial number of psuedorandom samples from a polynomial number of original samples. Thus, Lyubashevsky traded sample complexity for time complexity. More specifically, the algorithm solved LPN with constant error rate and parities of size n in time $2^{O(n/\log \log n)}$ using only $n^{1+\epsilon}$ samples.

In later work, Bogos et al. [26] presented a unified framework for various improvements and optimizations of BKW. Specifically, they focused on tightening the analysis of several previous works [63, 76] to give more accurate bounds for the time and sample complexity needed to solve the LPN problem. They improved the bounds of the variant of the BKW algorithm proposed by Leviel and Fouque [76] which is based on Walsh-Hadamard transform. Moreover, they analyzed the algorithm by Guo et al. [63] which used a "covering codes" technique to reduce the dimension of the problem. We note that the many of the improvements listed are heuristic in nature, while others provably improve the runtime. We also note that our usage of BKW in our algorithms is compatible with only some of these improvements. We only use the so-called "reduction" phase of the algorithm to generate our p-biased samples. Thus, improvements to this phase, such as covering codes, are applicable whereas others, such as the Walsh-Hadamard transform, are not.

LPN with Sparse Parities Grigorescu et al. [56] showed an improvement of learning sparse parities with noise over brute-force search, which has run time $\binom{n}{k}$. The algorithm ran in time $\mathsf{poly}\left(\log(\frac{1}{\delta}), \frac{1}{1-2\eta}\right) \cdot n^{\left(1+(2\eta)^2+o(1)\right)k/2}$ and had sample complexity of $\frac{k\log(n/\delta)\omega(1)}{(1-2\eta)^2}$ in the random noise setting under the uniform distribution, where η is the noise rate and δ is the confidence parameter.

Valiant [105] showed that the learning parity with noise problem can be solved in time $\approx n^{0.8k}\mathsf{poly}(\frac{1}{1-2\eta})$. He also showed that noisy k-juntas can be learned in time $n^{0.8k}\mathsf{poly}\left(\frac{1}{1-2\eta}\right)$ and r-term DNF can be (ε, δ)-PAC learned in time $\mathsf{poly}\left(\frac{1}{\delta}, \frac{r}{\varepsilon}\right) n^{0.8\log(\frac{r}{\varepsilon})}$, respectively. We note that the improvements of Grigorescu et al. [56] and Valiant [105] do not improve upon the runtime of brute-force search of n^k in terms of asymptotics in the exponent.

Learning DNF and Juntas Mossel et al. [91] showed the first learning algorithm which achieves a polynomial factor improvement over trivial brute-force algorithm which runs time $O(n^k)$. It shows that k-juntas can be learned in absence of noise with confidence $1-\delta$ from uniform random examples with run time of $\left(n^k\right)^{\frac{\omega}{\omega+1}} \cdot \mathsf{poly}\left(2^k, n, \log(1/\delta)\right)$ where $\omega < 2.376$ is the matrix multiplication exponent.

Feldman et al. [50] presented a foundational work for learning both DNFs and Juntas. They developed an oracle transformation procedure that enabled reductions from learning DNFs and Juntas to that of LPN. In addition, Feldman et al. presented a learning algorithm for agnostically learning parities by showing a reduction from learning parities with adverserial noise to learning parities with random noise. With this reduction, they showed that the algorithm by Blum et al. [24] can learn parities with an adverserial noise rate of η in time $O(2^{\frac{n}{\log n}})$. In a follow up work [51], Feldman et al. refined their reductions and included the influence of sample complexity on the the runtime. These reductions have streamlined the process of improving algorithms for learning DNFs and Juntas, as improved algorithms for learning parities can be directly applied to both problems. Both the work of Grigorescu et al. [56], and Valiant [105] were examples of this.

One can also consider natural restrictions to the Junta problem. For monotone Juntas, Dachman-Soled et al. [39] found lower bounds for solving monotone Juntas in the statistical query model. Lipton et al. [80] considered the problem of learning symmetric Juntas and showed they can be learned in $n^{o(k)}$ time. Note here that the symmetry requirement is orthogonal to restrictions on the size of k.

4.2 Constant Noise Setting

In the constant noise setting, our algorithm consists of two steps. First, using a modification of the acclaimed BKW algorithm [24], we implement a p-biased LPN Oracle with noise rate η' and secret value $\mathbf{s}$ which is denoted by $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ and is defined in Section 2.2.3. We present this modification, entitled $\mathsf{BKW_R}$ (BKW restricted to set R), in Section 4.2.1. In Section 4.2.2 we present the integration of our p-biased oracle into the learning algorithm based on Fourier analysis. Finally, in Sections 4.2.3 and 4.2.4, we combine our analysis to present the regime in which we can set the free parameter p in order to improve on both BKW and brute-force search asymptotically in the *exponent*.

4.2.1 $\mathsf{BKW_R}$ Algorithm

As a first step, we present our $\mathsf{BKW_R}$ algorithm in Figure 4.1. The $\mathsf{BKW_R}$ algorithm is given access to an unbiased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ and its goal is to produce a sample that is p-biased. The presented algorithm works similarly to BKW by successively taking linear combinations of samples to produce a sample with all zero entries one “block” at a time. The algorithm accomplishes this by maintaining successive tables such that samples in each table are combined to fill the next table. The number of tables is a parameter of the algorithm denoted $\mathfrak{a}$. The tables $T^{(1)}, \ldots, T^{(\mathfrak{a})}$ are each of size $2^{\mathfrak{b}}$, where $\mathfrak{b}$ is the size of each block, except the last table $T^{(\mathfrak{a})}$ which might have a smaller number of entries, specifically $2^{|R| \bmod \mathfrak{b}}$. Each table $T^{(j)}$ is indexed by the value of the coordinates in the j-th block of $\mathbf{x}|_R$, namely $\mathbf{x}|_R\left[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1\right]$. The element in row i of table j is denoted by $\left[T^{(j)}_i\right]$. Importantly, while the size of R may vary, $\mathfrak{a}$ remains constant each time the algorithm is called. This ensures that a constant number of samples are combined to produce the output. This decouples the noise present in the output from the size of R, ensuring that all generated samples are independent.

Construction of p-biased Oracle given $\mathsf{BKW_R}$ The construction of the p-biased Oracle is quite simple. We sample an index set R where each index is selected independently with probability p. The set R is then passed as input to $\mathsf{BKW_R}$. By bounding the size of the set R, we can ensure that with overwhelming probability $\mathsf{BKW_R}$ outputs a p-biased sample in $2^{O(np/\log(np))}$ time. If the size of the set R exceeds this bound (captured by the event $\mathsf{Event1}$ occurring), the runtime may be longer. Thus, when we invoke $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ multiple times to generate a large number of p-biased samples for the Fourier analysis, we need to ensure that with high probability $\mathsf{Event1}$ never occurs. We bound the probability of $\mathsf{Event1}$ in Theorem 22.

Lemma 21. *The samples $(\mathbf{x}', b')$ outputted by $\mathsf{BKW_R}$ Algorithm with access to $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ are independent and distributed identically to samples drawn from a p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$.*

Proof. The proof can be found in Appendix B.1. □

Theorem 22. *Given access to LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ which gives samples $\mathsf{s} = (\mathbf{x}, b)$, the oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ constructed from $\mathsf{BKW_R}$ requires $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ samples, and $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ runtime with probability at least $1 - 2\exp(-p \cdot n/8)$.*

Proof. The proof can be found in Appendix B.2. □

$\mathsf{BKW_R}$

Result: Sample $(\mathbf{x}', b')$ such that the coordinates of $\mathbf{x}'$, which are defined by set R are set to 0.

```
1:  if $|R| \geq 2np \vee |R| \leq pn/2$ then
2:      $\mathsf{Event1}$ occurs.
3:  end if
4:  Set $\mathfrak{a} := \lceil \log(2np)/2 \rceil$ and $\mathfrak{b} := \lceil |R|/\mathfrak{a} \rceil$
5:  Set $T^{(1)}, \dots, T^{(\mathfrak{a})}$ to empty tables
6:  while True do
7:      Query a new sample from unbiased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$
8:      $j := 1$
9:      while $j \leq \mathfrak{a}$ do
10:         if $\left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right] = \emptyset$ then
11:             $\left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right] := (\mathbf{x}, b)$
12:             break
13:         end if
14:         if $\mathbf{x}|_R\,[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1] \neq 0$ then
15:             $(\mathbf{x}', b') := \left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right]$
16:             $\mathbf{x}'' := \mathbf{x} + \mathbf{x}', \quad b'' := b + b'$
17:             $(\mathbf{x}, b) := (\mathbf{x}'', b'')$
18:         end if
19:         $j := j + 1$
20:     end while
21:     if $j = \mathfrak{a} + 1$ then
22:         break
23:     end if
24: end while
25: $(\mathbf{x}', b') := (\mathbf{x}, b)$
26: return $(\mathbf{x}', b')$
```

Figure 4.1: $\mathsf{BKW_R}$ "Zeroing" Algorithm

4.2.2 Learning Secret Coordinates

In this section we first present the Learning Algorithm in Figure 4.2. The Algorithm starts by sampling num number of samples from a p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$. As the samples are non-uniform, we can apply Fourier analysis technique described in Section 2.2.4.

The Learning Algorithm

The learning algorithm gets access to p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ which returns sample $\mathsf{s}^p = (\mathbf{x}, b)$.

```
1. Initialize $\mathcal{S}, \mathcal{S}' := \emptyset$
2. For $i \in \mathsf{num}$:
   (a) Set $\mathsf{s}_i^p \leftarrow \mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ to be the output sample from $p$-biased LPN Oracle.
   (b) Add $\mathsf{s}_i^p$ to the set $\mathcal{S}$.
3. Use the set $\mathcal{S}$ of $\mathsf{num}$ number of samples to estimate the Fourier coefficient of each coordinate of secret.
   • For each $j \in [n]$, approximate $\hat{b}_p(\{j\}) := \frac{1}{\mathsf{num}} \sum_{i=1}^{\mathsf{num}} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, where each coordinate of $\mathbf{x}_i, b_i$ is switched to $\{-1, 1\}$ from $\mathbb{F}_2$.
   • If $\hat{b}_p(\{j\}) > (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$, add $j$ to $\mathcal{S}'$.
4. Output $\mathbf{s}'$ such that $\mathbf{s}'[j] = 1$ for $j \in [n]$ if $j \in \mathcal{S}'$.
```

Figure 4.2: LPN Algorithm for Constant Noise

Lemma 23. *For $\delta \in [0,1]$, $p \in (0,1)$, the learning algorithm presented in Figure 4.2 uses samples from Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ to estimate the secret value $\mathbf{s}'$. The algorithm runs in time $\frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$, requires $\mathsf{num} = \frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$ number of samples and outputs the correct secret key, i.e. $\mathbf{s} = \mathbf{s}'$ with probability $1-\delta$.*

Proof. The proof can be found in Appendix B.3. □

4.2.3 Combining the Results

Combining the results of Sections 4.2.1 and 4.2.2 we obtain the following theorem:

Theorem 24. *For $\delta \in [0,1]$, $p \in (0,1)$, the Learning Parity with Noise algorithm presented in Figure 4.2, learns parity with k out of n variables with the total number of samples and* *total computation time of*

$$\mathsf{poly}\left(\frac{1}{(1-2\eta)^{\sqrt{np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{n}{\delta}) \cdot 2^{\frac{4np}{\log(2np)}} \cdot \log(2np)\right),$$

and achieves success probability of $1 - \delta - \left(\frac{16}{(1-2\eta)^{\sqrt{8np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{2n}{\delta}) \cdot \exp(\frac{-pn}{8})\right)$.

Proof. Using Lemma 23, we have that the number of p-biased samples required is $\mathsf{num} = \frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$ and using Lemma 21 we have that $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$. From Theorem 22, we have that with probability $1 - 2\exp(-p \cdot n/8)$ each p-biased sample can be obtained by an invocation of the $\mathsf{BKW_R}$ algorithm, which requires $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ samples and $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ runtime with probability $1 - 2\exp(-p \cdot n/8)$. Combining and taking a union bound, we have that the algorithm in Figure 4.2 requires at most $\mathsf{num} \cdot O\left(2^{\frac{4np}{\log(2np)}} \cdot \log(2np)\right)$ samples and run time and succeeds with probability $1 - \delta - (2 \cdot \mathsf{num} \cdot \exp(-p \cdot n/8))$. □

4.2.4 Parameter Settings

We consider the parameter setting for which our algorithm asymptotically outperforms the previous algorithms *in the exponent.* We consider two cases.

- The algorithm has to run faster than a brute-force algorithm which tries all the $\binom{n}{k}$ combination to find the sparse secret. Note that the best algorithms for k-sparse LPN achieve only a constant factor improvement *in the exponent* beyond brute-force search. Since we are concerned with asymptotic improvement in the exponent, these algorithms are equivalent to brute-force search.

- The algorithm should run faster than the BKW algorithm for the length-n LPN problem, as BKW is the asymptotically best algorithm for length-n LPN.

Corollary 25. *For the sparsity $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, where $c \in o(\log\log(n))$ and $c \in \omega(1)$, the runtime of our learning algorithm in Figure 4.2 is contained in both $\log(n)^{o(k)}$ and $2^{o(n/\log(n))}$, with constant failure probability. For this range of k, brute-force search requires runtime $\log(n)^{\Omega(k)}$ and BKW requires runtime of $2^{\Omega(n/\log(n))}$.*

Proof. Setting $1/p = \log^{1/(c)}(n)$ and $k = \frac{n}{\log^{(c+1)/c}(n)}$ in Theorem 24, we find that our LPN Algorithm for constant noise rate presented in Figure 4.2 succeeds with constant probability and has runtime

$$\left(\frac{1}{p}\right)^{2k} \cdot 2^{\frac{4np}{\log(2np)}} = \log(n)^{(1/c)\cdot\frac{n}{\log^{(c+1)/c}(n)}} \cdot 2^{\frac{4n/\log^{1/(c)}(n)}{\log(2n/\log^{1/(c)}(n))}} \in \log(n)^{O((1/c)\cdot k)}.$$

Note that if $c \in \omega(1)$, then our runtime is in $\log(n)^{o(k)}$. On the other hand, if $c \in o(\log\log(n))$ then our runtime

$$\log(n)^{O((1/c)\cdot k)} = 2^{O((\log\log(n)/c)\cdot k)} \in 2^{o(k)} \in 2^{o(n/\log(n))}$$

and so asymptotically beats the above two algorithms *in the exponent* for any $c = c(n)$ that satisfies $c \in \omega(1)$ and $c \in o(\log\log(n))$. Plugging the above parameter into Theorem 24 yields probability of success of $1 - \delta - \mathrm{negl}(n) = 1 - \delta$. $\square$

4.3 Low Noise Setting

In this section we present an improved learning algorithm for the low noise setting. The algorithm will draw only a *polynomial number of samples* from the given LPN oracle, use them to construct a much larger set of p-biased samples that are not independent, but have certain desirable properties, and then present a learning algorithm that succeeds with regard to a set of p-biased samples satisfying these properties.

4.3.1 Sample Partition

In this section we present the SamP algorithm which draws a polynomial-sized set of samples from the original LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$, and uses them to construct a far larger set of p-biased samples that are "close" to being pairwise independent. To achieve this, SamP constructs a large number of subsets of size $2np+1$ from the polynomial-sized set of samples, such that each pair of distinct subsets has at most $t \ll 2np+1$ number of samples in common. Then, from each subset of size $2np+1$, we construct a single p-biased sample $\mathsf{s}^p = (\mathbf{x}', b')$ as follows: First, a random subset $R \subseteq [n]$ of coordinates is chosen, by placing each index $i \in [n]$ in R with independent probability p. Note that with overwhelming probability, $|R| \leq 2np$. Thus, given our set of $2np+1 \geq |R|+1$ samples, we construct a matrix $\mathbf{M}$ that contains the samples as rows and we compute the left kernel of the matrix to find a vector $\mathbf{u}$ to zero out the coordinates of R – i.e. $(\mathbf{u} \cdot \mathbf{M})\,|_R = 0^{|R|}$ and the returned sample is $(\mathbf{x}', b') := \mathbf{u} \cdot \mathbf{M}$. This procedure is denoted by RLK (see Definition 12 for more details). Note that the procedure always succeeds when the size of R is at most $2np+1$. If the size of R is larger than this, a bad event $\mathsf{Event1}$ occurs, and we must draw new independent samples from the oracle. We will later show that Event1 occurs with negligible probability. We show that the samples resulting from distinct subsets are "close" to independent, due to the small intersection of any pair of subsets. We next provide some additional details on the construction and guarantees on independence, before formally describing the algorithm and its properties.

Constructing the subsets with small pairwise intersection Our algorithm given in Figure 4.3 constructs the subsets using the *designs* of Nisan and Wigderson [93]. Their approach works as follows. It first draws $(2np+1)^2$ samples from the original LPN distribution and associates each sample with an ordered pair (x, y) for $x, y \in \mathbb{F}$, for the field $\mathbb{F}$ of size $2np+1$. There are $(2np+1)^t$ polynomials of degree $t-1$ in $\mathbb{F}$, and each subset is associated with a particular polynomial, i.e. the samples contained in a particular subset correspond to the $2np+1$ points that lie on the associated polynomial. Note that the maximum number of subsets that can be constructed is $(2np+1)^t$ and that, furthermore, since any pair of distinct polynomials of degree $t-1$ in $\mathbb{F}$ intersect in at most t points, any two subsets have at most t samples in common. Note that this construction allows at most $\mathsf{maxnum} := (2np+1)^t$ number of p-biased samples to be generated. Looking ahead, in Section 4.3.2 we will present a learning algorithm that requires $O(\log(n))$ such independent sets of samples, each of size at most maxnum to learn the parity function.

Near pairwise independence We note that by construction, the Sample Partition Algorithm SamP presented in Figure 4.3 constructs sets of size $(2np+1)$ such that the intersection of any two sets is at most t for $t \leq (np+1)$. This will allow us to bound

Generating the p-biased samples

Obtain $(2np+1)^2$ independent samples $\mathcal{S} = \{\mathsf{s}_1, \ldots, \mathsf{s}_{(2np+1)^2}\}$ from the unbiased LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$. Run the following setup phase to create sets $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{\mathsf{maxnum}}$ each of size $2np+1$ such that for distinct i, j, $|\mathcal{O}_i \bigcap \mathcal{O}_j| \le t$.

Setup Phase :

1. Consider a Finite Field $\mathbb{F}$ of size $2np+1$. Define a bijection π from $[(2np+1)^2]$ to pairs $(x, y) \in \mathbb{F} \times \mathbb{F}$.
2. Consider all polynomials of degree $t-1$ in the ring $\mathbb{F}[x]$. There are $\mathsf{maxnum} := (2np+1)^t$ such distinct polynomials $\mathsf{poly}_1, \ldots, \mathsf{poly}_{\mathsf{maxnum}}$.
3. For $j \in [\mathsf{maxnum}]$, $\mathcal{O}_j$ contains s_i if and only if $\pi(i) = (x, y)$ and $\mathsf{poly}_j(x) = y$.

```
1: procedure SamP(j)                                    ▷ To respond to the $j$-th query
2:     if $j >$ maxnum then
3:         return $\perp$ and terminate
4:     else
5:         sample a set $R_j$ such that each $i \in [n]$ is selected independently into $R_j$ with
       probability $p$
6:     end if
7:     if $|R_j| \ge 2np \vee |R_j| \le pn/2$ then
8:         Event1 occurs
9:         Sample a fresh set of $|R_j|+1$ samples from the LPN oracle and arrange them in
       rows of matrix $\mathbf{A}$ of size $(|R_j|+1 \times n)$
10:        Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$;     ▷ RLK is defined in
       Section 2.2.5
11:        Go To L1
12:    end if
13:    Select set $\mathcal{O}_j$ and arrange them in rows of matrix $\mathbf{A}$ of size $(2np+1 \times n)$
14:    Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$;     ▷ RLK is defined in
       Section 2.2.5
15:    if $\mathbf{x}'|_{R_i} = 0^{|R_i|}$ for some $i \in [j-1]$ then
16:        Event2 occurs
17:        Sample a fresh set of $2np+1$ samples from the LPN oracle and arrange them in
       rows of matrix $\mathbf{A}$ of size $(2np+1 \times n)$
18:        Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$
19:    end if
20:    L1 : k := 1
21:    $(\mathbf{x}', b') := \mathbf{u} \cdot \mathbf{A}$
22:    while $k < 2np + 1 - \mathsf{weight}(\mathbf{u})$ do       ▷ weight is defined in Section 2.2.5
23:        $b' := b' + \tilde{\mathcal{O}}_\eta$
24:    end while
25:    return $(\mathbf{x}', b')$
26: end procedure
```

Figure 4.3: SamP "Zeroing" Algorithm

the covariance of the errors e'_i and e'_j obtained by taking linear combinations of elements in the sets $\mathcal{O}_i$, $\mathcal{O}_j$. Overall, the set of samples generated by SamP algorithm have certain properties enumerated in the following Lemma.

Lemma 26. *Consider an experiment in which the setup phase is run and two samples* $\mathsf{s}_i^p = (\mathbf{x}'_i, b'_i)$ *and* $\mathsf{s}_j^p = \left(\mathbf{x}'_j, b'_j\right)$ *are generated by running* $\mathsf{SamP}(i)$ *and* $\mathsf{SamP}(j)$ *for distinct* $i, j \leq \mathsf{maxnum}$ *then the following hold:*

1. *Each individual sample* $(\mathbf{x}'_i, b'_i)$ *(resp.* $\left(\mathbf{x}'_j, b'_j\right)$*) outputted is distributed identically to a sample drawn from a p-biased LPN Oracle* $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ *for* $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{2np+1}$.

2. $\mathbf{x}'_i$ *and* $\mathbf{x}'_j$ *are pairwise independent*

3. *Recall that* $b'_i = f_{\mathbf{s}}(\mathbf{x}'_i) + e'_i$ *and* $b'_j = f_{\mathbf{s}}(\mathbf{x}'_j) + e'_j$. *Then*

$$\mathrm{Cov}\left[e'_i, e'_j\right] \leq (1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}.$$

Proof. The proof can be found in Appendix B.4. □

Finally, we analyze the runtime and sample complexity for each invocation of SamP.

Theorem 27. *Given access to LPN Oracle* $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ *which gives samples* $\mathsf{s} = (\mathbf{x}, b)$*, the* SamP *algorithm requires* $O\left((np)^2\right)$ *samples in total, and* $\mathsf{poly}(np)$ *runtime per invocation with probability at least* $1 - 2\exp(-p \cdot n/8) - (np)^t \cdot \exp(-n/48) - (np)^t \cdot 1/2^{np/4}$.

Proof. The proof can be found in Appendix B.5. □

4.3.2 Learning Secret Coordinates

In this section we present our Learning Algorithm in Figure 4.4. The input to the algorithm is $8\log(n)$ independently generated sets of p-biased samples with the properties given in Lemma 26. The algorithm uses the p-biased samples to estimate the values of the Fourier Coefficients of the target function.

The Learning Algorithm

The learning algorithm starts by having access to $8\log(n)$ sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{8\log(n)}$ of randomly generated samples. Each set of samples is independent and satisfies the properties given in Lemma 26.

```
1. Initilizate set $\mathcal{S}' := \emptyset$.
2. For $j \in [n]$
   • $\mathsf{count} := 0$
   • $T := 8\log(n)$
   • For $i' \in T$:
      (a) Use the set $\mathcal{S}_{i'}$ of num number of samples to approximate $\hat{b}_p(\{j\}) := \frac{1}{\mathsf{num}}\sum_{i=1}^{\mathsf{num}} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, where each coordinate of $\mathbf{x}_i, b_i$ is switched to $\{-1,1\}$ from $\mathbb{F}_2$.
      (b) If $\hat{b}_p(\{j\}) > (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$, $\mathsf{count} := \mathsf{count} + 1$
   • if $\mathsf{count} \geq T/2$
      – add $j$ to $\mathcal{S}'$
3. Output $\mathbf{s}'$ such that $\mathbf{s}'[j] = 1$ for $j \in [n]$, if $j \in \mathcal{S}'$.
```

Figure 4.4: LPN Algorithm for Low-Noise

Lemma 28. *For $\delta \in [0,1]$, $p \in (0,1)$, given as input $8\log(n)$ independent sets of samples $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{8\log(n)}$ each of size $\mathsf{num} := O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and each satisfying the properties given in Lemma 26 for some $t \in \Theta(1/\eta)$, the Learning Algorithm presented in Figure 4.4 runs in time $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and outputs the correct secret key, i.e. $\mathbf{s} = \mathbf{s}'$ with probability $1-\delta$.*

Proof. The proof can be found in Appendix B.6. □

4.3.3 Combining the Results

Combining the results of Sections 4.3.1 and 4.3.2 we obtain the following theorem:

Theorem 29. *Assuming parameters are set such that*

$$\log\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \in o(1/\eta \cdot \log(np)), \tag{4.1}$$

and with $\delta \in [0,1]$, $p \in (0,1)$, the Learning Parity from Noise Algorithm presented in Figure 4.4, learns parity with k out of n variables and noise rate η using $(2np+1)^2 \cdot \log(n)$ number of samples, total computation time of $N := \mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and achieves success probability of

$$1-\delta-\left(N\cdot\left(2\cdot\exp(-p\cdot n/8)+\exp(-n/48)+1/2^{np/4}\right)\right)$$

Proof. Using Lemma 28, we have that, for some $t \in \Theta(1/\eta)$, the number of p-biased samples with the following properties needed to succeed with probability $1-\delta$ is $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$. From Theorem 27, we have that as long as $\mathsf{num} = \mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \leq \mathsf{maxnum} = (2np+1)^t$ we can generate the required samples using $(2np+1)^2$ samples from the unbiased LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$, and with $\mathsf{poly}(np)$ runtime per sample, with probability at least $1-2(np)^t\cdot\exp(-p\cdot n/8)-(np)^t\cdot\exp(-n/48)-(np)^t\cdot 1/2^{np/4}$. The fact that num and maxnum satisfy the above constraint is guaranteed by the assumption in the theorem on the setting of parameters and the fact that $t \in \Theta(1/\eta)$.

Combining and taking a union bound, we have that the algorithm in Figure 4.4 requires $(2np+1)^2 \cdot 8\log(n)$ samples, has run time $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)} \cdot \log(n)\right)$, and succeeds with probability $1-\delta-\left(N \cdot \left(2\cdot\exp(-p\cdot n/8)+\exp(-n/48)+1/2^{np/4}\right)\right)$. □

4.3.4 Parameter Settings

We consider the parameter setting for which our algorithm's runtime asymptotically outperforms the previous algorithms' runtime *in the exponent*. We consider two cases.

- The algorithm has to run faster than a brute-force algorithm which tries all the $\binom{n}{k}$ combinations to find the sparse secret. Note that there are known algorithms that improve upon brute-force search, but the improvement is a *constant factor in the exponent*. Since we are concerned with asymptotic improvement in the exponent, these algorithms are equivalent to brute-force search.

- The algorithm should run faster than the algorithm which just gets lucky and gets n noiseless samples, we call this algorithm "lucky brute-force". For this algorithm to succeed, it needs $\frac{n}{1-\eta}$ samples from LPN Oracle to ensures that there are approximately n noiseless samples. The next step is to just randomly select n out of these $\frac{n}{1-\eta}$ samples and try Gaussian elimination on them. The run time of such an algorithm for small η can be approximate by $e^{\eta n}$.

Corollary 30. *For sparsity $k(n)$ such that $k=\frac{1}{\eta}\cdot\frac{\log(n)}{\log(f(n))}$, noise rate $\eta \neq 1/2$ such that $\eta^2=\left(\frac{\log(n)}{n}\cdot f(n)\right)$for $f(n)\in\omega(1)\cap n^{o(1)}$, the Learning Algorithm of Figure 4.4 runs in time $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2k}}\cdot\log(n)\cdot(np)^3\right)\in\left(\frac{n}{k}\right)^{o(k)}$ with constant probability. In this setting, the running time of brute-force is $\binom{n}{k}\geq\left(\frac{n}{k}\right)^k$ and the running time of lucky brute-force is*

$e^{\eta n} \in \left(\frac{n}{k}\right)^{\omega(k)}$.

Proof. For k, η defined as above, we choose the biased $p = \frac{1}{f(n)}$ and $\frac{1}{p} \in \left(\frac{n}{k}\right)^{o(1)}$, we have constraint (4.1) from Theorem 29 satisfied as follows:

$$\log\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \approx 4np\eta + 2k\log(\frac{1}{p})$$
$$\in o(1/\eta \cdot \log(n)) \in o(1/\eta \cdot \log(np)),$$

the runtime of the Learning Algorithm of Figure 4.4 is bounded by

$$\frac{1}{(1-2\eta)^{2np}p^{2k}} \cdot \log(n) \cdot O\left((np)^3\right) \approx e^{4np\eta} \cdot \left(\frac{1}{p}\right)^{2k} \cdot \log(n) \cdot O\left((np)^3\right)$$
$$\in e^{o(k)\cdot\log(n/k)} \cdot \left(\frac{n}{k}\right)^{o(k)} \cdot \log(n) \cdot o(n^3)$$
$$\in \left(\frac{n}{k}\right)^{o(k)},$$

which outperforms brute-force and lucky brute-force under the same parameter settings. Plugging the above parameters into Theorem 29 yields probability of success of $1 - \delta - \text{negl}(n)$. □

4.4 Learning Other Classes of Functions

In the following we apply our LPN algorithms from Section 4.2 to learn other classes of functions. First, let us look at the reduction from learning DNFs to learning noisy parities.

Theorem 31 (Theorem 2 in [51]). *Let $\mathcal{A}$ be an algorithm that learns noisy parities of*

k variables on $\{0,1\}^n$ for every noise rate $\eta < 1/2$ in time $T(n, k, \frac{1}{1-2\eta})$ and using at most $S(n, k, \frac{1}{1-2\eta})$. Then there exists an algorithm that learns DNF expressions of size s in time $\tilde{O}\left(\frac{s^4}{\epsilon^2} \cdot T(n, \log B, B) \cdot S(n, \log B, B)^2\right)$, where $B = \tilde{O}(s/\epsilon)$.

We are interested in determining the parameter range for which our algorithm yields an asymptotic improvement over the state of the art in the *exponent*. The work of Grigorescu [56] is the current state-of-the-art. They present an improvement of the bound from [106] of $2^{O(\log(n)\log\frac{s}{\epsilon})}$ for $\frac{s}{\epsilon} \in o\left(\frac{\log^{1/3} n}{\log\log n}\right)$. As we are similarly applying the reductions from Feldman, our algorithm yields a similar improvement on the bounds in [106] for a different range of $\frac{s}{\epsilon}$.

Note the reduction in Feldman [51] relates the ratio of the size of the DNF and its approximation factor to both the noise rate and sparsity of the parity function. Thus, the parameter range for which our algorithm is optimal will be expressed in terms of this ratio.

We begin by extending the runtime analysis of our algorithm from Section 4.2, which dealt with the constant noise setting, to the arbitrary noise $\eta < 1/2$.

Theorem 32. *The learning algorithm described in Figure 4.2 has a runtime of*

$$T\left(n, k, \frac{1}{1-2\eta}\right) = \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

and requires

$$S\left(n, k, \frac{1}{1-2\eta}\right) = \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

LPN samples in the high noise setting, and achieves a success probability of

$$1 - \delta - \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{16\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2} e^{\frac{-np}{8}}$$

where $\mathfrak{a}\mathfrak{b} = np$.

Proof. The proof follows directly from Theorem 24. Instead of fixing a value for $\mathfrak{a}$ and $\mathfrak{b}$, we let them remain free parameters. As well, we no longer make assumptions on the noise rate η. Thus, we start with the runtime in terms of η'.

$$\begin{aligned} T(n,k,\eta') &= \frac{8\ln(2n/\delta)}{(1-2\eta')^2 p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right) \\ T(n,k,\eta) &= \frac{8\ln(2n/\delta)}{(1-2\eta)^{2^{\mathfrak{a}+1}} p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right) \\ T\left(n,k,\frac{1}{1-2\eta}\right) &= \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right) \end{aligned}$$

The sample complexity of the algorithm is equal to its runtime complexity, and thus we just need to consider the success probability. In the high noise setting, the p-biased LPN oracle is called $\mathsf{num} = \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2}$ times, and the success probability calculation follows the same formula from Theorem 24. □

As we are concerned with asymptotic improvement in the *exponent* of the runtime we will take the logarithm of the runtime and compare it to the state of the art for learning DNFs and Juntas.

Corollary 33. *The learning algorithm described in Figure 4.2 learns DNFs of size s and approximation factor ϵ, with asymptotic improvements over Verbeurgt's bound [106]* *of* $O\left(n^{\log\frac{s}{\epsilon}}\right)$*, and with negligible failure probability when* $\log\frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log\log c}\right)$*, and* $\log\frac{s}{\epsilon} \in n^{1-o(1)}$*, where* $c \in n^{1-o(1)}$*.*

Note here that the parameter regime in Corollary 33 requires setting the free parameters of the learning algorithm differently than in the constant noise setting. In order to minimize the runtime of the $\mathsf{BKW}_{\mathsf{R}}$ step of the algorithm in the high noise setting, the value for $\mathfrak{a}$ must be changed from the description in Section 4.2. Thus we set $\mathfrak{a} = (1/2)\log\log(np)$. This change necessitates considerations for δ, the Fourier analysis confidence. This ensures that the failure probability of the full algorithm remains small, even after increasing the number of samples required. We set $\delta = 2^{-n}$. The free parameter p is set to $n^{-o(1)}$ to satisfy asymptotic requirements. These parameters are set similarly for Corollary 35.

Aside from DNFs we can also use our LPN algorithm to learn Juntas. By applying Feldman's reduction we are able to yield an algorithm that, for certain ranges for k, is able to improve on the $O(n^{0.7k})$ runtime cited in [105] asymptotically, not just by reducing the constant factor in the exponent.

Theorem 34 (**Theorem 3 in [51]**)**.** *Let* $\mathcal{A}$ *be an algorithm that learns parities of* k *variables on* $\{0,1\}^n$ *for every noise rate* $\eta < 1/2$ *in time* $T(n, k, \frac{1}{1-2\eta})$*. Then there exists an algorithm that learns* k*-juntas in time* $O\left(2^{2k}k \cdot T(n, k, 2^{k-1})\right)$*.*

Corollary 35. *The learning algorithm described in Figure 4.2 learns Juntas of size* k *with a runtime of* $n^{o(k)}$ *and a negligible failure probability when* $k \in \omega\left(\frac{c}{\log n \log\log c}\right)$*, and* $k \in n^{1-o(1)}$*, where* $c \in n^{1-o(1)}$*.*

Chapter 5: (In)Security of Ring-LWE Under Partial Key Exposure

5.1 Introduction

There has been a monumental effort in the cryptographic community to develop "post-quantum" cryptosystems that remain secure even in the presence of a quantum adversary. One of the foremost avenues for viable post-quantum public key cryptography is to construct schemes from the Ring-Learning with Error (RLWE) assumption—currently 3 out of 26 of the second round NIST submissions are based on assumptions in the ring setting. RLWE is often preferred in practice over standard LWE due to its algebraic structure, which allows for smaller public keys and more efficient implementations. In the RLWE setting, we typically consider rings of the form $R_q := \mathbb{Z}_q[x]/(x^n + 1)$, where n is a power of two and $q \equiv 1 \mod 2n$. The (decisional) RLWE problem is then to distinguish $(a, b = a \cdot s + e) \in R_q \times R_q$ from uniformly random pairs, where $s \in R_q$ is a random secret, $a \in R_q$ is uniformly random and the error term $e \in R$ has small norm. A critical question is whether the additional algebraic structure of the RLWE problem renders it less secure than the standard LWE problem. Interestingly, to the best of our knowledge—for the rings used in practice and practical parameter settings—the best attacks on RLWE are generic and can equally well be applied to standard LWE [95]. In this work, we ask whether improved attacks on RLWE are possible when partial information about the RLWE secret is exposed, though the secret retains high entropy.

The NTT Transform One key method for speeding up computations in the RLWE setting is usage of the *NTT transform* (similar to the discrete Fourier transform (DFT), but over finite fields) to allow for faster polynomial multiplication over the ring R_q. Specifically, applying the NTT transform to two polynomials $\vec{p}, \vec{p'} \in R_q$—resulting in two n-dimensional vectors, $\widehat{\vec{p}}, \widehat{\vec{p'}} \in \mathbb{Z}_q^n$—allows for *component-wise* multiplication and addition, which is highly efficient. In this work, we consider leakage of a fraction of NTT coordinates of the RLWE secret. Since the RLWE secret will typically be stored in NTT form (to facilitate fast computation), [11, 14] leakage of coordinates of the NTT transform is a natural model for partial key exposure attacks.

This Work The goal of this work is to initiate a study of partial key exposure in RLWE based cryptosystems and explore both positive and negative results in this setting. Specifically, we (1) define search and decision versions of Leaky RLWE assumptions, where the structured leakage occurs on the coordinates of the NTT transform of the RLWE secret; (2) present partial key exposure attacks on RLWE, given 1/4-fraction of structured leakage on the secret key; (3) present a search to decision reduction for the Leaky RLWE assumptions; and (4) propose applications of the decision version of the assumption to practical RLWE-based cryptosystems.

5.1.1 Leaky RLWE Assumptions–Search and Decision Versions

We next briefly introduce the search and decision versions of the Leaky RLWE assumptions. For $\vec{p} \in R_q := \mathbb{Z}_q[x]/(x^n + 1)$, we denote $\widehat{\vec{p}} := \mathsf{NTT}(\vec{p}) := (\vec{p}(\omega^1), \vec{p}(\omega^3), \ldots, \vec{p}(\omega^{2n-1}))$, where ω is a primitive $2n$-th root of unity modulo q, and is guaranteed to exist by choice of prime q, s.t. $q \equiv 1 \mod 2n$. Note that $\widehat{\vec{p}}$ is indexed by the set $\mathbb{Z}_{2n}^*$.

The search version of the RLWE problem with leakage, denoted Leaky R-SLWE, is parametrized by $(n' \in \{1, 2, 4, 8, \ldots n\}, \mathcal{S} \subseteq \mathbb{Z}_{2n'}^*)$. The goal is to recover the RLWE secret $\vec{s} = \mathsf{NTT}^{-1}\left(\widehat{\vec{s}}\right)$, given samples from the distribution $D^{\vec{s}}_{real,n',\mathcal{S}}$ which outputs $\left(\widehat{\vec{a}}, \widehat{\vec{a}} \cdot \widehat{\vec{s}} + \widehat{\vec{e}}, [\widehat{s}_i]_{i \equiv \alpha \bmod 2n' \mid \forall \alpha \in \mathcal{S}}\right)$, where $\vec{a}$, $\vec{s}$, and $\vec{e}$ are as in the standard RLWE assumption (see Section 2.3.4 and [87] for the precise definition).

The decision version of the RLWE problem with leakage, denoted Leaky R-DLWE is parametrized by $(n' \in \{1, 2, 4, 8, \ldots n\}, \mathcal{S} \subseteq \mathbb{Z}_{2n'}^*)$. The goal is to distinguish the distributions $D^{\vec{s}}_{real,n',\mathcal{S}}$ and $D^{\vec{s}}_{sim,n',\mathcal{S}}$, where $D^{\vec{s}}_{real,n',\mathcal{S}}$ is as above and $D^{\vec{s}}_{sim,n',\mathcal{S}}$ outputs $\left(\widehat{\vec{a}}, \widehat{\vec{u}}, [\widehat{s}_i]_{i \equiv \alpha \bmod 2n' \mid \forall \alpha \in \mathcal{S}}\right)$, where $\widehat{u}_i = \widehat{a}_i \cdot \widehat{s}_i + \widehat{e}_i$ for $i \equiv \alpha \mod 2n'$, $\alpha \in \mathcal{S}$ and $\widehat{u}_i$ is chosen uniformly at random from $\mathbb{Z}_q$, otherwise. Note that only the coordinates of $\widehat{u}$ corresponding to *unleaked* positions are required to be indistinguishable from random.

When $\mathcal{S} = \{\alpha\}$ consists of a single element, we sometimes abuse notation and write the Leaky-RLWE parameters as (n', α). Due to automorphisms on the NTT transform, Leaky-RLWE with parameters $(n', \mathcal{S})$ where $\mathcal{S} = \{\alpha_1, \alpha_2, \ldots, \alpha_t\}$, is equivalent to Leaky-RLWE with parameters $(n', \mathcal{S}')$, where $\mathcal{S}' = \alpha_1^{-1} \cdot S$ (multiply every element of $\mathcal{S}$ by α_1^{-1}). It is also not hard to see that leaky search and decision are equally hard when secret $\vec{s}$ is uniform random from R_q versus drawn from the error distribution (the same reduction for standard RLWE works in our case).

5.1.2 Our Results

Partial Key Exposure Attacks We present attacks on Leaky R-SLWE and test them on various practical parameter settings, such as the NewHope [14] parameter settings as well as the RLWE challenges of Crockett and Peikert [38]. Our attacks demonstrate that Leaky R-SLWE is *easy* for leakage parameters $(n' = 4, \alpha = 1)$, $(n' = 8, \mathcal{S} = \{1, 7\})$ and $(n' = 8, \mathcal{S} = \{1, 15\})$, under (1) NewHope parameter settings of $n = 1024$, $q = 12289$, and $\chi = \Psi_{16}$ (centered binomial distribution of parameter 16); (2) The same parameters above, but with $\chi = D_{\sqrt{8}}$ (discrete Gaussian with standard deviation of $\sqrt{8}$, which has the same standard deviation as Ψ_{16}), since this is the recommended setting in the case where the adversary gets to see *many* RLWE samples [10]; (3) For parameters of several of the Crockett and Peikert challenges, including those classified as "very hard." In all the above cases, we fully recover the RLWE secret with high probability, given the corresponding 1/4-fraction of the positions in the NTT transform of the RLWE secret. See Section 5.2.2 for details on the experimental results.

A Search-to-Decision Reduction Define $T_{n'}(n)$ to be the time required to solve Leaky R-SLWE for dimension n, given positions $[\hat{s}_i]_{i \equiv \alpha \mod 2n'}$. Assuming search R-$\mathsf{LWE}$ *without leakage* is subexponentially $2^{\Omega(n^\epsilon)}$-hard for some constant $\epsilon \leq 1$ and polynomial modulus q, then $T_n(n) \in 2^{\Omega(n^\epsilon)}$, i.e. there is a constant c such that, for sufficiently large n, $T_n(n) \geq 2^{c(n^\epsilon)}$. Note that, search R-$\mathsf{LWE}$ can be solved given a subroutine that solves Leaky R-

SLWE by first guessing the leakage on $\vec{s}$, then running the Leaky R-SLWE attack. Thus, by guessing the value of the single leaked position we obtain a $(T_n(n) \cdot q)$-time attack on search R-LWE *without leakage*. Also, $T_1(n) \in \mathsf{poly}(n)$, since the entire $\vec{s}$ is leaked. So there is some constant c' such that, for sufficiently large n, there exists $n^* = n^*(n) \in \{2, 4, 8, 16, \dots, n\}$ such that $T_{n^*}(n) \geq 2^{c'(n^\epsilon)}$ and $\frac{T_{n^*}(n)}{T_{n^*/2}(n)} \geq n$. Otherwise for every $n_1 \in \mathbb{N}$, there exists an $n_2 \geq n_1$ such that $T_{n_2}(n_2) < 2^{c'(n_2^\epsilon)} \cdot n_2^{\log n_2} < 2^{c(n_2^\epsilon)}$, which is a contradiction and so the latter relation has to hold.

Theorem 36 (Informal). *Assume $n^* := n^*(n) > 4$, $\vec{s} \leftarrow R_q$, then:*

(1) $D^{\vec{s}}_{real,n^*,\{\alpha\}} \approx D^{\vec{s}}_{sim,n^*,\{\alpha\}}$ OR

(2) $D^{\vec{s}}_{real,n^*,\{\alpha,(n^*-1)\cdot\alpha\}} \approx D^{\vec{s}}_{sim,n^*,\{\alpha,(n^*-1)\cdot\alpha\}}$ OR

(3) $D^{\vec{s}}_{real,n^*,\{\alpha,(2n^*-1)\cdot\alpha\}} \approx D^{\vec{s}}_{sim,n^*,\{\alpha,(2n^*-1)\cdot\alpha\}}$.

While at first glance it may seem that the conclusions (1), (2), (3) are redundant, in fact they are incomparable; Indeed, conclusion (1) does not imply (2) (resp. (3)), since the adversary in (2) (resp. (3)) is given additional leakage. Conversely, conclusion (2) (resp. (3)) does not imply (1), since the set of NTT coordinates that are indistinguishable from random is smaller in (2).

Note that our experimental results show that for our chosen parameter settings $D^{\vec{s}}_{real,4,\{1\}} \not\approx D^{\vec{s}}_{sim,4,\{1\}}$, $D^{\vec{s}}_{real,8,\{1,7\}} \not\approx D^{\vec{s}}_{sim,8,\{1,7\}}$ and $D^{\vec{s}}_{real,8,\{1,15\}} \not\approx D^{\vec{s}}_{sim,8,\{1,15\}}$ (since we in fact fully recover the secret in all these cases). This indicates that $n^* \neq 4$ and, if $n^* = 8$ for our chosen parameter settings (as supported by our experiments), then it must be the case that $D^{\vec{s}}_{real,8,\{1\}} \approx D^{\vec{s}}_{sim,8,\{1\}}$.

Applications The Leaky R-DLWE assumption is a useful tool for analyzing the security of RLWE-based cryptosystems subject to partial key exposure, and guaranteeing a graceful degradation in security. In particular, the Leaky R-DLWE assumption was used to analyze the NewHope protocol of [14] in a work by Dachman-Soled et al. [42]. The assumption is applicable to schemes in which the RLWE assumption is used to guarantee that a certain outcome is high-entropy (as opposed to uniform random), such as NewHope without reconciliation [13].

Practicality of Our Attack We note that an attack on Leaky R-SLWE yields an attack on standard search R-LWE by guessing each possible leakage outcome, running the Leaky R-SLWE attack and checking correctness of the recovered secret. Therefore, we believe this line of research is interesting beyond the context of leakage resilience, since if the attack can be made to work successfully for sufficiently low leakage rate (far lower than the 1/4-leakage rate of our attacks), then one could potentially obtain an improved attack on standard search R-LWE.

We chose to consider partial exposure of the *NTT transform* of the R-LWE secret, since in practical schemes the secret key is often stored in the NTT domain and certain types of side-channel attacks allow recovering large portions of the secret key stored in memory. For example, Albrecht et al. [11], in their analysis of "cold boot attacks" on NTT cryptosystems, considered bit-flip rates as low as 0.2%. However, the highly structured leakage required for our attack is unlikely to occur in a practical leakage setting such as a "cold boot attack," where one expects to recover the values of random locations in memory. We leave open the question of reducing the structure of the leakage in our attack.

Specifically, as a starting point it will be interesting to see if our attack can extend to leakage patterns of $n' = 16$, $|S| = 4$ or $n' = 32$, $|S| = 8$, etc. While the leakage rate remains the same (1/4) in each case, these patterns capture leakage that is less and less structured, since at the extreme, one can view leakage of a random 1/4-fraction of the NTT coordinates as an instance of Leaky R-SLWE with parameters $n' = n$ and $|S| = n/4$. This direction was raised by an anonymous reviewer for our original submission of this work, and we want to thank them for bringing this research direction to our attention.

5.1.3 Related Work

Leakage-Resilient Cryptography The study of provably secure, leakage-resilient cryptography was introduced by the work of Dziembowski and Pietrzak in [48]. Pietrzak [96] also constructed a leakage-resilient stream-cipher. Brakerski et al. [32] showed how to construct a scheme secure against an attacker who leaks at each time period. There are other works as well considering continual leakage [46, 78]. There are also work on leakage-resilient signature scheme [31, 71, 88].

Leakage-Resilience and Lattice-Based Cryptography Goldwasser et al. [55], and subsequently [9, 45, 47] studied the leakage resilience of standard LWE based cryptosystems in the symmetric and public key settings.

Leakage Resilience of Ring-LWE Dachman-Soled et al. [41] considered the leakage resilience of a RLWE-based public key encryption scheme for specific leakage profiles. This was followed by Albrecht et al. [11], they investigated cold boot attacks and compared the number of operations for implementing the attack when the secret key is stored as polynomial coefficients versus when encoding of the secret key using a number theoretic transform (NTT) is stored in memory. Recently, Stange [102] showed that given multiple samples of RLWE instances such that the public key for every instance lies in some specific subring, one can reduce the original RLWE problem to multiple independent RLWE problems over the subring. In this work we do not place any such restriction on the RLWE samples required to mount partial key exposure attack.

Comparison with Concurrent Work of Bolboceanu et al. [27] One of the settings considered by authors in [27] is sampling the RLWE secret from an ideal $I \subseteq qR$. It is straightforward to see that sampling the RLWE secret uniformly at random from R_q and then leaking the NTT coordinates i such that $i = \alpha \mod 2n'$ is *equivalent* to sampling the RLWE secret from the ideal I that contains those elements whose NTT transform is 0 in positions i such that $i = \alpha \mod 2n'$. Nevertheless, our decisional assumption is *weaker* than the assumption of [27], since [27] require that the entire vector $\vec{u}$ be indistinguishable from uniform random, whereas we only require that the NTT transform of $\vec{u}$ is indistinguishable from uniform random at the positions i that are not leaked. Our assumption lends itself to a search-to-decision reduction while the assumption of [27] does not. While [27] do provide a direct security reduction for their decisional assumption, the required standard deviation of the error (in polynomial basis, tweaked and scaled by q) is $\omega(q^{1/n'} \cdot n^{3/2})$, which would be far higher than the noise considered in the NewHope and RLWE Challenges settings. In contrast, our assumption can be applied in practical parameter regimes and is sufficient to argue the security of several practical cryptosystems under partial key exposure. Finally, we compare our attack to that of [27]. For fixed n, q, our attack works for noise regimes that are not covered by the attack of [27]. For example, for NewHope settings of $n = 1024$, $q = 12289$, the attack of [27] has success rate at most 1/1000 when the standard deviation of noise distribution is less than 0.00562. Note that [27] provides an upper bound of norm of error with respect to canonical basis for its attack to succeed. Using a variant of Chernoff's bound, we derive an upper bound of standard deviation of error for success rate at most 1/1000. To make the bound comparable to NewHope setting, we further convert to tweaked polynomial representation and to RLWE instance in the form of $(as+e)$ instead of $(as/q + e)$. In contrast, our attack works (with success ranging from 82/200 to 2/1000) when the standard deviation of the noise is $\sqrt{8} \approx 2.83$. Also note that the standard deviation of $\sqrt{8}$ is the more conservative setting in the original NewHope specification [14]. The NIST submission uses lower standard deviation of 2, which is still not covered by the attack of [27]. Our attack applies only for certain leakage patterns corresponding to certain ideals I, whereas the attack of [27] works for any ideal. The techniques of the two attacks are entirely different. Bolboceanu et al. [27] obtain a "good" basis for the ideal via *non-uniform* advice, perform a change of basis and then use Babai's roundoff algorithm to solve the resulting BDD instance. We use the algebraic structure of the problem to convert RLWE instances over high dimension into CVP instances over constant dimension n'. We then exactly solve the CVP instances over constant dimension and determine the "high confidence" solutions that are likely to be the correct values of the RLWE error. Assuming all high confidence solutions are correct, we obtain a noiseless system of linear equations w.r.t. the RLWE secret, allowing efficient recovery of the secret.

5.2 Partial Key Exposure Attack on Ring-LWE

5.2.1 Reconstructing the Secret Given ($\alpha \mod n'$) Leakage.

Recall that for $\vec{p} \in \mathbb{Z}_q[x]/(x^n+1)$, the NTT transform, $\widehat{\vec{p}}$, is obtained by evaluating $\vec{p}(x) \bmod q$ at the powers ω^i for $i \in \mathbb{Z}_{2n}^*$, where ω is a $2n$-th primitive root in $\mathbb{Z}_q$. For $n' \in \{1,2,4,8,\dots,n\}$, let $u = n/n'$. For $\alpha \in \mathbb{Z}_{2n'}^*$, consider $\vec{p}_u^{\alpha}(x)$ be the degree $u-1$ polynomial that is obtained by taking $\vec{p}(x)$ modulo $(x^u - (\omega^\alpha)^u)$. We may assume without loss of generality that $\alpha = 1$. We abbreviate notation and write $\vec{p}_u$, instead of $\vec{p}_u^1$.

We consider attacks in which the adversary learns all coordinates i of $\widehat{\vec{s}}$ such that $i \equiv 1 \bmod 2n'$ where $n' \in \{1,2,4,8,\dots,n\}$, and aims to recover the RLWE secret $\vec{s}$. First, we note that in NTT transform notation the equation $\widehat{\vec{a}} \cdot \widehat{\vec{s}} + \widehat{\vec{e}} = \widehat{\vec{u}}$ holds component-wise. Therefore, given leakage on certain coordinates of $\widehat{\vec{s}}$, we can solve for the corresponding coordinates of $\widehat{\vec{e}}$. We also get to see multiple RLWE samples (which we write in matrix notation–where the $\vec{A}^j$ matrices are the circulant matrices corresponding to the ring element $\vec{a}^j$'s) as $(\vec{A}^1, \vec{A}^1\vec{s} + \vec{e}^1 = \vec{u}^1), \dots, (\vec{A}^\ell, \vec{A}^\ell\vec{s} + \vec{e}^\ell = \vec{u}^\ell)$. Thus, for the j-th RLWE sample we learn all the coordinates $\widehat{\vec{e}}_i^j$, for $i \equiv 1 \mod 2n'$. Note that the leaked coordinates are the evaluation of the polynomial $\vec{e}_u(x)$ at the ω^i for $i \equiv 1 \mod 2n'$. We can then reconstruct the polynomial $\vec{e}_u(x)$ using Lagrange Interpolation.

For $i \in \{0,\dots,u-1\}$, the $(i+1)$-st coefficient of $\vec{e}_u(x)$, i.e. $e_{u,i}$ is equal to

$$e_i + \omega^u \cdot e_{i+u} + \omega^{2\cdot u} \cdot e_{i+2\cdot u} + \dots + \omega^{(n'-1)\cdot u} \cdot e_{i+(n'-1)\cdot u}$$

The coefficients of $\vec{e}$ can be partitioned into u groups of size n', forming independent linear systems, each with n' variables and one equation. Given only the leakage, the set of feasible secret keys is a cartesian product $\mathcal{S}_1 \times \cdots \times \mathcal{S}_u$, where for $i \in [u]$, the set $\mathcal{S}_i$ is the set of vectors $\vec{\bar{e}}_i := \{e_i, e_{i+u}, e_{i+2u}, \ldots, e_{i+(n'-1)u}\}$ that satisfy the i-th linear system:

$$\begin{bmatrix} 1 & \omega^{u} & \omega^{2\cdot u} & \cdots & \omega^{(n'-1)\cdot u} \end{bmatrix} \cdot \begin{bmatrix} e_i & e_{i+u} & e_{i+2\cdot u} & \cdots & e_{i+(n'-1)\cdot u} \end{bmatrix}^T = \begin{bmatrix} e_{u,i} \end{bmatrix}$$

Since each coordinate of $\vec{e}$ is drawn independently from χ and since each linear system above has small dimension n', we can use a brute-force-search to find the most likely solution and calculate its probability.

Given this information, we will carefully choose the solutions $\vec{\bar{e}}_i^j$ (from all possible sets of solutions $\left[\vec{\bar{e}}_i^j\right]_{j\in[\ell], i\in[u]}$) that have a high chance of being the correct values of the RLWE error. To obtain a full key recovery attack, we require the followings.

(1) In total, we must guess at least u number of n'-dimensional solutions, $\vec{\bar{e}}_i^j$, from all the obtained solutions $\left[\vec{\bar{e}}_i^j\right]_{j\in[\ell], i\in[u]}$.

(2) With high probability *all* our guesses are correct.

Observe that if our guess of some $\vec{\bar{e}}_i^j$ is correct, we learn the following linear system of n' equations and n variables ($A^{j,i} \cdot \vec{s} = \vec{u}^{j,i} - \vec{e}^{j,i}$), where $A^{j,i}$ is the submatrix of A^j consisting of the n' rows $i, i+u, i+2\cdot u, \ldots, i+(n'-1)\cdot u$, and $\vec{u}^{j,i}, \vec{e}^{j,i}$ are vectors consisting of the $i, i+u, i+2\cdot u, \ldots, i+(n'-1)\cdot u$ coordinates of $\vec{u}^j$ and $\vec{e}^j$. So assuming items (1) and (2) hold, we learn u *noiseless* systems of n' linear equations, each with $n = u \cdot n'$ number of variables. We then construct a linear system of n variables and n equations, which can be solved to obtain the candidate $\vec{s}$.

In order to ensure that item (2) holds, we only keep the guess for $\vec{e}_i^j$ when we have "high confidence" that it is the correct solution. The probability of a particular solution $\vec{e}_i^j := \left(e_i^j, e_{i+u}^j, \ldots, e_{i+(n'-1)u}^j\right)$, is the ratio of the probability of $\vec{e}_i^j$ being drawn from the error distribution (which is coordinate-wise independent) over the sum of the probabilities of all solutions. For small dimension n', this can be computed via a brute-force method. In our case, we keep the highest probability solution when it has probability at least, say 0.98. The probability that *all* guesses are correct is therefore $0.98^u = 0.98^{n/n'}$.

Since computing the exact probability as above is computationally intensive, we develop a heuristic that performs nearly as well and is much faster. Note that finding the "most likely" solution is equivalent to solving a CVP problem over an appropriate n'-dimensional lattice. We then calculate the probability of the solution under the discrete Gaussian and set some threshold . If the probability of the solution is above the threshold we keep it, if not we discard it. Experimentally, we show that by setting the threshold correctly, we can still achieve high confidence. See Figure 5.1 for the exact settings of the threshold for each setting of parameters. Our experiments show that item (1) also holds given a reasonable number of RLWE samples. See Section 5.2.2 for a presentation of our experimental results. We describe our attack in cases where the leakage is on all coordinates i such that $i \equiv \alpha_1 \mod 2n'$ or $i \equiv \alpha_2 \mod 2n'$ in Appendix C.1.

Complexity of the Attack We provide the pseudocode for the attack in Appendix C.3, Figure C.1. While our attack works well in practice, we do not provide a formal proof that our attack is polynomial time for a given setting of parameters. Within the loop beginning on line 5, all the steps (or subroutines) shown in Figure C.1 can be computed in polynomial time. Note that even step 12 (`CVP.closest_vector`), which requires solving a CVP instance, can be computed in polynomial time because for the leakage patterns we consider, the dimension of the CVP instance will always be either 4 or 8–a constant, independent of n. However, our analysis does not bound the number of iterations of the loop beginning on line 5. Specifically, we do not analyze how large the variable `RLWESamples` must be set in order to guarantee that the attack is successful with high probability. Bounding this variable corresponds to bounding the number of RLWE samples needed in order to obtain a sufficient number of "high confidence" solutions. In practice, the number of RLWE samples was always fewer than 200 for all parameter settings. In future work, we plan to compute the expected number of RLWE samples needed to obtain a sufficient number of high confidence solutions for a given parameter setting. Assuming this expected number of samples is polynomial in n, we obtain an expected polynomial time attack.

5.2.2 Experimental Results

We first assess the performance of our attack on the RLWE challenges published by Crockett and Peikert [38], with various parameters, ranging from "toy" to "very hard" security levels. For each parameter setting, a cut-and-choose protocol was used by [38] to prove correctness of the challenges: They committed to some number (e.g. $N = 32$) of independent RLWE instances, a random index i was chosen, and the secret key for all except the i-th instance was revealed. For each of the 31 *opened* challenges, we simulate the Leaky RLWE experiment and attempt to recover the full secret $\vec{s}$ using our attack.

We next measure the performance of our attack on RLWE instances generated using the dimension, modulus and noise distribution proposed in the original NewHope scheme [14]. These parameters are more conservative than the ones chosen for the later submission to the NIST competition [12]. When multiple RLWE samples are released, bounded error distributions are less secure [10]. We therefore tested our attack in the *more difficult* setting of Gaussian error, in addition to the original binomial error distribution of [14].

The experiments were run using server with AMD Opteron 6274 processor, with a python script using all the cores with Sage version 8.1. We used fplll [44] library for CVP solver and the source code of all the attacks are available online at [4]. The results of our attacks are summarized in Figure 5.1. We report the total number of instances we broke and the average number of RLWE samples needed for those instances. To decide whether a solution is kept or discarded, its probability mass under the error distribution χ is calculated and compared to the threshold. The threshold for each parameter setting is set heuristically so that minimal weight solutions passing the threshold are correct with high confidence (see Figure 5.1 for the exact threshold settings). We tested leakage patterns of $(n' = 4, \mathcal{S} = \{1\})$, $(n' = 8, \mathcal{S} = \{1, 7\})$ and $(n' = 8, \mathcal{S} = \{1, 15\})$–all corresponding to 1/4-fraction leakage—for each parameter setting and were able to break multiple Leaky RLWE instances for every parameter setting/leakage pattern shown in Figure 5.1. We also report the maximum time it took to break a single instance for each parameter setting in Figure 5.1. Overall, the maximum amount of time to break a single instance was 6 hours for the hardest instance, i.e. Challenge ID 89. We attempted to launch our attack given only 1/8-fraction of leakage (leakage pattern $(n' = 8, \alpha = 1)$), but were only successful for the easiest case, i.e. Challenge ID 1. For, e.g. Challenge ID 89, the attack failed since for

5000 number of linear systems, the maximum confidence of any solution was 0.28, meaning that we expect to recover the secret key with probability at most $0.28^{2048/8} \approx 2^{-470}$, which is well beyond feasible.

Chall ID (hardness)	n	q	χ	n'	Pattern ($\mathcal{S}$)	min-max RLWE #	Avg. RLWE #	Broken Instances	Threshold	Maximum Time (s)
1 (toy)	128	769	$D_{0.40}$	4	$\{1\}$	2-2	2	31 of 31	7e-5	2.24
				8	$\{1, 15\}$	1-2	1.93	29 of 31	7e-6	2.18
				8	$\{1, 7\}$	1-2	1.93	29 of 31	7e-6	1.23
				8	$\{1\}$	1-1	1	4 of 31	1e-8	1.3
5 (toy)	128	3329	$D_{0.80}$	4	$\{1\}$	2-3	2.38	31 of 31	7e-5	2.53
				8	$\{1, 15\}$	2-3	2.09	31 of 31	7e-6	1.99
				8	$\{1, 7\}$	2-3	2.09	31 of 31	7e-6	1.88
45 (moderate)	256	7681	$D_{0.80}$	4	$\{1\}$	2-3	2.61	31 of 31	7e-5	8.83
				8	$\{1, 15\}$	2-2	2	31 of 31	7e-8	8.78
				8	$\{1, 7\}$	2-2	2	31 of 31	7e-8	6.97
85 (very hard)	1024	59393	$D_{3.59}$	4	$\{1\}$	6-7	6.05	17 of 31	7e-5	1914
				8	$\{1, 15\}$	39-60	51.88	26 of 31	7e-9	2000
				8	$\{1, 7\}$	39-59	50.76	17 of 31	7e-9	2682
89 (very hard)	2048	86017	$D_{3.59}$	4	$\{1\}$	6-7	6.16	30 of 31	7e-5	5523
				8	$\{1, 15\}$	44-58	52.29	31 of 31	7e-9	11766
				8	$\{1, 7\}$	44-58	52.29	31 of 31	7e-9	20837
NewHope	1024	12289	$D_{\sqrt{8}}$	4	$\{1\}$	35-37	36	3 of 200	3e-4	745
				8	$\{1, 15\}$	147-220	180.85	82 of 200	7e-8	2226
				8	$\{1, 7\}$	189-204	196.5	2 of 1000	7e-8	1238
			Ψ_{16}	4	$\{1\}$	34-36	34.16	6 of 200	3e-4	796
				8	$\{1, 15\}$	149-217	183.20	94 of 200	7e-8	2039
				8	$\{1, 7\}$	177-193	184.8	5 of 1000	7e-8	975

Figure 5.1: Performance of attack against RLWE Challenges [38] and NewHope [14] parameter settings. For each parameter setting, we report the following: min/max and average number of RLWE samples required for successful break, total number of broken instances, and max run-time (in seconds) for successful break. Threshold is set such that the minimal weight solutions to the linear systems given in Section 5.2 have high confidence with sufficiently high probability.

5.3 Search and Decisional Ring-LWE with Leakage

Definition 37 (Search RLWE (R-SLWE) with Leakage). *The* search *version of the R-LWE problem with leakage, denoted Leaky R-*$\mathsf{SLWE}_{q,\psi,n',\mathcal{S}}$*, is parameterized by* $(n' \in \{1, 2, 4, 8, \ldots n\}, \mathcal{S} \subseteq \mathbb{Z}^*_{2n'})$*. The experiment chooses* $\vec{s} \leftarrow R_q$ *uniformly at random, where*

$\vec{s} = \mathsf{NTT}^{-1}\left(\widehat{\vec{s}}\right)$. *The goal of the adversary is to recover* $\vec{s}$*, given independent samples from the distribution* $D^{\vec{s}}_{real,n',\mathcal{S}}$*, which outputs* $\left(\widehat{\vec{a}}, \widehat{\vec{a}} \cdot \widehat{\vec{s}} + \widehat{\vec{e}}, [\widehat{s}_i]_{i \equiv \alpha \bmod 2n' \mid \forall \alpha \in \mathcal{S}}\right)$ *where* $\vec{a}, \vec{e}$ *are obtained from* $A_{\vec{s},\psi}$ *as in standard RLWE (see Definition 15).*

Definition 38 (**Decision RLWE (R-DLWE) with Leakage**). *The* decision *version of the* R-LWE *problem with leakage, denoted Leaky* R-$\mathsf{DLWE}_{q,\psi,n',\mathcal{S}}$*, is parameterized by* $(n' \in \{1, 2, 4, 8, \ldots n\}, \mathcal{S} \subseteq \mathbb{Z}^*_{2n'})$*. The experiment chooses* $\vec{s} \leftarrow R_q$ *uniformly at random, where* $\vec{s} = \mathsf{NTT}^{-1}\left(\widehat{\vec{s}}\right)$*. The goal of the adversary is to distinguish between independent samples from the distributions* $D^{\vec{s}}_{real,n',\mathcal{S}}$ *and* $D^{\vec{s}}_{sim,n',\mathcal{S}}$*, where* $D^{\vec{s}}_{real,n',\mathcal{S}}$ *is the same as above, and* $D^{\vec{s}}_{sim,n',\mathcal{S}}$ *outputs* $\left(\widehat{\vec{a}}, \widehat{\vec{u}}, [\widehat{s}_i]_{i \equiv \alpha \bmod 2n' \mid \forall \alpha \in S}\right)$*, where* $\vec{a}, \vec{e}$ *are obtained from* $A_{\vec{s},\psi}$ *as in standard RLWE (see Definition 15) and*

$$\widehat{u_i} = \widehat{a_i} \cdot \widehat{s_i} + \widehat{e_i} \quad \mid \quad i \equiv \alpha \bmod 2n' \;\; \forall \alpha \in S \qquad \textit{and} \qquad \widehat{u_i} \leftarrow \mathbb{Z}_q$$

chosen uniformly random, otherwise.

5.4 Search to Decision Reduction with Leakage

Let the RLWE secret be denoted by $\hat{\vec{s}}$ and assume without loss of generality that there exists an adversary that obtains leakage $[\hat{s}_i]_{i \equiv 1 \mod 2n'}$ and distinguishes $\hat{\vec{u}} = \hat{\vec{a}} \cdot \hat{\vec{s}} + \hat{\vec{e}}$ from $\hat{\vec{u}}'$, where $\hat{u}_i = \hat{a}_i \cdot \hat{s}_i + \hat{e}_i$ for $i \equiv 1 \mod 2n'$ and otherwise is uniform random. Note that the problem is identical when the adversary obtains leakage $[\hat{s}_i]_{i \equiv \alpha \mod 2n'}$, for $\alpha \in \mathbb{Z}^*_{2n'}$ since, as we shall see next, an automorphism can be applied to shift all indices i such that $i \equiv \alpha \mod 2n'$ to positions $i \equiv 1 \mod 2n'$. It is not hard to see, using techniques of [83, 84, 85], that this implies an attacker that learns a single index $j \in \mathbb{Z}^*_{2n}$, $j \equiv b \mod 2n'$ of the RLWE secret, where $b \not\equiv 1 \mod 2n'$. We call this the **Basic Attack**. We refer readers to Appendix C.2 for description of **Basic Attack**.

Theorem 39 (Existence of Basic Attack)**.** *If, for any $(n', \mathcal{S} \subseteq \mathbb{Z}^*_{2n'})$ adversary $\mathcal{A}$ running in time $t := t(n)$ distinguishes $D^{\vec{s}}_{real,n',\mathcal{S}}$ from $D^{\vec{s}}_{sim,n',\mathcal{S}}$ with probability $p := p(n)$, then there is some index j such that $j \neq \alpha' \mod n$ for all $\alpha' \in \mathcal{S}$ and an attack **Basic Attack** with parameters $(n', \mathcal{S}, j, t, p)$, that learns NTT coordinate $\hat{s}_j$ with probability $1 - 1/\mathsf{poly}(n)$ and takes time $\mathsf{poly}(n) \cdot t \cdot 1/p$.*

Our attack **Attack 1** uses the **Basic Attack** to learn *all* the values $[\hat{s}_i]_{i \equiv b^r \mod 2n'}$ for $r \in [n'/2]$. Let $\hat{\vec{s}}^1 := \hat{\vec{s}}$. The main idea of **Attack 1** is to learn all $[\hat{s}^1_i]_{i \equiv b \mod 2n'}$ in the first round, then apply an automorphism to shift the positions $i \equiv b^2 \mod 2n'$ into the positions $i \equiv b \mod 2n'$, resulting in a permuted RLWE secret, denoted $\hat{\vec{s}}^2$. Note that applying the automorphism causes the positions $\hat{s}^1_i$ such that $i \equiv b \mod 2n'$ to shift into the positions $i \equiv 1 \mod 2n'$. This means that we are now back where we started, and the reduction is now able to provide the required leakage (on $[\hat{s}^2_i]_{i \equiv 1 \mod 2n'}$) to the adversary and thus can learn the values of $[\hat{s}^2_i]_{i \equiv b \mod 2n'} = [\hat{s}^1_i]_{i \equiv b^2 \mod 2n'}$ in the second iteration, $[\hat{s}^3_i]_{i \equiv b \mod 2n'} = [\hat{s}^1_i]_{i \equiv b^3 \mod 2n'}$ in the third iteration, etc. We next formalize the necessary properties of the automorphisms.

For $i, j \in \mathbb{Z}^*_{2n}$, let $\phi_{i \to j}$ be the automorphism that maps $\hat{\vec{v}}$ to $\hat{\vec{v}}'$ such that $\vec{v}(\omega^i) = \vec{v}'(\omega^j)$. Hence, $\phi_{i \to j}$ induces a permutation on the elements of $\hat{\vec{v}}$, denoted $\rho_{i \to j}$. Specifically, $\phi_{i \to j}(\hat{\vec{v}})$ maps $\hat{v}_\ell$ to $\hat{v}_{\rho_{i \to j}(\ell)}$ for $i, j, \ell \in \mathbb{Z}^*_{2n}$, where $\rho_{i \to j}(\ell) = i^{-1}\ell j$.

Definition 40. *A probability distribution $\psi : \mathbb{Z}(\zeta_m) \to \mathbb{R}$ is automorphically closed in K*

if for all $i, j \in \mathbb{Z}_m^*$, $\phi_{i \to j}(\psi) = \psi$.

We remark that RLWE error distribution χ is automorphically closed [83].

We formally define **Attack 1** in Figure 5.2. We next sketch how **Attack 1** can be used to complete the proof. For dimension n and parameter $n' \in \{1, 2, 4, 8, \dots n\}$, let $T_{n'} := T_{n'}(n)$ be the (non-uniform) time to solve Leaky R-SLWE for dimension n and parameters $(n', \mathcal{S} = \{\alpha\} = \{1\})$, i.e. given positions $[\hat{s}_i^1]_{i \equiv 1 \mod 2n'}$, with probability $1/2$.

Assume subexponential $2^{\Omega(n^\epsilon)}$-hardness of search RLWE *without leakage* for some constant $\epsilon \le 1$ and polynomial modulus q. Then we also have that $T_n(n) \in 2^{\Omega(n^\epsilon)}$, and as discussed in the intro, there must exist a constant c' such that for sufficiently large n, there exists $n^* = n^*(n) \in \{2, 4, 8, 16, \dots, n\}$ such that $T_{n^*}(n) \ge 2^{c'(n^\epsilon)}$ and $\frac{T_{n^*}(n)}{T_{n^*/2}(n)} \ge n$. The above implies that $T_{(n^*/2)} \in o(T_{n^*})$.

Now, if given $[\hat{s}_i^1]_{i \equiv 1 \mod 2n^*}$ leakage, there exists a $(t(n), p(n))$-distinguishing adversary (where $t(n) = \sqrt{T_{n^*}}/\mathsf{poly}(n)$ and $p(n) = 1/\sqrt{T_{n^*}}$), then we will show that there is an adversary solving the R-SLWE with high probability given positions $[\hat{s}_i^1]_{i \equiv 1 \mod 2n^*}$ in time *less than* T_{n^*}, leading to contradiction. We begin by running **Attack 1**, which takes time at most $o(T_{n^*})$ for our settings of $t(n)$ and $p(n)$. If $b \in \mathbb{Z}_{2n^*}^*$ is such that for some $r \in [n^*/2]$, $b^r \equiv n^* + 1 \mod 2n^*$, then we can combine the reconstructed values of $\hat{s}_i^1$ from **Attack 1** with our knowledge of $[\hat{s}_i^1]_{i \equiv 1 \mod 2n^*}$ to obtain all values $[\hat{s}_i^1]_{i \equiv 1 \mod n^*}$. This means that we can then run the search attack for $2/n^*$-fraction of leakage to recover all of $\hat{\vec{s}}$ in time $T_{(n^*/2)} \in o(T_{n^*})$. But then the entire attack for $(1 \mod 2n^*)$-leakage can be run in time $o(T_{n^*})$, contradicting the definition of T_{n^*}.

For $n^* > 4$, the only cases in which **Attack 1** does not recover $[\hat{s}_i]_{i \equiv n^*+1 \mod 2n^*}$, is

Attack 1

Given access to $D^{\vec{s}}_{real,n',\mathcal{S}=\{1\}}$ (i.e. RLWE samples with leakage $[\hat{s}_i]_{i\equiv 1 \mod 2n'}$) and the distinguishing index $j \in \mathbb{Z}^*_{2n}$, where $j \equiv b \mod 2n'$, for the **Basic Attack**:

```
1: for all Leaky-RLWE samples do
2:     Set $\hat{\vec{a}}^1 := \hat{\vec{a}}, \hat{\vec{u}}^1 := \hat{\vec{u}}$, $[\hat{s}^1_i := \hat{s}_i]_{i\equiv 1 \mod 2n'}$.
3: end for
4: for $r \in [1, 2, \ldots, n'/2]$ do        ▷ $[\hat{s}^r_i]_{i\equiv 1 \mod 2n'}$ are now known.
5:     for all $j'$ such that $j' \equiv j \mod 2n'$ do
6:         Run the Basic Attack with parameters $(n', \{1\}, j, t, p)$ on RLWE samples of the form $(\hat{\vec{a}} := \phi_{j'\to j}(\hat{\vec{a}}^r), \hat{\vec{u}} := \phi_{j'\to j}(\hat{\vec{u}}^r))$, leakage set $\left[\hat{s}_i := \hat{s}^r_{\rho_{j'\to j}(i)}\right]_{i\equiv 1 \mod 2n'}$ to recover $\hat{s}^r_{j'}$.
           ▷ All these values of $\hat{s}^r_{\rho_{j'\to j}(i)}$ are now known: If $i \equiv 1 \mod 2n'$ then $\rho_{j'\to j}(i) \equiv 1 \mod 2n'$, since $j \equiv j' \mod 2n'$.
7:     end for        ▷ w.h.p. all $\hat{s}^r_{j'}$ s.t. $j' \equiv b \mod 2n'$ are now known.
8:     Choose an $\ell \in \mathbb{Z}^*_{2n}$ such that $\ell \equiv b^2 \mod 2n'$.
9:     for all Leaky RLWE samples do
10:        Set $\hat{\vec{a}}^{r+1} := \phi_{\ell\to j}(\hat{\vec{a}}^r)$ and $\hat{\vec{u}}^{r+1} := \phi_{\ell\to j}(\hat{\vec{u}}^r)$.
11:    end for
       ▷ $\left[\hat{s}^{r+1}_i\right]_{i\equiv 1 \mod 2n'}$, are now known since $\hat{s}^r_{i'}$ s.t. $i' \equiv b \mod 2n'$ are now in position $\hat{s}^{r+1}_i$ s.t. $i \equiv 1 \mod 2n'$.
12: end for        ▷ All values $s_i$ such that $i \equiv b^r \mod 2n'$ and $r \in [n'/2]$ are now known.
```

Figure 5.2: Description of **Attack 1**.

when $b \in \{n^* - 1, 2n^* - 1\}$. For such b, we do not know how to rule out the possibility that given $[\hat{s}_i]_{i\equiv 1 \mod 2n^*}$, the positions $i \equiv b \mod 2n^*$ of $\hat{u}$ do not look random. In this case, however, we argue that given leakage on *both* $[\hat{s}_i]_{i\equiv 1 \mod n^*}$, and $[\hat{s}_i]_{i\equiv b \mod n^*}$, all *other* positions are indistinguishable from random, since otherwise a modified version of **Attack 1** can be run. We next state the formal theorem of this section.

Theorem 41. *Assume $n^* := n^*(n) > 4$, $\vec{s} \leftarrow R_q$, then:*

- $D^{\vec{s}}_{real,n^*,\{\alpha\}} \approx_{t(n),p(n)} D^{\vec{s}}_{sim,n^*,\{\alpha\}}$ *OR*
- $D^{\vec{s}}_{real,n^*,\{\alpha,(n^*-1)\cdot\alpha\}} \approx_{t(n),p(n)} D^{\vec{s}}_{sim,n^*,\{\alpha,(n^*-1)\cdot\alpha\}}$ *OR*

- $D^{\vec{s}}_{real,n^*,\{\alpha,(2n^*-1)\cdot\alpha\}} \approx_{t(n),p(n)} D^{\vec{s}}_{sim,n^*,\{\alpha,(2n^*-1)\cdot\alpha\}}$.

where, $t(n) = \sqrt{T_{n^*}}/\mathsf{poly}(n)$*,* $p(n) = 1/\sqrt{T_{n^*}}$*.*

Proof. We can assume without loss of generality that $\alpha = 1$. Furthermore, assume $D^{\vec{s}}_{real,n^*,\{1\}} \not\approx_{(\sqrt{T_{n^*}}/\mathsf{poly}(n),1/\sqrt{T_{n^*}})} D^{\vec{s}}_{sim,n^*,\{1\}}$. Then this means there must be an adversary A running in time $\sqrt{T_{n^*}}/\mathsf{poly}(n)$, that distinguishes on index $j \in \mathbb{Z}^*_{2n}$, where $j \equiv b \mod 2n'$ with probability at least $1/\sqrt{T_{n^*}}$.

Case 1: b is such that $b^r \equiv n^* + 1 \mod 2n^*$ for some $r \in [n^*/2]$. In this case, with appropriate setting of $\mathsf{poly}(n)$, we can use **Attack 1** to recover the positions i such that $i \equiv n^* + 1 \mod 2n^*$ (with high probability) in time $o(T_{n^*})$. Now we can run the attack that takes as input $[\hat{s}_i]_{i\equiv 1 \mod n^*}$ and recovers all of $\hat{\vec{s}}$. By assumption, this attack runs in time $T_{(n^*/2)} \in o(T_{n^*})$. Thus, we can recover the whole $\hat{\vec{s}}$ with high probability greater than $1/2$ in time $o(T_{n^*})$, which is a contradiction.

By properties of the group $\mathbb{Z}^*_{2n^*}$, where n^* is a power of two, for all $b \in \mathbb{Z}^*_{2n^*} \setminus \{1, n^* - 1, 2n^* - 1\}$, it is the case that $b^r \equiv n^* + 1 \mod 2n^*$ for some $r \in [n^*/2]$. Thus, Case 1 holds for all $b \in \mathbb{Z}^*_{2n^*} \setminus \{n^* - 1, 2n^* - 1\}$.

Case 2: $b = n^* - 1$. In this case, with appropriate setting of $\mathsf{poly}(n)$, we can use **Attack 1** to recover the positions i such that $i \equiv n^* - 1 \mod 2n^*$ (with high probability) in time $o(T_{n^*})$. Assume $D^{\vec{s}}_{real,n^*,\{1,(n^*-1)\}} \not\approx_{\sqrt{T_{n^*}}/\mathsf{poly}(n),\sqrt{T_{n^*}}/\mathsf{poly}(n)} D^{\vec{s}}_{sim,n^*,\{1,(n^*-1)\}}$, then there must be some adversary $\mathcal{A}'$ that distinguishes on index $j' \in \mathbb{Z}^*_{2n}$, where $j' \equiv b' \in \mathbb{Z}^*_{2n^*} \setminus \{1, n^* - 1\}$. We can combine this with the previous attack as follows:

Case 2(a): $b' \in \mathbb{Z}^*_{2n^*} \setminus \{1, n^* - 1, 2n^* - 1\}$. Due to essentially the same argument as before, by appropriately setting $\mathsf{poly}(n)$, we can with high probability learn all $[\hat{s}_i]_{i \equiv (b')^r \mod 2n^*}$ for $r \in [n^*/2]$ in time $o(T_{n^*})$ and then apply the same argument as above.

Specifically, given the initial leakage $[\hat{s}^1_i]_{i \equiv 1 \mod 2n^*}$, the attack will first learn $[\hat{s}^1_i]_{i \equiv n^*-1 \mod 2n^*}$, then learn $[\hat{s}^1_i]_{i \equiv b' \mod 2n^*}$, then, for some (j, j') such that $j \equiv b' \mod 2n^*$ and $j' \equiv 1 \mod 2n^*$, apply automorphism $\phi_{j \to j'}$ to get $\hat{\vec{s}}^2$, learn $[\hat{s}^2_i]_{i \equiv n^*-1 \mod 2n^*}$, then learn $[\hat{s}^2_i]_{i \equiv b' \mod 2n^*}$, etc. thus ultimately learning $[\hat{s}_i]_{i \equiv (b')^r \mod 2n^*}$ for $r \in [n^*/2]$. At this point, we will have $[\hat{s}_i]_{i \equiv 1 \mod n^*}$ and thus can learn all of $\hat{\vec{s}}$ in additional time $T_{(n^*/2)} \in o(T_{n^*})$. Thus, in total the attack takes time $o(T_{n^*})$, leading to contradiction.

Case 2(b): $b' = 2n^* - 1$**.** Due to essentially the same argument as before, with appropriate setting of $\mathsf{poly}(n)$, we can with high probability recover the positions i such that $i \equiv 2n^* - 1 \mod 2n^*$ in time $o(T_{n^*})$. The adversary now knows $[\hat{s}_i]_{i \equiv n^*-1 \mod n^*}$. We can thus learn all of $\hat{\vec{s}}$ in additional time $T_{(n^*/2)} \in o(T_{n^*})$. Thus, in total the attack takes time $o(T_{n^*})$, leading to contradiction.

Case 3: $b = 2n^* - 1$**.** This essentially follows identically to Case 2. □

Chapter 6: Conclusion and Future Directions

First in Chapter 3, we looked at a cache side-channel attack against the SQLite database management system. We developed two algorithms that approximately recover the database using the information leaked from the side-channel attack. Finally, we showed the effectiveness of closest vector problem (CVP) solvers in reducing the overall noise in the recovered databases to obtain databases with improved accuracy. We showed that for attributes with range of size 12 our algorithm can recover the approximate database in at most 190 seconds with maximum error percentage of 0.11%. We have also extended our analysis to study the effect of heavy load on the system as well as cases where some of the ranges are missing. We have shown that the error percentage for those cases remain below 2%. As a possible approach to mitigate the attacks presented in this work, we suggest that when processing a range query, a random number of dummy elements get appended to the results and returned in addition to the true matches. The effect of such a countermeasure is twofold. (1) It makes it difficult for the side-channel attacker to able to aggregate information over different runs to obtain good approximations of the volumes. (2) It makes the graph generation and clique-finding algorithms more expensive, as there will be a large number of additional nodes and edges in the graph (recall that each observed volume corresponds to a node in the graph). Since clique-finding is NP-hard, adding even a small fraction of nodes to the graph can make the attack infeasible.

As a future work, it would be interesting to explore the effectiveness of the attack using fewer traces; in the extreme case it is interesting to study the scenario where only 1 trace per query is given. Moreover, it would be of interest to study the performance of the Prime & Probe [94] which is a more generic type of cache side-channel attack that can be used even in scenarios where the victim and attacker do not have a shared library. Further, as mentioned previously, improved attacks on encrypted databases are possible when the full access pattern is revealed (cf. Grubbs et al. [59]). It will be interesting to explore whether partial information about the access pattern can be obtained via the cache side-channel and whether this information can be used to obtain improved attacks. We have simulated some non-uniform query distribution, and one area that can be explored more is to study what are the most realistic query distribution. One limitation we faced in the work is the scalability of solver for NP-hard problem, i.e. clique finding algorithm. The work of Grubbs et al. [58] tolerate this by having a prepossessing step. This step enables the algorithm to work even for the cases where the size of the graph is large and it is interesting to study whether a prepossessing step can be applied to cases where volumes are noisy.

Later in Chapter 4, we considered new algorithms for LPN when the parity function is sparse, i.e. has dimension n, error rate η, and Hamming weight at most k, where $k/n \ll \eta$. Prior to our work, the best algorithms for the sparse parity regime (up to constants in the exponent) were brute-force-search in time $\binom{n}{k}$ or, for larger k, simply using the best LPN algorithm, i.e. BKW, for the non-sparse secret case. In other words, restricting the LPN problem to k-sparse parities did not lead to any speedup (asymptotically in the exponent) other than the trivial brute-force-search over all parities of size k, for any sparsity regime. However in this work, we presented two new algorithms and exhibits two regimes where one can improve upon the above, asymptotically in the exponent. The first regime corresponds to constant noise rate, and parities of sparseness k such that k is (loosely) between $n/\log(n)$ and $n/\log^2(n)$. The second regime of improvement corresponds to low noise rate which can be, for example, $\log(n)/\sqrt{n}$ and parities of sparseness $k = \sqrt{n}/\log\log(n)$. In both of the cases we compared our proposed algorithm to the state-of-the-art algorithm. Specifically, for the first case we showed an improvement over BKW algorithm and brute-force algorithm. Similarly, for the second case, we showed an improvement over brute-force and a lucky brute-force (as it is defined in Section 4.3.4).

This work provides new insights into the LPN problem and the quantitative effects on security of sampling the secret from a high entropy distribution that is different from the noise distribution. We also motivated the study of improvements to other learning classes, i.e. DNF and Juntas. One natural extension of this work, is to explore what other type of learning algorithms can be improved by using the underlying LPN algorithm we developed in this work.

Finally, in Chapter 5, we motivated the study of partial key exposure in Ring-LWE (RLWE)-based cryptosystems. Specifically we presented and implemented an efficient key exposure attack that, given certain 1/4-fraction of the coordinates of the NTT transform of the RLWE secret, along with samples from the RLWE distribution, recovers the full RLWE secret for standard parameter settings. As it was mentioned in Section 5.1.2, our attacks work in a highly structured leakage scenario and reducing the structure of the leakage in our attack seems to be an interesting future work.

Appendix A: Appendix of Cache Side-Channel Attack on Database

A.1 Closest Vector Problem (CVP)

Given n-linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in \mathbb{R}^m$, the lattice generated by $\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_n$ is the set of all the integer linear combination of them i.e. $\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_n) = \{\sum_1^n \mathbf{b}_i x_i \mid x_i \in \mathbb{Z}\}$. The set $\{\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_n\}$ is called the basis of the lattice and is presented by matrix $\mathbf{B}$ in which basis $\mathbf{b}_i$ is i-th row of the matrix. In the closest vector problem target vector $\mathbf{y}$ is given. The target vector $\mathbf{y}$ does not necessarily belong to lattice $\mathcal{L}$. The solution is a lattice point $\mathbf{y}' = \mathbf{x}\mathbf{B}$ which is closest to target vector $\mathbf{y}$ and also $\mathbf{y}' \in \mathcal{L}$. Notice that a lattice point $\mathbf{y}'$ is a linear combination of basis, while the target vector $\mathbf{y}$ is not. The significance of the CVP problem is to find a closest vector to $\mathbf{y}$ such that the linear combination is satisfied. CVP problem is also known to be NP-complete and we use `fplll` [44] for finding the closest vector in lattice.

A.2 Error Reduction Step

As explained in Section 3.2.3 and Section 3.2.4 by using the noisy clique-finding and Match & Extend algorithms on the noisy data we get some *close* answer to the real database. Here we outline a technique which can reduce the noise and output a *more* *accurate* answer. The first step is to compute all the $\binom{N}{2} + N$ volumes corresponding to each range. Specifically, the ranges $[1, 1], [1, 2], \ldots, [1, N]$ are obtained using noisy clique-finding or Match & Extend. Each range $[i, j]$ can be computed from the elementary volumes as $\big|[i, j]\big| = \big|[1, j]\big| - \big|[1, i-1]\big|$. Instead of taking the computed value for range $[i, j]$, we choose the value in the set of volumes (obtained from the side-channel data) that is closest to this computed value. This procedure results in $N' = \binom{N}{2} + N$ volumes which we call *candidate volumes.* Now note that given the volumes of the ranges $[1, 1], [2, 2], \ldots, [N, N]$, the volume of any other range $[i, j]$ can be expressed as a *linear combination* of these values. Therefore, our variable $\vec{x} = (x_1, \ldots, x_N)$ corresponds to the volumes of the ranges $[1, 1], [2, 2], \ldots, [N, N]$ and our candidate volumes $\vec{v} = (v_1, \ldots, v_{N'})$, correspond to noisy linear combinations of the x_i's. Thus, solving for the $\vec{x}$ which yields the closest solution to $\vec{v} = (v_1, \ldots, v_{N'})$ under the linear constraints, corresponds to solving a Closest Vector Problem (CVP).

For example, if the range has size $N = 3$, then we obtain a total of 6 volumes $v_1, \ldots, v_6$ corresponding to the ranges $[1, 1]$, $[2, 2]$, $[3, 3]$, $[1, 2]$, $[2, 3]$, $[1, 3]$ and can construct the following system of equations:

$$\mathbf{A}\vec{x} + \vec{e} = \vec{v} \qquad \text{where} \qquad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$\vec{v} = (v_1, \ldots, v_6)$, $\vec{e}$ is the amount of error and $\vec{x}$ is unknown. To solve this problem, we can consider the lattice defined by $\mathbf{A}\vec{z}$, where $\mathbf{A}$ is the basis and $\vec{z}$ is any integer vector. Now, given $\vec{v}$, we would like to find the closest lattice vector $\vec{y} = \mathbf{A}\vec{x'}$ to $\vec{v}$. Once we have $\vec{y}$, we can solve to get $\vec{x'}$. To create a full rank matrix for our solver, we can modify matrix A as following:

$$A' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & T & 0 & 0 \\ 0 & 1 & 1 & 0 & T & 0 \\ 1 & 1 & 1 & 0 & 0 & T \end{bmatrix}$$

where $T \gg n$ and $\vec{v}$ stays the same.

Now we obtain a solution of dimension 6 (as opposed to dimension 3), but the last three coordinates should always be 0, since if they are non-zero there will be at least $\pm\ T$ in the corresponding coordinate of $\vec{v}$, which will clearly not be the closest vector.

A.3 Error Reduction Step Experiments

Table A.1 and Table A.2 compare the clique and Match & Extend algorithms and the improvement achieved by the error reduction step using the CVP solver. Recall that the error percentage is computed for each recovered coordinate. We measured the quality of the recovered databases in two ways: in Table A.1 we report the average value of the error percentage over all the volumes in all recovered databases. For Table A.2, we compute for

	Noisy Clique		Match & Extend	
Experiment	No CVP	CVP	No CVP	CVP
Experiment I	0.21%	0.19%	0.09%	0.07%
Experiment II	0.16%	0.10%	0.09%	0.08%
Experiment IV	0.80%	0.77%	0.09%	0.08%

Table A.1: Average Error Percentage

	Noisy Clique		Match & Extend	
Experiment	No CVP	CVP	No CVP	CVP
Experiment I	0.86%	0.86%	0.22%	0.20%
Experiment II	0.55%	0.38%	0.30%	0.22%
Experiment IV	2.29%	2.30%	0.24%	0.22%

Table A.2: Maximum Error Percentage

each database the largest error percentage of its coordinates, and we report the average of all these maxima over all databases in Experiments I, II and IV. The effectiveness of CVP is for the cases where the initial reconstructed database is within some *close* distance of the correct database. Hence for the other two experiments, the plain CVP is not effective as the initial recovered database is rather far from the correct answer, and generally CVP can not be effective. Moreover, the CVP algorithm needs to have all the initial volumes, so for the experiments where some of the volumes are missing, CVP can not be used.

It can be seen that for Match & Extend algorithm the average error percentage is reduced from 0.09 to 0.07, from 0.09 to 0.08 and from 0.09 to 0.08 in Experiments I, II and IV, respectively. Table A.2 shows similar results for maximum error percentage. For Match & Extend algorithm the maximum error percentage is reduced from 0.22 to 0.20, from 0.30 to 0.22 and from 0.24 to 0.22 in Experiments I, II and IV, respectively. Table A.1 and Table A.2 present the L_1 norm and L_∞ norm, respectively. The CVP solver optimizes for L_2 norm, so it has a larger effect on decreasing L_∞ norm than L_1 norm. In case the objective is to minimize the L_1 norm, an integer programming approach would be preferable.

Appendix B: Appendix of New Algorithms for LPN with Sparse Parities

B.1 Proof of Lemma 21

We first show that each coordinate of $\mathbf{x}'$ is set to 0 with independent probability $(1+p)/2$. The probability that a coordinate j of $\mathbf{x}'$ in sample s^p is set to 0 after running $\mathsf{BKW_R}$ can be computed as follows:

$$\begin{aligned}\Pr[\mathbf{x}'[j]=0] &= \Pr[\mathbf{x}'[j]=0 \mid j \in R]\cdot \Pr[j \in R] + \Pr[\mathbf{x}'[j]=0 \mid j \notin R]\cdot \Pr[j \notin R] \\ &= 1\cdot p + 1/2 \cdot (1-p) = (1+p)/2\end{aligned}$$

To show that the label b' is correct with probability η' and that the correctness of the label is independent of the instance $\mathbf{x}', \mathbf{s}$, note that $\mathbf{x}'$ is always constructed by XOR'ing a set of exactly $2^{\mathfrak{a}}$ number of samples and that the choice of the set of XOR'ed samples depends only on the random coins of the algorithm and on the $\mathbf{x}$ values, which are independent of the e value. Therefore, we can apply Lemma 9 to conclude that the noise is independent and that b' is correct with probability $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$.

B.2 Proof of Theorem 22

From the description of $\mathsf{BKW_R}$, it is clear to see that it takes $O(\mathfrak{a}2^{\mathfrak{b}})$ LPN samples and running time to generate a p-biased sample, where $\mathfrak{a} = \log(2np)/2, \mathfrak{b} = \lceil |R|/\mathfrak{a} \rceil$. Remember that the $\mathsf{BKW_R}$ algorithm will abort if $|R| \geq 2pn$ or $|R| \leq pn/2$, i.e. $\mathsf{Event1}$ occurs. By showing that $\mathsf{Event1}$ occurs with probability at most $2\exp(-p \cdot n/8)$, we obtain that $\mathsf{BKW_R}$ runs in time $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ with probability at least $1 - 2\exp(-p \cdot n/8)$.

To bound the probability of $\mathsf{Event1}$ occurring, we notice that by multiplicative Chernoff bounds in Theorem 1, we can bound the size of set R as follows:

$$\Pr\left[|R| \geq 2pn\right] \leq \exp(-p \cdot n/3)$$

$$\Pr\left[|R| \leq pn/2\right] \leq \exp(-p \cdot n/8)$$

$$\Pr\left[|R| \geq 2pn \vee |R| \leq pn/2\right] \leq \exp(-p \cdot n/3) + \exp(-p \cdot n/8) \leq 2\exp(-p \cdot n/8)$$

$$\Pr\left[pn/2 < |R| < 2pn\right] > 1 - 2\exp(-p \cdot n/8)$$

B.3 Proof of Lemma 23

Before proving Lemma 23, we present the following simple claims about the number of samples needed to estimate the Fourier Coefficient of a single index. Based on Claim 1, the magnitude of Fourier coefficient of the indexes with secret value of 0 is equal to 0, while for the secret coordinates 1 that is equal to $\varepsilon = (1 - 2\eta') \cdot p^{k-1}\sqrt{1-p^2}$. In the Following Claim we compute how many samples are needed to estimate the magnitude of Fourier coefficient within distance of $\varepsilon/2$ of correct value. We will bound the failure probability with δ/n.

Claim 3. *For every $j \in [n]$, $\hat{b}_p(\{j\}) = \mathbb{E}[b \cdot \chi_{\{j\},p}(\mathbf{x})]$, where $(\mathbf{x}, b) \sim \mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$, can be estimated within additive accuracy $\frac{\varepsilon}{2}$ and confidence $1 - \frac{\delta}{n}$ using $\frac{8}{\varepsilon^2} \cdot \frac{1+p}{1-p} \cdot \ln(2n/\delta)$ number of samples.*

Proof. The estimate of $\hat{b}_p(\{j\})$ based on the m samples $\mathbf{s}_i^p = (\mathbf{x}_i, b_i)$ is

$$\hat{b}_{\text{estimate}}(\{j\}) = \frac{1}{m} \sum_{i=1}^{m} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$$

and notice that $\mathbb{E}\left[\hat{b}_{\text{estimate}}(\{j\})\right] = \hat{b}_p(\{j\})$. Lets denote $X_i = \frac{1}{m} \cdot b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, then note that $|X_i| \leq (1/m)\sqrt{\frac{1+p}{1-p}}$. Finally by Chernoff-Hoeffding bound of Theorem 2, we have the following.

$$\Pr\left[\left|\hat{b}_{\text{estimate}}(\{j\}) - \hat{b}_p(\{j\})\right| \geq \varepsilon/2\right] \leq 2\exp\left(\frac{-m\varepsilon^2}{8} \cdot \frac{1-p}{1+p}\right)$$

Bounding the right hand side by δ/n and solving for m gives the desired value for number of samples. □

Proof of Lemma 23. Invoking Claim 1, we have that for j such that $\mathbf{s}[j] = 1$, $\hat{b}_p(\{j\}) = (1 - 2\eta') \cdot p^{k-1}\sqrt{1-p^2}$ while for j such that $\mathbf{s}[j] = 0$, $\hat{b}_p(\{j\}) = 0$. It is clear by inspection that Algorithm 4.2 succeeds when it correctly estimates the values of $\hat{b}_p(\{j\})$ to within additive $\varepsilon/2 := (1 - 2\eta') \cdot p^{k-1}\sqrt{1-p^2}/2$ for all $j \in [n]$. By Claim 3, $\frac{8}{\varepsilon^2} \cdot \frac{1+p}{1-p} \cdot \ln(2n/\delta)$ number of samples are sufficient to estimate a single coordinate within additive $\varepsilon/2$ of its correct value with confidence $1 - \frac{\delta}{n}$. By a union bound, the success probability of estimating all coordinates to within additive $\varepsilon/2$ is $1-\delta$. $\square$

B.4 Proof of Lemma 26

The proof is similar to the proof of Lemma 21 and noticing that the SamP algorithm uses $2np+1$ samples to generate a single p-biased sample. Two p-biased samples $\mathbf{x}'_i, \mathbf{x}'_j$, $j > i$ are pairwise independent, unless the same linear combination of samples in $\mathcal{S}$ was used to generate both of them. But in that case, during execution, the condition $\mathbf{x}'_j|_{R_i} = 0^{|R_i|}$ would evaluate to true, which means that Event2 occurred and so fresh samples (not from $\mathcal{S}$) would be used to generate $\mathbf{x}'_j$.

In the rest of the proof we switch to the ± 1 representation instead of the Boolean representation. The sample $\mathsf{s}_i^p = (\mathbf{x}'_i, b'_i)$ is obtained from the samples in set $\mathcal{O}_i$ alongside some extra error samples from Noise Oracle $\tilde{\mathcal{O}}_\eta$. In the following proof these are denoted by $e_1, e_2, \ldots, e_{2np+1}$. Moreover, notice that the sample $\mathsf{s}_j^p = \left(\mathbf{x}'_j, b'_j\right)$, obtained from set $\mathcal{O}_j$, has at most t elements in common with the sample obtained from the set $\mathcal{O}_i$. Hence we can represent the error in sample $\mathsf{s}_j^p = \left(\mathbf{x}'_j, b'_j\right)$ as $e_1, e_2, \ldots, e_t, e''_{t+1} \ldots e''_{2np+1}$. For the ease of notation we assumed that the t samples which are in common are at index 1 to t.

$$
\begin{aligned}
\mathrm{Cov}[e'_i, e'_j] &= \mathrm{Cov}[e_1 \cdot e_2 \ldots e_t \cdot e_{t+1} \ldots e_{2np+1}\ ,\ e_1 \cdot e_2 \ldots e_t \cdot e''_{t+1} \ldots e''_{2np+1}] \\
&= \mathbb{E}[e_1^2 \cdot e_2^2 \ldots e_t^2 \cdot e_{t+1} \ldots e_{2np+1} \cdot e''_{t+1} \ldots e''_{2np+1}] \\
&\qquad - \mathbb{E}[e_1 \cdot e_2 \ldots e_{2np+1}]\ \mathbb{E}[e_1 \cdot e_2 \ldots e_t \ldots e''_{t+1} \ldots e''_{2np+1}] \\
&= (1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}
\end{aligned}
$$

Where the last line follows from the independence of errors, $\mathbb{E}[e_i] = 1 - 2\eta$ and $\mathbb{E}[e_i^2] = 1$.

B.5 Proof of Theorem 27

Assuming $\mathsf{Event1}$ and $\mathsf{Event2}$ do not occur, the sample complexity and runtime can be verified by inspection and assuming RLK takes $\mathsf{poly}(np)$ time. It remains to bound the probability of $\mathsf{Event1}$ and $\mathsf{Event2}$. We can upper bound the probability of $\mathsf{Event1}$ by $2\exp(-p \cdot n/8)$, as in the proof of Theorem 22. To upperbound the probability of $\mathsf{Event2}$, we note that assuming $\mathsf{Event1}$ does not occur, $\mathsf{Event2}$ occurs only if one of the following two events occur:

- $\mathsf{Event'1}$: For some distinct $i, j \in \mathsf{maxnum}$, $|R_i \cap R_j| \geq np/4$.
- $\mathsf{Event'2}$: For some distinct $i, j \in \mathsf{maxnum}$, $|R_i \setminus R_j| \geq np/4$ and $\mathbf{x}'_j|_{R_i \setminus R_j} = 0^{|R_i \setminus R_j|}$.

Since for distinct i, j, each coordinate $\ell \in [n]$ is placed in *both* R_i and R_j with probability p^2, by a union bound over all pairs i, j and a standard Chernoff bound, $\mathsf{Event'1}$ can be upperbounded by:

$$\mathsf{maxnum}^2 \cdot \exp(-n/48) = (np)^t \cdot \exp(-n/48).$$

Since for any $\mathbf{x}'_j$, the coordinates outside of R_j are uniformly random, $\mathsf{Event'2}$ can be upperbounded by:

$$\mathsf{maxnum}^2 \cdot 1/2^{np/4} = (np)^t \cdot 1/2^{np/4}.$$

B.6 Proof of Lemma 28

Similar to Section 4.2.2, before proving Lemma 28, we first present the following claim about the number of samples needed to estimate the Fourier Coefficient of a single index. The algorithm gets access to $8\log(n)$ sets of p-biased samples. In the following claim we first prove how many samples are needed to be able to approximate the Fourier coefficient within additive distance of $\epsilon/2$ and later discuss how by repeating the approximation step, i.e. step 2b in Figure 4.4, will reduce the error in approximation even further.

Claim 4. *For $\delta \in [0,1]$, $p \in (0,1)$, given $8\log(n)$ independent sets of samples $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{8\log(n)}$ that each of size $\mathsf{num} := O\left(\frac{1}{(1-2\eta)^{4np+2}p^{2(k-1)}(1-p^2)}\right)$ and each satisfying the properties given in Lemma 26 for some $t \in \Theta(1/\eta)$, then for every $j \in [n]$, $\hat{b}_p(\{j\}) = \mathbb{E}[b \cdot \chi_{\{j\},p}(\mathbf{x}))]$ can be estimated within additive accuracy $\frac{\epsilon}{2} = (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{2np+1}$ with confidence $1 - \frac{\delta}{n}$.*

Proof. Let $X = \frac{1}{m}\sum_{i=1}^{m} b_i \cdot \chi_{S,p}(\mathbf{x}_i)$. Let f be a parity function. Assuming $S = \{k\}$, let $X_i = \frac{1}{m} \cdot b_i \cdot \chi_{\{k\},p}(\mathbf{x}_i)$. First we compute $\mathrm{Cov}[X_i, X_j]$ for k such that $\mathbf{s}[k] = 1$

$$\mathrm{Cov}[X_i, X_j] = \mathrm{Cov}\left[\frac{1}{m} \cdot b'_i \cdot \chi_{\{k\},p}(\mathbf{x}'_i)\ ,\ \frac{1}{m} \cdot b'_j \cdot \chi_{\{k\},p}(\mathbf{x}'_j)\right]$$

$$\begin{aligned}\mathrm{Cov}[X_i, X_j] &= \frac{1}{m^2} \cdot \mathrm{Cov}\left[b'_i \cdot \chi_{\{k\},p}(\mathbf{x}'_i)\ ,\ b'_j \cdot \chi_{\{k\},p}(\mathbf{x}'_j)\right] \\ &= \frac{1}{m^2} \cdot \mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u]\right) \cdot e'_i \cdot \frac{\mathbf{x}'_i[k]-p}{\sqrt{1-p^2}}\ ,\ \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v]\right) \cdot e'_j \cdot \frac{\mathbf{x}'_j[k]-p}{\sqrt{1-p^2}}\right]\end{aligned} \tag{B.1}$$

$$= \frac{1}{m^2} \cdot \frac{1}{1-p^2} \Bigg(\mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}'_i[u] \right) \cdot e'_i \ , \ \left(\prod_{v:\mathbf{s}[v]=1 \wedge v \neq k} \mathbf{x}'_j[v] \right) \cdot e'_j \right] -$$

$$\mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}'_i[u] \right) \cdot e'_i \ , \ p \cdot \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v] \right) \cdot e'_j \right] -$$

$$\mathrm{Cov}\left[p \cdot \left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u] \right) \cdot e'_i \ , \ \left(\prod_{v:\mathbf{s}[v]=1 \wedge v \neq k} \mathbf{x}'_j[v] \right) \cdot e'_j \right] +$$

$$\mathrm{Cov}\left[p \cdot \left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u] \right) \cdot e'_i \ , \ p \cdot \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v] \right) \cdot e'_j \right] \Bigg) \tag{B.2}$$

$$= \frac{1}{m^2} \cdot \frac{1}{(1-p^2)} \left(p^{2(k-1)} \mathrm{Cov}\left[e'_i, e'_j\right] - 2p^{2k} \mathrm{Cov}\left[e'_i, e'_j\right] + p^{2(k+1)} \mathrm{Cov}\left[e'_i, e'_j\right] \right) \tag{B.3}$$

$$= m^{-2} p^{2(k-1)} (1-p^2) \mathrm{Cov}\left[e'_i, e'_j\right]$$

$$= m^{-2} p^{2(k-1)} (1-p^2) \left[(1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2} \right] \tag{B.4}$$

Where equation (B.1) follows from definition of Fourier Coefficients and noting that b'_i is multiplications of $\mathbf{x}_i$s and error term e_i, equation (B.2) follows from properties of Covariance, equation (B.3) follows from independence of $\mathbf{x}'_i$s and equation (B.4) follows from Lemma 26. We can also bound $\mathrm{Var}[X_i]$ as follows

$$\mathrm{Var}[X_i] = \mathrm{Var}\left[\frac{1}{m} \cdot b'_i \cdot \chi_{\{k\},p}(\mathbf{x}'_i) \right]$$

$$= \frac{1}{m^2} \cdot \mathrm{Var}\left[\left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u] \right) \cdot e'_i \cdot \frac{\mathbf{x}'_i[k] - p}{\sqrt{1-p^2}} \right]$$

$$= \frac{1}{m^2} \cdot \frac{1}{1-p^2} \left(\mathrm{Var}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}'_i[u] \right) \cdot e'_i \right] - p^2 \cdot \mathrm{Var}\left[\left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_i[u] \right) \cdot e'_i \right] \right)$$

$$= \frac{1}{m^2} \cdot \frac{1}{1-p^2} \left(\mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}_i'^2[u] \right) \cdot e_i'^2 \right] - \mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}_i'[u] \right) \cdot e_i' \right]^2 - \right.$$

$$\left. p^2 \cdot \mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}_i'^2[u] \right) \cdot e_i'^2 \right] + p^2 \cdot \mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}_i'[u] \right) \cdot e_i' \right]^2 \right) \tag{B.5}$$

$$= \frac{1}{m^2} \cdot \frac{1}{1-p^2} \left(1 - p^{2(k-1)}(1-2\eta)^{2np} - p^2 + p^{2(k+1)}(1-2\eta)^{2np}\right) \tag{B.6}$$

$$= m^{-2}\left(1 - p^{2(k-1)}(1+p^2)(1-2\eta)^{2np}\right) \leq m^{-2}$$

Where equation (B.5) follows from properties of variance and equation (B.6) follows from independence of $\mathbf{x}_i'$s. Then we have the following bound from Chebyshev's bound of Theorem 3

$$\Pr\left[|X - \mathbb{E}[X]| \geq \varepsilon/2\right] \leq \frac{\sum_{i=1}^{m} \mathrm{Var}\left[X_i\right] + 2\sum_{i=1}^{m}\sum_{j>i} \mathrm{Cov}\left[X_i, X_j\right]}{\varepsilon^2/4}$$
$$\leq 4 \cdot \frac{m^{-1} + p^{2(k-1)}(1-p^2)\left[(1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}\right]}{\varepsilon^2}$$

By substituting $\varepsilon = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{2np+1}$, we can bound the right hand side by a constant less than 1/2 by setting $t < -\frac{\ln(9/8-1/c)}{2\ln(1-2\eta)}$ and setting $m = c \cdot \frac{1}{(1-2\eta)^{4np+2}p^{2(k-1)}(1-p^2)}$, where $c > 8$. We use random variable $Y_{i'}$ to represents whether the value of count in step i' is increased or not, specifically $Y_{i'} = 1$ represents the event that count is increased in step i'. Assume we repeat the protocol for T rounds in total. Let $Y = (1/T) \cdot \sum_{i'=1}^{T} Y_{i'}$. First, take the case that j such that $\mathsf{s}[j] = 0$, we know that in each step of loop over i', $\Pr[Y_{i'} = 1] = 1/2 - \epsilon$. Note that the algorithm is run T times using independent sets $\mathcal{S}_{i'}$ each time and index j is only added if in the majority of the runs its estimated Fourier coefficient is more than $\varepsilon/2$. Using Chernoff bound, we can bound $\Pr[Y \geq T/2] \leq 1/n$.

$$\begin{aligned}\Pr[\text{index } j \text{ is added to set } \mathcal{S}'] &= \Pr[\mathsf{count} \geq T/2] \\ &= \Pr\left[\frac{\sum_{i'=1}^{T} Y_{i'}}{T} \geq \frac{1}{2}\right] \leq \Pr\left[|Y - E[Y]| > \varepsilon\right] \leq 2\exp(-2T\varepsilon^2)\end{aligned}$$

We can bound the right hand side by $\frac{\delta}{n}$ for constant δ by setting $T = 8\log(n)$ and $\varepsilon = 1/4$. Similar argument applies to the case for j such that $\mathbf{s}[j] = 1$. □

Proof of Lemma 28. Invoking Claim 1, we have that for j such that $\mathbf{s}[j] = 1$, $\hat{b}_p(\{j\}) = (1 - 2\eta') \cdot p^{k-1}\sqrt{1 - p^2}$ while for j such that $\mathbf{s}[j] = 0$, $\hat{b}_p(\{j\}) = 0$. It is clear by inspection that Algorithm in Figure 4.4 succeeds when it correctly estimates the values of $\hat{b}_p(\{j\})$ to within additive $\varepsilon/2 := (1 - 2\eta') \cdot p^{k-1}\sqrt{1 - p^2}/2$ for all $j \in [n]$. By Claim 4, we need $8\log(n)$ sets such that each set has $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ number of p-biased samples. So in total $\mathsf{num} \cdot 8\log(n) = O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)} \cdot \log(n)\right)$ number of p-biased samples are sufficient to estimate a single coordinate within additive $\varepsilon/2$ of its correct value with confidence $1 - \frac{\delta}{n}$. By a union bound, the success probability of estimating all coordinates to within additive $\varepsilon/2$ is $1 - \delta$. □

Appendix C: Appendix of (In)Security of Ring-LWE Under Partial Key Exposure

C.1 Reconstructing the Secret Given ($\alpha_1, \alpha_2 \mod n'$) Leakage

Let $\vec{e}_u^{\alpha}(x)$ be the degree $u = n/n'$ polynomial that is obtained by taking $\vec{e}(x)$ modulo $x^u - (\omega^{\alpha})^u$. We consider two polynomials $\vec{e}_u^{\alpha_1}(x)$ and $\vec{e}_u^{\alpha_2}(x)$. We may assume without loss of generality, $\alpha_1 = 1$. We therefore set $\alpha := \alpha_2$. For $i \in \{0, \dots, u-1\}$, the $(i+1)$-st coefficient of $\vec{e}_u(x)$ and $\vec{e}_u^{\alpha}(x)$ are as follows, respectively

$$e_i + \omega^u \cdot e_{i+u} + \omega^{2 \cdot u} \cdot e_{i+2 \cdot u} + \ldots + \omega^{(n'-1) \cdot u} \cdot e_{i+(n'-1) \cdot u}$$

$$e_i + \omega^{\alpha \cdot u} \cdot e_{i+u} + \omega^{\alpha \cdot 2 \cdot u} \cdot e_{i+2 \cdot u} + \ldots + \omega^{\alpha \cdot (n'-1) \cdot u} \cdot e_{i+(n'-1) \cdot u}$$

Similar to the previous attack, we obtain the following constraints on the error, given leakage on the secret key and an RLWE sample,

$$\begin{bmatrix} 1 & \omega^{u} & \omega^{2\cdot u} & \cdots & \omega^{(n'-1)\cdot u} \\ 1 & \omega^{\alpha\cdot u} & \omega^{\alpha\cdot 2\cdot u} & \cdots & \omega^{\alpha\cdot(n'-1)\cdot u} \end{bmatrix} \cdot \begin{bmatrix} e_i \\ e_{i+u} \\ e_{i+2\cdot u} \\ \vdots \\ e_{i+(n'-1)\cdot u} \end{bmatrix} = \begin{bmatrix} e_{u,i} \\ e^{\alpha}_{u,i} \end{bmatrix}$$

We solve a corresponding CVP instance to find the "most likely" solution, $\vec{e}_i$ for $(e_i, e_{i+u}, e_{i+2\cdot u}, \ldots, e_{i+(n'-1)\cdot u})$, since the "most likely" solution is the one with smallest norm. Similar to our previous attack, our goal is to carefully choose the answers with "high confidence" such that

(1) In total, we must guess at least u number of n'-dimensional solutions, $\vec{e}_i^{\,j}$, from all the obtained solutions $\left[\vec{e}_i^{\,j}\right]_{j\in[\ell],i\in[u]}$.

(2) With high probability *all* our guesses are correct.

We choose the candidate which has probability of at least, say, 0.95 of being correct solution. The total probability of success for this case is $0.95^u = 0.95^{n/n'}$. Our experiments in Section 5.2.2 again show that we can obtain enough "high" confidence solutions, without requiring too large a number of RLWE instances.

C.2 Description of Basic Attack

In this section, we present the **Basic Attack**, following the description from [83, 84, 85] and using the fact that `NTT` coefficients form a CRT representation. We first recall definition of CRT representation in our setting of parameters.

Definition 42 (CRT Representation)**.** *For $\vec{p} \in R_q$, and ω a m^{th} primitive root of unity in $\mathbb{Z}_q^*$, CRT representation for $\vec{p}$ is defined as*

$$CRT(\vec{p}) = \left(\vec{p}(\omega^{j_1}), \ldots, \vec{p}(\omega^{j_n})\right),$$

for $j_i \in \mathbb{Z}_m^*$.

Remember our setting, $m = 2n$ for n a power of 2; hence, It is easy to see that $\text{CRT}(\vec{p}) = (\widehat{p}_0, \ldots, \widehat{p}_{n-1})$. We first introduce the following definition:

Definition 43 (Hybrid Leaky RLWE Distribution)**.** *For $j \in \mathbb{Z}_{2n}^* = \{1, 3, \ldots, 2n-1\}$, a "secret" $s \in R_q$, and a distribution χ over R_q, a sample from the distribution $D^{\vec{s},j}_{real,n',\mathcal{S}}$ is generated by choosing $\left(\widehat{\vec{a}}, \widehat{\vec{b}}\right) \leftarrow D^{\vec{s}}_{real,n',\mathcal{S}}$ and outputting $\left(\widehat{\vec{a}}, \widehat{\vec{b}} + \vec{u}\right)$, where $\vec{u} = (u_1, u_3, \ldots, u_{2n-1}) \in \mathbb{Z}_q^n$ with u_i, $i \in \mathbb{Z}_{2n}^*$ defined as follows: u_i is chosen uniformly at random from $\mathbb{Z}_q$ if $i \neq \alpha' \mod 2n'$ for all $\alpha' \in \mathcal{S}$ and $i \leq j$, $u_i = 0$ otherwise.*

Define $D^{\vec{s},-1}_{real,n',\mathcal{S}} := D^{\vec{s}}_{real,n',\mathcal{S}}$. Additionally, notice that $D^{\vec{s},2n-1}_{real,n',\mathcal{S}} = D^{\vec{s}}_{sim,n',\mathcal{S}}$. Thus if, for any $(n', \mathcal{S} \subseteq \mathbb{Z}_{2n'}^*)$ adversary $\mathcal{A}$ running in time $t := t(n)$ distinguishes $D^{\vec{s}}_{real,n',\mathcal{S}}$ from $D^{\vec{s}}_{sim,n',\mathcal{S}}$ with probability $p := p(n)$, then there is some index $j \in \mathbb{Z}_{2n}^*$ such that $j \neq \alpha' \mod n$ for all $\alpha' \in \mathcal{S}$ and a distinguisher $\mathcal{D}_j$ that is able to distinguish between the distribution $D^{\vec{s},j-2}_{real,n',\mathcal{S}}$ and $D^{\vec{s},j}_{real,n',\mathcal{S}}$ with probability at least p/n.

We now show the distinguisher $\mathcal{D}_j$ can be used to construct an algorithm that finds the value of $\hat{s}_j$. The idea of this algorithm is to try each of the possible values $\hat{s}_j$, constructing the samples on inputs from $D^{\vec{s}}_{real,n',\mathcal{S}}$, so that the samples are distributed according to $D^{\vec{s},j-2}_{real,n',\mathcal{S}}$ if $\hat{s}_j$ is guessed correctly, and the samples are distributed according to $D^{\vec{s},j}_{real,n',\mathcal{S}}$

otherwise. Then using the distinguisher $\mathcal{D}_j$, which runs in time t, for $\mathsf{poly}(n/p)$ times for each of the $q(= \mathsf{poly}(n))$ guesses for $\hat{s}_i$, we are able to find the correct value of $\hat{s}_j$ with probability $1 - 1/\mathsf{poly}(n)$ in time $t \cdot \mathsf{poly}(n) \cdot 1/p$.

Next we present the samples construction algorithm that takes a guess $g \in \mathbb{Z}_q$ and transform $D^{\vec{s}}_{real,n',\mathcal{S}}$ to either $D^{\vec{s},j-2}_{real,n',\mathcal{S}}$ or $D^{\vec{s},j}_{real,n',\mathcal{S}}$. On each sample $\left(\widehat{\vec{a}}, \widehat{\vec{b}}\right) \leftarrow D^{\vec{s}}_{real,n',\mathcal{S}}$, it outputs a sample

$$\left(\vec{a}', \vec{b}'\right) = \left(\widehat{\vec{a}} + \vec{v}, \widehat{\vec{b}} + \vec{u} + g\vec{v}\right),$$

where $\vec{u} = (u_1, u_3, \ldots, u_{2n-1}), \vec{v} = (v_1, v_3, \ldots, v_{2n-1}) \in \mathbb{Z}_q^n$ are chosen as follows: u_k is uniform in $\mathbb{Z}_q$ if $k < j$, $k \neq \alpha' \mod 2n'$ for all $\alpha' \in \mathcal{S}$, and the rest are 0; v_k is uniform in $\mathbb{Z}_q$ if $k = j$, and the rest are 0. Note that b'_j can be written as

$$b'_j = \hat{a}_j \hat{s}_j + \hat{e}_j + u_j + g v_j = a'_j \hat{s}_j + \hat{e}_j + u_j + (g - \hat{s}_j) v_j.$$

Observe that if g is the correct guess, then $(g - \hat{s}_j)v_j = 0$. The distribution of $\left(\vec{a}', \vec{b}'\right)$ is identical to $D^{\vec{s},j-2}_{real,n',\mathcal{S}}$. If g is a wrong guess, $(g - s_j)$ is non-zero. Since q is prime, $(g - \hat{s}_j)v_j$ is uniform in $\mathbb{Z}_q$. Thus the distribution of $\left(\vec{a}', \vec{b}'\right)$ is identical to $D^{\vec{s},j}_{real,n',\mathcal{S}}$.

C.3 Pseudocode of Attack from Section 5.2

Partial Key Exposure Attack

Given leaked coordinates on NTT version of secret key $\widehat{\vec{s}}$, public key a and a public value b, recover all coordinates of $\vec{s}$

```
1: [$\bar{\omega}$, B] = create_basis()
   ▷ create basis used in CVP solver and $\bar{\omega}$ being $\left[1, \omega^{u}, \omega^{2\cdot u}, \cdots, \omega^{(n'-1)\cdot u}\right]$
2: bTotal = [], aTotal = []
3: count = 0
4: $u = n/n'$
5: for $j \in [1, 2, \ldots,$ RLWESamples$]$ do
6:     $\vec{A}^j$ =create_a($\widehat{\vec{a}}^{j}$)        ▷ Create circulant matrix corresponding to $\widehat{\vec{a}}^{j}$
7:     $\widehat{\vec{e}}^{j} = \widehat{\vec{b}}^{j} - \widehat{\vec{a}}^{j} \cdot \widehat{\vec{s}}$
   ▷ For all leaked coordinate of $\widehat{\vec{s}}$ we compute the corresponding coordinates of error $\widehat{\vec{e}}^{j}$
8:     $\vec{e} =$ Lagrange polynomials($\widehat{\vec{e}}^{j}$)
   ▷ Recover the coeffiecient of polynomial obtained by taking $\vec{e}(x)$ modulo $(x^u - (\omega^\alpha)^u)$
9:     aMat = [], bTemp = []
10:    for $i \in [0, 1, 2, \ldots, u-1]$ do
11:        $X = \bar{\omega}$.solve_right($\vec{e}_i$)        ▷ Solving the system of equation explained in Section 5.2
12:        $Y$ =CVP.closest_vector(B, $X$)
13:        $\bar{\vec{e}}_i = X - Y$
14:        if Prob($\bar{\vec{e}}_i$) > Threshold then
15:            aMat.append$\left(\widehat{\vec{A}}^{j}[i, i+u, i+2\cdot u, \ldots, i+(n'-1)\cdot u][:]\right)$
   ▷ Select the corresponding rows from $\widehat{\vec{a}}^{j}$ and save them
16:            bTemp.append$\left(\widehat{\vec{b}}^{j}[i, i+u, i+2\cdot u, \ldots, i+(n'-1)\cdot u] - \bar{\vec{e}}_i\right)$
   ▷ Select the corresponding rows from $\widehat{\vec{b}}^{j}$, subtract $\bar{\vec{e}}_i$ from it to get noiseless system
17:            count += $n'$
18:        end if
19:    end for
20:    aTotal.append(aMat)
21:    bTotal.append(bTemp)
22:    if count == n then
23:        break
24:    end if
25: end for
26: try:
27:   sk = aTotal.solve_right(bTotal)        ▷ solve the noiseless system to recover key
28: except:
29:   return error        ▷ couldn't solve the system
30: return sk
```

Figure C.1: Description of Partial Key Exposure Attack from Section 5.2

Bibliography

[1] Cloud Services. `https://www.datamation.com/cloud-computing/slideshows/top-10-cloud-apps.html`. Accessed: 2020-05-08.

[2] Microsoft Wins Pentagon's $10 Billion JEDI Contract, Thwarting Amazon. `https://www.nytimes.com/2019/10/25/technology/dod-jedi-contract.html`. Accessed: 2020-05-08.

[3] The Experiments and the Code for Algorithms. `https://github.com/ariashahverdi/database_reconstruction`.

[4] Source Code, July 2019. `https://github.com/ariashahverdi/RLWE`.

[5] *Introduction to the HCUP Nationwide Inpatient Sample (NIS) 2008*, (accessed June 15, 2020).

[6] Onur Acıiçmez. Yet Another Microarchitectural Attack: Exploiting I-cache. In *Proceedings of the 2007 ACM workshop on Computer security architecture*, pages 11–18. ACM, 2007.

[7] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[8] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-Preserving Encryption for Numeric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 563–574, 2004.

[9] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 474–495. Springer, Heidelberg, March 2009.

[10] Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic Algorithms for LWE Problems. working paper or preprint, October 2014.

[11] Martin R. Albrecht, Amit Deo, and Kenneth G. Paterson. Cold boot attacks on ring and module LWE keys under the NTT. *IACR TCHES*, 2018(3):173–213, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/7273`.

[12] Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila. Newhope: Algorithm specification and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2017.

[13] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016. `http://eprint.iacr.org/2016/1157`.

[14] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[15] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop Van de Pol, and Yuval Yarom. Amplifying Side Channels Through Performance Degradation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 422–435. ACM, 2016.

[16] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2013.

[17] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

[18] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

[19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2:4, 2019.

[20] Naomi Benger, Joop Van de Pol, Nigel P Smart, and Yuval Yarom. "Ooh Aah... Just a Little Bit": A small amount of side channel can go a long way. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 75–92. Springer, 2014.

[21] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece: conservative code-based cryptography. *NIST submissions*, 2017.

[22] Avrim Blum, Merrick L. Furst, Jeffrey C. Jackson, Michael J. Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using fourier analysis. In *26th ACM STOC*, pages 253–262. ACM Press, May 1994.

[23] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Heidelberg, August 1994.

[24] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant Learning, the Parity Problem, and the Statistical Query Model. *J. ACM*, 50(4):506–519, 2003.

[25] Andrej Bogdanov, Manuel Sabin, and Prashant Nalini Vasudevan. XOR Codes and Sparse Learning Parity with Noise. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 986–1004, 2019.

[26] Sonia Bogos, Florian Tramer, and Serge Vaudenay. On Solving LPN using BKW and Variants. *Cryptography and Communications*, 8(3):331–369, 2016.

[27] Madalina Bolboceanu, Zvika Brakerski, Renen Perlman, and Devika Sharma. Order-LWE and the Hardness of Ring-LWE with Entropic Secrets. Cryptology ePrint Archive, Report 2018/494, 2018. `https://eprint.iacr.org/2018/494`.

[28] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 224–241. Springer, Heidelberg, April 2009.

[29] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[30] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 563–594, 2015.

[31] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. *Journal of Cryptology*, 26(3):513–558, July 2013.

[32] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51st FOCS*, pages 501–510. IEEE Computer Society Press, October 2010.

[33] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Heidelberg, April / May 2018.

[34] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679. ACM, 2015.

[35] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Annual Cryptology Conference*, pages 353–373. Springer, 2013.

[36] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[37] Jung Hee Cheon, Yongha Son, and Donggeon Yhee. Practical FHE Parameters Against Lattice Attacks. *IACR Cryptol. ePrint Arch.*, 2021:39, 2021.

[38] Eric Crockett and Chris Peikert. Challenges for Ring-LWE. *IACR Cryptology ePrint Archive*, 2016:782, 2016.

[39] Dana Dachman-Soled, Vitaly Feldman, Li-Yang Tan, Andrew Wan, and Karl Wimmer. Approximate Resilience, Monotonicity, and the Complexity of Agnostic Learning. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 498–511. SIAM, 2014.

[40] Dana Dachman-Soled, Huijing Gong, Hunter Kippen, and Aria Shahverdi. BKW Meets Fourier New Algorithms for LPN with Sparse Parities. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 658–688, Cham, 2021. Springer International Publishing.

[41] Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. On the leakage resilience of ideal-lattice based public key encryption. Cryptology ePrint Archive, Report 2017/1127, 2017. `https://eprint.iacr.org/2017/1127`.

[42] Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. *Security of NewHope Under Partial Key Exposure*, pages 93–125. Springer International Publishing, 2020.

[43] Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. (In)Security of Ring-LWE Under Partial Key Exposure. *Journal of Mathematical Cryptology*, 15(1):72–86, 2021.

[44] The FPLLL development team. FPLLL, a Lattice Reduction Library. Available at `https://github.com/fplll/fplll`, 2016.

[45] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 361–381. Springer, Heidelberg, February 2010.

[46] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *51st FOCS*, pages 511–520. IEEE Computer Society Press, October 2010.

[47] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 621–630. ACM Press, May / June 2009.

[48] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, October 2008.

[49] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *International Conference on Cryptology in Africa*, pages 282–305. Springer, 2018.

[50] V. Feldman, P. Gopalan, S. Khot, and A. K. Ponnuswami. New Results for Learning Noisy Parities and Halfspaces. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 563–574, 2006.

[51] Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On Agnostic Learning of Parities, Monomials, and Halfspaces. *SIAM Journal on Computing*, 39(2):606–645, 2009.

[52] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[53] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware. *Journal of Cryptographic Engineering*, 8(1):1–27, 2018.

[54] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. How to encrypt with the LPN problem. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 679–690. Springer, Heidelberg, July 2008.

[55] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 230–240. Tsinghua University Press, January 2010.

[56] Elena Grigorescu, Lev Reyzin, and Santosh Vempala. On Noise-tolerant Learning of Sparse Parities and Related Problems. In *International Conference on Algorithmic Learning Theory*, pages 413–424. Springer, 2011.

[57] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, Gauss, and Reload–a Cache Attack on the Bliss Lattice-based Signature Scheme. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 323–345. Springer, 2016.

[58] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 315–331. ACM, 2018.

[59] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In *IEEE Symposium on Security and Privacy (S&P) 2019*, 2019.

[60] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse Attacks Against Order-revealing Encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 655–672. IEEE, 2017.

[61] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache Template Attacks: Automating Attacks on Inclusive Last-level Caches. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 897–912, 2015.

[62] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache Games – Bringing Access-Based Cache Attacks on AES to Practice. In *2011 IEEE Symposium on Security and Privacy*, pages 490–505, 2011.

[63] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using Covering Codes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2014.

[64] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 346–365, 2012.

[65] Sanghyun Hong, Michael Davinroy, Yigitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitras. Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks. *CoRR*, abs/1810.03487, 2018.

[66] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Dana Dachman-Soled, and Tudor Dumitraş. How to 0wn the NAS in Your Spare Time. In *International Conference on Learning Representations*, 2020.

[67] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 52–66. Springer, Heidelberg, December 2001.

[68] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. S $ A: A shared cache attack that works across cores and defies VM sandboxing–and its application to AES. In *2015 IEEE Symposium on Security and Privacy*, pages 591–604. IEEE, 2015.

[69] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! A fast, Cross-VM Attack on AES. In *International Workshop on Recent Advances in Intrusion Detection*, pages 299–319. Springer, 2014.

[70] Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 293–308. Springer, Heidelberg, August 2005.

[71] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 703–720. Springer, Heidelberg, December 2009.

[72] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016.

[73] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data Recovery on Encrypted Databases with k-nearest Neighbor Query Leakage. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1033–1050. IEEE, 2019.

[74] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 599–616, 2020.

[75] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[76] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *International conference on security and cryptography for networks*, pages 348–359. Springer, 2006.

[77] Kevin Lewi and David J Wu. Order-revealing Encryption: New Constructions, Applications, and Lower Bounds. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1167–1178, 2016.

[78] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 725–734. ACM Press, June 2011.

[79] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *J. ACM*, 40(3):607–620, 1993.

[80] Richard J Lipton, Evangelos Markakis, Aranyak Mehta, and Nisheeth K Vishnoi. On the Fourier Spectrum of Symmetric Boolean Functions with Applications to Learning Symmetric Juntas. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 112–119. IEEE, 2005.

[81] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[82] Vadim Lyubashevsky. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In *Approximation, randomization and combinatorial optimization. Algorithms and techniques*, pages 378–389. Springer, 2005.

[83] Vadim Lyubashevsky. Search to decision reduction for the learning with errors over rings problem. In *2011 IEEE Information Theory Workshop, ITW 2011, Paraty, Brazil, October 16-20, 2011*, pages 410–414, 2011.

[84] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[85] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6):43:1–43:35, November 2013.

[86] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

[87] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. Cryptology ePrint Archive, Report 2013/293, 2013. `http://eprint.iacr.org/2013/293`.

[88] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 89–106. Springer, Heidelberg, March 2011.

[89] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis.* Cambridge university press, 2017.

[90] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 69–90. Springer, 2017.

[91] Elchanan Mossel, Ryan O'Donnell, and Rocco P Servedio. Learning Juntas. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 206–212, 2003.

[92] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[93] Noam Nisan and Avi Wigderson. Hardness vs Randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994.

[94] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: the Case of AES. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.

[95] Chris Peikert. How (not) to instantiate ring-LWE. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 411–430. Springer, Heidelberg, August / September 2016.

[96] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer, Heidelberg, April 2009.

[97] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[98] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM*, 56(6), September 2009.

[99] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[100] Aria Shahverdi, Mahammad Shirinov, and Dana Dachman-Soled. Database Reconstruction from Noisy Volumes: A Cache Side-Channel Attack on SQLite. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1019–1035. USENIX Association, August 2021.

[101] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[102] Katherine E. Stange. Algebraic aspects of solving Ring-LWE, including ring-based improvements in the Blum-Kalai-Wasserman algorithm. Cryptology ePrint Archive, Report 2019/183, 2019. `https://eprint.iacr.org/2019/183`.

[103] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 23(1):37–71, Jan 2010.

[104] Y. TSUNOO. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. *Proc. ISITA2002, Oct.*, 2002.

[105] Gregory Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2012.

[106] Karsten Verbeurgt. Learning DNF under the Uniform Distribution in Quasi-Polynomial Time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, COLT '90, page 314–326, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[107] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[108] Yuval Yarom. Mastik: A Micro-architectural Side-channel Toolkit. *Retrieved from School of Computer Science Adelaide: http://cs. adelaide. edu. au/˜ yval/Mastik*, 2016.

[109] Yuval Yarom and Naomi Benger. Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack. *IACR Cryptology ePrint Archive*, 2014:140, 2014.

[110] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, 2014. USENIX Association.

[111] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-VM Side Channels and their use to Extract Private Keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[112] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 990–1003, New York, NY, USA, 2014. Association for Computing Machinery.