Cryptography

Lecture 02

Software Security Engineering

Winter 2023
Thompson Rivers University

Imagine that Alice (A) is talking to Bob (B).

E.g., over the phone line.

E.g., Alice is a web browser, Bob a server.

Eve (E) is an eavesdropper, trying to

read data

modify data

inject data

delete data

Eve is called the adversary (theory) or attacker (systems)

Eve is called the adversary (theory) or attacker (systems)

If Eve only reads data,
it is called passive adversary

Eve is called the **adversary** (theory) or **attacker** (systems)

If Eve only reads data,
it is called **passive adversary**

If Eve can modify, inject, and delete data,
it is called **active adversary**

Eve is called the **adversary** (theory) or **attacker** (systems)

If Eve only reads data,
it is called **passive adversary**

If Eve can modify, inject, and delete data,
it is called **active adversary**

If Eve can inject but cannot read,
it is called **blind adversary**

Eve is called the **adversary** (theory) or **attacker** (systems)

If Eve only reads data,
it is called **passive adversary**

If Eve can modify, inject, and delete data,
it is called **active adversary**

If Eve can inject but cannot read,
it is called **blind adversary**

If Eve has infinite time and memory,
it is called information theoretic adversary

Eve is called the **adversary** (theory) or **attacker** (systems)

If Eve only reads data,
it is called **passive adversary**

If Eve can modify, inject, and delete data,
it is called **active adversary**

If Eve can inject but cannot read,
it is called **blind adversary**

If Eve has infinite time and memory,
it is called **information theoretic adversary**

If Eve does not have infinite time and memory,
it is called **computationally bounded adversary**

Communication threat modelling

- Alice and Bob are communicating
- Eve controls the communication channel
  - taps the lines
  - is in same broadcast range
    - e.g., WiFi
  - or the lines run through Eve
    - Eve is router, ISP, AS
    - “on-path” adversary

Adversary can be not on-path and not blind

Adversary can be not on-path and not blind

e.g., Alice, Bob, and Eve are all in same Wifi range

Adversary can be not on-path and not blind

e.g., Alice, Bob, and Eve are all in same Wifi range

Eve can inject and read traffic …

Adversary can be not on-path and not blind

e.g., Alice, Bob, and Eve are all in same Wifi range

Eve can inject and read traffic …

… but not easily modify or delete

Example Protocol

- file system protocol
  - Mount
  - Open
  - Read
  - Write
  - Unlink
  - Umount

Alice (user) connects to Bob (remote FS)

Alice (user) connects to Bob (remote FS)

Alice issues commands:

Alice (user) connects to Bob (remote FS)

Alice issues commands:

mount

Alice (user) connects to Bob (remote FS)

Alice issues commands:

mount

open somefile

Alice (user) connects to Bob (remote FS)

Alice issues commands:

mount

open somefile

write something to some file

Alice (user) connects to Bob (remote FS)

Alice issues commands:

mount

open somefile

write something to some file

umount

etc.

What are some things Eve could do?

What are some things Eve could do?

Assume there is no security

and Bob is reachable on the Internet

Suppose Eve is on-path

- man-in-the-middle attack
  - read Alice's data
  - read Alice's FS usage (metadata)
- impersonation attack
  - impersonate Bob to Alice
  - may not "look" like Bob
- denial of service attack
  - delete traffic so Bob can't be used

Suppose Eve is Blind

- impersonate Alice to Bob
  - can read all files
  - can unlink (delete) all files
    - denial of service

Why can’t Eve impersonate Bob to Alice when Eve is off-path?

Amend protocol

- assume Alice and Bob have a **secret number**
  - every command must include this magic number
  - Bob ignores messages that don’t have this magic number
- what can Eve do?

Attacks

- non-blind
  - eavesdrop on any message to learn magic number
    - can impersonate Alice after
- blind
  - try every possible number to see if its the right one
  - brute-force guessing attack

Enter cryptography

Cryptography – lit. secret writing

Cryptography is the art of sending messages
so that no one else can read them.

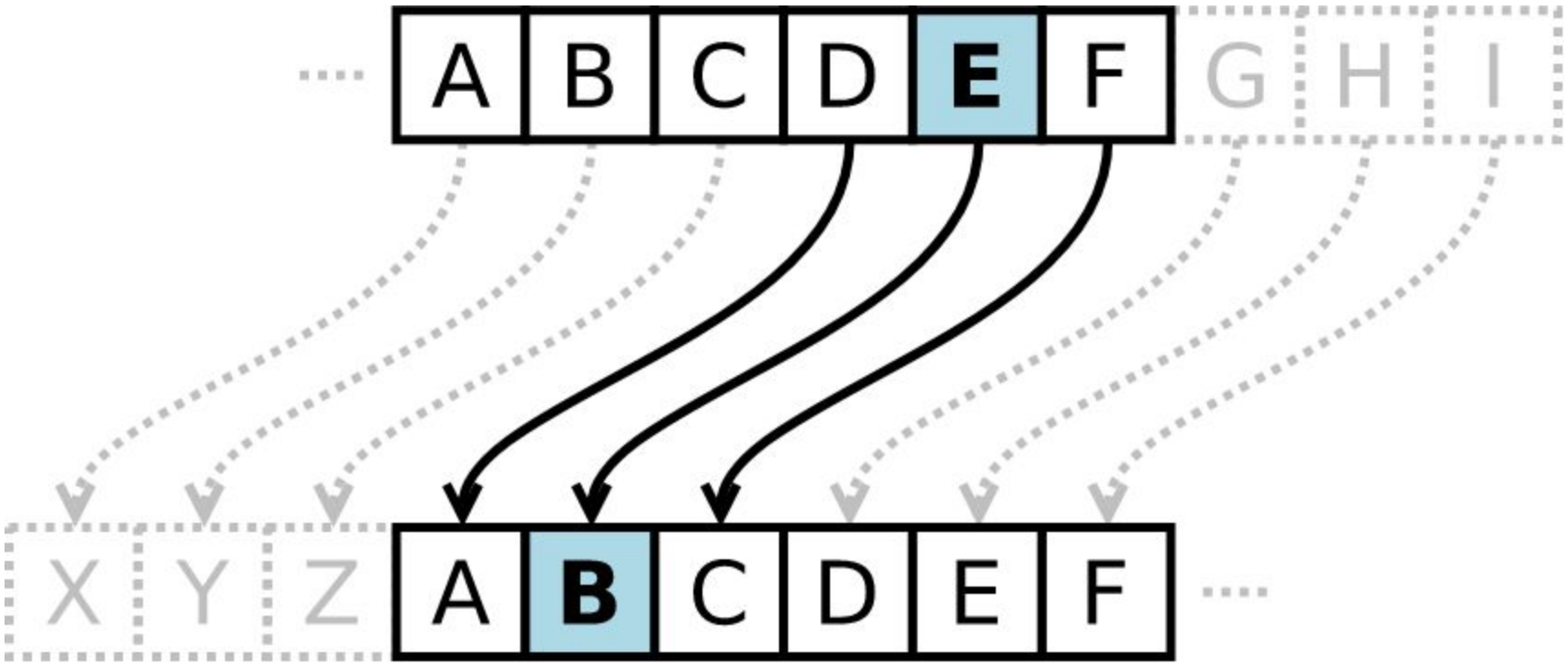
The actual message is called the **plain-text**.

The encrypted message is called the **cipher-text**.

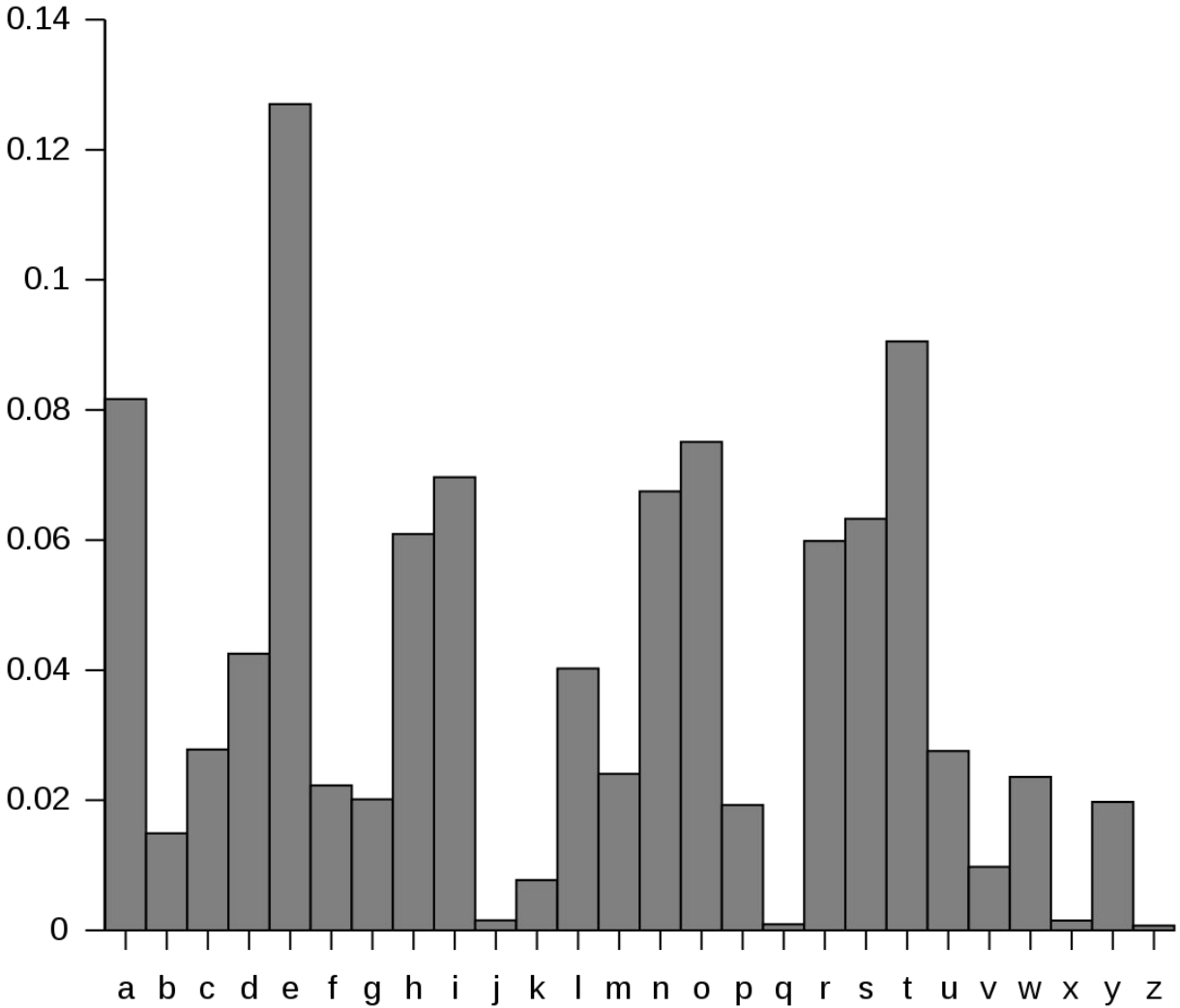
Cryptography has a long history, particularly for politics and warfare

Caesar Cipher

- Simple alphabet shift
- $E(x) = x + k \pmod{26}$
- $D(x) = x - k \pmod{26}$


A B C D E F G H I
X Y Z A B C D E F

Caesar Cipher is vulnerable to frequency analysis


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
a b c d e f g h i j k l m n o p q r s t u v w x y z

Eve without frequency analysis,

there’s not such much work to do:

only 25 key to try

Rhtsvvwz pz hdlzvtl

Breaking Caesar Cipher

- Qgsruuvy oy gckyusk
- Pfrqttux nx fbjxtrj
- Oeqpsstw mw eaiwsqi
- Ndporrsv lv dzhvrph
- Mconqqru ku cyguqog
- Lbnmppqt jt bxftpnf
- Kamloops is awesome
    - Key was 7

Letter Substitution Cipher

- better than a simple shift
- each letter has its own replacement
- ‘a’ = ‘r’, ‘b’ = ‘d’, etc.

GRAY FOX HAS ARRIVED
UKQN YGB IQL QKKOCTR

Is Letter Substitution Cipher vulnerable to frequency analysis?

Is Letter Substitution Cipher vulnerable to frequency analysis?

Yes.

Cracking Substitution

- most common is ‘e’
- guess for the next most common ones
- look to see if you get words or nonsense
- keep repeating until you work out the letters
- how many keys?

Cracking Substitution

- most common is ‘e’
- guess for the next most common ones
- look to see if you get words or nonsense
- keep repeating until you work out the letters
- how many keys? 26! (factorial)

Substitution

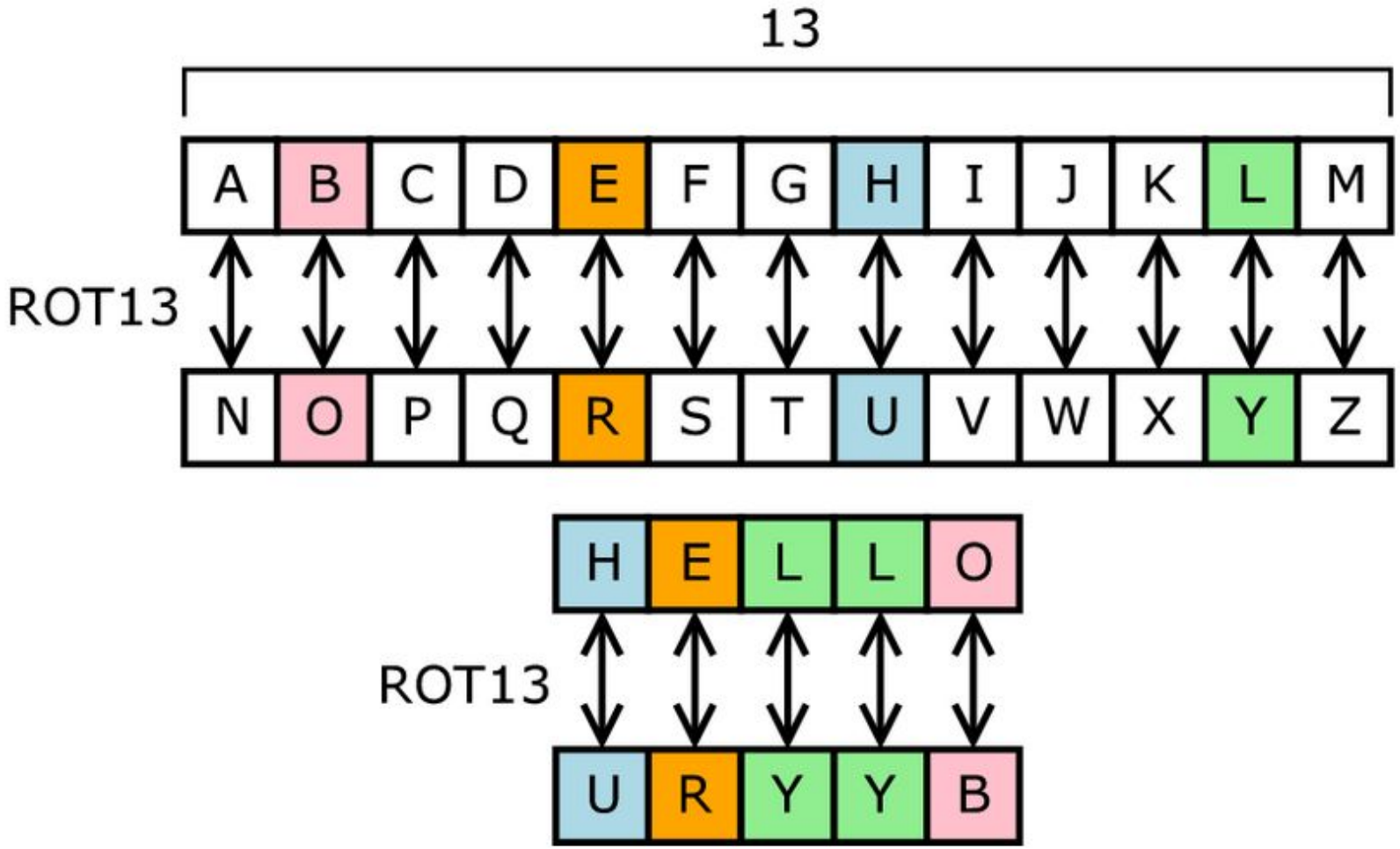
- can work for small encryptions
- e.g RTJA
  - this could be any four letter word
  - except for ones that have what property?

Substitution

- RTJA is EAST
  - when R=E, T=A, J=S, A=T
- RTJA is WEST
  - when R=W, T=E, J=S, A=T
- similarly, TGNNS can be JOLLY or WHEEL
  - impossible to prove which is which
- but longer messages or more messages make it clearer

ROT13

- a fixed-key variant of Caesar
- just add 13: A=N
- what is special about this encryption?


13
A B C D E F G H I J K L M
ROT13
N O P Q R S T U V W X Y Z
H E L L O
ROT13
U R Y Y B

Vigenere Cipher

- also known as “le chiffre indechiffrable”
- basically just repeated Caesar cipher
- the key is a sequence of shifts values that loop
  - e.g., Caesar cipher with key C is add CCCCCCCCCCC to the message
  - Vigenere cipher with key TRU is add TRUTRUTRUTRUTRU
- this worked from 1500s until 1800s
- so how do you break it?

Vigenere Cipher

- this worked from 1500s until 1800s
- so how do you break it?
  - ultimately it is a repeated Caesar cipher
  - assume key length is 5, the letters in position 1, 6, 11, 16, etc. are encrypted with same key
  - instead of solving Caesar cipher once, you need to solve it key length times
    - try by key length 1, then 2, 3, …

Key Count

- for short messages it can still work fine:
  - RTJA can be EAST or WEST for substitution
  - RTJA can be EAST or WEST or FALL for Vingenere

Key Count

- if the number of possible keys is more than the number of possible messages, it is possible to have perfect encryption
  - this is information-theoretically secure
- requires that for every same-length message M and encrypted message C there exists a key K so that $E_k(M) = C$
  - basically every possible message (of a particular size) is a candidate original message for the observed output
  - and barring any additional information, equally likely
  - but this means that the number of keys is at least as big as the number of messages

XOR (exclusive or)

- truth table
  - $0 \oplus 0 = 0$
  - $0 \oplus 1 = 1$
  - $1 \oplus 0 = 1$
  - $1 \oplus 1 = 0$
- properties
  - $x \oplus x = 0$
  - $x \oplus 0 = x$
  - $x \oplus 1 = \hat{}x$
  - $x \oplus (y \oplus z) = (x \oplus y) \oplus z$
  - $x \oplus y = y \oplus x$

XOR or binary strings

- operates bit-by-bit
  - $x_1\ x_2 \ldots x_n \oplus y_1\ y_2 \ldots y_n = (x_1 \oplus y_1)\ (x_2 \oplus y_2) \ldots (x_n \oplus y_n)$
- ASCII/UTF-8 for ‘b’ is 98, ‘h’ is 104
  - b: 0b01100010
  - h: 0b01101000
  - $b \oplus h$ = 0b00001010 = 8 + 2 = 10 or “newline” (\n)

One-Time Pad (OTP)

- encpty M by XORing it with K ($E_k(M) = M \oplus K$)
  - K is random binary string with the same length of M
- decrypt C by XORing it with K ($D_k(C) = C \oplus K$)
- $D_k(E_k(M)) = D_k(M \oplus K) = (M \oplus K) \oplus K =$
  $M \oplus (K \oplus K) = M \oplus 00\ldots0 = M$
- $|K| = |M| = |C|$
  - given a ciphertext c, choose any possible message m. Then $k_m = c \oplus m$ is a key that decrypts c to m
- another m' has a corresponding $k_m' = c \oplus m'$
- no way to "prove" that any particular k is the "correct" one if all are equally possible.

One-Time Pad

- an example of “perfect” security
- no frequency analysis can help decode it
- “information theoretically” secure
  - there is a key for all plaintext-ciphertext combinations
  - you can’t ever prove that it’s the “correct” deciphering

So why don’t we all use one-time pads?

So why don’t we all use one-time pads?

in fact some people do

e.g. diplomats, spies, etc.

use OTP for very important information

LFHHY ZAHBB JRNXK BYNFW KOZAT

VRETH JPCSU RUSYQ JWKMM WLQEL

PODYW JJLVJ XFSKL HPLGA ZXVZY

TSUIO XBMKI MBSND HPNPI OZVOZ

EYJWW OBXKR PKTVY YTKGK ATOPW

NHCJK FPNSV BMZZN QQZYN CYSDG

YIIUJ TWRRZ QHRDE YOVRJ HOCGY

HALOK NHIIN CAIDV RDTKH ZDZHP

OINDS CHOFE XGBVJ CAYSO IGBHU

K[illegible]SZX OZJIM DBRCY BNUVZ LFBXT

[illegible]KTI WWIFH IHNSF RUVVC UITRN

NQQNQ ZUBZB EPVJX NCZXY FBTEX

VEIOE HDVTN GSSNG LRZWG UKUQK

POFRI QCFAA NLTKE DXMDA QAIHU

HEIMQ LQTWP MVBNX MMUUK ACPXA

AYGFS ZNFDU SYMVX IYIPO RJCEK

PWOPQ JFWIO NYLIX GWTNC QQXXH

FSGNA UDTLB UHKAH HARHG TZVXN

UGBOA JXMFY HTUNH WCTXM OFLSY

A	ABCDEFGHIJKLMNOPQRSTUVWXYZ ZYXWVUTSRQPONMLKJIHGFEDCBA
B	ABCDEFGHIJKLMNOPQRSTUVWXYZ YXWVUTSRQPONMLKJIHGFEDCBAZ
C	ABCDEFGHIJKLMNOPQRSTUVWXYZ XWVUTSRQPONMLKJIHGFEDCBAZY
D	ABCDEFGHIJKLMNOPQRSTUVWXYZ WVUTSRQPONMLKJIHGFEDCBAZYX
E	ABCDEFGHIJKLMNOPQRSTUVWXYZ VUTSRQPONMLKJIHGFEDCBAZYXW
F	ABCDEFGHIJKLMNOPQRSTUVWXYZ UTSRQPONMLKJIHGFEDCBAZYXWV
G	ABCDEFGHIJKLMNOPQRSTUVWXYZ TSRQPONMLKJIHGFEDCBAZYXWVU
H	ABCDEFGHIJKLMNOPQRSTUVWXYZ SRQPONMLKJIHGFEDCBAZYXWVUT
I	ABCDEFGHIJKLMNOPQRSTUVWXYZ RQPONMLKJIHGFEDCBAZYXWVUTS
J	ABCDEFGHIJKLMNOPQRSTUVWXYZ QPONMLKJIHGFEDCBAZYXWVUTSR
K	ABCDEFGHIJKLMNOPQRSTUVWXYZ PONMLKJIHGFEDCBAZYXWVUTSRQ
L	ABCDEFGHIJKLMNOPQRSTUVWXYZ ONMLKJIHGFEDCBAZYXWVUTSRQP
M	ABCDEFGHIJKLMNOPQRSTUVWXYZ NMLKJIHGFEDCBAZYXWVUTSRQPO
N	ABCDEFGHIJKLMNOPQRSTUVWXYZ MLKJIHGFEDCBAZYXWVUTSRQPON
O	ABCDEFGHIJKLMNOPQRSTUVWXYZ LKJIHGFEDCBAZYXWVUTSRQPONM
P	ABCDEFGHIJKLMNOPQRSTUVWXYZ KJIHGFEDCBAZYXWVUTSRQPONML
Q	ABCDEFGHIJKLMNOPQRSTUVWXYZ JIHGFEDCBAZYXWVUTSRQPONMLK
R	ABCDEFGHIJKLMNOPQRSTUVWXYZ IHGFEDCBAZYXWVUTSRQPONMLKJ
S	ABCDEFGHIJKLMNOPQRSTUVWXYZ HGFEDCBAZYXWVUTSRQPONMLKJI
T	ABCDEFGHIJKLMNOPQRSTUVWXYZ GFEDCBAZYXWVUTSRQPONMLKJIH
U	ABCDEFGHIJKLMNOPQRSTUVWXYZ FEDCBAZYXWVUTSRQPONMLKJIHG
V	ABCDEFGHIJKLMNOPQRSTUVWXYZ EDCBAZYXWVUTSRQPONMLKJIHGF
W	ABCDEFGHIJKLMNOPQRSTUVWXYZ DCBAZYXWVUTSRQPONMLKJIHGFE
X	ABCDEFGHIJKLMNOPQRSTUVWXYZ CBAZYXWVUTSRQPONMLKJIHGFED
Y	ABCDEFGHIJKLMNOPQRSTUVWXYZ BAZYXWVUTSRQPONMLKJIHGFEDC
Z	ABCDEFGHIJKLMNOPQRSTUVWXYZ AZYXWVUTSRQPONMLKJIHGFEDCB


1 SYWAH TNMQG AIXSM MDBHQ TYRCO
2 OIBUA VBFAD HGSTR IQHJU CXHUW
3 ZIEZA UFOQY WMPKY SZWHJ LIYAS
4 JNHHY KCQKR ORPKJ PMYBU ZUKNA
5 JIHMW RVAUP XSHXR SGGSV
6 OPTQG PVCBT PMWLY MCEU
KBVT RXP
IRCXH
NXSDP
ACUPC
PIYJG
TARFP
Crypto Museum
cryptomuseum.com


RED PHONE


HOT LINE

Briefcases filled with one-time pad materials are exchanged by “diplomats” ahead of time

There’s also shortwave radio stations that broadcast out numbers called numbers stations.

(There were more during cold war)

The problem is that it’s not
usable for day-to-day needs.

One-Time Pad

- ensuring there is sufficient padding material ahead of time
  - i.e. before it is needed
  - why does this matter?
- securing the delivery of all the one-time pads
  - you don’t gain anything by sending a new one-time pad using old one-time pad (why?)
- making sure the pad is never used twice
- making sure the pad is generated randomly

Why is it called a one-time pad?

Two-time Pad

- $c_1 = m_1 \oplus p$
- $c_2 = m_2 \oplus p$
- $\longrightarrow c_1 \oplus c_2 = m_1 \oplus m_2,$
  - how can we break this? (hint: it is not straightforward)

Two-time Pad

- $c_1 = m_1 \oplus p$
- $c_2 = m_2 \oplus p$
- $\longrightarrow c_1 \oplus c_2 = m_1 \oplus m_2,$
  - how can we break this? (hint: it is not straightforward)
  - now, it is vulnerable to frequency analysis

How do you make sure a one-time pad is randomly generated?

Good Sources of Randomness

- physical phenomenon
  - dice rolling, coin flipping, radioactive decay
- HID events
  - HID stands for Human Interface Devices
  - time between keystrokes, mouse movements
- other I/O events
  - device interrupts, networking events

for something extremely important, like bank’s core encryption key,
use physical phenomenon like coin flipping

Types of Randomness

- true randomness
  - randomness from good sources
  - unpredictable from all information both before or after
    - to an information theoretic attacker
  - independent randomness
- pseudo randomness
  - numbers that “look” random but may fail some statistical tests
    - i.e. can be predicted with the right information
  - numbers not independent from other generated numbers

Pseudorandom Numbers

- pseudo random number generators (PRNGs) generate a stream of pseudo random using a **seed** and an **algorithm**
  - the **seed** should be truly random
  - the same seed generates the same stream
  - the algorithm generates a stream of numbers from the seed that are pseudorandom
- two types: **cryptographically suitable** and **not cryptographically suitable**

Cryptographically suitable is a requirement on the algorithm that a computationally bounded adversary cannot predict the next bit at any point in the stream

(knowing the stream before better than just guessing)

Cryptographically Suitable PRNG

- the stream of randomness cannot reveal the seed (why?)
- functions like `rand()` are not cryptographically secure
  - e.g. linear congruence generators (y = x * p (mod n))
  - given enough samples, you can start predicting the next ones, figure out seed
- if the seed is predictable then the stream is also predictable
  - e.g. using the time as the seed
  - does not mean algorithm isn't cryptographically suitable, only misused

What needs Randomness

- when using random numbers for cryptographic reasons it must be unpredictable
  - one-time pads, public keys, session keys, random tokens, web cookies
  - much of day to day Internet security relies on random numbers
  - when it needs to be unguessable
- when using random numbers for super important things use coin flips
  - e.g. long-lived signing keys for certificate authorities, banks, etc.
- when using random numbers for other purposes, they only need to be uniformly distributed
  - salts, challenges, nonces, initializations vectors, identifiers
  - if something should be unique use regular randomness
    - randomness based on time not guaranteed to be unique!
    - i.e. a function every other computer on the planet is trying to match
- or better yet just always use cryptographically secure randomness
  - LEAST SURPRISE and USABILITY, supports SAFE DEFAULTS

Where to find Randomness?

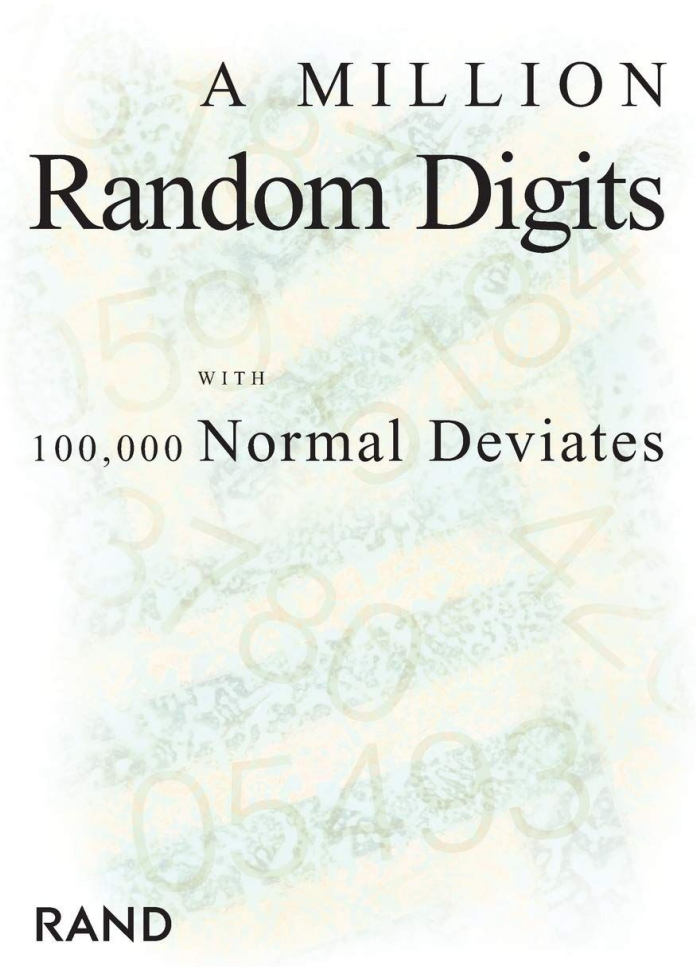
- /dev/random
  - bases on true source of randomness
  - try this command in linux -> `$ od /dev/random`
- /dev/urandom
  - use cryptographically suitable pseudo-random number generation (PRGN)
  - try this command in linux -> $ od /dev/urandom
  - this is okay for many uses
  - but not information theoretic uses!
    - KNOW YOUR ADVERSARY
- in higher-level programming languages random generators may use these
  - however many don’t by default or through the “obvious” way
  - SAFE DEFAULTS

Bad Sources of Randomness
is anything that can be predicted,
even if it started as a good source

A MILLION
Random Digits

WITH

100,000 Normal Deviates



RAND

If Eve can predict the next random bit that Alice chooses
even slightly better than random guess,
it is a bad source of randomness.

Basic Encryption Algorithms

- Alice and Bob share a Key *K*
- Alice generate a random one-time Pad by generating a random Pad with the K as the seed
- Bob decrypt by the same sequence of randomness

Stream Ciphers

- one-time pads in practice
  - XORs a long random string to the plaintext
- instead of a huge random string as the key, it is just a small seed
  - used for a cryptographically secure pseudo-random number generator (PRNG)
    - generates unpredictable numbers
    - provided adversary does not know the key
  - it creates the one-time pad
- this key cannot be reused (why?)

Stream Ciphers

- one-time pads in practice
  - XORs a long random string to the plaintext
- instead of a huge random string as the key, it is just a small seed
  - used for a cryptographically secure pseudo-random number generator (PRNG)
    - generates unpredictable numbers
    - provided adversary does not know the key
  - it creates the one-time pad
- this key cannot be reused (why?)
  - because it causes the two-time pad problem

Stream Ciphers are no longer secure against an information theoretic adversary. (Why?)

Stream Ciphers are no longer secure against an information theoretic adversary. (Why?)

key has a fixed length and it is vulnerable to the brute force attack

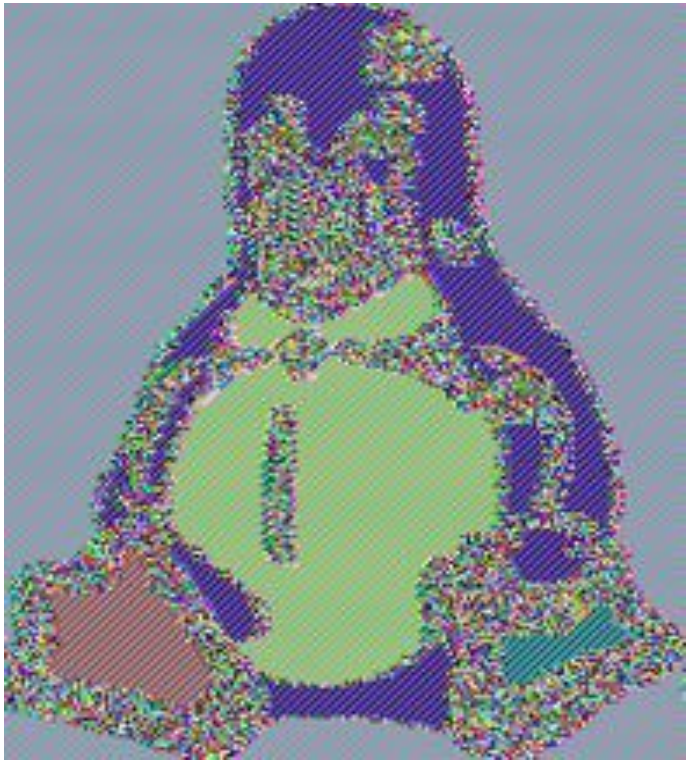
Block Cipher

- standard for modern crypto
  - was DES (data encryption standard)
  - now AWS (advanced encryption standard)
- block ciphers divide the message into blocks
  - block size 128 bit, 192 bit, 256 bit
- encrypts the blocks one by one
- decrypts the blocks one by one
- encryption is a function that maps some block value to another

What can go wrong?

Block cipher naively used to encrypt a picture:

Original image



Encrypted using Block Cipher -
ECB mode

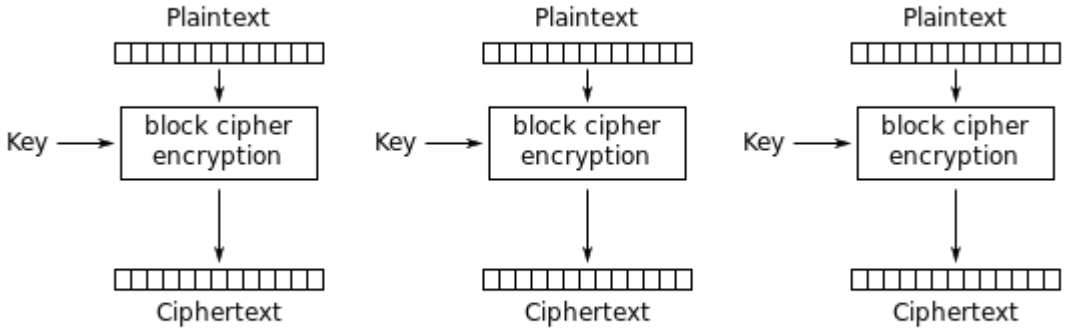
Clearly you can learn something about the plaintext by seeing the encrypted text

What happened?

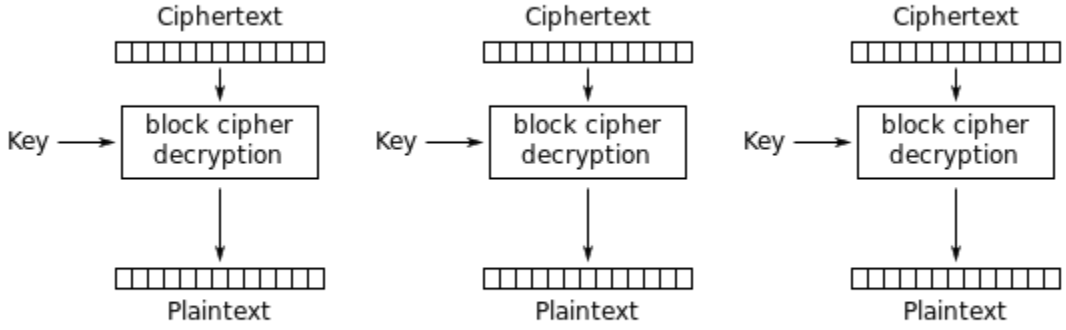
- each block is encrypted separately
- there's only so many different values that can appear as plaintext blocks
- if block value X to Y, and X appears a lot, the Y appears a lot in the ciphertext
- this is frequency analysis!
  - instead of a the level of letters it is at the level of blocks
- like encrypting a phone number digit by digit
  - if you know the area code is E(4) E(0) E(3), you learn where those appear in the rest

ECB Mode (never use this)

- Electronic CodeBook mode (ECB Mode)
- each block of data has an encrypted version and it’s looked up
- shows when the same original data is encrypted again
  - this is the encrypted penguin!



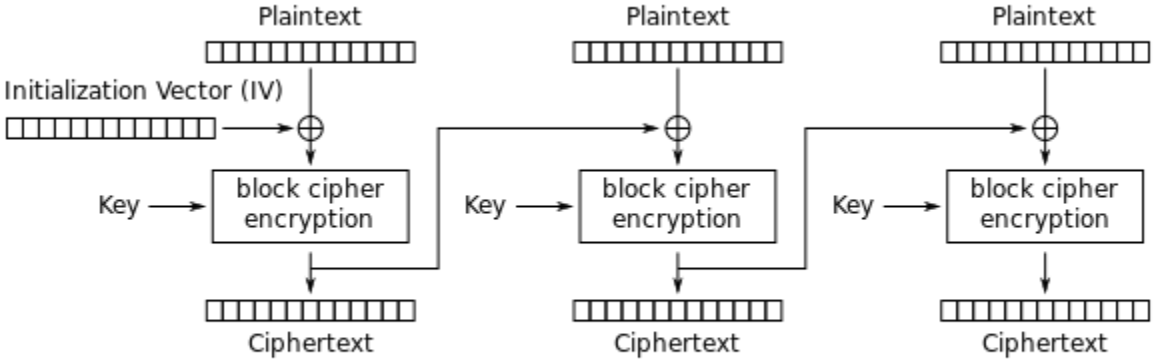
Electronic Codebook (ECB) mode encryption



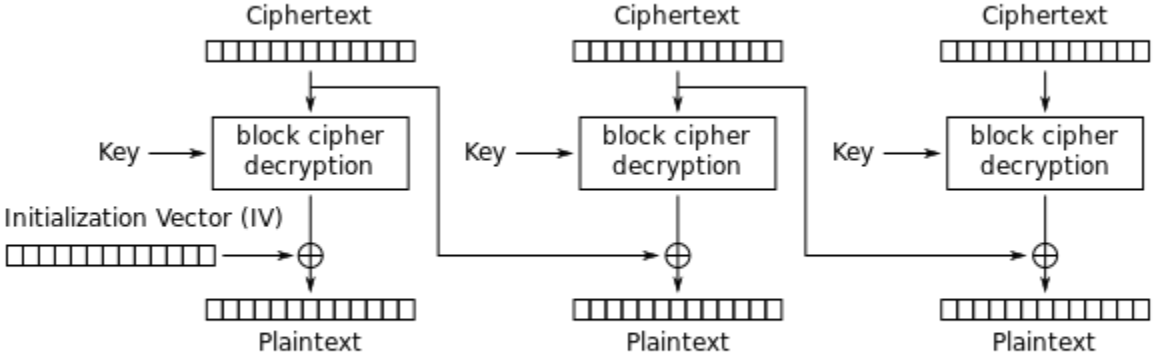
Electronic Codebook (ECB) mode decryption

CBC Mode (use this)

- CipherBlock Chaining (CBC) mode
- an **initialization vector** is used alongside the key
  - initialization vector is a uniform random bit string
  - it does not need to be a secret
- encrypting the block changes its value as it goes

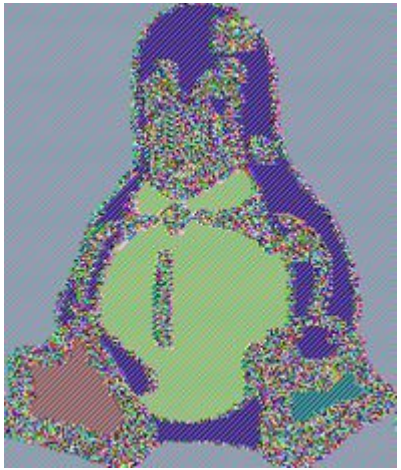


Cipher Block Chaining (CBC) mode encryption

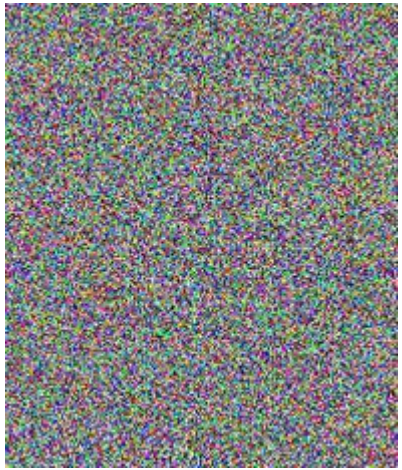


Cipher Block Chaining (CBC) mode decryption

Original image



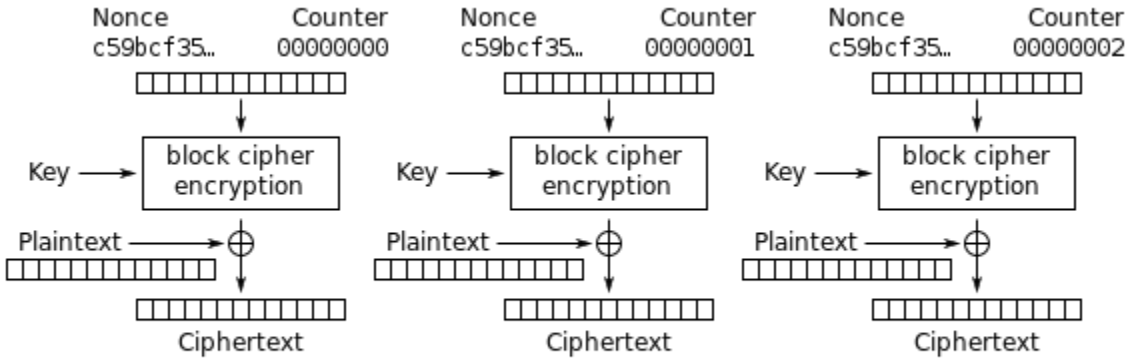
Encrypted using
ECB mode



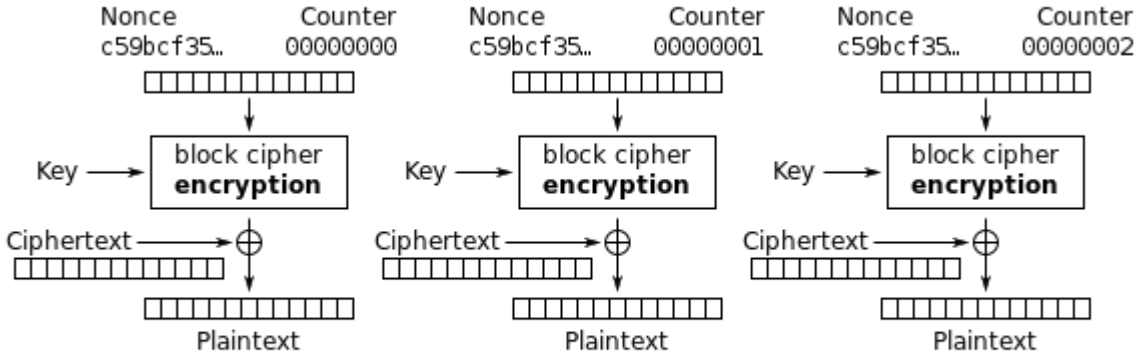
Modes other than ECB result in
pseudo-randomness

CTR Mode (also useful)

- counter mode
- has random access property
- turns block cipher into a stream cipher
- encrypt a sequence of numbers and XORs it to data



Counter (CTR) mode encryption



Counter (CTR) mode decryption

How do we decide if a crypto system is good?

How do we decide if a crypto system is good?

We define an adversary, give it capabilities,
and see if it works

How do we decide if a crypto system is good?

We define an adversary, give it capabilities,
and see if it works

The closer the adversary maps
to the real-world attacker the better

How do we decide if a crypto system is good?

We define an adversary, give it capabilities,
and see if it works

The closer the adversary maps
to the real-world attacker the better
(or more abilities we give it)
the better (if our system stays secure)

A strong adversary is a weak assumption.

A weak adversary is a strong assumption.

e.g. it is a strong assumption when you assume that the adversary cannot use a computer (is weak) but that makes security easy.

Some attacks on crypto

Known Plaintext Attack (KPA)

- you have some plaintext and its matching ciphertext
- if you can figure out the key then the encryption scheme is vulnerable to this attack
  - for these classical ciphers (like Caesar Cipher) we've seen they all reveal the key right away to KPA
- but why would you give away the plaintext?
  - isn't that the whole point of encryption?

Known Plaintext Attack (KPA)

- sometimes the plain-text is just public
  - e.g. encrypting HTTP GET or other fixed/structured headers
  - "*Defenders have protocols*"
- sometimes it can be guessed
  - if it is data with a limited range of values (like color) or corresponds to something observable
- or it can be leaked, published, compromised some other way
  - e.g. encrypted before publishing, or determined after the fact
- so protecting against this attack is a good idea

Chosen Plaintext Attack (CPA)

- you have access to a black box that will encrypt your messages using the same key
- you can feed it any input message
- so it’s like the known plaintext except you even get to choose the message
  - if having this choice doesn’t help you anymore than just having a known plaintext then it’s the same security as KPA

How can chosen plaintext attacks happen?

CPA examples

- Allies conspicuously mined the north sea and looked to see the description in encrypted German transmissions along with “all clear”
- US sending “low on supplies in Midway Island” in plaintext to see Japanese transmissions

secure against CPS $\longrightarrow$ secure against KPA

secure against CPS $\longrightarrow$ secure against KPA

adversary choice means always worst choice (for you)
if there is one

secure against CPS → secure against KPA

adversary choice means always worst choice (for you)
if there is one

ZERO PROBABILITY FAULTS

insecure against KPA $\rightarrow$ insecure against CPA

Chosen Ciphertext Attack (CCA) is similar to CPA except you pick message to be decrypted.

This is also called the lunchtime attack, as in,
using someone's computer when they're out for lunch
but you can't use it later when you need to decrypt a new message.

This is also called the lunchtime attack, as in,
using someone's computer when they're out for lunch
but you can't use it later when you need to decrypt a new message.

This is a form of an insider threat

Semantic Security:

a cryptosystem is **semantically secure** if
the only thing you can feasibly learn
about an encrypted message
without the key
is something about its length.

Security through Obscurity

- keeping how it works secret
  - secure as long as no one leaks
    - and then maybe still secure
  - no one figures out how it works and breaks it
  - some details about it are not stolen
- publishing how it works
  - secure as long as no one can figure it out
  - but all the world’s top cryptographers are trying
    - some do it in the open and would let us know

Kerckhoff's principle

- a system should be secure even when everything is know about the system except the key
- how it works is completely public
- a machine doesn’t need to be physically protected
- only a small secret key is what gives it security

Shannon’s maxim: the enemy knows the system

Security through Obscurity

- the opposite of Kerckhoff's principle
- relies on just not telling anyone how it works and hoping they don’t figure it out
- generally a bad idea and it fails frequently
- the crypto we all use is publicly inspected, and has been unsuccessfully attempted to be broken by the world’s best cryptographers
  - attacks aren’t in crypto; attacks bypass crypto
  - the cryptos that we used are the TIME-TESTED TOOLS
- NSA is an exception to this
  - LEVERAGE UNPREDICTABILITY

So we have encryption, but how do we manage the keys?

If I connect to GMail, I don’t need to already have a
secret key in my possession that
no one knows but me and Google.
How does it work?

Communication Channels

- insecure channel: standard, like speaking over the telephone
- secure channel: data is encrypted and you decrypt it to see
- authentic channel: data sent was not tampered with and you know the source of it
- secure authentic channel: data is secure and authentic

In practice, block ciphers like AES are
useful to provide security to te channel
but something else is needed for authenticity.

In practice, block ciphers like AES are
useful to provide security to te channel
but something else is needed for authenticity.

If only me and Google know the key that is a good start.

In practice, block ciphers like AES are
useful to provide security to te channel
but something else is needed for authenticity.

If only me and Google know the key that is a good start.

But how do we get an initial **key**?

Public Key Crypto

- Alice has a private secret key *K* and a public key *PK*
- Alice gives everyone *PK*
- to encrypt M, compute C = $E_{PK}(M)$
- to decrypt C, compute M = $D_K(C)$
- only Alice has K, so only Alice can compute D_k
- E and D are related, K and PK are related
- we assume that knowing E, D, PK, and lots of Cs and Ms does not reveal K

Public Key Crypto Swapped

- Alice has a private secret key K and a public key PK
- Alice gives everyone PK
- to encrypt M, compute $C = E_K(M)$
- to decrypt C, compute $M = D_{PK}(C)$
- only Alice has K, so only Alice can compute $C = E_K(M)$
  - why would we want this?

Public Key Crypto Swapped

- Alice has a private secret key K and a public key PK
- Alice gives everyone PK
- to encrypt M, compute $C = E_K(M)$
- to decrypt C, compute $M = D_{PK}(C)$
- only Alice has K, so only Alice can compute $C = E_K(M)$
  - why would we want this?

    Alice signatures the message

Public Key Signature

- I create a number that everyone can verify corresponds to a message
- everyone knows that only I can compute this number
- so everyone knows that only I can authorize its computation
  - i.e. signing the message by computing the number

Public Key Signature

- somewhat like a real-world signature
  - only I can create the signature
- exception
  - private keys can be stolen/copied
  - may not know it is compromised
  - signing things you do not agree with

Public key crypto is slow.

So usually we use it only to exchange encryption keys used for block ciphers like AES, and to sign short digests of messages (more on digests later).

Key Exchange

Alice wants to talk to Bob

She picks random key k
encrypts with Bob's public key PK $E_{PK}(k)$
and gives it to Bob.

Alice wants to talk to Bob

She picks random key k
encrypts with Bob's public key PK $E_{PK}(k)$
and gives it to Bob.

What can go wrong?

Perfect Forward Secrecy:
the compromise of a long-term secret does not impact the secrecy of previous (finished) sessions.

Example: suppose Google has a public key pair.

Everyone used the public key to send Google an encrypted session key.

Example: suppose Google has a public key pair.

Everyone used the public key to send Google
an encrypted session key.

If Google’s long-term private key is ever compromised
anyone who recorded previous traffic can decrypt
session key and then decrypt everything.

Key Negotiation and Discrete Logarithms

Modular Multiplication (mod 17)

- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 15$
- $2^6 = 13$
- $2^7 = 9$

Discrete Logarithms Trapdoor

- given 2 and 7, computing 2^7 mod 17 is easy
- given 9, working out that it is 2^7 (mod 17) is not
- given y, g, and n such that $y = g^x$ (mod n), find x
  - this is believed to be computationally infeasible in general for large n

Discrete Logarithms allow key negotiation!

Diffie-Hellman Key Negotiation

- we publicly agree on g and n (like 2 and 17, except 2048 bits long)
- I choose a random x, compute g^x (mod n) and share it.
- You choose a random y, compute g^y (mod n) and share it.
- from g^x you (or anyone) cannot compute x
- but I can compute $(g^y)^x$ and you $(g^x)^y$
- our key is therefore g^{xy}
- no message was ever sent containing g^{xy}
- what can go wrong?

Man-in-the-Middle (mitm)

- Eve sits between Alice and Bob and can modify their traffic
  - active network attacker
- Alice negotiates a DH key with “Bob” but it’s really Eve
- Eve the negotiates with Bob who thinks it’s Alice
- Eve decrypts all traffic from Alice and does what she wants, then relays it Bob
- e.g. `MITMPROXY` sits on a computer and can be used to read traffic that is encrypted

A

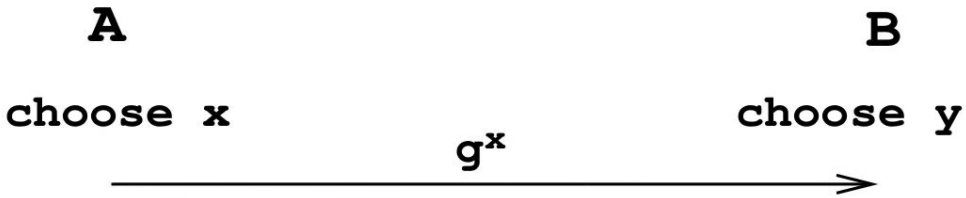
B

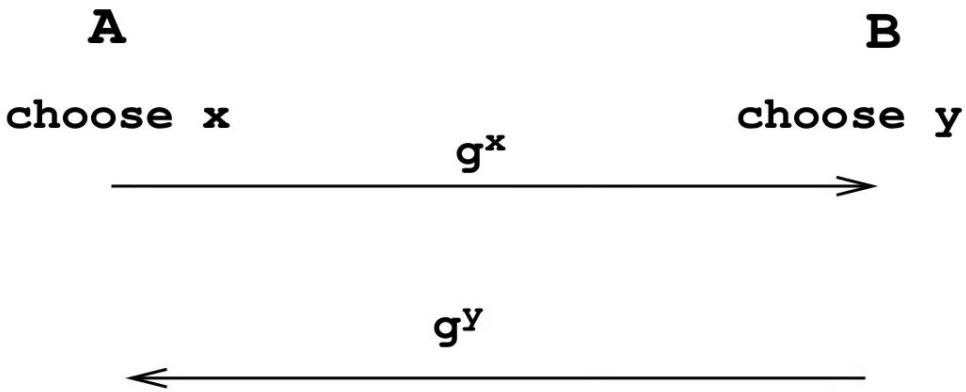
A

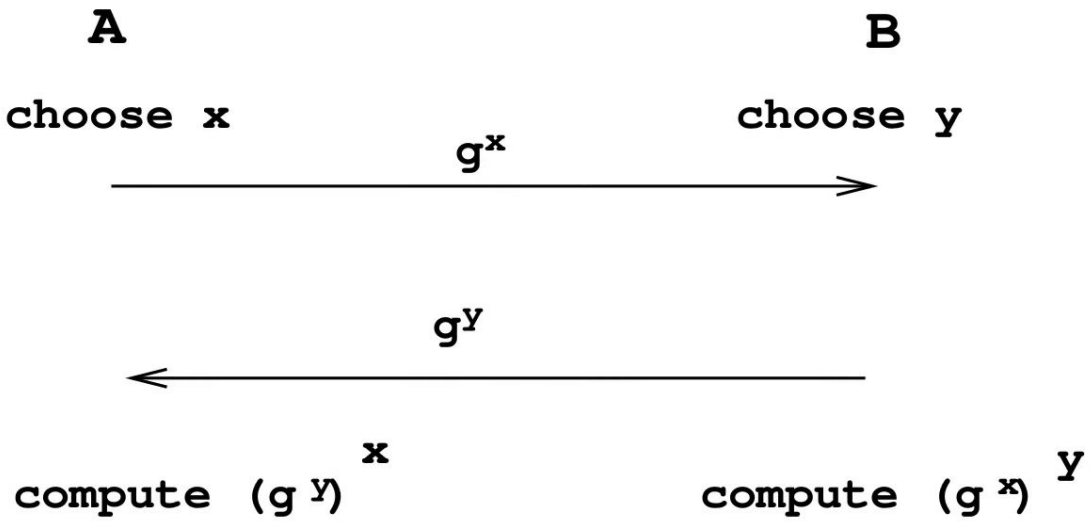
```
choose x
```

B

```
choose y
```


A
B
choose x
choose y
g^x


A
B
choose x
choose y
g^x
g^y


A
B
choose x
choose y
g^x
g^y
compute $(g^y)^x$
compute $(g^x)^y$

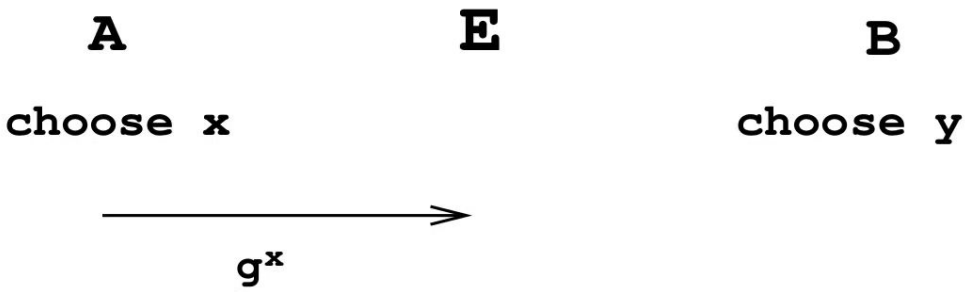
A

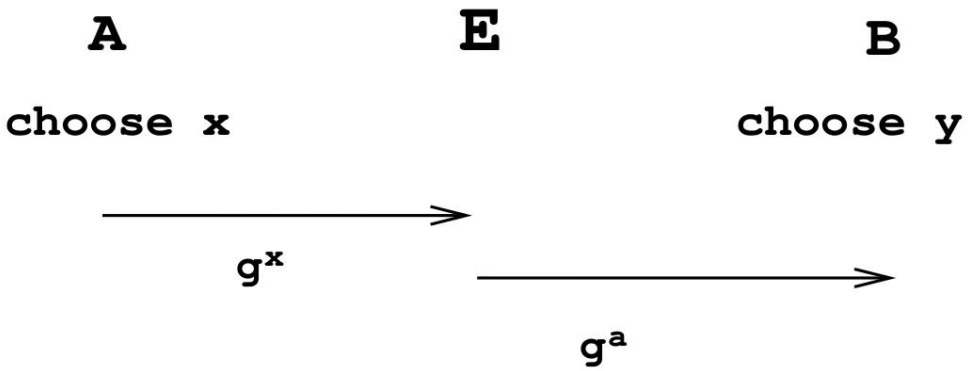
`choose x`

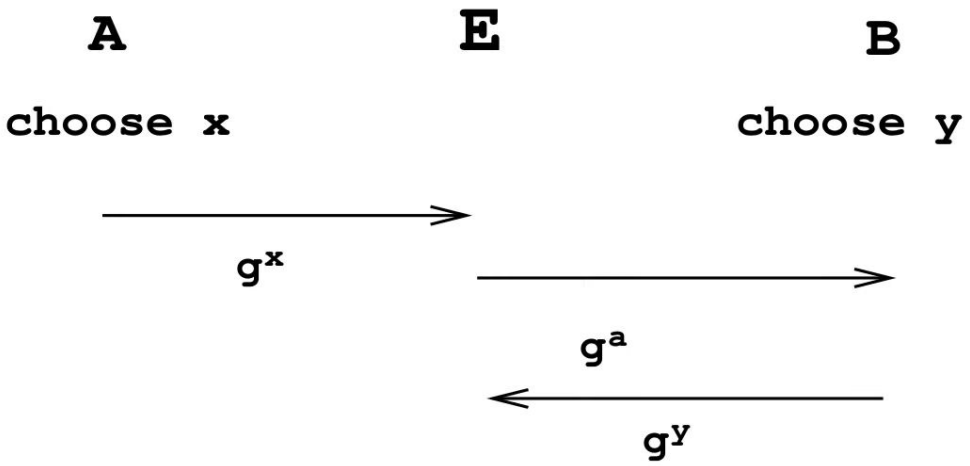
E

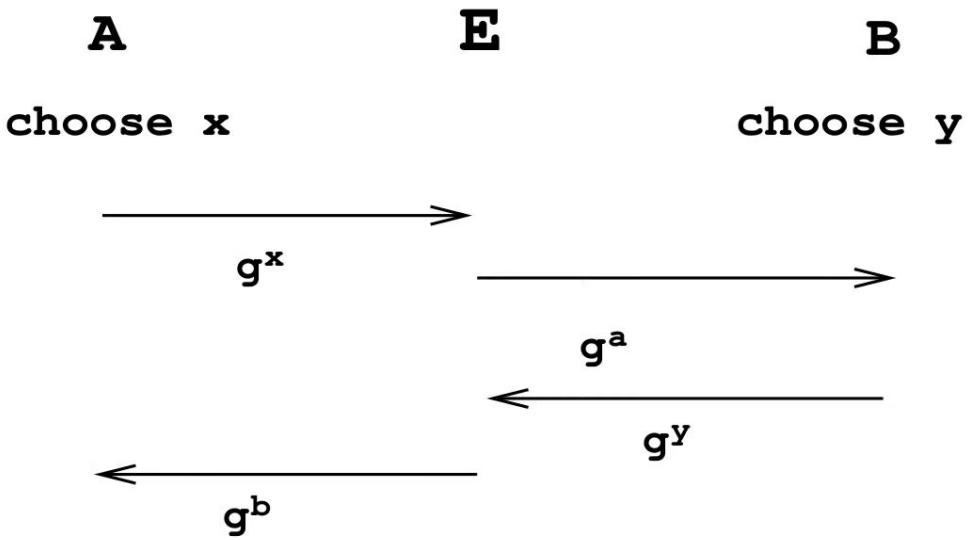
B

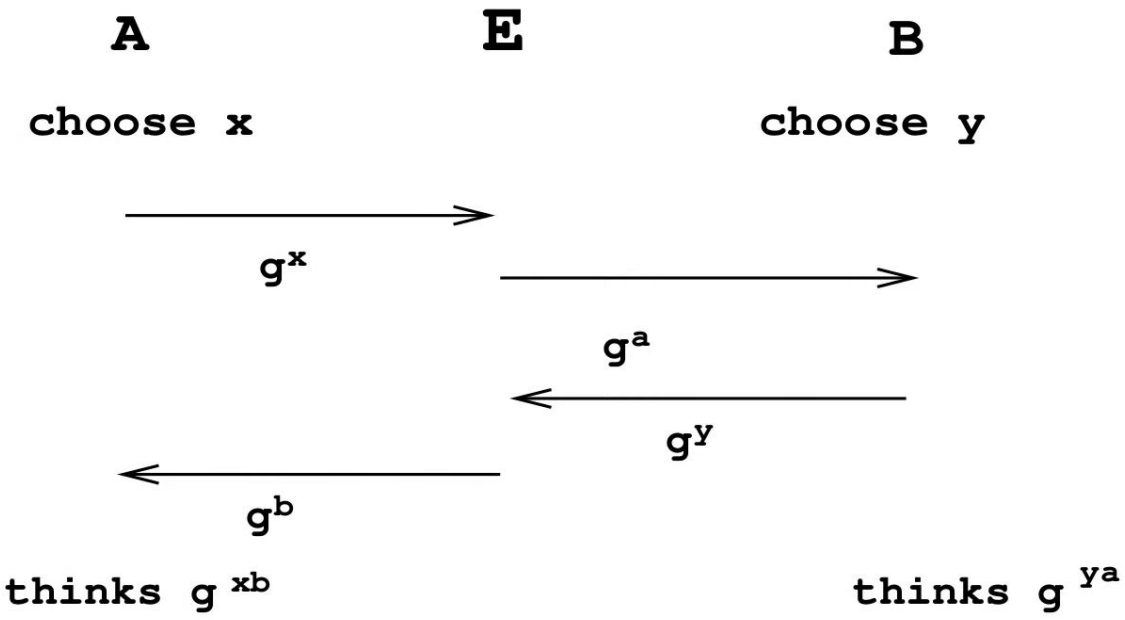
`choose y`

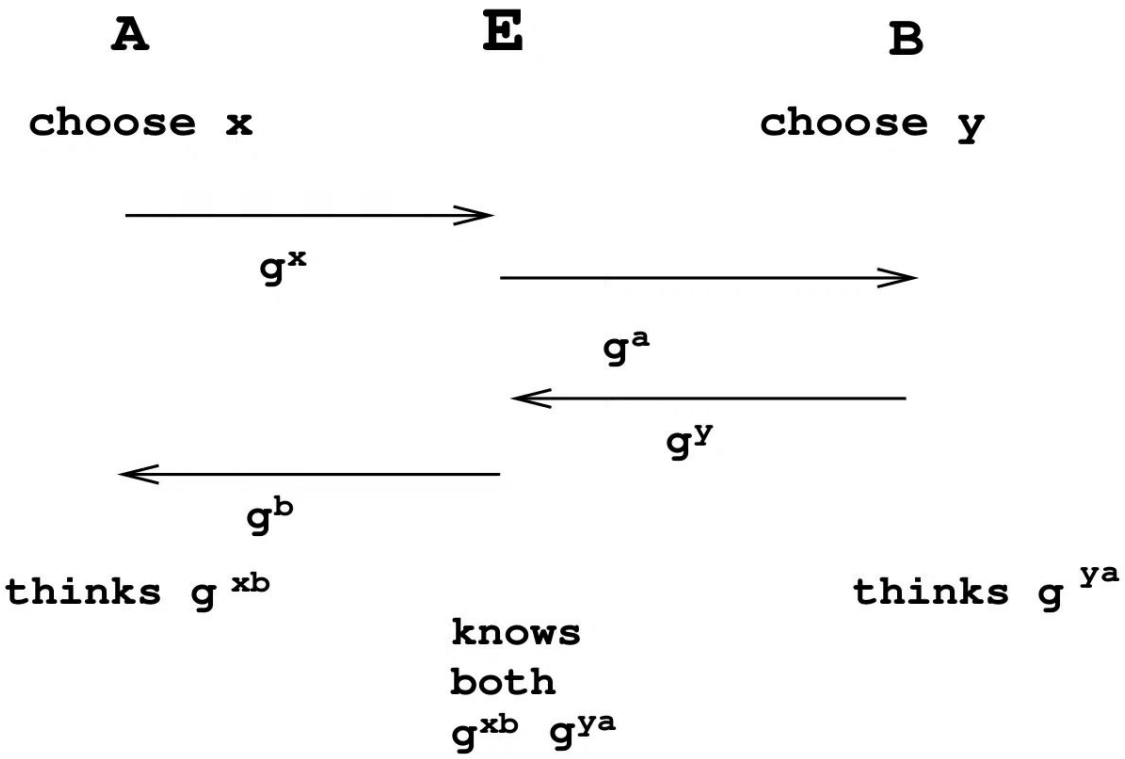

A
E
B
choose x
choose y
g^x


A
E
B
choose x
choose y
g^x
g^a

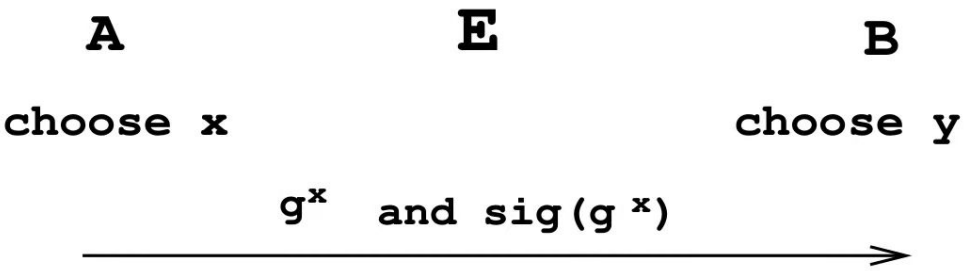

A
E
B
choose x
choose y
g^x
g^a
g^y

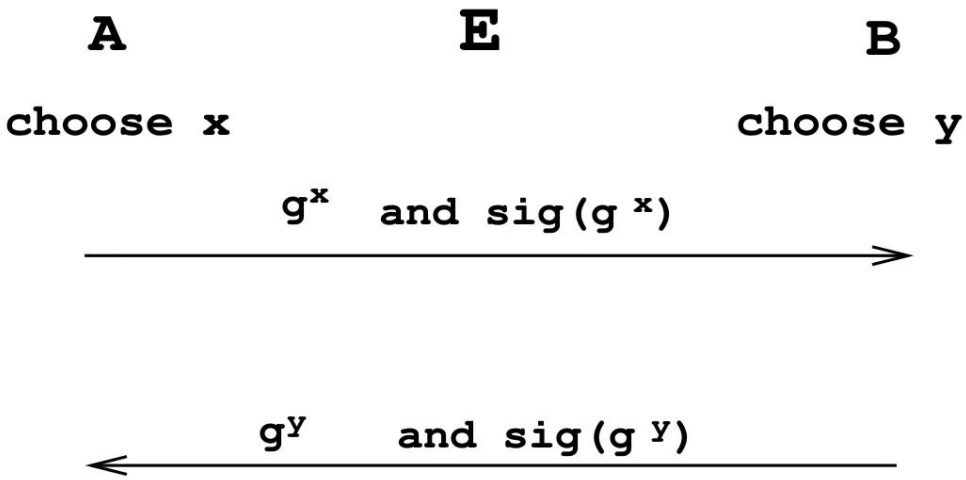

A
E
B
choose x
choose y
g^x
g^a
g^y
g^b

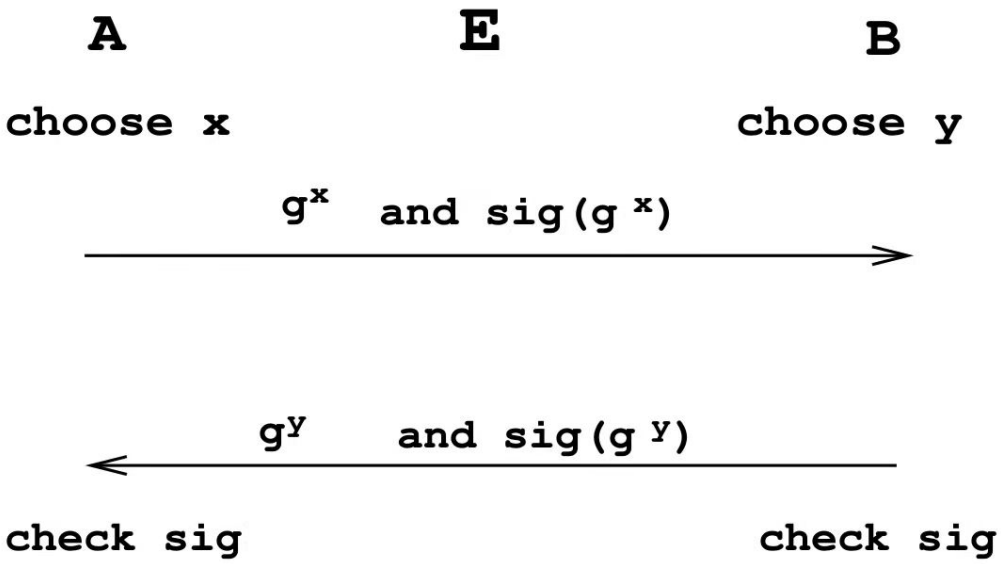

A
E
B
choose x
choose y
g^x
g^a
g^y
g^b
thinks g^{xb}
thinks g^{ya}


A
E
B
choose x
choose y
g^x
g^a
g^y
g^b
thinks g^{xb}
thinks g^{ya}
knows
both
g^{xb} g^{ya}

Another proposal


A
E
B
choose x
choose y
g^x and sig(g^x)


A
E
B
choose x
choose y
g^x and sig(g^x)
g^y and sig(g^y)

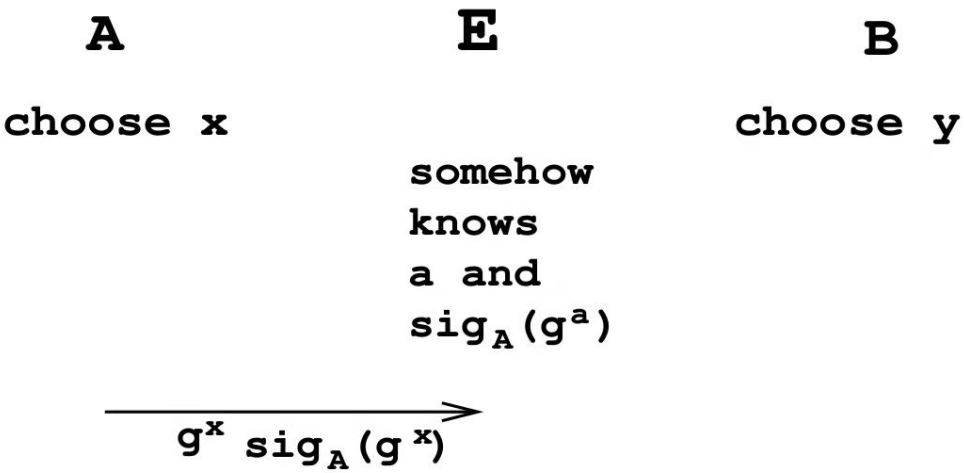

A
E
B
choose x
choose y
g^x and sig(g^x)
g^y and sig(g^y)
check sig
check sig

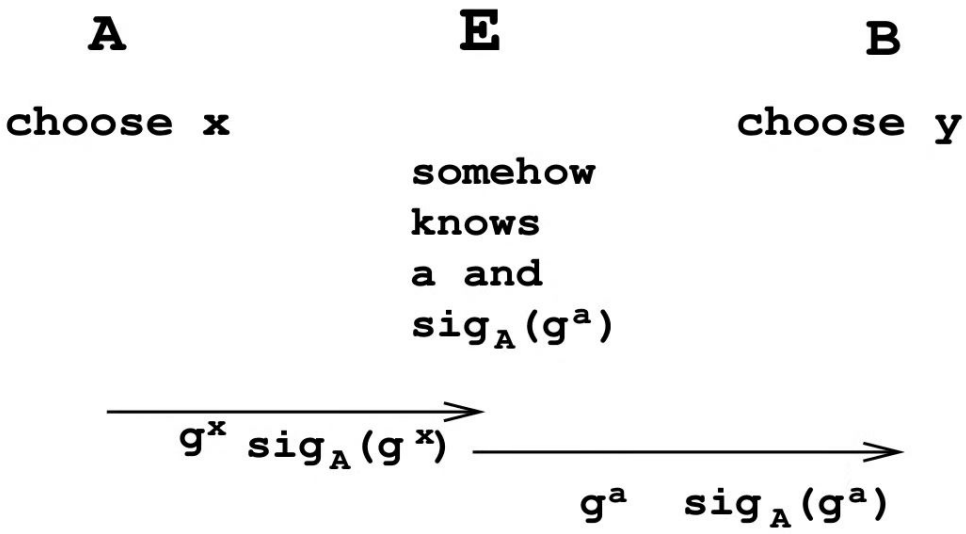
A E B

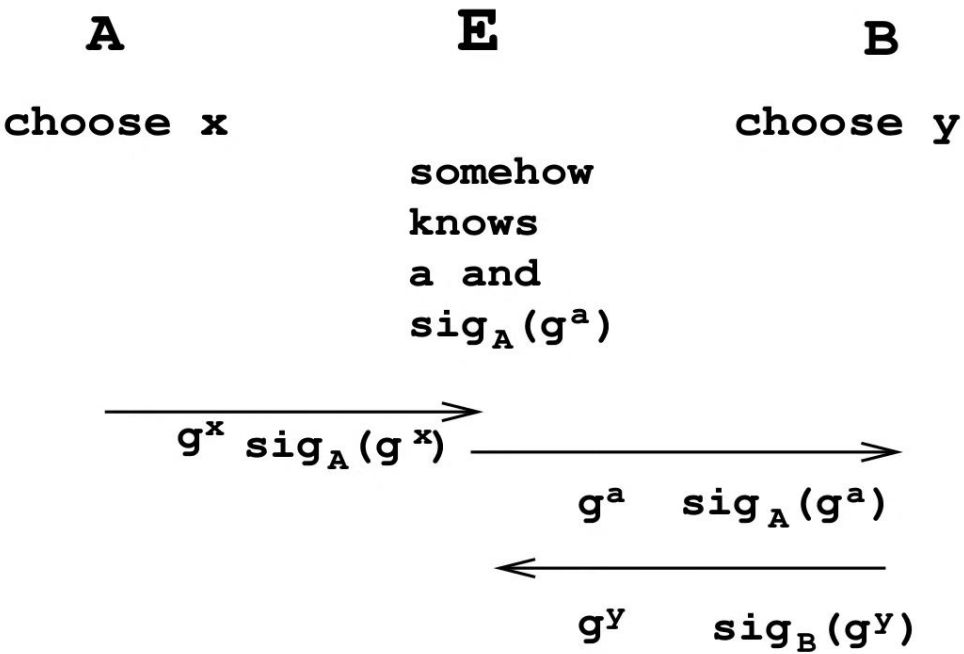
choose x choose y

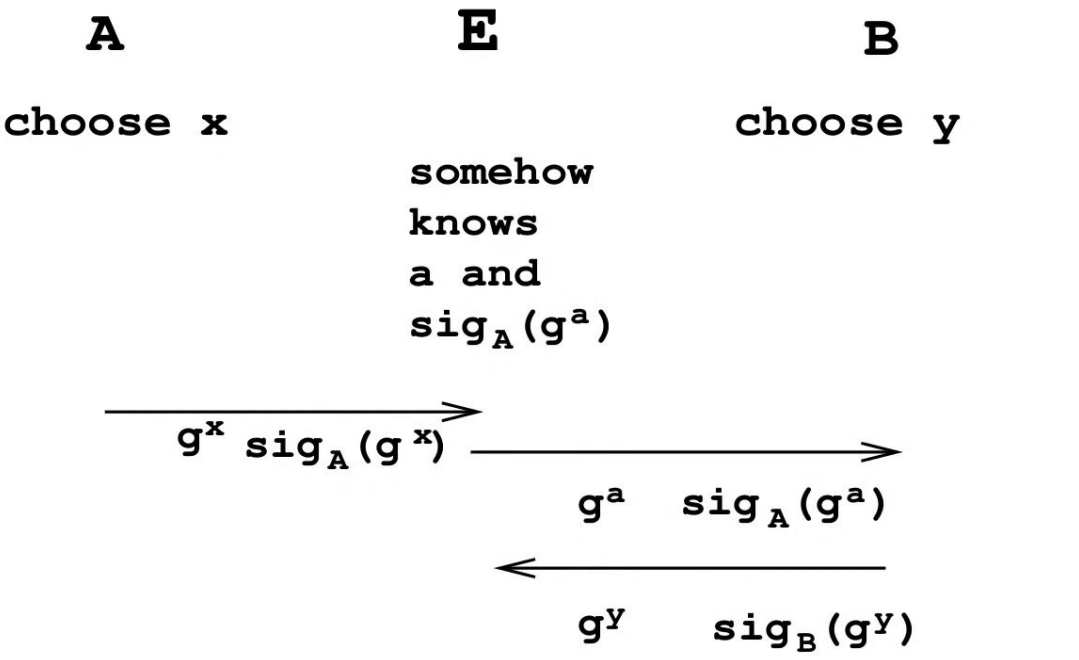
replay attack

A	E	B
choose x	somehow knows a and $sig_A(g^a)$	choose y


A
E
B
choose x
choose y
somehow
knows
a and
$sig_A(g^a)$
g^x $sig_A(g^x)$


A
E
B
choose x
choose y
somehow
knows
a and
$sig_A(g^a)$
g^x $sig_A(g^x)$
g^a $sig_A(g^a)$


A
E
B
choose x
choose y
somehow
knows
a and
$sig_A(g^a)$
g^x $sig_A(g^x)$
g^a $sig_A(g^a)$
g^y $sig_B(g^y)$


A
E
B
choose x
choose y
somehow
knows
a and
sig_A(g^a)
g^x sig_A(g^x)
g^a sig_A(g^a)
g^y sig_B(g^y)
compute g^ya
compute g^ya

We will cover the key negotiation in the next lecture