AY24/25 Sem 2 CS3235 MidTerms CheatSheet

Chapter 1: Secure Programming

A) Memory Vulnerability: Spatial Memory Errors

Buffer Overflows

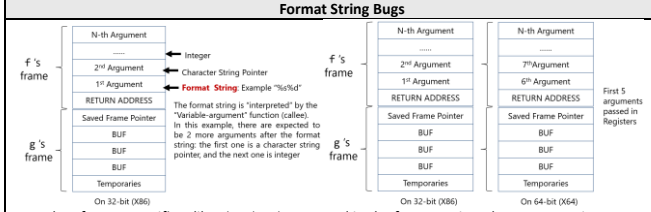
- Execution of code is done sequentially in the memory, with PC guiding the process.
  - **Direct Branch**: Replace by a constant value specified in the instruction
  - **Indirect Branch**: Replace by a value fetched from the memory. (jmp)
- Exploiting Buffer Overflow
  - The attacker can write to certain memory that the attacker originally doesn't have access to, if overflow is not checked. While possible, it is still not so easy for an attacker to compromise memory.

Possible Attacks

1. Overwrite existing code w malicious code
2. Overwrite a piece of control-flow mechanism
   a) Attack 1: Replace a **memory location** storing a code address that is used by a **direct jump**
   b) Attack 2: Replace a **memory location** storing a code addr that is used by an **indirect jump**

- Stack Smash has 2 effects:
  - ➢ If return address is modified, the control flow is compromised.
  - ➢ If local variables are modified, the computed result will be wrong
  - ➢ Attacker modifies the exec path, & modify mem values via buffer overflow

Format String Bugs



- When format specifiers like %x, %s, %n are used in the format string, the program tries to:
  1. **Read or write from the stack**, treating it as arguments for the format string.
  2. Access unintended memory locations or modify the program's memory.
     - "%x" reads data from the stack
     - "%s" reads char strings from the process' memory
     - "%n" writes an integer to locations in the process' memory.
     - If an application accepts the format and parses the format as shown in the table above, high chance it is vulnerable to format string attacks.
- The **format string vulnerability** occurs because of the misuse of format specifiers, not due to the differing byte sizes of data types. It allows attackers to:
  1. Leak sensitive information by reading from unintended memory locations.
  2. Modify memory through specifiers like %n.
- To prevent format string vulnerabilities:
  - Avoid directly passing user inputs as a format string. I.e use formatted characters

Common Parameters used in a Format String Attack:

Formatters	Output	Passed as:
%%	% character (literal)	Reference
%p	External representation of a pointer to void	Reference
%d	Decimal	Value
%c	Character	Value
%u	Unsigned decimal	Value
%x	Hexadecimal	Value
%s	String	Reference
%n	Writes the number of characters into a pointer	Reference

Integer Overflow

Integer overflow is often caused by hardware's limitation to represent digits (16-bit, 32-bit, etc).

- Limited representation of values.
- In 2s complement → different number of +'ve and –'ve numbers. Moreover, number values can be either signed or unsigned.
  - ➢ Signed Range (for *n* bits): -2^{n-1} to $2^{n-1} - 1$ | e.g 8bits → [-128, 127]
  - ➢ Unsigned Range (for n bits): 2^n | e.g 8bits → [0, 255]

This integer overflow is a problem if there are logic checks in place for value comparison.

- E.g in a 2s, 4-bit system, -3 + -6 should ideally = -9. But integer overflow →+7 instead.

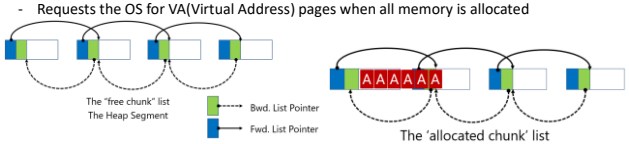
Type Promotions
bool → char → short int → int → unsigned int → long → unsigned → long long → float → double → long double

Best practice: Perform EXPLICIT conversion, don't rely on implicit conversion, as they are implementation dependent… All might have different behaviors. Explicit types will prevent such Integer overflows.

- Say there is a flow control to check for –'ve inputs. This would cause a wrong behavior in the system.

Heap Overflows

- Heap is used for dynamic memory allocation. The size of the heap is not fixed, and heaps may be fragmented.
- The glibc (std. C library) manages memory allocations in C
  - Stores each **allocated** memory chunk in a linked list
  - Stores each **unallocated** memory chunk in a linked list
  - Can re-allocate a previously freed chunks
  - Requests the OS for VA(Virtual Address) pages when all memory is allocated



- If the given memory block has size 40, but malloc(50) was used instead → heap overflow
- The program will be writing beyond the given limited space. → Can cause program hijack, seg fault, memory manipulation attacks.

These forms of attack are known as Data Oriented Attacks.

- Data-oriented attacks manipulate non-control data to alter a program's benign behavior without violating its control-flow integrity.
- There are no req for Data Oriented Attacks (Unlike Control Oriented Attacks)
  - No need to write any control data for exploit.
  - We can just manipulate the non-control data.

There is also no need to corrupt anything for Data Oriented Attacks. E.g Heartbleed in OpenSSL

Bad Casting / Type Confusion

- Recall: Every object(variable/method/class) in base class is accessible in derived/sub object.
- It is ALWAYS **SAFE** to **UPCAST** (Upcast → More generic, Downcast → More specific)
- DOWNCAST is DANGEROUS, esp static_cast, Imagine Big >> Small, not in scope. Data lossy.
  - May cause memory corruptions, as the object may not be a sub-object of the former → undefined behavior
  - Data might have multiple inheritances, which causes out-of-scope memory access.

TLDR:

- A pointer to class T' can safely point to a superclass T
- All functions in class T are implemented in subclass T'
- Upcasting is safe, down casting is not
- Alw choose dynamic_cast for polymorphic types. Static → Mem error, not enough space to cast.

Static Cast	Dynamic Cast
• Compile-time conversions • Fast: no extra verification in run-time • No information on actually allocated types in runtime.	• Run-time conversions • Requires Runtime Type Information (RTTI) • Slow: Extra verification by parsing RTT • Typically prohibited in performance critical applications

B) Memory Exploits

Code Injection

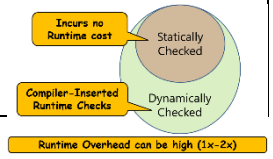
- Definition: A memory exploit that hijacks control to jump to attacker's data payload.
- Example, if you want to inject malicious code, Z, we need to change the RETURN ADDRESS of f to point to the start of your malicious code Z, say address: 0xbfffffff0. This can be done via a buffer overflow in g.
- From Fig 3, If you don't know "X", just set the shellcode to use NOP (No operation)
  - Tell the CPU to do nothing & fetch the next instruction.
  - Including a large block of NOP instructions in the injected code as landing area.
  - The execution will reach the shell code as long as the return address points to somewhere in the NOP Sled.
    - ➢ You can jump anywhere in the NOP sled.
    - ➢ **Note**: This only works IF the address is >= return payload address. If "0xbfffffX" hits below Z, then it will result in a segmentation fault.
    - ➢ **Usually,** we can just use GCB to check the return address… So this is not an issue.

Order of sequence during a code injection attack

1. A buffer overflow happened & writes the shell code (payload), NOP sled, and an attacker-controlled return address for the current function.
2. The code executes the return instruction of the current function and jumps to an attacker-controlled address
3. The code execution is transferred to the NOP sled. After which, the code execution is transferred to the shell code.

Requirements for Code Injections:

- **Req 1**: Write Attack Payload in memory
- **Req 2**: Have Attack Payload Be Executable
- **Req 3**: Divert control-flow to payload



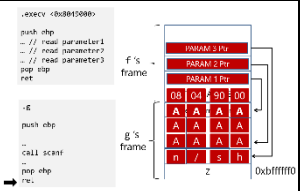
Code Reuse

***Definition**: A memory exploit that hijacks control to jump to attacker's controlled code address*

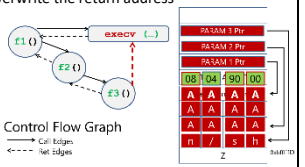
- Attacker hijacks the control flow.
- They then jump back to the code segment. (e.g Return-to-libc)

Say libC resides at 0x8049000.

- We first change the return address to the code segment's address. In this case, 0x8049000.
  - ➢ We can achieve this via buffer overflow vulnerability, and formulating a payload to overwrite the return address
- Afterwards, we setup arguments to execv()
  → executes a program, on the stack.



In the Control Flow Graph, the black arrows shows the ideal path, the red arrow shows the malicious path.



Requirements for Code Reuse:

- **Req 1**: Have Attack Payload Be Executable
- **Req 2**: Divert control-flow to payload
- **Note**: Re-use the existing code as the payload! Don't need to write Attack Payload in Mem.

More advanced: Return-Oriented Programming, e.g CTF Ret-2-Win

Other common libc functions include: system() → execute shell, and exit() → terminate prgm.

- This argument/param stack will be passed into the libc functions.
- When the function returns, the program will execute the attacker's chosen libc function, with the parameters in the payload as the arguments to the library calls.

C) Memory Vulnerability: Temporal Memory Errors

Types of Storage Durations (Static, Automatic, Dynamic)

Static	
Creation	The compiler only creates a static when it executes the **variable declaration.** • E.g static int num = 10; • If the declaration statement is executed > 1 time, the compiler will ignore the remaining calls & does not change the variable's value.
Storage	A static variable is stored in the **data segment** of the object file of a program.
Lifetime	A static variable will remain valid for the entire program's life, even though the parent block of code has finished executing.
Automatic	

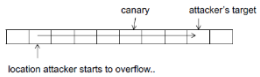
Creation	• **Automatic variable = Local variable**
	• Automatic variable is created when the program enters its parent block of code
	  - E.g int num = 10;
	  - If the statement is declared > 1 time, the compiler throws an error

Storage	A local(automatic) variable is stored in the function **call stack** of a program
Lifetime	An automatic variable is valid till the parent block of code stops running. The variable will be removed from memory when the parent block stops executing. (Pop off stack) **Note**: If however, the return address is overwritten, the normal "function return" cleanup process is skipped → local variables in the stack are NOT removed. - This can be exploited for attacks such as Return-to-libC.
Dynamic	
Creation	• A **dynamic variable** is created when we allocate the variable's memory. • Memory is allocated dynamically via pointer and functions such as calloc & malloc. - E.g int* ptr = (int*) malloc(sizeof(int)); • If we want to create another dynamic variable, we will need to use another ptr. • If the same pointer is used, the compiler will throw an error.
Storage	A dynamic variable is stored on the **heap** of the program.
Lifetime	The dynamic variable is only removed from the heap when it is deallocated with free. If we DON'T deallocate memory when the program ends → Memory Leak 😢

Use-After-Free

- A **use-after-free** vulnerability occurs when a program accesses memory after it has been freed. This can lead to **undefined behavior**, including crashes, data corruption, or even security vulnerabilities such as arbitrary code execution.
- Use-After-Free can be caused by programming bug such as:

Double Free	• **Cause**: Freeing the same memory location twice. • **Example**: If the program frees a ptr and then mistakenly frees it again later. • **Exploit**: An attacker could manipulate heap metadata to inject malicious code into the memory freed twice.
Dangling Pointers	• **Cause**: Keeping a reference to a memory location after it has been freed. • **Example**: Memory is freed, but a pointer to it still exists | That pointer is used to read/write to the now-invalid memory.



Heap Metadata Corruption	• When a program accesses freed memory, it can overwrite metadata maintained by the memory allocator. • This allows attackers to modify the behavior of future memory allocations or overwrite specific areas of memory.

Double Free

- A **double free** occurs when a program attempts to free the same mem location more than once.
  - **Explicit double free & Logic Errors**

Explicit Double Free	Logic Errors

`char *ptr = (char *)malloc(10);`
`free(ptr);`
`free(ptr);  // Double free`
	`void freeMemory(char *ptr) {`
	`  free(ptr);`
	`}`
	`int main() {`
	`  char *ptr = (char *)malloc(10);`
	`  freeMemory(ptr);  // First free`
	`  free(ptr);  // Double free`
	`  return 0;`
	`}`

  - **Shared pointer mismanagement** (Race conditions, multithreaded programs, improper semaphores, concurrency)
    - ➢ Recall, semaphore operations such as sem_wait or sem_post are atomic → Cannot be interrupted by context switches!
    - ➢ However, poorly written semaphores or concurrency management system may have flaws, leading to unintentional double-frees

```
char *ptr1 = (char *)malloc(10);
char *ptr2 = ptr1;  // Both pointers refer to the same block
free(ptr1);
free(ptr2);  // Double free
```

Note: It is possible that the OS allocates the same chunk of memory in p to q. In that case, the data in q will be the data in p. Leaked memory ☹ .

Write-Anything-Anywhere Exploit

BAD FUNCTION for deleting a node from a linked list → Enables write anything anythere.

```
void bad_delete_from_list (struct chunk * p)
{
    if (!p) return NULL;
    if (p->next) p->next->prev = p->prev;
    if (p->prev) p->prev->next = p->next;
    else free_list_head = p->next;
}
```

Why bad?
No pointer validity checks
- p->next can be **ANYWHERE**
- p->prev can be **ANYTHING**

Dangling pointers not cleared

It is **possible to hijack the delete function and write anything anywhere**.
- Because the code does not check if the pointer to the next node is correct/ wrong.

** Don't assume that all code is correct. Some code are vulnerable ☹

D) Full Memory Safety

i) Spatial Safety

FAT Pointer / Referent Objects

Fat pointers are extended pointers that store additional metadata about the memory region they point to. Typically, they include:
- **Base Address**: The starting address of the allocated memory region.
- **Bounds**: The size (or upper bound) of the memory region.
- **Pointer**: The current address the pointer is accessing.

How Fat Pointers Work:
1. When accessing memory via a fat pointer, bounds checks are performed to ensure the access is within the valid range.
2. Any attempt to access memory outside the bounds triggers a runtime error or exception.

Pros	Cons
• Detects out-of-bounds accesses (e.g., buffer overflows or underflows) at runtime. • Provides strong spatial safety guarantees.	• Increased memory usage (due to extra metadata). • Higher runtime overhead (due to bounds checks).

FAT Pointers/Referent Objects Provides:
- **Complete** Spatial Safety
- Code compiled for normal pointers will crash when passed fat pointers
- Object Layouts don't change
- Each pointer **dereference** operation needs a check
  - No check on pointer arithmetic *** FAT pointers do not check for pointer arithmetic.
- No check on type casts ****FAT pointers do not check for typecast safety.

NULLIFY Pointers

- In short, this means that we set the pointer = NULL after free().
  - Track creation and destruction of pointers
  - Ensure: De-allocated pointers are not accessed
- Nullifying pointers involves setting a pointer to NULL immediately after the memory it points to is freed. This prevents accidental or malicious dereferencing of dangling pointers.

How It Works:

1. When free() is called on a pointer, the pointer is set to NULL.
2. Any attempt to dereference a NULL pointer triggers a runtime error.

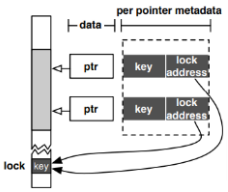
Pros	Cons
• Prevents dereferencing of dangling ptr. • Easy to implement in most programming languages.	Only addresses one aspect of temporal safety. Use-after-free bugs can still occur if multiple ptrs reference the same memory.

Note: If the pointer is reused but freed, it may still cause double-free issues.
ONLY PROVIDES **PARTIAL TEMPORAL SAFETY**!!!!

Lock & Key Mechanism

- CETS ensures temporal safety by associating each memory allocation with a unique "key" and checking whether a pointer's "lock" matches the memory's "key" during access.
  - Track creation and destruction of pointers
  - Ensure: De-allocated pointers are not accessed
- CETS: Lock-and-key [CETS] (Compiler-Enforced Temporal Safety)
  - Lock & Key values match *IFF* memory object is temporally safe to access via this pointer
  - Assume that Metadata cannot be corrupted spatially.

How It Works:
1. Each memory block is assigned a **unique key** when it is allocated.
2. Pointers to the block are assigned a **lock** that matches the block's key.
3. When accessing the memory, the lock and key are compared:
   - If the lock matches the key, access is allowed.
   - If the lock does not match the key (e.g., the mem was freed & reallocated), access is denied.
4. Upon freeing the memory, the key is invalidated.



- Associate a (key, lock) pair with each pointer
- Key: Unique value for each object allocated
- Pointer created from another inherit (key, lock) value.
- On **deference**, check pointer's key matches its lock
  - key =*(lock_address)
  - If key match, the dereference succeeds.
  - If key don't match, the operation is aborted.
- On **deallocation** (i.e free), change lock value in mem. obj. → Key is flagged as invalid.

- The key value stored at the lock address in CETS (Compiler-Enforced Temporal Safety) is typically stored in the heap.
- The key check happens dynamically at runtime whenever a pointer is dereferenced, ensuring that use-after-free and dangling pointer issues are caught before they cause undefined behavior.
  - Unlike static analysis (which happens at compile-time), CETS instruments the compiled code to perform runtime checks.
  - Every time a pointer is used, the runtime system checks whether the key stored at the lock address matches the pointer's key.

E) Memory Mitigations

Defensive Coding Practices for C/C++ Programmers

- **Use bug-finding tools**
  - Tools checking for vulnerabilities using static source code analysis:
    - ➢ Address Sanitizer; ITS4 (It is the Software, Stupid –Security Scanner); RATS (Rough Auditing Tool for Security); Flawfinder
- **Safe coding techniques**
  - Pay attention to loops
  - Explicitly specify size of destination buffer
- **Use safe libraries**

Unsafe C Functions	Safer C Function Versions
strcpy(char *dest, const char *src) strcat(char *dest, const char *src) gets(char *s) sprintf(conts char *format, …)	strncpy, strncat, fgets, snprintf

- **Using Security Extensions –Libsafe**
  - These extensions make unsafe functions safer. (E.g strcpy)
    - ➢ It intercepts calls to unsafe functions, e.g strcpy(dest, src)
    - ➢ Validate sufficient space in curr stack frame: $|frame - pointer - dest| > strlen(src)$
    - ➢ If so, do strcpy. Else terminate the application

Stack Canaries

- Canaries are secrets inserted at carefully selected memory locations during runtime.
  - They are used to protect nearby data from corruption
- Checks are carried out during runtime to make sure that the values are not being modified.
  - The random canary value is checked at RET (return).
  - Canary value is usually before the RET value.
- IF these values are touched/ modified → Program halts.

- Canaries can help to detect overflow, especially stack overflow.
  - In a typical buffer overflow, consecutive memory locations have to be over-ran.
  - If the attacker wants to write to a particular memory location via buffer overflow, the canaries would be modified.
- NOTE: It is important to keep the value as "secret". If the attacker happens to know the value, it may able to write the secret value to the canary while over-running it.

- In C, by default, canary is on. To off it: `gcc programname.c -fno-stack-protector`

Guard Pages

- **Guard pages** are a memory protection mechanism used to detect and prevent unintended memory access, such as **buffer overflows** or **stack overflows**, and to improve memory safety.
- They involve reserving a region of memory that is marked as **inaccessible** or **non-readable/non-writable**.
  - Certain pages are marked with NR, NW, NX (No read, No write, No execute)
  - Assumption:
    - ➢ The attacker can only write linearly.
    - ➢ All written values are not used in dereferences.
  - If a program attempts to access the memory in a guard page, a **segmentation fault (or access violation)** occurs, which halts the program or triggers a handler.

How Guard Pages Work
1. **Stack Protection**:
   - A guard page is typically placed at the end of a thread's stack to prevent **stack overflows** from corrupting adjacent memory regions. If the stack grows beyond its allocated size, it hits the guard page, triggering an exception.
2. **Heap Protection**:
   - Guard pages are sometimes placed around memory allocations in the heap to detect **out-of-bounds accesses** or buffer overflows.
3. **Dynamic Adjustments**:
   - When memory regions are expanded (e.g., stack growth), the guard page is moved to the new boundary to maintain protection.

Non-Executable Data (DEP) --- Check with `$ execstack -s c_binary_file_name`
- Defense: DEP (a.k.a W ⊕X) (DEP → Data Execution Prevention)
  - Setting regions of memory non-executable + Use NX bit
- Defense Goal →Prevents Foreign code Injection
- Blocks Requirement 2 of the Attack →"Need to have payload executable"

Address Randomization

- Assumption: Attacker can write arbitrary places
- Defense Goal: Attacker can't predict location accessed in attack
- Mechanism:
  - At load time, randomize stack, code, bss, etc & Randomize heap location at runtime

Address Space Layout Randomization (ASLR)
- **ASLR** is used to protect programs from memory-based attacks such as **buffer overflows**, **return-oriented programming (ROP)**, and other exploits that rely on predictable memory addresses.
- ASLR works by **randomizing the locations of key regions in a program's address space every time it is executed**.
  - The base of each stack segment is randomised.
  - With this random address location, attackers cannot predict the location accessed.

How ASLR Works

When ASLR is enabled, the operating system randomizes the starting addresses of various memory regions in a program's address space, including:
1. **Stack**: Randomizing the stack base address makes it harder for attackers to predict where variables and return addresses are located.
2. **Heap**: The heap base address is randomized to prevent predictable heap exploits.
3. **Shared Libraries**: Libraries like libc are loaded at randomized addresses, making it difficult for attackers to reuse code.
4. **Executable Code**: The starting address of the executable's code section is randomized.
5. **Memory-Mapped Files and Regions**: Randomizes the location of dynamically loaded libraries and other mapped files.

Run	ALSR ON	ALSR OFF
1	Stack address: 0x7ffcabc12345 Heap address: 0x55555575d000 Code segment: 0x555555554000 Library address: 0x7f81e0123000	
2	Stack address: 0x7ffd2bf45678 Heap address: 0x55555698e000 Code segment: 0x555556664000 Library address: 0x7f81f1237000	Stack address: 0x7ffcabc12345 Heap address: 0x55555575d000 Code segment: 0x555555554000 Library address: 0x7f81e0123000

** Notice how the Addresses changes in "ALSR ON" on the 2nd run.

- ASLR is not foolproof. It can be bypassed with information leaks or brute force or via offsets.
- It is dependent on Compiler and OS, and might not be supported on older devices.

Chapter 2: Depth in Defense

A) Inline Reference Monitors (IRM) and Process Sandboxing

Reference Monitors / Inline Reference Monitors (IRM)

- Reference Monitor: A piece of code that *checks all* **references** to an **object**
  - It is a system that controls the access to a computer resource/object.
  - It determines whether an action can be performed on an object or not.
  - It is a validation mechanism that enforces access control policy over a subject's ability to perform operations on objects. (More below)
    - ➢ E.g Subject A can r/w/x object A, but not object B. (Same for Subject B for obj A,B)

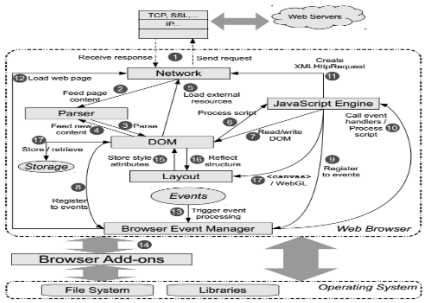
Syscall Sandboxing

- Syscall Sandbox: A reference monitor for protecting OS resource objects from an app
  - A sandbox in general, is a safe & isolated environment for executing untrusted programs.
  - Main goal is to prevent a bad process from compromising the entire system.
  - E.g chroot command, Linux namespaces, resource access control such as seccomp, and VMs.
  - Sandboxing is NOT entirely foolproof.
- Idea: To have syscall policies to defeat attacks (i.e restricts syscalls)
- Only safe syscalls are allowed
  - No exec system calls allowed → Prevents execution of arbitrary binary/shell code.
  - No exec-after-read system calls allowed → Prevents Just-In-Time (JIT) code execution.
  - Filtering filesystem & network syscalls → Blocks unauth access to sensitive files/data/servers.
  - This prevents malicious injected code from running & compromising the system.

Process Sandboxing

- A **sandbox** is a controlled environment that limits an application's access to the file system, network, memory, and other system components.
- Applications running in a sandbox operate with **least privilege**, reducing their ability to cause harm if exploited.
- Sandboxing can be implemented using various technologies:
  - **Virtual Machines (VMs):** Isolate entire operating systems.
  - **Containers (e.g., Docker, Kubernetes):** Provide lightweight, process-level isolation.
  - **Operating System Sandboxes:** Features like Windows Sandbox, macOS App Sandbox, or Linux namespaces.
  - **Browser Sandboxes:** Web browsers like Chrome and Edge sandbox web content to prevent malicious scripts from escaping.

B) Access Control & Capabilities



Issues with Security Policies

- Security policies are often hard to define correctly, as it is possible to have buggy policies, or there are ways to confuse the system, such that privilege escalation still occurs.
- Attackers can exploit these weaknesses to confuse the system, leading to privilege escalation or unauthorized access.

Model	Read?	Write?	For?
Bell-LaPadula	Read Down	Write Up	Confidentiality
Biba	Read Up	Write Down	Integrity

Access Control List (ACL) and Capabilities

Access Control List (ACL), Access Control Matrix (ACM)

An ACL stores the access rights to an object as a list.

my.c	→ (root, {r,w}) → (Bob, {r,w,o})
mysh.sh	→ (root, {r,x}) → (Alice, {r,x,o})
Sudo	→ (root, {r,s,o}) → (Alice, {r,s}) → (Bob, {r,s})
a.txt	→ (root, {r,w}) → (root, {r,w,o})

- Straightforward to manage the accesses to resources by associating permissions with objects.
- However, they can become complex to manage, and inefficient. (Per Object)

Object (ACM)

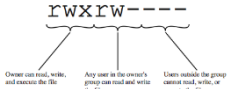
Principals	my.c	mysh.sh	sudo	a.txt
root	{r,w}	{r,x}	{r,s,o}	{r,w}
Alice	{}	{r,x,o}	{r,s}	{r,w,o}
Bob	{r,w,o}	{}	{r,s}	{}

Capability

- A subject is given a list of capabilities, where each capability is the access rights to an object.
- "a capability is an unforgeable token that gives the possessor certain rights to an object"

root	→ (my.c, {r,w}) → (mysh.sh, {r,x}) → (sudo, {r,s,o}) → (a.txt, {r,w})
Alice	→ (mysh.sh, {r,x,o}) → (sudo, {r,s}) → (a.txt, {r,w,o})
Bob	→ (my.c, {r,w,o}) → (sudo, {r,s})

- Provide flexibility in access control by associating permissions with subjects.
- However, they can be complex to track (Per Subject)
- A **Capability** is a (pointer, metadata) tuple, which
  - Created or modified **only** by querying the security monitor
  - **Sufficient & necessary** to have to access the object
  - Has access rights embedded in it, enforced by monitor



- Capability must be **Unforgeable**: Can't manufacture without explicitly getting it.

Security Policies

Separation of Concerns	Separate the policy from its enforcement.
Least Privilege	Give each component only the necessary privileges.
Minimize Trusted Code Base (TCB)	• Reduce what one needs to trust. • Separate the verifier from the enforcement.

Access Control Policies:

Discretionary access control (DAC)	• DAC is based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do. • No fixed policy, Each owner decides the access rules • Example: UNIX File Systems
Mandatory access control (MAC)	• MAC is based on comparing security labels (which indicate how sensitive or critical system resources are) with security clearances (which indicate system entities are eligible to access certain resources). • Policy fixed by the administrator. • Each owner cannot change access rights of objects created or owned by it. Example: Web protocols. E.g Bell-Lapula, Biba Model
Role-based access control (RBAC)	• RBAC is based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles. • E.g Discretionary Install-time permission

Bell-LaPula Model (For Data Confidentiality)

1. The Simple Security Property states that a subject at a given security level **may not read an object at a higher security level**.
2. The * (Star) Security Property states that a subject at a given security level **may not write to any object at a lower security level**.
3. The Discretionary Security Property uses an **access matrix** to specify the **discretionary access control (DAC).**

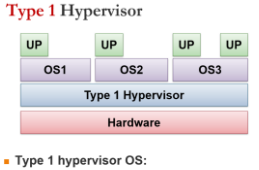
- This is known as "Write-Up, Read-Down" (WURD).
- Note: A subject can append to objects at higher security level. It is also possible that, by appending to an object, one could distort its original content. (Renegotiation Attacks)

Biba Model (For Process Integrity)

- The Biba model is designed so that subjects may not corrupt data in a level ranked higher than the subject, or be corrupted by data from a lower level than the subject.
- In general, preservation of data integrity has three goals:
  1. Prevent data modification by unauthorized parties
  2. Prevent unauthorized data modification by authorized parties
  3. Maintain internal and external consistency (i.e. data reflects the real world)
- This security model is directed toward data integrity (rather than confidentiality) and is characterized by the phrase: "Read-Up, Write Down". (RUWD).
- Restrictions imposed by Biba Model:No Write Up + No Read Down

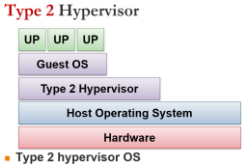
C) Virtual Machines

Full Virtualisation: Type 1 Hypervisor



- Type 1 hypervisor OS:
  - Provides individual ***virtual machines*** to guest OSes
- Runs directly on top of the hardware
- Generally more secure + higher performance (Type 1 tend to be used in cloud environments)
- Hypervisor is able to maintain separate hardware states each of the OSes.

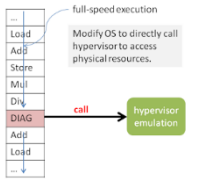
Full Virtualisation: Type 2 Hypervisor



- Type 2 hypervisor OS
  - Runs in host OS
  - Guest OS runs inside Virtual Machine
- Runs on the Host OS.
- Less secure. If host compromised, VM compromised.
- Type 2 Hypervisor interacts with the hardware not directly, but via the OS.

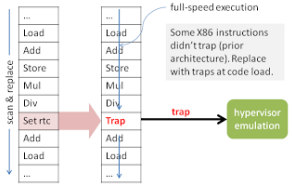
Paravirtualisation

- The guest OS is modified to work efficiently with the hypervisor
- Instead of emulating a full hardware, it cooperates with the hypercisor,reducint the overhead. Note: OS MUST BE MODIFIED to work with the hypervisor



Binary Translation (Software Virtualisation)

- The hypervisor dynamically translates privilege instructions from the guest OS to run on the host. → Allows unmodified guest OS to run on a virtualised system
- Note: Supports legacy systems, but a lot of overheads



Containerisation

- Shares the host OS kernel while isolating applications in a separate environment
  - Each container runs an app with its own deps, without a full OS overhead.
- Less isolated than full virtualisation. Compromised container → affect host OS.

Hardware Assisted Virtualisation (HAV)

- HAV allows the hypervisor to directly manage VMs using CPU extensions.
- CPUs adding support for better virtualisation.
- Better performance & lower latency than Binary translation.
- Requires external hardware support, not available on older processers.

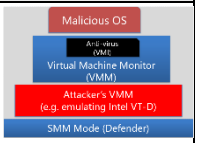
- **Limitations of Virtualisation**

Virtual Machine Based Rootkit (VMBR)

- VMBR → Malware which is installed below the OS, by running VMM underneath the original OS
- Dangers of VMBR:
  - **Full Control Over OS**: Since the OS runs as a **guest** inside the malicious hypervisor, all system calls and operations can be intercepted and manipulated.
  - **Undetectable by Traditional Security Tools**: Most antivirus (AV) & security software operate **inside the OS** and cannot detect activities at the hypervisor level.
  - **Persistence**: Since the rootkit **runs beneath the OS**, reinstalling the OS does not remove it.
  - **Invisible Hooks**: Unlike traditional rootkits that modify kernel structures, a VMBR **does not need to modify the kernel** of the guest OS → Hard to detect.

Root of Trust Problem

- If the highest privilege layer (i.e., the hypervisor or VMM) is compromised, all security mechanisms above it become ineffective.
- In this case, since the OS is malicious, an antivirus cannot detect the issue, or protect the system.
- If the VMM is compromised, it can modify all guest OS activities, bypassing security measures.
  - Containment is possible only if the highest privilege layer (VMM) is trusted.



VMM Detection: Red Pill Attack → To detect whether a system is running inside a VM.

Hardware Fingerprinting	• Some virtualized environments use outdated hardware emulation (e.g., VMware emulates an old i440BX chipset). • Malware can check for these unique hardware signatures to detect a VM.

Timing Analysis	• (Virtualisation Latency Measurement) Virtualized systems introduce extra delays in CPU execution, memory access, and I/O operations. • Malware measures execution times and compares them against known physical hardware speeds to determine if it's running inside a VM.
Side-Channel Attacks	• Cache-based detection: Malware executes cache timing operations that behave differently in a VM. • Instruction Behavior Analysis: Some CPU instructions behave differently when executed in a VM (e.g., CPUID and RDTSC instructions).

Problems of Covert Channels:

Timing-Based Covert Channels

Exploits shared hardware resources (e.g., CPU, cache, memory, disk, network) by modulating access timing to leak data.

- CPU Cache Timing Attacks
  - **Flush+Reload Attack:** The attacker VM loads a shared memory location, and the victim VM's usage of the same cache line affects access time.
  - **Prime+Probe Attack:** The attacker VM fills the CPU cache and waits for the victim to use it, measuring timing differences.
- Branch Target Buffer (BTB) Attacks
  Malware in an attacker VM pollutes the branch predictor used by the victim VM, measuring misprediction rates to extract information.
- Memory Bus Contention
  An attacker VM intentionally consumes memory bandwidth, causing delays that are detectable in the victim VM, allowing data exfiltration.
- Disk I/O Timing
  The attacker VM manipulates disk reads/writes, monitoring response times to infer victim activity (e.g., keystrokes, cryptographic operations).

Network-Based Covert Channels

Even if VMs are isolated from external networks, they can still communicate using **internal virtual networking** (e.g., virtual switches, shared buffers).

- **Traffic Shaping & Packet Timing:** The attacker VM **modulates network traffic timing**, encoding bits as delays (e.g., **fast = 1, slow = 0**).
- **ICMP/Ping Channel:** An attacker VM sends crafted **ping pkts** with encoded data to leak info.
- **ARP Spoofing within VMs:** The attacker VM manipulates **ARP cache** to redirect network traffic.

Side-Channel Attacks in Cloud Environments

Cloud providers like AWS, Azure, and Google Cloud rely on virtualization for multi-tenancy. Covert channels are especially dangerous in cloud environments because:

- ✔ **VMs belonging to diff users may share the same physical hardware** (CPU, cache, memory).
- ✔ **Attackers can rent VMs on same cloud host** to execute side-channel att against victim VMs.
- **Cross-VM Cache Attacks in the Cloud**
  - Attackers measure cache access patterns to steal **cryptographic keys** from victim VMs.
  - **Example**: Researchers have demonstrated AES key extraction using L1 cache timing attacks.
- **Virtualized CPU Scheduling Leaks**
  - By **measuring execution scheduling delays**, an attacker can infer **CPU usage patterns** of victim VMs, revealing sensitive workload activity.

Chapter 3: Network Attacks

OSI Model		
Layer	**Function**	**Example Protocols**
7. Application	User interface & network services	HTTP, FTP, DNS, SMTP
6. Presentation	Data translation & encryption	SSL/TLS, JPEG, ASCII
5. Session	Manages connections & sessions	NetBIOS, RPC
4. Transport	End-to-end communication, error recovery	TCP, UDP
3. Network	Routing & logical addressing	IP, ICMP, BGP
2. Data Link	MAC addressing, error detection	Ethernet, Wi-Fi, PPP
1. Physical	Electrical/physical transmission	Cables, Fiber, Radio Signals
TCP/IP Model		
Layer	**Function**	**Example Protocols**
Application	Direct interaction with user applications	HTTP, FTP, DNS, SMTP
Transport	Ensures data delivery & error handling	TCP, UDP
Internet	Logical addressing & routing	IP, ICMP, BGP
Link	Physical transmission & MAC addressing	Ethernet, Wi-Fi

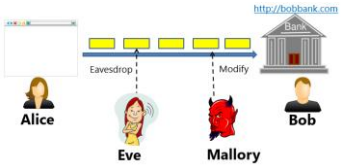
- Internet Routing

- Internet routing determines how data packets travel between networks using routers and protocols. There are 2 main types of routing:

Intra Domain Routing	• IGP → Interior Gateway Protocols • Used **within** an organisation or ISP
Inter Domain Routing	• EGP → Exterior Gateway Protocols • Used between ISPs and large networks (e.g BGP, more below)

- Routers **exchange reachability information** to forward packets efficiently.
- **Autonomous Systems** (AS) are independent networks managed by organizations or ISPs.
- **BGP** (Border Gateway Protocol) is the primary protocol used for **inter-domain routing**.
- The IP gateway serve as an entry/exit point for AS.

- Threat Model

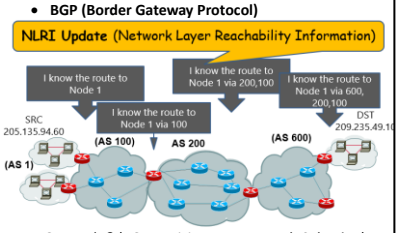


- Usual stuff
- Eve → Can Eavesdrop
- Mallory → Can modify/MITM
- An attacker aims to violate the CIAA:
- Confidentiality → Stealing or Intercepting data (e.g sniffing)
- Integrity → Tampering with data (E.g route hijacking)
- Authentication → Impersonation / identity theft
- Availability →Disrupting services (e.g DDoS attacks)

- **Network Attacker**

Off-Path Attacker	• Attacker cannot intercept packets but can send forged ones • Used for DDoS and IP spoofing.
On-Path Attacker	• Man-in-the-Middle • Attacker can intercept, modify and forward traffics. • Used for packet sniffing, IP hijacking and active surveillance.
Passive Attacker	• Listen/Reads traffic, but does not modify the data. • Used for eavesdropping, intelligence gathering.
Active Attacker	• Actively modifies, injects, or drop packets from traffic. • Used for hijacking connections, phishing, and DDoS attacks.
Individual vs Nation State Attackers	• Individuals (hacker, cybercriminals) focus on financial or personal gains. • Nation: (E.g NSA) conduct cyber espionage, sabotage and warfare.

- **BGP (Border Gateway Protocol)**



- BGP is the protocol that connects different networks (Autonomous Systems) on the Internet. (Large networks like ISPs, cloud providers, and enterprises) to exchange routing information.
- BGP ensures ISPs and networks **learn the best routes** to destinations.
  - It is a path vector protocol, meaning that it chooses routes based on a series of AS hops, rather than link-state or distance. SHORTEST PATH (Shortest Number of HOPS)

How BGP Works? (BGP Decision process, Path Selection)

1. **Highest Local Preference** (manually set by an admin).
2. **Shortest AS-Path** (fewer hops are preferred).
3. **Lowest Origin Type** (IGP < EGP < Incomplete).
4. **Lowest MED (Multi-Exit Discriminator)** (helps in multi-homed AS setups).
5. **Prefer eBGP over iBGP** (external routes preferred).
6. **Smallest Router ID** (used as a last resort).

BGP Operations:

- Network Layer Reachability Information Update (NLRI Update)
  - NLRI is the core data structure in BGP that describes how a router can reach a specific IP address. It contains:
    1. **IP Prefix** (Destination network, e.g 192.168.1.0/24)
    2. **Prefix Length** (Subnet Masks, e.g /24 → 255.255.255.0)
    3. **Attributes** (AS-Hops, Next-HOP, Local Preference etc…)

```
BGP Update:
  Withdrawn Routes: None
  NLRI: 192.168.10.0/24
  Attributes:
    - AS_PATH: 65001 65002
    - NEXT_HOP: 203.0.113.1
```

- This means that the network 192.168.10.0/24 is reachable via 203.0.113.1
- And the path to this network goes through AS 65001 → AS 65002.

How NLRI Works?

- BGP routers send Update messages containing NLRI data to inform peers about reachable networks.
- If a route becomes unavailable, router sends a withdraw msg to remove it from the routing table.
- BGP routers exchange routing tables & the update propagate prefix.

BGP Vulnerabilities & Exploits

Route Hijacking

- An attacker **announces ownership of an IP prefix they don't own**, causing traffic to be **misrouted or intercepted**.

How Route Hijacking Works:

1. Attacker **forges a BGP announcement** claiming they own a legitimate IP range.
2. Neighboring ASes **accept the fake announcement** if it offers a "shorter path."
3. Traffic destined for the real owner is **redirected through the attacker's AS**.

Mitigation:

- **RPKI (Resource Public Key Infrastructure)**: Cryptographically signs BGP prefixes.
- **Route Filtering**: ASes should **validate prefixes** before accepting them.

Re-Advertise Withdrawn Routes

A **BGP router re-announces** a withdrawn route, tricking routers to **send traffic to an invalid path**.

How it Works?

1. A router **withdraws a route** (due to network failure or administrative changes).
2. An attacker **falsely re-announces** the withdrawn route.
3. Neighboring routers **accept the fake announcement** and **resume forwarding traffic**.

Impact:

- **Network Instability:** Causes constant rerouting.
- **Man-in-the-Middle Attacks (MITM):** Intercepts data without disrupting connectivity.

Mitigation:

- **Route Flap Dampening**: Limits repeated updates for the same prefix.
- **Prefix Validation**: Only accept routes **from authorized ASes**.

BGP Link Swamping (DDoS Attack)

Attackers **flood a BGP router** with excessive updates, consuming CPU and memory.

How it Works?

1. Attackers **send thousands of fake route updates**.
2. The target **BGP router processes excessive updates**, leading to **high CPU usage**.
3. Legitimate routing updates **get delayed**, causing **network slowdowns or outages**.

Impact:

- Can **disconnect entire networks** by overloading BGP routers.
- Used in **state-sponsored cyber warfare** to disrupt internet access.

Mitigation: Rate-limiting BGP updates. & Use faster BGP processing hardware.

On path vs Off Path BGP Attacks

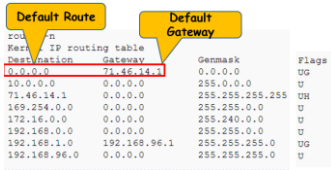
Attack Type	Description	Impact
On-Path Attacks	Attacker **controls** an AS or router, manipulating traffic in real-time.	Man-in-the-Middle, censorship.
Off-Path Attacks	Attacker **does not control** an AS but tricks routers via spoofed packets.	BGP hijacking, route injection.

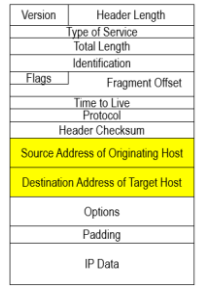
Summary of BGP:

- BGP is vital but insecure, as it was designed in an era without authentication or encryption.
- NLRI propagates network reachability info but lacks verification.
- BGP is vulnerable to hijacking, readvertising, and DDoS attacks.
- Solutions like RPKI and filtering can help, but global adoption is slow.

Internet Protocol (IP)

- On-Path Adversary (a.k.a. Man-in-the-Middle) ⟶ Has direct access to traffic.
- Off-Path Adversary ⟶ Cannot see traffic directly but can manipulate packet flow.





Field	Purpose
Version	IPv4 or IPv6
Source/Destination IP	Address of sender/receiver
TTL (Time To Live)	Prevents infinite loops
Protocol	Specifies transport layer (TCP/UDP)

Flaws in IP

- **Lack of Authentication**
  - IP **doesn't verify the sender's identity**, allowing attackers to **spoof source IP addresses**.
- **No Encryption**
  - All IP packets are **sent in plaintext** unless higher-layer encryption (like TLS/IPsec) is used.
- **Susceptibility to Fragmentation Attacks**
  - Attackers can **craft fragmented packets** to evade firewalls or inject malicious payloads.
- **No Built-in Integrity Checking:** Routers **don't verify if packets are altered** during transit.

On Path Adversary	Off Path Adversary
• **Packet Sniffing**: Reads unencrypted traffic. • **IP Data Pollution**: Corrupts packets. • **Source IP Forgery**: Spoofs sender address.	• **DDoS using Src IP Forgery** (e.g., Smurf Attack). • **Anonymous Infection** (e.g., Slammer Worm).

Attack Summary

Attack Type	On-Path Adversary	Off-Path Adversary
Packet Sniffing	Reads unencrypted packets (Wireshark)	❌ Not possible
IP Data Pollution	Modifies packet contents (MITM)	❌ Not possible
Src IP Forgery	Spoofs IP to impersonate someone	Spoofs IP for **DDoS/worm**
DDoS Attack	Uses fake IP to amplify attack	Uses **smurfing** to flood victim
Worm Prop	Injects malware into packets	Releases worms

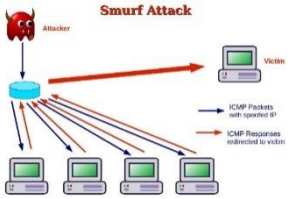
On Path Adversary Attacks (MITM)			
Attacks	**What it is**	**How it works**	**Prevention**
Packet Sniffing (Confidentiality Attack)	The attacker reads unencrypted packets in transit. Used in password theft, espionage, and data leakage.	The attacker gains access to a network (e.g., via ARP spoofing, rogue Wi-Fi, or ISP-level access). They capture packets using tools like Wireshark or tcpdump. Sensitive data (passwords, emails, banking info) can be extracted if not encrypted.	Use encryption (TLS, HTTPS, VPNs). Segment networks to limit sniffing. Monitor ARP/DNS poisoning attacks.
IP Data Pollution (Integrity Attack)	An attacker modifies packet contents to corrupt data or inject malware.	The attacker intercepts a packet. They modify the payload (e.g., injecting malware into a software download). The recipient receives tampered data without detecting the change.	Use integrity checks (e.g., TLS, digital signatures). Use VPNs to secure traffic. Enable DNSSEC for secure DNS responses.
Source IP Forgery (Spoofing)	The attacker fakes the sender's IP address, misleading the recipient.	The attacker crafts a packet with a fake source IP. The victim believes the packet came from a trusted source. This is used for bypassing firewalls, impersonation, and MITM attacks.	Implement Source Address Validation (SAV) (filters spoofed packets). Use cryptographic authentication in higher layers (TLS, SSH).
Off Path Adversary			
Attacks	**What it is**	**How it works**	**Prevention**
DDoS via Source IP Forgery (Smurf Attack)	Attacker spoofs a victim's IP and sends requests to multiple devices, overwhelming the victim with response traffic.	Attacker sends ICMP echo requests (ping) to thousands of devices. The source IP is spoofed to be the victim's address. All devices reply to the victim, flooding them with traffic (DDoS attack).	Disable ICMP responses if not needed. Use anti-spoofing filters (BCP38) to block fake source IPs. Rate-limit ICMP/DNS responses.
Anonymous Infection (Slammer Worm)	A self-propagating worm that spreads without revealing the attacker's location.	The attacker releases a worm with a spoofed source IP. The worm scans the internet, infecting vulnerable systems. Each infected machine continues spreading the worm, causing rapid global infection.	Patch vulnerabilities quickly. Use firewalls to block scans. Use intrusion detection systems (IDS).

Packet Capture Tools

- tcpdump → CLI-based packet analyzer. &
- Wireshark → GUI-based network analysis tool.

Smurf Attack (DDoS)

- Exploits **IP spoofing** to flood a target with **amplified responses**.



TCP (Transmission Control Protocol)

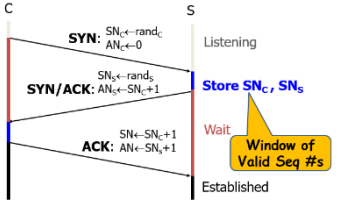
Recall TCP vs UDP:

- UDP → Unreliable Data Delivery over IP
- TCP → Reliable Data Delivery
  - Connection-oriented, Ordered Packets

Feature	TCP	UDP
Reliability	✅ Yes	❌ No
Connection	✅ Stateful	❌ Stateless
Speed	⏳ Slower	⚡ Faster

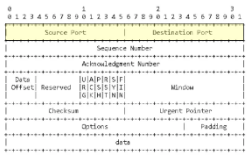
TCP Handshake

1. SYN → Client request connection
2. SYN-ACK → Server Acknowledges
3. ACK → Client Confirms



TCP Handshake & Header Consistency Across 3 Packets

Source and Destination Port & IP across the 3 packets must match



TCP Handshake

- The **TCP 3-Way Handshake** establishes a **connection** between a client and server.
- It **ensures synchronization** of sequence numbers, enabling **reliable data transfer**.

TCP Header

Each TCP segment contains a **TCP header**, which includes **critical metadata** for managing communication between sender and receiver.

Field	Size (Bits)	Description
Source Port	16	Identifies the sender's application process.
Destination Port	16	Identifies the receiver's application process.
Sequence Number	32	Ensures ordered delivery; represents the byte number in the stream.
Acknowledgment Number	32	Confirms receipt of data from the other side.
Data Offset (Header Len)	4	Indicates where the TCP header ends and data begins.
Reserved	3	Reserved for future use (usually set to 0).
Flags (Control Bits)	9	Controls TCP behavior (SYN, ACK, FIN, etc.).
Window Size	16	Specifies how much data the receiver is willing to accept.
Checksum	16	Used for error-checking the TCP header and data.
Urgent Pointer	16	Used for urgent data (rarely used).
Options	Variable	Used for extended functionality (e.g., Maximum Segment Size).

TCP Handshake Steps in Details:

Step 1: SYN (Client → Server)	• The client sends a SYN packet with an initial sequence number (ISN). • TCP Header Fields: - Source Port: Randomly chosen by the client. - Destination Port: Server's listening port (e.g., 80 for HTTP). - Sequence Number: A random initial number (e.g., 1000). - Flags: SYN = 1
Step 2: SYN-ACK (Server → Client)	• The server responds with a SYN-ACK, acknowledging the SYN. • TCP Header Fields: - Source Port: Server's listening port (e.g., 80). - Destination Port: Client's source port. - Sequence Number: A random ISN from the server (e.g., 3000). - Acknowledgment Number: ISN + 1 (e.g., 1001). - Flags: SYN = 1, ACK = 1
Step 3: ACK (Client → Server)	• The client sends an ACK, confirming the handshake. • TCP Header Fields: - Source Port: Same as the first packet. - Destination Port: Same as the first packet. - Sequence Number: ISN + 1 (e.g., 1001). - Acknowledgment Number: Server's ISN + 1 (e.g., 3001). - Flags: ACK = 1

Summary of TCP Handshake (If these don't match, the handshake fails, and the connection is rejected!)

Packet	Src Port	Dest Port	Src IP	Dest IP	Seq #	Ack #	Flags
SYN (C → S)	50000	80	192.168.1.2	10.0.0.1	1000	-	SYN
SYN-ACK (S → C)	80	50000	10.0.0.1	192.168.1.2	3000	1001	SYN,ACK
ACK (C→ S)	50000	80	192.168.1.2	10.0.0.1	1001	3001	ACK

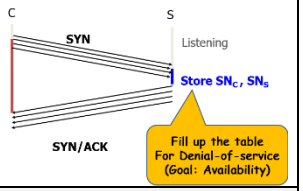
TCP Attacks

Attack 1: SYN Flood

- A **SYN Flood** attack is a **Denial of Service (DoS) attack** that **exploits the TCP three-way handshake** by sending a massive number of **SYN requests** without completing the handshake.
  - Overloads a server with **half-open TCP connections**.
- This **exhausts server resources**, making it unable to accept legitimate connections.

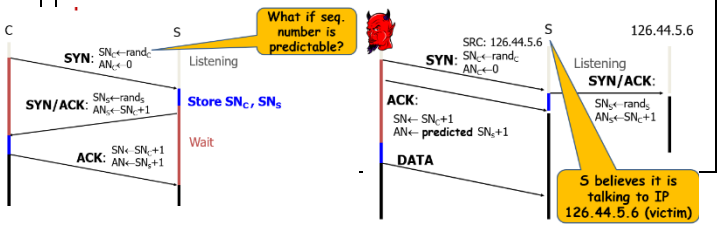
How SYN Flood Works:

1. **Attacker sends a large number of SYN packets** with fake or random source IP addresses.
2. **Server responds with SYN-ACK** to each req.
3. **Attacker never sends the final ACK**, leaving the server waiting.
4. **Server keeps half-open connections in memory**, consuming resources.
5. **Legitimate users can't connect** because the server's backlog is full.



Attack 2: Sequence Number Prediction

- A **Sequence Number Prediction attack** allows an attacker to **impersonate** a legitimate client by **guessing the TCP sequence number** used to establish a session.
- This can **bypass authentication mechanisms**, especially for services that rely on IP-based authentication. (E.g Static/Dynamic IP Auth, IPSec, Firewall rules, Geo-IP Auth, VPN)
  - Predicting TCP sequence numbers allows session hijacking.



How Sequence Number Prediction Works

1. **Attacker observes initial TCP sequence numbers** used by a victim and a server.
2. **If sequence numbers are predictable**, attacker forges a packet with the next expected number.
3. **Server accepts the forged packet**, assuming it came from the legitimate client.
4. **Attacker gains control of the session**, injecting malicious commands.

QNS	Why does this work?
ANS	• Early TCP implementations used predictable sequence numbers (e.g., incremented by a fixed value). • If an attacker can predict the next number, they can forge packets and hijack the connection.

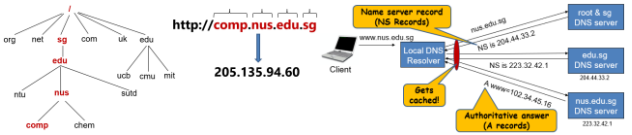
Predictable Sequence Numbers → Break IP-based Authentication

- IP-based authentication is common (and weak)! E.g. /etc/hosts, application layer gateways
- Personal Firewalls
- IP-based filtering on gateways/firewalls
- Custom User identification logic
- Apache

QNS	How Does Sequence Number Prediction Break IP-Based Authentication?
ANS	1. **Attacker Spoofs a Trusted IP** - The attacker sends a TCP **SYN packet** with a **fake src IP** (e.g., an IP from /etc/hosts). 2. **Victim Server Replies with SYN-ACK** - But the **real owner of the IP never sees this packet**, only the attacker does. 3. **Attacker Predicts the Sequence Number** - The attacker **guesses the correct sequence number** and sends a fake **ACK**, completing the handshake. 4. **Server Believes It's a Trusted User** - The server now thinks the attacker is a legitimate user. - If the service (e.g., rlogin, rsh, Apache access control) trusts this IP, **the attacker is granted access**. Real World Consequences: • **Firewall Bypass**: Attackers can spoof a trusted internal IP and get past IP-based firewall rules. • **Server Impersonation**: Attackers can inject malicious commands into an active session (e.g., rsh, telnet). • **Unauthorized Web Access**: If an **Apache** server restricts access by IP (Allow from 192.168.1.1), an attacker can spoof that IP and bypass security. Prevention (Modern Defenses) • **Use Encrypted & Strong Authentication** (e.g., SSH instead of rlogin). • **Randomize TCP Sequence Numbers** (modern OS kernels do this). • **Challenge-Response Authentication** (e.g., two-factor authentication). • **Deep Packet Inspection (DPI)** (firewalls that verify session consistency).

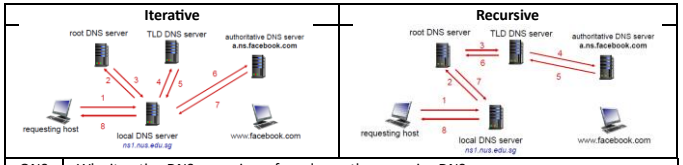
Domain Name System (DNS)

DNS translates **domain names**(e.g.google.com) to IP addr. E.g http://comp.nus.edu.sg → 205.135.94.60



- People remember **domain names**, not **IP addresses**.
  - IP addresses **change**, but domain names remain **constant**.
  - DNS enables **scalability** of the Internet by distributing name resolution.

DNS Name Resolution

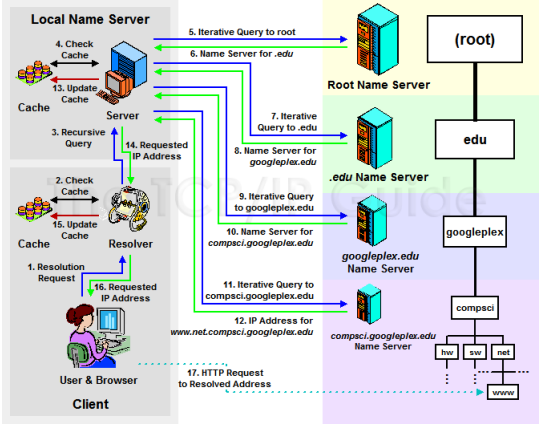


QNS	Why iterative DNS query is preferred over the recursive DNS query,
ANS	• It is basically because of security vulnerabilities. It is prone to DNS amplification and DNS cache poisoning attacks. • Just to give you an idea, lets say the recursive DNS in put in use. So, each of the DNS severs (root, TLD, authoritative name server (ANS)) communicate with other to obtain the IP.

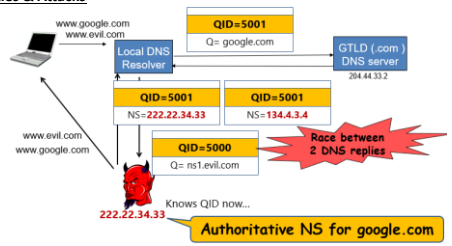
- Now if a DNS cache poisoning attack were to happen here, it would affect a much higher amount of users - as the same root/TLD/ANS servers would handle queries from many users around the world. But in the case of an iterative query, only the local DNS server's cache is poisoned - affects a much lower amount of users).

- Name servers get recorded in the Local DNS (Cache)
- Chain of Trust → Recursive, Iterative DNS Query

DNS Record Type	Purpose
A Record	Maps domain name to **IPv4 address**
AAAA Record	Maps domain name to **IPv6 address**
CNAME Record	Creates an **alias** for another domain
MX Record	Specifies **Mail Exchange** (email) servers
NS Record	Points to **Name Servers** for a domain
PTR Record	Reverse lookup (IP to Domain)
TXT Record	Stores arbitrary **text** data (e.g., SPF for email security)



DNS Vulnerabilities & Attacks



1. **DNS Cache Poisoning (DNS Spoofing)**
   - Attacker **injects fake DNS responses** into a resolver's cache.
   - Users are redirected to **malicious sites** without knowing.
   - Example: **Fake banking website** to steal credentials.
2. **DNS Pharming**
   - Modifies DNS settings **at the router level**.
   - Redirects all users of a network to a **malicious website**.
   - Common in **home routers with default passwords**.
3. **DNS Hijacking**
   - **ISP or attacker** redirects DNS queries to a malicious server.
   - Often used for **censorship** or **man-in-the-middle attacks**.
   - Example: China's **Great Firewall** modifies DNS responses.
4. **DNS Amplification Attack (DDoS)**
   - Attacker sends small DNS requests with a **spoofed source IP**.
   - The DNS server **sends large responses** to the **victim**.
   - **Floods the victim's network** (common in **botnet DDoS attacks**).

Consider a stateless network firewall that has an accuracy of 99%. 99% of the legitimate packets are allowed and 99% of malicious packets are dropped. 1,000,000 packets are passed through the firewall of which 100 of the packets are malicious and the rest are not. Answer the following.

1. How many malicious packets are dropped by the firewall?
   - 100 * 99% = 99
2. How many legitimate packets are dropped by the firewall?
   - 999,900 * 1% = 9,999
3. A packet is dropped by the firewall. What is the chance that the packet is actually legitimate (false positive)?
   - FP = FP/(TP + TN) = 9999/ (9999 + 99) = 99.02%

Chapter 4: Firewall & IDS

Firewalls

- Firewalls are security mechanisms designed to filter and control the flow of traffic between networks, based on predefined security rules.
- Their primary goal is to block unauthorized access while allowing legitimate communication.
  - Sitting at border between networks
  - Looking at services, addresses, data, and etc. of traffic
  - Deciding whether a packet should be allowed or dropped based on a firewall policy

Types of Firewalls / NIDS / IPS

Firewalls

- Used to filter and control traffic between networks.
- Enforce security policies based on predefined rules.
- **Traditional/Stateless Packet Filters** → Inspect only the header (mainly IP packet's header)
  - Applying rules to packets in/out of the firewall
  - Based on information in the packet header
- **Stateful Inspections** → Keep a state on previous received packets (e.g counting # connections from a particular IP address in the past hour).
  - Maintaining a state table of ALL active connections
  - Filtering packets based on connection states
- **Proxy-bases / Application Firewalls** → Modifies Packets
  - Understanding application logic + Acting as a relay of application level traffic

Network Intrusion Detection Systems (NIDS)

- Monitors network traffic for suspicious activities or policy violations.
- **Passive system**: detects and alerts but does not block threats.

Intrusion Prevention System (IPS)

- Monitors traffic like NIDS but can **actively block** malicious packets.
- Operates in-line with network traffic, adding latency but providing protection.

Firewall Design

- A firewall enforces a set of rules provided by the network administrator.
- Example on the 2-firewall setting:
  - Rules for Firewall1 → Block: HTTP +Allow internal to Mail Server: SMTP, POP3
  - Rules for Firewall2 → Allow from anywhere to Mail Server: SMTP only

How the rules are to be specified differ on different devices and software.

Rule	Type	Direction	Src Addr	Dest Addr	Designation Port	Action
1	TCP	In	*	192.168.1.*	25	Permit
2	TCP	In	*	192.168.1.*	69	Permit
3	TCP	Out	192.168.1.*	*	80	Permit
4	TCP	In	*	192.168.1.18	80	Permit
5	TCP	In	*	192.168.1.*	*	Deny
6	UDP	In	*	192.168.1.*	*	Deny

The rules are processed sequentially starting from rule 1, 2, …. The first matching rule determines the action. The symbol "*" matches any value. This is a symbol in "regular expression".

Firewall Rules: (3 main components of a firewall rule)

1. **Hooks to mount the rule**
   - ➢ Filtering packets for processes on the firewall computer: INPUT, OUTPUT
   - ➢ Filtering packets for other computers connected to the firewall: FORWARD
   - ➢ Network address translation (NAT): Prerouting, Postrouting
2. **Conditions** → IP address, ports, network interface, connection state
3. **Actions** → Drop, Reject, Change Packet information

Firewall Layers

Network Firewalls

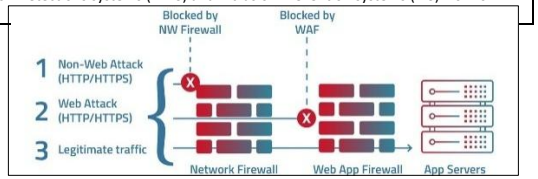
- Operate at network layer (Layer 3) and transport layer (Layer 4). →TCP/IP Layer
- Filter traffic based on IP addresses, ports, and protocols. E.g: Cisco ASA, Palo Alto, pfSense.

Network Firewalls uses Stateless Packet Filters

- These firewalls inspect each packet individually without tracking connections.
- They are fast but can be bypassed because they lack state awareness.
- Applies rules to packets in/out of firewall
  - Based on information in packet header like src/dest IP addr & port, IP protocol, interface
- Typically, there is a list of rules to check the packets if allow or deny.

Application Firewalls

- Operate at Layer 7 (Application Layer).
- Inspect traffic for specific applications (e.g., HTTP, FTP, DNS).
- Can perform deep packet inspection (DPI).
- Example: Web Application Firewall (WAF) for blocking SQL injection, XSS.
- Network Intrusion Detections systems (NIDS) and Intrusion Prevention systems (IPS) work on same principle.



Stateless Filter Packets ** Cannot prevent BGP attacks

- Stateless packet filters work based on predefined rules that either allow/block packets based on:
  - Source IP Address, Destination IP Address, Source Port
  - Destination Port, Protocol (e.g., TCP, UDP, ICMP), Interface (inbound/outbound), CtrlBit (ACK)
- Do not keep track of the state of network connections.
  - Only checks the headers of the packets
- Apply a set of filtering rules to each packet independently.
- Rules are based on fields in the packet headers (e.g., source/destination IP, port, protocol).

Stateless Filter Packet Policies:

Default = Discard (Deny by Default)	Default = Forward (Allow by Default)
• If a packet does not match any rule, it is discarded. • Provides higher security but may require frequent rule modifications. • Example: "Only allow HTTP and SSH traffic; everything else is blocked."	• If a packet does not match any rule, it is allowed. • Less secure but more user-friendly. • Example: "Block only known malicious traffic, allow everything else."

Note: Good Design Principle: Default fail-close policy

- Don't open a service to public unless it is necessary
- Good practice: Open as few ports as possible to external

Limitations of Stateless Packet Filters

- Cannot track the state of a connection (e.g., whether a connection is established or new).
- Susceptible to IP spoofing and other simple attack techniques.
- Cannot enforce complex security policies that require tracking session states.

Stateless Packet Filter Rules

- Firewall uses a list of filter rules to decide what to do with a packet
- For a packet, firewall apply the rules starting from the top of the list, and execute the operation specified by the first matching rule

Packet Filtering/Screening

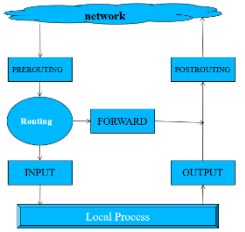
- Firewall's controls are achieved by "packets filtering" (aka screening).
- Filtering may occur in router, gateway/bridge, host, etc.
- Packet filtering inspects every packet, typically only on the TCP/IP packet's header information (network & transport layer).
- If the payload is inspected, we call it deep packet inspection (DPI).
- Action taken after inspection could be
  - To **allow** the packet to pass
  - Just **drop** the packet
    - ➢ Drop packets with "source ip-address" not within the organization's network. (To stop attacks originated within the network)

WhiteList	BlackList
Drop all packets except those specified in the white-list. (e.g. drop all except http, email protocol, and DNS)	Accept all packets except those specified in the black-list. (e.g. allow https except ip-address in the blacklist).

  - Other Actions: Reject the packet (i.e. drop and inform the sender), Log info. Notify system admin, Modify the packet (for more advanced device).

Linux Firewalls: Netfilter Framework

- Netfilter is a packet filtering framework built into the Linux kernel.
- It has 5 hooks to decide the packet's fate: (Drop, Reject, Change)
  1. Pre-routing
  2. Post-routing
  3. Forward (Something like a router)
  4. Input
  5. Output
- It allows various networking-related operations, such as:
  - Packet filtering, Network Address Translation (NAT)
  - Connection tracking



iptables Utility

- The iptables is a user-space command-line tool that configures firewall rules in the Linux Netfilter framework. E.G: allowing ssh connections to this computer

```
$ sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
```

Hook to mount the rule: INPUT
Conditions: TCP protocol: -p tcp, SSH port: --dport ssh
Action: Allow

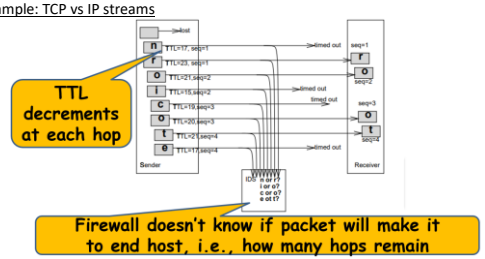
Key features:

Chains & Rules	Common Actions
• **INPUT**: Handles incoming packets • **OUTPUT**: Handles outgoing packets • **FORWARD**: Handles packets being routed through the firewall	• **ACCEPT**: Allow the packet • **DROP**: Discard the pkt without a response • **REJECT**: Discard the packet and send an error message to the sender

Firewalls & IDS/IPS: Threat Model

Goal	Stop attacker's packet from reaching the end application
Adversary's Capability	• Adversary can send malicious network packets • Adversary is outside the network perimeter
Assumptions	**1. The firewall is uncompromised** - Assumes no attacker has bypassed or disabled the firewall. **2. Defender's policy can tell bad from good traffic by inspecting pkt content** - Assumes that inspecting packet content is enough to detect attacks. **3. The firewall sees the same data as the end application** - Assumes that packets reaching the firewall are identical to those reaching internal hosts. **4. The network perimeter is correctly defined** - Assumes that all malicious traffic originates from outside the network.

Weaknesses of Firewall: In this course, weakness = defeating firewalls by violating the assumptions.

Assumption 1: The firewall is uncompromised	
Attack	• Denial of Service (DoS) attack on the firewall to overwhelm it. - Overloading the firewall with excessive connections. - Firewalls have a finite number of open TCP connections they can handle. • Distributed Denial of Service (DDoS) attack to bring down the firewall. Note: IP addresses can be rented. & Fast-Flux: Fast changing IPs for web domain
Mitigation	Rate-Limiting incoming connections, stateful filtering to track & drop anomalous traffic
Assumption 2: Firewall can distinguish between good and bad	
Challenge	• Firewall filters can look packet payloads for - **Anomalies:** Patterns that are unusual - **Signatures:** Patterns of known malicious traffic (e.g virus/exploit sigs) However… • Attackers can use the legitimate IP addresses to blend in with normal traffic. • Firewalls may inspect payloads for anomalies & known attack patterns (signatures). **Evade Signature**: • Evade signature-based detection/filtering • **Polymorphic malware**: Exploiting the same vulnerability with different inputs. (i.e modify itself, and attack again) **Evade Anomalies**: Benign but uncommon traffic may be misclassified as malicious (False Positives - FP). Difficulty in achieving low false positive rates (Base Rate Fallacy). The **Base Rate Fallacy** affects detection: • If 20 real attacks occur in **1,000,000** network events, but false positives occur **0.1%** of the time, then: - **True Positives (TP) = 20, False Positives (FP) = 1000** - Only **1 out of 50** alerts is a real attack! • In networks processing **10^8 packets per day**, excessive false positives can make detection impractical.
Mitigation	• Combining signature-based and anomaly-based filtering • Using machine learning models to improve accuracy
Assumption 3: The firewall sees the same data as end host	
Issues	Firewalls filter traffic **before** it reaches the application, but their interpretations of packets may differ due to: • **Differences in OS (host) implementations** (e.g., Windows & Linux handle packets differently). • **Network topology unknown** (e.g., additional routers may alter packet order). • **Lower-layer filtering mismatches**: - Filtering semantically at lower layers of network stack - A firewall may drop a packet, but an application might still receive some fragmented data. Example: TCP vs IP streams 
	• **TTL Behavior & Firewall Limitation:** - TTL decreases at each hop, and packets may time out before reaching the dest. - The firewall **cannot predict how many hops remain**, leading to potential discrepancies in traffic filtering. • **TCP vs. IP Streams Issue:** - Firewalls inspect packets at the **IP layer**, while end hosts **reassemble TCP streams**. - This can lead to situations where the firewall drops some packets, but the end host still reconstructs the session. • **Security Implications & Mitigation:** Firewalls may **see different data** than the end host, creating **opportunities for attackers to bypass filtering**.
Mitigation	• Ensuring firewall filtering **mirrors** host processing rules. • Using **deep packet inspection (DPI)** for context-aware filtering.
Assumption 4: The network perimeter is well defined	
Problem	• Bring Your Own Devices (BYOD) - Employees bring personal devices that connect to the internal network. - These devices may be compromised, bypassing perimeter-based security.
Mitigation	• **Zero Trust Network Architecture (ZTNA)** – Assume every device is untrusted. • Use **endpoint security agents** to enforce policies **on every device**. • Restrict access based on **identity verification and device security posture**.

Summary of Firewall Assumptions, Attack, Challenges & Mitigation Measures

Assumption	Attack	Challenge	Mitigation
Firewall is uncompromised	DoS/DDoS	Limited TCP connections, IP rental, fast-flux	Rate limiting, stateful filtering
Firewall distinguishes good from bad	Evasion techniques	Polymorphic malware, Base Rate Fallacy	Machine learning, anomaly + signature-based detection
Firewall & host see the same data	Ambiguities in packet handling	OS differences, network topology issues	Deep packet inspection, synchronization of rules
Network perimeter is well-defined	BYOD	Personal devices bypass security	Zero Trust Model, endpoint security

Chapter 5A: Secure Channels – Integrity

CIA TRIAD	Achieved Using
Confidentiality	Encryption
Integrity	MAC, Digital Signature
Authentication	MAC, Digital Signature

- To achieve Integrity, we need MAC & Digital Signatures.
- Encryption DOES NOT provide Integrity!

Message Authentication Code (MAC)

- MAC is a cryptographic scheme used to ensure both **integrity** and **authenticity** of messages. It consists of two algorithms: $MAC\ I = (S, V)\ defined\ over\ (K, M, T)\ is\ a\ pair\ of\ algs$

Signing Algorithm, S	**Verification Algorithm, V**
Input: secret key k & message m. **Outputs:** Tag t (AKA MAC or auth tag). **Notation**: $S(k, m) \rightarrow t$	**Input:** secret key k, message m, tag t **Output:** 'Yes'→ valid t, 'No' → Invalid t **Notation:** $V(k, m, t) \rightarrow \{yes, no\}$

- Ensure msg integrity & authenticity, such that the best possible attack is brute-force guessing of key.
  - A **secure MAC** ensures that no attack is more effective than **brute-force key guessing**.
- If an attacker finds a method **better than brute force**, MAC is **not secure** & **DONT provide integrity**.
- **Sender Authenticity (Signature)**
  - Sender is the entity with the shared key
  - Provides **authentication** that the msg was generated by someone with access to the secret key.
- **Integrity (MAC)**
  - Any changes to the sent message are detected during verification
  - Ensures that a message has **not been altered** (integrity).

Key properties of MAC:

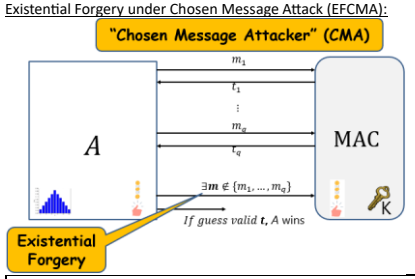
- **Unforgeability**
  - An adversary should not be able to generate a valid MAC for a new message unless they know the secret key.
  - The probability of successfully forging a MAC should be negligible (i.e., no attack should be significantly better than brute-force guessing).
- **Resistance to Chosen Message Attack (CMA)**
  - Even if an attacker can request MACs for messages of their choice, they **should not** be able to forge a MAC for a message they have not queried.
- **Brute-Force as the Best Attack**
  - The best attack an adversary can perform should be guessing the key, which requires **exponential time** (e.g., 2^n for an $n - bit$ key).
  - If an attack has a higher probability of success than brute-force guessing, the MAC is considered **insecure**.

Mathematical Definition of MAC Security:

A $MAC(S, V)$ is existentially unforgeable under a chosen message attack (EFCMA) if for any polynomial time adversary A, their probability of success forging a valid MAC for a new message m^* is negligible:

$$P[V(k, m^*, t) = yes \wedge m^* \notin Q] \leq \epsilon(n)$$

Existential Forgery under Chosen Message Attack (EFCMA):



- Q is the set of messages for which A has queried the MAC oracle.
- $\epsilon(n)$ is a negligible function.

- Chosen Message Attack → For $m_1, m_2, \ldots m_q$ attacker get $t_1, t_2, \ldots, t_q$.
- Existential Forgery → When an attacker can create a tag for another message m.

Security (integrity) of MAC	
EFCMA NOT Secure (Adversary Wins)	IF there exist an adversary A: $\exists m, \Pr_k[V(k, m, t) \rightarrow yes] > \boldsymbol{negligible}$
EFCMA Secure	IF for all adversaries A: $\forall m, \Pr_k[V(k, m, t) \rightarrow yes] \leq \boldsymbol{negligible}$

Perfectly Secure MAC

A **Perfectly Secure MAC** should ensure that an attacker **cannot forge a valid tag** for a new message, even if they have seen tags for other messages. → $|K| \geq |T|^2$ key size

Try: One-Bit MAC Note: Tag Space is size $|T| = 2^n$

Unsecure Attempts	**Explanation**
$S(m, k) = m$	• This is insecure because the MAC **reveals the message itself**. • An attacker can trivially forge a valid tag. • **Fails Existential Forgery (EFCMA Attack).**
$S(m, k) = m \oplus k$	• This is also insecure because an attacker can **observe the tag** for $m = 0$ and **flip** it to derive a valid tag for $m = 1$. • **Fails Existential Forgery (CMA Attack).**

Perfectly Secure MAC Construction:

$$S(m, k) := (k_a \cdot m + k_b)\ mod\ p$$

- The adversary **queries** $S(m_1, k)$ and $S(m_2, k)$
- Without knowledge of k_a & k_b, the adversary **cannot compute a valid tag** for a new msg m^*.
- Thus, **existential forgery fails**, making this MAC **secure**.

Where:
- p is a prime known publicly
- $\mathbb{Z}_p := \{0, \ldots, p-1\}$
- $k_a \in \mathbb{Z}_p$, and $k_b \in \mathbb{Z}_p$ chosen uniformly
- (k_a, k_b) are the shared secret key k

This ensures that:
- **Each tag looks random** because it depends on two unknown secret values.
- **An attacker cannot predict the MAC for a new message** without knowing k_a & k_b.
- **Even seeing multiple message-tag pairs doesn't help the attacker** derive new valid tags.

Formal Security Justification:

- The probability of a correct forgery attempt is at most $\frac{1}{2}$, which is equivalent to randomly guessing a bit. (Randomness over choices of (a, b)). P(successfully forging a valid tag) $at\ most, \leq \frac{1}{2^n}$.
- Formal Definition: $\forall m, m', \forall t, t'\ \Pr_{a,b}[S(m) = t \wedge S(m') = t'] = \frac{1}{|T|^2}$ **n is the bit size of tag
- This means no attacker can gain an advantage better than brute force, ensuring **perfect security** of the MAC. Note: The **probability statement holds for arbitrary prime 'p'**

Limitations of Perfectly Secure MAC

- Perfect Security has **impractical** key sizes
  - Perfect Secrecy has $|K| \geq |M|$
  - Perfect MACs have $|K| \geq |T|^2$
- Key lengths as large as message /tag length
  - **Can't reuse key bits** and preserve security claims!
- In a perfectly secure MAC, the keys must be sufficiently large to ensure that the probability of forging a tag (even with multiple attempts) remains at or below the threshold set by the definition of security.
- For a perfectly secure $n - bit$ MAC, the key size must be at least $2n$ bits to avoid vulnerability to attacks such as existential forgery.
- Probability for the best guess of t any adversary can make for any m
  - Should be at most $\frac{1}{2^n}$, as the adversary can always guess the right tag with Prob: $\frac{1}{|T|} = \frac{1}{2^n}$
- For a $1 - bit$ MAC, you need at least $2 - bit$ keys to maintain security.
- For an $n - bit$ MAC, you need at least $2n - bit$ keys.
- This ensures that the MAC remains resistant to existential forgery and that the adversary cannot do better than a random guess with a success probability of $\frac{1}{2^n}$.

Chapter 5B: Secure Channel – Intro to Symmetric Key Encryption

- A Secure Channel is a data communication protocol established between 2 programs which preserves the CIA, Confidentiality, Integrity, Authentication.
  - Note: Availability is not a goal. So, DOS attacks are permitted by the threat model.
  - Integrity is also referred to as "authenticity" sometimes, NOT "authentication"

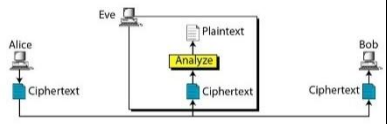
Kerchoff's Principle

- Kerckhoff's Principle states that the **security** of a cryptosystem **must lie in the choice of its keys only**; **everything else** (including the algorithm itself) should be **considered public knowledge**. we assume that the adversary knows all the algorithms.
  - A system should be secure even if everything about the system, except the secret key, is public knowledge. To hide the design of the system in order to achieve security.
- Encryption / Decryption operations are made **public**
- Cryptosystems are defined w.r.t. to an attacker model.

Models for Adversary's Capability

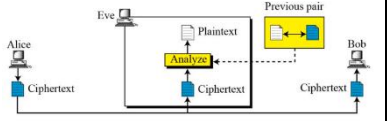
Ciphertext only Attack

- Given just the ciphertexts, guess the plaintext
- The adversary is given a collection of ciphertext c.
- The adversary may know some properties of the plaintext, for e.g. the plaintext is an English sentence. (adversary can't choose the plaintext).



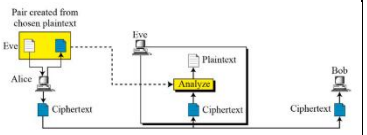
Known Plaintext Attack

- Given ciphertexts for certain plaintexts (not chosen by the adversary)
- The adversary is given a collection of plaintext m and their corresponding ciphertext c. (in this class, we assume that adversary can't choose the plaintext)



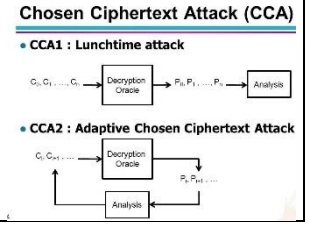
Chosen Plaintext Attack (CPA)

- Adversary gets to see ciphertexts for a chosen set of plaintext, then asked to guess for the decryption an unknown ciphertext.
- The adversary has access to a blackbox (i.e. the [encryption] oracle).
  - Can choose & feed any plaintext m to the blackbox & obtain the corresponding ciphertext c (all encrypt with the same key) for a reasonable large number of times.
  - Can see the ciphertext and then choose the next input.



Chosen Ciphertext Attack (CCA2)

- Adversary gets to see decryptions of ciphertexts, then asked to guess for plaintexts for an unknown ciphertext
- Similar to CPA, but the adversary chooses the ciphertext and the blackbox outputs the plaintext.
- We call the black-box a decryption oracle.
- The attacker can have some but not full decryption capability, e.g. padding oracle.
  - Consider oracle with full decryption capability to account for such attacks



Attack Models: Adversary's Goals

Total Break	If an attacker wants to find the key, we call this goal total break.
Partial Break	• The attacker may satisfy with a partial break. • There are a few definitions for that. For instance, the adversary may want to decrypt a ciphertext (but not interested in knowing the secret key), or the adversary may want to determine some coarse information about the plaintext (e.g., whether the plaintext is a jpeg image or a C program).
IND	• **Indistinguishability (IND)** The attacker may satisfy with distinguishability of ciphertext with some "non-negligible" probability more than ½, the attacker is able to distinguish the ciphertexts of a given plaintext (say, "Y") from the ciphertext of another given plaintext (say, "N"). • Precise notion of distinguishability not required in this course. • I.e The attacker is unable to distinguish the ciphertext of any 2 plaintexts. (OR unable to distinguish the ciphertext from a randomly sequenced text)

Note:

- Total break is the "most difficult" goal in the sense that, if an adversary is able to achieve total break, he is also able to achieve partial break and distinguishability.

	Hash Function	Symmetric Encrypt	Asymmetric Encrypt
# Crypto Keys used	0	1	2
Key Management	N/A	Big issue	Easy & Secure
Performance	Very High	High	Relatively Low
Primary Application	Password Storage Data Integrity Check	Symmetric Encryption is mostly required when dealing with transmitting bulk data, e.g banking	Digital Signatures Transport Layer Security (TLS)
Algorithms	SHA-256	AES, DES, 3DES	RSA, DHKE, ECC

Properties of Encryption Primitives: Confidentiality

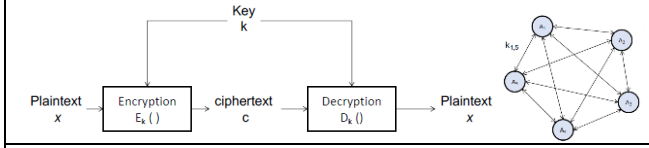
E and D are efficient algorithms, such that:

1. $D\left(k, E\left(k, m\right)\right) = m$
2. For k chosen uniformly at random, $E\left(k, m\right)$ gives no additional information about m (to the adversary who doesn't know k)

Key Encryption Schemes:

A symmetric-key encryption scheme uses the same key for encryption and decryption

- Every pair of entities require 1 key (E.g AES, DES, 3DES, RC5)
- Let $k_{i,j}$ be the keys to be shared by A_i and A_j
- Then there will be $k_{1,2}$, $k_{1,3}$, …, $k_{n,\,n-1}$ keys → Total Number of Keys: $\frac{n(n-1)}{2}$.
- Symmetric key: We need a secure channel to establish the secret key for any two entities.

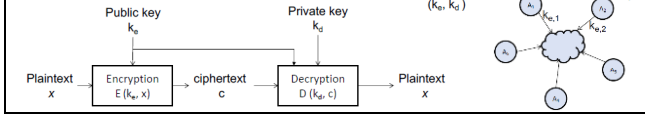


A public key (aka asymmetric-key) scheme uses different keys for encryption and decryption.

- Each entity publishes its public key (E.g RSA, DHKE, ECC)
- Entity A_i publish $k_{d,i}$ and keep $k_{e,i}$
- Total number of public keys: n; Total number of private keys: n

Note: Main advantage of PKC is that the entities do not need to know each other before broadcasting the public keys

- Public key: We only need a secure broadcast channel to distribute the public key.



Proving Cryptographic security (Cryptography uses precise threat models)

Set Up

Defines the environment & assumption under which the cryptographic system operates, including:

- Participants → Who are involved (Sender, Receiver, Adversary)
- Initialization → Starting state of the system (Key gen, Parameters etc)
- Assumptions → E.g Assume adversary knows how the system works

Example: Public Key Encryption → The key generation algorithm produces a public & private key
Assuming that the adversary cannot break the encryption easily (RSA, EC hard to resolve etc)

Adversary / Capability Model

Defines what the adversary can do, often categorized into 3 groups:

- Computational Power → State of the Art tech? computationally bounded (poly time) etc.
- Access to Information → What info will adversary have access? (e.g cipher text, oracles etc)
- Interaction → ability to interact with the system (e.g CPA, CCA etc)

Example: In the chosen plaintext attack model → The adversary can encrypt any plaintext of their choice using the public key. The adversary's goal is to learn info about a challenge ciphertext

Security Goals

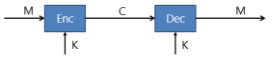
- Basic CIA Triad (PT → Plaintext, CT → Ciphertext)
  - Confidentiality → Adversary cannot learn new info about PT from CT.
  - Integrity → Adversary cannot tamper without detection
  - Authentication → Adversary cannot impersonate
- Perfect Secrecy → $\Pr[Guess = m|c] = \Pr\left[Guess = m\right]$. Does not depend on prior knowledge

Example: Indistinguishability under chosen-plaintext attack (IND-CPA):
The adversary cannot distinguish between the encryption of two chosen plaintexts.

- Once we have defined the threat models and goals, we can construct a cryptographic encryption scheme to satisfy the definition, which involves:
  - Algorithm Design → Algorithm to use (e.g KeyGen, Encryption, Decryption)
  - Correctness → Work as intended
  - Security Proof → Check if construction meets security goals under threat model
- Example: The RSA encryption scheme is a construction that aims to provide confidentiality under the assumption that factoring large integers is hard.

One Bit Encryption

- Simplest case of encryption → is 1 bit.
- Start from 1 bit and increase from there on
- Choose keys uniformly at random from {0,1}



How many deterministic functions of ENC type signature exists? → 2^4

M	K	AND
0	0	0
0	1	0
1	0	0
1	1	1

❌ Correctness
❌ Perfect Secrecy

M	K	MSG
0	0	0
0	1	0
1	0	1
1	1	1

✅ Correctness
❌ Perfect Secrecy

M	K	KEY
0	0	0
0	1	1
1	0	0
1	1	1

❌ Correctness
✅ Perfect Secrecy

M	K	RAND
0	0	$
0	1	$
1	0	$
1	1	$

❌ Correctness
✅ Perfect Secrecy

- Given C = M AND K, If K = 0, and C = 0, can we know M? → No. 3 options to choose from
- Given C = M XOR K, If K = 0, and C = 0, can we know M? → Yes. We can determine that M = 1

One Time Pad

- Encrypt: c := m⊕ k
  - k is the secret key, chosen uniformly at random
  - k changes for each plaintext message
  - ⊕ is the bitwise XOR operation (addition mod 2)
- Decrypt: m = c ⊕ k

OTP is a "Perfect" Cipher, where $\Pr[Guess = m|c] = \Pr\left[Guess = m\right]$

- One-to-One Function (Bijection)
  - ✅ Correctness

When k = 0 (k←0)	
M	C
0	0
1	1

When k = 1 (k←1)	
M	C
0	1
1	0

- Adversary cannot distinguish m = 0 vs m = 1 (i.e each value of c is equally likely with uniform k)
  - ✅ Perfect Secrecy

Proof: Let m ∈ {0,1} be with $\mathbf{Pr}[m = 0] = p$ and and $\mathbf{Pr}[m = 1] = (1 - p)$
Note: You can choose p to be any value!
For a one-time pad, what is:

1. $\boldsymbol{Pr}\left[\boldsymbol{m = 0 \mid c = 0}\right]$?
2. $Pr\left[m = 0 \mid c = 1\right]$?
3. $Pr\left[m = 1 \mid c = 0\right]$?
4. $Pr\left[m = 1 \mid c = 1\right]$?

Recall: $Pr\left[A \mid B\right] = \frac{Pr\left[A \cap B\right]}{Pr\left[B\right]}$

* The other 2,3,4 proofs follow the same logic.

Probability of $c = 0$ is defined over choices of key k and of m

- $\Pr[k = 0] = \Pr[k = 1] = \frac{1}{2}$
- $\mathbf{Pr}[\boldsymbol{c = 0}] = \Pr[k = 0 \cap m = 0] + \Pr[k = 1 \cap m = 1] = \frac{p}{2} + \frac{(1-p)}{2} = \frac{1}{2}$
- $\Pr[m = 0 \cap c = 0] = \Pr[m = 0 \cap k = 0] = \frac{p}{2}$
- $Pr[m = 0|c = 0] = \frac{\Pr[m=0 \cap c=0]}{\mathbf{Pr}[c=0]} = p$
- Proved that $\Pr[m = 0|c = 0] = \Pr[m = 0]$

Perfect Secrecy

Perfect secrecy, as defined by Shannon, ensures that an adversary with unlimited computational power gains **no information** about the plaintext from the ciphertext → $|K| \geq |M|$ key size

Limitations of Perfect Secrecy

Key Size Must be AT least as Large as the Message

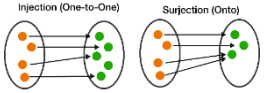
- Theorem: For any cipher to achieve perfect secrecy, the size of the key space $|K|$ must be at least as large as the size of the message space $|M|$. → $|K| \geq |M|$
- Proof: (by contradiction)

Part 1 (Proof of correctness): $|M| \leq |C|$

- ➢ For decryption to work, the encryption function MUST be injective (1-1) → Each msg maps UNIQUELY to a cipher text for a fixed key.
- ➢ If $|M| > |C|$, multiple messages will be mapped to the same cipher text & making decryption impossible (By PHP)

Part 2 (Proof of security property): $|C| \leq |K|$

- ➢ Every ciphertext MUST be EQUALLY LIKELY for any plaintext.
- ➢ If $|K| < |C|$ → C cannot be surjective (onto). Not all ciphertext in C can be mapped to a key.
- ➢ If K & C is not surjective, it violates perfect secrecy.
- ➢ Some ciphertext cannot be produced for certain plaintexts.



Combining $|M| \leq |C|$ & $|C| \leq |K|$, we get: $|K| \geq |M|$
Note: The key must be at least as long as the message. This makes key management impractical for large messages.

Key Cannot be Reused

- Perfect secrecy requires that the key is used only once.
- Reusing the key (even partially) compromises security.

Example: In the one-time pad, reusing the key allows an adversary to XOR two ciphertexts, canceling out the key and revealing information about the plaintexts.

Key Distribution problem
• Since the key must be as long as the message and cannot be reused, securely distributing keys becomes a major challenge. **Example**: If Alice and Bob want to communicate securely using a one-time pad, they must securely exchange a new key for every message, which is impractical for large-scale communication
No Practicality for Modern Cryptography
• Perfect secrecy is **theoretically sound** but **impractical** for most real-world applications due to the key size and distribution constraints. • Modern cryptography relies on **computational security**, where security is based on the assumption that certain problems (e.g., factoring large integers) are computationally hard to solve.

Can't repeat the pad bits across multiple encryptions!

- The **one-time pad** is the only cipher that achieves perfect secrecy, but it suffers from all the limitations mentioned above:
  - **Key Size**: The key must be as long as the message.
  - **Key Reuse**: Reusing the key breaks security.
  - **Key Distribution**: Securely distributing large keys is challenging.

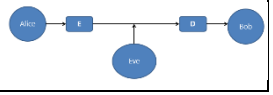
Chapter 5C: Secure Channel – Statistical vs Computational

1. **Perfect Security:** Impractical key sizes
   - **Perfect Secrecy** → has $|K| \geq |M|$ key size
   - **Perfect MACs** → have $|K| \geq |T|^2$ key size
   - Key lengths are as large as message/tag length → Can't reuse key bits & preserve security.
   - Very high overhead.
2. **Computational Hardness:**
   - Assumption that a system is safe just because it is hard to solve computationally.
   - E.g Diffie Hellman. If there are no efficient algorithm to "break in" in polynomial time = safe

Formal Definition of Computational Hardness Assumption

The security of these cryptographic primitives relies on the assumption that the adversary is computationally bounded

- Assumption 1: The adversary is efficient
  - An adversary is considered efficient if it can execute in polynomial time relative to the security parameter (e.g key size).
- Assumption 2: Polynomial Bounds
  - This means that the adversary can execute a **polynomial number** of steps.
  - The adversary should execute in a randomized, non-deterministic manner.
  - Adversary is computationally bounded.
    - ➢ Bounded queries in Chosen Plaintext Attack (CPA)/Chosen Ciphertext Attack(CCA) to polynomial in $|K|$.
- Assumption 3: Aim for Computational Secrecy instead of Perfect Secrecy
  - Ciphertext reveals NO information about the plaintext to any efficient adversary.
    - ➢ Covers all efficient attacks algorithms be definition
    - ➢ But, is less powerful than "perfect secrecy"

A Relaxed Definition of Secrecy: Computationally Hard
$\boldsymbol{E}$ and $\boldsymbol{D}$ are **computationally efficient** algorithms, such that: 1. $\boldsymbol{D}(\boldsymbol{E}(k,m)) = m$ 2. For k chosen uniformly at random, $\boldsymbol{E}(k,m)$ gives no additional information about m (to an "efficient" or "computational bounded" adversary who doesn't know k) 

Important Ideas

- Perfect secrecy, that means we don't have any assumption.
- Computational Hardness → Assume only poly time query & execution is possible. → $O(n^2)$ is secure

Practical Symmetric Encryption

- In practice, perfect security is often relaxed to computational security, where security is guaranteed only against computationally bounded adversaries.
- Use Computational Hardness Assumption

One-Way Functions (Pre-image resistant)	• These are functions that are easy to compute but hard to invert. • The existence of one-way functions is a fundamental assumption in cryptography and implies the existence of PRGs, cryptographic hash functions, and pseudorandom permutations (PRPs).
Pseudorandom Generators (PRG)	• PGR are algorithms that take a short, random seed and expand it into a longer pseudorandom string. • If a PRG is secure, the output should be indistinguishable from a truly random string to any efficient adversary. • Common Encryption scheme uses PGR: $Enc(k,m) \coloneqq PRG(k) \oplus m$ - Here, the PRG generates a pseudorandom pad that is XORed with the message m. - If the PRG is secure, this scheme provides computational secrecy.

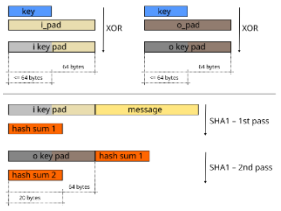
Practical MAC Constructions

- Message Authentication Codes (MACs) are used to ensure the integrity and authenticity of messages. They can be constructed using cryptographic hash functions or block ciphers.

Cryptographic Hash Functions	• These are functions that take an input and produce a fixed-size output (hash) that appears random. • They are used in constructions like HMAC (Hashed MAC).
Pseudorandom Permutations (PRP)	• These are keyed permutations that are indistinguishable from random permutations. • Block ciphers like AES are examples of PRPs and are used in MAC constructions like: OMAC (OneKey MAC), CCM (counter with CBC-MAC), and PMAC (parallelizable MAC)

Practical MAC:

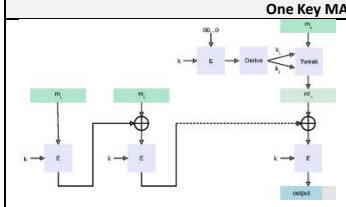
Hashed MAC (HMAC)



$$HMAC(k,m) = H((k \oplus opad)||H((k \oplus ipad)||m))$$

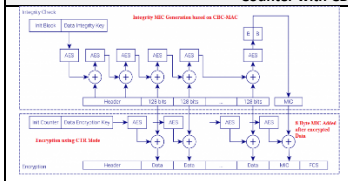
- HMAC is proven to be secure as long as the underlying hash function H is secure.
- HMAC is resistant to length extension attacks, which are a concern for some hash functions.
- Provides Integrity and Authenticity

One Key MAC (OMAC)



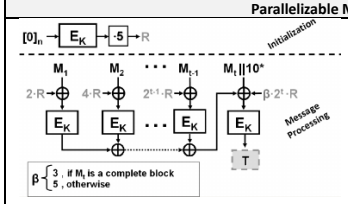
- OMAC uses a single key k and a block cipher E (e.g AES).
- The message m is divided into blocks $m_1, m_2, \ldots, m_n$.
- OMAC processes these blocks using the block cipher in CBC mode, with some additional steps to handle the final block and ensure security.
- OMAC is secure if the underlying block cipher E is a pseudorandom permutation (PRP). It is designed to be efficient and secure with a single key.

Counter with CBC-MAC (CCM)



- Operates in 2 phases:
  1. **CMC-MAC**: Compute CMC-MAC of the message m to produce a tag.
  2. **CTR mode Enc**: Encrypt msg m using CTR mode and append the tag.
- CCM is secure if the underlying block cipher E is a PRP.
- CCM provides both confidentiality and integrity, making it suitable for secure communication protocols.

Parallelizable MAC (PMAC)



- PMAC uses a block cipher E and a key k.
- It processes the message m in parallel by dividing it into blocks and applying a combination of the block cipher and a masking function to each block.
- The results are then combined to produce the final tag.
- PMAC is secure if the underlying block cipher E is a PRP.
- Its parallelizability makes it efficient for high-speed applications.

Chapter 5D: Secure Channel – Public Key Cryptography & Key Exchange

- In **asymmetric cryptography**, each party has a **public key** (known to everyone) and a **private key** (kept secret). The public key is used for encryption or verification, while the private key is used for decryption or signing.
- **Assumption**: Everyone knows each other's public keys, but private keys remain secret.

Public Key Encryption	• **Security Goal**: Confidentiality. • Analogous to symmetric-key encryption, but uses a pair of keys (public and private) instead of a shared secret key. • E.g: RSA Encryption (based on the hardness of factoring large integers).
Digital Signatures	• **Security Goal**: Integrity and Authenticity. • Analogous to symmetric-key MACs, but uses PK crypto to sign messages. • E.g: RSA Signatures or ElGamal Signatures (based on discrete logarithms).

Practical Constructions:

- **RSA**: Based on the difficulty of factoring large integers.
- **ElGamal**: Based on the hardness of the discrete logarithm problem.
- **Lattice-Based Cryptography**: Based on problems in vector geometry (used in post-quantum cryptography).

Under computational assumptions
- Difficulty of Factoring product of large primes
- Discrete Logarithm in Groups
- Problems in Lattices / Vector Geometry

Key Exchange Protocols

- Key Exchange Protocol is essential, as it enables us to **establish fresh shared secrets for each session.** → Ensures forward secrecy & limits the damage when a key is compromised.
- Even if we have pre-established secrets, we want to use refresh "keys" per session

Forward Secrecy

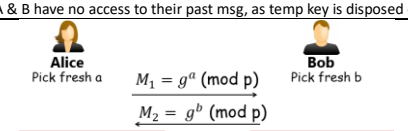
- Forward Secrecy ensures that session keys are not compromised even if long-term keys (e.g., private keys) are leaked in the future.
- Mechanism:
  - Session keys are derived from temporary keys.
  - After the session, the temporary keys are discarded.
- This protects past communications from future compromises of long-term keys

Computational Hardness Assumption: Discrete Log (DLOG)

- For an appropriately chosen group G
  - g is the generator or the group G
  - " $\cdot$ " is the group operator
  - g^a is repeated application of " $\cdot$ ", $g \cdot g \cdots (a \text{ times})$
- **Discrete Log (DLOG) Problem**:
  - Given $A \in G$ chosen randomly, it is difficult to find any $a \in \mathbb{Z}$ such that $g^a = A$
- There **exist** groups G in which DLOG is hard
  - Eg. $\mathbb{Z}_p^* \coloneqq \{1, .., p-1\}$ *with multiplication mod prime* p

Discrete Logarithm Problem (DLOG)	Given $g^a \bmod p$, it is computationally hard to find a	**Relation** • If DLOG is easy, then CDH is also easy. • The security of DHKE relies on CDH being computationally hard.
Computational Diffie-Hellman Problem (CDH)	Given $g^a \bmod p$ and $g^b \bmod p$, it is computationally hard to compute $g^{ab} \bmod p$.	

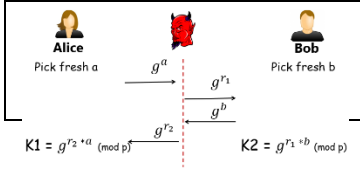
Diffie-Hellman Key Exchange ** Provides Forward Secrecy

Basic Idea	• Two parties (Alice and Bob) can securely establish a shared secret over an insecure channel using public-key principles. • Kerckhoff's Principle: (g,p,G) known to all • Alice and Bob compute the same key K. • K should not be computable by adversary. • A & B generates fresh keys everytime they communicate. → **Forward Secrecy** - These temp keys are discarded after each session. • Even A & B have no access to their past msg, as temp key is disposed of alr.
Mechanism	 1. Alice and Bob agree on a group G, a generator g, and a prime p (public info). 2. Alice chooses a private key a and sends $M_1 = g^a \bmod p$ to Bob. 3. Bob chooses a private key b and sends $M_2 = g^b \bmod p$ to Alice. 4. Both compute the shared secret: $K = g^{ab} \bmod p = (M_2)^a = (M_1)^b$.

Passive Adversary (Eve)

- Eve can only eavesdrop on the communication/traffic. **CDH → Computational Diffie-Hellman
- Eve can see $g^a \bmod p$ and $g^b \bmod p$. → Eve can see g, g^a, g^b for the chosen a, b
- Cannot compute $g^{ab} \bmod p$ if CDH is hard. → DLOG must be at least as hard as CDH
  - Breaking DHKE requires solving CDH, which is hard.
  - If DLOG is easy, we can easily get a from g^a and b from g^b and compute g^{ab}.
- Assuming CDH is hard → DHKE is secure against Eve (Passive Adversary).

Active Adversary (Mallory)



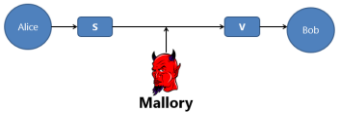
(K_1, K_2) are known to the adversary who can communications freely (MITM)

- Mallory can see and tamper with the communication/traffic (e.g., perform a MITM Attack).
- Mallory can intercept g^a and g^b, replace them with g^{r1} and g^{r2}, and compute separate keys with Alice and Bob. $K_1 = g^{r_2 * a} mod\ p$ & $K_2 = g^{r_1 * b} mod\ p$
- Alice & Bob would not even know that there is a MITM.
- DHKE does NOT achieve secure key exchange

Note: This security can be achieved by combining Digital Signatures with DHKE. (See Below)

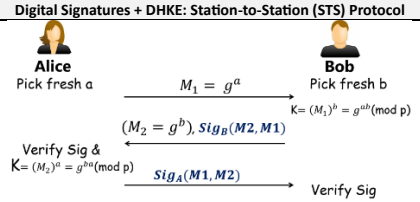
Authenticated Key Exchange

- As seen from the basic DHKE, MITM can bypass DHKE's computational hardness.
- To circumvent this issue, we will need to ensure that the communication parties are who they claim to be (Verification).
- Hence, we need to check 2 things:
  1. Are you communicating with the right person? → **Entity Authentication**
  2. Is the key used correct? → **Good Key**



An **authentication key exchange protocol** guarantees that:
- Entity Authentication: Entities are who they claim to be.
- Good Key : Only the intended parties derive shared new secret

Digital Signatures + DHKE: Station-to-Station (STS) Protocol

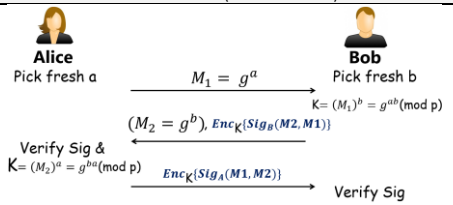


1. Alice and Bob perform DHKE to compute $K = g^{ab}\ mod\ p = (M_2)^a = (M_1)^b$.
2. They sign their messages with their private keys to prove their identities.
3. The signatures ensure that Mallory cannot impersonate Alice or Bob. (Not forgeable)

Security Properties:
- **Entity Authentication**: Ensures that Alice and Bob are who they claim to be.
- **Good Key**: Ensures that only Alice and Bob derive the shared secret.
- **Forward Secrecy**: Protects past sessions even if long-term keys are compromised.

Why does the signature help against MITM?
- The MITM will not be able to correctly sign for Alice or Bob.
- If the signature is wrong, either parties can easily detect it, and know about the MITM attack.

Full STS Procotol (Similar to above)



Why encrypt with Symmetric key?
- Symmetric key encryption is faster and more efficient for bulk data encryption vs asymmetric.
- The shared symmetric key K is used for encrypting the actual communication.

Additional Privacy
- Encrypting the signature hides the fact that A & B are communicating → Added layer of privacy.

Important Takeaways

- **Without computational assumptions**:
  - Symmetric cryptography requires impractical key sizes for perfect secrecy.
    - ➢ Secret key cryptography has impractical key sizes
  - Asymmetric cryptography relies on computational hardness assumption to achieve practical security.
- **With computational hardness assumptions**
  - Digital Signatures exist (E.g RSA, ElGamal)
  - DHKE → Secure against Eve (Passive Adversaries)
  - DHKE + Signatures → Secure against Mallory
  - Hierarchy of Security
    - ➢ DHKE + Signature → Authentication Key Exchange
    - ➢ Authenticated Key Exchange → Fresh Symmetric Keys

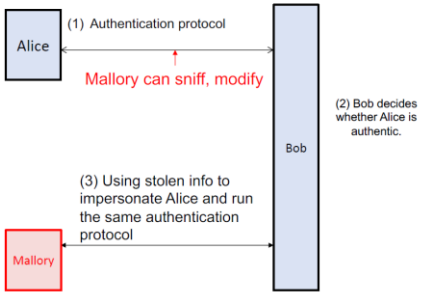
- **Why do we need KE protocols?**
  - **A way to establish fresh shared secrets**
  - Even if we have pre-established secrets, we want to use refresh "keys" per session
  - Enable Forward Secrecy to protect past communications
- Remark: (Perfect) Forward Secrecy
  - Protect Encrypted information, even **if long-term key (for client and server) is compromised**
  - **Idea**: Generate session keys, and throw away messages used to generate sessions keys…

Limitations/Attacks on PKI:

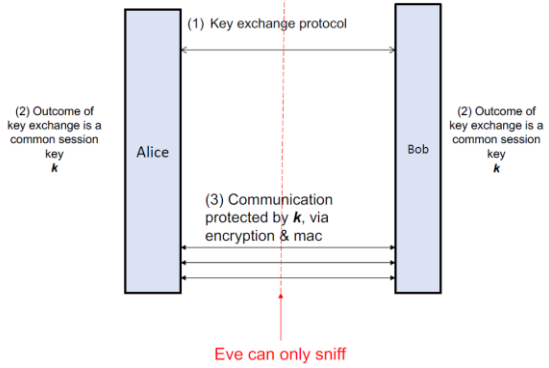
- Implementation Bugs (Browsers ignoring substrings in the name after null)
- Abuse by CA (Malicious CA → MITM → Confidentiality and Integrity compromised. E.g SuperFish)
- Social Engineering: (Generally: DNS spoofing, URL spoofing, Fake URL etc.)
  - E.g: Typosquatting, Sub-domains, Homograph attacks

Summary: (basic) Authentication

No good ☹
Attacker can sniff & modify



Summary: Key exchange



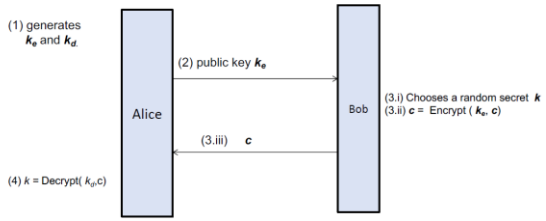
Using MAC's property, even if Eve sniffs the communication between A & B, and obtain multiple valid m & t, Eve still cant get the secret key, k, and cannot forge the MAC for the messages Eve has yet to see/sniff.

Eve can't conduct replay attacks

Unilateral Authentication via MAC
Mutual Authentication via PKC

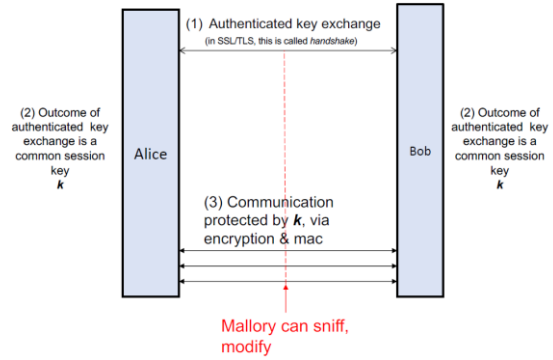
PKC-based Key-exchange

Here is a key-exchange that uses a PKC
1. Alice generates a pair of private/public key.
2. Alice sends the public key k_e to Bob.
3. Bob carries out the following
   - i. Randomly chooses a secret k,
   - ii. Encrypts k using k_e.
   - iii. Sends the ciphertext c to Alice.
4. Alice uses her private key k_d to decrypt and obtain k.



- Attacker will need to sniff, then steal session key.
- By the property of signatures, Eve can't derive Bob's private key and forge/imitate Bob's responses.
- Can use nonce to ensure freshness

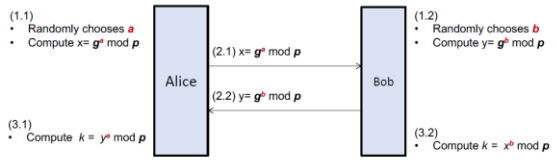
Summary: Authenticated Key exchange



- A & B authenticate via signature (PKC).
- Protection against MITM attacks
- By the security of PKC, from the public key and ciphertext, even if Eve sniffs and obtains the public key, K_e and ciphertext c, Eve can't get any information about the plaintext, which is the secret key, k.

Diffie-Hellman key-exchange

We assume both Alice and Bob have agreed on two *public* parameters, a *generator* g and a large (e.g. 1000 bits) prime p. Both g and p are not secret and known to the public.



Security relies on the CDH assumption.

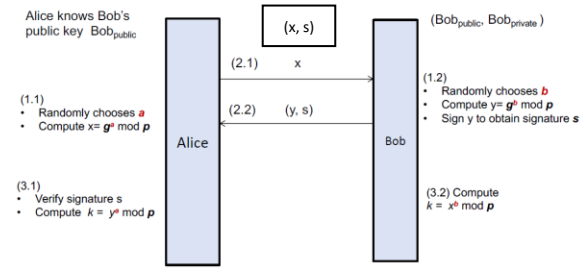
Computational Diffie-Hellman CDH assumption:
Given g, p, x= g^a mod p, y= g^b mod p, it is computationally infeasible to find k=g^{ab} mod p.

Remarks:
1. Step (1.1)&(1.2), (2.1)&(2.2), (3.1)&(3.2) can be carried out in parallel.
2. The assumption seems self-fulfilling. Nonetheless, there are many evidences that it holds.
3. The operation of "exponentiation" can be applied to any algebraic group, i.e. not necessary integers. CDH doesn't hold in certain groups. The crypto community actively searches for groups that CDH holds. E.g. Elliptic Curve Cryptography ECC is based on elliptic curve group where CDH believed to hold.

Station-to-station protocol (Authenticated key-exchange based on DH)

We assume both Alice and Bob have agreed on two *public* parameters, a *generator* g and a large (e.g. 1000 bits) prime p. Both g and p are not secret and known to the public.

Here, we consider unilateral authentication. Alice want to authenticate Bob. Can be easily extended to mutual authentication.



For mutual authentication, alice can sign at step 2.1

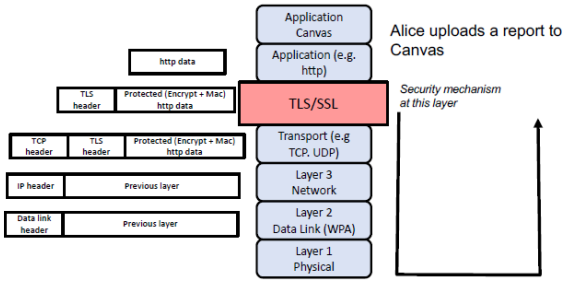
** Session keys help to contribute to forward secrecy.
Even if long-term secret keys are compromised (in the future) no one can use it to decrypt past communication sessions. Past communications are protected, and since session keys have been discarded, the attacker can only have access to current/future sessions, not past sessions.

Chapter 5E: Secure Channel – HTTPS(SSL/TLS)

- TLS/SSL sits on top of the Transport Layer.
- When an application sends data to the other end point, it first passes the data and the address (ip address, port) to the TSL/SSL.
- Then, the TLS/SSL will "protect" the data via encryption (achieving confidentiality) and MAC (achieving Authenticity). Thereafter, it instructs the transport layer to send the protected data to the corresponding end user.
- Note: HTTPS = HTTP on top of TLS

Alice using Canvas to upload a report a.pdf to the Canvas server. Note that Canvas uses https, which in turn uses SSL/TLS.

Alice's machine does the following	Canvas' Server does the following:
1. The "Canvas" application passes the file a.pdf to TLS. 2. TLS protects the data by encryption and mac. 3. TLS passes the protected data to the transport layer.	1. The transport layer passes the protected data to TLS. 2. TLS decrypt the data and verify the mac for integrity. 3. TLS passes the decrypted data Canvas's application.



The receiver end-point decrypts the received data at the corresponding layer.

A Very High-Level of SSL/TLS

SSL/TLS Protocol

The basic TLS/SSL handshake protocol is presented below. Note that we use $\{X\}_{K^{-1}}$ to denote an encrypted message X with private key K^{-1} and $\{X\}_K$ is the encrypted message with the public key K. In the Diffie-Hellman (DH) below we denote g as the group generator over a multiplicative group of integers modulo a prime p; values a and b are secrets. Assume that the browser, the CA and the CA keys are not compromised.

1. Client $\xrightarrow{\text{ClientHello}}$ Server: Client sends a **ClientHello** message containing random number R_b and supported cipher suites $Cipher_c$. The ciphersuite will select the key exchange protocol (e.g. RSA, Diffie-Hellman (DH)), the encryption algorithm for the data and the MAC algorithm.
2. Server $\xrightarrow{\text{ServerHello}}$ Client: Server sends a **ServerHello** message random number R_s, supported cipher suites $Cipher_s$.
3. Server $\xrightarrow{\text{Certificate}}$ Client: Server sends its **Certificate** containing the server's public key K_s signed with K_{auth}^{-1} where $auth$ is some signing authority (typically a CA) and ${K_{auth}}^{-1}$ is the authority's private key. If the certificate is invalid, client aborts.
4. Server $\xrightarrow{\text{ServerKeyExchange}}$ Client:
   If DH is selected, the server sends **ServerKeyExchange** message $\{g, p, g^a \bmod p\}_{K_s^{-1}}$
5. Server $\xrightarrow{\text{ServerHelloDone}}$ Client: Server signals end of handshake negotiation
6. Client $\xrightarrow{\text{ClientKeyExchange}}$ Server:
   If DH is selected, client sends $g^b \bmod p$.
   If RSA is selected, client sends $\{PS\}_{K_s}$, where PS is called PreMasterSecret and is a random value generated by the client.
   After the **ClientKeyExchange** the client and server will derive a shared key: $K_{sess} \leftarrow$ derived from DH messages 4, 6 via a standard Diffie Hellman Key Exchange procedure, or $K_{sess} = PS$ if RSA is selected. Then the client and server derive the cipher keys CK_c, CK_s and integrity keys I_c, I_s deterministically using function $\mathtt{f}()$:
   $(CK_c, CK_s, I_c, I_s) \leftarrow f(R_b, R_s, K_{sess})$
7. Client $\xrightarrow{\text{ChangeCipherSpec}}$ Server: Client sends MAC(msg, I_c)
8. Server $\xrightarrow{\text{ChangeCipherSpec}}$ Client: Server sends MAC(msg, I_s)

After the last step all data takes the form:

- Client $\rightarrow$ Server: $\{M1, \text{MAC}(M1, I_c)\}_{CK_c}$
- Server $\rightarrow$ Client: $\{M2, \text{MAC}(M2, I_s)\}_{CK_s}$

The private key is the secret that attackers don't have. Without it, the TLS handshake fails, preventing successful HTTPS impersonation.

Chain of Trust (PKI)

Public Key Infrastructure
• In Public Available Directory, its centralization may lead to server overload. As such, we require a distributed way to broadcast the keys securely. • **Centralized vs. Distributed**: - PKI provides a standardized system for securely distributing public keys, addressing the limitations of centralized methods like public directories. - Distribution of keys in a distributed manner, avoiding the potential for server overload. • **Main Objective of PKI**: - PKI aims to be deployable on a large scale, accommodating the needs of widespread use cases such as internet domains and email addresses. • **Secure Distribution**: - Users obtain public keys securely through digital certificates signed by trusted Cas. - These certs establish a chain of trust, ensuring the authenticity & integrity of public keys. • **Components**: - **Certificate**: A digital certificate contains information about the entity it belongs to (such as a website or an individual), along with the associated public key and the digital signature of the issuing CA. - **Chain of Trust**: PKI operates based on a chain of trust, where the authenticity of a certificate is verified by tracing it back to a trusted root CA. Intermediate CAs further validate and sign certificates, forming a hierarchical trust structure. **Public vs. Private PKI:** • **Public PKI**: This refers to the PKI adopted in the internet for domain names, email addresses, and other public-facing applications. It's commonly known simply as "PKI." • **Private PKI**: These are PKI systems used for specific applications or within organizations. Private PKI systems have their own set of CAs, allowing companies to establish their own trust infrastructure for internal use cases.

Certificate → A certificate is a digital document that contains **at least the following 4 main items**

Name(s)	Identifies the entity associated with the certificate, such as an email address, domain name, or wildcard domain (*.bbc.com).
Public Key	Contains the public key of the certificate **owner**, which can be used for encryption, decryption, and digital signatures.
Validity Period (Time Window)	Specifies the time window during which the certificate is considered valid.
CA Signature	Includes the digital signature of the Certificate Authority (CA), attesting to the authenticity and validity of the certificate.
Additional Info to add to Certificate	
Digest (Fingerprint)	Provides a cryptographic hash of the certificate data, enabling **verification without directly using the CA's public key**.
Metadata	Includes information about the algorithms used for encryption, key lengths, and other relevant details. (e.g ECC, RSA, key length etc)

Certificate Usage:

Type of Name	Indicates the type of entity identified by the name in the certificate, such as an email address or domain name.
Chain-of-Trust Role	Specifies whether the entity identified by the name is authorized to act as a Certificate Authority and participate in establishing the chain of trust.

Certificate Verification Process:

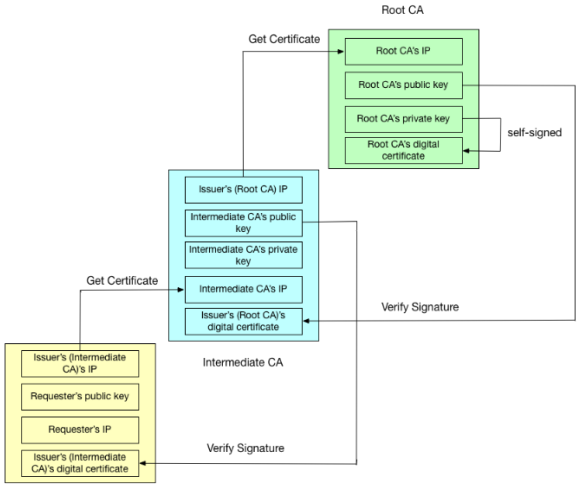
- From the certificate, Bob can obtain Alice's public key, and verifies its authenticity even without connection to the CA.
- We assume that Bob trusts the CA. → Hence, info signed/certified by the CA is also trusted.
- Since CA signed Alice's certificate, the public key extracted from the certificate is indeed Alice's.
  1. **Trust in CA:** Bob trusts the CA's public key pre-loaded in his machine.
  2. **Authenticity Verification**: Bob uses the CA's public key to verify the digital signature on Alice's certificate, ensuring its authenticity.
  3. **Public Key Extraction**: Bob extracts Alice's public key from the verified certificate
  4. **Email Authenticity Verified**: B uses A's public key to verify authenticity of email content.

Self-signed Certificate (aka root-certificate)

- A self-signed certificate is signed by its owner (i.e "name" in the cert) rather than by a trusted CA.
- The cert owner's name (the entity associated with the cert) signs the cert using its own private key.
  - It is to be verified using the "public key" listed in the certificate. Self-fulfilling verification process.

Chain of Trusted Certificates

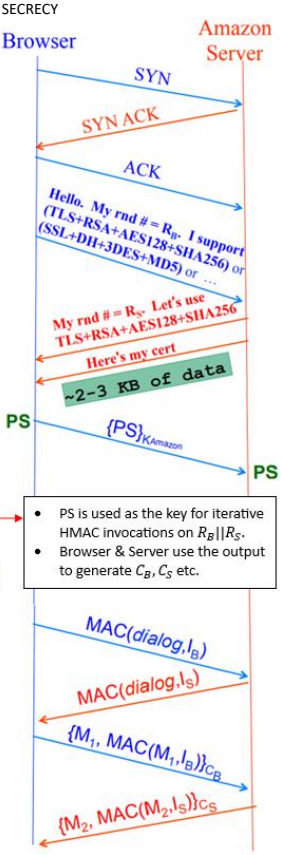
- Root CAs (e.g. GeoTrust) → Can designate Intermediate CA
  - E.g. Google Internet Authority
  - Restricted to signing certs for its subdomains
- Where does the browser start trusting?
  - Root CA's certificates are baked in your browser, ~50
- Who are the root CAs for the web?
  - Symantec (GeoTrust) – 38%; Comodo – 20%; GoDaddy – 13%; GlobalSign – 10%



Requester, Intermediate CA, and Root CA form a chain of trust. Root CA's digital certificate is self-signed. Intermediate CA get certificate from Root CA and Intermediate CA's public key is used to verify the digital certificate sent to the Requester.

HTTPS Using RSA *** DOES NOT PROVIDE FORWARD SECRECY

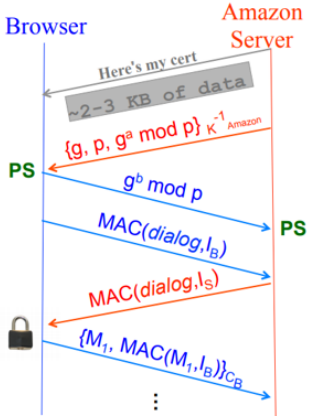
- Browser (client) connects via TCP to Amazon's **HTTPS** server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects *cipher suite* to use for this session
- Server sends over its certificate
- (all of this is in the clear)
- ***Client now validates cert***
- For RSA, browser constructs long (368 bits) "Premaster Secret" **PS**
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Using PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) & MAC *integrity keys* (I_B, I_S)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays 🔒
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Messages also numbered to thwart replay attacks



- PS is used as the key for iterative HMAC invocations on $R_B||R_S$.
- Browser & Server use the output to generate C_B, C_S etc.

HTTPS Using DHKE (----- Similar SSL Handshake Validation Logic Before DHKE ----)

- For Diffie-Hellman, server generates random a, sends public params and g^a mod p
  - Signed with server's public key
- Browser verifies signature
- Browser generates random b, computes **PS** = g^{ab} mod p, sends to server
- Server also computes **PS** = g^{ab} mod p
- Remainder is as before: from PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) and MAC *integrity keys* (I_B, I_S), etc…



Note:

- DHKE → Provides Forward Secrecy; ➔ Uses random numbers as "keys"
- RSA → DOES NOT Provide Forward Secrecy ➔Uses private keys for encryption

- RSA → PS generated by browser/client (PT) & encrypted with server's public Key
- DHKE → PS generated using random numbers, from c & s.

Security Analysis of HTTPS

Principles of Sound Logical Reasoning:

(1) State threat model, else there's no security argument!
(2) Even if assumptions fail, argument can remain valid
(3) Choose assumptions that are likely true (believable)

Chapter 5F: Secure Channel – Failures of HTTPS

Assumptions in the threat model

A threat model is a framework used to understand and assess the security risks of a system. The following are the assumptions made in a typical threat model for HTTPS:

User is using a secure channel	This assumes that the communication channel between the user and the website is encrypted (usually through HTTPS, which uses TLS).
Users check browser UI correctly	This means users are paying attention to security indicators in their browsers, like the padlock symbol and HTTPS in the URL bar, to ensure they're visiting a secure website.
Certificate issuers are uncompromised	This assumes the certificate authorities (CAs) who issue SSL/TLS certificates for websites are trustworthy and have not been hacked.
Alice & Bob's secrets are secure	Alice and Bob (in this case, the users of the system) have kept their private keys or secrets safe. If these secrets are leaked or compromised, security is broken.
Crypto primitives are secure	The cryptographic algorithms (e.g., encryption methods) used to secure the data are assumed to be strong and not vulnerable to attacks.
TLS Protocol design is secure	The design of the TLS (Transport Layer Security) protocol itself is assumed to be safe and free from design flaws that could allow attackers to exploit it.
TLS Protocol implementation is secure	The implementation of TLS on the actual server or client side is assumed to be free from bugs or vulnerabilities that attackers could exploit.
Entities are Authenticated Correctly	It is assumed that the website and users (or other entities involved) are properly authenticated, meaning they can be trusted and verified.

- DNS Cache Poisoning, BGP Route Hijacking, TCP/IP Attacks NOT possible under these assumptions.
- However, **in reality**, these assumptions often don't hold true. Attackers can still bypass strong defenses via two mechanisms: (Attackers go outside the threat model)
  1. **Falsify the assumptions**: Attackers might find ways to break or bypass one or more of the above assumptions, like faking authentication or exploiting weaknesses in the implementation.
  2. **Violate other security properties not captured by the threat model**: Sometimes, a threat model may overlook certain risks.
     - ➢ HTTPS only provides "CIA" for network (not availability)
     - ➢ Attack other layers! (Code on host, not the network)

Important Principles for Reasoning:

1) State threat model, else there's no security argument!
2) Assumptions can fail, but that's not a invalid argument
3) Choose sound assumptions in your threat model

Examples of Bypassing Threat Model:

Falsify the assumptions

In the context of HTTPS, the "threat model" refers to the set of assumptions and conditions that HTTPS is designed to protect against. However, these assumptions can sometimes be falsified or fail in practice. If any of the assumptions fail, HTTPS might not provide the level of security that it theoretically promises. Here's how:

a) User is using a secure channel
- In theory, HTTPS guarantees encrypted and secure communication over the network. However, **if the user is not using HTTPS**, then the communication will not be encrypted at all, and an attacker can intercept or manipulate the data.
- **Example**: If an attacker forces the connection to downgrade from HTTPS to HTTP (via an **HTTP downgrade attack**), the data will no longer be encrypted.

b) Users check browser UI correctly
- Users must be vigilant in checking the security of their connection. This includes looking for the padlock icon in the address bar and ensuring the URL is correct.
- **Example**: **Phishing attacks** (like **typosquatting** or **homograph attacks**) can deceive users by using domain names appearing almost identical to the intended one (e.g., www.gmaill.com vs. www.gmail.com), → fool users into trusting a malicious site, even though it uses HTTPS.

c) Certificate issuers are uncompromised
- HTTPS relies on Certificate Authorities (CAs) to issue certificates that validate the identity of websites. If a CA is **compromised** or fraudulent, an attacker could obtain a valid certificate for a malicious domain and impersonate a legitimate site.
- **Example**: If a CA is tricked into issuing a certificate for www.gmail.com to an attacker, the attacker could use this certificate for phishing or intercepting traffic.

d) Crypto primitives are secure
- HTTPS relies on strong encryption algorithms (e.g., RSA, AES) to protect the data. However, if an encryption algorithm or key length is **weak** (e.g., using deprecated algorithms like **MD5** or **SHA-1**), attackers can break the encryption.
- **Example**: A **collision attack** on a weak hash function like MD5 can allow an attacker to forge a valid certificate or data.

e) Entities are authenticated correctly
- HTTPS ensures that the entities on both ends of the communication (e.g., you & Gmail) are who they claim to be. However, if an attacker can bypass auth (e.g., by exploiting **side-channel attacks**/bugs in the TLS implementation), they can impersonate a legitimate service.
- **Example**: A **Man-in-the-Middle (MITM) attack** can occur if the attacker can intercept and alter data between the user and the website, even if HTTPS is used, by exploiting vulnerabilities in the authentication process.

In summary, if the assumptions upon which HTTPS relies are falsified (e.g., the user doesn't check certificates, the CA is compromised, or weak cryptographic primitives are used), HTTPS can be bypassed or rendered ineffective.

Violate other security properties not captured by the threat model

HTTPS primarily protects the **Confidentiality, Integrity, and Authentication (CIA)** of data in transit, but it **does not address availability** or other security properties that could affect the system overall.

a) Availability
- HTTPS focuses on ensuring the communication between the client and server is secure and private, but it doesn't guarantee that the service will always be available.
- **Example**: An attacker might perform a **Distributed Denial-of-Service (DDoS)** attack on Gmail's servers to make it unavailable, even though HTTPS protects the communication. In this case, the service is unavailable, but the communication over HTTPS, if it were still possible, would remain secure.

b) Security of Other Layers
- HTTPS primarily secures the **transport layer** (i.e., the communication between the client and the server). However, it does not protect other layers of the system that might be vulnerable. If an attacker can exploit other layers, such as the host machine or application, they can bypass the protections HTTPS offers.

Examples of Attacks on Other Layers:
- **Code on the host**: If an attacker compromises the host or the client device (e.g., by installing malware or exploiting a vulnerability), they can bypass HTTPS entirely. For example, if an attacker infects your browser with a **keylogger** or **Trojan horse**, they can steal your login credentials before they are sent over the HTTPS connection.
- **Application Layer Attacks**: Even though HTTPS ensures that the communication is secure, an attacker might exploit vulnerabilities at the application layer (e.g., **SQL injection**, **Cross-Site Scripting (XSS)**) to compromise the data being processed before it is transmitted. For instance, a vulnerable web application could allow an attacker to execute malicious code or steal data even though HTTPS is in place.
- **Local Attacks**: If an attacker has access to your machine or network (e.g., through physical access or by exploiting weak local security), they could tamper with data before it is encrypted or after it is decrypted. For example, an attacker who gains access to a device might install a **malicious root certificate**, allowing them to intercept and decrypt HTTPS traffic.

Example of a Layered Attack:

An attacker could use **Cross-Site Scripting (XSS)** to inject malicious code into a web page, which could then steal sensitive data, such as session cookies or credentials, directly from the user's browser before the data is encrypted by HTTPS. Even though HTTPS would secure the communication itself, the data could be compromised earlier in the process.

Key Takeaway:

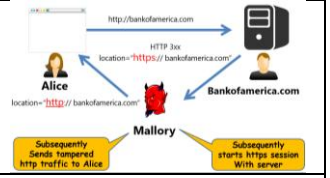
- HTTPS is not a silver bullet and must be part of a broader security strategy that includes defense-in-depth measures at other layers of the system (host, application, network, etc.).
  - HTTPS protects the CIA (Confidentiality, Integrity, Authentication) of your communication, but **it cannot protect availability**. It only ensures secure transmission of data between client and server.
  - HTTPS **does not protect against attacks on other layers of the system**, such as the host (e.g., malware, local code injections) or application (e.g., SQL injection, XSS).
- In practice, HTTPS can be bypassed if the assumptions in its threat model (e.g., proper certificate validation, strong encryption, etc.) are violated or falsified, or if attackers target other parts of the system that HTTPS does not secure.

More details on Attacks via Falsification of different assumptions

Falsify the assumptions: User is using a secure channel

HTTP Redirects

HTTPS Downgrade



- Sub-resources done have security warning for HTTP → More opportunities for downgrade.

Going from HTTP to HTTP sub-resource is NOT safe. → mixed-content issue.

- Mixed content occurs when an HTTPS page (which is secure) loads resources (such as images, scripts, stylesheets, etc.) over HTTP (insecure).

Active Mixed Contents	More dangerous, attacker inject/modify scripts E.g Javascript, CSS, iframes
Passive Mixed Contents	Less dangerous, e.g files ** Note files may be trojans E.g Images, videos, audio

- This is a security risk because the HTTP resources are not encrypted, potentially exposing the page to various types of attacks, such as:
  - **MITM Attacks**: If an attacker intercepts the HTTP traffic (which is unencrypted), they could modify or inject malicious content into the page.
  - **Data Integrity Issues**: Even if the page itself is loaded securely via HTTPS, any data sent over HTTP is not protected and could be tampered with.
  - **Privacy Violations**: Sensitive information loaded via HTTP (like user data or cookies) could be exposed to attackers on untrusted networks, e.g public Wi-Fi.
- **Protection:**
  - Ensure all resources are loaded over HTTPS
  - Use Content-Security Policy (CSP), to enforce HTTPS only & block mixed contents

QNS	If the website if using HTTP, but the login iframe uses HTTPS, is it safe? Can Mallory intercept Alice's password without Alice noticing?
ANS	Is the password safe from interception? • **HTTPS in the iframe ensures encryption**: Since the login form is loaded over HTTPS, the password is encrypted during transmission to the server. • **Mallory cannot directly intercept the password in transit** if the HTTPS implementation is correct (valid certificate, no vulnerabilities). Can Mallory modify the page to steal the password? • **Yes, because the main page is HTTP (insecure)**. Mallory can: - **Modify the HTTP page** (e.g., via a MITM attack) to inject malicious JavaScript that captures the password before it's sent via HTTPS. - **Replace the HTTPS iframe with a fake login page** (e.g., by changing the src attribute to a phishing site). - **Use a "stripping attack"** (downgrade HTTPS to HTTP if the iframe URL is not hardcoded with HTTPS). - **Cookies** →When Alice logins in, Mallory can use Alice's session cookies to resume session and login. Can Alice notice the attack? Not necessarily: • The browser **only shows a padlock for the HTTPS iframe**, not the main page. Alice might not realize the main page is insecure. • If Mallory replaces the iframe with a phishing page that also uses HTTPS (with a fake certificate), Alice might only notice if she checks the certificate details (which most users don't)

Defense Against HTTP Downgrade OR Mixing HTTP and HTTPS contents

- Use HSTS (HTTP Strict Transport Security) →Forces all connections to be HTTPS.
- Done by supplying a header over HTTPS. The browser will not issue any HTTP request if the header is in the request

Insecure Cookies

What do browsers do for mixed content?

- **Legacy**: Ignore, No security warning.
- **Recent**: Block and present new UI indicators

QNS	If website uses HTTPS, but an image uses http to load, is this safe?
ANS	**No, this is unsafe** due to cookie leakage, even though images can't execute code. 1. **Cookie Leakage (if Secure flag is unset)** - When the browser loads <img src=http://example.com/logo.gif>, it sends a request to http://example.com. - If the cookie for example.com does **not** have the Secure flag, the browser **includes it in the HTTP request**. - An attacker (Mallory) intercepting this HTTP traffic can steal the cookie. 2. **Even if the image can't run code**, the mere act of loading it **exposes sensitive cookies** over an insecure channel. How to Prevent? • Set 'secure' flag on cookies → Ensure they are ONLY sent over HTTPS • Use HSTS (HTTP Strict Transport Security). • Avoid Mixed contents and Use Content-Security-Policy (CSP)

Web Cookies (Secure Channel)

- **Cookies provide confidentiality** (if configured properly)
  - Secure cookies → Sent only over HTTPS
  - However, cookies can still be read by JS via DOM API (unless HttpOnly is set)
  - E.g: Set-Cookie: SID=abc123; Secure; HttpOnly; SameSite=Lax
    - Secure → Only via HTTPS; HttpOnly → Blocks JS; SameSite → Prevents XSS
- **Cookies DO NOT provide integrity**
  - Can be overwritten by HTTP/S requests via 'set-cookie'.
    - A secure cookie can we overwritten by an evilcooki
  - Can be modified in JS (If HttpOnly is not tagged)
  - Subdomain Takeover / Cross Site Cookie Setting
    - evil.example.com can set cookies for example.com (unless restricted by Domain attribute). →Set-Cookie: SID=malicious; Domain=example.com; Secure

Falsify the assumptions: Users check browser UI correctly

1. **Phishing** → Typo squatting (www.w.com vs www.vv.com)
2. **Spoofing** → www.google.com vs www.go0gle.com
   - Null-Byte Technique: [Don't work for modern browsers]
     - gmail.com\0.evil.com → Spoofing https://gmail.com
     - The null bytes trick some system to ignore the suffix of the domain, which inaccurately displays the url on the search bar. Attacker can clone a similar website to steal data.
3. **Homograph Attack**
   - Register domains with similar characters
   - pаypal.com → paypal.com (corresponds to Cyrillic small 'a')
   - Looks similar, but they are 2 different websites, hard to notice with quick glance
4. **Click Jacking**
   - Pages can embed iframes from 3rd-party
   - Any site can host another in <iframe>
   - Frames can overlap (multiple iframes in the same site)
   - CSS controls the transparency, location of frames & Set opacity: 0.1 or pointer-events: none

Coopting the User to Click-through

Operating System	SSL Warnings	
	Firefox	Chrome
Windows	32.5%	71.1%
MacOS	39.3%	68.8%
Linux	58.7%	64.2%
Android	NC	64.6%

Table 3: User operating system vs. clickthrough rates for SSL warnings. The Google Chrome data is from the stable channel, and the Mozilla Firefox data is from the beta channel.

Channel	SSL Warnings	
	Firefox	Chrome
Release	NC	70.2%
Beta	32.2%	73.3%
Dev	35.0%	75.9%
Nightly	43.0%	74.0%

Table 4: Channel vs. clickthrough rates for SSL warnings.

- The data from the study highlights a critical issue in web security: **users frequently ignore SSL/TLS certificate warnings**, making them ineffective at preventing phishing and man-in-the-middle (MITM) attacks.
  - Across all platforms, most users bypass warnings (e.g., 71.1% for Chrome on Windows, 64.6% on Android).
  - Firefox users are slightly more cautious (32.5% CTR on Windows) but still ignore warnings at alarming rates.

Why This Matters

- **Certificate warnings fail their purpose**: They're meant to stop users from proceeding on insecure/invalid connections, but most click through anyway.
- **Attackers exploit this**: Phishing sites with self-signed or expired certs can still trap users.
- **Blame "warning fatigue"**: Users are conditioned to ignore security dialogs due to overuse (e.g., Java updates, cookie notices).

Falsify the assumptions: Certificate issuers are uncompromised

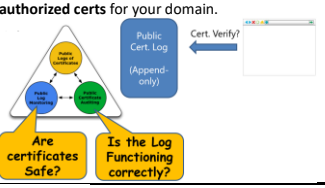
Ease of Access to SSL/TLS Certificates

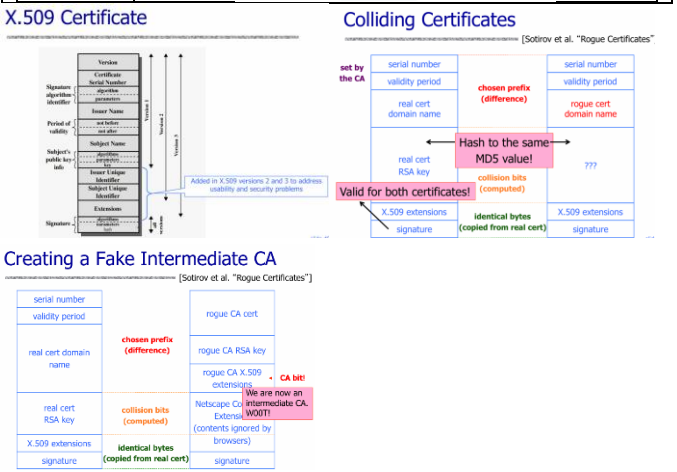
- Request a certificate from an Intermediate CA (e.g., Let's Encrypt, DigiCert, Sectigo).
- Some CAs offer free certificates (Let's Encrypt), while others charge (10–10–50/year).
- Verification Checks:
  - **Domain Validation (DV)**: Proves you control the domain (e.g., via DNS or HTTP challenge).
  - **Organization Validation (OV)**: Basic business verification (rarely used).
  - **Extended Validation (EV)**: More rigorous checks (but mostly deprecated due to low impact).
- Issue: If an attacker control evil.com, they can get a **valid HTTPS certificate** for it!

Anyone can be a CA

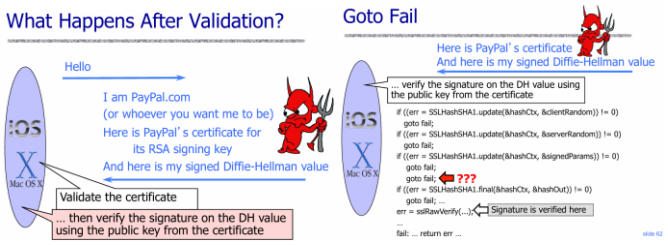
- You can **self-sign certificates** (but browsers won't trust them by default).
  - Customers would need to add you as root CA.
- Enterprises often run **private CAs** (employees must manually trust them).
- **Rogue CAs** (e.g., government-mandated CAs) can intercept traffic if forced into trust stores.
- If a CA is compromised (e.g., DigiNotar), attackers can issue **fraudulent certs** for any site.
- CA is usually built on the chain of trust. If all CA blindly trust each other, then all the CA's trust will be poisoned

Defense against Compromsied CA (How to Detect If Being Served Bad Cert)

Certificate Pinning	• Browser "pins"/caches certain certificates after the 1st visit (e.g. ab.com) • If a diff cert appears later, the browser blocks access (assuming MITM). • **Issues:** How many and which certs to pin? - **Which certs to pin?** Too many changes (e.g., CDNs, multi-CA setups) - **Misconfig risks** – A wrong pin can **break the website permanently**. - **Deprecated in favor of Expect-CT + Certificate Transparency (CT)**.
Certificate Revocation	• Idea: CA can revoke compromised certs. • Supported by OCSP (Online Certificate Status Protocol) - CA signs a revocation list - Problems? ➢ **Time delays** – Revocation isn't instant (attackers have a window). ➢ **Privacy leaks** – OCSP reveals which sites users visit. ➢ **Performance issues** – Extra network requests slow browsing. - Improvements: OCSP stapling (see Wikipedia) ➢ The **server** (not the browser) fetches & caches the OCSP response. ➢ Reduces latency and privacy leaks. (Network costs increase)
Certificate Transparency **(BEST)**	• Idea: Publicly audit all SSL certs. • All public certs are logged in public logs (e.g., crt.sh). • Browsers (Chrome, Safari) require CT for all new certificates. • Monitors can detect unauthorized certs (e.g., if facebook.com gets a cert from an unknown CA). *Anyone can audit the certs anytime in real time • Detects **unauthorized certs** for your domain. 



SSL/TLS Verification after handshake

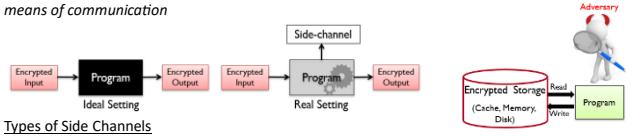


Broken TLS implementation provides no protection against the very attack it was supposed to prevent

Falsify the assumptions: Alice & Bob's secrets are secure

Side Channel Leakage

A side-channel is an unintended source of information leakage that is not designed as the primary means of communication



Types of Side Channels

- Attacker's knowledge in encrypted computation → Program logic is public
- Side-Channels → Size of Data, Timing Channels (more common), Data Access Patterns, Power Channel, Sound, Electromagnetic radiation.

Example: Timing Channel

Exponentiation is implemented via square-and-multiply RSA has $y = g^k \bmod N$

Algorithm 1 RSA - Left-to-Right Binary Algorithm

Inputs: $g, k = (k_{t-1}, \cdots, k_0)_2$ **Output:** $y = g^k$

Start:

1: $R_0 \leftarrow 1$; $R_1 \leftarrow g$
2: **for** $j = t - 1$ *downto* 0 **do**
3: $R_0 \leftarrow (R_0)^2$
4: **if** $k_j = 1$ **then** $R_0 \leftarrow R_0 R_1$ **end if**
5: **end for**
return R_0

Always executes (1 CPU cycle)

Executes conditionally (1 CPU cycle)

Leaks key via timing channel

Fix:

Same computation on both the branches
Is there any other leakage channel?

- YES

Memory access patterns reveal key bits

- Order of accessing R_0 and R_1.
- Need to be fixed using deterministic address patterns, or randomization

Algorithm 2 Montgomery Power Ladder Algorithm

Inputs: $g, k = (k_{t-1}, \cdots, k_0)_2$ **Output:** $y = g^k$

Start:

1: $R_0 \leftarrow 1$; $R_1 \leftarrow g$
2: **for** $j = t - 1$ *downto* 0 **do**
3: **if** $k_j = 0$ **then** $R_1 \leftarrow R_0 R_1$; $R_0 \leftarrow (R_0)^2$
4: **else** $R_0 \leftarrow R_0 R_1$; $R_1 \leftarrow (R_1)^2$
5: **end if**
6: **end for**
return R_0

Leakage via access patterns

- From the above, we can prevent CPU cycles analysis by having the 2 conditions run at the same CPU cycle → Attacker cannot deterministically know if the cycle = if or else.
- E.g Login page 1 sec = successful, 3 sec = unsuccessful. We want to make them both same is possible to prevent side channel attacks.
- **Defense**: Use techniques like randomization or constant-time operations to prevent attackers from gaining information through timing differences.

Falsify the assumptions: Crypto primitives are secure, TLS protocol design is secure, TLS protocol implementation is secure

Broken Crypto Primitives: E.g Broken MD5 leads to Forged Certs.

- Can attack the cryptographic signing [Sotirov et al.]
- MD5 = broken hash: can have collisions. MD5 signings can be forged, not crypto secure

Improper Use of Crypto Primitives (MAC → integrity, Enc → confidentiality)

MAC + Enc Scheme + Security	
SSH	**Encrypt – and – MAC**; E.g SSH (E (m) | MAC (m)) • Compute the MAC on the plaintext. Then encrypt the cleartext, and then append the MAC at the end of the ciphertext. • Plaintext integrity. Since MAC(Plaintext), once Ciphertext is encrypted, we can verify the MAC to see if it is correct. • No Integrity on Ciphertext, since the MAC is taken against the plaintext. This opens the door to some chosen-ciphertext attacks. • If Ciphertext is modified / cipher scheme is malleable → Plaintext will be invalid, but attackers may use this as an entry point for attack.
	• May reveal information about the plaintext in the MAC. → If plaintext msg are repeated, no counter → MAC same. - Note: For SSH, they not have a 32-bit counter. Prone to overflow attack, as the MAC will be the same. • However, it is INSECURE because the MAC does not prevent tampering with ciphertext before decryption.
SSL (Used in HTTPS)	**MAC – then – Encrypt**; E.g TLS/HTTPS (MAC (E (m))) • Compute the MAC the plaintext, append it to data, and then encrypt everything. • Does not provide any integrity on the ciphertext, since we have no way of knowing until we decrypt the msg whether it was indeed authentic or spoofed. • **Plaintext integrity**. If the cipher scheme is malleable it may be possible to alter the message to appear valid and have a valid MAC. • MAC cannot provide any info on the plaintext either, since it is encrypted. • Potentially insecure due to Padding Oracle Attacks in some implementations.
IPSec	**Encrypt – then – MAC**; E.g IPSec (E (MAC (m))) • Encrypt the cleartext, then compute the MAC on ciphertext, and append it to the ciphertext. • Provides integrity of Ciphertext. Assuming the MAC shared secret has not been compromised, we ought to be able to deduce whether a given ciphertext is indeed authentic or has been forged; for example, in public-key cryptography anyone can send you messages. It ensures you only read valid messages. • **Plaintext integrity**. If the cipher scheme is malleable we need not be so concerned since the MAC will filter out this invalid ciphertext. • The MAC does not provide any information on the plaintext since, assuming the output of the cipher appears random, so does the MAC. In other words, we haven't carried any structure from the plaintext into the MAC. • Most secure, as MAC verification happens before decrypt, preventing attacks.

Summary of MAC + Encryption Schemes

Scheme	Process	Integrity Provided?	Vulnerability?
SSH	• Encrypt-and-MAC 1. Encrypt msg, $E(ke, m)$ 2. Append tag = $S(ki, m)$ i.e compute tag on plaintext 3. Output $E(ke, m)\|\|S(ki, m)$	• Only **ciphertext integrity** (MAC checks after decryption) • MAC can leak info about PT – its not designed for secrecy.	Insecure if encryption is malleable, since ciphertext can be modified before MAC verification
SSL/TLS	• MAC-then-Encrypt 1. Add tag = $S(ki, m)$ to end of msg 2. Encrypt whole thing, $E(ke, m\|\|tag)$	• **Plaintext integrity**, but relies on encryption security • Have to decrypt first to see if the message is authentic.	Vulnerable to **Padding Oracle Attacks, BEAST Attack** (MAC check happens after decryption)
IPSec	• Encrypt-then-MAC 1. Encrypt msg, $E = (ke, m)$ 2. Append tag= $S(ki, c)$ i.e compute tag on ciphertext 3. Output $E(ke, m)\|\|S(ki, c)$	• **Provably secure integrity & confidentiality** • Ciphertext protected by MAC • Any tampering of msg → rejected immediately • MAC is applied after encryption → Cannot leak any info	No known vulnerabilities; the most secure approach

Other Crypto Implementation Flaws ** PT → Plain Text

Timing Side-Channels

Vulnerable RSA PKCS#1 v1.5 (1998)

- **Attack**: Bleichenbacher's **Million Message Attack**.
- **How it works**:
  - Observes **time differences** in RSA decryption failures.
  - If server checks padding b4 rejecting invalid ciphertexts, attacker can **iteratively guess PT**.
- **Impact**: Decrypts RSA-encrypted data (e.g., TLS session keys).
- **Fix**: Use **OAEP padding** (RSA-PSS) or constant-time checks.

Compression-Based Attacks

CRIME (2012) & BREACH (2013)

- **Attack**: Exploits **TLS compression + HTTP compression** to steal cookies.
- **How it works**:
  - Attacker injects guesses (e.g., Cookie: session=ABC...) into requests.
  - If guess matches, **compressed response is smaller** → leaks secrets.

Impact: Steals **session cookies, CSRF tokens**.

- **Fix**: **Disable TLS compression** (most browsers). + **Randomize secrets** (mask cookies w entropy).

Renegotiation Attacks (Rescorla 2009)

- **Attack**: Exploits **TLS renegotiation** to inject malicious requests.
- **How it works**:
  - Attacker starts a **fake session**, then renegotiates to a real one.
  - Server **merges both sessions**, allowing request smuggling.
- **Impact**: Man-in-the-middle (MITM) can inject commands (e.g., POST /transfer-money).
- **Fix**: **Secure Renegotiation Extension** (RFC 5746).

WPA2 Replay Attacks (2017)

- **Attack**: **Replays handshake messages** to reset nonces.
- **Impact**: **Decrypt packets, inject data**.
- **Fix**: **WPA3 (Simultaneous Authentication of Equals, SAE)**.

BEAST attack (2011)

- **Attack**: **CBC IV chaining** in TLS 1.0 allows plaintext recovery.
- **How it works**:
  - Predictable **IVs in CBC mode** let attackers XOR-guess plaintext byte-by-byte.
- **Impact**: Decrypts **HTTPS cookies, passwords**.
- **Fix**:
  - **TLS 1.1+ random IVs**.
  - **RC4 (temporarily, but RC4 is broken too!)**.

RC4 Weaknesses

No Padding used, BROKEN, gives biased stream

- **Problem**: **Biased keystream** → Statistical attacks recover plaintext.
- **Attacks**:
  - **Mantin's attack (2001)** – Predicts keystream bits.
  - **Bar Mitzvah attack (2015)** – Recovers cookies.
- **Impact**: **Totally broken** (RFC 7465 bans RC4 in TLS).

Browsers Ignoring SSL Errors (CS5331 Study)

- **Problem**: Users **click through cert warnings** 70%+ of the time.
- **Impact**: Phishing/MITM succeeds even with bad certs.
- **Fix**:
  - **HSTS (HTTP Strict Transport Security)** → Forces HTTPS.
  - **Remove "Proceed Anyway"** (Chrome now hides it).

Dual_EC Backdoor (ANSI/ISO Standard)

- **Problem**:
  - NSA **inserted a backdoor** in NIST's **Dual_EC_DRBG** (RNG).
  - Knowing a secret key, attacker can **predict all outputs**.
- **Impact**: Compromised VPNs, TLS, Bitcoin wallets.
- **Fix**: **Banned in 2014** (use AES-CTR_DRBG instead).

Note:

1. **Timing attacks** → Use constant-time crypto.
2. **Compression attacks** → Disable TLS compression.
3. **BEAST/RC4** → Use **AES-GCM (TLS 1.2+)**.
4. **Never ignore SSL errors** → Enforce HSTS.
5. **Avoid backdoored algorithms** (Dual_EC, RC4).

How to defeat strong defenses?

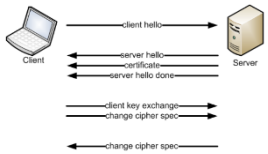
Important Meta-Point:

- We're not showing that the threat model is invalid (bad)!
- We're showing a violation of its assumptions

Principles for Reasoning:

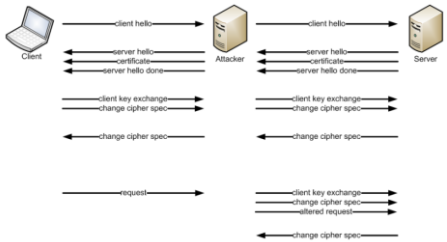
(1) **State threat model, else there's no security argument!**
(2) **Assumptions can fail, but that's not a invalid argument**
(3) **Choose sound assumptions in your threat model**

SSL Basics - Handshake



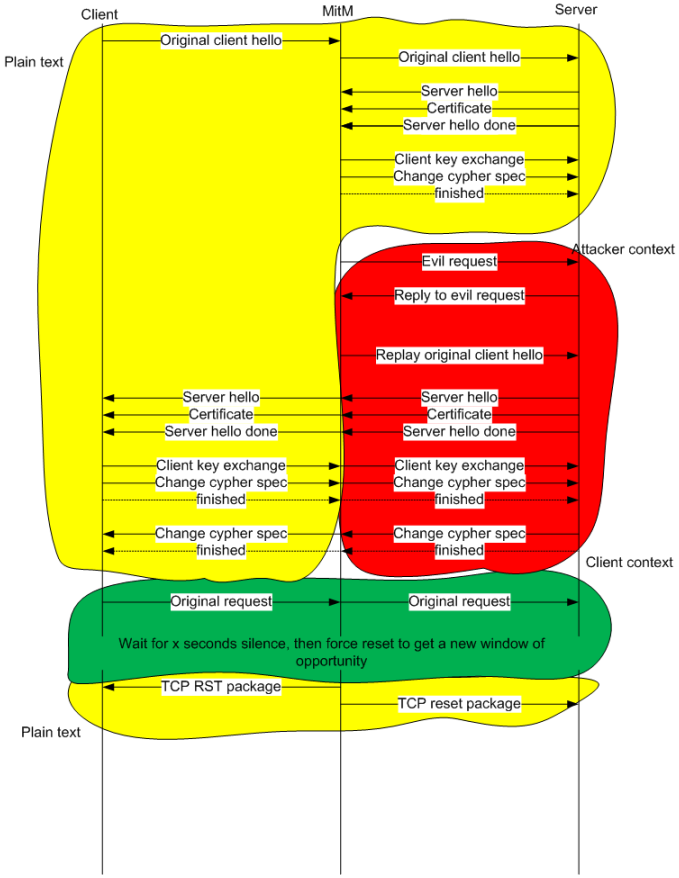
More on Renegotiation Attacks

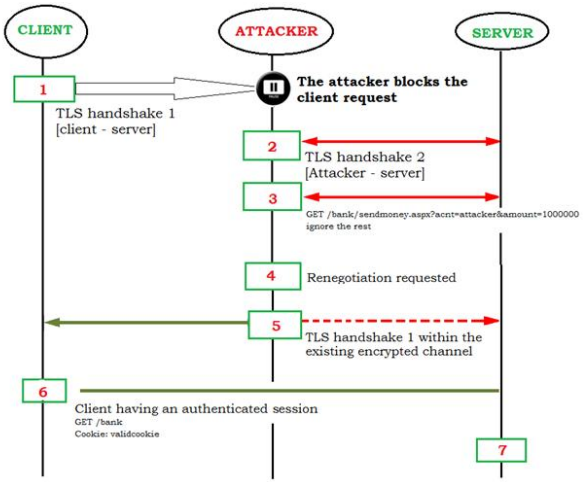
SSL Basics – Attack



- A renegotiation attack is a vulnerability in the process used to establish a secure connection with encryption protocols like SSL/TLS.
- Renegotiation attack compromises the integrity of the system.
- The vulnerability lies in how the renegotiation process is handled. The attacker might be able to manipulate or inject data during this handshake.

- **Downgrade Attack**: The attacker might downgrade the connection to a weaker encryption algorithm or even an unencrypted connection.
- **Man-in-the-Middle Attack**: The attacker might inject themselves into the renegotiation process, becoming a "man-in-the-middle" and intercepting encrypted traffic.
- **Denial-of-Service Attack**: In some cases, the attacker might exploit the renegotiation process to crash the server or client, causing a denial-of-service attack.





What can a Rogue CA do?

- The rogue CA can **create a fake certificate and private key**, to be used to **impersonate the server**.
- The **client will believe** that it talks to the **genuine** server, but in fact it talks to the impersonator.
- If the impersonator, **simultaneously, connects to the true sever** as if he was the client, this becomes a **Man-in-the-Middle attack**.
  - Client and server notice nothing, but the attacker sees everything, and can alter data at will.
- Data decryption and forward secrecy are relatively unrelated to that point.
  - In a PKI, the **CA does not normally see the servers' private keys**.
  - When a server owner wishes to **obtain an "SSL certificate**" for his server, he **generates a new key pair *locally* and sends only the public key to the CA**.
  - Since the **CA never sees the private key**, it **cannot** use it to **decrypt** recorded connections, regardless of whether the cipher suite enables forward secrecy (i.e. a "DHE" cipher suite) or not.
  - The **role of the CA** is to **guarantee the ownership of public keys**;
    - if the client really uses the true server's public key, then the cryptography protects the connection, rogue CA or not rogue CA.
    - However, there are existing CA who are on the habit of generating servers' private keys themselves.
    - They send to their customers the certificate *and* the private key together (usually as a PKCS#12 archive -- aka "pfx file").
    - They do that because it is simpler: many server administrators, with **limited competence** at what they do, have trouble handling private keys and certificates when the key pair was generated within their Web browser, on their desktop machine (not on their server).
  - When the C**A is rogue *and* generates the private key for the server, then it can keep a copy of the private key and use it to try to decrypt the data.**
    - In that case, **forward secrecy will be a problem** for that **eavesdropping rogue CA**: use of a DHE cipher suite will force the rogue CA to go to the trouble of running a fake server.
- Since **MitM attacks** are still quite feasible, a **rogue CA** is a huge problem, **and use of forward secrecy is *not* an adequate mitigation.**

Extras:

- **DNS Cache Poisoning**: Even if DNS cache poisoning redirects you to a malicious IP address, HTTPS prevents attackers from completing a TLS handshake without (e.g Google's) private key. This ensures confidentiality and integrity, with any certificate errors signaling an impersonation attempt.
- **BGP Hijacking**: Attackers can become "on-path" by manipulating BGP routing, but HTTPS ensures transmitted data remains encrypted and integrity-protected. Attackers cannot modify or intercept data unnoticed, though HTTPS doesn't guarantee availability (e.g., DoS could still occur).
  - ACK sequence property of HTTP will prevent client from receiving out-of-ordered packets and ping for a retransmission. So, if hacker drops packets, the client would not receive the proper files. Moreover, using HTTPS, the files are encrypted, so the hacker cannot modify the packet's sequence number to fudge the client's check.
    - **TCP Sequence Number Attacks**: While attackers might hijack TCP sessions at the network layer, HTTPS requires a valid TLS handshake involving Google's private key. This key is never shared publicly, making impersonation or data tampering impossible without detection.
    - **HTTPS Guarantees Confidentiality & Integrity**: HTTPS primarily ensures confidentiality and integrity but doesn't prevent attacks on availability. Successful connections (without certificate errors) guarantee that communication is secure and authentic.
      - Excessive packet dropping could disrupt communication → DOS. HTTPS DOES NOT GUARANTEE AVAILABILITY!
  - For tampered packets, HTTP has checksums built in place to check for any modified packets, and auto drops packets which are tampered + request for retransmission.
- **Precise Wording**: When discussing HTTPS protection, it's important to note that attackers can modify packets physically, but the modifications are detected by HTTPS and rendered ineffective. Terms like "tamper without detection" are more accurate than "cannot modify transmitted messages."

Memory Questions

QNS	In the x86 calling convention of GCC for LinuxLinks to an external site., to return from a function call, which of the following steps must be performed and in which order?
ANS	1. Save the return value to either a register or onto the stack as per calling convention 2. Copy stack base register to stack pointer 3. Pop stack base register from the stack 4. Pop the program counter from the stack (i.e. the RET instruction)
QNS & ANS	A process on a computer has allocated only the virtual memory region 0x6000000 - 0x8000000. What happens when it accesses 0x5000000 at runtime? a) The processor will raise a page fault and jump back to the process. b) The processor will halt. c) The processor will kill the process. d) The processor will raise a page fault and jump to the kernel.
QNS	In the x86 calling convention of GCC for LinuxLinks to an external site., to perform a function call, which of the following steps must be performed and in which order?
ANS	1. Push all function operands into registers or onto the stack 2. Push the value of the program counter onto the stack 3. Jump to the location of the desired subroutine code 4. Push the stack base pointer onto the stack 5. Copy the value of the stack pointer to the stack base register
QNS	Which of the assumptions Req1-Req3 are not required when a data-oriented attack works? • Req 1: Write Attack Payload in memory • Req 2: Have Attack Payload Be Executable • Req 3: Divert control-flow to payload
ANS	All 3 not required. Data-oriented attacks don't require payloads (req 1, 2), and there is no need to divert control flow. Data-oriented attacks manipulate existing memory data rather than introducing new malicious payloads. The program continues with its normal control-flow, but the behavior is altered by modifying data.
QNS	Which of the following are requirements of memory safety? a) The access to variables (e.g. arrays or struct) are always within its allocated range b) The access to variables are always within its lifetime (e.g., before their deallocation) c) There are no memory leaks (i.e. allocated memory should be freed when no longer being used)
ANS	a) **True**. Accessing variables within their allocated range prevents out-of-bounds errors, which are a critical component of memory safety. b) **True**. Ensuring variables are accessed only while they are valid prevents use-after-free vulnerabilities, which can lead to undefined behavior or security issues. c) **False**. While avoiding memory leaks is a good programming practice, it is not strictly a requirement of memory safety. Memory safety is primarily concerned with preventing illegal access (e.g., out-of-bounds or use-after-free), not the reclamation of memory
QNS	Which of the following is true about fat pointers? a) Fat pointers will add bound checks on every pointer arithmetic so that it should never point to out-of-bound locations b) Fat pointers take more memory bits to store than normal pointers c) The bound of a fat pointer is set once when the pointer is created and is never updated
ANS	a) **False**. While fat pointers can facilitate bounds checking, the actual implementation of bounds checks depends on the system or compiler. Fat pointers themselves do not inherently enforce bounds checks. b) **True**. Fat pointers store additional metadata, such as bounds information, alongside the pointer itself. This extra data increases the memory footprint compared to normal pointers. c) **False**. The bounds of a fat pointer can be updated depending on the implementation, especially in cases where the pointer is reassigned or manipulated.
QNS	Which of the following is true about temporal memory safety? a) Always setting a pointer to null when freeing it can prevent double free b) Lock-and-Key mechanism ensures temporal memory safety at compile time and has no runtime cost c) Lock-and-Key mechanism can prevent use-after-free
ANS	a) **False**. While setting a pointer to NULL after freeing it can prevent accidental double-free operations on that specific pointer, it does not address other pointers that may still reference the freed memory (dangling pointers). These dangling pointers can still lead to use-after-free vulnerabilities. b) **False**. The Lock-and-Key mechanism operates at runtime, as it involves dynamically associating a "key" with a memory object and a "lock" with pointers. This mechanism incurs runtime overhead due to the checks performed during pointer dereferencing. c) **True**. The Lock-and-Key mechanism ensures temporal memory safety by invalidating the "lock" associated with a memory object when it is freed. Any subsequent attempt to dereference a pointer with a mismatched "key" will be detected, preventing use-after-free vulnerabilities.
QNS	Which of the following statement(s) is/are correct? a) Stack canaries are secret data values on stack to protect against corruption of nearby data caused by a direct overflow b) Stack canaries can be used to detect out-of-bound memory read c) DEP can stop return-to-libc attacks and data-oriented attacks if the attacker can only corrupt memory pages protected by DEP (not executable). d) ASLR shuffles the relative positions of variables on stack, thus the attacker cannot predict what variables they are overwriting when buffer overflow happens. e) ASLR randomizes just the start addresses of multiple segments like the code, stack, heap, etc., but not the relative positions of the objects within each segment f) ASLR is performed at compile time so that the compiled binary must be kept secret
ANS	a) True. Stack canaries are used to detect stack buffer overflows by placing a secret value on the stack. If the value is altered, it indicates a buffer overflow attempt. b) False. Stack canaries are designed to detect buffer overflows (write operations) but cannot detect out-of-bound memory reads. c) False. DEP (Data Execution Prevention) prevents code execution in memory pages marked as non-executable. While this helps protect against direct code injection attacks, it cannot stop **return-to-libc** or **data-oriented attacks**, where attackers exploit existing code or manipulate program data without injecting executable code. These attacks work within the constraints of DEP and do not rely on executing code in non-executable memory regions, which is why DEP cannot stop them. d) False. ASLR (Address Space Layout Randomization) randomizes the start addresses of memory segments (e.g., stack, heap) but does not shuffle the relative positions of variables within a segment. e) True. ASLR randomizes the base addresses of memory segments but does not affect the relative layout of objects within those segments. f) False. ASLR is applied at runtime, not compile time, and does not require the binary to be kept secret.

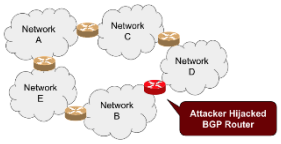
Networks Questions

QNS	Can packet sniffing be performed by an off-path or on-path attacker? • Off path attacker • On path attacker • None
QNS	Which security principles does a DoS attack primarily compromise? • Confidentiality • Integrity • Authentication • availability
QNS	Besides the network attacks taught in class, Distributed Denial of Service (DDoS) attacks are also common in practice. Please read about DoS and DDoS attacks. Which of the following statements accurately describe DDoS attacks and their characteristics? Choose all that apply. • DDoS attacks utilize a single source IP address to overwhelm a target with traffic to disrupt its services.

- DDoS attacks utilize a diverse set of IP addresses to overwhelm a target with traffic and exhaust its resources.
- Installing malware on lots of machines and BGP hijacking are both effective methods for conducting DDoS attacks.
- DDoS attacks primarily rely on exploiting vulnerabilities in a target's software and network infrastructure.

QNS

Assume that attackers control one BGP router between network B and network D as shown in the figure. Assume that benign BGP routers will forward the packets along the shortest path they have heard of. Here we consider the path length as the number of BGP router hops shown in the figure. What can attackers do with traffic from **network B to network A**? Choose all that apply.



- Attackers can perform IP forgery attack
- Attackers can perform packet sniffing attack
- Attackers can change the content of IP packets
- Attackers can drop packets

ANS

While BGP routing follows the shortest path routing, the traffic from Network B to Network A would not normally pass through the hijacked BGP router (B-x-D). Benign BGP routers will forward packets along the shortest path they have heard of, which in this case is A-E-B.

- But, attackers can still perform IP forgery attacks - DDoS via Source IP Forgery (Smurf Attack). → Attacker spoofs a victim's IP and sends requests to multiple devices, overwhelming the victim with response traffic.

However, if the attackers control the BGP router "x" between Network B and Network D, they could still potentially perform various attacks by advertising false shorter paths to divert traffic through their router. For instance, the hijacked router "x" could falsely advertise a very short route to Network A, causing benign BGP routers to reroute traffic through it. This can allow the attackers to:

- **Perform packet sniffing attacks:** By rerouting traffic through their controlled router, they can capture and inspect the packets.
- **Change the content of IP packets:** Once the traffic is diverted through the hijacked router, the attackers can modify the contents.
- **Drop packets:** They can selectively drop packets to cause disruptions.

Hence, all are possible.

DNS, Perfect Secrecy Questions

QNS

The DNS resolution is performed over transport layer protocols. Assuming the source port of **each query** sent by the local DNS resolver is chosen **independently** and **uniformly randomly** between port numbers 3001 and 60000 (including 3001 and 60000). The local DNS resolver will only listen to the response on the sending port. How many queries does an off-path attacker need **on expectation** to perform a successful attack? **Assuming that the attacker can only send one response per query.** (Please fill in a number without any space, e.g., 3235)

ANS

To determine how many queries an off-path attacker needs on expectation to perform a successful attack, we need to calculate the expected number of attempts required to guess the correct source port used by the local DNS resolver.

1. **Range of Ports**: The source port is chosen uniformly randomly between 3001 and 60000, inclusive. The total number of possible ports is: $60000 - 3001 + 1 = 57000$
2. **Probability of Success**: Since the port is chosen uniformly at random, the probability p that the attacker guesses the correct port in a single attempt is: $p = \frac{1}{57000}$
3. **Expected Number of Attempts**: The expected number of attempts E for a successful guess in a uniform distribution is the reciprocal of the probability of success: $E = \frac{1}{p} = 57000$

Therefore, the attacker needs **57000** queries on expectation to perform a successful attack. OR

1. **Source Port Range**: The source port is chosen uniformly randomly between 3001 and 60000, inclusive. This gives a total of: $60000 - 3001 + 1 = 56999$ *possible ports*
2. **Transaction ID Range**: DNS queries typically use a 16-bit Transaction ID (TXID), which has $2^{16} = 655362$ possible values.
3. **Total Combinations**: For each query, the attacker must guess both the source port and the TXID correctly. The total number of combinations is: $56999 \times 65536 = 3735552000$

QNS

Which of the following options is true about perfect secrecy?

a) Even if the attacker knows the message beforehand, after seeing the ciphertext, the attacker cannot gain any additional information about the key.
b) If the attacker has some prior knowledge of the message, after seeing the ciphertext, it is possible that the attacker can gain additional information about the key, but not the message.

ANS

- Perfect secrecy, as defined by Shannon, ensures that an adversary with unlimited computational power gains **no information** about the plaintext from the ciphertext.
- Theorem: For any cipher to achieve perfect secrecy, the size of the key space $|K|$ must be at least as large as the size of the message space $|M|$.
- Given the attacker has some prior knowledge of the message (e.g size), it can know how long the key is.
  - i.e perfect secrecy CAN reveal key length if attacker knows the message's size
- Hence, the answer is b) → "gain additional info about the key", while correct, but you can't really do much with this new information.
  - Perfect secrecy hides plaintext content and key values entirely.
  - It *may* reveal **plaintext length** (and thus minimum key length), but this is unavoidable. → This is why OTP requires $|k| \geq |m| |k| \geq |m|$.
  - **No ciphertext analysis** can reduce uncertainty about the msg or key beyond what was already known.

QNS

In the lecture, when talking about one-bit encryption, we calculated $Pr[m = 0 | c = 0]$ assuming that the key is uniformly at random from $\{0,1\}$ and $Pr[m = 0] = p$. Now let us slightly change the setup of the one-time pad. Now lets consider the case that the key k is not chosen uniformly at random. Instead, it is chosen from the Bernoulli distribution with $Pr[k = 1] = 0.25$ and $Pr[k = 0] = 0.75$. In this case, what if $Pr[m = 0 | c = 0]$ given $Pr[m = 0] = p$?

a) $\frac{3p}{2p+1}$ b) $\frac{1-p}{2p+1}$, c) $\frac{p}{3-2p}$, d) $\frac{3-3p}{3-2p}$

ANS

- The message mmm is either 0 or 1.
- The key k is sampled from a Bernoulli distribution with: $Pr[k = 1] = 0.25, \quad Pr[k = 0] = 0.75.$
- The ciphertext is computed as: $c = m \oplus k$
- The prior probability of $m = 0$ is $Pr[m = 0] = p$.
- We need to compute: $Pr[m = 0 | c = 0]$

Step 1: Compute $Pr[c = 0]$

The ciphertext $c = 0$ occurs in two cases:

1. $m = 0, k = 0$ → Probability: $Pr[m = 0] \cdot Pr[k = 0] = p \cdot 0.75 = 0.75p$
2. $m = 1, k = 1$ → Probability: $Pr[m = 1] \cdot Pr[k = 1] = (1 - p) \cdot 0.25 = 0.25(1 - p)$

Thus, the total probability of observing $c = 0$ is:

$$Pr[c = 0] = 0.75p + 0.25(1 - p) = 0.75p + 0.25 - 0.25p = 0.5p + 0.25$$

Step 2: Compute $Pr[m = 0 | c = 0]$

Using Bayes' rule:

$$Pr[m = 0 | c = 0] = \frac{Pr[m = 0, c = 0]}{Pr[c = 0]}$$

Since $Pr[m = 0, c = 0] = 0.75p$(from Step 1),

$$Pr[m = 0 | c = 0] = \frac{0.75p}{0.5p + 0.25}$$

Dividing numerator and denominator by 0.25:

$$Pr[m = 0 | c = 0] = \frac{3p}{2p + 1}$$

Hence, the answer is a)

QNS

Assuming an adversary is plotting a missile strike on an undisclosed target, the exact location of which is kept secret. The coordinates are represented as (x, y), where x and y are 32-bit signed integers in the standard two's complement representationLinks to an external site. (e.g., 1 is 0x00000001, and -1 is 0xffffffff). The adversary is transmitting this location to the missile launcher as a bit string with a length of 96. These 96 bits represent 3 signed integers concatenated in order: (x, y, checksum) where the checksum is (x XOR y).

This bit string is encrypted using a one-time pad (the encryption key is a secret random bit string with a length of 96). The encrypted message is sent via a satellite that is unaware of the encryption key, and only the missile launcher is capable of decrypting the message. You've gained unauthorized access to the adversary's satellite, enabling you to manipulate the transmitted message.

You've discovered that the satellite received the following ciphertext in hex: eaf51eb915f444ec77502396

1. Can you modify the ciphertext before retransmitting it, ensuring that the decrypted message swaps x and y, resulting in (y, x, y XOR x)? If it is possible, please provide the modified ciphertext in **lowercase hexadecimal (24 chars)**. Otherwise, please fill "NA" in the blank. _____
2. Can you alter the ciphertext before retransmitting it to achieve a decrypted message of (x, -y-1, x XOR (-y-1))? If it is possible, please provide the modified ciphertext in **lowercase hexadecimal (24 chars).** Otherwise, please fill "NA" in the blank. _____

ANS

Let the original plaintext be P = (x, y, x⊕y).

To force decrypted plaintext to be P' = (y, x, y⊕x), we need:

P' = C' ⊕ K => C' = P' ⊕ K

- But K is unknown, and we cannot compute C' without knowing K → NA.

1. Split ciphertext into 3 × 32-bit chunks (in hex):
   eaf51eb915f444ec77502396 → [eaf51eb9, 15f444ec, 77502396]
2. To get -y-1 (~y), flip all bits of the y ciphertext chunk (15f444ec):
   ~0x15f444ec = 0xea0bbb13 (bitwise NOT)
3. Update checksum:
   - Original checksum ciphertext: 77502396
   - New checksum should be x ⊕ (~y)
   - Since x ⊕ y is the original checksum, we can derive:
     x ⊕ (~y) = (x ⊕ y) ⊕ (~y ⊕ y) = (x ⊕ y) ⊕ 0xffffffff
   - Therefore, flip all bits of the checksum ciphertext:
     ~0x77502396 = 0x88afdc69
4. Construct modified ciphertext:
   [eaf51eb9, ea0bbb13, 88afdc69] → "eaf51eb9ea0bbb1388afdc69"

MAC Questions

Let's consider a MAC construction for a 1-bit message based on Perfectly Secure MAC $\boldsymbol{S(m,k) = (k_a m + k_b) \bmod p}$ we have learned in the lecture. Instead of using $p = 2$, let's consider $p = 5$.

1. How many possible key pairs (k_a, k_b)?

$$S(m,k) = (k_a \cdot m + k_b) \bmod p$$

Where $p = 5$, and $k_a, k_b \in \mathbb{Z}_5$ → $k_a, k_b \in \{0,1,2,3,4\}$

Hence, total number of possible key pairs $(k_a, k_b) = 5 \times 5 = 25$

2. Assume the CMA attacker gets to know that $t_1 = S(m,k) = S(0,k) = 3$. In the attacker's view, how many key pairs can produce $t_1 = 3$ for $m = 0$?

For $m = 0$, the MAC becomes:

$$S(0,k) = (k_a \cdot + k_b) \bmod 5 = k_b \bmod 5$$

The attacker observes $t_1 = S(0,k) = 3$ → $k_b \bmod 5 = 3$

- Thus, k_b MUST BE 3.
- However, k_a can still be any value where $k_a \in \{0,1,2,3,4\}$
- Since k_b only has 1 value, and k_a has 5 values → Number of key pairs that can produce $t_1 = 3$ for $m = 0$ is $1 \times 5 = 5$

3. What is the best possible probability for the CMA attacker to win the game (i.e., correctly forging $S(1,k)$)? Please write it as an irreducible fraction **in the form of A/B**.

- The attacker's goal is to forge $S(1,k)$ after observing $t_1 = S(0,k) = 3$.
- From the previous step (Qns 2), we derived that there are 5 possible key pairs that can be produce $t_1 = 3$ for $m = 0$.
- For $m = 1$, the MAC becomes:

$$S(1,k) = (k_a \cdot 1 + k_b) mod\ 5 = (k_a + k_b) mod 5$$

Since $k_b = 3$:

$$S(1,k) = (k_a + 3) mod 5$$

- The attacker need to guess $S(1,k)$.
- However, k_a can still be any value $k_a \in \{0,1,2,3,4\}$, so $S(1,k)$ can take any values in $\{0,1,2,3,4\}$ with equal probability.
- Hence, the probability of the attacker guessing the correct $S(1,k)$ is $\frac{1}{5}$

4. Is this construction perfectly secure (against the existential forgery attack under CMA)? (Please answer **lowercase "yes" or "no"**)

Yes.

A MAC construction is perfectly secure if the probability of the attacker successfully forging a MAC for a new message is no better than random guessing. In this case, the attacker's success probability is $\frac{1}{5}$, which is the same as random guessing (since there are 5 possible values for $S(1,k)$).

Thus, this construction is perfectly secure.

We learned the construction of perfectly secure MAC for one-bit messages in the course: $\boldsymbol{S(m,k) = (k_a m + k_b) \bmod p}$ where $p = 2$. Now let us consider a MAC construction for two-bit messages using this one-bit MAC as the constructing block.

- Assume that the two-bit message can be expressed as $(m_1||m_2)$, the concatenation of the message bits m_1 and m_2.
- The proposed way we produce a two-bit MAC is to call the perfectly secure one-bit MAC for each message bit. Assume that we have two pairs of keys for perfectly secure one-bit MAC: (a_1, b_1) and (a_2, b_2). We will use (a_1, b_1) on message bit m_1 to produce the tag t_1, and use (a_2, b_2) on message bit m_2 to produce tag t_2.
- The output is a two-bit MAC tag $(t_1||t_2)$ for the message $(m_1||m_2)$
- We refer to this as $MAC_{2bit}(m_1||m_2) = t_1||t_2$, ignoring the keys in the input.

Please answer the following questions (For probabilities, please write in the form of x/y):

1. What is the probability of guessing $MAC_{2bit}(1 \,||\, 1)$ correctly if the attacker knows that $MAC_{2bit}(0 \,||\, 0)$ =0 || 1? Please write it as an irreducible fraction **in the form of A/B.**

MAC_{2bit} construction uses 2 independent instances of the perfectly secure 1-bit MAC:

- For the first bit, m_1, the tag t_1 is computed as: $t_1 = (a_1 \cdot m_1 + b_1) \bmod 2$
- For the second bit m_2, the tag t_2 is computed as: $t_2 = (a_2 \cdot m_2 + b_2) \bmod 2$

The final MAC tag is the concatenation of t_1 and t_2 → $t_1||t_2$

$P(Guessing\ MAC_{2bit}(1 \,||\, 1)\ correctly\ if$ the attacker knows that $MAC_{2bit}(0 \,||\, 0) = 0 \,||\, 1)$:
The attacker knows $MAC_{2bit}(0 \,||\, 0) = 0 \,||\, 1$, which means that:

- For $m_1 = 0, t_1 = (a_1 \cdot 0 + b_1) \bmod 2 = b_1 \bmod 2 = 0 \rightarrow b_1 = 0$
- For $m_2 = 0, t_2 = (a_2 \cdot 0 + b_2) \bmod 2 = b_2 \bmod 2 = 1 \rightarrow b_2 = 1$

Not, the attacker wants to guess $MAC_{2bit}(1 \,||\, 1)$, for $m_1 = 1$, and $m_2 = 1$:

- $t_1 = a_1 \bmod 2 \rightarrow 2\ possibilities, t_1 \in \{0,1\}$
- $t_2 = (a_2 + 1) \bmod 2 \rightarrow 2\ possibilities, t_2 \in \{0,1\}$

Therefore, there can be $2 \times 2 = 4$ equally likely possibilities for $t_1||t_2$. Hence Prob = ¼

2. What is the probability of guessing $MAC_{2bit}(1 \,||\, 1)$ correctly if the attacker knows that $MAC_{2bit}(0 \,||\, 0)$ =0 || 1 and $MAC_{2bit}(0 \,||\, 0)$ =1 || 0 = 1 || 1? Please write it as an irreducible fraction **in the form of A/B.**

$P(Guessing\ MAC_{2bit}(1 \,||\, 1)\ correctly\ if$ the attacker knows that $MAC_{2bit}(0 \,||\, 0) = 0 \,||\, 1\ AND\ MAC_{2bit}(1 \,||\, 0) = 1 \,||\, 1)$:
The attacker knows $MAC_{2bit}(0 \,||\, 0) = t_1||t_2 = 0 \,||\, 1$, which means that:

- $b_1 = 0$ & $b_2 = 1$ (From Qns 1)

The attacker knows $MAC_{2bit}(1 \,||\, 0) = t_1||t_2 = 1 \,||\, 1$, which means that:

- For $m_1 = 1, t_1 = (a_1 \cdot 1 + b_1) \bmod 2 = (a_1 + 0) mod\ 2 = a_1 \bmod 2 = 1 \rightarrow a_1 = 1$
- For $m_2 = 1, t_2 = (a_2 \cdot 0 + b_2) \bmod 2 = (b_2) mod\ 2 = 1 \rightarrow b_2 = 1$

Now, the attacker knows: $a_1 = 1, b_1 = 0, b_2 = 1, but\ a_2\ is\ unknown$

The attacker wants to guess $MAC_{2bit}(1 \,||\, 1)$. For $m_1 = 1$ and $m_2 = 1$:

- $t_1 = (a_1 \cdot 1 + b_1) \bmod 2 = (1 + 0) \bmod 2 = 1$
- $t_2 = (a_2 \cdot 1 + b_2) \bmod 2 = (a_2 + 1) \bmod 2$
  - Since a_2 is unknown, t_2 can be:
    - If $a_2 = 0 \rightarrow t_2 = (0 + 1) \bmod 2 = 1$
    - If $a_2 = 1 \rightarrow t_2 = (1 + 1) \bmod 2 = 0$

Hence, there will be 2 possibilities for $t_1||t_2 \rightarrow \{0,1\}$. Hence Prob = ½

3. Is this construction perfectly secure (against the existential forgery attack under CMA)?

NO.

No, this construction is **not perfectly secure**. The attacker can exploit the structure of the MAC to gain information about the keys and forge tags for new messages. For example, in the second case, the attacker could deduce a_1 and b_1 and reduce the uncertainty about the tags. This violates the definition of perfect security, which requires that the attacker's success probability is no better than random guessing.

HTTPS Questions

The following are steps in an SSL/TLS handshake process based on DHKE between one client and one server for HTTPS connection. Assume that the public and private keys of the server are K_{spub} and K_{spriv} respectively.

Step 1	The server and the client send encrypted HTTP messages with the derived secret from PS in the session.
Step 2	The server sends $\{g, p, g^a \bmod p\}$ signed with K_{spriv}
Step 3	Both parties compute pre-master secret ($PS = g^{ab}$)
Step 4	The client sends $g^b \bmod p$
Step 5	The server and the client verify the integrity of their dialog till this point
Step 6	The client and the server negotiate the cipher suites. The client checks the validity of the certificate provided by the server

QNS

In the lecture, we learned that forward secrecy means the encrypted information is still protected, even if the long-term key (for the client or server) is compromised.
Which statement(s) is/are correct about forward secrecy? Choose all that apply.

a) The long-term key means the private key of the client or server
b) The long-term key means the shared premaster secret between the client and server
c) HTTPS based on DHKE provides forward secrecy
d) HTTPS based on RSA encryption on a client-chosen PS provides forward secrecy

ANS

a) Correct. Longterm Key → Persistent cryptographic material, e.g private keys.
b) Incorrect. Premaster Secret (PS) is a session key. Forward secrecy is not a long term key
c) Correct. DHKE uses session keys via random number generation. Provides Forward Sec.
d) Incorrect. RSA uses private keys enc/decr. Compromises when key is stolen. No FS.

Please choose all that apply for two blanks and **use capitalized characters concatenated with a comma without space**, e.g., "A" or "A,B,C".

- A: Prevent a passive attacker from eavesdrop the transmitted message encrypted with the keys derived by Alice and Bob
- B: Prevent an active attacker from performing an MITM impersonation attack
- C: Prevent an active attacker from dropping packets to block the communication

1) We have learned the full STS protocol with authentication in the lecture (page 19 of slide 4e). Which of the following attacks can it prevent? → A,B

The full Station-to-Station (STS) protocol involves:

- Diffie-Hellman key exchange for confidentiality.
- Signature verification for authentication.
- MAC (Message Authentication Code) for integrity.

What it prevents:

- **A:** Prevents passive eavesdropping—Diffie-Hellman ensures secure key derivation for encryption.
- **B:** Prevents MITM impersonation attacks—Signature verification authenticates the identities of Alice and Bob.
- The full STS protocol ensures mutual authentication and key agreement, which prevents passive eavesdropping (**A**) and active MITM (Man-in-the-Middle) impersonation attacks (**B**).
- However, it cannot prevent an active attacker from dropping packets (**C**) because dropping packets is a denial-of-service (DoS) attack, which is outside the scope of cryptographic protocols. **C is incorrect** in this context because while a MAC can detect and prevent tampering (ensuring integrity), it cannot prevent packet dropping. Dropped packets are not detectable as they never reach the recipient, and MACs cannot address this issue.

2) If we remove the signature verification from the full STS protocol, what can it prevent? → A.

- Removing signature verification means authentication is lost. However, Diffie-Hellman key exchange is still intact for encryption.
- Without signature verification, the protocol can still encrypt messages, preventing passive eavesdropping (A).
- However, it cannot prevent active attacks like MITM impersonation (B) because the lack of signature verification removes the ability to authenticate the parties.
- Similarly, it cannot prevent packet dropping (C) for the same reason as above.

We have learned about several network attacks before. Now, assuming the assumptions in the threat model of HTTPS hold (as shown on page 19 of the 4f-HTTPS slide), let's see whether HTTPS can beat these attacks and their effects. If so, how? Please choose the most suitable one.

1) **Defeats DNS Cache Poisoning?**

A. HTTPS can prevent the DNS cache poisoning to the local DNS resolver
B. After successful DNS cache poisoning for google.com with an IP controlled by the attackers, the HTTPS connection from a benign client to the malicious IP server will fail because the attacker does not have the certificate issued to google.com
C. After successful DNS cache poisoning for google.com with an IP controlled by the attackers, the HTTPS connection from a benign client to the malicious IP server will fail because the attacker does not have the private key of the google.com server

- A is false because HTTPS does not directly prevent DNS cache poisoning. While HTTPS ensures secure communication, it does not stop attackers from poisoning the DNS cache.
  - When your browser queries DNS for google.com, it has no way to cryptographically verify the response (unless DNSSEC is used).
  - HTTPS only verifies the server's identity after DNS resolution, using TLS certificates.
  - If DNS is poisoned, your browser will connect to the attacker's IP, but HTTPS will fail if the attacker lacks a valid cert
  - TLDR: HTTPS can detect if a DNS has been poisoned via the certificates (legit/fake), but it cannot prevent the DNS from being poisoned.
    - ➢ If self signed, the browser will show a warning → Indicate poisoned DNS.
- B is false because the failure of the HTTPS connection is not due to the lack of a certificate issued to "google.com." Attackers could potentially create a self-signed certificate, but this would trigger a browser warning.
- C is true because, after DNS cache poisoning, the attacker cannot establish a valid HTTPS connection with the client. This is because the attacker does not possess the private key corresponding to the legitimate certificate for "google.com." Without the private key, the attacker cannot complete the SSL/TLS handshake, causing the connection to fail.

2) **Defeats BGP Route Hijacking?**

A. HTTPS can prevent a router from being hijacked in a BGP hijacking attack
B. After a successful BGP hijacking attack, the attacker becomes on-path but cannot modify the transmitted messages because of HTTPS
C. After a successful BGP hijacking attack, the attacker cannot decrypt the transmitted messages or tamper the messages without being detected
D. After successfully hijacking traffic to google.com, the attacker can establish HTTPS connections with clients when impersonating google.com

- **A** is false because HTTPS operates at the application layer and does not prevent BGP hijacking, which occurs at the network layer. HTTPS cannot stop a router from being hijacked.
- **B** is false because while HTTPS encrypts messages, an on-path attacker could still modify the encrypted data (e.g., dropping packets) without decrypting it. However, they cannot tamper with the content without detection.
- **C** is true because HTTPS ensures that even if an attacker successfully hijacks traffic, they cannot decrypt the transmitted messages or tamper with them without being detected. This is due to the encryption and integrity checks provided by HTTPS.
- **D** is false because the attacker cannot impersonate a legitimate server like google.com without possessing the private key corresponding to the server's certificate. Without this, the HTTPS connection will fail.

Remarks:

- **BGP hijacking** allows an attacker to **reroute traffic** to a malicious network.
- However, **HTTPS encrypts all transmitted messages** end-to-end.
- The attacker can intercept traffic **but cannot decrypt or modify messages** without detection.
- If they attempt to impersonate the server, the HTTPS handshake **will fail** due to **invalid certificates**.

3) **Defeats TCP / IP attacks?**

A. IP packet sniffing attacks can be performed by off-path attackers and cannot be prevented by HTTPS
B. The attackers cannot impersonate the Google server in connection with a client using a TCP sequence number prediction attack because they do not have the certificate issued to google.com
C. The attackers cannot impersonate the Google server in connection with a client using a TCP sequence number prediction attack because they do not have the private key of google.com

- **A** is false because IP packet sniffing attacks cannot be performed by off-path attackers. Sniffing requires the attacker to be on-path or have access to the network traffic. HTTPS does not directly prevent sniffing but ensures that the sniffed data is encrypted and unreadable.
- **B** is false because the certificate alone is not sufficient for impersonation. An attacker would also need the private key associated with the certificate to establish a valid HTTPS connection.
- **C** is true because HTTPS relies on cryptographic keys for secure communication. Without the private key of google.com, an attacker cannot complete the SSL/TLS handshake, making it impossible to impersonate the Google server in a TCP sequence number prediction attack. This ensures the integrity and authenticity of the connection.

Remarks:

- TCP sequence number prediction attacks can allow attackers to **hijack** a session, but they **cannot bypass HTTPS encryption**.
- Even if the attacker predicts sequence numbers, they **cannot impersonate the real server** because they **do not have the private key of google.com**.
- Without the **private key**, the attacker **cannot complete the TLS handshake**, and the connection will fail.

Failures of HTTP Question

Question 1

QNS

Zoom allows joining a meeting with a hashed password specified in the URL. For example, the URL for one of the previous QA sessions on Feb 13th is https://nus-sg.zoom.us/j/83550259501?pwd=NjZiR0ZHZXlPL1RNVVB4TmJsVU4vdz09Links to an external site.. When you use the link above, which of the following can a network attacker who is eavesdropping on your Internet traffic learn?

- You are accessing zoom.us
- You are accessing the nus-sg sub-domain of zoom.us
- You are joining a meeting with ID 83550259501
- The hashed password for the meeting is NjZiR0ZHZXlPL1RNVVB4TmJsVU4vdz09

QNS

In the lecture, we have learned some practical pitfalls in HTTPS.
Assume that Bob has a website bob.com that supports both HTTP over port 80 and HTTPS over port 443. The website's login page can be accessed through either http://bob.com/login or https://bob.com/login. The submission link of the login form on the web page (that submits the username and password to the server) is always HTTPS (https://bob.com/auth?username=XXX&password=YYY) no matter whether the login page is accessed via HTTPS or not. The server will respond to such authentication requests with "Set-Cookie accesstoken=ZZZZ; Domain=bob.com" in the response header. The client will receive this temporary access token and attach it to all later requests to bob.com so that the user does not need to log in again for the next 3 days.
Which statement(s) is/are correct? Choose all that apply.

- When the user logs in using HTTP, an active network attacker named Mallory can perform a downgrade attack to steal the user's passwords.
- When the user logs in using HTTP, a passive network attacker named Eve can eavesdrop on the messages to get the temporary access token that can be used to impersonate the user for a short period.
- If Bob changes the response header to "Set-Cookie accesstoken=ZZZZ; Domain=bob.com; SecureLinks to an external site.", it is sufficient to stop both active network attackers and passive network attackers from impersonating a user logging in through HTTP.
- If Bob sets the HSTS Links to an external site. header with a period of 1 year (appending "Strict-Transport-Security: max-age=<expire-time>; includeSubDomains" to the response header of both HTTP and HTTPS), it is sufficient to stop both active network attackers like Mallory and passive network attackers like Eve from getting the password or the access token of a user logging in through HTTP for the first time.

ANS

- ✅ **A. If** the user accesses the login page over HTTP, Mallory (an active attacker) can modify the page to change the form submission URL from HTTPS to HTTP, intercepting the credentials. Even though the form submission is hardcoded to HTTPS, Mallory could manipulate the page before it loads (e.g., via a MITM attack on HTTP) to submit to HTTP instead.
  - Additionally, if the user is tricked into accessing the HTTP version of the login page, Mallory could intercept the credentials even if the form submits to HTTPS (e.g., by injecting malicious JavaScript to capture the credentials before submission).
- ✅ **B.** If the login page is accessed over HTTP, the authentication request is sent over HTTPS (as described), but the server's response (including the Set-Cookie header with the accesstoken) is sent over HTTPS. However, if the cookie is not marked as Secure, the browser may later send it over HTTP in subsequent requests, allowing Eve to eavesdrop on it.
  - Even if the cookie is marked as Secure, if the user initially accesses the login page over HTTP, Eve could still see the login page request (though not the token itself), which might leak other sensitive info.
- ❌ **C.** While the Secure flag ensures the cookie is only sent over HTTPS, it does not prevent:
  - An active attacker (Mallory) from intercepting the password if the login page is loaded over HTTP (e.g., by modifying the form submission).
  - A passive attacker (Eve) from seeing the password if the form is submitted over HTTP (though the question states the form submits over HTTPS, Mallory could still manipulate the page to submit over HTTP).
  - The Secure flag only protects the cookie, not the initial password submission or the login page itself.
- ❌ **D.** HSTS headers are **ignored** when sent over HTTP (they are only honored if sent over HTTPS). If the user initially visits http://bob.com/login, the HSTS header sent over HTTP has no effect, and Mallory/Eve can still attack the connection.
  - HSTS only works after the user has visited the site over HTTPS at least once (or the domain is preloaded in browsers). For a first-time visit over HTTP, HSTS provides no protection.

QNS

Wendy visits the public wifi network at an airport. She connects to https://free-air-wifi.com and encounters an HTTPS certificate error. Upon finding out that the certificate is self-signed by the airport authorities (call them airport.com), she decides to accept the issuer as a root CA permanently, because she urgently wants to connect to her work email.
Which of the following attacks become(s) possible? Please check all that apply. Besides the added root CA, assume that all other necessary assumptions in the HTTPS threat model hold.

a) Airport.com, if malicious, can decrypt Wendy's web session with https://gmail.com while she is on the airport wifi.
b) ASes (or ISPs) routing Wendy's traffic from the airport to https://gmail.com can decrypt her session without colluding with airport.com while she is on the airport wifi.
c) After Wendy leaves the airport, an AS (or ISP) routing Wendy's traffic can decrypt Wendy's web session with https://gmail.com if they collude with airport.com.
d) A malicious website https://evil.com, which doesn't collude with airport.com, can now intercept the user's visits to https://gmail.com

ANS

- ✅ **A.**The airport can issue a fake gmail.com certificate and perform a **MITM attack** in real time, decrypting all traffic.
  - By accepting the self-signed certificate from airport.com as a root CA, Wendy has effectively trusted airport.com to issue certificates for any domain, including gmail.com.
  - If airport.com is malicious, it can perform a man-in-the-middle (MITM) attack and decrypt her session with gmail.com while she is connected to the airport wifi.
  - Forward secrecy does **not** prevent this because the attack is active (not retroactive).
- ❌ **B.** ASes/ISPs **cannot** issue fake certificates unless they collude with airport.com (they don't have its private key).
- ❌/✅ **C:**
  - **Incorrect (in most cases)**:
    - ➢ If the session uses **forward-secure ciphers (DHE/ECDHE)**, even a colluding AS cannot decrypt past sessions.
    - ➢ If the session uses **non-forward-secure ciphers (e.g., RSA key exchange)** and the rogue CA also knows the server's private key (unlikely for gmail.com), then retroactive decryption is possible.
    - ➢ However, the question states that **all other HTTPS threat model assumptions hold**, which implies **forward secrecy is used** (modern browsers enforce this). Thus, this attack **fails**.
  - **For this case, CORRECT**
    - ➢ If airport.com colludes with an AS or ISP, it can issue a fraudulent certificate for gmail.com, allowing the AS or ISP to decrypt Wendy's session even after she leaves the airport.
- ❌ **D.** A malicious website like evil.com cannot intercept Wendy's visits to gmail.com unless it colludes with airport.com.
  - evil.com cannot issue fake certificates for gmail.com without airport.com's private key.

QNS

A content delivery network (CDNLinks to an external site.) refers to a geographically distributed group of servers that work together to provide fast delivery of Internet content and reduce the workload of the content providers (web domains).
Assume that you have a website with an HTTPS certificate. You want to use CDN for your HTTPS website. Which of the following about how the CDN caches resources for your website is true?

A) Due to the fact that the session key (symmetric key) of each SSL/TLS connection is different, the CDN needs to send different ciphertexts to different users if using HTTPS.
B) CDN can still send cached resources through SSL/TLS connection to the clients on behalf of your website (using your domain name) without knowing the private key corresponding to your domain name's certificate or communicating with your website.

ANS

A) True. different session keys will cause the cached data to be encrypted with different session keys for each user, hence the resulting ciphertext is different
B) False. CDN cant sned through SSL/TLS, since it doesnt have the private key to derive the session keys used to send the message via HTTPS

QNS & ANS

A network attacker Eve is eavesdropping on an HTTPS connection between a user Bob and YouTube. He finds that the patterns of the streaming data packets (frequencies, sizes, and duration of the streaming) are very similar to an online lecture on computer security that he's watching. Eve concludes that Bob is watching the same lecture video as him, without decrypting the network traffic. Why is Eve able to learn this information?

- Because the communication channel (HTTPS) does not provide integrity
- Because of side-channel information leakage from the encrypted HTTPS traffic
- Because even with HTTPS, Eve can know the requested URL of the video that Bob is watching