CSL7480 Cryptography **13 Jan 2023**

Lecture 5: Cryptography

Lecturer: Somitra Sanadhya *Scribe: Sukhman Preet Singh Sandhu (B20CS073)*

5.1 Hill Cipher

It is a polyalphabetic substitution cipher which uses a square invertible matrix as a key.

- **Encryption**: If the key used(H) is a $n \times n$ matrix then the message(m) is divided into parts of n size each and for each part the cipher text(C) which is also $n \times 1$ matrix is obtained by simple matrix multiplication mod 26. Given a message part $M_{n\times1}$ and a key $H_{n\times n}$, the cipher text $C_{n\times1}$ can be obtained as
$$C_{n\times1} = (H_{n\times n}.M_{n\times1})mod26$$
- **Decryption**: If we know the key then we can easily get the plain text from cipher text by taking the inverse of the key matrix and multiplying it with the cipher text.
$$M = (H^{-1}.C)$$
- **Attack**: Claim - By asking poly(n) queries we can break the hill cipher.
  - Hill Cipher is prone to **known Plain text attacks**. For a $n \times n$ key matrix there are only n^2 unknown variables which can be easily solved using Linear Algebra if we have n^2 equations in those variables.
- **key Space**: If we allow only alphabets as key matrix entries then for a $n \times n$ matrix there are $26^{n\times n}$ possible keys. Effective key size in bits is the number of bits required which equals $log_2(26^{n^2})$ which is approximately $4.7n^2$. But it is just an upper bound because all the matrices will not have there determinant non zero hence won't be invertible.

5.2 What we have covered so far?

1. Toy ciphers are weak and easy to attack.
2. Ultimate security can be formulated in 3 ways:
   - Shannon
   - Perfect
   - Indistinguishability game
3. One-Time Pad is able to achieve perfect security
4. There are various limitations of OTP like:

- Given key space of size K and message space of size M following condition must always hold

$$|K| \geq |M|$$

- Malleability of OTP is a big concern.

5.3 Relaxing the security definition

- For most applications in real life achieving perfect security is not possible hence we relax the definition of our security.
  1. **Computational security**: If the attacker is allowed only poly(n) queries he can not get any information about the plain text using the cipher text.
  2. Even if the attacker succeeds to break the cipher with a very small probability $\frac{1}{2^n}$ then we don't consider it as a valid attack.
     - As Attacker is allowed poly(n) computations he may win by repeating the attack multiple times hence we need negligible functions which remain very small even after polynomial multiplication. For eg, $\frac{1}{2^n}$, $\frac{1}{2^{\frac{n}{2}}}$

5.4 Generic Attacks

- **Brute force**: Try all the possible keys one by one. Requires exponential computations.

$$Prob_{success} = 1$$

- **Guessing the key**: Randomly guessing the key.

$$Prob_{success} = \frac{1}{|\mathcal{K}|}$$

5.4.1 Extending the OTP idea to computational security

We now have an idea that we need something similar to OTP but which is more feasible in real life applications. Having a key space greater than equal to the message space makes key generation, distribution, and management tough.

- This gives us the intuition that if we have a deterministic function which takes an n bit uniformly distributed seed S as an input and produces a bitstream much larger than n bit, then our problem would be solved.

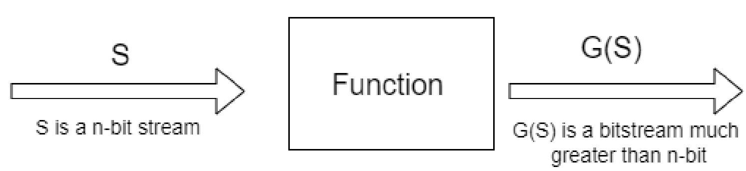

Figure 5.1: Definition of our function

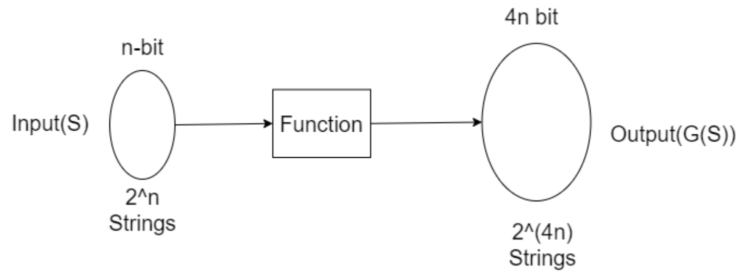

Figure 5.2: Function G(.) produces larger output

- For eg. say our seed S if of n-bit and $G(S)$ produced is of 4-n bit.
  - Since S is of n-bits the number of possible binary string can be 2^n and similarly possible output strings of $4n$ bits can be 2^{4n}.
  - As $G(.)$ is a **deterministic** function hence out of 2^{4n} possible outputs strings we will only get 2^n strings through $G(.)$.
  - Remaining strings will never occur in the output of G(.)
- But what is the usefulness of G?
  - If $G(.)$ produces an output which **"looks like random"** then we can encrypt $4n$ bit message using an n bit key.

5.4.2 Brute force Attack

If exponential computations are allowed then the attacker can easily guess whether the key is generated from $G(.)$ or through a uniformly random function.

1. Attacker will create a table using all the 2^n possible n bit S values and store the corresponding $4n$ bit value generated by $G(.)$.

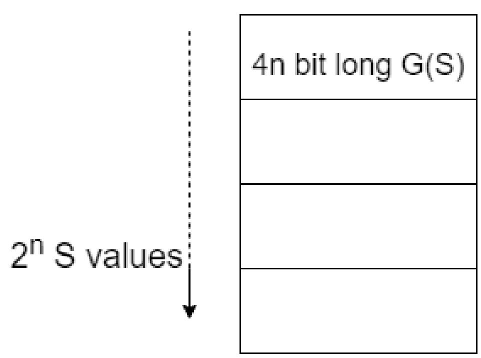

Figure 5.3: Attacker creates a table

2. Now for a given $4n$ bit key r generated uniformly at random.

$$r \xleftarrow{D} \{0,1\}^{4n}$$

3. The attacker checks if r is present in the table or not.
   - If it is not present then the attacker knows for sure that the string has been generated randomly. In this case, the probability of the attacker winning is:
   $$Prob_{win} \ = 1$$
   - If it is present then the attacker cannot distinguish whether the string has been generated through the function $G(.)$ or randomly hence he makes a random guess. In this case, the probability of the attacker winning is:
   $$Prob_{win} \ = \ \frac{1}{2}$$
4. What is the overall probability of winning of the attacker?
   - Since we toss a coin to decide whether to give a randomly generated or a PRG generated string to the attacker hence
   $$Prob_{win} \ = \ \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times 1 \ = \ \frac{3}{4}$$

5.4.3 Psuedo Random Genertor(PRG)

The function G(.) is a PRG if:

1. It is a deterministic length expanding function i.e for a n bit seed S
$$|G(S)| \ > n$$
2. The output of $G(.)$ should be indistinguishable from a uniformly random string for a computationally bound adversary. Say a game is played:
$$r \xleftarrow{D} \{0,1\}^{4n} -- (1)$$
$$S \xleftarrow{D} \{0,1\}^{n}, \ r = G(S) -- (2)$$
   - Out of (1) and (2) we randomly give one string to the adversary and ask whether the string was generated by the PRG or by a uniformly random function then if the probability of guessing correctly i.e :
   $$Prob_{win} \ = \ \frac{1}{2} \ + \ \epsilon(n)$$
   where $\epsilon(n)$ is negligible than we can say that function $G(.)$ produces string that is almost random and hence $G(.)$ is a PRG.
3. Another definition of PRG is:
   - G is a Pseudorandom generator if $\forall$ PPT(probabilistic polynomial time) distinguishers D, $\exists$ a negligible function $\epsilon(n)$ such that
   $$| \ Prob[\ D(r) == 1 \ ] - Prob[\ D(G(S)) == 1 \ ] \ | = |0.5 - (0.5 + \epsilon(n)) \ | \ = \ \epsilon(n)$$
   where, the first probability is taken over uniform choice of $r \leftarrow \ \{0,1\}^{l(n)}$ and the randomness of D the second probability is taken over uniform choice of $S \leftarrow \ \{0,1\}^{n}$ and the randomness of D.

5.4.4 Adverserial game 1

1. Say Attacker chooses two messages m_0 and m_1 and gives it to the challenger.
2. Now the Challenger generates n bit seed S and tosses a coin to generate b
$$S \xleftarrow{\$} \{0,1\}^n$$
$$b \xleftarrow{\$} \{0,1\}$$
3. Now based on value of b challenger chooses m_0 or m_1 and encrypts it using $G(S)$ to get cipher text c.
$$C \;=\; Enc(m_b, G(S)) \;=\; m_b \oplus G(S)$$
4. This c is given to Attacker who has to figure in polynomial computations that out of m_0 and m_1 which plain text message has been encrypted. Now if:
$$Prob_{win} \;=\; \frac{1}{2} \;+\; \epsilon(n)$$
where $\epsilon(n)$ is negligible than we can say that our encryption Scheme is computationally secure.

5.4.5 Adverserial game 2

Another game can be played to check the security of our encryption scheme.

1. Say Attacker chooses a message m of $4n$ bits and gives it to the challenger.
$$m \xleftarrow{\$} \{0,1\}^{4n}$$
2. Now the Challenger tosses a coin to generate b
$$b \xleftarrow{\$} \{0,1\}$$
3. Now encryption is done based on value of b.
4. Say if $b = 0$ then cipher text c is generated using a randomly chosen $4n$ bit key r.
$$r \xleftarrow{\$} \{0,1\}^{4n}$$
$$c \;=\; Enc(m, r) \;=\; m \oplus r - - \;(1)$$
5. Say if $b = 1$ then cipher text c is generated by passing an n bit randomly chosen seed r to the PRG $G(.)$
$$r \xleftarrow{\$} \{0,1\}^n$$
$$c \;=\; Enc(m, G(r)) \;=\; m \oplus G(r) - - \;(2)$$
6. This c is given to the Attacker who has to figure in polynomial computations that out of (1) and (2) which method has been used to encrypt the plain text. Now if:
$$Prob_{win} \;=\; \frac{1}{2} \;+\; \epsilon(n)$$
where $\epsilon(n)$ is negligible then we can say that our Encryption Scheme is computationally secure.

5.4.6 Security of OTP using PRG

Theorem: If a secure PRG $G(.)$ exists then the OTP encryption scheme using PRG is secure.
Proof:
As shown in $Fig : 5.4$

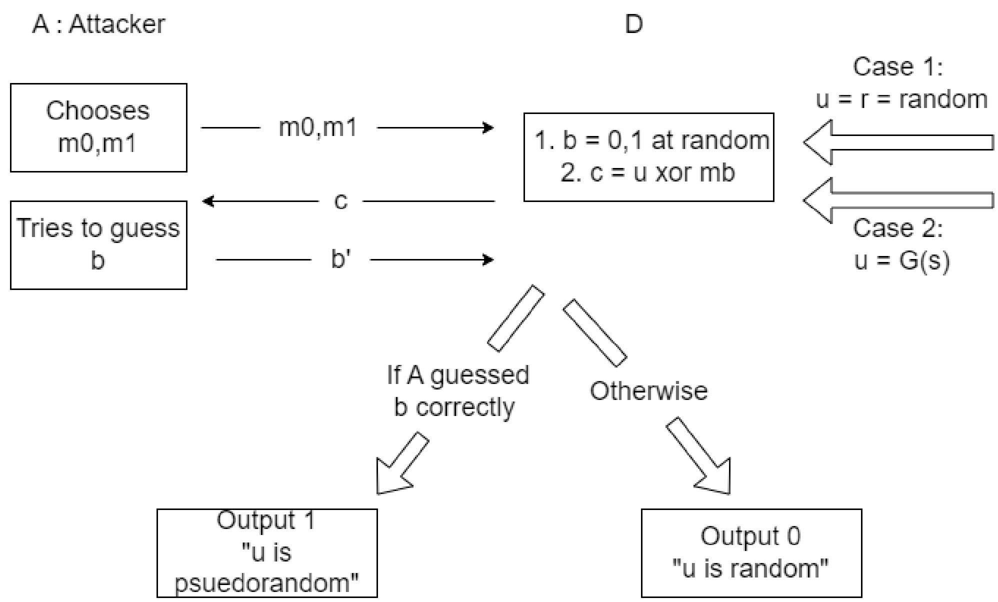

Figure 5.4: Distinguisher D interacts with A as challenger

1. Let A be a PPT(Probabilistic Polynomial time) attacker who can break the security of our encryption scheme as stated in section 5.4.4 i.e.

$$Prob_{win} \ = \ \frac{1}{2} \ + \ \epsilon(n)$$

   where $\epsilon(n)$ is **not** negligible.

2. Using above point we will try to design a PPT distinguisher D who can break the PRG security.

3. The cipher text c is generated as follows

$$c \ = \ u \oplus m_b$$

   where u can be either a randomly generated string or can be generated through PRG $G(.)$.

4. Say if the Attacker A guesses b correctly then it outputs 1 denoting that he correctly guessed that u is a pseudorandom string, else he outputs 0 denoting that "u is random".

5. Case 1: When we actually use a random string to generate the cipher text c.

$$c \ = \ r \oplus m_b \ , \ r \xleftarrow{\$} \{0,1\}^{l(n)}$$

   Here the attacker makes a random guess so the probability that he outputs 1 is

$$Prob_{win} \ = \ Prob[\ b == b' \ ] \ = \ \frac{1}{2}$$

This implies that

$$Prob[\ D(r) == 1\ ]\ =\ \frac{1}{2}\ --\ (1)$$

where D is a PPT distinguisher.

6. Case 2: When we use a pseudorandodm string to generate the cipher text c.

$$c\ =\ G(s) \oplus m_b\ ,\ s \xleftarrow{\$} \{0,1\}^n$$

Here by the assumption made in point 1 probability of output 1 is

$$Prob_{win}\ =\ Prob[\ b == b'\ ]\ =\ \frac{1}{2}\ +\ \epsilon(n)$$

where $\epsilon(n)$ is not negligible. This implies that

$$Prob[\ D(G(r)) == 1\ ]\ =\ \frac{1}{2}\ +\ \epsilon(n)\ \ --\ (2)$$

where D is a PPT distinguisher.

7. The difference between (1) and (2) must be very small for a PRG but

$$|\ Prob[\ D(r) == 1\ ] - Prob[\ D(G(r)) == 1\ ]\ | = |0.5 - (0.5 + \epsilon(n))\ |\ =\ \epsilon(n)$$

And as $\epsilon(n)$ is not negligible hence the PPT distinguisher D is able to distinguish between random string and pseudorandom string generated by $G(.)$ with a probability which is not negligible. This violates the definition of a PRG which implies that G is not a PRG.

8. From previous point we proved that if our encryption scheme is not secure implies that the function $G(.)$ cannot be a PRG. Hence taking contrapositive of this statement proves our Theorem.

References

- https://en.wikipedia.org/wiki/Hill_cipher#Key_space_size
- https://en.wikipedia.org/wiki/Pseudorandom_generator#:~:text=In%20theoretical%20computer%20science%20and,the%20generator%20and%20the%20uniform
- https://www.ccs.neu.edu/home/alina/classes/Spring2018/Lecture4.pdf