CS 370: Introduction to Security
04.25: RSA, Digital certificate

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab

Topics for today

- Public key cryptography
  - What is it?
  - What problem does it solve?
  - What is a popular public-key cryptography algorithm?
  - What can we do with the public-key crypto-algorithm in practice?

Symmetric key cryptography

- So far, we've talked about this world

Symmetric key cryptography

- Problems
  - How can we securely share the key between two parties?
  - How can we manage communications from/to multiple parties (100+)?

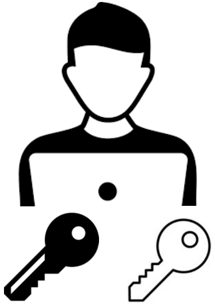
Symmetric key cryptography

- Problems
  - How can we securely share the key between two parties?
  - How can we manage communications from/to multiple parties (100+)?

- Solutions
  - What if I have two keys?
    - Key A that only can encrypt a message (but can’t decrypt)
    - Key B that can encrypt and decrypt a message
  - How can I leverage the two keys?
    - Share Key A to others
    - Do not share; keep Key B private

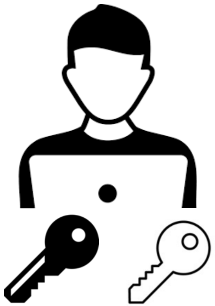
Public key cryptography

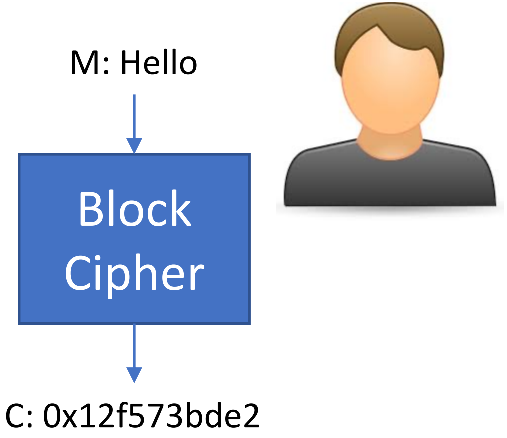
- The key idea
  - Asymmetric key cryptography
  - Use two different keys for encryption and decryption
    - Public key: share to others, only can encrypt a message
    - Private key: do not share, can encrypt and decrypt
  - What is possible?



Public key cryptography

- The key idea
  - Asymmetric key cryptography
  - Use two different keys for encryption and decryption
    - Public key: share to others, only can encrypt a message
    - Private key: do not share, can encrypt and decrypt
  - What is possible?





Public key cryptography

- The key idea
  - Asymmetric key cryptography
  - Use two different keys for encryption and decryption
    - Public key: share to others, only can encrypt a message
    - Private key: do not share, can encrypt and decrypt
  - What is possible?

Public key cryptography

- The key idea
  - Asymmetric key cryptography
  - Use two different keys for encryption and decryption
    - Public key: share to others, only can encrypt a message
    - Private key: do not share, can encrypt and decrypt
  - What is possible?
    - No one can decrypt a ciphertext unless they have the private key
    - We do not need to share the private key to anyone else
    - We share public key that can only encrypt the message

Public key cryptography: RSA

- RSA (Rivest, Shamir, Adleman)
  - A popular public key cryptography algorithm
  - It exploits the difficulty of prime factorization
    - To break RSA, an adversary solves the prime factorization of a large number
  - It is used for digital signature (we will revisit this later)

RSA

- Asymmetric key cryptography
  - Public key: e and N
  - Private key: d

- Key selection:
  - Choose two large prime number, p and q
    - Public key:
      - Set $N = pq$
      - Choose e as a coprime of $\phi = (p-1)(q-1)$
    - Private key:
      - Fine d that satisfies $de == 1 \pmod{\phi}$

RSA

- Key selection:
  - Choose two large prime number, p and q
    - Public key:
      - Set $N = pq$
      - Choose e (e.g., 65537) as a coprime of $\phi = (p-1)(q-1)$
    - Private key:
      - Fine d that satisfies $de == 1 \pmod{\phi}$
- Security
  - Concern: can an adversary guess the private key from the public key?
  - To do such an attack, the attacker needs to find ϕ
  - But we choose p and q as a large prime number; thus, it is difficult

RSA encryption

- Suppose we have
  - Public key: e, N
  - Message: M
  - Ciphertext: $M^e \bmod N$

RSA decryption

- We have
  - Public key: e, N
  - Message: M
  - Ciphertext: $M^e \bmod N$
- Suppose we also have
  - Public key: e N
  - Private key: d (that satisfies ed = 1)
  - Ciphertext: $C = M^e$
  - Plaintext: $C^d \bmod N$
    - $= (M^e)^d \bmod N$
    - $= M^{ed} \bmod N$
    - $= M \bmod N$ (N is a really large prime, so mostly it's N)

RSA-4096

- N

```
>>> n
94300162313076688501487627887743414045960390714020238212657873381998561681681396483075239596099727423614481972904672187684996447088678082779896359977776138389400094618
93811971402397228611383901773325321259010777553654865622984724944010604379009841301441764487806140346612303663579718455548165265742251289534980309219758481925957858787
99446168652865946938887547013421958335603541458898859985233102756405213301150045330555227176327316853262195678419436714942441571041724570176803145478446833917315101830
88856681940225944856271324669491185023754645727393394123350591112266076929457503053224635114890484540850553592509484360925706475886219500022922117482666285104831675845
83315533957568315104232320670250600708758347303059147821341336205419089515531617207836027717015263175059127264155564477809166344370523152038595667063337410819626147392
61504146573604212524025625329042730131363602682044377326795554520903135271401816609380989125787711356372203148662219805667048555875256480930486742228216374620641072794
39035295803547382839528902460696618996010706014197280097861310233823234888621192701394780193796900301510396063855789518617879808500828987759386908963639259712107524427
2314777803224755716476436654020264220108971589745725860770832311
```

- p and q

```
>>> p
32068656324685270630293083389512032043113021564958908104074082522961856175155506846871162451521328939255338658509282518527589856290667522356720088237485538660191295524
66763080165651305435511312859214781036375072132558927069941823437366007834867296636112996194202733617438183660982840127747653230903017526198432501279953198838940252489
07397379843971080938071066796458439488172567320447317484321566200567720344593323665558811276211955099547645761253947551466759960870349272433501174519655039023335818685
46315844543573625502694906678244707907306961176386182217799611306289079541727783724838441308692085353299440403808273
>>> q
29405710472654850958671060473587380272033499777349031897972774267355140300355329654820633532129222264029138740441499983175231104660614751766804322996494698305495888552
75830646534910098483628642910172110351604197235439018569363033804440884203852197669902443119162881776394993390408980913691022609261096791810408703245340903925873967271
31915873792822333318309536841669683057109843537055633086239079548671678418825449122417399716632535551668617851522599535745800792690727833156308278625231616345078778788
62534221887593666478665101149415792855605030513596595792804362412144174514663486222431955289564169364164960140372807
>>> p*q == n
True
```

- e and d

```
>>> e
65537
>>> d
18849384718575836845896027058446964676474069889583977304207060764212294704213288583979822675876320692881116222058550218329698543675506704371830299154811288476755744686
33122309922202781555596519725201973060912784929930859509301614305911758508400276501043488636804559844677375802873099344914795687645505468558344783111507028292873460032
08458501447312875244509618473165199111897193846400889052085900271738228698980203454999061232737047023478870918523805193489875712660735033176924808974589217149348121427
35615339936478018182357360775902688920932997528554152051755386172249044170100756671761641820728219898456478814438777554613519848613000196946332897123652219249131582620
18419600479189059635429627251759483101314923287035484839408409780479225523102668224739334748234253874639242903082590778667288728135476436491806847150495168056124486222
04967505131689556757368561851417851628405515960727363119875634155780743786262756894688981786624749458184135230246829545964509695021029098701475660700811758053804335333
34248118321456892350272682042203653282995523452563819502703960150673047768164870581311920830986731370034223814398610511624078011843818974900250458094933776149648209294
86455693297913060447945831290928211405464145580186799079751763
```

RSA-4096 (cont'd)

- Encryption

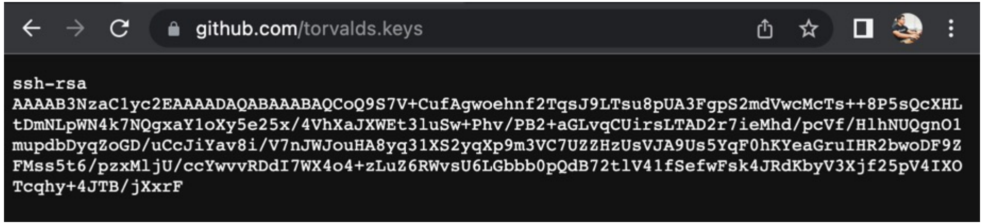
```
>>> m = 12345
>>> c = pow(m, e, n) # m ** e % n
>>> c
443784706377787352396086118109422823782079974501479680631743535246318476586590869537073655088774638869934500492985128744921003048888943298533486709895058774237372484178
040079575874499290413374847062808224508941879682031434528457814176924662901174674588647695046357307850784382343408121122220430499568600055483447589046769763784724684
486322116873999832137123999826443871919655383966531625400636903013509472170170783645161654011399643660644599309304122890591448950111746282695420794882322431789196358093
936982430073983958364444852924091631238933353822684767095879201080640206119212819746473211673614112631764365509831063157125125503814588660817393614276862437674744056716
969212531466451790154616008582626051573044131596636322530763889773664859797300591802077382373834580062670252137693091857292315461595441787257177342422993355220095280422
942476481060100821238941445817250493239517641421074872678593347890043728988899931048881456864826850140750986137042474998371731154209998675414707669442684372571942321
225313027498398049964794300732266201011397284720526380534464581760035084430444866978745447307757861316139293737708328715645502655290769724794147178642621760920172068445
561283568170818845183097678931422241453739518372479767621293111
```

- p and q

```
>>> _c = pow(c, d, n) # c ** d % n == m ** (de) % n == m % n == m
>>> _c
12345
>>> 
```

RSA-4096 (CONT'D)

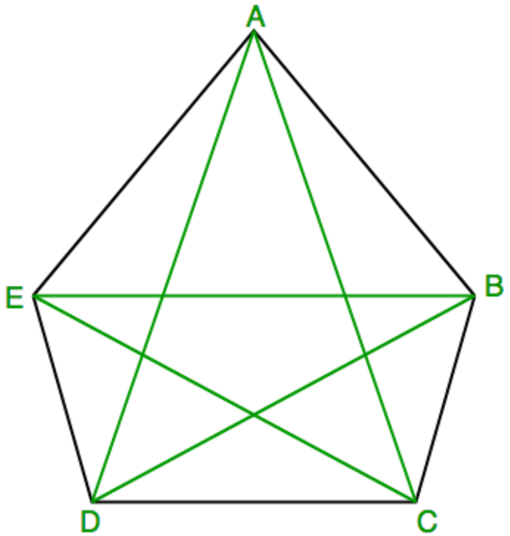
- Benefits:
  - We can publicize our public keys
  - Encryption/decryption
    - Anyone can encrypt their Ms with your public key: (e, N) is public and $C := M^e \bmod N$
    - Only you can decrypt this message: d is private $M^{ed} == M^1 == M \pmod N$



Public key cryptography and key exchange

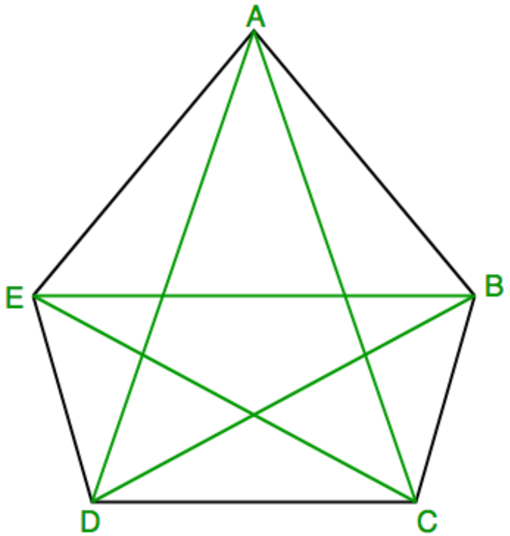
- Suppose we have five people (A, B, C, D, E)
  - How many keys do we need to make them communicate securely?
  - How can we make everybody be able to talk to anyone?

- In block cipher
  - Need 1 key for two of them (A and B) can talk securely
  - How many keys do we need for all?
    - A-B, A-C, A-D, A-E
    - B-C, B-D, B-E
    - C-D, C-E
    - D-E
    - 10 keys ($_5C_2 = 10$)



Key exchange in symmetric key cryptography

- Key exchange complexity
  - A key per each pair of people
  - ${}_nC_2 = N(N-1)/2$
  - $O(N^2)$



Key exchange in asymmetric key cryptography

- Key exchange complexity
  - Each person shares their public key to everybody
  - But they do not share their private key
  - We need O(N) keys

- Benefit: it scales!
  - Suppose we have a crypto conference with 400 folks
  - Symmetric key crypto: we need 400 x 399 / 2 keys for secure comm.
  - Asymmetric key crypto: we only need 400 public-private key pairs

RSA and Digital Signature

- Digital signature
  - A mathematical scheme for verifying the authenticity of digital messages
  - RSA can be used for the digital signature

- Recap: encryption and decryption
  - Encryption is applying the public exponent to a plaintext: $C = M^e \bmod N$
  - Decryption is applying the private exponent to a ciphertext: $M = C^d \bmod N$

RSA and Digital Signature

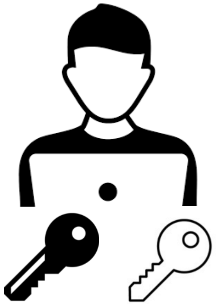
- Digital signature
  - A mathematical scheme for verifying the authenticity of digital messages
  - RSA can be used for the digital signature

- Recap: encryption and decryption
  - Encryption is applying the public exponent to a plaintext: $C = M^e \bmod N$
  - Decryption is applying the private exponent to a ciphertext: $M = C^d \bmod N$

- Encryption and decryption for digital signature
  - Encryption is applying the private exponent to a plaintext: $C = M^d \bmod N$
  - Decryption is applying the public exponent to a ciphertext: $M = C^e \bmod N$

RSA and digital signature

- Digital signature
  - A mathematical scheme for verifying the authenticity of digital messages
  - RSA can be used for the digital signature
- Encryption and decryption for digital signature
  - Encryption is applying the private exponent to a plaintext: $C = M^d \bmod N$
  - Decryption is applying the public exponent to a ciphertext: $M = C^e \bmod N$



RSA and Digital Signature

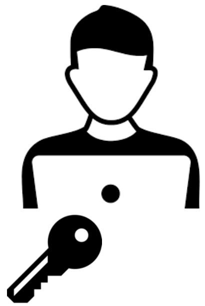
- Digital signature
  - A mathematical scheme for verifying the authenticity of digital messages
  - RSA can be used for the digital signature

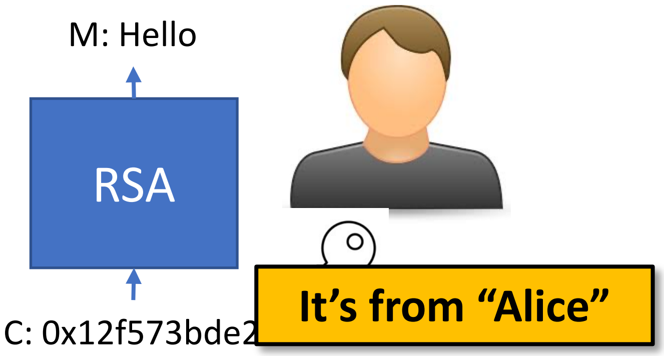
- Encryption and decryption for digital signature
  - Encryption is applying the private exponent to a plaintext: $C = M^d \bmod N$
  - Decryption is applying the public exponent to a ciphertext: $M = C^e \bmod N$

RSA AND DIGITAL SIGNATURE

- Digital signature
  - A mathematical scheme for verifying the authenticity of digital messages
  - RSA can be used for the digital signature

- Encryption and decryption for digital signature
  - Encryption is applying the private exponent to a plaintext: $C = M^d \bmod N$
  - Decryption is applying the public exponent to a ciphertext: $M = C^e \bmod N$





RSA and Digital Signature: Example

- Suppose:
  - A wants to send "I would like to buy a Pizza for Sanghyun if I get A from CS 370"
  - A encrypts the plaintext with their private key
  - $C = m^d \bmod N$

RSA and Digital Signature: Example

- Suppose:
  - You want to send "I would like to buy a Pizza for Sanghyun if I get A from CS 370"
  - You encrypt the plaintext with their private key
  - $C = m^d \bmod N$

- Now Sanghyun receives C
  - $C = m^d \bmod N$
  - SH has the public key e
  - and runs C^e
    - $== m^{de}$
    - $== m^1$
    - $== m \pmod{N}$
    - == "I would like to buy a Pizza for Sanghyun …"

Important:
- C only can be generated with the private key
- C can be decrypted by anyone who has "e"
- We know the private key owner endorsed M
- We call it as "signing"

RSA AND DIGITAL SIGNATURE

- Can we use symmetric key for “signing”?
  - A-B, A-C, A-D, A-E
  - B-C, B-D, B-E
  - C-D, C-E
  - D-E

- Suppose M was encrypted with the key shared between D and E
  - Either D or E can generate the message -> ambiguity
  - Only D or E can verify that -> it is not public
  - They must leak the key D-E for verification -> the secret key need to be publicized

Topics for today

- Public key cryptography
  - A symmetric key cryptography
  - Benefits:
    - We don’t need to share our private keys
    - Only the private key owners can decrypt C generated by the public key
  - RSA-4096
  - In practice, we use it:
    - For the secure communication
    - For the digital signature (i.e., “signing”)

Thank You!

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab