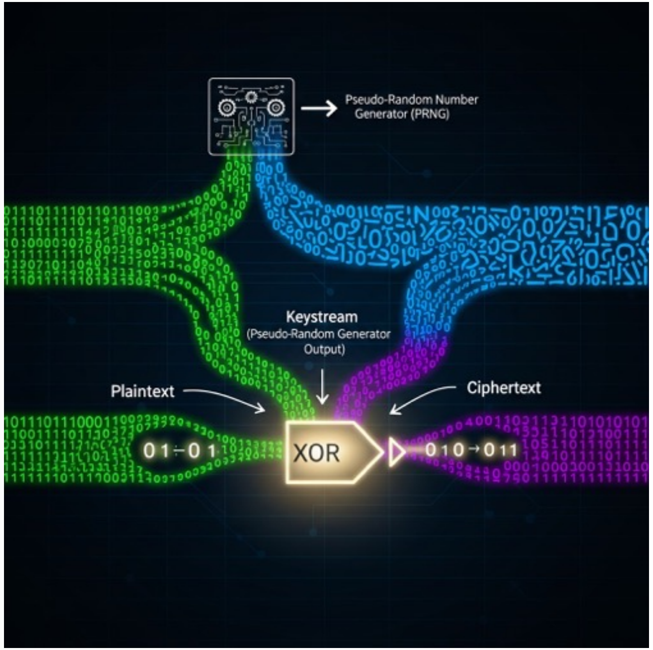
# Stream ciphers

old, but now back in the spotlight



# Secret key cryptography

Alice and Bob share

- A crypto protocol E
- A secret (symmetric) key K
- They communicate using E with key K
- Adversary knows E, knows some exchanged messages but ignores K

Two approaches:

- Stream Ciphers
- Block ciphers

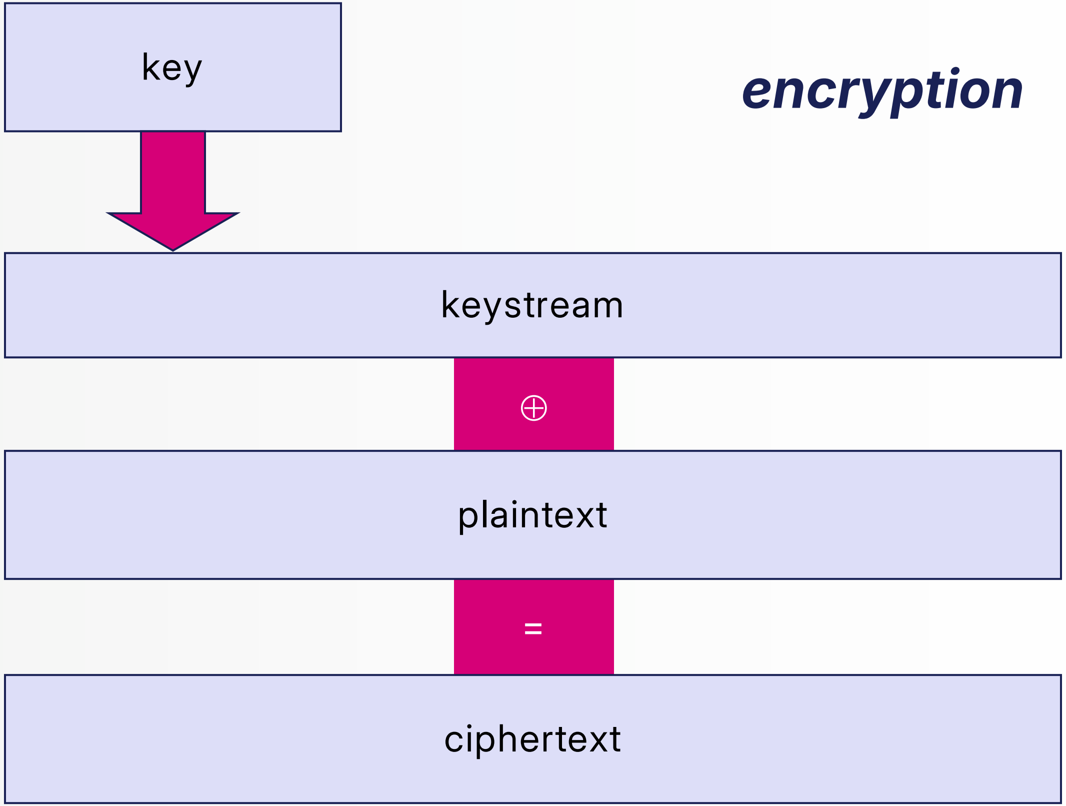
# Stream ciphers

- define a secret key (seed)
- using the seed generate a byte stream (*keystream*): *i*-th byte is function of
  - only key (*synchronous* stream cipher), or
  - both key and first *i*-1 bytes of ciphertext *asynchronous* stream cipher)
- obtain ciphertext by bitwise XORing plaintext and keystream

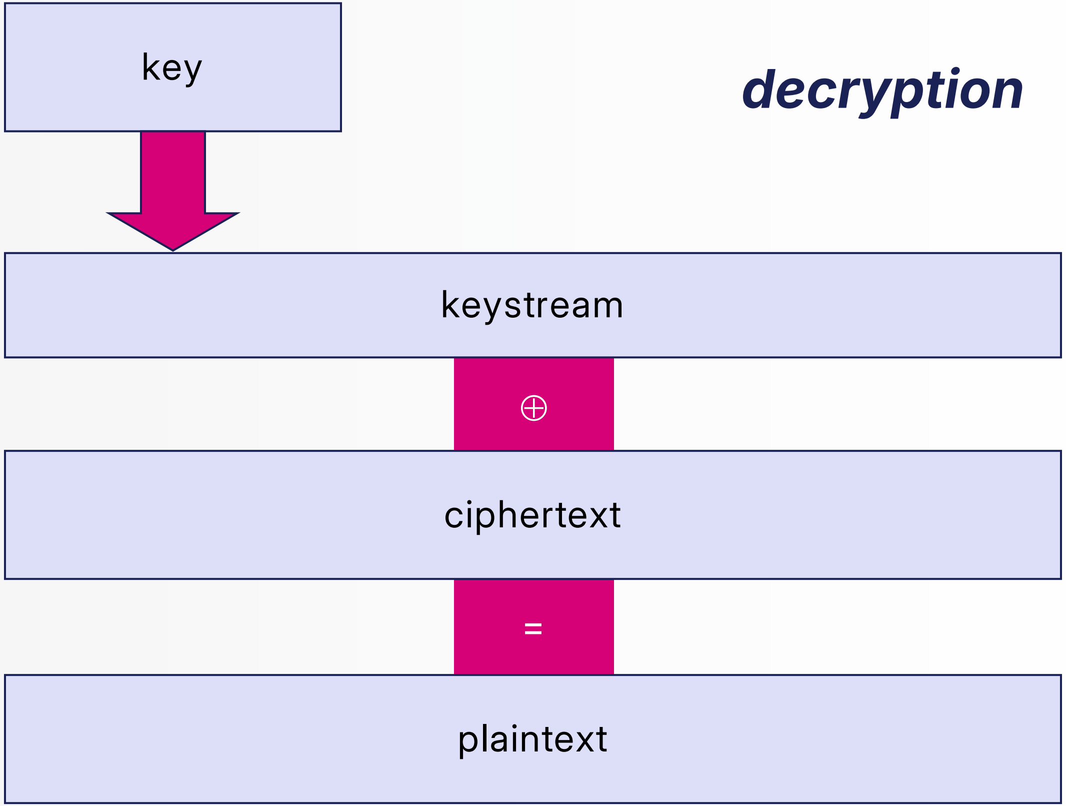
| x | y | x ⊕ y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Synchronous stream cipher

*encryption*



# Synchronous stream cipher



# Synchronous stream ciphers in practice

- Many ciphers before 1940
- Enigma - II world war (Germany)
- A5 – GSM (encryption cell phone-base station)
- WEP - used in Ethernet 802.11 (wireless)
- RC-4  (Ron’s Code, used by WEP)

# A5/1 stream cipher

- Developed in 1987, used to ensure over-the-air communication privacy in the GSM cellular telephone standard
- A major dispute arose in the mid-1980s between NATO signal intelligence agencies about whether GSM encryption should be strong or intentionally weakened
- Used in both Europe and the United States. A related variant, A5/2, was a deliberately weakened version intended for export to certain regions
- The design was initially kept secret, but leaked in 1994 and fully reverse-engineered in 1999 by Marc Briceno
- Multiple serious vulnerabilities in the cipher were subsequently identified

Rivest

# RC-4

- RC: Ron’s Code
  - (Ron = Ronald Rivest, MIT, born in 1947 in NY state)
- Considered safe: 1987 - 1994 kept secret, after ‘94 extensively studied
  - insecure from 2004
- Good for exporting (complying with US restrictions)
- Easy to program, fast
- Very popular: Lotus Notes, SSL, Wep etc.
- RC4's weak key schedule can give rise to a variety of serious problems

# RC-4 weaknesses

- Biased output in early keystream bytes
- Vulnerable if keys are reused
- Broken in protocols like WEP and TLS (early versions)

- Today deprecated

# RC-4 properties

- variable key length (byte)
- synchronous
- starting from the key, it generates an apparently random permutation
- eventually the sequence will repeat
- however, long period > $10^{100}$
- very fast: 1 byte of output requires 8-16 instructions

# RC-4 initialization

Goal: generate a (pseudo)random permutation of the first 256 natural numbers

1. $j=0$
2. $S_0=0$, $S_1=1$, … , $S_{255}=255$
3. Assume a key of 256 bytes $k_0, \dots, k_{255}$ (if the key is shorter, repeat)
4. for $i=0$ to 255 do
    1. $j = (j + S_i + k_i) \bmod 256$
    2. exchange $S_i$ and $S_j$

In this way we obtain a permutation of 0, 1, … ,255, the resulting permutation is a function of the key

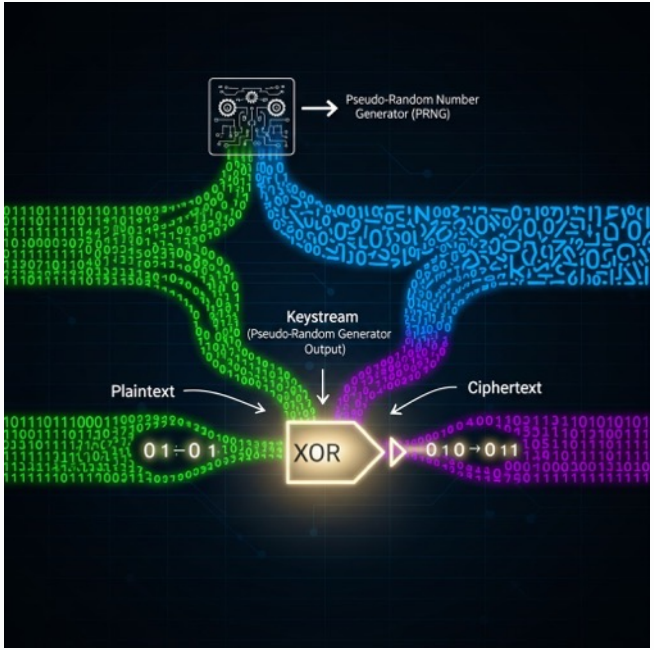
# RC-4 keystream generation

Input: permutation S of 0, 1, … , 255

1. $i = 0, j = 0$
2. while (true)
    1. $i = (i + 1) \bmod 256$
    2. $j = (j + S_i) \bmod 256$
    3. exchange $S_i$ and $S_j$
    4. $t = (S_i + S_j) \bmod 256$
    5. $k = S_t$ // compute XOR

at every iteration compute the XOR between k and next byte of plaintext (or ciphertext)

# Perfect ciphers and One-Time Pad



# Perfect ciphers

Plaintext space = $\{0, 1\}^n$, $D$ known

Given a ciphertext $C$ the probability that exists $k2$ such that $D_{k2}(C) = P$ for any plaintext $P$ is equal to the apriori probability that $P$ is the plaintext

In other words: *the ciphertext does not reveal any information on the plaintext*

Pr[plaintext = $P$ | ciphertext = $C$] = Pr[plaintext = $P$]

in short:
$\Pr[P|C] = \Pr[P]$

Probabilities are over the key space and the plaintext space

# Conditional probabilities

$Pr[P|C] = Pr[P \wedge C] / Pr[C]$
(def. of cond. pr.)

$Pr[P|C] * Pr[C] =$
$Pr[C|P] * Pr[P]$
(Th. Bayes)

In a perfect cipher
$Pr[P|C] = Pr[P]$

by replacing into Bayes
$Pr[P] * Pr[C] = Pr[C|P] * Pr[P]$

$Pr[C] = Pr[C \mid P]$

# One-time pad (OTP)

AKA Vernam Cipher, invented in 1917 and patented in 1919 while *Gilbert Vernam* was working at AT&T

Plaintext space: $\{0,1\}^n$

Keystream space: $\{0,1\}^n$

The scheme is symmetric; *keystream K is chosen at random*

$E_K(P) = C = P \oplus K$

$D_K(C) = C \oplus K = P \oplus K \oplus K = P$

# Features of OTP

Claim: one-time pad is a perfect cipher. [given a *k* bit cipher text every *k*-bit plain text has got same probability if key is random]

Problem: size of keystream space, as show by the following

**Theorem** (Shannon): A cipher cannot be perfect if the size of its key space is less than the size of its message space

# Proof of Shannon's theorem

By contradiction

Assume #keys (l) < #messages (n) and consider ciphertext $C_0$ s.t. $Pr[C_0] > 0$ ($C_0$ must exist!)

For some key K, consider $P = D_K(C_0)$. There exist at most l (#keys) such messages (one per each key)

Choose message $P_0$ s.t. it is not of the form $D_K(C_0)$ (there exist n-l such messages)

Hence $Pr[C_0|P_0]=0$

But in a perfect cipher $Pr[C_0|P_0]=Pr[C_0] > 0$. Contradiction

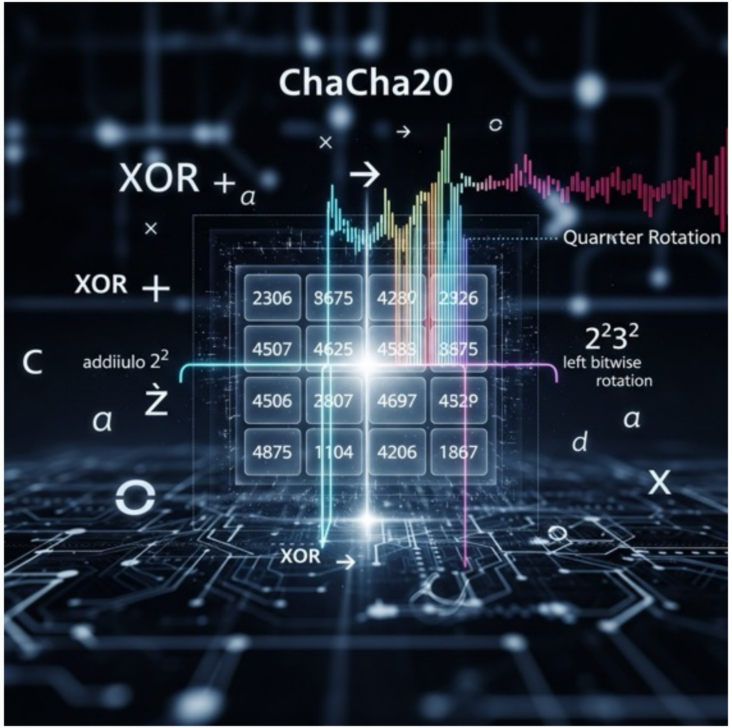
# Consequences

- OTP is perfect if
  - keystream truly random
  - keystream space not less message space
- if seed (key) is reused we get same keystream K for two different encryptions. Heavy consequences
  - $C1 = P1 \oplus K$, $C2 = P2 \oplus K$
  - Assume pair plaintext, ciphertext (P1, C1) known (KPA)
  - Attacker computes $C1 \oplus C2 = P1 \oplus K \oplus P2 \oplus K$
  - $\oplus$ commutative and associative, so: $C1 \oplus C2 = P1 \oplus P2 \oplus K \oplus K = = P1 \oplus P2$ (by properties of $\oplus$)
  - $P1 \oplus P2 \oplus P1$ (known) $= P2$ (Success!)

# Conclusion

- OTP is not obsolete
- it needs a truly random long keystream
    - PRNG not enough
- not to be reused
- This makes OTP unpractical
- Otherwise... OTP is theoretically unbreakable (the only with proved perfect security)

# ChaCha20


ChaCha20
XOR +
XOR +
left bitwise
rotation
2306
4507
4625
4506
4697
4875
4206
1867
XOR

# ChaCha20, a synchronous stream cipher

## Objectives

- Understand the design and purpose of ChaCha20
- Compare it to older stream cipher models
- Learn how ChaCha20 builds on the one-time pad concept

# Background & history

- Designed in 2008 by Daniel J. Bernstein as an improvement of Salsa20
- Standardized in RFC 8439 by the IETF
- Focus: high speed, simplicity, and resistance to timing attacks
- Adopted in security-critical applications like TLS 1.3, OpenSSH, WireGuard, Signal, etc.

# Stream ciphers recap

- Stream ciphers generate a keystream and encrypt data by XORing it with plaintext
- Fast, low-latency encryption ideal for real-time applications
- Example: the One-Time Pad — perfectly secure but impractical

# From OTP to ChaCha20

- ChaCha20 mimics the One-Time Pad by generating a pseudo-random keystream
- Key idea: instead of a long random keystream, we use a small key + counter + nonce (random number to be used once) to generate keystream blocks
- XOR remains the central operation, just like OTP

# Inputs to ChaCha20

- **256-bit key**: shared secret
- **96-bit nonce**: unique per message/session
- **32-bit block counter**: changes per block to ensure keystream uniqueness
    - a block is a chunk of 512 bits of plaintext
- These inputs determine the internal state that drives the keystream

# Block structure

- 512-bit state divided into 16 32-bit words
  - 384 bits are variable and 128 are coming from constants
- Layout
  - 4 constant words
  - 8 key words
  - 1 counter word
  - 3 nonce words
- This state is transformed to produce 64-byte keystream blocks

# Quarter-round function

- Core operation: processes 4 words (from state) with **Add, Rotate, XOR (ARX)**
- Lightweight and fast on all CPUs
- Provides diffusion and non-linearity
- Designed for simplicity and timing safety

# 10 Double-rounds and output

- ChaCha20 performs 20 operations, organized as 10 double-rounds
- Each double-round consists of
  - 1 column-wise operation: applies 4 Quarter-Round functions to the columns of the 4×4 word matrix
  - 1 diagonal-wise operation: applies 4 Quarter-Round functions to the diagonals of the matrix
- 1 double-round = 8 Quarter-Round functions
- These transformations ensure that all 16 words of the state are thoroughly mixed in a defined pattern
- After all 10 double-rounds: add the final state to the original state (mod 2^32) to get 64 bytes of keystream

# Encryption with XOR

- **ciphertext = plaintext $\oplus$ keystream**
- Same operation used for decryption: **plaintext = ciphertext $\oplus$ keystream**
- Stateless per block: no chaining between blocks required

# Security goals of ChaCha20

- Designed to resist
  - Key recovery
  - Keystream prediction
  - Timing attacks
- No known practical attacks on full 10 double-round version
- High performance across platforms, even without hardware support

# Real-world usage

- TLS 1.3: used as an option (in addition to others) for secure web traffic
- Signal, WhatsApp: encrypt messages using ChaCha20 (or other, depending on context)
- WireGuard VPN: lightweight and fast encryption
- OpenSSH: option for encrypted shell sessions, now less used