

# SABLE-HE: A Sparse-LPN and Code-LPN Candidate for All-Code-Based Leveled Homomorphic Encryption

Research design draft

Prepared as a mathematical construction and validation plan

Draft v0.7 C5 arithmetic, baselines, and public-surface diagnostics – June 19, 2026

## Abstract

This document proposes SABLE-HE, a candidate post-quantum leveled homomorphic encryption construction for bounded-depth arithmetic circuits over a prime field. The construction combines a sparse-LPN GSW-style nonlinear evaluation layer with an LPN/code-based linearly homomorphic compaction layer. The purpose is assumption diversification: the design avoids Ring-LWE, Module-LWE, DDH, DCR, RSA, pairings, and Paillier-type assumptions, and instead relies on sparse-LPN and q-ary LPN/code-decoding assumptions. We give formal syntax, algorithms, correctness invariants, a hybrid security proof under stated assumptions, efficiency formulas, parameter constraints, and a validation plan. The result should be read as a research-grade CANDIDATE construction, not as a deployed or peer-reviewed cryptographic standard.

**Status and intended use.** This is a theory-and-design manuscript. It is suitable as the basis for a research paper, prototype library, security-estimation work, and cryptanalysis. It does not certify concrete parameters as secure. Any real deployment would require independent review, parameter estimation against current LPN and information-set-decoding attacks, side-channel review, and implementation audits.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Research objective . . . . .	2
1.2	Contributions . . . . .	2
1.3	Non-goals . . . . .	2
1.4	Relationship to post-quantum standardization . . . . .	3
<b>2</b>	<b>Preliminaries</b>	<b>3</b>
2.1	Notation . . . . .	3
2.2	q-ary symmetric noise . . . . .	3
2.3	Sparse q-ary LPN . . . . .	4
2.4	Dense q-ary LPN/code encryption . . . . .	4
2.5	Security target . . . . .	4
<b>3</b>	<b>Design overview</b>	<b>5</b>
3.1	Three ciphertext forms . . . . .	5
3.2	Why the construction is code-based . . . . .	5
3.3	Main technical invariant . . . . .	6

<b>4</b>	<b>Core primitives</b>	<b>6</b>
4.1	Sparse-LPN zero encryption . . . . .	6
4.2	Compact input encryption . . . . .	6
4.3	Sparse-LPN GSW-style zero matrix . . . . .	6
4.4	GSW-style scalar encryption . . . . .	7
4.5	Expansion from compact input to GSW form . . . . .	7
4.6	Code-LPN linearly homomorphic encryption . . . . .	8
<b>5</b>	<b>Full construction</b>	<b>8</b>
5.1	Parameters . . . . .	8
5.2	KeyGen . . . . .	9
5.3	Enc . . . . .	9
5.4	Expand . . . . .	9
5.5	Homomorphic evaluation . . . . .	9
5.6	Compact . . . . .	10
5.7	Dec . . . . .	10
5.8	Correctness intuition . . . . .	10
<b>6</b>	<b>Correctness analysis</b>	<b>10</b>
6.1	Good evaluation ciphertexts . . . . .	10
6.2	Fresh expansion quality . . . . .	11
6.3	Addition and multiplication invariants . . . . .	11
6.4	Depth bound . . . . .	12
6.5	Compaction correctness . . . . .	12
<b>7</b>	<b>Security analysis</b>	<b>13</b>
7.1	Pseudorandomness of zero encryptions . . . . .	13
7.2	IND-CPA theorem . . . . .	13
7.3	What the proof does and does not cover . . . . .	14
7.4	Relevant attack families . . . . .	14
<b>8</b>	<b>Efficiency and parameter constraints</b>	<b>15</b>
8.1	Ciphertext and key sizes . . . . .	15
8.2	Evaluation costs . . . . .	15
8.3	Correctness budget . . . . .	15
8.4	Illustrative non-secure toy parameters . . . . .	16
8.5	Parameter-selection recipe . . . . .	16
8.6	Benchmark baselines . . . . .	16
<b>9</b>	<b>Validation plan</b>	<b>16</b>
9.1	Formal proof track . . . . .	17
9.2	Parameter-estimation track . . . . .	17
9.3	Prototype track . . . . .	17
9.4	Benchmark track . . . . .	18
9.5	Success criteria for a publishable paper . . . . .	18
<b>10</b>	<b>Limitations and future work</b>	<b>18</b>
10.1	Limitations . . . . .	18
10.2	Future work . . . . .	19
10.3	Recommended claim language . . . . .	19

<b>A</b>	<b>Implementation-oriented pseudocode</b>	<b>19</b>
A.1	Noise and sparse sampling . . . . .	20
A.2	Compact encryption . . . . .	20
A.3	GSW-style scalar encryption . . . . .	20
A.4	Code-LPN encryption . . . . .	21
A.5	Key generation and evaluation . . . . .	21
<b>B</b>	<b>Proof of the q-ary piling-up formula</b>	<b>22</b>
<b>C</b>	<b>Reviewer and cryptanalysis checklist</b>	<b>22</b>
C.1	Assumptions . . . . .	22
C.2	Correctness . . . . .	23
C.3	Security proof . . . . .	23
C.4	Implementation . . . . .	23
<b>D</b>	<b>Attack-Screening Addendum</b>	<b>23</b>
D.1	Screened attack surfaces . . . . .	23
D.2	Early finding: correctness/security tension . . . . .	24
D.3	Design consequence . . . . .	25
<b>E</b>	<b>C2 Block-Dictionary Compaction</b>	<b>25</b>
<b>F</b>	<b>C3 Seeded Block-Dictionary Storage</b>	<b>26</b>
<b>G</b>	<b>C2/C3 Public-Sample Surface Screen</b>	<b>27</b>
<b>H</b>	<b>C4 Projective Sparse Additive-Basis Compaction</b>	<b>28</b>
H.1	Projective dictionary idea . . . . .	29
H.2	C4 correctness theorem . . . . .	29
H.3	Public-surface comparison . . . . .	30
H.4	Validation status . . . . .	30
<b>I</b>	<b>C5 Arithmetic Operation Suite and External Baseline Plan</b>	<b>30</b>
<b>J</b>	<b>C5 Baseline and Public-Surface Diagnostics</b>	<b>31</b>

## 1 Introduction

Homomorphic encryption (HE) allows a server to evaluate functions on encrypted data and return an encrypted result. Modern practical HE is dominated by lattice assumptions, especially LWE, RLWE, and their ring/module variants. This dominance is scientifically understandable: lattice HE has a mature lineage from Gentry’s first FHE construction, BGV, GSW, BFV, CKKS, FHEW, and TFHE [16, 4, 17, 15, 11, 13, 10]. However, it also creates an assumption-diversity problem. Post-quantum cryptography already benefits from several hardness families: lattices, codes, multivariate equations, hash-based signatures, and isogeny-style problems. In contrast, post-quantum HE remains much more concentrated around lattices.

Recent work makes a code-based direction credible. Corrigan-Gibbs, Henzinger, Kalai, and Vaikuntanathan construct somewhat homomorphic encryption from sparse LPN plus a linearly homomorphic encryption primitive [12]. Their work is the closest technical ancestor of this document. It shows that sparse LPN can support bounded multiplication through a GSW-like approximate-eigenvector method. Separately, Bitzer, Egger, Liu, and Wachter-Zeh propose post-quantum secure aggregation via code-based homomorphic encryption under LPN-style assumptions [7]. A recent review also explicitly identifies code-based HE as an underexplored

direction for post-quantum homomorphic encryption [6]. Random-code/rank-metric SHE has also recently been studied [1].

The construction proposed here is called SABLE-HE, short for *Sparse-LPN All-Code Bounded-depth Leveled Homomorphic Encryption*. Its central idea is:

$$\boxed{\text{sparse-LPN GSW-style nonlinear evaluation} + \text{q-ary LPN/code-based linear compaction} = \text{all-code-based leveled SHE}}$$

The sparse-LPN layer handles bounded multiplication. The q-ary code/LPN layer homomorphically evaluates the final linear decryption map and compresses the output. The proposed compaction layer is the main design change relative to sparse-LPN SHE constructions that instantiate linear homomorphism using non-code assumptions such as DDH or DCR.

## 1.1 Research objective

The objective is not to claim a complete full FHE scheme. The objective is to design a mathematically coherent, non-lattice, post-quantum candidate for the following target functionality:

$$\mathcal{F}_{D,A,q} = \left\{ f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q \mid f \text{ is represented by an arithmetic circuit of multiplicative depth } D \text{ and addition budget } A \right\}$$

Here  $q \geq 3$  is prime. Boolean inputs can be embedded as  $0, 1 \in \mathbb{F}_q$ . Arithmetic multiplication gives AND, while OR, NOT, and XOR can be represented by low-degree polynomials:

$$\text{AND}(x, y) = xy, \quad \text{OR}(x, y) = x + y - xy, \quad \text{NOT}(x) = 1 - x, \quad \text{XOR}(x, y) = x + y - 2xy.$$

This makes SABLE-HE a candidate for private low-degree analytics, voting/counting with nonlinear checks, secure aggregation plus low-degree validation, private feature-interaction models, and small Boolean circuits.

## 1.2 Contributions

This draft makes four contributions.

1. It formulates an all-code-based leveled SHE candidate using two independent LPN-style layers: sparse LPN for nonlinear evaluation and q-ary LPN/code encryption for linear compaction.
2. It gives precise algorithms: **KeyGen**, compact input **Enc**, **Expand**, **EvalAdd**, **EvalMul**, **Compact**, and **Dec**.
3. It proves correctness by tracking a row-sparsity and coordinate-error invariant. The invariant is inherited from the approximate-eigenvector technique of GSW and from sparse-LPN SHE analysis [17, 12].
4. It gives a hybrid IND-CPA security proof under explicit sparse-LPN and q-ary LPN assumptions, plus an attack and validation checklist for parameter selection.

## 1.3 Non-goals

The following are intentionally outside the first version.

- **No full bootstrapping claim.** The construction is leveled and supports a predetermined bounded multiplicative depth.
- **No deployment-grade concrete parameters.** Parameter tables in this document are formulas and illustrative examples only. Security estimation must be done separately.

- **No CCA security.** The security target is IND-CPA for a secret-key HE scheme with public evaluation keys.
- **No claim of superiority over lattice FHE.** Mature lattice libraries remain the practical baseline. The scientific contribution is assumption diversity and a code-based path to bounded homomorphism.

## 1.4 Relationship to post-quantum standardization

NIST’s first post-quantum standards are for key establishment and signatures, not homomorphic encryption. FIPS 203 specifies ML-KEM and relates its security to Module-LWE, while FIPS 204 specifies ML-DSA for digital signatures [21, 22]. HE security guidance is maintained separately by the HomomorphicEncryption.org community [2, 18]. Thus SABLE-HE should not be read as a NIST-standardized primitive. It is a research proposal in the code-based HE family.

## 2 Preliminaries

### 2.1 Notation

Let  $q \geq 3$  be prime and let  $\mathbb{F}_q$  be the field of  $q$  elements. Vectors are column vectors unless stated otherwise. For  $x \in \mathbb{F}_q^n$ ,  $\text{wt}(x)$  denotes Hamming weight and  $\text{supp}(x)$  denotes support. For  $s \in \mathbb{F}_q^n$ , define the extended secret

$$\tilde{s} = (-s_1, \dots, -s_n, 1) \in \mathbb{F}_q^{n+1}.$$

For  $c = (a, b) \in \mathbb{F}_q^n \times \mathbb{F}_q$ , decryption-style inner products are written

$$c \cdot \tilde{s} = -a^\top s + b.$$

All probabilities are over the randomness of key generation, encryption, and noise unless otherwise specified.

### 2.2 q-ary symmetric noise

For  $0 \leq \eta \leq 1$ , define the  $q$ -ary symmetric noise distribution  $\chi_{q,\eta}$  over  $\mathbb{F}_q$  by

$$e = 0 \text{ with probability } 1 - \eta, \quad e = u \text{ with probability } \eta/(q - 1) \text{ for each } u \in \mathbb{F}_q^*.$$

This is the natural  $q$ -ary analogue of Bernoulli noise for LPN. It is also useful for coding-theoretic decoding because an error coordinate is either correct or replaced by a uniformly random nonzero field element.

**Lemma 2.1** ( $q$ -ary piling-up). *Let  $E_1, \dots, E_B$  be independent  $q$ -ary symmetric errors with error probability  $\eta$ . Let  $\alpha_i \in \mathbb{F}_q^*$  be nonzero coefficients. Then*

$$E = \sum_{i=1}^B \alpha_i E_i$$

*is  $q$ -ary symmetric with error probability*

$$\eta_B = \frac{q-1}{q} \left( 1 - \left( 1 - \frac{q\eta}{q-1} \right)^B \right).$$

*In particular,  $\eta_B \leq B\eta$  by the union bound.*

The proof is included in Appendix B. The exact expression is useful for compaction parameters; the union bound is useful for simple correctness estimates.

### 2.3 Sparse q-ary LPN

Let  $S_{n,k,q}$  be the distribution over  $\mathbb{F}_q^n$  that samples a uniformly random support of size  $k$  and assigns each supported position a uniformly random nonzero element of  $\mathbb{F}_q^*$ . A sparse q-ary LPN sample with secret  $s \in \mathbb{F}_q^n$  is

$$(a, a^\top s + e), \quad a \leftarrow S_{n,k,q}, \quad e \leftarrow \chi_{q,\eta}.$$

**Assumption 2.2** (Sparse q-ary LPN). For parameters  $(n, k, q, \eta, M)$ , the distribution of  $M$  sparse q-ary LPN samples

$$\{(a_i, a_i^\top s + e_i)\}_{i=1}^M$$

with  $s \leftarrow \mathbb{F}_q^n$ ,  $a_i \leftarrow S_{n,k,q}$ , and  $e_i \leftarrow \chi_{q,\eta}$  is computationally indistinguishable from

$$\{(a_i, u_i)\}_{i=1}^M, \quad u_i \leftarrow \mathbb{F}_q.$$

Sparse LPN is a coding-theoretic noisy-decoding assumption. Classical LPN and related noisy linear-equation assumptions are connected to learning parity with noise, random linear codes, and average-case coding hardness [8, 3, 5]. Sparse-LPN variants have recently been used for homomorphic encryption [12].

### 2.4 Dense q-ary LPN/code encryption

The compaction layer uses a standard q-ary LPN/code primitive. A dense q-ary LPN sample is

$$(A, Ar + e)$$

for secret  $r \in \mathbb{F}_q^{n_c}$ , matrix  $A \in \mathbb{F}_q^{m_c \times n_c}$ , and error vector  $e \in \mathbb{F}_q^{m_c}$  with independent  $\chi_{q,\eta_c}$  coordinates. Message encryption adds a codeword  $\text{Code}(x)$  to  $Ar + e$ . In the simplest prototype,  $\text{Code}(x) = x\mathbf{1}$  is a repetition code. A paper-grade version should replace repetition by a stronger q-ary error-correcting code.

**Assumption 2.3** (q-ary LPN/code pseudorandomness). For parameters  $(n_c, m_c, q, \eta_c, M_c)$ , polynomially many ciphertext blocks of the form

$$(A_j, A_j r + e_j + \text{Code}(x_j))$$

are semantically secure for messages  $x_j \in \mathbb{F}_q$  under the q-ary LPN assumption and the chosen code/decoder.

This assumption is intentionally stated at the primitive level because the eventual library may instantiate  $\text{Code}$  by repetition, BCH-like q-ary codes, LDPC-like codes, or CRT-based code decompositions.

### 2.5 Security target

The target is secret-key IND-CPA security with public evaluation key. The adversary may see the evaluation key, request polynomially many encryptions, choose two challenge messages, receive an encryption of one of them, and run arbitrary public homomorphic evaluation. Since evaluation is deterministic and public, security follows from ciphertext and evaluation-key pseudorandomness under the stated assumptions.

### 3 Design overview

SABLE-HE uses three independent secrets:

$$t \in \mathbb{F}_q^n, \quad s \in \mathbb{F}_q^n, \quad r \in \mathbb{F}_q^{n_c}.$$

The secret  $t$  is used for compact input encryption. The secret  $s$  is used for GSW-style matrix ciphertexts and nonlinear evaluation. The secret  $r$  is used only for the code/LPN compaction layer.

The key chain is

$$t \longrightarrow s \longrightarrow r.$$

An expansion key encrypts the coordinates of  $\tilde{t}$  under the GSW-style secret  $s$ . This turns compact sparse-LPN input ciphertexts into matrix ciphertexts. A compaction key encrypts the coordinates of  $\tilde{s}$  under the code/LPN compaction secret  $r$ . This lets the evaluator homomorphically compute the final linear decryption form and return a compact output.

#### 3.1 Three ciphertext forms

**Input ciphertexts.** A compact input ciphertext under  $t$  has the form

$$c = (a, b) = (a, a^\top t + \mu + e) \in \mathbb{F}_q^{n+1},$$

where  $a$  is  $k$ -sparse. It satisfies

$$c \cdot \tilde{t} = \mu + e.$$

This is small and suitable for data-owner encryption.

**Evaluation ciphertexts.** A GSW-style evaluation ciphertext under  $s$  is a matrix  $C \in \mathbb{F}_q^{N \times N}$ , where  $N = n + 1$ , satisfying

$$C\tilde{s} = \mu\tilde{s} + e.$$

Homomorphic addition and multiplication are simply

$$C_1 + C_2, \quad C_1 C_2.$$

The price is that row support and error grow with circuit depth.

**Compact output ciphertexts.** After evaluation, if

$$C_* \tilde{s} = f(\mu_1, \dots, \mu_m) \tilde{s} + e_*,$$

then the last row  $\rho_*$  satisfies

$$\rho_* \cdot \tilde{s} = f(\mu_1, \dots, \mu_m) + e_{*,N}.$$

The evaluator uses code/LPN encryptions of  $\tilde{s}_i$  to homomorphically compute an encryption of  $\rho_* \cdot \tilde{s}$ . This is the compaction step.

#### 3.2 Why the construction is code-based

Both hardness assumptions are coding-theoretic.

- Sparse LPN is a noisy sparse linear-code decoding problem. It supplies compact input encryption and the GSW-style nonlinear layer.
- $q$ -ary LPN/code encryption is a noisy random linear-code primitive. It supplies linearly homomorphic compaction.

No lattice, number-theoretic, or pairing assumption appears in the construction. This is the intended novelty.

### 3.3 Main technical invariant

The key invariant is that evaluation ciphertexts remain *approximately eigenvector matrices*. A ciphertext  $C$  encrypting  $\mu$  satisfies

$$C\tilde{s} = \mu\tilde{s} + e.$$

Then

$$(C_1 + C_2)\tilde{s} = (\mu_1 + \mu_2)\tilde{s} + (e_1 + e_2),$$

and

$$C_1C_2\tilde{s} = C_1(\mu_2\tilde{s} + e_2) = \mu_1\mu_2\tilde{s} + \mu_2e_1 + C_1e_2.$$

Thus multiplication is valid if  $C_1e_2$  remains sparse/noisy enough. This is why row support matters.

## 4 Core primitives

Let  $N = n + 1$ . For  $u \in \mathbb{F}_q^n$ , write  $\tilde{u} = (-u, 1) \in \mathbb{F}_q^N$ .

### 4.1 Sparse-LPN zero encryption

**Construction 4.1** (Sparse-LPN zero vector encryption). For secret  $u \in \mathbb{F}_q^n$ , sample  $a \leftarrow S_{n,k,q}$  and  $e \leftarrow \chi_{q,\eta}$ . Output

$$z = (a, a^\top u + e) \in \mathbb{F}_q^N.$$

Then

$$z \cdot \tilde{u} = e.$$

The row support is at most  $k + 1$ .

This is a noisy encryption of zero. Adding a message in the last coordinate gives compact message encryption.

### 4.2 Compact input encryption

**Construction 4.2** (Compact sparse-LPN input encryption). For secret  $t \in \mathbb{F}_q^n$  and message  $\mu \in \mathbb{F}_q$ :

1. Sample  $z = (a, a^\top t + e)$  as in Construction 4.1.
2. Output

$$c = z + \mu e_N = (a, a^\top t + e + \mu),$$

where  $e_N = (0, \dots, 0, 1) \in \mathbb{F}_q^N$ .

Then

$$c \cdot \tilde{t} = \mu + e.$$

The decryption error is not rounded away; it is corrected statistically by repetition and final plurality. This is natural in the LPN/code setting.

### 4.3 Sparse-LPN GSW-style zero matrix

**Construction 4.3** (Sparse-LPN GSW-style zero matrix). For secret  $s \in \mathbb{F}_q^n$ , sample  $N$  independent sparse-LPN zero rows

$$z_j = (a_j, a_j^\top s + e_j), \quad j = 1, \dots, N.$$

Let  $Z \in \mathbb{F}_q^{N \times N}$  be the matrix with rows  $z_j$ . Then

$$Z\tilde{s} = e,$$

where  $e = (e_1, \dots, e_N)^\top$ . Every row has support at most  $k + 1$ .



#### 4.4 GSW-style scalar encryption

**Construction 4.4** (Sparse-LPN GSW-style scalar encryption). To encrypt  $\alpha \in \mathbb{F}_q$  under  $s$ :

1. Sample  $Z$  as in Construction 4.3.
2. Output

$$X = Z + \alpha I_N.$$

Then

$$X\tilde{s} = \alpha\tilde{s} + e.$$

Each row has support at most  $k + 2$ .

This formulation is important for security:  $X$  is a pseudorandom zero-encryption matrix plus a public diagonal shift. This mirrors the approximate-eigenvector technique in GSW-style encryption [17] and its sparse-LPN adaptation [12].

#### 4.5 Expansion from compact input to GSW form

The expansion key contains GSW encryptions of the coordinates of  $\tilde{t}$ :

$$\text{EK}_i^{\text{exp}} = \text{GSWEnc}_s(\tilde{t}_i), \quad i = 1, \dots, N.$$

Given compact ciphertext  $c = (c_1, \dots, c_N)$ , define

$$\text{Expand}(c) = \sum_{i=1}^N c_i \text{EK}_i^{\text{exp}}.$$

Since  $c$  is sparse with at most  $k + 1$  nonzero coordinates, expansion sums at most  $k + 1$  GSW matrices.

**Lemma 4.5** (Expansion identity). *Let  $c$  be a compact encryption of  $\mu$  under  $t$ . Let  $C = \text{Expand}(c)$ . Then*

$$C\tilde{s} = \mu\tilde{s} + e',$$

where

$$e' = e_0\tilde{s} + \sum_{i \in \text{supp}(c)} c_i e^{(i)}.$$

Here  $e_0 = c \cdot \tilde{t} - \mu$  is the input error and  $e^{(i)}$  is the GSW error vector in  $\text{EK}_i^{\text{exp}}$ .

*Proof.* For each  $i$ ,

$$\text{EK}_i^{\text{exp}}\tilde{s} = \tilde{t}_i\tilde{s} + e^{(i)}.$$

Therefore

$$C\tilde{s} = \sum_i c_i \tilde{t}_i \tilde{s} + \sum_i c_i e^{(i)} = (c \cdot \tilde{t})\tilde{s} + \sum_i c_i e^{(i)}.$$

Since  $c \cdot \tilde{t} = \mu + e_0$ , the claim follows. □

## 4.6 Code-LPN linearly homomorphic encryption

Let  $\text{Code} : \mathbb{F}_q \rightarrow \mathbb{F}_q^{m_c}$  be a linear error-correcting encoding. For a first prototype,  $\text{Code}(x) = x\mathbf{1}$ . Let  $\text{Decode}$  be its decoder.

**Construction 4.6** (CLPN encryption). For secret  $r \in \mathbb{F}_q^{n_c}$  and message  $x \in \mathbb{F}_q$ :

1. Sample  $A \leftarrow \mathbb{F}_q^{m_c \times n_c}$ .
2. Sample  $e \in \mathbb{F}_q^{m_c}$  with independent  $\chi_{q, \eta_c}$  coordinates.
3. Output

$$Z = (A, b = Ar + \text{Code}(x) + e).$$

Decryption computes

$$y = b - Ar = \text{Code}(x) + e$$

and outputs  $\text{Decode}(y)$ .

**Construction 4.7** (CLPN linear evaluation). Given ciphertexts  $Z_i = (A_i, b_i)$  encrypting  $x_i$  and coefficients  $\alpha_i \in \mathbb{F}_q$ , output

$$\sum_i \alpha_i Z_i = \left( \sum_i \alpha_i A_i, \sum_i \alpha_i b_i \right).$$

This decrypts to

$$\text{Code} \left( \sum_i \alpha_i x_i \right) + \sum_i \alpha_i e_i.$$

## 5 Full construction

This section defines the SABLE-HE scheme. The scheme is secret-key encryption with a public evaluation key.

### 5.1 Parameters

A parameter set is

$$\Pi = (q, n, k, \eta, D, A, R, n_c, m_c, \eta_c, \text{Code}).$$

The parameters mean:

Parameter	Meaning
$q$	prime plaintext/noise field modulus, $q \geq 3$
$n$	sparse-LPN secret dimension
$k$	sparse row weight
$\eta$	sparse-LPN q-ary error probability
$D$	supported multiplicative depth
$A$	conservative addition/monomial budget
$R$	number of independent replicas for failure amplification
$n_c, m_c$	compactor secret length and code block length
$\eta_c$	compactor q-ary error probability
$\text{Code}$	linear q-ary error-correcting code for compaction

Let  $N = n + 1$ .

## 5.2 KeyGen

---

KeyGen

1. Choose parameters  $\Pi$  for security parameter  $\lambda$  and depth  $D$ .
2. Sample independent secrets

$$t \leftarrow \mathbb{F}_q^n, \quad s \leftarrow \mathbb{F}_q^n, \quad r \leftarrow \mathbb{F}_q^{n_c}.$$

3. Define  $\tilde{t} = (-t, 1)$  and  $\tilde{s} = (-s, 1)$ .
4. For each  $i \in [N]$ , generate an expansion-key matrix

$$\text{EK}_i^{\text{exp}} = \text{GSWEnc}_s(\tilde{t}_i).$$

5. For each  $i \in [N]$ , generate a compaction-key ciphertext

$$\text{EK}_i^{\text{cmp}} = \text{CLPNEnc}_r(\tilde{s}_i).$$

6. Output

$$\text{sk} = (t, s, r), \quad \text{ek} = (\Pi, \text{EK}^{\text{exp}}, \text{EK}^{\text{cmp}}).$$


---

## 5.3 Enc

For a message  $\mu \in \mathbb{F}_q$ , generate  $R$  independent compact ciphertexts:

$$c^{(j)} = \text{RegevEnc}_t(\mu), \quad j = 1, \dots, R.$$

Return

$$\text{ct} = (c^{(1)}, \dots, c^{(R)}).$$

Replicas are used only to reduce the decryption-failure probability. They may be omitted in toy experiments.

## 5.4 Expand

For each replica,

$$C^{(j)} = \sum_{i=1}^N c_i^{(j)} \text{EK}_i^{\text{exp}}.$$

Return

$$\text{CT} = (C^{(1)}, \dots, C^{(R)}).$$

The result is a vector of GSW-style matrix ciphertexts under secret  $s$ .

## 5.5 Homomorphic evaluation

For expanded ciphertexts

$$\text{CT}_1 = (C_1^{(1)}, \dots, C_1^{(R)}), \quad \text{CT}_2 = (C_2^{(1)}, \dots, C_2^{(R)}),$$

define

$$\text{EvalAdd}(\text{CT}_1, \text{CT}_2) = \left( C_1^{(j)} + C_2^{(j)} \right)_{j=1}^R,$$

and

$$\text{EvalMul}(\text{CT}_1, \text{CT}_2) = \left( C_1^{(j)} C_2^{(j)} \right)_{j=1}^R.$$

Multiplication may be oriented so that the matrix with smaller row support is placed on the left, because the error bound for multiplication is asymmetric.

## 5.6 Compact

For each evaluated replica  $C_*^{(j)}$ :

1. Let  $\rho_*^{(j)} \in \mathbb{F}_q^N$  be the last row of  $C_*^{(j)}$ .
2. Homomorphically evaluate the linear form  $x \mapsto \rho_*^{(j)} \cdot x$  on the compaction-key encryptions of  $\tilde{s}_i$ :

$$Z^{(j)} = \text{CLPN.EvalLin} \left( \rho_*^{(j)}; \text{EK}_1^{\text{cmp}}, \dots, \text{EK}_N^{\text{cmp}} \right).$$

Return

$$\text{ct}_{\text{out}} = (Z^{(1)}, \dots, Z^{(R)}).$$

## 5.7 Dec

For each replica, compute

$$y^{(j)} = \text{CLPNDec}_r(Z^{(j)}).$$

Return the plurality value among  $y^{(1)}, \dots, y^{(R)}$ .

## 5.8 Correctness intuition

If evaluation gives

$$C_*^{(j)} \tilde{s} = f(\mu_1, \dots, \mu_m) \tilde{s} + e_*^{(j)},$$

then the last row  $\rho_*^{(j)}$  satisfies

$$\rho_*^{(j)} \cdot \tilde{s} = f(\mu_1, \dots, \mu_m) + e_{*,N}^{(j)}.$$

The compaction key encrypts the coordinates of  $\tilde{s}$ . Therefore **Compact** produces a code/LPN encryption of the decryption value. If both the sparse-LPN evaluation error and the code/LPN compaction error are decoded correctly, decryption returns  $f(\mu_1, \dots, \mu_m)$ .

# 6 Correctness analysis

## 6.1 Good evaluation ciphertexts

**Definition 6.1** (Good GSW-style ciphertext). Let  $C \in \mathbb{F}_q^{N \times N}$ . We say that  $C$  is  $(w, \varepsilon)$ -good for  $\mu \in \mathbb{F}_q$  under secret  $s$  if:

1. every row of  $C$  has Hamming support at most  $w$ ;
2. writing

$$C\tilde{s} = \mu\tilde{s} + e,$$

every coordinate satisfies

$$\Pr[e_j \neq 0] \leq \varepsilon.$$

This definition tracks only row support and coordinatewise error probability. It deliberately does not require independence between all error coordinates after evaluation. The proofs use union bounds, so this weaker invariant is enough.

## 6.2 Fresh expansion quality

Let

$$w_0 = (k+1)(k+2), \quad \varepsilon_0 = (k+2)\eta.$$

**Lemma 6.2** (Fresh expanded ciphertext quality). *For a compact encryption  $c = \text{RegevEnc}_t(\mu)$ , the expanded ciphertext  $C = \text{Expand}(c)$  is  $(w_0, \varepsilon_0)$ -good for  $\mu$ .*

*Proof.* The compact ciphertext  $c$  has at most  $k+1$  nonzero coordinates: at most  $k$  from  $a$  and one final coordinate. Each expansion-key matrix has row support at most  $k+2$  because it is a sparse-LPN zero matrix plus a diagonal shift. Therefore each row of  $C = \sum_i c_i \text{EK}_i^{\text{exp}}$  has support at most  $(k+1)(k+2) = w_0$ .

By Lemma 4.5,

$$C\tilde{s} = \mu\tilde{s} + e_0\tilde{s} + \sum_{i \in \text{supp}(c)} c_i e^{(i)}.$$

For a fixed coordinate  $j$ , the term  $e_0\tilde{s}_j$  is nonzero only if  $e_0 \neq 0$  and  $\tilde{s}_j \neq 0$ , so it is nonzero with probability at most  $\eta$ . The summation contains at most  $k+1$  independent GSW error coordinates, each nonzero with probability  $\eta$ . A union bound gives

$$\Pr[e'_j \neq 0] \leq \eta + (k+1)\eta = (k+2)\eta.$$

□

## 6.3 Addition and multiplication invariants

**Lemma 6.3** (Homomorphic addition). *If  $C_1$  is  $(w_1, \varepsilon_1)$ -good for  $\mu_1$  and  $C_2$  is  $(w_2, \varepsilon_2)$ -good for  $\mu_2$ , then  $C_1 + C_2$  is  $(w_1 + w_2, \varepsilon_1 + \varepsilon_2)$ -good for  $\mu_1 + \mu_2$ .*

*Proof.* Row support is subadditive. Also,

$$(C_1 + C_2)\tilde{s} = C_1\tilde{s} + C_2\tilde{s} = (\mu_1 + \mu_2)\tilde{s} + e_1 + e_2.$$

For each coordinate,  $e_{1,j} + e_{2,j}$  is nonzero only if at least one of  $e_{1,j}, e_{2,j}$  is nonzero, except for possible cancellation. Thus

$$\Pr[e_{1,j} + e_{2,j} \neq 0] \leq \varepsilon_1 + \varepsilon_2.$$

□

**Lemma 6.4** (Homomorphic multiplication). *If  $C_1$  is  $(w_1, \varepsilon_1)$ -good for  $\mu_1$  and  $C_2$  is  $(w_2, \varepsilon_2)$ -good for  $\mu_2$ , then  $C_1 C_2$  is  $(w_1 w_2, \varepsilon_1 + w_1 \varepsilon_2)$ -good for  $\mu_1 \mu_2$ .*

*Proof.* A row of  $C_1 C_2$  is a linear combination of at most  $w_1$  rows of  $C_2$ , each of support at most  $w_2$ . Hence the row support is at most  $w_1 w_2$ .

For correctness,

$$C_2\tilde{s} = \mu_2\tilde{s} + e_2,$$

so

$$C_1 C_2 \tilde{s} = C_1(\mu_2\tilde{s} + e_2) = \mu_2 C_1 \tilde{s} + C_1 e_2.$$

Using  $C_1 \tilde{s} = \mu_1 \tilde{s} + e_1$ , we get

$$C_1 C_2 \tilde{s} = \mu_1 \mu_2 \tilde{s} + \mu_2 e_1 + C_1 e_2.$$

For a fixed output coordinate  $j$ , the term  $(\mu_2 e_1)_j$  is nonzero with probability at most  $\varepsilon_1$ . The  $j$ th coordinate of  $C_1 e_2$  depends on at most  $w_1$  coordinates of  $e_2$ , each nonzero with probability at most  $\varepsilon_2$ . By a union bound,

$$\Pr[(C_1 e_2)_j \neq 0] \leq w_1 \varepsilon_2.$$

Combining the two bad events gives the claimed bound.

□

## 6.4 Depth bound

For a balanced multiplication tree of depth  $D$  starting from fresh expanded ciphertexts, define recursively

$$w_{r+1} = w_r^2, \quad \varepsilon_{r+1} \leq (1 + w_r)\varepsilon_r,$$

with  $w_0 = (k+1)(k+2)$  and  $\varepsilon_0 = (k+2)\eta$ . Hence

$$w_D = w_0^{2^D}.$$

A conservative closed-form bound is

$$\varepsilon_D \leq \varepsilon_0 \prod_{r=0}^{D-1} (1 + w_r) \leq (k+2)\eta \prod_{r=0}^{D-1} (1 + w_0^{2^r}).$$

For  $w_0 \geq 2$ , this is roughly

$$\varepsilon_D = O\left(\eta w_0^{2^D-1}\right).$$

If at most  $A$  additive branches are combined, a conservative final bound is

$$\varepsilon_f \leq A(k+2)\eta \prod_{r=0}^{D-1} (1 + w_0^{2^r}).$$

## 6.5 Compaction correctness

Let  $B$  be the number of nonzero entries in the final row  $\rho_*$  used for compaction. For a repetition-code compactor,  $B$  independent nonzero scalar multiples of  $q$ -ary symmetric errors yield effective error probability

$$\eta_{c,B} = \frac{q-1}{q} \left( 1 - \left( 1 - \frac{q\eta_c}{q-1} \right)^B \right).$$

For a general code `Code`, let

$$\varepsilon_{\text{cmp}}(B)$$

be the decoder's failure probability at this effective error rate and block length  $m_c$ .

**Theorem 6.5** (One-replica correctness). *Suppose  $C_*$  is  $(w, \varepsilon)$ -good for  $\mu_* = f(\mu_1, \dots, \mu_m)$  and the final row has support  $B \leq w$ . Then one compacted replica decrypts correctly with probability at least*

$$1 - \varepsilon - \varepsilon_{\text{cmp}}(B).$$

*Proof.* Write

$$C_* \tilde{s} = \mu_* \tilde{s} + e_*.$$

Let  $\rho_*$  be the last row. Since  $\tilde{s}_N = 1$ ,

$$\rho_* \cdot \tilde{s} = \mu_* + e_{*,N}.$$

By goodness,  $\Pr[e_{*,N} \neq 0] \leq \varepsilon$ . If  $e_{*,N} = 0$ , the compactor homomorphically evaluates the linear form  $\rho_* \cdot x$  on encryptions of  $\tilde{s}_i$  and therefore decrypts to  $\mu_*$  unless the code decoder fails. The decoder failure probability is  $\varepsilon_{\text{cmp}}(B)$ . A union bound proves the theorem.  $\square$

**Theorem 6.6** (Replicated correctness). *Let*

$$p = \varepsilon + \varepsilon_{\text{cmp}}(B) < 1/2.$$

*With  $R$  independent replicas and plurality decoding, the final failure probability is at most*

$$\exp\left(-2R \left(\frac{1}{2} - p\right)^2\right).$$

*Proof.* Treat each replica as a Bernoulli trial that fails with probability at most  $p$ . Majority/plurality fails only if at least half the replicas fail. Hoeffding's inequality gives the stated bound. For  $q > 2$ , plurality failure is at most the majority-failure event in which the correct value appears in fewer than half the replicas.  $\square$

## 7 Security analysis

The security claim is conditional and modular. The construction is secure if the sparse-LPN and  $q$ -ary LPN/code distributions used in its layers are pseudorandom for the chosen parameter sizes and number of samples. Concrete parameters require separate estimation.

### 7.1 Pseudorandomness of zero encryptions

The following proposition is immediate from Assumption 2.2.

**Proposition 7.1** (Sparse-LPN zero ciphertext pseudorandomness). *Sparse-LPN zero vectors of the form*

$$(a, a^\top u + e)$$

*are indistinguishable from sparse-supported vectors whose final coordinate is uniform in  $\mathbb{F}_q$ . Likewise, sparse-LPN zero matrices  $Z$  sampled rowwise are indistinguishable from matrices with the same row-support distribution and uniform final row entries.*

Because  $\text{GSWEnc}_s(\alpha) = Z + \alpha I_N$ , semantic security of GSW-style scalar encryption follows by adding a fixed public shift to a pseudorandom matrix. If the encrypted scalar is secret but independent of the GSW secret, the same argument applies after conditioning on that scalar.

### 7.2 IND-CPA theorem

**Theorem 7.2** (IND-CPA security of SABLE-HE). *Assume:*

1. *sparse  $q$ -ary LPN pseudorandomness for all compact input ciphertext samples under  $t$ ;*
2. *sparse  $q$ -ary LPN pseudorandomness for all GSW-style zero rows under  $s$  in the expansion key;*
3.  *$q$ -ary LPN/code semantic security for all compaction-key ciphertexts under  $r$ .*

*Then SABLE-HE is IND-CPA secure for polynomially many encryption queries and polynomial-size public evaluation keys.*

*Proof.* Let  $\mathcal{A}$  be a polynomial-time IND-CPA adversary. We define hybrids.

**Hybrid H0: real experiment.** The adversary receives the real evaluation key

$$\text{ek} = (\text{EK}^{\text{exp}}, \text{EK}^{\text{cmp}})$$

and oracle encryptions under  $t$ . The challenge ciphertext encrypts either  $\mu_0$  or  $\mu_1$  under  $t$ .

**Hybrid H1: replace the compaction key.** Replace each compaction-key ciphertext

$$\text{EK}_i^{\text{cmp}} = \text{CLPNEnc}_r(\tilde{s}_i)$$

by an encryption of zero, or equivalently by a pseudorandom  $q$ -ary LPN/code ciphertext. The messages  $\tilde{s}_i$  may depend on  $s$  but are independent of the compaction secret  $r$ . By Assumption 2.3, the adversary's distinguishing advantage changes by at most a negligible quantity.

**Hybrid H2: replace the expansion key.** Replace each expansion-key matrix

$$\text{EK}_i^{\text{exp}} = \text{GSWEnc}_s(\tilde{t}_i) = Z_i + \tilde{t}_i I_N$$

by a pseudorandom matrix with the same public support profile. Conditioned on  $t$ , each  $\tilde{t}_i I_N$  is a fixed shift and  $Z_i$  is a sparse-LPN zero matrix under secret  $s$ . By Assumption 2.2, replacing all such matrices changes the adversary's advantage negligibly.

**Hybrid H3: replace encryption-oracle outputs.** Now the evaluation key reveals no computational information about  $t$ . Replace all compact input ciphertexts, including the challenge ciphertext, by sparse-supported pseudorandom vectors. Each compact encryption has the form

$$(a, a^\top t + e + \mu).$$

For any fixed  $\mu$ , this is a fixed shift of a sparse-LPN sample. By Assumption 2.2, encryptions of  $\mu_0$  and  $\mu_1$  are indistinguishable from pseudorandom vectors and hence from each other.

In Hybrid H3 the adversary's view is independent of the challenge bit. Therefore its advantage is zero in H3, and its advantage in H0 is bounded by the sum of the negligible hybrid gaps.  $\square$

### 7.3 What the proof does and does not cover

The proof covers passive IND-CPA privacy for the abstract construction. It does not cover:

- chosen-ciphertext attacks;
- maliciously generated public parameters;
- side channels or timing leakage;
- concrete attack costs for a particular parameter set;
- algebraic attacks exploiting unexpected structure in an optimized implementation;
- correctness failures that leak through repeated decryption attempts.

These issues must be addressed before any implementation is considered deployment-grade.

### 7.4 Relevant attack families

The most relevant attacks for concrete parameter selection include:

- BKW-style LPN attacks [8];
- information-set decoding and Prange/Stern-style decoding attacks [24, 25];
- low-noise LPN attacks when  $\eta$  or  $\eta_c$  is extremely small;
- sample-amplification attacks, because the evaluation key contains many LPN samples;
- distinguishing attacks against sparse public rows;
- attacks exploiting the chain of encrypted secrets  $t \rightarrow s \rightarrow r$ .

The construction deliberately avoids circular encryption of a secret under itself. The expansion key encrypts  $\tilde{t}$  under independent secret  $s$ , and the compaction key encrypts  $\tilde{s}$  under independent secret  $r$ .



## 8 Efficiency and parameter constraints

This section gives symbolic efficiency formulas. It does not certify concrete security. A real submission should include a parameter estimator and independent cryptanalysis.

### 8.1 Ciphertext and key sizes

Let  $N = n + 1$  and  $w_0 = (k + 1)(k + 2)$ .

Object	Dense size	Sparse/structured size
Compact input ciphertext	$N$ field elements	$O(k \log n + \log q)$ bits plus values
One GSW matrix ciphertext	$N^2$ field elements	$O(Nw)$ field/index pairs
Expansion key	$N$ GSW matrices	$O(N^2(k + 2))$ field/index pairs
Compaction key	$N$ CLPN ciphertexts	$O(Nm_c n_c)$ dense, seedable in practice
Compacted output	$R$ CLPN ciphertexts	independent of circuit size

The expansion key is large but amortized. It is published once per key and reused for many input ciphertexts. The final compact output does not grow with the evaluated circuit.

### 8.2 Evaluation costs

For row-sparse matrices with row supports  $w_1, w_2$ :

$$\text{EvalAdd} : O(N(w_1 + w_2)), \quad \text{EvalMul} : O(Nw_1w_2)$$

field operations, ignoring sorting/merging overhead. After depth  $D$ ,

$$w_D = w_0^{2^D}.$$

Thus SABLE-HE is a low-depth scheme. It is realistic to target degree-2, degree-4, or possibly degree-8 computations before considering further compression or bootstrapping.

### 8.3 Correctness budget

A conservative correctness target is

$$\varepsilon_f + \varepsilon_{\text{cmp}}(B) < \frac{1}{2}.$$

A stronger practical target is

$$\varepsilon_f + \varepsilon_{\text{cmp}}(B) \leq 0.1,$$

so that a moderate replica count  $R$  gives negligible final failure. Using the bound from Section 6,

$$\varepsilon_f \leq A(k + 2)\eta \prod_{r=0}^{D-1} (1 + w_0^{2^r}).$$

Therefore a simple design inequality is

$$\eta \leq \frac{\tau}{A(k + 2) \prod_{r=0}^{D-1} (1 + w_0^{2^r})}$$

for target sparse-LPN evaluation error  $\tau$ .

For the compactor, if the final row support is  $B$ , the effective q-ary error rate is

$$\eta_{c,B} = \frac{q-1}{q} \left( 1 - \left( 1 - \frac{q\eta_c}{q-1} \right)^B \right).$$

The code should be chosen so that decoding at this error rate fails with probability at most the desired  $\varepsilon_{\text{cmp}}(B)$ .

## 8.4 Illustrative non-secure toy parameters

The following parameters are only for debugging a Python prototype:

$$q = 127, \quad n = 64, \quad k = 3, \quad D = 1, \quad R = 9.$$

They are intentionally small and not secure. Their purpose is to validate algebraic correctness and serialization.

A research-grade estimator must choose  $n, k, q, \eta, n_c, m_c, \eta_c$  jointly. The estimator should output:

- estimated classical and quantum cost of sparse-LPN distinguishing;
- estimated cost of dense q-ary LPN/code distinguishing;
- estimated ISD cost for the relevant code parameters;
- total number of LPN samples exposed by input ciphertexts and evaluation keys;
- correctness failure estimates before and after replication.

## 8.5 Parameter-selection recipe

A practical research workflow is:

1. Fix the application class, especially multiplicative depth  $D$  and addition budget  $A$ .
2. Choose a small sparse row weight  $k$  that keeps multiplication feasible.
3. Compute  $w_0 = (k + 1)(k + 2)$  and the depth growth  $w_D = w_0^{2^D}$ .
4. Choose  $\eta$  small enough for correctness but not so small that low-noise LPN attacks become cheap.
5. Choose  $q$  to balance message space, q-ary error behavior, and implementation simplicity.
6. Choose `Code` and  $(n_c, m_c, \eta_c)$  so that compaction decoding succeeds at row support  $B \leq \min(N, w_D)$ .
7. Run attack estimators and increase dimensions until the desired security category is reached.

## 8.6 Benchmark baselines

The natural baselines are TFHE/FHEW for Boolean circuits, BFV/BGV for exact arithmetic, and CKKS for approximate arithmetic [13, 10, 15, 4, 11]. SABLE-HE should not be expected to dominate those mature lattice schemes broadly. It should be evaluated on the specific workloads where code-based assumption diversity and low-degree structure matter.

## 9 Validation plan

SABLE-HE should be validated in four parallel tracks: formal proofs, parameter estimation, prototype implementation, and benchmark comparison.

## 9.1 Formal proof track

The formal proof track should produce a machine-checkable or at least reviewer-checkable proof package for:

1. sparse-LPN zero-vector pseudorandomness;
2. GSW-style scalar encryption security as zero encryption plus diagonal shift;
3. expansion correctness;
4. row-support and coordinate-error invariants under addition and multiplication;
5. code/LPN compaction correctness;
6. hybrid IND-CPA security.

The proof should explicitly track sample counts. This matters because the evaluation key exposes  $O(N^2)$  sparse-LPN rows and the compaction key exposes  $O(Nm_c)$  dense  $q$ -ary LPN samples.

## 9.2 Parameter-estimation track

The estimator should be a separate, versioned artifact. It should compute at least:

- sparse-LPN distinguishing cost for  $(n, k, q, \eta, M)$ ;
- $q$ -ary LPN/code distinguishing cost for  $(n_c, m_c, q, \eta_c, M_c)$ ;
- information-set decoding estimates for the compaction code;
- low-noise attack costs as  $\eta$  and  $\eta_c$  decrease;
- correctness failure from the exact  $q$ -ary piling-up formula;
- final failure after  $R$  replicas.

No parameter set should be described as secure until this estimator and independent review agree.

## 9.3 Prototype track

The first Python library should implement the following objects:

- finite-field arithmetic modulo a prime  $q$ ;
- sparse vectors and row-sparse matrices;
- $q$ -ary symmetric noise sampling;
- compact sparse-LPN input encryption;
- GSW-style sparse-LPN matrices;
- expansion, addition, multiplication, and compaction;
- repetition-code decoding for initial tests;
- a pluggable code interface for BCH/LDPC/CRT-style compaction later.

Core unit tests should verify:

1.  $c \cdot \tilde{t} = \mu + e$  for compact input ciphertexts;
2.  $X\tilde{s} = \alpha\tilde{s} + e$  for GSW-style encryptions;
3.  $\text{Expand}(c)\tilde{s} = \mu\tilde{s} + e'$ ;
4. addition and multiplication decrypt to correct low-degree arithmetic values with expected failure rates;
5. compaction returns the same value as direct last-row decryption;
6. empirical failure rates match the formulas in Section 6.

## 9.4 Benchmark track

Benchmark on workloads that match the design:

- degree-2 private feature interaction:

$$f(x) = \sum_i a_i x_i + \sum_{i < j} b_{ij} x_i x_j;$$

- encrypted voting with polynomial validity checks;
- private counting with pairwise fraud/anomaly indicators;
- small Boolean circuits represented arithmetically over  $\mathbb{F}_q$ ;
- secure aggregation plus low-degree validation.

Metrics should include input ciphertext size, expansion-key size, compaction-key size, expansion time, multiplication time, compaction time, final ciphertext size, empirical failure probability, and estimated security bits.

## 9.5 Success criteria for a publishable paper

A strong paper based on SABLE-HE should provide:

1. a clean theorem statement under standard or carefully justified assumptions;
2. an implementation that is small enough for independent reproduction;
3. security estimates against known LPN and decoding attacks;
4. a comparison against sparse-LPN SHE with number-theoretic LHE compaction;
5. a comparison against TFHE/FHEW on small Boolean workloads;
6. evidence that the all-code compactor is not merely aesthetic but changes assumption structure meaningfully.

# 10 Limitations and future work

## 10.1 Limitations

The construction has several serious limitations.

**Depth growth.** Row support grows as  $w_D = w_0^{2^D}$ . This restricts SABLE-HE to low-depth computation unless a new compression or bootstrapping method is added.

**Low-noise tension.** Correctness prefers small  $\eta$  and  $\eta_c$ , but security against low-noise LPN attacks may prefer larger noise or larger dimensions. This tension is central and must be quantified.

**Large evaluation keys.** The expansion key contains  $N$  GSW matrices, each with  $N$  sparse rows. The compaction key also contains  $N$  code/LPN ciphertexts. Seeded randomness and structured sampling may reduce storage, but those optimizations need separate security analysis.

**Secret-key setting.** This draft gives a secret-key HE scheme with public evaluation keys. Public-key encryption is not included. A public-key variant may be possible using standard LPN public-key techniques, but it would need a new proof.

**Compactor quality.** A repetition-code compactor is simple but inefficient. A serious version should use stronger  $q$ -ary codes and possibly CRT decomposition, inspired by recent code-based secure aggregation work [7].

## 10.2 Future work

The most important next steps are:

1. design a stronger  $q$ -ary code compactor with efficient decoding and exact failure estimates;
2. build a sparse-LPN parameter estimator specialized to the exposed sample structure;
3. add seeded matrix generation to shrink evaluation keys;
4. study whether multiplication can be followed by a code-based refresh operation;
5. explore batching or packing across rows without introducing ring/lattice assumptions;
6. construct a public-key variant;
7. implement a Python prototype and then a C/Rust reference implementation.

## 10.3 Recommended claim language

For a research paper, use cautious claim language:

We propose a candidate all-code-based leveled somewhat homomorphic encryption scheme for bounded-depth arithmetic circuits. Under sparse-LPN and  $q$ -ary LPN/code assumptions, the construction achieves compact input encryption, GSW-style nonlinear evaluation, and code/LPN-based output compaction without lattice or number-theoretic assumptions.

Avoid stronger claims such as “practical full FHE from codes” or “secure against all quantum attacks.” The correct claim is conditional post-quantum plausibility under explicit code-based assumptions.

## A Implementation-oriented pseudocode

This appendix rewrites the construction in implementation-oriented form. Field operations are modulo prime  $q$ .

## A.1 Noise and sparse sampling

---

SampleNoise( $q, \eta$ )

1. With probability  $1 - \eta$ , return 0.
  2. Otherwise return a uniformly random element of  $\mathbb{F}_q^*$ .
- 

SampleSparse( $n, k, q$ )

1. Choose a uniformly random  $k$ -subset  $S \subset [n]$ .
  2. For each  $i \in S$ , choose  $a_i \leftarrow \mathbb{F}_q^*$ .
  3. Set all other coordinates to zero.
  4. Return sparse vector  $a$ .
- 

## A.2 Compact encryption

---

RegevEnc $_t(\mu)$

1.  $a \leftarrow \text{SampleSparse}(n, k, q)$ .
  2.  $e \leftarrow \text{SampleNoise}(q, \eta)$ .
  3.  $b \leftarrow a^\top t + \mu + e \pmod{q}$ .
  4. Return  $c = (a, b)$ .
- 

## A.3 GSW-style scalar encryption

---

GSWEnc $_s(\alpha)$

1. For  $j = 1, \dots, N$ :
    - (a) sample  $a_j \leftarrow \text{SampleSparse}(n, k, q)$ ;
    - (b) sample  $e_j \leftarrow \text{SampleNoise}(q, \eta)$ ;
    - (c) set row  $z_j = (a_j, a_j^\top s + e_j)$ .
  2. Let  $Z$  be the matrix with rows  $z_j$ .
  3. Return  $X = Z + \alpha I_N$ .
-

## A.4 Code-LPN encryption

---

CLPNEnc<sub>r</sub>( $x$ )

1. Sample  $A \leftarrow \mathbb{F}_q^{m_c \times n_c}$ .
  2. Sample  $e \leftarrow \chi_{q, \eta_c}^{m_c}$ .
  3. Set  $b = Ar + \text{Code}(x) + e$ .
  4. Return  $(A, b)$ .
- 
- 

CLPNDec<sub>r</sub>( $A, b$ )

1. Compute  $y = b - Ar$ .
  2. Return  $\text{Decode}(y)$ .
- 

## A.5 Key generation and evaluation

---

KeyGen

1. Sample  $t \leftarrow \mathbb{F}_q^n$ ,  $s \leftarrow \mathbb{F}_q^n$ ,  $r \leftarrow \mathbb{F}_q^{n_c}$ .
  2. Set  $\tilde{t} = (-t, 1)$  and  $\tilde{s} = (-s, 1)$ .
  3. For  $i = 1, \dots, N$ , set  $\text{EK}_i^{\text{exp}} = \text{GSWEnc}_s(\tilde{t}_i)$ .
  4. For  $i = 1, \dots, N$ , set  $\text{EK}_i^{\text{cmp}} = \text{CLPNEnc}_r(\tilde{s}_i)$ .
  5. Return  $\text{sk} = (t, s, r)$  and  $\text{ek} = (\text{EK}^{\text{exp}}, \text{EK}^{\text{cmp}})$ .
- 
- 

Expand( $c$ )

$$C \leftarrow \sum_{i=1}^N c_i \text{EK}_i^{\text{exp}}.$$

Return  $C$ .

---

---

Compact( $C$ )

1. Let  $\rho$  be the last row of  $C$ .
2. Return

$$\sum_{i=1}^N \rho_i \text{EK}_i^{\text{cmp}}.$$

---

## B Proof of the q-ary piling-up formula

We prove Lemma 2.1. Let  $E$  be a q-ary symmetric random variable with error probability  $\eta$ . Let  $\omega$  be a nontrivial additive character of  $\mathbb{F}_q$ . Its Fourier transform is

$$\mathbb{E}[\omega(aE)] = 1 - \eta + \sum_{x \in \mathbb{F}_q^*} \frac{\eta}{q-1} \omega(ax)$$

for  $a \neq 0$ . Since  $\sum_{x \in \mathbb{F}_q} \omega(ax) = 0$ , we have  $\sum_{x \in \mathbb{F}_q^*} \omega(ax) = -1$ . Therefore

$$\mathbb{E}[\omega(aE)] = 1 - \eta - \frac{\eta}{q-1} = 1 - \frac{q\eta}{q-1}.$$

Multiplication by a nonzero coefficient  $\alpha \in \mathbb{F}_q^*$  preserves the q-ary symmetric distribution. For independent variables  $E_1, \dots, E_B$ , the nontrivial Fourier coefficient of

$$S = \sum_{i=1}^B \alpha_i E_i$$

is

$$\left(1 - \frac{q\eta}{q-1}\right)^B.$$

A q-ary symmetric variable with error probability  $\eta_B$  has nontrivial Fourier coefficient

$$1 - \frac{q\eta_B}{q-1}.$$

Equating the coefficients gives

$$1 - \frac{q\eta_B}{q-1} = \left(1 - \frac{q\eta}{q-1}\right)^B,$$

and hence

$$\eta_B = \frac{q-1}{q} \left(1 - \left(1 - \frac{q\eta}{q-1}\right)^B\right).$$

This proves the formula.

## C Reviewer and cryptanalysis checklist

Before submitting a paper or releasing software, the following questions should be answered.

### C.1 Assumptions

1. Are the exact sparse-LPN parameters stated, including sample counts from all keys and ciphertexts?
2. Does the proof use only standard sparse-LPN/q-ary LPN assumptions, or does it introduce an additional hidden assumption?
3. Are low-noise regimes justified against known attacks?
4. Does any optimization introduce ring, module, cyclic, quasi-cyclic, or ideal structure that changes the assumption?



## C.2 Correctness

1. Are empirical failure rates consistent with the theoretical bounds?
2. Is the final row support measured rather than only upper-bounded?
3. Does the compactor decoder succeed at the observed effective q-ary error rate?
4. Is repetition/plurality independent across replicas?

## C.3 Security proof

1. Are  $t$ ,  $s$ , and  $r$  independent?
2. Does the expansion key encrypt  $\tilde{t}$  under  $s$  only, avoiding circularity?
3. Does the compaction key encrypt  $\tilde{s}$  under  $r$  only, avoiding circularity?
4. Are all hybrids efficient and simulatable?
5. Is the total exposed sample count within the assumed hardness regime?

## C.4 Implementation

1. Are field operations constant-time where needed?
2. Is noise sampling unbiased?
3. Are sparse rows stored canonically to avoid duplicate-index bugs?
4. Are decoding failures handled without leaking side information?
5. Are parameter files machine-readable and bound to test vectors?

# D Attack-Screening Addendum

This appendix records the first validation step after the initial construction: a conservative attack-screening layer for parameter rejection. The estimates here are not security proofs and do not certify concrete SABLE parameters. Their purpose is to expose weak settings before implementation or benchmarking. In particular, any final parameter set must still be analyzed by specialists in sparse-LPN, q-ary LPN, information-set decoding, and large-sample attacks.

## D.1 Screened attack surfaces

The validation repository models two public noisy-linear-equation surfaces. First, the expansion key exposes approximately

$$m_{\text{exp}} = (n + 1)^2$$

sparse-LPN-like rows related to the GSW expansion layer. Second, the compaction key exposes

$$m_{\text{cmp}} = (n + 1)m_c$$

q-ary LPN-like samples for the code/LPN compactor. These are only proxy surfaces; the exact joint distribution of the GSW rows and encrypted secret-key coordinates must still be handled in the formal proof and in cryptanalysis.

The repository now implements three first-pass attack screens.

**No-clean-subset screen.** A random set of  $n$  noisy equations is clean with probability roughly  $(1 - \eta)^n$ . The corresponding infinite-sample work proxy is

$$\log_2 W_{\text{clean}} \approx -n \log_2(1 - \eta) + \omega \log_2 n,$$

where the last term represents dense linear algebra. This screen is deliberately simple and is especially important when correctness pushes  $\eta$  to be very small.

**Prange/information-set screen.** Viewing the public samples as a noisy random linear code-word, a Prange-style information-set decoder succeeds when it chooses an error-free information set. With  $m$  samples and expected error weight  $t = \eta m$ , the proxy work is

$$\log_2 W_{\text{Prange}} \approx \log_2 \binom{m}{n} - \log_2 \binom{m-t}{n} + \omega \log_2 n.$$

This is the most basic ISD screen, following the information-set viewpoint of Prange and later code-decoding attacks [24, 25, 5].

**BKW-style bias screen.** For  $q$ -ary symmetric noise, the estimator uses the character-bias proxy

$$\beta_q = \left| 1 - \frac{q\eta}{q-1} \right|.$$

After approximately  $2^a$  combined samples, the proxy bias is  $\beta_q^{2^a}$  and a final distinguisher wants on the order of  $\beta_q^{-2^{a+1}}$  samples. The code minimizes a simple table-building and distinguishing proxy over the number of BKW levels. This follows the role of BKW-family algorithms in LPN cryptanalysis [8, 19, 9]. It is not a specialized sparse- $q$ -ary-LPN estimator.

## D.2 Early finding: correctness/security tension

The first estimator run rejects the earlier `prototype_medium` preset. The reason is not a software bug: the preset used very low noise for correctness. For  $q = 65537$ ,  $n = 512$ ,  $k = 3$ , and  $\eta = 2^{-20}$ , the expansion-key surface has expected total errors below one. The no-clean-subset and Prange-style proxies therefore fall far below a 128-bit target. The compaction-key surface has the same problem.

The repository records the following conservative feasibility grid. Here  $\eta$  is set to the largest value allowed by the conservative requirement that the evaluated GSW error be at most 0.10 for one additive term. The column  $n_{\min}$  is the smallest dimension making the no-clean-subset proxy reach 128 bits. The final column is the sparse expansion-key entry count  $(n+1)^2(k+1)$ .

$k$	depth	$\eta$ ceiling	$n_{\min}$	expansion entries
1	1	$6.67 \cdot 10^{-3}$	9,172	$1.68 \cdot 10^8$
2	1	$2.50 \cdot 10^{-3}$	23,388	$1.64 \cdot 10^9$
3	1	$1.18 \cdot 10^{-3}$	47,905	$9.18 \cdot 10^9$
1	2	$3.92 \cdot 10^{-4}$	135,802	$3.69 \cdot 10^{10}$
2	2	$3.05 \cdot 10^{-5}$	1,510,083	$6.84 \cdot 10^{12}$
3	2	$4.58 \cdot 10^{-6}$	8,895,213	$3.16 \cdot 10^{14}$

The table suggests that the current construction is most plausible for depth-one arithmetic, i.e., degree-two computations, unless a new refresh, coding, or compaction technique improves the correctness/security tradeoff. Depth two is not ruled out mathematically, but the naive expansion-key size becomes very large under this conservative screen.

### D.3 Design consequence

The next research problem is therefore sharper than the original construction problem:

Can SABLE reduce its dependence on extremely low LPN noise while preserving low-depth homomorphic correctness?

Promising directions are: using a stronger q-ary code rather than repetition in the compactor; reducing the public sample exposure through seeded or punctured evaluation keys; adding a lightweight code-based refresh for the GSW layer; deriving sharper circuit-aware error bounds instead of worst-case balanced-product bounds; and testing whether the sparse-LPN assumption used by Corrigan-Gibbs, Henzinger, Kalai, and Vaikuntanathan [12] admits concrete parameter regimes compatible with code/LPN compaction.

## E C2 Block-Dictionary Compaction

This appendix records the C2 refinement implemented in the validation repository. The original compaction layer publishes one linearly homomorphic code-LPN encryption of each coordinate of the extended GSW secret  $\tilde{s} \in \mathbb{F}_q^N$ . Compacting a final row  $r \in \mathbb{F}_q^N$  then evaluates

$$\sum_{i=1}^N r_i \cdot \text{Enc}(\tilde{s}_i),$$

so the number of accumulated compaction-noise terms is the row support  $B = \|r\|_0$ . Since q-ary symmetric noise piles up as

$$\eta_B = \frac{q-1}{q} \left( 1 - \left( 1 - \frac{q\eta_c}{q-1} \right)^B \right),$$

large  $B$  forces a very small compaction noise rate  $\eta_c$ .

The C2 block-dictionary compactor trades public-key size for fewer accumulated compaction-noise terms. Partition  $\tilde{s}$  into blocks

$$\tilde{s} = (\tilde{s}^{(1)}, \dots, \tilde{s}^{(L)}), \quad \tilde{s}^{(j)} \in \mathbb{F}_q^{\ell_j},$$

where  $\ell_j \leq \ell$  and  $L = \lceil N/\ell \rceil$ . For every block  $j$  and every nonzero coefficient vector  $u \in \mathbb{F}_q^{\ell_j}$ , publish

$$\text{EK}_{j,u}^{\text{C2}} = \text{CLPNEnc}_r(\langle u, \tilde{s}^{(j)} \rangle).$$

Given a final GSW row  $r$ , write it in matching blocks  $r = (r^{(1)}, \dots, r^{(L)})$  and output

$$\text{Compact}_{\text{C2}}(r) = \sum_{j:r^{(j)} \neq 0} \text{EK}_{j,r^{(j)}}^{\text{C2}}.$$

By linear homomorphism of the CLPN layer, the plaintext inside the compacted ciphertext is

$$\sum_{j:r^{(j)} \neq 0} \langle r^{(j)}, \tilde{s}^{(j)} \rangle = r \cdot \tilde{s}.$$

Thus C2 preserves the compaction semantics.

The number of compaction-noise terms becomes

$$t_{\text{C2}}(r) = \#\{j : r^{(j)} \neq 0\} \leq \min\{\|r\|_0, \lceil N/\ell \rceil\}.$$

For dense rows this improves the compaction-noise term by about a factor of  $\ell$ . For sparse rows there may be no improvement. The public-key cost is

$$\sum_{j=1}^L (q^{\ell_j} - 1)$$

CLPN ciphertexts, so this construction is only plausible for small  $q$  and small  $\ell$ , or for CRT-laned designs over small prime fields. This is consistent with the code-based secure aggregation literature, where CRT-style decompositions are used to reduce communication in LPN-based homomorphic aggregation [7].

The security proof remains a hybrid argument from CLPN semantic security: the evaluator sees additional CLPN encryptions of linear forms of  $\tilde{s}$ , but these messages are hidden under the compaction secret. The important change is quantitative rather than syntactic: C2 increases the number of public CLPN samples, so the attack surface must be re-estimated. In the validation repository the C2 estimator therefore reports both the reduced compaction-noise count and the enlarged public sample surface.

The executable validation reports that, for the toy setting  $q = 7$ ,  $N = 13$ , and  $\ell = 2$ , the block dictionary contains 294 ciphertexts and correctly evaluates clean multiplication and  $xy + z$  examples. For the larger design-screening setting  $q = 7$ ,  $N = 513$ , and  $\ell = 3$ , the dictionary contains 58,482 CLPN ciphertexts. At multiplicative depth two, the worst-case compaction terms drop from 513 to 171, but the public sample surface becomes very large. Therefore C2 should be treated as a research refinement, not as a certified parameter set.

## F C3 Seeded Block-Dictionary Storage

The C2 block-dictionary compactor reduces the number of accumulated CLPN noise terms by publishing encryptions of all nonzero block inner products  $\langle u, \tilde{s}^{(j)} \rangle$ . Its direct implementation, however, stores a full CLPN matrix  $A$  and vector  $b$  for every dictionary entry. This appendix records a storage refinement implemented in the validation repository.

**Seeded CLPN ciphertexts.** For each dictionary entry, sample a public seed  $\sigma$  and define  $A_\sigma \in \mathbb{F}_q^{m_c \times n_c}$  by a deterministic expander. Instead of storing  $(A_\sigma, b)$ , the public key stores

$$(\sigma, b), \quad b = A_\sigma r + x\mathbf{1} + e,$$

where  $x = \langle u, \tilde{s}^{(j)} \rangle$  is the dictionary plaintext and  $r$  is the CLPN compaction secret. Linear homomorphic addition concatenates seeded matrix terms and adds the corresponding  $b$  vectors. To decrypt an aggregate ciphertext, the holder of  $r$  regenerates the required rows of each  $A_\sigma$  and plurality-decodes the residual.

**Correctness.** Let a compacted C3 ciphertext be the linear combination

$$(b, \{(\alpha_t, \sigma_t)\}_{t=1}^T), \quad b = \sum_{t=1}^T \alpha_t b_t.$$

Then

$$b - \sum_{t=1}^T \alpha_t A_{\sigma_t} r = \left( \sum_{t=1}^T \alpha_t x_t \right) \mathbf{1} + \sum_{t=1}^T \alpha_t e_t.$$

Thus the plaintext and noise law are exactly those of dense C2. Seeded storage changes materialization cost but not the correctness invariant.

**Security accounting.** The public matrices are derivable from the seeds, so C3 exposes the same  $q$ -ary LPN sample surface as dense C2: if the dictionary contains  $M_{\text{dict}}$  entries, then the public compaction surface contains  $M_{\text{dict}}m_c$  LPN rows. Therefore seeding must not be claimed as a security improvement. It is a key-size and memory-traffic optimization only. The validation repository adds a dedicated  $q$ -ary/sparse-LPN surface estimator that reports the public sample count, sample-to-dimension ratio, no-clean-subset and Prange-style screens, and a  $q$ -ary block-BKW scan following the classical BKW line of work [8, 19, 9, 14].

**Storage comparison.** Ignoring small serialization overheads, dense C2 stores approximately

$$M_{\text{dict}}m_c(n_c + 1)$$

field elements. Seeded C3 stores approximately

$$M_{\text{dict}}m_c + M_{\text{dict}} \cdot \frac{\lambda_{\text{seed}}}{\log_2 q}$$

field-element equivalents, where  $\lambda_{\text{seed}}$  is the public seed length in bits. The saving is therefore close to a factor of  $n_c + 1$  when  $m_c$  is large relative to the seed length. The public attack surface remains unchanged.

**Implemented status.** The validation package includes seeded CLPN encryption, seeded block-dictionary key generation, end-to-end clean multiplication tests, and a seeded C3 estimator. The implementation is still a research prototype and uses SHA-256-derived rows for reproducibility; a production design would require a specified, reviewed, constant-time PRG or XOF interface and independent cryptanalysis.

## G C2/C3 Public-Sample Surface Screen

C2 and C3 reduce final compaction noise by replacing coordinate-wise compaction with a block dictionary. The security cost is a much larger public  $q$ -ary LPN surface. This appendix records the screening model used by the validation repository. It is not a certified cryptanalytic estimator; it is a conservative design screen intended to expose surfaces that must be analyzed before any concrete security claim.

**Dictionary row count.** For block widths  $\ell_j$ , the C2 dictionary publishes

$$E = \sum_j (q^{\ell_j} - 1)$$

CLPN ciphertexts. Each ciphertext contains  $m_c$  noisy rows under the same compaction secret  $r \in \mathbb{F}_q^{n_c}$ , so the direct dictionary surface contains  $Em_c$  public rows. Seeded C3 changes only the representation of the public matrices: it stores a seed for each matrix and the corresponding  $b$  vector. Since the adversary can regenerate the public matrices from the seeds, the LPN sample count is unchanged.

**Within-entry row differences.** A CLPN dictionary entry has rows

$$b_i = \langle A_i, r \rangle + x + e_i \pmod{q},$$

where  $x$  is the encrypted block inner product. Subtracting two rows from the same entry cancels  $x$ :

$$b_i - b_j = \langle A_i - A_j, r \rangle + (e_i - e_j) \pmod{q}.$$

Thus each dictionary entry yields up to  $\binom{m_c}{2}$  message-eliminating  $q$ -ary LPN-like rows. The effective noise rate is estimated by the  $q$ -ary piling-up formula with two terms,

$$\eta^{(2)} = \frac{q-1}{q} \left( 1 - \left( 1 - \frac{q\eta_c}{q-1} \right)^2 \right).$$

The validation code records the raw within-entry surface as

$$E \binom{m_c}{2}.$$

**Cross-entry dictionary differences.** Inside one block, two dictionary entries for coefficient tuples  $u$  and  $v$  encrypt

$$\langle u, s_{\text{block}} \rangle, \quad \langle v, s_{\text{block}} \rangle.$$

A row difference between these entries gives

$$b_{u,i} - b_{v,j} = \langle A_{u,i} - A_{v,j}, r \rangle + \langle u - v, s_{\text{block}} \rangle + e_{u,i} - e_{v,j}.$$

This is a noisy linear equation in the joint secret  $(r, s_{\text{block}})$  of dimension at most  $n_c + \max_j \ell_j$ . The raw ordered surface is conservatively screened using

$$\sum_j \binom{q^{\ell_j} - 1}{2} m_c^2$$

rows, again with two-term  $q$ -ary piling-up noise. These rows are not all independent, so the repository reports them as a red-flag screening surface rather than a proof of insecurity.

**Implication for C3.** Seeded C3 is still valuable because it avoids materializing  $Em_c n_c$  public field elements for dense matrices and keeps only  $Em_c$  field elements for the  $b$  vectors plus seed metadata. However, seeded storage does not reduce the within-entry or cross-entry surfaces above. Therefore the next construction-level problem is not storage but relation control: a publishable C4 variant should avoid a fully closed block dictionary, restrict tuple sets, randomize dictionary availability, or replace the compactor with a code/LPN LHE layer whose public entries do not create so many plaintext-cancelling row differences.

**Validation artifact.** The repository implements this screen in the module

`src/sable/c2_attack_surface.py`.

The generated toy and design reports in the documentation folder show that the current presets remain research-only. This is expected and useful: the screen identifies the next mathematical bottleneck rather than certifying parameters.

## H C4 Projective Sparse Additive-Basis Compaction

This appendix records the C4 compaction refinement implemented in the validation repository. The purpose of C4 is to reduce the public surface of the C2/C3 full block dictionary while preserving the small compaction-noise count that made block compaction attractive.

## H.1 Projective dictionary idea

Let a final GSW last-row block be

$$\alpha = (\alpha_1, \dots, \alpha_b) \in \mathbb{F}_q^b,$$

and let the corresponding block of the extended GSW secret be

$$s_B = (s_1, \dots, s_b) \in \mathbb{F}_q^b.$$

A full C2 dictionary stores an encryption of

$$\langle \alpha, s_B \rangle$$

for every nonzero  $\alpha \in \mathbb{F}_q^b$ . Hence one block requires

$$q^b - 1$$

public CLPN entries.

C4 observes that the linearly homomorphic compactor can scale ciphertexts. Therefore it is unnecessary to publish separate entries for all scalar multiples of the same coefficient vector. Let

$$\mathbb{P}^{b-1}(\mathbb{F}_q)$$

denote the set of one-dimensional subspaces of  $\mathbb{F}_q^b$ . For each projective line, choose one representative  $u$  whose first nonzero coordinate is normalized to one. The number of representatives is

$$|\mathbb{P}^{b-1}(\mathbb{F}_q)| = \frac{q^b - 1}{q - 1}.$$

For every representative  $u$ , the public C4 key stores

$$\text{Enc}_{\text{CLPN}}(\langle u, s_B \rangle).$$

For any nonzero  $\alpha \in \mathbb{F}_q^b$ , there exists a unique scalar  $\lambda \in \mathbb{F}_q^*$  and a unique normalized representative  $u$  such that

$$\alpha = \lambda u.$$

The evaluator then computes

$$\lambda \cdot \text{Enc}_{\text{CLPN}}(\langle u, s_B \rangle),$$

which is a CLPN encryption of

$$\lambda \langle u, s_B \rangle = \langle \alpha, s_B \rangle.$$

## H.2 C4 correctness theorem

**Theorem H.1** (Projective C4 block correctness). *Let  $q$  be prime and let  $B \subseteq \{1, \dots, N\}$  be a block of width  $b$ . Suppose the C4 public key contains CLPN encryptions of  $\langle u, s_B \rangle$  for every projective representative  $u \in \mathbb{P}^{b-1}(\mathbb{F}_q)$ . Then for every coefficient block  $\alpha \in \mathbb{F}_q^b$ , C4 homomorphically produces a CLPN encryption of  $\langle \alpha, s_B \rangle$  using zero CLPN terms if  $\alpha = 0$ , and one CLPN term otherwise.*

*Proof.* If  $\alpha = 0$ , the inner product is zero and the compactor contributes the zero CLPN ciphertext. If  $\alpha \neq 0$ , let  $i$  be the first index for which  $\alpha_i \neq 0$ . Set

$$\lambda = \alpha_i \quad \text{and} \quad u = \lambda^{-1} \alpha.$$

Then the first nonzero coordinate of  $u$  is one, so  $u$  is exactly the projective representative stored by the public key. By CLPN linear homomorphism,

$$\lambda \cdot \text{Enc}_{\text{CLPN}}(\langle u, s_B \rangle) \text{ag1}$$

is a CLPN encryption of

$$\lambda \langle u, s_B \rangle = \langle \lambda u, s_B \rangle = \langle \alpha, s_B \rangle.$$

This proves the claim. □

### H.3 Public-surface comparison

For a block of width  $b$ , the C4 projective dictionary has

$$M_{C4}(b, q) = \frac{q^b - 1}{q - 1}$$

entries, while C2/C3 full dictionaries have

$$M_{C2}(b, q) = q^b - 1.$$

Thus

$$\frac{M_{C4}}{M_{C2}} = \frac{1}{q - 1}.$$

The compaction-noise term count remains one per active block, as in C2/C3. Therefore C4 is not a storage-only transformation: it reduces both materialized key size and public CLPN sample count relative to full block dictionaries.

The cost is that C4 still publishes a structured family of related encryptions, one for every projective line in each block. This does not by itself prove insecurity, but it is a public relation surface that must be analyzed by sparse-LPN and q-ary-LPN attack screens.

### H.4 Validation status

The validation repository implements:

- projective representatives and sparse decomposition;
- random additive-basis coverage screens;
- C4 key generation, compaction, and decryption;
- C4 public-surface estimates against v1, C2, and C3;
- end-to-end toy arithmetic-circuit tests.

The implementation should be interpreted as a research validation prototype. The C4 construction improves the compaction-key public surface over C2/C3, but it does not remove the need for full LPN attack-cost estimation, large-sample analysis, and independent cryptanalysis.

## I C5 Arithmetic Operation Suite and External Baseline Plan

The core construction is not limited to multiplication. Once a compact input ciphertext is expanded into the GSW-style representation, the encrypted message space is the prime field  $\mathbb{F}_q$ . If  $C\tilde{s} = \mu\tilde{s} + e$ , then linear public operations are immediate and nonlinear multiplication is matrix multiplication.

**Native field operations.** For expanded ciphertexts  $C, C'$  and public  $a, b \in \mathbb{F}_q$ :

$$\begin{aligned} \text{Enc}(0) &:= 0, \\ \text{Enc}(b) &:= bI, \\ \text{Enc}(\mu) + \text{Enc}(\nu) &:= C + C', \\ \text{Enc}(\mu) - \text{Enc}(\nu) &:= C - C', \\ a\text{Enc}(\mu) &:= aC, \\ \text{Enc}(\mu)\text{Enc}(\nu) &:= CC'. \end{aligned}$$

The identity construction is exact because  $(bI)\tilde{s} = b\tilde{s}$ . These operations support addition, subtraction, negation, public-scalar multiplication, public-constant addition, multiplication, squaring, public powers, and arbitrary low-degree public polynomials by composition.



**Division and inversion.** Encrypted division is not a native operation. For a nonzero field element  $x \in \mathbb{F}_q^*$ , one may evaluate

$$x^{-1} = x^{q-2}$$

by Fermat’s little theorem. This consumes multiplicative depth  $O(\log q)$  by square-and-multiply and is undefined at  $x = 0$ . Therefore the prototype exposes nonzero inversion only as a validation tool for small  $q$ , not as a recommended primitive.

**Boolean gates.** Bits are encoded as  $0, 1 \in \mathbb{F}_q$ . The validation code tests the polynomial gates

$$\begin{aligned}\text{AND}(x, y) &= xy, \\ \text{OR}(x, y) &= x + y - xy, \\ \text{NOT}(x) &= 1 - x, \\ \text{XOR}(x, y) &= x + y - 2xy.\end{aligned}$$

Since the construction uses odd prime fields, XOR is not linear in this encoding. This is a design tradeoff of the current  $q$ -ary sparse-LPN formulation.

**Tested operations.** Version C5 adds tests for constants, zero, one, addition, subtraction, negation, public-scalar multiplication, public-constant addition/subtraction, multiplication, oriented multiplication, square, powers, nonzero inverse, nonzero division, affine polynomials, and Boolean AND/OR/XOR/NOT/NAND/NOR/XNOR.

**Baseline families.** The external comparison plan separates exact arithmetic, approximate arithmetic, and Boolean/integer circuits. OpenFHE documents support for BFV and BGV for integer arithmetic, CKKS for approximate real arithmetic, and FHEW/TFHE-style Boolean/arbitrary-function bootstrapping [23]. Microsoft SEAL supports BFV, BGV, and CKKS and is a natural exact/approximate arithmetic baseline [20]. TFHE-rs targets Boolean, short-integer, and integer circuits and documents common integer arithmetic operations [26]. The TFHE construction of Chillotti, Gama, Georgieva, and Izabachene remains the relevant Boolean-gate baseline [10].

**Interpretation of C5 measurements.** The included timings are pure-Python prototype timings on toy parameters. They validate operation semantics and expose relative costs inside the prototype. They must not be interpreted as speed claims against optimized OpenFHE, SEAL, or TFHE-rs implementations. The repository therefore includes an operation-support matrix and an optional external benchmark harness, but it does not fabricate wall-clock numbers for libraries that are not installed in the validation environment.

## J C5 Baseline and Public-Surface Diagnostics

C5 separates measured prototype timing from baseline comparison. The SABLE repository is a pure-Python validation prototype, whereas mature libraries such as OpenFHE, SEAL, Concrete, and TFHE-family implementations use optimized C, C++, Rust, compiler passes, and architecture-specific kernels. Therefore the current package reports operation-count and surface-area proxies rather than claiming wall-clock superiority.

The fair baseline mapping is workload-dependent. TFHE/FHEW-style schemes are the closest comparison for Boolean gates and programmable function evaluation [13, 10]. BFV and BGV are the closest comparison for exact modular or integer arithmetic [15, 4]. CKKS is the closest comparison for approximate real or complex arithmetic, but it is not the natural baseline for exact finite-field outputs [11].

The C5 public-surface screen also records that C4 projective compaction removes two-term projective duplicates but necessarily introduces many three-term linear relations once the block width is at least two. For public representatives  $u, v, w$  of projective lines, many choices satisfy

$$au + bv - cw = 0$$

for non-zero field elements  $a, b, c$ . This relation surface does not by itself break the CLPN layer, but it does give an adversary many public noisy linear combinations to examine. Consequently, the C5 conclusion is that C4 is smaller than C2/C3, but a security-grade claim requires a dedicated large-sample q-ary-LPN analysis of the projective public surface.

## References

- [1] C. Aguilar Melchor, V. Dyzeryn, and P. Gaborit. Somewhat homomorphic encryption based on random codes. *Designs, Codes and Cryptography*, 93(6):1645–1669, 2025. <https://doi.org/10.1007/s10623-024-01555-y>.
- [2] M. R. Albrecht et al. Security guidelines for implementing homomorphic encryption. IACR Cryptology ePrint Archive, Report 2024/463, 2024. <https://eprint.iacr.org/2024/463>.
- [3] M. Alekhnovich. More on average case vs approximation complexity. In *44th Annual IEEE Symposium on Foundations of Computer Science*, pages 298–307, 2003. <https://doi.org/10.1109/SFCS.2003.1238204>.
- [4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13:1–13:36, 2014. <https://doi.org/10.1145/2633600>.
- [5] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. <https://doi.org/10.1109/TIT.1978.1055873>.
- [6] S. S. Bhoi, A. Arakala, A. B. Corman, and A. Rao. Post-quantum homomorphic encryption: a case for code-based alternatives. *Cryptography*, 9(2):31, 2025. <https://doi.org/10.3390/cryptography9020031>.
- [7] S. Bitzer, M. Egger, M. Liu, and A. Wachter-Zeh. Post-quantum secure aggregation via code-based homomorphic encryption. arXiv:2601.13031, 2026. <https://arxiv.org/abs/2601.13031>.
- [8] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003. <https://doi.org/10.1145/792538.792543>.
- [9] S. Bogos, F. Tramer, and S. Vaudenay. On solving LPN using BKW and variants. *Cryptography and Communications*, 8(3):331–369, 2016. <https://doi.org/10.1007/s12095-015-0149-2>.
- [10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. <https://doi.org/10.1007/s00145-019-09319-x>.
- [11] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, LNCS 10624, pages 409–437, 2017. [https://doi.org/10.1007/978-3-319-70694-8\\_15](https://doi.org/10.1007/978-3-319-70694-8_15).

- [12] H. Corrigan-Gibbs, A. Henzinger, Y. T. Kalai, and V. Vaikuntanathan. Somewhat homomorphic encryption from linear homomorphism and sparse LPN. IACR Cryptology ePrint Archive, Report 2024/1760, 2024. <https://eprint.iacr.org/2024/1760>.
- [13] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology – EUROCRYPT 2015*, LNCS 9056, pages 617–640, 2015. [https://doi.org/10.1007/978-3-662-46800-5\\_24](https://doi.org/10.1007/978-3-662-46800-5_24).
- [14] A. Esser, R. Kuebler, A. May, and C. Zveydinger. LPN decoded. IACR Cryptology ePrint Archive, Report 2017/078, 2017. <https://eprint.iacr.org/2017/078>.
- [15] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, Report 2012/144, 2012. <https://eprint.iacr.org/2012/144>.
- [16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009. <https://doi.org/10.1145/1536414.1536440>.
- [17] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013*, LNCS 8042, pages 75–92, 2013. [https://doi.org/10.1007/978-3-642-40041-4\\_5](https://doi.org/10.1007/978-3-642-40041-4_5).
- [18] HomomorphicEncryption.org. Homomorphic encryption standard. Community white paper, 2018. <https://homomorphicencryption.org/standard/>.
- [19] E. Levieil and P.-A. Fouque. An improved LPN algorithm. In *Security and Cryptography for Networks*, LNCS 4116, pages 348–359, 2006. [https://doi.org/10.1007/11832072\\_24](https://doi.org/10.1007/11832072_24).
- [20] Microsoft Research. Microsoft SEAL: homomorphic encryption library. Project documentation and source repository, 2026. <https://github.com/microsoft/SEAL>.
- [21] National Institute of Standards and Technology. FIPS 203: module-lattice-based key-encapsulation mechanism standard. 2024. <https://doi.org/10.6028/NIST.FIPS.203>.
- [22] National Institute of Standards and Technology. FIPS 204: module-lattice-based digital signature standard. 2024. <https://doi.org/10.6028/NIST.FIPS.204>.
- [23] OpenFHE Project. OpenFHE documentation. Project documentation, 2026. <https://openfhe-development.readthedocs.io/>.
- [24] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.
- [25] J. Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, LNCS 388, pages 106–113, 1989. <https://doi.org/10.1007/BFb0019850>.
- [26] Zama. TFHE-rs documentation. Project documentation, 2026. <https://docs.zama.org/tfhe-rs>.