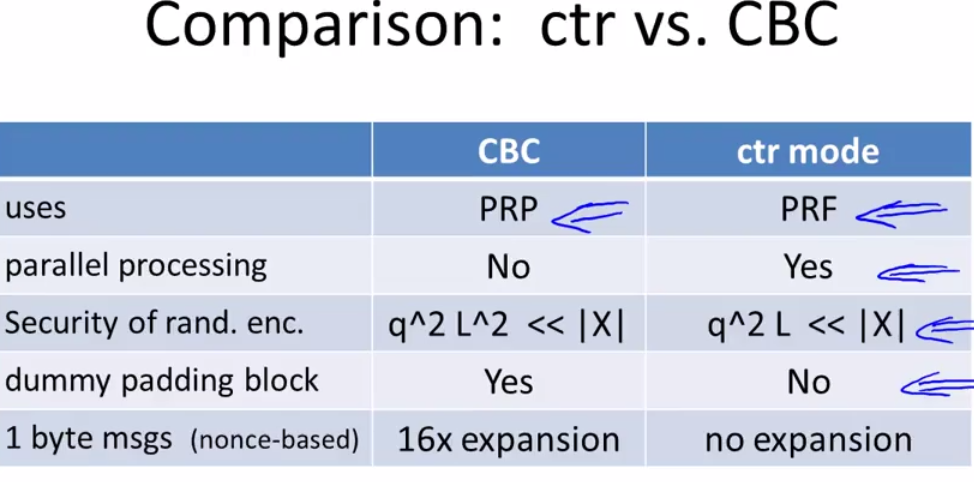
What are Block Ciphers

* Block ciphers: crypto work horse
  + Two algorithms takes n bits as input and a key outputs n bits
  + 3DES: n = 64 bits, k = 168 bits
  + AES: n = 128 bits, k = 128, 192, 256 bits
    - The longer the key the more secure the cipher
* Built by Iteration
  + Takes in key
    - Gets expanded to round keys using a round function R(k,m)
      * K = key
      * M = current state of message
* Performance
  + 3DES 13 mb/sec
  + AES 109 mb/sec
* Abstractly: PRPs and PRFs
  + Pseudo Random Function (PRF) defined over (k,x,y):
    - F: K x X -> Y such that exists “efficient” algorithm to evaluate F(k,x)
  + Pseudo Random Permutation (PRP) defined over (k,x):
    - E: K x X -> X
    - Such that:
      * Exists “efficient” deterministic algorithm to evaluate E(k,x)
      * The function E(k,) is one-to-one
      * Exists “efficient” inversion algorithm D(k,y)
        + Will output the original input
* Running Example
  + Functionally, any PRP is also a PRF
    - A PRP is a PRF where X=Y and is efficiently invertible
* Secure PRFs
  + Let F: K x X -> Y be a PRF
    - Funs[X,Y]: The set of all functions from X to Y
    - S\_f = {F(k,.) s.t. k element K } subset Funs[X,Y]
  + Intuition: a PRF is secure if
    - A random function in Funs[X,Y] is indistinguishable from a random function in S\_f
    - S\_f size = |K|
    - Funs[X,Y] size = |Y| ^ |X|
* An easy application: PRF -> PRG
  + Let F: K x {0,1}^n -> {0,1}^n be a secure PRF
    - Then the following G: K -> {0,1} ^ nt is a secure PRG:
      * G(k) = F(k,0) || F(k,1) || … || F(k,t)
    - Key property: parallelizable
    - Security from PRF property: F(k,.) indist. from random function f(.)
* **Data Encryption Standard (DES)**
* DES: core idea – Feistel Network
  + Given functions f1, .. ,fd: {0,1}^n -> {0,1}^n
  + Goal: build invertible function F; {0,1}^2n -> {0,1}^2n
  + Claims: for all f1, .. ,fd: {0,1}^n -> {0,1}^n
  + Feistel network F: {0,1}^2n -> {0,1}^2n is invertible
  + Proof: construct inverse
    - R\_i = L\_i+1
    - L\_i = f\_i+1(Li+1) xor R\_i+1
* Decryption circuit
  + Inversion is basically the same circuit, with f\_1, .. ,f\_d applied in reverse order
  + General method for building invertible functions (block ciphers) from arbitrary functions
  + Used for many block ciphers … but no AES
* Thm:
  + F: K x {0,1}^n -> {0,1}^n a secure PRF
    - 3-round Feistel F: K^3 x {0,1}^2n -> {0,1}^2n a secore PRP
* DES: 16 round Feistel network
  + F1, … ,f16: {0,1}^32 -> {0,1}^32, f\_i(x) = F(ki,x)
    - Each ki is a round key derived from the key k
    - 64 bit input initial permutation
    - then 16 round Feistel network
    - Final permutation inverse of the initial permutation
    - Final output
* The function F(ki,x)
  + Takes 32 bits input and maps to 48 bits using expansion box
  + Takes 48-bit round key
  + Compute xor of expansion and round key
  + 48 bits are broken into 8 groups of 6 bits
  + The bits go into s boxes
  + The outputs of s boxes map from 6 bits to 4 bits
  + Output is a permutation of the combined s boxes or the combined 32 bits
  + S-box function {0,1}^6 -> {0,1}^4, implemented as look up table
  + S\_i(x) = A\_i dot x (mod 2)
* Example: a bad S-box choice
  + The entire DES cipher would be linear: there is a fixed binary matrix B s.t.
  + DES(k,m) = 64 + (16 \* 48)= 832
  + DES(k,m1) xor DES(k,m2) xor DES(k,m3) = DES(k,m1 xor m2 xor m3)
  + If the s boxes were completely linear DES would be completely insecure
* Choosing the S-boxes and P-box
  + Choosing the S-boxes and P-box at random would result in an insecure block cipher (key recovery after approximately 2 ^ 34 outputs)
  + No output bit should be close to a linear function of the input bits
  + S-boxes are 4-to-1 maps
* **Exhaustive Search Attacks**
* Exhaustive Search for block cipher key
  + Goal: given a few input output pairs (mi, ci = E(k,mi)) I = 1, .. ,3 find key k
    - Find the key that does the mapping (m1 m2 m3) ->\_k (c1 c2 c3)
  + Lemma: Suppose DES is an ideal cipher
    - (2^56 random invertible functions {0,1}^64 -> {0,1}^64)
    - Then there is a m, c there is at most one key k s.t. c = DES(k,m)
      * With prob >= 1 – 1/256 approximately equal to 99.5%
  + Proof:
    - Pr[there is k’ != k: c = DES(k,m) = DES(k’,m)] <= summation of k’ over all the keys the probability that [DES(k,m1) = DES(k’,m)] <= 1/2^64 \* 2 ^56 = 1/ 256
    - This is the probability that the key is not unique
* Exhaustive Search for block cipher key
  + For two DES pairs (m1,c1=DES(k,m1)), (m2,c2 = DES(k,m2))
    - unicity prob approximately equal 1 – ½^71
    - the mapping from (m1, m2) -> (c1, c2)
  + For AES-128: given two input/output pairs, unicity prob a= 1 – ½^128
  + Two input / output pairs are enough for exhaustive key search
* DES challenge
  + Msg = “The unknown messages is: XXXX …”
  + CT = c1 c2 c3 c4
  + Goal find k element {0,1}^56 s.t. DES(k,mi) = ci for I = 1,2,3
* Strengthening DES against ex. Search
  + Method 1: Triple DES
    - Let E: K x M -> M be a block cipher
    - Define 3E:K^3 x M -> M as 3E((k1,k2,k3),m) =
      * E(k1,D(k2,E(k3,m)))
  + FOR 3DES: KEY-SIZE = 3 \* 56 = 168 3 \* slower
* Why not double DES?
  + Define 2E((k1,k2),m) = E(k1,E(k2,m))
  + Key-len = 112 bits for DES
  + M -> E(k2, ) -> E(k1, ) -> c
  + Attack: M = (m1,…..,m10), C=(c1,…,c10)
    - Find(k1,k2) s.t. E(k1,E(k2,M)) = C
    - K(k2,m) = D(k,c)
  + Attack
    - Step 1: build table sort on 2nd column
    - Step 2: for all k element {0,1}^56 do: test if D(k,C) is in 2nd column
      * If so then E(k^I,M) = D(k,C) => (k^I,k) = (k2,k1)
* Method 2: DESX
  + E: K x M -> M a block cipher
  + E: K x {0,1}^n -> {0,1}^n a block cipher
  + Define EX as EX((k1,k2,k3),m) = k1 xor E(k2,m xor k3)
  + For DESX: key-len = 64 + 56 + 64 = 184 bits
    - But easy attack in time 2^64 + 56 = 2^120
  + Note if xor only on the outside of the encryption or only on the inside of the encryption the cipher does nothing
* **More attacks on block ciphers**
* Attacks on the implementation
  + Side channel attacks:
    - Measure time to do enc/dec, measure power for enc/dec
  + Fault attacks
    - Computing errors in the last round expose the secret key k
* Linear and differential attacks
  + Given many inputs / outputs pairs, can recover key in time less than 2 ^ 56
  + Linear cryptanalysis (overview): let c = DES(k,m)
    - Suppose for random k,m:
    - Pr[m[i1]xor…xorm[ir] xor c[jj]xor…xorc[jv] = k[l1]xor..xork[lu]] = ½ + epsilon
  + For some epsilon. For DES, this exists with epsilon = ½^21
* Linear attacks
  + Relationship Thm: given 1/epsilon ^ 2 random (m, c=DES(k,m)) pairs then k[l1,…,lu] = MAJ[m[i1,…,ir] xor c[ji,…,jv]] = ½ + epsilon with prob. >= 97.7%
  + For DES, epsilon = ½^21 = with 2^42 input/output pairs can find k[l1,….,lu] in time 2^42
  + Roughly speaking: can find 14 key “bits” this way in time 2^42
  + Brute force remaining 56-14=42 bits in time 2^42
  + Total attack time approximately equal 2^43 with 2^42 random input/output pairs
* Lesson
  + A tiny bit of linearly in S\_5 lead to a 2^42 attack
    - Don’t design ciphers yourself
* Quantum attacks
  + Generic search problem
    - Let f: X -> {0,1} be a function
    - Goal: find x element of X s.t. f(x) = 1
  + Classical computer: best generic algorithm time = O(|X|)
  + Quantum computer: time = O(|X|^1/2)
  + Can quantum algorithms be built: unknown
* Quantum exhaustive search
  + Given m, c = E(k,m) define
    - F(k) = 1 if E(k,m) = c
    - 0 otherwise
    - k is element of K
  + Grover -> quantum computer can find k in time O(|k|^(1/2))
  + Quantum computer => 256-bits key ciphers (e.g. AES-256)
    - Secure
* The AES process
  + Key sizes 128, 192, 256 bits
  + Block size: 128 bits
* AES is a Subs-Perm network (not Feistel)
  + All the bits are changed in each round
  + Xor the current state with the round key
  + Blocks of state are replaced with other blocks
  + Permutation state bits are permuted and shuffled around
  + Repeat and then output
  + The whole process needs to be invertible
* AES 128 schematic
  + Operates on a 128 bit block which is 16 bytes, we write this as a 4 by 4 matrix
  + Xor with the first round key
  + byteSub, shiftRow, and MixColumn
  + repeat this process 10 times the last round however
    - byteSub
    - ShiftRow
  + Round keys themselves come from a 16 byte key
    - Key expansion: 16 bytes -> 176 bytes
      * 11 keys each being 16 bytes
* The round function
  + ByteSub: a 1 byte S-box. 256 byte table (easily computable)
    - For all I,j A[I,j] <- S[A[I,j]]
    - The lookup table is A containing a 4 by 4 byte matrix
  + ShiftRows
    - Cyclic shift of the rows in the matrix
  + MixColumns
    - Performs a linear transformation to the columns
    - Applied independently to each one of the columns
* Javascript AES
  + AES in the browser
    - The code that is sent to the browser has no pre-computed tables
      * Thus has fairly small code
    - Once the code lands on the browser the pre-computation of the tables is done
    - Once have the pre-computed tables encrypt
  + AES in hardware
    - AES instructions in intel Westmere:
      * Aesenc, aesenclast: do one round of AES 128-bit registers: xmm1=state, xmm2=round key
      * Aesenc xmm1, smm2 ; puts result in xmm1
    - Aeskeygenassist: performs AES key expansion
    - Claim 14 x speed-up over OpenSSL on same hardware
* Attacks
  + Best key recovery attack:
    - For times better than ex.search
    - 128-bit key => 126 bit key
  + Related key attack on AES-256
    - Given 2^99 input/output pairs from four related keys in AES-256 can recover keys in time approximately equal to 2^99
* Block ciphers
* Can we build a PRF from a PRG
  + Let G: K -> K^2 be a secure PRG
  + Define 1-bit PRF F:K x {0,1} -> K as F(k,x element {0,1}) = G(k)[x]
  + Theorem: If G is a secure PRG then F is a secure PRF
  + Can we build a PRF with larger domain?
* Extending PRG
  + Let G: k => k^2
  + Define G1:K->K^4 as G1(k) = G(G(k)[0]) || G(G(k)[1])
  + Output of the PRG is indistinguishable from two random values in k
  + The function G takes in the input k and creates two outputs using the generator twice we obtain the 4 output as desired.
  + We get a 2-bit PRF:
    - F(k, x element {0,1}^2) = G\_1(k)[x]
* G\_1 is a secure PRG
  + What we want to argue is that this distribution is indistinguishable from random four tuple in K^4
  + We know that the generator is secure so the output of the first level is indistinguishable from random.
  + Replace the first level by truly random strings
  + Output of the PRG is indistinguishable from random
    - So we replace the output with random
  + Replace the pseudo outputs with truly random outputs
  + Get the distribution that we want from replacing by truly random.
* Extending more
  + Gradually change the outputs in truly random outputs then can extend into a multiple of 2
  + We get a 3-bit PRF
    - F(k,101)
* Extending even more: the GGM PRF
  + Let G: K -> K ^ 2 define PRF F:K x {0,1} ^ n -> K as for input x = x0 x1 .. xn-1 element of {0,1}^n do:
  + Security: G a secure PRG => F is a secure PRF on {0,1}^n
  + Not used in practice due to slow performance
* Secure block cipher from a PRG?
  + Can we build a secure PRP from a secure PRG
    - Yes
* Using block ciphers: Crypto work horse
  + Canonical examples:
* Abstractly: PRPs and PRFs
  + Pseudo random Function (PRF) defined over (K,X,Y)
    - F: K x X -> Y
    - Such that exists “efficient” algorithm to evaluate F(k,x)
  + Pseudo random Permutation (PRP) defined over (K,X):
    - E: K x X -> X such that:
      * 1. Exists “efficient “ deterministic algorithm to evaluate E(k,x)
      * 2. The function E(k, .) is one to one
      * 3. Exists “efficient” inversion algorithm D(k,x)
* Secure PRFS
  + Let F: K x X -> Y be a PRF
    - Funs[X,Y]: the set of all functions from X to Y
    - S\_f = {F(k, .)s.t. k element K } subset Funs[X,Y]
  + Intuition: a PRF is secure if
    - A random function in Funs[X,Y] is indistinguishable from a random function in S\_f
    - S\_f 🡨 size |k|
    - Func[X,Y] <- size |Y|^|X|
* Secure PRF: definition
  + For b=0,1 define experiment EXP(b) as:
  + Challenger choose a random pusedo random function
    - B = 0: k <- K, f <- F(k, .)
    - B = 1: f <-Funs[X,Y]
  + Advisory outputs b’ element {0,1} EXP(b)
  + Def: F is secure PRF if for all efficient A:
    - Adv\_prf[A,F] := |Pr[EXP(0) = 1] – Pr[EXP(1) = 1] | is “negligible.”
* Secure PRP (secure block cipher)
  + Same as the experiment before setup for the Secure PRF except Perms[X]
  + Def: E is a secure PRP if for all “efficient” A:
    - Adv\_prp[A,E] = |Pr[EXP(0) = 1] – Pr[EXP(1)=1] | is “negligible.”
    - Pseudo random and random indistinguishable
* Example secure PRPs
  + 3DES, AES
  + AES-128: K x X -> X where K = X = {0,1}^128
  + An example concrete assumption about AES:
    - All 2^80 algs A have Adv\_prp[A,AES] < 2^-40
* Consider the 1-bit PRP from the previous question: E(k,x) = x xor k
  + Is it a secure PRF?
  + Note that Funs[X,X] contains four functions
    - No
  + Simple Attack
    - Attack A:
      * 1) query f(.) at x = 0 and x =1
      * 2) if f(0) = f(1) output “1” , else “0”
      * AdvPRF[A,E] = [0-1/2] = ½
* PRF Switching Lemma
  + Any secure PRP is also a secure PRF, if |X| is sufficiently large
  + Lemma: Let E be a PRP over (K,X)
    - Then for any q-query adversary A: (makes at most q queries)
      * |Adv\_PRF[A,E] – Adv\_PRP[A,E]| < q^2 / 2|X|
        + since X is very large this quantity is negligible
      * Suppose |X| is large so that q^2 / 2|X| is “negligible”
        + Then Adv\_prp[A,E] “negligible” => Adv\_prf[A,E] “negligible”
* Final Note
  + Suggestion:
    - Don’t thing about the inner-workings of AES and 3DES
  + We assume both are secure PRPs and will see how to use them
* **Modes of operation: One time key**
* Using PRPs and PRFs
  + Goal: build “secure” encryption from a secure PRP
  + This segment: **one-time keys**
    - Adversary’s power:
      * Adv sees only one ciphertext (one-time key)
    - Adversary’s goal:
      * Learn info about PT from CT (semantic security)
* Incorrect use of a PRP
  + Electronic Code Block (ECB)
    - Break message into blocks
      * In case of AES break message into 16 byte blocks
    - Then encrypt each block separately
  + Problem:
    - If m1=m2 then c1=c2
* Semantic Security (one-time key)
  + Challenger sends
  + Advisory outputs two messages m0, and m1 |m0| = |m1|
  + The advisory then gets the encryption of m0 and m1
    - Two different experiments
  + The goal is to say that the advisory cannot distinguish between these two experiments
  + Adv\_ss[A,OTP] = |Pr[EXP(0)=1] – Pr[EXP(1)=1]| should be “neg”
* ECB is not Semantically Secure
  + ECB is not semantically secure for messages that contain more than one block
  + When the advisory encrypts the message c1=c2 output 0, else output 1
  + Then Adv\_ss[A,ECB] = 1
* Secure Construction 1
  + Deterministic counter mode from a PRF F:
    - E\_DETCTR(k,m) = message xor function
      * Each block of the message is xor with the function(k,INT)
      * Obtain the cipher
    - Stream cipher built from a PRF (e.g. AES, 3DES)
* Det. Counter-mode security
  + Theorem: For any L > 0, If F is a secure PRF over (K,X,X) then E\_detctr is sem. Sec. cipher over (K,X^l,X^l). In particular, for any eff. Adversary A attacking E\_detctr there exists a n eff. PRF adversary B s.t.:
    - Adv\_ss[A,E\_dtctr] = 2 \* Adv\_prf[B,F]
  + Adv\_prf[B,F] is negligible (since F is a secure PRF) Hence, Adv\_ss[A,E\_detctr] must be negligible
* **Security for many-time key**
* Semantic Security for many-time key
  + Key used more than once => adv. Sees many CTs with the same key
  + Adversary’s power: chosen-plaintext attack (CPA)
    - Can obtain the encryption of arbitrary messages of his choice (conservative modeling of real life)
  + Adversary’s goal:
    - Break sematic security
* Semantic Security for many-time key
  + E = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as
    - Challenger
      * k <- K
    - Advisory
    - Advisory queries the challenger by submitting two messages m10 and m11 element of M |m10| = |m11|
    - Advisory receives the encryption of one of the two messages
    - Can does this for i=1,….,q
  + Chosen plain text attack
    - If adv. Wants c = E(k,m) it queries with mj0 = mj1 = m
    - Def: E is sem.sec. under CPA if for all ‘efficient’ A:
      * Adv\_cpa [A,E] = |Pr[EXP(0)=1] – Pr[EXP(1)=1]| is “negligible”
* Ciphers insecure under CPA
  + Suppose E(k,m) always outputs same ciphertext for msg m. Then:
    - Attack sends the same message as the query m0, m0 element M
    - Obtains the cipher text for E(k,m0) c0
    - Attacker sends a query m0 and m1 element of M
    - Obtains the encryption of either m0 or m1
    - The attacker checks if c = c0 then outputs 0 if c = c0
  + So what?
    - An attack can learn that two encrypted files are the same, two encrypted packet’s are the same, etc
    - Attacker’s advantage is 1 meaning that the system can not be CPA secure
    - Every message is always encrypted to the same cipher text
  + If secret key is to be used multiple times => given the same plaintext message twice, encryption must produce different outputs
* Solution 1: randomized encryption
  + E(k,m) is a randomized algorithm
    - When encrypting a message the message is mapped to a ball and outputs the encryption
    - When the decryption algorithm is running the algorithm will always map back to the original message
    - Encrypting same message twice gives different ciphertexts (w.h.p)
      * W.h.p meaning with high probability
    - Ciphertext must be longer than plaintext
      * Roughly speaking: CT-size = PT-size + “#random bits”
* Randomized encryption
  + Let F; K x R -> M be a secure PRF
  + For m element M define E(k,m) = [r <- R\_R, output (r,F(k,r) xor m)]
  + Is E semantically secure under CPA?
    - Yes, but only if R is large enough so r never repeats (w.h.p)
* Solution 2: nonce-based Encryption
  + Encryption algorithm takes in three inputs
    - E(k,m,n) = c
  + Decryption algorithm takes the nonce as input along with the cipher and obtains the original message
  + Nonce n: a value that changes from message to message. (k,n) pair never used more than once
  + Method1: nonce is a counter (e.g. packet counter)
    - Used when encryptor keeps state from message to message
    - If decryptor has same state, need not send nonce with CT
  + Method 2: encryptor choose a random nonce, n < N (w.h.p)
* CPA security for nonce-based encryption
  + System should be secure when nonces are chosen adversarially
  + Advisory
    - Sends the query containing the message and nonce
  + Challenger
    - Sends the encryption containing the message k and nonce
    - E(k,mib,ni)
  + All nonces {n1,…nq} must be distinct
  + Def: nonce-based E is sem.ec. under CPA if for all “efficient” A:
    - Adv\_cpa[A,E] = |Pr[EXP(0)=1 – Pr[EXP(1)=1]| is “negligible”
* **Modes of operation: many time key**
* Construction 1: CBC with random IV
  + Let(E,D) be a PRP.
  + E\_cbc(k,m) choose random IV element X and do:
  + IV is one block the message in the case of AES the IV would be 16 bytes
  + IV xor with m0
  + The result is then encrypted with the key and output is the cipher text
  + We then use the the first block of the cipher as a mask for the next message
  + M1 is xor with the first cipher block
  + It is then encrypted
  + Repeat the process for the entire message
  + Final cipher text is IV and all the cipher blocks
    - IV = Initialization vector
* Decryption circuit
  + In symbols: c[0] = E(k, IV xor m[0]) => m[0] = D(k,c[0]) xor IV
* CBC: CPA Analysis
  + CBC Theorem: For any L > 0
    - If E is a secure PRP over (K,X) then E\_cbc is a sem.sec. under CPA over (K,X^l,X^l+1). In particular, for a q-query adversary A attacking E\_cbc there exists a PRP adversary B s.t.:
      * Adv\_cpa[A,E\_cbc] <= 2 \* Adv\_prp[B,E] + 2 \* q^2 \* l^2 / |X|
  + Note: CBC is only secure as long as q^2L^2 << |X|
    - L = length of the message
    - Q is the number of cipher texts
      * The number of times we use the key k to encrypt messages
* An example
  + Adv\_cpa[A,E\_cbc] <= 2 \* Adv\_prp[B,E] + 2 \* q^2 \* l^2 / |X|
  + Q = the number of messages encrypted with k, L = length of the max message
  + Suppose we want Adv\_cpa[A,E\_cbc] <= 1/(2^32) <= q^2L^2 / |X| < 1/(2^32)
    - AES: |X| = 2^(128) => q L < 2 ^ (48)
      * So, after 2^48 AES blocks , must change key
    - 3DES: |X| = 2^64 => q L < 2 ^ 16
* Warning: an attack on CBC with rand. IV
  + CBC where attacker can predict the IV is not CPA-secure
  + Suppose given c <- E\_cbc(k,m) can predict IV for next message
  + Example
    - Advisory sends query 0 element of X
    - Advisory gets back encryption of the one block xor IV
      * C1 <- [IV1,E(k,0xorIV1]
    - Advisory sends message
      * M0 = IV xor IV1, m1 != m0
    - Challenger sends encryption to advisory [IV,E(k,IV1)]
    - What’s encrypted is (IV xor IV1) xor IV = IV1
      * If the IV is predictable there is no security
* Construction 1’: nonce-based CBC
  + Cipher block chaining with unique nonce: key=(k,k1)
    - Unique nonce means: (key,n) pair is used for only one message
  + Nonce is included if unknown to decryptor text
  + If nonce is not unique need to perform the extra encryption step
* An example Crypto API (OpenSSL)
  + Void AES\_cbc\_encrypt();
  + When nonce is non random need to encrypt it before use
* A CBC technicality: padding
  + If the last block is less then 16 bytes then add padding
  + TLS: for n>0 n byte pad is n|n|n|…|n
  + Pad removed during decryption
  + If no pad needed, add a dummy block
    - Adding dummy block of 16 blocks last block check last byte if 16 remove the blocks
* **Modes of Operation: Many Time Key (CTR)**
* Construction 2: rand ctr-mode
  + Let F be a secure PRF
  + F:K x {0,1}^n -> {0,1}^n
  + Method
    - Choose random IV
    - First encryption is IV then IV+1
    - Obtain the cipher text
    - IV – chosen at random for every message
      * Note: parallelizable (unlike CBC)
* Construction 2’: nonce ctr-mode
  + To ensure F(k,x) is never used more than once, choose IV as:
  + Have a normal nonce in the left hand side and the counter on the right hand side of IV
    - The counter is incremented for each block
* Rand ctr-mode (rand.IV): CPA analysis
  + Counter-mode Theorem: For any L>0 If F is a secure PRF over (K,X,X) then E\_ctr is a sem.sec. under CPA over (K,X^L,X^L+1)
  + In particular, for a q-query adversary A attacking E\_ctr there exists a PRF adversary B s.t.:
    - Adv\_cpa[A,E\_ctr] <= 2 \* Adv\_prf[B,F] + 2 q ^ 2 L / |X|
  + Note: ctr mode only secure as long as q^2 L << |X|. Better than CBC!
* An example
  + Adv\_cpa[A,E\_ctr] <= 2 \* Adv\_prf[B,E] + 2 q^2 L / |X|
  + Q number of messages encrypted with k, L = length of max message
  + Suppose we want Adv\_cpa[A,E\_ctr] <= 1/(2^32) <= q^2 L / |X| < 1 /(2^32)
* AES: |X| = 2 ^ 128 => q L ^ ½ < 2 ^ 48
  + So, after 2 ^ 32 CTs each of len 2^32, must change key (total of 2 ^ 64 AES blocks )
* Comparison: ctr vs. CBC



* Summary
  + PRPs and PRFs: a useful abstraction of block ciphers
  + We examined two security notions:
    - 1. Semantic security against one-time CPA
    - 2. Semantic security against many-time CPA.
    - Note: neither mode ensures data integrity
  + Stated security results summarized in the following table:

