Notes on Basefold (Part I): Foldable Linear Codes

- Yu Guo yu.guo@secbit.io
- Jade Xie jade@secbit.io

Basefold can be regarded as an extension of FRI, thereby supporting Proximity Proofs and Evaluation Arguments for Multi-linear Polynomials. Compared to Libra-PCS, Hyrax-PCS, and Virgo-PCS, Basefold does not rely on the MLE Quotients Equation to prove the value of an MLE at the point $\mathbf{u}$:

$$\tilde{f}(X_0, X_1, X_2, \ldots, X_{n-1}) = \sum_{i_0=0}^{n-1}(X_i - u_i)\cdot q_i(X_0, X_1, X_2, \ldots, X_{i-1}) \tag{1}$$

Instead, Basefold leverages the Sumcheck protocol to reduce the value of $\tilde{f}(\mathbf{X})$ at a pre-specified point to its value at a random point $\vec{\alpha}$. The latter can then utilize an FRI-like approach, where the verifier provides a challenge vector $\vec{\alpha}$ to recursively fold $\tilde{f}(\mathbf{X})$. This approach allows the prover to simultaneously demonstrate an upper bound on the degree of $\tilde{f}(\mathbf{X})$ (Proof of Proximity) and, based on the characteristics of MLE, prove the value of $\tilde{f}(\vec{\alpha})$. By combining the Sumcheck protocol with an FRI-style folding protocol, we elegantly obtain an Evaluation Argument for an MLE.

Another significant insight of Basefold is how to apply an FRI-style Proof of Proximity over arbitrary finite fields. It is known that the FRI protocol fully utilizes the structure of Algebraic FFTs over finite fields and performs interactive folding, which allows the codeword length to be exponentially reduced while maintaining the minimum relative Hamming distance bound, thereby enabling the verifier to easily verify a folded short codeword. However, for general finite fields where constructing FFTs is challenging, the FRI protocol cannot be directly applied.

Basefold introduces the concept of Random Foldable Codes. This is a framework for encoding using a recursive approach, where recursive encoding can be seen as the inverse process of recursive folding. In the Commit phase of the Basefold protocol, the prover first encodes each segment of the message using a base encoding scheme G_0, then encodes these codewords pairwise (a process similar to the butterfly operation in FFT computations), and finally obtains a single codeword. In the Commit-phase stage, the prover and verifier interactively perform half-folding on the committed single codeword and then commit to the half-folded codeword. It can be proven that the Relative Hamming Distance of this half-folded codeword remains above a clear lower bound. The parties then continue folding until the length is reduced to that of G_0-encoded length. In this way, both parties obtain a series of committed codewords. Note that since the encoding process is performed recursively, the codeword possesses folding capabilities similar to those of RS Code encoding. The advantage of this approach is evident: using Recursive Folding FRI protocols results in significantly smaller proof sizes compared to protocols like Tensor Codes. Additionally, Basefold's technique can be applied to any finite field ($|\mathbb{F}| > 2^{10}$), including extension fields $\mathbb{F}_{p^k}$. Naturally, similar to the FRI protocol, the commitment computation in the Basefold protocol requires $O(N\log(N))$ time complexity, while the prover's computational effort in the subsequent proof process remains at $O(N)$. If a Merkle Tree is used as the commitment scheme for the codeword, the proof size complexity is $O(\log^2(N))$.

In this article, we first explore the concept of Foldable Linear Codes.

What are Foldable Linear Codes

Suppose there is a linear code $C_0 : \mathbb{F}_p^{k_0} \to \mathbb{F}_p^{n_0}$ based on the finite field $\mathbb{F}_p$, where the message length is k_0, and the codeword length is n_0. Assume that the encoded length is amplified by a factor of R compared to the message length, i.e., $1/R$ is the traditional code rate. According to the definition of linear codes, there must exist an encoding matrix $G_0 : \mathbb{F}_p^{k_0 \times n_0}$ such that

$$\mathbf{m}G_0 = \mathbf{c} \tag{2}$$

Then, we can construct a new encoding matrix G_1 based on the encoding matrix G_0:

$$G_1 = \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_0 \cdot T_0' \end{bmatrix} \tag{3}$$

where T_0 and T_0' are two diagonal matrices with diagonal elements set to the parameters of the encoding scheme, which we will explain later.

If we consider G_1 as the encoding matrix of another linear code C_1, then the parameters of C_1 are $[R, 2k_0, 2n_0]$. That is, compared to C_0, the code rate of C_1 remains unchanged, but the message and codeword lengths are both doubled.

For example, suppose $\mathbf{m} = (m_0, m_1, m_2, m_3)$, and the base encoding scheme C_0 is a simple Repetition Code with $k_0 = 2$ and $n_0 = 4$. Thus, the message length of $\mathbf{m}$ exactly meets the requirements of G_1, i.e., $k_1 = 2 \cdot k_0 = 4$ and the encoded length is $n_1 = 2 \cdot n_0 = 8$.

The encoding matrix G_0 of the base code C_0 is defined as:

$$G_0 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{4}$$

Assume the diagonal elements of the system parameter T_0 are (t_0, t_1, t_2, t_3), and those of T_0' are (t_0', t_1', t_2', t_3') . We construct G_1 using the formula above:

$$G_1 = \left[\begin{array}{cccc|cccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline t_0 & 0 & t_2 & 0 & t_0' & 0 & t_2' & 0 \\ 0 & t_1 & 0 & t_3 & 0 & t_1' & 0 & t_3' \end{array}\right] \tag{5}$$

Substituting $\mathbf{m}$ directly, we obtain:

$$\mathbf{m}G_1 = [m_0 \quad m_1 \quad m_2 \quad m_3] \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ t_0 & 0 & t_2 & 0 & t_0' & 0 & t_2' & 0 \\ 0 & t_1 & 0 & t_3 & 0 & t_1' & 0 & t_3' \end{bmatrix} \tag{6}$$

We list the encoded codeword vector separately:

$$\begin{matrix} m_0 + t_0 m_2 & m_1 + t_1 m_3 & m_0 + t_2 m_2 & m_1 + t_3 m_3 \\ m_0 + t_0' m_2 & m_1 + t_1' m_3 & m_0 + t_2' m_2 & m_1 + t_3' m_3 \end{matrix} \tag{7}$$

Directly performing matrix operations, the structure of this encoding is not very clear. Let us approach it differently: the message $\mathbf{m}$ can be split into two equal-length parts, the left $\mathbf{m}_l = (m_0, m_1)$ and the right $\mathbf{m}_r = (m_2, m_3)$,

$$\mathbf{m}G_1 = (\mathbf{m}_l \parallel \mathbf{m}_r)G_1 = [1 \quad 1] \begin{bmatrix} \mathbf{m}_l & 0 \\ 0 & \mathbf{m}_r \end{bmatrix} \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_0 \cdot T_0' \end{bmatrix} = [1 \quad 1] \begin{bmatrix} \mathbf{m}_l G_0 & \mathbf{m}_l G_0 \\ \mathbf{m}_r G_0 T_0 & \mathbf{m}_r G_0 T_0' \end{bmatrix} \tag{8}$$

For the 2×2 matrix on the right side of the equation, we first compute the top-left and top-right submatrices, which are identical submatrices obtained by encoding $\mathbf{m}_l$ with G_0:

$$\mathbf{m}_l G_0 = [m_0 \quad m_1] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [m_0 \quad m_1 \quad m_0 \quad m_1] \tag{9}$$

And the bottom-left submatrix is:

$$\begin{aligned} \mathbf{m}_r(G_0 T_0) = (\mathbf{m}_r G_0) T_0 &= [m_2 \quad m_3] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} T_0 \\ &= [m_2 \quad m_3 \quad m_2 \quad m_3] \begin{bmatrix} t_0 & 0 & 0 & 0 \\ 0 & t_1 & 0 & 0 \\ 0 & 0 & t_2 & 0 \\ 0 & 0 & 0 & t_3 \end{bmatrix} \\ &= (t_0 m_2, t_1 m_3, t_2 m_2, t_3 m_3) \end{aligned} \tag{10}$$

The result for $\mathbf{m}_r(G_0 T_0')$ is almost the same, with t_i replaced by t_i':

$$\mathbf{m}_r(G_0 T_0') = (t_0' m_2, t_1' m_3, t_2' m_2, t_3' m_3) \tag{11}$$

It is easy to verify that we obtain the same result for $\mathbf{m}$ encoded with G_1.

We can simplify this computation process with the following equation:

$$\mathbf{m} G_1 = \mathbf{m}_l G_0 + (t_0, t_1, t_2, t_3) \circ \mathbf{m}_r G_0 \parallel \mathbf{m}_l G_0 + (t_0', t_1', t_2', t_3') \circ \mathbf{m}_r G_0 \tag{12}$$

If we consider k_0 as a system parameter (constant), the encoding computation in the above equation has a complexity of $O(n_0)$, including two G_0 encodings and two component-wise additions of length n_0. Additionally, this equation resembles the butterfly operation in (Multiplicative) FFT algorithms:

$$a' = a + t \cdot b, \quad b' = a - t \cdot b \tag{13}$$

In fact, as illustrated later, the encoding process of Foldable Codes is a generalized extension of RS-Code.

We can further recursively construct $G_2, G_3, \ldots, G_d$, eventually obtaining a linear code $C_d : \mathbb{F}_p^{k_0} \to \mathbb{F}_p^n$, where the message length is $k = k_0 \cdot 2^d$, and the encoding length is $n = c \cdot k_0 \cdot 2^d$. The code rate is $\rho = \frac{1}{R}$, where k_0 is the message length of the base encoding, n_0 is the base encoding length, and the choice of the base encoding is highly flexible.

We use the symbol G_d to denote the encoding matrix (or generator matrix) of C_d, $G_d \in \mathbb{F}_p^{k \times n}$. For $i \in \{1, 2, \ldots, d\}$, we have the following recursive relation:

$$G_i = \begin{bmatrix} G_{i-1} & G_{i-1} \\ G_{i-1} \cdot T_{i-1} & G_{i-1} \cdot T_{i-1}' \end{bmatrix} \tag{14}$$

Here, T_{i-1} and T_{i-1}' are two diagonal matrices,

$$T_{i-1} = \begin{bmatrix} t_{i-1,0} & 0 & \cdots & 0 \\ 0 & t_{i-1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & t_{i-1,n_i-1} \end{bmatrix}, \quad T_{i-1}' = \begin{bmatrix} t_{i-1,0}' & 0 & \cdots & 0 \\ 0 & t_{i-1,1}' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & t_{i-1,n_i-1}' \end{bmatrix}, \tag{15}$$

and their diagonal elements at the same positions are distinct:

$$\forall j \in [0, n_i - 1], t_{i,j} \neq t_{i,j}' \tag{16}$$

Assuming we can compute G_d using the above recursive equation, then for a message $\mathbf{m}_d$ of length k, the encoding process can be directly calculated as follows:

$$\mathbf{w}_d = \mathbf{m}_d G_d \tag{17}$$

However, encoding directly using the generator matrix G_d is relatively inefficient, with a computational complexity of $O(k \cdot n)$. Instead, based on the above derivation, if we do not directly use the G_d matrix but instead employ a recursive approach to encode $\mathbf{m}_d$, the basic idea is to split m into two parts m_l and m_r, then encode m_l and m_r separately using G_{d-1}, and finally concatenate the encoded C_l and C_r.

$$\mathbf{w}_d = \Big(m_l G_{d-1} + \mathsf{diag}(T_d) \circ m_r G_{d-1}\Big) \parallel \Big(m_l G_{d-1} + \mathsf{diag}(T'_d) \circ m_r G_{d-1}\Big) \tag{18}$$

In this way, the encoding process using G_d is transformed into two encoding processes using G_{d-1}. We can continue to recursively compute $m_l G_{d-1}$ and $m_r G_{d-1}$ until the split message length meets k_0, after which we simply use the generator matrix G_0 for the base encoding. The encoding time complexity at this stage is $O(k_0 \cdot n_0)$. With a total of d recursive rounds, the overall computational effort is reduced to $O(n \log(n))$.

RS code (foldable)

The RS code used in the FRI protocol satisfies the aforementioned conditions, making RS code foldable.

First, let us examine the generator matrix in the RS code:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{k-1} & \omega^{2(k-1)} & \cdots & \omega^{(n-1)(k-1)} \end{bmatrix} \tag{19}$$

where ω is an n-th root of unity, satisfying $\omega^n = 1$, with n being the codeword length and k the message length. We can observe that the above matrix is a Vandermonde matrix. We will now explain how the generator matrix of the RS code satisfies the definition of Foldable Codes. For demonstration purposes, assume $k_0 = 1$, $n_0 = 1$, $d = 3$, $k_3 = 8$, $n_3 = 8$, and $\omega^8 = 1$,

$$G^{(RS)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix} \tag{20}$$

First, we need to reorder the rows of G^{RS} according to the Reversed Bit Order (RBO) sequence. The reason for performing RBO reordering is related to the structure of Multiplicative FFTs, as explained in another article. The RBO sequence refers to reversing the binary representation of the indices and using these reversed binary numbers as the new indices. The RBO sequence is $(0, 4, 2, 6, 1, 5, 3, 7)$. By reordering the rows of the above Vandermonde matrix according to the RBO sequence, we obtain the following matrix, denoted as G_2:

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix} \tag{21}$$

Observing the reordered matrix, we can see that this matrix can be decomposed into submatrix operations involving G_1 and T_1, T_1':

$$G_2 = \begin{bmatrix} G_1 & G_1 \\ G_1T_1 & G_1T_1' \end{bmatrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ \hline 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{array}\right] \tag{22}$$

where G_1 is:

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \tag{23}$$

We can verify that the bottom-left submatrix can be expressed as G_1T_1:

$$G_1T_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \omega & & \\ & & \omega^2 & \\ & & & \omega^3 \end{bmatrix} = \begin{bmatrix} 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^5 & \omega^2 & \omega^7 \\ 1 & \omega^3 & \omega^6 & \omega^1 \\ 1 & \omega^7 & \omega^6 & \omega^5 \end{bmatrix} \tag{24}$$

Here, the matrix T_1 is indeed a diagonal matrix satisfying the requirement that its diagonal elements are distinct. The bottom-right submatrix can be decomposed as follows:

$$\begin{bmatrix} \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^4 & \omega & \omega^6 & \omega^3 \\ \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} \omega^4 & & & \\ & \omega^5 & & \\ & & \omega^6 & \\ & & & \omega^7 \end{bmatrix} = G_1T_1' \tag{25}$$

where T_1' is exactly $(-1)T_1$, ensuring that the elements of T_1' are distinct from those of T_1:

$$T_2' = \begin{bmatrix} \omega^4 & & & \\ & \omega^5 & & \\ & & \omega^6 & \\ & & & \omega^7 \end{bmatrix} = \begin{bmatrix} -1 & & & \\ & -\omega & & \\ & & -\omega^2 & \\ & & & -\omega^3 \end{bmatrix} = -T_2 \tag{26}$$

Next, we can continue to recursively decompose G_1:

$$G_1 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ \hline 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{array}\right] = \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_1 \cdot T_0' \end{bmatrix} \tag{27}$$

Here, G_0, T_0, and T_0' are:

$$G_0 = \begin{bmatrix} 1 & 1 \\ 1 & \omega^4 \end{bmatrix}, \quad T_0 = \begin{bmatrix} 1 & \\ & \omega^2 \end{bmatrix}, \quad T_0' = \begin{bmatrix} \omega^4 & \\ & \omega^6 \end{bmatrix} = -T_0 \tag{28}$$

By recursively decomposing, we reach the base encoding G_0, which remains an RS-Code, denoted as $RS[k_0 = 1, n_0 = 1]$. Thus, we have proven that RS-Code is a foldable code. However, it is evident that RS-Code imposes requirements on $\mathbb{F}_p$, needing it to contain a multiplicative subgroup of order 2^s, where s is a positive integer, and ensuring that s is large enough to accommodate a codeword length of $k_0 \cdot 2^{n_d}$.

References

- [ZCF23] Hadas Zeilberger, Binyi Chen, and Ben Fisch. "BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes." In *Annual International Cryptology Conference*, pp. 138-169. Cham: Springer Nature Switzerland, 2024.
- Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. "Doubly-efficient zkSNARKs without trusted setup." In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 926-943. IEEE, 2018.
- Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct zero-knowledge proofs with optimal prover computation." In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pp. 733-764. Springer International Publishing, 2019.
- Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Transparent polynomial delegation and its applications to zero knowledge proof." In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 859-876. IEEE, 2020.
- Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. "Signatures of correct computation." In *Theory of Cryptography Conference*, pp. 222-242. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.