Homomorphic Fingerprinting Theory for Veri-Store Implementation

1 Introduction

This document provides a technical foundation for implementing homomorphic fingerprinting in the `veri-store` distributed object storage system. Based on the work of Hendricks et al. (PODC 2007), the following outlines the mathematical theory, provides concrete examples, and specifies the implementation approach.

1.1 Problem Context

In erasure-coded storage systems using an m-of-n encoding, a data block B is encoded into n fragments $\{d_1, d_2, \ldots, d_n\}$ such that any m fragments suffice for reconstruction. The fundamental challenge is: **how do we verify that all fragments correspond to the same original block B without reconstructing it?** (Recontruction is expensive in terms of bandwidth and computation.)

Without such verification, different subsets of fragments could reconstruct different blocks, violating data consistency. Traditional approaches require either (1) sending the entire block to all servers (no bandwidth savings), or (2) reconstructing and re-encoding all n fragments during verification (high computational cost).

1.2 The Homomorphic Fingerprinting Solution

Homomorphic fingerprinting provides an elegant solution: compact fingerprints that preserve the algebraic structure of the erasure code. This allows each server to independently verify that its fragment was correctly encoded from the original block, using only small fingerprints rather than full fragments.

2 Core Definitions

2.1 Notation and Finite Fields

This document utilizes the following notation:

- $\mathbb{F}$: A finite field with operators "+" and "$\cdot$"
- $\mathbb{F}_{q^k}$: A finite field of order (size) q^k where q is prime
- $\mathbb{F}_{q^k}[x]$: Set of polynomials with coefficients in $\mathbb{F}_{q^k}$
- $d(x) \in \mathbb{F}_{q^k}[x]$: Polynomial representation of data vector d

For our implementation, we will use $\mathbb{F}_{2^8=256}$ (the field of bytes), making field arithmetic efficient on standard hardware.

2.2 Fingerprinting Functions

Definition 1 (Fingerprinting Function). *An ε-fingerprinting function $fp : \mathcal{K} \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ satisfies:*

$$\max_{\substack{d,d' \in \mathbb{F}^\delta \\ d \neq d'}} \Pr\left[fp(r,d) = fp(r,d') : r \xleftarrow{R} \mathcal{K}\right] \leq \varepsilon$$

where r is randomly selected from keyspace $\mathcal{K}$.

Translation: the probability that two different data fragments produce the same fingerprint is at most ε when the fingerprinting parameter r is chosen randomly. This is analogous to collision resistance in hash functions.

2.3 Homomorphic Property

Definition 2 (Homomorphic Fingerprinting). *A fingerprinting function $fp : \mathcal{K} \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ is* ***homomorphic*** *if for any $r \in \mathcal{K}$, any $d, d' \in \mathbb{F}^\delta$, and any scalar $b \in \mathbb{F}$:*

$$fp(r,d) + fp(r,d') = fp(r,d+d') \tag{1}$$

$$b \cdot fp(r,d) = fp(r, b \cdot d) \tag{2}$$

This property is *crucial*: it means fingerprints preserve the linear algebraic structure of the underlying data, which is exactly what erasure codes use.

3 Two Homomorphic Fingerprinting Functions

3.1 Division Fingerprinting

Division fingerprinting computes the fingerprint as the remainder when dividing a polynomial representation of the data by a random irreducible polynomial.

Theorem 1 (Division Fingerprinting). *Let $\mathbb{F}_{q^k}$ be a finite field of order q^k, and let $P_{q^k} : \mathcal{K} \to \mathbb{F}_{q^k}[x]$ select a monic irreducible polynomial of degree γ uniformly at random. Then:*

$$fp(r, d(x)) = d(x) \bmod p(x),$$

where $p(x) \leftarrow P_{q^k}(r)$ is an ε-fingerprinting function with $\varepsilon \approx \frac{\delta}{q^{k\gamma}}$.

Example 1 (Division Fingerprinting over $\mathbb{F}_{256}$). *Let's compute a fingerprint using division over $\mathbb{F}_{2^8}$ (bytes):*

- *Data: $d = [0x42, 0x7A, 0x3E]$ (3 bytes)*
- *Polynomial representation: $d(x) = 0x42 + 0x7A \cdot x + 0x3E \cdot x^2$*
- *Random irreducible polynomial: $p(x) = x^2 + x + 0x01$ (degree $\gamma = 2$)*
- *Fingerprint: $fp(r, d(x)) = d(x) \bmod p(x)$*

Computing the division (using $\mathbb{F}_{2^8}$ arithmetic):

$$d(x) = (0x3E \cdot x + \textit{quotient}) \cdot p(x) + \textit{remainder}$$

$$fp(r,d) = \textit{remainder} = [r_0, r_1] \text{ (2 bytes)}$$

The collision probability is $\varepsilon \approx \frac{3}{2^{16}} \approx 4.6 \times 10^{-5}$.

3.2 Evaluation Fingerprinting

Evaluation fingerprinting treats data as a bivariate polynomial and evaluates it at a random point.

Theorem 2 (Evaluation Fingerprinting). *Let $\mathbb{F}_{q^{k\gamma}} = \mathbb{F}_{q^k}[x]/p(x)$ be an extension field, and let $S : \mathcal{K} \to \mathbb{F}_{q^{k\gamma}}$ select a random element. Then:*

$$fp(r, d(y, x)) = d(s(x), x), \quad \textit{where } s(x) \leftarrow S(r)$$

is an ε-fingerprinting function with $\varepsilon = \frac{\delta/\gamma}{q^{k\gamma}}$.

4 Application to Erasure Codes

4.1 Linear Erasure Codes

Definition 3 (Linear Erasure Code). *An m-of-n erasure code is* ***linear*** *if there exist fixed constants $b_{ij} \in \mathbb{F}$ for $1 \le i \le n$ and $1 \le j \le m$ such that for any block B:*

$$d_i = \sum_{j=1}^{m} b_{ij} \cdot d_j$$

where $(d_1, \ldots, d_n) \leftarrow encode(B)$.

Translation: Each fragment d_i is a linear combination of the first m fragments. This is true for many common erasure codes, including Reed-Solomon codes and Rabin's Information Dispersal Algorithm. These are both linear erasure codes over finite fields, making them compatible with homomorphic fingerprinting.

4.2 The Key Theorem

Theorem 3 (Fingerprint Encoding Consistency). *Let $fp : \mathcal{K} \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ be a homomorphic ε-fingerprinting function, and let $(encode, decode)$ be a linear erasure code with coefficients $b_{ij} \in \mathbb{F}$. If $(d_1, \ldots, d_n) \leftarrow encode^\delta(B)$, then for any $r \in \mathcal{K}$ and any $1 \le i \le n$:*

$$fp(r, d_i) = encode_i^\gamma(fp(r, d_1), \ldots, fp(r, d_m))$$

Proof. By the linearity of the erasure code:

$$\begin{aligned} fp(r, d_i) &= fp\left(r, \sum_{j=1}^{m} b_{ij} \cdot d_j\right) \\ &= \sum_{j=1}^{m} b_{ij} \cdot fp(r, d_j) \quad \text{(by homomorphism)} \\ &= \text{encode}_i^\gamma(fp(r, d_1), \ldots, fp(r, d_m)) \quad \text{(by linearity of encoding)} \end{aligned}$$

□

Significance: This theorem states that the fingerprint of any fragment equals the encoding of the fingerprints of the first m fragments. Therefore, to verify fragment d_i:

1. Compute $fp(r, d_i)$

2. Retrieve fingerprints $fp(r, d_1), \ldots, fp(r, d_m)$ from servers
3. Compute $\text{encode}_i^\gamma(fp(r, d_1), \ldots, fp(r, d_m))$
4. Verify equality (succeeds with probability $\geq 1 - \varepsilon$ if d_i is correct)

Example 2 (3-of-5 Reed-Solomon Verification)**.** *Consider a 3-of-5 Reed-Solomon code over $\mathbb{F}_{256}$:*

- *Original block: $B = [B_0, B_1, B_2]$ (3 bytes)*
- *Encoded fragments: $d_1 = B_0$, $d_2 = B_1$, $d_3 = B_2$*
- *Parity fragments: $d_4 = B_0 + 2B_1 + 4B_2$, $d_5 = B_0 + 3B_1 + 9B_2$*

To verify fragment d_4 using division fingerprinting with $\gamma = 128$ bits:

1. *Compute fingerprints: $f_1 = fp(r, d_1)$, $f_2 = fp(r, d_2)$, $f_3 = fp(r, d_3)$*
2. *Compute expected fingerprint: $f_4' = f_1 + 2f_2 + 4f_3$ (16 bytes)*
3. *Compute actual fingerprint: $f_4 = fp(r, d_4)$ (16 bytes)*
4. *Verify: $f_4 \overset{?}{=} f_4'$ (succeeds with probability $\geq 1 - \varepsilon \approx 0.9999999$)*

Total communication: $3 \times 16 = 48$ bytes (fingerprints) instead of $3 \times 1MB$ (full fragments).

5 Implementation Plan

5.1 Technology Choices

Fingerprinting Function: Division fingerprinting over $\mathbb{F}_{2^8}$

- **Pros:** Fast implementation, efficient byte-wise operations, well-studied
- **Cons:** Slightly higher collision probability than evaluation fingerprinting

Finite Field: $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$

- Field elements are bytes (8 bits)
- Operations implementable via lookup tables
- Compatible with standard Reed-Solomon libraries

Security Parameters:

- Fingerprint size: $\gamma = 128$ bits (16 bytes)
- For 1MB fragments ($\delta = 2^{20}$ bytes): $\varepsilon \approx \frac{2^{20}}{2^{128}} \approx 2.3 \times 10^{-33}$
- Negligible collision probability for all practical purposes

5.2 Pseudocode

Algorithm 1 Division Fingerprinting Computation

```
Require: Data fragment $d$ (byte array), random key $r$
Ensure: Fingerprint $fp$ (16-byte array)
1: $p(x) \leftarrow \text{IrreduciblePoly}(r, \gamma = 128)$ {Degree-128 polynomial over $\mathbb{F}_{256}$}
2: $d(x) \leftarrow \text{BytesToPoly}(d)$ {Convert data to polynomial}
3: $fp(x) \leftarrow d(x) \bmod p(x)$ {Polynomial division over $\mathbb{F}_{256}$}
4: return PolyToBytes($fp(x)$) {Convert back to bytes}
```

Algorithm 2 Fragment Verification

```
Require: Fragment $d_i$, fingerprints $\{fp(r, d_1), \ldots, fp(r, d_m)\}$, encoding coefficients $\{b_{ij}\}$
Ensure: True if fragment is valid, False otherwise
1: $f_i \leftarrow \text{DivisionFingerprint}(d_i, r)$
2: $f_i' \leftarrow \sum_{j=1}^{m} b_{ij} \cdot fp(r, d_j)$ {Encode fingerprints}
3: return $(f_i == f_i')$
```

5.3 Python Libraries

We will use the following libraries for implementation:

- `galois`: Python package for finite field arithmetic (`https://github.com/mhostetter/galois`)
- `pyfinite`: Alternative for $\mathbb{F}_{2^8}$ operations
- `numpy`: Efficient array operations for polynomial manipulation

Example usage:

```
import galois
import numpy as np

# Define GF(2^8) with standard polynomial
GF256 = galois.GF(2**8)

# Generate random irreducible polynomial of degree 128
p = galois.irreducible_poly(2, 128)

# Compute fingerprint: d(x) mod p(x)
d_poly = GF256(data_bytes)  # Convert bytes to polynomial
fp = np.polydiv(d_poly, p)[1]  # Remainder is fingerprint
```

6 Security Analysis

6.1 Collision Probability

For our chosen parameters ($\gamma = 128$ bits, $\mathbb{F}_{2^8}$):

- Fragment size: $\delta = 2^{20}$ bytes (1 MB)
- Collision probability: $\varepsilon \approx \frac{2^{20}}{2^{128}} \approx 2.3 \times 10^{-33}$

- With 1 billion verification attempts: probability of any collision $\approx 2.3 \times 10^{-24}$

This provides a security level comparable to SHA-256 truncated to 128 bits, which is sufficient for our distributed storage application.

6.2 Attack Resistance

Fragment Forgery: An attacker attempting to create a fake fragment d'_i that passes verification must solve:

$$fp(r, d'_i) = \text{encode}_i^{\gamma}(fp(r, d_1), \ldots, fp(r, d_m))$$

Without knowing r, the attacker must guess a polynomial $d'_i(x)$ such that its remainder modulo a random irreducible polynomial equals a specific target value. Success probability is $\varepsilon \approx 2.3 \times 10^{-33}$.

Collision Attacks: Finding two fragments d, d' with $fp(r, d) = fp(r, d')$ requires trying approximately $\sqrt{2^{128}} = 2^{64}$ candidates (birthday paradox), which is computationally infeasible.

7 Conclusion

Homomorphic fingerprinting provides an elegant and efficient solution for verifying erasure-coded fragments without reconstruction. By leveraging the algebraic structure of linear codes, we can verify fragment consistency using compact 16-byte fingerprints instead of megabyte-sized fragments.

Our implementation plan uses division fingerprinting over $\mathbb{F}_{2^8}$ with 128-bit fingerprints, providing negligible collision probability ($\approx 10^{-33}$) while maintaining efficient computation through byte-wise operations.

Next Steps:

1. Implement `DivisionFingerprint` function using `galois` library
2. Test fingerprint computation on sample data
3. Implement fingerprint-based fragment verification
4. Integrate with Reed-Solomon encoding/decoding
5. Measure performance: fingerprint computation time, verification overhead