CSE 484 and CSE M 584 (Winter 2009)

Web Security
Symmetric Encryption &
Authentication

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

JavaScript

- Language executed by browser
  - Can run before HTML is loaded, before page is viewed, while it is being viewed or when leaving the page
- Often used to exploit other vulnerabilities
  - Attacker gets to execute some code on user's machine
  - Cross-scripting: attacker inserts malicious JavaScript into a Web page or HTML email; when script is executed, it steals user's cookies and hands them over to attacker's site

Scripting

```
<script type="text/javascript">
    function whichButton(event) {
            if (event.button==1) {
                    alert("You clicked the left mouse button!") }
            else {
                    alert("You clicked the right mouse button!")
            }}
</script>
…
<body onMouseDown="whichButton(event)">
…
</body>
```

Script defines a page-specific function

Function gets executed when some event happens (onLoad, onKeyPress, onMouseMove…)

JavaScript Security Model

- Script runs in a "sandbox"
  - Not allowed to access files or talk to the network
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

Risks of Poorly Written Scripts

- For example, echo user’s input

http://naive.com/search.php?term=“Britney Spears”

search.php responds with

<html> <title>Search results</title>

<body>You have searched for <?php echo $_GET[term] ?>… </body>

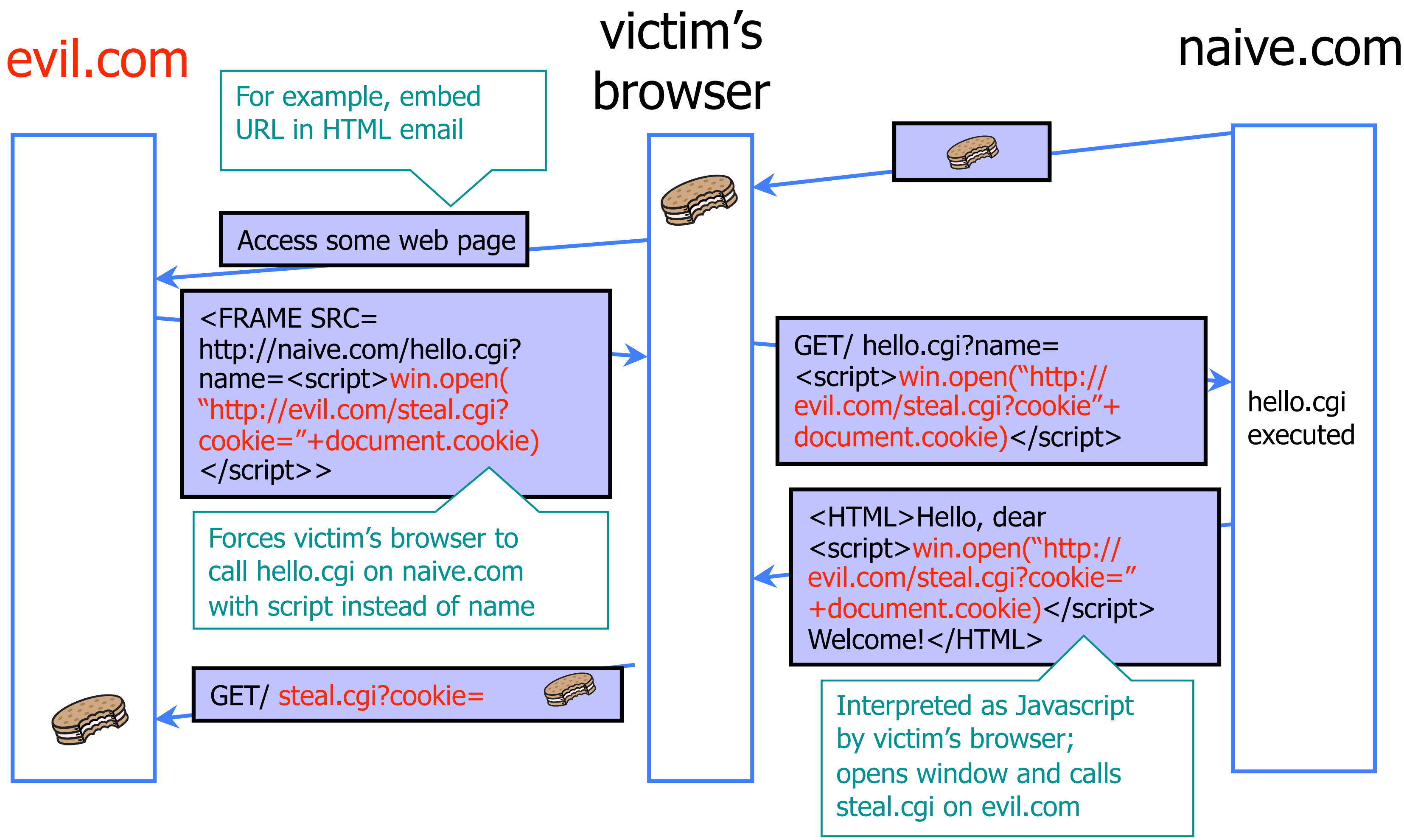
Or

GET/ hello.cgi?name=Bob

hello.cgi responds with

<html>Welcome, dear Bob</html>

Stealing Cookies by Cross Scripting



Inadequate Input Validation

- http://victim.com/copy.php?name=username
- copy.php includes

  Supplied by the user!

  system("cp temp.dat $name.dat")
- User calls

  http://victim.com/copy.php?name="a; rm *"
- copy.php executes

  system("cp temp.dat a; rm *");

URL Redirection

- http://victim.com/cgi-bin/loadpage.cgi?page=url
  - Redirects browser to url
  - Commonly used for tracking user clicks; referrals
- Phishing website puts

  http://victim.com/

  cgi-bin/loadpage.cgi?page=phish.com
- Everything looks Ok (the link is indeed pointing to victim.com), but user ends up on phishing site!

User Data in SQL Queries

- set UserFound=execute(

  SELECT * FROM UserTable WHERE

  username=′ ″ & form(“user”) & “ ′ AND

  password=′ ″ & form(“pwd”) & “ ′ ″);

  - User supplies username and password, this SQL query checks if user/password combination is in the database

- If not UserFound.EOF

  Authentication correct

  else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

SQL Injection

Always true!

- User gives username ′ OR 1=1 --
- Web server executes query

```
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username=′ ′ OR 1=1 -- ... );
```

Everything after -- is ignored!

- This returns the entire database!
- UserFound.EOF is always false; authentication is always "correct"

It Gets Better

- User gives username

  ’ exec cmdshell ’net user badguy badpwd’ / ADD --

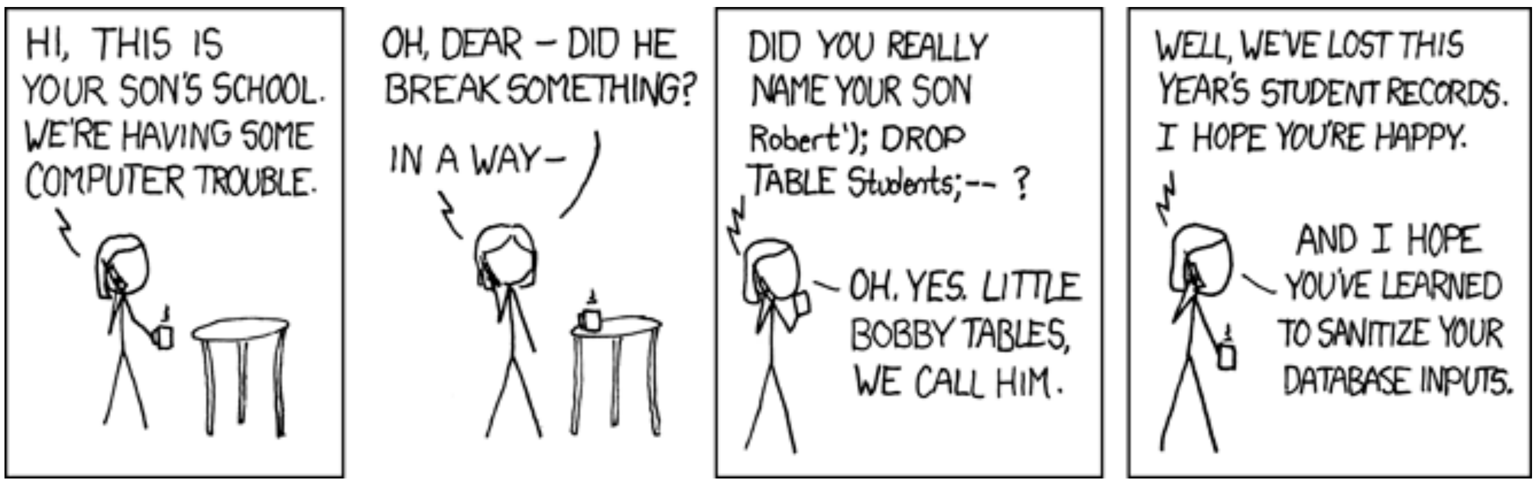
- Web server executes query

  set UserFound=execute(

  SELECT * FROM UserTable WHERE

  username=’ ’ exec … -- …);

- Creates an account for badguy on DB server



http://xkcd.com/327/

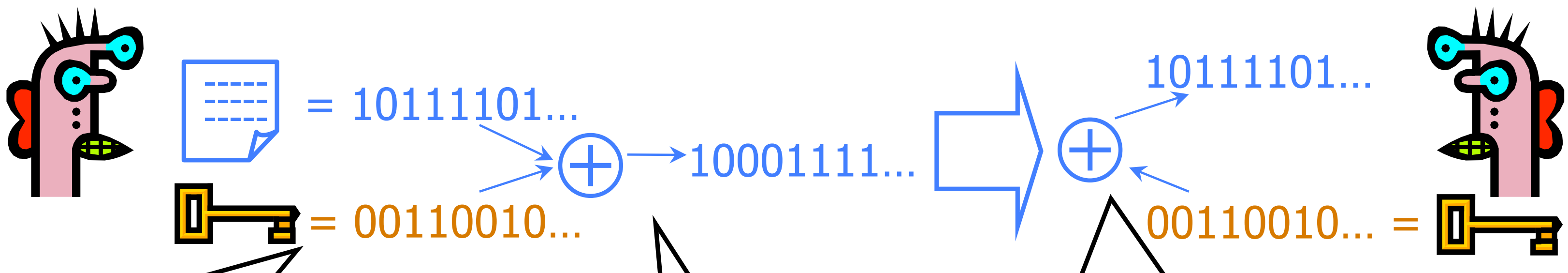
Other concerns

- Cross-site request forgery
- DNS rebinding
- …

Dangerous Websites

- Recent "Web patrol" study at Microsoft identified 752 unique URLs that could successfully exploit unpatched Windows XP machines
  - Many are interlinked by redirection and controlled by the same major players
- "But I never visit risky websites"
  - 11 exploit pages are among the top 10,000 most visited
  - Common trick: put up a page with popular content, get into search engines, page redirects to the exploit site
    - One of the malicious sites was providing exploits to 75 "innocuous" sites focusing on (1) celebrities, (2) song lyrics, (3) wallpapers, (4) video game cheats, and (5) wrestling
- Similar study at UW; Now through emails and ads

One-Time Pad



Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

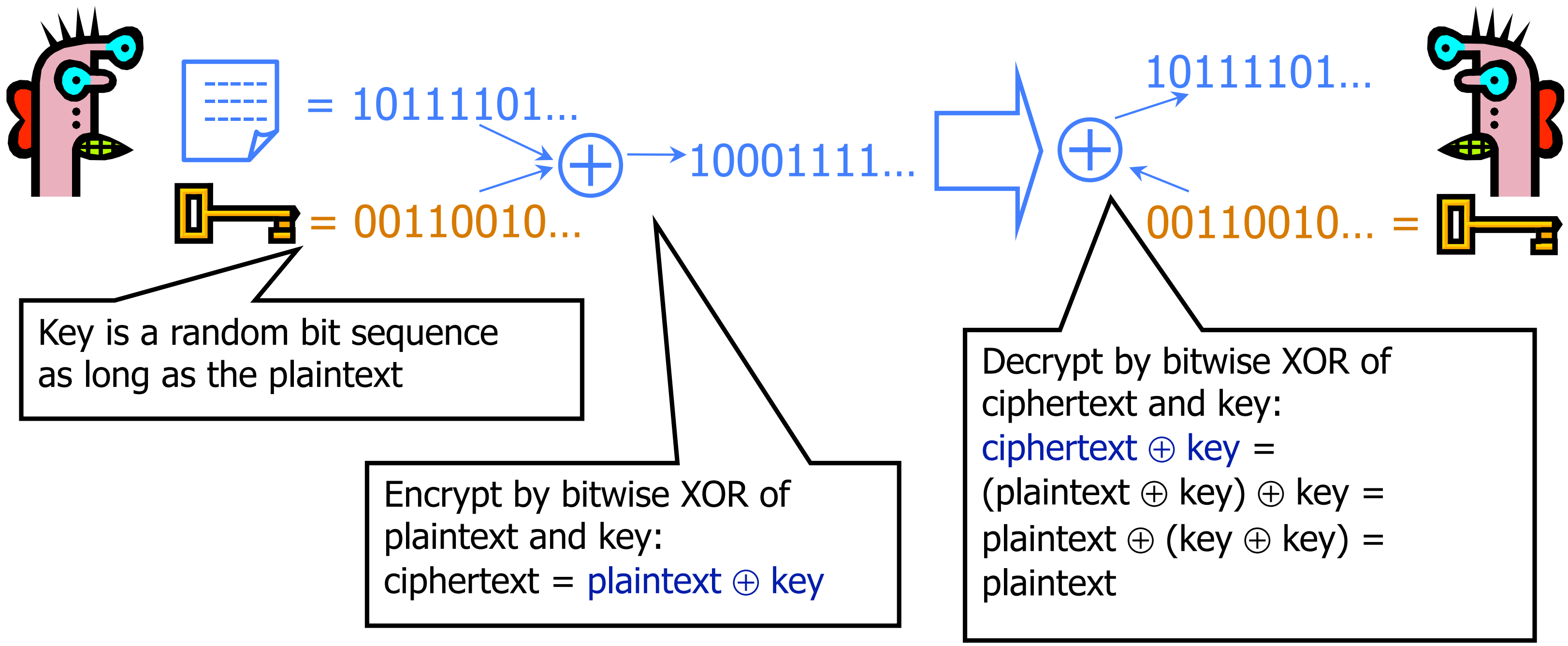
Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Cipher achieves perfect secrecy if and only if
there are as many possible keys as possible plaintexts, and
every key is equally likely (Claude Shannon)

Advantages of One-Time Pad

- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - …as long as the key sequence is truly random
    - True randomness is expensive to obtain in large quantities
  - …as long as each key is same length as plaintext
    - But how does the sender communicate the key to receiver?

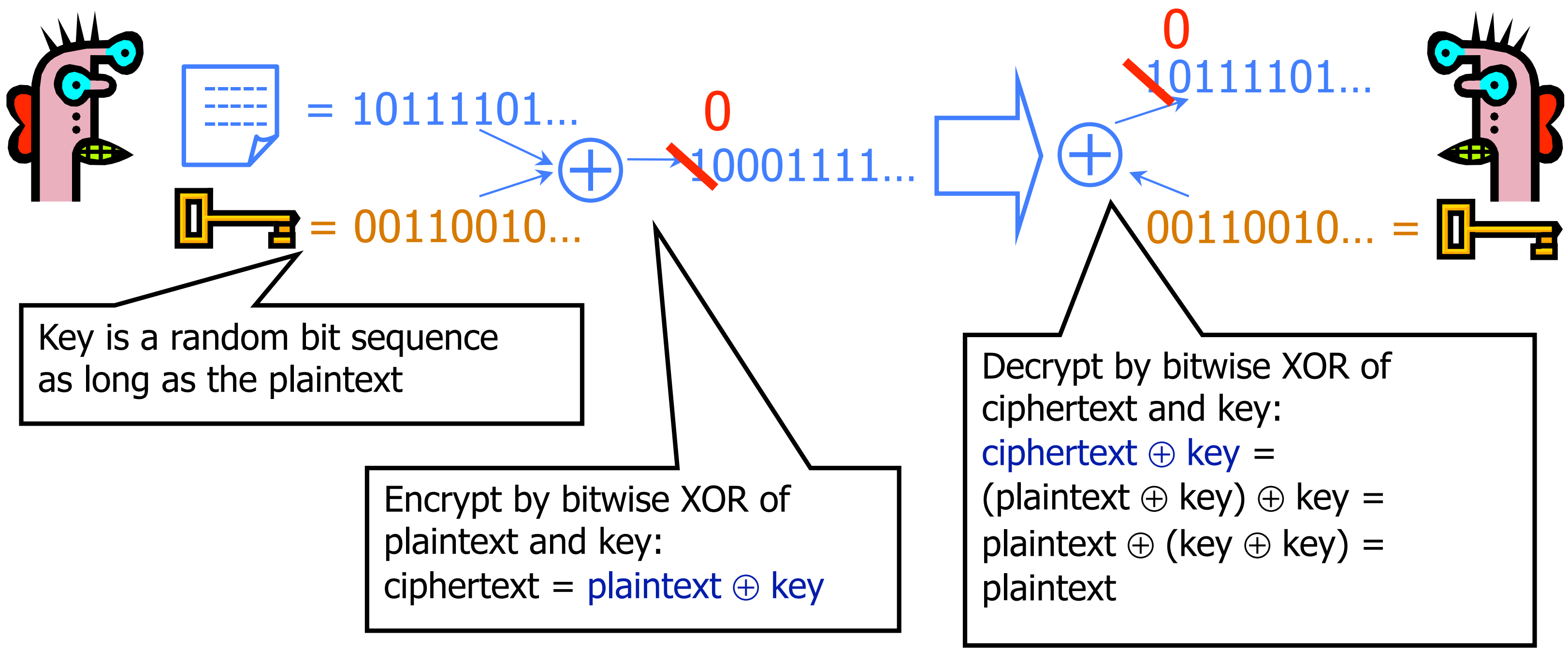
Disadvantages



Disadvantage #1: Keys as long as messages.

- Impractical in most scenarios
- Still used by intelligence communities

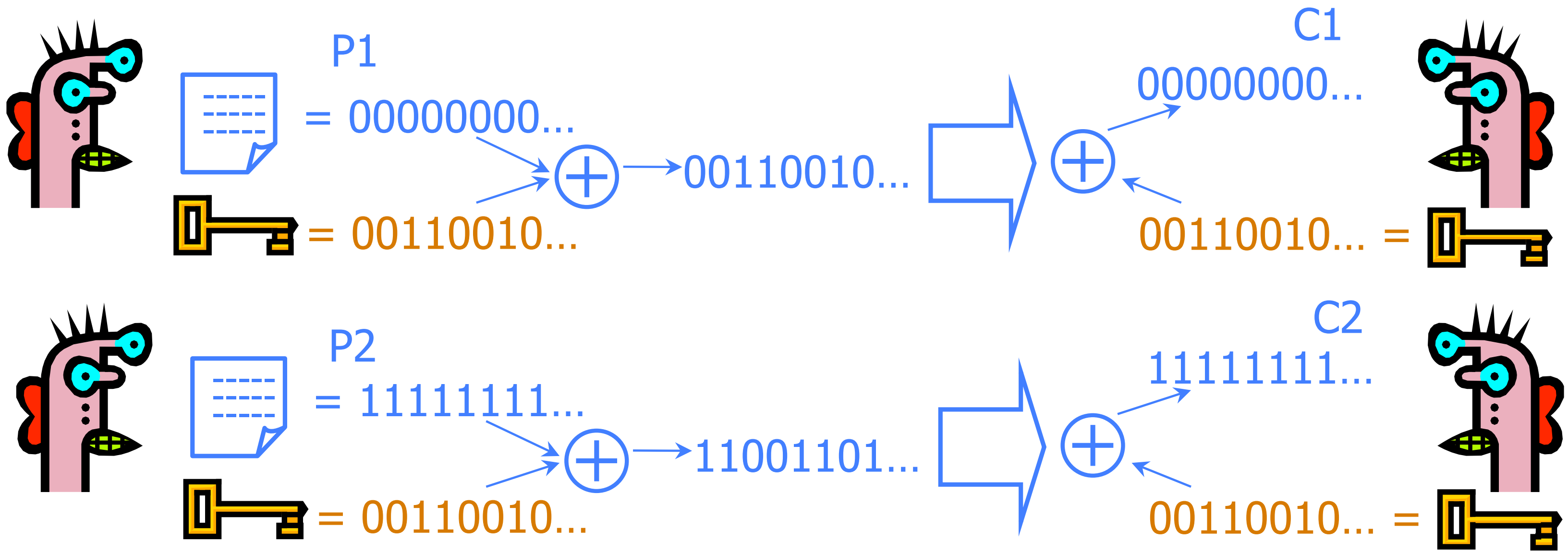
Disadvantages



Disadvantage #2: No integrity protection

Disadvantages

Disadvantage #3: Keys cannot be reused



Learn relationship between plaintexts:

C1⊕C2 = (P1⊕K)⊕(P2⊕K) = (P1⊕P2)⊕(K⊕K) = P1⊕P2

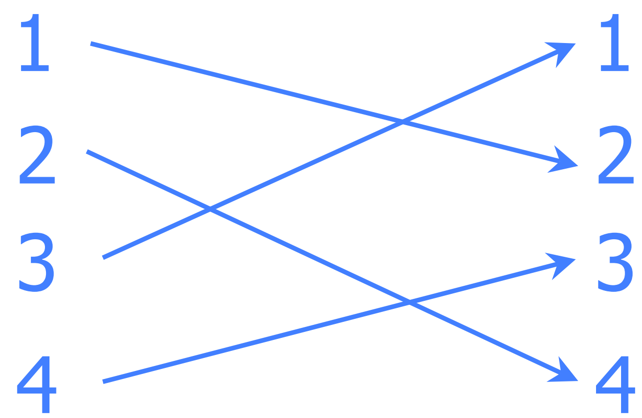
Reducing Keysize

- What do we do when we can’t pre-share huge keys?
  - When OTP is unrealistic

- We use special cryptographic primitives
  - Single key can be reused (with some restrictions)
  - But no longer provable secure (in the sense of the OTP)

- Examples: Block ciphers, stream ciphers

Background: Permutation

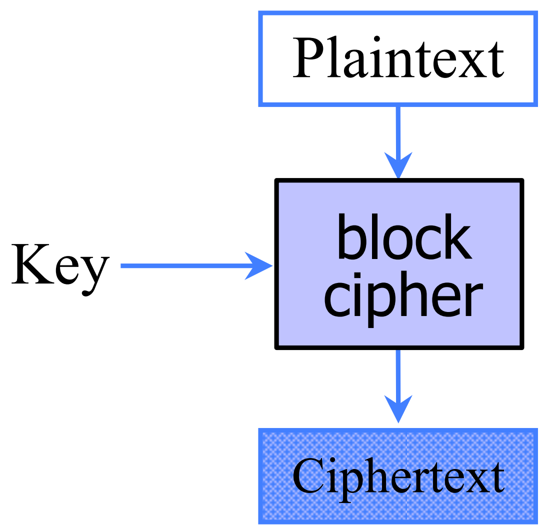


CODE becomes DCEO

- For N-bit input, N! possible permutations
- Idea: split plaintext into blocks, for each block use secret key to pick a permutation, rinse and repeat
  - Without the key, permutation should "look random"

Block Ciphers

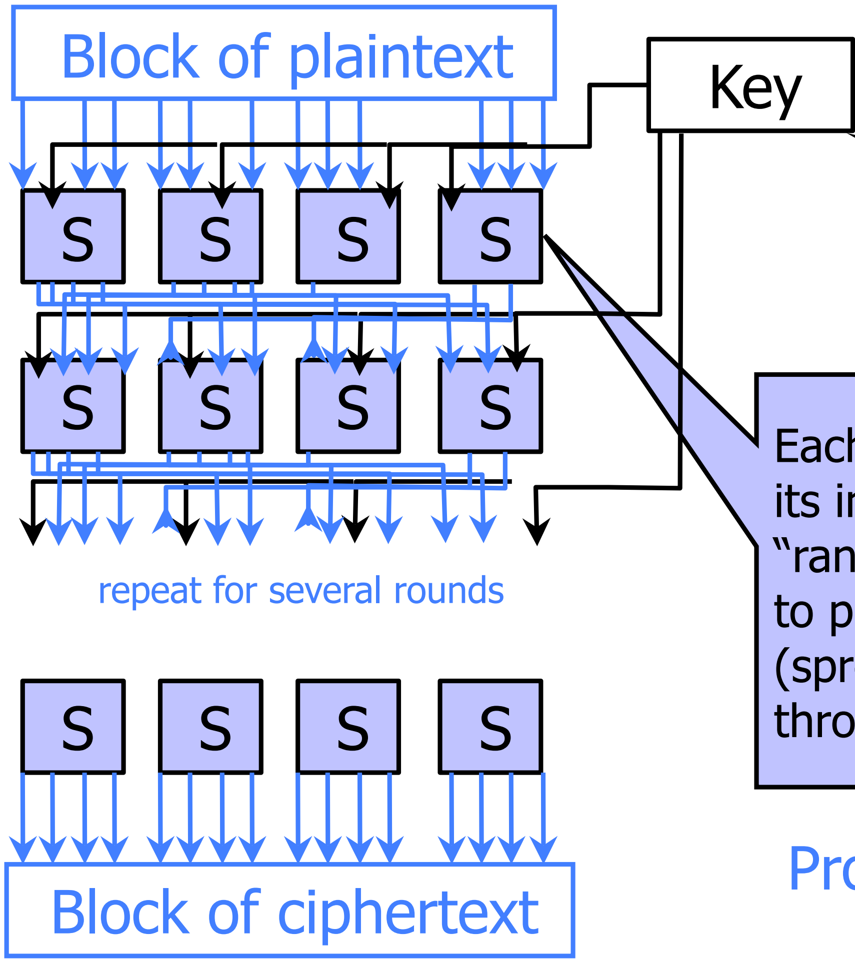
- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Same key is reused for each block (can use short keys)



Block Cipher Security

- Result should look like a random permutation
  - "As if" plaintext bits were randomly shuffled
- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

Block Cipher Operation (Simplified)

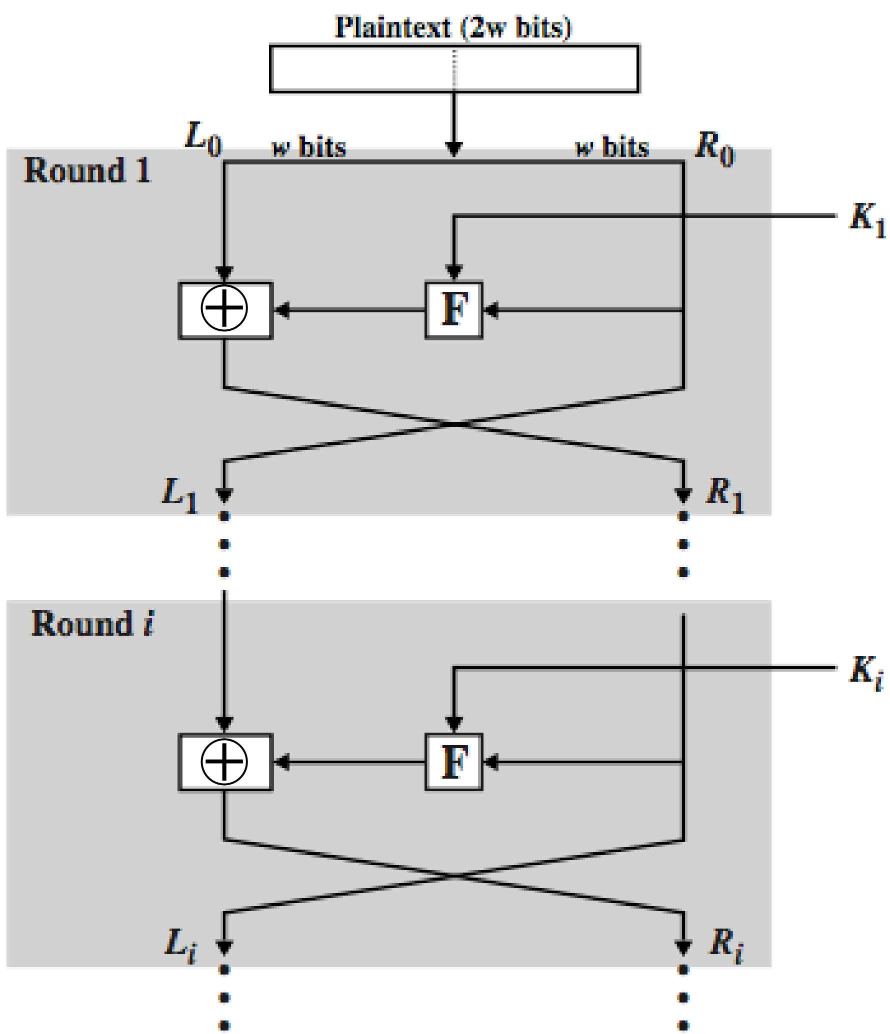


Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

Procedure must be reversible (for decryption)

Feistel Structure (Stallings Fig 2.2)



DES

- Feistel structure
  - "Ladder" structure: split input in half, put one half through the round and XOR with the other half
  - After 3 random rounds, ciphertext indistinguishable from a random permutation if internal F function is a pseudorandom function (Luby & Rackoff)
- DES: Data Encryption Standard
  - Feistel structure
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

DES and 56 bit keys (Stallings Tab 2.2)

- 56 bit keys are quite short

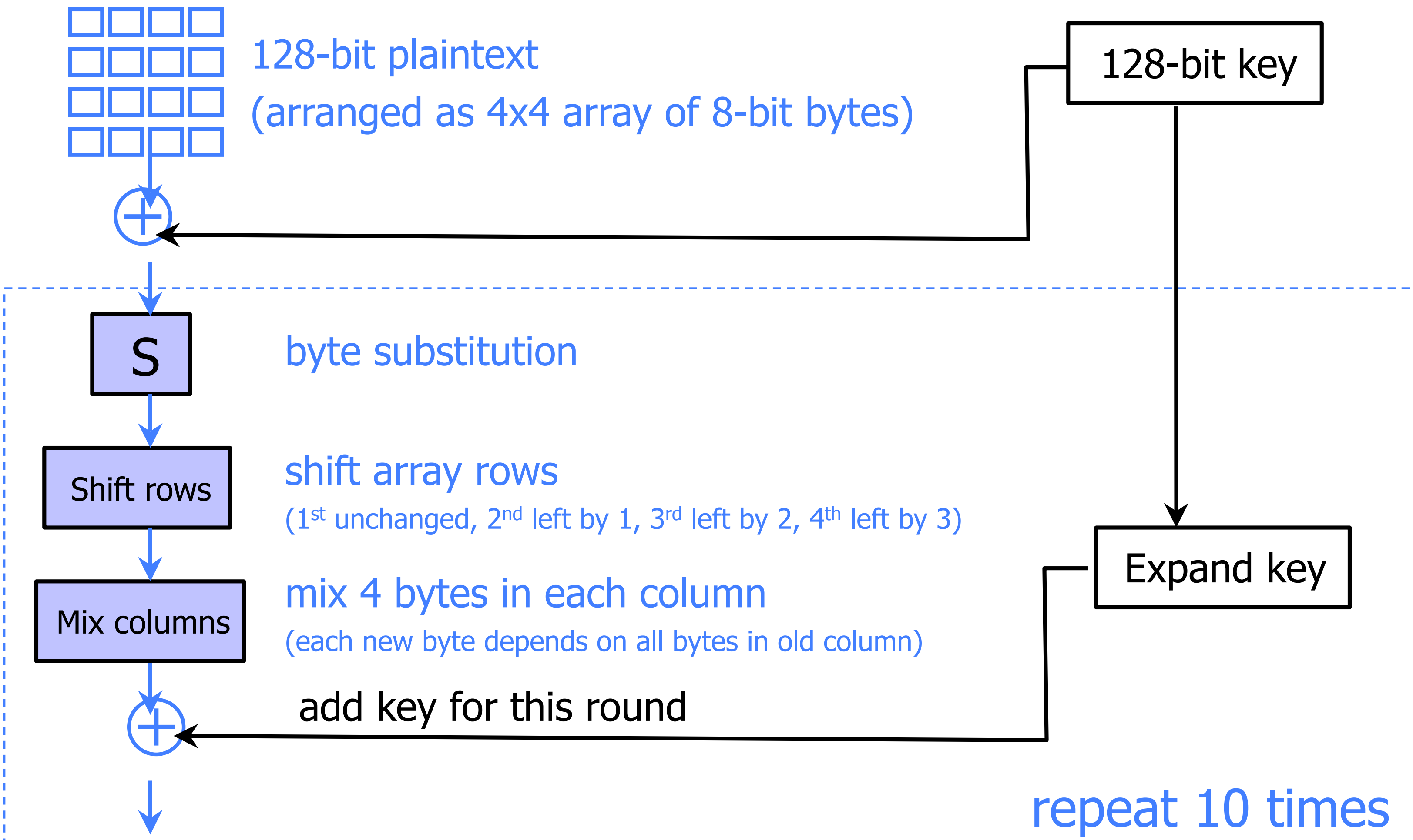
Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- 1999: EFF DES Crack + distributed machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

Advanced Encryption Standard (AES)

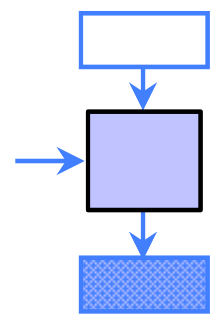
- New federal standard as of 2001
- Based on the Rijndael algorithm
- 128-bit blocks, keys can be 128, 192 or 256 bits
- Unlike DES, does <u>not</u> use Feistel structure
  - The entire block is processed during each round
- Design uses some very nice mathematics

Basic Structure of Rijndael



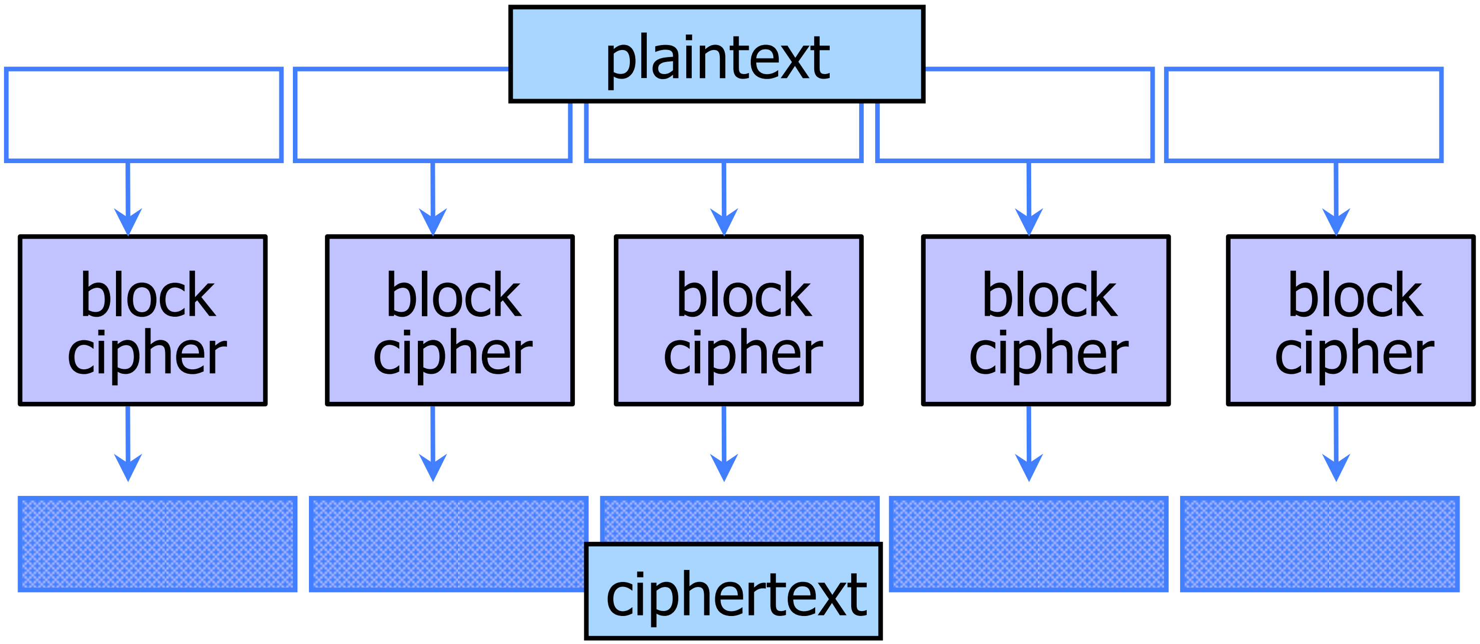
Encrypting a Large Message

- So, we've got a good block cipher, but our plaintext is larger than 128-bit block size
- Electronic Code Book (ECB) mode
  - Split plaintext into blocks, encrypt each one separately using the block cipher



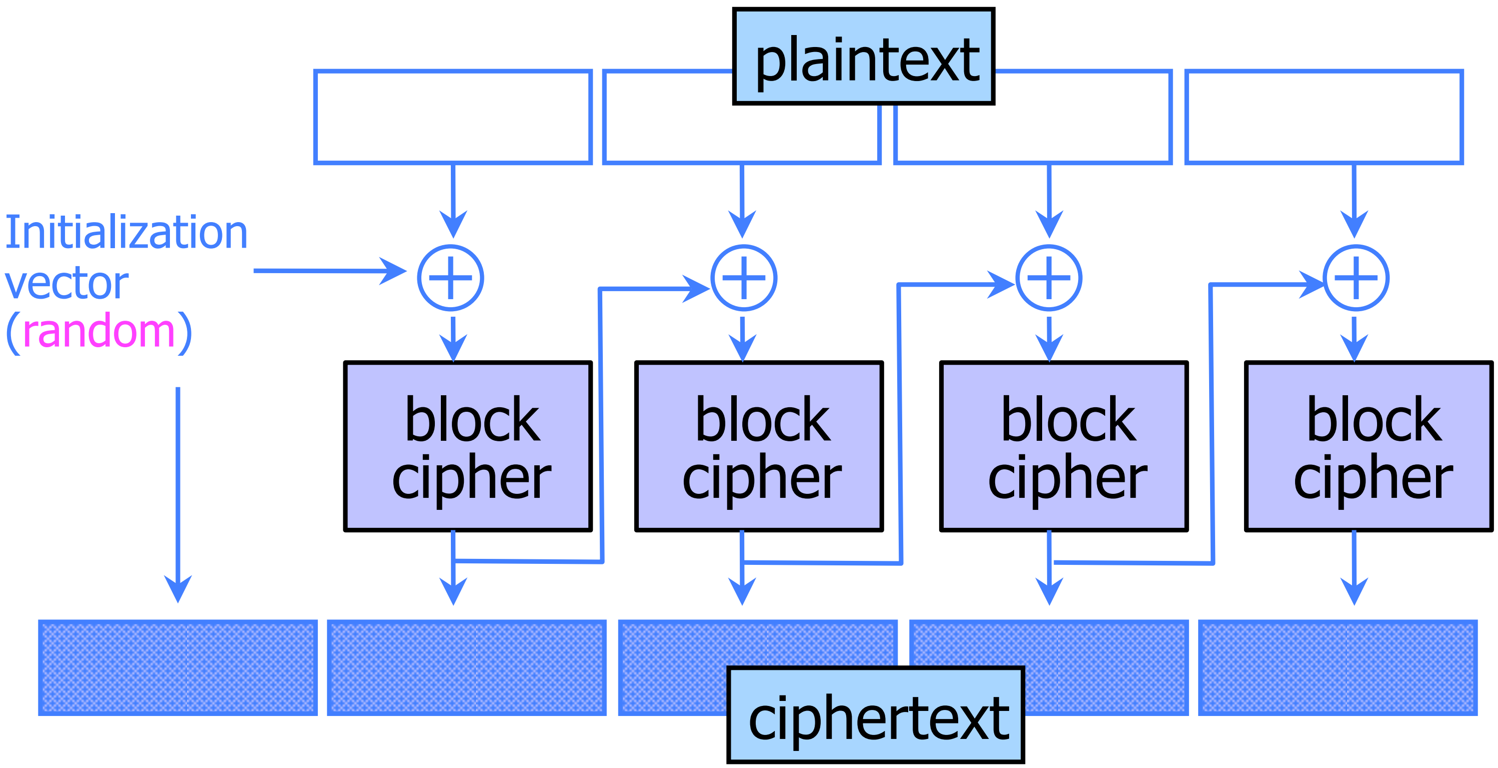
- Cipher Block Chaining (CBC) mode
  - Split plaintext into blocks, XOR each block with the result of encrypting previous blocks
- Counter (CTR) mode
  - Use block cipher to generate keystream, like a stream cipher
- …

ECB Mode



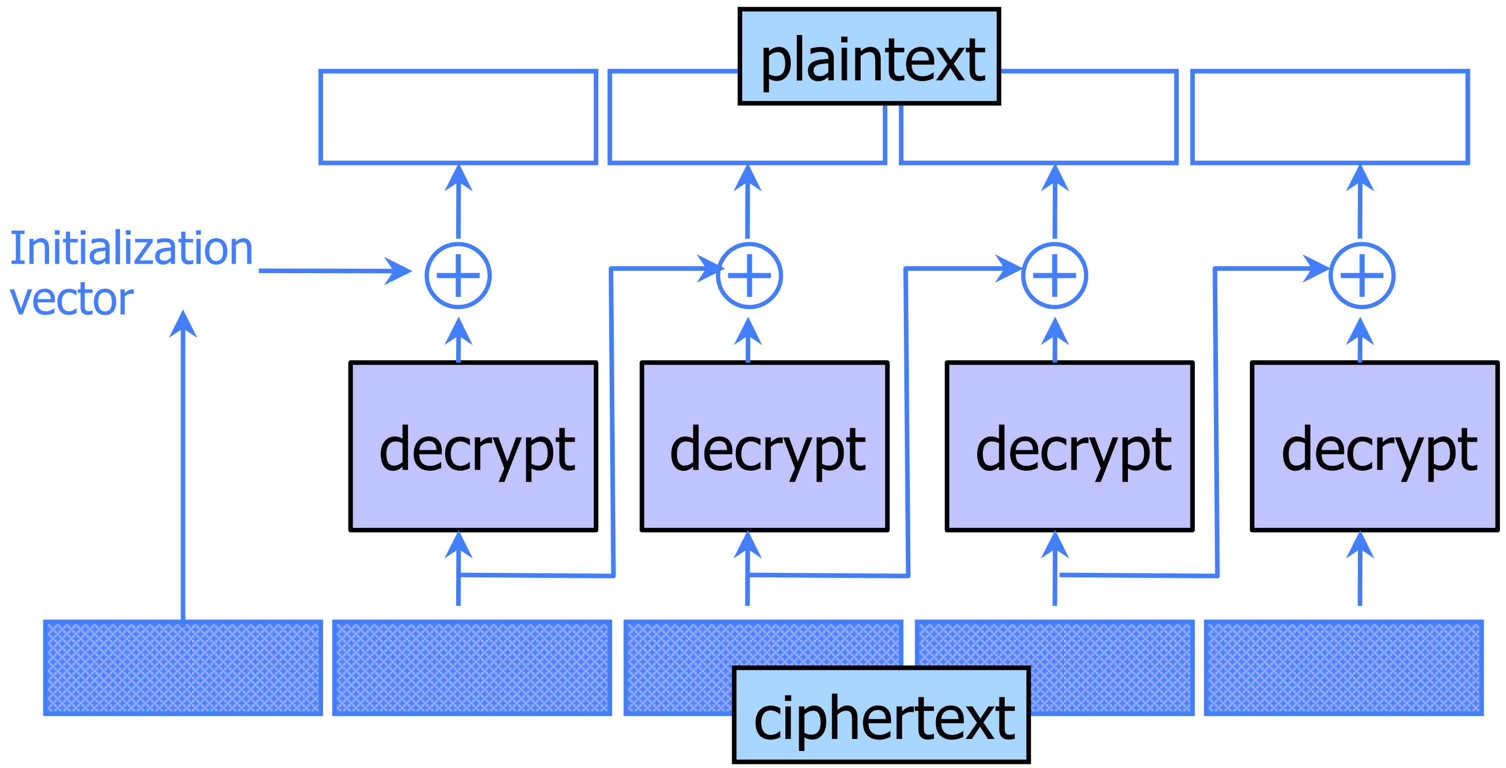
- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

CBC Mode: Encryption

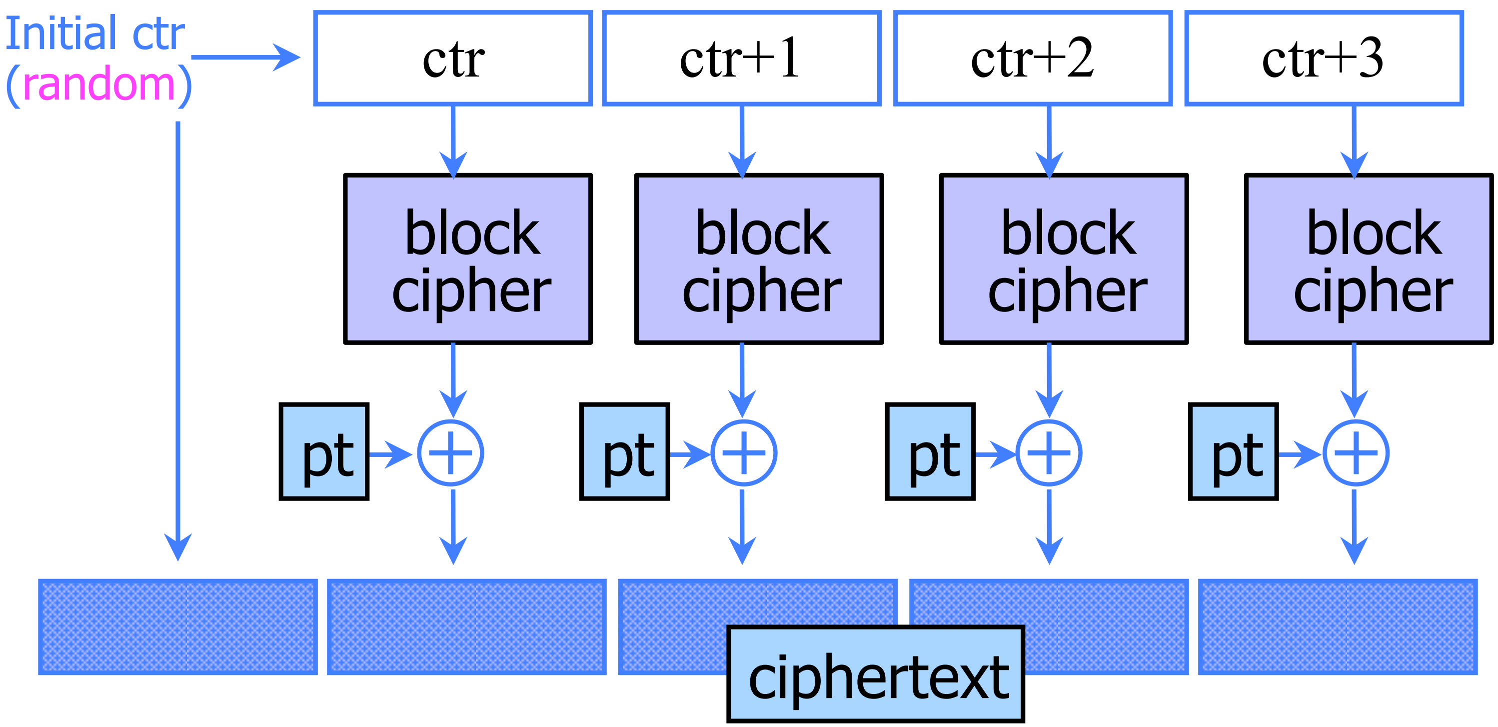


- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
    - Still does not guarantee integrity

CBC Mode: Decryption



CTR Mode: Encryption



- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity

CTR Mode: Decryption