**Discrete Logarithm Function (DL)**

Taking p as a prime, g is a generator such that and x as the initial seed.

Assuming discrete logarithm is a one-way function.

A function is one way if the following two conditions hold:

1. (Easy to compute): There exists a polynomial time algorithm Mf computing f that is Mf (x) =f (x) for all x.
2. (Hard to Invert): For every probabilistic polynomial time algorithm A, there exists a negligible function such that

**Hardcore Predicate**

A function is a hard-core predicate of a function f if

1. can be computed in polynomial time, and
2. for every probabilistic polynomial-time algorithm A there exists a negligible function such that

where the probability is taken over the uniform choice of x in and the random coin tosses of A.

**Pseudo Random Generator (PRG)**

PRG is obtained by taking DL as a one-way function and the hard-core predicate of the one-way function. Let f be a one-way permutation and let hc be a hard-core predicate of f. Then,

constitutes a pseudorandom generator with expansion factor .

In case of DL, MSB(x) is a hardcore predicate. Hence, the function G is as follows

Hardcore bit of DL

Let p be an n-bit prime, . Define as

B(x) is hard to compute from f(x), where f(x) is the DL function.

The above function G, can generate an expansion from n bit to n+1 bit. Assuming that there exists a pseudorandom generator with expansion factor them for any polynomial p(.), there exists a pseudorandom generator with expansion factor.

1. Take the last bit from l+1 length string for output.
2. Consider the remaining l length output as input of G for the next iteration.
3. Apply above 1,2 steps ntimes to get output of string n

**Pseudo Random Function (PRF)**

Let be an efficient, length-preserving, keyed function. We say that F is a pseudorandom function if *for all probabilistic polynomial-time distinguishers D, there exists a negligible function such that:*

Construction of PRF from PRG:

Let G be a PRG with an expansion factor . Denoted by the first half of G’s output, and by the second half of G’s output. For every , define the function as .

**CPA Security – Output Feedback Mode**

Using PRF alone will make a deterministic encryption which is prone to CPA attacks. To make an encryption CPA secure we need probabilistic encryption. There are multiple ways to obtain this.

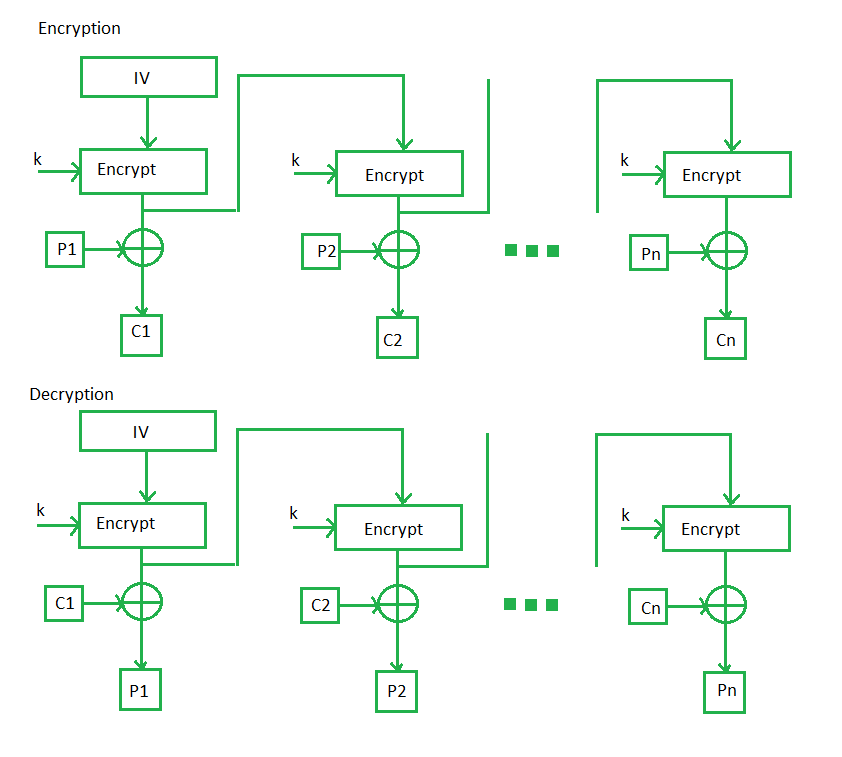
The following is one such approach

1. Simply generating random noise r, key k of length equal to length of plain text i.e. .
2. Encrypt r with a key k using PRF
3. XOR encrypted r with plain text .
4. Send the cipher text
5. At the receiver end, on input k and cipher text plain text

The above implementation will generate an output cipher text of double the length of the plain text.

To obtain probabilistic encryption without length doubling, we use one of the modes of operation.

Output Feedback Mode is used here.

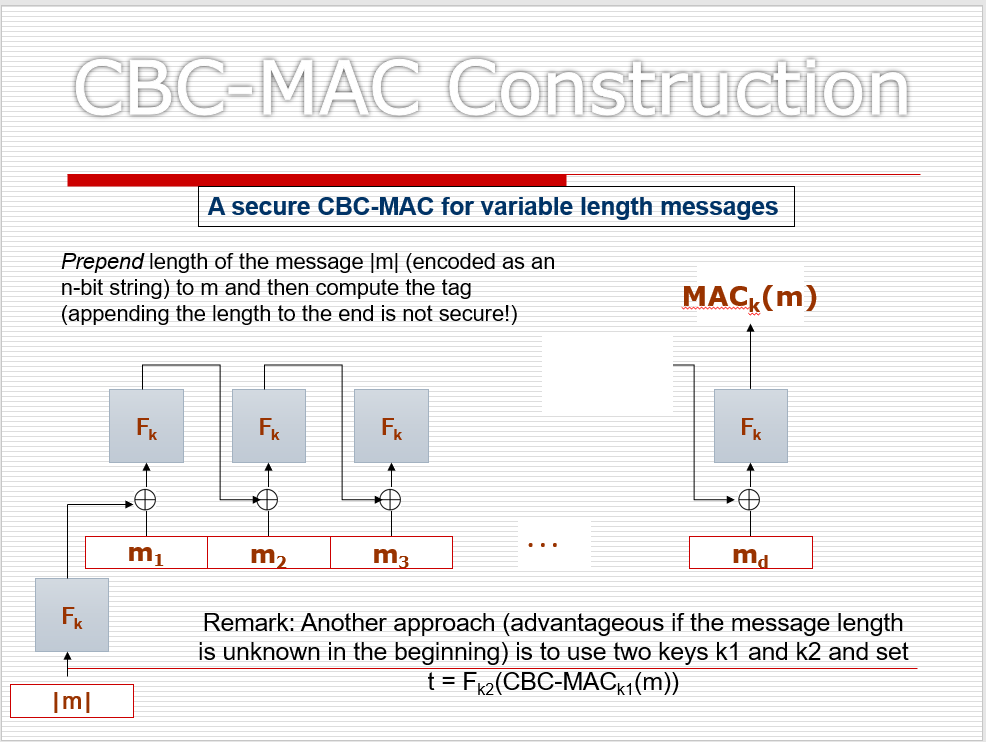


**CCA Security – CBC MAC**

Even though the system is CPA secure it only gives confidentiality. There are some known CCA attacks where the adversary knows that what changes made in cipher text will reflect in plain text. So, CCA security is required to provide integrity.

To obtain integrity preserving the length (prefix attack), sequence number (permutation attack), random identifier (interleaving attack across messages) we use a naïve method to generate tag (MAC). But, the tag (MAC) obtained by the naïve method is much larger than the original plain text. Hence, we use CBC-MAC commonly called as CMAC.

**CBC MAC Construction**



Cipher Block Chaining is used to create the MAC with the message and initialization vector (IV).

Here, is the PRF with the key k.

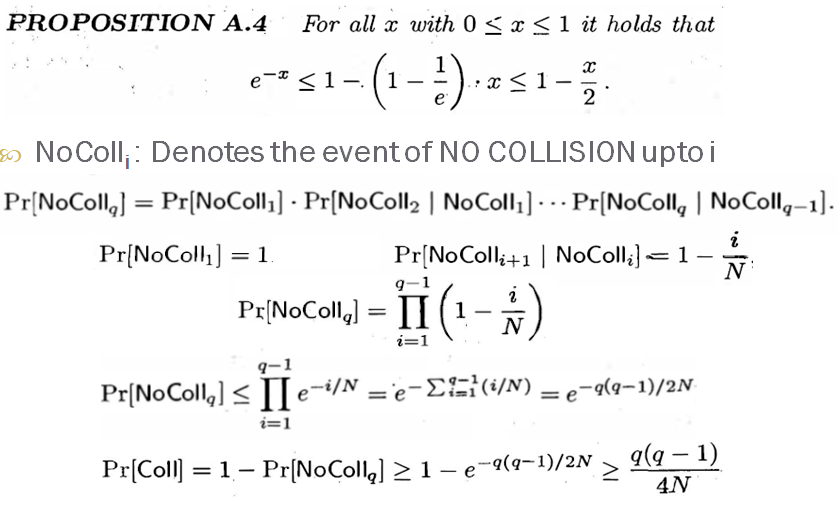
HMAC – FIXED\_HASH, VAR\_HASH, HMAC

Assuming DL is hard to invert, we built fixed length collision resistant hash function.

**Lemma:** Fix a positive integer N, and say elements are chosen uniformly and independently at random from a set of size N. Then the probability that there exist distinct with is at least . i.e.,

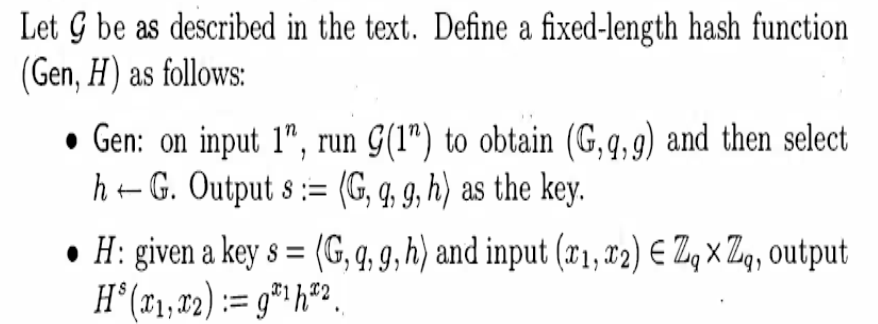
“Smaller the value of q w.r.t N, higher is the collision resistance”

Proof:



Construction of fixed length hash function

Assuming that DL is hard to invert (one-way function).



Proof by contradiction

If discrete logarithm problem is hard relative to G, then the following construction is a fixed-length collision-resistant hash function

For the given value of g, h we found an X, which contradicts the assumption of DL being a one-way function.

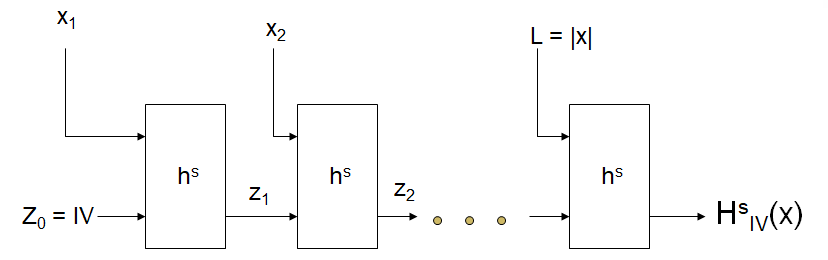
Hence, (Gen, H) is a collision resistant hash function.

**Construction of variable length hash function from fixed length hash function**

**Merkle Damgard Transform**

Theorem:

If (Gen, h) is a fixed length collision resistant hash function, then (Gen, H) is a collision resistant hash function



Let (Gen, h) be a fixed-length collision-resistant hash function for inputs of length and output length . Construct a variable-length hash function (Gen, H) as flows:

* Gen: remains unchanged
* H: on input a key s and a string of length in what follows:
  1. Set (i.e., the number of blocks in x). Pad x with zeroes so its length is a multiple of . Parse the padded result as the sequence of – bit blocks . Set where L is encoded using exactly bits.
  2. Set .
  3. For , compute
  4. Output .

**HMAC**

HMAC is the current industry standard as CBC-MAC is deemed to be slow

(Gen, h): A fixed length hash function

(Gen, H): Hash function after applying MD transform to (Gen, h)

Fixed constants: IV, opad (outer pad), ipad (inner pad)

Opad: 0x36 repeated as many times as needed

Ipad: 0x5C repeated as many times as needed

