2. HOMOMORPHIC FINGERPRINTING

This section defines homomorphic fingerprinting and its applications to erasure codes. Section 2.1 defines fingerprinting, providing two examples: division and evaluation fingerprinting. Section 2.2 defines homomorphic fingerprinting and shows that both division and evaluation fingerprinting are homomorphic fingerprinting functions. Section 2.3 explains the applications of homomorphic fingerprinting functions to erasure codes.

Throughout this paper, let $\mathbb{F}$ denote a finite field with operators "+" and "·", and let $\mathbb{F}_{q^k}$ denote such a field of order q^k where q is prime. Let $t \stackrel{R}{\leftarrow} T$ denote selection of an element from T uniformly at random and its assignment to t.

2.1 Fingerprinting

DEFINITION 2.1. An ε-*fingerprinting function* $fp : K \times \mathbb{F}^{\delta} \to \mathbb{F}^{\gamma}$ satisfies

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

In words, the probability under random selection of r that $fp(r,d) = fp(r,d')$ is at most ε.

Let $\mathbb{F}_{q^k}[x]$ denote the set of polynomials with coefficients in $\mathbb{F}_{q^k}$, with "+" and "·" defined as in normal polynomial arithmetic. A vector $d \in \mathbb{F}_{q^k}^{\delta}$ of δ elements of $\mathbb{F}_{q^k}$ has a natural representation as a polynomial $d(x) \in \mathbb{F}_{q^k}[x]$ of degree less than δ with coefficients in $\mathbb{F}_{q^k}$ where the j^{th} element of d is the coefficient in $d(x)$ of degree j, where $0 \leq j < \delta$. We will use these notations interchangeably, denoting d as $d(x)$ when it assumes this form.

EXAMPLE 2.2. [Rabin fingerprinting] Let $\mathbb{F}_2$ denote a field of order 2, let $K = \{2,3,4,\ldots,2^{\gamma}\}$, and let $P_2 : K \to \mathbb{F}_2[x]$ be a deterministic algorithm that outputs monic irreducible polynomials of prime degree γ with coefficients in $\mathbb{F}_2$ such that

$$\Pr\left[p(x) = P_2(r) : r \stackrel{R}{\leftarrow} K\right] = \Pr\left[p'(x) = P_2(r) : r \stackrel{R}{\leftarrow} K\right]$$

for all $p(x), p'(x) \in \mathbb{F}_2[x]$ of degree γ. That is, P_2 selects monic degree-γ irreducible polynomials uniformly at random, where probabilities are taken with respect to the uniformly random selection of r. Rabin showed that $fp : K \times \mathbb{F}_2^{\delta} \to \mathbb{F}_2^{\gamma}$ defined by

$$\begin{aligned} fp(r,d(x)) :\ & p(x) \leftarrow P_2(r); \\ & \text{return } (d(x) \bmod p(x)) \end{aligned}$$

is an ε-fingerprinting function for $\varepsilon = \frac{\delta}{2^{\gamma}-2}$ [24].

THEOREM 2.3. [Division fingerprinting] Let $\mathbb{F}_{q^k}$ denote a field of order q^k, let the size of K be the number of monic irreducible polynomials of degree γ with coefficients in $\mathbb{F}_{q^k}$, and let $P_{q^k} : K \to \mathbb{F}_{q^k}[x]$ be a deterministic algorithm that outputs monic irreducible polynomials of degree γ with coefficients in $\mathbb{F}_{q^k}$ chosen uniformly at random, with probabilities taken over the choice of input $r \in K$ uniformly at random. Then $fp(r,d) : K \times \mathbb{F}_{q^k}^{\delta} \to \mathbb{F}_{q^k}^{\gamma}$ defined by

$$\begin{aligned} fp(r,d(x)) :\ & p(x) \leftarrow P_{q^k}(r); \\ & \text{return } (d(x) \bmod p(x)) \end{aligned}$$

is an ε-fingerprinting function for $\varepsilon = \frac{\delta}{q^{k\gamma} - q^{\frac{k\gamma}{2}}} \approx \frac{\delta}{q^{k\gamma}}$.

PROOF. As in [24], this is because there are at least $\frac{q^{k\gamma}-q^{\frac{k\gamma}{2}}}{\gamma}$ monic degree-γ irreducible polynomials with coefficients in $\mathbb{F}_{q^k}$ [32], of which any nonzero degree-δ polynomial with coefficients in $\mathbb{F}_{q^k}$ may have at most $\lfloor \frac{\delta}{\gamma} \rfloor$ factors of degree-γ. Consider the difference of any two distinct polynomials with matching fingerprints. Let $d(x), d'(x) \in \mathbb{F}_{q^k}[x]$ and $d(x) \equiv d'(x) \bmod p(x)$ but $d(x) \neq d'(x)$. Then $(d(x) - d'(x)) \equiv 0 \bmod p(x)$, so $p(x)$ is a factor of $(d(x) - d'(x))$. Because $d(x) \neq d'(x)$, $(d(x) - d'(x)) \neq 0$. But there are at most $\frac{\delta}{\gamma}$ different monic degree-γ irreducible polynomials with coefficients in $\mathbb{F}_{q^k}$ that are factors of a nonzero degree-δ polynomial $(d(x) - d'(x))$. Hence, the probability that $p(x)$ is one of these polynomials is at most $\frac{\delta}{q^{k\gamma} - q^{\frac{k\gamma}{2}}}$. □

Division fingerprinting is a generalization of Rabin fingerprinting. Both are fast due to fast implementations of P_2 [24] and P_{q^k} [30].

Let $\mathbb{E}_{q^{k\gamma}} = \mathbb{F}_{q^k}[x]/p(x)$ denote the extension field of polynomials with coefficients in $\mathbb{F}_{q^k}$ of degree less than γ, with "+" defined as normal and "·" defined modulo a constant monic degree-γ irreducible polynomial $p(x) \in \mathbb{F}_{q^k}[x]$. Let $\mathbb{E}_{q^{k\gamma}}[y]$ denote the set of polynomials with coefficients in $\mathbb{E}_{q^{k\gamma}}$, with "+" and "·" defined as normal. It is convenient to consider $d \in \mathbb{E}_{q^{k\gamma}}[y]$ as a polynomial in two variables, $d(y,x)$.

A vector $d \in \mathbb{F}_{q^k}^{\delta}$ of δ elements of $\mathbb{F}_{q^k}$ has a natural representation as a polynomial $d(y,x) \in \mathbb{E}_{q^{k\gamma}}[y]$ of degree less than $\frac{\delta}{\gamma}$ in variable y. The j^{th} element of d is the coefficient in $d(y,x)$ of degree $\lfloor \frac{j}{\gamma} \rfloor$ in variable y and degree $j \bmod \gamma$ in variable x. We will use these notations interchangeably, denoting d as $d(y,x)$ when it assumes this form.

THEOREM 2.4. [Evaluation fingerprinting] Let $\mathbb{E}_{q^{k\gamma}} = \mathbb{F}_{q^k}[x]/p(x)$ denote a field of polynomials with coefficients in $\mathbb{F}_{q^k}$ of degree less than γ with "·" defined modulo $p(x)$, a constant monic degree-γ irreducible polynomial. Let $K = \{0, \ldots, q^{k\gamma} - 1\}$, and let $S : K \to \mathbb{E}_{q^{k\gamma}}$ be a deterministic algorithm that outputs an element of $\mathbb{E}_{q^{k\gamma}}$ chosen uniformly at random, with probabilities taken over the choice of input $r \in K$ uniformly at random. Then the function $fp(r,d) : K \times \mathbb{F}_{q^k}^{\delta} \to \mathbb{F}_{q^k}^{\gamma}$ defined by

$$\begin{aligned} fp(r,d(y,x)) :\ & s(x) \leftarrow S(r); \\ & \text{return } d(s(x),x) \end{aligned}$$

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

PROOF. As in [20], this is because any $\lceil \frac{\delta}{\gamma} \rceil$ points fully determine a polynomial of degree less than $\frac{\delta}{\gamma}$ over a field. Hence, any two distinct polynomials of degree less than $\frac{\delta}{\gamma}$ share fewer than $\frac{\delta}{\gamma}$ points. Because there are $q^{k\gamma}$ different points in $\mathbb{E}_{q^{k\gamma}}$, the probability that a randomly chosen point is shared between two distinct polynomials is at most $\frac{\delta/\gamma}{q^{k\gamma}}$. □

A trivial implementation of S is to return the polynomial representation of r divided into γ coefficients, where each coefficient is an element of $\mathbb{F}_{q^k}$.

Variants of division and evaluation fingerprinting known as the division and evaluation hashes can be used for message authentication. They are two of the fastest hashes, producing the smallest output and requiring the fewest bits of random input [22].

2.2 Homomorphism

Throughout this paper, let $b \cdot d$ denote the application of "·" by a scalar $b \in \mathbb{F}$ to each element in a vector $d \in \mathbb{F}^\sigma$ of σ elements of $\mathbb{F}$.

DEFINITION 2.5. A fingerprinting function $fp : K \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ is *homomorphic* if $fp(r,d) + fp(r,d') = fp(r,d+d')$ and $b \cdot fp(r,d) = fp(r, b \cdot d)$ for any $r \in K$ and any $d, d' \in \mathbb{F}^\delta$, $b \in \mathbb{F}$.

THEOREM 2.6. The fingerprinting functions given in Example 2.2 and Theorem 2.3 are homomorphic.

PROOF. For any $d, d' \in \mathbb{F}^\delta_{q^k}$ and any $r \in K$, $p(x) \leftarrow P_{q^k}(r)$,

$$\begin{aligned} fp(r,d(x)) + fp(r,d'(x)) &= d(x) \bmod p(x) + d'(x) \bmod p(x) \\ &= (d(x) + d'(x)) \bmod p(x) \\ &= fp(r, d(x) + d'(x)) \end{aligned}$$

Moreover, for any $b \in \mathbb{F}_{q^k}$,

$$\begin{aligned} fp(r, b \cdot d(x)) &= (b \cdot d(x)) \bmod p(x) \\ &= b \cdot (d(x) \bmod p(x)) \\ &= b \cdot fp(r, d(x)) \end{aligned}$$

□

THEOREM 2.7. The fingerprinting function given in Theorem 2.4 is homomorphic.

PROOF. For any $d, d' \in \mathbb{F}^\delta_{q^k}$ and any $r \in K$, $s(x) \leftarrow S(r)$,

$$\begin{aligned} fp(r,d(y,x)) + fp(r,d'(y,x)) &= d(s(x),x) + d'(s(x),x) \\ &= (d+d')(s(x),x) \\ &= fp(r, d+d') \end{aligned}$$

Moreover, for any $b \in \mathbb{F}_{q^k}$,

$$\begin{aligned} fp(r, b \cdot d) &= fp(r, (b \cdot d)(y,x)) \\ &= (b \cdot d)(s(x),x) \\ &= b \cdot (d(s(x),x)) \\ &= b \cdot fp(r, d(y,x)) \end{aligned}$$

□

The following lemma restates the properties of a homomorphic fingerprinting function.

LEMMA 2.8. Let $fp : K \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ denote a homomorphic ε-fingerprinting function. For any fixed constants $b_i \in \mathbb{F}$, $1 \le i \le m$,

$$\max_{\substack{d, d_1, \ldots, d_m \in \mathbb{F}^\delta \\ d \ne \sum_{i=1}^m b_i \cdot d_i}} \Pr\left[fp(r,d) = \sum_{i=1}^m b_i \cdot fp(r, d_i) : r \xleftarrow{R} K \right] \le \varepsilon$$

PROOF. Suppose otherwise. That is, suppose that there are $d, d', d_1, \ldots, d_m \in \mathbb{F}^\delta$ such that $d \ne d' = \sum_{i=1}^m b_i \cdot d_i$ and

$$\Pr\left[fp(r,d) = \sum_{i=1}^m b_i \cdot fp(r, d_i) : r \xleftarrow{R} K \right] > \varepsilon$$

By homomorphism, for any $r \in K$,

$$\sum_{i=1}^m b_i \cdot fp(r, d_i) = \sum_{i=1}^m fp(r, b_i \cdot d_i) = fp(r, \sum_{i=1}^m b_i \cdot d_i) = fp(r, d')$$

Then

$$\Pr\left[fp(r,d) = fp(r,d') : r \xleftarrow{R} K \right] > \varepsilon$$

in violation of Definition 2.1. □

Let $\mathsf{encode}(B_j)$ output

$$d_{j1} \;\big|\; \ldots \;\big|\; d_{jn}$$

Then $[\mathsf{encode}(B_1); \mathsf{encode}(B_2); \cdots; \mathsf{encode}(B_\sigma)]$ outputs

$$\begin{array}{c|c|c} d_{11} & \ldots & d_{1n} \\ d_{21} & \ldots & d_{2n} \\ \ldots & \ldots & \ldots \\ d_{\sigma 1} & \ldots & d_{\sigma n} \end{array}$$

Let $d_i = (d_{1i}, d_{2i}, \ldots, d_{\sigma i})$. That is, each d_i is a column vector from above. Then $\mathsf{encode}^\sigma(B)$ outputs

$$d_1 \;\big|\; \ldots \;\big|\; d_n$$

Figure 2.1: Encoding a vector

2.3 Applications to erasure codes

DEFINITION 2.9. An *m-of-n erasure coding scheme* is a pair of deterministic algorithms $(\mathsf{encode}, \mathsf{decode})$, where $\mathsf{encode} : \mathbb{F}^m \to \mathbb{F}^n$ and $\mathsf{decode} : (\mathbb{F} \times \{1, \ldots, n\})^m \to \mathbb{F}^m$. If $d_1, \ldots, d_n \leftarrow \mathsf{encode}(B)$, then $\mathsf{decode}(d_{i_1}, \ldots, d_{i_m}) = B$ for any distinct $i_1, \ldots, i_m$ $(1 \le i_j \le n)$.

Each fragment provided to decode is accompanied by its index $i \in \{1, \ldots, n\}$. For notational simplicity, let each index be implicitly provided to decode.

DEFINITION 2.10. An m-of-n erasure coding scheme $(\mathsf{encode}, \mathsf{decode})$ is *linear* if there exist fixed constants $b_{ij} \in \mathbb{F}$ for each $1 \le i \le n$ and $1 \le j \le m$ such that for any $B \in \mathbb{F}^m$, if $d_1, \ldots, d_n \leftarrow \mathsf{encode}(B)$ then $d_i = \sum_{j=1}^m b_{ij} \cdot d_j$.

Examples of linear erasure coding schemes are Reed-Solomon codes [26] and Rabin's Information Dispersal Algorithm [25].

The following three shorthands will be useful for the next theorem. First, to consider only the i^{th} encoded fragment, define the shorthand $d_i \leftarrow \mathsf{encode}_i(B)$. Second, we abbreviate $\mathsf{encode}(\mathsf{decode}(d_{i_1}, \ldots, d_{i_m}))$ as $\mathsf{encode}(d_{i_1}, \ldots, d_{i_m})$. Third, to apply encode or decode to each of the j^{th} elements of m σ-length vectors for every $j \in \{1, \ldots, \sigma\}$, define the shorthands $\mathsf{encode}^\sigma : (\mathbb{F}^\sigma)^m \to (\mathbb{F}^\sigma)^n$ and $\mathsf{decode}^\sigma : (\mathbb{F}^\sigma)^m \to (\mathbb{F}^\sigma)^m$. Then $d_1, \ldots, d_n \leftarrow \mathsf{encode}^\sigma(B)$ and $B \leftarrow \mathsf{decode}^\sigma(d_{i_1}, \ldots, d_{i_m})$, where $B \in (\mathbb{F}^\sigma)^m$ and $d_i \in \mathbb{F}^\sigma$. See Figure 2.1 for illustration.

THEOREM 2.11. Let $fp : K \times \mathbb{F}^\delta \to \mathbb{F}^\gamma$ be a homomorphic ε-fingerprinting function, and let $(\mathsf{encode}, \mathsf{decode})$ be a linear erasure code with coefficients $b_{ij} \in \mathbb{F}$, for $1 \le i \le n$ and $1 \le j \le m$. If $(d_1, \ldots, d_n) \leftarrow \mathsf{encode}^\delta(B)$, then for any $r \in K$ and any $1 \le i \le n$,

$$fp(r, d_i) = \mathsf{encode}^\gamma_i(fp(r, d_1), \ldots, fp(r, d_m))$$

PROOF.

$$\begin{aligned} fp(r, d_i) &= fp(r, \mathsf{encode}^\delta_i(B)) \\ &= fp(r, \sum_{j=1}^m b_{ij} \cdot d_j) && \text{(by Definition 2.10)} \\ &= \sum_{j=1}^m b_{ij} \cdot fp(r, d_j) && \text{(by Definition 2.5)} \\ &= \mathsf{encode}^\gamma_i(fp(r, d_1), \ldots, fp(r, d_m)) \\ & && \text{(by Definition 2.10)} \end{aligned}$$

□

COROLLARY 2.12. Let $fp : K \times \mathbb{F}^{\delta} \rightarrow \mathbb{F}^{\gamma}$ be a homomorphic ε-fingerprinting function, and let (encode, decode) be a linear erasure code. If $(d_1,\ldots,d_n) \leftarrow \mathsf{encode}^{\delta}(B)$, then for any $d \neq d_i$,

$$\Pr\left[\ fp(r,d) = \mathsf{encode}_i^{\gamma}(fp(r,d_1),\ldots,fp(r,d_m)) : r \stackrel{R}{\leftarrow} K\ \right] \leq \varepsilon$$

PROOF. Follows from Theorem 2.11 and Lemma 2.8. □

Theorem 2.11 and Corollary 2.12 state two useful facts about homomorphic fingerprinting functions. First, the fingerprints from an encoding of a block are equal to the encoding of the fingerprints of the block. That is, homomorphic fingerprinting functions are homomorphic. Second, if the fingerprint of a fragment is equal to the encoding of the fingerprints of other fragments, the fragment is, with high probability, the encoding of the other fragments. That is, homomorphic fingerprinting functions are fingerprinting functions.

3. FINGERPRINTED CROSS-CHECKSUM

The fault-tolerant data storage example we give in Section 4 utilizes a data structure that we call a *fingerprinted cross-checksum*. Before considering the contents of a fingerprinted cross-checksum, recall the following definition of a collision-resistant hash function (e.g., see [27]).

DEFINITION 3.1. A family of hash functions $\{hash_K : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}\}_{K \in K'}$ is (τ,ε')-*collision resistant* if for every probabilistic algorithm A that runs in time τ,

$$\Pr\left[\begin{array}{r} d' \neq d \ \wedge\ hash_K(d') = hash_K(d) : \\ K \stackrel{R}{\leftarrow} K',\ \langle d,d' \rangle \leftarrow A(K) \end{array}\right] \leq \varepsilon'$$

A fingerprinted cross-checksum then has the following form.

DEFINITION 3.2. An *m-of-n fingerprinted cross-checksum* fpcc consists of an array fpcc.cc[] of n values in $\{0,1\}^{\lambda}$ and an array fpcc.fp[] of m values in $\mathbb{F}^{\gamma}$.

The name "fingerprinted cross-checksum" derives from the fact that the array fpcc.cc[] is a cross-checksum [12, 15] and because fpcc.fp[] holds homomorphic fingerprints.

Let $hash : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ denote a random instance of a (τ,ε')-collision resistant hash function family, and let $fp : K \times \mathbb{F}^{\delta} \rightarrow \mathbb{F}^{\gamma}$ be a homomorphic ε-fingerprinting function. Let $\mathsf{random\_oracle} : (\{0,1\}^{\lambda})^n \rightarrow K$ denote a random oracle [3], which is a fixed, public function chosen uniformly at random from all functions from the same domain to the same range. The following definition specifies when a fragment is *consistent* with a fingerprinted cross-checksum.

DEFINITION 3.3. Let fpcc be a fingerprinted cross-checksum. A fragment $d \in \mathbb{F}^{\delta}$ is *consistent* with fpcc for index i, $1 \leq i \leq n$, if

$$\mathsf{fpcc.cc}[i] = hash(d)$$

and

$$fp(r,d) = \mathsf{encode}_i^{\gamma}(\mathsf{fpcc.fp}[1],\ldots,\mathsf{fpcc.fp}[m])$$

where $r = \mathsf{random\_oracle}(\mathsf{fpcc.cc}[1],\ldots,\mathsf{fpcc.cc}[n])$.

THEOREM 3.4. Let A be a probabilistic algorithm that runs in time τ, makes χ queries to random_oracle, and produces an m-of-n fpcc and fragments $d_{i_1},\ldots,d_{i_m}$, and $d'_{i'_1},\ldots,d'_{i'_m}$ such that each fragment is consistent with fpcc for its index. If

$$\begin{array}{rcl} B & \leftarrow & \mathsf{decode}^{\delta}(d_{i_1},\ldots,d_{i_m}) \\ B' & \leftarrow & \mathsf{decode}^{\delta}(d'_{i'_1},\ldots,d'_{i'_m}) \end{array}$$

then the probability that $B \neq B'$ is at most $\varepsilon' + M \cdot \varepsilon$ for constant $M = \chi\binom{n}{m+1}$.

PROOF. Suppose that A, running in time τ, produces some fpcc and fragments $d_{i_1},\ldots,d_{i_m}$ and $d'_{i'_1},\ldots,d'_{i'_m}$, each consistent with fpcc for its index, such that if $B \leftarrow \mathsf{decode}^{\delta}(d_{i_1},\ldots,d_{i_m})$ and $B' \leftarrow \mathsf{decode}^{\delta}(d'_{i'_1},\ldots,d'_{i'_m})$ then $B \neq B'$. $B \neq B'$ implies that for some j, $1 \leq j \leq m$, $d_{i_j} \neq \mathsf{encode}_{i_j}^{\delta}(B')$. Yet, because each fragment is consistent with fpcc, for each $\hat{d}_i \in \{d_{i_j}, d'_{i'_1},\ldots,d'_{i'_m}\}$, by Definition 3.3

$$fp(r,\hat{d}_i) = \mathsf{encode}_i^{\gamma}(\mathsf{fpcc.fp}[1],\ldots,\mathsf{fpcc.fp}[m])$$

where $r = \mathsf{random\_oracle}(\mathsf{fpcc.cc}[1],\ldots,\mathsf{fpcc.cc}[n])$. By Definition 2.9, we can rearrange this to

$$fp(r,d_{i_j}) = \mathsf{encode}_{i_j}^{\gamma}(fp(r,d'_{i'_1}),\ldots,fp(r,d'_{i'_m}))$$

We bound the probability with which A succeeds in producing such values. First suppose that A fails to produce a collision in *hash*. Then, for any random oracle query $\hat{r} \leftarrow \mathsf{random\_oracle}(\mathsf{h}_1,\ldots,\mathsf{h}_n)$, A possesses at most one $\hat{d}_i$ such that $hash(\hat{d}_i) = \mathsf{h}_i$, for each $1 \leq i \leq n$. Of these n fragments $\hat{d}_1,\ldots,\hat{d}_n$, consider each selection of $m+1$ of them, $\hat{d}_{i_0},\hat{d}_{i_1},\ldots,\hat{d}_{i_m}$, such that $\hat{B} \leftarrow \mathsf{decode}^{\delta}(\hat{d}_{i_1},\ldots,\hat{d}_{i_m})$ implies $\hat{d}_{i_0} \neq \mathsf{encode}_{i_0}^{\delta}(\hat{B})$. This selection satisfies $fp(\hat{r},\hat{d}_{i_0}) = \mathsf{encode}_{i_0}^{\gamma}(fp(\hat{r},\hat{d}_{i_1}),\ldots,fp(\hat{r},\hat{d}_{i_m}))$ with probability at most ε, by Corollary 2.12. Since there are at most $\binom{n}{m+1}$ such selections per random oracle query, and since there are χ queries to the random oracle, the probability with which A generates any such $\hat{d}_{i_0},\ldots,\hat{d}_{i_m}$ without finding a collision in *hash* is at most $M \cdot \varepsilon$ where $M = \chi\binom{n}{m+1}$. Adding the probability ε' that A finds a collision in *hash*, the total probability of A's success is bounded by $\varepsilon' + M \cdot \varepsilon$. □

4. EXAMPLE: IMPROVING AVID

This section illustrates how homomorphic fingerprinting can improve distributed protocols by modifying the AVID [5] protocol to make it more bandwidth efficient. Section 4.1 describes AVID. Section 4.2 highlights our modifications. Section 4.3 provides a complete description along with pseudo-code of the modified protocol, AVID-FP. Section 4.4 proves that AVID-FP satisfies the functional specification of an asynchronous verifiable information dispersal protocol given in [5]. Both the AVID and AVID-FP protocols can be used to build a Byzantine fault-tolerant distributed storage system using only $3f+1$ servers [6], where f is an upper bound on the number of faulty servers.

This section assumes that there are n servers and that a data block is erasure coded into fragments such that any m fragments suffice to decode it, where $m \geq f+1$ and $n = m+2f$. The system model is similar to that in [5]; there are authenticated, reliable, asynchronous point-to-point communications channels between all servers and clients, and all servers and clients are computationally limited so as to be unable to break the utilized cryptographic primitives.

4.1 AVID

AVID [5] is an asynchronous verifiable information dispersal protocol. In such a protocol, a client disperses some block B, which can later be retrieved by any client. The verifiability of the protocol ensures that any two clients retrieve the same block after dispersal.

For simplicity, the description of AVID in this section is restricted to $m = f+1$ and $n = 3f+1$. To write a block, a client encodes it into fragments and computes the hash of every fragment, creating a cross-checksum. The client sends to each server its fragment and the cross-checksum. Each server then echoes the cross-checksum and its fragment to all other servers in an `echo` message. After receiving $2f+1$ fragments and matching cross-checksums in `echo` messages, a correct server decodes the block, re-encodes it, and verifies each component of the cross-checksum, aborting if inconsistencies are found. A correct server then broadcasts this consistent cross-checksum and its fragment from the re-encoding to all other servers in a `ready` message. A correct server does likewise if it receives $f+1$ `ready` messages before it receives $2f+1$ `echo` messages. After receiving $2f+1$ `ready` messages, a correct server can conclude that $f+1$ servers broadcast `ready` messages that all correct servers will eventually receive. Hence, all correct servers will broadcast `ready` messages, and so all will receive at least $2f+1$ such messages and reach this point. The server can then reconstruct its fragment if needed and store this value. The bandwidth required to store block B is then $O(n^2 \frac{|B|}{m}) = O(n \frac{3f+1}{f+1} |B|) = O(n|B|)$, assuming the cross-checksum is of negligible size.

To read a block, a client retrieves a fragment and cross-checksum from each server until it finds a matching cross-checksum from $f+1$ servers and m fragments that are consistent with this cross-checksum. These fragments are decoded and returned.

4.2 AVID-FP

This section modifies AVID to utilize homomorphic fingerprinting, creating a new protocol, AVID-FP. AVID-FP differs from AVID in that servers agree upon a fingerprinted cross-checksum that is consistent with a block rather than on the block itself; servers need not echo fragments. The bandwidth required to store block B in AVID-FP is then $O(n \frac{|B|}{m}) = O(\frac{m+2f}{m} |B|) = O(|B|)$, assuming a fingerprinted cross-checksum is of negligible size.

In AVID-FP, each cross-checksum is replaced by the fingerprinted cross-checksum from Section 3. Unlike a cross-checksum, a server can verify with a fingerprinted cross-checksum that its fragment corresponds to a unique block without knowing the entire block. As a consequence, there is no need to send a fragment along with each `echo` or `ready` message, which saves substantial bandwidth. Furthermore, a server has nothing to re-encode and verify upon receiving an `echo` or `ready` message, saving a substantial amount of computation.

A less welcome consequence is that a correct server cannot reconstruct its fragment if it is not provided by the client. This is not a problem, however, because a server can still verify that enough other correct servers received consistent fragments such that a consistent block will always be retrievable in the future. Hence, after a block is dispersed, at least $f+1$ correct servers will know the agreed-upon fingerprinted cross-checksum and at least m will know their fragments. To read a block, a client retrieves these $f+1$ matching fingerprinted cross-checksums and m consistent fragments.

4.3 AVID-FP pseudo-code

Pseudo-code for AVID-FP can be found in Figure 4.2. In order to disperse a value B in AVID-FP, a client generates fragments (line 101) and the fingerprinted cross-checksum (lines 102–104). The client then sends each server its fragment and the fingerprinted cross-checksum.

Each server verifies that the fragment it receives is consistent with the fingerprinted cross-checksum (lines 600–603). If this is true, the server stores the fragment and sends an `echo` message containing the fingerprinted cross-checksum to all other servers (lines 604–605).

Upon receiving $m+f$ `echo` messages with matching fingerprinted cross-checksums from unique servers, a server can determine that at least m correct servers sent such messages and hence stored fragments consistent with the fingerprinted cross-checksum (line 701). The server then sends a `ready` message containing the fingerprinted cross-checksum to all other servers (line 702).

If a server receives $f+1$ `ready` messages with matching fingerprinted cross-checksums from unique servers (line 801), at least one must be from a correct server that determined that at least m correct servers stored consistent fragments. Hence, such a server can determine likewise and send a `ready` message to all other servers, if it has yet to do so (line 802).

If a server receives $2f+1$ `ready` messages with matching fingerprinted cross-checksums from unique servers, at least $f+1$ must be from correct servers (line 804). Hence, each correct server will receive at least these $f+1$ matching `ready` messages. Then each correct server will send a `ready` message (lines 801–802), so each correct server will actually receive at least $2f+1$ matching `ready` messages. Thus, a correct server can conclude upon receiving $2f+1$ `ready` messages that all correct servers will eventually receive $2f+1$ `ready` messages, as well. The server can then save the agreed upon fingerprinted cross-checksum and respond to the client (line 806).

Upon receiving $2f+1$ responses, the client is assured that $f+1$ correct servers have saved the same fingerprinted cross-checksums and that m correct servers have stored fragments consistent with this fingerprinted cross-checksum. To retrieve a block, then, a client retrieves a fragment and fingerprinted cross-checksum from each server, waiting for matching fingerprinted cross-checksums from $f+1$ servers (lines 407–408) and consistent fragments from m servers (line 411). These fragments are then decoded and the resulting block is returned.

4.4 AVID-FP correctness

To see why this is correct, recall the definition of an asynchronous verifiable information dispersal scheme given in [5]:

DEFINITION 4.1. An *(m,n)-asynchronous verifiable information dispersal scheme* is a pair of protocols $(\mathsf{disperse}, \mathsf{retrieve})$ that satisfy the following with high probability:

Termination: If $\mathsf{disperse}(B)$ is initiated by a correct client, then $\mathsf{disperse}(B)$ is eventually completed by all correct servers.

Agreement: If some correct server completes $\mathsf{disperse}(B)$, all correct servers eventually complete $\mathsf{disperse}(B)$.

Availability: If $f+1$ correct servers complete $\mathsf{disperse}(B)$, a correct client that initiates $\mathsf{retrieve}()$ eventually reconstructs some block B'.

Correctness: After $f+1$ correct servers complete $\mathsf{disperse}(B)$, all correct clients that initiate $\mathsf{retrieve}()$ eventually retrieve the same block B'. If the client that initiated $\mathsf{disperse}(B)$ was correct, then $B' = B$.

```
c_disperse(B):                                          /* Client disperse protocol */
100: store_count ← 0
101: d_1,...,d_n ← encode^δ(B)
102: for (i ∈ {1,...,n}) do fpcc.cc[i] ← hash(d_i)
103: r ← random_oracle(fpcc.cc[1],...,fpcc.cc[n])
104: for (i ∈ {1,...,m}) do fpcc.fp[i] ← fingerprint(r, d_i)
105: for (i ∈ {1,...,n}) do send(disperse, fpcc, d_i) to S_i
Upon receiving (stored) from S_i for the first time
200: store_count ← store_count + 1
201: if (store_count = 2f + 1) then return SUCCESS

c_retrieve():                                           /* Client retrieve protocol */
300: fpcc ← NULL;   State[*] ← ⟨NULL, NULL, NULL⟩
301: for (i ∈ {1,...,n}) do send(retrieve) to S_i
Upon receiving (retrieved, fpcc^, ⟨fpcc', d⟩) from S_i
400: if (NULL ≠ fpcc') then
401:     h ← hash(d)
402:     r ← random_oracle(fpcc'.cc[1],...,fpcc'.cc[n]);   fp ← fingerprint(r, d)
403:     fp' ← encode_i^γ(fpcc'.fp[1],...,fpcc'.fp[m])
404: if (NULL = fpcc' ∨ (fp = fp' ∧ h = fpcc'.cc[i])) then
405:     State[i] ← ⟨fpcc^, fpcc', d⟩
406:
407: if (|{j : State[j] = ⟨fpcc^, *, *⟩ ∧ fpcc^ ≠ NULL}| = f + 1) then
408:     fpcc ← fpcc^
409: if (fpcc ≠ NULL) then
410:     Frags ← {d_j : State[j] = ⟨*, fpcc, d_j⟩}
411:     if (|Frags| = m) then return decode^δ(Frags)

s_init():                                               /* Initialize server state */
500: echoed ← ⟨NULL, NULL⟩;   verified ← NULL
501: EchoSet_* ← ∅;   ReadySet_* ← ∅

/* Server i code to disperse data */
Upon receiving (disperse, fpcc, d_i) from client
600: h ← hash(d_i)
601: r ← random_oracle(fpcc.cc[1],...,fpcc.cc[n]);   fp ← fingerprint(r, d_i)
602: fp' ← encode_i^γ(fpcc.fp[1],...,fpcc.fp[m])
603: if (echoed = ⟨NULL, NULL⟩ ∧ fp = fp' ∧ h = fpcc.cc[i]) then
604:     echoed ← ⟨fpcc, d_i⟩
605:     for (j ∈ {1,...,n}) do send(echo, fpcc) to S_j
Upon receiving (echo, fpcc) from S_j
700: EchoSet_fpcc ← EchoSet_fpcc ∪ {j}
701: if (|EchoSet_fpcc| = m + f ∧ |ReadySet_fpcc| < f + 1) then
702:     for (j ∈ {1,...,n}) do send(ready, fpcc) to S_j
Upon receiving (ready, fpcc) from S_j
800: ReadySet_fpcc ← ReadySet_fpcc ∪ {j}
801: if (|ReadySet_fpcc| = f + 1 ∧ |EchoSet_fpcc| < m + f) then
802:     for (j ∈ {1,...,n}) do send(ready, fpcc) to S_j
803:
804: if (|ReadySet_fpcc| = 2f + 1) then
805:     verified ← fpcc
806:     send(stored) to client

/* Server i code to retrieve data */
Upon receiving (retrieve) from client
900: send(retrieved, verified, echoed) to client
```

Figure 4.2: AVID-FP pseudo-code

Termination is simple, as in the original AVID protocol. If a correct client initiates disperse, it erasure codes the block and computes a valid fingerprinted cross-checksum before dispersing fragments to each server (lines 101–105). Eventually, at least $m+f$ correct servers receive `disperse` messages, verify their fragments against the fingerprinted cross-checksum, and send `echo` messages to all other servers (line 605). Each correct server eventually receives at least $m+f$ `echo` messages; it will then send a `ready` message (line 702) unless it has already done so (line 802). Thus each correct server will eventually receive at least $2f+1$ `ready` messages, at which point it will send a `stored` message to the client and complete. Hence, all correct servers eventually complete.

Agreement is simpler than in the original AVID protocol because a server in AVID-FP need not reconstruct the block before returning a `ready` message. If some correct server completes disperse(B), then it received $2f+1$ `ready` messages (line 804). At least $f+1$ must have come from correct servers, so all correct servers will eventually receive `ready` messages from these servers. Then the condition satisfied on either line 801 or line 701 will be met for all correct servers, so all correct servers will send `ready` messages and receive at least $2f+1$ such messages, thus completing.

Availability is different than in the original AVID protocol. In AVID, fragments must be echoed such that a correct server can reconstruct its fragment if needed; in AVID-FP, fragments are not echoed. If any correct server completes disperse, it received $2f+1$ `ready` messages. Then at least one correct server received $m+f$ `echo` messages. If not, at most f `ready` messages would be received by any correct server, because no correct server would meet the condition on line 701. Hence, at least m correct servers stored consistent fragments (line 604). Then after $f+1$ correct servers complete disperse, a client that initiates retrieve will eventually receive $f+1$ matching fingerprinted cross-checksums (saved on line 805) along with m consistent fragments, which it will decode and return as some block B'.

Correctness is similar to the original AVID protocol except that the properties of the homomorphic fingerprint are required. Suppose some correct server saves fpcc_1 on line 805 and some other correct server saves $\mathsf{fpcc}_2 \neq \mathsf{fpcc}_1$. Then $m+f$ servers echoed fpcc_1, of which at least m were correct, and $m+f$ servers echoed fpcc_2, of which at least m were correct. Because a correct server will only echo once (line 603 will never be satisfied after line 604 is reached), there are at least $m+m+f$ servers involved, which is a contradiction (there are only $n < m+m+f$ servers in the system). Hence, any block decoded during retrieve is consistent with the same fpcc. Furthermore, if a correct client initiated disperse(B), this fpcc will be consistent with B. Then, by Theorem 3.4, the probability that $B \neq B'$ is negligible, for appropriately chosen parameters.

5. PERFORMANCE

Homomorphic fingerprinting is efficient, contributing little overhead to distributed protocols. To demonstrate that homomorphic fingerprinting is not a substantial computational burden in protocols such as the AVID-FP protocol given above, this section compares an implementation of the evaluation fingerprinting function against cryptographic hashing. The evaluation fingerprinting function implementation in this section is similar to the evaluation hash considered in [22] and [30].

A polynomial

$$d(y,x) = a_\sigma(x) \cdot y^\sigma + \ldots + a_0(x) \cdot y^0 \in \mathbb{E}_{q^{k\gamma}}[y]$$

can be evaluated using Horner's rule. To do so, let $\mathsf{fp} \leftarrow 0$, and for $j = \sigma, \ldots, 0$, iteratively compute $\mathsf{fp} \leftarrow \mathsf{fp} \cdot y + a_j(x)$. The efficiency of this implementation then depends on an efficient implementation of "+" and "·" for $a_j(x), y \in \mathbb{E}_{q^{k\gamma}} = \mathbb{F}_{q^k}[x]/p(x)$, where y, the point at which to evaluate, is the fixed random value $s(x) \leftarrow S(r)$.

Given an implementation of "+" and "·" for $\mathbb{F}_{q^k}$, construct "+" and "·" for $\mathbb{F}_{q^k}[x]/p(x)$ as follows. Consider the representation of $a(x) \in \mathbb{F}_{q^k}[x]/p(x)$ as a polynomial

$$a(x) = b_{\gamma-1} \cdot x^{\gamma-1} + \ldots + b_0$$

where $b_i \in \mathbb{F}_{q^k}$. The "+" operator is defined as the addition of same-degree terms. The "·" operator is defined as multiplication of two polynomials of degree less than γ modulo a constant monic degree-γ irreducible polynomial $p(x) \in \mathbb{F}_{q^k}[x]$.

For fixed $s(x)$, compute $a(x) \cdot s(x)$ as follows. For $0 \leq i < \gamma$, build γ lookup tables mapping each $b_i \in \mathbb{F}_{q^k}$ to $b_i \cdot x^i \cdot s(x) \bmod p(x)$; that is, compute the map $b_i \mapsto b_i \cdot x^i \cdot s(x) \bmod p(x)$. Each of these γ tables will contain q^k entries that are each $\lceil \log_2 q^{k\gamma} \rceil$ bits wide. A 128-bit fingerprint over $\mathbb{F}_{2^8}$ must compute 16 such tables after the random value r is selected; each table is 4 kB, for a total of 64 kB. Given these tables, one can compute $a(x) \cdot s(x)$ as the sum of γ lookups, $\Sigma_{i=0}^{\gamma-1}(b_i \cdot x^i \cdot s(x) \bmod p(x))$. For $\mathbb{F}_{2^8}$, this requires a table lookup plus an exclusive-or per byte of input.

For $\mathbb{F}_{2^8}$, building these tables is efficient: "+" is simply exclusive-or, and "·" can be implemented using a 64 kB lookup table. The "mod" operator can be defined using "+" and "·". Because $p(x)$ is constant, "mod" can be implemented with a lookup table for $b_i \mapsto b_i \cdot x^\gamma \bmod p(x)$. This table will contain 2^8 entries of γ bytes each, for a total of 4 kB for a 128-bit fingerprint, and it can be computed before the random value r is selected.

Gladman's implementation of SHA-1 [11] achieves a throughput of 110 megabytes per second on a 3 GHz Intel Pentium D. On this machine, the time to compute lookup tables for the evaluation fingerprint implementation presented here is 20 microseconds. After this computation, this implementation achieves a throughput of 410 megabytes per second.

6. OTHER PROTOCOLS

m-of-n erasure coding is used in many distributed systems (e.g., [1, 6, 8, 13, 18, 28]), because it reduces storage, network bandwidth, and I/O bandwidth. The savings approaches a factor of m when compared to replication. The division and evaluation fingerprinting functions are homomorphic over several popular erasure codes. Reed-Solomon codes [26] interpolate a polynomial over a field $\mathbb{F}_{q^k}$, and Rabin's Information Dispersal Algorithm [25] encodes using an $n \times m$ matrix over a field $\mathbb{F}_{q^k}$ where every $m \times m$ submatrix is invertible. Both are linear erasure codes over $\mathbb{F}_{q^k}$. A common field is $\mathbb{F}_{2^8}$ such that field elements are bytes. Rabin fingerprinting is homomorphic over many erasure codes based solely on exclusive-or, such as Online Codes [19] and parity.

Homomorphic fingerprinting provides benefits to erasure-coded Byzantine fault-tolerant storage systems [6, 13]. Section 4 demonstrated how the AVID protocol [5], used in [6], can exploit homomorphic fingerprinting to be more bandwidth efficient. Variants of the PASIS protocol [13, 14] can also exploit homomorphic fingerprinting. In the "non-repairable" protocol a writer sends fragments along with a cross-checksum to each server; a reader returns a block after finding sufficient servers with fragments and matching cross-checksums. Before accepting a value, a reader must reconstruct all fragments and recompute the cross-checksum, a significant computational overhead. This protocol can benefit directly by replacing the cross-checksum with a fingerprinted cross-checksum, obviating the need for fragment reconstruction and cross-checksum recomputation. The "repairable" protocol can also benefit, but requires further modifications.

Homomorphic fingerprinting may also provide benefits to erasure-coded broadcast [8], content distribution [17], and similar applications, if the encoding is not trusted to be consistent without verification.

7. RELATED WORK

A common cryptographic application of universal hashing is for message authentication codes (MACs) [22]. An early proposal by Krawczyk [16] included a MAC similar to Rabin's fingerprints. Shoup presented faster variants [30] along with implementation suggestions to optimize performance. Nevelsteen compares several other variants [22].

Homomorphic fingerprinting functions share homomorphic properties with incremental hashing functions [2]. Incremental hashing, however, is substantially slower because it is based on number-theoretic primitives. The homomorphic properties of incremental hashing are exploited in [17], which applies these homomorphic properties to Online Codes [19] in a peer-to-peer content distribution network.

The algebraic properties of certain universal hashes has been examined before. Rabin used these properties to update the fingerprint of a file [24]. In [29], a similar technique is used by a disk scrubber to check the consistency of erasure-coded data in a benign environment. In [4], algebraic properties are leveraged to permit fast updates of Rabin fingerprints of data structures such as trees.

More distantly related to this technique is verifiable secret sharing (e.g., [9, 10, 23, 31]), which allows correct participants to verify that a secret was shared among them consistently. The secrecy of the shared value, however, which must be preserved throughout the share distribution and verification process, drives these protocols to employ number-theoretic techniques that are significantly heavier-weight than considered here.

It is worth mentioning that a random oracle, as in Section 3, can be replaced with an evaluation of a distributed pseudo-random function [21] in a protocol such as AVID-FP. This construction has the benefit of requiring only standard cryptographic assumptions.

8. CONCLUSION

Homomorphic fingerprinting enables efficient verification that fragments have been correctly generated by an erasure-coding of a particular data block. A high level of security can be achieved with small fingerprints, and fingerprint generation has lower computational overhead than cryptographic hashing. This technique provides benefits to several distributed protocols. In particular, distributed storage systems capable of tolerating Byzantine clients, which may attempt to write sets of fragments that reconstruct different values depending upon which subset is used, can benefit significantly from this mechanism.

9. ACKNOWLEDGMENTS

We would like to thank the PODC reviewers for their feedback as well as the early reviewers of the ideas in this paper, including Raja Sambasivan, Matthew Wachs, Michael Abd-El-Malek, Theodore Wong, and Jay Wylie. We thank the members and companies of the CyLab Corporate Partners and the PDL Consortium (including APC, Cisco, EMC, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Network Appliance, Oracle, Panasas, Seagate, and Symantec) for their interest, insights, feedback, and support.

This material is based on research sponsored in part by the National Science Foundation, via grants CNS-0326453 and CCF-0424422, and by the Army Research Office, under agreement number DAAD19–02–1–0389. James Hendricks is supported in part by a National Science Foundation Graduate Research Fellowship.

10. REFERENCES

[1] M. K. Aguilera, R. Janakiraman, and L. Xu. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 336–345. IEEE Computer Society, 2005.

[2] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology – CRYPTO '94*, pages 216–233. Springer-Verlag, 1994.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[4] A. Z. Broder. Some applications of Rabin's fingerprinting method. *Sequences II: Methods in Communications, Security, and Computer Science*, pages 143–152, 1993.

[5] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 191–202. IEEE Press, 2005.

[6] C. Cachin and S. Tessaro. Optimal resilience for erasure-coded Byzantine distributed storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 115–124. IEEE Computer Society, 2006.

[7] J. L. Carter and M. N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the 9th ACM Symposium on Theory of Computing*, pages 106–112. ACM Press, 1977.

[8] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 298–313. ACM Press, 2003.

[9] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science*, pages 383–395. IEEE Press, 1985.

[10] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, pages 427–437. IEEE Press, 1987.

[11] B. Gladman. SHA1, SHA2, HMAC and key derivation in C. http://fp.gladman.plus.com/cryptography_technology/sha.

[12] L. Gong. Securely replicating authentication services. In *Proceedings of the 9th International Conference on Distributed Computing Systems*, pages 85–91. IEEE Computer Society, 1989.

[13] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 135–144. IEEE Computer Society, 2004.

[14] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. The safety and liveness properties of a protocol family for versatile survivable storage infrastructures. Technical Report CMU–PDL–03–105, Parallel Data Laboratory, Carnegie Mellon University, 2004.

[15] H. Krawczyk. Distributed fingerprints and secure information dispersal. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 207–218. ACM Press, 1993.

[16] H. Krawczyk. LFSR-based hashing and authentication. In *Advances in Cryptology – CRYPTO '94*, pages 129–139. Springer-Verlag, 1994.

[17] M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Press, 2004.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201. ACM Press, 2000.

[19] P. Maymounkov. Online codes. Technical Report TR2002–833, Secure Computer Systems Group, New York University, 2002.

[20] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21(4):339–374, 1984.

[21] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Advances in Cryptology – EUROCRYPT '99*, pages 327–346. Springer-Verlag, 1999.

[22] W. Nevelsteen and B. Preneel. Software performance of universal hash functions. In *Advances in Cryptology – EUROCRYPT '99*, pages 24–41. Springer-Verlag, 1999.

[23] T. Pedersen. Distributed provers with applications to undeniable signatures. In *Advances in Cryptology – EUROCRYPT '91*, pages 221–242. Springer-Verlag, 1991.

[24] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[25] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[26] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM Journal of Applied Mathematics*, 8:300–304, 1960.

[27] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Proceedings of the 11th International Workshop on Fast Software Encryption*. Springer-Verlag, 2004.

[28] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: Building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 48–58. ACM Press, 2004.

[29] T. Schwarz. Verification of parity data in large scale storage systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, 2004.

[30] V. Shoup. On fast and provably secure message authentication based on universal hashing. In *Advances in Cryptology – CRYPTO '96*, pages 313–328. Springer-Verlag, 1996.

[31] M. A. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology – EUROCRYPT '96*, pages 190–199. Springer-Verlag, 1996.

[32] D. Travis. On irreducible polynomials in Galois fields. *The American Mathematical Monthly*, 70(10):1089–1090, 1963.