cs 591B1: ALGORITHMIC ASPECTS OF COMPUTER NETWORKS	Spring 2002

Lecture 6 — February 25, 2002

Lecturer: John Byers *Scribe: Ryan Mahon*

In today's lecture we examined a method for information dispersal of a file with consideration to security, load balancing, and fault tolerance. First we solidified our background by examining Shamir's Secret Sharing Algorithm and then we went on to discuss Rabin's information dispersal methodology.

6.1 Shamir's Secret Sharing Algorithm

Motivating Example: How do you create and disperse the keys needed to launch the nuclear weapons of your country? You do not want the loss of one of the keys to impede your country's defense, nor do you want somebody who acquires just a few of the keys to be capable of launching your weapons.

Suppose we have a certain piece of data D (the launch code in the above example). We wish to separate D into n pieces such that any K of the n pieces is sufficient to recover D, but no information is revealed from any $K-1$ of the keys. This can be accomplished leveraging a basic mathematical principle known as Polynomial Interpolation.

6.1.1 Polynomial Interpolation

Given any set of pairs $\{(x_0, f(x_0))(x_2, f(x_2))...(x_k, f(x_k)\}$ there exists a **unique** polynomial $P(x)$ of degree $k-1$ which interpolates the tuples. For instance, examining figure 1, we see that there is only one P of degree four, the one drawn, that will fit the five points shown. Note that any k of the tuples will yield exactly one polynomial of degree $k-1$, but $k-1$ of the tuples yield an infinite number of polynomials of degree $k-1$. We will use this premise to disperse our secret.

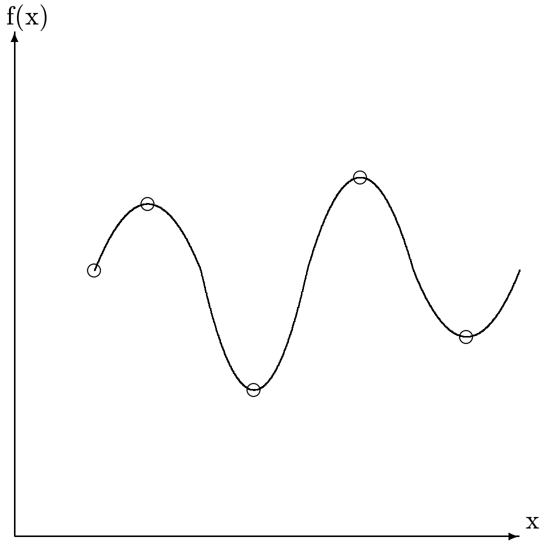

Figure 6.1. Polynomial Graph

6.1.2 A Simple Numerical Secret

Let us create our first secret now. We will let D, an integer, be our secret. First create a secret polynomial, $P(x) = A_0 + A_1x + A_2x^2 + ...$ of degree $k-1$ (k can been viewed as a security parameter, the bigger it is, the more points will be needed to discover D). The first coefficient(a_0) should be set to D and the remaining coefficients should be randomly and independently chosen.

Example 6.1

- $k = 2$: Two tuples will be required to discover the secret
- $D = 51$: Our secret
- $P(x) = 51 + 3x$: Our secret polynomial (Graph Shown in Figure 6.2)

We can generate an infinite number of tuples $(x, P(x))$ based on our secret polynomial (i.e. $(1,54)(2,57)(3,60)...$). Any two of these tuples will suffice to construct our polynomial (in this case a line), but no information can be gathered about our tuple from only one tuple. This follows from the geometric idea that any two points define exactly one line in the plane.

Note: If you always intend to use a_0 as your code word, then never generate $P(0)$ or you will have given away your code word!

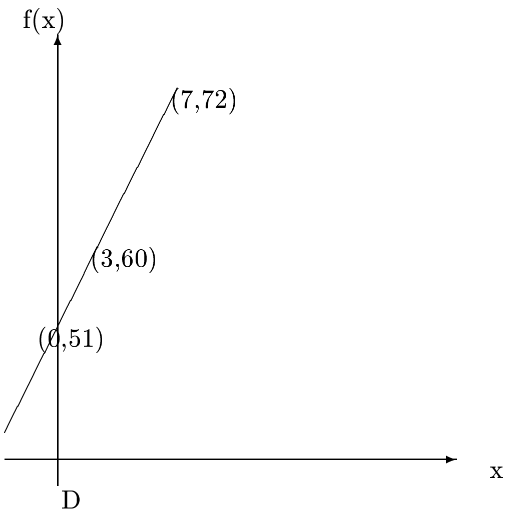

Figure 6.2. Simple Secret Graph

6.1.3 Lagrange Interpolation: Retrieving $P(x)$

To decode the polynomial $P(x)$ of degree d once we have all the required tuples $(a_0, b_0), (a_1, b_1), ..., (a_{d-1}, b_{d-1})$, we use equation 6.1. This is known as Lagrange Interpolation. A short example is shown below.

$$P(x) = \sum_{j=0}^{d-1} b_j \prod_{j \neq k} \left(\frac{x - a_k}{a_j - a_k}\right) \tag{6.1}$$

Example 6.2

- $d = 3$: Our polynomial is quadratic

- $(0,2)(1,0)(3,2)$: These are the three tuples we have received
- $P(x) = 2(\frac{x-1}{0-1})(\frac{x-3}{0-3}) + 0(\frac{x-0}{1-0})(\frac{x-3}{1-3}) + 2(\frac{x-0}{3-0})(\frac{x-1}{3-1}) = x^2 - 3x + 2$: Note that $P(a_i) = b_i$ so $P(x)$ works!

6.1.4 Polynomial Interpolation in a Finite World

How do we do polynomial interpolation in finite terms?

- Let
  - D be the data to be coded
  - n be the number of tuples to be generated
  - P be the polynomial
  - k be the degree of the polynomial P
- Pick a prime number p that is greater than both D and n. We will then work our arithmetic over Z_p (i.e. take everything mod p). Note:
  - p must be prime so that Z_p will be a field.
  - p must be greater than n so that we have distinct pieces. Otherwise we may not be able to discover the secret with $k-1$ pieces.
  - p must be greater than D so that our secret is not corrupted by the modulus operation.
- Note that polynomial interpolation still applies, although we now know that there are only p possible secrets.

6.2 Rabin's Method for Information Dispersal

Motivation: We want to be able to divide a file into pieces to provide for fault tolerance, load balancing, and security. Rabin's algorithm builds off of Shamir's ideas

6.2.1 Rabin's Idea

- Let
  - F be a file
  - L be the length of F (packets, bytes, etc.)
  - M be the number of pieces needed to reconstruct F (also the number of characters in each block)
  - N be the number of pieces that F will be split into
  - p will be a prime number larger than the coded characters of the file (i.e. 256 possible characters, use 257 or greater)
  - $\frac{L}{M}$ be the number of blocks of F
- We will manipulate F into N pieces of length $\frac{L}{M}$ such that any M pieces suffice to recover F.
- We will take F as a string of residues mod p: [1] $(f_1, ..., f_M), (f_{M+1}, ..., f_{2M}), ...(..., f_L)$
- We will take $N = M + K$ such that $\frac{N}{M} \leq 1 + \epsilon$ for some $\epsilon > 0$ [2]

[1] each block is a string of elements in the finite field Z_p

[2] the stretching is bounded

6.2.2 Encoding

Let us suppose that we have a special $N \times M$-matrix A that is decided in advance. We will discover how to create A and the properties it must have later on. Now, using matrix multiplication, let us multiply the matrix A by a $M \times 1$ matrix B where B's components consist of the elements of a block (segmented sequence) of F. (i.e. $(f_{iM+1}, ..., f_{iM})$) This matrix multiplication will produce a coded matrix C as illustrated in figure 6.3. We will repeat this multiplication for all $\frac{L}{M}$ of the blocks of F to arrive at the C matrix shown in figure 6.4. Each row of this matrix represents a coded piece of our file and will be a packet. Note that there are exactly N pieces and each piece is of size $\frac{L}{M}$.

$$[\ A\ ] * \begin{bmatrix} f_1 \\ f_2 \\ ... \\ f_M \end{bmatrix} = \begin{bmatrix} c_{1,1} \\ c_{2,1} \\ ... \\ c_{M,1} \end{bmatrix} \qquad [\ A\ ] * \begin{bmatrix} f_{M+1} \\ f_{M+2} \\ ... \\ f_{2M} \end{bmatrix} = \begin{bmatrix} c_{1,2} \\ c_{2,2} \\ ... \\ c_{M,2} \end{bmatrix}$$

Figure 6.3. The Coding Process

$$C = \begin{vmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,L/M} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,L/M} \\ ... & ... & ... & ... \\ c_{N,1} & c_{N,2} & \cdots & c_{N,L/M} \end{vmatrix}$$

Figure 6.4. The Matrix C: Each of the N rows is a packet (M required to reconstruct the file)

6.2.3 Decoding

Claim: Any M of the pieces suffice to recover all the f_is

Let us now assume that of the N rows(packets), M have been recovered. We then reconstruct the matrix C from figure 6.4 ignoring the missing $(N - M)$ rows to form the matrix C'. Next we remove the corresponding missing rows from the matrix A which will now become an $M \times M$ matrix which we shall term A'. We then take the matrix inverse of A' to form the matrix $A^{'-1}$. We use $A^{'-1}$ to solve for the F matrix as shown in equation 6.2. Note that in order for this to be possible, A' must be invertible. This means that the determinant of A' cannot be zero and that the columns of A' must be linearly independent. We will have to keep this in mind when we decide how to select A, which we describe later.

$$\begin{aligned} A' * F &= C' \\ A^{'-1} * A' * F &= A^{'-1} * C' \\ F &= A^{'-1} * C' \end{aligned} \tag{6.2}$$

6.2.4 Creating the Matrix A

Creating the matrix A is left to our discretion with one stringent requirement: The matrix A and any possible $M \times M$ submatrix of A must have linearly independent columns and thus be nonsingular. This is a requirement so that A' will be invertible and so equation 6.1 can be solved. We will examine two methods to generate A. The elements of A will be denoted as a_{ij} where i is the row number and j is the column number.

The first method will be to set each of the a_{ij} using equation 6.3 with the requirements in equation 6.4. Note that both x_i and y_j are elements in the integer field of size p. It is easy to prove the determinant of such a matrix is non-zero. Note the amount of calculation required to create such a matrix.

$$\begin{aligned} a_{ij} &= 1/(x_i + y_j) && (6.3) \\ x_i + y_j &\neq 0 && (6.4) \\ i &\neq j \\ x_i &\neq x_j \\ y_i &\neq y_j \end{aligned}$$

A simpler method would be to randomly choose the a_{ij}. Certainly it is no longer true that any $M \times M$ submatrix is invertible. However it is claimed that with high probability that a randomly generated matrix will be invertible. (The proof was given as an exercise and the solution is located in section 6.2.5). The exact probability bounds are given in equation 6.5 Note that for a field size of greater than 100 there is less than a one percent chance of selecting a singular submatrix!

$$1 - (\frac{1}{p})(\frac{p}{p-1}) \leq \{Pr|A| \neq 0\} \leq 1 - \frac{1}{p} \tag{6.5}$$

Note the above A's will take on average cubic time to calculate A's inverse. For $M = 14$, $N = 10$ this is acceptable, but not when M and N grow as moderately large as 1000. Questions to be studied in future lectures include if there is a better A and if there is a more efficient procedure for encoding and decoding.

6.2.5 Is a Random Matrix Nonsingular?

The following claim from the Rabin paper was assigned as a homework exercise. This proof was distilled from solutions by John Byers and Jef Considine.

To show: Given a square matrix A in which the entries are chosen uniformly at random from Z_p,

$$1 - \frac{1}{p-1} \leq \Pr[A : |A| \neq 0] \leq 1 - \frac{1}{p}.$$

Proof: We start with analysis relevant to both inequalities. Define the event C_i to be the bad event that the first $i-1$ rows of the matrix are linearly independent, but the ith row is linearly dependent on the first i rows. The probability that a random $m \times m$ matrix is non-singular is equivalent to the probability that *none* of the bad events C_i occur, i.e.: $\Pr[A : |A| \neq 0] = 1 - \Pr\left[\bigcup_{i=1}^m C_i\right]$.

The key to the analysis is the following lemma.

Lemma 6.1. $\Pr[C_{i+1}] = \frac{p^i}{p^m}$.

Proof: (of Lemma) Assume that there are i linearly independent rows present in the matrix. Then there are p^i possible linear combinations of these rows, since each row can be multiplied by any of the p possible coefficients from Z_p before these i rows are added together. (Note that this statement also holds for the all zeroes case at row 1). Moreover, each of these p^i linear combinations produce *distinct* vectors. If not, then there exist distinct sets of coefficients $\{a_j\}$ and $\{b_j\}$ which when applied to row vectors x_j yield $\sum_{j=1}^i a_j x_j = \sum_{j=1}^i b_j x_j$. This further implies that there exists a set of coefficients $\{c_j\}$, where $c_j = a_j - b_j$, such that $\sum_{j=1}^i c_j x_j = \vec{0}$. Since the set $\{c_j\}$ is not all zero, this contradicts the assumption that these rows are linearly independent. Therefore, the probability that the $(i+1)$st row is linearly dependent on the first i (linearly independent) rows is exactly $\frac{p^i}{p^m}$ when rows are chosen uniformly at random, i.e. from a set of p^m possibilities. □

Returning to the main claim, to prove the right inequality, we have that $\Pr\left[\bigcup_{i=1}^m C_i\right] \geq \Pr[C_m] = \frac{1}{p}$. Therefore, $\Pr[A : |A| \neq 0] = 1 - \Pr\left[\bigcup_{i=1}^m C_i\right] \leq 1 - \frac{1}{p}$.

Now we work toward the lower bound expressed by the left inequality. Using a union bound, i.e. for any set of events E_i, $\Pr[\bigcup_{i=1}^m E_i] \leq \sum_{i=1}^m \Pr[E_i]$, we have that $\Pr[\bigcup_{i=1}^m C_i] \leq \sum_{i=0}^{m-1} \frac{p^i}{p^m}$. This sum can be rewritten as $\sum_{i=1}^m \frac{1}{p^i}$. A useful bound which came up several times in CS 555, is $\sum_{i=0}^\infty x^i = \frac{1}{1-x}$, for $0 < x < 1$. Clearly,

$$\sum_{i=1}^m \left(\frac{1}{p}\right)^i = \sum_{i=0}^m \left(\frac{1}{p}\right)^i - 1 \leq \sum_{i=0}^\infty \left(\frac{1}{p}\right)^i - 1 = \frac{1}{1-1/p} - 1 = \frac{p}{p-1} - 1 = \frac{1}{p-1}.$$

Plugging back in to our original equation gives the result:

$$\Pr[A : |A| \neq 0] = 1 - \Pr\left[\bigcup_{i=1}^m C_i\right] \geq 1 - \sum_{i=0}^{m-1} \frac{p^i}{p^m} = 1 - \sum_{i=1}^m \left(\frac{1}{p}\right)^i \geq 1 - \left(\frac{1}{p-1}\right).$$

□

6.3 Bibliography

[1] M. Rabin "*Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance,*" Journal of the ACM, 36(2):335-348, 1989.

[2] A. Shamir "*How to Share a Secret,*" Communications of the ACM, 22(11):612-613, 1979.