`secret sharing lab`

Coding the Matrix, Summer 2013

Please fill out the stencil file named "`secret_sharing_lab.py`". While we encourage you to complete the Ungraded Problems, they do not require any entry into your stencil file.

1 *Threshold Secret-Sharing*

Recall we had a method for splitting a secret into two pieces so that both were required to recover the secret. The method used $GF(2)$. We could generalize this to split the secret among, say, four teaching assistants (TAs), so that jointly they could recover the secret but any three cannot. However, it is risky to rely on all four TAs showing up for a meeting.

We would instead like a *threshold* secret-sharing scheme, a scheme by which, say, we could share a secret among four TAs so that any three TAs could jointly recover the secret, but any two TAs could not. There are such schemes that use fields other than $GF(2)$, but let's see if we can do it using $GF(2)$.

1 *First attempt*

Here's a (doomed) attempt. I work with five 3-vectors over $GF(2)$: $\boldsymbol{a}_0, \boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4$. These vectors are supposed to satisfy the following requirement:

> **Requirement:** every set of three are linearly independent.

These vectors are part of the scheme; they are known to everybody. Now suppose I want to share a one-bit secret s among the TAs. I randomly select a 3-vector $\boldsymbol{u}$ such that $\boldsymbol{a}_0 \cdot \boldsymbol{u} = s$. I keep $\boldsymbol{u}$ secret, but I compute the other dot-products:

$$\begin{aligned} \beta_1 &= \boldsymbol{a}_1 \cdot \boldsymbol{u} \\ \beta_2 &= \boldsymbol{a}_2 \cdot \boldsymbol{u} \\ \beta_3 &= \boldsymbol{a}_3 \cdot \boldsymbol{u} \\ \beta_4 &= \boldsymbol{a}_4 \cdot \boldsymbol{u} \end{aligned}$$

Now I give the bit β_1 to TA 1, I give β_2 to TA 2, I give β_3 to TA 3, and I give β_4 to TA 4. The bit given to a TA is called the TA's *share*.

First I argue that this scheme allows any three TAs to combine their shares to recover the secret.

Suppose TAs 1, 2, and 3 want to recover the secret. They solve the matrix-vector equation

$$\left[\begin{array}{c} \boldsymbol{a}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \boldsymbol{a}_3 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} \beta_1 \\ \beta_2 \\ \beta_3 \end{array}\right]$$

The three TAs know the right-hand side bits, so can construct this matrix-vector equation. Since the vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ are linearly independent, the rank of the matrix is three, so the columns are also linearly independent. The matrix is square and its columns are linearly independent, so it is invertible, so there is a unique solution. The solution must therefore be the secret vector $\boldsymbol{u}$. The TAs use `solve` to recover $\boldsymbol{u}$, and take the dot-product with $\boldsymbol{a}_0$ to get the secret $\boldsymbol{s}$.

Similarly, any three TAs can combine their shares to recover the secret vector $\boldsymbol{u}$ and thereby get the secret.

Now suppose two rogue TAs, TA 1 and TA 2, decide they want to obtain the secret without involving either of the other TAs. They know β_1 and β_2. Can they use these to get the secret s? The answer is no: their information is consistent with both $s = 0$ and $s = 1$: Since the matrix

$$\begin{bmatrix} \boldsymbol{a}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{a}_2 \end{bmatrix}$$

is invertible, each of the two matrix equations

$$\begin{bmatrix} \boldsymbol{a}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{a}_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{a}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{a}_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

has a unique solution. The solution to the first equation is a vector $\boldsymbol{v}$ such that $\boldsymbol{a}_0 \cdot \boldsymbol{v} = 0$, and the solution to the second equation is a vector $\boldsymbol{v}$ such that $\boldsymbol{a}_0 \cdot \boldsymbol{v} = 1$.

2 *Scheme that works*

So the scheme seems to work. What's the trouble?

The trouble is that there are no five 3-vectors satisfying the requirement. There are just not enough 3-vectors over $GF(2)$ to make it work.

Instead, we go to bigger vectors. We will seek ten 6-vectors $\boldsymbol{a}_0, \boldsymbol{b}_0, \boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{a}_2, \boldsymbol{b}_2, \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{a}_4, \boldsymbol{b}_4$ over $GF(2)$. We think of them as forming five pairs:

- Pair 0 consists of $\boldsymbol{a}_0$ and $\boldsymbol{b}_0$,
- Pair 1 consists of $\boldsymbol{a}_1$ and $\boldsymbol{b}_1$,
- Pair 2 consists of $\boldsymbol{a}_2$ and $\boldsymbol{b}_2$,
- Pair 3 consists of $\boldsymbol{a}_3$ and $\boldsymbol{b}_3$, and
- Pair 4 consists of $\boldsymbol{a}_4$ and $\boldsymbol{b}_4$.

The requirement is as follows:

> **Requirement:** For any three pairs, the corresponding six vectors are linearly independent.

To use this scheme to share two bits s and t, I choose a secret 6-vector $\boldsymbol{u}$ such that $\boldsymbol{a}_0 \cdot \boldsymbol{u} = s$ and $\boldsymbol{b}_0 \cdot \boldsymbol{u} = t$. I then give TA 1 the two bits $\beta_1 = \boldsymbol{a}_1 \cdot \boldsymbol{u}$ and $\gamma_1 = \boldsymbol{b}_1 \cdot \boldsymbol{u}$, I give TA 2 the two bits $\beta_2 = \boldsymbol{a}_2 \cdot \boldsymbol{u}$ and $\gamma_2 = \boldsymbol{b}_2 \cdot \boldsymbol{u}$, and so on. Each TA's share thus consists of a pair of bits.

Recoverability: Any three TAs jointly can solve a matrix-vector equation with a 6×6 matrix to obtain $\boldsymbol{u}$, whence they can obtain the secret bits s and t. Suppose, for example, TAs 1, 2, and 3 came together. Then they would solve the equation

$$\left[\begin{array}{c} \boldsymbol{a}_1 \\ \hline \boldsymbol{b}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \boldsymbol{b}_2 \\ \hline \boldsymbol{a}_3 \\ \hline \boldsymbol{b}_3 \end{array}\right] \left[\begin{array}{c} \\ \\ \boldsymbol{x} \\ \\ \\ \end{array}\right] = \left[\begin{array}{c} \beta_1 \\ \gamma_1 \\ \beta_2 \\ \gamma_2 \\ \beta_3 \\ \gamma_3 \end{array}\right]$$

to obtain $\boldsymbol{u}$ and thereby obtain the secret bits. Since the vectors $\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{a}_2, \boldsymbol{b}_2, \boldsymbol{a}_3, \boldsymbol{b}_3$ are linearly independent, the matrix is invertible, so there is a unique solution to this equation.

Secrecy: However, for any two TAs, the information they possess is consistent with any assignment to the two secret bits s and t. Suppose TAs 1 and 2 go rogue and try to recover s and t. They possess the bits $\beta_1, \gamma_1, \beta_2, \gamma_2$. Are these bits consistent with $s = 0$ and $t = 1$? They are if there is a vector $\boldsymbol{u}$ that solves the equation

$$\left[\begin{array}{c} \boldsymbol{a}_0 \\ \hline \boldsymbol{b}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{b}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \boldsymbol{b}_2 \end{array}\right] \left[\begin{array}{c} \\ \\ \boldsymbol{x} \\ \\ \\ \end{array}\right] = \left[\begin{array}{c} 0 \\ 1 \\ \beta_1 \\ \gamma_1 \\ \beta_2 \\ \gamma_2 \end{array}\right]$$

where the first two entries of the right-hand side are the guessed values of s and t.

Since the vectors $\boldsymbol{a}_0, \boldsymbol{b}_0, \boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{a}_2, \boldsymbol{b}_2$ are linearly independent, the matrix is invertible, so there is a unique solution. Similarly, no matter what you put in the first two entries of the right-hand side, there is exactly one solution. This shows that the shares of TAs 1 and 2 tell them nothing about the true values of s and t.

3 *Implementing the scheme*

To make thing simple, we will define $\boldsymbol{a}_0 = [1, 1, 0, 1, 0, 1]$ and $\boldsymbol{b}_0 = [1, 1, 0, 0, 0, 1]$:

```
>>> a0 = list2vec([one, one, 0, one, 0, one])
>>> b0 = list2vec([one, one, 0, 0, 0, one])
```

Remember, `list2vec` is defined in the module `vecutil` and `one` is defined in `GF2`.

4 *Generating* $\boldsymbol{u}$

Task 1: Write a procedure `choose_secret_vector(s,t)` with the following spec:

- *input:* $GF(2)$ field elements s and t (i.e. bits)
- *output:* a random 6-vector $\boldsymbol{u}$ such that $\boldsymbol{a}_0 \cdot \boldsymbol{u} = s$ and $\boldsymbol{b}_0 \cdot \boldsymbol{u} = t$

Why must the output be random? Suppose that the procedure was not random: the output vector $\boldsymbol{u}$ was determined by the two secret bits. The TA could use the information to make a good guess as to the value of $\boldsymbol{u}$ and therefore the values of s and t.

For this task, you can use Python's `random` module to generate pseudorandom elements of $GF(2)$.

```
>>> import random
>>> def randGF2(): return random.randint(0,1)*one
```

However, be warned: Don't use this method if you really intend to keep a secret. Python's `random` module does not generate cryptographically secure pseudorandom bits. In particular, a rogue TA could use his shares to actually figure out the state of the pseudorandom-number generator, predict future pseudorandom numbers, and break the security of the scheme. (Figuring out the state of the pseudorandom-number generator uses—you guessed it—linear algebra over $GF(2)$.)

5 *Finding vectors that satisfy the requirement*

Task 2: We have decided that $\boldsymbol{a}_0 = [1, 1, 0, 1, 0, 1]$ and $\boldsymbol{b}_0 = [1, 1, 0, 0, 0, 1]$. Your goal is to select vectors $\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{a}_2, \boldsymbol{b}_2, \boldsymbol{a}_3, \boldsymbol{b}_3, \boldsymbol{a}_4, \boldsymbol{b}_4$ over $GF(2)$ so that the requirement is satisfied:

> For any three pairs, the corresponding six vectors are linearly independent.

Hint: try selecting eight random vectors and testing whether they satisfy the requirement. Repeat until you succeed. Use the `independence` module.

6 *Sharing a string*

Now that we can share two bits, we can share an arbitrarily long string. For this part, you don't have to do any work; it is provided simply for those who are interested.

The module `bitutil` defines procedures

- `str2bits(str)`, which converts a string to a list of $GF(2)$ values
- `bits2str(bitlist)`, the inverse of `str2bits`
- `bits2mat(bitlist, nrows)`, which uses the bits in `bitlist` to populate a matrix with `nrows` rows.
- `mat2bits(M)`, which is the inverse of `bits2mat`

You can use `str2bits` to transform a string, say `"Rosebud"`, into a list of bits, and use `bits2mat` to transform the list of bits to a $2 \times n$ matrix.

For each column of this matrix, you can use the procedure `choose_secret_vector(s,t)` of Task 1 to obtain a corresponding secret vector $\boldsymbol{u}$, constructing a matrix U whose columns are the secret vectors.

To compute the shares of the TAs, multiply the matrix

$$\left[\begin{array}{c} \boldsymbol{a}_0 \\ \hline \boldsymbol{b}_0 \\ \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{b}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \boldsymbol{b}_2 \\ \hline \boldsymbol{b}_3 \\ \hline \boldsymbol{a}_4 \\ \hline \boldsymbol{b}_4 \end{array}\right]$$

times U. The second and third rows of the product form the share for TA 1, and so on.