Batch Codes and Their Applications

Yuval Ishai* Eyal Kushilevitz† Rafail Ostrovsky‡ Amit Sahai§

Abstract

A *batch code* encodes a string x into an m-tuple of strings, called *buckets*, such that each batch of k bits from x can be decoded by reading at most one (more generally, t) bits from each bucket. Batch codes can be viewed as relaxing several combinatorial objects, including expanders and locally decodable codes.

We initiate the study of these codes by presenting some constructions, connections with other problems, and lower bounds. We also demonstrate the usefulness of batch codes by presenting two types of applications: trading maximal load for storage in certain load-balancing scenarios, and amortizing the computational cost of private information retrieval (PIR) and related cryptographic protocols.

*Department of Computer Science, Technion. E-mail: yuvali@cs.technion.ac.il. Some of this research was done while at AT&T Labs – Research and Princeton University. Research supported in part by the Israel Science Foundation.

†Department of Computer Science, Technion. E-mail: eyalk@cs.technion.ac.il. Some of this research was done while the author was a visiting scientist at IBM T.J. Watson Research Lab. Research supported in part by the Israel Science Foundation.

‡Dept. of Computer Science, UCLA. E-mail: rafail@cs.ucla.edu. Supported in part by a gift from Teradata.

§Dept. of Computer Science, Princeton University. E-mail: sahai@cs.princeton.edu. Research supported in part by grants from the Alfred P. Sloan Foundation and the NSF ITR program.

1 Introduction

In this paper we introduce and study a new coding problem, the interest in which is both purely theoretical and application-driven. We start by describing a general application scenario.

Suppose that a large database of n items (say, bits) is to be distributed among m devices.[1] After the data has been distributed, a user chooses an arbitrary subset (or *batch*) of k items, which she would like to retrieve by reading the data stored on the devices. Our goal is to minimize the worst-case maximal *load* on any of the m devices, where the load on a device is measured by the number of bits read from it, while also minimizing the total amount of storage used.[2]

To illustrate the problem, consider the case $m = 3$. A naive way to balance the load would be to store a copy of the entire database in each device. This allows to reduce the load by roughly a factor of 3, namely any k-tuple of items may be obtained by reading at most $\lceil k/3 \rceil$ bits from each device. However, this solution triples the total amount of storage relative to the original database, which may be very expensive in the case n is large. A natural question is whether one can still achieve a significant load-balancing effect while reducing the storage requirements. For instance, suppose that only a 50% increase in the size of the original database can be afforded (i.e., a total of $1.5n$-bit storage). By how much can the maximal load be reduced under this constraint?

For these parameters, no clever way of replicating individual data bits (or "hashing" them to the three devices) can solve the problem. Indeed, any such replication scheme would leave at least $n/6$ bits that can only be found on one particular device, say the first, and hence for $k \leq n/6$ there is a choice of k items which incurs a load of k on this device.[3]

In light of the above, we need to consider more general distribution schemes, in which each stored bit may depend on more than one data bit. A simple construction proceeds as follows. Partition the database into two parts L, R containing $n/2$ bits each, and store L on the first device, R on the second, and $L \oplus R$ on the third. Note that the total storage is $1.5n$ which satisfies our requirement. We argue that each *pair* of items i_1, i_2 can be retrieved by making at most *one* probe to each device.

[1] The term "device" can refer either to a *physical* device, such as a server or a disk, or to a completely virtual entity, as in the application we will describe in Section 1.3.

[2] Both our measure of load and the type of tradeoffs we consider are similar to Yao's *cell-probe* model [33], which is commonly used to model time-storage tradeoffs in data structure problems.

[3] One could argue that unless the k items are *adversarially* chosen, such a worst-case scenario is very unlikely to occur. However, this is not the case when k is small. More importantly, if the queries are made by *different* users, then it is realistic to assume that a large fraction of the users will try to retrieve the same "popular" item, which has a high probability of being stored only on a single device. Such a multi-user scenario will be addressed in the sequel.

Consider two cases. If i_1, i_2 reside in different parts of the database, then it clearly suffices to read one bit from each of the first two devices. On the other hand, if i_1, i_2 both reside in the same part, say L, then one of them can be retrieved directly from the first device, and the other by reading one bit from each of the other devices and taking the exclusive-or of the two bits. Thus, the worst-case maximal load can be reduced to $\lceil k/2 \rceil$. This achieves significant reduction in load with a relatively small penalty in storage.

1.1 Batch Codes

We abstract the problem above into a new notion we call a *batch code*, and we give several constructions for these new objects.

An (n, N, k, m, t) *batch code* over an alphabet Σ encodes a string $x \in \Sigma^n$ into an m-tuple of strings $y_1, \ldots, y_m \in \Sigma^*$ (also referred to as *buckets*) of total length N, such that for each k-tuple (*batch*) of distinct indices $i_1, \ldots, i_k \in [n]$, the entries $x_{i_1}, \ldots, x_{i_k}$ can be decoded by reading at most t symbols from each bucket. Note that the buckets in this definition correspond to the devices in the above example, the encoding length N to the total storage, and the parameter t to the maximal load. Borrowing from standard coding terminology, we will refer to n/N as the *rate* of the code.

When considering problems involving several parameters, one typically focuses the attention on some "interesting" settings of the parameters. In this case, we will mostly restrict our attention to a binary alphabet Σ and to the case $t = 1$, namely at most *one* bit is read from each bucket. This case seems to most sharply capture the essence of the problem and, as demonstrated above, solutions for this case can also be meaningfully scaled to the general case.[4] Moreover, the case $t = 1$ models scenarios where only a single access to each device can be made at a time, as is the case for the cryptographic application discussed in Section 1.3. From now on, the term "batch code" (or (n, N, k, m) batch code) will refer by default to the above special case.

We will typically view n, k as the given parameters and try to minimize N, m as a function of these parameters. Note that in our default setting we must have $m \geq k$. It is instructive to point out the following two (trivial) extreme types of batch codes: (1) $C(x) = (x, x, \ldots, x)$, i.e., replicate x in each bucket; in this case we can use an optimal m (i.e., $m = k$) but the rate $1/k$ is very low. (2) $C(x) = (x_1, x_2, \ldots, x_n)$, i.e., each bit of x is put in a separate bucket; in this case

[4] The decoding procedure in the above example can be viewed as $\lceil k/2 \rceil$ repetitions of decoding a batch code with parameters $(n, 1.5n, 2, 3, 1)$, yielding decoding with parameters $(n, 1.5n, k, 3, \lceil k/2 \rceil)$.

the rate, 1, is optimal but m is very large. Our goal is to obtain good intermediate solutions which are close to being optimal in both aspects.

MULTISET BATCH CODES. The load-balancing scenario described above involves a *single* user. It is natural to consider a scenario where k distinct users, each holding some query i_j, wish to directly retrieve data from the same devices. There are two main differences between this setting and the default one. First, each selected item x_{i_j} should be recovered from the bits read by the jth user alone, rather than from *all* the bits that were read. Second, while previously the k queries were assumed to be distinct, this assumption cannot be made in the current setting. Since the indices i_j now form a *multiset*, we use the term *multiset batch code* to refer to such a stronger type of batch code.

In defining multiset batch codes, we make the simplifying assumption that prior to the decoding process the users can coordinate their actions in an arbitrary way; we only "charge" for the bits they read.[5] We note, however, that most of our constructions can be modified to require little or no coordination between the users with a small degradation in performance.

Aside from their direct application in a multi-user scenario, an additional motivation for multiset batch codes is that their stronger properties make them easier to manipulate and compose. Hence, this variant will be useful as a building block even in the single-user setting.

1.2 Our Results

We have already insisted on minimal load per device – every batch is processed with only *one bit* being read from each device. Therefore, the two quantities of interest are: (1) Storage overhead, and (2) the number of devices m (which must be at least k in our setting) . This leads to two fundamental existential questions about batch codes: First, can we construct codes with arbitrarily low storage overhead (rate $1 - \epsilon$) as the number of queries k grows, but with the number of devices m still being "feasible" in terms of k? Second, can we construct codes with essentially the optimal number of devices ($m \approx k$) with storage overhead $o(k)$? We resolve both of these questions affirmatively, and also show a number of interesting applications of batch codes and our constructions. Our techniques and precise results are outlined below:

BATCH CODES FROM UNBALANCED EXPANDERS. In the above example we first considered a replication-based approach, where each item may be replicated in a carefully selected subset of buckets but no functions (e.g., linear combinations) of

[5] This is a reasonable assumption in some scenarios (e.g., if such a coordination is cheaper than an access to the device, or if it can be done off-line).

several items can be used. For the specific parameters of that example it was argued that this restricted approach was essentially useless. However, this is not always the case. We observe that if the assignment of items to buckets is specified by an *unbalanced expander graph* (with a weak expansion property), then one obtains a batch code with related parameters.[6] Using random graphs of polynomial size (where the random graph can be chosen and fixed "once and for all"), we obtain batch codes with parameters $N/n = O(\log n)$ and $m = O(k)$. This answers Question 2 above affirmatively for non-multiset batch codes. The code can be made explicit by using explicit constructions of expanders [31, 8], but with weaker parameters.

This expander-based construction has some *inherent* limitations. First, it cannot be used for obtaining codes whose rate exceeds $1/2$ (unless $m = \Omega(n)$). Second, even for achieving a smaller *constant* rate, it is required that m depend not only on k but also on n (e.g., the random graph achieves rate $1/3$ with $m = k^{3/2}n^{1/2}$). Third, this approach cannot be used to obtain *multiset* batch codes, since it cannot handle the case where many users request the same item. These limitations will be avoided by our other constructions.

THE SUBCUBE CODE. Our second batch code construction may be viewed as a composition of (a generalization of) the code from the above "$(L, R, L \oplus R)$" example with itself. We refer to the resulting code as the *subcube code*, as it admits a nice combinatorial interpretation involving the subcubes of a hypercube. The subcube code is a *multiset* batch code, furthermore it can achieve an arbitrarily high constant rate. Specifically, any constant rate $\rho < 1$ can be realized with $m = k^{O(\log \log k)}$. While the asymptotic dependence of m on k will be improved by subsequent constructions, the subcube code still yields our best results for some small values of k, and generally admits the simplest and most explicit batch decoding procedure.

BATCH CODES FROM SMOOTH CODES. A q-query *smooth code* (a close relative of *locally decodable codes* [18]) maps a string x to a codeword y such that each symbol of x can be decoded by probing at most q random symbols in y, where the probability of any particular symbol being probed is at most $q/|y|$.[7] We establish a two-way general relation between smooth codes and multiset batch codes. In particular, any smooth code gives rise to batch codes with related parameters. However, this connection is not sufficiently tight to yield the parameters we seek. See Section 1.4 for further discussion.

[6] This is very different from the construction of standard error-correcting codes from expanders (cf. [29]), in which the graph specifies parity-checks rather than a replication pattern.

[7] Using the more general terminology of [18], this is a $(q, q, 1/2)$-smooth code.

BATCH CODES FROM REED-MULLER[8] CODES. By exploiting the structure of Reed-Muller codes (beyond their smoothness), we obtain batch codes with excellent parameters. In particular, for any constant $\epsilon > 0$, we obtain a multiset batch code with rate $n/N = \Omega(1/k^{\epsilon})$ and $m = k \cdot \log^{2+1/\epsilon+o(1)} k$. Thus, the number of devices is within a polylogarithmic factor from optimal, while the storage overhead is only k^{ϵ} – answering Question 2 above affirmatively for *multiset* batch codes. Using Reed-Muller codes we also get multiset batch codes with rate $n/N = 1/(\ell!+\epsilon)$ and $m = k^{1+1/(\ell-1)+o(1)}$ for any constant $\epsilon > 0$ and integer $\ell \geq 2$.

THE SUBSET CODE. The batch codes we have constructed so far either require the rate to be below 1/2 (expander, Reed-Muller codes), or achieve high rates at the expense of requiring m to be (slightly) super-polynomial in k (subcube codes). Our final construction, which admits a natural interpretation in terms of the subset lattice,[9] avoids both of these deficiencies. Specifically, we get the following result, answering Question 1 above in the affirmative:

> For any constant rate $\rho < 1$ there is a constant $c > 1$ such that for every k and sufficiently large n there is an (n, N, k, m) multiset batch code with $n/N \geq \rho$ and $m = O(k^c)$.

In other words, one can insist on adding only an arbitrarily small percentage to the original storage, yet reduce the load by any desired amount k using only poly(k) devices.

The parameters of the different constructions are summarized in the following table.

Code	rate	m	multiset?
Expander	$1/d < 1/2$	$O(k \cdot (nk)^{1/(d-1)})$	No
	$\Omega(1/\log n)$	$O(k)$	
Subcube	$\rho < 1$	$k^{O(\log\log k)}$	Yes
RM	$1/\ell! - \epsilon < 1/2$	$k \cdot k^{1/(\ell-1)+o(1)}$	Yes
	$\Omega(1/k^{\epsilon})$	$k \cdot (\log k)^{2+1/\epsilon+o(1)}$	
Subset	$\rho < 1$	$k^{O(1)}$	Yes

NEGATIVE RESULTS. The focus of this paper has primarily been constructions of batch codes and their applications (see below), but as with most interesting combinatorial objects, finding optimal lower bounds is an intriguing open question. We give some initial lower bounds (tight in some instances) based on elementary combinatorial and information-theoretic arguments.

[8] A Reed-Muller code is one whose codewords correspond to all ℓ-variate polynomials of total degree at most d over a finite field F, where $|F| > d + 1$.

[9] The subset code may be viewed as a subcode of the *binary* Reed-Muller code. The latter, however, does not suffice for our purposes.

ADDITIONAL EFFICIENCY CONCERNS. While we have mainly focused on the most basic efficiency measures of batch codes, there are several other natural measures to be considered. These include efficiency of encoding and decoding, amount of coordination between the users in the multi-user setting (or the *distributed* complexity of decoding), efficiency of handling online additions and deletions of queries, average case performance, and so forth. We note that most of our solutions perform favorably in most of these aspects.

1.3 Cryptographic Applications

In addition to their immediate application to the general load-balancing scenario discussed above, batch codes are also motivated by the following cryptographic problem. A *private information retrieval* (PIR) protocol allows a user to retrieve an item i from a database of size n while hiding i from the servers storing the database. There are two main settings for PIR. In the information-theoretic setting, there are two or more servers holding copies of the database and the default privacy requirement is that each *individual* server learn no information about i. In the computational setting for PIR, there is typically only a single server holding the database and the privacy requirement is relaxed to *computational* privacy, which should hold against computationally bounded servers and under some cryptographic assumption.

The current state-of-the-art PIR protocols can achieve a very low *communication complexity* (cf. [9, 4, 20, 6]), but on the other hand they are inherently very expensive in terms of *computation* and require $\Omega(n)$ operations on the servers' part [5]. It is thus highly desirable to *amortize* the computational cost of PIR over k queries made by the user. An initial step in this direction was made in [5], where it was shown that the computational cost of handling multiple queries in certain PIR protocols can be *slightly* amortized by using fast matrix multiplication.

Batch codes can be used to obtain much better amortization of the computational cost of PIR while only moderately increasing the (low) communication. Specifically, an (n, N, k, m) batch code with bucket sizes N_j, $1 \leq j \leq m$, provides a reduction from k-query PIR to m invocations of standard PIR on databases (buckets) of size N_j. Any nontrivial batch code, satisfying $\sum_{j=1}^{m} N_j \ll nk$, implies amortized savings to the time complexity. (This assumes that the database has already been preprocessed to its batch encoding.) In terms of asymptotics, the amortized savings are most appealing when k is large, e.g., $k = n^{\epsilon}$ for some constant $0 < \epsilon < 1$. In this case one can combine the single-server PIR protocols of [20, 6] with our batch code constructions to get protocols that are "essentially optimal" with respect to both communication and computation. Specifically, k items can be privately retrieved using $k^{1+o(1)}$ communication and $n^{1+o(1)}$ compu-

tation. We stress that even when k is a small constant, batch codes still allow to obtain significant concrete savings. Also, the use of batch codes applies to both the information-theoretic and computational settings for PIR, and does not introduce any error probability or privacy loss. The reader is referred to Section 5 for a more detailed discussion of this application, including its comparison to an alternative hashing-based approach.

Our amortization results for PIR substantially improve the previous ones from [5]. In contrast to [5], however, they do not directly apply to the case where the k queries originate from different users. They also do not apply to the "PIR with preprocessing" model considered in [5], which allows to preprocess the database but requires the savings to kick in immediately (starting with the user's first query). Still, our techniques have an interesting corollary for this setting as well, discussed in Section 5.

ADDITIONAL CRYPTOGRAPHIC APPLICATIONS. PIR can be a useful building block in other cryptographic protocols. Hence, amortization results for PIR carry over to various other protocol problems. For instance, using efficient reductions to k-query PIR [15, 24, 25, 11], one can get protocols for $\binom{n}{k}$-*Oblivious-Transfer* [26, 12] which are essentially optimal with respect to *both* time and communication. Previous solutions to this problem achieved *either* (essentially) optimal communication or (essentially) optimal computation, but not both simultaneously. Significant savings are also possible in other contexts where PIR is used (e.g., [10, 23, 13, 7]). Again, the reader is referred to Section 5 for more details.

1.4 Related Notions

Below we survey a few of the interesting relations between batch codes and other primitives.

RELATION WITH INFORMATION DISPERSAL. Similarly to the application of erasure codes to information dispersal [27], batch codes also have applications to distributed data storage. However, the two problems differ both in their main goal (fault tolerance vs. load balancing) and in the type of data to which they cater: the former can be meaningfully applied to a *single, large* item, whereas batch codes are most interesting in the case of *many small* items.

RELATION WITH RANDOMNESS CONDUCTORS. The entropy smoothening property of expanders, extractors, and similar objects (generalized under the term "randomness conductors" [8]) makes them intuitively related to batch codes. In fact, replication-based batch codes with $t = 1$ are *equivalent* to unbalanced expanders with expansion factor 1. However, when dropping the (very restrictive) replication requirement, batch codes seem to no longer have analogues in the world of

randomness conductors.

RELATION WITH LOCALLY-DECODABLE/SMOOTH CODES. As noted above, smooth codes naturally give rise to batch codes. However, batch codes and smooth/locally-decodable codes are very different objects. In particular, the smoothness property implies significant fault tolerance, whereas batch codes require virtually none (an extreme example being the expander-based construction). Intuitively, smooth decoding requires a highly *random* probing pattern, whereas batch decoding only require the existence of *one* such good pattern for any k-tuple of items. An additional separating example is given by high degree *binary* Reed-Muller codes. In Section 3.5 we show that despite their superiority as smooth codes, they cannot achieve the batch decoding parameters we obtain via subset codes.

The relation of our problem to the last two notions is quite interesting. First, batch codes provide in some sense a *common relaxation* of expander-type objects and smooth codes. While many other connections between these types of problems exist (e.g., both can be constructed from multivariate polynomials [3, 32, 28] and both are useful in the context of derandomization [1, 17, 30, 21]), we are not aware of another problem whose formulation provides an almost direct relaxation of these two notions. Second, viewing (certain) expanders as a *restricted* class of batch codes, whose performance can be improved via *generalization*, suggests that it might be fruitful to investigate similar relaxations of related notions, or to look for additional applications of randomness conductors which can benefit from a similar relaxation.

2 Preliminaries

In this section we define the variants of batch codes we will be interested in and note some simple relations between the parameters. We start by defining the default notion of batch codes.

Definition 2.1 (batch code) *An (n, N, k, m, t) batch code over Σ is defined by an encoding function $C : \Sigma^n \to (\Sigma^*)^m$ (each output of which is called a* bucket*) and a decoding algorithm A such that:*

- *The total length of all m buckets is N (where the length of each bucket is independent of x);*
- *For any $x \in \Sigma^n$ and $\{i_1, \ldots, i_k\} \subseteq [n]$, $A(C(x), i_1, \ldots, i_k) = (x_{i_1}, \ldots, x_{i_k})$, and A probes at most t symbols from each bucket in $C(x)$ (whose positions are determined by $i_1, \ldots, i_k$).*

We will sometimes refer to x as the database*. By default, we assume batch codes to be* systematic*, i.e., the encoding should contain each symbol of x in some fixed*

position. Finally, an (n, N, k, m) batch code *is an* $(n, N, k, m, 1)$ *batch code over* $\Sigma = \{0, 1\}$.

For "multi-user" applications, it is natural to consider the following stronger variant of batch codes.

Definition 2.2 (multiset batch code) *An* (n, N, k, m) multiset batch code *is an* (n, N, k, m) *batch code satisfying the following additional requirement. For any* multiset $i_1, i_2, \ldots, i_k \in [n]$ *there is a* partition *of the buckets into subsets* $S_1, \ldots, S_k \subseteq [m]$ *such that each item* x_{i_j}, $j \in [k]$, *can be recovered by reading (at most) one symbol from each bucket in* S_j. *This can be naturally generalized to* $t > 1$.

The following special case of (multiset) batch codes will be particularly useful:

Definition 2.3 (primitive batch code) *A* primitive batch code *is an* (n, N, k, m) *batch code in which each bucket contains a single symbol, i.e.* $N = m$.

Note that primitive batch codes are trivial in the single-user case, but are non-trivial for multiset batch codes because of multiple requests for the same item. Next, we give some simple relations between our default choice of the parameters ($\Sigma = \{0, 1\}, t = 1$) and the general one.

Lemma 2.4 *The following holds both for standard batch codes and for multiset batch codes:*

1. *An* (n, N, k, m, t) *batch code (for an arbitrary* t*) implies an* (n, tN, k, tm) *code (with* $t = 1$*).*
2. *An* (n, N, k, m) *batch code implies an* (n, N, tk, m, t) *code and an* $(n, N, k, \lceil m/t \rceil, t)$ *code.*
3. *An* (n, N, k, m) *batch code implies an* (n, N, k, m) *code over* $\Sigma = \{0, 1\}^w$, *for an arbitrary* w.
4. *An* (n, N, k, m) *batch code over* $\Sigma = \{0, 1\}^w$ *implies a* (wn, wN, k, wm) *code over* $\Sigma = \{0, 1\}$.

3 Constructions

In this section we describe our different batch code constructions. Due to lack of space, some of the proofs have been omitted from this extended abstract and can be found in the full version.

3.1 Batch Codes from Unbalanced Expanders

Consider the case of "replication-based" batch codes, where each bit of the encoding is a physical bit of x. Then, we may represent the code as a bipartite graph, where the n vertices on the left correspond to the data bits, the m vertices on the right correspond to the buckets, and there is an edge if the bit is in the corresponding bucket; in this case N is the number of edges. By Hall's theorem, the graph represents an (n, N, k, m) batch code if and only if each set S of at most k vertices on the left has at least $|S|$ neighbors on the right. In the following we use standard probabilistic arguments to estimate the tradeoffs between the parameters we can get using this approach.

Parameters. Fix parameters n, k, d. The expander will have n vertices on the left vertex set A, and m (to be specified) on the right vertex set B. The graph is constructed as follows. For each vertex $u \in A$ on the left, the following procedure is repeated d times: Choose a uniformly selected element $v \in B$, and add the edge (u, v) to the graph. (If it already exists do nothing.) A standard union bound analysis gives the following:

Theorem 3.1 *Let $m \geq k \cdot (nk)^{1/(d-1)} \cdot t$. Then, with probability at least $1 - t^{-2(d-1)}$, the neighborhood of every sets $S \subset A$ such that $|S| \leq k$ contains at least $|S|$ vertices in B.*

Remark 3.2 We make several remarks concerning the expander-based approach to batch codes:

1. For the single-user case, the expander-based approach (which is equivalent to the replication-based approach) offers several practical advantages. For instance, once a good constant-degree expander graph is fixed, the encoding function can be computed in linear time, and only a constant number of bits in the encoding need to be updated for any change in a single bit of x.
2. When d is constant, the value of m in the above analysis depends not only on k, but also on n. We note that this is not an artifact of the analysis, but an inherent limitation.
3. The above bound can be made fully explicit if k is a constant, because the expansion properties can be checked in polynomial time.
4. We call the reader's attention to the following setting of parameters: Let $d = O((1/\epsilon) \log nk)$, in which case we obtain $m = (1 + \epsilon)k$. Note that this is only possible because of our very weak expansion requirement. A lossless expander, for instance, would trivially require $m \geq (1 - \epsilon)dk$. Thus, it is important to make use of the weak expansion condition to get optimal parameters.

5. Known explicit constructions of unbalanced expanders yield various interesting settings of parameters, though all of these are quite far from optimal:

   - The explicit construction of unbalanced expanders of [8], Theorem 7.3, yields $d = 2^{(\log\log n)^3}$ and $m = O(kd)$.
   - The explicit construction of unbalanced expanders of [31], Theorem 3, yields two possible settings of parameters: (1) $d = \log^c n$ for some constant $c > 1$, and $m = 2^{(\log k)^{1+\epsilon}}$, which is superpolynomial in k; (2) $d = 2^{(\log\log n)^2}$, and $m = k^c$, for some constant $c > 1$.

3.2 The Subcube Code

Expander-based batch codes have two inherent limitations: their rate cannot exceed $1/2$ and they cannot satisfy the stronger multiset property. We now describe a simple (and fully explicit) batch code construction which can overcome both of these limitations.

The underlying idea is to recursively apply the "$(L, R, L \oplus R)$" encoding described in the introduction. For instance, suppose that each of the 3 buckets is again encoded using the same encoding function. The resulting code has 9 buckets of size $n/4$ each. Now, a batch of $k = 4$ items can be decoded as follows. First, arbitrarily partition them into two pairs and for each pair find the positions in the "high-level" buckets that need to be read for decoding this pair. (Note that the high-level buckets are just logical entities and are not part of the final encoding.) Combining the two pairs, at most two items need to be read from each high-level bucket. We can now apply again the same procedure, decoding the pair in each high level bucket by probing at most one position in each of the corresponding low-level buckets. Hence we get a (multiset) code with $N = (9/4)n$, $k = 4$, and $m = 9$. In what follows we formally describe a generalization of this idea.

Here and in the following, it will be useful to first construct a "gadget" batch code for a small database of size n_0, and then extend it to a larger code attaining the same rate. The following simple lemma crucially relies on the multiset property of the code, and does not apply to the default notion of batch codes.

Lemma 3.3 (Gadget lemma) *Let C_0 be an (n_0, N_0, k, m)* multiset *batch code. Then, for any positive integer r there is an (n, N, k, m) multiset batch code C with $n = rn_0$ and $N = rN_0$. We denote the resulting code C by $(r \cdot C_0)$.*

Let ℓ denote a parameter which, for fixed n, k, will allow to trade between the rate and the number of buckets.

Lemma 3.4 *For any integers $\ell \geq 2$ and n, there is a primitive (n, N, k, m) multiset batch code $\mathbf{C}_\ell$ with $n = \ell$, $N = m = \ell + 1$, and $k = 2$.*

Proof: The encoding function of $\mathbf{C}_\ell$ is defined by $\mathbf{C}_\ell(x) = (x_1, x_2, \ldots, x_\ell, x_1 \oplus x_2 \oplus \ldots \oplus x_\ell)$. To decode a multiset $\{i_1, i_2\}$ we distinguish between two cases. If $i_1 \neq i_2$, then the two bits can be directly read from the two corresponding buckets (and there is no need to read bits from the remaining buckets). For a pair of identical bits $\{i, i\}$, one of them can be read directly from the ith bucket, and the other can be decoded by taking the exclusive-or of the bits in the remaining buckets. □

To make this construction general, we should extend it to handle larger database size n and number of queries k. Lemma 3.3 can be used for increasing the database size using the same number of buckets. Towards handling larger values of k, we define the following composition operator for batch codes.

Lemma 3.5 (Composition lemma) *Let C_1 be an (n_1, N_1, k_1, m_1) batch code and C_2 an (n_2, N_2, k_2, m_2) batch code such that the length of each bucket in C_1 is n_2 (in particular, $N_1 = m_1 n_2$). Then, there is an (n, N, k, m) batch code C with $n = n_1$, $N = m_1 N_2$, $k = k_1 k_2$, and $m = m_1 m_2$. Moreover, if C_1 and C_2 are* multiset *batch codes then so is C, and if all buckets of C_2 have the same size then this is also the case for C. We will use the notation $C_1 \otimes C_2$ to denote the composed code C.*

To construct a batch code with general parameters k, n, we first compose the code $\mathbf{C}_\ell$ with itself $\log_2 k$ times, obtaining a primitive code with parameters n_0, k, and then apply Lemma 3.3 with $r = \lceil n/n_0 \rceil$.

Lemma 3.6 *For any integers $\ell \geq 2$ and $d \geq 1$ there is a (primitive) multiset batch code $\mathbf{C}_\ell^d$ with $n = \ell^d$, $N = m = (\ell+1)^d$, and $k = 2^d$.*

Proof: $\mathbf{C}_\ell^d$ is defined inductively as follows: $\mathbf{C}_\ell^1 = \mathbf{C}_\ell$, and $\mathbf{C}_\ell^d = (\ell \cdot \mathbf{C}_\ell^{d-1}) \otimes \mathbf{C}_\ell$ (where '$\cdot$' is the gadget operator from Lemma 3.3 and '$\otimes$' is the composition operator from Lemma 3.5). It can be easily verified by induction on d that this composition is well defined and that $\mathbf{C}_\ell^d$ has the required parameters. □

In the full version we give a combinatorial interpretation of $\mathbf{C}_\ell^d$ in terms of the subcubes of the hypercube $[\ell]^d$. Using $\mathbf{C}_\ell^d$ with $d = \log_2 k$ as a gadget and applying Lemma 3.3, we get:

Theorem 3.7 *For any integers k, n and $\ell \geq 2$ there is an explicit multiset batch code with parameters $m = (\ell+1)^{\lceil \log_2 k \rceil} \approx k^{\log_2(\ell+1)}$ and $N = \lceil n/\ell^d \rceil \cdot m \approx k^{\log_2(1+1/\ell)} \cdot n$.*

By setting $\ell = O(\log k)$, the rate of the code can be made arbitrarily close to 1. Specifically:

Corollary 3.8 *For any constant $\rho < 1$ and integer k there is an integer $m(= k^{O_\rho(\log\log k)})$ such that for all sufficiently large n there is an (n, N, k, m) multiset batch code with $n/N \geq \rho$.*

In the following sections we will be able to achieve a constant rate with m being polynomial in k.

3.3 Batch Codes from Smooth Codes

The notion of smooth decoding arose from the context of *locally-decodable* error-correcting codes [18]. Intuitively, a smooth code is one where any input symbol can be decoded by looking at a small subset of symbols, such that every symbol in the encoding is looked at with roughly the same probability. Formally, a q-query smooth code over Σ is defined by an encoding function $C : \Sigma^n \to \Sigma^m$ together with a randomized, non-adaptive decoding procedure D satisfying the following requirement. For all $x \in \Sigma^n$ and indices $i \in [n]$, we have that $D^{C(x)}(i)$ always reads at most q symbols of $C(x)$ and correctly outputs x_i. Moreover, for each $j \in [m]$ the probability of $C(x)_j$ being read by $D^{C(x)}(i)$ is at most q/m. We will also consider *expected* q-query smooth codes, where the expected (rather than worst-case) number of queries made by D is bounded by q. In contrast to most of the literature on locally-decodable codes, we will typically be interested in smooth codes where q is quite large (say, $q = O(n^c)$ for some $0 < c < 1$).

We suggest two simple generic approaches for converting a smooth code into a primitive multiset batch code. In fact, both approaches do not modify the encoding function, and only make an oracle use of the smooth decoder.

The first approach applies the following greedy strategy. The batch decoder processes the items sequentially. For each item i_j, the smooth decoder is repeatedly invoked until it produces a q-tuple of queries that have not yet been used. The batch decoder reads the corresponding symbols and recovers x_{i_j}. This process continues until all k items have been successfully decoded. This approach yields the following theorem:

Theorem 3.9 *Let $C : \Sigma^n \to \Sigma^m$ be a q-query smooth code. Then C describes a primitive multiset batch code with the same parameters as the smooth code, where $k = \lfloor m/q^2 \rfloor$.*

The gap between $k = m/q^2$ and $k = m/q$ (the best one could hope for) is significant when q is large. In particular, it makes Theorem 3.9 totally useless when $q > m^{1/2}$. In the next sections, we will see two cases where this gap can be narrowed down using specific properties of the underlying codes, and one where it cannot.

When Theorem 3.9 cannot be used at all, the following alternative decoding strategy may be used. The batch decoder independently invokes the smooth decoder on each of the k items. Call such an experiment t-successful if no symbol is requested more than t times. Using a Chernoff bound and a union bound one can estimate the minimal t for which the experiment is t-successful with positive probability. For such t, the code may be viewed as a primitive (n, m, k, m, t) multiset batch code, which can be converted into a standard batch code using Lemma 2.4 and Lemma 3.3. An unfortunate byproduct of this conversion is that it decreases the rate of the code by a factor of t. Hence, the current approach is unsuitable for obtaining constant-rate batch codes with $t = 1$. An analysis of the second approach, applied to a typical range of parameters, gives the following.

Theorem 3.10 *Let $C : \Sigma^n \to \Sigma^m$ be a q-query smooth code. Then, for any k such that $kq/m > \log m$, the code C describes a primitive (n, m, k, m, t) multiset batch code over Σ with $t = kq/m + 2(kq \log m/m)^{1/2}$. Hence for the same t there is also a primitive (n, tm, k, tm) multiset batch code.*

Remark 3.11 Both of the above batch decoding algorithms (corresponding to Theorems 3.9, 3.10) are described as randomized Las-Vegas algorithms. However, they can be derandomized using limited independence. The same holds for randomized decoding algorithms that will be presented in the next sections.

We end this section by noting a weak converse of Theorem 3.9. The decoding procedure of an (n, m, k, m) primitive multiset batch code gives rise to an *expected* (m/k)-query smooth decoding procedure: to smoothly decode x_i, run the batch decoder on the multiset $\{i, i, \ldots, i\}$, and pick a random set S_j of buckets from the k disjoint sets allowing to decode x_i. We stress though that even the specific notion of a *primitive multiset* batch code is quite loosely related to smooth codes. Moreover, for general (non-primitive) batch codes, the above converse of Theorem 3.9 is essentially vacuous.

3.4 Batch Codes from Reed-Muller Codes

Reed-Muller (multivariate polynomial) codes are a well known example for smooth codes. Hence, one can apply the generic transformations from the previous section to get batch codes with related parameters. We will show that their special structure can be used to obtain significantly better batch decoding procedures.

Let ℓ denote the number of variables, where $\ell \geq 2$, and d a bound on the total degree of the polynomials we consider. We use F to denote the field over which the code will be defined, where $|F| \geq d + 2$. We assume by default that $|F|$ is not much larger than $d + 2$ (e.g., $|F| = 2^{\lceil \log_2(d+2) \rceil}$).

Recall that the Reed-Muller (RM) code is defined as the evaluation of all degree d polynomials on all $|F|^\ell$ evaluation points. Each such polynomial can be defined not only by picking (arbitrary) coefficients for each of the $\binom{\ell+d}{d}$ monomials of degree at most d, but also by picking (arbitrary) values of the polynomial evaluated at some specified subset S of $\binom{\ell+d}{d}$ points in F^ℓ. The existence of such a subset of F^ℓ is a simple consequence of the linear independence of the monomials of degree at most d, when viewed as vectors of their evaluations on F^ℓ. Thus, we associate the $n = \binom{\ell+d}{d}$ input values with the evaluations of a degree (at most) d polynomial at the points in S. Note that this yields a systematic code of length $m = |F|^\ell = (\alpha d)^\ell$. We refer to this code as an (ℓ, d, F) RM code. For any constant ℓ, the rate of the (ℓ, d, F) RM code is roughly $1/\ell!$ and its degree satisfies $d = \Theta(m^{1/\ell})$.

We start by quoting the standard smoothness property of RM codes.

Lemma 3.12 *Any (ℓ, d, F) RM code (with $|F| \geq d + 2$) is a q-query smooth code with $q = d + 1$.*

Our first goal is to maximize the rate. Hence, we would like to use $\ell = 2$ variables. However, in this case Lemma 3.12 gives smooth decoding with $q = \Theta(m^{1/2})$, and so Theorem 3.9 can only support a constant k. The following specialized batch decoding procedure gets around this barrier and, more generally, obtains better asymptotic bounds on m in terms of k when ℓ is small. The high-level geometric idea is to decode each target point using a random line passing through this point, where by slightly increasing the field size one can afford to "delete" all intersections between different lines. This yields the following theorem.

Theorem 3.13 *For any constants $\beta, \epsilon > 0$, an (ℓ, d, F) RM code with $|F| = \alpha d$, where $\alpha = 1 + \beta(1 + \epsilon)$ and $d = \omega(\ell \log d)$, defines a primitive multiset batch code over F with parameters $n = \binom{\ell+d}{d}$, $m = N = (\alpha d)^\ell$, and $k = \beta d(\alpha d)^{\ell-2}$.*

PARAMETERS ACHIEVED. The improved analysis yields the following for the case where ℓ is constant: Let β, ϵ be set to arbitrarily small constants. The rate of the code then will be arbitrarily close to $(1/\ell!)$. On the other hand, $m = O(k \cdot k^{1/(\ell-1)})$. In particular, this code is interesting even for the bivariate case. Again using Lemma 3.3, we obtain codes with rate arbitrarily close to $1/2$, and $m = O(k^2)$. Note that the alphabet size for these codes is $q = O(\log |F|) = O(\log k)$. The alphabet can be turned to binary using Lemma 2.4, increasing m by a factor of $q = O(\log k)$.

Finally, by combining Lemma 3.12 with Theorem 3.10 one gets codes with sub-constant rate, but where m can be made very close to k:

Theorem 3.14 *An (ℓ, d, F) RM code defines a primitive multiset batch code over F with parameters $n = \binom{\ell+d}{d}$, $m = N = (\alpha d)^\ell$, $k = \Omega((m \log m)/d)$, and $t = \log m$.*

PARAMETERS ACHIEVED. Suppose we set parameters as follows: $\ell = \frac{\epsilon \log n}{\log \log n}$ and $d = O(\log^{1+1/\epsilon} n)$. Then the theorem above, together with Lemma 3.3, yields a multiset batch code with $N = O(n \cdot k^\epsilon)$, $m = O(k \cdot \log^{1+1/\epsilon} k)$, and $t = (1+\epsilon)\log k$. If we reduce t to 1 using Lemma 2.4, we obtain multiset batch codes with $N = O(n \cdot k^\epsilon \log k)$, and $m = O(k \cdot \log^{2+1/\epsilon} k)$. Note that the alphabet size for these codes is $O(\log |F|) = O(\log \log k)$. Using Lemma 2.4, the alphabet can be turned to binary, increasing m by a factor of $O(\log \log k)$.

3.5 The Subset Code

In this section we describe our final construction of batch codes. In contrast to all previous constructions, it will simultaneously achieve an arbitrary constant rate and keep m polynomial in k.

Let ℓ, w be parameters, where $0 < w < \ell$. A typical choice of parameters will be $w = \alpha\ell$ for some constant $0 < \alpha < 1/2$. While we are primarily interested in codes over the binary alphabet, it will be convenient to view the alphabet as an arbitrary Abelian group Σ (where $\Sigma = Z_2$ by default).

The (ℓ, w) *subset code* is a primitive batch code with $n = \binom{\ell}{w}$ and $N = m = \sum_{j=0}^{w} \binom{\ell}{j}$. We index each data item by a unique set $T \in \binom{[\ell]}{w}$ and each bucket by a unique subset $S \subseteq [\ell]$ of size at most w. The content of bucket S is defined by: $Y_S \stackrel{\text{def}}{=} \sum_{T \supseteq S,\ |T|=w} x_T$. That is, each bucket receives the sum (or exclusive-or) of the data items labelled by its supersets. Before describing a batch decoding procedure for the subset code, it will be instructive to analyze its performance as a smooth code.

Definition 3.15 *For any $T \in \binom{[\ell]}{w}$ and $T' \subseteq T$, let $L_{T,T'} \stackrel{\text{def}}{=} \{S \subseteq [\ell] : S \cap T = T' \wedge |S| \leq w\}$. We will sometimes refer to $L_{T,T'}$ as the* space *defined by the point T and the* direction *T'.*

The following lemma follows immediately from the definition.

Lemma 3.16 *If T', T'' are distinct subsets of T, then $L_{T,T'} \cap L_{T,T''} = \emptyset$.*

The following lemma is crucial for establishing the smoothness property of the subset code.

Lemma 3.17 *For any $T \in \binom{[\ell]}{w}$ and $T' \subseteq T$, the item x_T can be decoded by reading all values Y_S such that $S \in L_{T,T'}$.*

Proof: Using the inclusion-exclusion principle, one may express x_T as a function of $Y_{T'}$ and the values Y_S such that $T' \subset S \not\subseteq T$ as follows:

$$x_T = Y_{T'} \quad - \quad \sum_{j_1 \notin T} Y_{T' \cup \{j_1\}} + \sum_{j_1 < j_2,\ j_1, j_2 \notin T} Y_{T' \cup \{j_1, j_2\}} - \tag{1}$$
$$\ldots \quad + (-1)^{w - |T'|} \sum_{j_1 < \ldots < j_{w-|T'|},\ j_h \notin T} Y_{T' \cup \{j_1, \ldots, j_{w-|T'|}\}}$$

Note that the subsets S involved in the right hand side of Eq. (1) are precisely those in $L_{T,T'}$. □

Lemma 3.18 *The (ℓ, w) subset code is an* expected $(m/2^w)$*-query smooth code, where $m = \sum_{j=0}^{w} \binom{\ell}{w}$.*

Proof: From the previous two lemmas, for each item x_T there are 2^w disjoint spaces $L_{T,T'}$ of values Y_S, from each of which x_T can be decoded. The smooth decoder can now proceed by picking $T' \subset T$ at random, reading all the values Y_S such that $S \in L_{T,T'}$, and recovering x_T from the values read. Since the spaces $L_{T,T'}$ form a perfect tiling of all sets $S \subset [\ell]$ such that $|S| \leq w$, the expected number of queries is exactly $m/2^w$. □

We now look at the asymptotic parameters achieved by the subset code. Let $H(\cdot)$ denote the binary entropy function. Set $w = \alpha\ell$ for some $0 < \alpha < 1/2$. (Choosing $\alpha < 1/2$ is necessary to ensure constant rate.) Using the approximation $\sum_{j=0}^{w} \binom{\ell}{j} \approx 2^{H(\alpha)\ell}$ and the inequality $\binom{\ell}{w-1} \leq \frac{w}{\ell - w}\binom{\ell}{w}$ we get:

Claim 3.19 *For any constant $0 < \alpha < 1/2$, the $(\ell, w = \alpha\ell)$ subset code has length $m \approx 2^{H(\alpha)\ell}$ and rate $n/m \geq 1 - \alpha/(1-\alpha)$. It is an expected q-query smooth code with $q \approx m^{1-\alpha/H(\alpha)}$.*

It follows that we cannot apply the generic transformations from Section 3.3 to get batch codes with a constant rate, regardless of the relation between k and m. Theorem 3.9 cannot be applied because $q > m^{1/2}$ (and moreover, q is the *expected* number of queries). Theorem 3.10 cannot be applied because it results in codes with sub-constant rates.

BATCH DECODING THE SUBSET CODE. Given a multiset $T_1, \ldots, T_k$ of items to be decoded, we would like to assign to each T_j a direction $T'_j \subseteq T_j$ such that the spaces $L_{T_1,T'_1}, \ldots, L_{T_k,T'_k}$ will be pairwise disjoint. One approach that comes to mind is to pick the directions T'_j uniformly at random independently of each other. However, it is easy to verify that if T_a, T_b are disjoint (or even nearly disjoint) then L_{T_a,T'_a} and L_{T_b,T'_b} will intersect with high probability. Another reasonable approach would be to greedily assign to each item T_j a direction T'_j of the highest

available weight, such that no point in the space L_{T_j,T'_j} has been used before. (The advantage of heavy sets T'_j is the small size of the corresponding spaces.) This approach as well is too crude, since it may cause adjacent sets to totally block each other at an early stage of the process. However, these two approaches provide the intuition we need for our solution. The former approach works well whenever the sets are "not too far" from each other, whereas the latter approach works well whenever the sets are "not too close" to each other.

We combine the two approaches by picking each direction independently at random from a distribution which is biased towards heavy directions. Specifically, let each T'_j be obtained from T_j by selecting each element with probability $3/4$. We analyze the probability that the spaces L_{T_a,T'_a} and L_{T_b,T'_b} intersect by first considering the case where T_a, T_b are close, and then the case they are far.

Lemma 3.20 *Suppose* $|T_a \cap T_b| \leq w/3$. *Then* $\Pr[L_{T_a,T'_a} \cap L_{T_b,T'_b} \neq \emptyset] = 2^{-\Omega(w)}$.

Proof: The random variable $|T'_a \cup T'_b|$ is larger than a binomial variable with $5w/3$ trials and success probability $3/4$. By Chernoff's bound, $\Pr[|T'_a \cup T'_b|] \leq w] < 2^{-\Omega(w)}$. The lemma follows by noting that whenever $|T'_a \cup T'_b| > w$, the spaces L_{T_a,T'_a} and L_{T_b,T'_b} must be disjoint. □

Lemma 3.21 *Suppose* $|T_a \cap T_b| > w/3$. *Then* $\Pr[L_{T_a,T'_a} \cap L_{T_b,T'_b} \neq \emptyset] = 2^{-\Omega(w)}$.

Proof: For the spaces L_{T_a,T'_a} and L_{T_b,T'_b} to intersect, the sets T'_a and T'_b must contain precisely the same elements from the intersection $T_a \cap T_b$. The probability of the latter event is clearly bounded by $2^{-\Omega(w)}$. □

By combining Lemmas 3.20, 3.21 and taking the union over all $\binom{k}{2}$ bad events we may conclude that there is an efficient (Las-Vegas) algorithm for batch decoding $k = 2^{\Omega(w)}$ items . Substituting the code parameters we get:

Theorem 3.22 *For any* $0 < \alpha < 1/2$, k, *and sufficiently large* ℓ, *the* $(\ell, w = \alpha\ell)$ *subset code is a primitive multiset batch code with* $m \approx 2^{H(\alpha)\ell}$, *rate* $n/m \geq 1 - \alpha/(1-\alpha)$, *and batch size* $k = 2^{\Omega(w)} = m^{\Omega(\alpha/H(\alpha))}$.

Finally, using Lemma 3.3 we obtain non-primitive codes with an arbitrarily high constant rate and $m = \text{poly}(k)$.

Corollary 3.23 *For every* $\rho < 1$ *there is some* $c > 1$ *such that for every* k *and sufficiently large* n *there is an* (n, N, k, m) *multiset batch code with rate* $n/N \geq \rho$ *and* $m = O(k^c)$.

3.5.1 Relation with binary Reed Muller codes

The subset code may be viewed as a subcode of the binary Reed-Muller code. Specifically, when $\Sigma = Z_2$ the (ℓ, w) subset code is defined by the ℓ-variate polynomials over Z_2 whose monomials all contain *exactly* $d = \ell - w$ distinct variables (rather than *at most* d variables). Because of this restriction, one can truncate all evaluation points of weight less than d.

It is thus natural to compare the performance of subset codes to binary RM codes. It is implicit in a recent work of Alon et al. [2] that the binary Reed-Muller code defined by all ℓ-variate polynomials of degree (at most) d is $(2^{d+1} - 2)$-smooth. However, we show that when $d > \ell/2$ (which is necessary for achieving rate above $1/2$) any systematic[10] binary RM code cannot be batch decoded.

Claim 3.24 *Let C be a systematic binary Reed Muller code defined by ℓ-variate degree-d polynomials where $d > \ell/2$. Then, viewed as a primitive multiset batch code, C does not support decoding even $k = 3$ items.*

Proof: Let p_x denote the polynomial encoding x. Let $i \in [n]$ and $v \in Z_2^\ell$ be such that for all x we have $p_x(v) = x_i$. (Such i exist since C is systematic.) Let S_1, S_2, S_3 denote the disjoint subsets of evaluation points used for decoding the multiset $\{i, i, i\}$. By linearity we may assume wlog that for each S_j, the bit x_i can be decoded by taking the *sum* (over Z_2) of the evaluations of p_x on points in S_j, and by disjointness of the sets we may assume that $v \notin S_1 \cup S_2$. Let $S_1' = S_1 \cup \{v\}$ and $S_2' = S_2 \cup \{v\}$. It follows that the characteristic vectors of S_1', S_2' are codewords in the dual code, hence each contains the evaluations of a degree-$(\ell - d - 1)$ polynomial on all points in Z_2^ℓ. (The dual code of a binary ℓ-variate RM code of degree d is a binary ℓ-variate RM code of degree $\ell - d - 1$, cf. [22].) Let q_1, q_2 denote the polynomials of the dual code corresponding to S_1', S_2'. Since $S_1' \cap S_2' = \{v\}$ the polynomial $q_1 q_2$ must evaluate to 1 on v and to 0 on all other points. Note that the unique polynomial satisfying this has degree ℓ. But since $d > \ell/2$, the degree of $q_1 q_2$ must be less than ℓ – a contradiction. □

4 Negative Results

In the full version of this paper we obtain several simple lower bounds for batch codes, some of which are tight for their setting of parameters. Summarizing, our bounds cover the following cases:

[10] Recall that a code is systematic if each entry of x appears in some fixed position of the encoding. In fact, it suffices in the following that *some* entry of x appear as an entry of the encoding.

- First, we show a general bound for *multiset* batch codes, relating their rate and k to the minimum distance of the batch code treated as an error-correcting code. Then, we go on to study cases when m is close to k:
- We show that if one is only willing to have $m = k$ servers, then the trivial $N = nk$ bound is essentially optimal.
- For *multiset* batch codes, we observe (trivially) that $N \geq (2k - m)n$ holds. For the special case of exactly one additional server ($m = k + 1$), we further improve this bound to $N \geq (k - 1/2)n$, and show that this is tight. In particular, this shows that the simple "$(L, R, L \oplus R)$" batch code mentioned in the introduction is optimal for the case $m = 3, k = 2$.
- All our constructions of multiset batch codes go through *primitive* batch codes. However, we show that this is *not* without loss of generality, because for primitive codes, a stronger bound holds. In general, in order to have $N < kn$, we show that $m \geq \lfloor (3k + 1)/2 \rfloor$. This is also tight, and the resulting primitive batch code for this value of m has $N/n = \frac{1}{2}\lfloor (3k + 1)/2 \rfloor$.

All formal statements and proofs can be found in the full version. Below we give a representative lower bound proof, establishing the tightness of the "$(L, R, L \oplus R)$" construction from the Introduction.

Theorem 4.1 *Let C be an $(n, N, 2, 3)$ multiset batch code. Then, $N \geq 1.5n$.*

Proof: We consider only multisets of two identical queries i. For each such pair, the decoder should recover x_i from two disjoint subsets of buckets. Hence, for each i there is a bucket b_i, such that x_i can be decoded in two possible ways: (1) by reading one bit from b_i; (2) by reading one bit from each of the remaining buckets.

For $j = 1, 2, 3$, let n_j count the number of indices i such that $b_i = j$. Let X be a uniformly distributed string (from $\{0, 1\}^n$) and X_j its restriction to the bits i such that $b_i = j$. Note that $H(X_j) = n_j$. Let (B_1, B_2, B_3) denote the joint distribution $C(X)$, where B_j is the content of bucket j.

We are now ready to derive the lower bound. We have assumed that all bits in X_1 can be recovered from B_2, B_3. Since X_1 is independent of X_2, X_3, we have:

$$\begin{aligned} n_1 &\leq H(B_2B_3 \mid X_2X_3) \\ &= H(B_2 \mid X_2X_3) + H(B_3 \mid B_2X_2X_3) \\ &\leq H(B_2 \mid X_2) + H(B_3 \mid X_3) \end{aligned} \tag{2}$$

Similarly,

$$n_2 \leq H(B_1 \mid X_1) + H(B_3 \mid X_3) \tag{3}$$

and

$$n_3 \leq H(B_1 \mid X_1) + H(B_2 \mid X_2) \tag{4}$$

Summing Eq. 2,3,4, we have:

$$n = n_1 + n_2 + n_3 \leq 2 \left(\sum_{j=1}^{3} H(B_j \mid X_j) \right) \tag{5}$$

Finally, since

$$H(B_j) = I(B_j \,;\, X_j) + H(B_j \mid X_j) = n_j + H(B_j \mid X_j)$$

by summing over j and substituting Eq. 5 we get:

$$H(B_1) + H(B_2) + H(B_3) \geq 1.5n$$

as required. □

5 Cryptographic Applications

In this section we describe the application of batch codes for amortizing the time complexity of private information retrieval (PIR), $\binom{n}{1}$-OT, and related cryptographic protocol problems. We refer the reader to Section 1.3 for background on the PIR problem and relevant previous work.

Amortized PIR. Recall that a PIR protocol allows a user $\mathcal{U}$ to retrieve the i-th bit (more generally, the i-th item) from a database x of length n while keeping the value i private. (The following discussion applies to both the computational setting for PIR, where typically there is only a single server holding x, and the information-theoretic setting where x is held by several servers.) We consider the $\binom{n}{k}$-PIR problem where the user is interested in retrieving k bits from the n-bit string x. This problem can obviously be solved by picking an arbitrary PIR protocol $\mathcal{P}$ and invoking it (independently) k times. The complexity of the resulting protocol is k times that of $\mathcal{P}$; in particular, the servers' time complexity is at least $k \cdot n$. Our goal is to obtain significant savings in the time complexity in comparison to the above naive solution, while only moderately increasing the communication complexity.

We start by observing that such an amortization can be achieved using *hashing*. This can be done with various choices of parameters; we outline a typical solution of this kind. The user, holding indices $i_1, \ldots, i_k$ of items it would like to retrieve, picks at random a hash function $h : [n] \to [k]$ from an appropriate family $\mathcal{H}$. (The

choice of h is independent of the indices $i_1, \ldots, i_k$.) It sends h to the server(s) and from now on both the user and the server(s) use h as a random partition of the indices of x into k buckets of size (roughly) n/k. This ensures that, except with probability $2^{-\Omega(\sigma)}$, the number of items hashed to any particular bucket is at most $\sigma \log k$. Next, to retrieve the k indices of x, the user applies the PIR protocol $\mathcal{P}$ to each bucket $\sigma \log k$ times. Except for $2^{-\Omega(\sigma)}$ probability, it will be able to retrieve all k items. It is not hard to see that the above hashing-based solution indeed achieves the desired amortization effect: the total size of all databases on which we invoke PIR is only $\sigma \log k \cdot n$, in comparison to kn in the naive solution.

The above hashing-based method has several disadvantages. First, even if the original PIR scheme is perfectly correct, the amortized scheme is not. (Alternatively, it is possible to modify this solution so that perfect correctness will be achieved, but at the expense of losing perfect privacy.) Second, the computational overhead over a single PIR invocation involves a multiplicative factor of $\sigma \log k$ – this is undesirable in general, and in particular makes this solution useless for small values of k. Finally, for efficiency reasons it might be necessary to reuse h, e.g., to let the server pick it once and apply it to the database in a preprocessing stage; however, for any fixed h there is (an efficiently computable) set of queries for which the scheme will fail.

Below, we show that *batch codes* provide a general reduction from $\binom{n}{k}$-PIR to standard $\binom{n}{1}$-PIR which allows to avoid the above disadvantages. More specifically, to solve the $\binom{n}{k}$-PIR problem, we fix some (n, N, k, m) batch code which will be used by the server(s) to encode the database x. The user, given the k indices $i_1, \ldots, i_k$ that it wants to retrieve, applies the code's batch-decoding procedure to that set; however, rather than directly read one bit from each bucket, it applies the PIR protocol $\mathcal{P}$ on each bucket to retrieve the bit it needs from it while keeping the identity of this bit private. Denoting by $C(n)$ and $T(n)$ the communication and time complexity of the underlying PIR protocol $\mathcal{P}$ and by $N_1, \ldots, N_m$ the sizes of buckets created by the batch code, the communication complexity of this solution is $\sum_{i=1}^{m} C(N_i)$ and its time complexity is $\sum_{i=1}^{m} T(N_i)$.[11] This reduction is *perfect* in the sense that it does not introduce any error nor compromise privacy. Hence, it can be applied to both information-theoretic and computational PIR protocols.

Batch codes may also be applied towards amortizing the *communication complexity* of PIR. This implies significant asymptotic savings in the information-theoretic setting, but is less significant in the computational setting (since there the communication complexity of retrieving a single item depends very mildly on

[11] For the purpose of this analysis, we ignore the computational overhead incurred by the encoding and decoding procedures. It is important to note though that encoding is applied to x only once (as long as the database is not changed) and that the cost of decoding, including the decision of which bit to read from each bucket, is quite small in our constructions.

n).

Two additional consequences for PIR. In addition to the direct application of batch codes to amortizing the cost of PIR, our techniques (specifically, the constructions of very short smooth codes) have two qualitatively interesting applications to PIR. The first is to PIR with preprocessing. In the model considered in [5], the servers preprocess the database in order to reduce the time it takes to answer a user's query. In contrast to the question of amortized PIR considered here, the savings in the time complexity should apply to each single query (rather than to a batch of queries together). The goal in this model is to minimize time, extra storage (in excess of n), and communication. The subset code construction yields the following interesting corollary: there exist PIR protocols with preprocessing in which all three quantities are *simultaneously* sublinear.

The idea is the following. Let $C(x)$ be the (ℓ, w) subset-encoding of the database x. It follows from the proof of Lemma 3.18 that the code is *perfectly* smooth, in the sense that its smooth decoding procedure probes each bit in the encoding with *exactly* the same probability. Hence, one can obtain PIR protocol with preprocessing as follows. At the preprocessing stage, compute $C(x)$ and store a *single* bit of the encoding at each server. (Note that this approach is radically different from the one in [5], where at least n bits are stored at each of a small number of servers.) Applying the smooth decoding procedure, the user approaches only the servers storing the bits it needs to read. Thus, the communication complexity is equal to the query complexity of the decoder. Privacy follows directly from the perfect smoothness requirement: each individual server is approached with equal probability, independently of the retrieved item i.

By Lemma 3.18, the expected number of bits read by the smooth decoder is $q = m/2^w$, where $m = \sum_{j=0}^{w} \binom{\ell}{j}$ is the length of the code (or the total storage). Also, $n = \binom{\ell}{w}$ is the length of the database. By an appropriate choice of parameters (e.g., $w = \sqrt{\ell}$) we have sublinear extra storage ($m = \sum_{j=0}^{w} \binom{\ell}{j} = (1 + o(1))n$), and sublinear communication complexity and time complexity ($q = m/2^w = o(n)$).

Another interesting corollary is that sublinear-communication information-theoretic PIR is possible even when the total number of bits stored at the servers is significantly smaller than $2n$. In all previous information-theoretic protocols from the literature, each server stores at least n bits of data (even when these bits are not necessarily physical bits of x [14, 5]), hence the minimal amount of possible storage is at least $2n$.

Applications to Oblivious Transfer and to other protocol problems. $\binom{n}{k}$-OT (k-out-of-n Oblivious Transfer) strengthen $\binom{n}{k}$-PIR by requiring that the user does not learn any information about x other than the k (physical) bits that it chose to retrieve [26, 12, 16]. Note that the above reduction from $\binom{n}{k}$-PIR to $\binom{n}{1}$-PIR

(using batch codes) cannot be directly applied for reducing $\binom{n}{k}$-OT to $\binom{n}{1}$-OT, since it allows the user to get m bits of information (rather than k), and even these are not necessarily physical bits of x. However, it is possible to obtain similar amortization for $\binom{n}{k}$-OT by using efficient reductions from this primitive to $\binom{n}{k}$-PIR (e.g., using [15, 24, 25, 19, 11]). Thus, the application of batch codes carries over to the $\binom{n}{k}$-OT primitive as well.

PIR is a useful building blocks in other cryptographic protocols. In particular, PIR has been used for various special-purpose secure computation tasks such as keyword search [10], distance approximations [13], statistical queries [7], and even for generally compiling a class of communication-efficient protocols into secure ones [23]. Most of these applications can benefit from the amortization results we obtain for PIR, at least in certain scenarios. For instance, in the keyword search application the cost of searching several keywords by the same user is amortized to the same extent as for the underlying PIR primitive.

Acknowledgements. We thank Amos Beimel and the anonymous reviewers for helpful comments.

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi. Deterministic simulation in LOGSPACE. In *Proc. 19th STOC*, pages 132-140, 1987.

[2] N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn, and D. Ron. Testing low-degree polynomials over $GF(2)$. In *Proc. RANDOM 2003*, pages 188-199.

[3] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proc. 7th STACS*, LNCS 415, pages 37–48, 1990.

[4] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond. Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval. In *Proc. 43rd FOCS*, pages 261-270, 2002.

[5] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. In *Proc. CRYPTO 2000*, LNCS 1880, pages 56–74. To appear in *Journal of Cryptology*.

[6] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proc. EUROCRYPT '99*, LNCS 1592, pages 402–414.

[7] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective Private Function Evaluation with Applications to Private Statistics In *Proc. 20th PODC*, pages 293-304, 2001.

[8] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness Conductors and Constant-Degree Expansion Beyond the Degree/2 Barrier. In *Proc. 34th STOC*, pages 659-668, 2002.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. 36th FOCS*, pages 41–51, 1995. Journal version: *J. of the ACM*, 45:965–981, 1998.

[10] B. Chor, N. Gilboa, and M. Naor Private information retrieval by keywords. Manuscript, 1998.

[11] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single Database Private Information Retrieval Implies Oblivious Transfer. In *Proc. EUROCRYPT 2000*, pages 122-138.

[12] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *C. ACM*, 28:637–647, 1985.

[13] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure Multiparty Computation of Approximations. In *Proc. 28th ICALP*, pages 927-938, 2001.

[14] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *Proc. 2nd RANDOM*, LNCS 1518, pages 200–217, 1998.

[15] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proc. 30th STOC*, pages 151-160, 1998. Journal version: *J. of Computer and System Sciences*, 60(3):592–629, 2000.

[16] O. Goldreich. Secure multi-party computation. Available at http://philby.ucsb.edu/cryptolib/BOOKS, February 1999.

[17] R. Impagliazzo, A. Wigderson. P=BPP unless E has Subexponential Circuits: Derandomizing the XOR Lemma. In *Proc. 29th STOC*, pages 220-229, 1997.

[18] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd STOC*, pages 80-86, 2000.

[19] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In *Proc. 24th STOC*, pages 723-732, 1992.

[20] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. 38th FOCS*, pages 364-373, 1997.

[21] Chi-Jen Lu, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Extractors: optimal up to constant factors. In *Proc. 35th STOC*, pages 602-611, 2003.

[22] F.J. MacWilliams and N.J. Sloane. The Theory of Error Correcting Codes. North-Holland, Amsterdam, 1977.

[23] M. Naor, and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proc. 33rd STOC*, pages 590-599, 2001.

[24] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. 31st STOC*, pages 245–254, 1999.

[25] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In *Proc. CRYPTO '99*, LNCS 1666, pages 573-590.

[26] M. O. Rabin. How to exchange secrets by oblivious transfer. Technical report TR-81, Harvard Aiken Computation Laboratory, 1981.

[27] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM* 38, pages 335-348, 1989.

[28] R. Shaltiel and C. Umans. Simple Extractors for All Min-Entropies and a New Pseudo-Random Generator. In *Proc. 42nd FOCS*, pages 648-657, 2001.

[29] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710-1722, 1996.

[30] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom Generators Without the XOR Lemma (Extended Abstract). In *Proc. 31st STOC*, pages 537-546, 1999.

[31] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proc. 33rd STOC*, pages 143-152, 2001.

[32] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller Codes. In *Proc. 42nd FOCS*, pages 638-647, 2001.

[33] A. C. C. Yao. Should tables be sorted? *J. of the ACM*, 28(3):615–628, 1981.