# Amortized Locally Decodable Codes for Insertions and Deletions

**Jeremiah Blocki**
Department of Computer Science, Purdue University, West Lafayette, IN, USA

**Justin Zhang**
Department of Computer Science, Purdue University, West Lafayette, IN, USA

## Abstract

Locally Decodable Codes (LDCs) are error correcting codes which permit the recovery of any single message symbol with a low number of queries to the codeword (the locality). Traditional LDC tradeoffs between the rate, locality, and error tolerance are undesirable even in relaxed settings where the encoder/decoder share randomness or where the channel is resource-bounded. Recent work by Blocki and Zhang initiated the study of Hamming *amortized* Locally Decodable Codes (aLDCs), which allow the local decoder to amortize their number of queries over the recovery of a small subset of message symbols. Surprisingly, Blocki and Zhang construct asymptotically ideal (constant rate, constant amortized locality, and constant error tolerance) *Hamming* aLDCs in private-key and resource-bounded settings. While this result overcame previous barriers and impossibility results for Hamming LDCs, it is not clear whether the techniques extend to Insdel LDCs. Constructing Insdel LDCs which are resilient to insertion and/or deletion errors is known to be even more challenging. For example, Gupta (STOC'24) proved that no Insdel LDC with constant rate and error tolerance exists even in relaxed settings.

Our first contribution is to provide a Hamming-to-Insdel compiler which transforms any amortized Hamming LDC that satisfies a particular property (consecutive interval querying) to amortized Insdel LDC while asymptotically preserving the rate, error tolerance and amortized locality. Prior Hamming-to-Insdel compilers of Ostrovsky and Paskin-Cherniavsky (ICITS'15) and Block et al. (FSTTCS'20) worked for arbitrary Hamming LDCs, but incurred an undesirable polylogarithmic blow-up in the locality. Our second contribution is a construction of an ideal amortized Hamming LDC which satisfies our special property (consecutive interval querying) in the relaxed settings where the sender/receiver share randomness or where the channel is resource bounded. Taken together, we obtain ideal Insdel aLDCs in private-key and resource-bounded settings with constant amortized locality, constant rate and constant error tolerance. This result is surprising in light of Gupta's (STOC'24) impossibility result which demonstrates a strong separation between locality and amortized locality for Insdel LDCs.

**2012 ACM Subject Classification** Mathematics of computing → Coding theory

**Keywords and phrases** Amortized Locally Decodable Codes, Insertion and Deletion Errors

**Digital Object Identifier** 10.4230/LIPIcs.ITC.2025.1

**Related Version** *Full Version*: `https://arxiv.org/abs/2507.03141`

**Funding** *Jeremiah Blocki*: Supported in part by NSF CAREER Award CNS-2047272.

## 1 Introduction

Locally Decodable Codes (LDCs) are a variant of error correcting codes which provide single-symbol recovery with highly efficient query complexity over a (possibly corrupted) codeword. Specifically, a LDC over alphabet $\Sigma$ is defined as a tuple of algorithms the encoding algorithm $\mathsf{Enc} : \Sigma^k \to \Sigma^n$ and the local decoder $\mathsf{Dec}^{\tilde{\boldsymbol{y}}} : [k] \to \Sigma$. Here, $k$ denotes the message length, $n$ denotes the codeword length and $\tilde{\boldsymbol{y}} \in \Sigma^n$ is a (possibly corrupted) codeword — for any message $\boldsymbol{x} \in \Sigma^k$, its corresponding codeword is $\boldsymbol{y} = \mathsf{Enc}(\boldsymbol{x})$ and $\tilde{\boldsymbol{y}}$ denotes a (possibly corrupted) copy of $\boldsymbol{y}$. The local decoder $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(i)$ is a randomized oracle algorithm that is given an index $i \in [k] := \{1, 2, \ldots, k\}$ as input and has oracle access to a (possibly corrupted) codeword $\tilde{\boldsymbol{y}}$. $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(i)$ attempts to output the $i$'th symbol of the message $\boldsymbol{x}$ after making *at most* $\ell$ queries to the symbols of $\tilde{\boldsymbol{y}}$. We require that for any index $i \in [k]$ and *any* $\tilde{\boldsymbol{y}}$ that is not too far from the uncorrupted codeword $\boldsymbol{y}$ (i.e $\mathsf{Dist}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) \leq \delta|\boldsymbol{y}|$ for a chosen distance metric $\mathsf{Dist}$) that $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(i)$ outputs the correct symbol $\boldsymbol{x}[i]$ with probability at least $1 - \varepsilon$. Broadly, codes which satisfy the above parameters are called $(\ell, \delta, \varepsilon)$-LDCs. The main parameters of interest are the *query locality* $\ell$, *error tolerance* $\delta$ and *rate* $k/n$. Ideally, we would like an LDC with constant locality, constant error tolerance, and constant rate simultaneously.

Within the classical setting of worst-case Hamming errors, i.e when the distance metric is chosen to be the Hamming distance metric and error patterns are introduced in a worst-case fashion, the trade-offs between the locality, error tolerance, and rate has been extensively studied. Even so, achievable parameters in this setting remain undesirably sub-optimal. Katz and Trevisan show that any LDC with constant locality $\ell \geq 2$ and constant error tolerance $\delta > 0$ must have non-constant rate [17], immediately ruling out the existence of an ideal LDC for worst case Hamming errors. Moreover, the best known constructions with constant locality and error tolerance have super-polynomial rate [24, 11]. Several relaxations have been made, such as allowing the local decoder to reject corrupted codewords [13, 2, 9, 18, 10], the assumption that the encoder and decoder share a cryptographic key [20, 14, 15] that is unknown to a probabilistic polynomial time channel, or the assumption that the channel is resource-bounded in other ways [1, 6]. Yet, even under these assumptions, we do not have an LDC with constant locality, constant error tolerance, and constant rate. The literature on LDCs and relaxed variants is vast. We provide an expanded discussion of the work related to LDCs in the full version of the paper [8].

Blocki and Zhang [7] introduced the notion of an *amortized* locally decodable code where the total number of queries to the codeword can be amortized by the total number of message bits recovered (see Definition 2). The local decoder for an amortized LDC takes as input a range $[L, R]$ (we will assume $R - L + 1 \geq \kappa$ for a suitable parameter $\kappa$) instead of a single index $i$ and attempts to output the entire substring $\boldsymbol{x}[L, R]$ instead of the single message symbol $x_i$. The amortized locality is $\ell/(R - L + 1)$ i.e., the total number of queries $\ell$ divided by the number of message symbols recovered. Surprisingly, they showed how to construct ideal amortized locally decodable codes in relaxed settings where the channel is resource-bounded or where the sender/receiver share a secret key. In particular, given any (sufficiently large) interval $[L, R] \subseteq [k]$ the local decoder can recover all of the corresponding message symbols $x[L], \ldots, x[R]$ while making at most $O(R - L)$ queries to the corrupted codeword. They also observed that the Hadamard code can have amortized locality approaching 1. By contrast, Katz and Trevisan show that codes with locality $\ell < 2$ do not exist unless the message length is a constant [17].

Given the surprising success in construction ideal amortized LDCs for Hamming channels, even in relaxed settings, it is natural to wonder whether or not it is possible to obtain similar results for insertions and deletions (*Insdel*) channels.

*Is it possible to obtain constant amortized locality, constant error tolerant, and constant rate Insdel* LDC*s when the channel is oblivious, resource-bounded, or when the sender/receiver share a cryptographic key?*

A construction of an ideal amortized LDC for Insdel channels would be especially surprising since when compared to known lowerbounds of Hamming LDCs, the current state of affairs is even worse for Insdel LDCs. Blocki et al. [5] proved the first separation result between Insdel and Hamming LDCs by proving that linear Insdel LDCs with locality $\ell = 2$ do not exist. This stands in contrast to Hamming LDCs, where constructions do exist, albeit with exponential rate (e.g. the Hadamard code). They additionally provide a exponential rate lowerbound for any constant locality Insdel LDC. Their lower bound holds in both traditional and relaxed (private-key and oblivious channel) settings, making progress towards a conjecture raised by Block et al. [5] that constant locality Insdel LDCs do not exist. Recently, Gupta resolved this conjecture by showing that constant query deletion codes do not exist by providing a randomized, oblivious adversarial deletion strategy [12]. The results of Gupta extend even to relaxed settings where the sender/receiver share random coins or where the channel is resource bounded i.e., even in relaxed settings it is impossible to construct constant query Insdel LDCs. Thus, it would be especially surprising to construct an ideal amortized LDC for Insdel channels even in relaxed settings with shared randomness or resource bounded channels.

On the side of constructions, Ostrovsky and Paskin-Cherniavsky provide the first InsDel LDC via a compilation of a Hamming LDCs into an Insdel LDCs [22] with a polylogarithmic blowup to the query complexity and only a constant blowup to the error-tolerance and rate. Their construction was revisited by Block et al. [4] who reproved and extended their results to construct Locally Correctable Codes (LCCs), a variant of LDCs which considers the local decodability of any codeword symbol. At a high level, their compiler takes a Hamming LDC encoding of the message, partitions it into blocks, and applies an Insdel code to each block. The local decoder then simulate the underlying Hamming decoder's queries by retrieving and decoding the corresponding Insdel-encoded block to recover the queried Hamming code symbol, where each answered Hamming query incurs a polylogarithmic query blowup.

Unfortunately, a black box approach of instantiating their Insdel LDC constructions with an amortizable Hamming LDC does not preserve the amortized locality. Even if all symbols recovered by the decoder are used, the amortized locality will still suffer a polylogarithmic blowup from the Hamming decoder simulation. Thus, it is unclear whether the Insdel LDC constructions of Ostrovsky and Paskin-Cherniavsky or Block et al. are amortization friendly.

## 1.1 Our Contributions

We provide the first framework for constructing amortized Insdel LDCs by modifying the compiler of Block et al. [4] to be amenable to amortization. Our resulting compiler takes in an amortized Hamming LDC, whose decoder satisfies a properties we define as *consecutive interval querying*, and outputs an Insdel amortized LDC with asymptotically equivalent parameters (amortized locality, rate, and error tolerance). More specifically, given an amortized Hamming LDC with constant rate, constant error tolerance, and constant amortized locality, whose decoder queries contiguous blocks of polylogarithmic size instead of individual message symbols, our compiler will yield an amortized Insdel LDC with constant rate, constant error tolerance, and constant amortized locality. While prior Insdel LDC compilers suffer from a polylogarithmic blow-up in query complexity of the underlying Hamming LDC, our amortized compiler preserves amortized locality as well as rate and error tolerance (up to constant factors). Thus, given any ideal Hamming amortized LDC which satisfies our consecutive interval querying property, we obtain an ideal Insdel amortized LDC achieving constant rate, error tolerance, and amortized locality.

▶ **Theorem** (Informal, see Corollary 13). *If there exists ideal (constant rate, constant error tolerance, and constant amortized locality) Hamming amortized* LDC*s whose decoder only queries consecutive blocks of size $\Omega(\text{polylog } n)$, then there exists ideal Insdel amortized* LDC*s (with asymptotically equivalent parameters).*

Furthermore, in the private-key setting, we construct the first consecutive interval querying Hamming LDC with constant rate, error tolerance, and constant *amortized* locality. We leave it as an open question whether one can construct a consecutive inverval querying Hamming LDC with constant *amortized* locality in the information theoretic setting. However, using our updated Hamming-to-Insdel compiler we obtain an ideal amortized Insdel LDC construction (constant rate, constant error tolerance and constant amortized locality) in the private-key setting. Specifically, the local decoder $\mathsf{Dec}^{\tilde{\boldsymbol{Y}}}(a,b)$ makes at most $O(b-a+\text{polylog}(n))$ queries to the corrupted codeword[1] $\tilde{\boldsymbol{Y}}$ and (whp) outputs all of the correct symbols $x[a], x[a+1], \ldots, x[b]$ in the interval $[a,b]$. As long as $b-a \geq \text{polylog}(n)$ the amortized locality is $O(1)$ per symbol recovered.

▶ **Theorem** (Informal, see Theorem 16 and Corollary 19). *For any $t, n \in \mathbb{N}$ such that $t \mid n$, there exists ideal private-key Hamming amortized* LDC*s with codeword length $n$ whose decoder only queries consecutive blocks of size $t$ and has constant amortized locality whenever decoding any consecutive interval of size at least $\omega(t \log n)$. This implies that there exists ideal private-key Insdel amortized* LDC*s.*

We show further that our private-key construction implies an ideal amortized Insdel LDC in resource-bounded settings i.e. when the channel has a bounded amount of some resource (e.g. space, circuit-depth, cumulative memory, etc.).

▶ **Theorem** (Informal, see Corollary 24). *If there exists ideal private-key Insdel amortized* LDC*s, there exists ideal resource-bounded Insdel amortized* LDC*s (with asymptotically equivalent parameters).*

Our results demonstrate definitively strong separations between locality and amortized locality for Insdel LDCs. In particular, Gupta [12] rules out the existence of *any* Insdel LDC with constant locality and error tolerance, even in relaxed settings and even with exponential rate.

## 1.2 Technical Overview

### Recalling the BBGKZ Compiler

Block et al. give a construction of a compiler taking any Hamming LDC and compiling it into an Insdel LDC[4]. We refer to their construction as the BBGKZ compiler. In the compiled encoding procedure, the message $\boldsymbol{x}$ is first encoded under the Hamming LDC as codeword $\boldsymbol{y}$, where $\boldsymbol{y}$ is then partitioned into size $\tau$ *Hamming* blocks $\boldsymbol{y} = \boldsymbol{w}_1 \circ \ldots \boldsymbol{w}_B$. Each Hamming block $\boldsymbol{w}_j$ along with its index $j$ is encoded using a constant rate Insdel code with the special property stating that the relative Hamming weight of every logarithmic-sized interval is at most 2/5 (The Schulman-Zuckerman-code [23] satisfies this property). Lastly, each block is pre/postpended with a zero buffer $0^{\phi\tau}$ of length $\phi\tau$, where $\phi \in (0,1)$, resulting in the *Insdel* block $\boldsymbol{w}'_j$ and the Insdel codeword $\boldsymbol{Y} = \boldsymbol{w}'_1 \circ \cdots \circ \boldsymbol{w}'_B$. Intuitively, the zero buffers $0^{\phi\tau}$ help to identify the beginning/end of each block after insertions/deletions.

Then, the compiled Insdel local decoding procedure, when given oracle access to the (possibly corrupted) Insdel codeword $\tilde{\boldsymbol{Y}}$, attempts to simulate the Hamming decoder with access to a corrupted codeword $\tilde{\boldsymbol{y}}$ that is close to the Hamming Codeword $\boldsymbol{y}$ in Hamming Distance. Specifically, when the Hamming decoder queries symbol $i$ of its expected oracle

[1] $\tilde{\boldsymbol{Y}}$ is the corrupted codeword output by the PPT-bounded channel who does not have the secret key under the constraint that the edit distance fraction between $\boldsymbol{Y}$ and $\tilde{\boldsymbol{Y}}$ is at most $2\delta$.

access to (possibly corrupted) codeword $\tilde{\boldsymbol{y}}$, the Insdel decoding procedure will identify the block $\boldsymbol{w}_j$ which contains $\boldsymbol{y}[i]$, (attempt to) locate the corresponding padded Insdel block $\boldsymbol{w}'_j$, undo the padding, run the Insdel decoder to recover the pre-compiled block $\boldsymbol{w}_j$, and return the corresponding symbol $\tilde{\boldsymbol{y}}[i]$. This query simulation process is facilitated by a subroutine we refer to as the `RecoverBlock` procedure which (attempts to) locate a particular Insdel block $\boldsymbol{w}'_j$ in the presence of corruptions using a noisy binary search procedure.

## Challenge 1: `RecoverBlock` is not Amortizable

Our first challenge in constructing amortizable Insdel LDCs is the observation that `RecoverBlock` performs asymptotically strictly more queries than the symbols it recovers. Since the codeword $\boldsymbol{Y}$ may contain insertions and deletions, in order for `RecoverBlock` to locate the Hamming decoder's queries, it performs a *noisy binary search* procedure and makes $O(\tau + \tau\text{polylog}(n))$ queries to recover the block corresponding to each query made by the Hamming decoder. The Hamming decoder only uses one of these symbols, but even if the decoder utilized all $\tau$ symbols from the recovered symbols, the amortized locality is still *at least* $O(\text{polylog}(n))$ per symbol recovered. Thus, we cannot achieve constant amortized locality by using the BBGKZ compiler as a black box.

Our insight to address this challenge is that `RecoverBlock` can be modified to recover a *consecutive interval* of blocks rather than just a single block. We refer to this modified procedure as `RecoverBlocks`, where on input block interval $[a, b] := \{a, \dots, b\}$, `RecoverBlocks` will recover pre-compiled blocks $\boldsymbol{w}_a, \dots, \boldsymbol{w}_b$ and make at most $O((b - a + 1)\tau + \tau\text{polylog}(n))$ queries. Hence, as long as $b - a + 1 \geq \text{polylog}(n)$, the ratio of recovered symbols to queried symbols will be constant.

## Challenge 2: Proving Correctness of `RecoverBlocks`

We show that the modified procedure `RecoverBlocks` preserves correctness. That is, a majority of the (corrupted) Insdel blocks $\tilde{\boldsymbol{w}}'_j$ satisfy the property running the search `RecoverBlocks` with any interval $[a, b]$ that contains $j \in [a, b]$ will yield the original Hamming block $\boldsymbol{w}_j$ except with negligible probability. Block et al. [4] proved that the procedure `RecoverBlock` will correctly find any *local-good block* and that most blocks will be *local-good* as long as the number of corruptions is bounded. The main technical challenge results from an inherent mismatch for `RecoverBlocks` because an interval $[a, b]$ may contain a mixture of blocks that are (resp. *are not*) locally-good. We show that `RecoverBlocks` successfully recovers any *local-good blocks* in its interval except with negligible probability.

We prove that correctness in a similar manner to the analysis of Block et al. [4]. We also utilize the notion of *local-good blocks*, where if block $\boldsymbol{w}'_j$ is $(\theta, \gamma)$-local-good, it will have at most $\gamma\tau$ corruption with the additional desirable property that any block interval $[a, b]$ surrounding the block, i.e. $j \in [a, b]$, has at least $(1 - \theta) \times |b - a + 1|$ blocks that also have at most $\gamma\tau$ corruption. Utilizing analysis from [4] we can pick our parameters to ensure that at least $(1 - \delta_h)$ fraction of our blocks are $(\theta, \gamma)$-local good. What we prove is that if block $\boldsymbol{w}'_j$ is $(\theta, \gamma)$-local good and if $j \in [a, b]$ then calling `RecoverBlocks` for the interval $[a, b]$ will recover the original (uncorrupted) $\boldsymbol{w}_j$ with high probability along with any other $(\theta, \gamma)$-good block in the interval $[a, b]$. Thus, for any partition $[a_1, b_1], \dots, [a_z, b_z]$ of $[B]$ calling `RecoverBlocks` with each interval $[a_i, b_i]$ would recover the corrupt (uncorrupted) $\boldsymbol{w}_j$ for at least $(1 - \delta_h)B$ blocks. We will use this observation to reason about the correctness of our decoding procedure. Intuitively, we can view the compiled Insdel decoder as simulating the Hamming decoder with a corrupted codeword whose fractional hamming distance is at most $\delta_h$ from the original codeword.

## Challenge 3: Consecutive Hamming Queries Needed For Amortized Decoding

The next challenge is utilizing the RecoverBlocks procedure to obtain amortized locality in the Insdel local decoding procedure. Similar to the BBGKZ compiler, our Insdel decoder with oracle access to (possibly corrupted) codeword $\tilde{\boldsymbol{Y}}$ simulates the compiled Hamming decoder's oracle access to (possibly corrupted) codeword $\tilde{\boldsymbol{y}}$. Our hope is that if the underlying Hamming LDC is amortizable then the compiled Insdel LDC will also be amortizable. In particular, we hope to group the Hamming decoder's queries $q_1, \ldots, q_\ell$ together into corresponding (disjoint) index intervals $[a_1, b_1], \ldots, [a_p, b_p]$ such that (1) $\sum_i |b_i + 1 - a_i| = \Theta(\ell)$, (2) $\{q_1, \ldots, q_\ell\} \subseteq \bigcup_i [a_i, b_i]$, and (3) the average size $\frac{1}{p} \sum_i |b_i + 1 - a_i|$ of each interval is large enough we can amortize over the calls to RecoverBlocks e.g., we need to ensure that the average interval size is at least $\frac{1}{p} \sum_i |b_i + 1 - a_i| = \Omega(\text{polylog} n)$. Unfortunately, in general, we cannot place any such structural guarantees on the queries made by an (amortizable) Hamming LDC.

We address this problem by introducing the notion of a *$t$-consecutive interval querying* Hamming LDCdecoder which is amortization-friendly under our Insdel compiler. Intuitively, the local decoder for a $t$-consecutive interval querying Hamming LDC accesses the (possibly corrupted) codeword $\tilde{\boldsymbol{y}}$ by querying for $t$-sized intervals instead of individual message symbols i.e., the decoder may submit the query $[i, i+t-1]$ and the output will be $\tilde{\boldsymbol{y}}[i, i+t-1]$. In more detail the Hamming decoder takes as input an interval $[L, R]$ of message symbols we would like to recover, makes queries for $t$-sized intervals of the (possibly corrupted) codeword $\tilde{\boldsymbol{y}}$ and then attempts to output $\boldsymbol{x}[L, R]$. Note that if the Hamming decoder makes $\ell/t$ interval queries to $\tilde{\boldsymbol{y}}$ this still corresponds to locality $\ell$ as the total number of codeword symbols accessed is $t \times (\ell/t) = \ell$.

Correspondingly, when the Hamming decoder is $t$-consecutive interval querying, our Insdel decoder will make at most $2\lceil \ell/t \rceil$ calls to RecoverBlocks on intervals of size $t$. The resulting query complexity is roughly $\ell \times \left(\frac{\tau \log^3 n}{t} + O(1)\right)$, and so, in other words, the query complexity blow-up is by a factor of $O\left(\frac{\tau \log^3 n}{t}\right)$. This implies that if there exists a $t$-consecutive interval querying Hamming decoder for $t = \Omega(\tau \log^3 n)$ with constant amortized locality, our compiled Insdel construction will have constant amortized locality. Thus, the primary question is whether or not it is possible to construct $t$-consecutive interval querying Hamming LDCs with constant amortized locality. We leave this as an open question for worst-case errors, but show that it is possible in settings where the channel is computationally/resource restricted.

## Challenge 4: A $t$-consecutive interval querying aLDC in Private-Key Settings

We present a $t$-consecutive interval querying private-key amortized Hamming LDC (paLDC) construction with constant rate, constant amortized locality, and constant error tolerance i.e. ideal parameters. As a starting point we use the one-time private-key Hamming LDC constructed by Ostrovsky et al. [21]. Blocki and Zhang show that their construction is amortizable and that their construction may be extended to multi-round secret-key reuse with cryptographic building blocks [7]. For simplicity, we focus on the single round construction, which is information-theoretically secure as long as the channel is computationally bounded.

The single round construction first generates the secret key as a randomly drawn permutation $\pi$ and a random string $\boldsymbol{r}$. To encode the message $\boldsymbol{x}$, it is partitioned into blocks $\boldsymbol{x} = \boldsymbol{e}_1 \circ \ldots \boldsymbol{e}_B$ and each block $\boldsymbol{e}_j$ is encoded as $\boldsymbol{e}'_j = \mathsf{Enc}(\boldsymbol{e})$ by a constant rate, constant error tolerance Hamming code Enc. Lastly, the bits of the encoded blocks are permuted by $\pi$ and the random string $\boldsymbol{r}$ is xor'd i.e. the resulting codeword is $\boldsymbol{y} = \pi(\boldsymbol{e}'_1 \circ \cdots \circ \boldsymbol{e}'_B) \oplus \boldsymbol{r}$. The local decoder can recover any block $\boldsymbol{e}_j$ by finding the indices of its corresponding encoded

$$\begin{tikzcd}
\boldsymbol{x} = & \boldsymbol{e}_1 \arrow[d, "\mathsf{Enc}"] & \circ \cdots \circ \arrow[d, "\mathsf{Enc}"] & \boldsymbol{e}_B \arrow[d, "\mathsf{Enc}"] & \\
& \boldsymbol{e}'_1 \arrow[d, phantom, "\shortparallel"] & \circ \cdots \circ \arrow[d, phantom, "\shortparallel"] & \boldsymbol{e}'_B \arrow[d, phantom, "\shortparallel"] & \\
& (\boldsymbol{\epsilon}_1 \circ \cdots \circ \boldsymbol{\epsilon}_\beta) \arrow[d, "\pi(\cdot)\oplus\boldsymbol{r}"] & \circ \cdots \circ \arrow[d, "\pi(\cdot)\oplus\boldsymbol{r}"] & (\boldsymbol{\epsilon}_{(B-1)\beta+1} \circ \cdots \circ \boldsymbol{\epsilon}_{B\beta}) \arrow[d, "\pi(\cdot)\oplus\boldsymbol{r}"] & \\
\boldsymbol{y} = & \Big( \boldsymbol{\epsilon}_{\pi(1)} \circ \cdots \circ \boldsymbol{\epsilon}_{\pi(\beta)} & \circ \cdots \circ & \boldsymbol{\epsilon}_{\pi((B-1)\beta+1)} \circ \cdots \circ \boldsymbol{\epsilon}_{\pi(B\beta)} \Big) & \oplus \boldsymbol{r}
\end{tikzcd}$$

**Figure 1** An overview of our private-key Hamming aLDC decoder, satisfying correct decoding (of any $\boldsymbol{e}_j$ Hamming block) and consecutive interval querying (of any $\boldsymbol{\epsilon}_{\pi(r)}$ sub-block) simultaneously.

block $\boldsymbol{e}'_j$, reversing the permutation and random masking, and running the Hamming code decoder $\mathsf{Dec}$ to recover $\boldsymbol{e}_j$. Unfortunately, the local decoder is not $t$-consecutive interval querying since its queries are not in a contiguous interval. The main issue is that the application of the permutation $\pi$ removes any pre-existing block structure in the codeword $\boldsymbol{y}$ i.e., if the local decoder wants to query for consecutive (pre-permutation) code symbols $(\boldsymbol{e}'_1 \circ \cdots \circ \boldsymbol{e}'_B)[s+1, s+\ell]$, it would have to query $\boldsymbol{y}[\pi(s+1)], \ldots, \boldsymbol{y}[\pi(s+\ell)]$, which are no longer consecutive with high probability.

A first attempt to remedy this issue is to permute at the block level instead of the bit level i.e. we draw permutation $\pi$ over $[B]$ and set the codeword instead as $\boldsymbol{y} = (\boldsymbol{e}'_{\pi(1)} \circ \cdots \circ \boldsymbol{e}'_{\pi(B)}) \oplus \boldsymbol{r}$. While this admits contiguous access when recovering any block $\boldsymbol{e}_j$ it defeats the original purpose of the permutation. The permutation $\pi$ (and the one-time pad $\boldsymbol{r}$) ensure that (whp) an adversarial channel cannot produce too many corruptions in any individual block. If we permute at the block level then it is trivial for an attacker to corrupt entire blocks. In particular, the adversary could focus their entire error budget to corrupt symbols at the start of the codeword. This simple strategy will always corrupt a constant fraction of blocks $\boldsymbol{e}'_j$. Thus, the proposed code will not satisfy correct decoding.

Our primary insight is that we can apply the permutation at an intermediate "sub-block" granularity to ensure correctness and simultaneously support consecutive interval queries. In particular, each encoded block $\boldsymbol{e}'_j$, for $j \in [B]$, is further partitioned into $\beta$ sub-blocks $\boldsymbol{e}'_j = (\boldsymbol{\epsilon}_{(j-1)\beta+1} \circ \cdots \circ \boldsymbol{\epsilon}_{j\beta})$ and the permutation is applied over sub-blocks $\boldsymbol{\epsilon}_r$, for $r \in [B\beta]$, rather than the Hamming blocks $\boldsymbol{e}'_j$ themselves i.e. the codeword is

$$\boldsymbol{y} = \left(\boldsymbol{\epsilon}_{\pi(1)} \circ \cdots \circ \boldsymbol{\epsilon}_{\pi(\beta)} \circ \cdots\cdots \circ \boldsymbol{\epsilon}_{\pi(B\beta)}\right) \oplus \boldsymbol{r}.$$

Intuitively, while a constant fraction of the sub-blocks $\boldsymbol{\epsilon}_r$ can still be corrupted with non-negligible probability, as long as the parameter $\beta$ is suitably large (e.g., $\beta = O(\text{polylog} n)$), we can ensure that, except with negligible probability, the overall error in each encoded block $\boldsymbol{e}'_j$ will still be at most a $\delta$-fraction. Thus, except with negligible probability we can recover $\boldsymbol{e}_j$ (the uncorrupted content in block $j$) by making $\beta$ consecutive interval queries to obtain $\boldsymbol{\epsilon}_r$ for each $r \in [(j-1)\beta+1, j\beta]$. This is exactly what we needed to achieve amortization with our Ideal Insdel compiler.

The above construction assumes that the sender and receiver share a secret key and only allows the sender to transmit one encoded message. However, if the channel is probabilistic polynomial time (PPT) then the construction can be extended allow for the sender to transmit multiple messages using standard cryptographic tools e.g., pseudo-random functions[21, 7]. Furthermore, if we assume stronger resource bounds on the channel (space, sequential depth etc...) then one can use cryptographic puzzles to completely eliminate the requirement for a secret key using ideas from prior works e.g., [1, 6, 3].

### Putting it All Together: Ideal Insdel aLDCs in Relaxed Settings

With the building blocks established prior, the aLDC Hamming-to-Insdel compiler and the consecutive interval querying, ideal Hamming paLDC, we construct the first ideal Insdel aLDC within relaxed settings (private-key and resource-bounded). Naturally, we first construct a private-key, ideal Insdel aLDC by directly using our compiler on the $t$-consecutive interval querying, ideal Hamming paLDC. For compiler block size $\tau$, our construction achieves constant amortized locality by setting $t = \Omega(\tau \log^3 n)$ as discussed previously.

The remaining challenge lies in showing that the compiler's correctness is preserved in private-key settings. Intuitively, we argue via a reduction to the security of the underlying Hamming paLDC. Suppose there exists a probabilistic polynomial time adversary $\mathcal{A}$ who introduces an Insdel corruption pattern into compiled codeword $\boldsymbol{Y}$ of their chosen message $\boldsymbol{x}$ that causes a decoding error with non-negligible probability. Then, we can construct a probabilistic polynomial time adversary $\mathcal{B}$ who introduces a Hamming error pattern into (pre-compiled) codeword $\boldsymbol{y}$ by (1) using our compiler procedure to transform Hamming codeword $\boldsymbol{y}$ into the compiled Insdel codeword $\boldsymbol{Y}$, (2) Giving compiled Insdel codeword $\boldsymbol{Y}$ to adversary $\mathcal{A}$ and receiving corrupted Insdel codeword $\tilde{\boldsymbol{Y}}$, and finally, (3) constructing the pre-compiled codeword $\tilde{\boldsymbol{y}}$ by the same simulation process done by the compiled decoder via the `RecoverBlocks` procedure. To streamline the reduction argument, we draw inspiration from Block and Blocki [3] and repackage our compiler into a convenient form for our reduction. All together, we have an Ideal Insdel paLDC.

Next, we generalize the paLDC construction to construct an aLDC for resource-bounded settings (raLDC). Similarly, our first step is constructing a $t$-consecutive interval querying Hamming raLDC. To do this, we make use of the Hamming private-key-to-resource-bounded LDC compiler by Ameri et al. [1], which was observed by Blocki and Zhang to be amortizable with only a constant blow-up to the amortized locality [7]. At a high level, their compiler makes use of a cryptographic primitive known as a *cryptographic puzzle* to embed the secret key within the codeword.

More specifically, cryptographic puzzles consists of two algorithms PuzzGen and PuzzSolve. On input seed $\boldsymbol{c}$, PuzzGen outputs a puzzle $\boldsymbol{Z}$ with solution is $\boldsymbol{c}$ i.e. $\mathsf{PuzzSolve}(\boldsymbol{Z}) = \boldsymbol{c}$. The security requirement states that adversary $\mathcal{A}$ in a defined algorithm class $\mathbb{R}$, cannot solve the puzzle $\boldsymbol{Z}$ and cannot even distinguish between tuples $(\boldsymbol{Z}_0, \boldsymbol{c}_0, \boldsymbol{c}_1)$ and $(\boldsymbol{Z}_1, \boldsymbol{c}_0, \boldsymbol{c}_1)$ for random $\boldsymbol{c}_i$ and $\boldsymbol{Z}_i = \mathsf{PuzzGen}(\boldsymbol{c}_i)$.

Then, the Hamming raLDC compiler takes a paLDC $(\mathsf{Gen}_\mathsf{p}, \mathsf{Enc}_\mathsf{p}, \mathsf{Dec}_\mathsf{p})$, and on on message $\boldsymbol{x}$ will (1) generate the secret key $\mathsf{sk} \leftarrow \mathsf{Gen}_\mathsf{p}(1^\lambda; \boldsymbol{c})$ with some random coins $\boldsymbol{c}$ (2) Compute the paLDC encoding of the message $\boldsymbol{y}_\mathsf{p} \leftarrow \mathsf{Enc}_{\mathsf{p},\mathsf{sk}}(\boldsymbol{x})$ (3) Compute an encoding $\boldsymbol{y}_*$ of the puzzle $\boldsymbol{Z}$ using a code with low locality that recovers the entire message (known as an LDC$^*$) and (4) output $\boldsymbol{y} = \boldsymbol{y}_\mathsf{p} \circ \boldsymbol{y}_*$. Intuitively, a resource-bounded adversary will not be able to solve the puzzle, while the raLDC decoder Dec will have sufficient resources to solve the puzzle, recover the secret key sk, and run the amortized local decoder $\mathsf{Dec}_{\mathsf{p},\mathsf{sk}}$.

We make an additional observation that if the compiler is instantiated with a $t$-consecutive interval querying paLDC, then the resulting raLDC will also be $t$-consecutive interval querying. Thus, when applied to our amortized Hamming-to-Insdel compiler with $t = \Omega(\tau \log^3 n)$, we acheive constant amortized locality. Lastly, the same reduction argument as in the ideal

Insdel paLDC construction may be used, with an additional subtle detail of allowing sufficient resources for the reduction. That is, for the class of resource-bounded channels/adversaries $\mathbb{R}$ that our Insdel aLDC is secure against, the underlying Hamming aLDC must be secure against a wider class of adversaries $\overline{\mathbb{R}}$, where $\overline{\mathbb{R}}$ include adversaries $\mathcal{A} \in \mathbb{R}$ whom can additionally compute the simulation process in the private-key reduction.

## 1.3 Preliminaries

### Notation

Let $x \leftarrow y$ represent a variable assignment of $x$ as $y$ and $x \overset{\$}{\leftarrow} X$ denote a uniform random assignment of $x$ from space $X$. Let $\circ$ denote the concatenation function. For $l \leq r$, $[l, r] := \{l, l+1, \ldots, r\}$ denotes an interval from $l$ to $r$ inclusive of its ends points. Similarly, $[l, r) := \{l, l+1, \ldots, r-1\}$ denotes an interval from $l$ to $r$ exclusive of the right endpoint. Over an alphabet $\Sigma$, bolded notations $\boldsymbol{x}$ denote vectors/words while non-bolded $x \in \Sigma$ denote single symbols. For a word $\boldsymbol{x}$, $x[i]$ denotes the symbol of $\boldsymbol{x}$ at index i, while $\boldsymbol{x}[l, r] := (x[l], x[l+1], \ldots, x[r])$ denotes the subword of $\boldsymbol{x}$ projected to indices in interval $[l, r]$. It will also be useful to define a short-hand for retrieving symbols from "blocks" of a word $\boldsymbol{x}$: $\boldsymbol{x}\langle i \rangle_b := \boldsymbol{x}[(i-1)b+1, ib]$ for any block size $b$ and index $i$.

For any words $\boldsymbol{x}, \boldsymbol{y} \in \Sigma^n$, $\mathsf{Ham}(\boldsymbol{x}, \boldsymbol{y})$ denotes the hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ i.e. $|\{i \in [n] : x_i \neq y_i\}|$. Further, for $\boldsymbol{z} \in \Sigma^*$, $\mathsf{ED}(\boldsymbol{x}, \boldsymbol{z})$ denotes the edit distance between $\boldsymbol{x}$ and $\boldsymbol{z}$ i.e. the number of symbol insertions and deletions to transform $\boldsymbol{x}$ into $\boldsymbol{z}$. We say $\boldsymbol{x}$ and $\boldsymbol{y}$ (resp. $\boldsymbol{x}$ and $\boldsymbol{z}$) are $\delta$-hamming-close (resp. $\delta$-edit-close) when $\mathsf{Ham}(\boldsymbol{x}, \boldsymbol{y}) \leq \delta|x|$ (resp. $\mathsf{ED}(\boldsymbol{x}, \boldsymbol{z}) \leq 2\delta|\boldsymbol{x}|$).

We recall the formal definitions of an LDC and an amortized LDC.

▶ **Definition 1** (LDC). *A $(n, k)$-code $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ (over $\Sigma$) is a $(\ell, \delta, \varepsilon)$-locally decodable code* (LDC) *for hamming errors (resp. insDel errors) if for every $\boldsymbol{x} \in \Sigma^k$, $\tilde{\boldsymbol{y}} \in \Sigma^*$ such that $\tilde{\boldsymbol{y}}$ is $\delta$-hamming-close (resp. $\delta$-edit-close) to $\mathsf{Enc}(\boldsymbol{x})$, and every index $i \in [k]$, we have $\Pr[\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(i) = x[i]] \geq 1 - \varepsilon$, and $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}$ makes at most $\ell$ queries to $\tilde{\boldsymbol{y}}$.*

▶ **Definition 2** (aLDC [7]). *A $(n, k)$-code $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ (over $\Sigma$) is a $(\alpha, \kappa, \delta, \epsilon)$-amortizeable LDC (*aLDC*) for hamming errors (resp. insDel errors) if for every $\boldsymbol{x} \in \Sigma^k$, $\tilde{\boldsymbol{y}} \in \Sigma^*$ such that $\tilde{\boldsymbol{y}}$ is $\delta$-hamming-close (resp. $\delta$-edit-close) to $\mathsf{Enc}(\boldsymbol{x})$, and every interval $[L, R] \subseteq [k]$ or size $R - L + 1 \geq \kappa$ we have $\Pr[\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(L, R) = (x_i : i \in [L, R])] \geq 1 - \varepsilon$, and $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}$ makes at most $\alpha \times (R - L + 1)$ queries to $\tilde{\boldsymbol{y}}$.*

Throughout the paper, it will be assumed codes are over the binary alphabet i.e. $\Sigma = \{0, 1\}$ unless specified otherwise. The primary metrics of interest for Hamming/Insdel aLDCs are the rate $k/n$, amortized locality $\alpha$, and the error/edit tolerance $\delta$.

## 1.4 Organization

We start with reintroducing and modifying the encoding and decoding procedures of the BBGKZ compiler in Section 2. The modification will take various steps organized into the following subsections: in Subsection 2.1, the notion of good blocks and intervals are introduced to analyze our new `RecoverBlocks` procedure in Subsection 2.2. We then present the overall construction of our Hamming-to-Insdel aLDC compiler in Subsection 2.3.

In Section 3, we present our construction of an ideal Insdel aLDCs in settings where the encoder/decoder share a secret-key (paLDC). More specifically, this construction first relies on a modified Hamming paLDC construction in Subsection 3.1, where the local decoder performs consecutive interval queries. The ideal Hamming paLDC is then compiled into our ideal Insdel paLDC in Subsection 3.2. Lastly, in Section 4, we present our construction of an ideal Insdel aLDCs in settings where the channel is resource-bounded (raLDC). Section 4 follows a symmetric structure to the previous section: we present a consecutive interval querying Hamming raLDC construction in Subsection 4.1, which is then compiled into an ideal Insdel raLDC construction in Subsection 4.2.

## 2 The Hamming aLDC to Insdel aLDC Compiler

Our results crucially rely on a modified BBGKZ compiler suited for amortized local decoding. We start with the encoding procedure. Interestingly, we will not need to make any changes to the encoding procedure of the BBGKZ compiler, which takes as parameters the block size $\tau \in \mathbb{N}$ and padding rate $\phi \in (0,1)$: the message $\boldsymbol{x}$ will first be encoded into a codeword $\boldsymbol{y}$ using a Hamming LDC. Next, the codeword $\boldsymbol{y}$ is broken up into blocks of equal size $\tau$ as $\boldsymbol{y} = \boldsymbol{w}_1 \circ \dots \boldsymbol{w}_B$, where each block and its index $(i \circ \boldsymbol{w}_i)$ is 1) encoded with a specifically chosen Insdel code 2) pre-and-post-pended with $\phi\tau$ many 0s. The result is the overall codeword $\boldsymbol{Y}$. Formally, we describe the encoder Enc as first computing the Hamming codeword $\boldsymbol{y}$, then applying a Hamming-to-Insdel compiler EncCompile to transform it into an Insdel codeword $\boldsymbol{Y}$. The encoder Enc and compiler EncCompile are formally described below.

Let $\mathcal{C}_{\mathrm{h}} = (\mathsf{Enc}_{\mathrm{h}}, \mathsf{Dec}_{\mathrm{h}})$ be a Hamming LDC. Let $\mathcal{C}_{\mathrm{insdel}} = (\mathsf{Enc}_{\mathrm{insdel}}, \mathsf{Dec}_{\mathrm{insdel}})$ be an Insdel binary code.

```
EncCompile_ϕ(y)
Input: Hamming codeword $\boldsymbol{y} \in \{0,1\}^m$, padding rate $\phi \in (0,1)$.
Output: Insdel codeword $\boldsymbol{Y} \in \{0,1\}^n$.
 1. Parse $\boldsymbol{y} = \boldsymbol{w}_1 \circ \cdots \circ \boldsymbol{w}_{|\boldsymbol{y}|/\tau}$ where $\boldsymbol{w}_j \in \{0,1\}^\tau$ for all $j \in [\ |\boldsymbol{y}|/\tau\ ]$.
 2. Compute $\boldsymbol{w}'_j \leftarrow 0^{\phi\tau} \circ \mathsf{Enc}_{\mathrm{insdel}}(j \circ \boldsymbol{w}_j) \circ 0^{\phi\tau}$ for all $j \in [\ |\boldsymbol{y}|/\tau\ ]$.
 3. Output $\boldsymbol{Y} = \boldsymbol{w}'_1 \circ \cdots \circ \boldsymbol{w}'_{|\boldsymbol{y}|/\tau}$.
```

```
Enc(x)
Parameters: Block size $\tau \in \mathbb{N}$, Padding rate $\phi \in (0,1)$.
Input: message $\boldsymbol{x} \in \{0,1\}^k$.
Output: Insdel codeword $\boldsymbol{Y} \in \{0,1\}^n$.
 1. Compute $\boldsymbol{y} \leftarrow \mathsf{Enc}_{\mathrm{h}}(\boldsymbol{x})$.
 2. Output $\boldsymbol{Y} \leftarrow \mathsf{EncCompile}(\boldsymbol{y})$.
```

Our main modifications to the BBGKZ compiler will be in the construction of the decoder Dec. Recall that the original BBGKZ decoder simulated oracle access to the Hamming LDC decoder $\mathsf{Dec}_{\mathrm{h}}$ by recovering the Hamming blocks corresponding to the requested queries of $\mathsf{Dec}_{\mathrm{h}}$. Since insertions and deletions in the corrupted codeword $\tilde{\boldsymbol{Y}}$ modify the structure of the blocks, this simulation is underpinned by a *noisy binary search* procedure we denote as the `RecoverBlock` procedure. More specifically, the `RecoverBlock` procedure on input block index $j$ will iteratively cut an initial search radius $[1, \tilde{n}]$ by a fractional amount, until the search radius is a constant size larger than the block size $\tau$. To decide how to cut the search radius on each iteration, the procedure samples $N = \theta(\log^2 n)$ blocks indices by decoding $N$ noisy $\boldsymbol{w}'_{j_1}, \dots \boldsymbol{w}'_{j_b}$ blocks within the middle of the search radius to $(j_b \circ \boldsymbol{w}_{j_b})$. On the block indices $j_1, \dots, j_b$, their median $j_{\mathsf{med}}$ is compared to the desired block index $j$ : depending if $j < j_{\mathsf{med}}$, `RecoverBlock` will cut off a fraction of the front or the back of the search radius.

Since the search radius decreases by a constant fraction each iteration, there are $O(\log n)$ iterations, where on each iteration, $\theta(log^2 n)$ blocks are read of $O(\tau)$ size. Thus, the query complexity to answer a single query of $\mathsf{Dec}_{\mathrm{h}}$ is $O(\tau \log^3 n)$.

Unfortunately, amortization is not feasible with the current Hamming decoder simulation method. Intuitively, since the `RecoverBlock` procedure must query a multiplicative polylog $n$ factor to recovers $\tau$ symbols, the amortized locality will be at least $\Omega(\text{polylog } n)$.

Our solution will recover multiple blocks at once using the observation that the `RecoverBlocks` procedure can be extended from recovering a single block to an interval of contiguous blocks. That is, we will modify the noisy binary search procedure to take in a block interval $a \le b \in [B]$, and we change the stopping condition to be linear to the interval size $b - a + 1$. Further, the consideration on how to shrink the search radius will take into account the respective endpoints $a$ and $b$ of the interval rather than a single block $j$.

While this change is relatively straight forward on paper, the analysis for the noisy binary search procedure BBGKZ compiler does not immediately follow for our modified procedure `RecoverBlocks`. Thus, to prove correctness for the modified `RecoverBlocks` procedure, we reintroduce the good block analysis from Block et al. [4], and modify their proof structure to take into account the recovery of an interval of blocks instead of a single block.

## 2.1 Good Blocks and Intervals

In this subsection, we introduce good block notation to analyze the modifications we will make to the BBGKZ decoder and its sub-routine `RecoverBlock`.

### Additional Notation

First, we fix notation consistent with the encoder definition above for the rest of the paper. We will refer to the Hamming encoding $\boldsymbol{y}$ in step 1 as the *Hamming encoding* and the final codeword $\boldsymbol{Y}$ as the *Insdel encoding* or simply as the *encoding*. Let $\boldsymbol{x} \in \{0,1\}^k$ be the message of size $k$ and let $\boldsymbol{y} = \mathsf{Enc}_{\mathrm{h}}(\boldsymbol{x}) \in \{0,1\}^m$ be the Hamming encoding with size $m$. Let $B = m/\tau$ be the number of blocks. Define $\beta$ such that the rate of the overall encoding is $1/\beta$. Similarly, define $\beta_{\text{insdel}}$ such that the rate of the Insdel code $\mathcal{C}_{\text{insdel}}$ is $1/\beta_{\text{insdel}}$. Then, the rate of the Hamming encoding is $(\beta_{\text{insdel}} \log B + 2\phi)/\beta$. Let $\delta_{\mathrm{h}}$ and $\delta_{\text{insdel}}$ be the error tolerances of the Hamming encoding and the Insdel code respectively. Let each $\boldsymbol{w}_j = \boldsymbol{y}\langle j \rangle_\tau \in \{0,1\}^\tau$ denote *Hamming block* $j$. Similarly, each $\boldsymbol{w}'_j \in \{0,1\}^{\beta\tau}$ denotes the *Insdel block/ block* $j$. Then, the overall codeword is $\boldsymbol{Y} \in \{0,1\}^n$ has length $n = \beta m$. We consider any corrupted codeword $\tilde{\boldsymbol{Y}} \in \{0,1\}^{\tilde{n}}$ such that $\mathsf{ED}(\boldsymbol{Y}, \tilde{\boldsymbol{Y}}) \le 2\delta n$.

We define the *block decomposition* as a mapping from codeword symbols to their blocks post corruption.

▶ **Definition 3.** *A block decomposition of* $\tilde{\boldsymbol{Y}}$ *is a non-decreasing mapping* $\phi : [\tilde{n}] \to [B]$.

By definition, the pre-image of each block decomposition defines a partition of $[\tilde{n}]$ into $B$ intervals $\{\phi^{-1}(j) : j \in [B]\}$. Each of these intervals define the block structure, where the j'th interval defines the j'th block in the corrupted codeword. In other words, for each $j \in [B]$, $\phi^{-1}(j)$ are all the indices corresponding to block $j$.

As in the BBGKZ compiler, our modified decoder will rely on a noisy recovery process, where the decoder will need to locate blocks without knowing the block boundaries that have been modified by insdel errors. For any search interval $[l, r) \subseteq [\tilde{n}]$, it will be helpful to consider its corresponding block interval i.e. the smallest interval $[L, R-1] \subseteq [B]$ such that $[l, r) \subseteq [\tau L, \tau(R-1)]$. Block intervals are defined formally below.

▶ **Definition 4** (Block Interval). *The block interval of an interval $I = [l, r) \subseteq [\tilde{n}]$ is defined as $\bigcup_{i=l}^{r-1} \phi^{-1}(\phi(i)) \subseteq [\tilde{n}]$. An interval $I$ is a block interval if the block interval of $I$ is itself. Equivalently, every block interval has the form $\mathcal{BI}[a, b] := \bigcup_{i=a}^{b} \phi^{-1}(j)$ for some $a, b \in [B]$.*

We say a block is *good* if it does not have too many edit corruptions. Intuitively, good blocks will correspond to Insdel blocks that are correctly decodable, except with negligible probability.

▶ **Definition 5** ($\gamma$-Good Block). *For $\gamma \in (0, 1)$ and $j \in [B]$, a block $j$ is $\gamma$-good (with respect to $\tilde{\boldsymbol{Y}}$) if $\mathsf{ED}(\tilde{\boldsymbol{Y}}[\phi^{-1}(j)], \boldsymbol{w}'_j) \leq \gamma\tau$. Otherwise, block $j$ is $\gamma$-bad.*

*Good blocks* may be naturally extended to *good intervals*, where the summed edit corruptions of each block within the interval is not too much. Additionally, we constrain the number of bad blocks in any good interval.

▶ **Definition 6** ($(\theta, \gamma)$-Good Interval). *A block interval $\mathcal{BI}[a, b]$ is $(\theta, \gamma)$-good (with respect to $\tilde{\boldsymbol{Y}}$) if the following hold:*
1. $\sum_{j=a}^{b} \mathsf{ED}\left(\tilde{\boldsymbol{Y}}[\phi^{-1}(j)], \boldsymbol{w}'_j\right) \leq \gamma\tau(b - a + 1)$.
2. *The number of $\gamma$-bad blocks in the block interval $\mathcal{BI}[a, b]$ is at most $\theta \times (b - a + 1)$.*

*Otherwise, the interval is $(\theta, \gamma)$-bad.*

Lastly, we define the notion of a locally good block, which captures the idea that any interval including such a block must also be good.

▶ **Definition 7** ($(\theta, \gamma)$-Local Good Block). *For $\theta, \gamma \in (0, 1)$, block $j \in [B]$ is $(\theta, \gamma)$-local good (with respect to $\tilde{\boldsymbol{Y}}$) if for every $a, b \in [B]$ such that $a \leq j \leq b$, the interval $\mathcal{BI}[a, b]$ is $(\theta, \gamma)$-good. Otherwise, block $j$ is $(\theta, \gamma)$-locally bad.*

## 2.2 Extending `RecoverBlock` to `RecoverBlocks`

We start with recalling specific details of the `RecoverBlock` procedure. The `RecoverBlock` procedure for recovering a single block takes as input the desired block index $j \in [B]$ and the initial search radius $[1, \tilde{n}]$. The procedure will iteratively split the search radius into three contiguous, linear-sized segments, a beginning, a middle, and an end segment, where it will choose to either cut the beginning or the end segment. To decide which, it randomly samples $N = \theta(\log^2 n)$ indices $i_1, \ldots, i_N$ within the middle segment and recover their respective block indices $j_1, \ldots, j_N$. More specifically, for each index $i_b$, a subroutine `Block-Decode` is called which queries a $O(\tau)$ interval around $i_b$ to decode the noisy block containing $(j \circ \boldsymbol{w}_j)$. The median $j_{\mathsf{med}}$ of these retrieved indices is then calculated, and, depending if $j < j_{\mathsf{med}}$ or $j \geq j_{\mathsf{med}}$, cuts either the end or beginning segment from the search interval respectively. The binary search continues until the search radius is linear in the size of the block for a sufficiently small constant.

Lastly, an interval decoding procedure is then run over the symbols in the final search interval, where it is guaranteed to recover any local-good block located within the interval. The subroutines used, `Block-Decode` and the final interval decoding, are based on local and global variations of the SZ-code decoding algorithm, finding the pre/post-pended $0's$ on the Insdel blocks to locate the blocks in between.

Intuitively, we may modify `RecoverBlock` to recover an interval $[a, b] \subseteq [B]$ of contiguous Hamming blocks instead of a single block by stopping the search once its search radius is linear to the *size of the interval* versus the size of a single block. Lastly, we run a procedure we call `Sim-RecoverBlock` to recover each block $j \in [a, b]$ within the final search radius $[l, r]$. If we naively call $\texttt{RecoverBlock}^{\tilde{\boldsymbol{Y}}}(l, r, q)$ repeatedly for every block $q \in [a, b]$ then each call to `RecoverBlock` will execute its own noisy binary search procedure in the interval $[l, r]$ resulting in redundant queries which would blow up our amortized locality i.e., the total number of queries would be too large i.e. $\Omega((l - r)\text{polylog}(l - r))$. Instead, we simply make $l - r + 1$ queries to $\tilde{Y}$ to recover the entire substring $\tilde{Y}[l, r]$ and then run $\texttt{Sim-RecoverBlock}(\tilde{\boldsymbol{Y}}[l, r], l, r, q)$ for each $q \in [a, b]$ where $\texttt{Sim-RecoverBlock}(\tilde{\boldsymbol{Y}}[l, r], l, r, q)$ uses the hardcoded string $\tilde{Y}[l, r]$ to simulate the execution of $\texttt{RecoverBlock}^{\tilde{\boldsymbol{Y}}}(l, r, q)$. We call this overall extended procedure `RecoverBlocks`, and present the construction below (modifications to the `RecoverBlock` procedure are in blue).

```
RecoverBlocks^{Ỹ}(l, r, a, b)
Parameters: Block size $\tau \in \mathbb{N}, \gamma \in (0, 1)$
Input: Search range $l < r \in [\tilde{n} + 1]$, block range $a \le b \in [B]$
Output: pre-compiled blocks $\boldsymbol{s} \in \{0, 1\}^{(b-a+1)\tau}$
Let $c = 36(\beta - \gamma)$.
Let $\rho = \min\left\{\frac{1}{4} \times \frac{\beta-\gamma}{\beta+\gamma}, 1 - \frac{3}{4} \times \frac{\beta+\gamma}{\beta-\gamma}\right\}$.

1. While $r - l > c(b - a + 1)\tau$:
   a. Let $m_1 \leftarrow (1 - \rho)l + \rho r$, and $m_2 \leftarrow \rho l + (1 - \rho)r$.
   b. For $p = 1, \ldots, N = \theta(\log^2 n)$,
      i. randomly sample $i_p \xleftarrow{\$} [m_1, m_2)$
      ii. set $S \leftarrow \texttt{Block-Decode}^{\tilde{\boldsymbol{Y}}}(i_p)$
      iii. If $S = \perp$, ignore and continue
      iv. Else, parse $(j_p, \boldsymbol{w}_{j_p}) \leftarrow S$
   c. Let $j_{med} \leftarrow \text{median}(j_1, \ldots, j_N)$ (ignore $\perp$)
   d. If $a > j_{med}$, set $l = m_1$. Otherwise, set $r = m_2$.
2. Make $(l - r + 1)$ queries to recover $\tilde{\boldsymbol{Y}}[l, r]$.
3. Output $\left[\texttt{Sim-RecoverBlock}(\tilde{\boldsymbol{Y}}[l, r], l, r, q) : q \in [a, b]\right]$.
```

The first step is to prove that `RecoverBlocks` on input $(1, \tilde{n} + 1, a, b)$ for any block interval $[a, b] \subseteq [B]$ will return all Hamming blocks $\boldsymbol{w}_j$ within that interval, given that the blocks are good. We will then argue that by a careful setting of parameters, the number of correctly recovered blocks is sufficient for an overall correct decoding process. We refer to the full version [8] for the proof of the following theorem and lemma on the correctness and query complexity of the `RecoverBlocks` procedure.

▶ **Theorem 8.** *Let $\mathcal{P} = \{[a_i, b_i] : 1 \le a_i \le b_i < a_{i+1} \le B\}$ be any partition of $[B]$. Then, let $\boldsymbol{s}_j^{[a_i,b_i]}$ be the random variable defined as the output of $\texttt{RecoverBlocks}^{\tilde{\boldsymbol{Y}}}(1, \tilde{n} + 1, a_i, b_i)\langle j - a_i + 1\rangle_\tau$. If $\delta = \frac{\delta_h \phi \gamma}{8\beta(1+1/\theta)}$, then*

$$\Pr\left[\sum_{[a_i,b_i]\in\mathcal{P}} \sum_{j=1}^{b_i-a_i+1} 1\left(\boldsymbol{s}_j^{[a_i,b_i]} \ne \boldsymbol{w}_j\right) \ge \delta_h B\right] \le negl(n),$$

*where the probability is taken over the joint distribution $\left\{\boldsymbol{s}_j^{[a_i,b_i]} : [a_i, b_i] \in \mathcal{P}, j \in [b_i - a_i + 1]\right\}$.*

▶ **Lemma 9.** *For any word $\tilde{\boldsymbol{Y}} \in \{0, 1\}^{\tilde{n}}$, interval $[a, b] \subseteq [B]$, and constants $\beta, \gamma > 0$, on input $(1, \tilde{n} + 1, a, b)$, `RecoverBlocks` has query complexity $O((b - a)\tau + \tau \log^3 n)$.*

## 2.3 The Amortized Insdel Decoder

We construct an amortized Insdel local decoder Dec utilizing `RecoverBlocks` to simulate oracle access for the underlying Hamming decoder $\mathsf{Dec}_{\mathrm{h}}$. Our Insdel decoder Dec follows a similar procedure to the BBGKZ compiler decoder. Dec calls the underlying Hamming decoder $\mathsf{Dec}_{\mathrm{h}}$, and on its queries $i_1, \ldots, i_q \in [m]$ to the (possibly corrupted) Hamming codeword $\tilde{\boldsymbol{y}}$, will respond by calling `RecoverBlocks` to recover the corresponding Hamming blocks and queried symbols. Correctness of our procedure will follow from Theorem 8, which roughly states that the symbols returned by `RecoverBlocks` are equivalent to what $\mathsf{Dec}_{\mathrm{h}}$ expects from true oracle access to the (possibly corrupted) Hamming codeword $\tilde{\boldsymbol{y}}$.

Our next step is to ensure that the queries made by our Insdel decoder amortize over the `RecoverBlocks` calls by restricting our Hamming decoder to only make contiguous queries, where each contiguous interval is of a fixed size. Suppose the Hamming decoder $\mathsf{Dec}_{\mathrm{h}}$ is guaranteed to make queries in contiguous intervals of size $t$, say $[u_1+1, u_1+t], \ldots, [u_p+1, u_p+t] \subseteq [m]$ for some $t$. Intuitively, when the Insdel decoder Dec is simulating the Hamming decoder's queries with `RecoverBlocks`, it can then make $O(p)$ calls to `RecoverBlocks` to recover codeword intervals of size $O(t)$. For large enough $t$, the number of `RecoverBlocks` calls $O(p)$ decreases and the Insdel decoder's queries will amortize.

We formally describe a Hamming decoder with this desirable property $t$-consecutive interval querying Hamming decoder.

▶ **Definition 10** (Consecutive Interval Querying). *Consider any $(m,k)$-code $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ that is a $(\alpha, \kappa, \delta, \varepsilon)$-*aLDC*. For any word $\tilde{\boldsymbol{y}}$, interval $[L,R] \subseteq [k]$ with $R - L + 1 \geq \kappa$, and random coins $r$, let* $\mathsf{Query}(\tilde{\boldsymbol{y}}, [L,R], r) := \{i_1, \ldots, i_q\} \subseteq [\tilde{n}]$ *denote the codeword indices queries when* $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(L,R)$ *is ran with randomness $r$. Code $\mathcal{C}$ and decoder* Dec *are $t$-*consecutive interval querying *if the set* $\mathsf{Query}(\tilde{\boldsymbol{y}}, [L,R], r)$ *can be partitioned into $q/t$ disjoint intervals $[u_1, v_1], \ldots, [u_{q/t}, v_{q/t}]$ of size $t$ i.e.,*

1. *$v_j - u_j + 1 = t$ for all $j \leq q/t$,*
2. $\mathsf{Query}(y, [L,R], r) = \bigcup_{j \leq q/b} [u_j, v_j]$, *and*
3. *$[u_{j_1}, v_{j_1}] \cap [u_{j_2}, v_{j_2}] = \emptyset$ for all $j_1 \neq j_2$.*

We now show how to construct the amortized local decoder Dec using `RecoverBlocks` and a $t$-consecutive interval querying Hamming aLDC decoder $\mathsf{Dec}_{\mathrm{h}}$ for a value of $t$ to be specified in the analysis. We choose the SZ-code as the Insdel code $\mathcal{C}_{\mathrm{insdel}}$, which has constant rate $1/\beta_{\mathrm{insdel}} = \Omega(1)$ and constant error-tolerance $\delta_{\mathrm{insdel}} = \theta(1)$.

▶ **Construction 11.** Let $\mathcal{C}_{\mathrm{h}} = (\mathsf{Enc}_{\mathrm{h}}, \mathsf{Dec}_{\mathrm{h}})$ be a $t$-consecutive interval querying Hamming LDC such that $\tau$ divides $t$. Let $\mathcal{C}_{\mathrm{insdel}} = (\mathsf{Enc}_{\mathrm{insdel}}, \mathsf{Dec}_{\mathrm{insdel}})$ be an Insdel binary code.

```
EncCompile_$\phi$($\boldsymbol{y}$)
Input: Hamming codeword $\boldsymbol{y} \in \{0,1\}^m$, padding rate $\phi \in (0,1)$.
Output: Insdel codeword $\boldsymbol{Y} \in \{0,1\}^n$.
1. Parse $\boldsymbol{y} = \boldsymbol{w}_1 \circ \cdots \circ \boldsymbol{w}_{|\boldsymbol{y}|/\tau}$ where $\boldsymbol{w}_j \in \{0,1\}^\tau$ for all $j \in [\,|\boldsymbol{y}|/\tau\,]$.
2. Compute $\boldsymbol{w}'_j \leftarrow 0^{\phi\tau} \circ \mathsf{Enc}_{\mathrm{insdel}}(j \circ \boldsymbol{w}_j) \circ 0^{\phi\tau}$ for all $j \in [\,|\boldsymbol{y}|/\tau\,]$.
3. Output $\boldsymbol{Y} = \boldsymbol{w}'_1 \circ \cdots \circ \boldsymbol{w}'_{|\boldsymbol{y}|/\tau}$.
```

```
Enc($\boldsymbol{x}$)
Parameters: Block size $\tau \in \mathbb{N}$, Padding rate $\phi \in (0,1)$.
Input: message $\boldsymbol{x} \in \{0,1\}^k$.
Output: Insdel codeword $\boldsymbol{Y} \in \{0,1\}^n$.
1. Compute $\boldsymbol{y} \leftarrow \mathsf{Enc}_{\mathrm{h}}(\boldsymbol{x})$.
2. Output $\boldsymbol{Y} \leftarrow \mathsf{EncCompile}(\boldsymbol{y})$.
```

```
Dec$^{\tilde{\boldsymbol{Y}}}(L, R)$
Parameters: Block size $\tau \in \mathbb{N}$, Padding rate $\phi \in (0,1)$.
Input: $1 \le L \le R \le \tilde{n}$
Output: word $\tilde{\boldsymbol{s}} \in \{0,1\}^{(R-L+1)\tau}$
1. Suppose $\mathsf{Dec}_{\mathrm{h}}(L, R)$ queries disjoint intervals $[u_1, v_1], \ldots, [u_p, v_p] \subseteq [m]$ of size $t$.
2. For each $r \in [p]$,
   a. Compute $j \in [\ \lfloor n/t \rfloor\ ]$ such that $[u_r, v_r] \subset [(j-1)t+1, (j+1)t]$.
   b. Let $t' = \lceil t/\tau \rceil$ and compute
        $\boldsymbol{s}^{(j-1)} \leftarrow \mathtt{RecoverBlocks}^{\tilde{\boldsymbol{Y}}}(1, \tilde{n}+1, (j-1)t'+1, jt')$
        $\boldsymbol{s}^{(j)} \leftarrow \mathtt{RecoverBlocks}^{\tilde{\boldsymbol{Y}}}(1, \tilde{n}+1, jt'+1, (j+1)t')$.
3. From $\{\boldsymbol{s}^{(j)}\}$, send $\mathsf{Dec}_{\mathrm{h}}$ the bits corresponding to intervals $[u_1, v_1], \ldots, [u_p, v_p]$.
4. Output the output of $\mathsf{Dec}_{\mathrm{h}}$.
```

Our Insdel decoder $\mathsf{Dec}$ processes the $t$-sized intervals $[u_1, v_1], \ldots, [u_p, v_p] \subseteq [m]$ queried by the Hamming decoder $\mathsf{Dec}_{\mathrm{h}}$ into a corresponding $t' = \lceil t/\tau \rceil$-sized block interval inputs for `RecoverBlocks` (step 2.a). Note that for ease of presentation, the decoder always computes `RecoverBlocks` on two $t'$ sized block intervals (step 2.b). The amortized locality can be optimized by a constant factor by calling `RecoverBlocks` on the exact block intervals queried by the Hamming decoder $\mathsf{Dec}_{\mathrm{h}}$. However, it will be convenient to assume the Insdel decoder $\mathsf{Dec}$ calls `RecoverBlocks` in a predictable manner for all inputs when proving correctness i.e. the input interval to `RecoverBlocks` is always $((j-1)t'+1, jt')$ for some $j$, and asymptotically, there is no change to the amortized locality.

▶ **Theorem 12.** *Let $(m, k)$-code $\mathcal{C}_h = (\mathsf{Enc}_h, \mathsf{Dec}_h)$ be a $t$-consecutive interval querying Hamming $(\alpha_h, \kappa_h, \delta_h, \varepsilon_h)$-aLDC. Then, for any block size $\tau \in \Omega(\log n)$ such that $\tau \mid n$, $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ in Construction 11 is a $(\alpha, \kappa, \delta, \varepsilon)$-aLDC for $\alpha = O\left(\frac{\alpha_h \tau \log^3 n}{t}\right)$, $\kappa = \kappa_h$, $\delta = \Omega(\delta_h)$, and $\varepsilon = \varepsilon_h + negl(n)$.*

**Proof.** Suppose $\mathsf{Dec}_{\mathrm{h}}^{\tilde{\boldsymbol{y}}}(L, R)$ queries disjoint intervals $[u_1, v_1], \ldots, [u_p, v_p] \subset [m]$. For each $r \in [p]$, we compute `RecoverBlocks` on block intervals $[(j-1)t'+1, jt']$ and $[jt'+1, (j+1)t']$ such that $[u_r, v_r] \subset [(j-1)t+1, (j+1)t]$. Let $\mathcal{P} = \{[1, t'], [t'+1, 2t'], \ldots, [B-t'+1, B]\}$ be a partition of $[B]$ in equal $t'$-sized intervals. Let $\boldsymbol{s}^{(j)}$ be the random variable defined as the output of $\mathtt{RecoverBlocks}^{\tilde{\boldsymbol{Y}}}(1, \tilde{n}+1, (j-1) \times t'+1, j \times t')$. Define $\tilde{\boldsymbol{y}}$ as the random string defined by $\tilde{\boldsymbol{y}}\langle j \rangle_{t'} = \boldsymbol{s}^{(j)}$ for each $j \in [\lfloor n/t' \rfloor]$. Then, since $\mathsf{Ham}(\tilde{\boldsymbol{y}}_u, \boldsymbol{y}) \le \delta_{\mathrm{h}} m$ is implied by the event $\sum_{[(j-1)t'+1,jt']\in\mathcal{P}}\sum_{r=1}^{t} 1\left(\boldsymbol{s}_r^{(j)} \neq \boldsymbol{w}_j\right) \leq \delta_{\mathrm{h}}B$, by Theorem 8, $\Pr[\mathsf{Ham}(\tilde{\boldsymbol{y}},\boldsymbol{y}) \leq \delta_{\mathrm{h}}m] \geq 1-\mathrm{negl}(n)$. Thus, from the view of $\mathsf{Dec}_{\mathrm{h}}$, it is interacting with $\tilde{\boldsymbol{y}}$ over partition $\mathcal{P}$, and so by definition, for any $R-L+1 \geq \kappa_{\mathrm{h}}$,

$$\Pr\left[\mathsf{Dec}_{\mathrm{h}}^{\tilde{\boldsymbol{y}}}(L,R) = (x_i : i\in[L,R]) \;\middle|\; \mathsf{Ham}(\tilde{\boldsymbol{y}},\boldsymbol{y}) \leq \delta_{\mathrm{h}}m\right] \geq 1-\varepsilon_u.$$

By construction of $\mathsf{Dec}$, for any $R-L+1 \geq \kappa_{\mathrm{h}}$,

$$\begin{aligned}&\Pr\left[\mathsf{Dec}^{\tilde{\boldsymbol{Y}}}(L,R) = (x_i : i\in[L,R])\right]\\ &\quad\geq \Pr[\mathsf{Ham}(\tilde{\boldsymbol{y}}_u,\boldsymbol{y}) \leq \delta_{\mathrm{h}}m]\Pr\left[\mathsf{Dec}_{\mathrm{h}}^{\tilde{\boldsymbol{y}}_u}(L,R) = (x_i : i\in[L,R]) \;\middle|\; \mathsf{Ham}(\tilde{\boldsymbol{y}}_u,\boldsymbol{y}) \leq \delta_{\mathrm{h}}m\right]\\ &\quad\geq (1-\mathrm{negl}(n))\times(1-\varepsilon_{\mathrm{h}}) \geq 1-\varepsilon_{\mathrm{h}}-\mathrm{negl}(n).\end{aligned}$$

Since $\mathcal{C}_{\mathrm{h}}$ is $t$-consecutive interval querying which makes at most $\alpha_{\mathrm{h}}(R-L+1)$ queries, the decoder $\mathsf{Dec}$ calls `RecoverBlocks` at most $\frac{2\alpha_{\mathrm{h}}(R-L+1)}{t}$ times on block intervals of size $t' = \lceil t/\tau\rceil$. By Lemma 9, the query complexity of $\mathsf{Dec}$ is

$$\alpha(R-L+1) = \frac{2\alpha_{\mathrm{h}}(R-L+1)}{t}\times O\left((\lceil t/\tau\rceil\times\tau)+\tau\log^3 n\right) \text{ and so } \alpha = O\left(\frac{\alpha_{\mathrm{h}}\tau\log^3 n}{t}\right). ◀$$

If a $t$-consecutive interval querying, ideal Hamming $\mathsf{aLDC}$ does exists, we obtain the following corollary.

▶ **Corollary 13.** *If there exists a $t$-consecutive interval querying, ideal Hamming* $\mathsf{aLDC}$ *for $t = \Omega(\tau\log^3 n)$, then there exists an ideal Insdel* $\mathsf{aLDC}$.

We construct $t$-consecutive interval querying, ideal Hamming $\mathsf{aLDC}$s in private-key and resource-bounded settings and leave a construction for worst-case Hamming errors as an open question.

## 3 Ideal Insdel aLDCs in Private-key Settings

Given the compiler and resulting Theorem 12 in the previous section converting an ideal Hamming $\mathsf{aLDC}$ $\mathcal{C}_{\mathrm{h}}$ to ideal Insdel $\mathsf{aLDC}$ $\mathcal{C}$ whenever $\mathcal{C}_{\mathrm{h}}$ is $\Omega(\tau\log^3 n)$-consecutive interval querying, we show that such an $\mathsf{aLDC}$ exist with private-key assumptions. We start by recalling the definition of a private-key $\mathsf{aLDC}$ ($\mathsf{paLDC}$).

▶ **Definition 14** ($\mathsf{paLDC}$ [7]). *Let $\lambda$ be the security parameter. A triple of probabilistic polynomial time algorithms* $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ *over $\Sigma$ is a private $(\alpha,\kappa,\delta,\epsilon,q)$-amortizeable LDC* ($\mathsf{paLDC}$) *for Hamming errors (resp. Insdel errors) if*

- *for all keys* $\mathsf{sk} \in$ *Range*$(\mathsf{Gen}(1^\lambda))$ *the pair* $(\mathsf{Enc}_{\mathsf{sk}},\mathsf{Dec}_{\mathsf{sk}})$ *is a $(n,k)$ code $\mathcal{C}$ over $\Sigma$, and*
- *for all $\tilde{\boldsymbol{y}} \in \Sigma^n$ and all $L,R\in[k]$ with $R-L+1 \geq \kappa$ the local decoding algorithm* $\mathsf{Dec}_{\mathsf{sk}}^{\tilde{\boldsymbol{y}}^{(h)}}(L,R)$ *makes at most $(R-L+1)\alpha$ queries to $\tilde{\boldsymbol{y}}$*
- *for all probabilistic polynomial time algorithms $\mathcal{A}$ there is a negligible function $\mu$ such that* $\Pr[$*`paLDC-Sec-Game`*$(\mathcal{A},\lambda,\alpha,\kappa,\delta,\alpha,q) = 1] \leq \mu(\lambda)$*, where the probability is taken over the randomness of $\mathcal{A}$,* $\mathsf{Gen}$*, and* *`paLDC-Sec-Game`*. *The experiment* *`paLDC-Sec-Game`* *is defined as follows:*

***paLDC-Sec-Game***$(\mathcal{A}, \lambda, \alpha, \kappa, \delta, q)$

*The challenger generates secret key* $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$. *For* $q$ *rounds, on iteration* $h$, *the challenger and adversary* $\mathcal{A}$ *interact as follows:*

1. *The adversary* $\mathcal{A}$ *chooses a message* $\boldsymbol{x}^{(h)} \in \Sigma^k$ *and sends it to the challenger.*
2. *The challenger sends* $\boldsymbol{y}^{(h)} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\boldsymbol{x}^{(h)})$ *to the adversary.*
3. *The adversary outputs* $\tilde{\boldsymbol{y}}^{(h)} \in \Sigma^*$ *that is* $\delta$*-Hamming-close (resp.* $\delta$*-edit-close) distance to* $\boldsymbol{y}^{(h)}$.
4. *If there exists* $L^{(h)}, R^{(h)} \in [k]$ *such that* $R^{(h)} - L^{(h)} + 1 \geq \kappa$ *and* $\Pr\left[\mathsf{Dec}_{\mathsf{sk}}^{\tilde{\boldsymbol{y}}^{(h)}}(L^{(h)}, R^{(h)}) \neq \boldsymbol{x}[L^{(h)}, R^{(h)}]\right] > \varepsilon(\lambda)$, *then this experiment immediately terminates and outputs* $1$.

*If the experiment did not output* $1$ *on any iteration* $h$, *then output* $0$.

Note that for paLDCs, we assume that the local decoder takes in a consecutive interval $[L, R]$ rather than an arbitrary subset $Q$ as input. Blocki and Zhang show that explicit, ideal Hamming paLDC constructions exist for such decoders, and the existence of ideal paLDCs whose local decoders take in arbitrary subsets $Q$ is left as an open problem.

## 3.1 A consecutive interval querying, Ideal Hamming paLDC

We construct a $t$-consecutive interval querying, ideal Hamming paLDC by modifying of the private-key LDC construction by Ostrovsky et al. [21]. Recall that the encoding procedure on input message $\boldsymbol{x}$ first partitions it into equal-sized blocks $\boldsymbol{x} = \boldsymbol{e}_1 \circ \dots \boldsymbol{e}_B$, for all $j \in [B]$. Each block is then individually encoded as $\boldsymbol{e}'_j = \mathsf{Enc}(\boldsymbol{e}_j)$ for each $j \in [B]$ by a constant rate, constant error tolerant, and constant alphabet size Hamming code encoding Enc (e.g. the Justesen code) to form the encoded message $\boldsymbol{x}' = \boldsymbol{e}'_1 \circ \dots \circ \boldsymbol{e}'_B$. Lastly, an additional secret-key permutation $\pi$ and random mask $\boldsymbol{r}$ are applied i.e. the codeword is $\boldsymbol{y} = \pi(\boldsymbol{x}') \oplus \boldsymbol{r}$, which effectively makes the codeword look random from the view of a probabilistic polynomial time channel.

Blocki and Zhang [7] observe that the encoding procedure by Ostrovsky et al. is amenable to amortization when the recovered symbols are in a consecutive interval $[L, R]$. Their amortized local decoder, with access to the secret key, undoes the permutation $\pi$ and random mask $\boldsymbol{r}$, recovers the contiguous blocks $\boldsymbol{e}'_{s+1}, \dots, \boldsymbol{e}'_{s+\ell}$ corresponding to requested $[L, R]$, decodes each block, and returns the corresponding queried symbols. Unfortunately, since we applied a permutation $\pi$ in the encoding procedure, the amortized local decoder does not make consecutive interval queries to the (corrupted) codeword $\tilde{\boldsymbol{y}}$.

To add $t$-consecutive interval querying to the current decoder, we modify the encoding scheme permutation step to permute $t$-sized *sub-blocks* of the blocks $\boldsymbol{e}'_j$ in encoded message $\boldsymbol{x}' = \boldsymbol{e}'_1 \circ \dots \circ \boldsymbol{e}'_B$, rather than permuting individual bits. We highlight in blue significant changes made from the original paLDC presented by Blocki and Zhang.

▶ **Construction 15.** Suppose that $\mathcal{C} = (\mathsf{Enc}_{\mathcal{C}}, \mathsf{Dec}_{\mathcal{C}})$ is a Hamming $(A, a)$-code over an alphabet $\Sigma$ with rate $R$. Let $c = \log|\Sigma|$. Define $\mathcal{C}_{\mathsf{p}} = (\mathsf{Gen}_{\mathsf{p}}, \mathsf{Enc}_{\mathsf{p}}, \mathsf{Dec}_{\mathsf{p}})$ with message length $k$, codeword length $m = \frac{k}{R}$, block size $cA$, and sub-block size $t$ (dividing $cA$) as follows:

$\mathsf{Gen}(1^\lambda)$

1. Generate $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^m$ and uniformly random permutation $\pi : [m/t] \to [m/t]$.
2. Output $\mathsf{sk} \leftarrow (\boldsymbol{r}, \pi)$.

```
$\mathsf{Enc}_{\mathsf{sk}}(\boldsymbol{x})$
Input: Message $\boldsymbol{x} \in \{0,1\}^k$
Output: Codeword $\boldsymbol{y} \in \{0,1\}^m$
1. Parse $(\boldsymbol{r}, \pi) \leftarrow \mathsf{sk}$.
2. Let $\boldsymbol{x} \leftarrow \boldsymbol{e}_1 \circ \boldsymbol{e}_2 \circ \cdots \circ \boldsymbol{e}_B$ where each $\boldsymbol{e}_s \in \Sigma^a$ (and $B = k/ca$).
3. For each $s \in [B]$:
   a. set $\boldsymbol{e}'_s \leftarrow \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{e}_s)$, and
   b. let $\boldsymbol{e}'_s = \boldsymbol{\epsilon}_{(s-1)\beta+1} \circ \cdots \circ \boldsymbol{\epsilon}_{s\beta}$ where each $\boldsymbol{\epsilon}_{s'} \in \{0,1\}^t$ (and $\beta = cA/t$).
4. Output $\boldsymbol{y} = \left(\boldsymbol{\epsilon}_{\pi(1)} \circ \cdots \circ \boldsymbol{\epsilon}_{\pi(B\beta)}\right) \oplus \boldsymbol{r}$.
```

```
$\mathsf{Dec}^{\tilde{\boldsymbol{y}}}_{\mathsf{sk}}(L, R)$
Input: Interval $[L, R] \subseteq [k]$
Output: word $\tilde{\boldsymbol{x}}[L, R] \in \{0,1\}^{R-L+1}$.
1. Parse $(\boldsymbol{r}, \pi) \leftarrow \mathsf{sk}$ and interpret $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{\epsilon}}_{\pi(1)} \circ \cdots \circ \tilde{\boldsymbol{\epsilon}}_{\pi(B\beta)}$ where each $\tilde{\boldsymbol{\epsilon}}_j \in \{0,1\}^t$.
2. Let the bits of $\boldsymbol{x}[L, R]$ lie in blocks $\boldsymbol{e}_{s+1}, \ldots, \boldsymbol{e}_{s+\ell}$. For each $j = s+1, \ldots, s+\ell$ :
   a. Compute $s'_1, \ldots, s'_\beta$ such that $\pi(s'_r) = (s+j-1)\beta + r$ for all $r \in [\beta]$.
   b. Query $\tilde{\boldsymbol{y}}$ to compute $\tilde{\boldsymbol{e}}'_j \leftarrow \tilde{\boldsymbol{\epsilon}}_{\pi(s'_1)} \circ \cdots \circ \tilde{\boldsymbol{\epsilon}}_{\pi(s'_\beta)}$.
   c. Compute $\tilde{\boldsymbol{e}}_j \leftarrow \mathsf{Dec}_{\mathcal{C}}(\tilde{\boldsymbol{e}}'_j)$.
From $\tilde{\boldsymbol{e}}_{s+1}, \ldots, \tilde{\boldsymbol{e}}_{s+\ell}$ output bits corresponding to interval $[L, R]$.
```

Any queries made by the decoder Dec are to the $t$-sized (corrupted) sub-blocks $\tilde{\boldsymbol{\epsilon}}_{\pi(r)}$, so Construction 15 is $t$-consecutive interval querying. The main challenge is choosing the overall block size $cA$ such that decoding error remains negligible. We show that by increasing the original setting of block size $\tau$ in the work of Blocki and Zhang [7] by a multiplicative $t$-factor, negligible decoding error follows.

▶ **Theorem 16.** *Let $\lambda \in \mathbb{N}$. In Construction 15, suppose the $(A, a)$-code $\mathcal{C}$ has (constant) rate $R$, (constant) error tolerance $\delta_{\mathsf{p}}$, and (constant) alphabet $\Sigma$ with $c = \log|\Sigma|$. Then, for any $t \in \mathbb{N}$ such that $t \mid cA$ and $k \in poly(\lambda)$, $\mathcal{C}_{\mathsf{p}} = (\mathsf{Gen}_{\mathsf{p}}, \mathsf{Enc}_{\mathsf{p}}, \mathsf{Dec}_{\mathsf{p}})$ is a $t$-consecutive interval querying $(2/R, O(a), \theta(\delta_{\mathsf{p}}), negl(\lambda), 1)$-paLDC when $a = \omega(t \log \lambda)$.*

**Proof.** We show that the probability of an incorrect decoding $\Pr[\texttt{paLDC-Sec-Game}(\mathcal{A}, \lambda, \alpha, \kappa, \delta, 1) = 1]$ is negligible for $\delta = \theta(\delta_{\mathsf{p}})$ and any probabilistic polynomial time adversary $\mathcal{A}$. Define the event $\texttt{Bad} = \bigcup_{j \in [k/ca]} \texttt{Bad}_j$, where $\texttt{Bad}_j$ is the event that $\boldsymbol{e}'_j$ has more than a $\delta_{\mathsf{p}}$ fraction of errors. We show that the probability $\Pr[\texttt{Bad}]$ is negligible.

Since the $t$-sized sub-blocks of the codeword are permuted and a random mask is applied, any errors added by a probabilistic polynomial time adversary $\mathcal{A}$ to the corrupted codeword $\tilde{\boldsymbol{y}}$ are added uniformly at random over $t$-bit intervals. This follows from generalizing the argument given by Lipton [19], where we interpret each sub-block as a $t$-bit symbol and the observation that the probability that event $\texttt{Bad}$ occurs given that $\mathcal{A}$ *does not* apply errors in a $t$-bit interval is at most the probability that event $\texttt{Bad}$ occurs given that $\mathcal{A}$ *does* apply errors in a $t$-bit interval.

Then, the number of errors in any given block $\boldsymbol{e}'_j$ follow a Hypergeometric$(m/t, \delta m/t, cA/t)$, which by the CDF bound of [15, 16], we have $\Pr[\texttt{BAD}_j] < \exp\left(\frac{-2(((\delta_{\mathsf{p}}-\delta)(cA/t))^2-1)}{(cA/t)+1}\right)$. Thus, for $\delta_{\mathsf{p}} > \delta$ and $a/t \in \omega(\log n)$, this probability is negligible. By a union bound, $\Pr[\texttt{Bad}]$ is also negligible, so $\varepsilon \le \Pr[\texttt{Bad}] < \mathrm{negl}(\lambda)$.

The proof of the query parameters $\alpha = 2/R$ and $\kappa = a$, is the same as in Theorem 16 in the work of Zhang and Blocki [7] (see full version [8]). ◀

We note that the poly-round paLDC construction of Blocki and Zhang [4] can also be made $t$-consecutive interval querying by the same technique of applying a higher-order permutation. We omit the construction and proof from this work since it will follow an almost-identical modification.

## 3.2 The Ideal Insdel paLDC Construction

We compile the $t$-consecutive interval querying, ideal Hamming paLDC in the prior subsection into an ideal Insdel paLDC using the Hamming-to-Insdel compiler EncCompile in Section 2.

▶ **Construction 17.** Suppose $(m,k)$-code $\mathcal{C}_\mathsf{p} = (\mathsf{Gen}_\mathsf{p}, \mathsf{Enc}_\mathsf{p}, \mathsf{Dec}_\mathsf{p})$ is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{p}, \kappa_\mathsf{p}, \delta_\mathsf{p}, \varepsilon_\mathsf{p}, q)$-paLDC. Then define $\mathcal{C} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as $\mathsf{Gen}(1^\lambda) := \mathsf{Gen}_\mathsf{p}(1^\lambda)$, $\mathsf{Enc}_\mathsf{sk}(\boldsymbol{x}) := \mathsf{EncCompile}(\mathsf{Enc}_{\mathsf{p},\mathsf{sk}}(\boldsymbol{x}))$, and $\mathsf{Dec}_\mathsf{sk}^{\tilde{\boldsymbol{Y}}}(L,R) := \mathsf{Dec}_{\mathsf{p},\mathsf{sk}}^{\tilde{\boldsymbol{Y}}}(L,R)$.

We will need to show that the Hamming-to-Insdel compiler EncCompile, when used within the private-key setting, retains correctness. See the full version [8] for the reduction argument proving the following theorem.

▶ **Theorem 18.** *Let $\mathcal{C}_\mathsf{p} = (\mathsf{Gen}_\mathsf{p}, \mathsf{Enc}_\mathsf{p}, \mathsf{Dec}_\mathsf{p})$ be a $(m,k)$-code that is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{p}, \kappa_\mathsf{p}, \delta_\mathsf{p}, \varepsilon_\mathsf{p})$-paLDC. Then, $\mathcal{C} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in Construction 17 with is a $(n,k)$-code that is an Insdel $(\alpha, \kappa, \delta, \varepsilon)$-paLDCwith $\alpha = O\left(\frac{\alpha_\mathsf{p}\tau\log^3 n}{t}\right)$, $\kappa = \kappa_\mathsf{p}, \delta = \Omega(\delta_\mathsf{p})$, and $\varepsilon = \varepsilon_\mathsf{p} + negl(n)$.*

By instantiating our private-key Insdel compiler with our $t$-querying Hamming paLDC, we construct an ideal Insdel paLDC.

▶ **Corollary 19.** *If $\mathcal{C}_\mathsf{p}$ of Construction 15 instantiated with a constant rate, constant error tolerant, and constant size alphabet code (e.g. a Justesen code), $t = \theta(\tau\log^3 n)$ and $\tau = \theta(\log n)$, then code $\mathcal{C}$ in Construction 17 is an (Ideal) Insdel $(O(1), O(\log^5 n), O(1), negl(n))$-paLDC with constant rate.*

## 4 Ideal Insdel aLDCs for Resource-bounded Channels

In this section, we present an ideal insdel aLDC in settings where the channel is bounded for some resource, such as parallel time or circuit depth. As in the construction of an ideal Insdel paLDC, our construction for resource-bounded channels will rely on a construction of a consecutive interval querying Hamming aLDC that will allow the compiler to amortize over its queries. Then, we formally prove the Compiler we constructed in Section 2 is secure in resource-bounded settings, which results in our ideal Insdel construction for resource-bounded channels.

We start by recalling the definition of an aLDC for resource-bounded channels (raLDC).

▶ **Definition 20** (raLDC [1]). *A $(n,k)$ code $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ is a $(\alpha, \kappa, \delta, \epsilon, \mathbb{R})$-resource-bounded amortizeable LDC (raLDC) for hamming errors (resp. insDel errors) if for all $L, R \in [k]$ with $R - L + 1 \geq \kappa$ the local decoding algorithm $\mathsf{Dec}_\mathsf{sk}^{\tilde{\boldsymbol{y}}^{(h)}}(.)$ makes at most $(R-L+1)\alpha$ queries to $\tilde{\boldsymbol{y}}$ $\mathsf{Dec}^{\tilde{\boldsymbol{y}}}$ and for any $\tilde{\boldsymbol{y}} \in \Sigma^*$ and for any resource bounded algorithm $\mathcal{A} \in \mathbb{R}$, there is a negligible function $\mu$ such that $\Pr[\texttt{raLDC-Game}(\mathcal{A}, \lambda, \delta, \kappa) = 1] \leq \mu(\lambda)$, where the probability is taken over the randomness of $\mathcal{A}$, $\mathsf{Gen}$, and $\texttt{paLDC-Sec-Game}$. The experiment $\texttt{raLDC-Game}$ is defined as follows:*

**raLDC-Game**$(\mathcal{A}, \lambda, \delta, \kappa)$

1. *The adversary* $\mathcal{A}$ *chooses a message* $\boldsymbol{x} \in \{0,1\}^k$ *and sends it to the encoder.*
2. *The encoder sends* $\boldsymbol{y} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\boldsymbol{x})$ *to the adversary.*
3. *The adversary outputs* $\tilde{\boldsymbol{y}} \in \{0,1\}^*$ *that is* $\delta$*-hamming-close (resp.* $\delta$*-edit-close) distance to* $\boldsymbol{y}$*.*
4. *If there exists* $L, R \in [k]$ *such that* $\Pr\left[\mathsf{Dec}^{\tilde{\boldsymbol{y}}}(L,R) \neq \boldsymbol{x}[L,R]\right] > \varepsilon(\lambda)$ *and* $R - L + 1 \geq \kappa$*, then this experiment outputs* $1$*. Otherwise, output* $0$*.*

## 4.1 A consecutive interval querying, Ideal raLDC

To construct a $t$-consecutive interval querying, ideal raLDC, we employ the Hamming paLDC to raLDC compiler of Blocki and Zhang [7], which was a modification of the original compiler by Ameri et al. [1]. The construction makes use of two other building blocks: the first is cryptographic puzzles, consisting of two algorithms PuzzGen and PuzzSolve. On input seed $\boldsymbol{c}$, PuzzGen outputs a puzzle $\boldsymbol{Z}$ with solution is $\boldsymbol{c}$ i.e. $\mathsf{PuzzSolve}(\boldsymbol{Z}) = \boldsymbol{c}$. The security requirement states that adversary $\mathcal{A}$ in a defined algorithm class $\mathbb{R}$, cannot solve the puzzle $\boldsymbol{Z}$ and cannot even distinguish between tuples $(\boldsymbol{Z}_0, \boldsymbol{c}_0, \boldsymbol{c}_1)$ and $(\boldsymbol{Z}_1, \boldsymbol{c}_0, \boldsymbol{c}_1)$ for random $\boldsymbol{c}_i$ and $\boldsymbol{Z}_i = \mathsf{PuzzGen}(\boldsymbol{c}_i)$. Ameri et al. are able to construct *memory-hard* puzzles, i.e. cryptographic puzzles unsolvable by algorithm class $\mathbb{R}_{\mathsf{cmc}}$ of algorithms with bounded cumulative memory complexity, under standard cryptographic assumptions. We generalize their definition for any class of algorithms $\mathbb{R}$, where if a cryptographic puzzle is unsolvable by any algorithm in $\mathbb{R}$, we say the puzzle is $\mathbb{R}$-hard.

The second building block is a variant of a locally decodable code, referred to as LDC$^*$, which recovers the entire (short) encoded message $\boldsymbol{s}$ with locality only scaling linearly with the message length. The original LDC$^*$ construction given by Blocki et al. [6] is a repetition code of a constant rate, constant error tolerance code (e.g. the Justesen code), where Dec$^*$ queries a constant number of repeated codewords and performs a majority vote decoding. On message length $k$, their LDC$^*$decoder makes $\theta(k)$ queries, where the codeword length $n \gg k$ can be arbitrarily large.

Given a paLDC code $\mathcal{C}_{\mathsf{p}} = (\mathsf{Gen}_{\mathsf{p}}, \mathsf{Enc}_{\mathsf{p}}, \mathsf{Dec}_{\mathsf{p}})$ and a LDC$^*$ code $\mathcal{C}_* = (\mathsf{Enc}_*, \mathsf{Dec}_*)$, the constructed Hamming raLDC encoder Enc, on message $\boldsymbol{x}$ will (1) generate the secret key $\mathsf{sk} \leftarrow \mathsf{Gen}_{\mathsf{p}}(1^\lambda; \boldsymbol{c})$ with some random coins $\boldsymbol{c}$ (2) Compute the paLDC encoding of the message $\boldsymbol{y}_{\mathsf{p}} \leftarrow \mathsf{Enc}_{\mathsf{p},\mathsf{sk}}(\boldsymbol{x})$ (3) Compute the LDC$^*$ encoding of the puzzle $\boldsymbol{y}_* \leftarrow \mathsf{Enc}_*(\boldsymbol{Z})$, where the LDC$^*$ encoding has codeword length $\theta(|\boldsymbol{y}_{\mathsf{p}}|)$ (4) and finally, output $\boldsymbol{y} = \boldsymbol{y}_{\mathsf{p}} \circ \boldsymbol{y}_*$. Intuitively, a resource-bounded channel will not be able to solve the puzzle to retrieve the secret key from $\boldsymbol{y}_*$. Hence, the message encoding $\boldsymbol{y}_{\mathsf{p}}$ looks random to the channel, reducing the resource-bounded channel's view of the codeword to the view a computationally-bound channel in the private-key setting. On the other hand, the raLDC decoder Dec will have sufficient resources to solve the puzzle and locally decode the codeword on input interval $[L,R]$ i.e. Dec computes $\tilde{\boldsymbol{c}} \leftarrow \mathsf{PuzzSolve}(\mathsf{Dec}_*(\tilde{\boldsymbol{y}}_*))$, $\tilde{\mathsf{sk}} \leftarrow \mathsf{Gen}_{\mathsf{p}}(1^\lambda; \tilde{\boldsymbol{c}})$, and outputs $\mathsf{Dec}^{\tilde{\boldsymbol{y}}_{\mathsf{p}}}_{\mathsf{p}\tilde{\mathsf{sk}}}(L,R)$. By choosing an ideal paLDC code $\mathcal{C}_{\mathsf{p}}$ and an appropriate LDC$^*$ code $\mathcal{C}_*$, the resulting compiled code $(\mathsf{Enc}, \mathsf{Dec})$ is an ideal raLDC.

We observe that the compiled raLDC decoder Dec satisfies $t$-consecutive interval query when instantiated with a $t$-consecutive interval querying paLDC and the original LDC$^*$ construction by Blocki et al. [6]. First, observe that the decoder Dec queries the message encoding $\boldsymbol{y}_{\mathsf{p}}$ and the secret-key randomness encoding $\boldsymbol{y}_*$ disjointly using $\mathsf{Dec}_{\mathsf{p}}$ and $\mathsf{Dec}_*$ respectively. Second, $\mathsf{Dec}_*$ is also $t$-consecutive interval querying as long as the encoding used in the repetition encoding has codeword length $t$. Thus, the overall raLDC decoder Decis $t$-consecutive interval querying. We summarize this result formally below.

▶ **Theorem 21.** *Let $\lambda \in \mathbb{N}$. Let $\mathcal{C}_\mathsf{p} = (\mathsf{Gen}_\mathsf{p}, \mathsf{Enc}_\mathsf{p}, \mathsf{Dec}_\mathsf{p})$ be a $(n_\mathsf{p}, k_\mathsf{p})$-code that is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{p}, \kappa_\mathsf{p}, \delta_\mathsf{p}, \varepsilon_\mathsf{p})$-paLDC. For any algorithm class $\mathbb{R}$ such that there exists $\mathbb{R}$-hard cryptographic puzzles, there exists a $(n_\mathsf{r}, k_\mathsf{r})$-code $\mathcal{C}_\mathsf{r} = (\mathsf{Enc}_\mathsf{r}, \mathsf{Dec}_\mathsf{r})$ that is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{r}, \kappa_\mathsf{r}, \delta_\mathsf{r}, \varepsilon_\mathsf{r}, \mathbb{R})$-raLDC for $n_\mathsf{r} = \theta(n_\mathsf{p}), k_\mathsf{r} = k_\mathsf{p} = poly(\lambda), \alpha_\mathsf{r} = \theta(\alpha_\mathsf{p}), \kappa_\mathsf{r} = \kappa_\mathsf{p}, \delta_\mathsf{r} = \theta(\delta_\mathsf{p})$, and $\varepsilon_\mathsf{r} = \varepsilon_\mathsf{p} + negl(n)$.*

## 4.2 The Ideal Insdel raLDC Construction

We present our final construction of an ideal Insdel raLDC. We will proceed similarly to our construction of an ideal Insdel paLDC in Section 3, where we use our Hamming-to-Insdel compiler EncCompile with the ideal, $t$-consecutive interval querying Hamming raLDC constructed in the prior subsection.

▶ **Construction 22.** Suppose $(m, k)$-code $\mathcal{C}_\mathsf{r} = (\mathsf{Enc}_\mathsf{r}, \mathsf{Dec}_\mathsf{r})$ is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{r}, \kappa_\mathsf{r}, \delta_\mathsf{r}, \varepsilon_\mathsf{r}, \mathbb{R})$-raLDC. Then define $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ as $\mathsf{Enc}(\boldsymbol{x}) := \mathsf{EncCompile}(\mathsf{Enc}_\mathsf{r}(\boldsymbol{x}))$ and $\mathsf{Dec}^{\tilde{\boldsymbol{Y}}}(L, R) := \mathsf{Dec}_\mathsf{r}^{\tilde{\boldsymbol{Y}}}(L, R)$.

Just as before, we will need to show that security holds when we use the Hamming-to-Insdel compiler in a resource-bounded setting by giving a reduction argument. One additional nuance we will need to handle is to allow any adversary for the Hamming code sufficient resources to perform the reduction. Let $T(n)$ (resp. $S(n)$) be the running time (resp. space usage) of the computation of $\mathsf{EncCompile}(\boldsymbol{y})$ and $\tilde{\boldsymbol{y}}_\mathsf{sim}$ . Let $\mathcal{B} = \mathtt{reduce}_n(\mathcal{A})$ be a reduction from algorithm $\mathcal{A}$ to $\mathcal{B}$ in in time $T(n)$ time with space usage $S(n)$. Then, for any class of algorithms $\mathbb{R}(n)$, denote its closure class $\overline{\mathbb{R}}(n)$ with respect to $\mathtt{reduce}_n$ defined as the minimum class of algorithms such that $\mathtt{reduce}_n(\mathcal{A}) \in \overline{\mathbb{R}}(n)$ for all $\mathcal{A} \in \mathbb{R}(n)$. See the full version [8] for the detailed proof.

▶ **Theorem 23.** *Let $\mathcal{C}_\mathsf{r} = (\mathsf{Enc}_\mathsf{r}, \mathsf{Dec}_\mathsf{r})$ be a $(m, k)$-code that is a $t$-consecutive interval querying Hamming $(\alpha_\mathsf{r}, \kappa_\mathsf{r}, \delta_\mathsf{r}, \varepsilon_\mathsf{r}, \mathbb{R}(n))$-raLDC. Then, $\mathcal{C} = (\mathsf{Enc}, \mathsf{Dec})$ in Construction 22 is a $(n, k)$-code that is an Insdel $(\alpha, \kappa, \delta, \varepsilon, \overline{\mathbb{R}}(n))$-raLDC with $\alpha = O\left(\frac{\alpha_\mathsf{r} \tau \log^3 n}{t}\right), \kappa = \kappa_\mathsf{r}, \delta = \Omega(\delta_\mathsf{r})$, and $\varepsilon = \varepsilon_\mathsf{r} + negl(n)$.*

▶ **Corollary 24.** *Let $t = \theta(\tau \log^3 n)$, $\tau = \theta(\log n)$, and define any algorithm class $\mathbb{R}$ such that there exists $\mathbb{R}$-hard cryptographic puzzles. Then by Theorem 21, code $\mathcal{C}$ in Construction 22 is an Insdel $(O(1), O(\log^5 n), O(1), negl(n), \mathbb{R})$-raLDC with constant rate.*

## References

1 Mohammad Hassan Ameri, Alexander R. Block, and Jeremiah Blocki. Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 45–68, Amalfi, Italy, september 12–14 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-14791-3_3.

2 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In László Babai, editor, *36th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, june 13–16 2004. ACM Press. doi:10.1145/1007352.1007361.

3 Alexander R. Block and Jeremiah Blocki. Private and resource-bounded locally decodable codes for insertions and deletions. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1841–1846, 2021. doi:10.1109/ISIT45174.2021.9518249.

4 Alexander R Block, Jeremiah Blocki, Elena Grigorescu, Shubhang Kulkarni, and Minshen Zhu. Locally decodable/correctable codes for insertions and deletions. *arXiv preprint arXiv:2010.11989*, 2020. arXiv:2010.11989.

5 Jeremiah Blocki, Kuan Cheng, Elena Grigorescu, Xin Li, Yu Zheng, and Minshen Zhu. Exponential lower bounds for locally decodable and correctable codes for insertions and deletions. In *62nd Annual Symposium on Foundations of Computer Science*, pages 739–750, Denver, CO, USA, february 7–10 2022. IEEE Computer Society Press. doi:10.1109/FOCS52979.2021.00077.

6 Jeremiah Blocki, Shubhang Kulkarni, and Samson Zhou. On locally decodable codes in resource bounded channels. In Yael Tauman Kalai, Adam D. Smith, and Daniel Wichs, editors, *ITC 2020: 1st Conference on Information-Theoretic Cryptography*, volume 163 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:23, Boston, MA, USA, june 17–19 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITC.2020.16.

7 Jeremiah Blocki and Justin Zhang. Amortized locally decodable codes, 2025. doi:10.48550/arXiv.2502.10538.

8 Jeremiah Blocki and Justin Zhang. Amortized locally decodable codes for insertions and deletions. *arXiv preprint*, 2025. doi:10.48550/arXiv.2507.03141.

9 Alessandro Chiesa, Tom Gur, and Igor Shinkar. Relaxed locally correctable codes with nearly-linear block length and constant query complexity. In Shuchi Chawla, editor, *31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1395–1411, Salt Lake City, UT, USA, january 5–8 2020. ACM-SIAM. doi:10.1137/1.9781611975994.84.

10 Gil Cohen and Tal Yankovitz. Asymptotically-good rlccs with (log n)^(2+ o(1)) queries. In *39th Computational Complexity Conference (CCC 2024)*, pages 8:1–8:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CCC.2024.8.

11 Klim Efremenko. 3-query locally decodable codes of subexponential length. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 39–44, Bethesda, MD, USA, May 31 – june 2 2009. ACM Press. doi:10.1145/1536414.1536422.

12 Meghal Gupta. Constant query local decoding against deletions is impossible. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing*, pages 752–763, Vancouver, BC, Canada, june 24–28 2024. ACM Press. doi:10.1145/3618260.3649655.

13 Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94, pages 27:1–27:11, Cambridge, MA, USA, january 11–14 2018. Leibniz International Proceedings in Informatics (LIPIcs). doi:10.4230/LIPIcs.ITCS.2018.27.

14 Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 126–143, Santa Barbara, CA, USA, august 17–21 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-85174-5_8.

15 Brett Hemenway, Rafail Ostrovsky, Martin J Strauss, and Mary Wootters. Public key locally decodable codes with short keys. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 605–615. Springer, 2011. doi:10.1007/978-3-642-22935-0_51.

16 Don Hush and Clint Scovel. Concentration of the hypergeometric distribution. *Statistics & Probability Letters*, 75(2):127–132, November 2005. doi:10.1016/j.spl.2005.05.019.

17 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *32nd Annual ACM Symposium on Theory of Computing*, pages 80–86, Portland, OR, USA, May 21–23 2000. ACM Press. doi:10.1145/335305.335315.

18 Vinayak M. Kumar and Geoffrey Mon. Relaxed local correctability from local testing. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing*, pages 1585–1593, Vancouver, BC, Canada, june 24–28 2024. ACM Press. doi:10.1145/3618260.3649611.

19 Richard J. Lipton. A new approach to information theory. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '94, pages 699–708, Berlin, Heidelberg, 1994. Springer-Verlag. doi:10.1007/3-540-57785-8_183.

20 Rafail Ostrovsky, Omkant Pandey, and Amit Sahai. Private locally decodable codes. Cryptology ePrint Archive, Report 2007/025, 2007. URL: https://eprint.iacr.org/2007/025.

21 Rafail Ostrovsky, Omkant Pandey, and Amit Sahai. Private locally decodable codes. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 387–398, Wroclaw, Poland, july 9–13 2007. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-73420-8_35.

22 Rafail Ostrovsky and Anat Paskin-Cherniavsky. Locally decodable codes for edit distance. In Anja Lehmann and Stefan Wolf, editors, *ICITS 15: 8th International Conference on Information Theoretic Security*, volume 9063 of *Lecture Notes in Computer Science*, pages 236–249, Lugano, Switzerland, May 2–5 2015. Springer, Cham, Switzerland. doi:10.1007/978-3-319-17470-9_14.

23 L.J. Schulman and D. Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, 1999. doi:10.1109/18.796406.

24 Sergey Yekhanin et al. Locally decodable codes. *Foundations and Trends® in Theoretical Computer Science*, 6(3):139–255, 2012. doi:10.1561/0400000030.