SETH[*]

A VLSI Chip for the Real-time Information Dispersal and Retrieval for Security and Fault-Tolerance

Azer Bestavros
azer@harvard.edu

Harvard University
Aiken Computation Lab.
Cambridge, MA 02138

September, 1989

[*]©1988 Harvard University. All rights reserved. No part of the SETH project may be reproduced, or manufactured, in any form or by any means without the prior written permission of the designers: Azer Bestavros, Steve Morss, and Adam Strassberg.

Abstract

In this paper, we describe SETH, a hardwired implementation of the recently proposed "Information Dispersal Algorithm" (IDA)[1] [Rabin:87]. SETH allows the real-time dispersal of information into different pieces as well as the retrieval of the original information from the available pieces. SETH accepts a stream of data and a set of "keys" so as to produce the required streams of dispersed data to be stored on (or communicated with) the different sinks. The chip might as well accept the streams of data from the different sinks along with the necessary controls and keys so as to reconstruct the original information.

We begin this paper by introducing the Information Dispersal Algorithm and give an overview of SETH operation. The different functions are described and system block diagrams of varying levels of details are presented. Next, we present an implementation of SETH [Bestavros:89] using Scalable CMOS technology that has been fabricated using a MOSIS 3-micon process. We conclude the paper with potential applications and extensions of SETH. In particular, we emphasize the promise of the Information Dispersal Algorithm in the design of I/O subsystems, Redundant Array of Inexpensive Disks (RAID) systems, reliable communication and routing in distributed/parallel systems. SETH demonstrates that using IDA in these applications is feasible.

Key words: Communication, Fault-tolerance, Security, VLSI, Finite fields, Parallel and distributed systems.

Acknowledgments

This project has been a collaboration involving many people. We would like to thank them all. In particular, we would like to express our deepest gratitude to Prof. Michael O. Rabin and Prof. James J. Clark for their great help and support during the course of this work. The techniques used for computing using irreducible polynomial arithmetic were developped by Azer Bestavros. The design and layout of SETH were done by Azer Bestavros and Steve Morss. Logic and circuit level simulations were done by Adam Strassberg. The design of fault-detection experiments and the actual testing of the SETH chip were done by Azer Bestavros.

SETH was laid out using Berkeley MAGIC. Simulations were done using ESIM and SPICE. The layout as well as the simulation results are available in the VLSI Lab, Harvard University [BMS:88]. SETH was fabricated by MOSIS on a 3-micron SCMOS-technology process. Twelve packages were returned and the functionality of the chip was verified.

[1]The IDA (Information Dispersal Algorithm) has been developped by Prof. Michael O. Rabin at Harvard University (Patent pending for Harvard University). A special permission to use the IDA in SETH has been obtained from Prof. Michael O. Rabin and Harvard University.

1 Introduction

The storage and transmission of data in computer systems raises significant security and reliability problems. In particular, data might be lost due to hardware failures, it might be accidentally (or even maliciously) garbled or destroyed, and it might be read and interpreted by unauthorized users.

The common solution to the afforementioned security problems is to have users store and communicate their data using some form of encryption, where only authorized users are enabled to decrypt the information throught the use of appropriate *Secret Keys* [Shamir:79]. The proven difficulty of decrypting the information without knowing the secret key guarantees a high level of *security*. On the other hand, to protect against possible failures, redundancy is often used as the only alternative to achieve fault-tolerance. This is usually done by having the users replicate their (possibly encrypted) data into n different machines [Gibson:88]. The independency of the failure modes of these machines guarantees a high level of *reliability*. The main disadvantages of this *encryption and replication* technique is that it results in an n-fold blowup of the total storage required in the system. Moreover, the existence of *all* the information (althought encrypted) in one site[2] for long periods might make it possible for adversaries to break the secret key.

Recently, Michael Rabin [Rabin:87] proposed the "Information Dispersal Algorithm (IDA)" as a potential technique that would achieve the required security and reliability using a much smaller level of redundancy and by keeping only *partial* information at a specific site. The main idea is to use a secret key to disperse the information of a file F into n pieces which are transmitted and stored on n different machines (or disks) such that the contents of the original file, F, can be reconstructed from any m of its pieces, where $m \leq n$. On the one hand, the proposed technique guarantees the confidentiality of the dispersed information. As a matter of fact, it is hard to get any clue about the original information unless at least m pieces from the dispersed file are collected. This makes the task of the adversaries more difficult, since they have to control m of the sites and not only one. Even if this happens, it is provably very difficult to reconstruct the original file unless the secret key is known. On the other hand, the proposed technique guarantees a higher availability, since it tolerates up to $(n - m)$ failures. The salient feature of the "Information Dispersal Algorithm" is that each of the dispersed pieces is of size $\frac{|F|}{m}$, where $\mid F \mid$ is the size of the original file. Hence, the added redundancy is $(\frac{n}{m} - 1) * 100\%$.

In this paper, we present a hardwired implementation of the "Information Dispersal Algorithm" that would allow the execution of the algorithm in real-time. A chip, which

[2]whether stored in or communicated through that site

we called SETH[3], has been designed in the VLSI Lab of Harvard University using Scalable CMOS technology and has been fabricated by the MOSIS 3-micron process. SETH accepts a stream of data and a set of vector-keys (see the description below) along with the necessary controls so as to produce the required streams of encrypted data to be stored on (or communicated with) the different machines. The chip might as well accept the streams of data from the different machines along with the necessary controls and vector-keys so as to reconstruct the original information.

We begin this paper by introducing the Information dispersal algorithm and presenting an overview of the SETH design. Next, we describe how the different operations are to be done using "irreducible polynomial arithmetic". In particular, we outline general methods for efficient addition and multiplication in these systems. Next, functional units are described and system block diagrams and schematics of varying levels of details are presented. We conclude the paper by examining potential applications and extensions for SETH. In particular, we consider the use of the Information Dispersal Algorithm in the design of I/O subsystems, Redundant Array of Inexpensive Disks (RAID) systems, reliable communication and routing in distributed/parallel systems. SETH demonstrates that using IDA in these applications is feasible.

SETH was designed using MAGIC. The design was tested using ESIM and SPICE. The MAGIC layout of the chip as well as the simulation results are available in the VLSI Lab, Harvard University [BMS:88]. The chip was fabricated by MOSIS on a 3-micron SCMOS-technology process. Twelve packages were returned and the functionality of the chip was verified.

2 Overview of SETH

SETH is an interface that allows the reliable and secure storage and communication of information between the different units of a computing system. Reliability is guaranteed by using redundant communication and/or storage, while security can be achieved using encryption. A major objective in SETH design was *versatility* and *flexibility.* As a matter of fact, SETH can be viewed as a *basic building block*; so that by using many of these blocks in

[3]According to an old Egyptian legend, SETH (an Egyptian prince) killed his brother OSIRIS and cut his body into small pieces and "dispersed" it all around the Eastern Mediteranean. Later, his loving wife, ISIS, "collected" all the pieces together, bringing OSIRIS to life again; and OSIRIS became the Evil God of the Underworld.

different configurations, one can achieve different design objectives (namely different levels of fault-tolerance, recoverability, redundancy and security).

SETH can be operated in two modes; *Write-mode* (Disperse-mode or SETH-mode) or *Read-mode* (Recombine-mode or ISIS-mode). In write-mode, the data given to SETH is encrypted, dispersed and *written* to the different redundant sinks. In read-mode, the data from a number of intact sinks is *read*, decrypted and recombined to yield the originally written information. The encrypt/decrypt and disperse/recombine functions are achieved using a set of *secret keys* that must be supplied to SETH. Figure-1 below, shows the read and write modes of SETH.

Keeping the afforementioned versatility and flexibility in mind, we decided to make the SETH chip interface an 8-bit bus to four 4-bit buses – using four 8-bit keys. Hence, a stream of 8 bits can be dispersed into four streams of 4 bits each, so that any two of these four streams is sufficient to reconstruct the original 8-bit stream. Different series/parallel configurations using SETH are possible. For instance, two SETH chips in parallel can be used to interface a 16-bit bus to eight 4-bit buses or to four 8-bit buses or to two 16-bit buses. In any case the level of redundancy is 100% and the achievable fault-tolerance is 50% (that is we tolerate the loss of 50% of the storage sinks or channels). These, however, are still controllable. For instance by using only three of the four output buses, the redundancy drops to 50%, wherea the level of fault-tolerance becomes 33%. In any of the above cases security is guaranteed since the dispersed data is actually encrypted. Moreover, by using exactly two of the output buses, SETH can be used just for encryption and dispersal (with no added redundancy and no support for failures). Another degree of freedom (both in the redundancy/fault-tolerance and security levels) can be achieved by using series configuration of SETH.

3 Dispersal and Retrieval of Information

In this section we explain how the Information Dispersal Algorithm works. We single out the different operations to be done and which must be implemented in real-time by SETH. We refer the reader to the original paper on IDA, [Rabin:87], for a thorough presentation of the algorithm. In the appendix, we use some examples to show data scattering and gathering using IDA.

Let F be the original file (information) we want to disperse. The main idea behind IDA is to split this file into n different pieces so that recombining any m of these pieces, $m \leq n$,

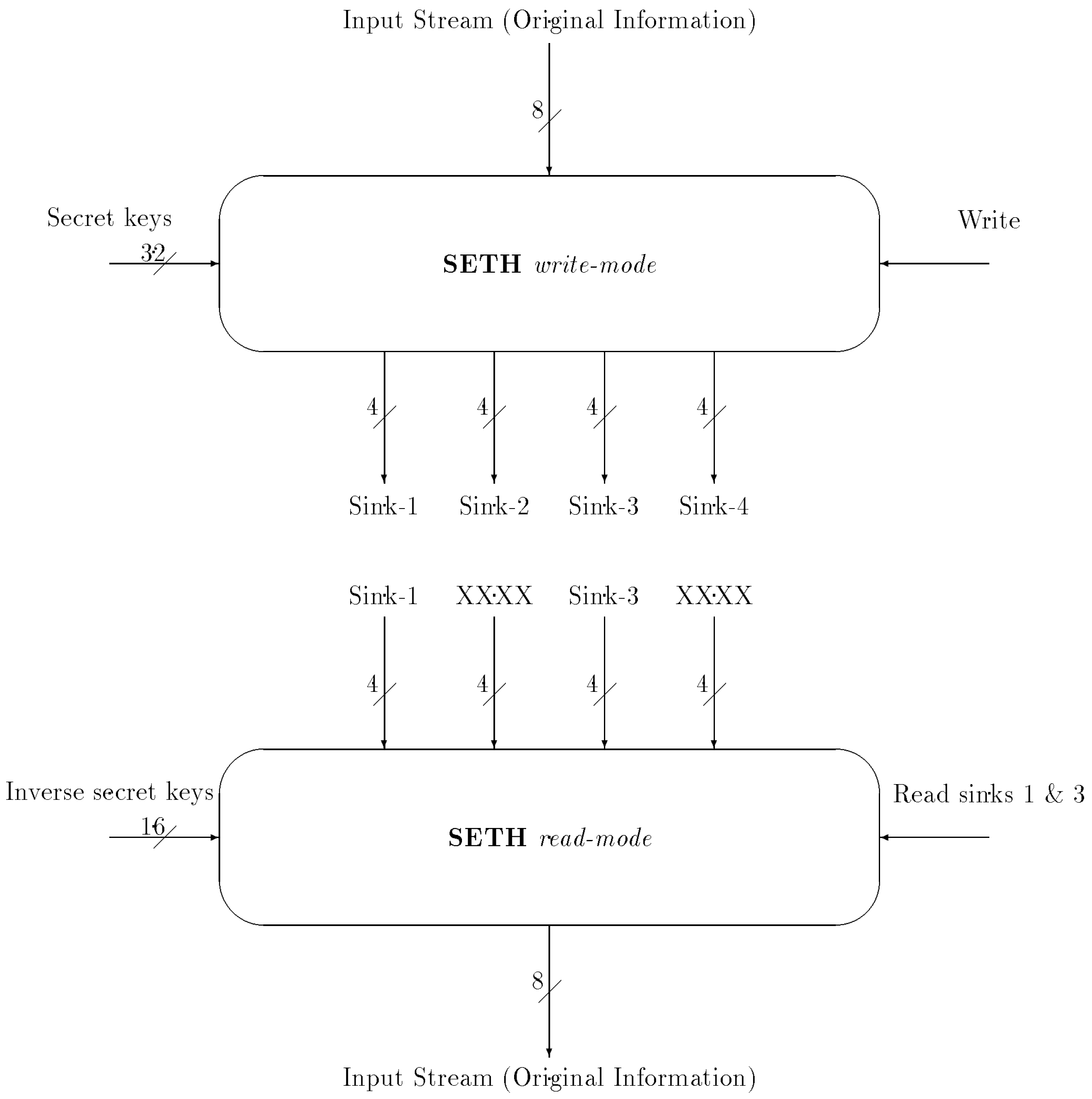

Figure 1: The Read and Write modes of SETH

should be sufficient to reconstruct F, wherea any number of pieces less than m would not be sufficient to reveal any information about the contents of any portion of F.

The original file F can be viewed as a sequence (or stream) of data on the form $F = b_1 b_2 b_3 b_4 \ldots \ldots etc.$, where each b_i in this stream can be viewed as an integer. In order to disperse this stream, we choose a set of n vectors, *secret keys* $(V_1, V_2, ..., V_n)$ each of length m. Theses keys, of course, have to meet certain conditions (see Appendix-A for details). Let A_{nm} be the array whose rows are the selected vectors. A represents a mapping from an m-dimensional space to an n-dimensional space, or in other words, from a sequence of m elements to another sequence of n elements. To disperse the file F, we simply map each sequence of m elements from F into a new sequence of n elements using the transformation A. Each element from the resulting sequence is sent to a different site and kept there. So, for each m elements of F we send *one* element to each of the n sites. Therefore, to disperse the whole file F we will need to send $\frac{|F|}{m}$ elements to each of the n sites.

Now suppose that we want to reconstruct the original file F from the pieces dispersed as described above. This is done by reading any m of these pieces (if less than m pieces are available then the file cannot be reconstructed, and if more than m pieces are available then the use of any subset of m pieces will suffice). Let the pieces be from sites $s1, s2, s3, ..., sm$. Let B_{mm} be the array whose rows are $(V_{s1}, V_{s2}, V_{s3}, ..., V_{sm})$. Thus, B maps sequences of m elements from F into sequences of m elements, which by virtue of the dispersion step above, are kept at sites $s1, s2, s3, ..., sm$. Now, to reconstruct the first m elements of F we need simply to collect the first element from each of m different sites and use the appropriate inverse transformation $(T = B^{-1})$. Note that if the keys were appropriately chosen, such an inverse is guaranteed to exist (see Appendix-A for details).

In SETH we decided to pick $n = 4$ and $m = 2$, so that F is encrypted and dispersed into 4 different pieces and using no less than 2 of these pieces would be sufficient to reconstruct F. Moreover, we decided to represent F as a stream of hexadecimal digits, *nibbles*, (integers in the range [0..15]). Hence, each *byte (8 bits)* of F can be viewed as 2 nibbles and we use the IDA described above to produce 4 nibbles that are dispersed to the 4 different sites. This is illustrated in Figure-2. It is to be noted, however, that the design techniques that we present in this paper are independent of the choices above and can be easily applied to any other choices.

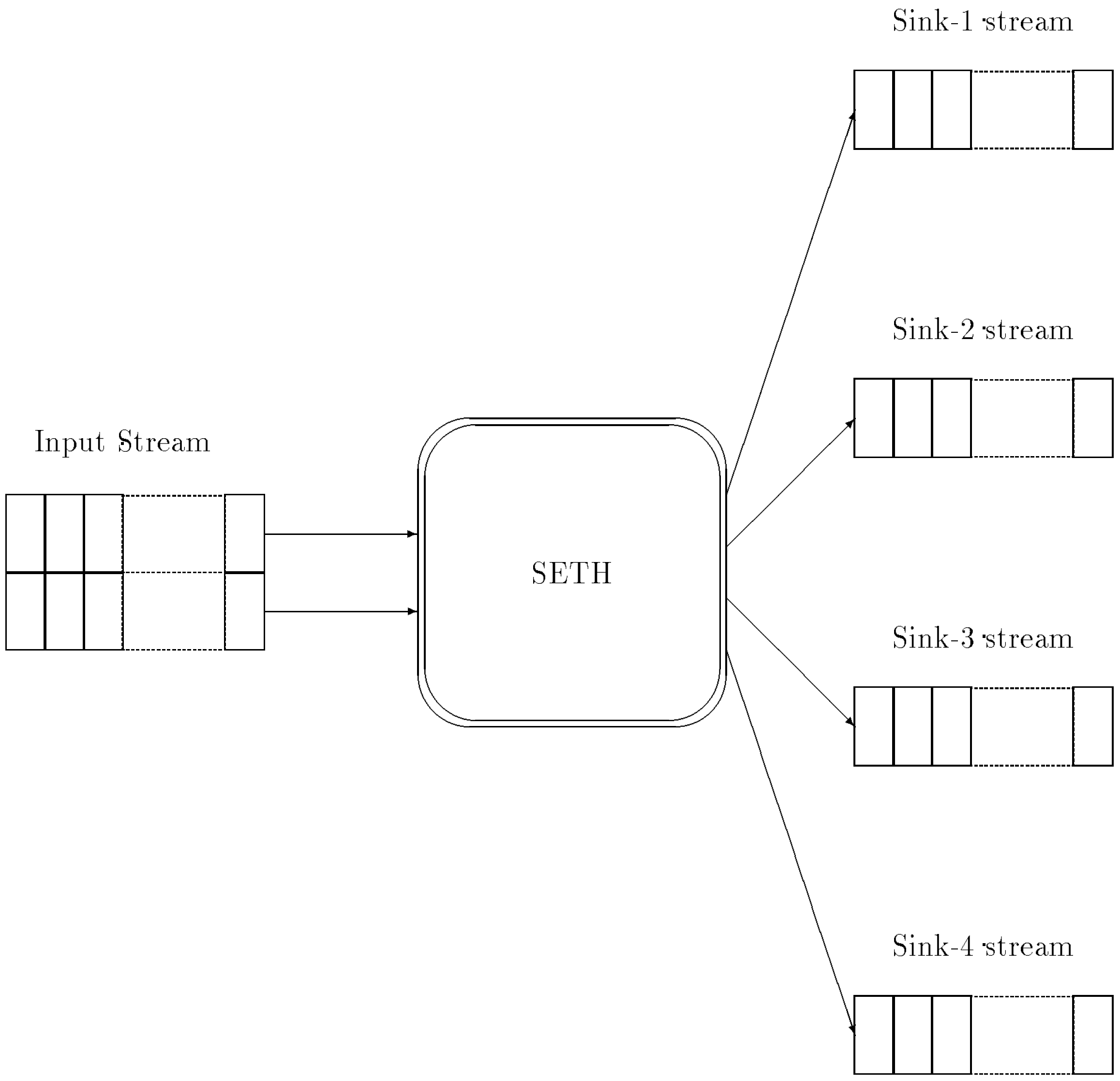

Figure 2: Dispersing a stream of 2 nibbles (1 byte) to 4 streams of nibbles

3.1 The Disperse operation

The *Disperse* transformation is simply a 4×2 matrix A,

$$A = \begin{pmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \\ a_{30} & a_{31} \end{pmatrix}$$

where, V_i is the i^{th} key-vector and each a_{ij} is a hexadecimal (4-bit) number. Let $(b_0 b_1)^T$ represent the two hexadecimal digits (a total of 8 bits) from the input stream. To disperse this piece of information, we use the transformation A as follows:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \\ a_{30} & a_{31} \end{pmatrix} \times \begin{pmatrix} b_0 \\ \\ \\ b_1 \end{pmatrix}$$

where, c_i is a hexadecimal digit (nibble) to be sent to the i^{th} site.

3.2 The Reconstruct operation

Given that the dispersed data at sites $s1$ and $s2$ is intact, the *Reconstruct* transformation is simply a 2×2 matrix T,

$$T = \begin{pmatrix} V_{s1}^T \\ V_{s2}^T \end{pmatrix}^{-1} = \begin{pmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{pmatrix}$$

Let $(c_0 c_1)^T$ represent the two hexadecimal digits (a total of 8 bits) from the two intact sites. To reconstruct the original byte of information, we use the transformation T as follows:

$$\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{pmatrix} \times \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

3.3 The irreducible polynomial arithmetic

All operations (namely additions and multiplications) needed for the IDA have to be carried-out using the *irreducible polynomial arithmetic* where integers are viewed as polynomials over some finite field Z_p, p is a prime number (see Appendix-B for a more detailed discussion and for a complete example). Taking $p = 2$ (an obvious choice for digital applications), integers are represented as polynomials with binary coefficients. For instance, the hexadecimal digits from 0 to 15 are represented as follows:-

$$\begin{pmatrix} 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0100 \\ 0101 \\ 0110 \\ 0111 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ x \\ x+1 \\ x^2 \\ x^2+1 \\ x^2+x \\ x^2+x+1 \end{pmatrix} \qquad \begin{pmatrix} 1000 \\ 1001 \\ 1010 \\ 1011 \\ 1100 \\ 1101 \\ 1110 \\ 1111 \end{pmatrix} = \begin{pmatrix} x^3 \\ x^3+1 \\ x^3+x \\ x^3+x+1 \\ x^3+x^2 \\ x^3+x^2+1 \\ x^3+x^2+x \\ x^3+x^2+x+1 \end{pmatrix}$$

In this system, all operations on hexadecimal digits should be done modulo an irreducible 4th degree polynomial (a polynomial that cannot be divided by any polynomial of 3rd or lesser degree). For instance, it can be easily shown that the polynomial $(x^4 + x + 1)$ is indeed an irreducible 4th degree polynomial. This is the irreducible polynomial used in SETH.

3.3.1 Addition

Addition is really straightforward. To add two integers, we add their corresponding polynomials. This is done by adding (modulo-2) the coefficients of the corresponding powers. The following are examples of additions done in this system:

$$5 + 6 = (x^2 + 1) + (x^2 + x) = (x + 1) = 3$$
$$6 + 7 = (x^2 + x) + (x^2 + x + 1) = 1$$

It is obvious that addition is just the bitwise "exclusive-or" of the binary representation of the integers. This makes the hardware implementation straightforward.

3.3.2 Multiplication

Multiplication is a little bit more complicated. To multiply two integers, we multiply their corresponding polynomials. If the resulting polynomial is of degree less than 4 then we got the polynomial representation of the result. Otherwise, the result is divided by the irreducible polynomial (x^4+x+1) to get the residue. Again, all additions and multiplications are done (modulo-2). The following are examples of multiplications done in this system:

$$2*13 = (x)(x^3+x^2+1) = (x^4+x^3+x) mod (x^4+x+1) = (x^3+1) = 9$$

$$3*10 = (x+1)(x^3+x) = (x^4+x^3+x^2+x) mod (x^4+x+1) = (x^3+x^2+1) = 13$$

Implementing multiplication in this system is no big deal !!.. The most straightforward implementation would be using table lookup. However, a more elegant implementation can be done using “shifts” and selective “exclusive-or’s”.

4 SETH functional units

Given the 2 hexadecimal digits from the input stream b_0, b_1, and given the 8 hexadecimal digits which define the transformation $A = [a_{ij}], 0 \leq i, j \leq 3$ described above, SETH should be able to compute 4 hexadecimal digits c_0, c_1, c_2, c_3 using the relation:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \\ a_{30} & a_{31} \end{pmatrix} \times \begin{pmatrix} b_0 \\ \\ \\ b_1 \end{pmatrix}$$

Moreover, using the appropriate inverted transformation $T = [t_{ij}], 0 \leq i, j \leq 1$, the data c_0, c_1 returned from any two different sinks can be recombined using the relation:

$$\begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{pmatrix} \times \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

In the above matrix operations, all the additions and multiplications are to be done using the “irreducible polynomial arithmetic” described in the previous section. It is obvious that the main functional units in SETH are the adder and multiplier (which themselves might be realized using other basic building blocks). By using an array of eight multipliers and by adding the approriate results together using 4 adders, a data path that implements matrix multiplication can be realized. To be able to use the same data path for both the read and write modes of SETH, some control logic needs to be added.

4.1 The Adder

Let $X = x_3x_2x_1x_0$ and $Y = y_3y_2y_1y_0$ be the binary representation of the two hexadecimal digits X, Y to be added. As we have stated earlier, addition using the "irreducible polynomial arithmetic" in the binary field Z_2, reduces to the bitwise "exclusive-or". Thus, the logical equations for the result $Z = z_3z_2z_1z_0$ is given by:

$$\begin{aligned} z_3 &= x_3 \oplus y_3 \\ z_2 &= x_2 \oplus y_2 \\ z_1 &= x_1 \oplus y_1 \\ z_0 &= x_0 \oplus y_0 \end{aligned}$$

4.2 The Multiplier

Let $X = x_3x_2x_1x_0$ and $Y = y_3y_2y_1y_0$ be the binary representation of the two hexadecimal digits to be multiplied, where the result is $Z = z_3z_2z_1z_0$.

First, we notice that multiplication of binary numbers actually reduces to successive shifts and adds; in our case exactly four shift/add stages are required (one stage for each bit of Y). In each of these stages, the accumulated value $W = w_3w_2w_1w_0$, initialy being 0000, is shifted left one bit and if the corresponding bit of Y is 1, X is added to the accumulated value and the result is propagated to the next "lower" stage. Of course all additions have to be done using the technique described above. Second we notice that the resulting number (using the described four-stage shift/add) cannot be used directly since the polynomial representation of this result might now be of the fourth degree (or higher) and the residue of this result should be computed using the "irreducible polynomial $x^4 + x + 1 (\equiv 10011)$. To compute this residue means that we need to perform successive subtractions. Fortunately, subtraction (modulo-2) is just the same as addition (modulo-2). Moreover, we note that these subtractions can be actually done within each stage of the above shift/add steps (and hence need not be accumulated till the end of the multiplication. Finally, we notice that at most one subtraction is needed within each stage of the multiplier since in each such stage, the degree of the accumulated result cannot be increased by more than one, and consequently, by subtracting the irreducible polynomial 10011 just once (whenever an overflow is detected) guarantees that the accumulated result will remain of the third degree (or less).

An algorithm to implement the above technique is shown in figure-3. In this algorithm, each iteration corresponds to one stage of the multiplier. Several optimizations can be applied to the algorithm. First, we notice that the test for whether y_i is equal to 0 or not and doing the shift-only or shift-and-add operation accordingly can be replaced by a shift and add to the bitwise product ("and") of y_i and X. Second, we notice that the test for whether a subtraction is needed or not can be replaced by always doing an "exclusive-or" of the result with w_4 (since if $w_4 = 0$ the "exclusive-or" won't change anything). Finally, we notice that $a \oplus 0 = a$. The optimized version of the algorithm is given in Figure-4. In Figure-5 we illustrate the use of this optimized and systematic method in computing the product of 1011 by 1100.

The above algorithm is the basis for our multiplier design. As a matter of fact the whole multiplier is constructed using four identical stages stacked together. Each stage accepts the accumulated value computed from the previous stage (stage #3 being fed with 0000). Also each stage accepts the value of X and one of the bits of Y. The result of the multiplication is the accumulated result from stage #0. The design of each of the multiplier stages requires exactly four "and" gates and five "exclusive-or" gates. A multiplier stage is shown in Figure-6 and the block diagram representation for a multiplier is shown in Figure-7.

4.3 The matrix multiplication unit

The matrix multiplication unit can be simply built using 8 multipliers to multiply in parallel the 2-element input vector with the corresponding elements of the transformation matrix, yielding 8 different products. Each couple of these products is added in parallel using a separate adder to produce the required 4-element output. Figure-8 shows the matrix multiplication unit.

4.4 Data flow Control

SETH is a bidirectional interface between two data streams. These streams are fed to SETH using two bidirectional ports B & C. Both the dispersal and retrieval modes of SETH involve matrix multiplication. This suggests that the same matrix multiplication unit could be used

```
Begin
    $w_4 = 0$ ;                                Initialize result
    $w_3 = 0$ ;
    $w_2 = 0$ ;
    $w_1 = 0$ ;
    $w_0 = 0$ ;
    for $(i = 3, i \geq 0, i--)$ {             For each stage
        if $(y_i = 0)$ {                       Is $y_i = 0$ ?
            $w_4 = w_3$ ;                      If Yes then just shift left
            $w_3 = w_2$ ;
            $w_2 = w_1$ ;
            $w_1 = w_0$ ;
            $w_0 = 0$ ;
        }
        else {
            $w_4 = w_3 \oplus 0$ ;             If No then shift left and add
            $w_3 = w_2 \oplus x_3$ ;
            $w_3 = w_1 \oplus x_2$ ;
            $w_3 = w_0 \oplus x_1$ ;
            $w_3 = 0 \oplus x_0$ ;
        }
        if $(w_4 = 1)$ {                       Is there a need to subtract ?
            $w_4 = w_3 \oplus 1$ ;             If Yes then do it !!
            $w_3 = w_3 \oplus 0$ ;
            $w_2 = w_2 \oplus 0$ ;
            $w_1 = w_1 \oplus 1$ ;
            $w_0 = w_0 \oplus 1$ ;
        }
    }                                          Done.
End
```

Figure 3: An algorithm for multiplication using irreducible polynomial arithmetic

```
Begin
    $w_4 = 0$ ;                                  Initialize result
    $w_3 = 0$ ;
    $w_2 = 0$ ;
    $w_1 = 0$ ;
    $w_0 = 0$ ;
    for ($i = 3, i \geq 0, i--$) {               For each stage
        $w_4 = w_3$ ;                            Add $X$ if $y_i = 1$ and add 0 otherwise
        $w_3 = w_2 \oplus (x_3.y_i)$ ;
        $w_3 = w_1 \oplus (x_2.y_i)$ ;
        $w_3 = w_0 \oplus (x_1.y_i)$ ;
        $w_3 = (x_0.y_i)$ ;
        $w_1 = w_1 \oplus w_4$ ;                 Adjust result (if necessary)
        $w_0 = w_0 \oplus w_4$ ;
    }                                            Done.
End
```

Figure 4: Optimized multiplication algorithm using irreducible polynomial arithmetic

```
0 0 0 0              ; Stage #3
  1 0 1 1
  -------
  1 0 1 1
      0 0
  -------
  1 0 1 1            ; Stage #2
    1 0 1 1
    -------
    1 1 0 1
        1 1
    -------
    1 1 1 0          ; Stage #1
      0 0 0 0
      -------
      1 1 0 0
          1 1
      -------
      1 1 1 1        ; Stage #0
        0 0 0 0
        -------
        1 1 1 0
            1 1
         -------
        1 1 0 1      ; Result
```

Figure 5: The multiplication of 1011 by 1100 using the irreducible polynomial arithmetic

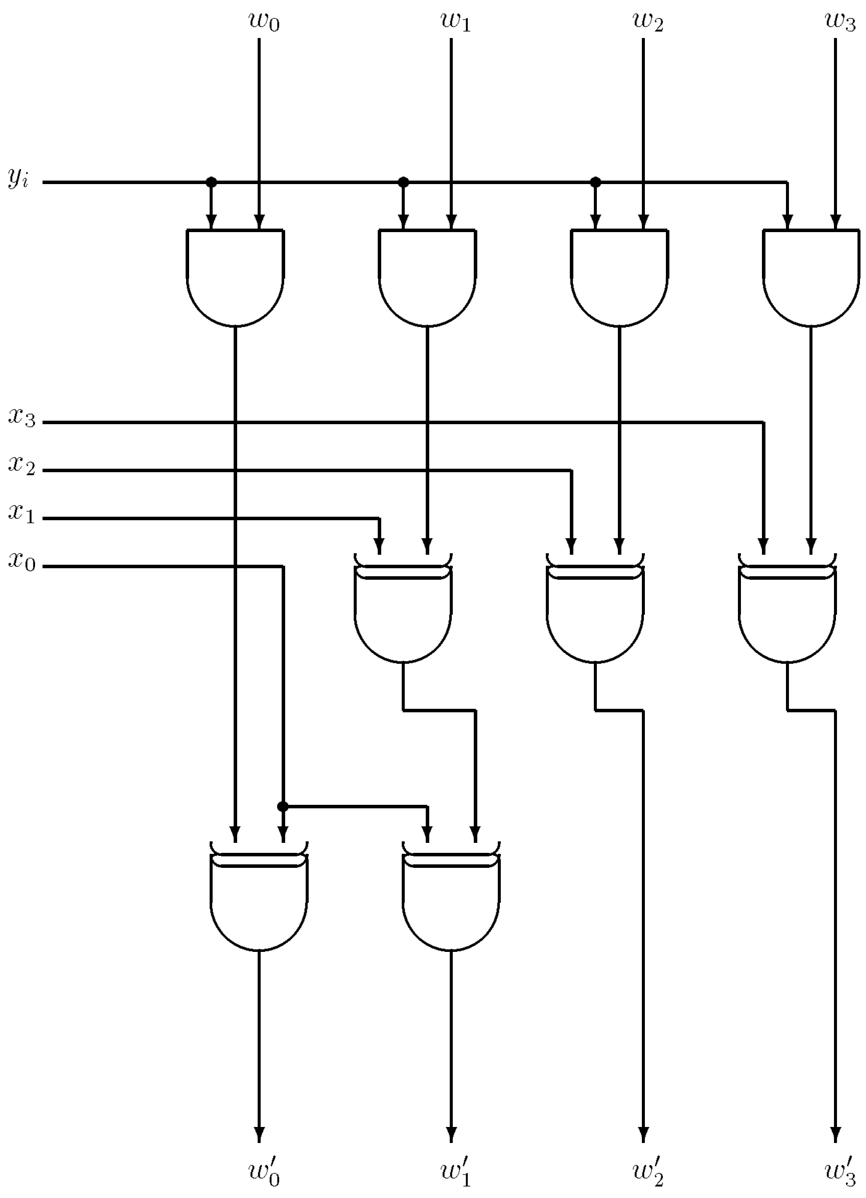

Figure 6: The mult-stage

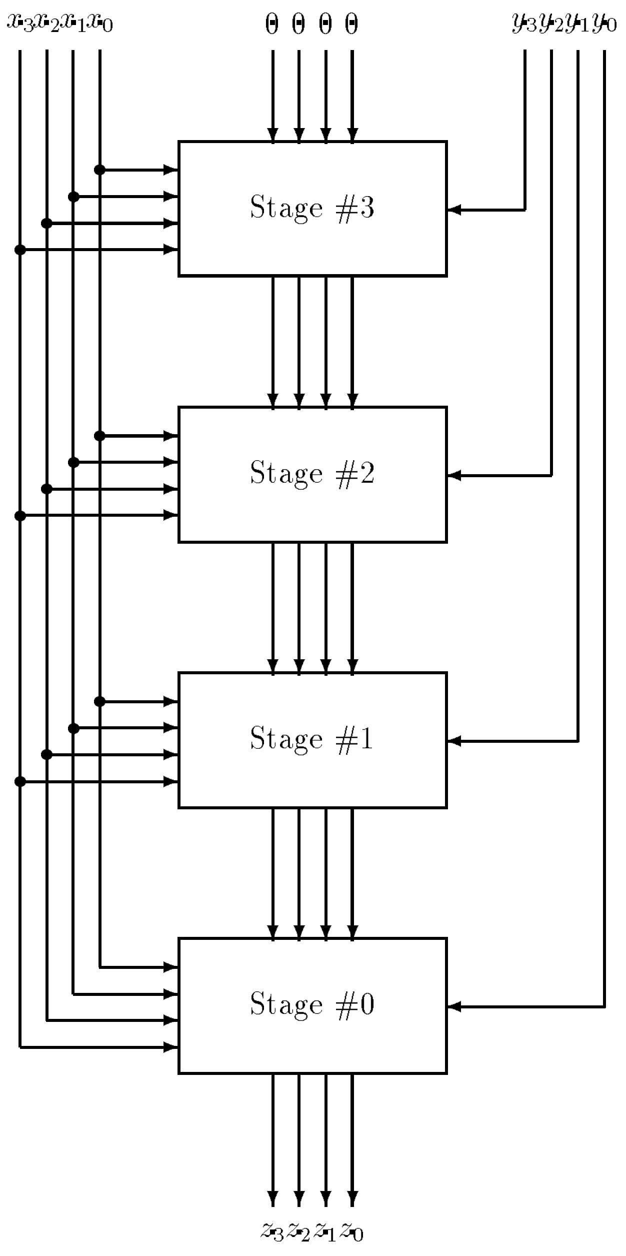

Figure 7: The multiplier functional unit

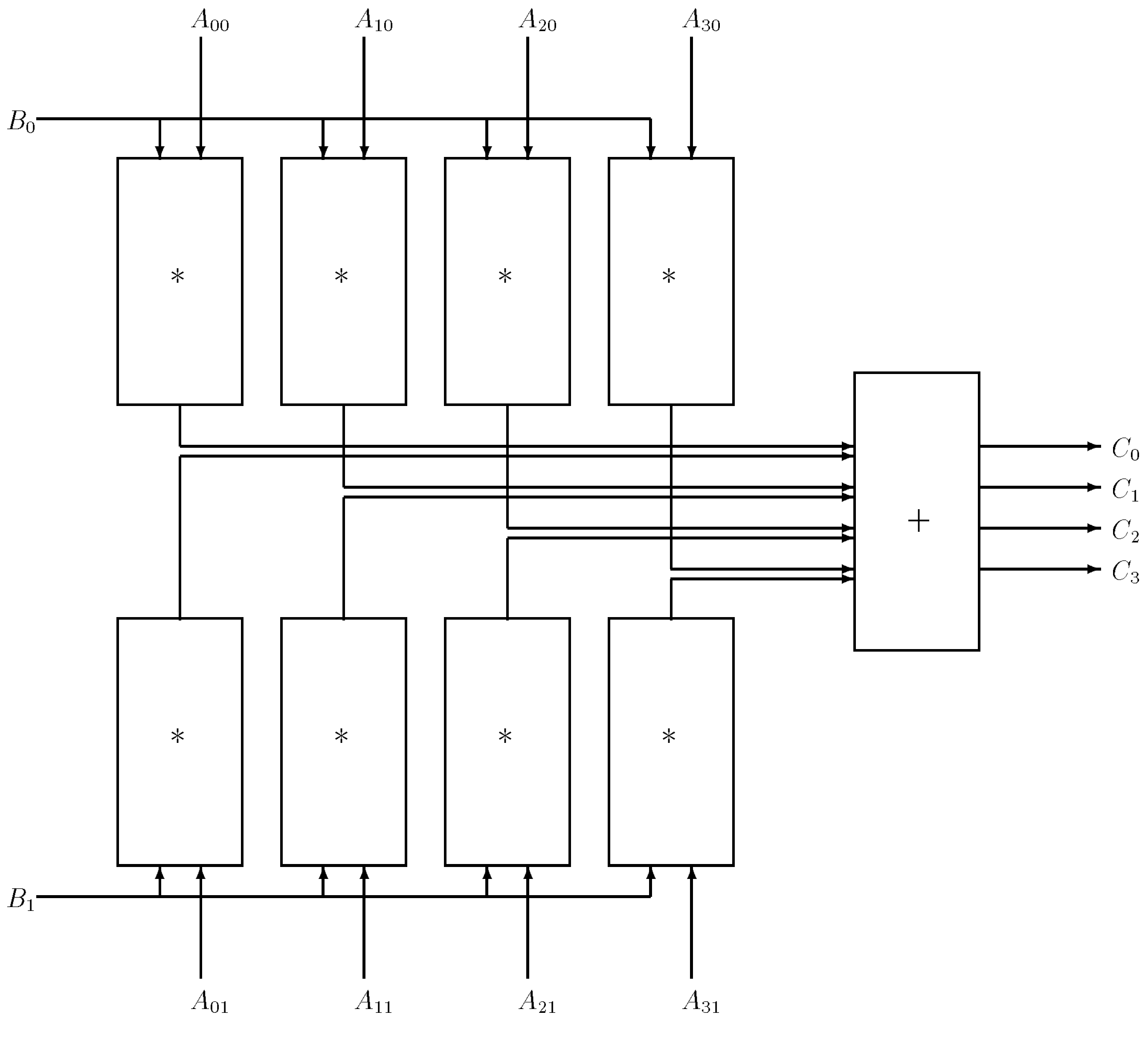

Figure 8: The matrix multiplication unit

for both modes with the appropriate control to forward the data in and out from the matrix multiplication unit.

In write-mode (dispersal operation), SETH accepts 8-bit inputs (2 hexadecimal digits) from its B-port. These inputs, as well as the 32-bit (8 hexadecimal digits) transformation matrix are forwarded to the matrix multiplication unit to produce the 16-bit (4 hexadecimal digits) output which is forwarded to SETH's C-port. In read-mode, SETH accepts 16-bit inputs (4 hexadecimal digits) from its C-port. Depending on the control lines applied, only 2 hexadecimal digits from these 4 are forwarded to the matrix multiplication unit along with the appropriate inverted secret keys provided to the chip. Finally 8 of the 16-bit output produced are routed to SETH's B-port (since only half of the matrix multiplication unit is used in this case).

5 Applications for SETH

There are basically two areas where SETH may be used; Data storage and retrieval, and Data communication. In a storage system, SETH would be located between the storage device and the data bus. In the case of a communication system, SETH would be placed at each end of a SETH bus with the system bus at each of the remaining ends. A benefical side effect of using IDA is improved load balancing in both storage and communication.

5.1 Data Storage and Retrieval using SETH

The integrety of stored information could be improved greatly by using SETH. Mechanical storage devices are the weakest components in a computer system due to their intolerance to shocks, vibration, dust, and their inherently unreliable moving parts.

Two SETH chips configured in parallel, and three disks will make a fault tolerant system with two times the storage of a single disk. In addition the system is secure from information thefts. During normal operation, data will be read from any two of the three drives on the system, while data (encrypted and dispersed using SETH) will be written to all three disks at one time. In the case of a drive failure (or even a bad track or sector on a single drive), the SETH chips will read data from the remaining two good drives until the bad drive can be replaced (or reformatted). When a new drive is installed, reading and writting all data back to the three disks will result in an initialization of the newly installed disk. It is important to realize that if the failure modes of the three disks is independent (which is normally the case), then the probability that the SETH-based design will fail is extremely small. For instance assume that the probability of loosing a specific track (or sector) is P (P is usually in the order of 10^{-6}). The SETH-based design will fail to read a specific track if and only if the *same* track (or sector) in two or more of the disks will be lost. This probability can be shown to be $3P^2(1-P)$ (in the order of 10^{-12}). A detailed analysis of the potential gains are discussed in [BCW:89].

Data storage and retrieval using SETH is secure as well. When the system needs to be secured, the three disks can be locked in three diffrent places (maybe under different machines – or at different sites). An adversary will need to access at least two of the disks (assuming that he knows what SETH is really doing !!) before he can retrieve any "meaningful" information. In addition, to do that, the conversion matrix used for dispersal needs to be known. This makes it more difficult especially if such information is generated randomly by the operating system or even the underlying hardware.

The use of the Information Dispersal Algorithm in the design of RAID systems (Redundant Arrays of Inexpensive Disks) has been investigated in [BCW:89]. It has been demonstrated that such an approach is superior to previously suggested techniques, namely shadowing and parity [Patterson:89].

5.2 Data communication using SETH

Placing SETH chips on both sides of an information bus will increase the reliability of the bus and make it harder for information thieves to tap the information thereon. For example, one SETH chip may interface with an 8-bit data bus and send 16 bits to the other end, where another SETH chip would be used to recombine the data. The security of the bus results from the difficulty to decipher the information. The coding matrices ("secret keys") used in SETH can be changed at frequent intervals to make deciphering still more difficult. The bus is highly fault-tolerant since the original information may be reconstructed from any two of the four groups of nibbles available on the bus.

In [Rabin:87], IDA has been used in routing packets on cube-based architectures. The suggested technique is fully described in that paper. An obvious extension of this work would be to consider network topologies other than the n-cube. Also, fine tuning the routing algorithms to the system parameters[4] is another interesting problem.

[4]number of processors, size of packets, failure rates, ..., *etc.*

6 Conclusion

In this paper, we have presented "SETH" – a hardwired implementation of the Information Dispersal Algorithm. SETH allows the real-time dispersal of information into different pieces as well as the retrieval of the original information from the available pieces. SETH accepts a stream of data and a set of "secret keys" so as to produce the required streams of dispersed data to be stored on (or communicated with) the different machines. SETH might as well accept the streams of data from the different machines along with the necessary controls and keys so as to reconstruct the original information. The design of SETH involved finding efficient techniques for computing using irreducible polynomial arithmetic. In particular, we have outlined general methods for addition and multiplication in these systems. SETH was designed using the Berkeley MAGIC layout software. The design was tested using ESIM and SPICE simulators [BMS:88]. The chip was fabricated by MOSIS on a 3-micron SCMOS-technology process. Twelve packages were returned and the functionality of the chip was verified.

The design of SETH can be extended in several ways. In our current implementation, an outside mechanism is assumed to be responsible for providing the secret keys to be used in the dispersal of the original information as well as providing the inverse keys to be used in the retrieval of the dispersed information. The outside mechanism, however, can be relieved from the burden of computing the inverse keys if the design is made so that the inverse keys are computed on the fly (given the original secret keys and the set of intact sinks). This is quite feasible. Moreover, the possibility of "automatically" generating the secret keys for each file (or set of packets) is very attractive. In this case, information about these keys has to be included in the dispersed data so as to be used later in the computation of the inverse keys.

Our choices for n (the number of sinks), m (the minimum number of sinks required to reconstruct F) and k (the character size) can also be changed. It can be shown that the size of the chip scales linearly with $n \times m \times k^2$ and that the propagation delay scales linearly with k. Increasing n and m would result in a more flexible design in terms of the achievable levels of redundancy and fault-tolerance. On the other hand, increasing the value of k would significantly enhance the security of the dispersal algorithm at the expense of a blowup in the size of the chip as well as an increase in the propagation delay. The increase in the propagation delay is not a critical factor, since by using pipelining the overall delay in communicating messages (especially long ones) can be downplayed. The number of pins required for the chip is another crucial factor. In particular, if the chip is to be used for parallel communication, then the number of pins required for data i/o is $(n+m) \times k$. For large values of n, m, and k, serial communication is likely to be used. Still, another alternative would be to partition the computations so as more than one chip could be used in parallel.

The potential applications of IDA are numerous. In particular, the use of IDA in I/O systems, RAID designs, distributed communication and routing is promising. SETH demonstrates that using IDA in these applications is feasible.

References

[BMS:88] Azer Bestavros, Steve Morss, and Adam Strassberg, *SETH: a chip for reliable and secure communication using the Information Dispersal Algorithm* Internal report, VLSI Lab., Harvard University - May 1988.

[Bestavros:89] Azer Bestavros, *SETH: A VLSI chip for the real-time Information Dispersal and retrieval for security and fault-tolerance* Technical Report, TR-06-89, Harvard University - January 1989.

[BCW:89] Azer Bestavros, Danny Chen, and Wing Wong, *The reliability and performance of Redundant Arrays of Inexpensive Disks* Technical Memorundum, AT&T Bell Labs (Holmdel) – To appear.

[Gibson:88] Garth Gibson, Lisa Hellerstein, Richard Karp, Randy Katz and David Patterson, *Coding Techniques for Handling Failures in Large Disk Arrays,* Technical Report UCB/CSD 88/477, Computer Science Division, University of California, July 1988.

[Patterson:89] David A. Patterson, Peter Chen, Garth Gibson, and Randy H. Katz, *Introduction to Redundant Arrays of Inexpensive Disks (RAID),* COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference, March 1989.

[Rabin:87] Michael O. Rabin, *Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance* Technical Report, TR-02-87, Department of Computer Science, DAS, Harvard University - April 1987. Also appeared in the Journal of the Association for Computing Machinery, Vol. 36, No. 2, April 1989, pp. 335-348. April 1989.

[Shamir:79] A. Shamir, *How to share a secret ?*, Communication of the ACM 22, 11 (Nov. 1979), 612-613. November 1979.

APPENDICES

A Splitting and Recombining Files using IDA

Let F be the file we want to disperse. Let $F = b_1 b_2 b_3 b_4 \ldots\ldots etc.$, where b_i is an integer taken from a certain range $[0..(2^r - 1)]$. In particular, for the intended design b_i is a hexadecimal digit ($r = 4$) and the range is $[0..15]$. Let p be a prime number greater than any possible b_i. In our case p might be selected to be 17. We might view any b_i as an element of the finite field Z_p (Z_{17} in our case), where all operations (namely addition and multiplication) are done in $mod(p)$. Hence, for $p = 17$, $7 + 12 = 2$ (since $7 + 12 = 19$ which is 2 in Z_{17}), and $7 * 3 = 4$ (since $7 * 3 = 21$ which is 4 in Z_{17}).

In order to split F, we choose a set of n vectors $(V_1, V_2, ..., V_n)$ each of length m and whose elements are from Z_p. Any subset of m of these vectors should be linearly independent. These vectors represent the "keys" that will be used to disperse F. Let $A_{n.m}$ be the array whose rows are the selected vectors. Hence A represents a mapping from an m-dimensional space to an n-dimensional space, or in other words, from a sequence of m elements to another sequence of n elements (all the elements are from Z_p). To disperse the file F, we simply map each sequence of m elements from F into a new sequence of n elements using the transformation A. Each element from the resulting sequence is sent to a different site and kept there. So, for each m elements of F we send *one* element to each of the n sites. Therefore, to disperse the whole file F we will need to send $\frac{|F|}{m}$ elements to each of the n sites.

Now suppose that we want to reconstruct the original file F from the pieces dispersed as described above. This is done by reading any m of these pieces (if less than m pieces are available then the file cannot be reconstructed, and if more than m pieces are available then the use of any subset of m pieces will suffice). Let the pieces be from sites $s1, s2, s3, ..., sm$. Let $B_{m.m}$ be the array whose rows are $(V_{s1}, V_{s2}, V_{s3}, ..., V_{sm})$. Thus, B maps sequences of m elements from F into sequences of m elements, which by virtue of the dispersion step above, are kept at sites $s1, s2, s3, ..., sm$. Now, to reconstruct the first m elements of F we need simply to collect the first element from each of m different sites and use the appropriate inverse transformation ($T = B^{-1}$). Note that such an inverse is guaranteed to exist since by hypothesis, the subset of m key-vectors is necessarily independent.

- **An example**

We are going to use $n = 4$, $m = 2$, $p = 17$. Furthermore, let $F = 2, 4, 1, 14, 6, 8, \ldots\ldots etc.$ and assume that we selected the secret key-vectors to be:-

$$V_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$V_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$V_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$V_3 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Hence, the transformation A is:-

$$A = \begin{pmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}$$

To disperse F we divide it into sequences of 2 elements as shown below:-

$$F = \underbrace{2, 4,} \underbrace{1, 14,} \underbrace{6, 8,} \ldots\ldots etc..$$

Next, each of these sequences is transformed using A and we obtain the new sequences shown below (note that all arithmetic is done (modulo 17):-

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \\ 10 \\ 14 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 14 \end{pmatrix} = \begin{pmatrix} 1 \\ 15 \\ 14 \\ 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 6 \\ 8 \end{pmatrix} = \begin{pmatrix} 6 \\ 14 \\ 5 \\ 13 \end{pmatrix}$$

Hence the resulting sequence will be:-

$$F' = \underbrace{2, 6, 10, 14}, \underbrace{1, 15, 14, 9}, \underbrace{6, 14, 5, 13}, \ldots\ldots etc..$$

Hence the first site will be given the 1^{st} element from each resulting sequence, i.e. $(2, 1, 6, \ldots\ldots etc.)$ Similarily, the second site will be given the 2^{nd} element from each resulting sequence, i.e. $(6, 15, 14, \ldots\ldots etc.)$, $\ldots\ldots etc.$

$$\begin{aligned} F'_1 &= 2, 1, 6, \ldots\ldots etc. \\ F'_2 &= 6, 15, 14, \ldots\ldots etc. \\ F'_3 &= 10, 14, 5, \ldots\ldots etc. \\ F'_4 &= 14, 9, 13, \ldots\ldots etc. \end{aligned}$$

Now, suppose that the second and fourth sites fail (or an error has been detected in their data) and we want to reconstruct the original file F. This is done by first computing the transformation B which consists of the key-vectors for the available sites (namely the first and third) as shown below:-

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$$

Second, we compute the inverse of this transformation $T = B^{-1}$ as follows:- (remember that all arithmetic is in (modulo 17) including the reciprocal)

$$T = B^{-1} \begin{pmatrix} 1 & 0 \\ 8 & 9 \end{pmatrix}$$

Now, to reconstruct the first sequence of m elements from the original file, we transform the sequence consisting of the first element from the first and third sites, namely $(2, 6)$, using T as follows:-

$$\begin{pmatrix} 1 & 0 \\ 8 & 9 \end{pmatrix} \times \begin{pmatrix} 2 \\ 6 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

Thus obtaining the first two elements $(2, 4)$ of the original file.

Similarily, to get the next two elements of F, we transform the sequence consisting of the second element from the first and third sites, namely $(6, 14)$, using T as follows:-

$$\begin{pmatrix} 1 & 0 \\ 8 & 9 \end{pmatrix} \times \begin{pmatrix} 6 \\ 14 \end{pmatrix} = \begin{pmatrix} 1 \\ 14 \end{pmatrix}$$

Thus obtaining the third and fourth elements of F, namely $(1, 14)$ and the process continues for the rest of the file.

B Using Irreducible Polynomials for IDA

In the previous section, we have assumed that the elements of the file F might be considered integers from a finite range $[0-(2^r-1)]$. Thus for $r=4$, we have integers ranging from 0 to 15. The algorithm however, requires computations to be done in $mod(p)$ where $p > (2^r-1)$. Thus for $r=4$, the minimum p is 17. Obviously, 2^r (for any r) is not prime. Thus, it is possible during the intermediate computations (as well as the final result) to get integers $> (2^r-1)$ and thus requiring more than r bits to be represented and stored. In the case of $r=4$, the added bit will only be used to represent the integer 16 and obviously results in a waisted space on each site. In particular, instead of increasing the redundancy by a factor of $(\frac{n}{m}-1)*100\%$, we will actually increase it by a factor of $\frac{5}{4}(\frac{n}{m}-1)*100\%$.

To overcome this difficulty, instead of representing integers as residues in the field of the prime p, we represent these integers as polynomials with binary coefficients (i.e. polynomials $\in Z_2[x]$) and use an irreducible polynomial of a larger degree (instead of the prime p) to perform the arithmetic. For instance, if $a_3a_2a_1a_0$ is the binary representation of a hexadecimal digit, we view it as a polynomial $a_3x^3+a_2x^2+a_1x+a_0$. Now, in order to perform any arithmetic on such polynomials, we must find a 4^{th} degree polynomial that is irreducible (analoguous to the prime number used before). An irreducible polynomial of degree r is one that cannot be divided by any other polynomial of lesser degree. Once this is found, we perform all the computations using the above polynomials in modulo-2 arithmetic, but representing any results by their residue when divided by the irreducible polynomial. For the case where $r=4$, it is easy to show that the polynomial (x^4+x+1) is irreducible. We have used this polynomial in our SETH implementation. This polynomial has the binary representation 10011. The following is a list of the diffrent polynomial representation for the digits [0..15]. These are actually all the possible polynomials of third degree (or less) that have binary coefficients.

$$\begin{pmatrix} 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0100 \\ 0101 \\ 0110 \\ 0111 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ x \\ x+1 \\ x^2 \\ x^2+1 \\ x^2+x \\ x^2+x+1 \end{pmatrix} \qquad \begin{pmatrix} 1000 \\ 1001 \\ 1010 \\ 1011 \\ 1100 \\ 1101 \\ 1110 \\ 1111 \end{pmatrix} = \begin{pmatrix} x^3 \\ x^3+1 \\ x^3+x \\ x^3+x+1 \\ x^3+x^2 \\ x^3+x^2+1 \\ x^3+x^2+x \\ x^3+x^2+x+1 \end{pmatrix}$$

- **An example**

We are going to use $n = 3$, $m = 2$, and the irreducible polynomial $(x^4 + x + 1)$. Let $F = 2, 4, 1, 14, 6, 8, \ldots\ldots etc..$ (as before) and assume that we selected the secret key-vectors to be:-

$$V_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$V_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$V_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$V_3 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Hence, the transformation A is:-

$$A = \begin{pmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}$$

To disperse F we divide it into sequences of 2 elements as shown below:-

$$F = \underbrace{2, 4,} \underbrace{1, 14,} \underbrace{6, 8,} \ldots\ldots etc..$$

Next each of these sequences is transformed using A and we obtain the new sequences shown below (note that all arithmetic is done using the irreducible polynomial $(x^4 + x + 1)$:-

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \\ 10 \\ 14 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 14 \end{pmatrix} = \begin{pmatrix} 1 \\ 15 \\ 14 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \times \begin{pmatrix} 6 \\ 8 \end{pmatrix} = \begin{pmatrix} 6 \\ 14 \\ 5 \\ 13 \end{pmatrix}$$

Hence, the resulting sequence will be:-

$$F' = \underbrace{2, 6, 10, 14}, \underbrace{1, 15, 14, 0}, \underbrace{6, 14, 5, 13}, \ldots\ldots etc..$$

Hence, the first site will be given the 1st element from each resulting sequence, i.e. $(2, 1, 6, \ldots\ldots etc.)$ Similarily, the second site will be given the 2nd element from each resulting sequence, i.e. $(6, 15, 14, \ldots\ldots etc.)$, $\ldots\ldots etc.$

$$F_1' = 2, 1, 6, \ldots\ldots etc.$$

$$F_2' = 6, 15, 14, \ldots\ldots etc.$$

$$F_3' = 10, 14, 5, \ldots\ldots etc.$$

$$F_4' = 14, 9, 13, \ldots\ldots etc.$$

Now, suppose that the second and fourth site fail and we want to reconstruct the original file F. This is done by first computing the transformation B which consists of the key-vectors for the available sites (namely the first and third) as shown below:-

$$B = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$$

Second, we compute the inverse of this transformation $T = B^{-1}$ as follows (this inverse is w.r.t. the irreducible polynomial 10011):-

$$T = B^{-1} = \begin{pmatrix} 1 & 0 \\ 9 & 9 \end{pmatrix}$$

Now, to reconstruct the first sequence of m elements from the original file, we transform the sequence consisting of the first element from the first and third sites, namely $(2, 6)$, using T as follows:-

$$\begin{pmatrix} 1 & 0 \\ 9 & 9 \end{pmatrix} \times \begin{pmatrix} 2 \\ 6 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

Thus obtaining the first two elements $(2, 4)$ of the original file. Similarily, to get the next two elements of F, we transform the sequence consisting of the second element from the first and third sites, namely $(6, 14)$, using T as follows:-

$$\begin{pmatrix} 1 & 0 \\ 9 & 9 \end{pmatrix} \times \begin{pmatrix} 6 \\ 14 \end{pmatrix} = \begin{pmatrix} 1 \\ 14 \end{pmatrix}$$

Thus obtaining the third and fourth elements of F, namely $(1, 14)$ and the process continues for the rest of the file.