Algebraic Signatures
for
Scalable Distributed Data Structures

Witold Litwin
CERIA
University Paris 9[1]

Thomas Schwarz, S.J.
Department of Computer Engineering
Santa Clara University[2]

Technical Report

[1] Prof. Witold LITWIN
Université Paris 9 Dauphine
Pl. du Mal. de Lattre de Tassigny
75016 Paris FRANCE
Witold.Litwin@dauphine.fr
[2] Prof. Thomas Schwarz, S.J.
Dept. of Computer Engineering
Santa Clara University
500 El Camino Real
tschwarz@calprov.org, tjschwarz@scu.edu

Abstract

We describe novel schemes for the signatures of data objects in a Scalable Distributed Data Structure (SDDS), which we call *algebraic signatures*. An algebraic signature exhibits different properties than a cryptographically secure signature, best exemplified by the well-known standard SHA1. We calculate the signature as a power series in a primitive element of a Galois Field (GF) of 2^8 or 2^{16} elements. We use our scheme for the SDDS file transfer to the disk and of the record update management in presence of concurrent SDDS clients and of bucket splits.

One salient consequence of the algebraic properties of our schemes is that the probability that two objects have the same signature while they differ by only a few symbols, can be strictly zero. Otherwise, our scheme offers a very small collision (error) probability. Like the cryptographic schemes in general. Another consequence is that one can perform algebraic operations on the signatures themselves.

We describe the rationale for our schemes, and we recall the theory of the Galois Fields. Next, we present our schemes. Next, we discuss the implementation issues, and overview the main actual performance results of the experimental validation in the SDDS-2000 system. We conclude with the directions for the further work.

1 Introduction

A signature identifies a larger data object (a record, a page, a file, etc.) with relatively few bytes. Comparing signatures should determine equality with high probability at least. The primary application is the *a posteriori* detection of updates to the object. Especially, between the distributed versions of it, e.g., multi-node copies of a file, [Me83], [AA93], [BGMF88], [BL91], [FWA86], [Me91], [SBB90]...

The updates often follow common patterns. For instance, in a text document the cut and paste of *n* successive symbols usually dominates. Likewise, a database record update often changes only relatively few bytes. Typical updates should be most likely to change the signatures, when, as usual, there is no way to guarantee the signature change for every potential update. Besides this common need, further "wish lists" of constraints on "good" signatures can be formulated for specific applications, e.g., for documents [FC87], [KC96], [KC99].

Several algorithms for signature calculus are known. Perhaps the most applied is the well-known standard SHA1. This is a cryptographically secure signature scheme to also prevent the malicious alterations. A standard SHA1 signature is usually 128 byte long.

Signatures are a useful tool for a Scalable Distributed Data Structure (SDDS), e.g., in the SDDS-2000 system [C02]. One need is the backup of RAM bucket to the disk, where possibly only the updated areas should be copied. Next, signing the concurrent record update avoids the lost updates. A high-availability SDDS may have a specific need for testing the consistency of application and parity data etc.

The property of a cryptographic signature, e.g., the standard size of SHA1 signature do not fit best this context. We describe a novel method which generates, as we call it an *algebraic signature.* We calculate the *n*-symbol signature as a power series of typically primitive elements in a Galois Field (GF) of 2^8 or 2^{16} elements. Our symbols are accordingly successive bytes or two-byte words of the object.

The algebraic signatures are not cryptographically secure. They exhibits in turn novel properties, attractive in our context. Short algebraic signatures, e.g. 4-byte long, should

usually suffice. Next, the probability that two objects differing by at most *n* symbols have the same signature of *n* symbols is zero. One can also perform algebraic operations on the signatures themselves. An updated signature of a structured object can then be calculated only from the update to a component signature. This paves the way towards tree structures over the collections of signatures, speeding up the localization of the updated ones.

Below, we describe more in depth our motivating SDDS signature needs. We then recall the theory of the Galois Fields. Next we present our schemes. Afterwards, we overview the implementation tuning, and the experimental results. Finally we provide our conclusion and directions for the further work.

Section 2 presents the motivating SDDS signature needs. Section 3 discusses the Galois Fields. Section 4 introduces our schemes. Section 5 discusses selected the implementation issues, and the experimental performance measurements. Section 6 sums up our conclusions.

2 Signatures for an SDDS

We recall that a Scalable Distributed Data Structure (SDDS) uses the *server* nodes of a multicomputer to store a file consisting of records or more generally objects. Records or objects have a unique key and are stored on each server in *buckets*. The data structure implements the key-based operations of inserts, deletes, and updates as well as scan queries. These operations are requested by an application from the SDDS *client* at its node. The client manages the query delivery to the appropriate server and receives the reply if any. The file scales with the inserts through the splits. Each split sends about half of a bucket to a new one, dynamically appended to the file. Typically the buckets reside for the processing entirely in the distributed RAM.

We apply the signatures to an SDDS at present to the RAM file backup to the disk and to the management of the concurrent updates.

2.1 File backup

We wish to store or backup an SDDS RAM buckets on disks. Perhaps to reuse the RAM for another SDDS bucket, brought from the disk. For this purpose, we backup in parallel each RAM bucket *B* of the file. . We want to copy to the disk possibly only the parts of *B* that changed with respect to the disk content. The whole capability should be an enhancement to the running prototype SDDS-2000 [CERIA].

The traditional approach that comes to mind is to divide the bucket into pages of some relatively small granularity. Then to simply index each page by a dirty/clean bit *B*. *B* is set to “clean” when the page is saved to the disk. It becomes “dirty” at the next write to the page. We copy only the dirty pages.

The implementation of this approach requires that every part of the running SDDS server code is checked for its writes of the bucket. If it does so, it is modified so that it makes dirty the appropriate bit. The SDDS server code has dozens of thousands of lines of code produced by various people over years. The discussed revision would be a major and error prone work. At best it would constitute a long effort, at worst it would be the enhancement that breaks the application.

Another approach is to provide each page with a signature. We compute all the signatures only when we save the bucket. This computation is therefore independent of

all those writing anywhere to the bucket, in any part of the already running code. It is the crucial advantage of the method over the traditional approach in our case.

More in detail, we provide the disk bucket with a *signature map,* which is simply the collection of all its page signatures. When we wish to save the bucket, we compute the signature of each page and compare it to its "buddy" in the signature map. We save the page only if the signatures are different. We update the map in consequence.

The slicing of the bucket into pages is somewhat arbitrary. A changed region might span over several pages. The slicing should be such that the map is entirely in RAM. Or better, it enters the L1 or L2 cache during the comparison phase, perhaps with the help of the **Prefetch** command. The practical page size can then be expected somewhere between 256 B and 64 KB depending on the bucket size. In our case, it will appear that the size of GF used also limits the page size. On the other hand, it will also appear that larger pages decrease the total signature calculus time. In contrast, larger granularity obviously increases the size of transfers to the disk. The best choice is likely to be application dependent.

The signature map can be simply a table. This is our basic choice for the present implementation. The use of algebraic signatures allows however to structure it into a *signature tree* as well. We compute then a signature at the node at next higher level from all those under the node. This may speed up the identification of the portions of the map where the signatures have changed. It should be especially useful if the map in its entirety does not fit into RAM.

2.2 Concurrent Record Updates

Several SDDS clients may attempt to read or update the same record R. It is best for the access performance to let every client read any record without any wait. However, the subsequent update should not override any other. A simple way to ensure this classical requirement for an SDDS could be as follows. It is freely inspired by the *optimistic* option of the concurrency control of MS-Access[3].

We recall that an update operation may only change the non-key part of R. Let R_b denote the *before-image* of R and S_b its signature. We recall that before-image is the content of R that is subject to the update by a client. We call the result of the update *after-image* of R and note it and its signature respectively R_a and S_a. The update can be *conditional* in which case R_a depends on R_b. For instance through the condition: Salary := Salary + 0.01*Sales. Alternatively, the update may be *unconditional*. Then, R_a is set regardless of R_b. For instance : Salary := 1000 or Image (c) := Refresh (Image (c)) for images of a surveillance camera. The application needs R_b for a conditional update. It may not need it for an unconditional one. In both cases, it may not be aware whether actually $R_a \neq R_b$ as the result. As perhaps in the above example, for unlucky salesmen in these hard times, or as long as there is no burglar in the house under surveillance.

The signatures may potentially help managing the concurrent updates to an SDDS as follows. If the application needs R_b, it requests the client to key search R. If R_b might have been updated, the application provides to the client R_b and R_a. The client computes S_a and S_b. If $S_a = S_b$, then the update in fact did not change the record. Such an update terminates at the client. Only if $S_a \neq S_b$, the client sends R_a and S_b to the server. The

[3] The optimistic concurrency control of MsAccess is not the optimistic approach traditionally presented in the database books, e.g., [LBK02].

server accesses R and computes its signature S. If $S_b = S$, then the server updates R to $R := R_a$. Otherwise, it abandons the update. A concurrent update had to happen to R in the meantime, since the client read R_b and the server received its update R_a. If the new update proceeded, it would override that one, making the result non-serializable. The server notifies the client about the rollback which in turn alerts the application. The application may read R again and redo the update accordingly.

If the update is unconditional, then the application may provide only R_a to the client. The client computes S_b and sends the key of R_a to the server requesting S. The server computes S and send it to the client as S_b. From this point, the client and the server can proceed as described above. Calculating and sending S alone as S_b, avoids the transfer of entire R_b to the client, to possibly avoid the useless transfer of R_a to the server in turn. These can be substantial savings, e.g., for our surveillance images.

The scheme does not need locks. Also, as we have seen, the signature calculus saves the useless record transfers. Besides, neither the key search, nor the insert or deletion needs the signature calculus. Hence, none of these operations incurs the concurrency management overhead. All together, the degree of concurrency can be potentially high. The scheme roughly corresponds to the R-Committed isolation level of the SQL3 standard. Its properties make it attractive to many applications that do not need the transaction management. Especially, if the search is the predominant operation, as one considers in general for an optimistic scheme.

The scheme does not store the signatures. Hence, the storage overhead can be zero, unlike, e.g., for timestamps, probably used by MsAccess. Nevertheless, it can still be advantageous to modify the scheme so to store the signatures with their records. The client sends then also S_a to the server which stores it in the file with R_a if it accepts the update. When the client requests R it gets it with S. If the client requests S alone, the server simply extracts S from R, instead of dynamically calculating it. All together, one saves the S_b calculus at the client and that of S at the server. Also, and more significantly perhaps in practice, the signature calculus becomes *entirely deported at the client*. Hence, it becomes entirely parallel among the concurrent clients. As we show later, the storage cost at the server can be about 4-bytes per signature.

Whether one stores the signature or not, the speed of the signature calculus is clearly THE challenge. The resulting access performance should in particular about match the current one of an SDDS. The calculus time should not be longer than a fraction of a millisecond per record in practice.

3 Galois Fields

A Galois field (GF), is finite field. As any field, it supports addition and multiplication. Addition and multiplication are associative, commutative, and distributive, there exists neutral elements called zero and one for addition and multiplication respectively, and there exist inverse elements regarding addition and multiplication.

We denote GF(2^f) a GF defined over the set of all binary strings of a certain length f. We deal only with these GFs unless we state otherwise. The fields GF (2^8) and GF (2^{16}) are our main concern. Their elements are respectively byte and 2-byte strings.

We identify each binary string with a binary polynomial in one formal unknown x. For example, we identify the string 101001 with the polynomial x^5+x^3+1. We further associate with the GF the *generator polynomial g(x)*, which is a polynomial of degree f

that cannot be written as a product of two other polynomials other than the trivial result of a multiplication of 1 with itself.

The addition of two elements in our GF is that of their binary polynomials. In practice, it boils down to the XOR of the two strings. The product of two elements is formally corresponds to the binary polynomial obtained by multiplying the two operand polynomials and taking the remainder modulo *g(x)*, the generator polynomial. . There are several ways to implement this calculus in practice. We use the *logarithmic* multiplication method we detail later on. This seems most practical for our GF (2^8) and GF (2^{16}) fields. It uses the *primitive* elements of a GF which are elements with the following properties.

The *order* of a non-zero element α of a Galois field is the smallest exponent non-zero i such that that:

$$\mathrm{ord}(\alpha) := \min\{i > 0 : \alpha^i = 1\} \quad .$$

All non-zero elements in a GF have a finite order. An element $\alpha \neq 0$ of a GF of size s is *primitive*, if ord(α) = s-1. It well known that for any given primitive element α in a Galois field with s elements, all the non-zero elements in the field are different powers α^i, each with a uniquely determined exponent i, $0 \leq i \leq s$-1. Furthermore, a GF can have several primitive elements.

In particular, any α^i is also a primitive element if i and s-1 are coprime (meaning that they have no non-trivial factors in common). Since our GFs contain 2^f elements, the prime decomposition of 2^f-1 does not contain the prime 2. For our values of f = 8,16, 2^f-1 contains furthermore a few factors Hence there are relatively many primitive elements.

For instance, for f = 255 the prime factorization is 3·5·15. There are 85 + 51 + 15 – 17 – 5 – 3 + 2 = 128 numbers between 2 and 255 that have a prime factor in common with 255. Hence, 255 – 128 = 127 or roughly half of the non-zero elements are primitive. For any a and any f furthermore, a^2, a^4, a^8… are the primitive elements.

The logarithmic multiplication uses the primitive elements as follows. Let α be a primitive element – as before. Every non-zero element β is a power of α. If $\beta=\alpha^i$, we call i the logarithm of β with respect to α and write $i=\log_\alpha(\beta)$ and we call β the antilogarithm of i with respect to α and write $\beta=\mathrm{antilog}_\alpha(i)$. The logarithms are uniquely determined if we choose i to be $0 \leq i \leq 2^f-2$. We set $\log_\alpha(0)=-\infty$.

The multiplication is now given by the following formula:

$$\beta \cdot \gamma = \mathrm{antilog}_\alpha(\log_\alpha(\beta) + \log_\alpha(\gamma)).$$

Here, the addition is to be performed modulo 2^f-1. We can now implement Galois field multiplication by keeping both a table for logarithms and antilogarithms. These tables are of size $2^f f$ and for moderate f (i.e. up to f=16) fit in the L1 or L2 cache during execution on current μ-processors. To accommodate the awkward addition modulo 2^f-1 in the product formula, we can double the size of the antilog table to accommodate indices up to size $2^f \cdot 2$.

The complete implementation of multiplication needs to check for the special case of one of the operands being equal to 0. Assuming that antilog is a table of size $2^f \cdot 2$, we then obtain the following C-pseudo-code for the multiplication:

```
GFElement mult(GFElement left, GFElement right)      {
 if(left==0||right == 0) return 0;
 return antilog[log[left]+log[right]];
}
```

In terms of Assembly instructions, the typical execution costs of the body of the sub-program are two comparisons, four additions (three for table-look-up), three memory fetches and the return statement.

4 Algebraic Signatures

4.1 Basic properties

We call *page P* a string of l symbols p_i ; l = 0,2..l-1. In our case, the symbols p_i are bytes or 2-byte words. The symbols are elements of a Galois field, GF (2^f) for us ; f = 8, 16 basically. We assume that $l < 2^f$-1.

Let $\boldsymbol{\alpha} = (\alpha_1 \ldots \alpha_n)$ be a vector of different non-zero elements of the Galois field. We call $\boldsymbol{\alpha}$ the *n-symbol signature base*, or simply the *base*. The (*n-symbol*) *P signature*, or *P n-signature*, or simply *P signature*, based on $\boldsymbol{\alpha}$, is vector :

$$\mathrm{sig}_{\boldsymbol{\alpha}}(P) = (\mathrm{sig}_{\alpha_1}(P), \mathrm{sig}_{\alpha_2}(P), ..., \mathrm{sig}_{\alpha_n}(P)).$$

Here, for each α, $\mathrm{sig}_{\boldsymbol{\alpha}}(P)$ denotes :

$$\mathrm{sig}_{\alpha}(P) = \sum_{i=0}^{l-1} p_i \alpha^i .$$

Some choices of $\boldsymbol{\alpha}$ coordinates, perhaps pseudo-random, are possibly interesting for cryptographic needs. It is not our goal here. Besides, it is quite clear that best randomizing (hashing) of P should occur when all α are primitive. Our primary interest is nevertheless in $\boldsymbol{\alpha}$ exhibiting the following pattern for some <u>primitive</u> α :

$\boldsymbol{\alpha} = (\alpha, \alpha^2, \alpha^3 \ldots \alpha^n)$ with $n << \mathrm{ord}(a) = 2^f - 1$.

We denote $\mathrm{sig}_{\boldsymbol{\alpha}}$ in this case as $\mathrm{sig}_{\alpha,n}$. Clearly, the collision probability of $\mathrm{sig}_{\alpha,n}$ can be at best 2^{-nf}. If n = 1, this may be insufficient in practice for our f values. Value $n \geq 2$ appears the least practical choice from this perspective.

Next, our interest is in signatures we denote $\mathrm{sig}^2_{\alpha,n}$ whose <u>all</u> coordinates of $\boldsymbol{\alpha}$ are primitive as we have discussed and which is:

$\boldsymbol{\alpha} = (\alpha, \alpha^2, \alpha^4, \alpha^8 \ldots \alpha^{2n})$.

The $\mathrm{sig}^2_{\alpha,n}$ calculus schema offers potentially better randomization for $n > 2$, (for $n \leq 2$, we have $\mathrm{sig}^2_{\alpha,n} = \mathrm{sig}_{\alpha,n}$, since α^2 is primitive in any GF(2^f)). Intuitively, the highest possible order of each element should avoid some collisions that $\mathrm{sig}_{\alpha,n}$ must leads to. We prove it more formally for the cut and paste operation in pages of length smaller than 2^f - 1 below.

The basic new property of $\mathrm{sig}_{\alpha,n}$ calculus scheme is that any change of up to n symbols within P changes the signature <u>for sure</u>. This is our primary rationale in this scheme. More formally we stay this property as the following proposition.

Proposition 1: Provided the page length l is $l < \mathrm{ord}(\alpha)$ which is $2^f - 1$ for $\mathrm{sig}_{\alpha,n}$ signature as α is primitive, the $\mathrm{sig}_{\alpha,n}$ signature discovers any change of up to n symbols per page.

Proof: Assume that the file symbols at locations i_1, i_2, … i_n has been changed, but that the signatures of the original and the altered file are the same. Call d_v the difference

between the respective symbols in position i_ν. By taking the difference of the component signatures, we conclude :

$$\sum_{\nu=1}^{n} \alpha^{i_\nu} d_\nu = 0$$

$$\sum_{\nu=1}^{n} \alpha^{2i_\nu} d_\nu = 0$$

$$\vdots$$

$$\sum_{\nu=1}^{n} \alpha^{ni_\nu} d_\nu = 0$$

Thus, the d_ν are the solutions of a homogeneous linear system:

$$\begin{pmatrix} \alpha^{i_1} & \alpha^{i_2} & \alpha^{i_3} & \alpha^{i_4} & \cdots & \alpha^{i_n} \\ (\alpha^{i_1})^2 & (\alpha^{i_2})^2 & (\alpha^{i_3})^2 & (\alpha^{i_4})^2 & \cdots & (\alpha^{i_n})^2 \\ (\alpha^{i_1})^3 & (\alpha^{i_2})^3 & (\alpha^{i_3})^3 & (\alpha^{i_4})^3 & \cdots & (\alpha^{i_n})^3 \\ (\alpha^{i_1})^4 & (\alpha^{i_2})^4 & (\alpha^{i_3})^4 & (\alpha^{i_4})^4 & \cdots & (\alpha^{i_n})^4 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ (\alpha^{i_1})^n & (\alpha^{i_2})^n & (\alpha^{i_3})^n & (\alpha^{i_4})^n & \cdots & (\alpha^{i_n})^n \end{pmatrix} \cdot \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ \vdots \\ d_n \end{pmatrix} = 0.$$

The coefficients in the first row are all different, since the exponents $i_\nu < \mathrm{ord}(\alpha)$. Therefore the matrix is of Vandermonde type and hence invertible. Hence, the vector of differences $(d_1 \quad d_2 \quad d_3 \quad d_4 \quad \cdots \quad d_n)^t$ is the zero-vector. This contradicts our assumption. Therefore the $\mathrm{sig}_{\alpha,n}$ signature can detect any up to n-symbol change. CQFD

Notice that Proposition 1 holds also for a non-primitive α, as long as one chooses smaller l which should be $l < \mathrm{ord}(\alpha) < 2^f\text{-}1$. However, a generalization of $\mathrm{sig}_{\alpha,n}$ scheme to a base using a non-primitive α does not seem of practical interest. We now prove our intuitive claim with respect to the collision probability of $\mathrm{sig}_{\alpha,n}$ and, naturally, of $\mathrm{sig}^2{}_{\alpha,n}$ with $n \leq 2$.

Proposition 2: Assuming that page length is $l < \mathrm{ord}(\alpha)$ and that every possible page content is equally likely, the probability that the signatures of two different pages coincide is 2^{-nf}.

Proof: The n-symbol signature is a linear mapping between the vector spaces $\mathrm{GF}(2^f)^l$ and $\mathrm{GF}(2^f)^n$. This mapping is an epimorphism, i.e., every element in $\mathrm{GF}(2^f)^n$ is the signature of some page, element of $\mathrm{GF}(2^f)^l$. Consider indeed map ϕ producing the signatures of pages where all but the first n symbols are zeros. We have thus ϕ: $\mathrm{GF}(2^f)^n \rightarrow \mathrm{GF}(2^f)^n$ mapping any $(x_1,\ldots,x_n)$ to the signature of a page with symbols $(x_1,\ldots,x_n,0,\ldots 0)$. Furthermore:

$$\varphi((x_1,x_2,...,x_n)) = \begin{pmatrix} \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \cdots & \alpha^n \\ \alpha^2 & (\alpha^2)^2 & (\alpha^3)^2 & (\alpha^4)^2 & \cdots & (\alpha^n)^2 \\ \alpha^3 & (\alpha^2)^3 & (\alpha^3)^3 & (\alpha^4)^3 & \cdots & (\alpha^n)^3 \\ \alpha^4 & (\alpha^2)^4 & (\alpha^3)^4 & (\alpha^4)^4 & \cdots & (\alpha^n)^4 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \alpha^n & (\alpha^2)^n & (\alpha^3)^n & (\alpha^4)^n & \cdots & (\alpha^n)^n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{pmatrix}.$$

The matrix is of Vandermonde type, hence is invertible. Consequentially, ϕ is an isomorphism and every Galois field element is the signature of a page with all but the first *n* symbols equal to zero. According to linear algebra, the set of pages with a given signature *s* is the coset (p_1, … p_n, 0, 0,… 0) + kernel(sig), where (p_1,…p_n, 0, 0, … 0) is the uniquely determined page with all but the first *n* elements dummy that has signature *s*. All cosets have the same cardinality. The selections of all possible pages being equally likely to appear, our proposition follows. CQFD

We qualified our schemes *algebraic* claiming that they support some form of the algebraic calculus of new signatures from the signatures only. Here is first motivating property of $\text{sig}_{\alpha,n}$ scheme. Its practical importance concerns pages where the change is localized to a rather small substring (or several substrings). **The case is usual in databases**, with the changes localized to the attributes with a few symbols. We show that one may then update the $\text{sig}_{\alpha,n}$ signature only knowing the changed symbols and the before signature. This may clearly largely speed up the signature calculus with respect to the full calculus again (necessary for the more traditional signature schemes we are aware of, SHA1 in particular). Formally:

Proposition 3: Let us change $P = (p_0, p_1, \ldots p_{l-1})$ to page P' where we replace the symbols starting in position *r* and ending with position *s*-1 with the string $q_r, q_{r+1}, \cdots, q_{s-1}$. We define *Δ-string* as $\Delta = (\delta_0, \delta_1, \ldots \delta_{s-r-1})$ with $\delta_i = p_{r+i} - q_{r+i}$. Then for each α in our base α we have:

$$\text{sig}_\alpha(P') = \text{sig}_\alpha(P) + \alpha^r \text{sig}_\alpha(\Delta).$$

Proof: The difference between the signatures is :

$$\begin{aligned} \text{sig}_\alpha(P') - \text{sig}_\alpha(P) &= \sum_{i=r}^{s-1} (q_i - p_i)\alpha^i \\ &= \alpha^r \left(\sum_{i=r}^{s-1} (q_i - p_i)\alpha^{i-r}\right) \\ &= \alpha^r \left(\sum_{i=r}^{s-1} \delta_{i-r}\alpha^{i-r}\right) \\ &= \alpha^r \left(\sum_{i=0}^{s-r-1} \delta_i \alpha^i\right) \\ &= \alpha^r \text{sig}_\alpha(\Delta). \end{aligned}$$

CQFD.

Back to Proposition 1, beyond its explicitly stated goal, it proves also the sure detection of any cut and paste, the switch, in other words, of length up to int (n/2). We now finally for this section prove the practical interest of $sig^2_{\alpha,n}$ schema for these frequent operations of length above that one.

Proposition 4 Let it be an arbitrary page P, and three indices r, s, t of appropriate sizes. We cut a string T of length t beginning with position r and move it into position s in P. We use of a scheme sig_{α} with arbitrary base $\alpha = \alpha_0, \alpha_1, \ldots \alpha_{n-1}$. Assume that ord($\alpha_i$) is above P length, $0 \le i \le n$-1. Next, B contains at least n symbols or T contains at least n symbols. Then, the probability that $sig_{\alpha}(P)$ changes is 2^{-nf}.

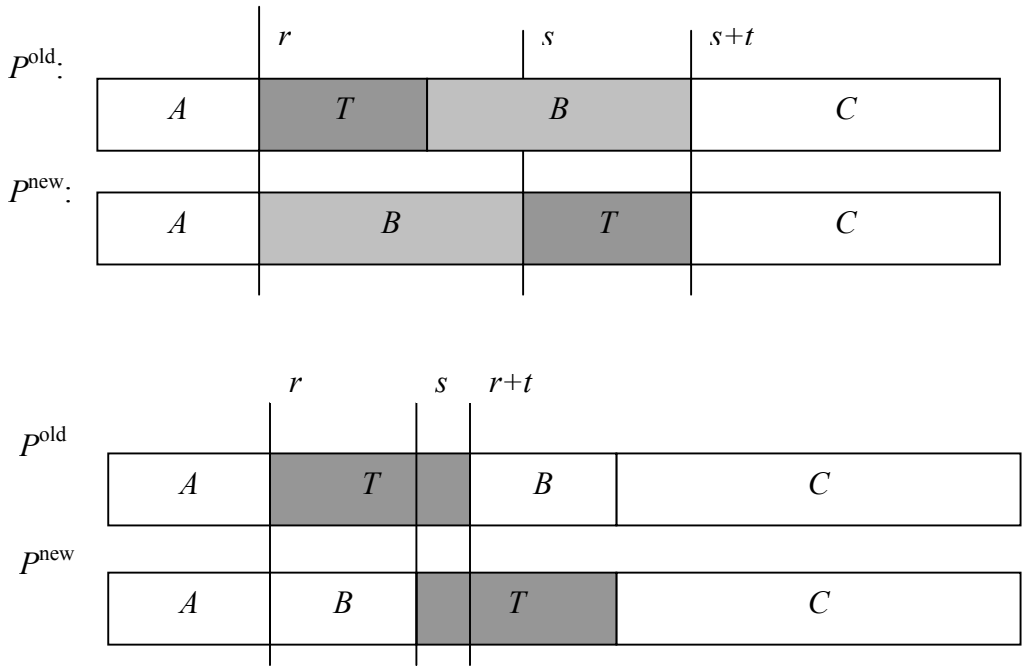

Figure 1: Cut and Paste Illustration for Proposition 4.

Proof: We only treat the case of length(B) ≥ n, the other one being analogue. Next, without loss of generality we assume a forward move of a region T within the file from position r to position s. A backward move just undoes this operation and thus has the same effect on the signature. Figure 1 defines names for the regions of the block and makes a spurious case distinction depending on whether $r+t < s$ or not. For any $\alpha \in \{\alpha_0, \alpha_1, \ldots \alpha_{n-1}\}$, the α signature of the "before" page (the top scheme for both situations) is :

$$sig_{\alpha}(P^{old}) = sig_{\alpha}(A) + \alpha^{r} sig_{\alpha}(T) + \alpha^{r+t} sig_{\alpha}(B) + \alpha^{s+t} sig_{\alpha}(C).$$

The after page signature is:

$$sig_{\alpha}(P^{new}) = sig_{\alpha}(A) + \alpha^{r} sig_{\alpha}(B) + \alpha^{s} sig_{\alpha}(T) + \alpha^{s+t} sig_{\alpha}(C).$$

The difference of the two signatures is:

$$\begin{aligned}\mathrm{sig}_\alpha(P^{\mathrm{new}}) - \mathrm{sig}_\alpha(P^{\mathrm{old}}) &= \alpha^r \mathrm{sig}_\alpha(T) + \alpha^{r+t}\mathrm{sig}_\alpha(B) + \alpha^r \mathrm{sig}_\alpha(B) + \alpha^s \mathrm{sig}_\alpha(T) \\ &= (\alpha^r + \alpha^s)\mathrm{sig}_\alpha(T) + (\alpha^r + \alpha^{r+t})\mathrm{sig}_\alpha(B) \\ &= \alpha^r \left((1+\alpha^s)\mathrm{sig}_\alpha(T) + (1+\alpha^t)\mathrm{sig}_\alpha(B)\right).\end{aligned}$$

This expression is zero only if the right hand side, or the following expression where we use γ_i as an abbreviation is zero:

$$\begin{aligned}&(1+\alpha_i^{\,s})(1+\alpha_i^{\,t})^{-1}\mathrm{sig}_{\alpha_i}(T) + \mathrm{sig}_{\alpha_i}(B) \\ &= (1+\alpha_i^{\,s})(1+\alpha_i^{\,t})^{-1}\mathrm{sig}_{\alpha_i}(T) + \sum_{\nu=n}^{\mathrm{size}(B)\text{-}1} \alpha_i^{\,\nu} b_\nu + \sum_{\nu=0}^{n-1} \alpha_i^{\,\nu} b_\nu \\ &= \gamma_i + \sum_{\nu=0}^{n-1} \alpha_i^{\,\nu} b_\nu.\end{aligned}$$

We now fix the whole situation with the exception of the first n symbols in B. The change in signature is:

$$\begin{aligned}&\left(\gamma_0 + \sum_{\nu=0}^{n-1} \alpha_0^{\,\nu} b_\nu, \gamma_1 + \sum_{\nu=0}^{n-1} \alpha_1^{\,\nu} b_\nu, \ldots, \gamma_{n-1} + \sum_{\nu=0}^{n-1} \alpha_{n-1}^{\,\nu} b_\nu\right) \\ &= (\gamma_0, \gamma_1, \ldots, \gamma_{n-1}) + \left(\sum_{\nu=0}^{n-1} \alpha_0^{\,\nu} b_\nu, \sum_{\nu=0}^{n-1} \alpha_1^{\,\nu} b_\nu, \ldots, \sum_{\nu=0}^{n-1} \alpha_{n-1}^{\,\nu} b_\nu\right)\end{aligned}$$

which is zero if and only if:

$$\left(\sum_{\nu=0}^{n-1} \alpha_0^{\,\nu} b_\nu, \sum_{\nu=0}^{n-1} \alpha_1^{\,\nu} b_\nu, \ldots, \sum_{\nu=0}^{n-1} \alpha_{n-1}^{\,\nu} b_\nu\right) = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1}).$$

The left hand side is a linear mapping in the (b_0, b_1, … b_{n-1}), which has a matrix that is invertible, because it has a Vandermonde type determinant. Therefore, there exists only one combination (b_0, b_1, … b_{n-1}) that is mapped by the mapping onto the right hand vector. This combination will be attained for a randomly picked B with probability 2^{-nf}. CQFD

To obtain the strongest property of a $\mathrm{sig}_{\boldsymbol{\alpha},n}$ signature schema, one should thus choose the defining Galois field elements $\boldsymbol{\alpha}$ whose α_i have the maximal order. The natural choice is that of primitive α_i. This is precisely the rationale in $\mathrm{sig}^2_{\alpha,n}$, assuming the need for $n > 2$. Notice that for GF (2^{16}), the collision probability is already small enough in practice, as we discuss more in Section 5.2.

At this stage of our research, the choice of $\mathrm{sig}^2_{\alpha,n}$ appears only as a trade-off between smaller probability of collision for any update, and the zero probability of collision for updates up to any n symbols. We are able only to conjecture that there is α in GF(2^8) or GF(2^8) for which Proposition 1 and 2 holds. We did not pursue the investigation further, since for our needs $n = 2$ for GF (2^{16}) sufficed (Section 5.2). Since for $n \leq 2$, $\mathrm{sig}^2_{\alpha,n} = \mathrm{sig}_{\alpha,n}$, all the properties of both schemes coincide as well.

4.2 Compound Algebraic Signatures

Our signature schemes keep the property of sure detection of *n*-symbol change as long as the page size in symbols is at most $2^f - 2$. For $f = 16$, the limit on the page size is almost 128 KB. For computing the signature of a larger area one needs a larger field that is 2^{24} at least in practice. Manipulating such fields is less convenient in practice. An alternative approach is to rather to divide the area into largest possible pages and to use the vector of the signatures of those pages. We call such vector *compound* signature. More specifically, we qualify a compound signature of *m* pages, as *m*-fold. We refer to the implementation of a compound signature as to the *signature map* like in Section 2.1.

The practical interest of the compound signatures stretches beyond the motivating case above. One may usefully apply the concept also as an alternative to a signature of an area *A* not exceeding the limit of ord (α)-1. To use an *m*-fold signature $sig_{\alpha,n}$ for instance, one may divide *A* into equally sized pages each provided with $sig_{\alpha,n}$. One locates then for sure and with the *granularity* of *l /m* any up to *n*-symbol change with *a priori* unknown location (hence Proposition 3 does not apply). The price with respect to $sig_{\alpha,n}(A)$, i.e., $sig_{\alpha,n}$ over entire *A* with the granularity thus of *l*, is mainly the about *m* times larger storage overhead. In practice, one can search for *m* leading to a reasonable compromise. Notice that a yet alternative choice for using the *m* times larger overhead if acceptable, is to enhance the sure change detection *resolution* to *mn* symbols anywhere in *A*, using $sig_{\alpha,mn}(A)$.

For larger *m,* further practical importance of the algebraic properties of *m*-fold $sig_{\alpha,n}$ scheme (and in fact similarly of $sig^2_{\alpha,n}$ scheme) is that we may implement signature maps as trees speeding up the search for a changed $sig_{\alpha,n}$. As we show below, with our schemes, we may algebraically, i.e., without reexamining the pages themselves compute the higher level signatures (unlike for more traditional signature schemes we are aware of). If a $sig_{\alpha,n}$ changes, we may update the higher level once again only algebraically. All these capabilities of compound signatures can be of obvious interest to our SDDS file backup application.

The following proposition proves the algebraic properties we discuss for an area partitioned into two pages. Those can be furthermore of different sizes. This is sometimes a useful capability as well, e.g., when *A* starts with a relatively small index of the data the follow in *A*. It generalizes trivially to any larger *m*. It holds pages of different sizes as well.

Proposition 5: Consider that we concatenate two pages P_1 and P_2 of length *l* and *m*, $l + m \leq 2^f$-1, into page (area) denoted $P_1 \mid P_2$. Then, the signature $sig_{\alpha,n}(P_1 \mid P_2)$ is as follows, where sig denotes $sig_{\alpha,n}$:

$$sig_{\alpha,n}(P_1 \mid P_2) = (sig(P_1)+a^l \cdot sig(P_2), sig(P_1)+a^{2l} \cdot sig(P_2), \ldots, sig(P_1)+a^{nl} \cdot sig(P_2).$$

Proof: The proof consists in applying Lemma that follows to each α coordinate α. We note $sig_{\alpha,1}$ as simply sig_α.

Lemma. The 1-symbol $sig_\alpha(P_1 \mid P_2)$ signature is :

$$sig_\alpha(P_1|P_2) = sig(P_1)+a^l\, sig(P_2).$$

Proof: Assume that $P_1 = \{s_1, s_2, \ldots, s_l\}$ and $P_2 = \{s_{l+1}, s_{l+2}, \ldots, s_{l+m}\}$. Then:

$$\begin{aligned}
\mathrm{sig}(\mathbf{P}_1 \mid \mathbf{P}_2) &= \sum_{\nu=1}^{\mathbf{l}+\mathbf{m}} \mathbf{s}_\nu \alpha^\nu \\
&= \sum_{\nu=1}^{\mathbf{l}} \mathbf{s}_\nu \alpha^\nu + \sum_{\nu=\mathbf{l}+1}^{\mathbf{l}+\mathbf{m}} \mathbf{s}_\nu \alpha^\nu \\
&= \sum_{\nu=1}^{\mathbf{l}} \mathbf{s}_\nu \alpha^\nu + \sum_{\nu=1}^{\mathbf{m}} \mathbf{s}_{\nu+\mathbf{l}} \alpha^{\nu+\mathbf{l}} \\
&= \sum_{\nu=1}^{\mathbf{l}} \mathbf{s}_\nu \alpha^\nu + \alpha^{\mathbf{l}} \cdot \sum_{\nu=1}^{\mathbf{m}} \mathbf{s}_{\nu+\mathbf{l}} \alpha^\nu \\
&= \mathrm{sig}(\mathbf{P}_1) + \alpha^{\mathbf{l}} \mathrm{sig}(\mathbf{P}_2)
\end{aligned}$$

CQFD.

Proposition 5 holds analogously for $\mathrm{sig}^2_{\alpha,n}$. Together, all the propositions we have formulated prove the potential of our two schemes. They have further algebraic properties we are currently investigating.

5 Experimental Implementation

5.1 Calculus tuning

One can tune the signature calculus. First, one may interpret the page symbols directly as logarithms. This saves a table look-up. The logarithms range from 0 to 2^f-2 (inclusively) with the additional value for log(0). One can set this one to 2^f-1.

Next, the signature calculations form the product with α^i. This one has i as the logarithm. One does not need to look this value up neither.

The following pseudo-code for $\mathrm{sig}_{\alpha,1}$ applies these properties. It uses as parameters the address of an array representing the bucket and the size of the bucket. The constant TWO_TO_THE_F is 2^f. The type GFElement is an alias for the appropriate integer type.

```
GFElement signature(GFElement *page, int pageLength){
    GFElement returnValue = 0;
    for(int i=0; i< pageLength; i++) {
       if(page[i]!=TWO_TO_THE_F-1)
         returnValue ^= antilog[i+page[i]];
    }
    return returnValue;
}
```

The application to the calculation of $\mathrm{sig}_{\alpha,n}$ is easy.

In our file backup application, the bucket usually contains several pages so we typically calculate the compound signature. To tune this calculus, one should consider the best use of the processor caches, i.e., L1 and L2 caches on our Pentium machines. It seems advantageous to explore the cache lines on the log table. Then, it may be gainful to first loop upon the calculus of $\mathrm{sig}_{\alpha,1}$ for all the pages, then move to $\mathrm{sig}_{\alpha^2,1}$ and so on. Our experiments confirmed this intuition.

5.2 Experimental Performance

We have implemented the $\mathrm{sig}_{\alpha,1}$ schemes in our motivating applications for the experimental analysis. The testbed configuration consisted from 1.8 GHz nodes and from 700 Mhz nodes over a 100 Mb Ethernet. One implementation concerned the signature calculus schemes alone with simulated data. In this series of experiments, we examined

variants of the $sig_{\alpha,n}$ calculus with respect to implementation issues and some differences with respect to the basic scheme. We have also experimented with the $sig^2_{\alpha,n}$ whose calculus time were understandably the same. Next, we have ported best calculus of $sig_{\alpha,n}$ to our SDDS-2000 prototype.

In both cases, we have set up for dividing the bucket into the pages of size of 16 KB with the 4-byte signature per page. This choice appears as a reasonable compromise between the signature size, hence its calculus time, and the overall collision probability of order 2^{-32}, i.e. over $4*10^{-9}$. For the record updates, we use the same signature size, but the record size is of 100 bytes.

Internally, the bucket in SDDS-2000 has a RAM index as it is structured into a RAM B-tree. The index is small, a few KB at largest. Bucket size page does not make sense there. We set up for the page size for the index of 128 B.

For the concurrency control, we set up for storing the signature in each record. Alternatively, it was also possible to compute it on-the-fly any time needed, at the server and perhaps the client to unload the former. The actual computation took place only for the updates. Inserts were not affected.

This analysis is presented full in [M02]. The main results are as follows. The stand-alone experiments showed, somehow surprisingly, a large variation of the calculus time depending on the data symbols. The reason seemed to be the influence of the caches L1 and L2. For a given page size, the calculus time was linear with n for $sig_{\alpha,n}$ used.

Next, the SDDS-2000 implementation led to the actual calculus times of 20-30 ms per 1 MB of RAM bucket, manipulated as a mapped file. It was a fraction of ms for the index page and records tested for the concurrent updates. This timing was linear in size of the bucket, and, also somehow surprisingly, rather stable regardless of the signature scheme tested. The calculus time was smaller for a larger page: 64 KB versus 16 KB. It is probably due to the better cache use. The actual transfer time of 1 Mb of RAM to the disk is about 300 ms. *Thus the backup using our signature scheme offered the expected gains. Likewise, the concurrency control scheme proved a practical solution as well.*

In both cases the use of the GF (2^{16}) was somehow more effective than that of GF (2^8). This despite the fact that the logarithm table of the latter could entirely enter the cache, accelerating thus notably the calculus, while not the former. The former used in turn the calculus using at least 2-byte words, actually 4-byte words. This design appears thus a preferable mode and was our final choice for the SDDS-2000.

6 Conclusion

We have present new signature schemes that we have developed for our SDDS manipulation needs. The scheme exhibit new properties interesting for database and other application. These are sure detection of limited size updates and potential for algebraic operations over the signatures themselves. The scheme present also good overall behavior with respect to the probability of detection of typical changes such as through the cut/paste operation. Finally, our signatures may introduce a very small overhead in practice, e.g., 4 bytes per 16 KB page in our case. The experimental implementation in the SDDS-2000 system has confirmed the practical interest of the schemes.

There are further directions for perfecting the schemes. One concerns the various theoretical issues with respect to the algebraic properties. Variants of the basic schemes remain also for deeper study. The mutual behavior of the $sig_{\alpha,n}$ and $sig^2_{\alpha,n}$ with respect to

the practical collision probability, apart of our assumption of the uniform probability of every page content, also remains to be explored. Finally, we did not study the explicit use of the cache managing **Prefetch** macro, providing perhaps substantial further calculus time reduction.

Perspective applications of our scheme go beyond our motivating ones. There is in particular a relationship between our calculus and that of parity calculus we use for the LH*$_{RS}$, based on the Reed-Salomon erasure correcting encoding. The signatures can help preserving the mutual consistency of data and parity records in presence of lost messages etc. Interesting possibilities appear also for the transactional concurrency. control beyond the prevention of the lost updates. We explore all these avenues at present.

Acknowledgments

We thank Lionel Delafosse and Peter Scheuerman for fruitful discussions This work was partly supported by the research grants from Microsoft Research, and from the European Commission project ICONS project no. IST-2001-32429 and the SCU IBM grant 41102-COEN-RSCH-IG-IG09.

Bibliography

[AA93] K. A. S. Abdel-Ghaffar, A. El-Abbadi: Efficient Detection of Corrupted Pages in a Replicated File", ACM Symp. Distributed Computing, 1993, p. 219-227.

[BGMF88] D. Barbara, H. Garcia-Molina, B. Feijoo: "Exploiting Symmetries for Low-Cost Comparison of File Copies", Proc. Int. Conf. Distributed Computing Systems, 1988, p. 471-479.

[BL91] D. Barbara, R. J. Lipton: "A class of Randomized Strategies for Low-Cost Comparison of File Copies", IEEE Trans. Parallel and Distributed Systems, vol. 2(2), 1991, p. 160-170.

[C02] Ceria Web site. http://ceria.dauphine.fr/.

[FC87] C. Faloutsos, S. Christodoulakis: "Optimal Signature Extraction and Information Loss", ACM Trans. Database Systems, Vol. 12(5), p. 395-428.

[FWA86] W. Fuchs, K. L. Wu, J. A. Abraham: "Low-Cost Comparison and Diagnosis of Large, Remotely Located Files", Proc. Symp. Reliability Distributed Software and Database Systems, p. 67-73, 1986.

[KC 95] Jeong-Ki Kim, Jae-Woo Chang: "A new parallel Signature File Method for Efficient Information Retrieval", CIKM 95, p. 66-73.

[KC99] S. Kocberber, F. Can: "Compressed Multi-Framed Signature Files: An Index Structure for Fast Information Retrieval", SAS 99, p. 221-226.

[LBK02] Lewis, Ph., M., Bernstein. A. *Database and transaction Processing.* Addison & Wesley, 2002.

[Me83] J. Metzner: A Parity Structure for Large Remotely Located Data Files, IEEE Transactions on Computers, Vol. C – 32, No. 8, 1983.

[MA00] J. Michener, T. Acar: "Managing System and Active-Content Integrity", Computer, July 2000, p. 108-110.

[M02] R., Mokadem. Stockage de Données en utilisant les Signatures dans les SDDS. Mémoire DEA, U. Paris 9, Dauphine.

[Sch95] B. Schneider: Applied Cryptography: Protocols, Algorithms, and Source Code, Wiley, 2nd edition, 1995, ISBN 0471117099.

[SBB90] Schwarz, Bowdidge, Burkhard, Low Cost Comparison of Files, ICDCS 90, Paris, p. 196-201.