Review: Mitigating Memory Safety Vulnerabilities. Introduction to Cryptography

CMPSC 403 Fall 2021

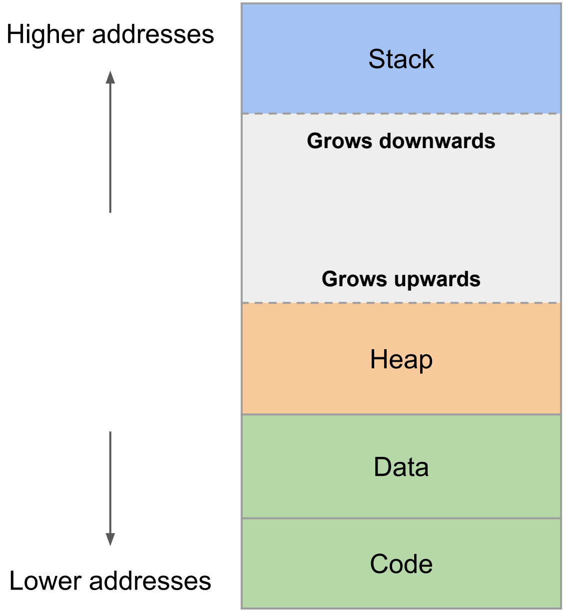
October 5, 2021

Recall: Putting Together an Attack

1. Find a memory safety (e.g. buffer overflow) vulnerability
2. Write malicious shellcode at a known memory address
   - Mitigation: Address-space layout randomization
3. Overwrite the RIP with the address of the shellcode
   - Mitigation: Stack canaries
   - Mitigation: Pointer authentication
4. Return from the function
5. Begin executing malicious shellcode
   - Mitigation: Non-executable pages

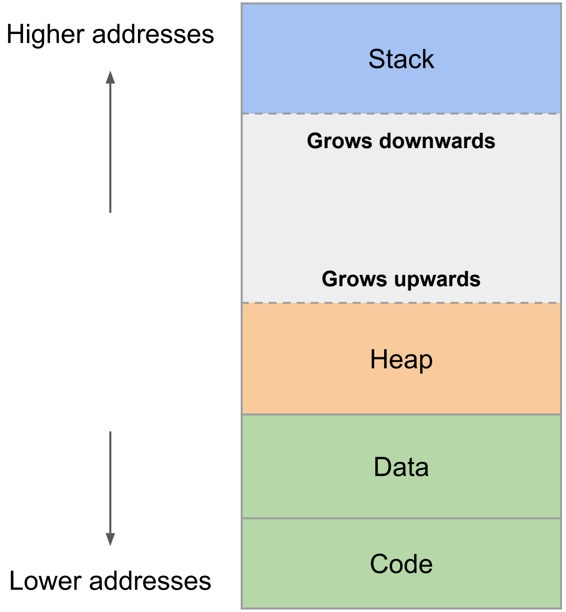
We can defend against memory safety vulnerabilities by making each of these steps more difficult (or impossible)!

Recall: x86 Memory Layout

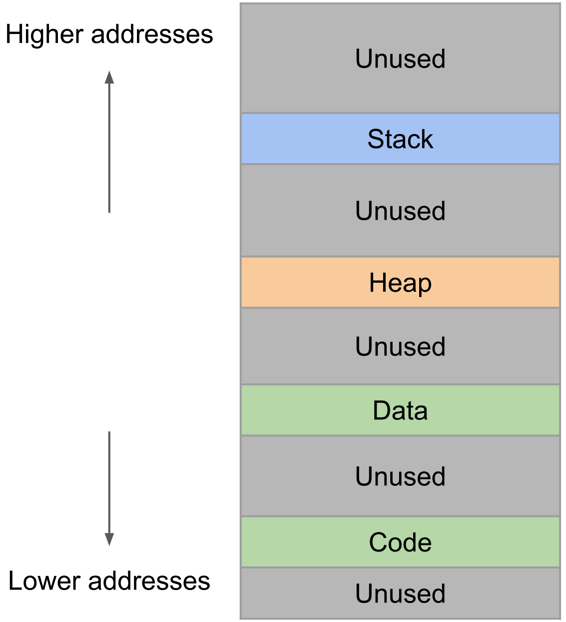


In theory, x86 memory layout looks like this...

Recall: x86 Memory Layout

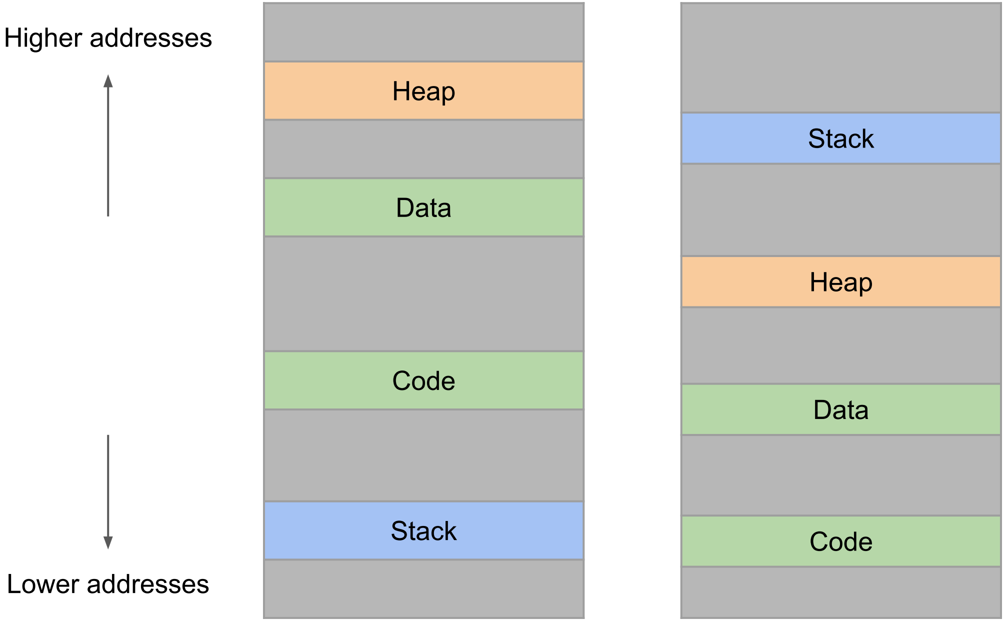


In theory, x86 memory layout looks like this...



...but in practice, it usually looks like this (mostly empty)!

Recall: x86 Memory Layout



Idea: Put each segment of memory in a different location each time the program is run

Address Space Layout Randomization

- **Address space layout randomization (ASLR)**: Put each segment of memory in a different location each time the program is run
  - The attacker can't know where their shellcode will be because its address changes every time you run the program
- ASLR can shuffle all four segments of memory
  - Randomize the stack: Can't place shellcode on the stack without knowing the address of the stack
  - Randomize the heap: Can't place shellcode on the heap without knowing the address of the heap
  - Randomize the code

Combining Mitigations

Combining Mitigations

- Recall: We can use multiple mitigations together
    - Synergistic protection: one mitigation helps strengthen another mitigation
    - Force the attacker to find multiple vulnerabilities to exploit the program
    - Defense in depth
- Example: Combining ASLR and non-executable pages
    - An attacker can't write their own shellcode, because of non-executable pages
    - An attacker can't use existing code in memory, because they don't know the addresses of those code (ASLR)
- To defeat ASLR *and* non-executable pages, the attacker needs to find two vulnerabilities
    - First, find a way to leak memory and reveal the address randomization (defeat ASLR)
    - Second, find a way to write to memory and defeat non-executable pages

Enabling Mitigations

- Many mitigations are effectively free today (insignificant performance impact)
- The programmer sometimes has to manually enable mitigations
  - Example: Enable ASLR and non-executable pages when running a program
  - Example: Setting a flag to compile a program with stack canaries (secret value placed on the stack which changes every time the program is started to protect RIP)
- If the default is disabling the mitigation, the default will be chosen
  - Recall: Consider human factors!
  - Recall: Use fail-safe defaults!

Enabling Mitigations: Internet of Things

Qualys Security Blog [Link]

CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)

Animesh Jain *January 26, 2021*

The Qualys Research Team has discovered a heap overflow vulnerability in sudo, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default sudo configuration by exploiting this vulnerability.

Takeaway: Many (most?) IoT devices don't enable basic mitigations

Summary: Memory Safety Mitigations

- Mitigation: **Non-executable pages**
  - Make portions of memory either executable or writable, but not both
  - Defeats attacker writing shellcode to memory and executing it
  - Subversions
    - **Return-to-libc**: Execute an existing function in the C library
    - **Return-oriented programming** (**ROP**): Create your own code by chaining together small gadgets in existing library code
- Mitigation: **Stack canaries**
  - Add a sacrificial value on the stack. If the canary has been changed, someone’s probably attacking our system
  - Defeats attacker overwriting the RIP with address of shellcode
  - Subversions
    - An attacker can write around the canary
    - The canary can be leaked by another vulnerability (e.g. format string vulnerability)
    - The canary can be brute-forced by the attacker

Summary: Memory Safety Mitigations

- Mitigation: **Pointer authentication**
  - When storing a pointer in memory, replace the unused bits with a pointer authentication code (PAC). Before using the pointer in memory, check if the PAC is still valid
  - Defeats attacker overwriting the RIP (or any pointer) with address of shellcode
- Mitigation: **Address space layout randomization** (**ASLR**)
  - Put each segment of memory in a different location each time the program is run
  - Defeats attacker knowing the address of shellcode
  - Subversions
    - Leak addresses with another vulnerability
    - Brute-force attack to guess the addresses
- Combining mitigations
  - Using multiple mitigations usually forces the attacker to find multiple vulnerabilities to exploit the program (defense-in-depth)

Next: Cryptography

What is cryptography?

- Older definition: the study of secure communication over insecure channels
- Newer definition: provide rigorous guarantees on the security of data and computation in the presence of an attacker
- Modern cryptography involves a lot of math
    - We'll review any necessary material as they come up

Don't try this at home!

A Brief History of Cryptography

Cryptography by Hand: Caesar Cipher

- One of the earliest cryptographic schemes was the **Caesar cipher**
  - Used by Julius Caesar over 2,000 years ago
- Choose a key *K* randomly between 0 and 25
- To encrypt a plaintext message *M*:
  - Replace each letter in *M* with the letter *K* positions later in the alphabet
  - If *K* = 3, plaintext DOG becomes GRJ
- To decrypt an encrypted ciphertext *C*:
  - Replace each letter in *C* with the letter *K* positions earlier in the alphabet
  - If *K* = 3, ciphertext GRJ becomes DOG

K = 3			
M	*C*	*M*	*C*
A	D	N	Q
B	E	O	R
C	F	P	S
D	G	Q	T
E	H	R	U
F	I	S	V
G	J	T	W
H	K	U	X
I	L	V	Y
J	M	W	Z
K	N	X	A
L	O	Y	B
M	P	Z	C

Cryptography by Hand: Substitution Cipher

- A better cipher: create a mapping of each character to another character.
  - Example: A = N, B = Q, C = L, D = Z, etc.
  - Unlike the Caesar cipher, the shift is no longer constant!
- Key generation algorithm: KeyGen()
  - Generate a random, one-to-one mapping of characters
- Encryption algorithm: Enc(*K*, *M*)
  - Map each letter in *M* to the output according to the mapping *K*
- Decryption algorithm: Dec(*K*, *C*):
  - Map each letter in *C* to the output according to the **reverse** of the mapping *K*

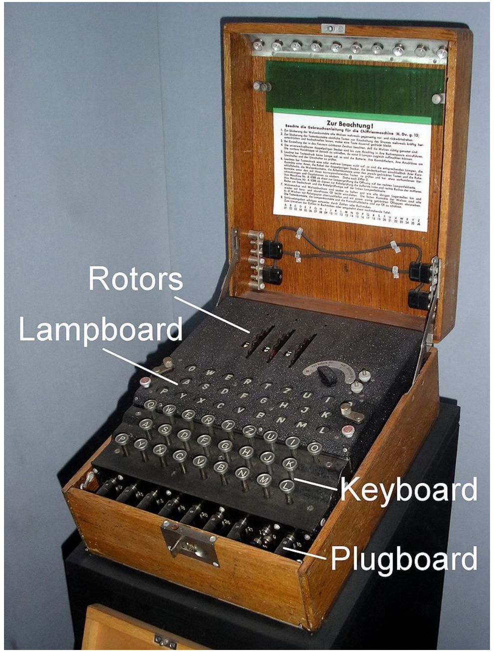
K			
M	*C*	*M*	*C*
A	N	N	G
B	Q	O	P
C	L	P	T
D	Z	Q	A
E	K	R	J
F	R	S	O
G	V	T	D
H	U	U	I
I	E	V	C
J	S	W	F
K	B	X	M
L	W	Y	X
M	Y	Z	H

Cryptography by Machines: Enigma

- A mechanical encryption machine used by the Germans in WWII

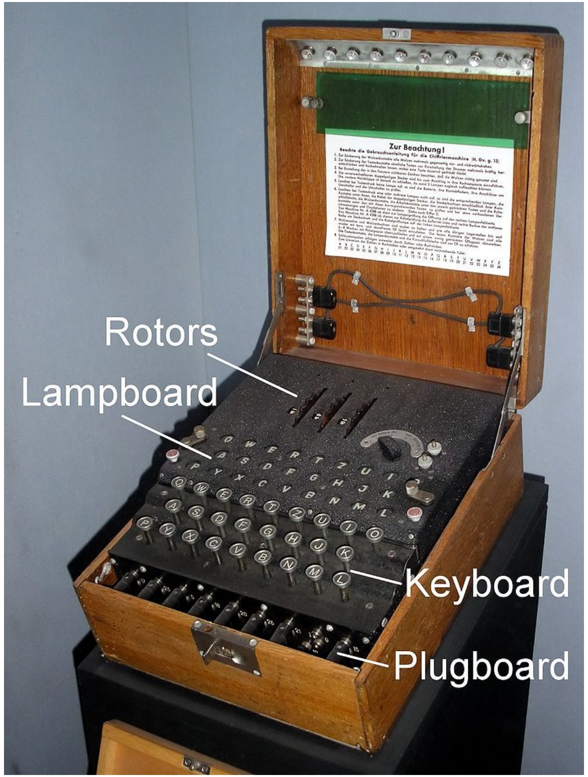
Rotors

Plugboard



Cryptography by Machines: Enigma

- KeyGen():
  - Choose rotors, rotor orders, rotor positions, and plugboard settings
  - 158,962,555,217,826,360,000 possible keys
- Enc(*K*, *M*) and Dec(*K*, *C*):
  - Input the rotor settings *K* into the Enigma machine
  - Press each letter in the input, and the lampboard will light up the corresponding output letter
  - Encryption and decryption are the same algorithm!
- Germans believed that Enigma was an "unbreakable code"



Cryptography by Computers

- The modern era of cryptography started after WWII, with the work of Claude Shannon
- “New Directions in Cryptography” (1976) showed how number theory can be used in cryptography
    - Its authors, Whitfield Diffie and Martin Hellman, won the Turing Award in 2015 for this paper

Takeaways

- Cryptography started with paper-and-pencil algorithms (Caesar cipher)
- Then cryptography moved to machines (Enigma)
- Finally, cryptography moved to computers (which we're about to study)

Definitions

Meet Alice, Bob, Janyl, and Jada

- Alice and Bob: The main characters trying to send messages to each other over an insecure communication channel
- Janyl: An **eavesdropper** who can read any data sent over the channel
- Jada: A **manipulator** who can read and modify any data sent over the channel

Meet Alice, Bob, Janyl, and Jada

- We often describe cryptographic problems using a common cast of characters
- One scenario:
  - Alice wants to send a message to Bob.
  - However, Janyl is going to *eavesdrop* on the communication channel.
  - How does Alice send the message to Bob without Janyl learning about the message?
- Another scenario:
  - Bob wants to send a message to Alice.
  - However, Jada is going to *tamper* with the communication channel.
  - How does Bob send the message to Alice without Jada changing the message?

Three Goals of Cryptography

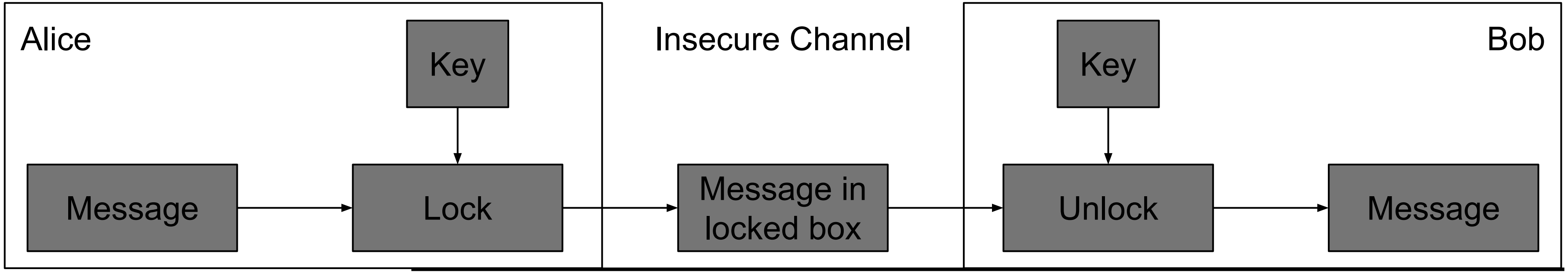
- In cryptography, there are three main properties desired in data
- **Confidentiality**: An adversary cannot **read** our messages.
- **Integrity**: An adversary cannot **change** our messages without being detected.
- **Authenticity**: I can prove that this message came from the person who claims to have written it.
    - Integrity and authenticity are closely related properties…
        - Before I can prove that a message came from a certain person, I have to prove that the message wasn't changed!
    - … but they're not identical properties

Keys

- The most basic building block of any cryptographic scheme: The **key**
- We can use the key in our algorithms to secure messages
- Two models of keys:
    - **Symmetric key model**: Alice and Bob both know the value of the same secret key.
    - **Asymmetric key model**: Everybody has two keys, a secret key and a public key.
        - Example: RSA encryption

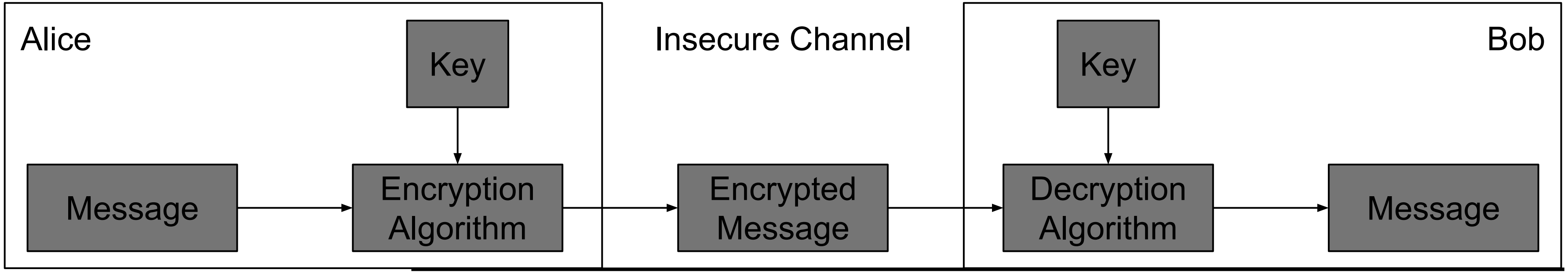
Confidentiality

- **Confidentiality**: An adversary cannot **read** our messages.
- Analogy: Locking and unlocking the message
  - Alice uses the key to lock the message in a box
  - Alice sends the message (locked in the box) over the insecure channel
  - Janyl sees the locked box, but cannot access the message without the key
  - Bob receives the message (locked in the box) and uses the key to unlock the message



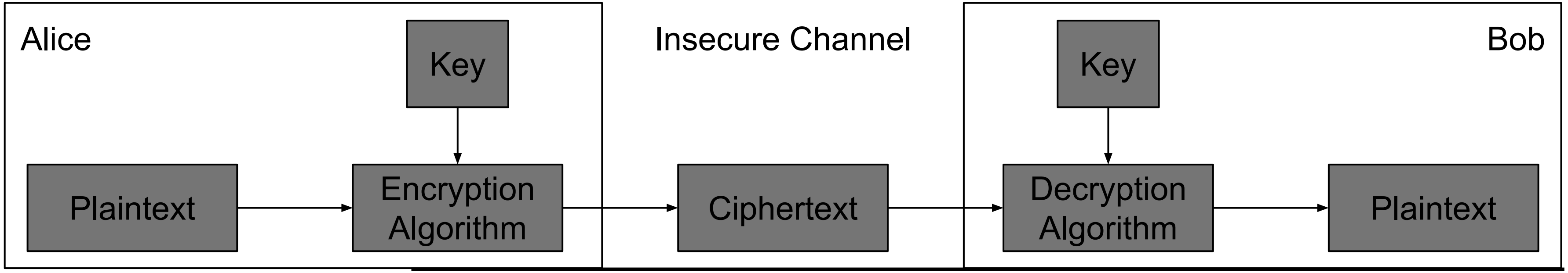
Confidentiality

- **Confidentiality**: An adversary cannot **read** our messages.
- Schemes provide confidentiality by **encrypting** messages
    - Alice uses the key to **encrypt** the message: change the message into a scrambled form
    - Alice sends the encrypted message over the insecure channel
    - Janyl sees the encrypted message, but cannot figure out the original message without the key
    - Bob receives the encrypted message and uses the key to **decrypt** the message back into its original form



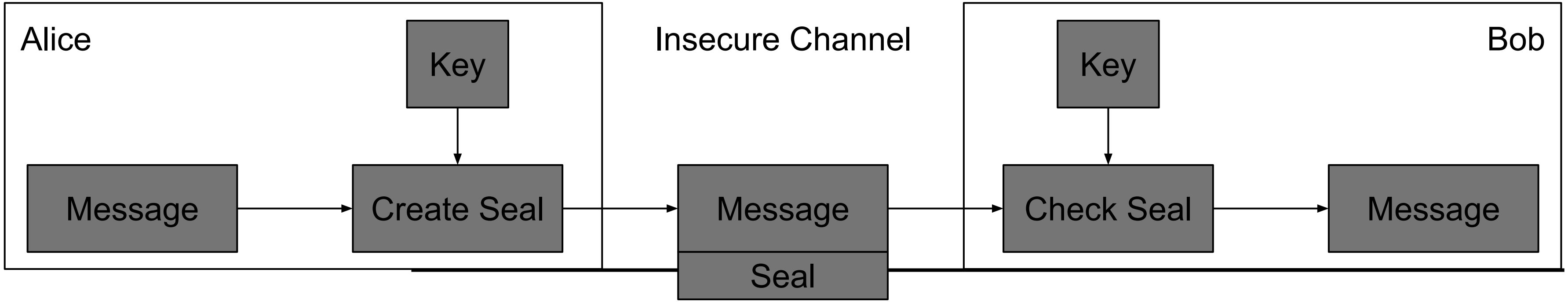
Confidentiality

- **Plaintext**: The original message
- **Ciphertext**: The encrypted message



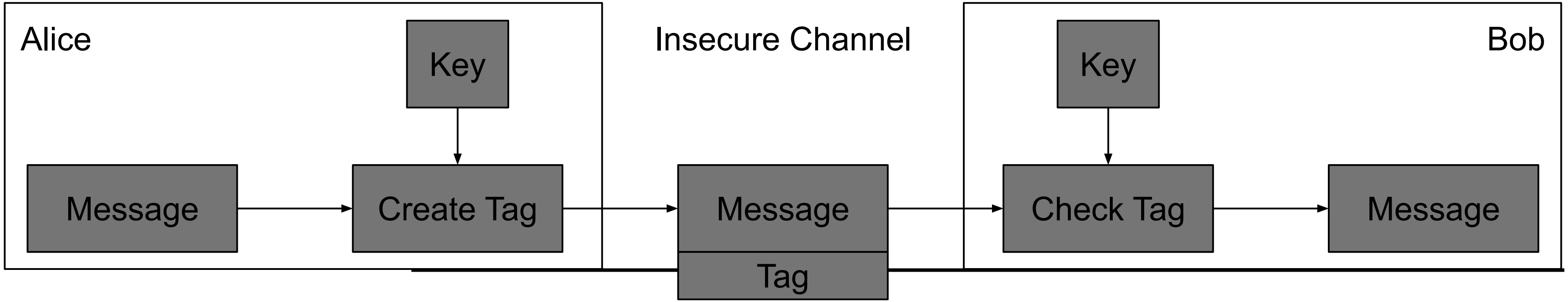
Integrity (and Authenticity)

- **Integrity**: An adversary cannot **change** our messages without being detected.
- Analogy: Adding a seal on the message
  - Alice uses the key to add a special seal on the message (e.g. puts tape on the envelope)
  - Alice sends the message and the seal over the insecure channel
  - If Jada tampers with the message, she'll break the seal (e.g. break the tape on the envelope)
  - Without the key, Jada cannot create her own seal
  - Bob receives the message and the seal and checks that the seal has not been broken



Integrity (and Authenticity)

- **Integrity**: An adversary cannot **change** our messages without being detected.
- Schemes provide integrity by adding a **tag** or **signature** on messages
  - Alice uses the key to generate a special tag for the message
  - Alice sends the message and the tag over the insecure channel
  - If Jada tampers with the message, the tag will no longer be valid
  - Bob receives the message and the tag and checks that the tag is still valid
- More on integrity in a future lecture



Threat Models

- What if Janyl can do more than eavesdrop?
- Real-world schemes are often vulnerable to more sophisticated attackers, so cryptographers have created more sophisticated threat models too
- Some threat models for analyzing confidentiality:

	Can Janyl trick Alice into encrypting messages of Janyl's choosing?	Can Janyl trick Bob into decrypting messages of Janyl's choosing?
Ciphertext-only		
Chosen-plaintext	✓	
Chosen-ciphertext		✓
Chosen plaintext-ciphertext	✓	✓

Cryptographic Hash Function: Definition

- Hash function: $H(M)$
  - Input: *Arbitrary* length message M
  - Output: *Fixed* length, n-bit hash
  - Sometimes written as $\{0, 1\}^* \to \{0, 1\}^n$
- Properties
  - **Correctness**: Deterministic
    - Hashing the same input always produces the same output
  - **Efficiency**: Efficient to compute
  - **Security**: One-way-ness ("preimage resistance")
  - **Security**: Collision-resistance
  - **Security:** random/unpredictability, no predictable patterns for how changing the input affects the output
    - Changing 1 bit in the input causes the output to be completely different
    - Also called "random oracle" assumption

Hash Function: Intuition

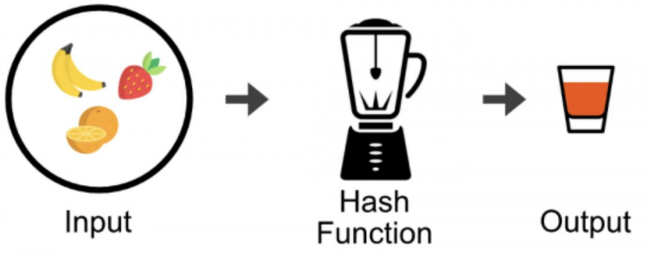
- A hash function provides a fixed-length “fingerprint” over a sequence of bits
- Example: Document comparison
  - If Alice and Bob both have a 1 GB document, they can both compute a hash over the document and (securely) communicate the hashes to each other
  - If the hashes are the same, the files must be the same, since they have the same “fingerprint”
  - If the hashes are different, the files must be different

Hash Function: One-way-ness

- **Informal:** Given an output y, it is infeasible to find **any** input x such that $H(x) = y$
- **Intuition**: Here's an output. Can you find an input that hashes to this output?
  - Note: The adversary just needs to find *any* input, not necessarily the input that was actually used to generate the hash

Is this function one-way?

- The constant function H(x) = 1
  - No, because an attacker can output any x, and that leads to 1. It does not have the original x some challenger thought about
- Take fruit and make a smoothie: yes



Ref: https://computersciencewiki.org/index.php/One-way_function

- Block cipher, for k random and secret: E
  - Yes, if an attacker can invert E_k, then an attacker can distinguish it from a random permutation which breaks the security of the block cipher