CRYPTOGRAPHY HANDOUT 06

BINARY AND ONE-TIME PADS

1. Binary

Convert the following numbers to binary:

1. 58 =

2. 47 =

3. 30 =

4. 31 =

Question: What do you notice about the right-most bit for even versus odd numbers?

2. Pseudo-Random Bit Generation

Linear Congruential Generator.

1. Start with an initial seed x_0.
2. Generate a sequence of numbers $x_1, x_2, \cdots$ in which $x_n = ax_{n-1} + b \bmod m$ for parameters a, b and m.

Example. Suppose $x_0 = 2, a = 3, b = 7, m = 5$. Generate x_1, x_2, x_3, x_4, x_5.

$$x_1 =$$

$$x_2 =$$

$$x_3 =$$

$$x_4 =$$

$$x_5 =$$

Question: What are potential problems with this method?

Blum-Blum-Shub (BBS) Pseudo-Random Bit Generator.

1. Choose two large prime numbers p and q in which both primes are congruent to 3 mod 4. Multiply them together to get $n = pq$.
2. Choose a seed $x_0 \equiv x^2 \bmod n$ where x is a number in which $\gcd(n, x) = 1$.
3. Generate a sequence of numbers $b_1, b_2, \cdots$ in which
   a. $x_j \equiv x_{j-1}^2 \bmod n$
   b. b_j is the least significant bit or right-most bit of x_j

Example. (from T & W 2.10) Suppose we choose $p = 24672462467892469787$ and $q = 39673689456783458980 3$. Then

$$n = 978847614085311079416885521741371578196 1.$$

One possible choice of x is $x = 873245647888478349013$.
The initial seed is $x_0 = 8845298710478780097089917746010122863172$.

We can compute $x_1, x_2, \cdots$ to get:

$$x_1 \equiv 7118894281131329522745962455498123822408$$

$$x_2 \equiv 3145174608888893164151380152060704518227$$

$$x_3 \equiv 4898007782307156233272233185574899430355$$

$$x_4 \equiv 3935457818935112922347093546189672310389$$

$$x_5 \equiv 675099511510097048901761303198740246040$$

What are b_1, b_2, b_3, b_4, b_5?

$$b_1 =$$

$$b_2 =$$

$$b_3 =$$

$$b_4 =$$

$$b_5 =$$

3. One-Time Pads

1. Write your plaintext as a sequence of 0s and 1s (binary, or use ASCII, etc.).
2. The key is a random sequence of 0s and 1s of the same length as the message. Once a key is used, it is discarded and not used again.
3. Encryption: Add the key to the message using XOR (exclusive or) or adding mod2.
4. Decryption: Add the same key to the ciphertext to get the plaintext back.

Example. `10110011+11111111=`

Variation.

1. Write your plaintext message.
2. Count the number n of letters you have in your plaintext.
3. The key is a random sequence of shifts between 0 and 25. On SageMathCell, type `randint(0,25)`. Make sure your key is as long as the message (so there are n shifts).
4. Encryption: follow your sequence of shifts.
5. Decryption: subtract instead of adding your shifts.

Question: How might you go about sending your key to a recipient of your message securely?

Now your goal today is to work individually on the following:

1. Create a plaintext message (about 10 letters long) of length n.
2. Generate a sequence of n shifts between 0 and 25. This is your key.
3. Use your sequence of shifts to encrypt your message.
4. Find a way to deliver your encrypted message to your intended recipient. They will do the same and will try to give you a message. Remember that your recipient must get a copy of your key as well as your ciphertext! You are not allowed to just hand them your plaintext.
5. Decrypt the message that you are given.

Alternatively...

If you don't want to go through all that work, you can intercept someone else's message, and if you decrypt it before the intended recipient, you win.

Discussion.

- Which issues did you come across when you were trying to deliver your message?

- How might this generalize to issues in the 'real world?'

- Do you have any ideas on modifying the one-time pad to make the delivery of the key more secure?