CSE 127: Computer Security

Symmetric-key Cryptography

Deian Stefan

Some slides adopted from Nadia Heninger, Kirill Levchenko and Dan Boneh

Cryptography

- Is:
  - ➢ A tremendous tool
  - ➢ The basis for many security mechanisms
- Is not:
  - ➢ The solution to all security problems
  - ➢ Reliable unless implemented and used properly
  - ➢ Something you should try to invent yourself
  - ➢ Another word for blockchain

How Does It Work?

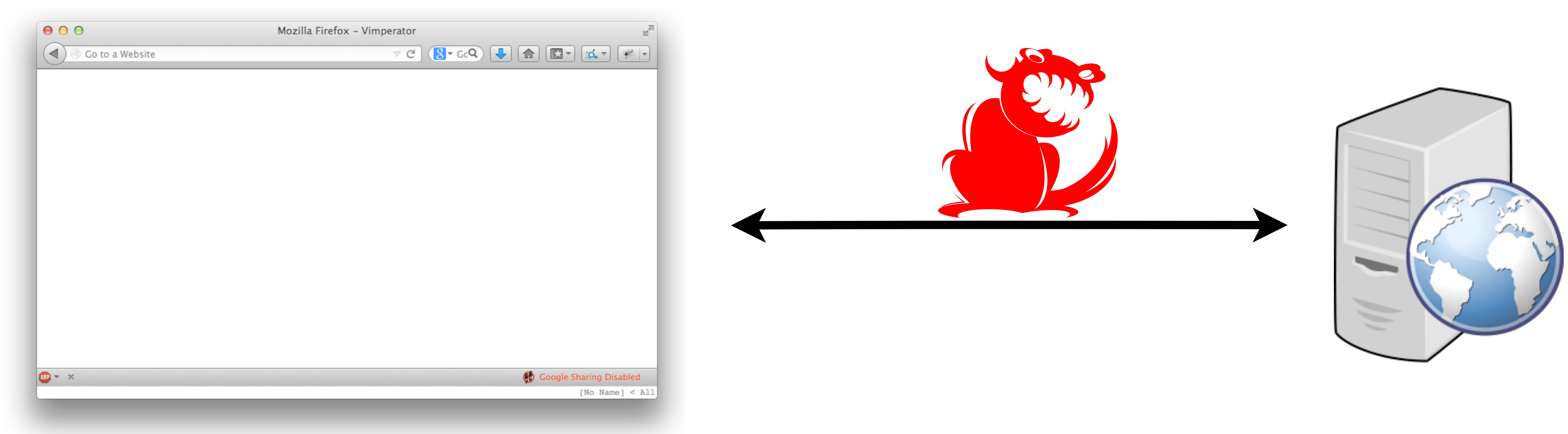
- Goal: learn how to use crypto primitives correctly
  - We will treat them as a black box that mostly does what it says
- To learn what's inside black box take CSE 107

How Does It Work?

- Goal: learn how to use crypto primitives correctly
  - We will treat them as a black box that mostly does what it says
- To learn what's inside black box take CSE 107
- Do not roll your own crypto*

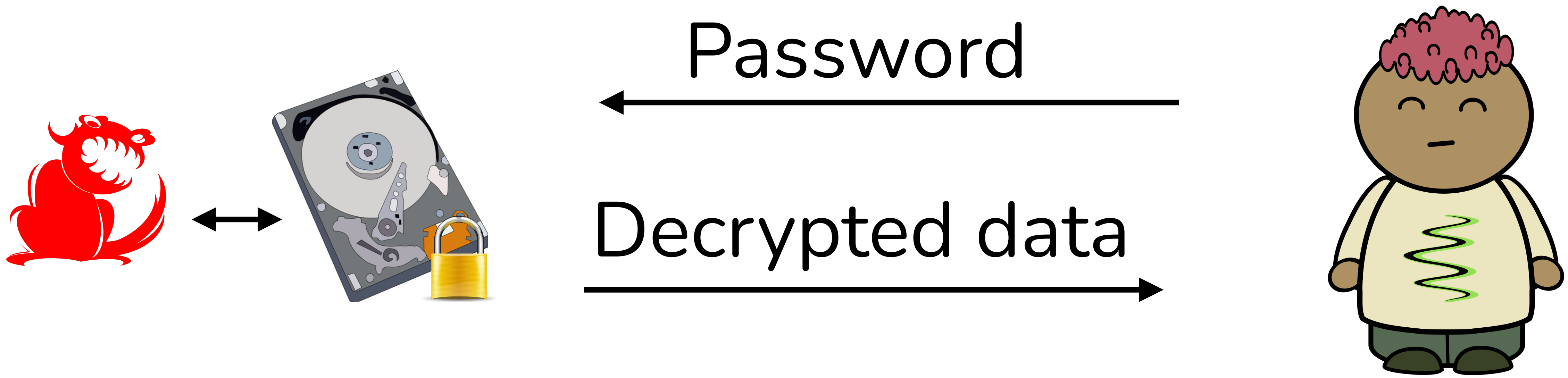
* Exceptions: You are Daniel J. Bernstein, Joan Daemen, Neal Koblitz, Dan Boneh, or similar, or you have finished your PhD in cryptography under an advisor of that caliber, and your work has been accepted at Crypto, Eurocrypt, Asiacrypt, FSE, or PKC and/or NIST is running another competition, and then wait several years for full standardization and community vetting.

Real-world crypto: SSL/TLS



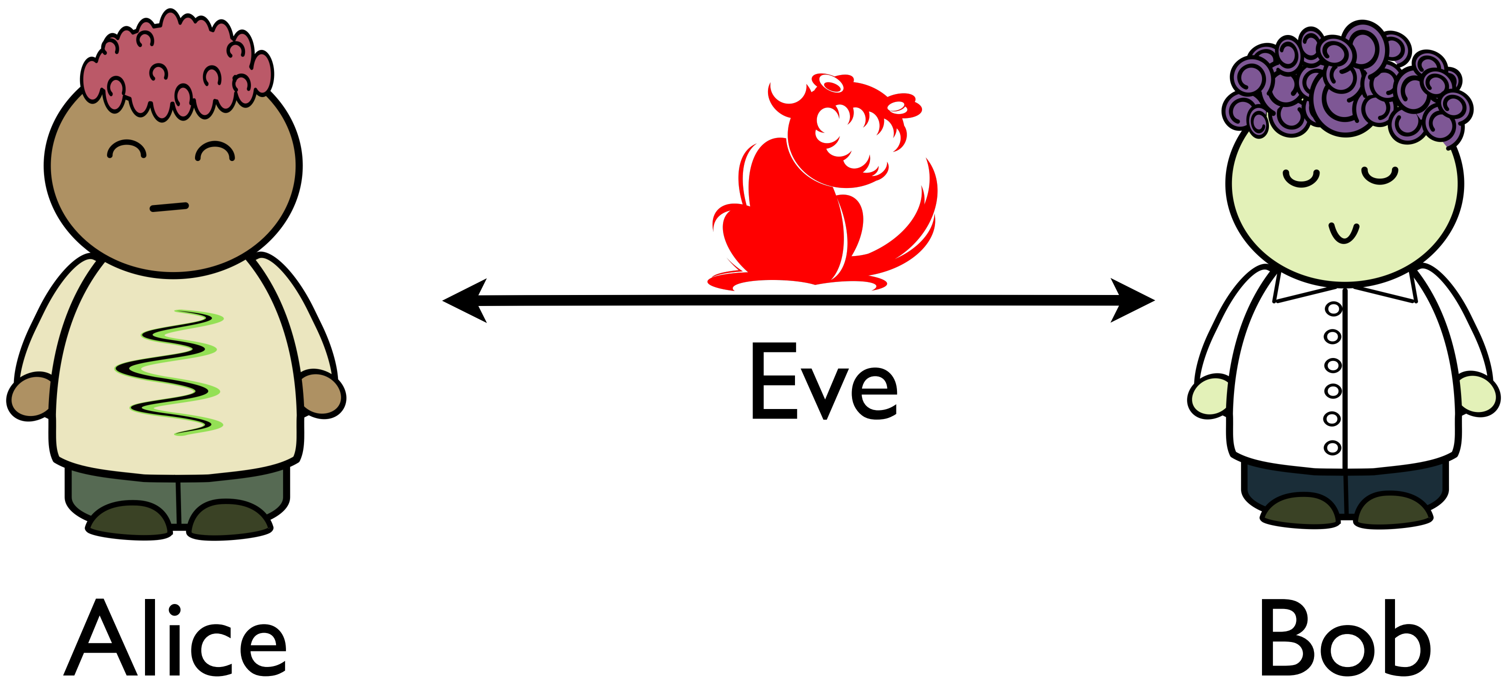
1. **Browser and web server run "handshake protocol":**
   - Establishes shared secret key using public-key cryptography (next lecture)
2. **Browser and web server use negotiated key to symmetrically encrypt** data ("Record layer")

Real-world crypto: File encryption



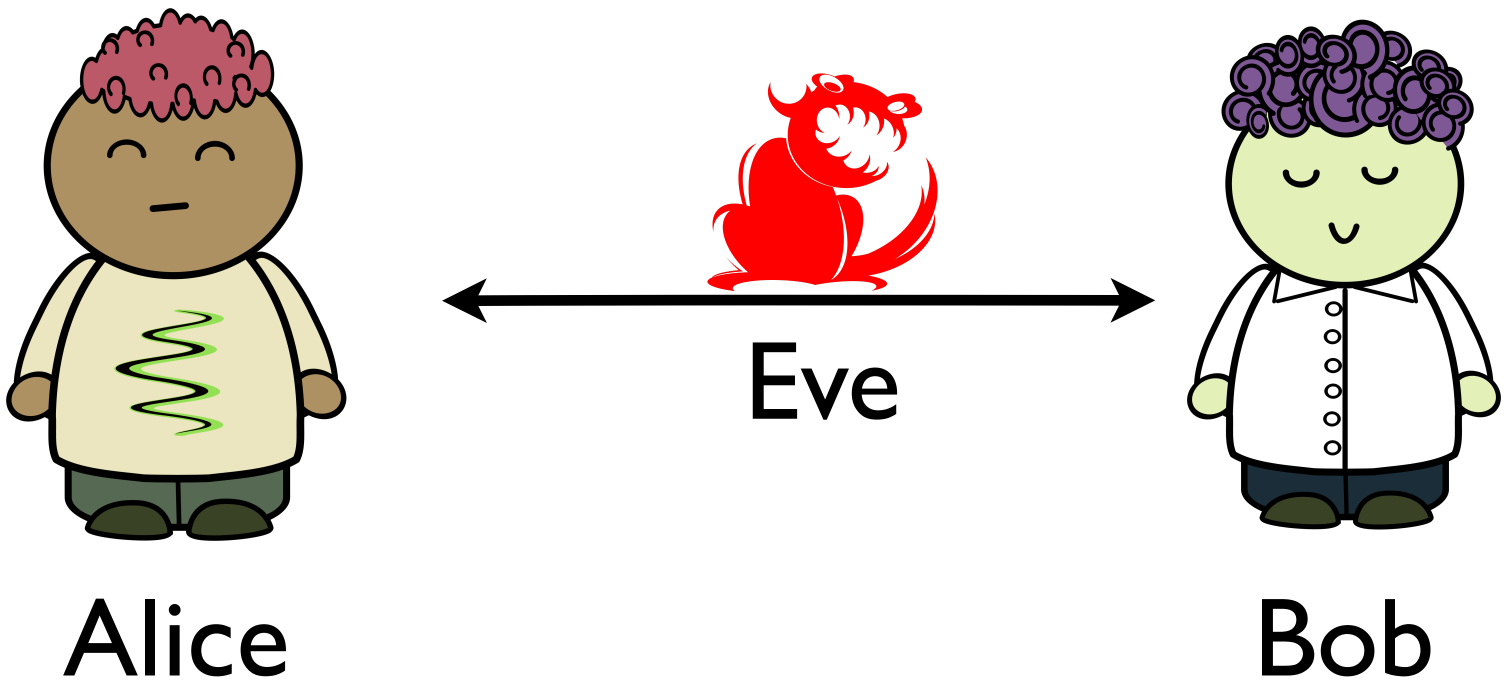
- Files are symmetrically encrypted with a secret key
- The symmetric key is stored encrypted or in tamperproof hardware.
- The password is used to unlock the key so the data can be decrypted.

This class: secure communication



- *Authenticity:* Parties cannot be impersonated
- *Secrecy:* No one else can read messages
- *Integrity:* Messages cannot be modified

Attacker models



- *Passive attacker: Eve only snoops on channel*
- *Active attacker: Eve can snoop, inject, block, tamper, etc.*

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Next time: asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

Symmetric-key encryption

- **Encryption:** (key, plaintext) → ciphertext
  - ➤ $E_k(m) = c$
- **Decryption:** (key, ciphertext) → plaintext
  - ➤ $D_k(c) = m$
- **Functional property:** Where $D_k(E_k(m)) = m$

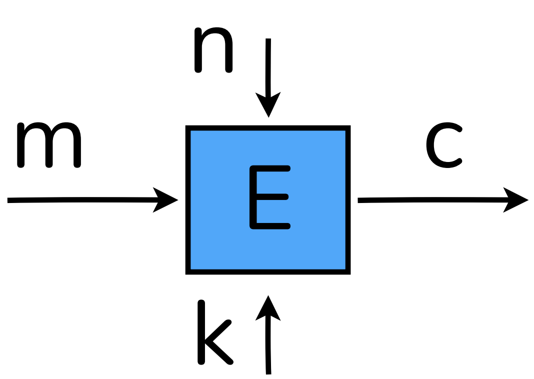
Symmetric-key encryption

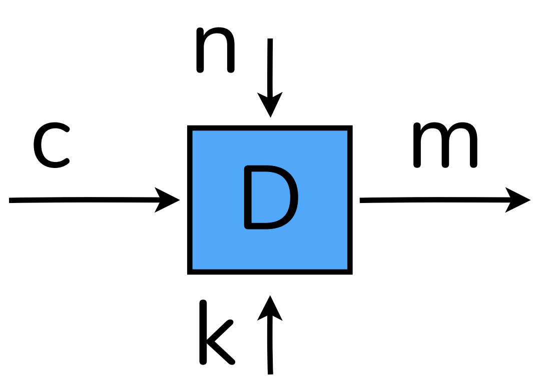
- **One-time key:** used to encrypt one message
  - E.g., encrypted email, new key generate per email

Symmetric-key encryption

- **One-time key:** used to encrypt one message
  - ➢ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➢ E.g., same key used to encrypt many packets

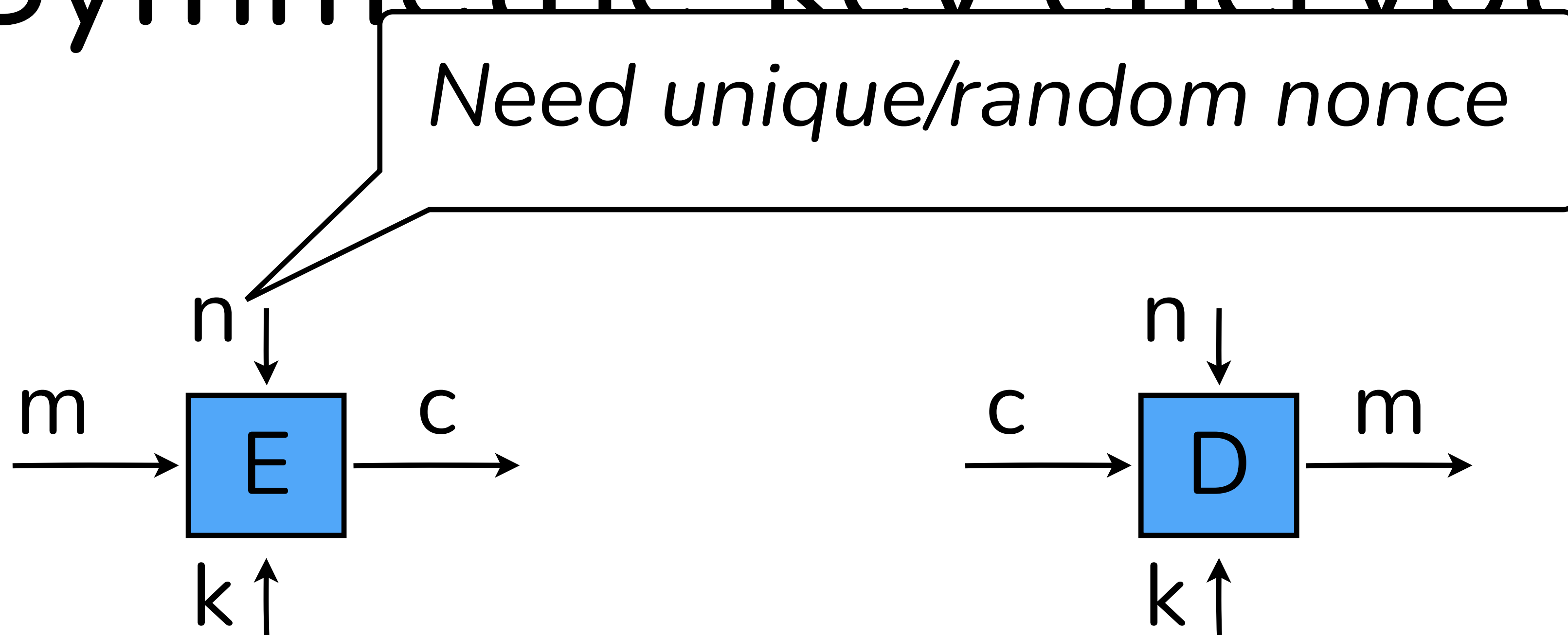
Symmetric-key encryption





- **One-time key:** used to encrypt one message
  - ➢ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➢ E.g., same key used to encrypt many packets

Symmetric-key encryption

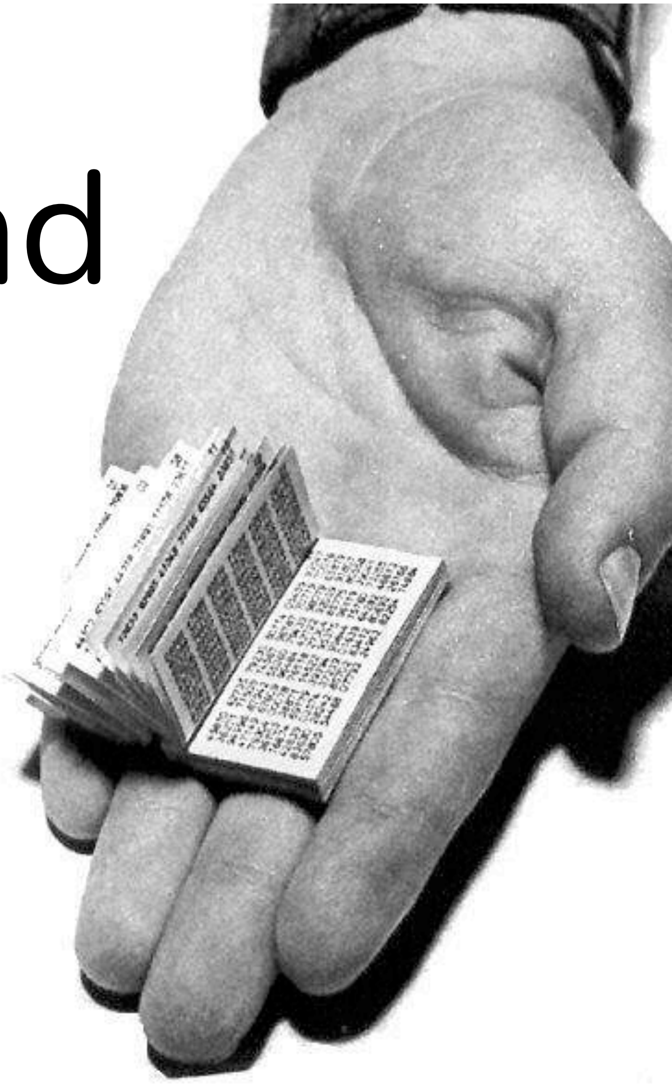


- **One-time key:** used to encrypt one message
  - ➤ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➤ E.g., same key used to encrypt many packets

Security definition: Passive eavesdropper

- Simplest security definition
  - How do you know an encryption scheme is secure against a passive eavesdropper?
  - Want: "Ciphertext reveals nothing about plaintext"
  - Informal formal definition: Given $E_k(m_1)$ and $E_k(m_2)$, attacker can't distinguish which ciphertext encrypts which plaintext without key

Example: One Time Pad



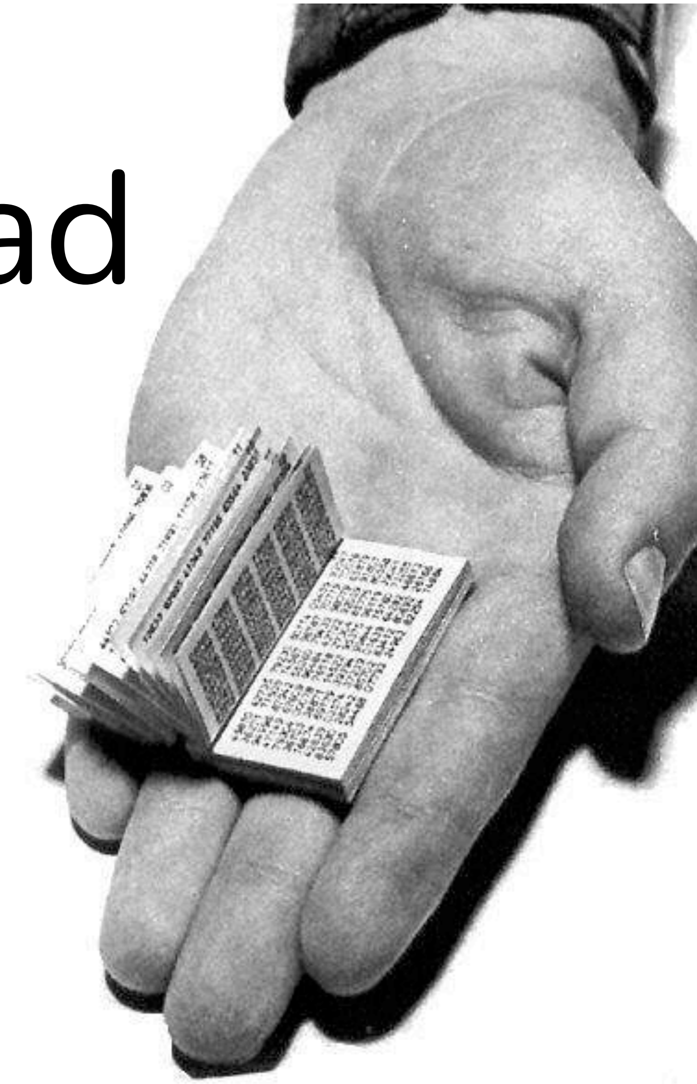
Vernam (1917)

Key:	0	1	0	1	1	1	0	0	1	0
Plaintext:	1	1	0	0	0	1	1	0	0	0
Ciphertext:	1	0	0	1	1	0	1	0	1	0

$\oplus$

- **Encryption:**
- **Decryption:**

Example: One Time Pad



Vernam (1917)

Key:	0	1	0	1	1	1	0	0	1	0
Plaintext:	1	1	0	0	0	1	1	0	0	0
Ciphertext:	1	0	0	1	1	0	1	0	1	0

$\oplus$

- **Encryption:** $c = E_k(m) = m \oplus k$
- **Decryption:** $D_k(c) = c \oplus k = (m \oplus k) \oplus k = m$

OTP security

- Shannon (1949)
  - ➤ Information-theoretic security: without key, ciphertext reveals no "information" about plaintext
- Problems with OTP
  - ➤ Can only use key once
  - ➤ Key is as long as the message

Computational cryptography

- Want to encrypt with shorter keys
  - Problem: information-theoretic security is impossible if key space is smaller than message space.
- Solution: Use a more practical security notion
  - It should be infeasible for a computationally bounded attacker to violate security
  - In practice: attacks should take at least e.g., 2^{128} time

Stream ciphers

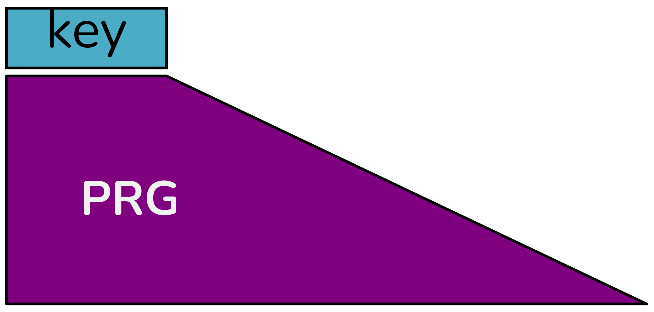
- Problem: OTP key is as long as message
- Solution: Pseudo random generator

Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator

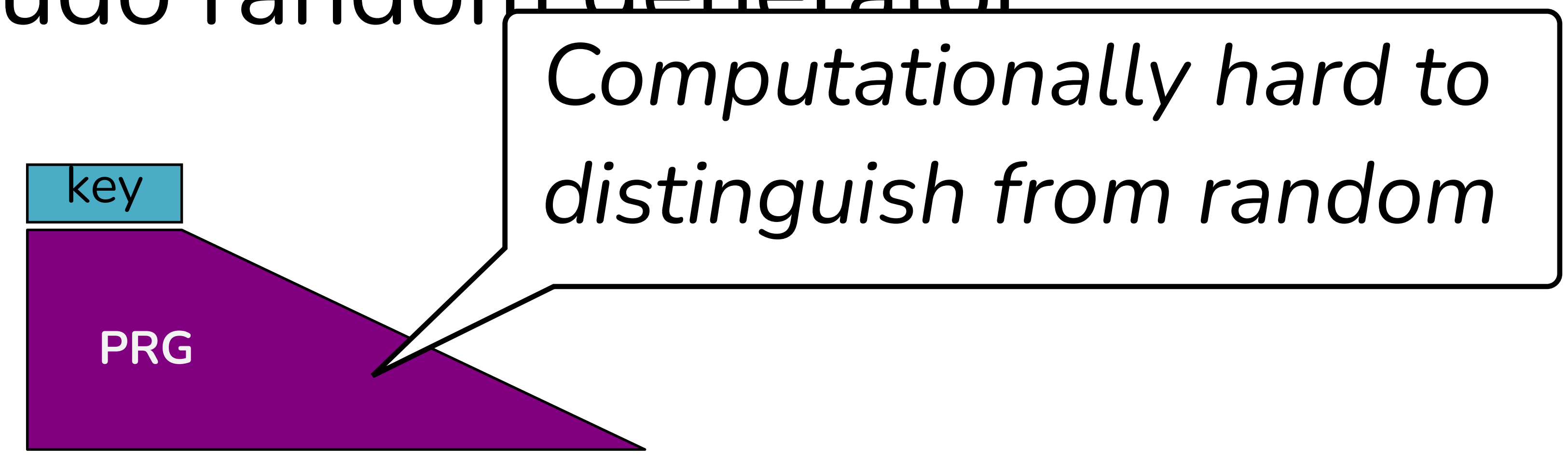
Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator



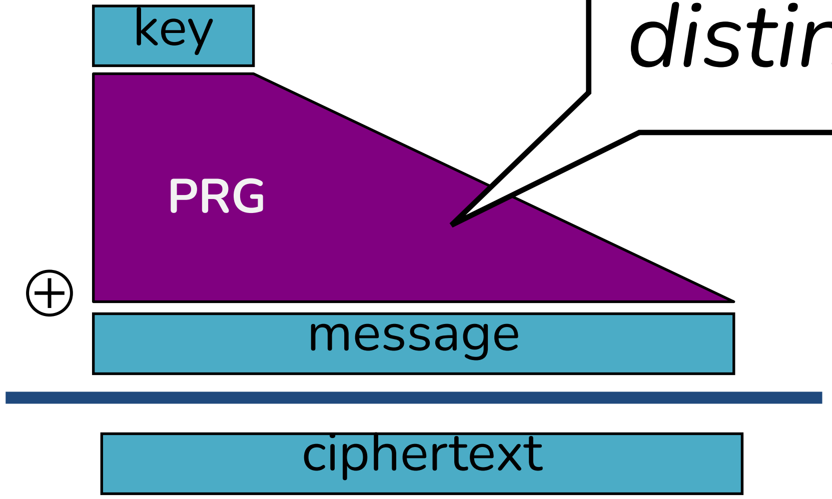
Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator



Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator

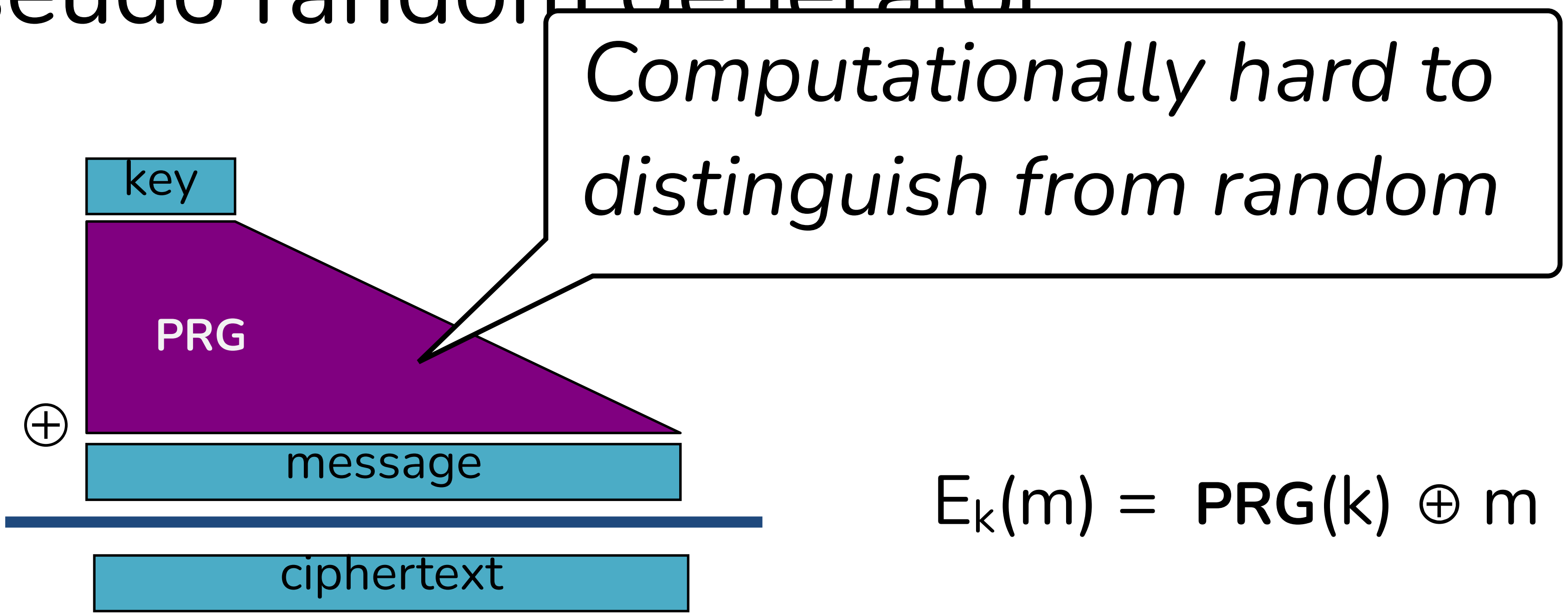


Computationally hard to distinguish from random

$E_k(m) = \textbf{PRG}(k) \oplus m$

Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator



$E_k(m) = \mathbf{PRG}(k) \oplus m$

  - ➤ Examples: ChaCha, Salsa, etc.

Dangers in using stream ciphers

- Can we use a key more than once?
  - E.g., $c_1 \leftarrow m_1 \oplus PRG(k)$

    $c_2 \leftarrow m_2 \oplus PRG(k)$
  - Yes? No?

Dangers in using stream ciphers

- Can we use a key more than once?
  - E.g., $c_1 \leftarrow m_1 \oplus PRG(k)$

    $c_2 \leftarrow m_2 \oplus PRG(k)$
  - Yes? No?
  - Eavesdropper does: $c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$
  - Enough redundant information in English that:
    $m_1 \oplus m_2 \rightarrow m_1, m_2$

Chosen plaintext attacks

- Attacker can learn encryptions for arbitrary plaintexts
- Historical example:
  - ➢ During WWII the US Navy sent messages about Midway Island and watched Japanese ciphertexts to learn codename (“AF”)
- More recent (but still a bit old) example:
  - ➢ WEP WiFi encryption has poor randomization and can result in the same stream cipher used multiple times

Block ciphers: crypto work horses

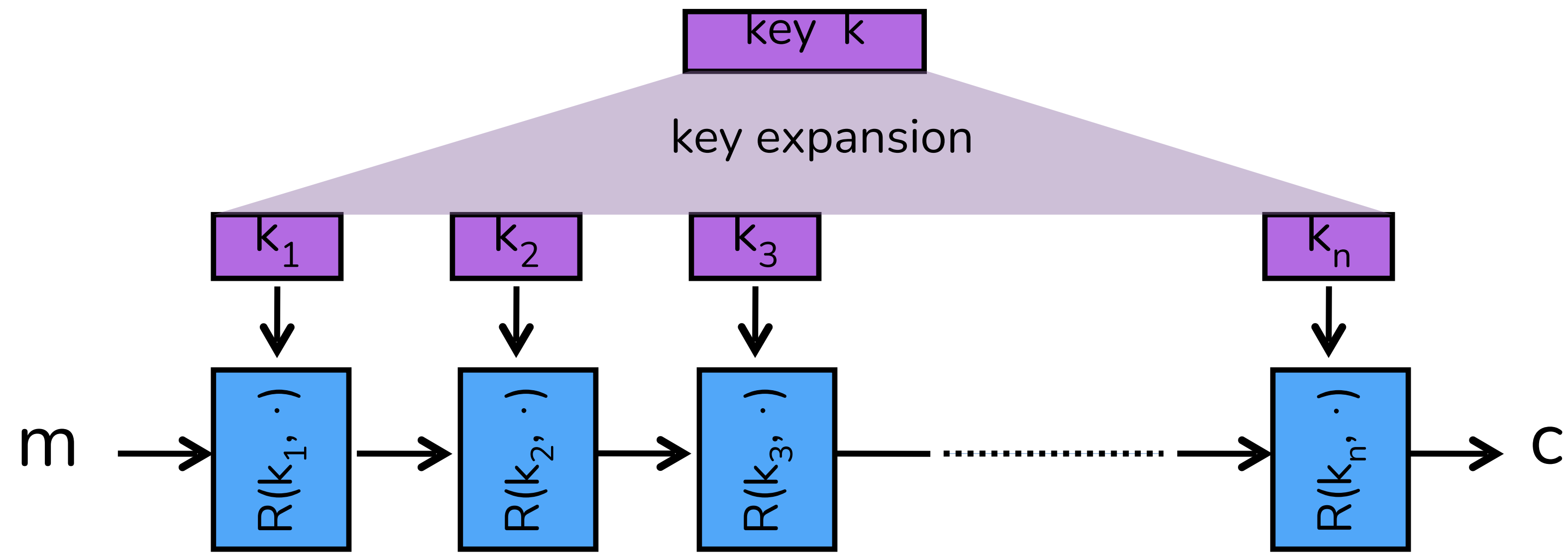
- Block cipher: permutation of fixed-size input block
  - ➤ Each input is mapped to one output (depends on key)
- Common examples:
  - ➤ E.g., 3DES: $|m| = |c| = 64$ bits, $|k| = 168$ bits
  - ➤ E.g., AES: $|m| = |c| = 128$ bits, $|k| = 128, 192, 256$

Block ciphers: crypto work horses

- Block cipher: permutation of fixed-size input block
  - ➤ Each input is mapped to one output (depends on key)
- Common examples:
  - ➤ E.g., 3DES: |m| = |c| = 64 bits, |k| = 168 bits
  - ➤ E.g., AES: |m| = |c| = 128 bits, |k| = 128, 192, 256

Correct block cipher choice: AES

What's inside the box?



R(k,m): round function
for AES-128 (n=10)

What's inside that?

Challenges with block ciphers

Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?

Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?
  - Several modes of operation for longer messages

Challenges with block ciphers

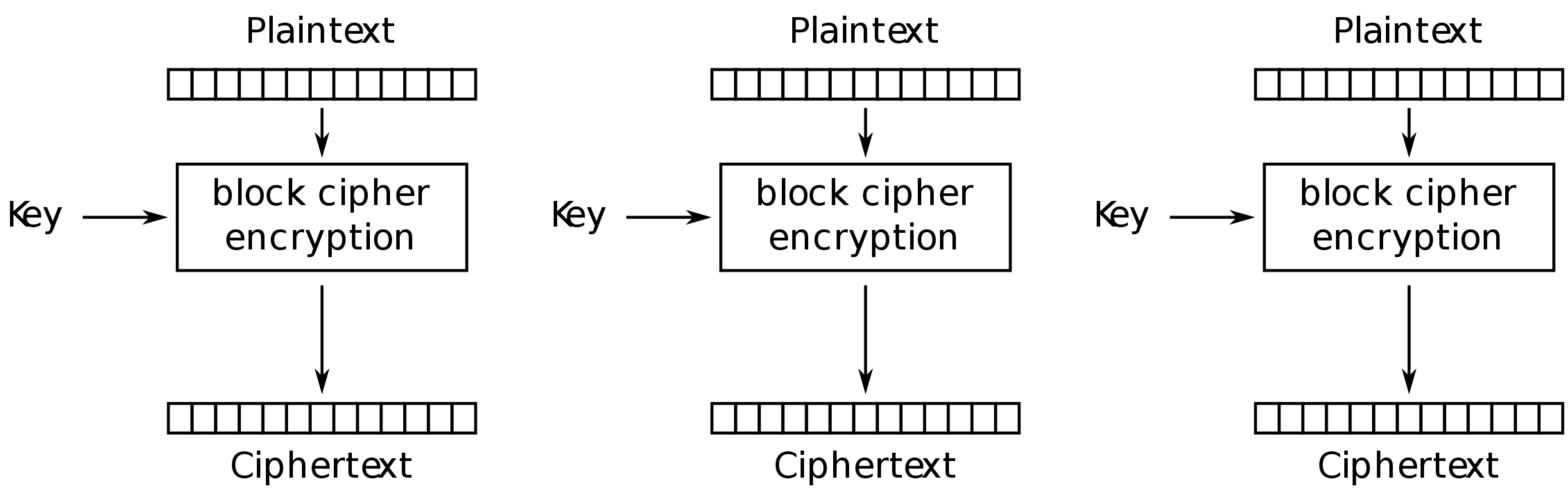
- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?
  - Several modes of operation for longer messages
- How do we deal with messages that are not block-aligned?

Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?
    - Several modes of operation for longer messages
- How do we deal with messages that are not block-aligned?
    - Must pad messages in a distinguishable way

Insecure block cipher usage:
ECB mode

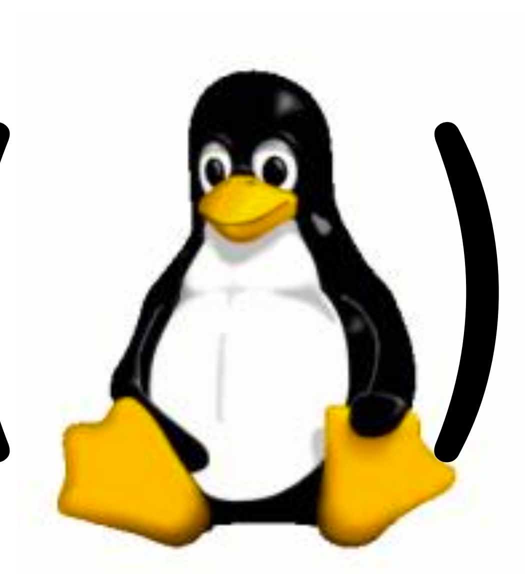
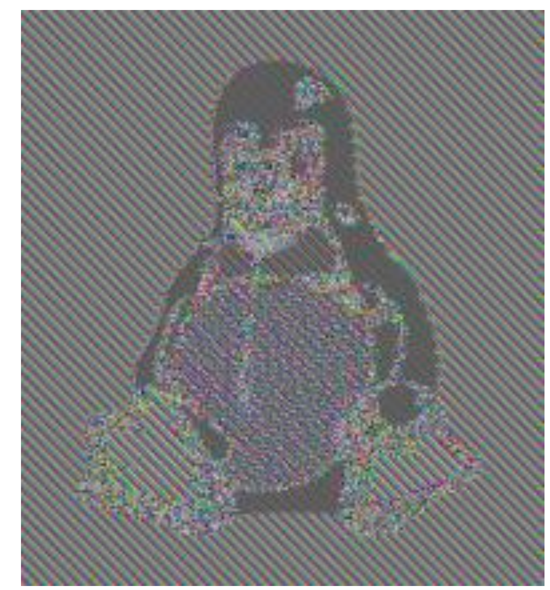
Insecure block cipher usage: ECB mode



Electronic Codebook (ECB) mode encryption

Source: wikipedia

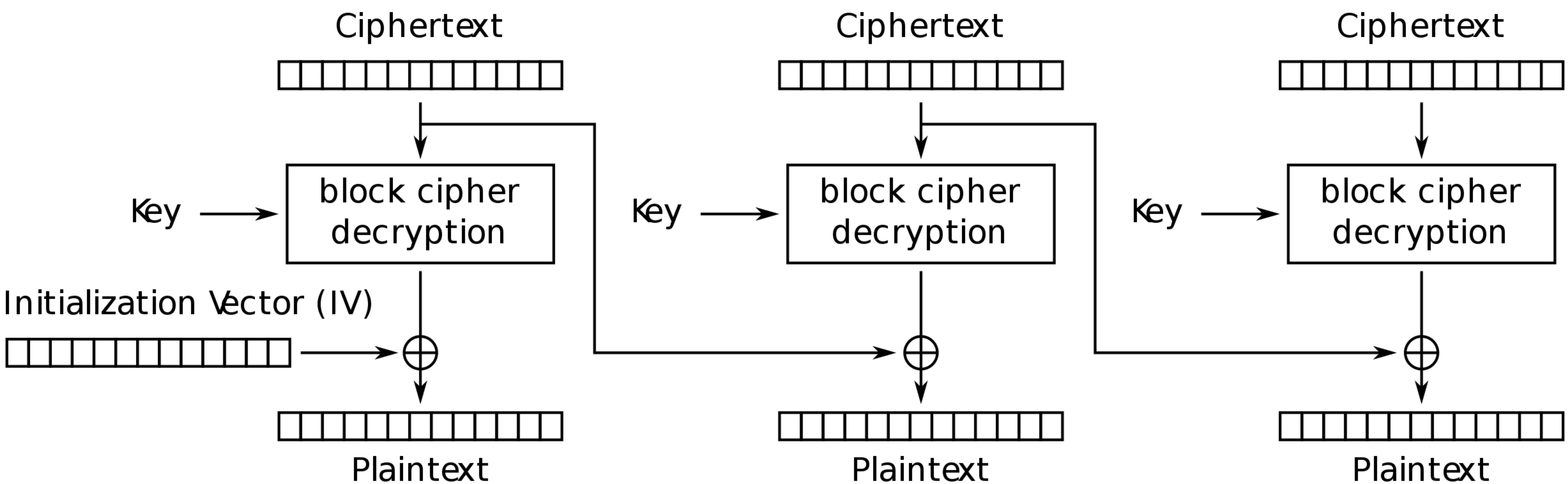
Why is ECB so bad?

$E_k($  $) =$ 

Source: wikipedia

Moderately secure usage:
CBC mode with random IV

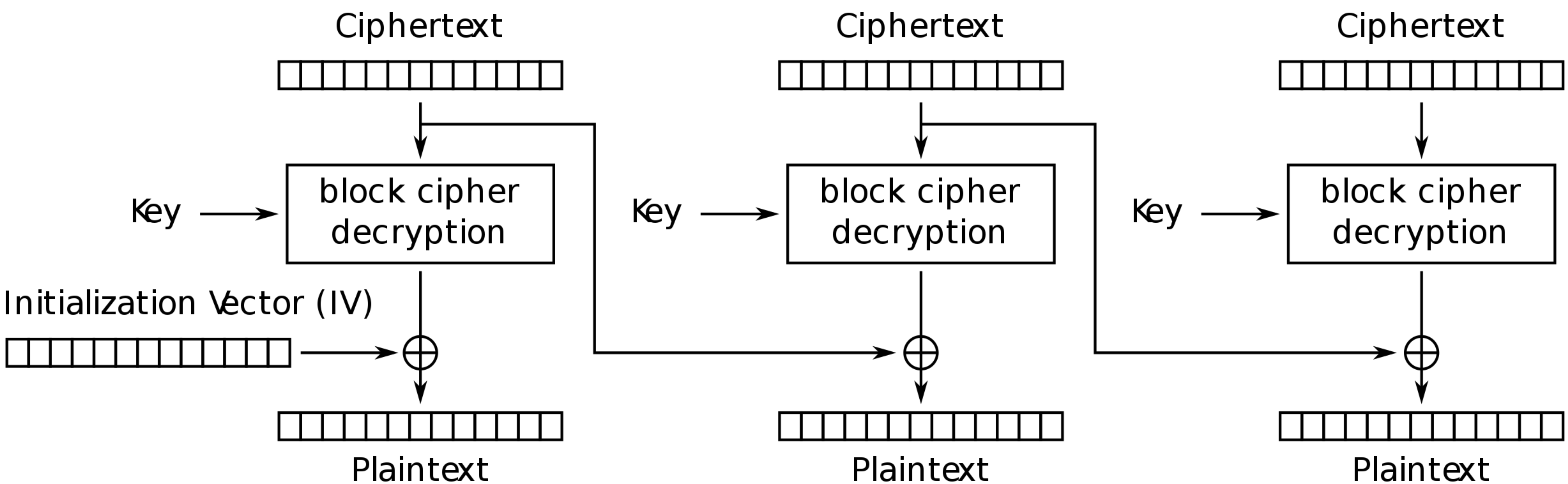
Moderately secure usage: CBC mode with random IV



Cipher Block Chaining (CBC) mode decryption

Source: wikipedia

Moderately secure usage: CBC mode with random IV



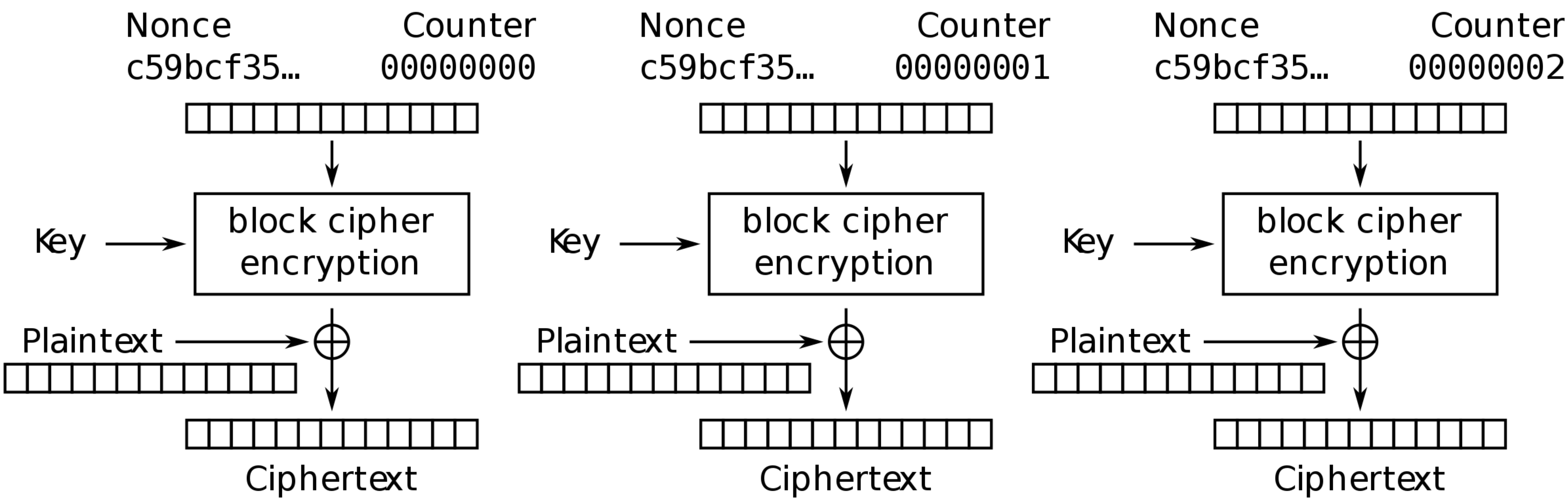
Cipher Block Chaining (CBC) mode decryption

Subtle attacks that abuse padding possible!

Source: wikipedia

Better block cipher usage:
CTR mode with random IV

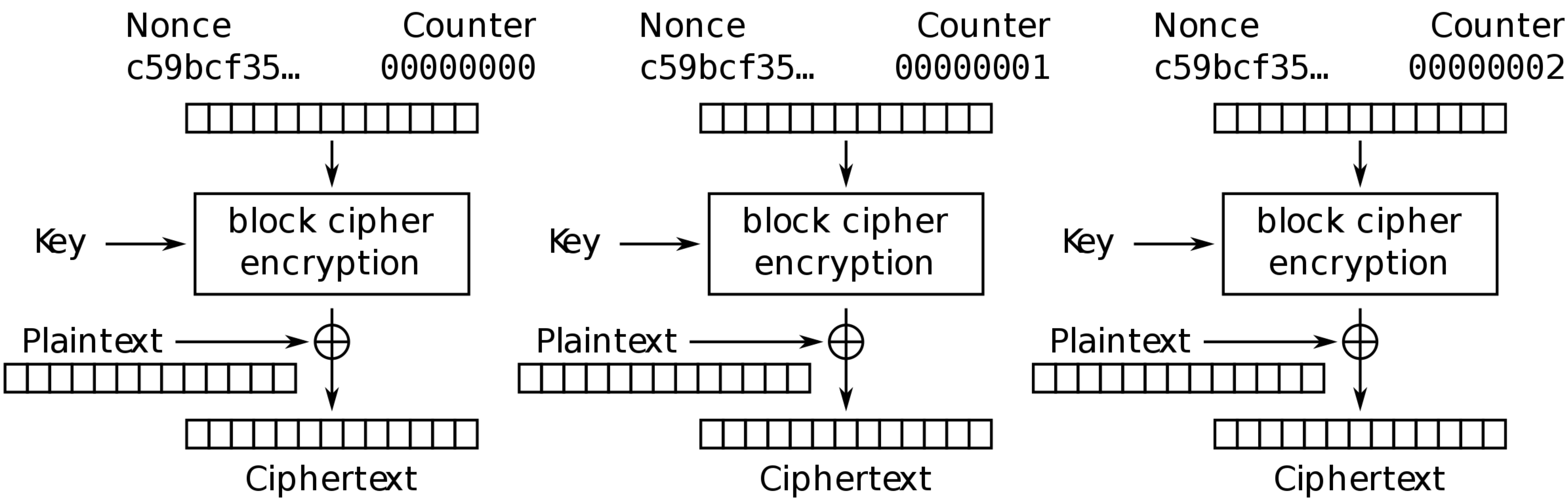
Better block cipher usage: CTR mode with random IV



Counter (CTR) mode encryption

Source: wikipedia

Better block cipher usage: CTR mode with random IV



Counter (CTR) mode encryption

Essentially use block cipher as stream cipher!

Source: wikipedia

What mode should you choose?

If your crypto library is making you choose a block cipher mode of operation, use a different library.

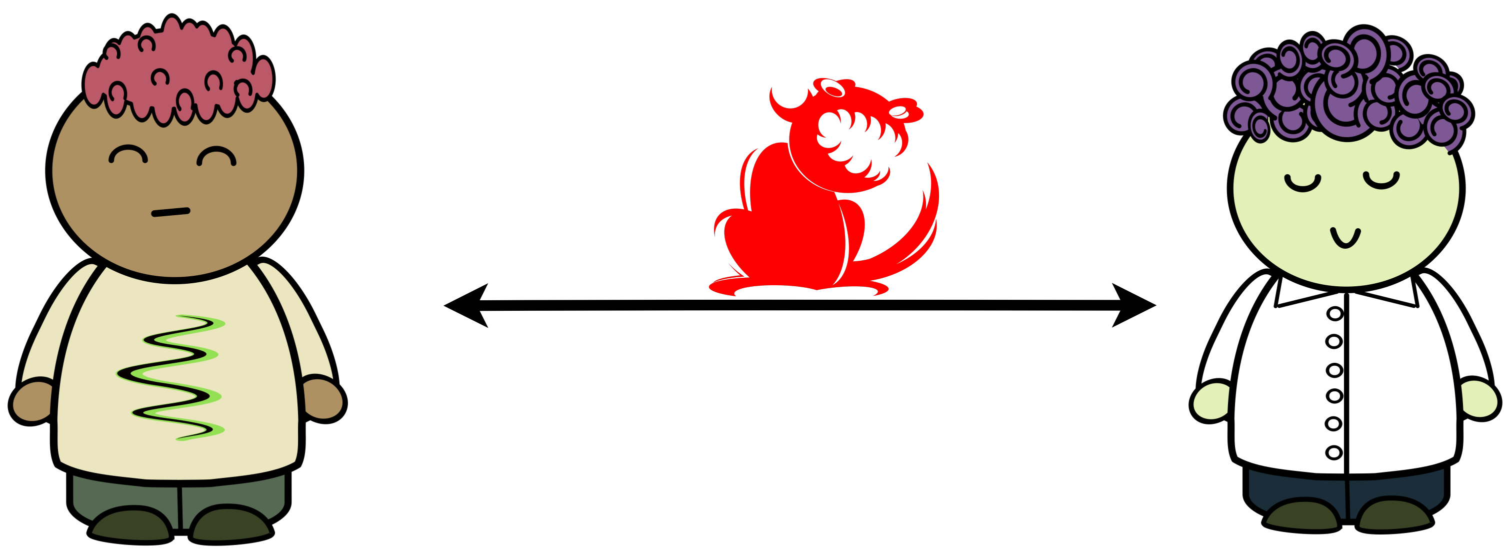
(Right answer: block cipher mode of operation can be built into an AEAD mode (end of lecture).)

What security do we get?

- All encryption breakable by brute force given enough knowledge about plaintext
  - Try to decrypt ciphertext with every possible key until a valid plaintext is found
- Attack complexity proportional to size of key space
  - 128-bit key requires 2^{128} decryption attempts

Chosen ciphertext attacks

- What if Eve can alter the ciphertexts sent between Alice and Bob?



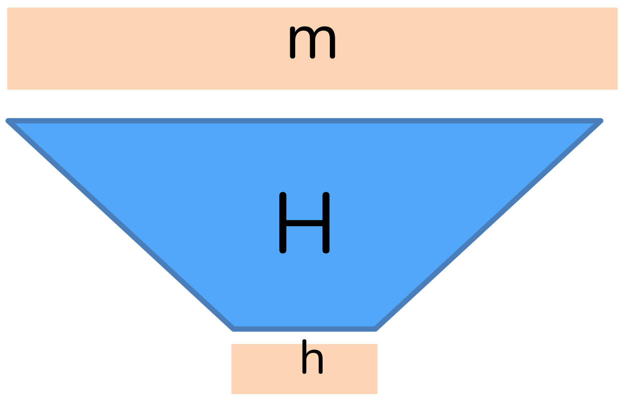
- Symmetric encryption alone is not enough to ensure security.
  - ➤ Need to protect *integrity of ciphertexts (and thus underlying encrypted messages)*

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

Hash Functions

- A (cryptographic) hash function maps arbitrary length input into a fixed-size string



$h = H(m)$

  - ➢ $|m|$ is arbitrarily large
  - ➢ $|h|$ is fixed, usually 128-512 bits

Hash Function Properties

- Finding a preimage is hard
  - Given h, find m such that $H(m)=h$
- Finding a second preimage is hard
  - Given m_1, find m_2 such that $H(m_1)=H(m_2)$
- Finding a collision is hard
  - Find m_1 and m_2 such that $H(m_1)=H(m_2)$

Hash function security

- A 128-bit hash function has 64 bits of security
  - Birthday bound: find collision in time 2^{64}

Real-world crypto: Hash functions

- Versioning systems (e.g., git)
  - Better than _1, _final, _really_final
- Sub-resource integrity
  - Integrity of files you include from CDN
- File download integrity
  - Make sure the thing you download is the thing you thought you were downloading
- Blockchain

blob: 41732ca416bc88034636778b4a76fa0ea03c4ebc (plain)

```
# Maintainer: Deian Stefan

pkgname=xwrits
pkgver=2.26
pkgrel=1
pkgdesc="reminds you to take wrist breaks "
arch=('any')
url="http://www.lcdf.org/xwrits/"
license=('GPLv2')
depends=()
makedepends=()
conflicts=()
source=("http://www.lcdf.org/xwrits/$pkgname-$pkgver.tar.gz")
sha256sums=('aaca4809b4cd62a627335ca14a231d4ab556fc872458bdb6fdbf6e76b103fed8')
sha512sums=('c8beeca957e41468d85819a7d6d4475c83a99735ff17d13d724658a421d1d3b9a15191ee8ab903104ab19b869a4832103dbe7d3ec2a9bf89ae95a7899e92f927')

build() {
  cd "$pkgname-$pkgver"
  ./configure --prefix=/usr
  make
}

check() {
  cd "$pkgname-$pkgver"
  make -k check
}

package() {
  cd "$pkgname-$pkgver"
  make DESTDIR="$pkgdir/" install
}
```

Popular broken hash gunctions

- MD5: Message Digest
  - Designed by Ron Rivest
  - Output: 128 bits
- SHA-1: Secure Hash Algorithm 1
  - Designed by NSA
  - Output: 160 bits

Hash functions

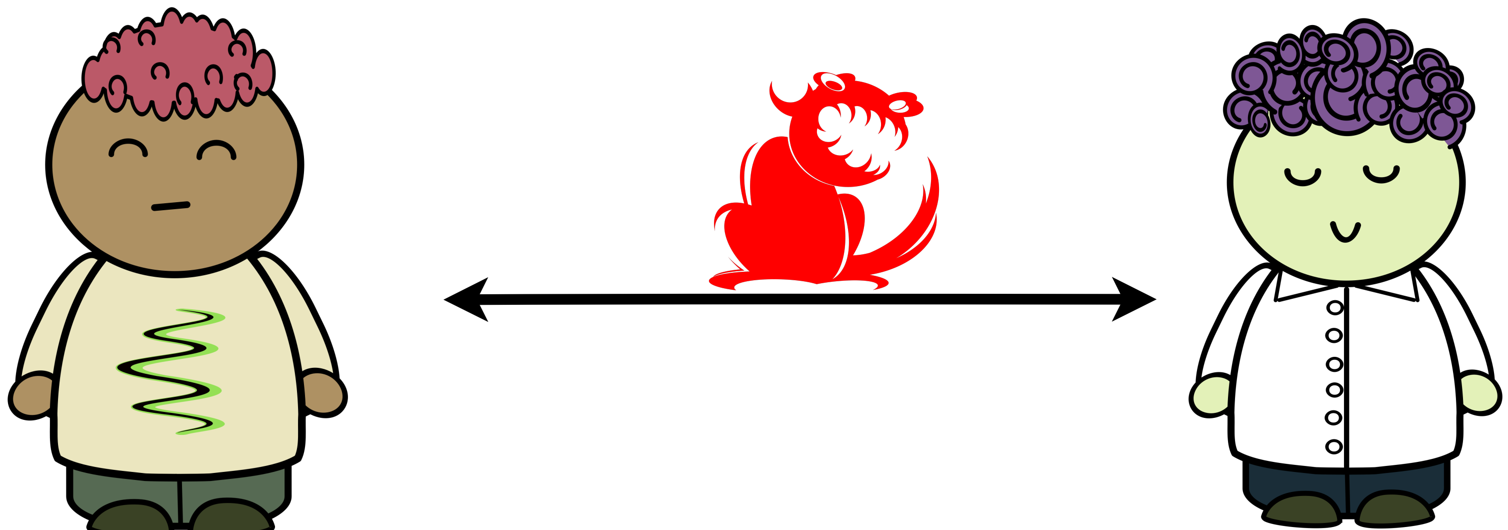
- SHA-2: Secure Hash Algorithm 2
  - Designed by NSA
  - Output: 224, 256, 384, or 512 bits
- SHA-3: Secure Hash Algorithm 3
  - Result of NIST SHA-3 contest
  - Output: arbitrary size
  - Replacement once SHA-2 broken

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication code
- Next time: asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

Chosen ciphertext attacks

- What if Eve can alter the ciphertexts sent between Alice and Bob?



- Symmetric encryption alone is not enough to ensure security.
  - ➤ Need to protect *integrity of ciphertexts (and thus underlying encrypted messages)*

MACs

- Validate message integrity based on shared secret
- MAC: Message Authentication Code
  - Keyed function using shared secret
  - Hard to compute function without knowing key

$$a=MAC_k(m)$$

HMAC construction

- HMAC: MAC based on hash function

  $$MAC_k(m) = H(\ k \oplus opad \parallel H(\ k \oplus ipad \parallel m\ )\ )$$

  - HMAC-SHA256: HMAC construction using SHA-256

Other MAC constructions

- In 2009, Flickr required API calls to use authentication token that looked like:

  MD5(secret || arg1=val1&arg2=val2&…)

- Is $MAC_k(m) = H(k \| m)$ a secure MAC?

Other MAC constructions

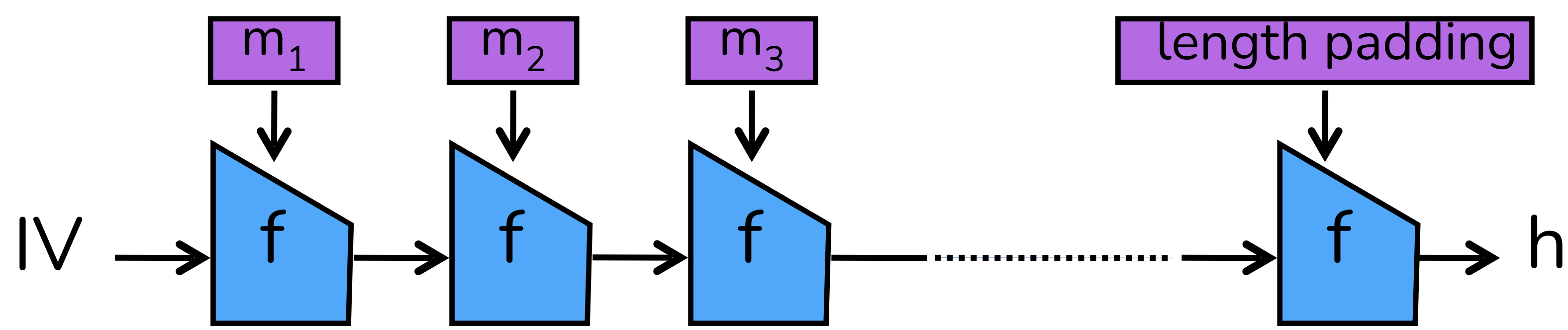
- In 2009, Flickr required API calls to use authentication token that looked like:

  MD5(secret || arg1=val1&arg2=val2&...)

- Is $MAC_k(m) = H(k \mid\mid m)$ a secure MAC?
  - No! If H is MD5, SHA1 or SHA2
  - Use HMAC!

Length extension attack

- Merkle-Damgård construction: hash function from collision-resistant compression function f

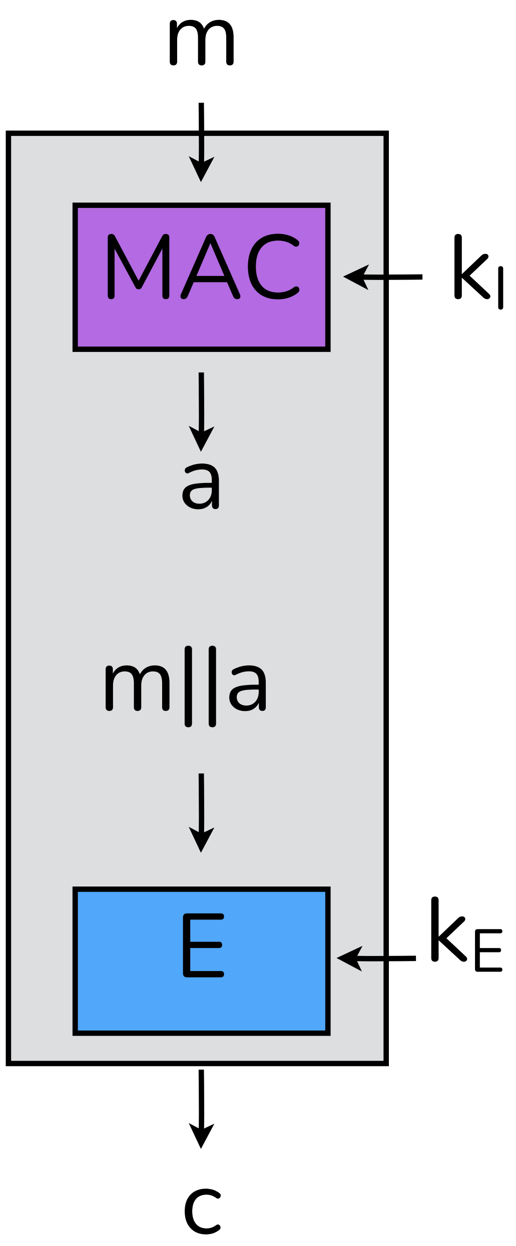


- Attacker that can observe $MAC_k(m)$ can forge $MAC_k(m||padding||r)$ for an r of their choice

Combining MAC with encryption

MAC then Encrypt (SSL)

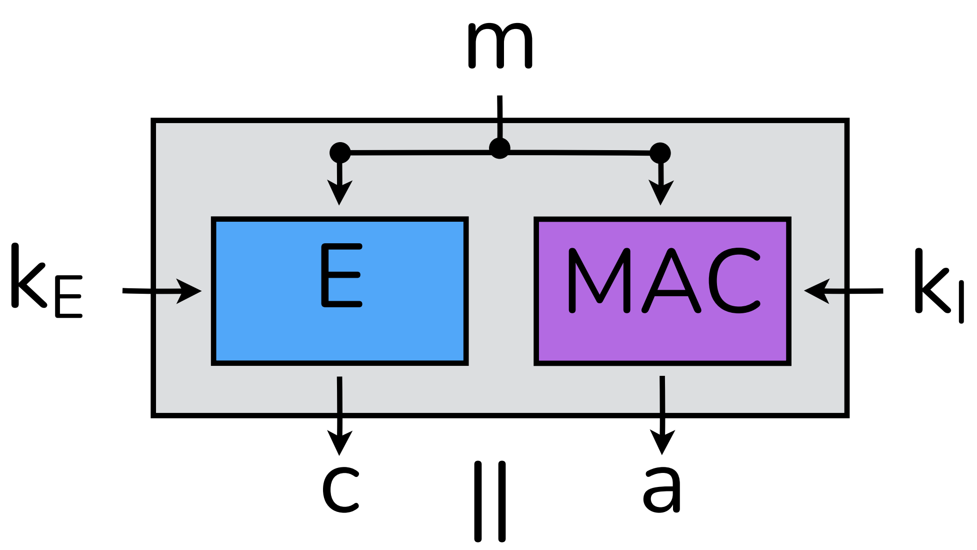
- Integrity for plaintext not ciphertext
- Issue: need to decrypt before you can verify integrity
- Hard to get right!



Combining MAC with encryption

Encrypt and MAC (SSH)

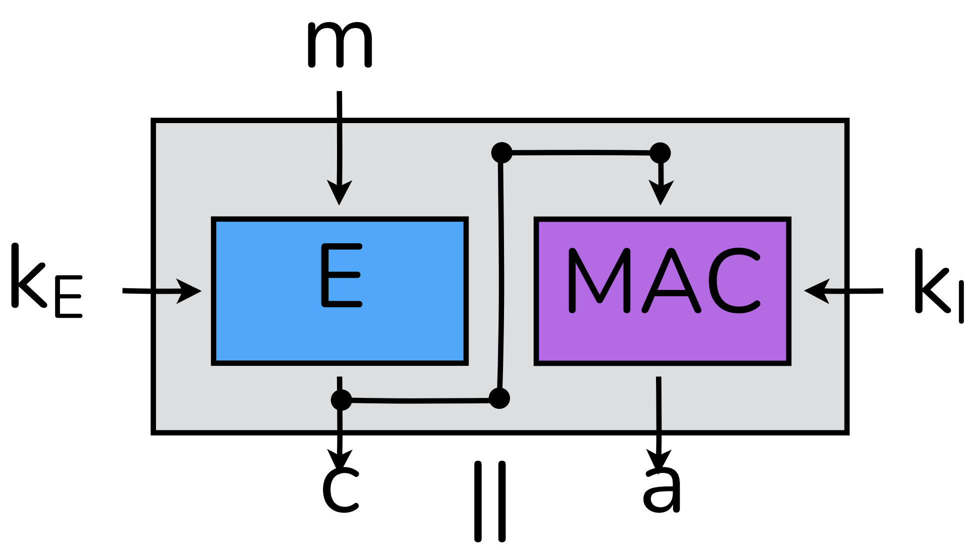
- Integrity for plaintext not ciphertext
- Issue: need to decrypt before you can verify integrity
- Hard to get right!



Combining MAC with encryption

Encrypt then MAC (IPSec)

- Integrity for plaintext and ciphertext
- Almost always right!



AEAD construction

- Authenticated Encryption with Associated Data
  - AES-GCM, AES-GCM-SIV
- Always use an authenticated encryption mode
  - Combines mode of operation with integrity protection/MAC in the right way

Good libraries have good defaults

Libsodium documentation

GitHub repository Download Quickstart Libhydrogen

Search...

- Introduction
- Installation
- Quickstart and FAQ
- Projects using libsodium
- Commercial support
- Bindings for other languages
- Usage
- Helpers
- Padding
- Secure memory
- Generating random data
- Secret-key cryptography
  - Authenticated encryption
  - Encrypted streams and file encryption
  - Encrypting a set of related messages

Authenticated encryption

Example

```c
#define MESSAGE ((const unsigned char *) "test")
#define MESSAGE_LEN 4
#define CIPHERTEXT_LEN (crypto_secretbox_MACBYTES + MESSAGE_LEN)

unsigned char key[crypto_secretbox_KEYBYTES];
unsigned char nonce[crypto_secretbox_NONCEBYTES];
unsigned char ciphertext[CIPHERTEXT_LEN];

crypto_secretbox_keygen(key);
randombytes_buf(nonce, sizeof nonce);
crypto_secretbox_easy(ciphertext, MESSAGE, MESSAGE_LEN, nonce, key);

unsigned char decrypted[MESSAGE_LEN];
if (crypto_secretbox_open_easy(decrypted, ciphertext, CIPHERTEXT_LEN, nonce, key) != 0)
    /* message forged! */
}
```