

Symmetric Encryption

Introduction to AES and ChaCha20

Jack P. (Master of Coin)

**I am not an
expert**

I just like maths :)

Synopsis

(aka what are we talking about)

- What is a symmetric cipher?
- Understanding when symmetric ciphers are practical (and secure!)
- What are the differences between a block and stream cipher?
- When to use a stream cipher or a block cipher?
- Brief overview of popular symmetric ciphers: AES and ChaCha20
- Understanding limitations of “naked” symmetric ciphers
- Knowing which cipher-suite to use when securing data

What is a symmetric cipher?

- A symmetric cipher relies on a shared key to encrypt and decrypt data.

$$\text{Enc}(k, M) = C \text{ and } \text{Enc}^{-1}(k, C) = M$$

where k is a random bit vector and M is the plaintext

- The key is used for both encryption and decryption.

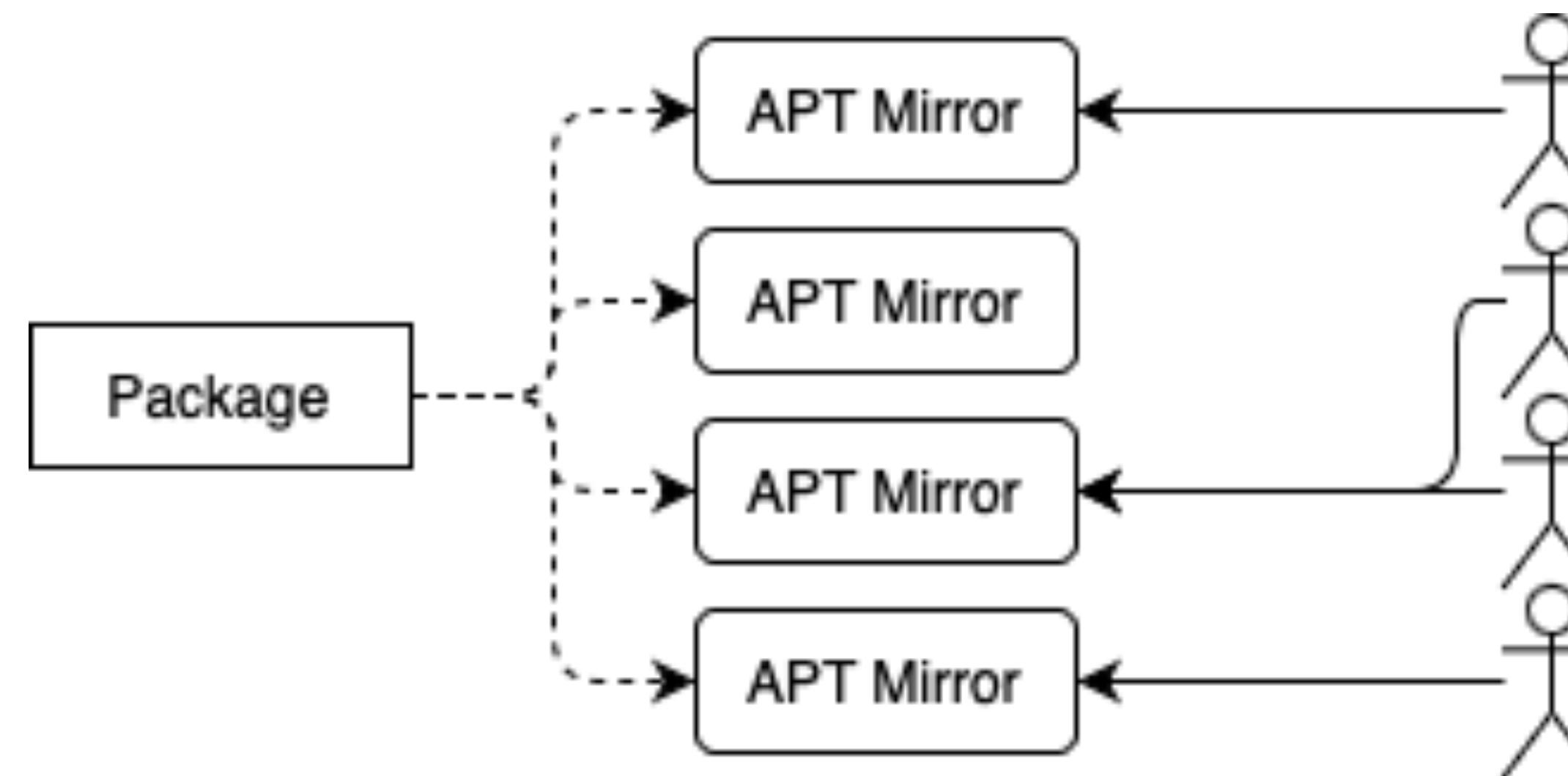
When are symmetric ciphers practical?

- If the data is static (e.g., encrypted hard-drive or encrypted backups).
- If the parties can securely transmit a key (e.g., PSK or via AKE).
- If the encryption must be performant (e.g., RTP).
- If the encryption must be efficient (e.g., encryption on embedded devices).



When are symmetric ciphers impractical?

- If data is communicated between unknown parties (e.g., package mirror and user).
- If only one peer should produce data (e.g., software updates).



Are symmetric ciphers secure?

- Symmetric ciphers are not invulnerable to attacks:
TEA, XTEA, DES, Blowfish, RC4, etc.
- There are concerns regarding Post-Quantum-Cryptography (PQC):
Grover's algorithm

Not inherently. All cryptographic protocols must be designed with upgradability in mind to adapt as research advances.

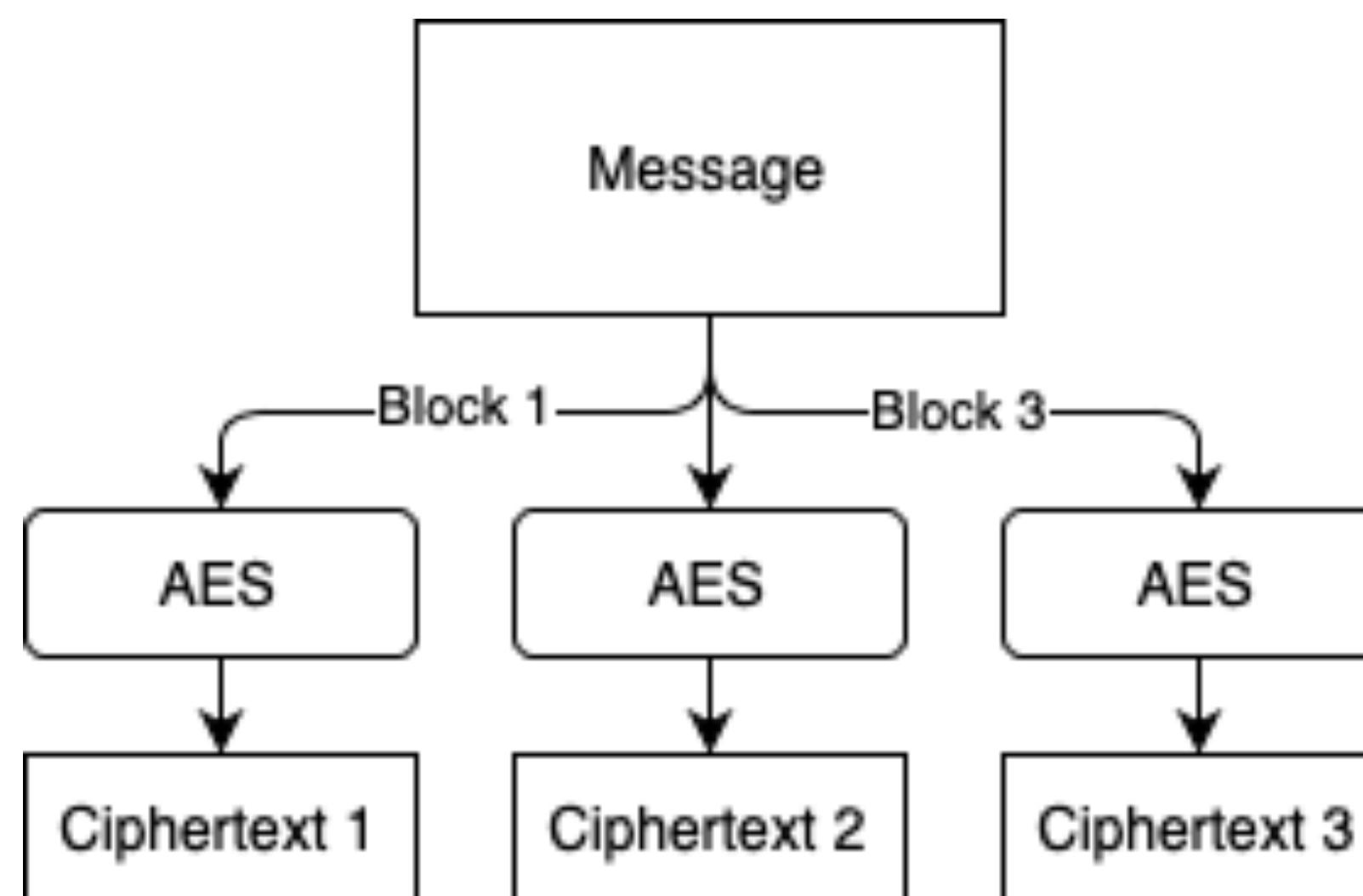
What is a block cipher and stream cipher?

- A block cipher encrypts data with a fixed key whereas a stream cipher encrypts using a continuous key stream.
- Block ciphers use a shared key to create an initial state which is mixed with the plaintext to produce ciphertext.
(e.g., XOR-ing plaintext against a random key)
- Stream ciphers use a shared key to seed an initial state which produces a stream of bits which are mixed with the plaintext to produce the ciphertext.
(i.e., seed a CSPRNG to generate a keystream)
- **The input length to a block cipher must be a multiple of the block size.**

What is AES?

Overview

- AES (“Rijndael”) is a block cipher developed for NIST in 1998.
- AES supports 128, 192, and 256-bit block sizes (applies to key sizes).
- AES has shown to be secure against all public research.



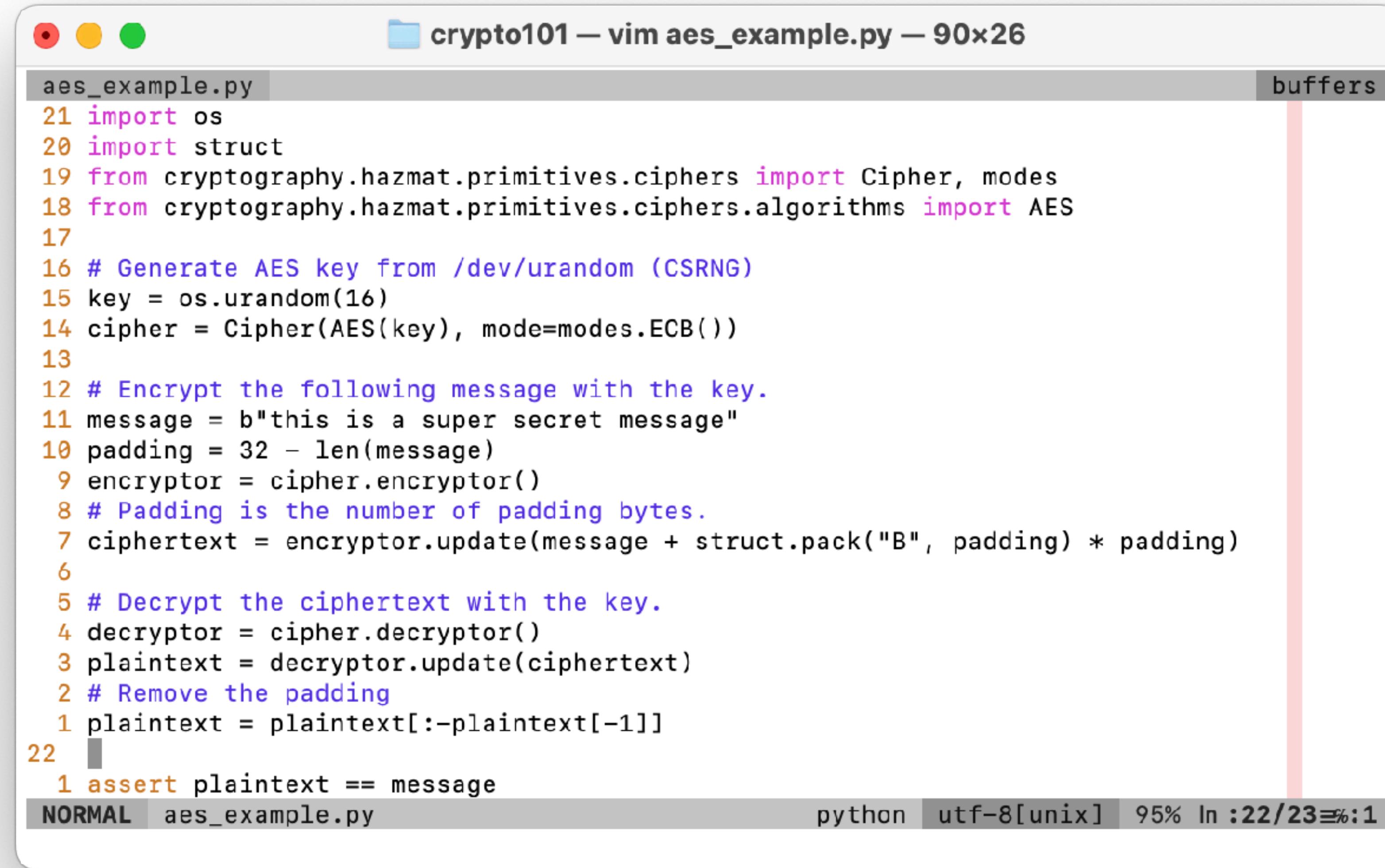
What is AES?

Design

- The input is split into block sized chunks which are independently encrypted. If the input length is incongruent with the block size then **the input is padded to the next full block.**
- AES performs a series of operations on 4x4 matrices (a block).
- AES implements a series of “rounds” which shuffle the matrix elements along the columns and rows, influenced by a round key derived from the input key.
- AES uses an S-box (substitution box) to swap elements of the matrix against a hard-coded table for *advanced maths reasons*.

What is AES?

Application



The screenshot shows a terminal window titled "crypto101 – vim aes_example.py – 90x26". The window displays a Python script named "aes_example.py". The script demonstrates the use of the "cryptography" library to perform AES encryption and decryption. It includes importing os, struct, and the necessary modules from cryptography. It generates a random key, creates a cipher object in ECB mode, and encrypts a message. The message is padded to 32 bytes. It then decrypts the ciphertext back to the original message and asserts that the decrypted message matches the original.

```
aes_example.py
21 import os
20 import struct
19 from cryptography.hazmat.primitives.ciphers import Cipher, modes
18 from cryptography.hazmat.primitives.ciphers.algorithms import AES
17
16 # Generate AES key from /dev/urandom (CSRNG)
15 key = os.urandom(16)
14 cipher = Cipher(AES(key), mode=modes.ECB())
13
12 # Encrypt the following message with the key.
11 message = b>this is a super secret message"
10 padding = 32 - len(message)
 9 encryptor = cipher.encryptor()
 8 # Padding is the number of padding bytes.
 7 ciphertext = encryptor.update(message + struct.pack("B", padding) * padding)
 6
 5 # Decrypt the ciphertext with the key.
 4 decryptor = cipher.decryptor()
 3 plaintext = decryptor.update(ciphertext)
 2 # Remove the padding
 1 plaintext = plaintext[:-plaintext[-1]]
22
 1 assert plaintext == message
NORMAL  aes_example.py
python  utf-8[unix]  95%  ln :22/23=1
```

What is ChaCha20?

Overview

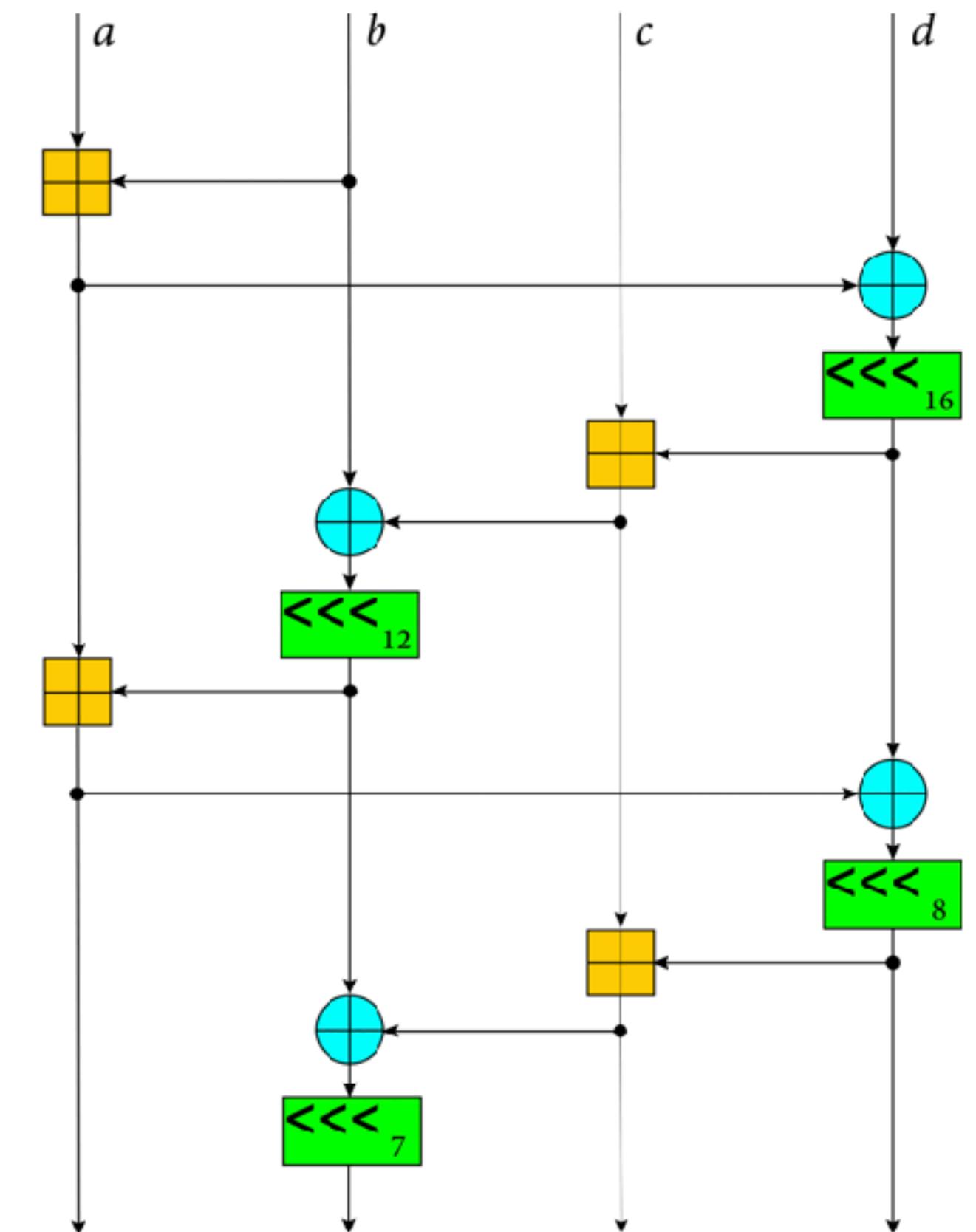
- ChaCha20 is a stream cipher designed by Daniel J. Bernstein in 2008 as an adaptation of Salsa20.
- ChaCha20 has 128 and 256 bit keys with a 512-bit state (for the stream).
- ChaCha20 has shown to be secure against all public research.

(the IETF has a proposal for XChaCha20 with better nonce misuse protection)

What is ChaCha20?

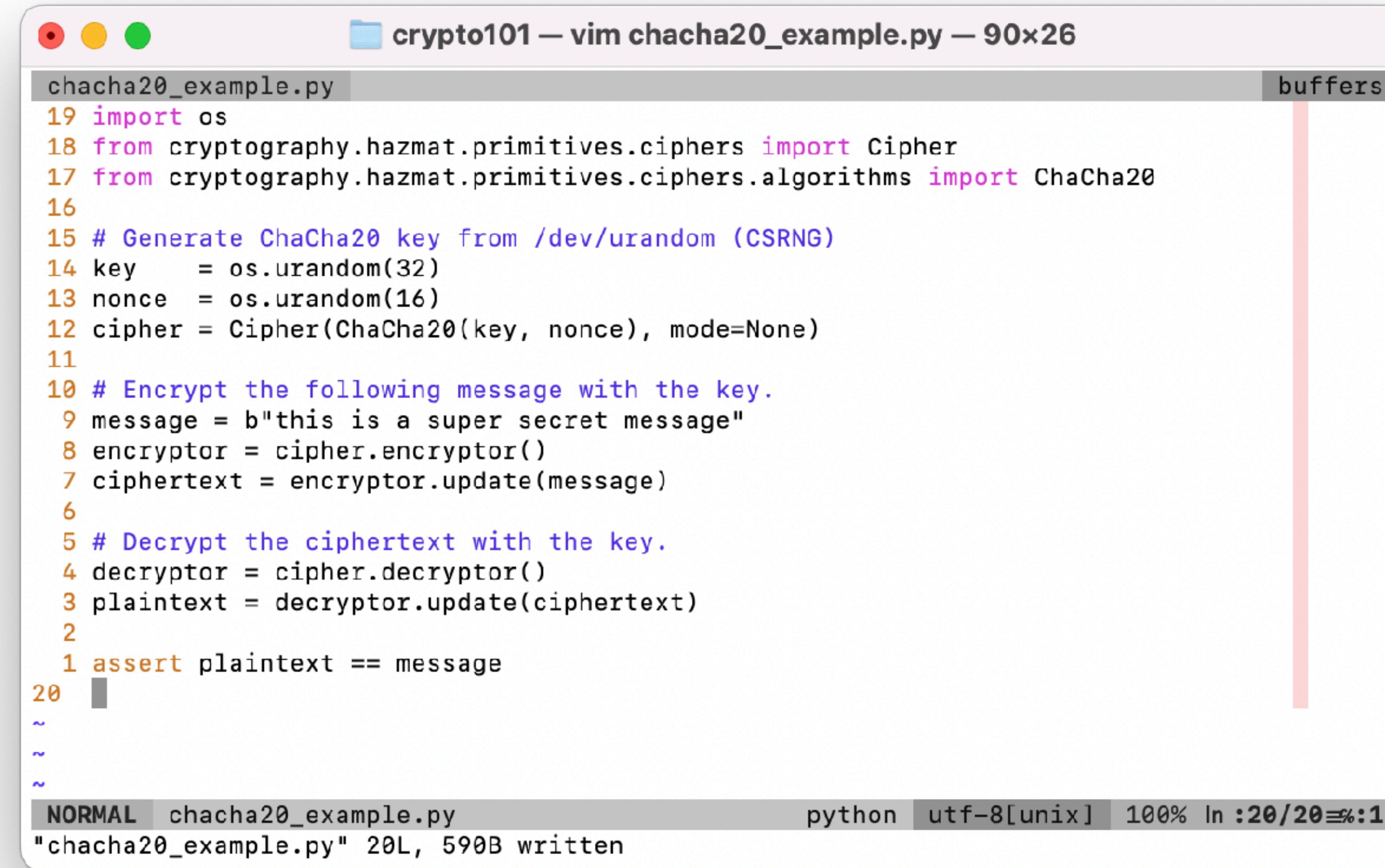
Design

- ChaCha20 generates a keystream from a nonce (we'll get to that), counter, and key.
- The keystream is updated for every block of input encrypted by ChaCha20.
- Each plaintext block is mixed against the current state to generate ciphertext.



What is ChaCha20?

Application



The screenshot shows a terminal window titled "crypto101 – vim chacha20_example.py – 90x26". The file "chacha20_example.py" contains the following code:

```
chacha20_example.py
19 import os
18 from cryptography.hazmat.primitives.ciphers import Cipher
17 from cryptography.hazmat.primitives.ciphers.algorithms import ChaCha20
16
15 # Generate ChaCha20 key from /dev/urandom (CSRNG)
14 key    = os.urandom(32)
13 nonce  = os.urandom(16)
12 cipher = Cipher(ChaCha20(key, nonce), mode=None)
11
10 # Encrypt the following message with the key.
 9 message = b>this is a super secret message"
 8 encryptor = cipher.encryptor()
 7 ciphertext = encryptor.update(message)
 6
 5 # Decrypt the ciphertext with the key.
 4 decryptor = cipher.decryptor()
 3 plaintext = decryptor.update(ciphertext)
 2
 1 assert plaintext == message
20
```

The terminal status bar at the bottom indicates:

NORMAL chacha20_example.py python utf-8[unix] 100% In :20/20 È%:1
"chacha20_example.py" 20L, 590B written

ChaCha20 vs. AES

Which to choose and why

- Hardware accelerated AES is faster than software implementations of ChaCha20 (if you support AES-NI then AES is more efficient).
- Software implementations of AES are outperformed by ChaCha20.
- Legal compliance: FIPS 💩
- Using ChaCha20 to avoid AES S-box timing attacks.
- ChaCha20 has built-in nonce protection unlike raw AES.

Practical Stuff :)

AES distinguishability

Characterising the message plaintext from ciphertext.

- Clone the repository: <https://github.com/gusecurity/crypto101>
- Install dependencies:
\$ python -m venv .venv
\$ source .venv/bin/activate
\$ pip install -r requirements.txt
- Read the script in ex1/election.py then determine who voted A or B.
60% voted for A and 40% voted for B.
- The “packet capture” of the voting machine is in ex1/votes.txt

AES distinguishability

Characterising the message plaintext from ciphertext.

Voted for A	Voted for B
1	0
2	5
3	8
4	9
6	
7	

AES distinguishability

Characterising the message plaintext from ciphertext.



The terminal window shows a command-line session in a zsh shell on a laptop named crypto101. The user is in a virtual environment (.venv). The command `sort -k2 ex1/votes.txt | uniq -cf1 | cut -wf4,6` is run to process the file `ex1/votes.txt`. The output consists of two lines of hex-encoded data, each starting with a number from 0 to 7 followed by a series of characters. The first line starts with 0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482... and the second line starts with a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482... . The terminal window has a title bar "crypto101 -- zsh -- 90x24".

```
[(.venv) jack@laptop crypto101 % sort -k2 ex1/votes.txt | uniq -cf1 | cut -wf4,6
0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482...
a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
[.venv) jack@laptop crypto101 % sort -k2 ex1/votes.txt
0 0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482...
5 0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482...
8 0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482...
9 0484013d842617ea0ae14e9701f5d0f039733ff4e7836c207a3aa1d089a80482...
1 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
2 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
3 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
4 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
6 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
7 a5e7809ea74040bbae2d755c7f5336b639733ff4e7836c207a3aa1d089a80482...
(.venv) jack@laptop crypto101 %
```

AES distinguishability

Characterising the message plaintext from ciphertext.

- This can be fixed by using a key per user (since we only encrypt a single block) but to decrypt a vote we must know who voted!
- **Using a cipher mode like CBC will not help since we have a single block!**
- Use paper ballots... but that's too reasonable
- Use nonces!

AES distinguishability (nonces)

Characterising the message plaintext from ciphertext.

Not this one



AES distinguishability (nonces)

Characterising the message plaintext from ciphertext.

- Nonces are a way to scramble the ciphertext such that two identical blocks of plaintext correspond to distinct ciphertext blocks.
- **Read the standardisation material for your ciphersuite for proper usage.**
- Common implementations randomly generate the nonce or use a deterministic counter.
- For some algorithms, reusing a nonce can be catastrophic (see AESGCM and ECDSA).
- **We can use a vote counter as a nonce for the ciphertext.**

Ciphertext Malleability

Blind modifications to ciphertext to exploit applications.

- Use the same repository as before.
- Run the server with:
\$ python ex2/server.py 8080
- Read the script `ex2/server.py` then modify the captured ciphertext in `ex2/message.txt` to login as `WebAdmin` to get the flag.
- If you are not confident with Python then feel free to adapt `ex2/partial_sol.py`.

Ciphertext Malleability

Blind modifications to ciphertext to exploit applications.

The flag is in the source 5head.

Ciphertext Malleability

Blind modifications to ciphertext to exploit applications.

```
e/sol.py
19 import socket
18 from random import randint
17
16 N_REQUESTS = 2_000
15
14 for i in range(N_REQUESTS):
13     with socket.create_connection(("localhost", 8080)) as s:
12         ciphertext = list(s.recv(64))
11         for _ in range(randint(0, 5)):
10             ciphertext[randint(0, len(ciphertext) - 1)] ^= 1 << randint(0, 7)
9             s.send(bytarray(ciphertext))
8
7         try:
6             buf = s.recv(64)
5             if b"flag" in buf:
4                 print(buf)
3                 break
2         except:
1             continue
20
~
~
~
```

NORMAL ex2/sol.py python [unix] 100% In :20/20 =%:1
"ex2/sol.py" 20L, 505B written

Ciphertext Juggling

I have no idea what the proper term is.

- The third exercise is the most subtle (and appears in real scenarios, such as NFC payments).
- The bank teller sends transactions to a central banking service.
- Figure out a way to transfer £500 from Alice to Eve from the captured transactions in ex3/transactions.txt.
- The script ex3/bank.py is responsible for creating the transactions.
- **This is hard so please ask for hints if you are stuck.**
- **The captured messages use a different key so the correct solution will not work!**

Ciphertext Juggling

I have no idea what the proper term is.

A screenshot of the Vim text editor displaying a Python script named `e/sol.py`. The title bar shows "crypto101 – vim ex3/sol.py – 90x26". The buffer tab bar shows "buffers". The code itself is as follows:

```
e/sol.py
14 with open("ex3/transactions.txt") as f:
13     transactions = [bytes.fromhex(l.strip()) for l in f.readlines()[1::2]]
12
11 # ciphertext structure of transaction:
10 # block 1: sender
  9 # block 2: amount|sender_bank|receiver_bank|timestamp
  8 # block 3: receiver
  7
  6 bob    = transactions[0][64: ]
  5 eve    = transactions[1][:32]
  4 amount = transactions[0][32:64]
  3
  2 transaction = bob + amount + eve
  1 print(transaction.hex())
15
```

The status bar at the bottom shows "NORMAL ex3/sol.py" and "python utf-8[unix] 100% In :15/15=%:1". There are also several cursor icons on the left side of the editor window.

Optional Exercise (Advanced)

Cipher Modes

WTF is this CBC thing?

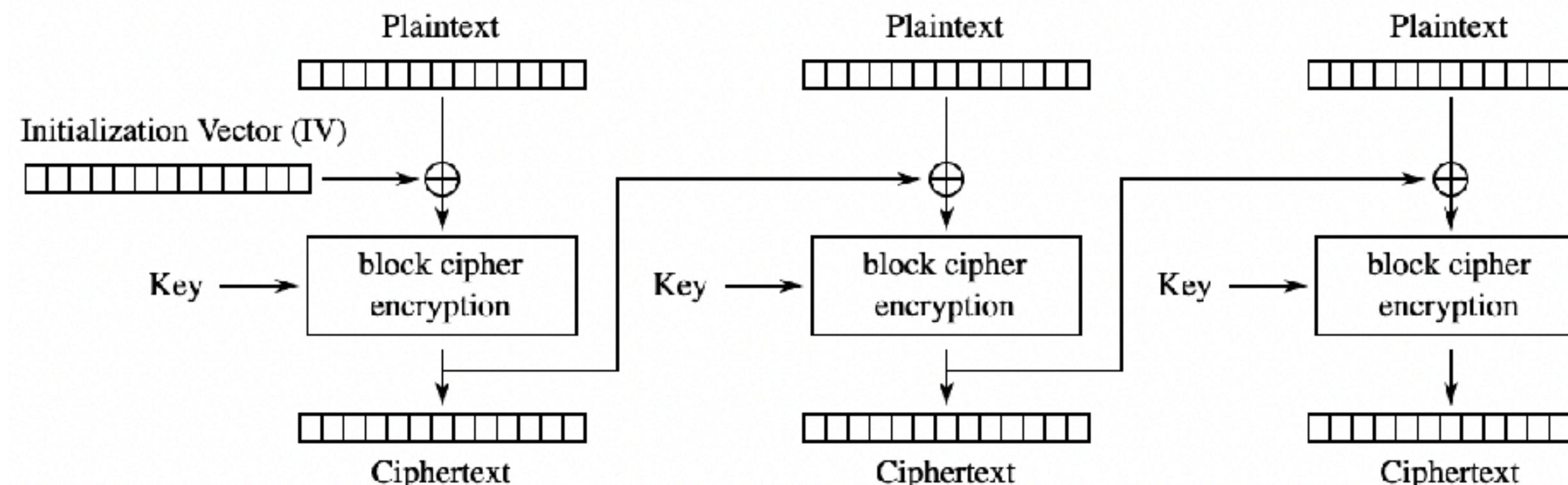
- Block ciphers have modes designed to scramble the ciphertext to prevent reliable malleability and distinguishing ciphertext.
- Cipher modes can be thought of as a way to transform a primitive cipher like AES by adding desirable properties (such as the above).
- Not all cipher modes are perfect: CBC does not have parallelised encryption.
- **Additional functionality can introduce insecurities.**

AES-CTR can turn AES into a pseudo-stream cipher!

Cipher Modes

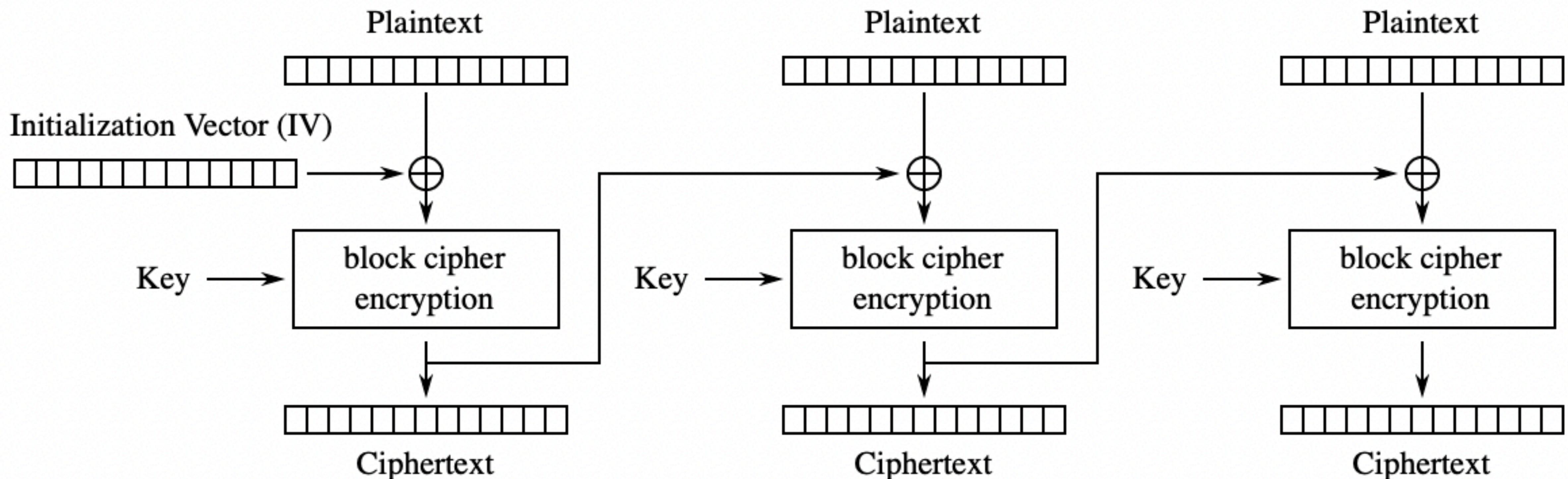
WTF is this CBC thing?

- CBC (Cipher-Block-Chaining) uses the previous ciphertext block to scramble the subsequent block by XOR-ing the plaintext.
- CBC implements padding by padding up to the next full block.
If the plaintext is a multiple of the block size then a block of padding is added.
- The padding is the number of padding bytes appended to the **plaintext (PKCS#7)**.



Cipher Modes

WTF is this CBC thing?



Padding Oracle

You thought CBC could save you?

- The final exercise is a walkthrough showing how to leak the plaintext of the message between Alice and Bob.
- The `ex4/decryptor.py` script will attempt to decrypt an input message.
- The captured message between Alice and Bob is in `ex4/message.txt`.
- Partial solution in `ex4/partial_sol.py`.
- **Feel free to give it a shot if you feel confident.**

Padding Oracle

You thought CBC could save you?

- This attack is online and requires interaction with the decryption oracle.
- This can be prevented using **authentication**.
- **Note:** padding oracle can serve as an *encryption oracle* (CBC-R).

Summary

Authentication

(and nonces)

Epilogue

(the boring bit)

- Please participate in the poll on the Discord channel to vote for the next workshop topic!
- If you have any suggestions then reach out to any of the committee members.
- If I could have done better then let me know so I can improve :)

If you enjoyed this topic then I am happy to do future presentations on other cryptographic concepts (and even cryptanalysis).