COMP4109 Midterm 1 General Notes

William Findlay

October 12, 2019

1 Types of Cryptography

- symmetric key
  - ▶ shift ciphers
  - ▶ block ciphers
  - ▶ stream ciphers
- asymmetric key (public-private key)
- hashing
- protocols

2 Security Notions Models

- three components of a model
  1. attack model
     - ▶ ciphertext only attack (P) (COA)
       - ◦ attacker attempts to decrypt ciphertext to plaintext
     - ▶ known plaintext attack (P) (KPA)
       - ◦ attacker knows one or more plaintext-ciphertext pairs
     - ▶ chosen plaintext attack (A) (CPA)
       - ◦ attacker chooses a plaintext and encrypts it to receive ciphertext
     - ▶ chosen ciphertext attack (A) (CCA)
       - ◦ attacker chooses a ciphertext and decrypts it to receive plaintext
  2. security goal
     - ▶ (IND) indistinguishability
       - ◦ ciphertext should be indistinguishable from random string
     - ▶ (NM) non-malleability
       - ◦ cannot modify ciphertext so it decrypts to another plaintext that makes sense
  3. level of security
     - ▶ information theoretic
       - ◦ attacker has unlimited resources at their disposal
     - ▶ complexity theoretic
       - ◦ attacker has resources bounded $O(p)$ where p is the security parameter
     - ▶ computational (realistic)
       - ◦ attacker has the resources of n computers
- two components of a notion
  - ▶ goal + attack model
  - ▶ e.g. IND-COA or NM-COA or IND-KPA, etc.

3 Unicity Distance

- expected minimum length of ciphertext needed to uniquely compute a secret key
- $\frac{\log_2 |K|}{R_L \log_2 |P|}$
  - ▶ where R_L is redundancy of the language
  - ▶ R_{English} is about 0.75

4 Definition of Symmetric Key Crypto

- 5-tuple (P, C, K, E, D)
  - P is plaintext space
  - C is ciphertext space
  - K is keyspace
  - E is encryption functions $e_k \in E$
  - D is decryption functions $d_k \in D$
  - where:
    - $\forall\, k \in K, m \in P,\ d_k(e_k(m)) = m$

5 Shift Ciphers

5.1 Caesar Cipher

- choose a key from $\mathbb{Z}_{|P|}$
- $c_i = p_i + k \bmod |P|$

5.1.1 Strengths

- none really, this sucks

5.1.2 Weaknesses

- easy to brute force
- weak to frequency analysis

5.2 Affine Cipher

- choose any a and b mod 26
  - except a gcd(a,26) must be 1
- $k = (a, b)$ where
  - $E_k(m) = (am + b) \bmod 26$
  - $D_k(c) = a^{-1}(c - b) \bmod 26$

5.2.1 Strengths

- better than caesar cipher
- two unknowns

5.2.2 Weaknesses

- use frequency analysis to solve for a and b
- not much better than Caesar really

5.3 Substitution Cipher

- permute P to get $\mathcal{A}$
- sub P_i fo $\mathcal{A}_i$

5.3.1 Strengths

- no strengths, don't use this

5.3.2 Weaknesses

- weak to CPA
- weak to KPA
- weak to COA
  - ▶ frequency analysis
  - ▶ exhaustive search won't work though

5.4 Vigenère Cipher

- choose some k_l as a plaintext string of length l
- encrypt $c_i = p_i + k_{i \bmod l} \bmod |P|$

5.4.1 Strengths

- much better than what we've seen so far
- if the length of the key is equal to the length of the message, very strong

5.4.2 Weaknesses

- can find candidate key lengths by factoring
- weak to frequency analysis
- multiple encryptions with same key opens up attacks

5.5 One-Time Pad

- like Vigenère except:
  - ▶ change key each time
  - ▶ perfect security if key length is equal to message length

5.5.1 Strengths

- perfect security for key length = message length
  - ▶ semantically secure in information theoretic security against COA

5.5.2 Weaknesses

- key can only be used one time
- key length the same as message length is kind of silly
  - ▶ why not just send the message over the secure channel in the first place
  - ▶ very long keys are impractical
- each key needs to be truly random
- has malleability
  - ▶ no authentication, only confidentiality

6 Block Ciphers

- confusion
  - ▶ many bits of c should depend on one bit of k
- diffusion
  - ▶ changing one bit of m should change about 1/2 bits of c
- considerations
  - ▶ key size not too small or too big
  - ▶ block size not too small or too big
  - ▶ high encryption/decryption rates

- ▶ easy to implement and analyze

6.1 Sub-Perm Networks (AES)

- some choices
  - ▶ 128 bit key with 10 rounds
  - ▶ 192 bit key with 12 rounds
  - ▶ 256 bit key with 14 rounds
- new gold standard for encryption
  - ▶ full version will never be broken
  - ▶ there are known attacks for reduced versions though

6.2 Feistel (DES)

1. right goes into function F
2. then F output gets xor'ed with left
3. swap left and right

- for DES, we do a 16-round feistel
  - ▶ 64 bit block size
  - ▶ 56 bit keylength
- standard for a long time
  - ▶ eventually replaced by AES
- small keysize is a problem
  - ▶ but what if we encrypt twice with two keys
  - ▶ 2DES ($E_{k_2}(E_{k_1})$)
  - ▶ but this allows meet in the middle attack (3 pair KPA attack)
- what about 3 keys
  - ▶ 3DES ($E_{k_3}(D_{k_2}(E_{k_1}))$)
  - ▶ now meet in the middle takes 2^{112} steps
  - ▶ no proof more secure, but fairly widely used

6.3 Lai-Massey (IDEA, FOX)

- don't worry about these

6.4 ARX (ChaCha20)

- don't worry about these

6.5 Block Cipher Modes

6.5.1 ECB

- encrypt each block independently
- no semantic security
- this sucks, don't use it

6.5.2 CBC

1. pad if necessary
2. xor m_0 with IV
3. encrypt new m_0
4. xor m_1 with c_0
5. and so on...

- this is pretty decent with random IV

6.5.3 CTR

- actually a stream cipher (size doesn't matter)

1. run a nonce appended with a counter through the encryption
2. xor that with plaintext
3. now you have ciphertext

- this is another good choice

6.6 Block Cipher Padding

6.6.1 Normal Padding

- take x is number of bytes smaller than block size
- append x sets of x
- if we had a perfect match, append one full block of padding
- this is kind of wasteful

6.6.2 Ciphertext Stealing (CBC)

- append all 0's
- encrypt as normal
- swap last two blocks
- truncate new last block by the number of 0's you appended

7 Stream Ciphers

- secret key and nonce generate a pseudorandom keystream
- encrypts a single digit at a time
  - ▶ keystream xor plaintext gives ciphertext
- should have long period
- should be IND truly random sequence
- forward and next-bit security

7.1 Synchronous

- keystream generated independently of p and c
- Alice and Bob must be synchronized
  - ▶ missing a single bit corrupts decryption
- single errors are not propagated

7.2 Asynchronous

- keystream depends on previous bits of c
- synchronizes itself
- bit errors will cause some bits to decrypt incorrectly, but can self-synchronize
- causes diffusion of plaintext

7.3 Stateful

- secret internal state that changes as keyystream computed

7.4 Counter-Based

- no internal state
- each block is defined by k, a nonce, and a counter

8 Hashing Functions

- one-way function
- easy to compute
- hard to reverse
- we don't know if these really exist
  - ▶ if $P \neq NP$, they exist
  - ▶ we're pretty certain they do

8.1 Properties

8.1.1 All Hashes

- compression
  - ▶ take any length and compress to some fixed length
- ease of computation

8.1.2 Cryptographic Hashes

- Preimage Resistance (PIR)
  - ▶ hard to find x given $h(x)$
- Second Preimage Resistance (SPIR)
  - ▶ given x, hard to find $x\prime$ such that $h(x) = h(x\prime)$
- Collision Resistance (CR)
  - ▶ hard to find any pair $\{x, x\prime\}$ such that $h(x) = h(x\prime)$

8.1.3 Desirable Properties

- resists length extension attacks
- hard to find messages with similar hashes
- non-malleable
- ideally, acts like random function

8.2 Finding Preimages

1. given y
2. try different $h(x)$ until $h(x) = y$

- we expect to be done in 2^n steps for binary alphabet

8.3 Finding Collisions (Naive)

1. store $(x, h(x))$ pairs
2. keep going until we find (x, y) and $(x\prime, y)$

- about 2^{n/2} steps (birthday attack)
- say n was large
  - ▶ this could take petabytes of storage

8.4 Rho Method for Finding Collision (Space-Efficient)

1. choose a random input x
2. set $H\prime_1 = H_1 = h(x)$
3. set $H_2 = h(H_1)$
4. set $H\prime_2 = h(h(H\prime_1))$
5. until we find $H\prime_i = H_i$
   - set $H_{i+1} = h(H_i)$
   - set $H\prime_{i+1} = h(h(H\prime_i))$

- summary of the method:
  - ▶ basically init H and $H\prime onsomerandomhash$
  - ▶ compute hashes for H and double hashes for $H\prime$ until we find $H = H\prime$
  - ▶ only store four values here, so $O(1)$ space
  - ▶ apparently it can be shown this takes $2^{n/2}$ steps still

8.5 Merkle-Damgard

- MD5
- SHA-1
- SHA-2
- (but not SHA-3, they realized it sucks by then)

1. pad last message block
2. take IV and message block 1 as input to compression function
3. take output of previous compression function and next block as input
4. keep going until last output is hash (sometimes extra work at this step)

- this SUCKS, don't use it
  - ▶ because of length extension attacks

8.5.1 Length Extension on Merkle-Damgard

- all you need to do is join mid-way through the hash function and keep hashing as normal
- append your message on the end

9 MACs

- three efficient algorithms (G, S, V)
  - ▶ G generates a key k
  - ▶ $S(k, m)$ generates a tag t
  - ▶ $V(k, m, t)$ verifies m with tag t
- provides data integrity and data origin authentication

9.1 Keyed Hashing

- use a secret symmetric key as well as message as input
- used for MACs and PRFs

9.2 Forgeries

- existential
  - ▶ there exist some forgeries
- selective
  - ▶ can create a message-tag pair for some chosen message m
- universal

- ▶ can create a message-tag pair for any message m

9.3 Generic Attacks of MACs

- make a MAC from an unkeyed has function if Merkle-Damgard
  - ▶ $H_k(m) = h(k||m)$
  - ▶ now we can compute $H_k(m||y)$ as selective forgery
  - ▶ only stipulation is we have to prepend original m
  - ▶ length-extension attack

9.4 HMAC

- hash (k xored with outter pad appended to the hash of (k xored with inner pad) appened to m)
  - ▶ only as secure as the hash function used

9.5 CBC-MAC

1. encrypt message as normal
2. discard every c_i except for last which is out MAC

- this is fine with **FIXED LENGTH**
  - ▶ we are as secure as underlying block cipher
- if we allow variable length messages, there is a length extension attack
  1. take some m, t pair and...
  2. send $m||m_1 \oplus t||m_2||...||m_l$ which will have same tag t

9.6 EMAC

- same as CBC MAC but we encrypt one extra time at the end with another key
  - ▶ allows arbitrary length
- can also do encrypt with k, decrypt with k2, then encrypt again with k

10 Randomness and PRFs (Chapter 7)

- next-bit security
  - ▶ given all previous bits, we shouldn't know anything more about next bit
- forward security
  - ▶ given current bit, we can't know anything about previous bits

10.1 Some PRNGs That Suck

- ASF software poker
  - ▶ never swapped the last card, so deck was not fully random
- Original Netscape SSL
  - ▶ relied on values that were not secret
  - ▶ relied on values that were not evenly distributed
  - ▶ relied on modular values that didn't have high period

10.2 Some PRNGs That Don't Suck (We Think)

- blum-blum-shub
  - ▶ difficulty of factoring
- blum-micalo
  - ▶ computing discrete logarithms modulo p is infeasible

10.3 PRFs

- use keyed hashing like MACs
- but they have stronger security requirements
  - ▶ they need IND with random sequence and unpredictablility
  - ▶ MACs just can't have any forgeries
  - ▶ if we took $PRF_2(k, x) = PRF_1(k, x)||0$, PRF_2 would be unacceptable as a PRF even if PRF_1 was secure
  - ▶ but it would still be acceptable as a MAC because we can't make any new forgeries