

DNSPIR: Private Information Retrieval Optimized for Privacy-Preserving DNS Lookups*

Lea Nürnberger

Independent Researcher

lea2.nuernberger@web.de

Mattia Veroni

Norwegian University of Science and Technology

mattia.veroni@ntnu.no

Simon Pohmann

Royal Holloway, University of London

simon@pohmann.de

Christian Weinert

Royal Holloway, University of London

Christian.Weinert@rhul.ac.uk

Abstract

The Domain Name System (DNS) was conceived with reliability, speed, and scalability in mind, with limited consideration for privacy. Different solutions have been proposed to make DNS more privacy preserving, mostly focusing on confidentiality in transit and networking approaches to achieve unlinkability. However, DNS queries are still observable in plaintext during processing.

In this work, we investigate how to achieve privacy against all involved servers through private DNS lookups via single-server private information retrieval (PIR). For this, we design a new PIR protocol that is optimized for the DNS setting. In addition to using ideas from NTTlessPIR (Li et al., CRYPTO 2024), we design a new algorithm to efficiently perform RLWE-based matrix-vector multiplication. As opposed to previous methods, it does not require NTT-friendly plaintext moduli, uses only a single key-switching key, and causes significantly lower noise growth.

For a single database of 2 billion entries (the estimated number of domains in the world), our protocol allows clients to perform PIR queries with about 80KB of communication, and less than 5 seconds of server computation time. This outperforms all previous PIR schemes in terms of communication cost, at only slightly increased runtime. Additionally, our end-to-end evaluation of iterative DNS resolution across three nameservers shows sub-second lookups in smaller settings and under 5 seconds with 264KB communication for querying .com domains, which can be practical when used selectively for privacy-sensitive web browsing activities.

Keywords

domain name system, private information retrieval, homomorphic encryption

1 Introduction

The *Domain Name System* (DNS) was first defined in 1987 (RFC1033, RFC1035). A client that wants to connect to a website usually knows the domain name of the website, but to connect, they require the IP address of the server hosting the sought website. The

task of the DNS structure is therefore to provide a map between memorable domain names and IP addresses. This is done in a distributed fashion, for efficiency and robustness: the information is redundantly spread over many thousand servers, all often holding no more than a thousand entries each, and are selectively contacted. However, in an era of large-scale data analysis and user profiling, DNS query data could potentially be leveraged to additionally infer users' browsing behaviour.

Several solutions to achieve some level of privacy in DNS have been proposed over the past 35 years [67]. We give an overview in Table 1. Since DNSSEC (RFC2065, proposed in 1997) only guarantees authentication and integrity, schemes like DNSCurve [24], DNSCrypt [25], and the standardized DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH) introduced encryption of DNS packets during transmission, but do not provide privacy against the DNS resolver. Other solutions change the DNS structure and still allow a single resolver or two colluding servers to link the client to the queried domain. Examples for this are the Paged Domain Name System [6], which follows a “hiding-in-the-crowd” approach by collecting information on authoritative nameservers at TLD level, Oblivious-DNS (ODNS) [39, 68], which splits the resolver role between two servers, each oblivious of either the identity of the client or the content of the query, and EncDNS [33], which forwards encrypted queries from a conventional resolver to an additional EncDNS server using a sender-anonymous tunnel. Finally, Cloudflare introduced a DNS resolver that is reachable as an onion service, allowing DNS queries to be routed entirely through the Tor network [19].

We observe that being in essence a database query, although a distributed one, the DNS lookup can be additionally secured by handling the DNS resolution via a private information retrieval (PIR) approach. Thus, in this work, we investigate how large a step we can take towards private DNS lookup via PIR.

1.1 PIR for Private DNS Lookups

In most deployments today, a client issues a *recursive* DNS query to a resolver, typically operated by the user's ISP or a public DNS provider, thus delegating the entire resolution process and expecting a complete response. This resolver maintains a cache of past queries for efficiency and, upon a cache miss, resolves the requested

*Please cite the conference version of this paper published in PoPETS'27 [56].

	Leakage to		Collusion resistant	Changes to		
	Resolver	Nameserver		Client	Resolver	Nameserver
DoH, DoT	full query	full query	NA	use encryption	use encryption	use encryption
Paged DNS [6]	full query	query group	✓	none	query pages	serve pages
Oblivious DNS [68]	anon. query	anon. query	✗	use encryption	none	special new server
EncDNS [33]	anon. query	anon. query	✗	use encryption	none	special new server
DNS over TOR [19]	anon. query	anon. query	✗	use TOR	none	none
Xiao et. al. [73]	none	truncated query	✓	use PIR	use PIR	none
Ours (DNSPIR)						
Iterative	NA	query $\in$ domain	✓	use PIR	removed	use PIR
Global	NA	none	✓	use PIR	removed	use centralized PIR

Table 1: Comparison of different approaches for DNS privacy and required infrastructure changes. “Anon. query” means that the adversary learns that a DNS record has been queried, but not by whom. “Truncated query” means that the nameserver learns the query, truncated one level below the current zone; in particular, the final nameserver learns the whole query. Finally, “query $\in$ domain” means that the nameserver only learns a single bit: that the client queried some name belonging to the subtree of the DNS namespace that it serves, which is the minimum possible amount of leakage in a decentralized setting.

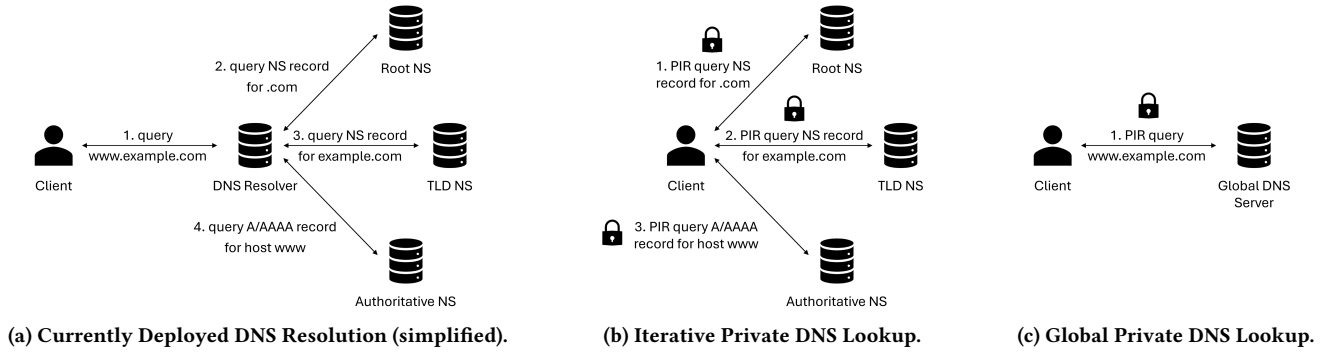


Figure 1: Simplified currently deployed DNS resolution and our privacy-preserving PIR-based alternatives.

domain name, typically in an *iterative* fashion. In doing so, the resolver may contact one of the DNS root servers, which refer to the appropriate top-level domain (TLD) nameservers; these, in turn, point to the nameservers for the queried domain. Ultimately, the authoritative nameserver returns the relevant DNS records.

While this design is efficient and scalable, it has important privacy implications. During recursive resolution, the resolver learns the full queried domain name. In addition, each nameserver learns information about the query specific to its role in the hierarchy: Root nameservers learn the queried top-level domain, while TLD nameservers learn the second-level domain and can infer that the query concerns one of the domains under their administration. Finally, authoritative nameservers learn the full hostname.

To mitigate this leakage, we consider a client-driven iterative DNS resolution combined with private information retrieval (PIR). In this setting (“iterative DNS”), each nameserver returns either the requested DNS record (if found) or a reference to the next nameserver in the resolution chain. The client then continues the resolution process by querying the next nameserver directly. By using PIR for each of these queries, the client can retrieve the required information without revealing which specific entries are requested.

This approach preserves the decentralized structure of DNS while minimizing the information gained by nameservers. In particular, TLD nameservers, which are currently able to track web activities for a large number of users, gain almost no relevant information in the new setting. Moreover, this appears to be the strongest form of privacy that can be achieved without requiring non-collusion assumptions or transitioning to a centralized DNS architecture. It is also possible to layer our privacy-preserving client-driven iterative resolution with networking approaches such as ODoH [39, 68] or DNS over Tor [19] to achieve unlinkability. Finally, we remark that the complete removal of DNS resolvers has been proposed before by Schomp et al. [69]; they emphasized improved security (unrelated to privacy) as the main motivation, and argued that the impact on the global DNS deployment, also in terms of performance, would be less dramatic than it might seem.

In addition to this decentralized approach, we also consider a centralized DNS model (“global DNS”), in which all domain-to-IP mappings and auxiliary domain information are stored on a single logical server (potentially replicated and distributed for redundancy and throughput). While such a model would require a substantial departure from the current DNS infrastructure and is therefore

less practical, it enables very strong privacy guarantees. In particular, the security properties of PIR ensure that the server learns no information about the queried domain during the lookup phase.

In this work, we study both the client-driven iterative approach and the centralized alternative (as visualized in Figure 1), and benchmark our PIR protocol in each setting.

1.2 Requirements for PIR Selection

For a PIR protocol to be applicable in the private DNS lookup case, it must satisfy the following properties:

- Efficiency on large databases with small entries. For this work, we estimate the number of publicly accessible and regularly queried hostnames registered in DNS to be about 2^{31} , which is consistent with the number reported by different research tools [40, 65] and the number reported for certificate transparency logs [37]. Each consists of at most 255 octets, linked to an IP-address of at most 32 bits for IPv4 and 128 bits for IPv6.
- Low communication overhead. While server-side computation costs are an important consideration for PIR protocols, they can be mitigated substantially through implementation and deployment choices: latency can be reduced with well-engineered GPU implementations, and monetary costs for compute time can be amortized by batching queries from multiple clients. Communication overhead, in contrast, is largely inherent to the protocol and cannot be reduced through faster hardware or implementation optimizations. We therefore consider communication efficiency a particularly important requirement for PIR in the private DNS setting. This is especially true for clients on mobile devices, for which we expect bandwidth to be the dominant cost factor. In Appendix D, we estimate the worldwide median cost of mobile data at approximately \$1.46 per GB; for PIR schemes, the resulting communication costs can thus significantly exceed server-side computation costs.
- No client-dependent offline phase. In DNS, nameservers serve many different clients, and it is not realistic for a server to store data that is specific to each client. Likewise, clients usually query many different nameservers, without knowing in advance which server will be contacted next. Therefore, it would often be the case that the per-client offline phase can only be run when the query is known, which essentially makes it a part of the online phase. In particular, this excludes schemes like SimplePIR [32] or FrodoPIR [21], which requires the client to download a hint during an offline phase, and schemes like Spiral [51], which require the client to upload key material during an offline phase.
- Efficient updateability or re-preprocessing of the database. DNS records are frequently added, updated, or deleted. Therefore, the PIR scheme should be able to accommodate frequent updates, although a limited latency until changes become visible is usually acceptable.

- Single-server PIR. Multi-server PIR protocols often imply a higher communication cost, and their security relies on the different servers to be non-colluding, an assumption that may be hard to justify in practice.
- Keyword-based search. The client has little to no knowledge on the DNS database, and cannot be expected to know at which index the IP-address of the queried domain name is stored. Thus, the PIR protocol needs to support keyword-based lookups.

1.3 Relevant Related Work on PIR

The field of PIR has seen vast development since the problem was originally proposed in 1998 [18]. Besides multi-server protocols (e.g., [1, 36, 42, 70]), computationally secure single-server constructions exist with preprocessing (cf. [4, 8, 27, 29, 30, 47, 52, 57]) and without preprocessing (cf. [32, 41, 49, 50, 55]). As shown in our survey of single-server PIR protocols Table 2, the protocols that closest match our criteria laid out above are NTTlessPIR and HintlessPIR [41], as well as InsPIRe [49]. While NTTlessPIR was proposed as a subroutine for HintlessPIR, it is also suitable as a standalone PIR protocol. Its main advantages are simplicity and performance, but at the cost of requiring many megabytes of communication. Therefore, decreasing this communication cost is the main goal of our work.

We note that reducing the communication cost of [41] has already been the objective of previous work, most notably InsPIRe [49] and [45]. InsPIRe did so by proposing a novel LWE-to-RLWE packing technique, which requires very little key material and can significantly decrease response sizes. On the other hand, [45] focuses on TensorPIR, which was also proposed by [41].

Despite having a similar goal, the techniques we propose in this work are completely disjoint from the techniques in prior works. In particular, we optimize NTTlessPIR instead of the more complex HintlessPIR or TensorPIR schemes, completely avoid the use of large GSW-style keys, and remove the query expansion phase to reduce noise growth. As a result, we achieve even lower communication costs than InsPIRe and [45], albeit at somewhat increased computation costs. We present a quantitative comparison with InsPIRe in Section 4.4. Unfortunately, we cannot compare with [45], since [45] does not report results for the database sizes we consider, and no open-source implementation is available. However, for a 16GB database, the authors report a communication cost of 7.57MB, which is far higher than what our protocol achieves.

Keyword PIR. In many PIR applications, the database is indexed by keywords (in our case hostnames), and thus it is necessary to use a keyword-PIR protocol. Apart from early proposals based on homomorphic comparisons [48], modern keyword-PIR protocols are usually built from standard index-PIR schemes using hashing. For example, ChalametPIR [15] proposed a general transformation from index-PIR to keyword-PIR, based on any key-value filter. Concrete instantiations most commonly use Binary Fuse Filters, as done by [15], or Cuckoo filters [58], as done for PIR by, e.g., [3] and for private set intersection protocols [16, 17, 61–63]. However, since Binary Fuse Filters and Cuckoo Filters use multiple hash functions (usually 2-3), many of these protocols have the disadvantage

Protocol	Offline Phase		Client Storage	Online Phase	
	Per-client	Time/Comm.		Communication	Server time
SealPIR (Angel et al., S&P’18) [4]	✗	low	✗	moderate	high
Spiral (Menon and Wu, S&P’22) [51]	✓	large	✗	small	moderate
FrodoPIR (Davidson et al., PETS’23) [21]	✓	moderate	✓	large	moderate
SimplePIR (Henzinger et al., USENIX Security’23) [32]	✓	large	✓	moderate	low
WhisPIR (de Castro et al., ePrint’24) [23]	✗	low	✗	moderate	moderate
PianoPIR (Zhou et al., S&P’24) [74]	✓	large	✓	small	very low
HintlessPIR (Li et al., CRYPTO’24) [41]	✗	moderate	✗	large	low
YPIR (Menon and Wu, USENIX Security’24) [52]	✗	low	✗	large	very low
KsPIR (third proposal; Luo et al., CCS’24) [44]	✗	low	✗	large	moderate
Respire (Burton et al., CCS’24) [13]	✓	moderate	✗	small	moderate
ShiftPIR (Wang et al., CCS’25) [72]	✓	large	✓	small	moderate
InsPIRe (Mahdavi et al., S&P’26) [49]	✗	low	✗	moderate	low
Ours (DNSPIR)	✗	moderate	✗	small	moderate

Table 2: Qualitative comparison of single-server PIR protocols with open-source implementation in chronological order. Communication is categorized as small (< 100 KB), moderate (100 – 1000 KB), and large (> 1000 KB), using the values reported by the original work (for an 8 GB database), and extrapolated if not available. Similarly, (single threaded) server time is categorized as very low (< 1 s), low (1 – 5 s), moderate (5 – 30 s), and high (> 30 s). Finally, we classify offline cost based on one-time server-side preprocessing time for schemes without per-client preprocessing as low (< 1000 s) and moderate (1000 – 3600 s); for schemes with per-client preprocessing, on the other hand, it is more natural to consider offline communication per client, which we classify as low (< 1 MB), moderate (1 – 10 MB), and large (> 10 MB).

of performing multiple index-PIR queries, which increases cost by the same factor. An exception is the recent work of Patel et al. [59], who use simple hashing together with a novel way of encoding buckets within the linear structure of PIR schemes based on matrix multiplication. In this paper, we use simple hashing to derive indices from domain names, but discuss incorporating the techniques of [59] in Section 3.4.

1.4 Related Work on PIR-based Private DNS

Recently, Xiao et al. [73] made an alternative proposal for privacy-preserving DNS based on PIR. In particular, they use PIR to achieve privacy against curious DNS resolvers by protecting queries made against comparatively small resolver caches. In case of cache misses, this however still requires unprotected iterative queries against nameservers, and the following cache update protocol is vulnerable to time correlation attacks. Moreover, they do not provide innovations on a PIR-level but instead make black-box use of Spiral [51], which requires per-client preprocessing.

In contrast, we provide privacy against curious nameservers and a significantly improved PIR protocol that is optimized for the DNS use case. As part of future work, our PIR protocol could also be used to query resolver caches, and our iterative approach for querying nameservers could be combined with the work of Xiao et al. [73] to reduce their privacy leakage. A significant research question though is how to construct an oblivious cache update protocol (potentially involving ORAM techniques) when using PIR both for privacy-preserving resolver cache and iterative nameserver queries.

1.5 Contributions and Technical Overview

Our main contribution is the design and implementation of a new PIR scheme, which is explicitly optimized for the DNS use case. To this end, we propose novel techniques to further improve the NTTlessPIR protocol [41], in particular to reduce communication costs. We believe these techniques are of general interest, and might also be useful in other settings. In more detail, our contributions are as follows.

1.5.1 Using non-NTT-friendly moduli. SimplePIR [32] is based on LWE, but most PIR schemes building on top of it, in particular NTTlessPIR [41], are based on Ring-LWE, and thus the ring $\mathcal{R}_t := \mathbb{Z}_t[X]/(X^N + 1)$ as plaintext space (for a power of two N). To make use of the large plaintext space while still having database entries in $\mathbb{Z}_t$, we need to pack a vector of elements of $\mathbb{Z}_t$ into $\mathcal{R}_t$. In NTTlessPIR and many other schemes, this is done using the slot decomposition of $\mathcal{R}_t$ for suitable t . Indeed, if $t \equiv 1 \pmod{2N}$, then $\mathcal{R}_t$ is naturally isomorphic to $\mathbb{F}_t^N$, and thus well suited to pack vectors.

The disadvantage of this method is that it requires $t \equiv 1 \pmod{2N}$, so in particular $t > 2N$. Unfortunately, the noise growth of homomorphic operations scales with t , and so choosing $t > 2N$ is undesirable. To avoid this, we propose a new matrix-vector multiplication algorithm (cf. Lemma 3.2) that assumes that the vector $a \in \mathbb{Z}_t^N$ is embedded into $\mathcal{R}_t$ using the coefficient embedding, i.e., as $\sum_i a_i X^i \in \mathcal{R}_t$. Our algorithm is as efficient as the classical, slot-based method, and works with any odd t .

1.5.2 Sparse secrets, and revisiting ciphertext splitting. In this work, we follow the common approach to PIR based on matrix multiplication, which is also used by NTTlessPIR. The idea is to arrange the database DB into a $n_{\text{rows}} \times n_{\text{cols}}$ matrix, and have the server compute

the homomorphic product $\text{resp} = \text{DB} \cdot \text{qry}$, where qry is an encryption of a unit vector. However, this means that qry and resp are vectors of length n_{cols} and n_{rows} respectively, and since both have to be transmitted between server and client, the communication cost is at least $\max\{n_{\text{rows}}, n_{\text{cols}}\} \geq \sqrt{n}$.

In order to further reduce it, two techniques have been proposed. SealPIR [4] and DoublePIR [32] split the database into smaller matrices $\text{DB}_1, \dots, \text{DB}_r$, and compute a matrix-vector product $\text{resp}_i = \text{DB}_i \cdot \text{qry}$ for each of them. The client then retrieves only the resp_i it is interested in, by considering $(\text{resp}_1, \dots, \text{resp}_r)$ to be a new database, and running a second instance of PIR on it. The main disadvantage of this method is that the resp_i are ciphertexts, and thus larger (by a factor F , called *ciphertext expansion factor*) than the original plaintexts. Hence, every stage of PIR on the result of the previous stage introduces an overhead of F in terms of response size. On the other hand, most modern PIR schemes (e.g. [49, 51, 54]) use the technique introduced by OnionPIR [54], based on GSW-style external products. This has the disadvantage of requiring larger queries, but can achieve much lower response sizes.

Nevertheless, we argue that in our case, the ciphertext splitting technique of SealPIR [4] can be optimized to outperform the GSW-style external product for two reasons. Firstly, our algorithm causes much lower noise growth than previous work. Secondly, we propose to use a sparse secret key, which is common in the FHE literature [10, 20], but to our knowledge has not previously been done for PIR. These two choices combined give us a ciphertext expansion ratio of only $F = 3$ (as a comparison, the parameters of SealPIR required $F = 7$). With such a small ciphertext expansion ratio, we find that the query size is the main bottleneck of our protocol, hence a ciphertext-splitting approach is advantageous.

1.5.3 Avoiding query expansion. Most modern PIR schemes [51, 54] first expand the client’s query, and use the expanded query in the matrix-vector product. This means that the relatively expensive expansion procedure only has to be performed once, and its result can be reused for every part of the database $\text{DB}_1, \dots, \text{DB}_r$. However, this method also comes with a disadvantage. Usually, the expansion procedure requires homomorphic Galois automorphisms, which add a large noise term to the ciphertexts. In the matrix-vector product step, this noise is then multiplied by the database entries, thus necessitating the choice of relatively large cryptographic parameters to avoid noise overflow.

On other hand, the matrix multiplication algorithm we propose does not require query expansion, and homomorphically applies Galois automorphisms after the ciphertexts have been multiplied with the database entries. This means the noise from the automorphisms only accumulates additively, which allows us to choose significantly smaller parameters. This further decreases the communication bandwidth of our method, but at the cost of higher online computation, since we apply homomorphic Galois automorphisms for each of the database slices. However, since we use the NTTless-paradigm introduced by NTTlessPIR, Galois automorphisms are relatively cheap to compute.

1.5.4 Using only rotations by a single Galois key. So far, we still need $N - 1$ rotations and therefore need to transmit $N - 1$ Galois

keys, which together have a size equivalent to that of the whole database. Therefore, to prevent excessive query sizes, we adapt our algorithm to compute the linear combination of query conjugates, as required by our matrix multiplication algorithm, to only use a single Galois key. This was previously done by WhisPIR [23] and InsPIRe [49], but since they both use a query expansion step, we cannot reuse their algorithm. The concrete algorithm is given in Lemma 3.1.

1.5.5 Practical Optimizations. In order to reach a response time that might be considered acceptable in the context of DNS lookup, we also optimize our scheme with regards to concrete parameters and hardware. In particular, our implementation uses AVX512 instructions to speed up arithmetic operations. This raises the question of how to do finite field arithmetic, since AVX512 does not support integer multiplication whose output has 128 bits of precision. Experimenting with parameters for our scheme would suggest a ciphertext modulus q of around 50 bits. This means that, unfortunately, we cannot directly use Barrett [7] or Montgomery [53] reductions, since their algorithms require intermediate results of size at least q^2 . We solve this issue by choosing a very specific modulus q , slightly above 50 bits of size, that factors as

$$q = 2251387614257153 = (2^{25} - 2^{12} + 1) \cdot (2^{26} - 2^{12} + 1)$$

In particular, elements modulo q can be represented in RNS format, i.e., modulo both $2^{25} - 2^{12} + 1$ and $2^{26} - 2^{12} + 1$. Both these factors are NTT friendly, and their special form allows a very fast reduction algorithm, which uses only $32 \times 32 \rightarrow 64$ bit multiplications. This reduction algorithm is similar (but simpler) to the techniques proposed in [43].

1.5.6 Evaluation. We implement and benchmark our PIR protocol, both in isolation (cf. Section 4.4) and as part of a full end-to-end iterative DNS resolution (cf. Section 4.5). Due to the above techniques, we achieve significantly lower communication cost than previous work [4, 41, 49, 54]. The best previous PIR protocol in terms of communication cost is InsPIRe [49], but nevertheless, our implementation achieves more than $3\times$ lower communication. To the best of our knowledge, only protocols that transmit key material during an offline phase [13, 38], and thus do not count it towards their total communication, have lower communication overhead.

For the end-to-end DNS benchmark, we consider three nameservers on root-, TLD-, and authoritative level, respectively, each serving their zone through PIR. The lookup performance varies significantly, mainly depending on the size of the TLD zone, and ranges from less than 5 seconds and 264KB of communication (for queries involving .com domains) to less than one second for the much smaller .org TLD. While the performance overhead is still too high for general use, we believe that our protocol can be practical when used selectively, e.g., for privacy-sensitive web browsing activities.

2 Preliminaries

2.1 Homomorphic Encryption

Homomorphic Encryption (HE) schemes are encryption schemes that allow an untrusted party to perform computations on encrypted data without ever decrypting. Many HE schemes have been proposed in the literature, but in this work, we focus on the BFV

scheme [11, 26]. BFV is defined over a number ring $\mathcal{R}$, which for this work will always be the ring of integers of a power-of-two cyclotomic number field, i.e., $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ with N a power of two. For an element $x \in \mathcal{R}$, we define the norm $\|x\|$ as the ℓ_∞ -norm of the coefficients x_i in the representation $x = \sum_{i=0}^{N-1} x_i X^i$. Finally, for an integer $t \geq 1$, we write $\mathcal{R}_t$ for the quotient $\mathcal{R}/t\mathcal{R}$.

The Galois group of $\mathcal{R}$ is isomorphic to $(\mathbb{Z}/2N\mathbb{Z})^*$, and thus for each $g \in (\mathbb{Z}/2N\mathbb{Z})^*$, there is a Galois automorphism

$$\sigma_g : \mathcal{R} \rightarrow \mathcal{R}, \quad X \mapsto X^g \bmod (X^N + 1).$$

For $N \geq 4$, the element 5 generates an index-2 subgroup of $(\mathbb{Z}/2N\mathbb{Z})^*$, and the whole group is thus generated by 5 and -1 .

In addition to the ring $\mathcal{R}$, the BFV scheme is parameterized by a plaintext modulus t and a ciphertext modulus q . The secret key is then chosen as a small element $s \in \mathcal{R}$, usually with ternary coefficients, i.e. in $\{-1, 0, 1\}$. Encryption of a plaintext $m \in \mathcal{R}_t$ then produces a tuple $\text{Enc}(m, s) = (c_0, c_1) \in \mathcal{R}_q^2$ with $c_0 + c_1 s = \frac{q}{t}m + e$ for a small error $e \in (\mathcal{R} \otimes \mathbb{Q})/q\mathcal{R}$. If the error is chosen from a suitable distribution, this results in a secure scheme, assuming the hardness of the Ring Learning with Errors (RLWE) problem [46, 66]. For more details on the security of BFV, we refer the reader to the original work [26].

As long as the error term e remains small, BFV is linearly homomorphic: for ciphertexts $(c_0, c_1), (c'_0, c'_1)$ with $c_0 + c_1 s = m + e$ and $c'_0 + c'_1 s = m' + e'$, clearly $(c_0 + c'_0, c_1 + c'_1)$ is an encryption of $m + m' \in \mathcal{R}_t$ with error term $e + e'$. Similarly, $(m'c_0, m'c_1)$ is an encryption of $m'm$ with error term $m'e$.

In addition to this, BFV also supports homomorphic multiplications and Galois automorphisms, based on the key-switching machinery of [12]. Since only Galois automorphisms are relevant for this work, we skip the presentation of homomorphic ciphertext-ciphertext multiplications here, and refer the reader to [26] instead. Let (c_0, c_1) be a BFV ciphertext w.r.t. secret key s and σ be a Galois automorphism of $\mathcal{R}$. Then, since σ preserves the size of e , we find that $(\sigma(c_0), \sigma(c_1))$ is an encryption of $\sigma(m)$ w.r.t. secret key $\sigma(s)$. To obtain the desired encryption of $\sigma(m)$ under s , we can perform key-switching: let $\beta \geq 2$ and $\text{ksk}_{s' \rightarrow s}$ be a key-switching key from s' to s , consisting of tuples $(k_{i0}, k_{i1}) \in \mathcal{R}_q^2$ with $k_{i0} + k_{i1}s = \beta^i s' + e_i$. For $x \in \mathcal{R}_q$, write further $\text{Decomp}(x)_{\beta,q}$ for a vector $(x_i)_{i \leq r}$, $r = \lceil \log_\beta(q) \rceil - 1$ with $x = \sum_i x_i \beta^i$ and each $\|x_i\| \leq \beta$. Then $\text{KeySwitch}((c_0, c_1), \text{ksk}_{s' \rightarrow s})$ is defined as the inner product

$$\left\langle c_0 + \left\langle (k_{i0})_i, \text{Decomp}_{\beta,q}(c_1) \right\rangle, \left\langle (k_{i1})_i, \text{Decomp}_{\beta,q}(c_1) \right\rangle \right\rangle.$$

This is an encryption of m under s , with the error term increased by an additive amount (scaling with β). Combined, this yields the algorithm $\text{Auto}((c_0, c_1), g, \text{ksk}_{\sigma_g(s) \rightarrow s}) := \text{KeySwitch}((\sigma_g(c_0), \sigma_g(c_1)), \text{ksk}_{\sigma_g(s) \rightarrow s})$ for homomorphic Galois automorphism evaluation.

2.2 Private Information Retrieval

We now recall the definition and the security notion of a generic private information retrieval (PIR) protocols.

Definition 2.1 (PIR Protocol). Let $\lambda \in \mathbb{N}$ be a security parameter and let DB be a database with n entries managed by the server. A PIR

protocol is a tuple of three algorithms $\Pi = (\text{Query}, \text{Respond}, \text{Extract})$ defined as follows:

Query($1^\lambda, i$): Given an index i , the algorithm outputs a query qry and a secret extraction key s .

Respond(qry, DB): Given a query qry , the algorithm returns a response resp .

Extract(resp, s): Given a response resp and an extraction key s , the algorithm outputs a potential database entry.

Note that some PIR protocols apply a preprocessing algorithm on the database. Moreover, if the database is shaped as a matrix, the query may consist of two indices (row and column) instead of a single one.

Definition 2.2 (Correctness of a PIR Protocol). A PIR protocol $\Pi = (\text{Query}, \text{Respond}, \text{Extract})$ is correct if

$$\text{Extract}(\text{Respond}(\text{qry}, \text{DB}), s) = \text{DB}[i] \quad \text{for } (\text{qry}, s) = \text{Query}(1^\lambda, i)$$

for all $i \in \{1, 2, \dots, N\}$ and all databases DB with overwhelming probability.

We consider a PIR protocol secure, if PIR queries for different indices are indistinguishable to a honest-but-curious adversary. This security notion is equivalent to the one considered in HintlessPIR [41, Def. 7] and InsPIRE [49, Appendix 1].

Definition 2.3 (Indistinguishability under chosen-query attacks (IND-CQA)). Let $\lambda \in \mathbb{N}$ be the security parameter. We define the indistinguishability game played by an adversary $\mathcal{A}$ against protocol Π in Figure 2.

IND-CQA $_{\Pi}^{\mathcal{A}}(\lambda)$

```

01  $b \xleftarrow{\$} \{0, 1\}$ 
02  $(i_0, i_1, \text{DB}) \leftarrow \mathcal{A}(1^\lambda)$ 
03  $\text{qry} \leftarrow \Pi.\text{Query}(i_b, 1^\lambda)$ 
04  $\text{resp} \leftarrow \Pi.\text{Respond}(\text{qry}, \text{DB})$ 
05  $b' \leftarrow \mathcal{A}(\text{qry}, \text{resp})$ 
06 return  $(b = b')$ 

```

Figure 2: The IND-CQA game.

The PIR protocol Π is *indistinguishable under chosen-query attacks* (IND-CQA) if, for any polynomial-time adversary $\mathcal{A}$ trying to win the above game, its advantage

$$\text{Adv}_{\Pi}^{\text{IND-CQA}}(\mathcal{A}) := \left| \Pr [\text{IND-CQA}_{\Pi}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right|$$

is negligible in λ .

3 Our Protocol

In this section, we describe the design of our PIR protocol. Before moving to the technical details, we set the scene by describing a naive RLWE-based PIR protocol, following the matrix multiplication idea established by SealPIR [4] and SimplePIR [32]. We then

present the technical details of our protocol in Section 3.2 and Section 3.3. We include a full proof of correctness and security in Appendix A and Appendix B, respectively. Finally, we discuss how to transform our index-based PIR protocol into a keyword-based protocol in Section 3.4 and database updates in Section 3.5.

3.1 A naive matrix-based protocol from RLWE

The standard approach for Homomorphic Encryption-based PIR protocols is to arrange the database entries in the form of a matrix

$$\text{DB} = \begin{pmatrix} \text{db}_{0,0} & \dots & \text{db}_{0,\sqrt{n}-1} \\ \vdots & & \vdots \\ \text{db}_{\sqrt{n}-1,0} & \dots & \text{db}_{\sqrt{n}-1,\sqrt{n}-1} \end{pmatrix} \in \mathbb{Z}_t^{\sqrt{n} \times \sqrt{n}}.$$

Based on this, it is possible to obviously select the j -th column of this matrix, by using an encryption $\text{qry} = \text{Enc}(e_j)$ of the j -th unit vector $e_j \in \mathbb{Z}_t^{\sqrt{n}}$. Due to linearity of LWE encryption, we find that

$$\text{DB} \cdot \text{qry} = \text{Enc}(\text{DB} \cdot e_j)$$

is indeed an encryption of the j -th column of DB.

SimplePIR [54] uses LWE-based encryption, but subsequent work relies on RLWE-based encryption, which can significantly reduce the number of required ciphertexts. The reason is that, under RLWE-based encryption, the plaintext space is $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N + 1)$, thus can store up to N elements from $\mathbb{Z}_t$. The most common approach [4, 32, 41] is to choose parameters that yield a slot decomposition $\mathcal{R}_t \cong \mathbb{Z}_t^N$, and then store one database element in each slot. If $N = \sqrt{n}$, we can then, for each row, pack the row's entries into a single element of $\mathcal{R}_t$. Similarly, we can encrypt the coefficients of the unit vector e_j within the slots of a single RLWE ciphertext. Formally, this means the database is of the form

$$\tilde{\text{DB}} = \begin{pmatrix} \tilde{\text{db}}_0 \\ \vdots \\ \tilde{\text{db}}_{N-1} \end{pmatrix} \in \mathcal{R}_t^N$$

and the PIR query qry sent by the client is a ciphertext packing the j -th unit vector e_j . When using this packing approach, we require homomorphic rotations HomRot_k , which shift the value of each slot by k slots, to compute the matrix-vector product. Note that, since a homomorphic rotation is implemented using homomorphic Galois automorphisms, it requires an evaluation key evk_k . Then the server can compute the linear combination

$$\begin{aligned} \text{resp} &= \sum_{k=0}^{N-1} \text{HomRot}_k(\tilde{\text{db}}_k \cdot \text{qry}, \text{evk}_k) \\ &= \sum_{k=0}^{N-1} \text{Enc}_s(\underbrace{0, \dots, 0, \text{db}_{k,j}, 0, \dots, 0}_{(k+j)\text{-times}}) \\ &= \text{Enc}_s(\text{db}_{-j,j}, \text{db}_{1-j,j}, \dots, \text{db}_{N-1-j,j}), \end{aligned}$$

which is a ciphertext that packs (a permutation of) the coefficients of the j -th column. Note here that the indices in $\text{db}_{\cdot,\cdot}$ should be understood modulo N .

Modulo some very important computation optimizations, this is the general approach taken by NTTlessPIR [41].

The main inefficiencies of this simple protocol are the following.

- To obtain a decomposition into slots that allows rotations, we require $t \equiv 1 \pmod{2N}$, so in particular $t > 2N$. However, a large t will cause significant noise growth during homomorphic operations, and thus lead to larger and less performant parameters.
- Evaluating these formulas as given requires computing $N - 1$ rotations, hence the server needs $N - 1$ rotation keys. However, this would incur prohibitively large communication costs.
- A single ciphertext has size $O(N)$, thus choosing $N = \sqrt{n}$ means the communication cost is of order $O(\sqrt{n})$. This is not desirable, and multiple techniques to avoid this are known in the literature. For example, [41] proposes TensorPIR to solve this, but it introduces significant overhead, which we would like to reduce.

In the next sections we will therefore study how this naive construction can be optimised.

3.2 Our base-level protocol

In this section, we present a “base protocol” that solves the first two issues. We describe the motivation and technical correctness, and the resulting algorithm is shown in Figure 3. To address the last point, we then show in Section 3.3 how to use this base protocol recursively.

Recall that σ_g is the Galois automorphism such that $\sigma_g(X) = X^g$, in particular σ_1 is the identity. The basic idea is to compute the linear combination

$$\sum_{i=0}^{N/2-1} c_i \sigma_{5^i}(\alpha)$$

of an (encrypted) ring element $\alpha \in \mathcal{R}_t$ as given by the following lemma.

LEMMA 3.1 (LINEAR COMBINATION OF GALOIS CONJUGATES). *Let $\alpha, c_0, \dots, c_{N/2-1} \in \mathcal{R}_t$. Define the recursive sequence*

$$\beta_0 := 0, \quad \beta_{i+1} := \sigma_5(\beta_i) + \sigma_{5^{N/2-i-1}}^{-1}(c_{N/2-i-1})\alpha \quad (1)$$

Then

$$\beta_{N/2} = \sum_{i=0}^{N/2-1} c_i \sigma_{5^i}(\alpha) \quad (2)$$

PROOF. We show

$$\beta_k = \sum_{i=0}^{k-1} \sigma_{5^{k-N/2}}(c_{N/2-i-1}) \sigma_{5^{k-i-1}}(\alpha)$$

by induction on k . The base case $k = 0$ is clear, hence assume $k > 0$. We find

$$\begin{aligned} \beta_{k+1} &= \sigma_5 \left(\sum_{i=0}^{k-1} \sigma_{5^{k-N/2}}(c_{N/2-i-1}) \sigma_{5^{k-i-1}}(\alpha) \right) \\ &\quad + \sigma_{5^{N/2-k-1}}^{-1}(c_{N/2-k-1})\alpha \\ &= \sum_{i=0}^{k-1} \sigma_{5^{(k+1)-N/2}}(c_{N/2-i-1}) \sigma_{5^{(k+1)-i-1}}(\alpha) \\ &\quad + \sigma_{5^{(k+1)-N/2}}(c_{N/2-k-1})\alpha \end{aligned}$$

$$= \sum_{i=0}^k \sigma_{5(k+1)-N/2} (c_{N/2-i-1}) \sigma_{5(k+1)-i-1} (\alpha)$$

In particular, for $k = N/2$, we have

$$\beta_{N/2} = \sum_{i=0}^{N/2-1} c_{N/2-i-1} \sigma_{5N/2-i-1} (\alpha) = \sum_{i=0}^{N/2-1} c_i \sigma_{5i} (\alpha) \quad \square$$

This only requires a single Galois key during the online phase. Indeed, since the values $c_{N/2-i-1}$ are available as plaintexts, we can compute their Galois conjugates without a corresponding Galois key. Furthermore, they depend only on the database, and can be precomputed. Hence, evaluating Equation (1) on an encryption of α requires only $N - 1$ homomorphic applications of σ_5 .

One small issue remains, however. Since 5 only generates an index-2 subgroup of $(\mathbb{Z}/2N\mathbb{Z})^*$, it is necessary for the client to provide encryptions of both α and $\sigma_{-1}(\alpha)$ so that the server can homomorphically generate all Galois conjugates.

While similar methods were used previously [41], they always required the plaintext modulus t to be NTT-friendly, i.e., $t \equiv 1 \pmod{2N}$. Hence, we propose to perform matrix multiplication on coefficient-embedded vectors. Our next lemma is based on the known theory of performing general matrix multiplication using Galois structure [28, 31]: any $\mathbb{Z}_t$ -linear transform $\mathcal{R}_t \rightarrow \mathcal{R}_t$ can be expressed as a $\mathcal{R}_t$ -linear combination of Galois automorphisms. When the linear transform is given as a matrix w.r.t. a specified basis, the involved coefficients are elegantly described in terms of the dual basis [60, Lemma 4.2]. In a sense, our next result is a specialization of that lemma: we observe that when using the power basis, this expression involving the dual basis can be re-written as an NTT. This formulation avoids solving a linear system in the preprocessing phase, and thus makes the approach computationally feasible for the database sizes N we consider. We also remark that we do not use the baby-step-giant-step optimizations of [31], since these reduce the number of used Galois automorphisms at the cost of requiring more Galois keys, but due to our use of the NTTless paradigm [41], Galois automorphisms are cheap to evaluate in our setting.

LEMMA 3.2 (HOMOMORPHIC MATMUL FOR GENERAL MODULI). *Let $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N + 1)$, with N a power of two and t any integer ≥ 2 . Let $a_0(X), \dots, a_{N-1}(X) \in \mathcal{R}_t$ and set*

$$\hat{a}_i(X) = \sum_{k=0}^{N-1} X^{-ki} a_k(X) \in \mathcal{R}_t$$

for $i \in (\mathbb{Z}/2N\mathbb{Z})^*$. Then for all $0 \leq l < N$ we find

$$\sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} \hat{a}_i(X) \sigma_i(X^l) = N a_l(X) \in \mathcal{R}_t \quad (3)$$

PROOF. Plugging the first equation into the second, we get

$$\begin{aligned} & \sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} \left(\sum_{k=0}^{N-1} X^{-ki} a_k(X) \right) \sigma_i(X^l) \\ &= \sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} \sum_{k=0}^{N-1} \sigma_i(X^l) X^{-ki} a_k(X) \end{aligned}$$

$$= \sum_{k=0}^{N-1} a_k(X) \sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} X^{li} X^{-ki}$$

Now observe that

$$\sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} X^{i(l-k)} = \begin{cases} N & \text{if } l = k \\ \sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} X^i = 0 & \text{if } l \neq k \end{cases}$$

Together, we find

$$\sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} \hat{a}_i(X) \sigma_i(X^l) = N a_l(X) \quad \square$$

This can be seen as a matrix multiplication algorithm, where X^l takes the role of the l -th unit vector. More concretely, for a matrix $A \in \mathbb{Z}_t^{N \times N}$ and a vector $b \in \mathbb{Z}_t^N$, define

$$\begin{aligned} a_i(X) &:= \sum_{j=0}^{N-1} A_{ji} X^j, & b(X) &:= \sum_{j=0}^{N-1} b_j X^j, \\ c(X) &:= \sum_{j=0}^{N-1} c_j X^j, & \text{where } c &= Ab. \end{aligned}$$

Since Galois automorphisms are linear, Equation (3) implies that

$$\sum_{i \in (\mathbb{Z}/2N\mathbb{Z})^*} \hat{a}_i(X) \sigma_i(b(X)) = N c(X)$$

hence the sum computes the matrix-vector multiplication $c = Ab$.

This removes all restrictions on the plaintext modulus t , except that it has to be odd (otherwise, the factor N introduced in Equation (3) will annihilate information). We do require the server to precompute the $\hat{a}_k$, which is relatively expensive, but does not depend on any data received from clients, and thus can be done offline just once. With this method, the result of the matrix multiplication will be packed into the coefficients instead of the slots of the ring element.

3.3 Recursively using the base-level protocol

Next, we discuss how the communication cost can be decreased below $O(\sqrt{n})$. The basic idea, as done for example in [4, 54] and many others, is to divide the database into chunks of size N^2 , use the above strategy on each of the chunks, and then privately select one of the results. The literature provides two approaches for this private selection, pioneered by SealPIR [4] and OnionPIR [54], respectively. SealPIR just arranges the individual results into a new database, and recursively performs the base PIR protocol on this new database. This approach is conceptually simple, but has some drawbacks. In particular, the base-level PIR protocol takes a database whose values are plaintexts in $\mathbb{Z}_t$ (respectively $\mathcal{R}_t$), but the results are ciphertexts $\mathcal{R}_q^2$. Thus, to apply the same protocol on a database that is now composed of ciphertexts instead of plaintexts, a ciphertext has to be “split” into F plaintexts before applying the PIR protocol, where F is the ciphertext expansion factor. The other option is the approach first proposed by [54], which is based on GSW-style external products. Most recent works prefer this method, as it avoids the $F \times$ blow-up of the response size. However, it requires the query to consist of GSW ciphertexts, which are significantly larger than BFV ciphertexts for the same parameters.

QueryBase(i)	PreprocessBase(DB)	RespondBase($qry, qry', \tilde{DB}, evk$)
01 $s := \text{KeyGen}(1^\lambda)$	01 for $i \in (\mathbb{Z}/2N\mathbb{Z})^*$	01 $resp_{N/2} := 0$
02 Generate evaluation key evk	02 $\tilde{DB}[i] := \sum_{j=0}^{N-1} DB[j]X^{ij}$	02 $resp'_{N/2} := 0$
03 $qry := \text{Enc}(X^i, s)$	03 for $0 \leq i < \frac{N}{2}$	03 for $i = N/2 - 1$ to 0
04 $qry' := \text{Enc}(X^{-i}, s)$	04 $\tilde{DB}[5^i] := \sigma_{5^i}^{-1}(\tilde{DB}[5^i])$	04 $resp_i := \text{Auto}(resp_{i+1}, 5, evk) + \tilde{DB}[5^i] \cdot qry$
05 return (qry, qry', evk, s)	05 $\tilde{DB}[-5^i] := \sigma_{5^i}^{-1}(\tilde{DB}[-5^i])$	05 $resp'_i := \text{Auto}(resp'_{i+1}, 5, evk) + \tilde{DB}[5^{-i}] \cdot qry'$
ExtractBase($resp, s$)	06 return $\tilde{DB}$	06 return $resp_0 + resp'_0$
01 return $N^{-1}\text{Dec}(resp, s)$		

Figure 3: Algorithms for the base-level protocol from Section 3.2.

Preprocess(DB)	Query(i, j)
01 for $0 \leq i < N$	01 (qry_1, qry'_1, evk, s) := QueryBase(i)
02 $\tilde{DB}_i := \text{PreprocessBase}(DB[iN], \dots, DB[iN + N - 1])$	02 $qry_2 := \text{Enc}(X^j, s)$
03 return $\tilde{DB}_0, \dots, \tilde{DB}_{N-1}$	03 $qry'_2 := \text{Enc}(X^{-j}, s)$
Respond($qry_1, qry'_1, qry_2, qry'_2, \tilde{DB}, evk$)	04 return ($qry_1, qry'_1, qry_2, qry'_2, evk, s$)
01 for $0 \leq i < N$	Extract($(resp'_0, \dots, resp'_{F-1}), s$)
02 $resp_i := \text{RespondBase}(qry_1, qry'_1, \tilde{DB}_i, evk)$	01 for $0 \leq i < F$
03 Mod-switch $resp_i$ to smaller ctxt moduli q_a, q_b	02 $\beta_i \leftarrow \text{ExtractBase}(resp'_i, s)$
04 Encode the result into F ptxts $\beta_{i,0}, \dots, \beta_{i,F-1}$	03 Recombine $\beta_0, \dots, \beta_{F-1}$ to $resp$
05 for $0 \leq k < F$	04 return $\text{ExtractBase}(resp)$
06 $\tilde{DB}'_k := \text{PreprocessBase}(\beta_{0,k}, \dots, \beta_{N-1,k})$	
07 $resp'_k = \text{RespondBase}(qry_2, qry'_2, \tilde{DB}'_k, evk)$	
08 return ($resp'_0, \dots, resp'_{F-1}$)	

Figure 4: Algorithms for our new high-level protocol from Section 3.3.

Due to our new matrix multiplication method Lemma 3.2, and the use of sparse secrets, we are able to choose parameters that keep the ciphertext expansion factor very low, and thus opt for the ciphertext splitting approach. More concretely, for the parameters proposed in Table 3, the ciphertext expansion factor is only $F = 3$, which in our case leads to significantly smaller total communication costs than the GSW approach and the algorithms defined in Figure 4.

3.3.1 Preprocessing NTTs. As in NTTlessPIR [41], we globally fix the a -component of RLWE ciphertexts (which can be done securely, as long as the secret key is different for every PIR request). During the first stage (i.e. when the database does not depend on the client queries), this allows to precompute most of the operations required in RespondBase, in particular including all the NTTs. As a result, this dramatically speeds up the evaluation RespondBase in these cases. Since this is achieved by just using the techniques of NTTlessPIR [41], we do not go into any details here but refer the reader to the original work.

3.4 Transforming an Index-based Protocol into a Keyword-based Protocol

So far, we constructed a PIR protocol that allows the client to look up the database entry at a specific index. In the DNS use case however, the client wants to perform a so-called keyword PIR lookup, i.e.,

get the database entry associated to a certain keyword (in this case, a domain name).

Fortunately, there are standard techniques to build keyword PIR from standard PIR, usually based on hashing, as discussed in Section 1.3. Most of those, however, use multiple hash functions, which results in performing multiple queries using the underlying PIR scheme. While optimal in other settings, it is possible to avoid this overhead in the DNS use case, due to the fact that our PIR scheme natively returns database elements that are large enough to store many IP addresses. Consequently, we can use simple hashing to distribute IP addresses into buckets, and return a whole bucket as PIR response without introducing additional overhead.

For the DNS use case, this very simple idea outperforms techniques based on Cuckoo Filters or Binary Fuse Filters. The exception is the work [59], which is also based on simple hashing. Unfortunately, their algorithms do not apply to the DNS setting in a straightforward way, as their efficiency relies on the fact that keyword hashes are smaller than database values, which is not the case for IP addresses. However, it may be the case that the underlying ideas can be adjusted to be efficient in this case as well, which might result in a denser packing of IP addresses to buckets. We consider this an interesting direction for future work.

To analyse the expected maximum size of a bucket in the hashtable, we model the hash of every domain name as a uniform element in $\{1, \dots, B\}$, where B is the number of buckets. Clearly, in the best case where each bucket receives the same number of elements, we need B buckets of size n/B . In practice, we will choose buckets of size k , such that with high probability, no more than k elements are placed in the same bucket.

A good, asymptotic estimate for the size of γ has been given by [64].

COROLLARY 3.2.1. *Let n and B be positive integers with $n \in \omega(B \log B)$. Let $X_1, \dots, X_n$ be identically and independently distributed (i.i.d.) random variables on $\{1, \dots, B\}$ (these model the hashes of all stored domain names). Then for any $\epsilon > 0$, with probability $1 - o(1)$, we find that*

$$\max_{i \in [B]} \# \{j \mid X_j = i\} \leq \frac{n}{B} + \sqrt{(2 + \epsilon) \frac{n}{B} \ln B}$$

PROOF. Follows directly from [64, Thm. 1]. $\square$

While this is an asymptotic statement, it has been reported to give very good estimates for concrete values [34]. We include some additional discussion in Appendix C.

For $n = 2^{31}$ (the estimated number of domain names) and $B = 2^{22}$ (as for the parameters in Section 4), we find that $\gamma = 0.244$. Hence, a bucket must store $\lceil (1 + \gamma)2^{29} \rceil \leq 637$ IP addresses, which is possible in 2560 bytes.

3.5 Updatability

With our scheme, to update a single entry, it is sufficient to only update the corresponding first-stage database $\text{DB}[iN], \dots, \text{DB}[iN + N - 1]$. This can be done in less than a second. However, in DNS, changes to the database occur frequently. Therefore, it is a better approach to re-preprocess the entire database regularly, including all updates that occurred since the last time.

This implies that DNS updates only become visible after the next re-preprocessing step. For the standard use cases of DNS, this is not an issue, since newly created names are unlikely to be queried immediately. For example, the ICANN Registry Agreement [35, Spec. 10, §2] requires DNS propagation time to be less than one hour, and TLD records are often cached for up to 48 hours. However, these are worst-case bounds (with propagation times usually being in the minutes), and some use cases of DNS require significantly lower propagation times, primarily when proving ownership of a domain via DNS, e.g., for certificate issuance, or as a client to SaaS platforms. Fortunately, the DNS zones involved in such use cases are usually much smaller than 2GB, and the preprocessing time of our protocol is therefore much lower (cf. Table 5).

Therefore, while the preprocessing times of our protocol are relatively high compared to previous work, we nevertheless believe them to be practical for the majority of private DNS use cases. Instead, we consider query latency, server operation cost, and particularly client-server communication to be more critical bottlenecks.

4 Implementation and Experiments

We implement the protocol as described in Section 3 in Rust. Our code is publicly available at <https://github.com/FeanorTheElf/dnspsir>. We report the benchmark results of the PIR protocol in isolation in Section 4.4 and of the end-to-end DNS resolution in Section 4.5.

4.1 Optimizations and Multithreading

We want to highlight two techniques that we use to enable practical running times.

Modular arithmetic. Firstly, we use the NTTless-paradigm of [41] to completely avoid the computation of NTTs during the main loop in `RespondBase` (cf. Figure 3). Since this loop computes essentially a matrix multiplication, all modular reductions can be deferred until the end, which is the same strategy as used by [41, 49]. However, modular reductions are still required, in particular for the single `PreprocessBase` call during `Respond`. To perform these as efficiently as possible, we choose the ciphertext modulus $q = p_1 p_2$ for the NTT-friendly primes $p_1 = 2^{26} - 2^{12} + 1$ and $p_2 = 2^{25} - 2^{12} + 1$. These particular primes allow us to compute the modular reduction x_{reduced} of $x \leq 4p_1^2$ using only bitshifts and $32 \times 32 \rightarrow 64$ bit multiplications, as $x' := x - (x \gg 26)p_1$ and $x_{\text{reduced}} := x' - (x' \gg 26)p_1$. This technique can be seen as a special and more efficient case of [43].

Multithreading. In order to reduce online computation time, our implementation utilizes multithreading. This is not the case for the open-source implementations of HintlessPIR [41], YPIR [52] or InspiRe [49], even though the paper of [41] reports multithreaded timings. Nevertheless, since all of these schemes are based on matrix-vector multiplication, there are no intrinsic issues with parallelizing them. We remark, however, that when using multithreading, our protocol (and most forms of matrix multiplication in general) become memory bound, and the runtime will primarily scale with the speed of RAM and the size of the preprocessed database.

4.2 Parameter Choices

We choose parameters for our protocol to accommodate the required database sizes as shown in Table 3. Hence, to benchmark our protocol with databases of sizes $2^{29} \times 4B$ and $2^{31} \times 4B$ as required for Scenarios I and II (cf. Sections 4.3.1 and 4.3.2), we arrange them as databases of sizes $2^{20} \times 2560B$ and $2^{22} \times 2560B$, respectively. The “compressed” ciphertext moduli $q_{\text{evk}}, q_{\text{qry}}, q_a, q_b$ are used to modulus-switch the query and reply to smaller ciphertext moduli before transmission (they are modulus-switch back to q at the receiver’s side). Using the lattice estimator [2], we find that these parameters provide 128 bits of security. The number of database entries can be configured by varying the number of executions of the base-level protocol, i.e., execute the loop in `Respond()` in Figure 4 possibly less than N times. Note that this affects server computation time, but not communication bandwidth.

4.3 Benchmark Design

We base our benchmarks on estimates of the size and structure of DNS deployments in the real world: As outlined in Section 1.2, we estimate the global number of publicly accessible and regularly queried hostnames to be 2^{31} . Based on this, we consider the following two scenarios.

Parameter	Chosen Value
Ring dimension N	2^{11}
Plaintext modulus t	1025
Database entry size	$2^{11} \cdot 10$ bits
Ciphertext modulus size $\log_2(q)$	≈ 51.0
Compressed ciphertext moduli	
Evaluation key q_{evk}	2^{50}
Query q_{qry}	2^{34}
First reponse part q_b	2^{12}
Second response part q_a	2^{17}
Ciphertext expansion factor F	3
Secret key hamming weight	128
Error standard deviation σ	3.2
Security level	128
Digits during key-switching	2
Base for gadget decomposition	2^{26}

Table 3: Parameters for our protocol in the DNS lookup use case. The ciphertext expansion factor refers to the expansion factor of a “compressed” ciphertext $\in \mathcal{R}_{q_b} \times \mathcal{R}_{q_a}$.

4.3.1 Scenario I (Iterative DNS). In this scenario (cf. Section 1.1 and Figure 1b), we assume that the DNS database remains distributed between root, TLD, and many authoritative name servers and that DNS resolution is performed by clients iteratively via PIR. This has the advantage that no major changes to the DNS infrastructure are required, however, only guarantees privacy for the client if the queried servers do not collude and if the adversary is not able to monitor the network traffic of clients.

The number of DNS records for the approximately 1,500 registered TLDs in the root zone is comparatively small and can be compressed to less than 1 MB, and even be reduced to less than 100 KB when ignoring DNSSEC records (i.e., RRSIG, DS, and DNSKEY). Therefore, we assume the client can use the “trivial” PIR method of downloading the entire root zone for a local lookup. It would still be possible to slightly reduce communication overhead via PIR, even on databases as small as 100KB. However, we expect these improvements to be minimal. For example, InsPIRe [49] reports a reduction of about 30% compared to the full database download.

For the number of DNS records managed by a single TLD-level name server, we base our estimate on the zone file for the .com domain¹. The file has 384,493,438 NS entries in total, which warrants the use of our PIR protocol to query the TLD database with sub-linear communication complexity. When queried with an encrypted domain name, we assume that NS entries are returned as IP addresses (as opposed to hostnames, as currently done in DNS unless glue IPs are necessary), since this will reduce the number of DNS queries the client has to perform, and avoids the need for compressing long hostnames. Under these assumptions, based on the structure of the .com zone file, our estimate for the required database size is $2^{29} \times 4B$.

¹Accessed 08/10/25 through the The Centralized Zone Data Service (CZDS).

Protocol	Threads	Database Scenario I and II	Comm Key/Query/Resp	Time	Preproc
HintlessPIR [41]	1	$2^{28} \times 8B$ (I)	3.2 MB 1.7*/1.5	0.5s	346s
	1	$2^{30} \times 8B$ (II)	7.0 MB 5.5*/1.5	1.1s	1450s
YPIR [52]	1	$2^{31} \times 1B$ (I)	1.0 MB 1.0*/0.01	0.2s	47s
	1	$2^{33} \times 1B$ (II)	1.5 MB 1.5*/0.01	0.7s	179s
InsPIRe ⁽²⁾ [49] $\gamma = (64, 1024, 64)$	1	$2^{28} \times 8B$ (I)	190 KB 82/54/54	1.9s	169s
	1	$2^{30} \times 8B$ (II)	234 KB 71/109/54	4.0s	356s
Ours (DNSPIR)	1	$2^{20} \times 2560B$ (I)	74 KB 26/26/22	4.5s 3.2/1.2	346s
	1	$2^{22} \times 2560B$ (II)	82 KB 26/35/22	17.9s 13.1/4.8	1388s
HintlessPIR [41]	multithreading not implemented				
YPIR [52]					
InsPIRe ⁽²⁾ [49]					
Ours (DNSPIR)	8	$2^{20} \times 2560B$ (I)	74 KB 26/26/22	1.3s 0.8/0.6	47s
	8	$2^{22} \times 2560B$ (II)	82 KB 26/35/22	4.5s 3.0/1.5	182s

Table 4: Performance comparison of our PIR scheme with previous work. For our protocol, online times are additionally split between loop 1 and loop 2 in Respond (cf. Figure 4). *HintlessPIR and YPIR do not break down upload communication into keys and query.

Finally, when a client has reached the level of an authoritative name server and wants to query A/AAAA entries for a host, there are two options: (a) For authoritative name servers maintaining only a few hundred or thousands of entries, the client can again use “trivial” PIR to download the entire database for a local lookup. (b) For large name servers, the client can again use our PIR protocol for private lookup with sublinear communication complexity. Unfortunately, there are no reliable statistics available on the scale of databases maintained by authoritative name servers. While we assume the majority is in the smaller category (simply by dividing the estimated total number of host names by the number of unique name servers listed in top-level zone files), clients in the worst case can perform a PIR-based lookup using the same large-scale parameters as for TLD level.

4.3.2 Scenario II (Global DNS). In this scenario (cf. Section 1.1 and Figure 1c), we assume that a global DNS database (containing all A and AAAA entries for all hosts) is constructed and available through a single logical server (which can be replicated and distributed for redundancy and throughput). This allows the client to translate a hostname to an IP address by performing only a single PIR-based query to that database. Furthermore, there is no need for including NS entries since every existing hostname can be resolved immediately. Based on our estimate for the total number of publicly accessible hostnames, it is sufficient to have a database of size $2^{31} \times 4B$.

Protocol	Threads	Database	Comm Key/Query/Resp	Time	Preproc
InsPIRe ⁽²⁾ [49] $\gamma = (64, 1024, 64)$	1	$2^{29} \times 8B$	216 KB 81/81/54	2.1s	186s
	1	$2^{25} \times 8B$	162 KB 81/27/54	0.9s	0.09s
	1	$2^{21} \times 8B$	162 KB 81/27/54	0.9s	0.09s
Ours (DNSPIR)	1	$2^{21} \times 2560B$	74 KB 26/26/22	9.1s 6.8/2.3	699s
	1	$2^{17} \times 2560B$	65 KB 26/17/22	0.6s 0.4/0.2	42s
	1	$2^{13} \times 2560B$	65 KB 26/17/22	0.08s 0.04/0.04	0.08s

Table 5: Performance comparison of our PIR scheme with InsPIRe [49] for additional database sizes. For our protocol, online times are additionally split between loop 1 and 2 in Respond (cf. Figure 4).

4.4 PIR Benchmarks

We benchmark the implementation of our PIR protocol in both settings for the DNS use case alongside the closest competitors, namely HintlessPIR [41], YPIR [52] and InsPIRe [49].

InsPIRe comes in multiple modes, of which we use InsPIRe⁽²⁾, since it is optimized for low bandwidth, which is the same setting we optimize for. We focus our attention on databases of size 8GB, with entries that should be able to store at least a single IP address. For YPIR, which supports entries of at most one byte, we use $2^{33} \times 1B$ as the closest supported database configuration.

All benchmarks are run on a system with 32 Intel(R) Xeon(R) Gold 5318S cores and 256GB of RAM. All benchmarked implementations utilize AVX-512 instructions. The results of our benchmarks are shown in Table 4.

For the global setting (Scenario II in Table 4), we can see that our PIR protocol outperforms prior work by factors $3.0\times$ (InsPIRe) to $85.4\times$ (HintlessPIR) in terms of communication overhead at the cost of $4.5\times$ (InsPIRe) to $25.6\times$ (YPIR) more server online run-time. However, unlike these prior works, our scheme supports and has implemented multi-threading (cf. Section 4.1), which reduces this overhead to $1.1 - 6.4\times$, highlighting the favourable trade-off with significant reductions in communication overhead.

For the iterative setting (Scenario I in Table 4), we can see that the reduced database size results in about a quarter of the processing time. However, note that the results in Table 4 do not include the private lookups for the root zone and authoritative name server, which we consider in the next section.

Finally, we also benchmark InsPIRe⁽²⁾ and our protocol for a range of other database sizes (cf. Table 5). However, both protocols soon hit a baseline, at which query and response size are determined by factors that do not scale with database size, including the security parameter and the noise growth during key-switches and plaintext-ciphertext multiplications. Furthermore, for InsPIRe, the LinPIR step does not depend on the database size and dominates both preprocessing- and online time for these database sizes. This would only change if the reduced noise growth made it possible to work

Queried Domain queried nameserver zones	Root cached?	Subdomain #records	Ours (DNSPIR)		InsPIRe ⁽²⁾	
			Comm Req/Resp	Time	Comm Req/Resp	Time
www.private-dns.com	✗	4	190 KB 60/130	3.4s	322KB 160/162	1.2s
. → .com → private-dns.com	✓	4	82 KB 60/22	3.4s	214KB 160/54	1.1s
www.private-dns.com	✗	4	173 KB 43/130	1.1s	278KB 116/162	0.6s
.com → private-dns.com	✓	4	65 KB 43/22	0.9s	170KB 116/54	0.5s
www.private-dns.org	✗	11753034	255 KB 103/152	4.0s	492KB 276/216	1.5s
. → .com → private-dns.com	✓	11753034	148 KB 103/45	3.9s	384KB 276/108	1.4s
www.private-dns.org	✗	11753034	238 KB 86/152	1.6s	448KB 232/216	1.0s
. → .org → private-dns.org	✓	11753034	131 KB 86/45	1.4s	340KB 232/108	0.9s

Table 6: End-to-end performance measurements of our private DNS protocol. “Subdomain #records” refers to the number of unique domain names in the private-dns.org resp. private-dns.com zone file. All server-side computations were parallelized on 8 threads. For comparison, we also provide the expected timings when using InsPIRe⁽²⁾ to perform PIR queries in our implementation. Reported values for InsPIRe are extrapolated based on the sizes of the involved databases and performance data from Table 4, assuming that InsPIRe can achieve the same speedup through multithreading as our PIR protocol.

with a smaller ring dimension (as [49] proposed for a 384KB database), but this does not yet seem possible for the database sizes reported in Table 5. Apart from that, it may be possible to slightly decrease key and query precision (i.e., q_{evk} , q_{qry} in Table 3), but the decrease would be limited to 2-4 bits, thus impacting communication by less than 10%.

4.5 End-to-End Benchmarks

In addition to comparing our PIR protocol with previous work, we also run benchmarks for a complete DNS resolution as specified in the pseudocode in Appendix E. More concretely, we assume Scenario I, and set up three name servers for the DNS root, for the .org resp. .com TLDs, and for a fictional domain private-dns.org resp. private-dns.com. The used root zone file, and the zone files for .org and .com, are slight modifications of the real zone files², which additionally include NS and glue entries for the domains that are resolved in the benchmark. The three name servers are hosted on three different machines (with the same specifications as for Section 4.4) with the client connecting over the Internet. The results are shown in Table 6.

In total, all resolutions take less than 5 seconds and less than 300KB of total communication. Moreover, when all involved domains have only moderately large zone files (like .org, which has 29,388,168 NS entries), the total resolution time is about 1.5 seconds.

4.6 Limitations and Future Work

We acknowledge that our work is limited in several aspects, mainly through simplifications of how DNS works in practice, how DNS

²Accessed 08/10/25 through the The Centralized Zone Data Service (CZDS).

integrity is protected, and the additional crucial services it provides for web security. In the following, we discuss these aspects in more detail and hint at potential solutions that can be integrated into our PIR-based approach as part of future work.

The integrity of returned records for DNS is ensured by DNSSEC. A similar, signature-based approach could be used to add integrity assurance to our protocol, e.g., by adding a small signature to every bucket. However, this would inherit some of the issues of DNSSEC, like vulnerability to replay attacks [9]. Stronger security guarantees could be obtained by using verifiable PIR protocols such as VeriSimplePIR [22]. However, verifiable PIR schemes currently require client-dependent preprocessing or introduce unrealistic performance overheads. Thus, designing an efficient verifiable PIR protocol without client-dependent preprocessing is an open problem for future work.

A somewhat related limitation is that our evaluation focuses on queries for NS and A/AAAA records. In practice, however, many other DNS record types are widely used. For example, email authentication mechanisms such as SPF, DKIM, and DMARC rely primarily on TXT records, which can be comparatively large (e.g., for DKIM public keys). Extending our PIR protocol to efficiently support larger record payloads and a broader set of record types is left for future work, potentially by storing large TXT-like records in a separate database or using size-aware encoding.

Finally, the response time of our current DNS resolution prototype is above one second, which users generally do not perceive as an immediate response during interactive tasks such as web browsing [14]. Such latency may only be acceptable to highly privacy-conscious users who are willing to tolerate additional delays in exchange for strong privacy guarantees. Reducing the server-side computation time of our PIR protocol therefore remains an important direction for future work. Beyond protocol- and FHE-level optimizations, standard engineering techniques such as database sharding and parallelization across multiple servers (potentially using dummy queries for shards not relevant to the query) could further improve response times. In addition, we expect that a well-engineered implementation for GPUs or other hardware accelerators could result in realistic latencies.

Acknowledgments

Simon Pohmann was supported by the EPSRC and the UK Government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/S021817/1).

Ethical Considerations

This work proposes and evaluates privacy-preserving DNS lookup based on single-server PIR. As such, our work is primarily systems and cryptographic research that does not involve human-subject experiments, user studies, or the collection of real DNS queries. Instead, our experimental performance evaluation uses only publicly available DNS zone data and synthetically generated databases. All experiments for performance measurements were conducted in an isolated and controlled lab environment, not involving public DNS infrastructure.

We have also considered broader impact of private DNS on a range of stakeholders. These include: (i) Internet users whose DNS queries can reveal sensitive browsing activities; (ii) DNS operators, including operators of recursive resolvers, root and TLD nameservers, and authoritative nameservers; (iii) website and domain operators whose domains may appear in DNS datasets or be queried; (iv) researchers who may build on our protocol and code; and (v) society at large, which generally benefits from privacy but may also be affected by abuse of such privacy technologies.

For these stakeholders, we have identified three main risks and propose suitable mitigations:

- (1) First, there is the risk that the same privacy properties that protect ordinary users may also be used by malicious actors to hide parts of their network activity. This risk is not unique to our work, but as a general tension in the design of PETs, including related technologies such as encrypted DNS, anonymous communication networks, E2E-encrypted messaging, etc. However, we believe the dual-use risk for our DNSPIR is limited as it does not hide subsequent network connections, payment trails, malware behaviour, or other observable activities.
- (2) A second risk is operational: PIR queries incur significant overhead compared to regular DNS lookups. Real-world deployment could therefore increase costs for DNS operators, degrade service availability, and may lead to reduced sustainability through increased resource consumption, especially if clients use PIR for all lookups. To provide guidance and manage expectations, we report concrete server-side computation and communication costs that can help in resource planning. Moreover, we suggest to use DNSPIR only selectively for privacy-sensitive queries rather than as an immediate drop-in replacement for all DNS traffic. We also recommend that any real-world deployments include standard protection mechanisms such as rate limiting and abuse monitoring to avoid overloading DNS infrastructure.
- (3) Finally, there are implementation risks. If the cryptographic parameters for our PIR protocol are altered and not chosen correctly, security guarantees in terms of input privacy and correctness will no longer hold. We thus provide a set of default parameters for which security is proven under today's understanding of the underlying cryptographic hardness assumptions. We also stress that our prototype implementation is intended as a research artifact and should not be used for real-world deployment until it has been thoroughly audited, hardened, and undergone additional testing.

Open Science

One of the main contributions of our work is the implementation of our proposed PIR scheme in Rust. Our code is publicly available at <https://github.com/FeaorTheElf/dnsplr>. We tested the implementation with the `nightly-2026-05-14-x86_64` version of Rust, but expect any recent nightly version of Rust to work. We benchmarked the related work using the following implementations.

- HintlessPIR and NTTlessPIR with the implementation at https://github.com/google/hintless_pir (commit 4be2ae81a2a7999cd53f92a2c80f036ea3f7ab26).
- YPIR with the implementation at <https://github.com/menonsamir/ypir> (commit a73e550a469605c2fa23945a4d96c20fb2f4a839).
- InsPIRe with the implementation at <https://github.com/google/private-membership/tree/main/research/InsPIRe> (commit 55da7a7971eeaeff0edc533a81cc69c70f125ed1).

AI Use

AI-based tools (Claude) were used while conducting this research, in the following ways.

- To obtain feedback and find mistakes in the written paper. However, no part of this submission is AI-generated.
- To obtain feedback and find bugs in the core implementation of the PIR protocol. Except for some documentation, the implementation of the PIR protocol (the code in `/src/pir`) does not contain AI-generated code.
- To implement the higher-level private DNS protocol used for the end-to-end benchmarks of Section 4.5 (the code in `/src/dns`). Most of this code has been generated completely by Claude Code, only given high-level instructions and the implementation of the core PIR protocol. All code was manually reviewed and checked for correctness.

References

- [1] Dorit Aharonov, Zvika Brakerski, Kai-Min Chung, Ayal Green, Ching-Yi Lai, and Or Sattath. 2019. On Quantum Advantage in Information Theoretic Single-Server PIR. In *Advances in Cryptology – EUROCRYPT 2019, Part III* (LNCS, Vol. 11478), Yuval Ishai and Vincent Rijmen (Eds.). Springer, Cham, 219–246. doi:10.1007/978-3-030-17659-4_8
- [2] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On The Concrete Hardness Of Learning With Errors. Cryptology ePrint Archive, Report 2015/046. <https://eprint.iacr.org/2015/046>
- [3] Asra Ali, Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Philipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR. In *USENIX Security Symposium*. USENIX Association, 1811–1828.
- [4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 962–979. doi:10.1109/SP.2018.00062
- [5] Richard Arratia and Louis Gordon. 1989. Tutorial on Large Deviations for the Binomial Distribution. *Bulletin of Mathematical Biology* 51, 1 (1989), 125–131.
- [6] Daniele Enrico Asoni, Samuel Hitz, and Adrian Perrig. 2017. A Paged Domain Name System for Query Privacy. In *CANS (Lecture Notes in Computer Science, Vol. 11261)*. Springer, 250–273.
- [7] Paul Barrett. 1987. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology – CRYPTO ’86* (LNCS, Vol. 263), Andrew M. Odlyzko (Ed.). Springer, Berlin, Heidelberg, 311–323. doi:10.1007/3-540-47721-7_24
- [8] Amos Beimel, Yuval Ishai, and Tal Malkin. 2000. Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In *Advances in Cryptology – CRYPTO 2000* (LNCS, Vol. 1880), Mihir Bellare (Ed.). Springer, Berlin, Heidelberg, 55–73. doi:10.1007/3-540-44598-6_4
- [9] Daniel Bernstein. 2009. Breaking DNSSEC. 3rd Usenix Workshop on Offensive Technologies.
- [10] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *ACNS 2022: 20th International Conference on Applied Cryptography and Network Security* (LNCS, Vol. 13269), Giuseppe Ateniese and Daniele Venturi (Eds.). Springer, Cham, 521–541. doi:10.1007/978-3-031-09234-3_26
- [11] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology – CRYPTO 2012* (LNCS, Vol. 7417), Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 868–886. doi:10.1007/978-3-642-32009-5_50
- [12] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *52nd Annual Symposium on Foundations of Computer Science*, Rafail Ostrovsky (Ed.). IEEE Computer Society Press, 97–106. doi:10.1109/FOCS.2011.12
- [13] Alexander Burton, Samir Jordan Menon, and David J. Wu. 2024. Respire: High-Rate PIR for Databases with Small Records. In *CCS*. ACM, 1463–1477.
- [14] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. 1991. The information visualizer, an information workspace. In *CHI*. ACM, 181–186.
- [15] Sofia Celi and Alex Davidson. 2024. Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries. In *ACM CCS 2024: 31st Conference on Computer and Communications Security*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 4107–4121. doi:10.1145/3658644.3670271
- [16] Gowri R. Chandran, Thomas Schneider, Maximilian Stillger, and Christian Weinert. 2025. Concretely Efficient Private Set Union via Circuit-Based PSI. In *AsiaCCS*. ACM, 149–162.
- [17] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1223–1237. doi:10.1145/3243734.3243836
- [18] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *J. ACM* 45, 6 (1998), 965–981.
- [19] Inc. Cloudflare. 2018. Introducing DNS Resolver for Tor. Blog post. <https://blog.cloudflare.com/welcome-hidden-resolver/>.
- [20] Benjamin R. Curtis and Rachel Player. 2019. On the Feasibility and Impact of Standardising Sparse-secret LWE Parameter Sets for Homomorphic Encryption. In *WAHC@CCS*. ACM, 1–10.
- [21] Alex Davidson, Gonçalo Pestana, and Sofia Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proceedings on Privacy Enhancing Technologies* 2023, 1 (Jan. 2023), 365–383. doi:10.56553/popets-2023-0022
- [22] Leo de Castro and Keewoo Lee. 2024. VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers. In *USENIX Security 2024: 33rd USENIX Security Symposium*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. <https://www.usenix.org/conference/usenixsecurity24/presentation/de-castro>
- [23] Leo de Castro, Kevin Lewi, and Edward Suh. 2024. WhisPIR: Stateless Private Information Retrieval with Low Communication. Cryptology ePrint Archive, Report 2024/266. <https://eprint.iacr.org/2024/266>
- [24] M. Dempsy. 2009. DNSCurve: Link-level security for the domain name system. Internet Eng. Task Force, Fremont, CA, USA. <https://tools.ietf.org/html/draft-dempsy-dnscurve-00>
- [25] Frank Denis and Yecheng Fu. 2013. DNSCrypt Version 2 Protocol Specification. <https://dnscrypt.info/protocol/>
- [26] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. <https://eprint.iacr.org/2012/144>
- [27] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. 2021. Random-Index PIR and Applications. In *TCC (3) (Lecture Notes in Computer Science, Vol. 13044)*. Springer, 32–61.
- [28] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Fully Homomorphic Encryption with Polylog Overhead. In *Advances in Cryptology – EUROCRYPT 2012* (LNCS, Vol. 7237), David Pointcheval and Thomas Johansson (Eds.). Springer, Berlin, Heidelberg, 465–482. doi:10.1007/978-3-642-29011-4_28
- [29] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. 2024. Efficient Pre-processing PIR Without Public-Key Cryptography. In *Advances in Cryptology – EUROCRYPT 2024, Part VI* (LNCS, Vol. 14656), Marc Joye and Gregor Leander (Eds.). Springer, Cham, 210–240. doi:10.1007/978-3-031-58751-1_8
- [30] Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. 2022. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *USENIX Security Symposium*. USENIX Association, 1759–1776.
- [31] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HELIB. In *Advances in Cryptology – CRYPTO 2018, Part I* (LNCS, Vol. 10991), Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Cham, 93–120. doi:10.1007/978-3-319-96884-1_4
- [32] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023: 32nd USENIX Security Symposium*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905. <https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger>
- [33] Dominik Herrmann, Karl-Peter Fuchs, Jens Lindemann, and Hannes Federrath. 2014. EncDNS: A Lightweight Privacy-Preserving Name Resolution Service. In *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part I* (LNCS, Vol. 8712), Mirosław Kutylowski and Jaideep Vaidya (Eds.). Springer,

- Cham, 37–55. doi:10.1007/978-3-319-11203-9_3
- [34] ShreevatsaR (<https://math.stackexchange.com/users/205/shreevatsar>). [n.d.]. Throwing balls into b buckets: when does some bucket overflow size s ? Mathematics Stack Exchange. <https://math.stackexchange.com/q/172744> URL:<https://math.stackexchange.com/q/172744> (version: 2012-07-19).
- [35] Internet Corporation for Assigned Names and Numbers. 2024. Base gTLD Registry Agreement. <https://itp.cdn.icann.org/en/files/registry-agreements/base-registry-agreement-21-01-2024-en.html>.
- [36] Yuval Ishai, Elaine Shi, and Daniel Wichs. 2024. PIR with Client-Side Preprocessing: Information-Theoretic Constructions and Lower Bounds. In *Advances in Cryptology – CRYPTO 2024, Part IX (LNCS, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 148–182. doi:10.1007/978-3-031-68400-5_5
- [37] Liz Izhikevich, Gautam Akiwate, Briana Berger, Spencer Drakontaidis, Anna Ascherman, Paul Pearce, David Adrian, and Zakir Durumeric. 2022. ZDNS: a fast DNS toolkit for internet measurement. In *IMC*. ACM, 33–43.
- [38] Jiayi Kang and Leonard Schild. 2025. Pihouette: Query Efficient Single-Server PIR. Cryptology ePrint Archive, Paper 2025/680. <https://eprint.iacr.org/2025/680>
- [39] Eric Kinnear, Tommy Pauly, Tanya Verma, and Christopher A. Wood. 2022. RFC 9230: Oblivious DNS over HTTPS. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/html/rfc9230>.
- [40] Calvin Kranig, Eric Pauley, Wei-Shiang Wung, Paul Barford, Mark Crovella, and Joel Sommers. 2025. Toward a Representative DNS Data Corpus: A Longitudinal Comparison of Collection Methods. In *TMA*. IEEE, 1–11.
- [41] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. 2024. Hintless Single-Server Private Information Retrieval. In *Advances in Cryptology – CRYPTO 2024, Part IX (LNCS, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 183–217. doi:10.1007/978-3-031-68400-5_6
- [42] Hsuan-Yin Lin, Siddhartha Kumar, Eirik Rosnes, Alexandre Graell i Amat, and Eitan Yaakobi. 2022. Multi-Server Weakly-Private Information Retrieval. *IEEE Trans. Inf. Theory* 68, 2 (2022), 1197–1219.
- [43] Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *CANS 16: 15th International Conference on Cryptology and Network Security (LNCS, Vol. 10052)*, Sara Foresti and Giuseppe Persiano (Eds.). Springer, Cham, 124–139. doi:10.1007/978-3-319-48965-0_8
- [44] Ming Luo, Feng-Hao Liu, and Han Wang. 2024. Faster FHE-Based Single-Server Private Information Retrieval. In *ACM CCS 2024: 31st Conference on Computer and Communications Security*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 1405–1419. doi:10.1145/3658644.3690233
- [45] Yingchu Lv, Yanbin Pan, and Huaxiong Wang. 2026. Improved TensorPIR: Single-Server PIR with Lower Communication Cost. Cryptology ePrint Archive, Paper 2026/912. <https://eprint.iacr.org/2026/912>
- [46] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology – EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Berlin, Heidelberg, 1–23. doi:10.1007/978-3-642-13190-5_1
- [47] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. 2022. Incremental Offline/Online PIR. In *USENIX Security 2022: 31st USENIX Security Symposium*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1741–1758. <https://www.usenix.org/conference/usenixsecurity22/presentation/ma>
- [48] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *USENIX Security 2022: 31st USENIX Security Symposium*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1723–1740. <https://www.usenix.org/conference/usenixsecurity22/presentation/mahdavi>
- [49] Rasoul Akhavan Mahdavi, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2026. InsPIRE: Communication-Efficient PIR with Server-side Preprocessing. In *2026 IEEE Symposium on Security and Privacy (SP)*.
- [50] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 155–174.
- [51] Samir Jordan Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 930–947. doi:10.1109/SP46214.2022.9833700
- [52] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. In *USENIX Security 2024: 33rd USENIX Security Symposium*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. <https://www.usenix.org/conference/usenixsecurity24/presentation/menon>
- [53] Peter L. Montgomery. 1985. Modular Multiplication without Trial Division. *Math. Comp.* 44, 170 (1985), 519–521.
- [54] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *ACM CCS 2021: 28th Conference on Computer and Communications Security*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2292–2306. doi:10.1145/3460120.3485381
- [55] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized Batch Private Information Retrieval. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 437–452. doi:10.1109/SP46215.2023.10179392
- [56] Lea Nürnberg, Simon Pohmann, Mattia Veroni, and Christian Weinert. 2027. DNSPIR: Private Information Retrieval Optimized for Privacy-Preserving DNS Lookups. *Proc. Priv. Enhancing Technol.* 2027, 1 (2027).
- [57] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. 2024. Towards Practical Doubly-Efficient Private Information Retrieval. In *FC 2024: 28th International Conference on Financial Cryptography and Data Security, Part II (LNCS, Vol. 14745)*, Jeremy Clark and Elaine Shi (Eds.). Springer, Cham, 264–282. doi:10.1007/978-3-031-78679-2_14
- [58] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *J. Algorithms* 51, 2 (2004), 122–144.
- [59] Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Don’t be Dense: Efficient Keyword PIR for Sparse Databases. In *USENIX Security Symposium*. USENIX Association, 3853–3870.
- [60] Chris Peikert and Zachary Pepin. 2025. Vive Galois! Part 1: Optimal SIMD Packing and Packed Bootstrapping for FHE. In *TCC (1) (Lecture Notes in Computer Science)*. Springer, 185–219.
- [61] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT (3) (Lecture Notes in Computer Science, Vol. 11478)*. Springer, 122–153.
- [62] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT (3) (Lecture Notes in Computer Science, Vol. 10822)*. Springer, 125–157.
- [63] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.* 21, 2 (2018), 7:1–7:35.
- [64] Martin Raab and Angelika Steger. 1998. “Balls into Bins” - A Simple and Tight Analysis. In *RANDOM (Lecture Notes in Computer Science, Vol. 1518)*. Springer, 159–170.
- [65] Rapid7. 2013. Project Sonar – Gaining insights into global exposure. <https://www.rapid7.com/research/project-sonar/>
- [66] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th Annual ACM Symposium on Theory of Computing*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. doi:10.1145/1060590.1060603
- [67] Giovanni Schmid. 2021. Thirty Years of DNS Insecurity: Current Issues and Perspectives. *IEEE Commun. Surv. Tutorials* 23, 4 (2021), 2429–2459.
- [68] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: practical privacy for DNS queries: published in PoPETS 2019. In *ANRW*. ACM, 17–19.
- [69] Kyle Schomp, Mark Allman, and Michael Rabinovich. 2014. DNS Resolvers Considered Harmful. In *HotNets*. ACM, 16:1–16:7.
- [70] Jaspal Singh, Yu Wei, and Vassilis Zikas. 2024. Information-Theoretic Multi-server Private Information Retrieval with Client Preprocessing. In *TCC 2024: 22nd Theory of Cryptography Conference, Part IV (LNCS, Vol. 15367)*, Elette Boyle and Mohammad Mahmoody (Eds.). Springer, Cham, 423–450. doi:10.1007/978-3-031-78023-3_14
- [71] International Telecommunication Union. 2025. ICT Price Baskets: Country-Level Data, 2008–2025. <https://www.itu.int/en/ITU-D/Statistics/Pages/ICTPrices/default.aspx>
- [72] Zihan Wang, Lutan Zhao, Ming Luo, Zhiwei Wang, Haoqi He, Wenzhe Lv, Xuan Ding, Dan Meng, and Rui Hou. 2025. ShiftPIR: An Efficient PIR System with Gravity Shifting from Client to Server. In *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, Chun-Ying Huang, Jyh-Cheng Chen, Shih-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 1143–1157. doi:10.1145/3719027.3765153
- [73] Yunming Xiao, Peizhi Liu, Ruijie Yu, Chenkai Weng, Matteo Varvello, and Aleksandar Kuzmanovic. 2025. Collusion Resistant DNS With Private Information Retrieval. *CoRR* abs/2507.20806 (2025).
- [74] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2024. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4296–4314. doi:10.1109/SP54263.2024.00055

A Correctness

The core parts of our algorithm follow Lemma 3.1 and Lemma 3.2. Based on this, we can conclude the correctness of our scheme.

THEOREM A.1. *Let DB be a database storing N^2 entries from $\mathcal{R}_l$, and let $kN + l$ be an index into the database, with $0 \leq k, l < N$. Let*

$$(\text{qry}_1, \text{qry}'_1, \text{qry}'_2, \text{evk}), s \leftarrow \text{Query}(k, l)$$

be a PIR query, computed as in Figure 4. Then, we have that

$$\text{Extract}(\text{Respond}(\text{qry}_1, \text{qry}'_1, \text{qry}'_2, \text{evk}), s)$$

is equal to $\text{DB}[iN + k]$ with overwhelming probability. In other words, the protocol as given in Figure 4 is correct.

We analyse the noise growth of our protocol in Appendix F.

PROOF. First, we show the correctness of RespondBase (Figure 3). More concretely, we argue that for encryptions $\text{qry}_1, \text{qry}'_1$ of X^l resp. X^{-l} , the response $\text{resp} = \text{RespondBase}(\text{qry}_1, \text{qry}'_1, \hat{\text{DB}}_i, \text{evk})$ is indeed an encryption of $N \cdot \text{DB}[iN + l]$.

By Lemma 3.1, instantiated with $\alpha = X^l$ and $c_j = \hat{\text{DB}}_i[5^j]$, observe that once we reach line 6 of RespondBase in Figure 3, it holds that resp_0 is an encryption of

$$\sum_{j=0}^{N/2-1} \hat{\text{DB}}_i[5^j] \sigma_{5^j}(X^l)$$

Similarly, by Lemma 3.1 with $\alpha = X^{-l}$ and $c_j = \hat{\text{DB}}_i[-5^j]$, we find that resp'_0 is an encryption of

$$\sum_{j=0}^{N/2-1} \hat{\text{DB}}_i[-5^j] \sigma_{5^j}(X^{-l})$$

In particular, we find $\text{resp}_0 + \text{resp}'_0$ is an encryption of

$$\begin{aligned} & \sum_{j=0}^{N/2-1} \hat{\text{DB}}_i[5^j] \sigma_{5^j}(X^l) + \hat{\text{DB}}_i[-5^j] \sigma_{-5^j}(X^l) \\ &= \sum_{j \in (\mathbb{Z}/2N\mathbb{Z})^*} \hat{\text{DB}}_i[j] \sigma_j(X^l) \end{aligned}$$

Now Lemma 3.2 yields that this, indeed, is equal to $N \cdot \text{DB}[iN + l]$.

In particular, this means that the resp_i in line 2 of Respond (Figure 4) are indeed encryptions of $N \cdot \text{DB}[iN + l]$. Furthermore, we find that the resp'_j in line 6 of Respond (Figure 4) are encryptions of $N \cdot \beta_{k,j}$, respectively. Thus, when the client calls ExtractBase on the resp'_j , they successfully retrieve the $\beta_{k,j}$. Recombining them thus gives resp_k , which is an encryption of $N \cdot \text{DB}[kN + l]$. Calling ExtractBase on resp_k then returns $\text{DB}[kN + l]$, as claimed. $\square$

B Security

In this section we prove the security of our protocol. For this, we use the following definition for the IND-CPA security of the (public-key) BFV scheme.

Definition B.1 (Indistinguishability under chosen-plaintext attacks (IND-CPA)). Let $\lambda \in \mathbb{N}$ be the security parameter. We define the indistinguishability game played by an adversary $\mathcal{A}$ against a public-key encryption scheme Π .

$\text{IND-CPA}_{\Pi}^{\mathcal{A}}(\lambda)$
01 $b \xleftarrow{\$} \{0, 1\}$
02 $(\text{sk}, \text{pk}) \leftarrow \Pi.\text{KeyGen}(1^\lambda)$
03 Define the oracle $\mathcal{O} : (m_0, m_1) \mapsto \Pi.\text{Enc}(m_b, \text{sk})$
04 $b' \leftarrow \mathcal{A}(\text{pk}, \mathcal{O})$
05 return $(b = b')$

Figure 5: The IND-CPA game.

If Π is a Homomorphic Encryption scheme, we additionally consider the evaluation key evk to be part of the public key pk .

A public-key encryption scheme Π is *indistinguishable under chosen-plaintext attacks* (IND-CPA) if, for any polynomial-time adversary $\mathcal{A}$ trying to win the above game, its advantage

$$\text{Adv}_{\Pi}^{\text{IND-CPA}}(\mathcal{A}) := \left| \Pr [\text{IND-CPA}_{\Pi}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right|$$

is negligible in λ .

We have the following theorem.

THEOREM B.1. Let π be the protocol as described in Section 3. Let $\mathcal{A}$ be an efficient adversary against the IND-CQA game for π . Then there exists an efficient adversary $\mathcal{B}$ against the IND-CPA security of BFV such that

$$\text{Adv}_{\pi}^{\text{IND-CQA}}(\mathcal{A}) \leq \text{Adv}_{\text{BFV}}^{\text{IND-CPA}}(\mathcal{B}).$$

In particular, if the underlying instantiation of BFV is IND-CPA secure, then π is IND-CQA secure.

Together with the security proof of BFV [26], this implies that our scheme is secure under the RLWE assumption with circular security.

PROOF. Given the adversary $\mathcal{A}$ against the IND-CQA security of our protocol, construct an adversary $\mathcal{B}$ against the IND-CPA security of BFV as follows.

- Take $\text{pk}, \mathcal{O}$ as input.
- Extract a Galois key evk for the automorphism σ_5 from the public key pk .
- Call $\mathcal{A}$ to get two query indices $(i_0, j_0), (i_1, j_1)$ and a database DB .
- Call the encryption oracle $\mathcal{O}$ on $(X^{i_0}, X^{i_1}), (X^{-i_0}, X^{-i_1}), (X^{j_0}, X^{j_1}), (X^{-j_0}, X^{-j_1})$ to obtain encryptions $\text{Enc}(X^{i_b}), \text{Enc}(X^{-i_b}), \text{Enc}(X^{j_b}), \text{Enc}(X^{-j_b})$.
- Set $\text{qry} = (\text{Enc}(X^{i_b}), \text{Enc}(X^{-i_b}), \text{Enc}(X^{j_b}), \text{Enc}(X^{-j_b}))$.
- Compute $\text{resp} := \text{Respond}(\text{qry}, \text{Preprocess}(\text{DB}), \text{evk})$.
- Compute $b' := \mathcal{A}((\text{qry}, \text{evk}), \text{resp})$.
- Return b' .

To prove the theorem, it now suffices to show

$$\text{Adv}_{\pi}^{\text{IND-CQA}}(\mathcal{A}) = \text{Adv}_{\text{BFV}}^{\text{IND-CPA}}(\mathcal{B}).$$

To this end, we claim it suffices to show that qry, resp is distributed exactly like an honest PIR query and corresponding response for index (i_b, j_b) under protocol π , evaluation key evk and database DB . Indeed, by definition of IND-CQA security (cf. Definition 2.3), once shown, this would imply that

$$\begin{aligned} \text{Adv}_{\pi}^{\text{IND-CQA}}(\mathcal{A}) &= \left| \Pr [\text{IND-CQA}_{\Pi}^{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| \\ &= \left| \Pr [b' = b] - \frac{1}{2} \right| \\ &= \left| \Pr [b = b'] - \frac{1}{2} \right| \\ &= \text{Adv}_{\text{BFV}}^{\text{IND-CPA}}(\mathcal{B}). \end{aligned}$$

To see now that the distributions match as claimed, consider the distribution of $(\text{qry}', \text{evk}', \text{sk}') := \text{Query}(i_b, j_b)$ (where Query is the algorithm in Figure 4). Then, for all x , we have

$$\begin{aligned}
& \Pr[\text{qry}' = x \mid \text{evk}' = \text{evk}, \text{sk}' = \text{sk}] \\
&= \Pr[\text{BFV}.\text{Enc}(X^{i_b}, \text{sk}) = x_0 \wedge \text{BFV}.\text{Enc}(X^{-i_b}, \text{sk}) = x_1 \\
&\quad \wedge \text{BFV}.\text{Enc}(X^{j_b}, \text{sk}) = x_2 \wedge \text{BFV}.\text{Enc}(X^{-j_b}, \text{sk}) = x_3 \\
&\quad \mid \text{evk}' = \text{evk}, \text{sk}' = \text{sk}] \\
&= \Pr[\text{BFV}.\text{Enc}(X^{i_b}, \text{sk}) = x_0 \wedge \text{BFV}.\text{Enc}(X^{-i_b}, \text{sk}) = x_1 \\
&\quad \wedge \text{BFV}.\text{Enc}(X^{j_b}, \text{sk}) = x_2 \wedge \text{BFV}.\text{Enc}(X^{-j_b}, \text{sk}) = x_3] \\
&= \Pr[\text{qry} = x]
\end{aligned}$$

Consequently, $(\text{qry}, \text{evk}, \text{sk})$ is distributed identically to the output of Query . Moreover, resp is distributed identically to $\text{Respond}(\text{qry}, \text{DB}, \text{evk})$, by definition of $\mathcal{B}$. This shows that qry, resp is distributed identically to an honest PIR query-response pair, completing the proof. $\square$

C Estimates for Bucket size of Simple Hashing

In this section, we give concrete evidence for our estimate of the size of buckets required for simple hashing. More concretely, based on Corollary 3.2.1, we argue that if n elements are distributed uniformly on B buckets, then the maximal size of buckets can be estimated as $(1 + \gamma) \frac{n}{B}$ for $\gamma = \sqrt{2B \ln B/n}$ for the size of n and B that is relevant to our work.

More concretely, for $n = 2^{31}$ and $B = 2^{22}$, we generate uniformly random values $X_1, \dots, X_n$ in $\{1, \dots, B\}$ and compute the largest size of a bucket $\max_k \# \{j \mid X_j = k\}$. As shown in Figure 6, the maximal bucket size for these parameters is distributed between 620 and 660. Therefore, the estimate

$$\frac{n}{B} (1 + \sqrt{2B \ln B/n}) \approx 637.0$$

is indeed very precise.

With this bound, there is still a small probability that the hashing fails, because one bucket is too large. In such a case, the server can re-run the operation with a different hash function seed. Note that unlike for private set intersection protocols, where rehashing would leak information about sensitive inputs and thus is a major concern, this is not the case for PIR where the database is public.

C.1 Worst-case bounds on failure probability

Alternatively, it is also possible to avoid rehashing by choosing slightly larger parameters, which make the probability of exceeding the size of any bucket extremely small.

LEMMA C.1. *Let n, k and B be positive integers. Let $X_1, \dots, X_n$ be identically and independently distributed (i.i.d.) random variables on $\{1, \dots, B\}$ (these model the hashes of all stored domain names). Then*

$$\Pr[\exists i : \# \{j \mid X_j = i\} > k] \leq B \exp\left(-nD\left(\frac{k}{n} \parallel \frac{1}{B}\right)\right)$$

where $D(a \parallel p) := a \ln \frac{a}{p} + (1 - a) \ln \frac{1-p}{1-a}$ is the Kullback-Leibler divergence.

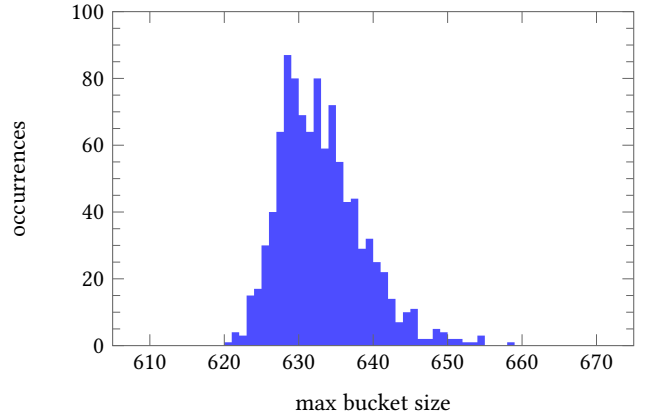


Figure 6: Distribution of maximal bucket size when hashing $n = 2^{31}$ values to $B = 2^{22}$ buckets, over 1000 trials.

PROOF. Using the union bound, we find

$$\Pr[\exists i : \# \{j \mid X_j = i\} > k] \leq B \cdot \Pr[\# \{j \mid X_j = 1\} > k] \quad (4)$$

Clearly the random variable $\# \{j \mid X_j = 1\}$ follows a binomial distribution $\text{Binom}(n, \frac{1}{B})$, and using the standard tail bound for the binomial distribution (see e.g. [5])

$$\Pr[\# \{j \mid X_j = 1\} \geq k] \leq \exp\left(-nD\left(\frac{k}{n} \parallel \frac{1}{B}\right)\right)$$

the result follows. $\square$

The use of the union bound in Equation (4) slightly overestimates the error probability, but apart from that, this estimate is quite tight. Setting $k = 640$, $n = 2^{31}$ and the acceptable failure probability to 2^{-40} , we find that a bucket count of $B = 19 \cdot 2^{18}$ is sufficient. In other words, we would require a database size of $19 \cdot 2^{18} \times 2560B$ instead of $2^{22} \times 2560B$, as used in Section 4. Running our protocol on 8 threads on the same system as in Section 4 results in a preprocessing time of 195s, an average server computation time of 4.9s, a query size of 60 KB and a response size of 44 KB.

Therefore, in applications where rehashing should be avoided, these parameters provide a reasonable alternative to the previous scenario with a noticeable failure probability. Nevertheless, we believe that potential rehashing does not cause issues in the DNS use case, and thus use the benchmarks from Table 4 as our main results.

D Monetary Cost Estimates for DNSPIR with Mobile Devices

We estimate the worldwide median cost for one GB of mobile data as \$1.46. Opposed to the cost of bandwidth in cloud computing, this number is much harder to determine due to large differences across countries and data plans. We base our estimate on the 2025 study of the ITU [71]: They report the cost of 1GB, 5GB, and 10GB monthly mobile data subscriptions across global economies, in \$-PPP, i.e., USD dollars normalized to *purchasing power parity*. Taking the median over the values reported for all economies, we obtain a monthly cost of \$-PPP 12.6, 18.32, and 25.74 for 1GB, 5GB, and 10GB subscriptions, respectively. These costs closely fit a linear model

	Server		Client Comm.	Total
	Computation	Comm.		
InsPIRe ⁽²⁾ [49]	7	5	265	277
Ours (DNSPIR)	18	2	103	123

Table 7: Comparison of monetary cost estimates for a PIR query with our scheme and InsPIRe. All costs given in $\$10^{-6}$.

with a fixed subscription cost of $\$$ -PPP 11.1 and an additional $\$$ -PPP 1.46 per GB of provided bandwidth.

We acknowledge that this simplified analysis has significant limitations, by not accounting for variations in population or DNS use across economies, and not considering unmetered mobile data plans. Furthermore, it varies significantly between countries. Nevertheless, we believe that our analysis is sufficient to motivate our focus on reducing communication overhead as opposed to server computation time.

Based on the above estimate, and the estimates for server-side costs of $\$0.09$ per GB of outbound traffic and $\$0.014$ per CPU hour from [59], we derive the cost of a PIR query on a 2GB database (cf. Scenario I in Table 4) for our protocol and InsPIRe⁽²⁾. The results are provided in Table 7.

E Pseudocode for Complete DNS Resolution via DNSPIR

In Figure 7, we provide the pseudocode for complete DNS resolution based on our DNSPIR protocol from Figure 4. This is used for the implementation and execution of end-to-end benchmarks as reported in Section 4.5.

F Noise Growth of Our Protocol

To analyse the noise growth, we introduce the following notation. For a ciphertext ct , we denote its first component by $ct[0]$ and its second component by $ct[1]$. For any ciphertext $ct \in \mathcal{R}_q^2$, we denote its *critical quantity* as $\gamma(ct) = ct[0] + ct[1]s$. The critical quantity is a polynomial in $\mathcal{R}$, and for the purpose of noise analysis we assume that its coefficients are identically and independently distributed according to some distribution with variance $\sigma_{\gamma(ct)}^2$. For any polynomial $a \in \mathcal{R}$, $\lfloor a \rfloor$ means the coefficient-wise rounding of a to the nearest integer. We denote by τ the incurred rounding error, that is $\lfloor a \rfloor + \tau = a$. The coefficients of τ are in the interval $(-\frac{1}{2}, \frac{1}{2}]$, and we can model them as being uniformly randomly distributed over this interval. Thus, the coefficient variance of the rounding polynomial is given by $\sigma_\tau^2 = \frac{1}{12}$.

It is common to assume that the coefficients of the critical quantity are normally distributed. Since this eases the analysis of the noise, we will make this same assumption here. It is well known that, for a univariate normally distributed random variable X with standard deviation σ ,

$$\Pr(-\alpha\sigma < X < \alpha\sigma) \leq \text{erf}\left(\frac{\alpha}{\sqrt{2}}\right),$$

for $\text{erf}(\cdot)$ the error function, and $\alpha \in \mathbb{R}$ some parameter. For a normally distributed random vector V with N elements and standard

deviation σ , we correspondingly have

$$\Pr(\|V\|_\infty > \alpha\sigma) \leq 1 - \text{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^N.$$

Thus, we can bound the magnitude of $\gamma(ct)$ by $\alpha\sigma$ with probability $1 - \text{erf}\left(\frac{\alpha}{\sqrt{2}}\right)^N$. For $N = 2^{11}$, we find that the probability of this bound not holding is $\approx 2^{-40}$, for $\alpha = 8$. Since the noise growth only impacts correctness and not security, this failure probability is acceptable. Thus, we will choose this value for α . Then, if we are able to determine the standard deviation of $\gamma(ct)$, we can bound its magnitude by $\|\gamma(ct)\|_\infty \leq 6\sigma_{\gamma(ct)}$. The remainder of this section will be dedicated to determining $\sigma_{\gamma(ct)}$.

There are three major parts of the protocol that incur noise: the modulus switching up and down during transmission from client to server, the response algorithm, and the modulus switching up and down from the server to the client. We will look at those parts separately.

F.1 Transmission from Client to Server

All ciphertexts and the evaluation key that are created by the client are secret-key ciphertexts whose randomness is drawn from a discrete normal distribution with standard deviation $\sigma_e = 3.2$. For any secret-key BFV ciphertext ct encrypting a message m , we have

$$\gamma(ct) = ct[0] + ct[1]s = m - as + e + as = m + e.$$

Since the message is deterministic and does therefore influence the mean but not the variance of the distribution of $\gamma(ct)$, we have $\sigma_{\gamma(ct)}^2 = \sigma_e^2 = 3.2^2$.

Let qry^c be any query on the client-side, and let qry^s be the same query as received by the server. The b -component of the query is first modulus-switched down to q_{qry} before transmission and mod-switched up to q by the server.

$$\text{qry}^s = \left(\left\lfloor \frac{q}{q_{\text{qry}}} \left\lfloor \frac{q_{\text{qry}}}{q} \text{qry}^c[0] \right\rfloor \right\rfloor, \text{qry}^c[1] \right).$$

Thus, we have for the critical quantity of qry^s

$$\begin{aligned} \gamma(\text{qry}^s) &= \frac{q}{q_{\text{qry}}} \left(\frac{q_{\text{qry}}}{q} \text{qry}^c[0] + \tau_1 \right) + \tau_1 \\ &= \gamma(\text{qry}^c) + \frac{q}{q_{\text{qry}}}(\tau_1) + \tau_0 \end{aligned}$$

Since the rounding errors have variance $\frac{1}{12}$, we get for the variance of $\gamma(\text{qry}^s)$

$$\sigma_{\gamma(\text{qry}^s)}^2 = \sigma_{\gamma(\text{qry}^c)}^2 + \frac{q^2}{12q_{\text{qry}}^2} \approx \sigma_e^2 + \frac{q^2}{12q_{\text{qry}}^2}$$

In a similar way, let evk^c be the evaluation key generated on the client-side. It is then in transit switched down to a modulus $q_{\text{evk}} = 2^{50}$ bits and then switched up to modulus q . Thus, we have

$$\sigma_{\gamma(\text{evk}^s)}^2 \approx \frac{q^2}{12q_{\text{evk}}^2}.$$

NameserverSetup(zone file) <pre> 01 Estimate number of required buckets 02 Choose random hash function h, hashing into buckets 03 for each A, AAAA or NS record in zone file 04 Get IP for entry; in case of NS record, resolve delegation IP 05 Append (hostname, record type, IP) to bucket $h(\text{hostname})$ 06 If bucket overflows, go to step 2 07 for each bucket B 08 Choose a bucket hash function h_B 09 Concatenate all tuples $(h_B(\text{hostname}), \text{record type}, \text{IP})$ 10 If h_B collides on two different hostnames, go to step 8 11 $\text{DB}[B] := \text{concatenated tuples, prefixed by } h_B$ 12 return (Preprocess(DB), h) NameserverReply((DB, h), qry) 01 if qry = INFO 02 return (zone name, DB shape, h) 03 else 04 return Respond(qry, DB) </pre>	ResolveStep(hostname, nameserver, n_{labels}) <pre> 01 truncated := hostname truncated to $n_{\text{labels}} + 1$ labels 02 Query nameserver with INFO; obtain (zone name, DB shape, h) 03 Derive PIR parameters from DB shape 04 Create query qry = Query($h(\text{truncated})$) 05 Query nameserver for qry to get resp 06 Obtain bucket payload as Extract(resp) 07 Extract h_B and all tuples $(h_b(\text{hostname}_i), \text{record type}_i, \text{IP}_i)$ 08 return ($\text{IP}_i, \text{record type}_i$) where $h_b(\text{hostname}_i) = h_b(\text{truncated})$ Resolve(hostname) 01 $n_{\text{max}} := \text{labels in hostname}$ 02 nameserver ← address of root nameserver 03 $n_{\text{labels}} \leftarrow 0$ 04 while hostname not resolved 05 Run ResolveStep(hostname, nameserver, n_{labels}) 06 if some IP is A or AAAA, and $n_{\text{labels}} + 1 = n_{\text{max}}$ 07 return this IP 08 elseif list of IPs is not empty 09 nameserver ← some returned IP 10 $n_{\text{labels}} \leftarrow n_{\text{labels}} + 1$ 11 abort if $n_{\text{labels}} \geq n_{\text{max}}$ </pre>
--	--

Figure 7: Client and nameserver pseudocode for complete DNS resolution via DNSPIR.

F.2 The Response Algorithm

The response algorithm is executed twice. First, on the DNS database. This database has been preprocessed, but since it consists of plaintexts this does not incur noise. After the first execution of the response algorithm, it is called again on a database consisting of the responses of the first round that have been split into plaintexts. Since the split up responses are considered plaintexts, thus the preprocessing step in Line 5 of the response algorithm in Figure 4 again does not incur noise. However, since they result from ciphertexts, they carry noise itself which we need to consider when analysing the noise of the response algorithm. We first look at the noise after applying an automorphism and key switching. Then, we analyse the noise growth of the first application of the response algorithm, and then that of the second.

The application of the automorphism itself does not incur any noise. The noise growth is only due to the key switching, which we analyse now. Let ct be a ciphertext, and $\text{ct}_{ks} = \text{KeySwitch}(\text{ct}, s_1, s_2)$. Let $\ell = \lceil \log_\beta(q) \rceil - 1$.

$$\begin{aligned}
\text{ct}_{ks} &= \langle \text{Decomp}(\text{ct}_1)^T, \text{evk}_{s_1 \rightarrow s_2} \rangle + (\text{ct}_0, 0) \\
&= \langle (\text{ct}_{1,0}, \dots, \text{ct}_{1,\ell}), \\
&\quad ((s_1\beta^0 - a_0s_2 + e_0, a_0), \dots, (s_1\beta^\ell - a_\ell s_2 + e_\ell, a_\ell)) \rangle \\
&\quad + (\text{ct}_0, 0) \\
&= \sum_{i=0}^{\ell} \text{ct}_{1,i} (s_1\beta^i - a_i s_2 + e_i, a_i) + (\text{ct}_0, 0).
\end{aligned}$$

Thus, we obtain for the critical quantity

$$\begin{aligned}
\gamma(\text{ct}_{ks}) &= \sum_{i=0}^{\ell} \text{ct}_{1,i} (s_1\beta^i - a_i s_2 + e_i) + \text{ct}_0 + \sum_{i=0}^{\ell} \text{ct}_{1,i} a_i \\
&= \sum_{i=0}^{\ell} \text{ct}_{1,i} \beta^i s_1 + \text{ct}_{1,i} e_i + \text{ct}_0 \\
&= \text{ct}_0 + \text{ct}_1 s_1 + \sum_{i=0}^{\ell} \text{ct}_{1,i} e_i \\
&= \gamma(\text{ct}) + \sum_{1,i} \text{ct}_{1,i} e_i.
\end{aligned}$$

Denote by $\gamma_{ks} = \sum_{i=0}^{\ell} \text{ct}_{1,i} e_i$ the additional noise term that occurs during key switching. Since we can model the $\text{ct}_{1,i}$ as being uniformly randomly distributed in the interval $[0, \beta - 1]$, they have variance $\frac{\beta^2 - 1}{12} \approx \frac{\beta^2}{12}$. We have seen that since the evaluation key is compressed and decompressed, the noise terms e_i have variance $\sigma_{e_i}^2 \approx \frac{q^2}{12q_{\text{evk}}^2}$. Thus, we have for the variance of the additional noise term

$$\sigma_{ks}^2 \approx \frac{(\ell + 1)N\beta^2 q^2}{144q_{\text{evk}}^2}.$$

We can now analyse the response algorithm. The database is in plaintext, so we will have products between deterministic polynomials and random polynomials. Note, that the variance of the product of a deterministic polynomial c and a random polynomial a with coefficient variance σ_a^2 is given by $\sigma_{ca}^2 = \|c\|_2^2 \sigma_a^2$.

We first analyse the first call to the response algorithm. Then, the responses resp_0 and resp_1 are calculated respectively as

$$\begin{aligned}\text{resp}_0 &= \text{Auto}(\text{resp}_1, 5, \text{evk}) + \tilde{D}B[5^0]\text{qry} \\ &= \text{Auto}(\text{Auto}(\text{resp}_2, 5, \text{evk}) + \tilde{D}B[5^1]\text{qry}) + \tilde{D}B[5^0]\text{qry} \\ &= \dots \\ &= \text{Auto}(\dots (\text{Auto}(\text{resp}_{\frac{N}{2}}, 5, \text{evk}) + \tilde{D}B[5^{\frac{N}{2}-1}]\text{qry}) \\ &\quad + \dots) + \tilde{D}B[5^1]\text{qry}) + \tilde{D}B[5^0]\text{qry}\end{aligned}$$

and

$$\begin{aligned}\text{resp}'_0 &= \text{Auto}(\dots (\text{Auto}(\text{resp}_{\frac{N}{2}}, 5, \text{evk}) + \tilde{D}B[5^{-\frac{N}{2}+1}]\text{qry}') \\ &\quad + \dots) + \tilde{D}B[5^{-1}]\text{qry}') + \tilde{D}B[5^0]\text{qry}'.\end{aligned}$$

Then, we have for the respective critical quantities

$$\begin{aligned}\gamma(\text{resp}_0) &= \gamma(\text{Auto}(\dots (\text{Auto}(\text{resp}_{\frac{N}{2}}, 5, \text{evk}) + \tilde{D}B[5^{\frac{N}{2}-1}]\text{qry}) \\ &\quad + \dots) + \tilde{D}B[5^1]\text{qry}) + \tilde{D}B[5^0]\text{qry}) \\ &= \gamma(\text{Auto}(\dots (\text{Auto}(\text{resp}_{\frac{N}{2}}, 5, \text{evk}) + \tilde{D}B[5^{\frac{N}{2}-1}]\text{qry}) \\ &\quad + \dots) + \tilde{D}B[5^1]\text{qry}) + \tilde{D}B[5^0]\gamma(\text{qry}) \\ &= \gamma(\text{Auto}(\dots (\text{Auto}(\text{resp}_{\frac{N}{2}}, 5, \text{evk}) + \tilde{D}B[5^{\frac{N}{2}-1}]\text{qry}) \\ &\quad + \dots) + \gamma_{\text{ks},1} + \sigma_5(\tilde{D}B[5^1])\gamma(\sigma_5(\text{qry})) \\ &\quad + \tilde{D}B[5^0]\gamma(\text{qry}) \\ &= \dots \\ &= \sigma_{5^{\frac{N}{2}-1}}(\tilde{D}B[5^{\frac{N}{2}-1}])\gamma(\sigma_{5^{\frac{N}{2}-1}}(\text{qry})) \\ &\quad + \sigma_{5^{\frac{N}{2}-2}}(\tilde{D}B[5^{\frac{N}{2}-2}])\gamma(\sigma_{5^{\frac{N}{2}-2}}(\text{qry})) \\ &\quad + \dots + \tilde{D}B[5^0]\gamma(\text{qry}) + \gamma_{\text{ks},1} + \dots + \gamma_{\text{ks},N/2}.\end{aligned}$$

Correspondingly, we have for the critical quantity of resp'_0 that

$$\begin{aligned}\gamma(\text{resp}'_0) &= \sigma_{5^{\frac{N}{2}-1}}(\tilde{D}B[5^{-\frac{N}{2}+1}])\gamma(\sigma_{5^{\frac{N}{2}-1}}(\text{qry}')) \\ &\quad + \sigma_{5^{-\frac{N}{2}+2}}(\tilde{D}B[5^{\frac{N}{2}-2}])\gamma(\sigma_{5^{\frac{N}{2}-2}}(\text{qry}')) + \dots \\ &\quad + \tilde{D}B[5^0]\gamma(\text{qry}') + \gamma'_{\text{ks},1} + \dots + \gamma'_{\text{ks},N/2}.\end{aligned}$$

The application of the automorphism has no effect on the standard deviation of the noise terms. Therefore, we obtain for the respective variances

$$\begin{aligned}\sigma_{\gamma(\text{resp}_0)}^2 &= \frac{N}{2}\sigma_{\text{ks}}^2 + \left(\|\sigma_{5^{\frac{N}{2}-1}}(\tilde{D}B[5^{\frac{N}{2}-1}])\|_2^2\right. \\ &\quad \left.+ \dots + \|\tilde{D}B[5^0]\|_2^2\right)\sigma_{\gamma(\text{qry})}^2 \\ &\leq \frac{N}{2}\sigma_{\text{ks}}^2 + \left(N\frac{t^2}{4} + \dots + N\frac{t^2}{4}\right)\sigma_{\gamma(\text{qry})}^2 \\ &= \frac{N}{2}\sigma_{\text{ks}}^2 + \frac{N^2t^2}{8}\sigma_{\gamma(\text{qry})}^2,\end{aligned}$$

and

$$\sigma_{\gamma(\text{resp}'_0)}^2 = \frac{N}{2}\sigma_{\text{ks}}^2 + \frac{N^2t^2}{8}\sigma_{\gamma(\text{qry}')}^2.$$

Using the results from the previous section, the variance after the first call to the response algorithm is

$$\begin{aligned}\sigma_{\text{resp}_0+\text{resp}'_0}^2 &= N\sigma_{\text{ks}}^2 + \frac{N^2t^2}{8}(\sigma_{\gamma(\text{qry})}^2 + \sigma_{\gamma(\text{qry}')}^2) \\ &= N\sigma_{\text{ks}}^2 + \frac{N^2t^2q^2}{96q_{\text{qry}}^2}.\end{aligned}$$

The response is then modulus switched down to $q_b = 2^{12}$ (first component) and $q_a = 2^{17}$ (second component) respectively. In the second call to the respond algorithm, we consider the ciphertexts obtained as responses in the first call as plaintexts. Thus, the second call produces a double encryption of the result. We will decrypt these layers individually, that is we are interested whether each layer has small enough noise to decrypt correctly. Thus, the noise analysis for both calls is the same. We will therefore analyse the modulus switching down together with the modulus switching up before decryption.

F.3 Transmission from Server to Client

The server transmits a ciphertext that has been modulus switched down to the compressed moduli, before decryption the client modulus switches the ciphertexts up to q again. That is, the ciphertext that is decrypted (both layers) have critical quantity

$$\begin{aligned}\gamma(\text{resp}^c) &= \left\lfloor \frac{q}{q_b} \left\lfloor \frac{q_b}{q} (\text{resp}_0 + \text{resp}'_0)[0] \right\rfloor \right\rfloor \\ &\quad + \left\lfloor \frac{q}{q_a} \left\lfloor \frac{q_a}{q} (\text{resp}_0 + \text{resp}'_0)[1] \right\rfloor \right\rfloor s \\ &= \frac{q}{q_b} \left(\frac{q_b}{q} (\text{resp}_0 + \text{resp}'_0)[0] + \tau_2 \right) \\ &\quad + \frac{q}{q_a} \left(\frac{q_a}{q} (\text{resp}_0 + \text{resp}'_0)[1] + \tau_3 \right) s + \tau_0 + \tau_1 s \\ &= \gamma(\text{resp}_0 + \text{resp}'_0) + \frac{q}{q_b}\tau_2 + \frac{q}{q_a}\tau_3 s.\end{aligned}$$

The secret key has been drawn from a ternary distribution with hamming weight h and therefore has variance $\sigma_s^2 = \frac{h}{N}$. The variance of the product of two independent random polynomials a and b of degree N is given by $\sigma_{ab}^2 = \delta_{\mathcal{R}}\sigma_a^2\sigma_b^2$, where $\delta_{\mathcal{R}}$ is the ring expansion factor. For a cyclotomic power of two ring, it can be theoretically shown that $\delta_{\mathcal{R}} \leq N$. In practice, we have $\delta_{\mathcal{R}} \approx \sqrt{N}$. Since we are interested in correctness, not security, we will use this value. Thus we get

$$\sigma_{\gamma(\text{resp}^c)}^2 = \sigma_{\gamma(\text{resp}_0+\text{resp}'_0)}^2 + \frac{q^2}{12q_b^2} + \frac{hq^2}{12q_a^2\sqrt{N}}.$$

F.4 Determining the Noise Budget

We have just determined the variance of the noise of the final ciphertext. It decrypts correctly, if the invariant noise budget $-\log_2(2\frac{t}{q}\|\gamma(\text{resp}^c)\|_\infty)$ is positive. We have

$$\begin{aligned}\|\gamma(\text{resp}^c)\|_\infty &\leq \alpha\sigma_{\gamma(\text{resp}^c)} \\ &= 8\sqrt{\sigma_{\gamma(\text{resp}_0+\text{resp}'_0)}^2 + \frac{q^2}{12q_b^2} + \frac{hq^2}{12q_a^2\sqrt{N}}}\end{aligned}$$

$$\begin{aligned}
 &= 8 \sqrt{N\sigma_{ks}^2 + \frac{N^2 t^2 q^2}{96q_{\text{qry}}^2} + \frac{q^2}{12q_b^2} + \frac{hq^2}{12q_a^2 \sqrt{N}}} \\
 &= 8 \sqrt{\frac{(\ell + 1)N^2 \beta^2 q^2}{144q_{\text{evk}}^2} + \frac{N^2 t^2 q^2}{96q_{\text{qry}}^2} + \frac{q^2}{12q_b^2} + \frac{hq^2}{12q_a^2 \sqrt{N}}}
 \end{aligned}$$

$$\approx 2^{40.976}.$$

This is smaller than $\frac{q}{t} \approx 2^{40.998}$. Thus, with a probability of $\approx 1 - 2^{-40}$ our protocol returns the correct IP address.