week1

Secure sockets layer / TLS
- Handshake protocol: Establish shared secret key using public-key cryptography
- Record Layer: Transmit data using shared secret key
  - Ensure confidentiality and integrity

Protect files on disk
- no eavesdropping
- no tampering
- analogous to secure communication
  - alice today sends message to alice tomorrow

sym. encryption
- two parties share secret key k
- use cipher E, D
  - E encryption algorithm
  - D decryption algorithm
- Encryption algorithm is publicly known
  - never use a proprietary cipher

Use Cases
- single use key
  - key is only used to encrypt one message
- multi use key
  - key used to encrypt multiple message
    - encrypted files: same key used to encrypt many files
  - need more machinery than for one-time key

What is Cryptography
- Crypto core
  - secret key establishment
  - secure communication
- But crypto can do much more
  - Digital signatures
  - Anonymous communication
  - Anonymous digital cash
    - can I spend “digital coin” without anyone knowing who I am?
    - How to prevent double spending?
- Protocols
  - Elections
    - votes sent to center encrypted outputs winner
  - Private auctions
    - auction center obtains encrypted bids computes the highest bidder and the 2nd highest bid
  - Secure multi-party computation
    - compute f(inputs)
    - trusted authority
      - collects individual inputs
      - publishes the value of the function
  - Theorem. anything the can done with trusted auth. can also be done without
    - instead the parties talk to each other using some protocol
    - nothing other than the value of the function is revealed

- Crypto magic
  - privately outsourcing computation
    - send encrypted message to google
    - google responds with the encrypted message indicating the results of the search
    - google doesn't know what the search was for
  - zero knowledge
    - alice N = p * q (product of two large primes)
    - bob just has the number N
- A rigorous science
  - precisely specify threat model
  - propose a construction
  - prove that breaking construction under threat mode will solve an underlying hard problem

Discrete Probability

- U: universe finite set (e.g. U = {0,1}^n)
- Def: Probability distribution P over U is a function P:U -> [0,1] such that summation P(x) = 1
- Uniform distribution:
  - for all x element U: P(x) = 1 / |U|
- Point distribution
  - x_0: P(x_0) = 1, for all x != x_0: P(x) = 0
- Events
  - For a set A subset U:
    - Pr[A] = summation x element A P(x) element [0,1]
  - The set A is called an event
  - Example: U = {0,1}^8
    - A = { al x in U such that lsb_2(x) = 11 } subset U
      - for the uniform distribution on {0,1}^8: Pr[A] = 1/4
    - |U| = 256
    - 64 strings in 11 256 total 64 / 256
- The union bound
  - For events A_1 and A_2 subset U
    - Pr[A_1 subset A_2] <= Pr[A_1] + Pr[A_2]
  - If A_1 intersect A_2 = empty set => Pr(A_1 union A_2) = Pr[A_1] + Pr[A_2]
  - Pr[lsb_2(x) = 11 or msb_2(x) = 11] = Pr[A_1 union A_2] <= 1/4 + 1/4 = 1/2
- Random Variables
  - Def: a random variable X is a function X: U -> V (set V where the random variable takes its values)
  - Example: X:{0,1}^n -> {0,1}; X(y) = lsb(y) element of {0,1}
  - For the inform distribution on U:
    - Pr[X=0] = 1/2, Pr[X=1] = 1/2
- The uniform random variable
  - Let U be some set, e.g. U = {0,1}^n
  - We write r <- r U to denote a uniform random variable over U for all a element of U:
    - Pr[r = a] = 1/ |U|
  - formally,r is the identity function: r(x) = x for all x in U
  - Let r be a uniform random variable on {0,1}^2
  - Define the random variable X = r_1 + r_2
  - Then Pr[X=2] = 1/4
- Randomized algorithms
  - Deterministic algorithm: y <- A(m)

  - Randomized algorithm
    - y <- A(m;r) where r <- r {0,1}^n
    - output is a random variable
      - y <- r A(m)
  - Example: A(m;k) = E(k,m), y <- r A(m)

Discrete Probability

- Recap
  - U: finite set (e.g. U = {0,1}^n)
  - Prob. distr. P over U is a function P: U -> [0,1] s.t. summation x element U P(x) = 1
  - A subset U is called an event and Pr[A] = summation x element A P(x) element of [0,1]
  - Pr[U] = 1
  - A random variable is a function X:U -> V
  - X takes values in V and defines a distribution on V
- Independence
  - Def: events A and B are independent if Pr[A and B] = Pr[A] * Pr[B]
  - random variables X,Y taking values in V are independent if for all a and b elements V: Pr[X = a and Y = b] = Pr[X=a] * Pr[Y=b]
  - Example: U = {0,1}^2 = {00,01,10,11} and r <- r U
    - Define r.v. X and Y as: X = lsb(r), Y = msb(r)
    - Pr[X = 0 and Y = 0] = Pr[r =00] = 1/4 = Pr[X = 0] * Pr[Y = 0]
- Review: XOR
  - XOR of two strings in {0,1}^n is their bit-wise addition mod 2
- An important property of XOR
  - Theorem: Y a rand. var. over {0,1}^n, X an independent. uniform variable on {0,1}^n
  - Then Z:= Y XOR X is uniform var. on {0,1}^n
  - Proof: (for n = 1)
    - Pr[Z=0] = 1/2
- The birthday paradox
  - Let r1, …. rn element of U be independent identically distributed random variables
  - Theorem: when n = 1.2 * |U|^1/2 then Pr[there is an i != j: r_i = r_j] >= 1/2
  - Example: Let U = {0,1}^128 |U| = 2^128
    - after sampling about 2^64 random messages from U, some two sampled messages will likely be the same

Information Theoretic Security and The One Time Pad

- Symmetric Ciphers; definition
  - Def: a cipher defined over (k,m,c) (the set of all possible keys, messages and ciphers)
  - is a pair of “efficient” algorithms (E,D) (“efficient” means runs in polynomial time to the size of their inputs)
    - E: K * M -> C
    - D: K * C -> M
    - s.t. for all messages element M, key element K:L
      - D(k, E(k,m)) = m (consistency equation)
  - E is often randomized. D is always deterministic
- The One Time Pad
  - First example of a “secure” cipher
  - M = C = {0,1}^n
  - K = {0,1} ^ n
  - key = (random bit string as long as msg)
  - C := E(k,m) = k xor m

    - D(k,c) = k xor c
    - Indeed: D(k,E(k,m)) = D(k,k xor m) = k xor (k xor m) = (k xor k) xor m = 0 xor m = m
    - fast encryption and decryption but hard to use because keys are long
- Information Theoretic Security
    - Basic idea: CT should reveal no “info” about PT
    - Def: A cipher (E,D) over (K,M,C) has perfect secrecy if for every m0, m1 element of M len(m0) = len(m1) and forall c element of C Pr[E(K,m0) = c] = Pr[E(K,m1) = c] where k is uniform in K. (k <- r K (k is a random variable that uniformly sampled in the key space K))
    - Given CT can’t tell if m is mo or m1
        - true for all m0, or m1
    - no CT only attack on a cipher that has perfect secrecy
- Lemma: OTP has perfect secrecy
    - for every m, c Pr[E(k,m) = c] = (#keys k element k s.t. E(c,m) = C) / |K|
    - So for all m,c: #{k element K: E(k,m) = c} = constant
    - Proof:
        - For OTP if E(k,m) = c
        - k xor m = c => k = m xor c
        - #{k element K: E(k,m) = c } = 1 for all m,c
        - OTP has perf. sec.
- The bad news..
    - Theorem: perfect secrecy => |K| >= |M|
    - perf-sec => key-ken >= len-msg

Stream Ciphers and Pseudo Random Generators

- Review
    - Cipher over (k,m,c): a pair of “efficient” algorithms (E,D) s.t. for all m element M, k element K: D(k,E(k,m)) = m
    - weak ciphers
    - a good ciphers: OTP M = C = K = $\{0,1\}^n$
        - E(k,m) = k xor m, D(k,c) = k xor c
    - Lemma: OTP has perfect secrecy (i.e. no CT only attacks)
    - Bad news: perfect-secrecy -> key-len >= msg-len
- Stream Ciphers: making OTP practical
    - idea: replace “random” key by “pseudorandom” key
    - PRG: is a function G:$\{0,1\}^s$ -> $\{0,1\}^n$, n >> s (n is much larger than s)
        - (“eff” computable by deterministic algorithm)
    - c = E(k,m) := m xor G(k)
    - D(k,c) := c xor G(k)
    - stream ciphers cannot have perfect secrecy
        - need a different definition of security
        - security will depend on specific PRG
- PRG must be unpredictable
    - suppose PRG is predictable
    - then there is some i: G(k)|1,…i -> algorithm G(k) | i+1,….n
    - if can predict first G(k)|i,…,i -> G(k)|i+1

- PRG must be unpredictable
  - we say that G: k -> {0,1}^n is predictable if:
  - there is “eff” algorithm A and there is 1 < = i <= n - 1 s.t. Pr[A(G(k))I1,..,i = G(k)Ii+1] >= 1/2 + E
    - for some “non- negligible E (E >= 1/2^30)
    - the ability to predict the next i bit of the G(k) for some non negligible value
  - Def: PRG is unpredictable if it is not predictable
    - there is for all i: no “efficient” advisory or algorithm that can predict bit (i+1) for “non=negligible” E
- Weak PRGS (do not use for crypto)
  - linear congruential generator parameters a,b,p a and integers and p a prime
  - r[0] = seed of generator
    - compute r[i] <- a*r[i-1] + b mod p
    - output few bits of r[i]
    - i++
  - easy to predicate
  - glibc random():
    - r[i] <- (r[i-3] + r[i-31]) % 2^32
    - output r[i] >> 1
  - never use random() for crypto
- Negligible and non-negligible
  - In practice: E is a scalar and
    - - E non-neg: E >= 1/2 ^ 30 (likely to happen over 1GB of data)
    - - E negligible: E <= 1/2 ^ 80 (won’t happen over life of key)
  - In theory: E is a function E Z ^ >= 0 -> R ^ >= 0 and
    - E non-neg: there is a d: E(lambda) >= 1/(lambda^d) inf.often (E >= 1 / poly, for many lambda)
    - E negligible: for all d, lambda >= lambda_d: E(lambda) <= 1/lambda^d (E <= 1 / poly, for large lambda)
- Few Examples
  - E(lambda) = 1/2^lambda :negligible
  - E(lambda) = 1/lambda^1000: non-negligible
  - negligible to mean less than an exponential
  - non negligible to mean more than 1 / polynomial

Attacks on Stream Ciphers and The One Time Pad

- Review
  - OTP: E(k,m) = m xor k, D(k,c) = c xor k
  - Making OTP practical using PRG: G: K -> {0,1}^n
  - Stream cipher: E(k,m) = m xor G(k), D(k,c) = c xor G(k)
  - Security: PRG must be unpredictable
- Attack 1: two time pad is insecure
  - never use stream cipher key more than once
  - c1 <- m1 xor PRG(k)
  - c2 <- m2 xor PRG(k)
  - Eavesdropper does
    - c1 xor c2 -> m1 xor m2
  - Enough redundancy in English and ASCII encoding that:
    - m1 xor m2 -> m1, m2

- Real world examples
  - Project Venona
  - MS-PPTP (windows NT): (point to point transfer protocol)
    - from client to server concatenating messages then encrypting using one long key
    - all messages from the server are then encrypted using the same key
    - Need different keys for client -> server and server -> client
    - K = (K_sc, K_cs)
      - both sides know these keys
  - 802.11b WEP
    - avoid related keys
    - client and access point both have the key k
    - client m appends CRC(m) xor with PRG(IV ll k) sent using stream cipher
    - length of IV: 24 bits
      - repeated IV after 2^24 approximately equal 16m frames
      - changes after every packet is sent
      - on some 802.11 cards: iV resets to 0 after power cycle
    - For PRG in WEP
      - after about 10^6 frames can recover k
- A better construction
  - k -> (PRG) to frames
  - each frame has a pseudorandom key
- Yet another example: disk encryption
  - file encrypt to blocks on disk
- Two time pad: summary
  - never use stream cipher key more than once
  - network traffic: negotiate new key for every session (e.g. TLS)
  - disk encryption: typically do not use a stream cipher
    - as changes are made to the file will be leaking information about the contents of the file
- Attack 2: no integrity (OTP is malleable)
  - m encrypt(xor k) -> m xor k
  - (m xor k) xor p decrypt(xor k) m xor p
    - modifications to cipher text are undetected and have predictable impact on plaintext

Real-World Stream Ciphers

- Old example (software): RC4
  - expands from 128 bits to 2048 bits
  - simple generate 1 byte at a time
  - Used in HTTPS and WEP
  - Weaknesses
    - Bias in initial output: Pr[2nd byte = 0] = 2/256
    - Prob. of (0,0) is 1/256^2 + 1/256 ^ 3
    - Related key attacks
- Old example (hardware) CSS (badly broken)
  - linear feedback shift register (LFSR):
    - consists of cells where each cell contains one bit
    - tabs feed into xor
    - shifts the last bit falls off
    - seed = init state of LFSR
  - DVD encryption (CSS): 2 LFSRs
  - GSM encryption (A5/1,2): 3 LFSRs

- Bluetooth (E0): 4 LFSRs
- CSS: seed = 5 bytes = 40 bits
- Modern Stream ciphers: eStream
  - PRG: {0,1}^s * R -> {0,1}^n
  - Nonce: a non-repeating value for a given key
  - R = nonce
  - Nonce a value that is never going to repeat as long as the key is the same
  - E(k,m;r) = m xor PRG(k;r)
  - The pair (k,r) is never used more than once
    - reuse the key because (k,r) are unique
- eStream: Salsa 20 (SW+HW)
  - Salsa20: {0,1} ^ (128 or 256) * {0,1} ^ 64 -> {0,1}^n (max n = 2 ^ 73 bits)
  - first part is the seed 128 or 256 bits
  - second part nonce 64 bits
  - Salsa20(k;r) := H(k,(r,0)) || H(k,(r,1))
    - 64 bytes long

PRG Security Definitions

- Let G:k - > {0,1}^n be a PRG
- Goal: define what it means that
  - [k <-_r K, output G(K)]
    - is "indistinguishable" from a [r <-_r{0,1}^n, output r]
- Statistical Tests
  - statistical test on {0,1}^n
  - an algorithm A s.t. A(x) outputs "0" or "1"
  - "0" outputs not random
  - "1" outputs random
  - Examples
    - A(x) = 1 iff |#0(x) - #1(x)| <= 10 * sqrt(n)
    - A(x) = 1 iff |#00(x) - n/4| <= 10 * sqrt(n)
    - A(x) = 1 iff max-run-of-o(x) <= 10 * log_2(n)
- Advantage
  - Let G:K -> {0,1}^n be a PRG and A a stat, test on {0,1}^n
  - Define: advantage[A,G] := | Pr[A(G(K)) = 1] (k is chosen uniformly randomly from the seed space) - Pr[A(r) = 1] (r is a truly random string {0,1}^n) | element of [0,1]
  - if advantage is close to 1 -> A can distinguish the output of the generator from random
  - if advantage is close to 0 -> A cannot distinguish the generator from random
  - Example
    - A(x) = 0 -> Advantage[A,G] = 0
  - Example
    - Suppose G:K -> {0,1}^n satisfies msb(G(k)) = 1 for 2/3 of keys in K
    - Define statistical test A(x) as:
      - if[msb(x) = 1] output "1" else output "0"
    - Then
      - Advantage[A,G] = |Pr[A(G(k))=1] - Pr[A(r) = 1]| = 1/6
        - 1/6 is non-negligible
        - A breaks the generator g with advantage 1/6
- Secure PRGs: crypto definition
  - Def: We say that G:k -> {0,1}^n is a secure PRG if for all "efficient " statistical tests A: Advantage[A,G] is negligible (very close to 0)

- are there provably secure PRGs? unknown!!
  - but we have heuristic candidates
- Easy fact: a secure PRG is unpredictable
  - We show: PRG predictable -> PRG is insecure
  - Suppose A is an efficient algorithm s.t. Pr[A(G(K)|1…n) = G(k)|i+1] = 1/2 + epsilon
- Easy fact: a secure PRG is unpredictable
  - Define statistical test B as:
    - B(x) = [if A(x|1,…i) = x_i + 1 output 1 else output 0
    - r truly random string
    - Pr[B(r) = 1] = 1/2
    - k pseudorandom sequence
    - Pr[B(G(k)) = 1] > 1/2 + epsilon
    - Advantage[B,G] > epsilon
- Unpredictable PRG is secure
  - Theorem: if for all i element {0,…n-1} PRG G is unpredictable at pos. i then G is a secure PRG
  - If next-bit predictors cannot distinguish G from random then statistical test can
- More Generally
  - Let P_1 and P_2 be two distributions over {0,1}^n
  - Def: We say that P_1 and P_2 are computationally indistinguishable
    - if for all "efficient " statistical tests A |Pr[A(X) = 1] (from x <- P_1) - Pr[A(x) = 1] (from x <- P_2) | < negligible
  - Example:
    - a PRG is secure if {k <-_r K: G(k) } computationally indistinguishable uniform({0,1}^n)

Semantic security

- What is a secure cipher?
  - attacker's abilities: obtains one cipher text
  - possible security requirements:
    - attempt #1 attack cannot recover secret key
      - E(k,m) = m
    - attempt #2 attacker cannot recover all of the plaintext
      - E(k,m_0||m_1) = m_0 || E(k,m_1)
  - Recall shannon's idea:
    - CT should reveal no "info" about PT
- Recall Shannon's perfect secrecy
  - Let (E,D) be a cipher over (K,M,C)
  - (E,D) has perfect secrecy if for all m_0, m_1 element of M (|M_0| = |M_1|)
    - {E(k,m_0)} = {E(k,m_1)} where k <- K
  - (E,D) has perfect secrecy if for all m_0, m_1 element of M (|M_0| = |M_1|)
    - {E(k,m_0)} computationally indistinguishable {E(k,m_1)} where k <- K
    - but also need adversary to exhibit m_0 and m_1 element of M explicitly

- Semantic Security (one time key)
  - For b =0,1 define experiments EXP(0) and EXP(1) as:
  - b element of {0,1} two challengers
    - picks random key
  - adversary A outputs two messages element of M
  - adversary is trying to break the key
  - The challenger outputs the encryption of m_0 or m_1
  - encryption <- E(k,m_b)
  - for b = 0,1 : W_b := [event that EXP(b) = 1] b' element of {0,1}
  - Advantage[A,E] = |Pr[w_0] - Pr[w_1]| element of [0,1]
    - the output of the adversary
- Semantic Security (one time key)
  - Def: E is semantically secure if for all "efficient" A
    - Advantage[A,E] is negligible
      - no efficient advisory can distinguish the encryption of m_0 from m_1
  - for all explicit m_0 and m_1 element of M: E(k,m_0) computationally indistinguishable E(k,m_1)
- Examples
  - Suppose efficient A can always deduce LSB of PT from CT
  - E = (E,D) is not semantically secure
  - challenger choses random key
    - one message ends in 0 and one ends 1
  - forward cipher text advisory A
  - Advisory A outputs the LSB(m_b) = b
  - Advantage[B,E] = | Pr(EXP(0) = 1] = Pr[EXP(1) = 1] | = 1
- OTP is semantically secure
  - challenger adversary
  - advisory sends the messages to the challenger m_0 and m_1
  - challenger sends the encryption of the messages to the adversary
    - OTP is the xor of the key and the message
  - For all A: Advantage[A,E] = | Pr[A(k xor m_0) = 1] - Pr[A(k xor m_1) = 1] | = 0
    - the distribution of k with anything we get uniform distribution
    - for both cases algorithm A is given the same distribution of inputs

Stream ciphers are semantically secure

- Stream ciphers are semantically secure
  - Theorem: G:K -> {0,1}^n is a secure PRG -> stream cipher E derived from G is semantically secure
    - for all semantically secure adversary A, there is an a PRG adversary B s.t.
      - Advantage[A,E] <= 2 * Advantage[B,G]
- Proof: Let A be a semantic security adversary
  - challenger outputs the encryption of message
    - choses a random stream r
    - m xor G(k)
    - adversary cannot tell that we switched from pseudorandom to truly random encryption
    - encrypt using r instead G(k)
      - m xor r <= OTP
  - adversary outputs message to the challenger
  - For b=0,1: W_b := [event that b' = 1].
  - Advantage_ss[A,E] = |Pr[W_0] - Pr[W_1] |

- Proof: Let A be a semantic secure adversary
  - Claim 1: |Pr[R_0] - Pr[R1] | = Advantage_ss(A,OTP) = 0
  - Claim 2: there is a B: |Pr[W_b] - Pr[R_B]| = Advantage_prg[B,G] for b = 0,1
  - Advantage_ss[A,E] = |Pr[W_0] - Pr[W_1]| <= 2 * Advantage_prg[B,G]
- Proof of claim 2: There is a B: |Pr[W_0] - Pr[R_0}\ = Advantage_prg[B,G]
  - algorithm B:
    - adversary A outputs two messages
-
-
-
-
-
-
- sdfsdf
-
-