Syed Mahbub Hafiz and Ryan Henry*

A Bit More Than a Bit Is More Than a Bit Better

Faster (essentially) optimal-rate many-server PIR

Abstract: We study both the practical and theoretical efficiency of private information retrieval (PIR) protocols in a model wherein several untrusted servers work to obliviously service remote clients' requests for data and yet no pair of servers colludes in a bid to violate said obliviousness. In exchange for such a strong security assumption, we obtain new PIR protocols exhibiting remarkable efficiency with respect to every cost metric—download, upload, computation, and round complexity—typically considered in the PIR literature.

The new constructions extend a multiserver PIR protocol of Shah, Rashmi, and Ramchandran (ISIT 2014), which exhibits a remarkable property of its own: to fetch a b-bit record from a collection of r such records, the client need only download $b + 1$ bits total. We find that allowing "a bit more" download (and optionally introducing computational assumptions) yields a family of protocols offering very attractive trade-offs. In addition to Shah et al.'s protocol, this family includes as special cases (2-server instances of) the seminal protocol of Chor, Goldreich, Kushilevitz, and Sudan (FOCS 1995) and the recent DPF-based protocol of Boyle, Gilboa, and Ishai (CCS 2016). An implicit "folklore" axiom that dogmatically permeates the research literature on multiserver PIR posits that the latter protocols are the "most efficient" protocols possible in the perfectly and computationally private settings, respectively. Yet our findings soundly refute this supposed axiom: These special cases are (by far) the *least* performant representatives of our family, with essentially *all other* parameter settings yielding instances that are *significantly* faster.

Keywords: Private information retrieval; distributed point functions; provable security; optimality

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

Private information retrieval (PIR) is a cryptographic primitive enabling clients to fetch records from a remote database without revealing to the database holder(s) which records they fetch. This paper concerns both the practical and theoretical efficiency of so-called "1-private PIR protocols", wherein the database is hosted jointly by several, say $\ell > 1$, *computationally bounded* and *pairwise non-colluding* servers.

In recent work, Shah, Rashmi, and Ramchandran [28] proposed a construction in this model[1] with the following remarkable property: to fetch a b-bit record from a database comprising r such records, the client need only download $b+1$ bits total. In other words, their construction imposes *just one bit* of download overhead compared with the most efficient conceivable non-private protocol, thereby establishing that the *(download) capacity* of 1-private multiserver PIR is effectively perfect. Inspired by Shah et al.'s result, the information-theory research community has produced a steady succession of papers characterizing the download capacity of multiserver PIR protocols under a variety of constraints—when the number of servers is fixed [31]; when a threshold number of servers can collude [30]; when a subset of the servers may be non-responsive [32] or Byzantine [3]; and so on. One can justifiably view Shah et al.'s result and the body of work it inspired as a breakthrough in our understanding of multiserver PIR's theoretical underpinnings and yet, to date, this work has had essentially zero impact on PIR research within the cryptography and applied privacy research communities. Indeed, so aggressively optimizing download cost at the expense of all else imposes rather exorbitant costs for the other metrics of interest—so exorbitant, in fact, that the constructions in each of the aforementioned papers end up falling short of satisfying any reasonable definition of "non-triviality" for a PIR protocol (see Definition 3 below).

Our contributions

In this paper, we revisit the "one-extra-bit" construction of Shah et al. with an eye toward mathematical rigor and concrete practicality. Specifically, we present and analyze a novel family of what we call "*one-extra-word*" protocols, of which the one-extra-bit construction is an extreme special case. Interestingly, the new family also captures as special cases the 2-server perfectly 1-private protocol of Chor, Goldreich, Kushilevitz, and Sudan [11, §2.1] and the computationally 1-

Syed Mahbub Hafiz: Indiana University, E-mail: shafiz@indiana.edu
***Corresponding Author: Ryan Henry:** University of Calgary, E-mail: ryan.henry@ucalgary.ca

1 In fact, the servers in Shah et al.'s construction—much like the servers in all but the most efficient of our constructions—need not be computationally bounded for the privacy guarantee to hold.

private protocol of Boyle, Gilboa, and Ishai [8, §3.2]. In terms of practicality, the fact that our family captures the latter two protocols is significant; indeed, an implicit "folklore" conjecture positing that these protocols are the "most efficient" possible appears to underlie a good deal of existing PIR research [20, 27, 34]. Yet our findings soundly refute this folklore conjecture: These special cases are among the *least* practical representatives of our family, with essentially *all other* reasonable parameter settings yielding instances that are *significantly* more practical. Specifically, the performance improvements we obtain relative to these special cases grows in proportion to the number of pairwise non-colluding servers that participate in each query, up to some inflection point after which further marginal improvements to download and computation costs are dwarfed by rapidly growing upload costs to an unrealistically large number of servers. (Continuing well beyond this inflection point eventually yields the one-extra-bit construction.) The new family sports detailed security and concrete efficiency analyses, which help to elucidate (and improve on) some curious properties of the efficiency and privacy (or lack thereof) of the original one-extra-bit protocol.

As our main contribution, we exhibit, for any $L \geq 1$, a 2^L-server one-extra-word protocol instance that, although falling short of imposing only a single bit of download overhead, nonetheless (i) provides superior privacy guarantees compared to the one-extra-bit construction and (ii) is *shockingly* efficient (when $L > 1$) with respect to every cost metric—download, upload, computation, and round complexity—typically considered in the PIR literature. For a database comprising r-many b-bit records, the client in our protocol uploads just $130L(\lceil \lg r \rceil - 7)$ bits and downloads just $b/(2^L - 1)$ bits of data per server, while each server performs $L\lceil r/64 \rceil$ AES-128 evaluations and an expected $rb/(2^L - 1)$ bit operations (at the 128-bit security level). These download and computation costs sharply match known lower bounds, suggesting that our protocols are among the most efficient possible. We also extend this protocol to arbitrarily many (non-power-of-2) servers, at the cost of increasing the upload cost and AES-128 evaluation count each by a (small) factor polynomial in the security parameter. This all compares *very* favorably with prior work; in fact, based on our analysis and empirical performance measurements from our prototype implementation, we hazard to claim that, all told, ours is the most practical multiserver PIR protocol proposed and implemented to date—by a significant margin.

In short: Our main contribution is to transform Shah et al.'s one-extra-bit construction from a (highly impractical) theoretical result into what we believe to be the most efficient PIR protocol currently in existence, albeit under the same very strong trust assumption employed by the original.

Outline: Section 2 gives formal definitions for private information retrieval. Section 3 presents the one-extra-word family of protocols (deferring its "one-extra-bit" instantiation to Appendix B and a detailed security analysis for the family as a whole to Appendix C). Section 4 introduces the bit-more-than-a-bit construction, a subfamily of one-extra-word protocols that improves on one-extra-bit in several respects. Section 5 describes an efficient, computationally 1-private bit-more-than-a-bit construction, first for 2^L servers (§5.2) and then for arbitrarily many servers (§5.3). Section 6 presents findings from our prototype implementation, including head-to-head comparisons with the "folklore" protocols of Chor et al. and Boyle et al. in addition to a selection of multiserver protocols from the open-source Percy++ (§6.4) and RAID-PIR (§6.5) libraries. Section 7 wraps up with some concluding remarks and directions for future work. Appendix A briefly discusses of our results in the context of related work.

2 Private information retrieval

Let $\mathbb{F}$ be a finite field and let $\boldsymbol{D} \in \mathbb{F}^{r \times s}$ be a database consisting of r many s-element records (called *blocks*), each of which is indexed by a positive integer less than or equal to r. Intuitively, PIR is a cryptographic technique that allows a client to fetch one or more blocks of its choosing from $\boldsymbol{D}$ without disclosing to the remote server (or servers) holding $\boldsymbol{D}$ any information about which blocks it fetches. We formalize this intuitive notion of privacy using an *indistinguishability*-based definition, as described below.

Formally defining PIR: Consider the interaction that occurs between a client who seeks the block $\vec{\boldsymbol{D}}_i \in \mathbb{F}^s$ indexed by i and a remote database server (or servers) who holds $\boldsymbol{D}$. The client initiates the interaction by sending a *query* string $q \in \{0,1\}^*$ to the server(s), who return a *response* string $z \in \{0,1\}^*$ from which the client extracts its desired block. Let I, Q, R, and E respectively denote the random variables that describe (i) the index $i \in [1..r]$ of the block the client seeks, (ii) the query string $q \in \{0,1\}^*$ the client sends to the server, (iii) the response string $z \in \{0,1\}^*$ the server(s) returns to the client, and (iv) the string $d \in \mathbb{F}^s$ the client ultimately outputs. From the client's perspective, the 4-tuple (I, Q, R, E) completely characterizes the interaction that occurs in a given protocol run.

"Impossibly perfect" privacy: Ideally, a PIR protocol would guarantee that an adversary who controls the database server(s)—no matter how powerful and well-positioned to launch an attack—cannot possibly deduce *anything* from a client's requests (beyond the trivial fact that the client is looking for *something*). One could formalize this requirement by insisting that the distribution of I be statistically independent

from that of Q. This requirement leads to a notion that we refer to as *"impossibly perfect" privacy*, so called because Chor et al. observed in their seminal paper [11, §5.1] that this level of privacy is impossible for any "reasonable" PIR construction to provide. (We elaborate on what we mean by "reasonable" shortly.) It is nevertheless instructive to state a formal definition for impossibly perfect privacy—no matter how unsatisfiable—as it will serve as a useful starting point for our actual, satisfiable privacy definition (Definition 4) below.

Definition 1. *The interaction described by (I, Q, R, E) provides* impossibly perfect privacy *if, for all block indices $i, i^* \in [1..r]$ and for all query strings $q \in \{0,1\}^*$, we have*

$$\Pr[Q = q \mid I = i] = \Pr[Q = q \mid I = i^*],$$

where the probability is over all the random coin tosses made by the client.

Definition 1 insists that the conditional distribution of Q given $I = i$ be *identical in its entirety* to that of Q given $I = i^*$ (as opposed to merely insisting that these distributions be *ε-close* or *polynomial-time indistinguishable*, or insisting that certain *substrings* of Q be identically distributed); that is, it insists that the client's query string q contains "no information"—in a strict, information-theoretic sense—about which block the client is using that query string to fetch. This suggests two natural relaxations that might lead to "possible" notions of privacy: (i) insist that the above distributions be merely computationally indistinguishable or (ii) restrict the attacker to observing only part of any given sample from Q.

In fact, the privacy definition we ultimately use—called *computational 1-privacy*—adopts both relaxations. Before stating that definition, we define two additional requirements (correctness and non-triviality) for a PIR protocol, which collectively describe what we meant above when we spoke of "reasonable" PIR protocols with respect to which Definition 1 is impossible to satisfy.

Correctness: One trivial way to satisfy Definition 1 is to have the client choose $q \in \{0,1\}^*$ arbitrarily and without regard for i. Upon receiving q from the client, the server(s) likewise responds with arbitrary $z \in \{0,1\}^*$, unrelated to q or $\boldsymbol{D}$. This interaction clearly offers perfect privacy, but it is not terribly useful: the response z provides nothing to help the client correctly extract its desired block. The following *correctness* criterion disqualifies such "useless" protocols, requiring that the client actually learns its desired block.

Definition 2. *The interaction described by (I, Q, R, E) provides* (perfect) correctness *if*

$$\Pr[E = \vec{\boldsymbol{D}}_i \mid I = i] = 1,$$

where $\vec{\boldsymbol{D}}_i \in \mathbb{F}^s$ denotes the block indexed by i within $\boldsymbol{D}$.

Notice that the distribution of E is dependent not only on those of I and Q but also on that of R; thus, correctness is necessarily contingent on the faithful execution of the protocol by the server(s).

Non-triviality: Another trivial way to satisfy Definition 1, while simultaneously satisfying Definition 2, is to again have the client choose $q \in \{0,1\}^*$ arbitrarily and without regard for i, but then to have the server(s) respond with *the entire database*; that is, with $z = \boldsymbol{D}$. Upon receiving $\boldsymbol{D}$, the client would then extract the block $\vec{\boldsymbol{D}}_i$ from $\boldsymbol{D}$ on her local machine, thus ensuring perfect privacy in exchange for considerable download overhead. The following *non-triviality* criterion disqualifies such "trivially correct" protocols by insisting that the (bidirectional) communication cost of the interaction be asymptotically smaller than the size of $\boldsymbol{D}$.

Definition 3. *The interaction described by (I, Q, R, E) provides* non-trivial communication *if*

$$\sum_{n \in \mathbb{N}} n \cdot \Pr[|Q| + |R| = n] \in o(|\boldsymbol{D}|);$$

that is, if the expected (combined) bitlength *of the query and response strings scale sublinearly with the bitlength $|\boldsymbol{D}|$ of $\boldsymbol{D}$.*[2]

We remark that the various capacity-achieving PIR constructions described in the body of work [3, 28, 30–32] cited in Section 1 satisfy a weaker definition of non-triviality wherein only *download* (i.e., $|Q|$) is required to scale sublinearly with $|\boldsymbol{D}|$. In fact, in several of those constructions—e.g., the capacity-achieving constructions of Sun and Jafar [30–32]—the upload cost $|R|$ scales *(super-)exponentially* in $|\boldsymbol{D}|$.

Computational and perfect C-privacy: The confluence of Definitions 2 and 3 rules out the "useless" and "trivial" PIR candidates we have considered thus far; indeed, as noted previously, Chor et al. proved in their seminal paper on PIR that it would be *impossible* to construct any protocol—no matter how clever—that satisfies Definitions 1–3 simultaneously. Nevertheless, in the very same paper, they proposed a (simultaneously *correct* and *non-trivial*) protocol involving several (say, $\ell > 1$) non-colluding but otherwise untrusted servers that *does* perfectly hide the client's request from each of the ℓ servers in isolation. In such a multiserver interaction, the query and response are both ℓ-tuples, say $q = (q_1, \ldots, q_\ell)$ and $z = (z_1, \ldots, z_\ell)$, from which the client sends $q_j \in \{0,1\}^*$ to and receives $z_j \in \{0,1\}^*$ from the jth server.

It is clear that a *grand coalition* comprising all ℓ servers in such a multiserver interaction wields privacy-infringing powers tantamount to those of the lone server in a single-

2 One could also consider a variant of Definition 3 that considers the *maximum* possible—rather than *expected*—communication cost of an interaction.

server interaction, in which case Chor et al.'s impossibility result applies.[3] Consequently, no multiserver protocol can hope to provide perfect privacy unless certain coalitions among the servers do not form. In order to capture this limitation, it is necessary to both extend and relax Definition 1. Let $C \subseteq [1..\ell]$ be an arbitrary coalition among the ℓ servers and let Q_C denote the random variable that describes the subsequence of query strings $(q_j)_{j \in C}$ that the client sends to the servers in C.

Definition 4. *The interaction described by* (I, Q, R, E) *provides* perfect privacy with respect to C *(or* perfect C-privacy*) if, for all block indices* $i, i^* \in [1..r]$,

$$\Pr[Q_C = (q_j)_{j \in C} \mid I = i] = \Pr[Q_C = (q_j)_{j \in C} \mid I = i^*],$$

where the probability is over all the random coin tosses made by the client.

Notice that the conditional probabilities on Q_C depend only on the distributions of I and Q: Definition 4 permits no restrictions on the computational abilities of nor strategies employed by the servers in C—it merely assumes that they learn nothing of the q_j for $j \in [1..\ell] \setminus C$. Many of the PIR constructions considered herein are provably private in the sense of Definition 4; however, proving that our most efficient construction (see Section 5) is private requires—in addition to a non-collusion assumption—a *computational* assumption (namely, the existence of fixed-expansion pseudorandom generators). Such a "computational" notion of privacy for multiserver PIR protocols was first considered by Chor and Gilboa [9] shortly after the introduction of perfectly private PIR.

Definition 5. *The interaction described by* (I, Q, R, E) *provides* computational privacy with respect to C *(or* computational C-privacy*) if, for all block indices* $i, i^* \in [1..r]$*, the distribution ensembles* $\{Q_C \mid I = i\}_{\lambda \in \mathbb{N}}$ *and* $\{Q_C \mid I = i^*\}_{\lambda \in \mathbb{N}}$ *are computationally indistinguishable (with security parameter* λ*).*

We stress that the hardness of distinguishing distribution ensembles depends not on $\boldsymbol{D}$ but on some underlying security parameter (the size of which will invariably impact the computationally burden on the client and honest servers). Looking ahead, the security parameter in our computationally 1-private bit-more-than-a-bit constructions will dictate the seed-length for some fixed-expansion pseudorandom generator (which we concretely realize using AES).

3 Indeed, given an ℓ-server protocol that maintains privacy against such a grand coalition, one could easily construct a single-server protocol in which the lone server takes on the roles of all ℓ servers. The resulting protocol would clearly contradict Chor et al.'s impossibility result.

An interaction that satisfies either Definition 4 or 5 with respect to *all* coalitions in the family of subsets $\Gamma = \{C \subseteq [1..\ell] \mid |C| \leq t\}$ for some positive-integer threshold $t < \ell$ is colloquially said to be (*perfectly* or *computationally*) *t-private*. Given Definitions 2–5, we can now provide a formal definition for 1-private PIR protocols.

Definition 6. *The interaction described by* (I, Q, R, E) *is a (*perfectly *or* computationally*)* 1-private information retrieval protocol *if it provides (i) (*perfect*)* correctness*, (ii)* non-trivial communication*, and (iii) (*perfect *or* computational*)* 1-privacy *(in the senses of Definitions 2–5, respectively).*

3 "One-extra-word" protocols

We now present our generalization of the "one-extra-bit" construction over an arbitrary finite field $\mathbb{F}$; we refer to this generalization as the "one-extra-word" family of PIR protocols. Members of the one-extra-word family are parametrized by (i) the underlying field $\mathbb{F}$, (ii) the number s of $\mathbb{F}$ elements needed to represent each block, (iii) the number of participating servers $\ell > 1$, and (iv) a vector $\vec{\boldsymbol{v}} \in \mathbb{F}^s$ and mapping φ to be described below. We call such a tuple of parameters $(\mathbb{F}, s, \ell, \varphi, \vec{\boldsymbol{v}})$ a *one-extra-word instance*. (To recover one-extra-bit as a one-extra-word instance, we simply set $\mathbb{F} = \mathbf{GF}(2)$ and $\ell = s + 1$, and then instantiate the $\vec{\boldsymbol{v}}$ and φ in a particular way—more on this in Appendix B.)

Notation and preliminaries: Recall that the database $\boldsymbol{D}$ is an $r \times s$ matrix over a finite field $\mathbb{F}$, in which each of the r rows is an s-word block of fetchable data. For each $j \in [1..s+1]$, let $\vec{\boldsymbol{e}}_j$ denote the jth standard basis vector of $\mathbb{F}^{s+1}$. (We also overload $\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_s$ to denote the standard basis vectors in $\mathbb{F}^s$; the dimension of the vector space in which a given $\vec{\boldsymbol{e}}_j$ resides will always be clear from context.)

Denote by $\mathbf{M}^{(r,s)} \subseteq \mathbb{F}^{r \times (s+1)}$ the set of all height-r matrices whose rows are vectors from the standard basis $\{\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_{s+1}\}$, and consider the family (indexed by $i \in [1..r]$) of equivalence relations $\equiv_i$ defined on the $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ as

$$\boldsymbol{A} \equiv_i \boldsymbol{B} \text{ iff } \mathrm{Row}_{i^*}(\boldsymbol{A} - \boldsymbol{B}) \neq \vec{\boldsymbol{0}} \text{ implies } i^* = i,$$

where $\mathrm{Row}_{i^*}(\boldsymbol{A} - \boldsymbol{B})$ denotes the i^*th row of $\boldsymbol{A} - \boldsymbol{B}$ and $\vec{\boldsymbol{0}}$ denotes the zero vector in $\mathbb{F}^{s+1}$. Read the expression $\boldsymbol{A} \equiv_i \boldsymbol{B}$ as "$\boldsymbol{A}$ is *i-equivalent* to $\boldsymbol{B}$".

The following two-part observation is an immediate consequence of the above definitions of $\mathbf{M}^{(r,s)}$ and the $\equiv_i$.

Observation 1. *Given any* $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ *and* $i, i^* \in [1..r]$,

(i) there exists an equivalence class $\mathrm{Eq}(i; \boldsymbol{A}) \subseteq \mathbf{M}^{(r,s)}$ *comprising precisely* $s + 1$ *matrices (including* $\boldsymbol{A}$ *itself) such that* $\boldsymbol{A} \equiv_i \boldsymbol{B}$ *iff* $\boldsymbol{B} \in \mathrm{Eq}(i; \boldsymbol{A})$*, and*

(ii) if $\boldsymbol{A} \equiv_i \boldsymbol{B}$ *and* $\boldsymbol{A} \equiv_{i^*} \boldsymbol{B}$*, then* $i = i^*$ *or* $\boldsymbol{A} = \boldsymbol{B}$*.*

The $s+1$ matrices in $\mathrm{Eq}(i;\boldsymbol{A})$ whose existences are guaranteed by Part (i) of Observation 1 are precisely those matrices in $\mathbf{M}^{(r,s)}$ that are identical to $\boldsymbol{A}$ in all but (perhaps) their ith rows. Moreover, the ith rows of the matrices in $\mathrm{Eq}(i;\boldsymbol{A})$ collectively span the entire standard basis for $\mathbb{F}^{s+1}$; in other words, if $\mathrm{Eq}(i;\boldsymbol{A}) = \{\boldsymbol{B}_1,\ldots,\boldsymbol{B}_{s+1}\}$, then $\{\mathrm{Row}_i(\boldsymbol{B}_1),\ldots,\mathrm{Row}_i(\boldsymbol{B}_{s+1})\} = \{\vec{\boldsymbol{e}}_1,\ldots,\vec{\boldsymbol{e}}_{s+1}\}$ up to ordering. By convention, we assume the subscripts indexing the $\boldsymbol{B}_j \in \mathrm{Eq}(i;\boldsymbol{A})$ are selected so as to induce a "sorted" view in which $\mathrm{Row}_i(\boldsymbol{B}_j) = \vec{\boldsymbol{e}}_j$ for each $j \in [1..s+1]$. Part (ii) of Observation 1 merely adds that a given pair of distinct matrices can be i-equivalent for *at most one* index $i \in [1..r]$. An easy counting argument establishes that a given pair $(\boldsymbol{A},\boldsymbol{B})$ need not be i-equivalent for *any* of the $i \in [1..r]$.

Encoding the database: The database encoding algorithm takes as input the database $\boldsymbol{D} \in \mathbb{F}^{r\times s}$, a vector $\vec{\boldsymbol{v}} \in \mathbb{F}^s$, and a (surjective) function $\varphi\colon \mathbf{M}^{(r,s)} \to [1..\ell]$ mapping each matrix $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ to a positive integer; it outputs a collection of ℓ buckets—one per server—indexed by the $j \in [1..\ell]$. The algorithm first initializes the ℓ buckets to empty and "augments" the database $\boldsymbol{D} \in \mathbb{F}^{r\times s}$ by affixing an $(s+1)$th column, which it computes as the matrix-vector product $\boldsymbol{D}\vec{\boldsymbol{v}}^{\mathrm{T}} \in \mathbb{F}^{s\times 1}$. Denote the augmented database by $\boldsymbol{D}^* := \boldsymbol{D}\|(\boldsymbol{D}\vec{\boldsymbol{v}}^{\mathrm{T}}) \in \mathbb{F}^{r\times(s+1)}$ and observe that, by construction, the final column of $\boldsymbol{D}^*$ is just a linear combination of its first s columns.

Next, to populate the buckets, the algorithm computes, for each of the $(s+1)^r$ matrices $\boldsymbol{A}$ in $\mathbf{M}^{(r,s)}$, the *Frobenius inner product*[4], $\langle \boldsymbol{D}^*,\boldsymbol{A}\rangle_{\mathrm{F}}$, of $\boldsymbol{D}^*$ with $\boldsymbol{A}$, and then it places the result (a *scalar* from $\mathbb{F}$) in the bucket indexed by $\varphi(\boldsymbol{A})$. In the simplest possible instantiations, φ is a bijection so that each of the $(s+1)^r$ resulting scalars constitutes the sole contents of its own bucket (thus necessitating $\ell = (s+1)^r$ non-colluding servers). In the sequel, we discuss more efficient instantiations of φ (including the one employed by one-extra-bit), which allow the scheme to use significantly fewer servers.

Fetching a block: Recall that the database $\boldsymbol{D}$ comprises r blocks and that we defined r equivalence relations $\equiv_i$ over $\mathbf{M}^{(r,s)}$. A query for $\vec{\boldsymbol{D}}_i$, the block at index i within $\boldsymbol{D}$, corresponds to a random equivalence class of $\mathbf{M}^{(r,s)}$ under $\equiv_i$. Specifically, the client fetches $\vec{\boldsymbol{D}}_i$ by first selecting a uniform random matrix $\boldsymbol{A} \in_{\mathrm{R}} \mathbf{M}^{(r,s)}$, and then retrieving $\langle \boldsymbol{D}^*,\boldsymbol{B}_j\rangle_{\mathrm{F}}$ from bucket $\varphi(\boldsymbol{B}_j)$ for each $\boldsymbol{B}_j \in \mathrm{Eq}(i;\boldsymbol{A})$. Because $\equiv_i$ is symmetric (so that $\boldsymbol{B} \in \mathrm{Eq}(i;\boldsymbol{A})$ iff $\boldsymbol{A} \in \mathrm{Eq}(i;\boldsymbol{B})$), for any given index i, there exist precisely $(s+1)^r/(s+1) = (s+1)^{r-1}$ distinct queries with which the client can request $\vec{\boldsymbol{D}}_i$. The next observation is a direct consequence of the remarks following Observation 1.

Observation 2. *If* $\mathrm{Eq}(i;\boldsymbol{A}) = \{\boldsymbol{B}_1,\ldots,\boldsymbol{B}_{s+1}\}$, *then the* s *equations* $(\vec{\boldsymbol{e}}_j - \vec{\boldsymbol{v}})\cdot\langle x_1,\ldots,x_s\rangle^{\mathrm{T}} = \langle \boldsymbol{D}^*,\boldsymbol{B}_j\rangle_{\mathrm{F}} - \langle \boldsymbol{D}^*,\boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}$ *for* $j \in [1..s]$ *form a system of* s *linear equations in* s *unknowns. Moreover,* $\vec{\boldsymbol{D}}_i = \langle x_1,\ldots,x_s\rangle$ *is in the solution set of this system.*

Note that Observation 2 implies that the client can compute $\vec{\boldsymbol{D}}_i$ given $\vec{\boldsymbol{v}}$ and the sequence of Frobenius products $\langle \boldsymbol{D}^*,\boldsymbol{B}_1\rangle_{\mathrm{F}},\ldots,\langle \boldsymbol{D}^*,\boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}$ whenever the system of linear equations

$$\begin{pmatrix}\vec{\boldsymbol{e}}_1 - \vec{\boldsymbol{v}}\\ \vdots \\ \vec{\boldsymbol{e}}_s - \vec{\boldsymbol{v}}\end{pmatrix}\vec{\boldsymbol{D}}_i^{\mathrm{T}} = \begin{pmatrix}\langle \boldsymbol{D}^*,\boldsymbol{B}_1\rangle_{\mathrm{F}} - \langle \boldsymbol{D}^*,\boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}\\ \vdots \\ \langle \boldsymbol{D}^*,\boldsymbol{B}_s\rangle_{\mathrm{F}} - \langle \boldsymbol{D}^*,\boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}\end{pmatrix} \tag{2.1}$$

has full rank and, therefore, a unique solution.

Theorem 1. *A one-extra-word instance provides* (perfect) correctness *(in the sense of Definition 2) provided* $\|\vec{\boldsymbol{v}}\|_1 \neq 1$.[5]

Proof. Write $\sum_{j=1}^s c_j(\vec{\boldsymbol{e}}_j - \vec{\boldsymbol{v}}) = \vec{\boldsymbol{0}}$, so that $\sum_{j=1}^s c_j\vec{\boldsymbol{e}}_j = \sum_{j=1}^s c_j\vec{\boldsymbol{v}}$. If $\sum_{j=1}^s c_j = 0$, then $\sum_{j=1}^s c_j\vec{\boldsymbol{e}}_j = \vec{\boldsymbol{0}}$, which (due to the linear independence of $\vec{\boldsymbol{e}}_1,\ldots,\vec{\boldsymbol{e}}_s$) can only happen when $c_j = 0$ for every $j \in [1..s]$. Otherwise, if $\sum_{j=1}^s c_j \neq 0$, then $\vec{\boldsymbol{v}} = \left(\sum_{j=1}^s c_j\vec{\boldsymbol{e}}_j\right)/\left(\sum_{j=1}^s c_j\right)$, in which case $\|\vec{\boldsymbol{v}}\|_1 = \sum_{j=1}^s \left(c_j/\left(\sum_{j=1}^s c_j\right)\right) = \left(\sum_{j=1}^s c_j\right)/\left(\sum_{j=1}^s c_j\right) = 1$. □

We note that the choice $\vec{\boldsymbol{v}} = \vec{\boldsymbol{0}}$ yields an especially convenient (i.e., "already solved") system of equations in which the matrix on the left-hand side of Equation (2.1) is just the identity matrix in $\mathbb{F}^{s\times s}$ so that $\langle \boldsymbol{D}^*,\boldsymbol{B}_j\rangle_{\mathrm{F}} - \langle \boldsymbol{D}^*,\boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}$ is equal to the jth word of $\vec{\boldsymbol{D}}_i$ for each $j \in [1..s]$. We consider arbitrary $\vec{\boldsymbol{v}}$ not because it is ever beneficial to use a $\vec{\boldsymbol{v}} \neq \vec{\boldsymbol{0}}$—indeed, it seems always best to set $\vec{\boldsymbol{v}} = \vec{\boldsymbol{0}}$—but to ensure that the one-extra-bit construction remains a special case of our own.

Communication cost analysis: By inspection, the download cost aggregated across all servers (i.e., $|R|$) is $s+1$ field elements—one element from each of $s+1$ many buckets. When $\mathbb{F} = \mathbf{GF}(2)$, this yields a download cost of just $b+1$ bits to fetch a b-bit block, which is the lowest possible download cost for any PIR protocol [28, Theorem 1].

For each $j \in [1..s+1]$, the client must specify to the server holding bucket $\varphi(\boldsymbol{B}_j)$ which of the $|\varphi^{-1}(\varphi(\boldsymbol{B}_j))|$ Frobenius products from that bucket it seeks; thus, the upload cost aggregated across all servers (i.e., $|Q|$) is

$$F(\boldsymbol{A}) := \sum_{j=1}^{s+1} \max\left(\left\lceil \lg\left|\varphi^{-1}(\varphi(\boldsymbol{B}_j))\right|\right\rceil, 1\right) \text{ bits},$$

where the max is due to the fact that the client must still send at least one bit to the server holding bucket $\varphi(\boldsymbol{B}_j)$, as a "signal", even if $|\varphi^{-1}(\varphi(\boldsymbol{B}_j))| = 1$ so that $\lceil \lg|\varphi^{-1}(\varphi(\boldsymbol{B}_j))|\rceil = 0$.

4 The *Frobenius inner product* of $\boldsymbol{D}^*$ and $\boldsymbol{A}$ is $\langle \boldsymbol{D}^*,\boldsymbol{A}\rangle_{\mathrm{F}} := \mathrm{tr}(\boldsymbol{D}^*\boldsymbol{A}^{\mathrm{T}})$ or, equivalently, the sum of the products of each pair of corresponding components in $\boldsymbol{D}^*$ and $\boldsymbol{A}$.

5 Here $\|\vec{\boldsymbol{v}}\|_1 := \sum_{j=1}^s v_j$ denotes the L_1-norm of $\vec{\boldsymbol{v}} = \langle v_1,\ldots,v_s\rangle$.

Theorem 2. *A one-extra-word instance provides* non-trivial communication *(in the sense of Definition 3) provided the mapping φ satisfies*

$$\max_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\left\{\lg\left|\varphi^{-1}\big(\varphi(\boldsymbol{A})\big)\right|\right\}\in o(rw),$$

where $w\in\Omega(\lg|\mathbb{F}|)$ is the bitlength of (the representation used to describe) each $\mathbb{F}$ element.

Proof. Since $|R| = (s+1)w < 2sw \in o(|\boldsymbol{D}|)$, it will suffice to show that $|Q| \in o(rws)$. Let $x = \max_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\{\lg|\varphi^{-1}(\varphi(\boldsymbol{A}))|\}$. The size of a query $q = (q_1,\ldots,q_{s+1})$ involving the servers holding buckets $\mathrm{Eq}(i;A) = \{\boldsymbol{B}_1,\ldots,\boldsymbol{B}_{s+1}\}$ is

$$\begin{aligned}\left|(q_1,\ldots,q_{s+1})\right| &= \textstyle\sum_{j=1}^{s+1}\max\left(\left\lceil\lg\left|\varphi^{-1}\big(\varphi(\boldsymbol{B}_j)\big)\right|\right\rceil,1\right)\\ &\le \textstyle\sum_{j=1}^{s+1}\max\big(\lceil x\rceil,1\big)\\ &= (s+1)\max\big(\lceil x\rceil,1\big)\\ &= (s+1)\,o(rw)\\ &\in o(rws).\end{aligned}$$

□

The condition on $\max_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\{\lg|\varphi^{-1}(\varphi(\boldsymbol{A}))|\}$ given by Theorem 2 is *sufficient but far from necessary* to obtain non-triviality. A necessary condition would require only that the function $F(\boldsymbol{A})$ mapping each choice of $\boldsymbol{A}\in\mathbf{M}^{(r,s)}$ to the upload cost conditioned on that choice satisfies $\mathbf{Exp}[F(\boldsymbol{A})]\in o(|\boldsymbol{D}|)$. However, the simpler (and stricter) condition in Theorem 2 immediately yields the following corollary.

Corollary 1. *A one-extra-word instance provides* non-trivial communication *(in the sense of Definition 3) if each bucket holds a number of Frobenius products in $2^{o(rw)}$.*

Note that Corollary 1 implies that a one-extra-word instance provides non-trivial communication under the (perfectly reasonable, yet much stronger) assumption that the storage capacity of each server is polynomially bounded in $|\boldsymbol{D}|$.

Computation cost analysis: The (online) computation cost for each server is effectively nil. The computation cost for the client consists of the cost of setting up the system of equations—which is s subtractions in $\mathbb{F}$—plus the cost of solving that system—which is at most an additional $O(s^{2+\epsilon})$ field operations, for some $\epsilon \ge 0$. In the special case where $\vec{\boldsymbol{v}} = \vec{\boldsymbol{0}}$, the latter step is "free" and the total client-side computation cost is just s subtractions in $\mathbb{F}$. (When $\mathbb{F}$ is a binary field, as in the one-extra-bit construction and our own bit-more-than-a-bit constructions, each of the s subtractions becomes a bitwise exclusive-OR.)

Security analysis: Privacy of a one-extra-word instance depends on the choice of φ. Note that each server holds one and only one bucket; for convenience, we equate each server with the bucket it holds. The next theorem gives necessary and sufficient conditions for the interaction to provide perfect privacy against singleton coalitions (i.e., against *individual* servers).

Theorem 3. *A one-extra-word instance provides* perfect 1-privacy *(in the sense of Definition 4) if and only if the mapping $\varphi\colon \mathbf{M}^{(r,s)}\to[1..\ell]$ satisfies the following condition:*

$$\forall\boldsymbol{A}\in\mathbf{M}^{(r,s)}\text{ and }i\in[1..r],\ \left|\varphi\big(\mathrm{Eq}(i;\boldsymbol{A})\big)\right| = s+1.$$

Intuitively, Theorem 3 says that a one-extra-word instance is private (with respect to singleton coalitions) provided φ never maps two distinct matrices from the *same equivalence class* (under *any* of the $\equiv_i$) to the same bucket. We note that this condition is trivially satisfied when φ is a bijection; in the sequel, we describe some other choices for φ that also satisfy the condition set forth in Theorem 3.

The proof of Theorem 3 uses Lemma 1, whose own proof follows easily from Part (i) of Observation 1. Before stating and proving the lemma, it will be helpful to briefly recall the relevant portions of the process by which the client requests a block $\vec{\boldsymbol{D}}_i$: the client selects a matrix $\boldsymbol{A}$ *uniformly* at random from $\mathbf{M}^{(r,s)}$, and then it retrieves $\langle\boldsymbol{D}^*,\boldsymbol{B}_j\rangle_{\mathrm{F}}$ for each $\boldsymbol{B}_j\in\mathrm{Eq}(i;\boldsymbol{A})$.

Lemma 1. *For any $i\in[1..r]$ and any $\boldsymbol{B}\in\mathbf{M}^{(r,s)}$, a client seeking to fetch $\vec{\boldsymbol{D}}_i$ retrieves the Frobenius product $\langle\boldsymbol{D}^*,\boldsymbol{B}\rangle_{\mathrm{F}}$ from $\varphi(\boldsymbol{B})$ with probability $1/(s+1)^{r-1}$.*

Proof. Fix $\boldsymbol{B}\in\mathbf{M}^{(r,s)}$ and $i\in[1..r]$, and let X be the (uniform) random variable describing the client's initial selection of $\boldsymbol{A}\in\mathbf{M}^{(r,s)}$. We need to prove that $\Pr[B\in\mathrm{Eq}(i;X)] = 1/(s+1)^{r-1}$. From the Law of Total Probabilities, we have

$$\begin{aligned}\Pr\big[\boldsymbol{B}\in\mathrm{Eq}(i;X)\big] &= \textstyle\sum_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\Pr\big[\boldsymbol{B}\in\mathrm{Eq}(i;X)\;\big|\;X=\boldsymbol{A}\big]\cdot\Pr[X=\boldsymbol{A}]\\ &= \textstyle\sum_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\Pr\big[\boldsymbol{B}\in\mathrm{Eq}(i;X)\;\big|\;X=\boldsymbol{A}\big]/(s+1)^r,\end{aligned}$$

which, since $\mathrm{Eq}(i;\boldsymbol{A}) = \mathrm{Eq}(i;\boldsymbol{B})$ iff $\boldsymbol{B}\in\mathrm{Eq}(i;\boldsymbol{A})$, equals

$$= \textstyle\sum_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\Pr\big[X\in\mathrm{Eq}(i;\boldsymbol{B})\;\big|\;X=\boldsymbol{A}\big]/(s+1)^r.$$

Now, X is a uniform random variable on a sample space of size $(s+1)^r$ and we know, from Part (i) of Observation 1, that $|\mathrm{Eq}(i;\boldsymbol{B})| = s+1$; hence, it follows that X takes on one of the $s+1$ values in $\mathrm{Eq}(i;\boldsymbol{B})$ with probability $(s+1)/(s+1)^r = 1/(s+1)^{r-1}$. Therefore,

$$\begin{aligned}\Pr\big[\boldsymbol{B}\in\mathrm{Eq}(i;X)\big] &= \textstyle\sum_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}}\big(1/(s+1)^{r-1}\big)/(s+1)^r\\ &= 1/(s+1)^{r-1}.\end{aligned}$$

□

We emphasize that the probability in Lemma 1 depends neither on the particular choice of $\boldsymbol{B}\in\mathbf{M}^{(r,s)}$ nor on $i\in[1..r]$. We are now ready to prove Theorem 3, which follows from Lemma 1 and Observation 1.

Proof of Theorem 3. Let I be the random variable describing the index $i \in [1..r]$ of the block the client seeks and, for each $j \in [1..\ell]$, let $Q_{\{j\}} \colon \mathbf{M}^{(r,s)} \to 2^{\varphi^{-1}(j)}$ be the random variable that associates the client's (uniform random) selection of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ with the subset of matrices $\boldsymbol{B} \in \varphi^{-1}(j)$ for which the client retrieves $\langle \boldsymbol{D}^*, \boldsymbol{B} \rangle_{\mathrm{F}}$ from bucket j as part of its query.

"If": The "if" direction is an easy corollary to Lemma 1 and Part (i) of Observation 1. Fix a bucket index $j \in [1..\ell]$ and note that, by construction, $Q_{\{j\}}(\boldsymbol{A}) = \mathrm{Eq}(I; \boldsymbol{A}) \cap \varphi^{-1}(j)$.

By Lemma 1, for any given $i \in [1..r]$ and $\boldsymbol{B} \in \varphi^{-1}(j)$, we have $\Pr[\boldsymbol{B} \in Q_{\{j\}} \mid I = i] = 1/(s+1)^{r-1}$; thus, it will suffice to show that $|Q_{\{j\}}(A)| \leq 1$. But this follows contrapositively from the condition $|\varphi(\mathrm{Eq}(i; \boldsymbol{A}))| = s + 1$, since $|Q_{\{j\}}| \geq 2$ would imply the existence of $\boldsymbol{B}, \boldsymbol{B}' \in \mathrm{Eq}(i; X)$ with $\boldsymbol{B} \neq \boldsymbol{B}'$ and $\varphi(\boldsymbol{B}) = \varphi(\boldsymbol{B}')$ so that $|\varphi(\mathrm{Eq}(i; X))| < |\mathrm{Eq}(i; X)| = s + 1$.

"Only if": The "only if" direction follows from Part (ii) of Observation 1. Suppose there is an $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ with $|\varphi(\mathrm{Eq}(i; \boldsymbol{A}))| < s + 1$. Then, by the Pigeonhole Principle, there must be a pair of distinct matrices $\boldsymbol{B}, \boldsymbol{B}' \in \mathrm{Eq}(i; \boldsymbol{A})$ such that $\varphi(\boldsymbol{B}) = \varphi(\boldsymbol{B}')$; so, set $j = \varphi(\boldsymbol{B})$ and let $q^* := Q_{\{j\}}(\boldsymbol{A})$ be the query string for server j arising when the client selects $\boldsymbol{A}$ in a query for $\vec{\boldsymbol{D}}_i$. By Part (ii) of Observation 1, we know that $\boldsymbol{B} \notin \mathrm{Eq}(i^*; \boldsymbol{B}')$ for any $i^* \neq i$; hence, we have $\Pr[Q_{\{j\}} = q^* \mid I = i] > 0$ and $\Pr[Q_{\{j\}} = q^* \mid I = i^*] = 0$, which proves that the construction is not perfectly $\{j\}$-private. □

Taken together, Theorems 1–3 imply that, given a suitable vector $\vec{\boldsymbol{v}}$ and mapping φ, the one-extra-word construction yields a perfectly 1-private PIR protocol. Moreover, as noted previously, setting $\mathbb{F} = \mathbf{GF}(2)$ yields a protocol having the lowest download cost possible. The next section discusses possible choices for φ.

4 "Bit-more-than-a-bit" protocols

We now describe our new "bit-more-than-a-bit" construction, a family of perfectly 1-private one-extra-word protocols parametrized by the number of buckets $\ell \geq 2$ and the number of words per block s. (Recall that the number of buckets equals the number of servers.) Each member of the bit-more-than-a-bit family uses a binary field $\mathbb{F} = \mathbf{GF}(2^w)$ where $w = \lceil \frac{b}{s} \rceil$, the all-0s vector $\vec{\boldsymbol{v}} = \vec{\boldsymbol{0}}$, and the mapping $\varphi \colon \mathbf{M}^{(r,s)} \to [1..\ell]$ defined in Equation (1) below.

The "bit-more-than-a-bit" mapping: The new mapping is much simpler than the one-extra-bit mapping described in Appendix B. It represents each matrix $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ as an r-digit integer expressed in radix $(s+1)$ using a "natural" bijection, and then it assigns each integer to a bucket according to its congruence class modulo ℓ. The mapping is given by

$$\varphi(\boldsymbol{A}) := \textstyle\sum_{i=1}^{r} (s+1)^{i-1} \, \mathrm{Ord}_i(\boldsymbol{A}) \bmod \ell, \tag{1}$$

where $\mathrm{Ord}_i(\boldsymbol{A}) = j$ iff $\mathrm{Row}_i(\boldsymbol{A}) = \vec{\boldsymbol{e}}_{j+1}$.

This mapping induces ℓ buckets, compared with $(s+1)^{r-1}$ buckets for the one-extra-bit mapping (cf. Equation (2)); hence, relative to the one-extra-bit construction, it reduces the number of servers required from $(b+1)^{r-1}$—which is super-exponential in $|\boldsymbol{D}|$—to an arbitrary constant $\ell \geq 2$.

A consequence of this super-exponential reduction in the number of buckets is that each bucket now contains about $(s+1)^r/\ell$ distinct w-bit Frobenius products. Specifically, the buckets now have a size that grows (at least formally) exponentially with the number of blocks comprising $\boldsymbol{D}$. Fortunately, because each Frobenius product is easily computed as a sum of just r-many words from $\mathbf{GF}(2^w)$, it suffices for the servers to store a copy of $\boldsymbol{D}$ and then construct needed Frobenius products on the fly. We discuss this idea further in Section 4.1.

Communication cost analysis (or "choosing a field size"): As each bit-more-than-a-bit instance is just a special instance of the one-extra-word construction, the analysis in Section 3 implies that the download cost (i.e., $|R|$) to fetch a b-bit block is just $(s+1)w \approx (b+w)$ bits. This suggests that setting w to be very small—for example, setting $w = 1$, as in the one-extra-bit construction—gives an ideal download cost.

Yet small constants like $w = 1$ lead to enormous bucket sizes and, thereby, fail to yield protocols that are non-trivial (in the sense of Definition 3). Indeed, to query for block i, the client must uniquely reference a specific Frobenius product from among the $\approx (s+1)^r/\ell$ such products held by each of the $s+1$ buckets in an i-equivalence class. To reference the Frobenius product $\langle \boldsymbol{D}^*, \boldsymbol{A} \rangle_{\mathrm{F}}$, the client uses an ordered pair (x, j) in which $x = \lfloor \frac{a}{\ell} \rfloor$ and $j = a \bmod \ell$ respectively denote the quotient and remainder obtained upon dividing $a := \sum_{i=1}^{r} (s+1)^{i-1} \mathrm{Ord}_i(\boldsymbol{A})$ by ℓ; hence, for any choice of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, the client sends $(r-1)\lceil \lg(s+1) \rceil$ bits to each of the $s+1$ servers in $\varphi(\mathrm{Eq}(i; \boldsymbol{A}))$. This gives an upload cost aggregated across all servers (i.e., $|Q|$) of $(s+1)(r-1)\lceil \lg(s+1) \rceil$ bits. Notably, if $w = 1$, then this is $(b+1)(r-1)\lceil \lg(b+1) \rceil \in \omega(|\boldsymbol{D}|)$ and the construction fails to provide non-trivial communication. To yield a non-trivial protocol it is therefore necessary to set w somewhat larger (thereby incurring higher download overhead). The next theorem characterizes the choices of w (and, consequently, s) that yield protocols providing non-trivial communication.

Theorem 4. *A bit-more-than-a-bit instance provides* non-trivial communication *(in the sense of Definition 3) if and only if $w \in \omega(1)$ or, equivalently, if and only if $s \in \mathrm{o}(b)$.*

In light of Theorem 4, we suggest choosing a desired (say, small constant) number of words per block $s \in \mathbb{N}$ and then selecting the field to be $\mathbf{GF}(2^w)$ for $w := \lceil \frac{b}{s} \rceil$. The next observation follows by inspection.

Observation 3. *Let ℓ and s be positive integers such that $\ell \geq 2$ and $s < \min(\ell, b+1)$. A bit-more-than-a-bit instance with ℓ buckets and s words per b-bit block has upload and download costs of $(s+1)(r-1)\lceil \lg(s+1) \rceil$ and $(s+1)w \approx \frac{s+1}{s} b$ bits, respectively.*

Notice that setting $s = 1$ yields a 2× download overhead (ignoring ceilings) with an upload cost linear in r—indeed, one can effectively view a protocol instance with $s = 1$ as a 2-server instance of the seminal PIR protocol for blocks due to Chor et al. [10, §6]. Increasing s decreases the download overhead *geometrically* while only increasing the upload cost *arithmetically* (ignoring the logarithmic term).

Computation cost analysis: As each Frobenius product is precomputed during the setup phase, the (online) computation cost for each server remains effectively nil. The client-side computation cost is s additions in $\mathbf{GF}(2^w)$, which equates to about b bit operations—one exclusive-OR for each bit in the requested block.

Security analysis: We establish sufficient conditions for the perfect 1-privacy of our bit-more-than-a-bit constructions in Theorem 5 below, which is an easy corollary to Theorem 3 and the following lemma.

Lemma 2. *Let s and ℓ be positive integers such that $s < \ell$ and $\gcd(s+1, \ell) = 1$. If a and b are distinct positive integers satisfying $a \equiv b \bmod \ell$, then the radix-$(s+1)$ representation of a and b differ in at least two digit positions.*

Proof. Assume without loss of generality that $a > b$. If a and b differ in just one digit position, then there must exist integers $c \in [1..s]$ and $d \geq 0$ such that $a - b = c(s+1)^d$. Now, since $a \equiv b \bmod \ell$, it follows that $\ell \mid c(s+1)^d$. But $\gcd(s+1, \ell) = 1$, so by a generalization of Euclid's Lemma to arbitrary integers, we have $\ell \mid c$, which contradicts the restriction that $c \in [1..s]$ and the fact that $s < \ell$. □

Theorem 5. *Let s and ℓ be positive integers such that $s < \ell$ and $\gcd(s+1, \ell) = 1$. A bit-more-than-a-bit instance with ℓ buckets and s words per block provides* perfect 1-privacy *(in the sense of Definition 4).*

Proof. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ be any pair of distinct matrices. From Lemma 2, $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ implies that $a \coloneqq \sum_{i=1}^{r} (s+1)^{i-1} \operatorname{Ord}_i(\boldsymbol{A})$ and $b \coloneqq \sum_{i=1}^{r} (s+1)^{i-1} \operatorname{Ord}_i(\boldsymbol{B})$ differ in at least two digits or, equivalently, that $\boldsymbol{A}$ and $\boldsymbol{B}$ differ in at least two rows. In turn, this implies that $\boldsymbol{B} \notin \operatorname{Eq}(i; \boldsymbol{A})$ for any $i \in [1..r]$. Since this is true for all distinct $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$, it follows that each element of $\operatorname{Eq}(i; \boldsymbol{A})$ resides in a different bucket; i.e., that $|\operatorname{Eq}(i; \boldsymbol{A})| = s+1$. Hence, by Theorem 3, it follows that the construction provides perfect 1-privacy. □

Combining Corollary 5 and Theorem 4 with the fact that $\|\vec{\mathbf{0}}\|_1 = 0$ (cf. Theorem 1) yields the following theorem.

Theorem 6. *Let s and ℓ be positive integers such that $s < \ell$ and $\gcd(s+1, \ell) = 1$. A bit-more-than-a-bit instance with ℓ buckets and s words per block is a* perfectly 1-private ℓ-server PIR protocol. *Moreover, the protocol has download rate $\frac{s+1}{s} + o(1)$.*

We note that, in contrast to the one-extra-bit construction, our bit-more-than-a-bit constructions maintain perfect privacy in the face of "traffic analysis" attacks (cf. Appendix C). Indeed, if we let X denote the random variable describing which of the ℓ servers receive no message in a given query, then for all $i \in [1..r]$ and $j \in [1..\ell]$, we find that $\Pr[j \in X \mid I = i] = 1 - |\varphi^{-1}(j)|/(s+1)^{r-1}$. The latter probability is a consequence of (i) the one-to-one correspondence between queries for block i and equivalence classes under $\equiv_i$, and (ii) the fact that server j holds exactly one element from each of $|\varphi^{-1}(j)|$ out of $(s+1)^{r-1}$ such equivalence classes.

4.1 Virtual buckets

Recall that each bucket in a bit-more-than-a-bit instance having s words per block contains about $(s+1)^r/\ell$ distinct w-bit Frobenius products, a quantity that scales exponentially with the number of blocks in $\boldsymbol{D}$. Fortunately, because each Frobenius product is easily computed as a sum of just r-many words from $\mathbf{GF}(2^w)$—i.e., one word from each row of $\boldsymbol{D}^*$—it suffices for a server to store a copy of $\boldsymbol{D}$ and then construct needed Frobenius products on the fly. In fact, having all servers store $\boldsymbol{D}$ eliminates the need to associate each server with a fixed bucket, thereby eliminating the requirement from Theorems 5 and 6 that $\gcd(s+1, \ell) = 1$. Indeed, with this approach, we are free to set $\ell = s+1$ so that the client simply involves all servers in all queries. For this reason, we say that a server who stores all of $\boldsymbol{D}$ and computes arbitrary Frobenius products on the fly holds a *virtual bucket* of $\boldsymbol{D}$.

Virtual buckets fix the per-server storage cost as $|\boldsymbol{D}|$ bits, while increasing the online computation cost to at most r additions in $\mathbf{GF}(2^w)$, each of which is implemented as a bitwise exclusive-OR of w-bit binary strings. However, on average a fraction $1/(s+1)$ of the rows of $\boldsymbol{A}$ contain $\vec{\boldsymbol{e}}_{s+1}$, which, because we set $\vec{\boldsymbol{\nu}} = \vec{\mathbf{0}}$, induces a no-op; hence, the *expected* computation cost per server is closer to $\frac{rs}{s+1}$ additions in $\mathbf{GF}(2^w)$. In particular, the expected computation cost aggregated across all servers is about rs additions in $\mathbf{GF}(2^w)$, or about one exclusive-OR per bit in $\boldsymbol{D}$, with each of $s+1$ participating servers shouldering a roughly equal computational burden, and the *worst-case* computation cost is only nominally higher. This compares favorably with other ℓ-server protocols in the literature; indeed, Beimel, Ishai, and Malkin [4, Theorem 6.4]

proved that using *even one fewer bit operation* in expectation is impossible without increasing the storage cost by having each server store extra, pre-processed information about $\boldsymbol{D}$.

Observation 4. *Suppose $\ell - 1 \mid b$. A bit-more-than-a-bit construction with ℓ virtual buckets and $s = \ell - 1$ words per b-bit block has download cost of exactly $(1 + \frac{1}{\ell-1})b$ bits.*

The $(1 + \frac{1}{\ell-1})b$-bit download cost from Observation 4 matches the best possible download for any ℓ-server PIR protocol, provided that r is sufficiently large [6, Theorem 2.5].

5 Computational 1-privacy

This section presents our most efficient construction, a *computationally* 1-private variant of the bit-more-than-a-bit family of PIR protocols with virtual buckets. The simplest form of this computationally 1-private construction, which can be instantiated with $\ell = 2^L$ servers for any positive integer L, reduces the per-server upload cost from rL bits in the perfectly 1-private bit-more-than-a-bit construction to just $(\lambda + 2)\lceil \lg \frac{r}{\lambda} \rceil L$ bits, where λ is a security parameter (say, $\lambda = 128$ or 256). The download cost remains unchanged and the computation cost increases only modestly in practice (assuming both clients and servers are equipped with modern CPUs that support the AES-NI instruction set).

The computationally 1-private construction replaces the *uniform random* query strings from our perfectly 1-private construction with *pseudorandom* query strings generated by an L-tuple of 2-*out-of*-2 *distributed point functions* [17], each mapping integers from $[1..r]$ to 1-bit outputs. A *point function* in this context refers to a function $p_i : [1..r] \to \{0,1\}$ such that

$$p_i(i^*) = \begin{cases} 1 & \text{if } i^* = i, \text{ and} \\ 0 & \text{otherwise;} \end{cases}$$

while one can think of a *distributed* point function as an extremely compact, secret-shared representation of a point function.

We provide the following formal definition for a 2-out-of-2 distributed point function with 1-bit outputs, which we adapt from Definition 2.2 of Boyle, Gilboa, and Ishai [8, §2] and Definition 1 of Gilboa and Ishai [17, §2].

Definition 7. *Let $\mathbb{P}(r)$ denote the set of all point functions $p_i : [1..r] \to \{0,1\}$ with domain $[1..r]$ and 1-bit outputs. A pair of probabilistic polynomial-time algorithms* (Gen, Eval) *with syntax*

- $\mathsf{Gen}(1^\lambda; i)$*, taking $\lambda \in \mathbb{N}$ and $i \in [1..r]$, outputs a* DPF key pair*, $(k_0, k_1) \in \{0,1\}^* \times \{0,1\}^*$; and*
- $\mathsf{Eval}(k, i^*)$*, taking $k \in \{0,1\}^*$ and $i^* \in [1..r]$, outputs a bit,*

is a 2-out-of-2 distributed point function*, or* $(2,2)$-DPF*, for $\mathbb{P}(r)$ if it provides the following two guarantees:*

(i) ***(Perfect) correctness****: For all inputs $i, i^* \in [1..r]$, if $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i)$, then*

$$\mathsf{Eval}(k_0, i^*) \oplus \mathsf{Eval}(k_1, i^*) = p_i(i^*).$$

(ii) ***(Computational) secrecy****: There exists a PPT algorithm* Sim *such that the distribution ensembles*

$$\{\mathsf{Sim}(1^\lambda; b)\}_{b \in \{0,1\}}$$

and

$$\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i)\}_{b \in \{0,1\}, i \in [1..r]}$$

are computationally indistinguishable.

The following (rather trivial) lemma will be convenient for proving that our DPF-based query construction provides computational 1-privacy (à la Definition 5), which is an indistinguishability-based (rather than simulation-based) notion of privacy.

Lemma 3. *If* (Gen, Eval) *is a* $(2,2)$*-DPF for $\mathbb{P}(r)$, then for all $i, i^* \in [1..r]$ and for all $b \in \{0,1\}$, the distribution ensembles*

$$\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i)\}_{b \in \{0,1\}}$$

and

$$\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i^*)\}_{b \in \{0,1\}}$$

are computationally indistinguishable.

Proof (sketch). This follows immediately from the definition of computational secrecy for a $(2,2)$-DPF (Part (ii) of Definition 7) and the triangle inequality. □

Our construction uses the efficient $(2,2)$-DPF construction of Boyle, Gilboa, and Ishai [8, Figure 1] with 1-bit outputs. The computational secrecy of that construction relies only on the existence of a fixed-expansion pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$; a computationally efficient candidate for such a PRG is easily constructed from any block cipher. A typical choice for concrete instantiations is to use 128- or 256-bit AES (hence our reliance on CPUs with the AES-NI instruction set for efficiency in practice).

The efficiency of our construction also benefits from the fast *full-domain evaluation* algorithm of Boyle et al. [8, §3.2.1], which expands a $(2,2)$-DPF key k into the length-r bit vector

$$\mathsf{EvalFull}(k) := \langle \mathsf{Eval}(k, 1), \mathsf{Eval}(k, 2), \ldots, \mathsf{Eval}(k, r) \rangle$$

using just $\lceil r/\lambda \rceil$ PRG evaluations [7, Theorem 3.3].

We also introduce the '$[\!]$' operator to denote the component-wise concatenation of length-r vectors over $\{0,1\}^*$; in particular, if $\vec{u} = \langle u_0, \ldots, u_n \rangle$ and $\vec{v} = \langle v_0, \ldots, v_n \rangle$, then

$$\vec{u} \,[\!]\, \vec{v} := \langle u_0 \| v_0, \ldots, u_n \| v_n \rangle,$$

where $v_i \| u_i$ denotes concatenation of u_i and v_i.

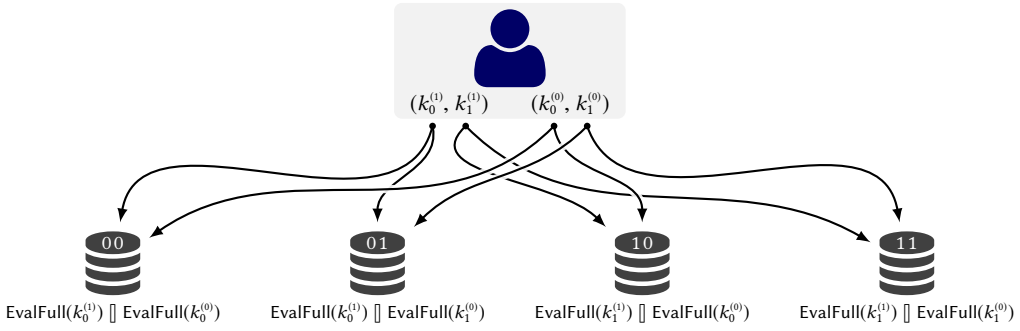


Fig. 1. $(2,2)$-DPF key distribution and query expansion procedure for $\ell = 2^2$ servers.

5.1 DPF-based computational 1-privacy

We are now ready to describe our computationally 1-private query construction. As previously explained, the objective of the construction is to leverage $(2,2)$-DPFs to provide extremely compact representations of the client's query strings in an otherwise standard bit-more-than-a-bit instantiation.

The main technical challenge to overcome is the impedance mismatch between the "1-out-of-2" secrecy of the $(2,2)$-DPFs and the desired "1-out-of-ℓ" privacy of the PIR queries. We resolve this by using an L-tuple of $(2,2)$-DPF key pairs (all realizing the same underlying point function), which the client samples independently and distributes to the servers using a strategy reminiscent of Beimel and Stahl's generic transformation for 2-out-of-ℓ "robust" PIR from any 2-server PIR protocol [5, §3.1]. We first examine the simplest and most efficient case in which $\ell = 2^L$ for some $L \geq 1$; an extension to the case of arbitrary ℓ follows in Section 5.3.

5.2 Power-of-2 number of servers

Query construction (aka "DPF key distribution"): The DPF-based query construction for $\ell = 2^L$ servers works as follows. Assign to each of the ℓ servers a numeric label j between 0 and $\ell - 1$ and then, for each $j \in [0\,..\,\ell-1]$, consider the L-bit binary representation $(j_{L-1} \cdots j_1 j_0)_2$ of j, where j_e denotes the eth-least-significant bit of j. To query for block $\vec{D}_i$, the client samples L independent $(2,2)$-DPF key pairs for the point function p_i, say

$$(k_0^{(L-1)}, k_1^{(L-1)}), \ldots, (k_0^{(0)}, k_1^{(0)}) \leftarrow \mathsf{Gen}(1^\lambda;\, i) \times \cdots \times \mathsf{Gen}(1^\lambda;\, i),$$

and then, to each server $j \in [0\,..\,\ell-1]$, it sends the query string

$$q_j := \big(k_{j_{L-1}}^{(L-1)}, \ldots, k_{j_0}^{(0)}\big),$$

where, as above, j_e is the eth-least-significant bit of j.

Because each server receives *either* the 0th or 1th key from each DPF key pair, and because *which* key a given server receives is determined by the corresponding bit in its label, it follows that (i) no server receives both keys from a single DPF key pair, and (ii) no two servers receive the same sequence of DPF keys.

Query expansion: Upon receiving the query string q_j from the client, server j parses it as a sequence of DPF keys

$$q_j := \big(k^{(L-1)}, \ldots, k^{(0)}\big),$$

and then it performs a full-domain evaluation on each of the keys and concatenates the resulting bit vectors componentwise to obtain a length-r vector of L-bit integers; that is, it computes

$$\tilde{q}_j := \mathsf{EvalFull}(k^{(L-1)}) \,▯\, \cdots \,▯\, \mathsf{EvalFull}(k^{(0)}).$$

From here, the server proceeds exactly as it would upon receiving the query string $\tilde{q}_j$ directly from the client in a perfectly 1-private bit-more-than-a-bit instance with virtual buckets.

Figure 1 illustrates the DPF key distribution and query expansion procedure for a protocol instance involving $\ell = 2^2$ servers. In that figure, the ordered pairs $(k_0^{(1)}, k_1^{(1)})$ and $(k_0^{(0)}, k_1^{(0)})$ are both $(2,2)$-DPF key pairs sampled independently using $\mathsf{Gen}(1^\lambda;\, i)$.

Correctness analysis: We now analyze the correctness of the computationally 1-private bit-more-than-a-bit construction. In particular, Theorem 7 establishes that the vectors $\tilde{q}_0, \ldots, \tilde{q}_{\ell-1}$ produced by the ℓ servers indeed correspond to an i-equivalence class of $\mathbf{M}^{(r,s)}$ where $s = \ell - 1$.

Theorem 7. *If* $\big(k_0^{(L-1)}, k_1^{(L-1)}\big), \ldots, \big(k_0^{(0)}, k_1^{(0)}\big) \leftarrow \big(\mathsf{Gen}(1^\lambda;\, i)\big)^L$ *is an* L*-fold sequence of* $(2,2)$*-DPF key pairs for a point function* $p_i \colon [1\,..\,r] \to \{0,1\}$ *and if* $i^* \in [1\,..\,r]$*, then for any pair of distinct* L*-bit binary strings* $w = (w_{L-1}\, w_{L-2} \cdots w_0)_2$ *and* $x = (x_{L-1}\, x_{L-2} \cdots x_0)_2$*, the vectors*

$$\vec{\boldsymbol{w}} := \mathsf{EvalFull}(k_{w_{L-1}}^{(L-1)}) \,▯\, \cdots \,▯\, \mathsf{EvalFull}(k_{w_0}^{(0)})$$

and

$$\vec{\boldsymbol{x}} := \mathsf{EvalFull}(k_{x_{L-1}}^{(L-1)}) \,▯\, \cdots \,▯\, \mathsf{EvalFull}(k_{x_0}^{(0)})$$

differ in column i^* *if and only if* $i^* = i$.

Proof. We prove the "if" and "only if" directions separately.

"If": Let e be a bit position at which $w_e \neq x_e$. From the correctness criterion for a $(2,2)$-DPF, we have $\mathsf{Eval}(k^{(e)}_{w_e}, i) \oplus \mathsf{Eval}(k^{(e)}_{x_e}, i) = 1$. Hence, $\mathsf{Eval}(k^{(e)}_{w_e}, i) \neq \mathsf{Eval}(k^{(e)}_{x_e}, i)$ and it follows that the ith components of $\vec{\boldsymbol{w}}$ and $\vec{\boldsymbol{x}}$ differ in at least one bit.

"Only if": Suppose that $\vec{\boldsymbol{w}}$ and $\vec{\boldsymbol{x}}$ differ in column i^* and let $e \in [0\,..\,L-1]$ be the position of a bit at which they differ; i.e., let e be such that $\mathsf{Eval}(k^{(e)}_{w_e}, i^*) \oplus \mathsf{Eval}(k^{(e)}_{x_e}, i^*) = 1$. Then, from the correctness criterion of a $(2,2)$-DPF, $p_i(i^*) = 1$ so that $i^* = i$. □

The preceding theorem together with a simple counting argument yields the following corollary.

Corollary 2. *If* $\big(k^{(L-1)}_0, k^{(L-1)}_1\big), \ldots, \big(k^{(0)}_0, k^{(0)}_1\big) \leftarrow \big(\mathsf{Gen}(1^\lambda; i)\big)^L$ *is an L-fold sequence of $(2,2)$-DPF key pairs for a point function $p_i : [1\,..\,r] \to \{0,1\}$, then the ith components of the 2^L vectors*

$$\big\{\mathsf{EvalFull}(k^{(L-1)}_{j_{L-1}}) \,\|\, \cdots \,\|\, \mathsf{EvalFull}(k^{(0)}_{j_0}) \;\big|\; j_{L-1}, \ldots, j_0 \in \{0,1\}^L\big\}$$

collectively span the interval $[0\,..\,2^L - 1]$.

The next theorem is an immediate consequence of Corollary 2 and the "only if" direction of Theorem 7.

Theorem 8. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a $(2,2)$-DPF for $\mathbb{P}(r)$ providing* (perfect) correctness *(in the sense of Part (i) of Definition 7), then the 2^L-server DPF-based bit-more-than-a-bit construction provides* (perfect) correctness *(in the sense of Definition 2).*

Communication cost analysis: The download cost of a DPF-based bit-more-than-a-bit protocol instance is identical to that of a perfectly 1-private instance with the same number of virtual buckets. In terms of upload cost, the client in a DPF-based bit-more-than-a-bit instance sends L distinct DPF keys to each server. Assuming that the DPFs are implemented with the optimized $(2,2)$-DPF construction of Boyle, Gilboa, and Ishai [8, Figure 1] with 1-bit outputs, the total upload cost is then $L(\lambda + 2)(\lceil \lg \frac{r}{\lambda} \rceil)$ bits [7, Theorem 3.3]—in contrast to the $r\lceil \lg(s+1) \rceil = rL$ bits needed for the perfectly 1-private construction with $\ell = 2^L$ buckets and $s = \ell - 1$ words per block—and the following theorem is immediate.

Theorem 9. *The 2^L-server DPF-based bit-more-than-a-bit construction provides* non-trivial communication *(in the sense of Definition 3).*

Computation cost analysis: The DPF-based bit-more-than-a-bit construction imposes some (modest) computation overhead relative to a perfectly 1-private bit-more-than-a-bit instance. The overhead consists of (i) the cost for the client to sample the DPF key pairs, and (ii) the cost for each server to expand its DPF keys using a full-domain evaluation (plus nominal costs associated with component-wise concatenation). Sampling the DPF key pairs is an extremely lightweight operation: each sample takes about $2\lceil \lg \frac{r}{\lambda} \rceil$ evaluations of a PRG and about $2\lambda\lceil \lg \frac{r}{\lambda} \rceil$ additional bit operations [7, Figure 1]. The full-domain evaluations for query expansion are also quite efficient: each uses just $\lceil \frac{r}{\lambda} \rceil$ evaluations of the PRG [7, Theorem 3.3]. All told, when the PRG is implemented using AES-128, this equates to a total of about $4L(\lceil \lg r \rceil - 7)$ AES-128 evaluations and $256L(\lceil \lg r \rceil - 7)$ additional bit operations for the client, and about $L\lceil \frac{r}{64} \rceil$ AES-128 evaluations for each server.

In Section 6, we present experimental evidence from our prototype implementation. Our findings suggest that, for plausible parameter settings, the computation overhead associated with sampling DPF key pairs and query expansion are insignificant relative to the time required to compute the Frobenius products specified by the expanded query vectors. In fact, the DPF-associated computation times are low enough that we can reasonably surmise that reductions in upload time will vastly overshadow them for all but the most contrived of network settings and parameter choices.

Security analysis: A trivial reduction proves the computational 1-privacy of a DPF-based bit-more-than-a-bit construction follows from the perfect 1-privacy of regular bit-more-than-a-bit and the computational secrecy (Part (ii) of Definition 7) of the DPFs. In particular, given blackbox oracle access to a PPT algorithm $\mathcal{A}$ distinguishing between the distribution ensembles $\{Q_{\{j\}} \mid I = i\}$ and $\{Q_{\{j\}} \mid I = i^*\}$ with advantage $\mu(\lambda)$, we obtain a PPT algorithm $\mathcal{B}$ distinguishing between the distribution ensembles $\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i)\}_{b \in \{0,1\}}$ and $\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i^*)\}_{b \in \{0,1\}}$ with the same advantage (cf. Definition 5 and Lemma 3). The reduction $\mathcal{B}$ works by (i) sampling L DPF keys from its distribution, (ii) concatenating the L samples to form q_j, (iii) passing q_j to $\mathcal{A}$, and then (iv) returning whatever $\mathcal{A}$ returns. This reduction and Theorems 8 and 9 proves the following.

Theorem 10. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a $(2,2)$-DPF for $\mathbb{P}(r)$ providing* computational secrecy *(in the sense of Part (ii) of Definition 7), then a 2^L-server DPF-based bit-more-than-a-bit instance is a* computationally 1-private 2^L-server PIR protocol.

5.3 Arbitrary number of servers

We now describe how to extend the computationally 1-private bit-more-than-a-bit protocol to an arbitrary number of servers, at the expense of some additional upload and computation overhead. The challenge to overcome is the existence of

so-called *modulo bias* introduced when reducing integers distributed uniformly in $[1..2^L]$ modulo ℓ, when $\ell \nmid 2^L$.

Query construction (aka "DPF key distribution"): Let $poly\colon \mathbb{N} \to \mathbb{N}$ be some positive integer-valued polynomial. The basic idea behind the extension is quite simple: Set $L = \lceil \lg \ell \rceil + poly(\lambda)$ and, as before, have the client sample an L-fold sequence of DPF key pairs and then send a query string q_j comprising one DPF key from each key pair to each server $j \in [0..\ell-1]$. Note that although there are just ℓ servers, there are more than $\ell + 2^{poly(\lambda)}$ possible combinations of DPF keys from which the client may choose. The client chooses which sequences of DPF keys to send to each server *uniformly*, subject to the values in column i of the ℓ resulting vectors being pairwise incongruent modulo ℓ. The additional $poly(\lambda)$ DPF keys serve to "smooth out" the distribution, reducing the magnitude of the modulo bias to a negligible level; thus, we refer to $poly(\lambda)$ as the *smoothing parameter*. We stress that the smoothing parameter need not be large: Formally proving computational privacy requires only that $poly(\lambda) \in \Omega(\lambda^\epsilon)$ for some $\epsilon > 0$; in practice it is sufficient to regard $poly(\lambda)$ as a small(ish) constant, say $poly(\lambda) = 64$ or 80.

Query expansion: Upon receiving the query string q_j from the client, server j parses it as a sequence of DPF keys and then performs L full-domain evaluations and an L-ary component-wise concatenation to obtain a length-r vector of L-bit strings (which it regards as integers). From here, server j reduces the vector component-wise modulo ℓ to obtain the desired length-r vector of integers in $[0..\ell-1]$.[6]

Correctness analysis: The "if" direction of Theorem 7 guarantees that the servers generate vectors of L-bit integers containing identical values in all but the ith column; thus, after reduction modulo ℓ the vectors remain identical in all but (perhaps) the ith column. Furthermore, because the client selects DPF keys so as to ensure that the entries in column i at the servers are pairwise incongruent modulo ℓ, the values in column i remain pairwise distinct after reduction modulo ℓ. This proves the following theorem.

Theorem 11. *If* (Gen, Eval) *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$ *providing* (perfect) correctness *(in the sense of Part (i) of Definition 7), then the* ℓ*-server DPF-based bit-more-than-a-bit construction provides* (perfect) correctness *(in the sense of Definition 2).*

Communication and computation cost analysis: Relative to the 2^L-server construction, the construction for arbitrary ℓ increases upload cost and computation cost associated with sampling DPF key pairs and performing full-domain evaluations each by a factor of approximately $\big(\ell + poly(\lambda)\big)/\ell$; the download cost is unaffected. Additionally, each server must perform a component-wise reduction modulo ℓ, although in implementations it is possible to interleave this operation with the component-wise concatenation so that it introduces essentially no overhead (at most one conditional subtraction) per DPF key.

Theorem 12. *The* ℓ*-server DPF-based bit-more-than-a-bit construction provides* non-trivial communication *(in the sense of Definition 3).*

Security analysis: In terms of privacy, the ℓ-server construction is nearly identical to the 2^L-server construction, with one notable exception: the expanded query vectors in the ℓ-server construction *exhibit a (small) statistical bias*. Specifically, writing $2^L = Q\cdot\ell + R$ with $0 \le R < \ell$ using the Division Algorithm, each integer $j \in [0..R-1]$ is congruent to $Q+1$ integers from $[0..2^L-1]$, while each $j \in [R..\ell-1]$ is congruent to just Q integers from $[0..2^L-1]$; thus, the probability mass function describing each component of an expanded query vector assigns a probability of $(Q+1)/2^L$ to each value in $[0..R-1]$ and $Q/2^L$ to each value in $[R..2^L-1]$—except for column i, where the distribution is uniform (since clients choose the DPF key sequences uniformly). Fortunately, as the smoothing parameter $poly(\lambda)$ guarantees that the statistical distance between these two distributions is bounded above by $2^{-poly(\lambda)}$, we still obtain following theorem.

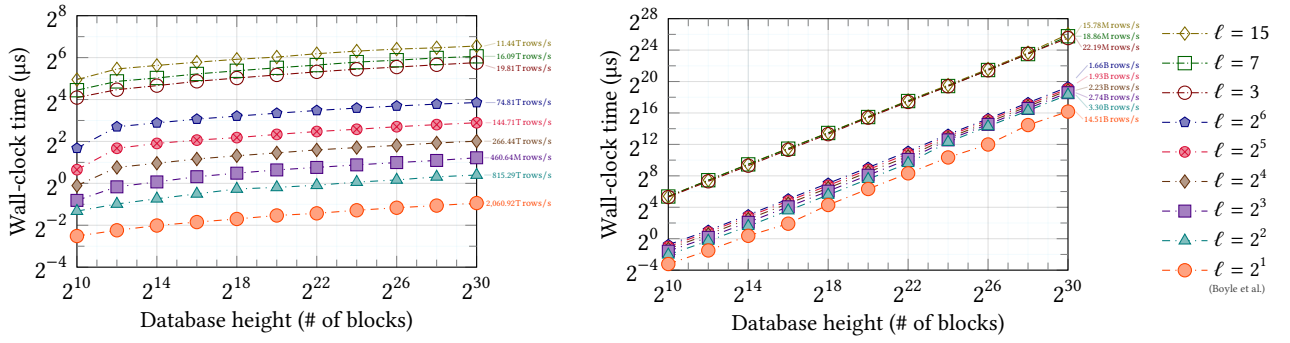
Theorem 13. *If* (Gen, Eval) *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$ *providing* computational secrecy *(in the sense of Definition 7) and if* $poly(\lambda) \in \Omega(\lambda^\epsilon)$ *for some* $\epsilon > 0$*, then the* ℓ*-server DPF-based bit-more-than-a-bit construction provides* computational 1-privacy *(in the sense of Definition 4).*

6 Implementation & evaluation

To validate the efficiency of the constructions presented in Sections 4 and 5, we have implemented them as an open-source C++ library called libbitmore [23], which we built atop our own dpf++ library [22]. To facilitate a fair head-to-head comparison with the "folklore optimal" protocols, our codebase also incorporates hand-optimized implementations of both the 2-server variant of Chor et al.'s protocol [10] and the basic 2-server DPF-based protocol incorporating all optimizations suggested by Boyle et al. [7].[7] We also compare

6 This is always possible as (i) Corollary 2 ensures the 2^L possible sequences of DPF keys collectively span $[0..2^L-1]$, and (ii) $[0..\ell-1] \subseteq [0..2^L-1]$.

7 Although the new constructions include the latter two protocols as special cases, our standalone implementations are slightly faster than fixing $\ell = 2$ in our implementation of the general constructions.



(a) DPF key sampling (client).

(b) DPF key expansion (server).

Fig. 2. Wall-clock time for client-side DPF key sampling (a) and server-side DPF key expansion (b) as the database height grows. All experiments use $\lambda = 128$; for computationally 1-private protocols with ℓ not a power of 2, we set the smoothing parameter to $poly(\lambda) = 80$.

our implementation with Percy++ v1.0 [19] and RAID-PIR v0.9.5 [13], a pair of open-source multiserver PIR libraries that are widely used as benchmarks in the PIR research community.

The remainder of this section presents selected findings from a pool of experiments designed to measure the performance implications of various parameter choices and resulting speedups relative to prior work. We conducted all experiments on a workstation running Ubuntu 18.04.2 LTS on an AMD Ryzen 7 2700x eight-core CPU @ 4.30 GHz with 16 GiB of RAM and a 1 TB NVMe M.2 SSD. (We had this workstation custom-built for CA$1050 in April 2019.) All experiments measured the running times for a single client or server instance running in isolation on a single CPU core. We repeated all experiments for 100 trials and report herein the sample mean (wall-clock) timings across all trials. With the exception of RAID-PIR, all experiments exhibited sample standard deviations that were small relative to the sample means; thus, we omit error bars in our plots to reduce clutter and improve visual clarity. When discussing selected statistics in the text, we write $M \pm s$ to indicate a sample mean of M and sample standard deviation of s. We report all such statistics to one digit of precision in the sample standard deviation.

6.1 DPF key sampling and expansion

Our first set of experiments measures the time required for DPF key sampling (by the client) and expansion (by the server) for various numbers of servers (ℓ) and database heights (r) in our computationally 1-private protocols. We provide log-log plots of our findings in Figure 2. We emphasize that the costs for DPF key sampling and expansion do *not* depend on the bitlength (b) of the database blocks. As expected, the costs for both operations are consistently low, even for databases with relatively large numbers (i.e., hundreds of millions) of blocks—notice that the y-axis on both plots measures wall-clock time in *microseconds*.

DPF key sampling: For $\ell = 2^L$, the running times reported in our DPF key sampling plot (Figure 2a) consist only of sampling L independent $(2, 2)$-DPF key pairs. For ℓ not a power of 2, they also include the time required for (i) the sampling of $poly(\lambda) = 80$ extra DPF key pairs for smoothing and (ii) the uniform random selection of key sequences to satisfy the requisite pairwise-incongruence property (see Section 5.3); consequently, the running times for these experiments are an order of magnitude higher than for $\ell = 2^L$. Nonetheless, all experiments complete in *under 100 microseconds*.

DPF key expansion: The DPF key expansion plot (Figure 2b) reports the time required for both the full-domain evaluation of each DPF key and the subsequent component-wise concatenation (including modular reductions).

We implement the component-wise concatenation using our own algorithms based on 256-bit vectorized (AVX2) instructions. For $\ell = 2^L$, our algorithm iterates over the bits obtained via full-domain evaluation of each DPF key in 32-bit increments, using AVX2 intrinsics to expand each 32-bit segment into a 32-byte vector type. It then uses simple (vectorized) masking and bitwise logical ORs to effectively concatenate 32 components in parallel. This yields an order-of-magnitude speedup versus an "obvious" implementation of component-wise concatenation, helping to ensure that DPF key expansion contributes insignificantly to the servers' total computation time; however, it also limits our implementation to at most $L = 8$ keys per server, thus imposing a limit of $\ell = 256$ on the total number of servers.

Our algorithm for ℓ not a power of 2 is similar, except it periodically performs (parallel) partial modulo-ℓ reductions and uses lookup tables to simulate expanding each bit in the full-domain evaluations beyond the architecture-imposed 1-byte boundary. This approach significantly reduces the overhead from modular reductions, though it imposes an even stricter limit of $\ell = 127$ on the number of servers (to avoid overflowing bytes between partial reductions).

As expected, the costs for DPF key expansion (Figure 2b) are significantly higher than for sampling (Figure 2a); indeed, the cost for DPF key expansion scales with r, whereas DPF key sampling scales with $\lg r$. Nonetheless, all experiments with $\ell = 2^L$ complete in *under 1 second* and all experiments with ℓ not a power of 2 still complete within *about 1 minute*. Focusing on a "modestly" sized database with just $r = 2^{20}$ rows, we find that all experiments for $\ell = 2^L$ complete within *just over half a millisecond* and all experiments with ℓ not a power of 2 complete in *well under 50 milliseconds*.[8]

Looking ahead to Figure 3, we observe (perhaps surprisingly) that in all cases DPF key expansion ends up being (slightly) faster than sampling a pseudorandom query of the same length using `arc4random`; thus, in addition to significantly reducing upload costs (e.g., from 0.5 GiB to just 31.66 KiB per server when $\ell = 15$ and $r = 2^{30}$), the computationally 1-private protocols actually have a modestly lower aggregate computation cost relative to their perfectly 1-private counterparts! Looking even further to Figure 4, we conclude that DPF key sampling and expansion contribute insignificantly to the total computation cost of a PIR query.

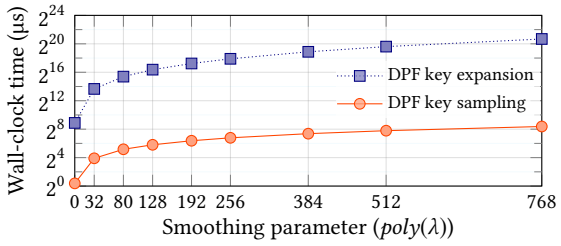


Fig. 3. Wall-clock time for client-side DPF key sampling and server-side expansion as the smoothing parameter grows. All experiments fix $\ell = 3$ and $r = 2^{20}$.

Varying the "smoothing parameter": Note that the plots in Figure 2 all fix the smoothing parameter as $poly(\lambda) = 80$. We also conducted a series of experiments to study the implications of varying $poly(\lambda)$, from $poly(\lambda) = 0$ up to $poly(\lambda) = 6\lambda$ (i.e., from $poly(128) = 0$ up to $poly(128) = 768$) for a fixed number of servers ($\ell = 3$) and database height ($r = 2^{20}$). We provide a log-linear plot of the results in Figure 3. The results are unsurprising: increasing $poly(\lambda)$ effects a corresponding increase in DPF key sampling and expansion times, with key expansion times ranging from 0.470 ± 0.006 ms when $poly(\lambda) = 0$ up to 1681 ± 3 ms when $poly(\lambda) = 768$. In a practical deployment, we recommend setting $poly(\lambda)$ between 64 and 96 to strike a good balance between performance and security, which in these experiments yielded expansion times from 31.4 ± 0.1 ms up to 55.4 ± 0.2 ms.

In terms of upload cost, setting $poly(\lambda) = 0$ yields queries of size 0.41 KiB, while $poly(\lambda) = 80$ yields queries of size 16.92 KiB and setting $poly(\lambda) = 768$ yields 158.85 KiB. Thus, even for "unreasonably" large values of the smoothing parameter, DPF sampling times, DPF expansion times, and per-server upload costs remain quite modest for a database having on the order of a few millions of blocks.

6.2 Computational vs. perfect privacy

Our second set of experiments compares the cost of (client-side) query generation for DPF-based computationally 1-private protocol instances versus perfectly 1-private protocol instances as the database height (r) grows. We provide a log-log plot of our findings in Figure 4. For $\ell = 2^L$, perfectly 1-private query generation is implemented using a single call to `arc4random_buf`, relying on the Linux kernel to provide the required number of pseudorandom bits as efficiently as it knows how. For ℓ not a power of 2, it is implemented using r consecutive calls to `arc4random_uniform`, which involves a modular reduction and the occasional re-sampling (to account for modulo bias) for each query component; thus, similar to the computationally 1-private protocol, we observe a notable performance penalty when ℓ not a power of 2.

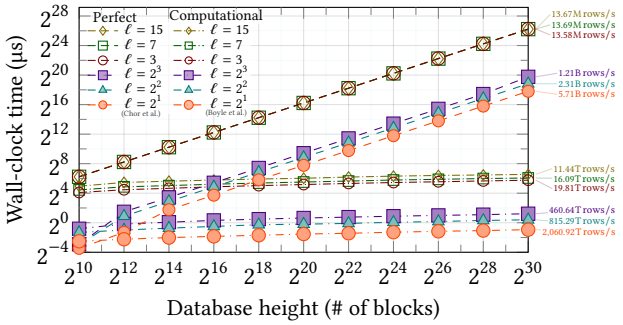


Fig. 4. Wall-clock time for client-side query construction in both computationally and perfectly 1-private protocols. For computationally 1-private instances with ℓ not a power of 2, we set the smoothing parameter to $poly(\lambda) = 80$.

As expected, query generation times in the computationally 1-private protocols scale quite well (indeed, logarithmically) relative to query generation in the perfectly 1-private protocols. Indeed, as noted in Section 6.1, even the linear-cost DPF key expansion consistently outperforms query sampling (by a factor 2–3× for $\ell = 2^L$ and by 1.15–1.63× for ℓ not a power of 2). We conclude that, for any realistic parameter settings, our computationally 1-private protocol is strictly faster than its perfectly 1-private counterpart.

8 Astute readers may notice that, although the cases of $\ell = 3$, 7, and 15 respectively expand and concatenate 82, 83, and 84 DPF keys, the differences in their throughputs are nontrivial (ranging from 22.192 ± 0.006M rows/s for $\ell = 3$ to 15.78 ± 0.01M rows/s for $\ell = 15$). This larger-than-expected difference owes to the frequency with which our implementation must perform partial modular reductions to avoid overflowing bytes, which is dependent on the bit length of the modulus.

6.3 Response generation

Our third set of experiments compares the time required for each of $\ell = 2^L$ servers to respond to a query, for various choices of L and a fixed number of blocks ($r = 256$), as the database width (b) grows. We measure here only the time required to respond to an "expanded" query; thus, we do not distinguish between perfectly and computationally 1-private protocols, which perform identically (indeed, run the same code on inputs from computationally indistinguishable distributions) for this step of the protocol. We emphasize that response generation is far and away the most computationally intensive step in all constructions we consider (notice that the y-axis on Figure 5 is in *milliseconds* versus *microseconds* in all the other plots); indeed, response-generation speedups from increasing ℓ dwarf the comparatively tiny query costs in Figures 2–4. When $\ell = 2^1$ (i.e., in the "folklore" protocols), the servers process the database at a rate of 5.7 ± 0.5 GiB/s, whereas when $\ell = 2^6$ this throughput increases to a whopping 144 ± 3 GiB/s—more than a 25× speedup!

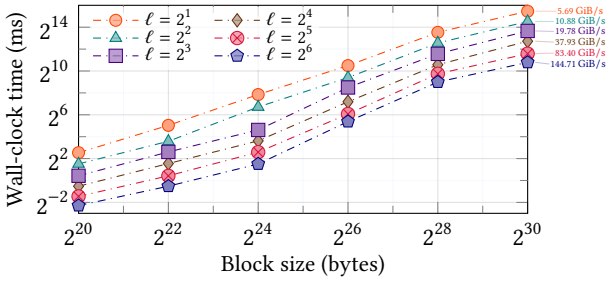


Fig. 5. Wall-clock time for server-side response generation as block size grows. All experiments fix $r = 256$.

Transitioning from $b = 2^{24}$ to $b = 2^{26}$ bytes per block yields a database that exceeds our 16 GiB of physical memory, resulting in a steepening of each trendline. The trendlines gradually converge to a new slope—reflecting that the computation rate is now limited by disk-read throughput—with instances having fewer servers converging more quickly. The differing convergence rates stem from our using `posix_madvise` to help the Linux kernel prefetch memory pages we will soon need—with $\ell = 2^6$ servers, the (at most) 256 required words occupy about 8 GiB and still fit in physical memory.

The 25× speedup for $\ell = 2^6$ versus $\ell = 2^1$ servers is comparable to the increase in the total number of servers and, hence, in the total amount of parallelism introduced. This is not surprising, as the total (expected) number of bit operations done by all servers is independent of ℓ. It is worth noting, however, that one cannot simply match the throughput of our single-threaded $\ell = 2^6$ computation by running 32 threads on each of $\ell = 2^1$ servers, since the response-generation computation is I/O bound; rather, to match the speed of the $\ell = 2^6$ protocol, one would need to divy up the work of each server among multiple compute nodes (à la MapReduce) and allow each "thread" to saturate its own memory bus. We hasten to add that, although this would roughly match the *throughput* of our new protocols with an equivalent hardware investment, the result would remain non-competitive in terms of communication and client-side reconstruction costs.

Client-side reconstruction: We also measured the time required for the client to reconstruct the responses. We provide log-log plots of our findings in Figure 6. As expected, the cost of reconstruction is essentially constant across all experiments, at one (vectorizable) bit operation for each bit in a block. When $b = 1$ GiB, took about 128.7 ± 0.7 ms.

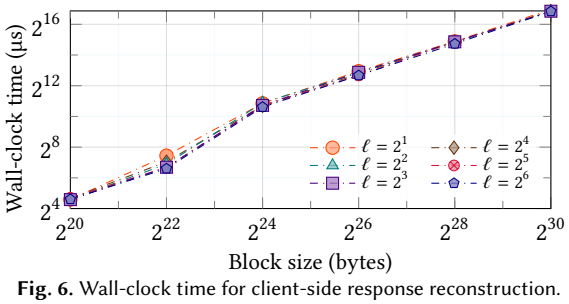
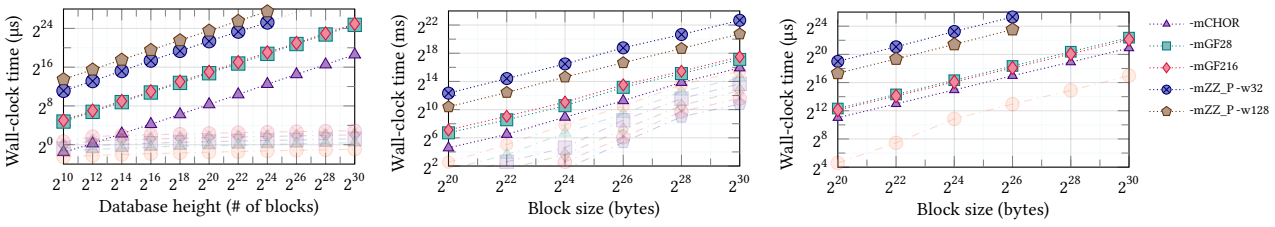


Fig. 6. Wall-clock time for client-side response reconstruction.

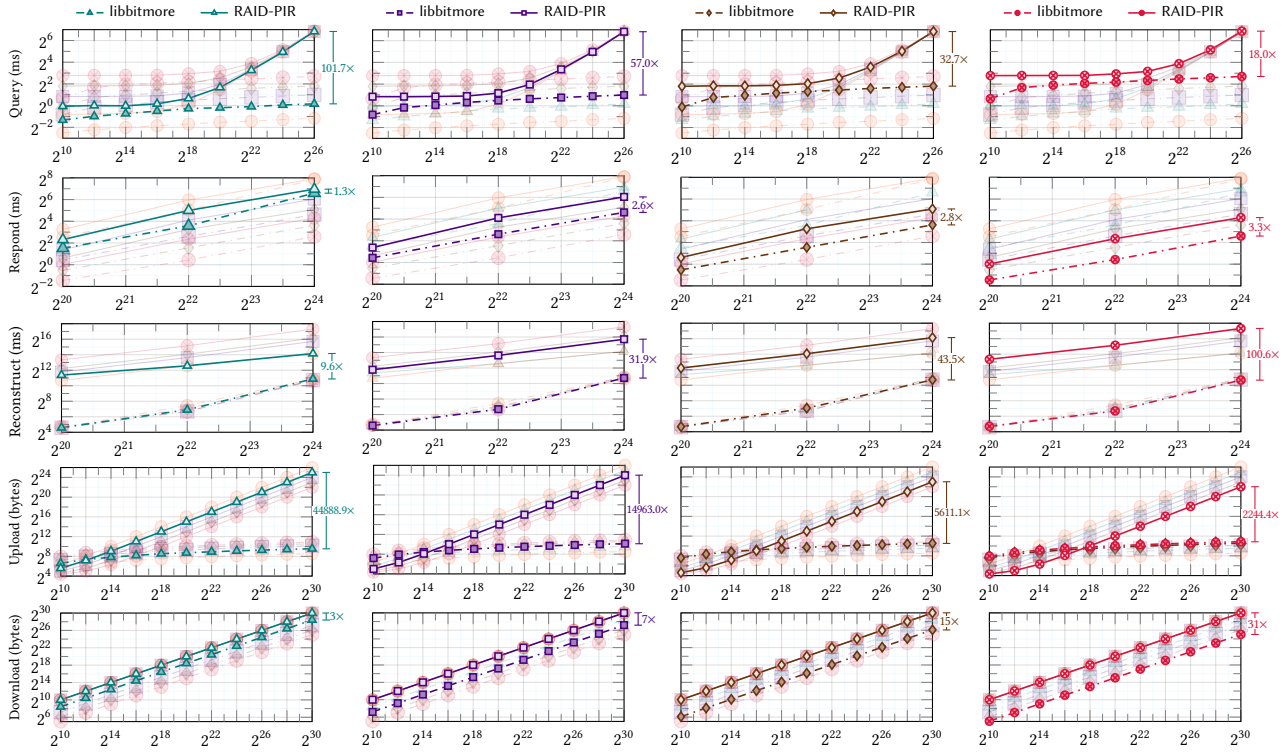
6.4 Head-to-head with Percy++

Our fourth set of experiments compares the performance of libbitmore with that of Percy++ v1.0 [19], an open-source C++ implementation of various PIR protocols. Percy++ supports numerous protocols with varying privacy thresholds [1, 10, 15, 18], Byzantine robustness guarantees [15], query batching [21, 24, 26], τ-independence [16], and more. Here we repeat a subset of our libbitmore experiments on a selection of "basic" protocols, fixing the privacy threshold and number of



(a) Query construction **(b)** Response generation ($r = 256$) **(c)** Reconstruction

Fig. 7. Head-to-head comparison with various 1-private, 2-server instances from Percy++ v1.0. The faint plots near the bottoms Figures 4a, 4b, and 4c show corresponding costs from Figures 2a, 5, and 6, respectively.



(a) $\ell = 2^2$ **(b)** $\ell = 2^3$ **(c)** $\ell = 2^4$ **(d)** $\ell = 2^5$

Fig. 8. Head-to-head comparison with computationally 1-private RAID-PIR v0.9.5 instances for ℓ ranging from 2^1 to 2^{32}. The scale of some experiments was limited because RAID-PIR v0.9.5 cannot handle databases that exceed physical memory.

servers in all Percy++ experiments as $t = 1$ and $\ell = 2$, respectively. These parameter choices provide the best possible performance for query generation and response reconstruction (whereas server-side computation and per-server communication cost are both agnostic to these settings [21]).

The fastest protocol Percy++ supports is the folklore-optimal protocol of Chor et al. [11, 27]—by far the *slowest* protocol in libbitmore. Figure 7 shows that the implementation of that protocol in libbitmore is noticeably faster than the one in Percy++. We also plot running times for Goldberg's protocol [18] with arithmetic in $\mathbf{GF}(2^8)$, $\mathbf{GF}(2^{16})$, and integers modulo 32- and 128-bit primes. We stress although libbitmore is much faster, many useful features in Percy++ cannot be realized in its simpler setting.

6.5 Head-to-head with RAID-PIR

Our fifth and final set of experiments compares the performance of libbitmore with that of RAID-PIR v0.9.5 [13]. RAID-PIR is highly configurable: ℓ-server instances can be configured with a tunable privacy level ranging from 1-privacy through to $(\ell - 1)$-privacy. Here we repeat a subset of our libbitmore experiments, fixing the privacy threshold (which they refer to as the *redundancy parameter* [14]) as $t = 1$ to yield instances with computational 1-privacy. Unlike with Percy++, a direct comparison between libbitmore and RAID-PIR is apt: in both cases, the cost of a (1-private) query decreases as the number of servers grows. We find that, given equivalent parameter choices, libbitmore consistently outperforms RAID-PIR *on every metric*, with improvements ranging from modest (1.3× faster when $\ell = 2^2$ and $b = 4$ MiB) to extreme (44 888.9× less upload when $\ell = 2^2$ and $r = 2^{30}$ blocks).

7 Conclusion

In this paper, we revisited the one-extra-bit PIR construction of Shah et al. with an eye toward rigor and efficiency, transforming that construction from a (highly impractical) theoretical result into what we believe to be the most efficient PIR protocol currently in existence, albeit under a very strong trust assumption. We have implemented our protocols as an open-source library and demonstrated their efficiency relative to existing techniques, including the "folklore optimal" protocols of Chor et al. and Boyle et al. For future work, we plan to explore how our approach extends to the setting of computationally t-private protocols for thresholds $t > 1$.

Acknowledgements: This work is based upon work supported by the National Science Foundation under Grant No. 1718595 and an NSERC Discovery Grant.

References

[1] Carlos Aguilar-Melchor and Philippe Gaborit. A fast private information retrieval protocol. In *Proceedings of ISIT 2008*, pages 1848–1852, Toronto, ON, Canada (July 2008).

[2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *Proceedings of IEEE S&P 2018*, pages 962–979, San Francisco, CA, USA (May 2018).

[3] Karim A. Banawan and Sennur Ulukus. Private information retrieval from Byzantine and colluding databases. In *Proceedings of Allerton 2017*, pages 1091–1098, Monticello, IL, USA (October 2017).

[4] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151 (March 2004).

[5] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. In *Proceedings of SCN 2002*, volume 2576 of *LNCS*, pages 326–341, Amalfi, Italy (September 2002).

[6] Simon R. Blackburn, Tuvi Etzion, and Maura B. Paterson. PIR schemes with small download complexity and low storage requirements. In *Proceedings of ISIT 2017*, pages 146–150, Aachen, Germany (June 2017).

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *Advances in Cryptology: Proceedings of CRYPTO 2016 (Part I)*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA (August 2016).

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of CCS 2016*, pages 1292–1303, Vienna, Austria (October 2016).

[9] Benny Chor and Niv Gilboa. Computationally private information retrieval (Extended abstract). In *Proceedings of STOC 1997*, pages 304–313, El Paso, TX, USA (May 1997).

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of FOCS 1995*, pages 41–50, Milwaukee, WI, USA (October 1995).

[11] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981 (November 1998).

[12] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of IEEE S&P 2015*, pages 321–338, San Jose, CA, USA (May 2015).

[13] Daniel Demmler, Amirr Herzberg, and Thomas Schneider. RAID-PIR; version 0.9.5 [computer software]. Available from: https://github.com/encryptogroup/RAID-PIR (October 2016).

[14] Danniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: Practical multi-server PIR. In *Proceedings of CCSW 2014*, pages 45–56, Scottsdale, AZ, USA (November 2014).

[15] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *Proceedings of USENIX Security 2012*, pages 269–283, Bellevue, WA, USA (August 2012).

[16] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Science (JCSS)*, 60(3):592–629 (June 2000).

[17] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Advances in Cryptology: Proceedings of EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658, Copenhagen, Denmark (May 2014).

[18] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of IEEE S&P 2007*, pages 131–148, Oakland, CA, USA (May 2007).

[19] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ / PIR in C++; version 1.0 [computer software]. Available from: git://git-crysp.uwaterloo.ca/percy (October 2014).

[20] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *Proceedings of NSDI 2016* (March 2016).

[21] Ryan Henry. Polynomial batch codes for efficient IT-PIR. In *Proceedings on Privacy Enhancing Technologies (PoPETS)*, volume 2016(4), pages 202–218, Darmstadt, Germany (July 2016).

[22] Ryan Henry. dpf++; version 0.0.1 [computer software]. Available from: https://www.github.com/rh3nry/dpfplusplus (July 2019).

[23] Ryan Henry and Syed Mahbub Hafiz. libbitmore; version v0.0.1 [computer software]. Available from: https://www.github.com/rh3nry/libbitmore (July 2019).

[24] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In *Proceedings of NDSS 2013*, San Diego, CA, USA (February 2013).

[25] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. In *Proceedings of PETS 2015*, volume 2, pages 222–243, Philadelphia, PA, USA (June–July 2015).

[26] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *Proceedings of FC 2015*, volume 8975 of *LNCS*, pages 168–186, San Juan, Puerto Rico (January 2015).

[27] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Proceedings of FC 2011*, volume 7035 of *LNCS*, pages 158–172, Gros Islet, St. Lucia (February 2011).

[28] Nihar B. Shah, K. V. Rashmi, and Kannan Ramchandran. One extra bit of download ensures perfectly private information retrieval. In *Proceedings of ISIT 2014*, pages 856–860, Honolulu, HI, USA (June–July 2014).

[29] Radu Sion and Bogdan Carbunar. On the practicality of private information retrieval. In *Proceedings of NDSS 2007*, San Diego, CA, USA (March 2007).

[30] Hua Sun and Syed Ali Jafar. The capacity of private information retrieval with colluding databases. In *Proceedings of GLOBECOM 2016*, pages 941–946, Washington, DC, USA (December 2016).

[31] Hua Sun and Syed Ali Jafar. The capacity of private information retrieval. *IEEE Transactions on Information Theory*, 63(7):4075–4088, 2017.

[32] Hua Sun and Syed Ali Jafar. The capacity of robust private information retrieval with colluding databases. *IEEE Transactions on Information Theory*, 64(4):2361–2370, 2018.

[33] Luqin Wang, Trishank Karthik Kuppusamy, Yong Liu, and Justin Cappos. A fast multi-server, multi-block private information retrieval protocol. In *Proceedings of GLOBECOM 2015*, pages 1–6, San Diego, CA, USA (December 2015).

[34] Sergey Yekhanin. Private information retrieval. *Communications of the ACM (CACM)*, 53(4):68–73 (April 2010).

A Related Work

The literature on PIR is vast, though relatively few works focus specifically on 1-private multi-server PIR or on (practical) protocols exhibiting provably optimal costs. Before concluding, we briefly highlight a few of the most closely related efforts in this space.

Kiayias, Leonardos, Lipmaa, Pavlyk, and Tang [25] propose a single-server PIR protocol with asymptotically optimal download cost. For sufficiently large records—beyond around 1 GiB each—the download overhead of their scheme is at most a few percent; however, their reliance on number-theoretic assumptions yields computation costs that are prohibitively high for large-scale deployment [29].

In the multi-server setting, Henry, Huang, and Goldberg [24] proposed *batch queries* for Goldberg's multi-server protocol, which allows to amortize the costs associated with fetching multiple blocks at a time. In followup work, both Wang, Kuppusamy, Liu, and Cappos [33] and Demmler, Herzberg, and Schneider [14] extend batch queries to Chor et al.'s folklore protocol. For maximally large batch sizes, this approach significantly reduces the efficiency gaps between our bit-more-than-a-bit protocols and those in Figures 7 and 8—provided the client wishes to make sufficiently many queries in parallel.

In the 2-server setting, several works have employed $(2, 2)$-DPFs to achieve low upload costs [8, 12, 17], while Angel, Chen, Laine, and Setty [2] proposed a novel technique to obtain low upload costs in single-server FHE-based PIR using so-called plaintext slot permutations. For $\ell > 2$ servers, there does not appear to be any prior work that provides upload costs scaling sublinearly in r.

B The "one-extra-bit" construction

Let $\vec{\mathbf{1}} \coloneqq \sum_{j=1}^{s} \vec{\boldsymbol{e}}_j$ be the all-1s vector in $\mathbb{F}^s$ and, for each $i \in [1..r]$, define $\mathrm{Ord}_i \colon \mathbf{M}^{(r,s)} \to [0..s]$ such that

$$\mathrm{Ord}_i(\boldsymbol{A}) = j \text{ iff } \mathrm{Row}_i(\boldsymbol{A}) = \vec{\boldsymbol{e}}_{j+1}.$$

The "one-extra-bit" construction, as originally proposed by Shah et al., is a special case of our one-extra-word construction. It uses the binary field $\mathbb{F} = \mathbf{GF}(2)$, the all-1s vector $\vec{\boldsymbol{v}} = \vec{\mathbf{1}}$, and a special choice for the mapping $\varphi \colon \mathbf{M}^{(r,s)} \to [1..\ell]$ as defined in Equation (2) below.

The "one-extra-bit" mapping: Denote by $\mathbf{M}_0^{(r,s)} \coloneqq \{\boldsymbol{A} \in \mathbf{M}^{(r,s)} \mid \mathrm{Ord}_1(\boldsymbol{A}) = 0\}$ the subset of matrices $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ having $\vec{\boldsymbol{e}}_1 \in \mathbb{F}^{s+1}$ as their first rows. The following observation is immediate.

Observation 5. *There are* $(s+1)^{r-1}$ *distinct matrices in* $\mathbf{M}_0^{(r,s)}$.

Indeed, a natural bijection associates with each matrix $\boldsymbol{A} \in \mathbf{M}_0^{(r,s)}$ a non-negative integer less than $(s+1)^{r-1}$ by treating the sequence $\mathrm{Ord}_2(\boldsymbol{A}), \ldots, \mathrm{Ord}_r(\boldsymbol{A})$ as the low- through high-order digits of an $(r-1)$-digit integer expressed in base $(s+1)$; that is, it associates with $\boldsymbol{A}$ the integer $\sum_{i=2}^{r}(s+1)^{i-2}\,\mathrm{Ord}_i(\boldsymbol{A})$.

The one-extra-bit mapping extends the aforementioned bijection between $\mathbf{M}_0^{(r,s)}$ and $[0..(s+1)^{r-1}-1]$ to an injection from all of $\mathbf{M}^{(r,s)}$ to $[0..(s+1)^{r-1}-1]$ as follows. For each $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, let $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbb{F}^{(s+1)\times(s+1)}$ denote the *cyclic permutation matrix* that acts on $\boldsymbol{A}$ (via right multiplication) by rotating it column-wise to the left by $\mathrm{Ord}_1(\boldsymbol{A})$ positions. (In other words, $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}}$ is the cyclic permutation matrix having the property that $\mathrm{Row}_1(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}) = \vec{\boldsymbol{e}}_1$, thereby ensuring that $\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbf{M}_0^{(r,s)}$ for every $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$.)

The one-extra-bit mapping is then given by

$$\varphi(\boldsymbol{A}) \coloneqq \textstyle\sum_{i=2}^{r}(s+1)^{i-2}\,\mathrm{Ord}_i(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}). \tag{2}$$

The notation and definition in Equation (2) are quite unlike the ones proposed by Shah et al. [28, Algorithm 1]; however, it is a straightforward, if tedious, exercise to verify that the two definitions are, in fact, equivalent.

Analysis of the "one-extra-bit" construction: Observe that $\varphi(\boldsymbol{A}) = \sum_{i=2}^{r}(s+1)^{i-2}\,\mathrm{Ord}_i(\boldsymbol{A})$ if and only if $\boldsymbol{A} \in \mathbf{M}_0^{(r,s)}$. More generally, we have the following observation.

Observation 6. *Let* $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$. *Then* $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ *iff* $\boldsymbol{B}$ *is among the* $s+1$ *column-wise cyclic permutation of* $\boldsymbol{A}$.

An immediate consequence of Observation 6 is that any two (distinct) matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ having $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ necessarily differ on *every single row*, a fact from which the next corollary directly follows.

Corollary 3. *If* $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ *such that* $\boldsymbol{A} \neq \boldsymbol{B}$ *but* $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$, *then* $\boldsymbol{A} \notin \mathrm{Eq}(i; \boldsymbol{B})$ *for any* $i \in [1..r]$.

Note, in particular, that Corollary 3 implies $|\varphi(\mathrm{Eq}(i; \boldsymbol{A}))| = s+1$ for every $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$; hence, this choice of φ satisfies the necessary and sufficient conditions given by Theorem 3 and the next corollary is immediate.

Corollary 4. *The one-extra-bit construction provides* perfect 1-privacy *(in the sense of Definition 4).*

The (online) computation cost for each server is effectively nil;[9] the computation cost for the client to set up and solve the resulting system of equations is about s^2 bit operations. The client can uniquely reference a specific Frobenius product $\langle \boldsymbol{D}^*, \boldsymbol{A} \rangle_{\mathrm{F}}$ via the ordered pair (x, j) in which $x = \mathrm{Ord}_1(\boldsymbol{A})$ and $j = \varphi(\boldsymbol{A})$; hence, for any $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, the client need only send $\lceil \lg(s+1) \rceil$ bits to each of the $s+1$ servers in $\varphi\big(\mathrm{Eq}(i; \boldsymbol{A})\big)$. More precisely, $|\varphi^{-1}\big(\varphi(\boldsymbol{A})\big)| = s+1$ for all $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ so that φ satisfies the condition given by Theorem 2 as long as s is polynomial in r. This proves the following.

Lemma 4. *The one-extra-bit construction provides* non-trivial communication *(in the sense of Definition 3), provided* $s \in 2^{o(r)}$.

Combining Corollary 4 and Lemma 4 with the fact that $\|\vec{\mathbf{1}}\|_1 = s$ (cf. Theorem 1) yields the following theorem.

Theorem 14. *If* $s \in 2^{o(r)}$ *with* $s > 1$, *then the one-extra-bit construction is a perfectly* 1*-private* $(s+1)^{r-1}$*-server PIR protocol.*

The preceding theorem establishes that the one-extra-bit construction provides perfect 1-privacy, which is the strongest privacy result possible for the construction. Indeed, as we argue below, there necessarily exist (many) doubleton coalitions $C \subseteq [1..\ell]$ with respect to which the construction is clearly *not* C-private. Yet it is apparent that the one-extra-bit construction *must* be private against *at least some* non-singleton coalitions. Appendix C analyzes the necessary and sufficient conditions on a coalition $C \subseteq [1..\ell]$ under which a one-extra-word instance is C-private.

B.1 Trading a bit of work for a bit less storage

In this sub-appendix, we briefly recall and generalize the following simple observation of Shah et al. regarding the one-extra-bit construction, which leads to a modest improvement in the per-server storage cost. Recall that one-extra-word uses $\vec{\boldsymbol{\nu}} = \vec{\mathbf{1}}$

Observation 7. *Fix* $j \in [1..\ell]$ *and consider the set* $\varphi^{-1}(j) = \{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{s+1}\}$ *comprising the* $s+1$ *distinct matrices in* $\mathbf{M}^{(r,s)}$ *that map to bucket* j. *We have*

$$\sum_{i=1}^{s+1} \langle \boldsymbol{D}^*, \boldsymbol{A}_i \rangle_{\mathrm{F}} = 0, \tag{7.1}$$

or, equivalently,

$$\sum_{i=1}^{s} \langle \boldsymbol{D}^*, \boldsymbol{A}_i \rangle_{\mathrm{F}} = \langle \boldsymbol{D}^*, \boldsymbol{A}_{s+1} \rangle_{\mathrm{F}}. \tag{7.2}$$

Observation 7 follows from the choice of $\vec{\boldsymbol{\nu}} = \vec{\mathbf{1}}$ and the fact that $\mathbb{F} = \mathbf{GF}(2)$ is a field of characteristic 2. Shah et al. point out that each server can store just s bits (rather than $s+1$ bits), say, $\langle \boldsymbol{D}^*, \boldsymbol{A}_1 \rangle_{\mathrm{F}}, \ldots, \langle \boldsymbol{D}^*, \boldsymbol{A}_s \rangle_{\mathrm{F}}$, and then use Equation (7.2) to compute the discarded bit $\langle \boldsymbol{D}^*, \boldsymbol{A}_{s+1} \rangle_{\mathrm{F}}$ on the fly, as needed. The tradeoff for this modest optimization is an increase in the amortized server-side computation cost, which increases from nil to (just under) one $\mathbf{GF}(2)$ addition per request (on average): Exactly one out of $s+1$ servers must reconstruct its discarded bit for each request and, on average, each server must reconstruct its discarded bit for one out of every $s+1$ requests in which it participates.

We remark (sans formal proof) that the same idea generalizes easily to an arbitrary field $\mathbb{F}$ by setting $\vec{\boldsymbol{\nu}} = (p-1)\vec{\mathbf{1}}$, where $p = \mathrm{char}(\mathbb{F})$ is the characteristic of $\mathbb{F}$.[10]

C Necessary and sufficient conditions for C-privacy

The privacy guarantees of a one-extra-word instance are completely determined by the choice of φ. Intuitively, for any coalition C, a one-extra-word instance is C-private if (i) it provides 1-privacy, and (ii) no two servers in C ever participate in the same query, regardless of which block the client is requesting nor the outcomes of any coin tosses the client makes. The next theorem formalizes this intuition.

Theorem 15. *Let* $C \subseteq [1..\ell]$ *be a coalition. A one-extra-word instance is* C-private *if and only if, for each server* $j \in C$,

1. *the construction is* $\{j\}$*-private (cf. Theorem 3), and*
2. *for all* $\boldsymbol{A} \in \varphi^{-1}(j)$ *and* $i \in [1..r]$, $\varphi\big(\mathrm{Eq}(i; \boldsymbol{A})\big) \cap C = \{j\}$.

Proof (sketch). Let $n = \sum_{j \in C} |\varphi^{-1}(j)|$. The first condition guarantees that no client can ever request two or more distinct Frobenius products from a single coalition member (as part of the same query), while the second condition extends the first to ensure that no client will ever request two or more distinct Frobenius products from two or more different coalition members. It follows that there are precisely $n+1$ possible "joint views" that can arise for coalition C in each query: namely,

9 Appendix B.1 recalls a modest optimization due to Shah et al., which slightly reduces the per-server storage cost of any one-extra-word construction that uses $\vec{\boldsymbol{\nu}} = \vec{\mathbf{1}}$ together with the one-extra-bit mapping φ (as defined in Equation (2)) at the expense of requiring a small (amortized) computation cost per query.

10 Intuitively, for any $\varphi^{-1}(j) = \{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{s+1}\}$, summing $\boldsymbol{A}_1 + \cdots + \boldsymbol{A}_{s+1}$ yields the all-1s matrix in $\mathbb{F}^{r \times (s+1)}$ so that $\sum_{i=1}^{s+1} \langle \boldsymbol{D}^*, \boldsymbol{A}_i \rangle_{\mathrm{F}}$ is just a summation over all entries in $\boldsymbol{D}^*$. Now, by construction, the last entry in each row of $\boldsymbol{D}^*$ is a product of the sum of all earlier entries in the row with $p-1$; thus, each row sums to p so that $\sum_{i=1}^{s+1} \langle \boldsymbol{D}^*, \boldsymbol{A}_i \rangle_{\mathrm{F}} = rp$, which, since $\mathbb{F}$ has characteristic p, is equal to 0 in $\mathbb{F}$.

the client can request any one of the n Frobenius products that coalitions' members hold, or it can request nothing at all from any of the coalition members.

By Lemma 1, the first n events each happen with probability $1/(s+1)^{r-1}$, regardless of which block the client seeks. Moreover, because these $n+1$ events are pairwise disjoint and mutually exhaustive, it follows that the remaining ("nothing requested") event occurs with probability $1 - n/(s+1)^{r-1}$. As all probabilities are independent of the block being fetched, this establishes what we set out to prove. □

Theorem 15 and the remarks following Theorem 14 suggest that the privacy guarantees of the one-extra-bit construction may, in fact, be much *stronger* than Corollary 4 and Theorem 14 suggest; the reality, however, is rather nuanced.

Privacy of the "one-extra-bit" construction, revisited: We first note that if $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbf{M}^{(r,s)}$ are distinct matrices such that $\boldsymbol{A}_1 \in \mathrm{Eq}(i; \boldsymbol{A}_2)$ for some $i \in [1..r]$, then the construction cannot be $\{\varphi(\boldsymbol{A}_1), \varphi(\boldsymbol{A}_2)\}$-private; indeed, letting I denote the random variable that describes the index of the block the client seeks, and letting F_1 and F_2 respectively denote the events that the client requests $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ as part of its query, it follows from Lemma 1 that

$$\Pr\big[F_1 \cap F_2 \;\big|\; I = i\big] = 1/(s+1)^{r-1},$$

whereas it follows from Part (ii) of Observation 1 that

$$\Pr\big[F_1 \cap F_2 \;\big|\; I \neq i\big] = 0.$$

Hence, there exist *many* doubleton coalitions with respect to which privacy cannot be guaranteed. (Indeed, each $j \in [1..\ell]$ belongs to rs distinct doubleton coalitions C with respect to which the construction is not C-private.) Thus, Theorem 14 gives the strongest "threshold" notion of privacy one can hope for.

What's more, the one-extra-bit mapping φ, as defined in Equation (2), results in a protocol with the peculiar property that simply knowing that *any given pair* of servers holding a pair of i-equivalent Frobenius products is (or is not) involved in a query (without necessarily knowing what *query strings* those servers receive) is sufficient for a *passive observer*—such as the querier's Internet service provider—to compromise the querier's privacy! We are unaware of any other PIR protocol whose privacy falls to such *traffic analysis* attacks when the observer may be limited to seeing only encrypted query strings. This surprising and troubling property is due to the following observation and its corollary.

Observation 8. *Let φ be as defined in Equation (2). If $\boldsymbol{B}_1 \in \mathrm{Eq}(i; \boldsymbol{A}_1)$ and $\boldsymbol{B}_2 \in \mathrm{Eq}(i; \boldsymbol{A}_2)$, then $\varphi(\boldsymbol{A}_1) = \varphi(\boldsymbol{A}_2)$ if and only if $\varphi(\boldsymbol{B}_1) = \varphi(\boldsymbol{B}_2)$.*

Recalling that we defined $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbb{F}^{(s+1)\times(s+1)}$ as the permutation matrix such that $\mathrm{Row}_1(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}) = \vec{\boldsymbol{e}}_1$, an equivalent formulation of Observation 8 is:

$$\boldsymbol{B} \in \mathrm{Eq}(i; \boldsymbol{A}) \text{ iff } \boldsymbol{B} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{B}} \in \mathrm{Eq}(i; \boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}).$$

That is, if *some* Frobenius product in bucket j_1 has an i-equivalent partner in bucket j_2, then (i) *every* Frobenius product in j_1 has an i-equivalent partner in j_2, and (ii) no Frobenius product in j_1 has a i^*-equivalent partner in j_2 for any $i^* \neq i$ (see Part (ii) of Observation 1).

Corollary 5. *Let I denote the random variable that describes the index of the block the client seeks, and let F_1 and F_2 denote the events that a client requests a Frobenius product from bucket j_1 and from j_2, respectively. Then, we have that either (i) $\Pr[F_1 \cap F_2] = 0$, or (ii) there exists $i \in [1..r]$, such that*

$$\Pr\big[I = i \;\big|\; F_1 \cap F_2\big] = 1 \tag{5.1}$$

and

$$\Pr\big[I = i \;\big|\; F_1 \cap \bar{F}_2\big] = 0. \tag{5.2}$$

Colloquially, Equation (5.1) says that knowing *any pair* of servers involved in a given query is sufficient to uniquely determine the index of the block sought. Moreover, Equation (5.2) says that if $\boldsymbol{B} \in \mathrm{Eq}(i; \boldsymbol{A})$, then knowing that $\varphi(\boldsymbol{A})$ is involved in a query, while $\varphi(\boldsymbol{B})$ is not, is sufficient to deduce that the request is not for block i.