

Math Companion to Soundcalc

December 8, 2025

1 Preliminaries

1.1 Fields

Fields of size q are denoted as \mathbb{F}_q or simply \mathbb{F} .

1.2 Regime-specific bounds

1.2.1 Proximity parameter

`unique_decoding.py/get_proximity_parameter()`

In UDR

$$\xi_U(\rho) = (1 - \rho)/2 \quad (1)$$

In JBR we compute the ξ differently.

`johnson_bound.py/get_proximity_parameter()`

$$\xi_J(\rho, N, q) = \begin{cases} 1 - \sqrt{\rho} - \frac{1}{N} & \text{if } q > 2^{150} \\ 1 - \sqrt{\rho} - \frac{\rho}{20} & \text{otherwise} \end{cases} \quad (2)$$

1.2.2 List sizes

`unique_decoding.py/get_max_list_size()`

$$\ell = 1$$

`johnson_bound.py/get_max_list_size()`

We compute

$$\ell(\rho, N) = \frac{1}{2(1 - \rho - \xi)\sqrt{\rho}}.$$

2 WHIR-based VM security level calculation

2.1 Notation and parameters

- Number of iterations: $M = \text{num_iterations}$
- Iteration index: $i \in \{0, \dots, M - 1\}$
- Folding round index: $s \in \{0, \dots, k - 1\}$
- Field size: $|\mathbb{F}| = q$
- Constraint degree: $d = \text{constraint_degree}$
- Folding factor: $k = \text{folding_factor}$
- Log-degree in iteration i : $m_i = \text{log_degrees}[i]$, and we set $m_i = m_0 - k \cdot i$
- Log-inverted rate in iteration i : $\mu_i = \text{log_inv_rates}[i]$, and we set $\mu_i = \log_2 \frac{1}{\rho} + (k - 1) \cdot i$
- Iteration-round-specific RS codes:

$$\mathcal{C}_{i,s} = (\mathcal{C}_{i,s}[\rho], \mathcal{C}_{i,s}[N])_{i,s} = (2^{-\mu_i}, 2^{m_i-s})$$

- Number of OOD samples in iteration i : $w_i = \text{num_ood_samples}[i]$
- Number of queries in iteration i : $t_i = \text{num_queries}[i]$
- Grinding bits:

$$g_{\text{batch}}, \quad \{g_{i,s}^{\text{fold}}\}_{i \in [M], s \in [k]}, \quad \{g_i^{\text{ood}}\}_{i \in M}, \quad \{g_i^{\text{qry}}\}_{i \in M}.$$

2.2 Bits of Security in UDR and JBR

In order to compute the bits of security in UDR and JBR regimes, we compute the following error terms using the parameters from section 2.1.

Computed in `get_security_levels_for_regime()`.

For any error term:

$$\lambda(\epsilon) = \lfloor -\log_2 \epsilon \rfloor.$$

Overall security:

$$\lambda_{\text{total}} = \min\{\lambda_{\text{batch}}, \lambda_{i,s}^{\text{fold}}, \lambda_i^{\text{out}}, \lambda_i^{\text{shift}}, \lambda_i^{\text{fin}}\}.$$

2.3 Batching Error ϵ_{batch}

This computes the batching error with parameters from section 2.1. `get_batching_error()`

In the Johnson Bound Regime (JBR) we proceed as follows:

- Set code parameters

$$(\rho, N) = (2^{-\mu_0}, 2^{m_0})$$

- Compute proximity parameter ξ as in (2):

- Compute m as in `get_m()`:

$$m = 0.5 + \max \left(\left\lceil \frac{\sqrt{\rho}}{1 - \sqrt{\rho} - \xi} \right\rceil, 3 \right)$$

- Compute linear error¹:

$$\epsilon_{\text{batch,lin,J}}(\rho, N, q) = \frac{N(2m^5 + 3m\xi\rho) + 3m\rho}{3\rho^{1.5}q}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow,J}}(\rho, N, q, B) = \epsilon_{\text{batch,lin,J}}(\rho, N, q) \cdot (B - 1)$$

¹Code refers to Theorem 4.2 from BCHKS25

- Compute base error:

$$\varepsilon_{\text{batch},JBR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},J} & (\text{power batching}), \\ \epsilon_{\text{batch,lin},J} & (\text{linear batching}). \end{cases}$$

Whereas in the UDR we do as follows:

- Compute linear error:

$$\epsilon_{\text{batch,lin},U}(\rho, N, q) = \frac{N}{q\rho}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow},U}(\rho, N, q, B) = \epsilon_{\text{batch,lin},J}(\rho, N, q) \cdot B$$

- Compute base error:

$$\varepsilon_{\text{batch},UDR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},U} & (\text{power batching}), \\ \epsilon_{\text{batch,lin},U} & (\text{linear batching}). \end{cases}$$

In both regimes we do as follows after grinding:

$$\epsilon_{\text{batch}} = \varepsilon_{\text{batch}}^{\text{base}} \cdot 2^{-g_{\text{batch}}}.$$

2.4 Folding Error

`epsilon_fold()`

For iteration $i \in [M]$ and folding round $s \in [k]$ in JBR we do:

- Get list size

$$\ell_{i,s} = \ell(\mathcal{C}_{i,s}[\rho])$$

- Base error (two terms):

$$\varepsilon_{i,s}^{\text{fold,base,J}} = d \cdot \frac{\ell_{i,s}}{q} + \epsilon_{\text{batch,pow},J}(\mathcal{C}_{i,s+1}[\rho], \mathcal{C}_{i,s+1}[N], q, B = 2).$$

And in UDR the base error is

$$\varepsilon_{i,s}^{\text{fold,base,U}} = \frac{d}{q} + \epsilon_{\text{batch,pow},U}(\mathcal{C}_{i,s+1}[\rho], \mathcal{C}_{i,s+1}[N], q, B = 2).$$

After grinding:

$$\epsilon_{i,s}^{\text{fold}} = \varepsilon_{i,s}^{\text{fold,base}} \cdot 2^{-g_{i,s}^{\text{fold}}}.$$

2.5 OOD error

`epsilon_out()`

For iteration $i \in \{1, 2, \dots, M - 1\}$

- Base error in JBR:

$$\varepsilon_i^{\text{out,base,J}} = \ell_{i,0}^2 \left(\frac{2^{m_i}}{2q} \right)^{w_{i-1}}.$$

- Base error in UDR:

$$\varepsilon_i^{\text{out,base,U}} = \left(\frac{2^{m_i}}{2q} \right)^{w_{i-1}}.$$

- After grinding:

$$\epsilon_i^{\text{out}} = \varepsilon_i^{\text{out,base}} \cdot 2^{-g_i^{\text{out}}}.$$

2.6 Shift Error in JBR

`epsilon_shift()`

For iteration $i \in \{1, 2, \dots, M - 1\}$ in JBR

- Get δ for the iteration using (??):

$$\delta_i = \min_{s \in [k]} \delta_J(\mathcal{C}_{i-1,s}[\rho])$$

- Base error (two terms):

$$\varepsilon_i^{\text{shift,base}} = (1 - \delta_i)^{t_{i-1}} + \ell_{i,0} \cdot \frac{t_{i-1} + 1}{F}.$$

Same procedure in UDR:

- Get δ for the iteration using (??):

$$\delta_i = \min_{s \in [k]} \delta_U(\mathcal{C}_{i-1,s}[\rho])$$

- Base error :

$$\varepsilon_i^{\text{shift,base}} = (1 - \delta_i)^{t_{i-1}} + \frac{t_{i-1} + 1}{F}.$$

After grinding:

$$\epsilon_i^{\text{shift}} = \varepsilon_i^{\text{shift,base}} \cdot 2^{-g_{i-1}^{\text{qry}}}.$$

2.7 Final Round Error

`epsilon_final()`

JBR:

- Get δ for the iteration using (??):

$$\delta_{M-1} = \min_{s \in [k]} \delta_J(\mathcal{C}_{M-1,s}[\rho])$$

UDR:

- Get δ for the iteration using (??):

$$\delta_{M-1} = \min_{s \in [k]} \delta_U(\mathcal{C}_{M-1,s}[\rho])$$

Then the base error is :

$$\varepsilon^{\text{fin,base}} = (1 - \delta_{M-1})^{t_{M-1}}.$$

After grinding:

$$\epsilon^{\text{fin}} = (1 - \delta_{M-1})^{t_{M-1}} \cdot 2^{-g_{M-1}^{\text{qry}}}.$$

3 Proof Size Calculations

This section summarizes the proof-size formula computed in `get_proof_size_bits()`.

3.1 Initial Function Size

$$S_{\text{if}} = b_{\text{hash}}.$$

3.2 Initial Sumcheck Size

Thus the folding proof size per round is

$$S_{\text{sumcheck}} = kd \log_2 |\mathbb{F}|.$$

3.3 OOD Proof Size for Iteration i

$$S_{\text{ood},i} = b_{\text{hash}} + w_i \cdot \log_2 |\mathbb{F}| + kd \log_2 |\mathbb{F}|$$

$$S_{\text{ood},M} = +2^{m_M} \log_2 |\mathbb{F}|$$

3.4 Query Proof Size for Iteration i

Thus

$$S_{\text{qry},i} = t_i \cdot MP(2^{m_i+\mu_i-k}, 2^k, \log_2 q, b_{\text{hash}}).$$

3.5 Total Proof Size

Collecting all terms and summing over all M iterations:

$$S_{\text{total}} = S_{\text{if}} + S_{\text{sumcheck}} + \sum_{i \in [M]} (S_{\text{ood},i} + S_{\text{qry},i})$$

Expanding,

$$S_{\text{total}} = b_{\text{hash}} + kd \log_2 |\mathbb{F}| + 2^{m_M} \log_2 |\mathbb{F}| + \sum_{i \in [M-1]} (b_{\text{hash}} + w_i \cdot \log_2 |\mathbb{F}| + kd \log_2 |\mathbb{F}|) + \sum_{i \in [M]} (t_i \cdot MP(2^{m_i+\mu_i-k}, 2^k, \log_2 q, b_{\text{hash}}))$$

4 FRI-based VM security level calculation

This section calculates the security level for a FRI-based VM in section 4.3.

4.1 FRI parameters

Global parameters used in the FRI analysis:

- m_J — Johnson parameter.
- r_{FRI} — number of FRI rounds.
- Folding factors $\widehat{\text{folds}} = [k_0, k_1, \dots, k_{r_{\text{FRI}}-1}]$;
- t — number of queries.
- θ .
- δ .
- ρ — rate of the Reed-Solomon code.
- N — trace length.
- L — list size.
- $b_{\text{grind},Q}$ — grinding parameter for the query phase.
- n — witness size.
- b_{hash} — number of bits in the hash function output.
- b_{proof} — proof size in bits.
- B — batch size.

Notation specific to the Johnson bound:

-

4.2 Fixed constants

We fix the following constants for the soundness calculator:

- $m_J = 16$. Set in

```
fri.py/get_johnson_parameter_m()
```

4.3 Security level for a FRI-based VM

The security level is calculated in

```
zkvms/fri_based_vm.py/get_security_levels()
```

It is done separately for two different regimes: UDR and JBR — using the same procedure:

1. Calculate the FRI round-by-round soundness errors $\epsilon_{FRI,U}, \epsilon_{FRI,J}$ using the formula from section 4.3.1 and section 4.3.2.
2. Obtain optimal δ_U, δ_J parameters using the formula from section 1.2.1 and section 1.2.1.
3. Obtain the list sizes L_U, L_J for the respective δ_U, δ_J .
4. Obtain the DEEP-ALI soundness errors $\epsilon_{D-A,U}, \epsilon_{D-A,J}$ using the formulas from Section ??.
5. Compute the total soundness errors as

$$\epsilon_U = \min(\epsilon_{FRI,U}, \epsilon_{D-A,U}), \quad \epsilon_J = \min(\epsilon_{FRI,J}, \epsilon_{D-A,J}).$$

Then the full security level in bits is the maximum of the two regimes:

$$\text{Security level} = \max(-\log_2 \epsilon_U, -\log_2 \epsilon_J).$$

4.3.1 RBR soundness in UDR

```
fri_based_vm.py/get_security_levels_for_regime()
```

4.3.2 RBR soundness in JBR

```
fri_based_vm.py/get_security_levels_for_regime()
```

4.3.3 DEEP-ALI errors

```
fri_based_vm.py/get_DEEP_ALI_errors()
```

4.4 Batching Error ϵ_{batch}

This computes the batching error with parameters from section 2.1. `get_batching_error()`

In the Johnson Bound Regime (JBR) we proceed as follows:

- Find η as a function of ρ :

$$\eta(\rho) = \frac{\sqrt{\rho}}{32}$$

- Compute linear error:

$$\epsilon_{\text{batch,lin},J} = \frac{N}{q(2 \min(\eta(\rho), \sqrt{\rho}/20))^5}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow},J}(B) = \epsilon_{\text{batch,lin},J} \cdot (B - 1)$$

- Compute base error:

$$\varepsilon_{\text{batch},JBR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},J} & (\text{power batching}), \\ \epsilon_{\text{batch,lin},J} & (\text{linear batching}). \end{cases}$$

Whereas in the UDR we do as follows:

- Compute linear error:

$$\epsilon_{\text{batch,lin},U} = \frac{N}{q\rho}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow},U}(B) = \epsilon_{\text{batch,lin},J} \cdot B$$

- Compute base error:

$$\varepsilon_{\text{batch},UDR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},U} & (\text{power batching}), \\ \epsilon_{\text{batch,lin},U} & (\text{linear batching}). \end{cases}$$

4.5 Commit phase error

For round i we set

$$N_i = \frac{N}{k_i^i}$$

4.6 Soundness formula

This is calculated in

`fri.py/get_FRI_query_phase_error()`

Query phase error:

$$\epsilon_{\text{query}} = (1 - \theta)^t \cdot 2^{-b_{\text{grind},Q}} \quad (3)$$

The query phase error without grinding is computed as per [?]²

4.7 Proof size

This calculation is performed in

`fri.py/get_FRI_proof_size_bits()`

. The FRI proof contains two parts: Merkle roots, and one "openings" per query, where an "opening" is a Merkle path for each folding layer. For each layer we count the size that this layer contributes, which includes the root and all Merkle paths.

Initial round: one root and one path per query. We assume that for the initial functions, there is only one Merkle root, and each leaf i for that root contains symbols i for all initial functions.

Folding rounds: we assume that "siblings" for the following layers are grouped together in one leaf. This is natural as they always need to be opened together.

²Code refers to (7) and Th2 of [Hab22]

The proof size is calculated as follows:

$$\begin{aligned}
b_{\text{proof}} = & b_{\text{hash}} + t \cdot MP\left(\underbrace{\frac{n}{\widehat{\text{folds}}[0]}}_{\text{Initial round}}, B, \log_2 |\mathbb{F}|, b_{\text{hash}}\right) + \\
& + \underbrace{\sum_{1 \leq i \leq r_{FRI}-2} \left(b_{\text{hash}} + t \cdot MP\left(\frac{n}{\prod_{1 \leq j \leq i} \widehat{\text{folds}}[j]}, B, \log_2 |\mathbb{F}|, b_{\text{hash}}\right) \right)}_{\text{Folding rounds but last}} + \\
& + \underbrace{\left(b_{\text{hash}} + t \cdot MP\left(\frac{n}{\widehat{\text{folds}}[r_{FRI}-1] \prod_{1 \leq j \leq r_{FRI}-1} \widehat{\text{folds}}[j]}, B, \log_2 |\mathbb{F}|, b_{\text{hash}}\right) \right)}_{\text{Last folding round}}
\end{aligned} \tag{4}$$

where $MP(n, s, q, b)$ is the Merkle path size calculated as

$$MP(n, s, q, b) = \underbrace{sq}_{\text{leaf size}} + \underbrace{sq}_{\text{sibling}} + \underbrace{[\log_2 n] \cdot b}_{\text{co-path}} \tag{5}$$