Notes on Basefold (Part II): IOPP

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

Proof of Proximity

Below, we present a proof of implementing IOPP using Foldable codes.

Suppose there is an MLE polynomial $\tilde{f}(\mathbf{X})$ represented as follows:

$$\tilde{f}(X_0, X_1, \ldots, X_{d-1}) = f_0 + f_1 X_0 + f_2 X_1 + f_3 X_0 X_1 + \cdots + f_{2^d-1} X_{d-1} \tag{1}$$

Since $\tilde{f}(X)$ is a multivariate polynomial, there are $d-1$ unknowns, making the length of its coefficient vector 2^d. Note that we choose the Lexicographic Order as the sorting method for the polynomial.

We encode the coefficient vector $\mathbf{f}$ of $\tilde{f}(\mathbf{X})$ to obtain the codeword $c_{\mathbf{f}} = \mathsf{Enc}(\mathbf{f})$, which has a length of n_d. Then, we use a Hash-based Merkle Tree to generate the commitment:

$$\mathbf{cm}(\mathbf{f}) = \mathsf{Merklize}(\mathsf{Enc}(f_0, f_1, f_2, \ldots, f_{2^d-1})) \tag{2}$$

Similar to the FRI protocol, the Basefold-IOPP protocol is used to prove that a commitment $\pi_d = \mathbf{cm}(\mathbf{f})$ is with high probability "close" to a vector encoded by C_d. Therefore, this protocol is called a Proof of Proximity. This protocol is one of the core protocols for constructing the Evaluation Argument.

Proof of Proximity leverages a remarkable property of linear codes: the "Proximity Gap." Specifically, if two vectors π, π' are far from the legitimate codeword space, then their random linear combination $\pi + \alpha \cdot \pi'$ will either have a very low probability of becoming legitimate or will remain far from the legitimate codeword space:

$$\begin{cases} \Delta(\pi_i, C_i) = 0 & \text{(with negligible probability)} \\ \Delta(\pi_i, C_i) \leq \Delta(\pi_{i+1}, C_{i+1}) & \text{(with non-negligible probability)} \end{cases} \tag{3}$$

This indicates that the folding process of the codeword does not disrupt the distance between the vector and the legitimate codeword space. By folding the vector sufficiently, the Verifier can use a very short code to verify whether the final folded vector is a legitimate codeword, thereby determining whether the original vector is a legitimate codeword.

> 📖 **Notes on Proximity Gap** Proof of Proximity utilizes a remarkable property of linear codes: the "Proximity Gap." Specifically, for two vectors π, π', folding them with a random scalar $\alpha \in \mathbb{F}$ yields a set $A = \{\pi + \alpha \cdot \pi' : \alpha \in \mathbb{F}\}$. Different α correspond to different elements in set A. The "Proximity Gap" theorem states that the elements in this set are either all close to the legitimate codeword space C_i or only a negligible fraction of the elements are close to C_i, while the majority are at a distance of δ from C_i. In probabilistic terms:

> $$\Pr_{a \in A}[\Delta(a, C_i) \leq \delta] = \begin{cases} \epsilon & \text{(small enough)} \\ 1 \end{cases} \tag{4}$$
>
> Thus, the Verifier can confidently use a random scalar α for folding, because even if only one of the two vectors π, π' provided by a cheating Prover is at a distance δ from C_i, the probability that the folded result is close to C_i is only ϵ, which is very small. In other words, a cheating Prover would need to be as lucky as winning the lottery to evade detection by the Verifier's scrutiny. Therefore, if the Prover initially selects a π_d that is far from the legitimate codeword space, the Verifier selects a series of random scalars to iteratively fold it until obtaining π_0. During this process, there is a high probability that π_0 does not become close to the legitimate codeword space, allowing the Verifier to detect cheating.

> The "Proximity Gap" theorem provides a significant advantage to the Verifier: instead of verifying all elements in the set $A = \{\pi + \alpha \cdot \pi' : \alpha \in \mathbb{F}\}$ to check their proximity to the legitimate codeword space, the Verifier only needs to randomly select one point for verification. This greatly reduces the Verifier's computational load.

The Proof of Proximity protocol consists of two phases: the Commit-phase and the Query-phase. The former involves the subprotocol that performs multiple folding processes of the codeword and generates commitments (or oracles) for each folded codeword. The latter, the Query-phase, involves the Verifier performing random sampling to verify the legitimacy of each folding step.

Commit-phase

First, we explain the Commit-phase. The Prover performs multiple foldings of the encoded π_d (with length n_d), obtaining codewords of lengths $n_{d-1}, \ldots, n_0$, denoted as $(\pi_{d-1}, \pi_{d-2}, \ldots, \pi_0)$, and then sends them to the Verifier.

Remember that this is an interactive protocol with a total of d rounds of interaction. In each round (assume the i-th round, $0 \leq i < d$), the Prover folds π_{i+1} based on the random scalar α_i sent by the Verifier to obtain a new codeword, denoted as π_i. After d rounds, the Prover obtains a codeword of length n_0, denoted as π_0. The Prover then commits to each $(\pi_d, \ldots, \pi_0)$ and sends $\mathsf{cm}(\pi_d), \ldots, \mathsf{cm}(\pi_0)$ as the output of $\mathsf{IOPP.Commit}$.

Next, we analyze the technical details of a single folding π_i. Suppose $\pi_i \in C_i$ is a legitimate codeword (i.e., satisfying $\pi_i = \mathbf{m}G_i$), with length n_i:

$$\pi_i = (c_0, c_1, c_2, \ldots, c_{n_i-1}) \tag{5}$$

We split this vector into two parts and stack them:

$$\begin{pmatrix} c_0, & c_1, & \ldots, & c_{n_{i-1}-1} \\ c_{n_{i-1}}, & c_{n_{i-1}+1}, & \ldots, & c_{n_i-1} \end{pmatrix} \tag{6}$$

At this point, the Verifier needs to provide a random scalar $\alpha^{(i)}$. We perform a random linear combination of the two rows, or in other words, fold them:

$$\pi_{i-1} = \big(\mathsf{fold}_{\alpha_i}(c_0, c_{n_{i-1}}), \mathsf{fold}_{\alpha_i}(c_1, c_{n_{i-1}+1}), \ldots, \mathsf{fold}_{\alpha_i}(c_{n_{i-1}}, c_{n_i-1})\big) \tag{7}$$

The above is the folded vector π_{i-1}. Assuming the Prover is honest, the folded vector should be a legitimate C_{i-1} codeword. The function fold_{α_i} in the above equation is defined as follows:

$$\mathsf{fold}_\alpha(c_j, c_{n_{i-1}+j}) = \frac{t_j \cdot c_{n_{i-1}+j} - t'_j \cdot c_j}{t_j + t'_j} + \alpha \cdot \frac{(c_j - c_{n_{i-1}+j})}{t_j - t'_j} \tag{8}$$

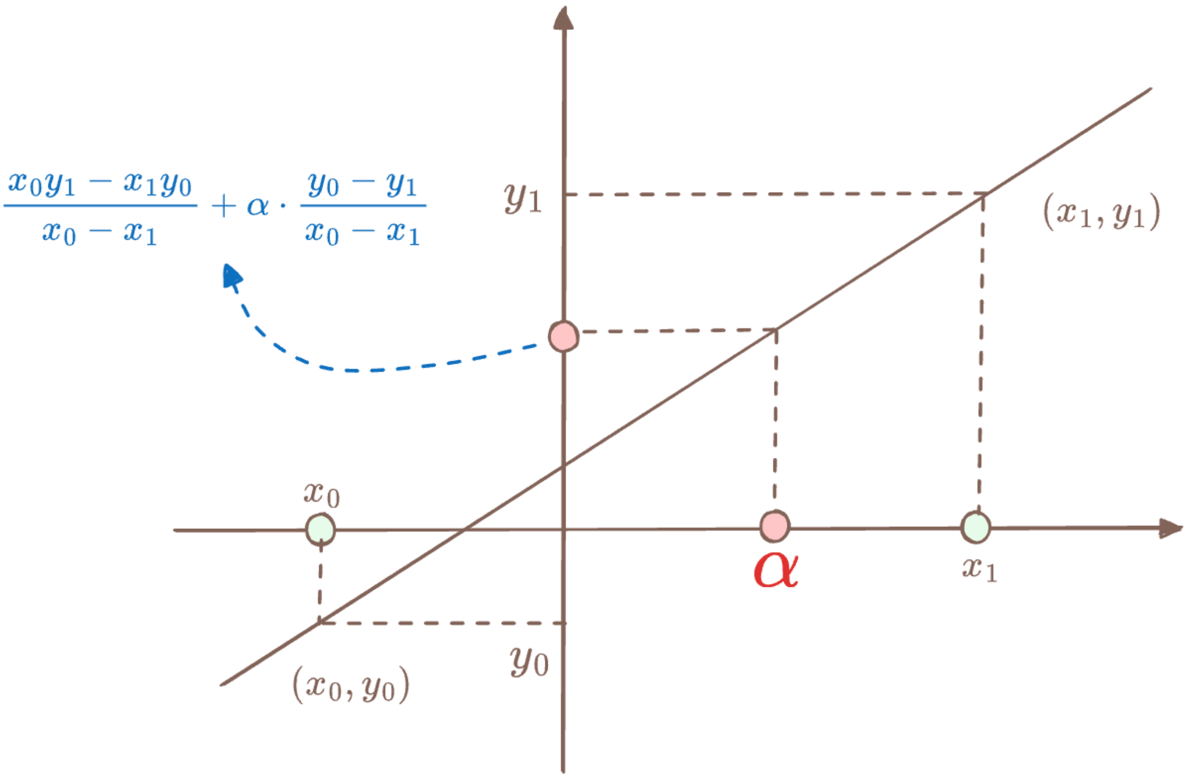
How should we understand the $\mathsf{fold}_\alpha(\cdot, \cdot)$ function? It is essentially a polynomial interpolation process. We treat the two rows to be folded as sets of points on two separate domains, specifically the $\mathsf{diag}(T_i) = (t_0, t_1, \ldots, t_{n_{i-1}-1})$ and $\mathsf{diag}(T'_i) = (t'_0, t'_1, \ldots, t'_{n_{i-1}-1})$ used in the recursive encoding process:

$$\begin{pmatrix} (t_0, c_0), & (t_1, c_1), & \ldots, & (t_{i-1}, c_{n_{i-1}-1}) \\ (t'_0, c_{n_{i-1}}), & (t'_1, c_{n_{i-1}+1}), & \ldots, & (t'_{i-1}, c_{n_i-1}) \end{pmatrix} \tag{9}$$

We then interpolate each column of the above matrix over the domain (t_j, t'_j) to produce a set of $n_{i-1} = n_i/2$ polynomials, denoted as $p_j^{(i-1)}(X)$, where $0 \leq j < n_{i-1}$. The Prover then evaluates each $p_j^{(i-1)}(X)$ at $X = \alpha_i$, resulting in n_{i-1} values at $X = \alpha_i$. These values constitute the new codeword π_{i-1}.

The definition of the folding function aligns with the linear polynomial interpolation process. We can manually derive the origin of the folding function definition. Since we are performing a half-folding of π_i, the folded codeword will have n_{i-1} values corresponding to "linear polynomials." Suppose the j-th polynomial describes a line passing through two points (x_0, y_0) and (x_1, y_1). The interpolating polynomial $p(X)$ for these two points can be defined as:

$$\begin{aligned} p(X) &= \frac{y_0}{x_0 - x_1}(X - x_1) + \frac{y_1}{x_1 - x_0}(X - x_0) \\ &= \frac{x_0 y_1 - x_1 y_0}{x_0 - x_1} + \frac{y_0 - y_1}{x_0 - x_1} \cdot X \end{aligned} \tag{10}$$



Substituting $x_0 = t_j, x_1 = t'_j$, and $X = \alpha$ yields the definition of the folding function $\text{fold}_\alpha(y_0, y_1)$ as above.

$$\text{fold}_\alpha(y_0, y_1) = p(\alpha) = \frac{t_j \cdot y_1 - t'_j \cdot y_0}{t_j - t'_j} + \frac{y_0 - y_1}{t_j - t'_j} \cdot \alpha \tag{11}$$

If x_0 and x_1 are negatives of each other, i.e., $t_j = -t'_j$, then $\text{fold}_\alpha(y_0, y_1)$ becomes the familiar definition from the FRI protocol:

$$\text{fold}_\alpha(y_0, y_1) = \frac{1}{2}(y_0 + y_1) + \alpha \cdot \frac{y_0 - y_1}{2 \cdot t_j} \tag{12}$$

Since we have defined the folded codeword $\mathbf{c}^{(i-1)}$, the definition of the folding function needs to be consistent with the codeword space generated by the generator matrix G_{i-1}. Continuing with this intuition, assume that in the i-th round, if $\mathbf{c}^{(i)}$ is indeed the encoding of $\mathbf{m}$, then by definition, it satisfies the properties of Foldable Codes:

$$\begin{aligned}
\pi_i &= \mathbf{m}G_i \\
&= (\mathbf{m}_l \parallel \mathbf{m}_r) \begin{bmatrix} G_{i-1} & G_{i-1} \\ G_{i-1} \cdot T_{i-1} & G_{i-1} \cdot T'_{i-1} \end{bmatrix} \\
&= \Big(\mathbf{m}_l G_{i-1} + \mathbf{m}_r G_{i-1} \circ \mathbf{diag}(T_{i-1})\Big) \parallel \Big(\mathbf{m}_l G_{i-1} + \mathbf{m}_r G_{i-1} \circ \mathbf{diag}(T'_{i-1})\Big)
\end{aligned} \tag{13}$$

The Prover folds it in half to obtain the new codeword:

$$\text{fold}_\alpha(\pi_i) = \Big(\text{fold}_\alpha(\pi_i[0], \pi_i[n_{i-1}]), \text{fold}_\alpha(\pi_i[1], \pi_i[n_{i-1}+1]), \ldots, \text{fold}_\alpha(\pi_i[n_{i-1}-1], \pi_i[n_i - 1])\Big) \tag{14}$$

We now verify that each $\text{fold}_\alpha(\pi_i[j], \pi_i[n_{i-1}+j])$ is a linear combination of $\mathbf{m}_l G_{i-1}[j]$ and $\mathbf{m}_r G_{i-1}[j]$ with respect to α:

$$\begin{aligned}
\text{fold}_\alpha(\pi_i[j], \pi_i[n_{i-1}+j]) &= \frac{1}{t_j - t'_j} \cdot \Big(t_j \cdot (\mathbf{m}_l G_{i-1}[j] + t'_j \cdot \mathbf{m}_r G_{i-1}[j]) - t'_j \cdot (\mathbf{m}_l G_{i-1}[j] + t_j \cdot \mathbf{m}_r G_{i-1}[j])\Big) \\
&\quad + \frac{\alpha}{t_j - t'_j} \cdot \Big(\mathbf{m}_l G_{i-1}[j] + t_j \cdot \mathbf{m}_r G_{i-1}[j] - \mathbf{m}_l G_{i-1}[j] - t'_j \cdot \mathbf{m}_r G_{i-1}[j]\Big) \\
&= \mathbf{m}_l G_{i-1}[j] + \alpha \cdot \mathbf{m}_r G_{i-1}[j]
\end{aligned} \tag{15}$$

Thus, the entire folding of π_i is equivalent to a linear combination of $\mathbf{m}_l G_{i-1}$ and $\mathbf{m}_r G_{i-1}$ with respect to α:

$$\mathsf{fold}_\alpha(\pi_i) = \mathbf{m}_l G_{i-1} + \alpha \cdot \mathbf{m}_r G_{i-1} = (\mathbf{m}_l + \alpha \cdot \mathbf{m}_r) G_{i-1} \tag{16}$$

The folded codeword is exactly the half-folded version of $\mathbf{m}$ with respect to α, denoted as $\mathbf{m}^{(i-1)}$, and then encoded with G_{i-1} to obtain π_{i-1}. This is not surprising because Foldable Codes and the recursive folding of codewords are inverse processes, so the parameters T_i and T_i' introduced by the encoding are eliminated after folding.

Below, we walk through a simple example to illustrate how the Commit-phase of the Basefold-IOPP protocol operates.

Public Input

- The codeword of the MLE polynomial $\tilde{f}$, $\pi_3 = \mathsf{Enc}_3(\mathbf{f}) = \mathbf{f} G_3$

Witness

- The coefficient vector of the MLE polynomial $\tilde{f}$, $\mathbf{f} = (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7)$

First Round: Verifier sends a random scalar α_2

Second Round: Prover computes $\pi_2 = \mathsf{fold}_\alpha(\pi_3)$ and sends it to the Verifier

The process of computing π_2 is as follows:

$$\pi_2[j] = \mathsf{fold}_\alpha(\pi_3[j], \pi_3[j+4]), \quad j \in \{0, 1, 2, 3\} \tag{17}$$

The computed $\pi_2 = \mathsf{Enc}_2(f^{(2)})$, meaning π_2 is the codeword of $f^{(2)}$, where $f^{(2)}$ is the folding of f with respect to α_2:

$$f^{(2)}(X_0, X_1) = f(X_0, X_1, \alpha_2) = (f_0 + f_4\alpha_2) + (f_1 + f_5\alpha_2)X_0 + (f_2 + f_6\alpha_2)X_1 + (f_3 + f_7\alpha_2)X_0X_1 \tag{18}$$

Third Round: Verifier sends a random scalar α_1

Fourth Round: Prover computes $\pi_1 = \mathsf{fold}_\alpha(\pi_2)$ and sends it to the Verifier

The process of computing π_1 is as follows:

$$\pi_1[j] = \mathsf{fold}_\alpha(\pi_2[j], \pi_2[j+2]), \quad j \in \{0, 1\} \tag{19}$$

The computed $\pi_1 = \mathsf{Enc}_1(f^{(1)})$, meaning π_1 is the codeword of $f^{(1)}$, where $f^{(1)}$ is the folding of $f^{(2)}$ with respect to α_1:

$$f^{(1)}(X_0) = f(X_0, \alpha_1, \alpha_2) = (f_0 + f_4\alpha_2 + \alpha_1(f_2 + +f_6\alpha_2)) + (f_1 + f_5\alpha_2 + \alpha_1(f_3 + +f_7\alpha_2))X_0 \tag{20}$$

Fifth Round: Verifier sends a random scalar α_0

Sixth Round: Prover computes $\pi_0 = \mathsf{fold}_\alpha(\pi_1)$ and sends it to the Verifier

The process of computing π_0 is as follows:

$$\pi_0[j] = \mathsf{fold}_\alpha(\pi_1[j], \pi_1[j+1]), \quad j = 0 \tag{21}$$

Similarly, $\pi_0 = \mathsf{Enc}_0(f^{(0)})$, meaning π_0 is the codeword of $f^{(0)}$, where $f^{(0)}$ is the folding of f with respect to $(\alpha_0, \alpha_1, \alpha_2)$:

$$f^{(0)} = f(\alpha_0, \alpha_1, \alpha_2) = f_0 + f_1\alpha_0 + f_2\alpha_1 + f_3\alpha_0\alpha_1 + f_4\alpha_2 + f_5\alpha_0\alpha_2 + f_6\alpha_1\alpha_2 + f_7\alpha_0\alpha_1\alpha_2 \tag{22}$$

At this point, the Commit-phase ends, and the Prover has sent (π_2, π_1, π_0) to the Verifier. Upon receiving them, the Verifier first checks whether π_0 is a constant polynomial. However, this alone is insufficient; the Verifier also needs to validate that the Prover's folding operations were honest. If all foldings π_i were to be verified, the Verifier would lose succinctness and, consequently, verification efficiency. Due to the Proximity Gap property, the Verifier only needs to perform a limited number of validations to ensure that π_i is a legitimate codeword.

Query-phase

Similar to the FRI protocol, in the Query-phase, the Verifier conducts multiple rounds of random sampling on the $(\pi_d, \pi_{d-1}, \ldots, \pi_0)$ sent by the Prover to verify the honesty of the folding process. We now discuss each round of the sampling process.

The Verifier will randomly select a position μ within π_d and send it to the Prover, noting that $0 \leq \mu < n_{d-1}$, where n_{d-1} pertains only to π_d. The Prover opens the points $\pi_d[\mu]$ and $\pi_d[\mu + n_{d-1}]$ and also sends the value at position μ in the folded codeword π_{d-1}, i.e., $\pi_{d-1}[\mu]$, along with the Merkle Path for these three points.

Upon receiving these, the Verifier first verifies that these three points correspond correctly to the codewords π_d and π_{d-1}. Then, the Verifier checks whether they satisfy the folding relationship:

$$\pi_{d-1}[\mu] \stackrel{?}{=} \mathsf{fold}_{\alpha_{d-1}}(\pi_d[\mu], \pi_d[\mu + n_{d-1}]) \tag{23}$$

Merely verifying the folding relationship from π_d to π_{d-1} is insufficient. The Verifier must also validate the folding relationships from π_{d-1} to π_0. The Prover must additionally provide the points from π_{d-1} to π_{d-2}. Here, the Verifier does not need to select new random scalars but continues to use μ, because in the next round of folding, the position $\pi_{d-1}[\mu]$ will be folded with another symmetrical point regarding α_{d-2}. The specific symmetrical position depends on the situation: if $\mu < n_{d-2}$, then $\pi_{d-1}[\mu + n_{d-2}]$ is the symmetrical point; if $\mu \geq n_{d-2}$, then $\pi_{d-1}[\mu - n_{d-2}]$ is the symmetrical point. Assume $\mu \geq n_{d-2}$; then the Prover sends $\pi_{d-1}[\mu - n_{d-2}]$ and its Merkle Path to the Verifier to allow the Verifier to check the folding relationship from π_{d-1} to π_{d-2}.

In this way, by providing a single random scalar μ, the Verifier can verify all the folding relationships from π_d to π_0. This verification process constitutes one round.

To elevate reliability to a sufficient level, the Verifier must perform multiple rounds to ensure that the Prover has no room to cheat. The Query-phase leverages the Proximity Gap property. A cheating Prover who alters the codeword is likely to be far from the legitimate encoding space, enabling the Verifier to detect cheating with only a small number of sampling attempts.

Summary

This article described the framework of the Commit-phase and Query-phase of the Basefold-IOPP protocol. This framework generalizes and extends the FRI protocol, expanding from RS-Codes to any Foldable Linear Codes. However, it is important to note that Basefold does not support codeword folding of degree greater than 2. This is because the Basefold-IOPP protocol must not only perform Proximity Testing but also provide an operational result of an MLE polynomial. This will be the topic of the next article in this series.

References

- [ZCF23] Zeilberger, H., Chen, B., Fisch, B. (2024). BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes. In: Reyzin, L., Stebila, D. (eds) Advances in Cryptology – CRYPTO 2024. CRYPTO 2024. Lecture Notes in Computer Science, vol 14929. Springer, Cham.
- [BCIKS20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity Gaps for Reed–Solomon Codes. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*, pages 900–909, 2020.