Foundations

first, some terms
- cryptography: the art and science of encryption; it is performed by cryptographers
- cryptanalysis: the art of science of breaking the encryption; it is performed by cryptanalysts
  - this is you this term!
- cryptology: involves both cryptography and cryptanalysis; it is performed by cryptologists
- cryptosystem: a set of cryptographic algorithms needed to implement security for a service

so, what's the purpose of cryptography?
- primarily, to achieve confidentiality
- but also for authentication, integrity verification, and nonrepudiation

- confidentiality: the information is hidden from parties who cannot decrypt
- authentication: the receiver can verify the origin of the data
- integrity verification: ensures that the data was not tampered with during transit
- nonrepudiation: the sender must be unable to deny having sent a message

parties involved
- usually, only two parties: sender (Alice) and receiver (Bob)
- but there are protocols that require more than two parties

basic working mechanism
- enciphering: plaintext → ciphertext (through encryption)
- if E is an encryption function, P is plaintext, and C is ciphertext, then: $E(P)=C$

- deciphering: ciphertext → plaintext (through decryption)
- if D is a decryption function, C is ciphertext, and P is plaintext, then: $D(C)=P$

cipher
- an encryption algorithm
- most ciphers require a key for encryption and decryption

restricted algorithms
- do not use a key
- security depends on the secrecy of their implementation
  - i.e., their methods are “secret”

- issues
  - each algorithm must be unique
  - useless for teamwork
    - a new algorithm is needed every time someone leaves
    - Gourd: Quickbooks escrow account
  - security of the algorithm is in question
    - can't be verified by third-party experts
  - plus, reverse engineering (CSC 448/CYEN 404)

algorithms with key(s)
- modern cryptography introduces a key to tackle to issues involved with restricted algorithms
- the security then relies on the secrecy of the key
  - instead of the secrecy of an algorithm

keyspace

- the range of keys an algorithm can use for encryption and decryption purposes
  - i.e., all possible keys for any algorithm

unconditionally and conditionally secure algorithms

- unconditionally secure algorithms
  - can never be broken, even with an infinite computing power and time
  - only one-time pad (OTP) is proven to be unconditionally secure
  - you can try all possible keys, but that would give you all possible outcomes
    - so you would not know which one is an actual message
    - but what if we know something about the original message?
- conditionally/computationally secure algorithms
  - can be broken; however, the cost of breaking them is higher than the value of the message
  - most modern algorithms take billions of years to break
    - when an adequate-length key is used

key lengths, keyspace, and time complexity

- space
  - 4-digit numeric key: $10^4 = 10{,}000$
  - 10-digit numeric key: $10^{10} = 10{,}000{,}000{,}000$ (10 billion)
  - 4-character lowercase key: $26^4 = 456{,}976$
  - 4-character mixed case key: $52^4 = 7{,}311{,}616$ (7.31 million)
  - 4-character alphanumeric key: $62^4 = 14{,}776{,}336$ (14.78 million)
  - 4-character (with symbols) key: $94^4 = 78{,}074{,}896$ (78.07 million)
  - 5-character (with symbols) key: $94^5 = 7{,}339{,}040{,}224$ (7.34 billion)
  - 6-character (with symbols) key: $94^6 = 689{,}869{,}781{,}056$ (689.87 billion)
  - 8-character (with symbols) key: $94^8 = 6.10 * 10^{15}$ (6.10 quadrillion)
  - 12-character lowercase: $26^{12} = 9.54 * 10^{16}$ (95.43 quadrillion)
  - 15-character lowercase: $26^{15} = 1.68 * 10^{21}$ (1.68 sextillion)
  - 20-character lowercase: $26^{20} = 1.99 * 10^{28}$ (19.93 octillion)

- time?
  - CPU: 100 million per second
  - GPU: 3 billion per second
  - distributed: 12 billion per second
  - specialized hardware: 90 billion per second
  - fastest so far (cluster): 350 billion per second

  - let's use the CPU option
    - 4-digit numeric key: 0.0001 seconds (0.1 milliseconds – or one 10,000th of a second)
    - 10-digit numeric key: 100 seconds
    - 4-character lowercase key: 4.57 milliseconds
    - 4-character mixed case key: 0.07 seconds
    - 4-character alphanumeric key: 0.15 seconds
    - 4-character (with symbols) key: 0.78 seconds
    - 5-character (with symbols) key: 73.39 seconds
    - 6-character (with symbols) key: 1.92 hours
    - 8-character (with symbols) key: 1.93 years
    - 12-character lowercase: 30.26 years
    - 15-character lowercase: 531,855.45 years
    - 20-character lowercase: $6.32 * 10^{12}$ years (6.32 trillion years)

attack

any attempted cryptanalysis on a cryptosystem

complexity of an attack

depends mainly on three factors:

data complexity: the amount of data needed to break an encryption
processing complexity: the time needed to perform an attack
storage requirements: the memory needed to perform an attack

a good cryptosystem would not just worry about today
but also consider the computing power of tomorrow

security of an algorithm

depends on how difficult an algorithm is to break
if the cost to break an algorithm is higher than the value of the message, then it is considered safe
an algorithm can be compromised in the following ways:

total break: a cryptanalyst finds a key, *k*, such that $D_k(C)=P$
global deduction: a cryptanalyst finds another equivalent algorithm, without knowing a key
local deduction: a cryptanalyst finds the plaintext of an intercepted ciphertext
information deduction: a cryptanalyst gains some information about the key or plaintext

one-time pad

it is the only known unconditionally secure encryption algorithm
process:

a one-time pad, containing some long randomly generated keys, is shared beforehand
the key used has the same length as the message/plaintext
Gourd's XOR!
each key is used only once, then discarded
OK, not like Gourd's XOR

e.g.:

OTP: 2 3 5 1 20 2 5 6 2 1 8 23 4 9 6 4 5 3 15 1 12 7 16 12 15
first message: HEY
encryption: H + 2 = J, E + 3 = H, Y + 5 = D → JHD
decryption: J – 2 = H, H – 3 = E, D – 5 = Y → HEY

before sending the next message, discard the keys already used
to brute force this, you could try every possible key
but you would get every possible result
can you think of a way to work through this anyways?
at least with some constraints (e.g., language)

XOR

theoretically as secure as OTP
the nifty reversible property of XOR makes it easy to implement
if *E* is an encryption function, we can use this same function to encrypt and decrypt
e.g.:

$E_k(P)=C$
$E_k(C)=P$

k is the random string of bits used as a key

with XOR there's a 50/50 distribution of the bits 0 and 1 in the output
this shuffles the output bits quite well
which other bitwise operators like OR and AND fail to do

OR

input	key	output
0	0	0
0	1	1
1	0	1
1	1	1

75% 1s

AND

input	key	output
0	0	0
0	1	0
1	0	0
1	1	1

25% 1s

XOR

input	key	output
0	0	0
0	1	1
1	0	1
1	1	0

50% 1s

despite being a promising concept, XOR has several limitations related to the key generation
true random keys are almost impossible to generate
key length: keys have to be as long as the message
key exchange
…

but for limited (personal) uses (with long enough keys/plaintexts), it works fine