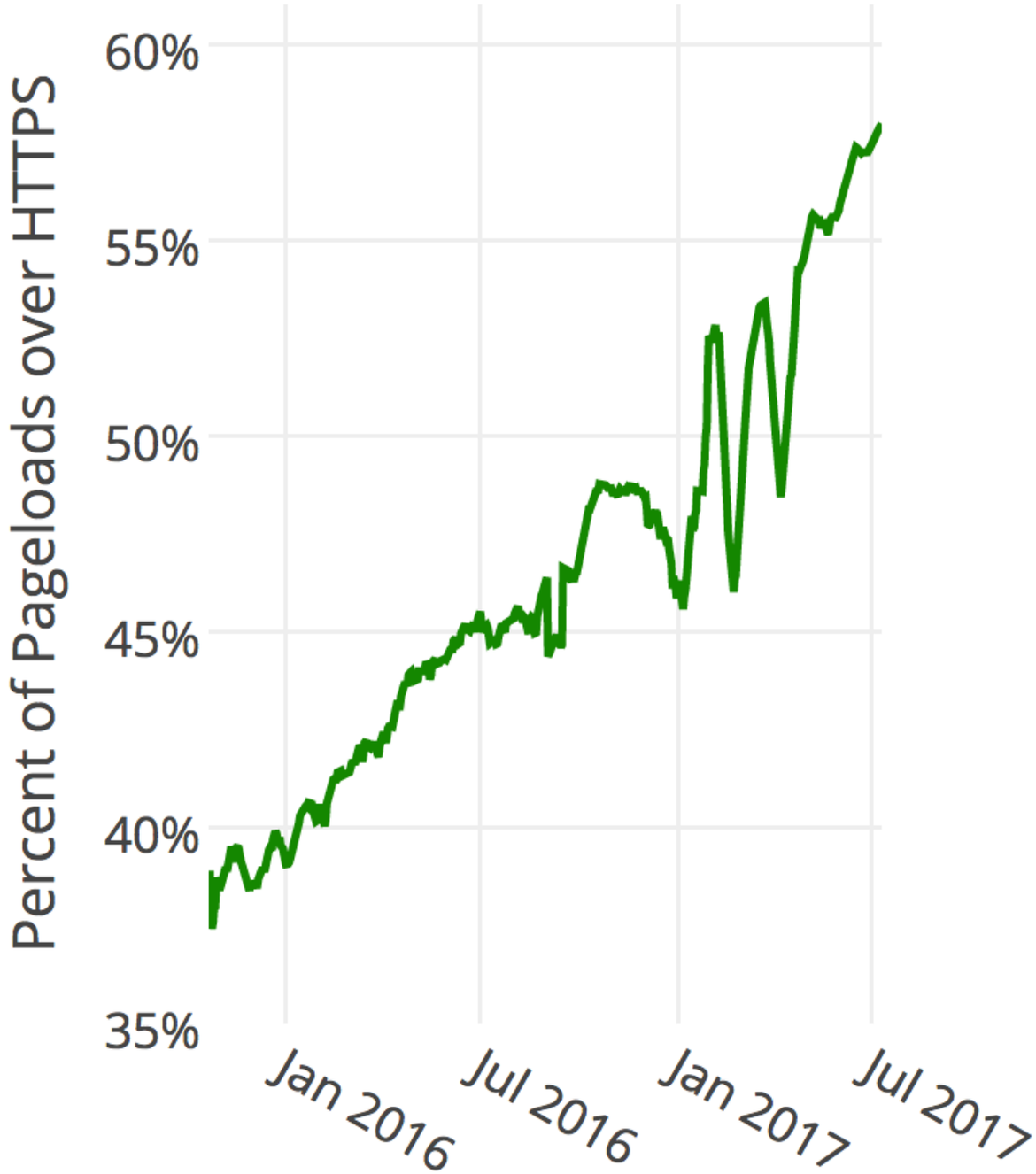
Breaking the Internet: How Cryptography Fails in Practice

Henry Corrigan-Gibbs
Dept. of Computer Science
Stanford University
July 28, 2017

Breaking the Internet: How Cryptography Fails in Practice

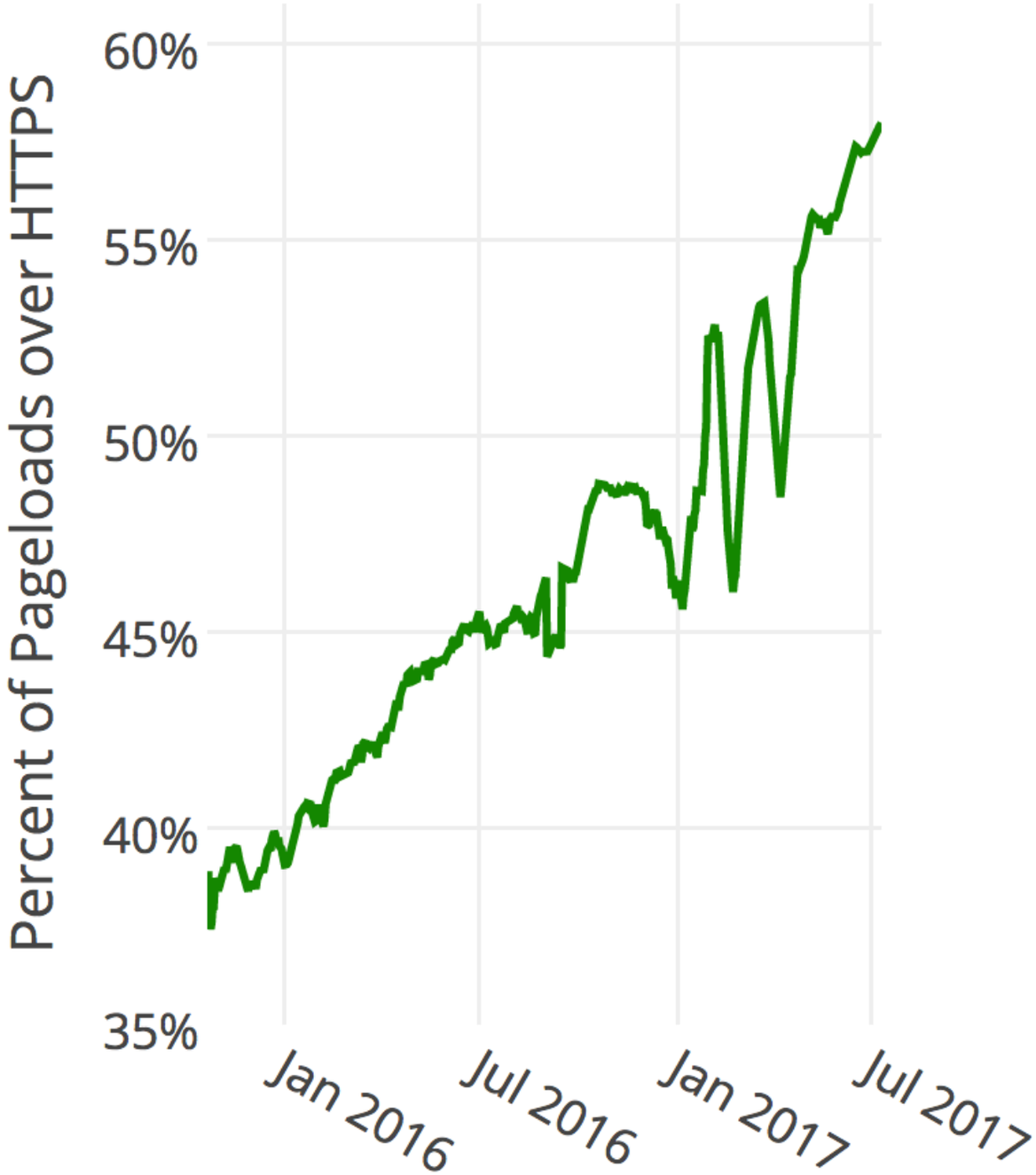
Henry Corrigan-Gibbs
Dept. of Computer Science
Stanford University
July 28, 2017

Encryption is becoming ubiquitous



Mozilla Telemetry

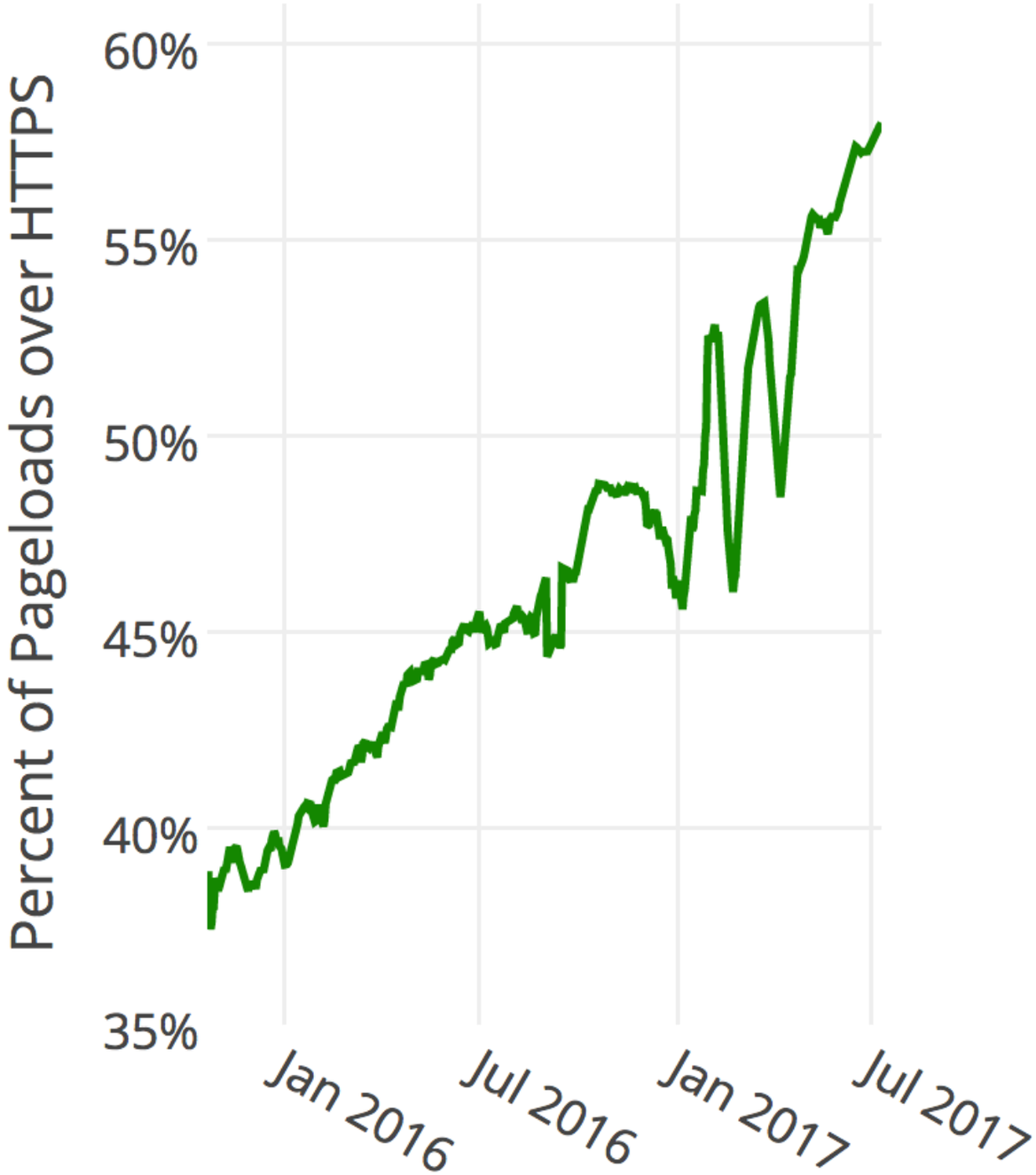
Encryption is becoming ubiquitous



Factors

Mozilla Telemetry

Encryption is becoming ubiquitous

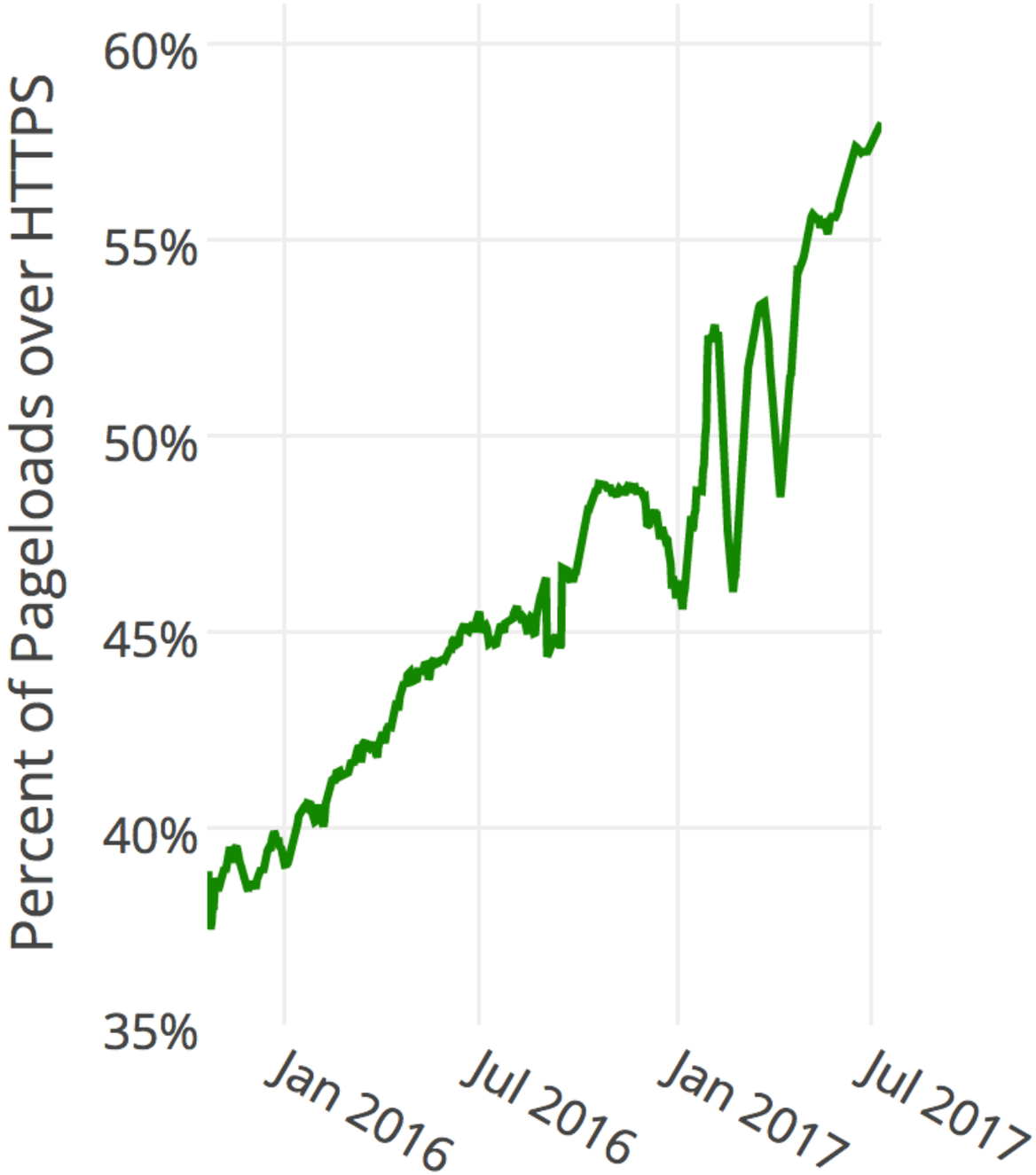


Factors

- WiFi

Mozilla Telemetry

Encryption is becoming ubiquitous

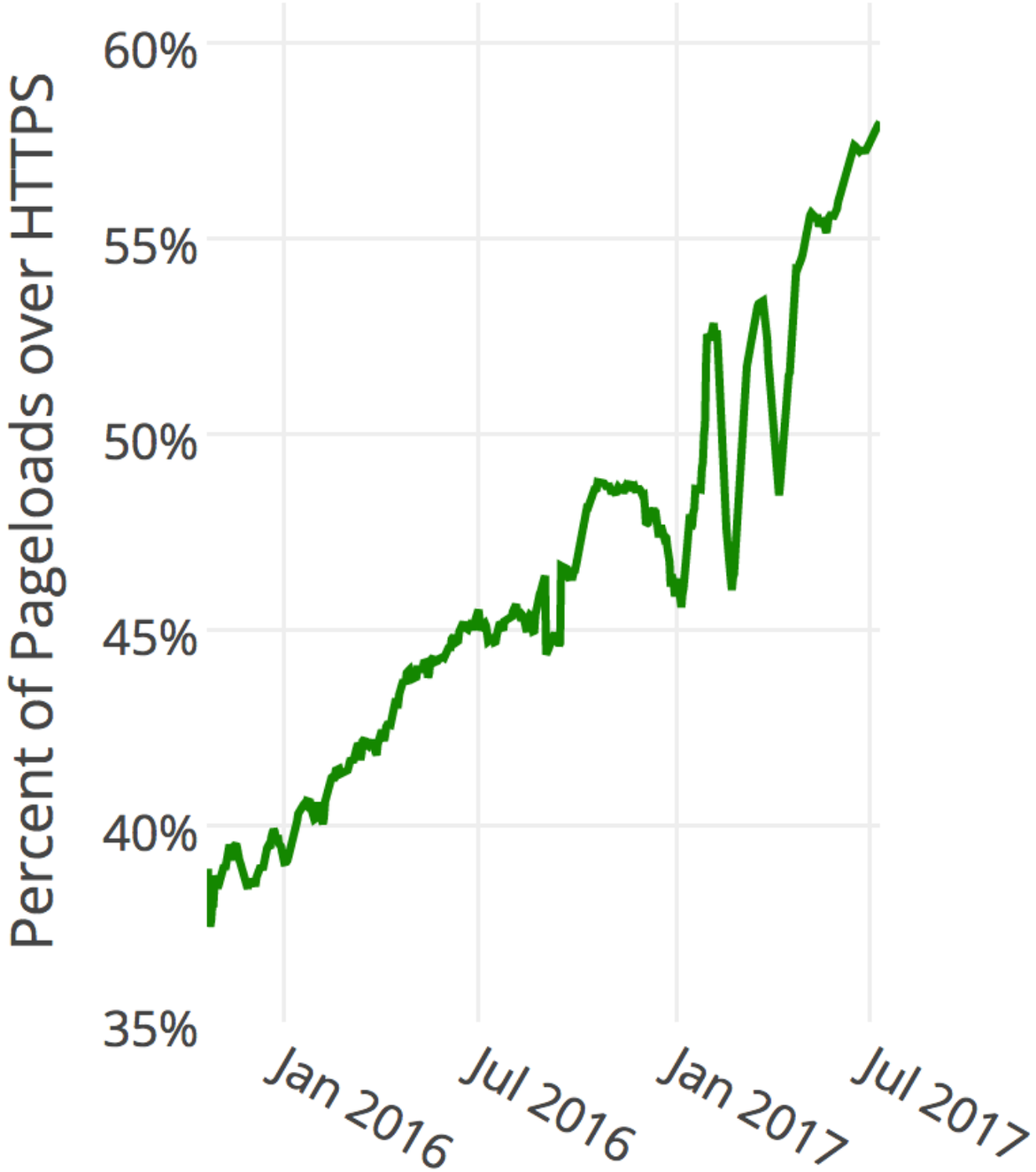


<u>Factors</u>

- WiFi
- Intel’s AES hardware

Mozilla Telemetry

Encryption is becoming ubiquitous

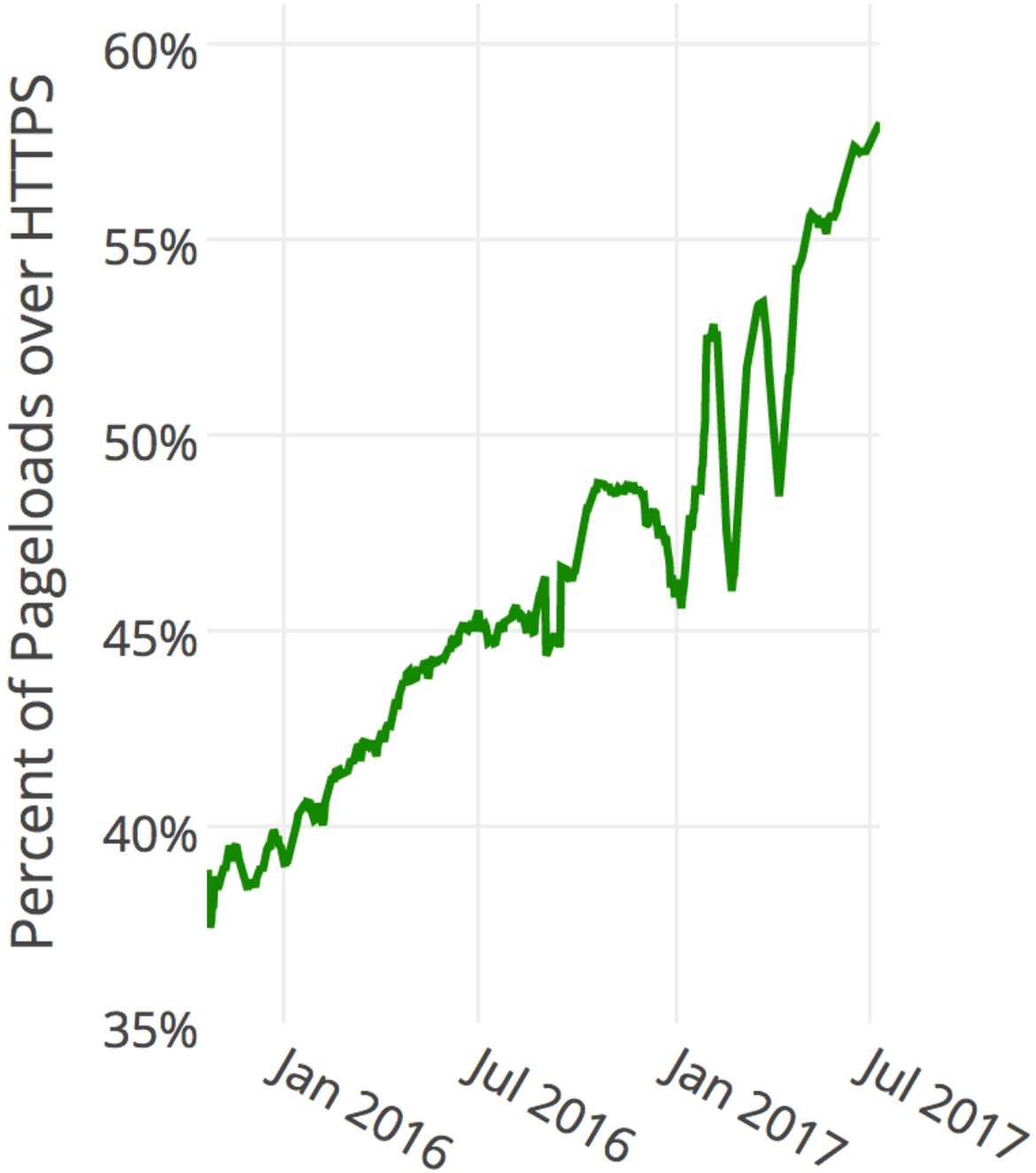


<u>Factors</u>

- WiFi
- Intel's AES hardware
- Coercion by Google

Mozilla Telemetry

Encryption is becoming ubiquitous

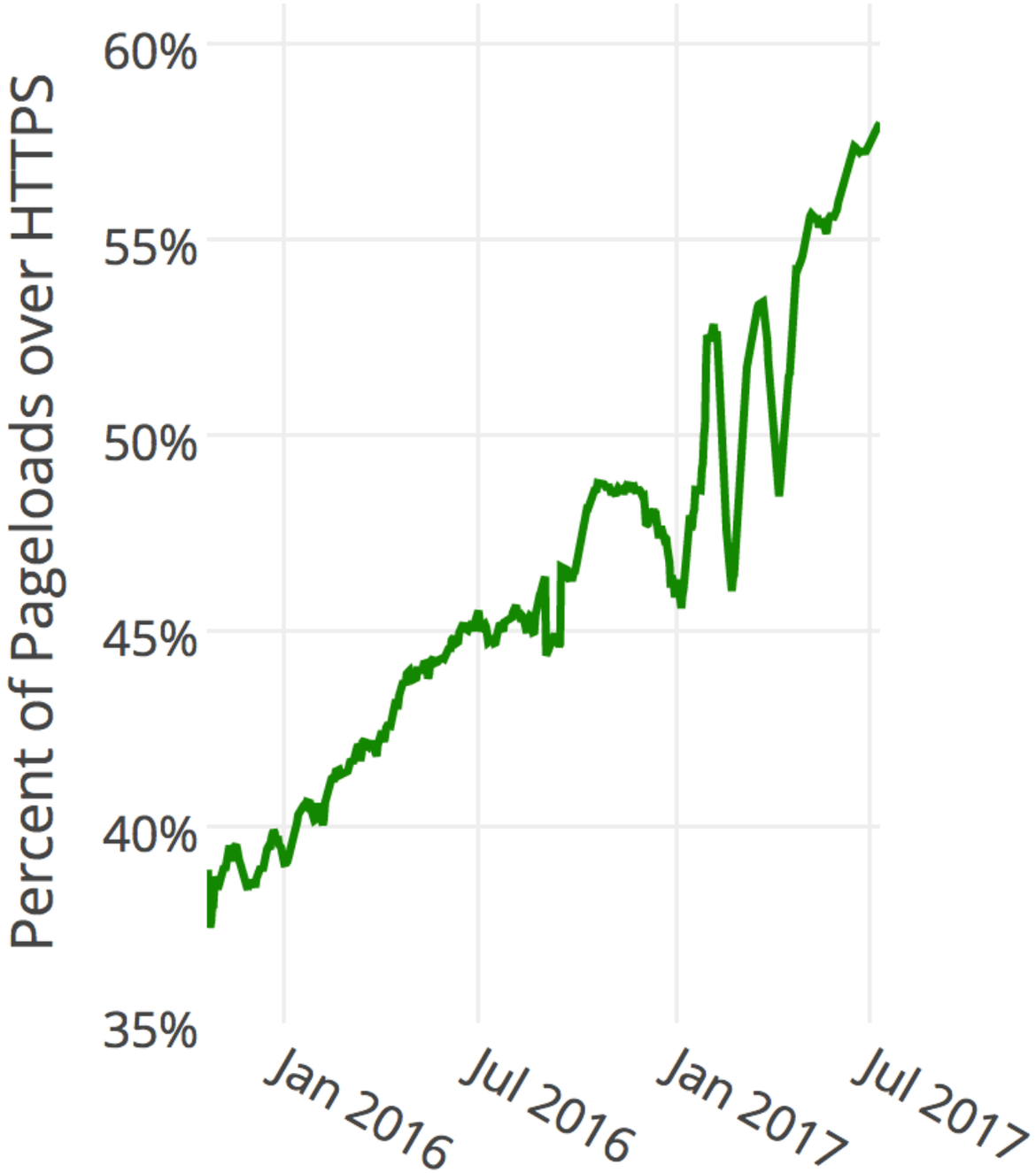


<u>Factors</u>

- WiFi
- Intel’s AES hardware
- Coercion by Google
- Better tools
  -> Let’s encrypt, etc.

Mozilla Telemetry

Encryption is becoming ubiquitous

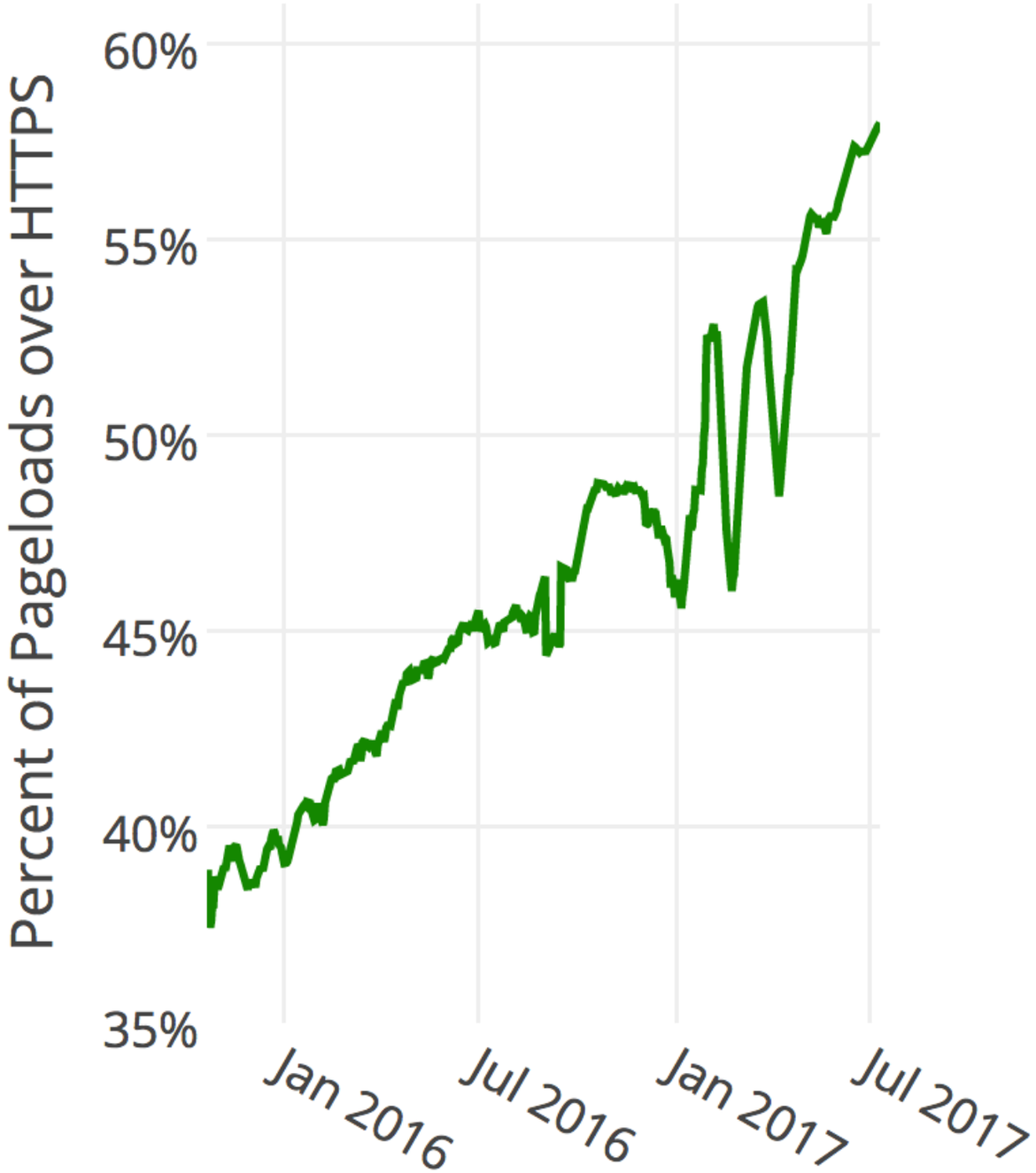


<u>Factors</u>

- WiFi
- Intel's AES hardware
- Coercion by Google
- Better tools
  -> Let's encrypt, etc.
- Snowden revelations

Mozilla Telemetry

Encryption is becoming ubiquitous



<u>Factors</u>

- WiFi
- Intel's AES hardware
- Coercion by Google
- Better tools
  -> Let's encrypt, etc.
- Snowden revelations
- Cyberattacks?

Mozilla Telemetry

Without encryption

Web browser

bank.com

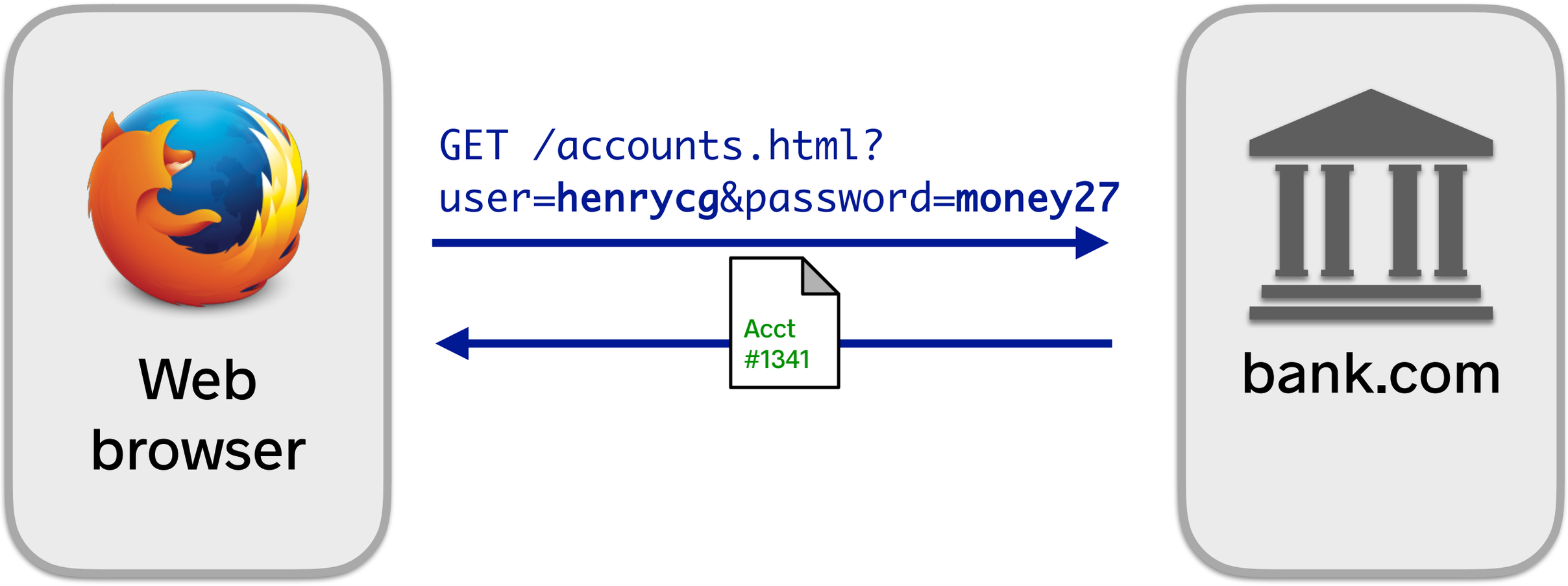
Without encryption

Web browser

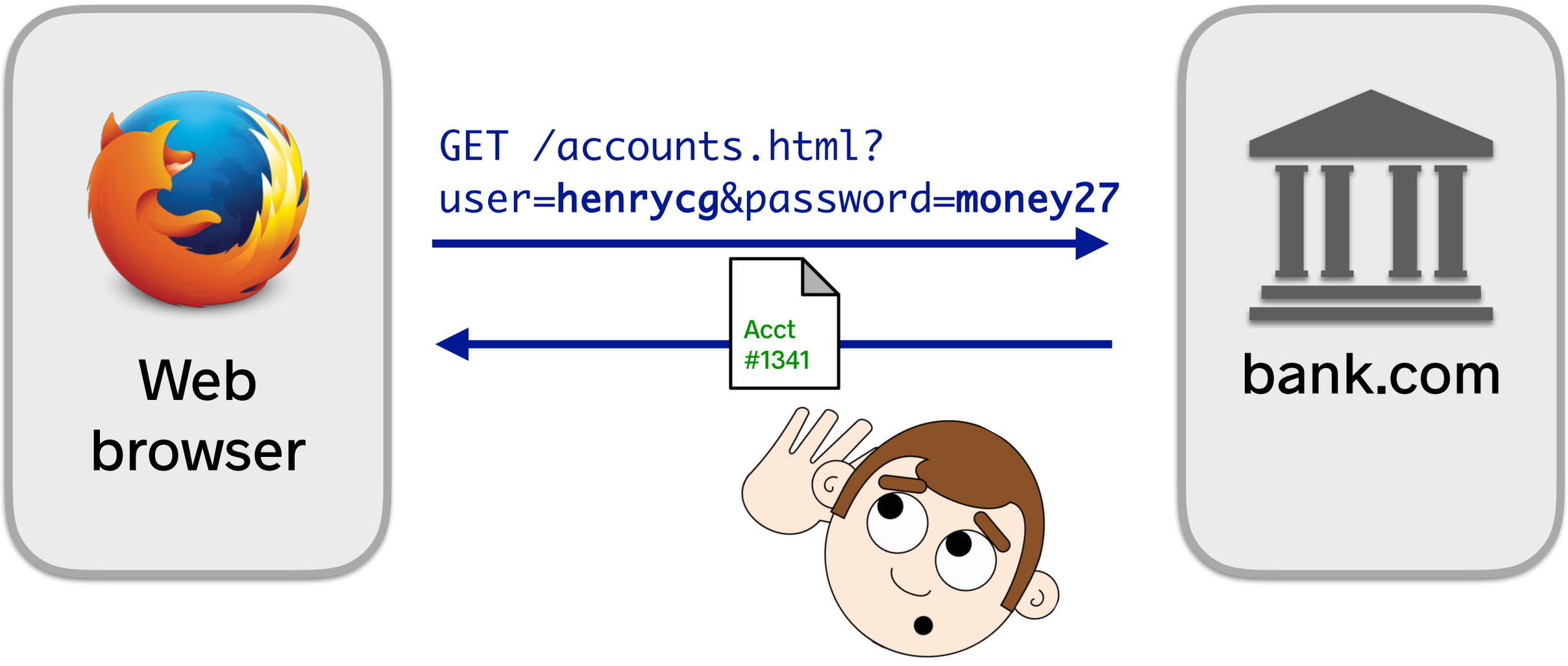
GET /accounts.html?
user=**henrycg**&password=**money27**

bank.com

Without encryption

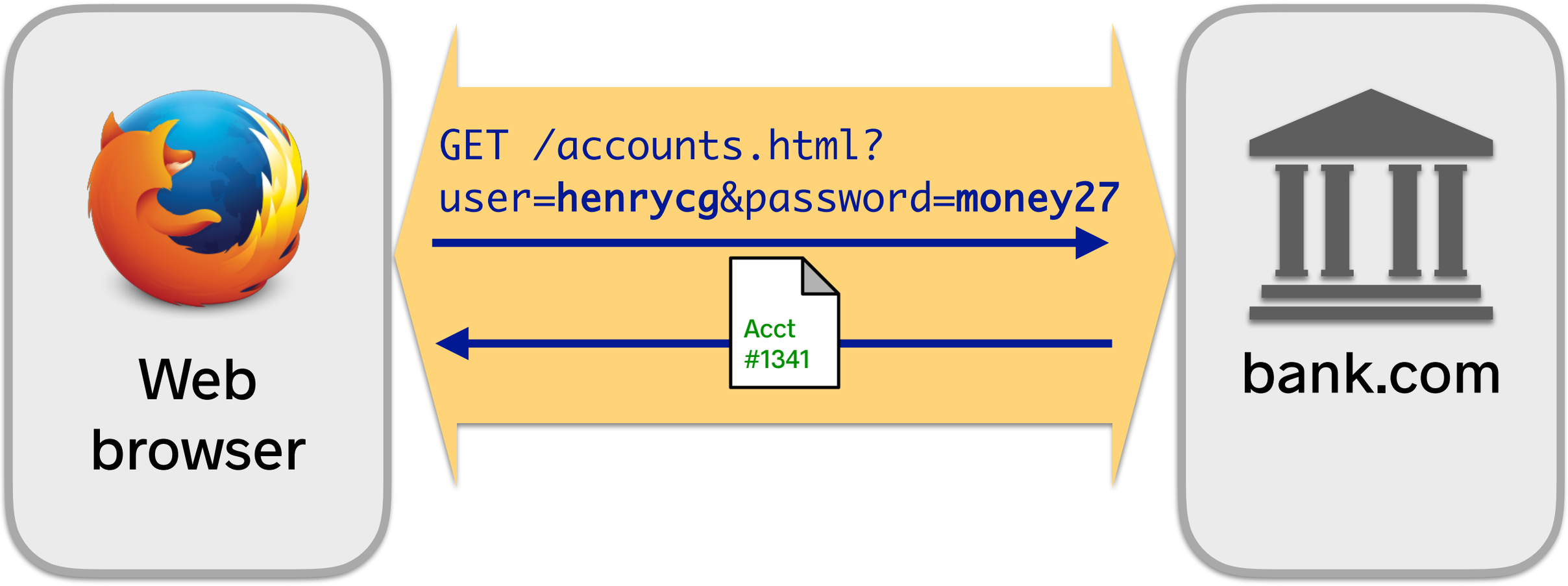


Without encryption

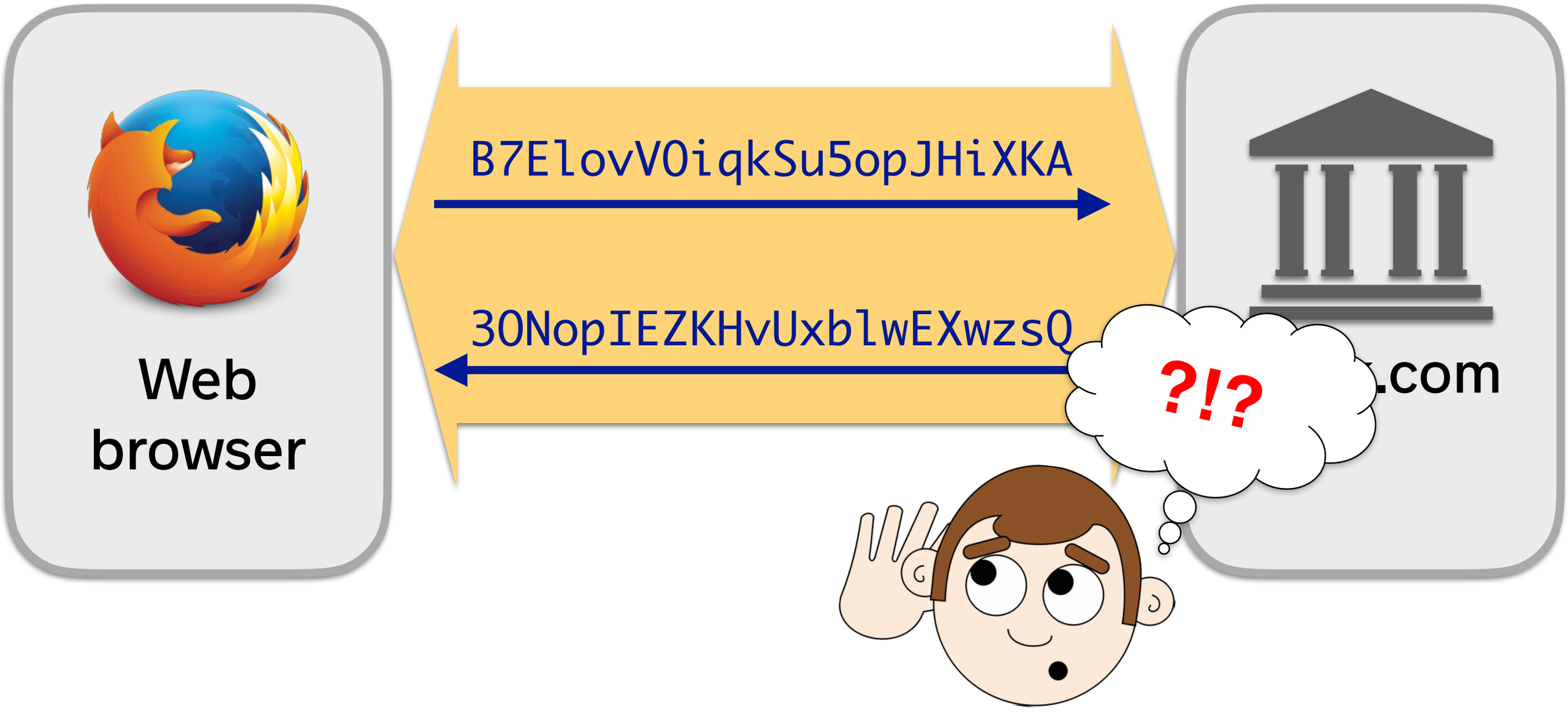


- Internet service provider
- Person on your WiFi network
- Government running fiber link

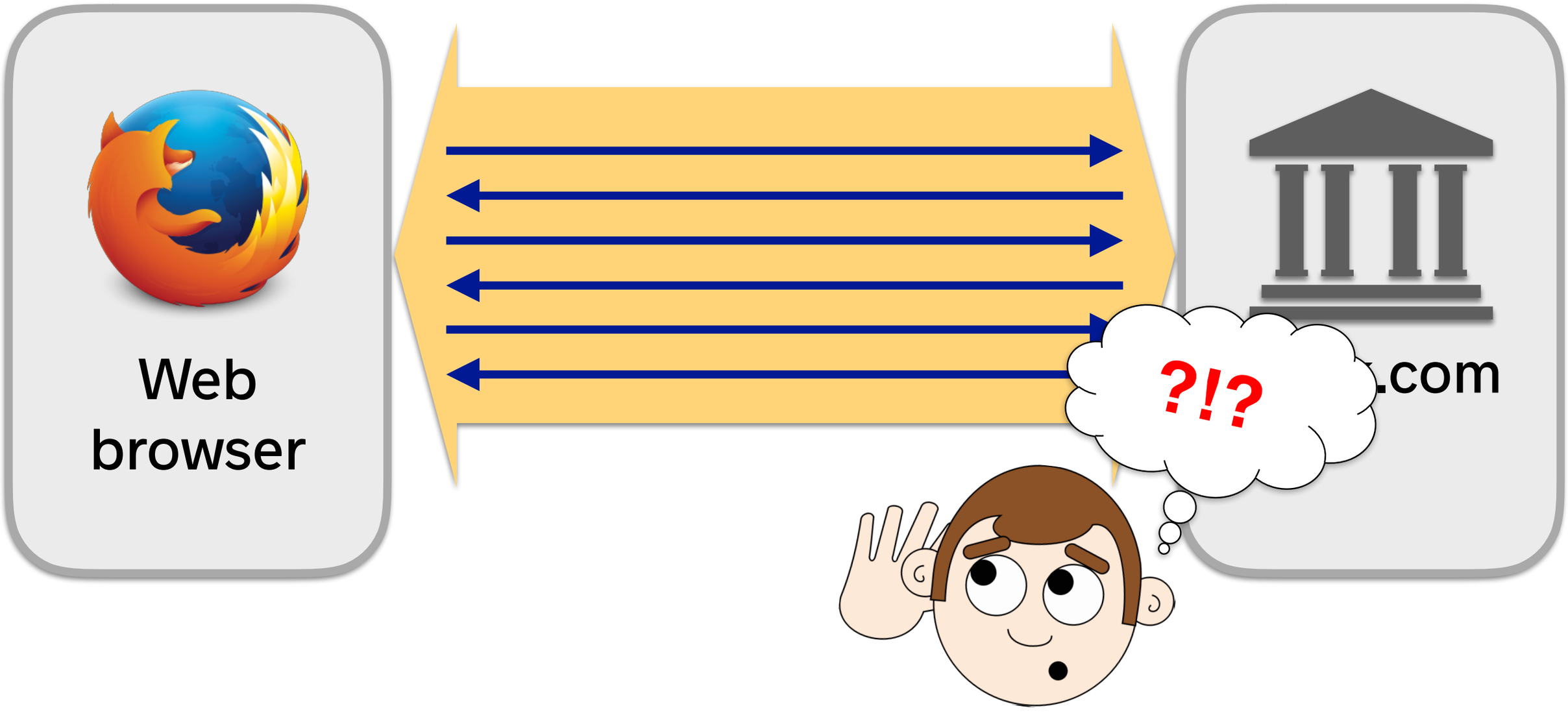
With encryption



With encryption

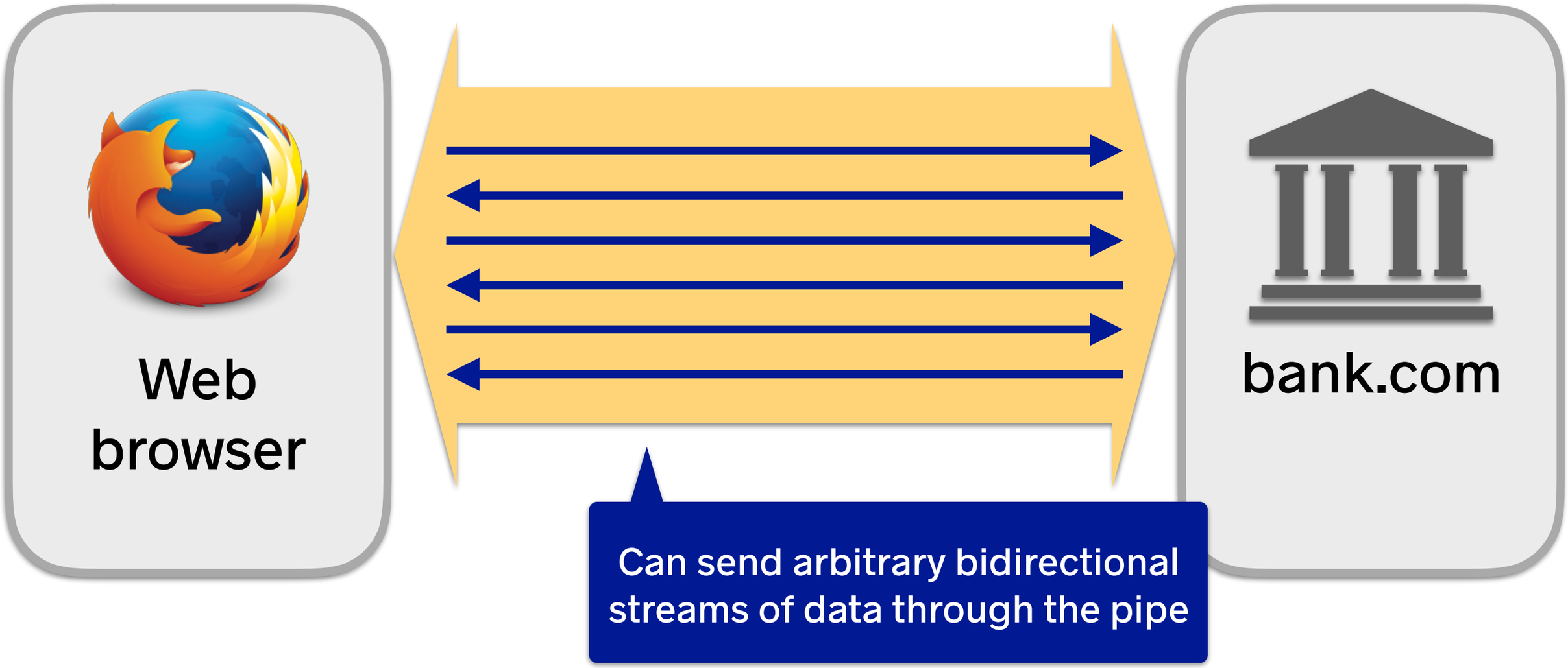


Want an “encrypted pipe” to your bank



*Precisely specifying the security property we want is not so easy...

Want an “encrypted pipe” to your bank



*Precisely specifying the security property we want is not so easy...

Goals of this talk

1. Explain ideas behind link encryption (TLS).
2. Show why it’s hard to get right.
3. Highlight open problems and research directions.

Goals of this talk

Will focus on the concepts, rather than the protocol specifics.

1. Explain ideas behind link encryption (TLS).
2. Show why it’s hard to get right.
3. Highlight open problems and research directions.

Goals of this talk

1. Explain ideas behind link encryption (TLS).
2. Show why it’s hard to get right.
3. Highlight open problems and research directions.

Goals of this talk

1. Explain ideas behind link encryption (TLS).
2. Show why it’s hard to get right.
3. Highlight open problems and research directions.

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an "encrypted pipe" abstraction on the Internet

- When you visit https://www.stanford.edu/, your traffic is flowing over TLS

- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an "encrypted pipe" abstraction on the Internet

- When you visit https://www.stanford.edu/, your traffic is flowing over TLS

- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS

- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

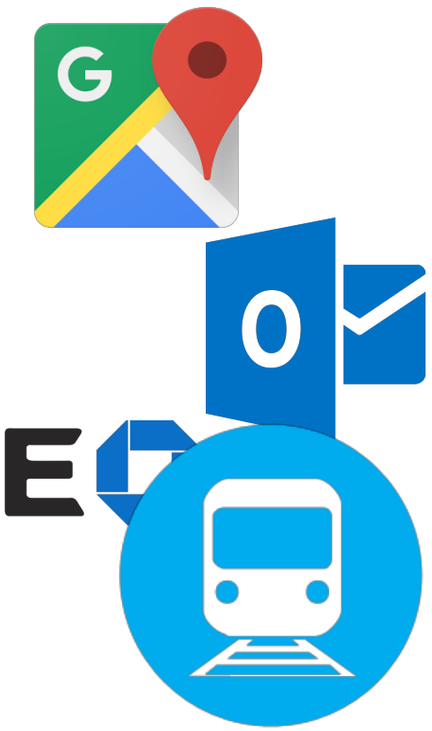
- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS
- TLS is used everywhere! Not just in the browser…
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS
- TLS is used everywhere! Not just in the browser…
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS
- TLS is used everywhere! Not just in the browser…
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)



Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS
- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet
- When you visit https://www.stanford.edu/, your traffic is flowing over TLS
- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Transport layer security (TLS)

- TLS (formerly SSL) is the primary protocol that implements an “encrypted pipe” abstraction on the Internet

- When you visit https://www.stanford.edu/, your traffic is flowing over TLS

- TLS is used everywhere! Not just in the browser...
  - Phone apps to backend servers
  - Mail client (e.g., Outlook) to mail server
  - ATM to bank
  - Smart car to telemetry service
  - Laptop to software update server
  - Ground station to expensive satellite (?)

Problem overview

Problem overview

Problem overview

- Endpoints have a shared secret key k (e.g., a 128-bit string)

Problem overview

- Endpoints have a shared secret key k (e.g., a 128-bit string)

Problem overview

- Endpoints have a shared secret key k (e.g., a 128-bit string)
- Have a “block cipher” that encrypts/decrypts 128-bit messages

```
E(k, m)    -> c        // Encrypt message m
D(k, c)    -> m        // Decrypt ciphertext c
```

Problem overview

Fixed size

- Endpoints have a shared secret key k (e.g., a 128-bit stri
- Have a “block cipher” that encrypts/decrypts 128-bit messages

```
E(k, m)    -> c        // Encrypt message m
D(k, c)    -> m        // Decrypt ciphertext c
```

Problem overview

- Endpoints have a shared secret key k (e.g., a 128-bit string)
- Have a “block cipher” that encrypts/decrypts 128-bit messages

```
E(k, m)    -> c        // Encrypt message m
D(k, c)    -> m        // Decrypt ciphertext c
```

Problem overview

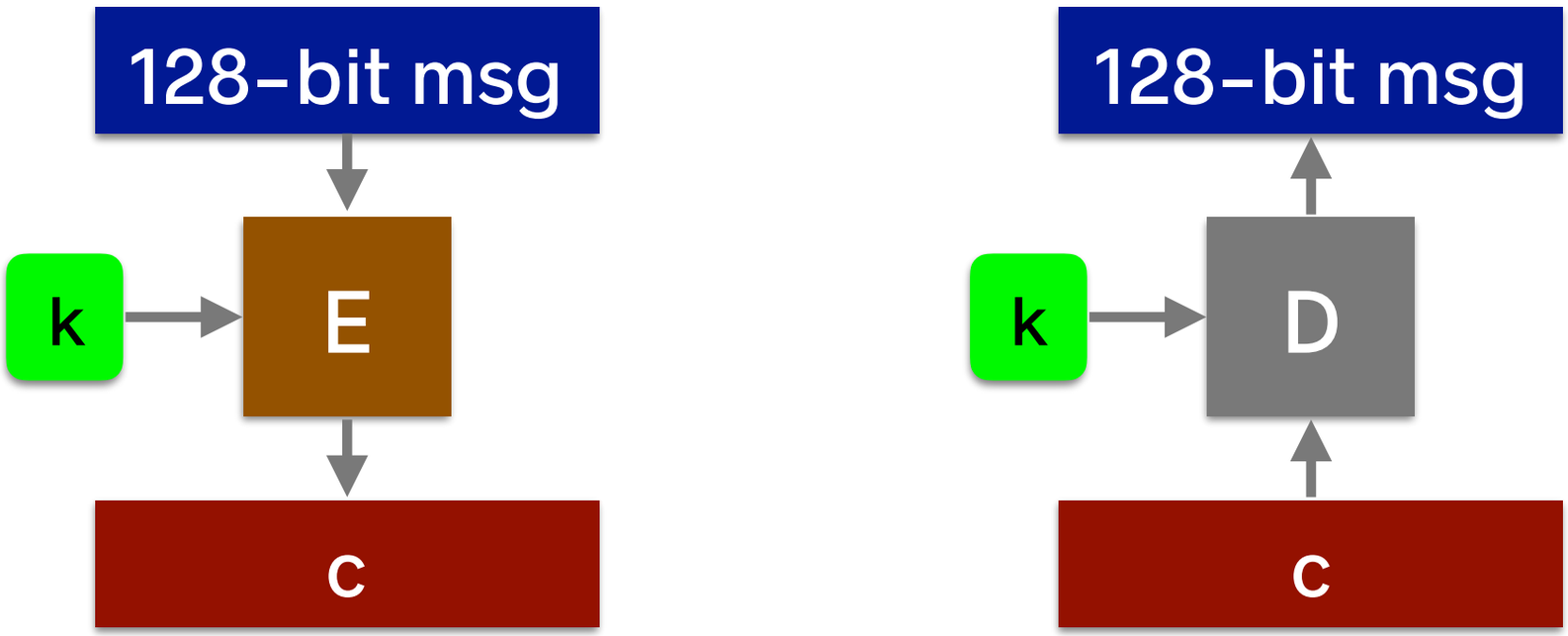
- Endpoints have a shared secret key k (e.g., a 128-bit string)
- Have a “block cipher” that encrypts/decrypts 128-bit messages

  E(k, m) -> c // Encrypt message m

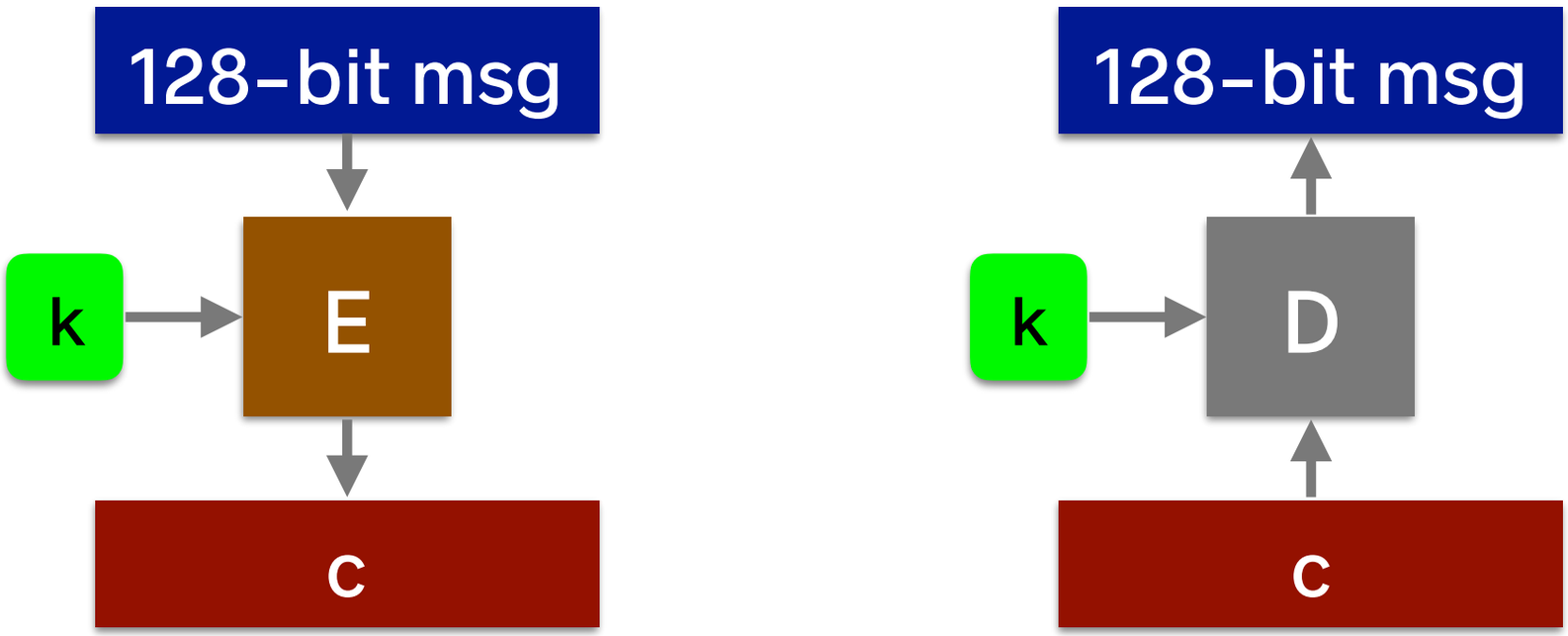
  D(k, c) -> m // Decrypt ciphertext c
- Want to build a scheme that encrypts *arbitrary length* messages

  — Web pages, Netflix movies, software updates, etc.

Block Cipher



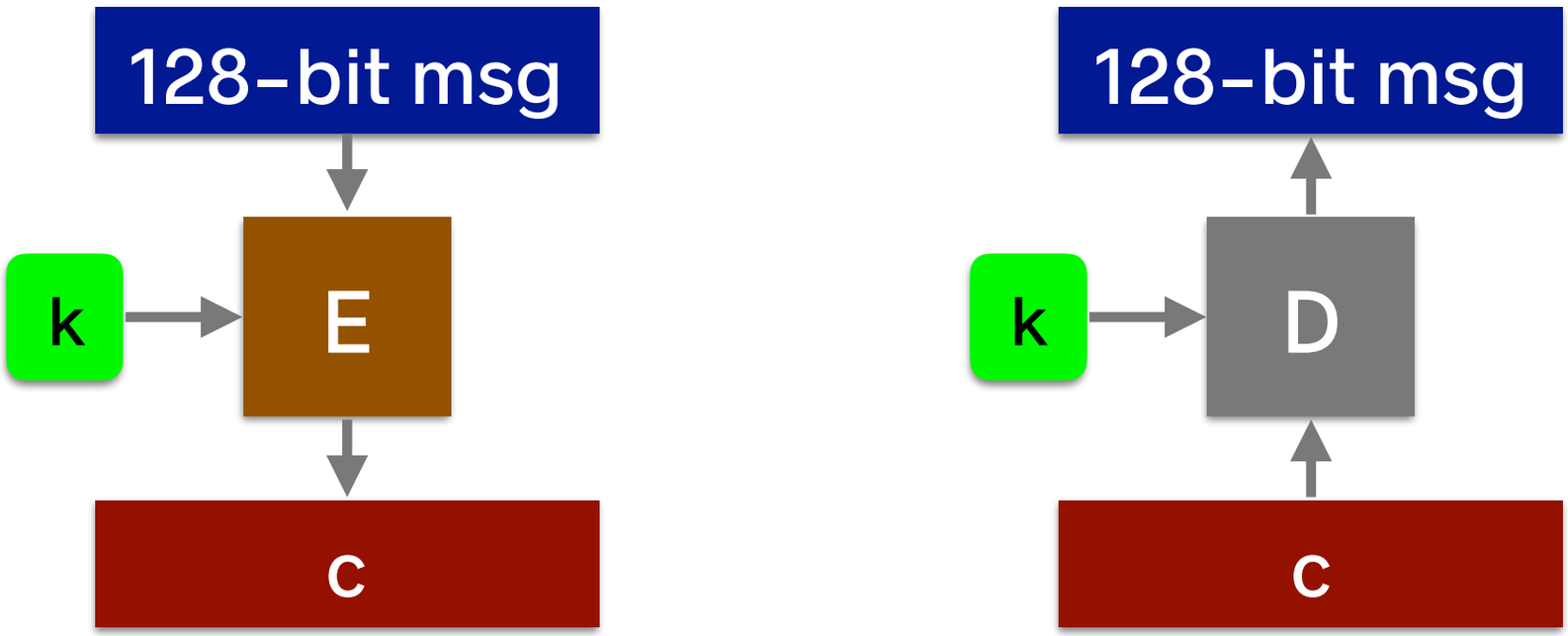
Block Cipher



Correctness. For all keys k and messages m:

$$D(k, E(k, m)) = m$$

Block Cipher



Correctness. For all keys k and messages m:

$$D(k, E(k, m)) = m$$

Security. For random key k, for all messages m,

$$c = E(k, m) \text{ “looks like random string”}$$

TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

- Even if the adversary gets to tamper with network traffic,

TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

- Even if the adversary gets to tamper with network traffic,
- Even if the adversary gets to choose a part of the message, and

TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

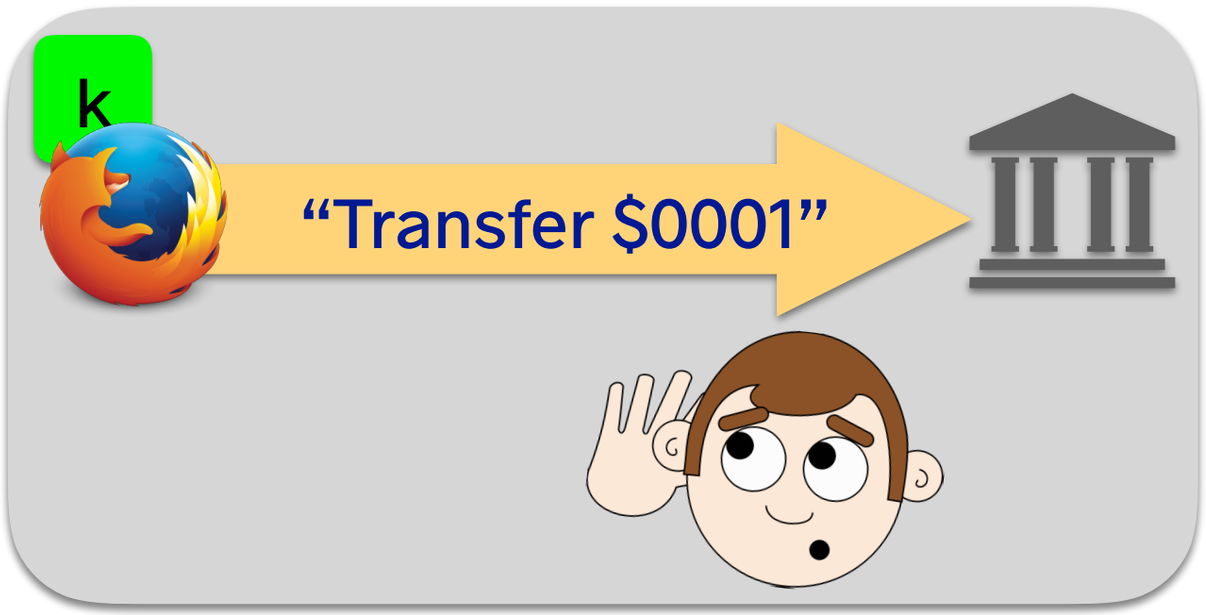
- Even if the adversary gets to tamper with network traffic,
- Even if the adversary gets to choose a part of the message, and
- Even if the client sends the same message many times.

TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

- Even if the adversary gets to tamper with network traffic,
- Even if the adversary gets to choose a part of the message, and
- Even if the client sends the same message many times.

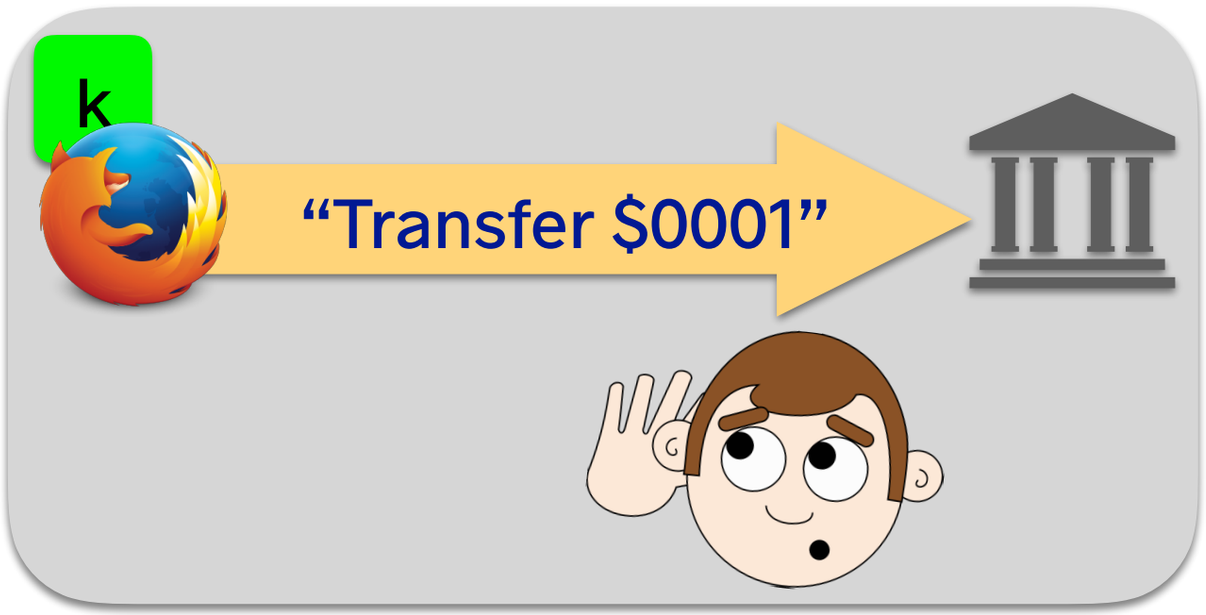


TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

- Even if the adversary gets to tamper with network traffic,
- Even if the adversary gets to choose a part of the message, and
- Even if the client sends the same message many times.

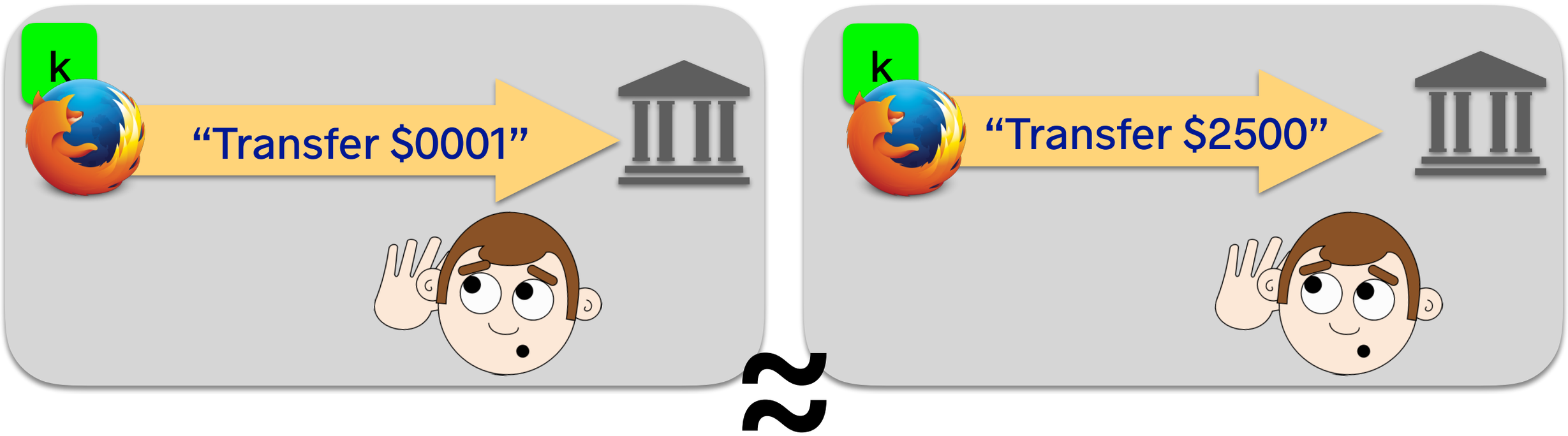


TLS Security Goal

[Very informal]

"Adversary learns nothing*
about the message being sent."

- Even if the adversary gets to tamper with network traffic,
- Even if the adversary gets to choose a part of the message, and
- Even if the client sends the same message many times.



TLS protocol

TLS protocol

- TLS/SSL has been around since 1995
- It is the backbone of Internet security

... important to get right

TLS protocol

- TLS/SSL has been around since 1995
- It is the backbone of Internet security

... important to get right

And yet, many flaws:

TLS protocol

- TLS/SSL has been around since 1995
- It is the backbone of Internet security

... important to get right

And yet, many flaws:

The New York Times

Poodle Bug Marks Third Major Security Flaw Discovered This Year

BY NICOLE PERLROTH OCTOBER 15, 2014 1:58 PM

TLS protocol

- TLS/SSL has been around since 1995
- It is the backbone of Internet security

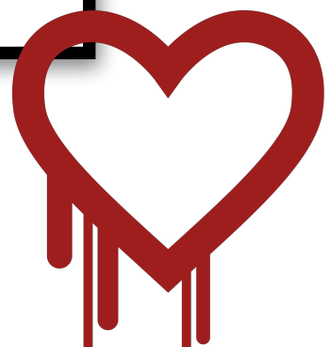
... important to get right

And yet, many flaws:

The New York Times

Poodle Bug Marks Third Major Security Flaw Discovered This Year

BY NICOLE PERLROTH OCTOBER 15, 2014 1:58 PM



Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Goals of this talk

1. Explain ideas behind link encryption (TLS).
2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack
3. Highlight open problems and research directions.

Warm up

Warm up

- Say we have a 7 MB message and a 128-bit cipher
- How do we encrypt a long message with a short cipher?

Warm up

- Say we have a 7 MB message and a 128-bit cipher
- How do we encrypt a long message with a short cipher?

Electronic Codebook Mode (ECB)

1. Break up message into 128-bit blocks
2. Encrypt message block by block

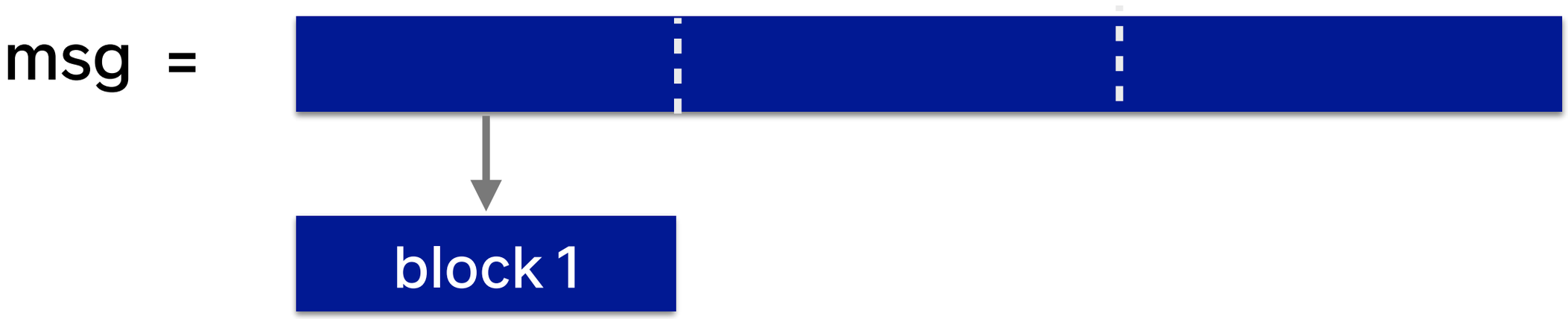
Warm up: ECB encryption

msg =

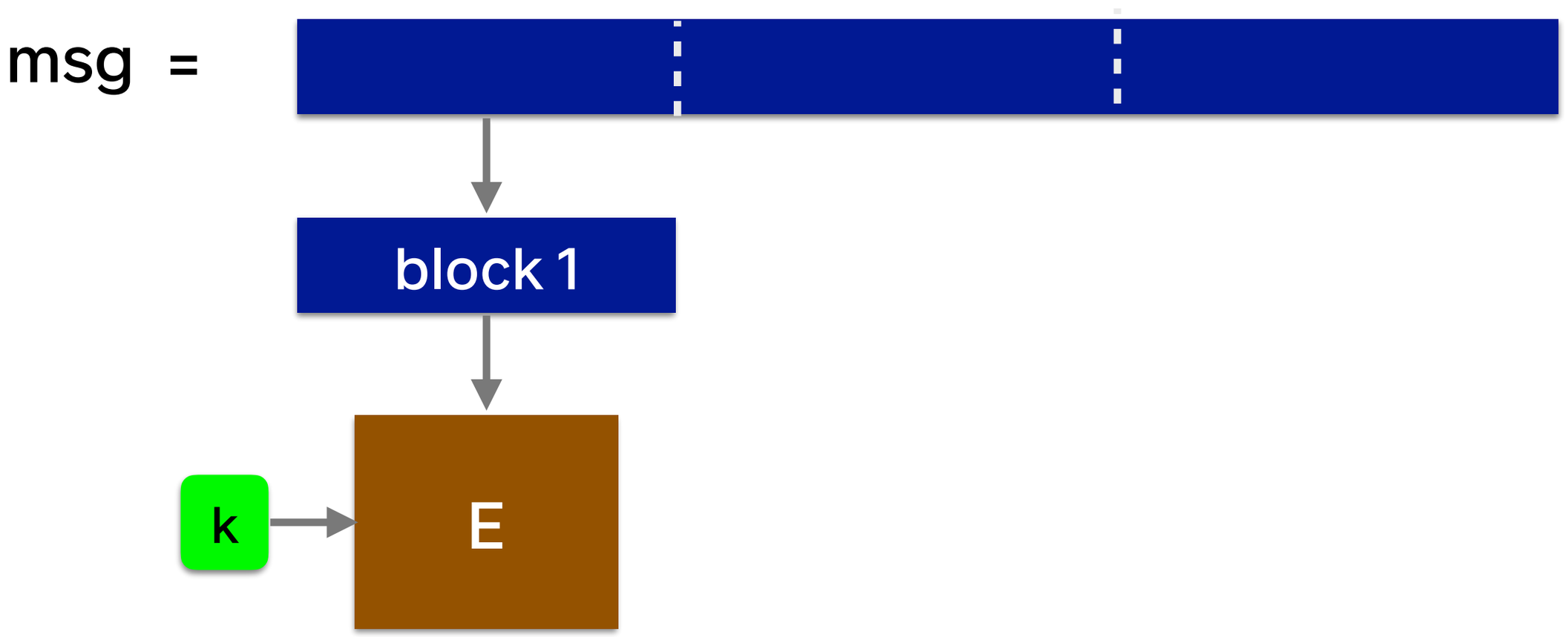
Warm up: ECB encryption

msg =

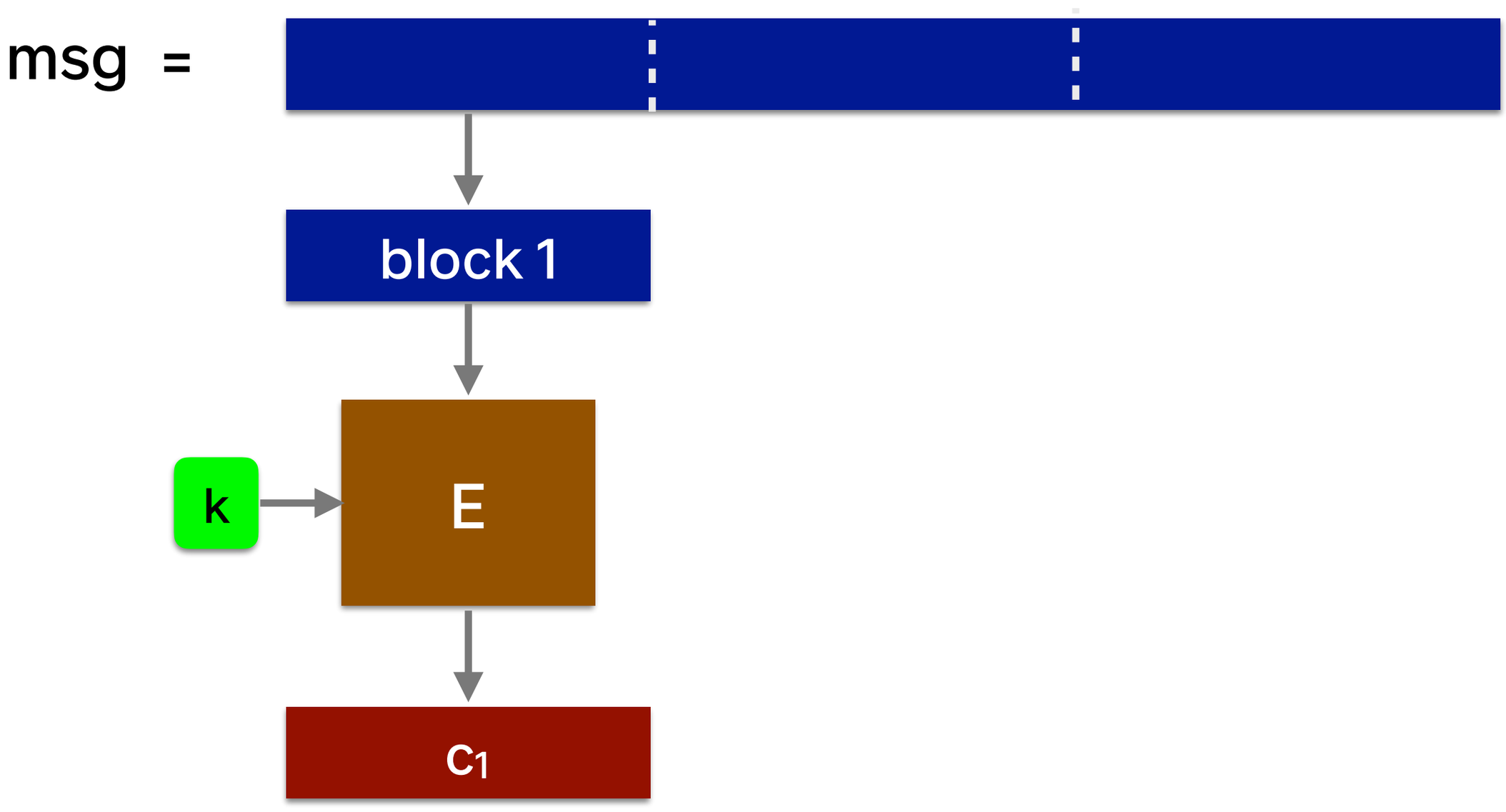
Warm up: ECB encryption



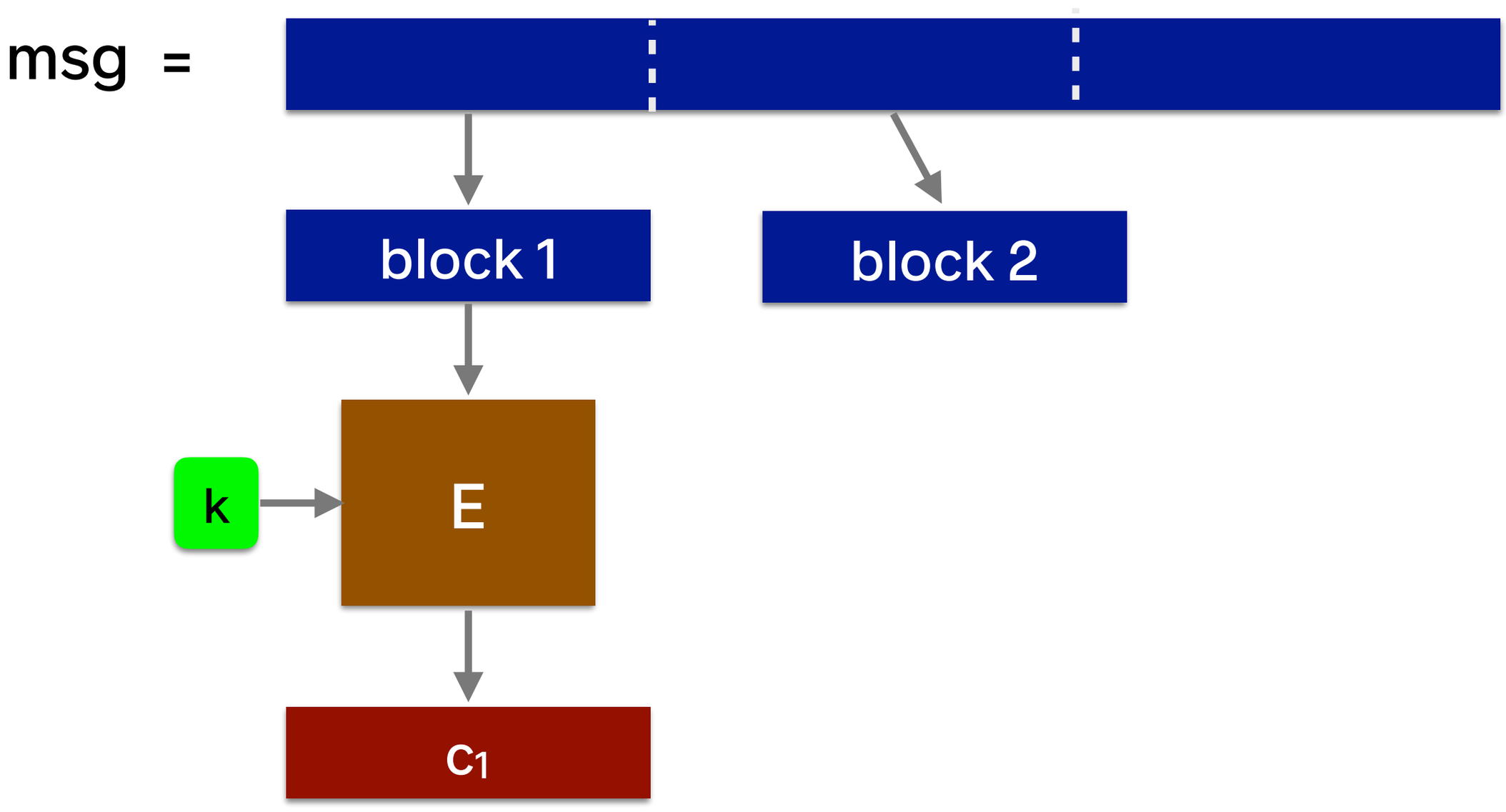
Warm up: ECB encryption



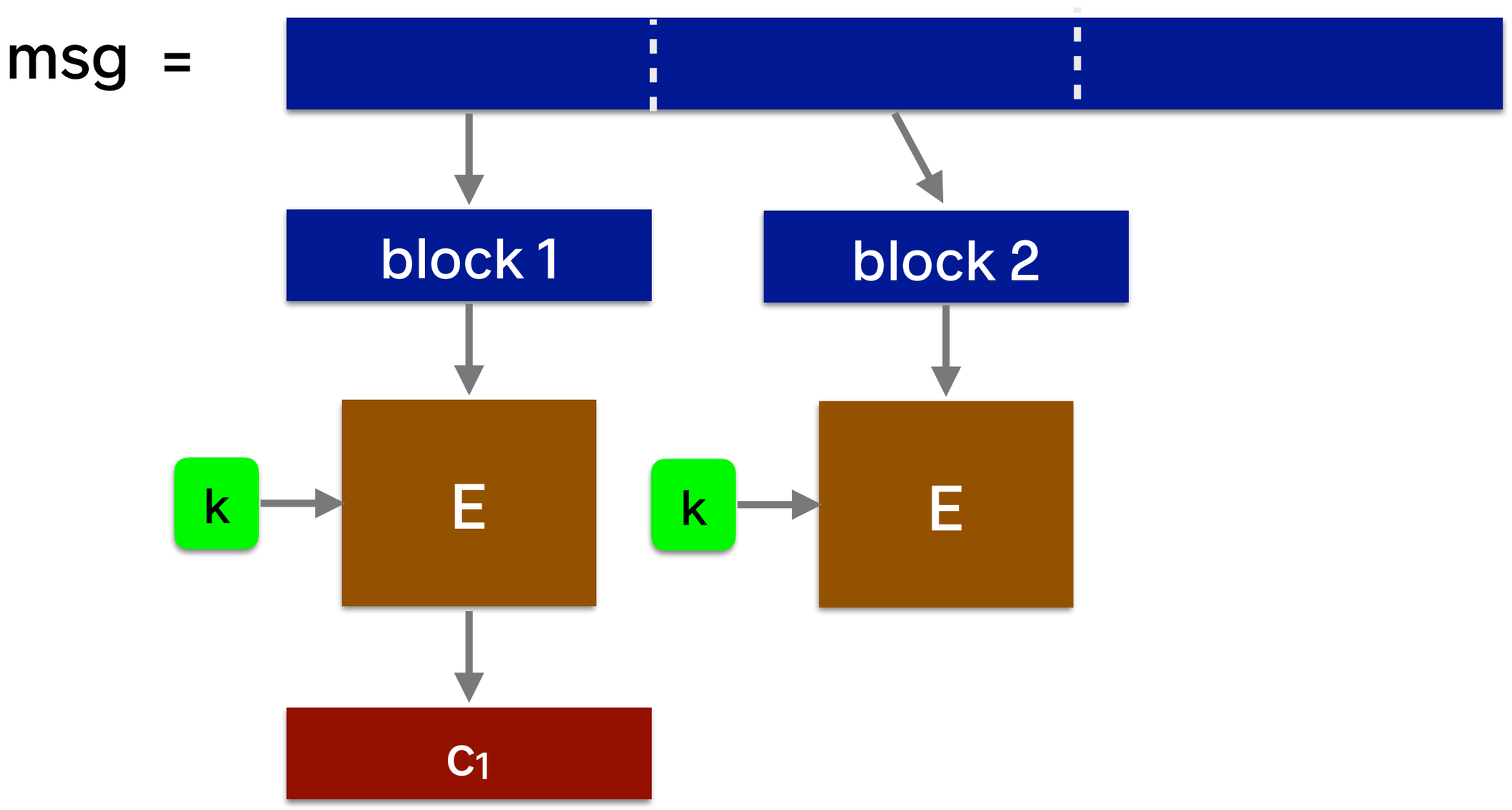
Warm up: ECB encryption



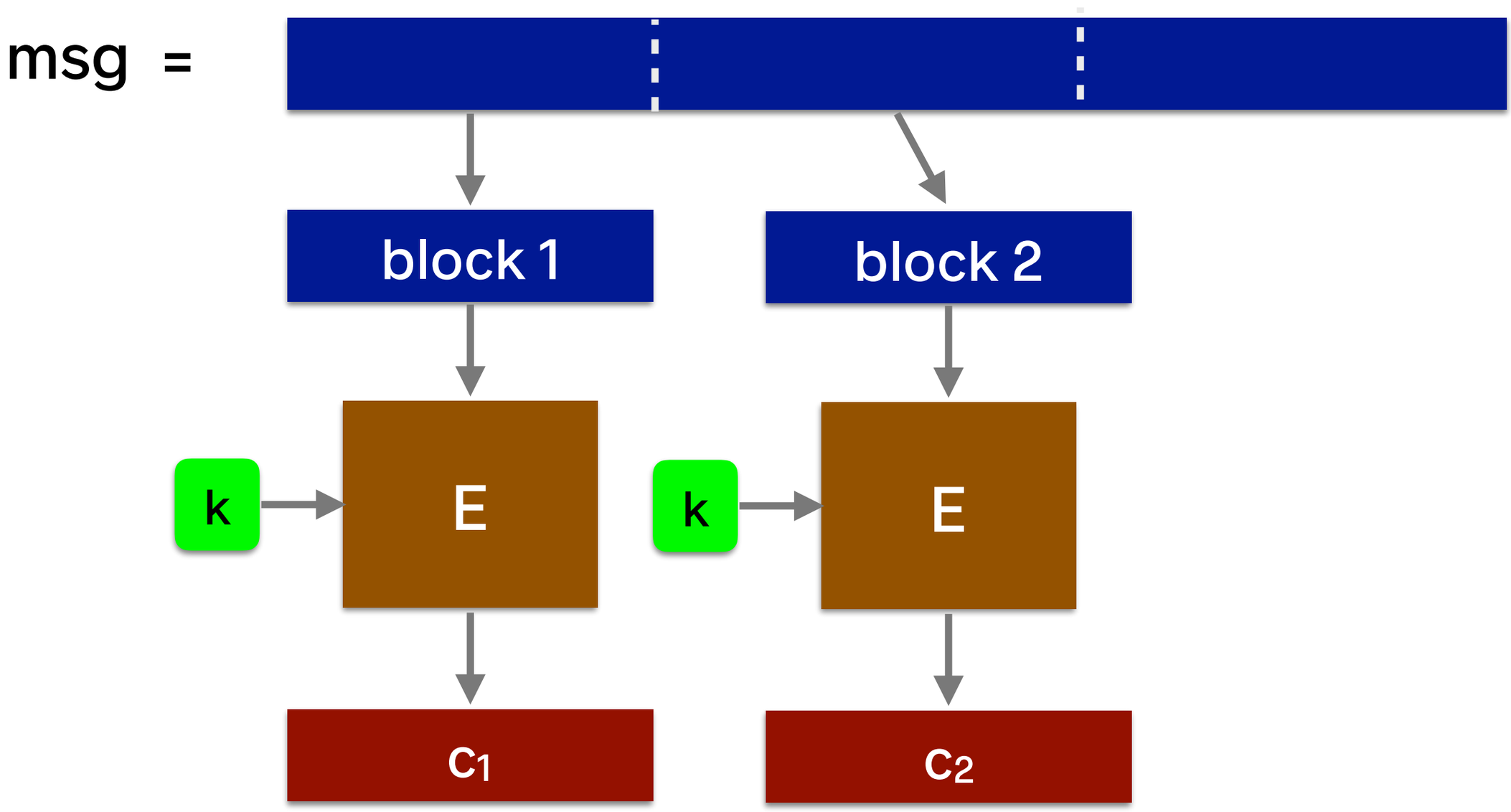
Warm up: ECB encryption



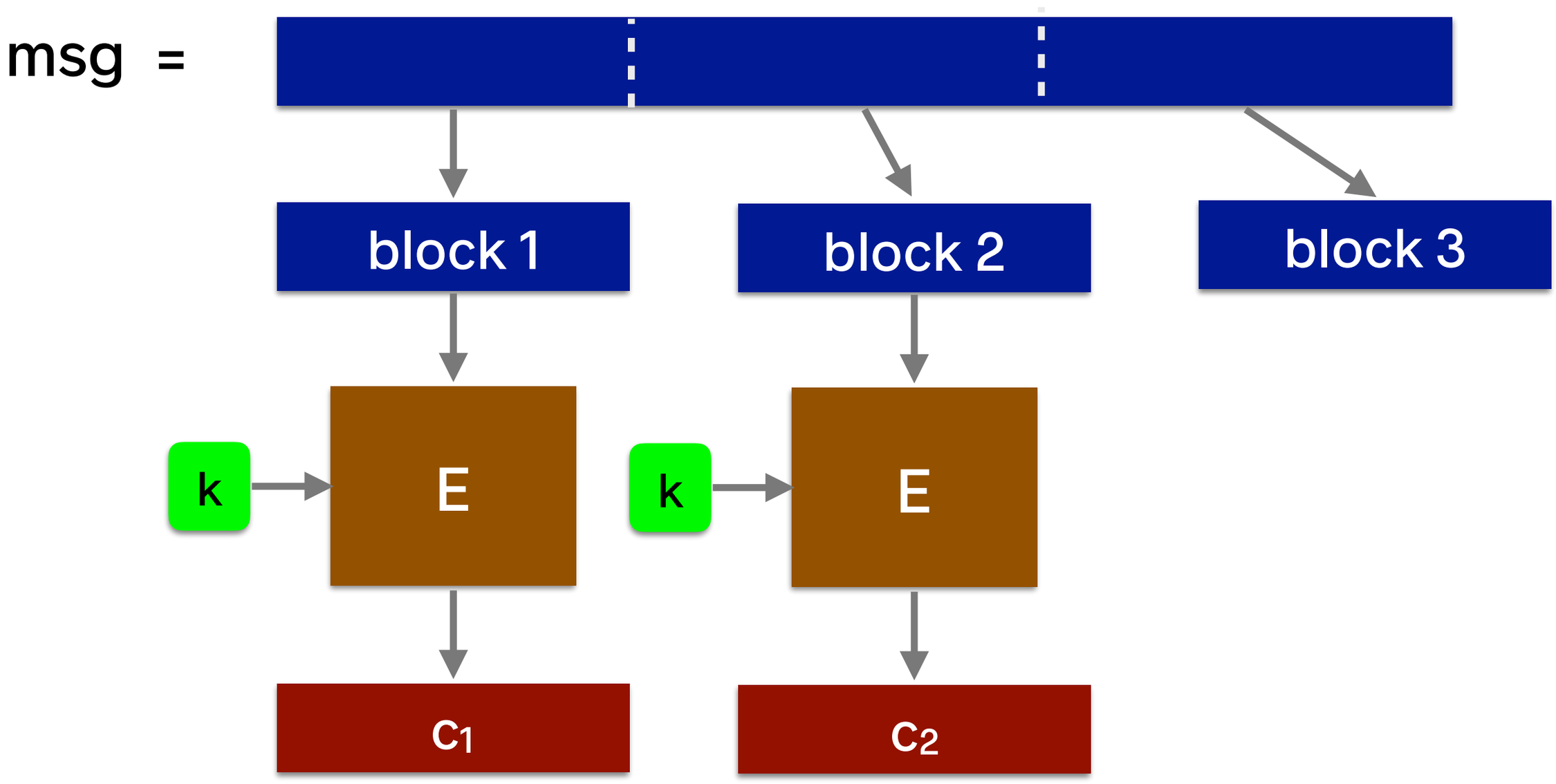
Warm up: ECB encryption



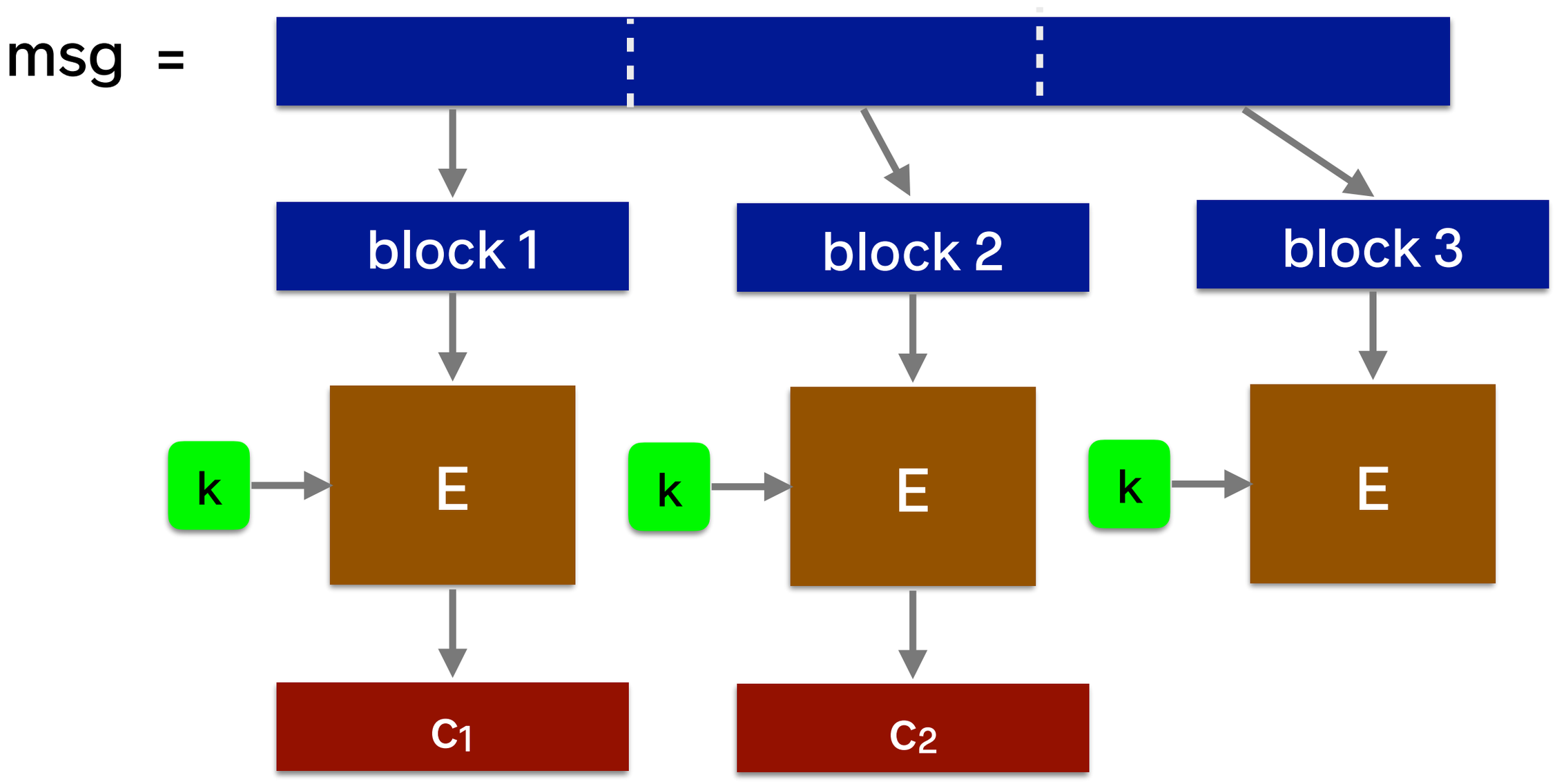
Warm up: ECB encryption



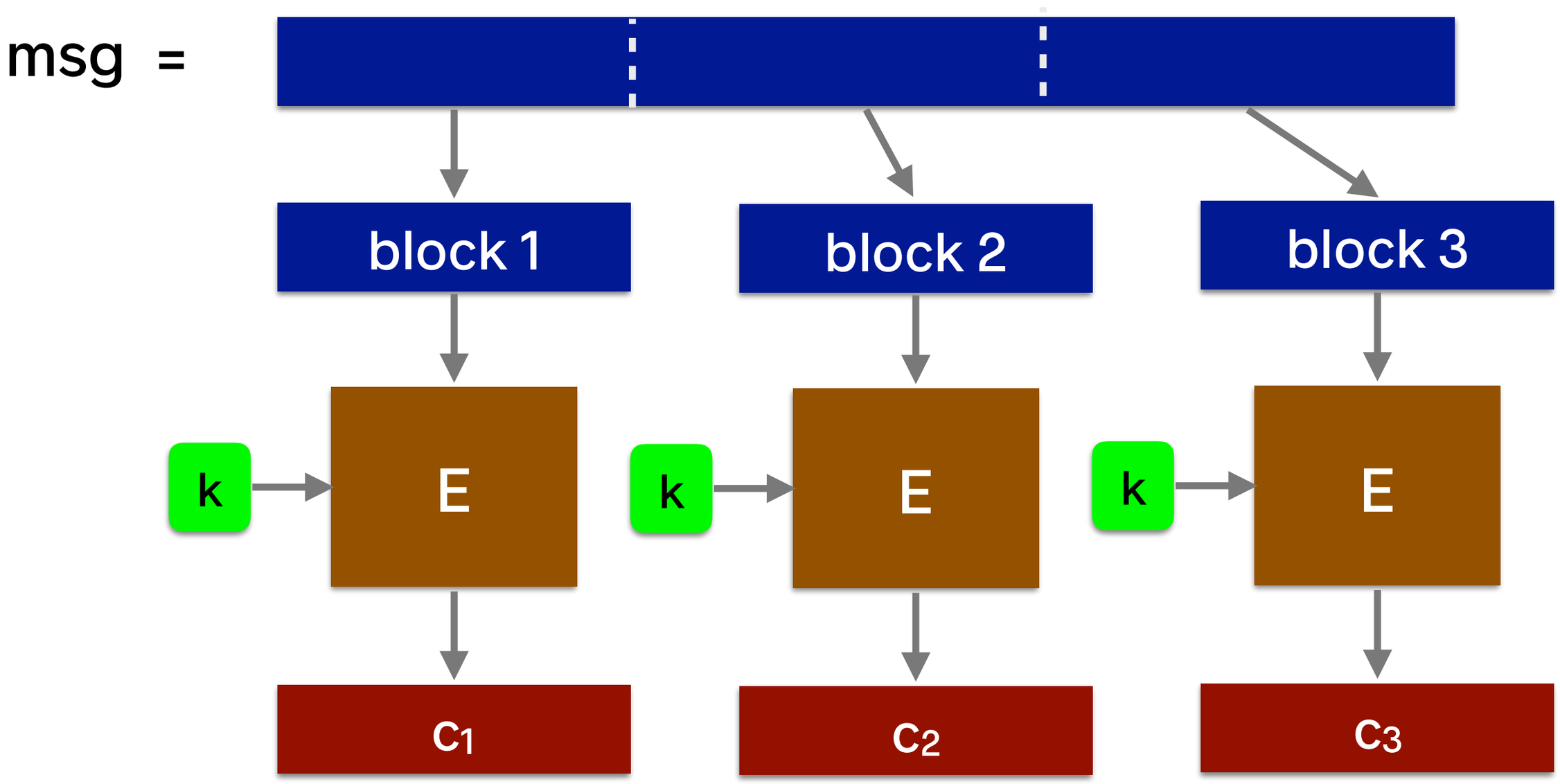
Warm up: ECB encryption



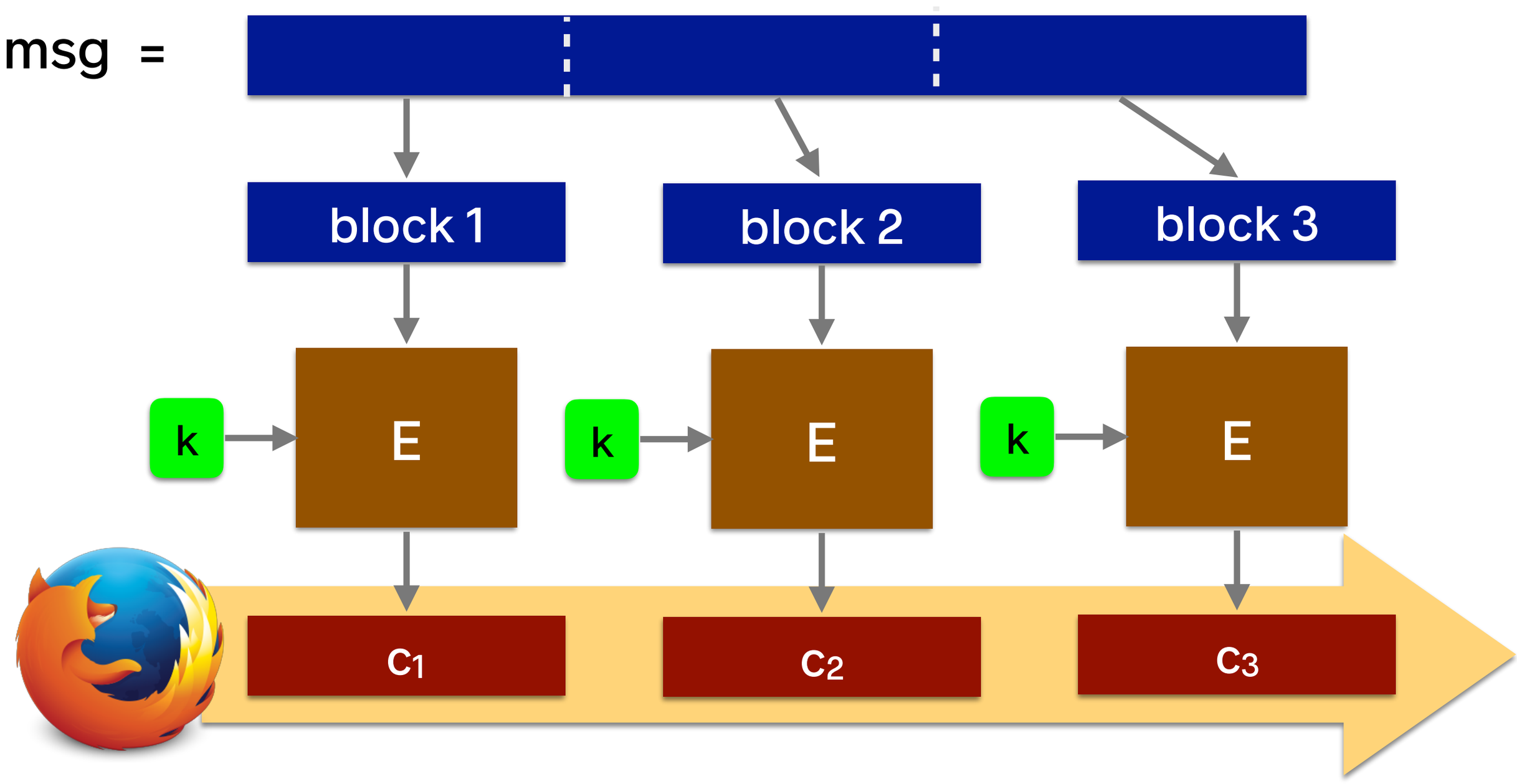
Warm up: ECB encryption



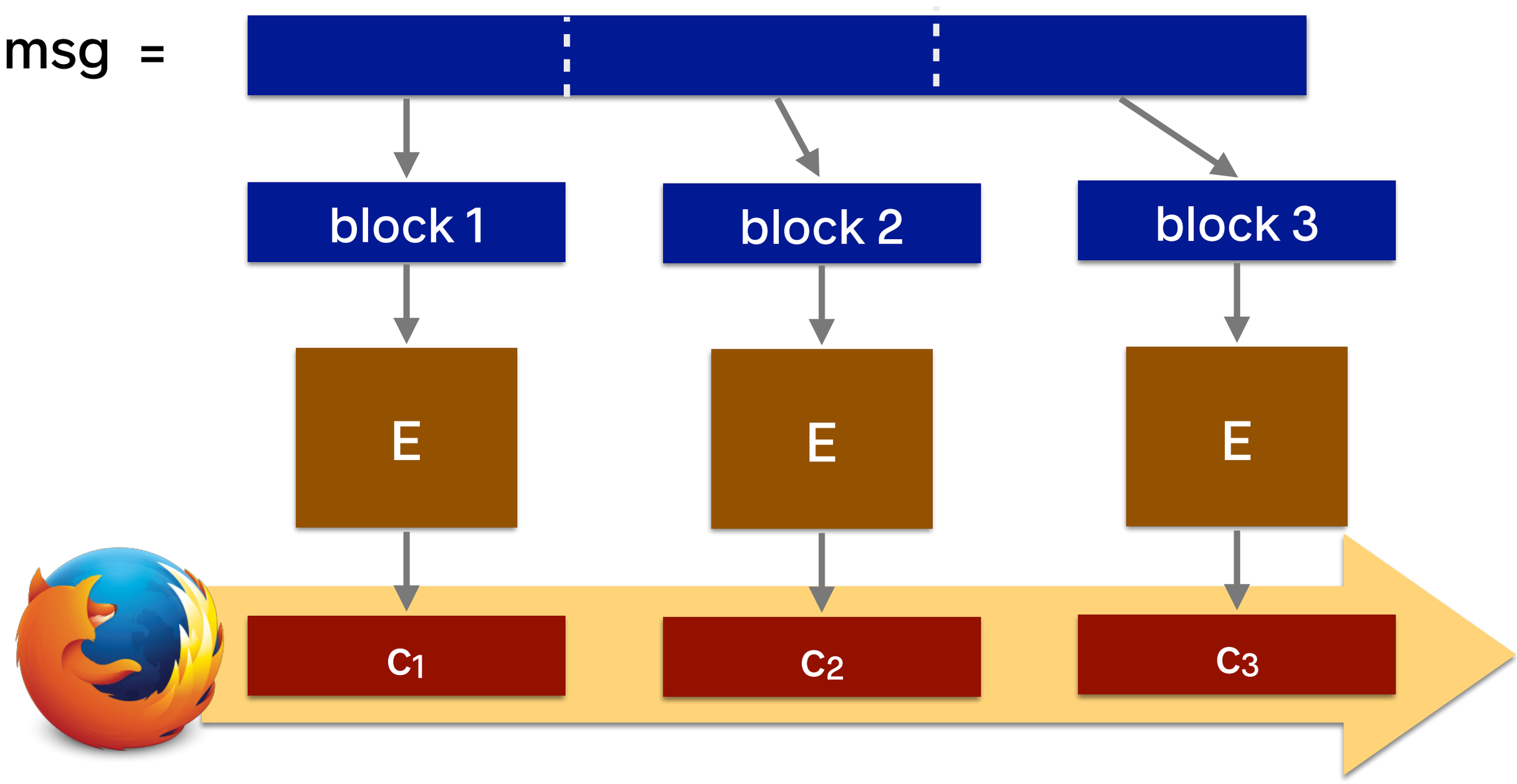
Warm up: ECB encryption



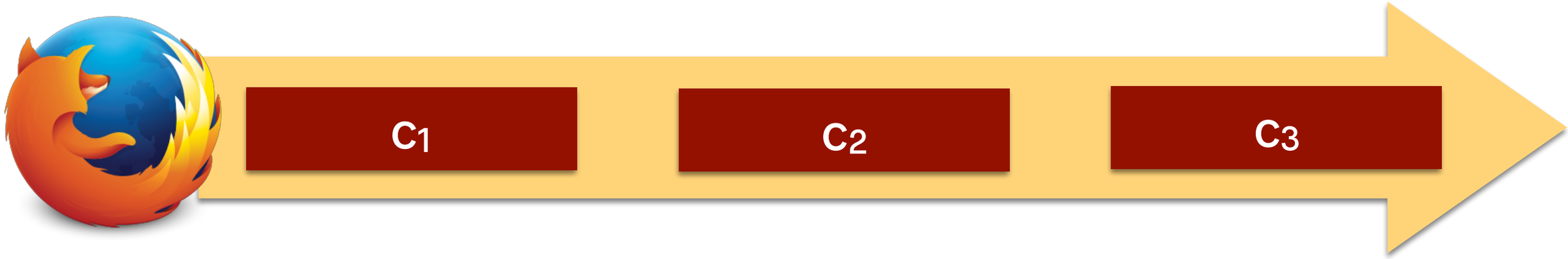
Warm up: ECB encryption



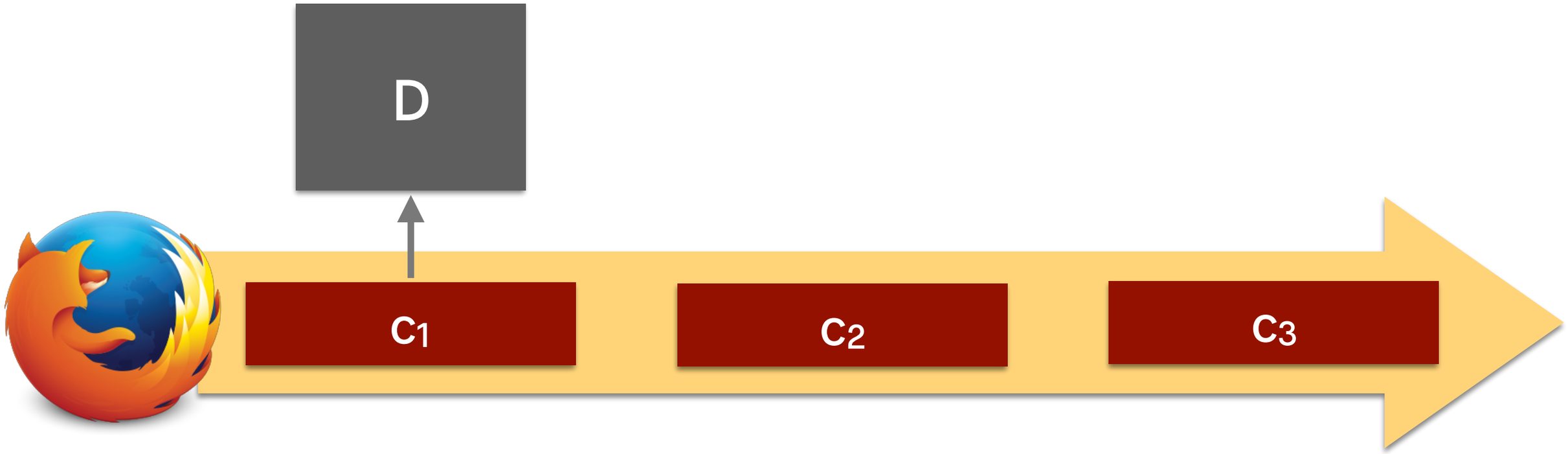
Warm up: ECB encryption



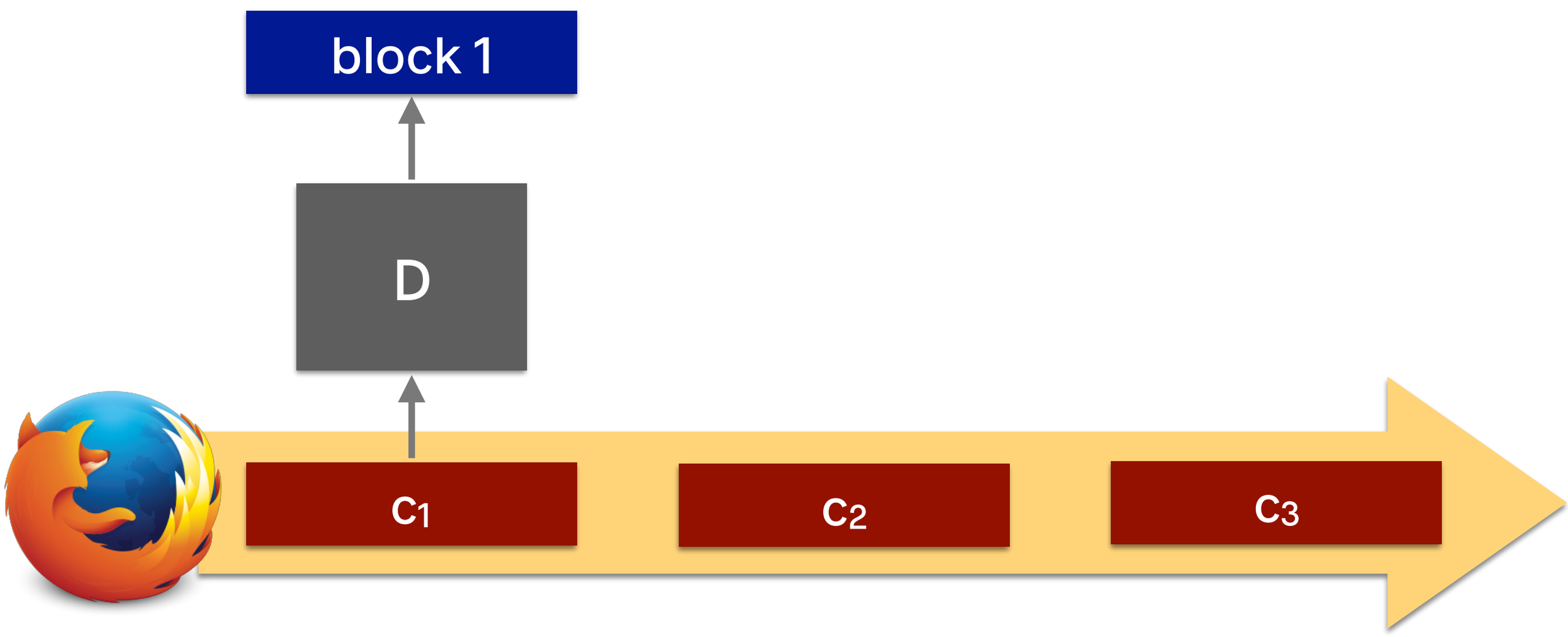
Warm up: ECB decryption



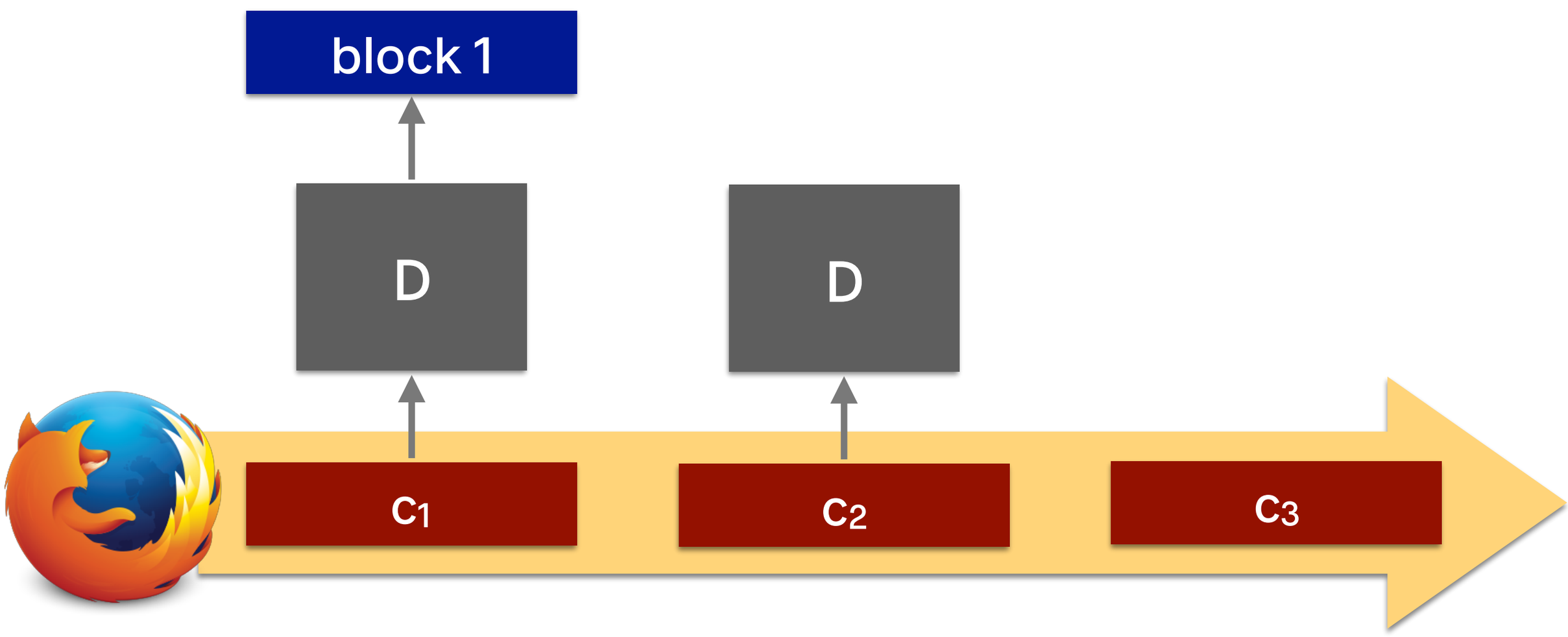
Warm up: ECB decryption



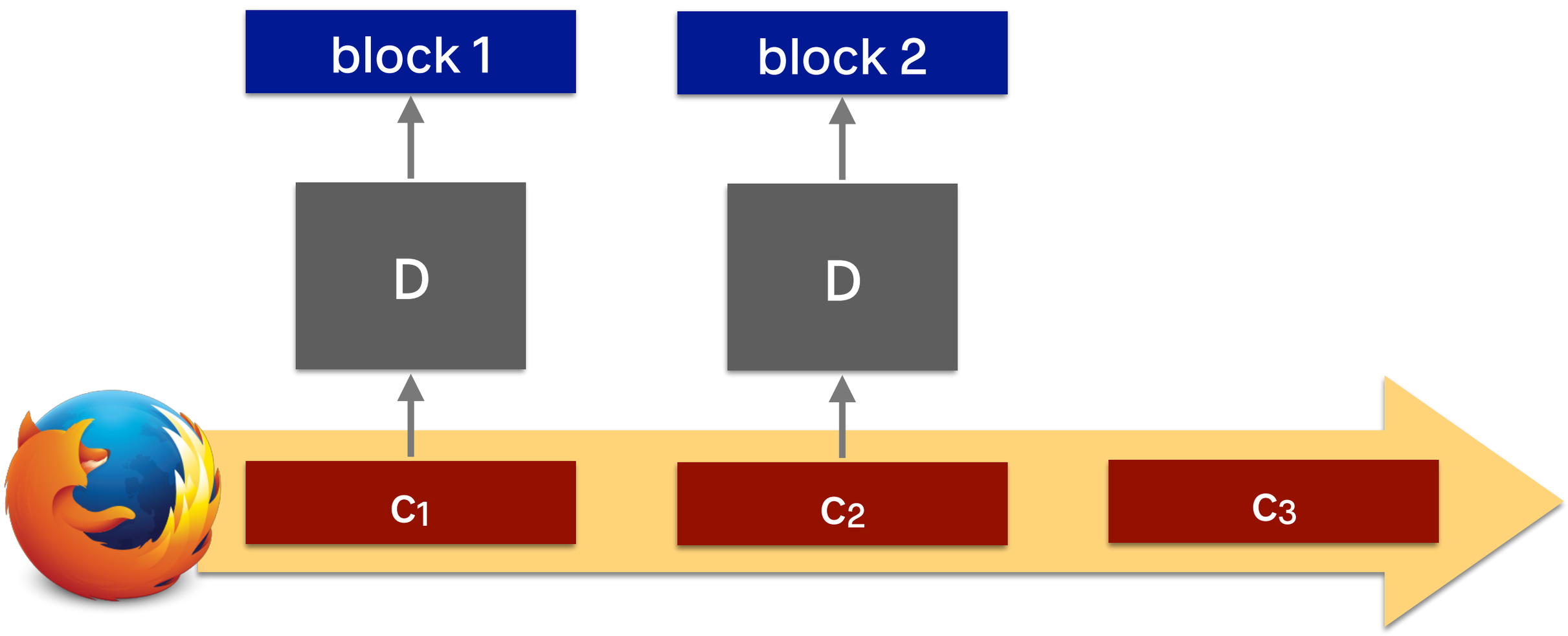
Warm up: ECB decryption



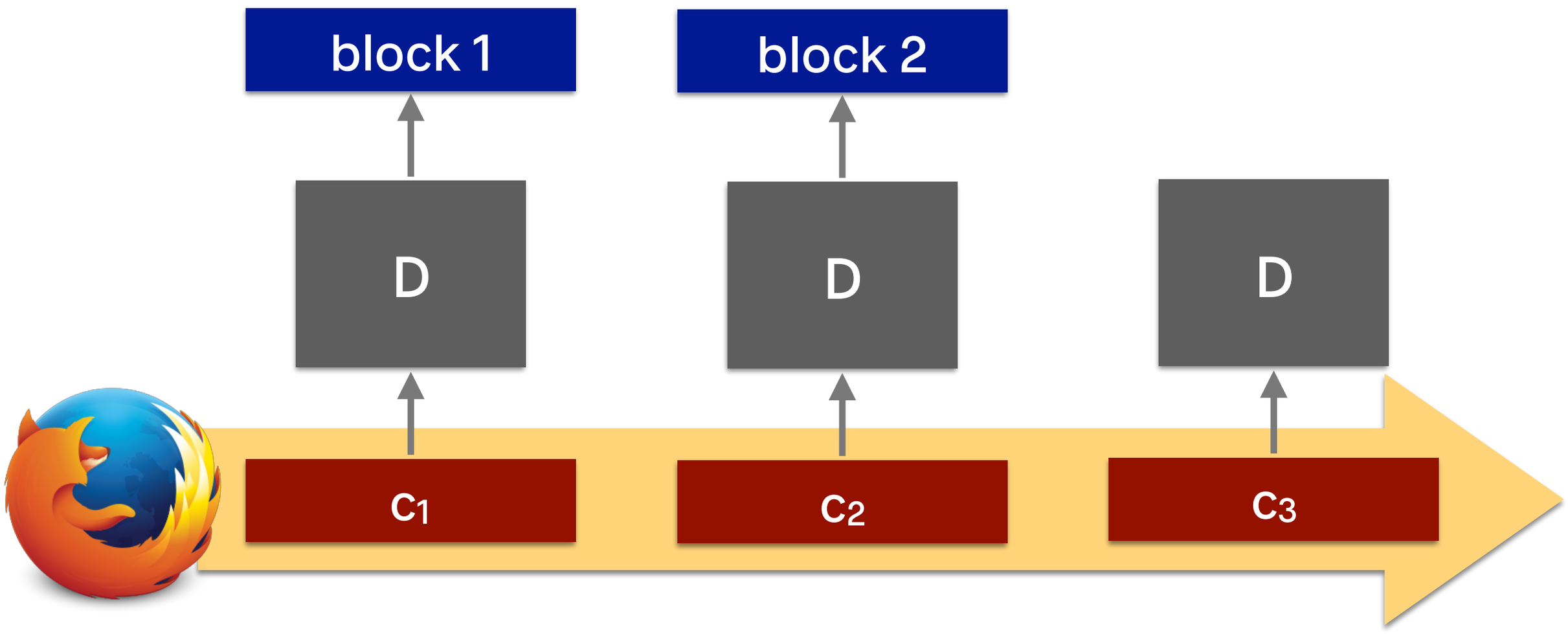
Warm up: ECB decryption



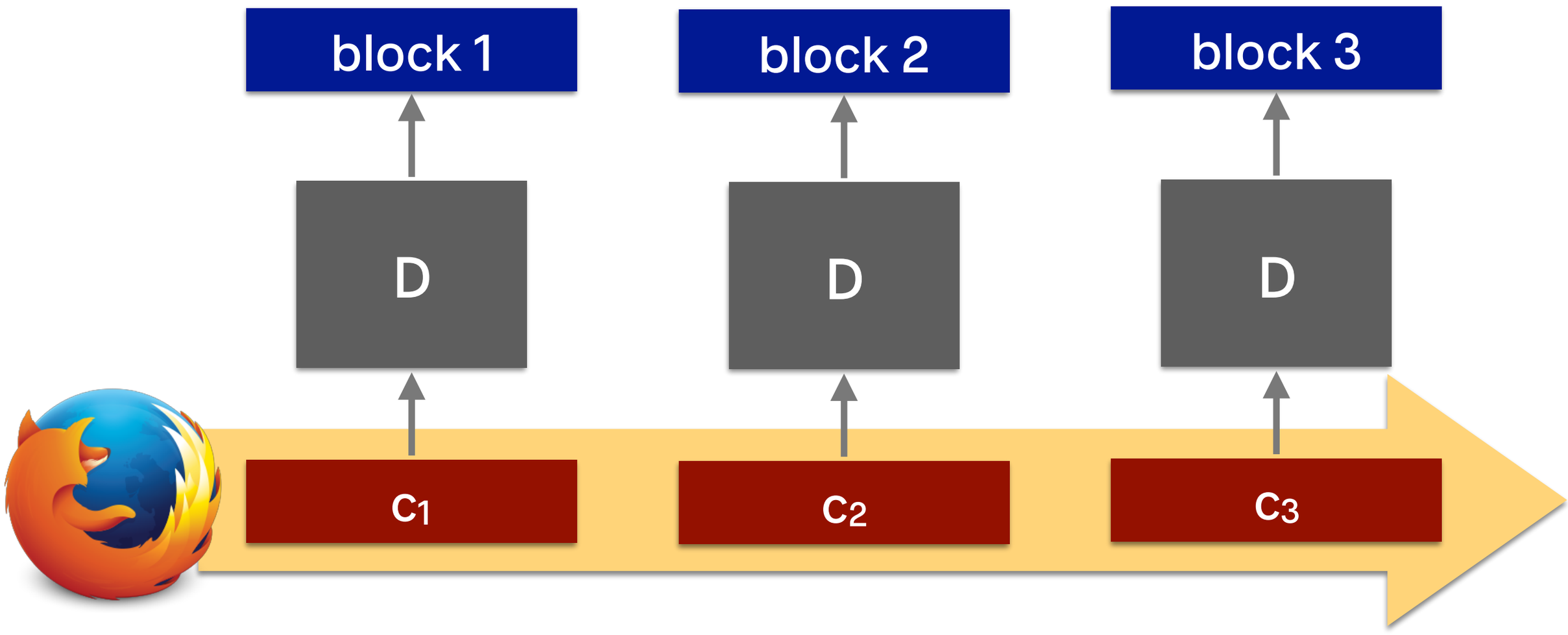
Warm up: ECB decryption



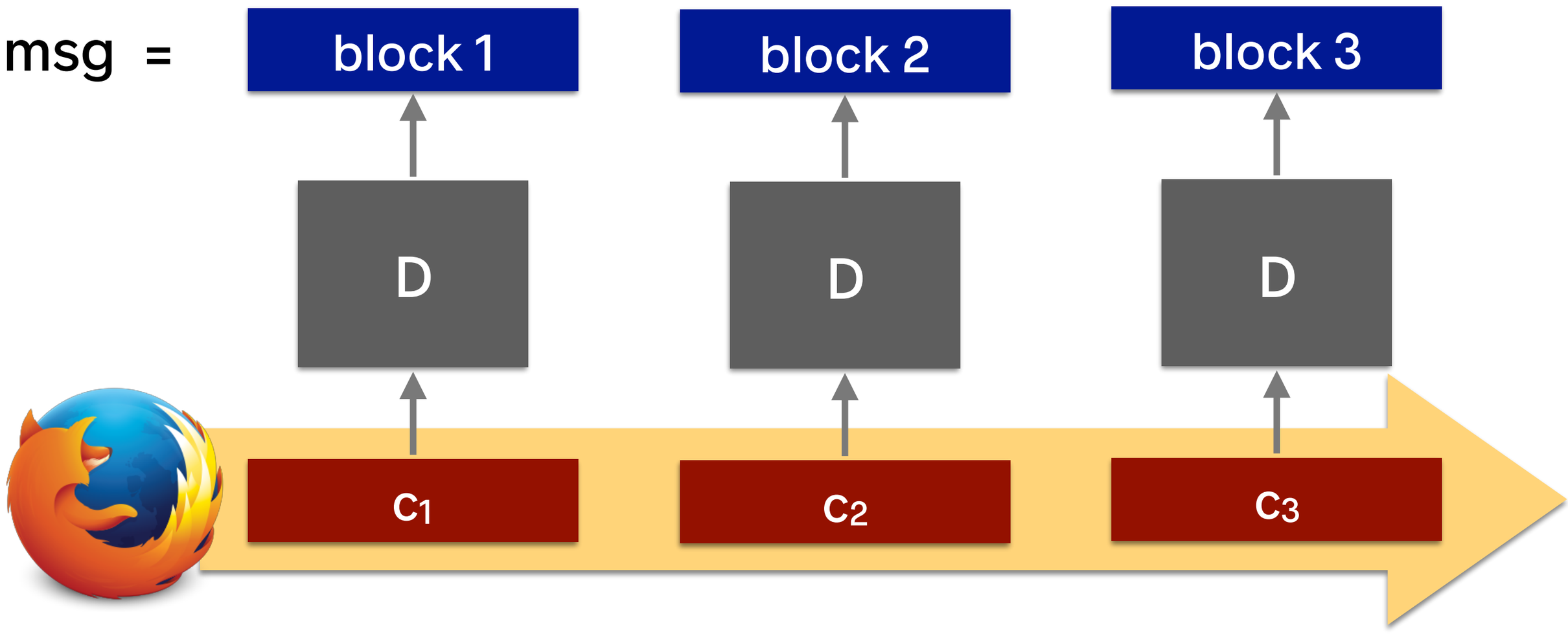
Warm up: ECB decryption



Warm up: ECB decryption



Warm up: ECB decryption



Warm up: ECB mode

Warm up: ECB mode

Problem?

Warm up: ECB mode

Problem?

- Repeated patterns in message show up as repeated patterns in ciphertext!

Warm up: ECB mode

Problem?

- Repeated patterns in message show up as repeated patterns in ciphertext!
  - “cow” | “cow” | “cow” → 5d4f 5d4f 5d4f

Warm up: ECB mode

Problem?

- Repeated patterns in message show up as repeated patterns in ciphertext!
  - “cow” | “cow” | “cow” → 5d4f 5d4f 5d4f
  - “cow” | “dog” | “cat” → 5d4f 1c91 f93e

Warm up: ECB mode

Problem?

- Repeated patterns in message show up as repeated patterns in ciphertext!
  - “cow” | “cow” | “cow” → 5d4f 5d4f 5d4f
  - “cow” | “dog” | “cat” → 5d4f 1c91 f93e

Does this little bit of leakage matter?

HUGHES®

The famous penguin: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

HUGHES®

256-bit AES encryption
in ECB mode

The famous penguin: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

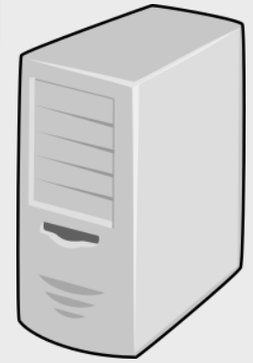
256-bit AES encryption
in ECB mode

The famous penguin: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

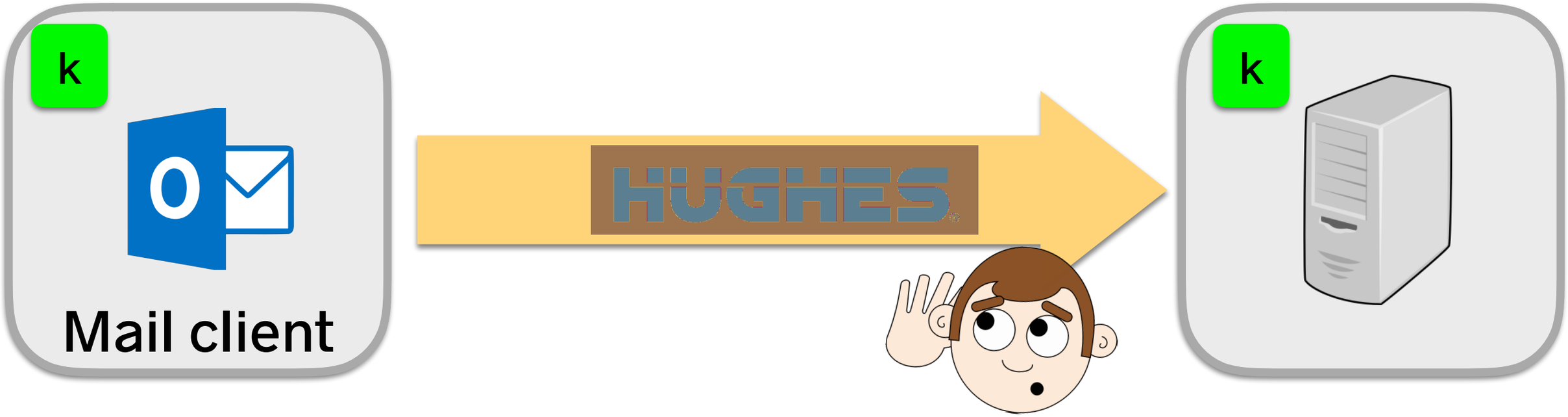
k

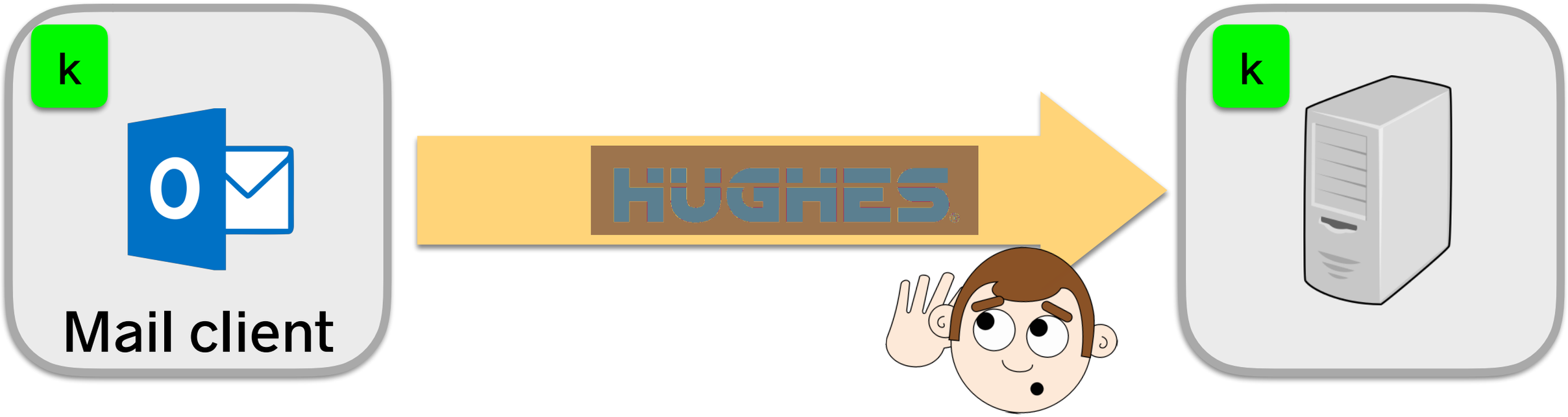
Mail client

k



k
Mail client
HUGHES
k


k
Mail client
HUGHES
k



Moral: A little bit of leakage goes a long way.

Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Second attempt: CBC mode

- Naive ECB mode (encrypt chunk-by-chunk) leaks repeated message blocks

- If we “mix the message blocks together” properly, these attacks will no longer apply

- Cipher block chaining mode (CBC) does just this
  — One of two major options in TLS until ~2008

CBC encryption

block 1

block 2

block 3

CBC encryption

block 1

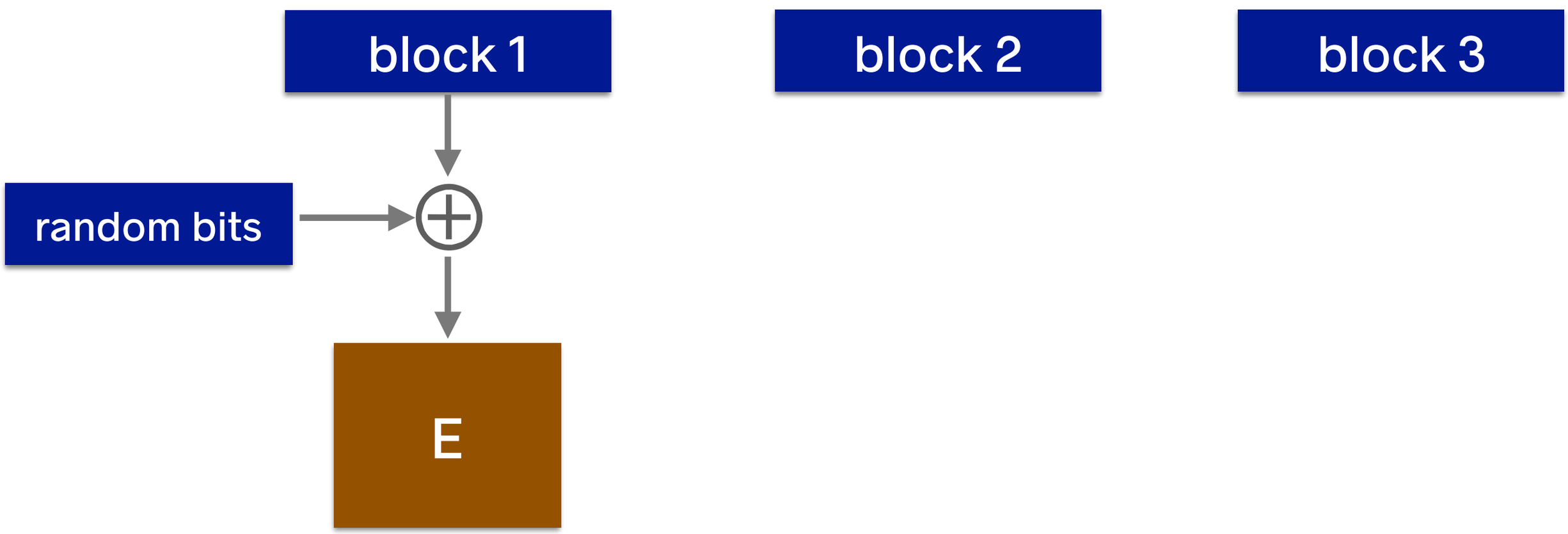
block 2

block 3

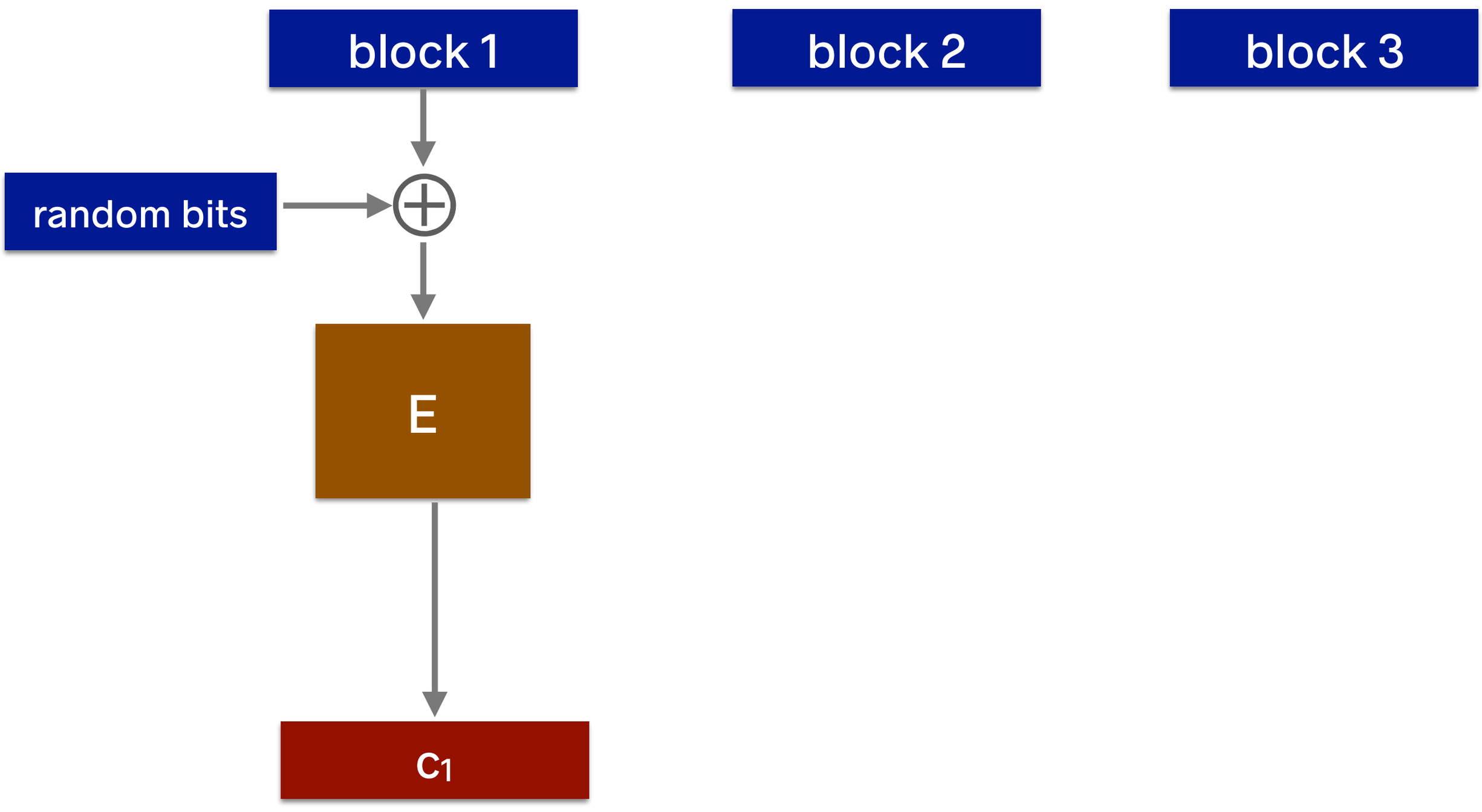
random bits

CBC encryption

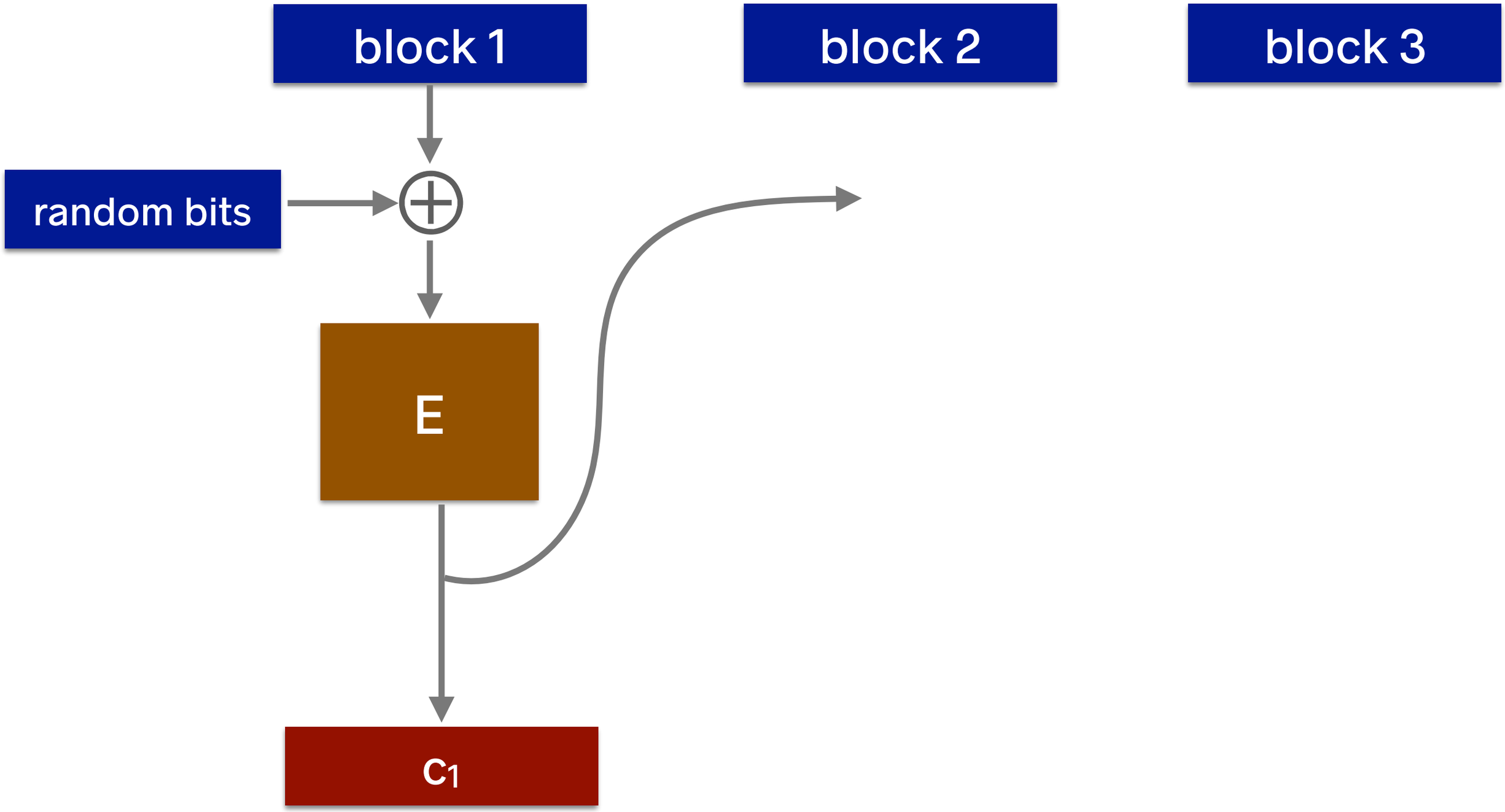
CBC encryption



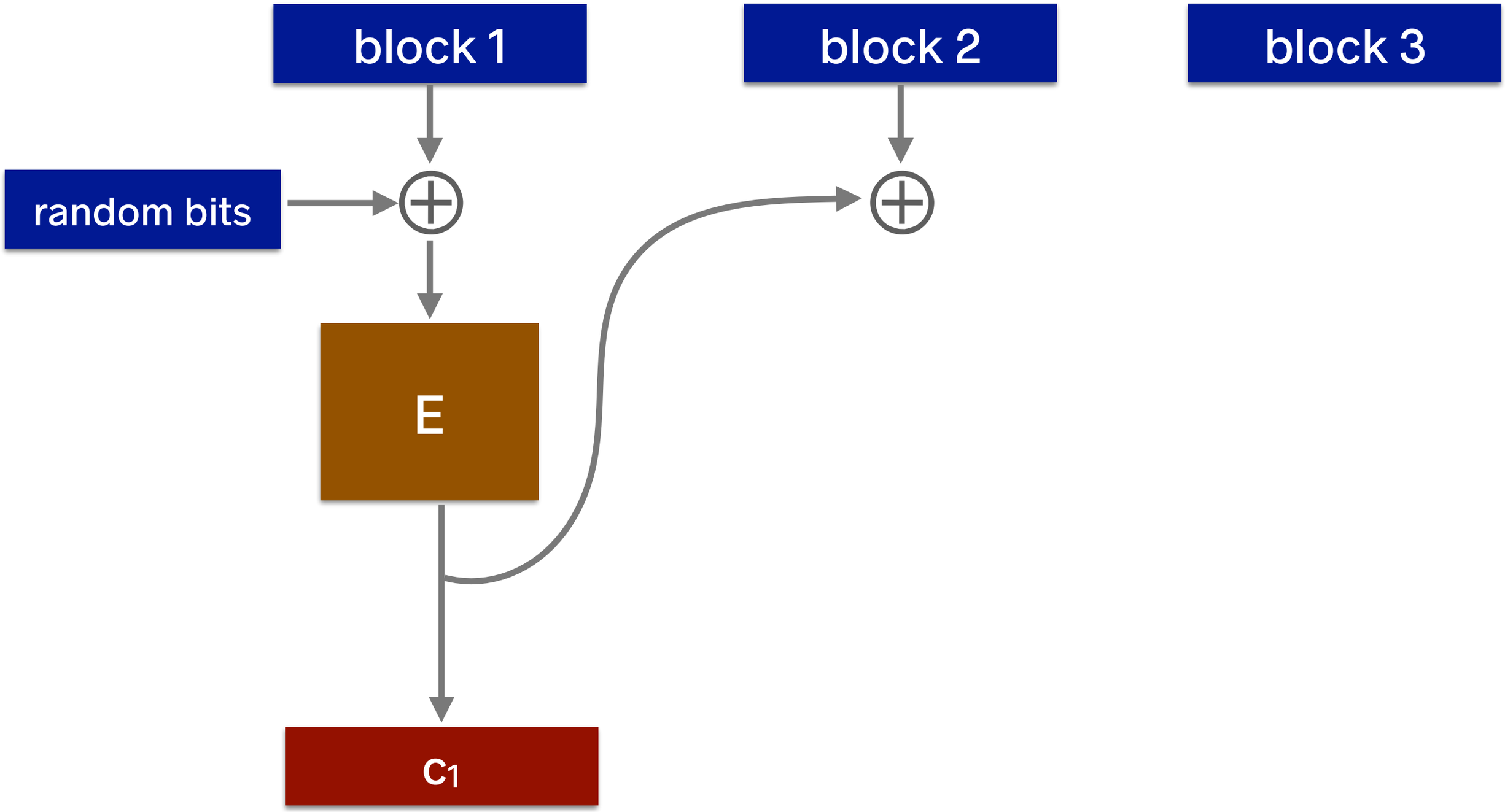
CBC encryption



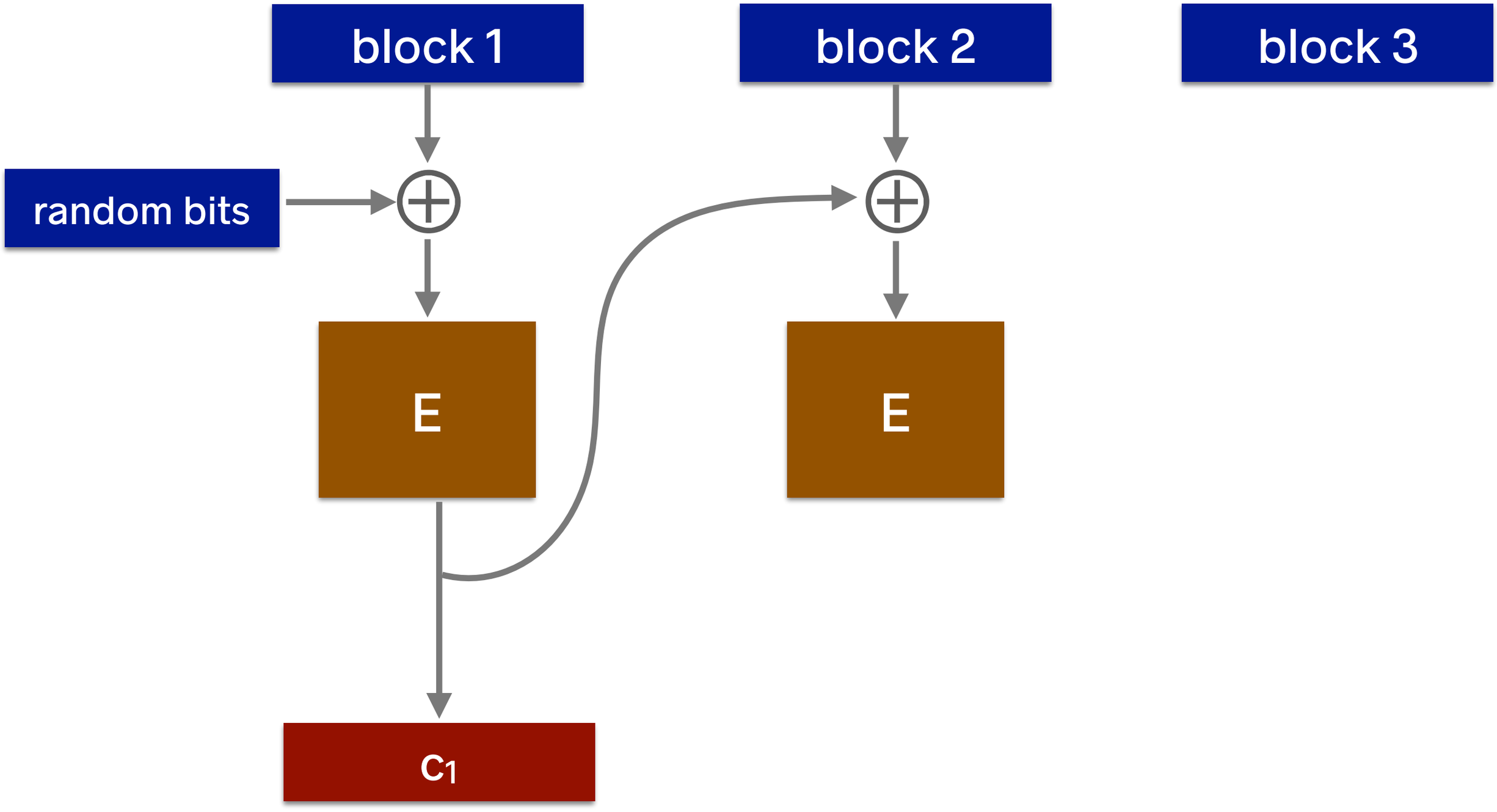
CBC encryption



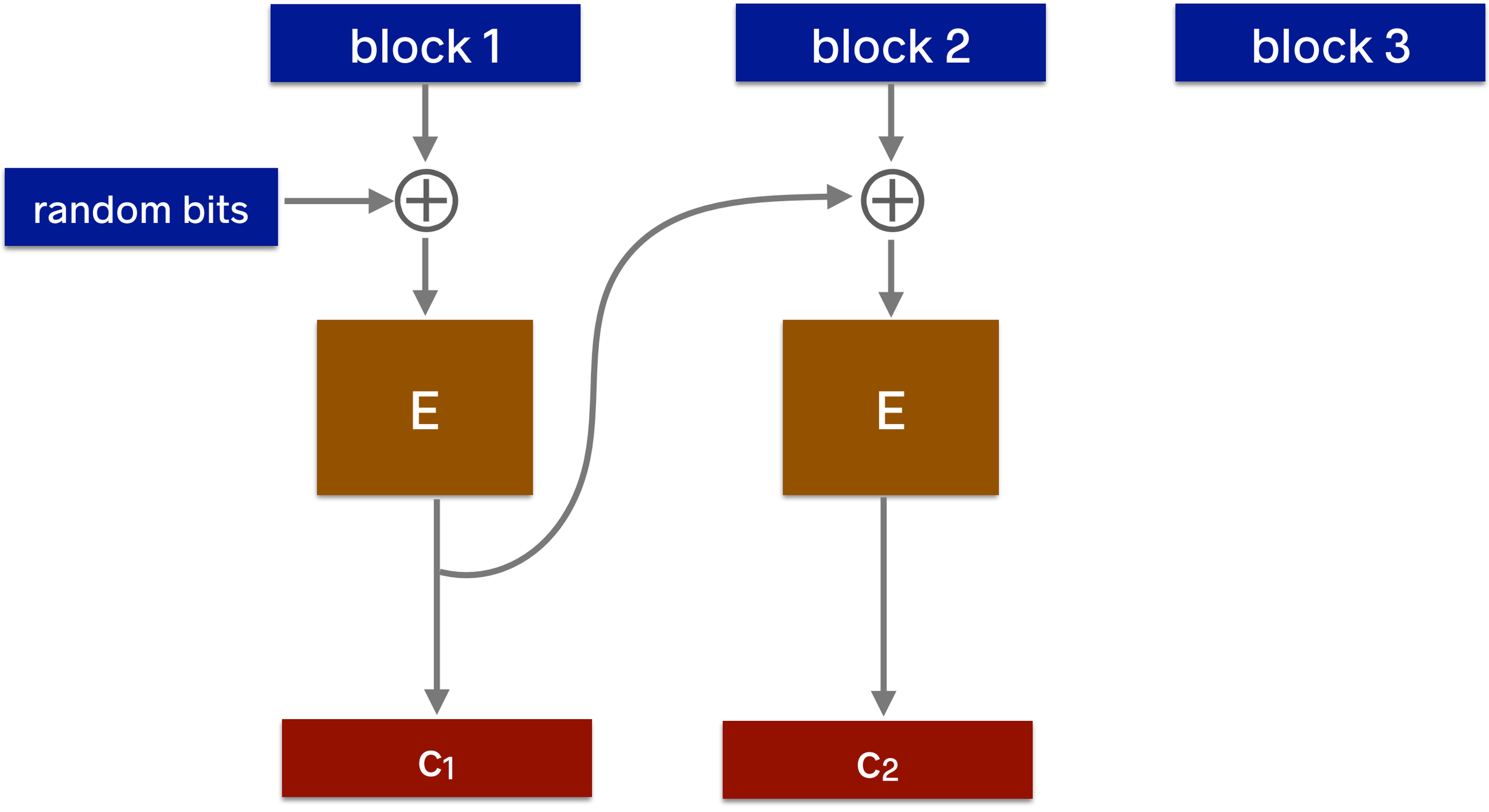
CBC encryption



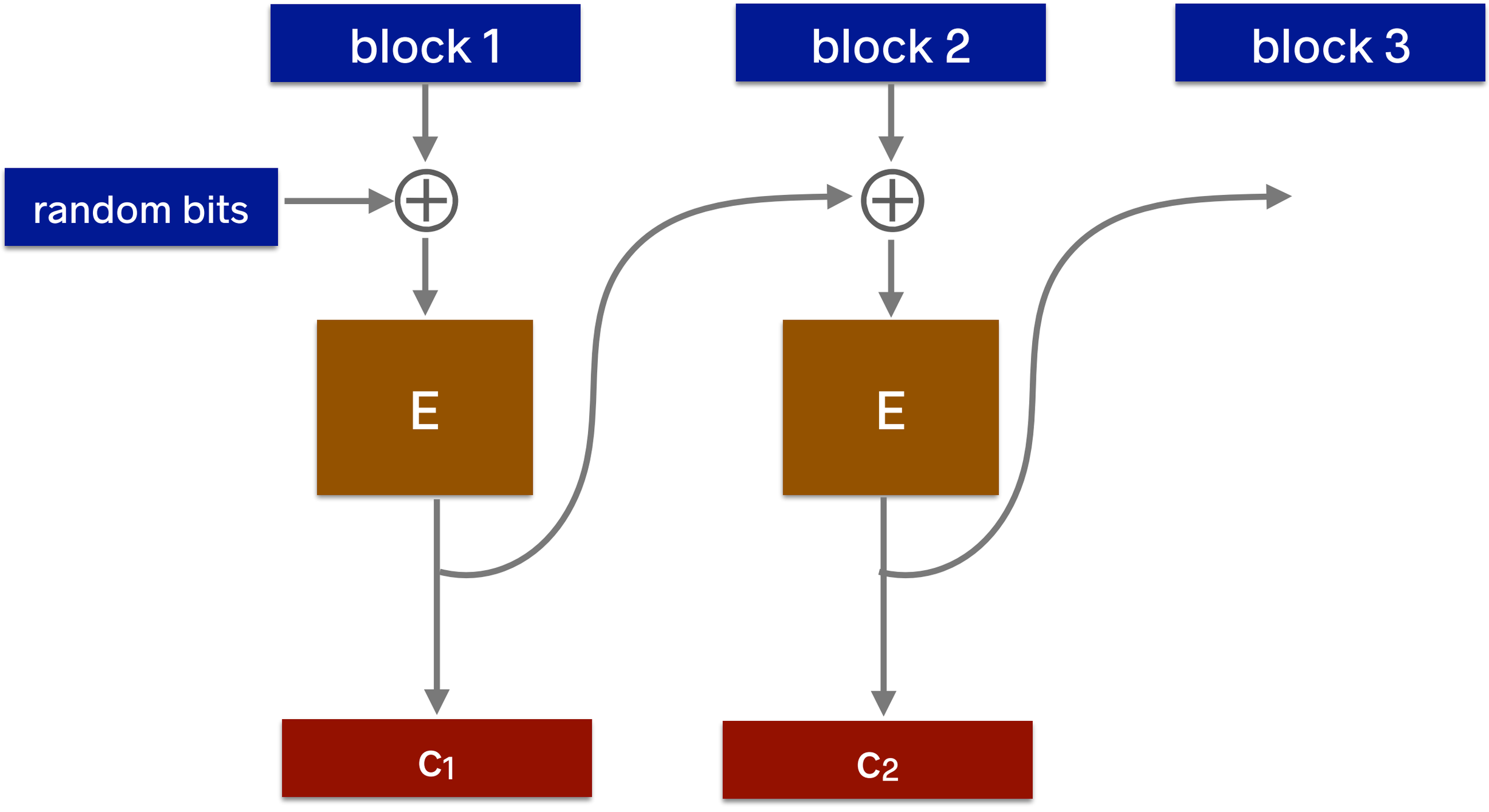
CBC encryption



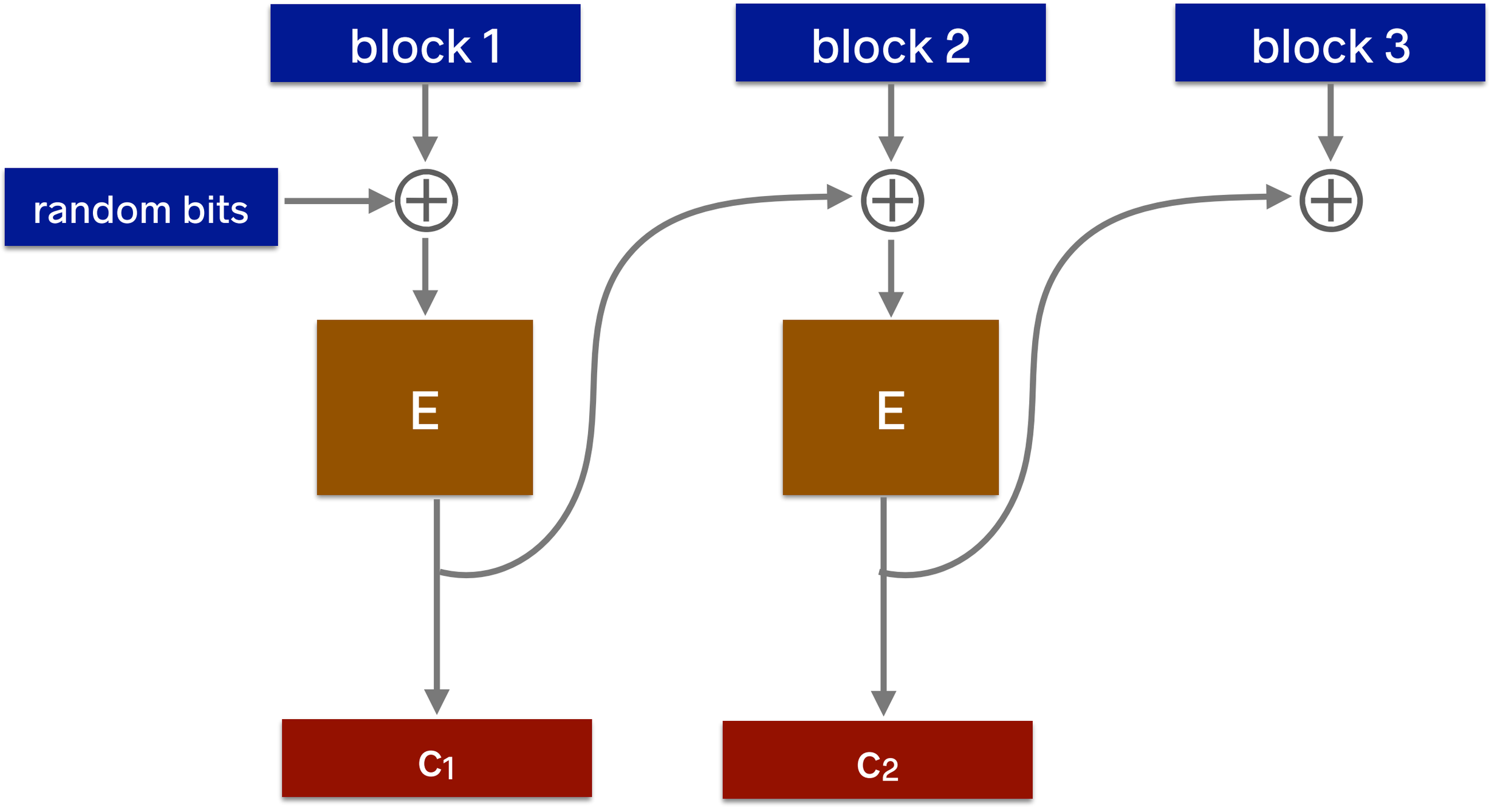
CBC encryption



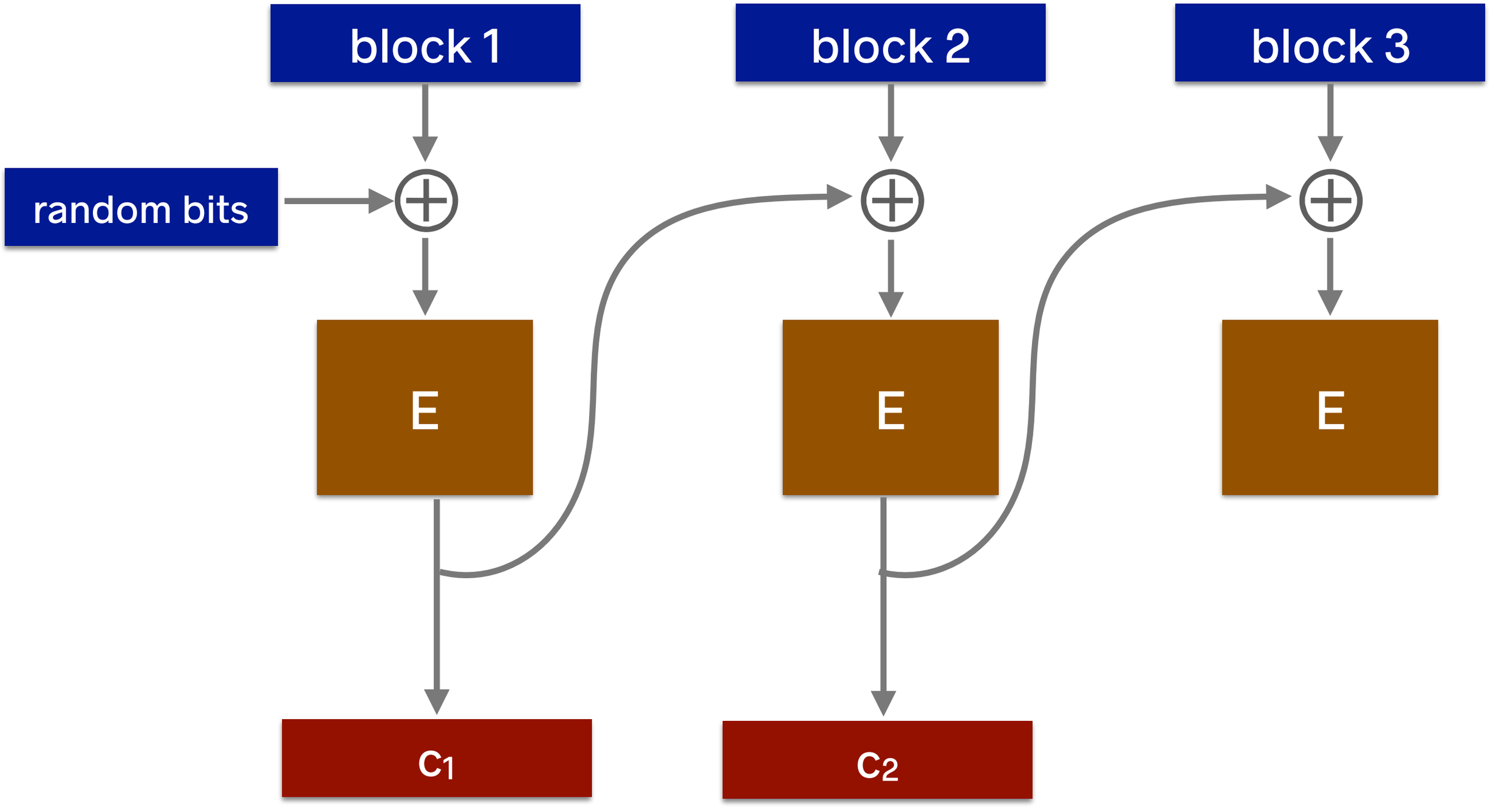
CBC encryption



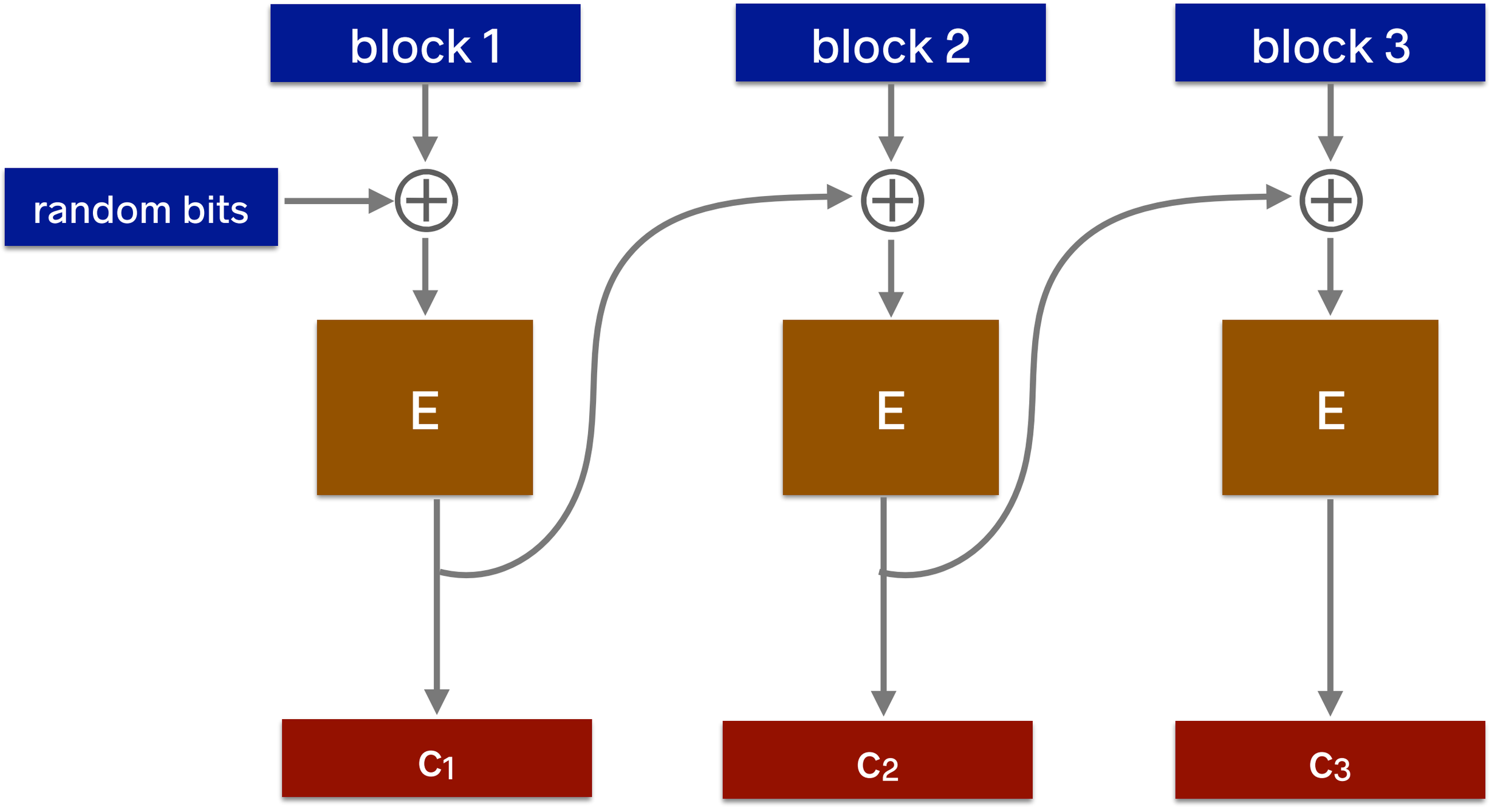
CBC encryption



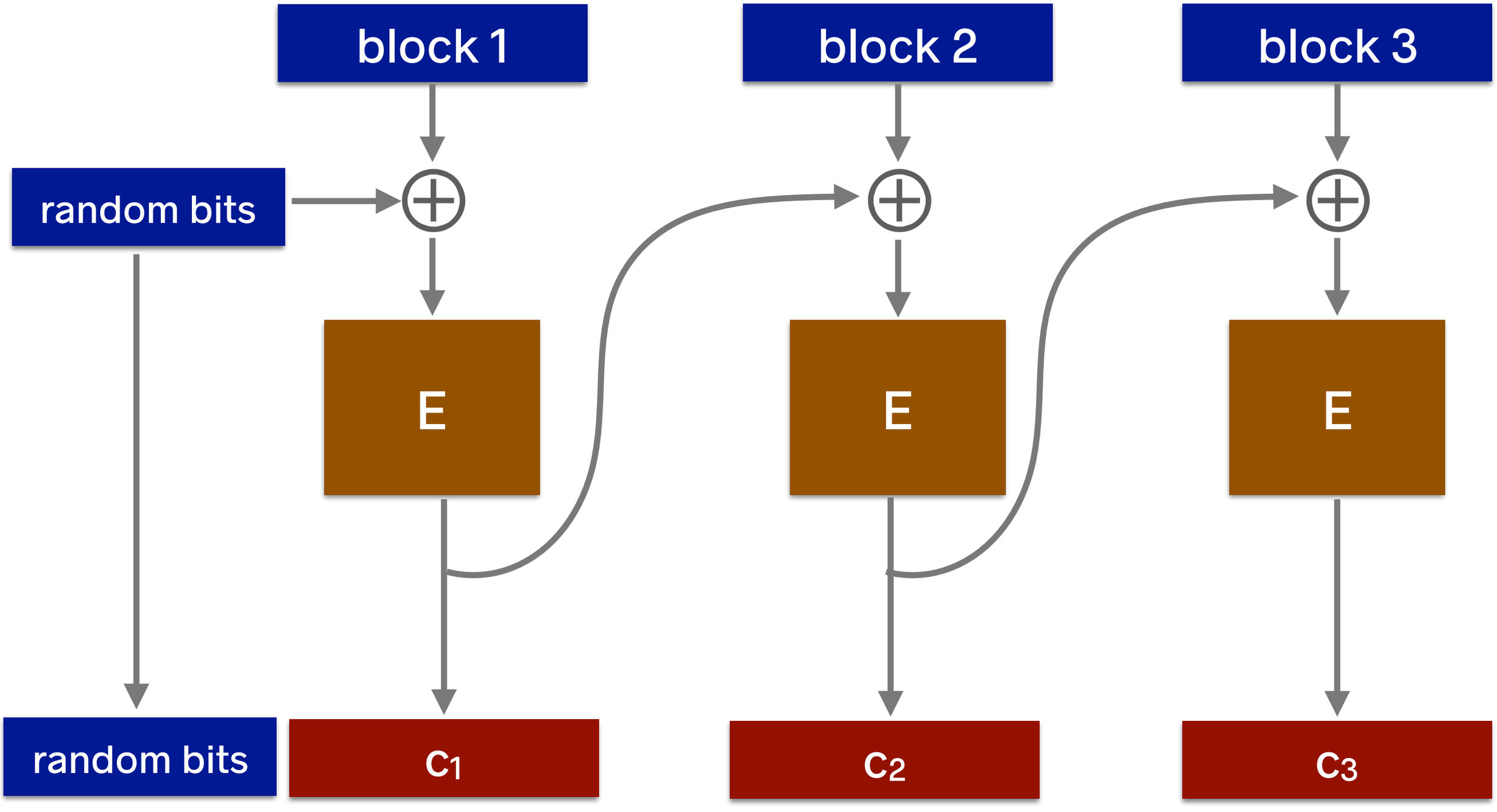
CBC encryption



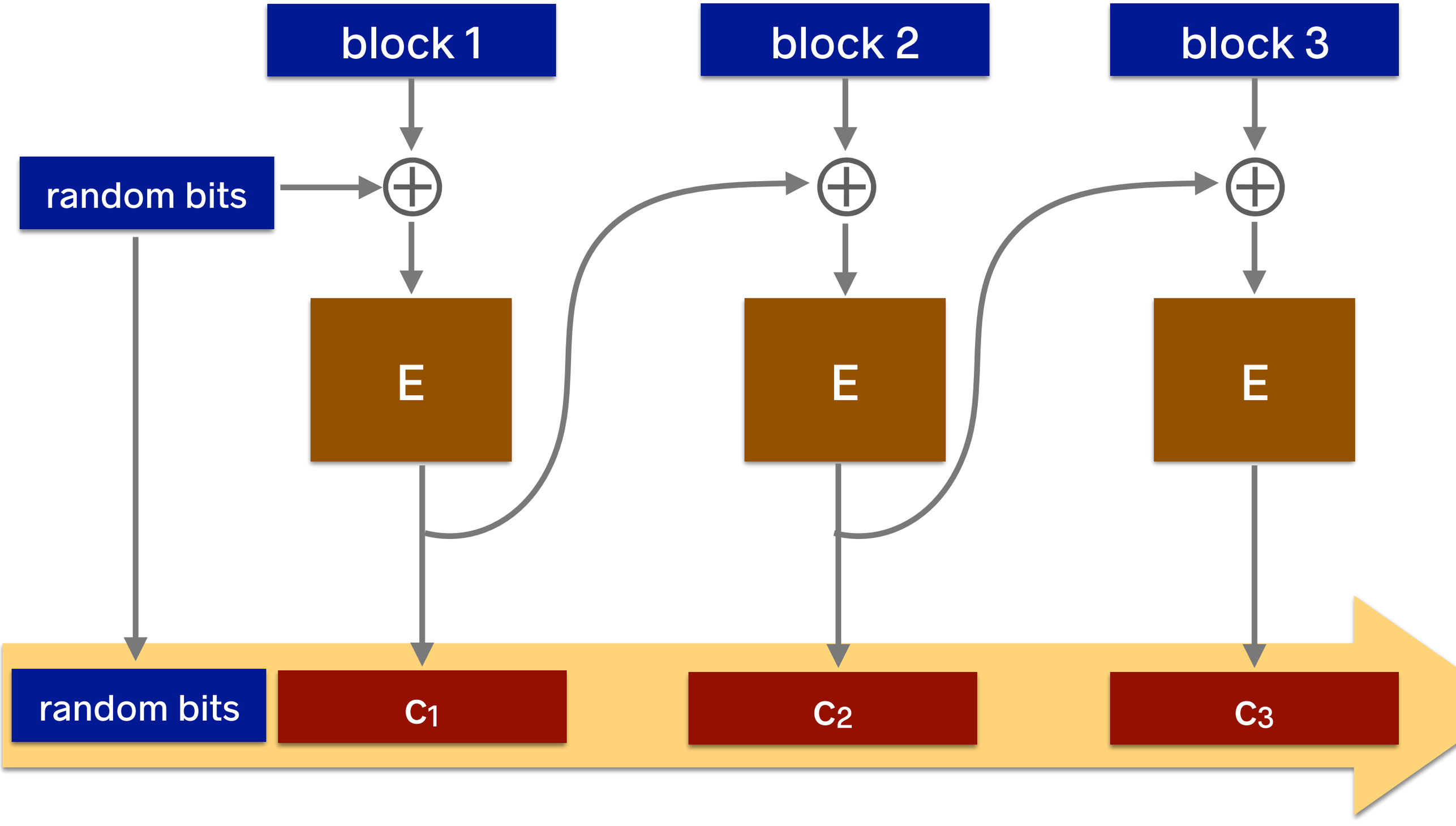
CBC encryption



CBC encryption

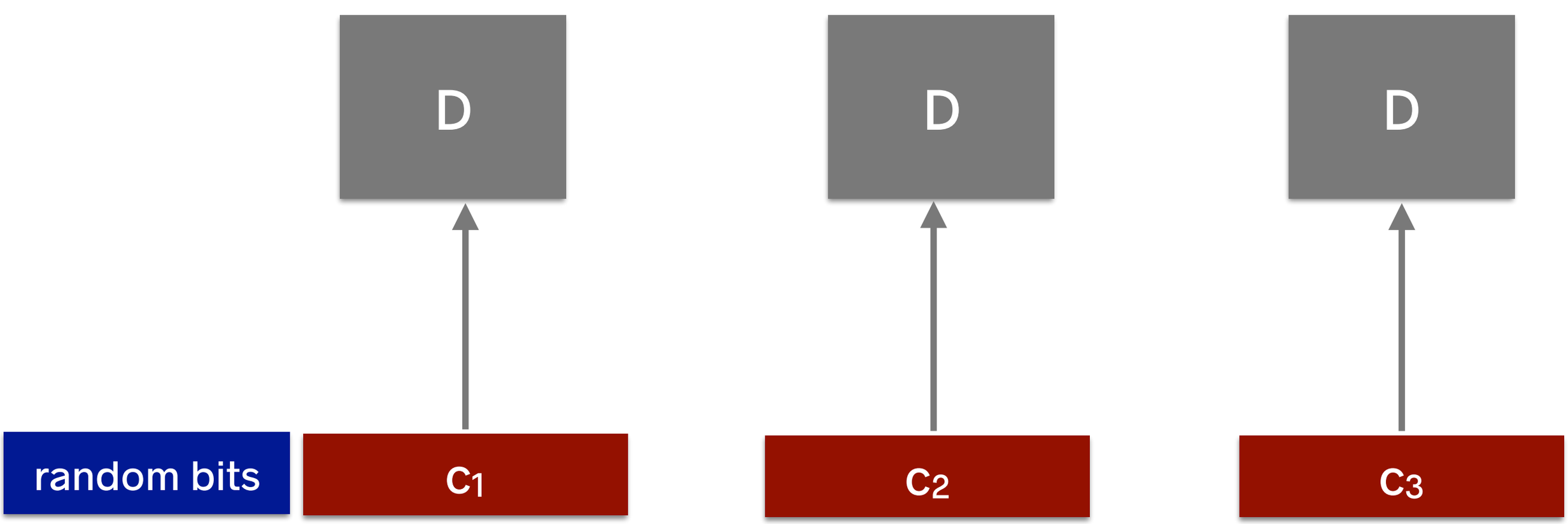


CBC encryption

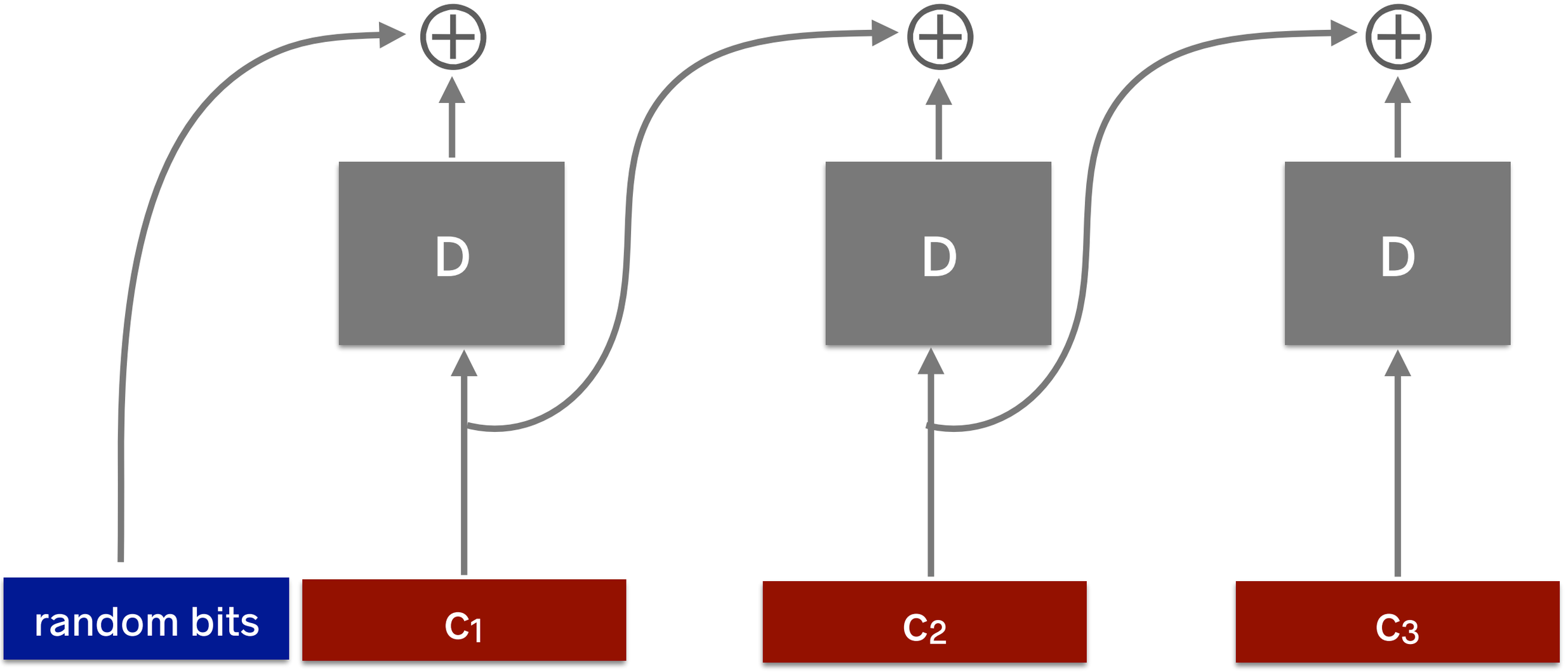


CBC decryption

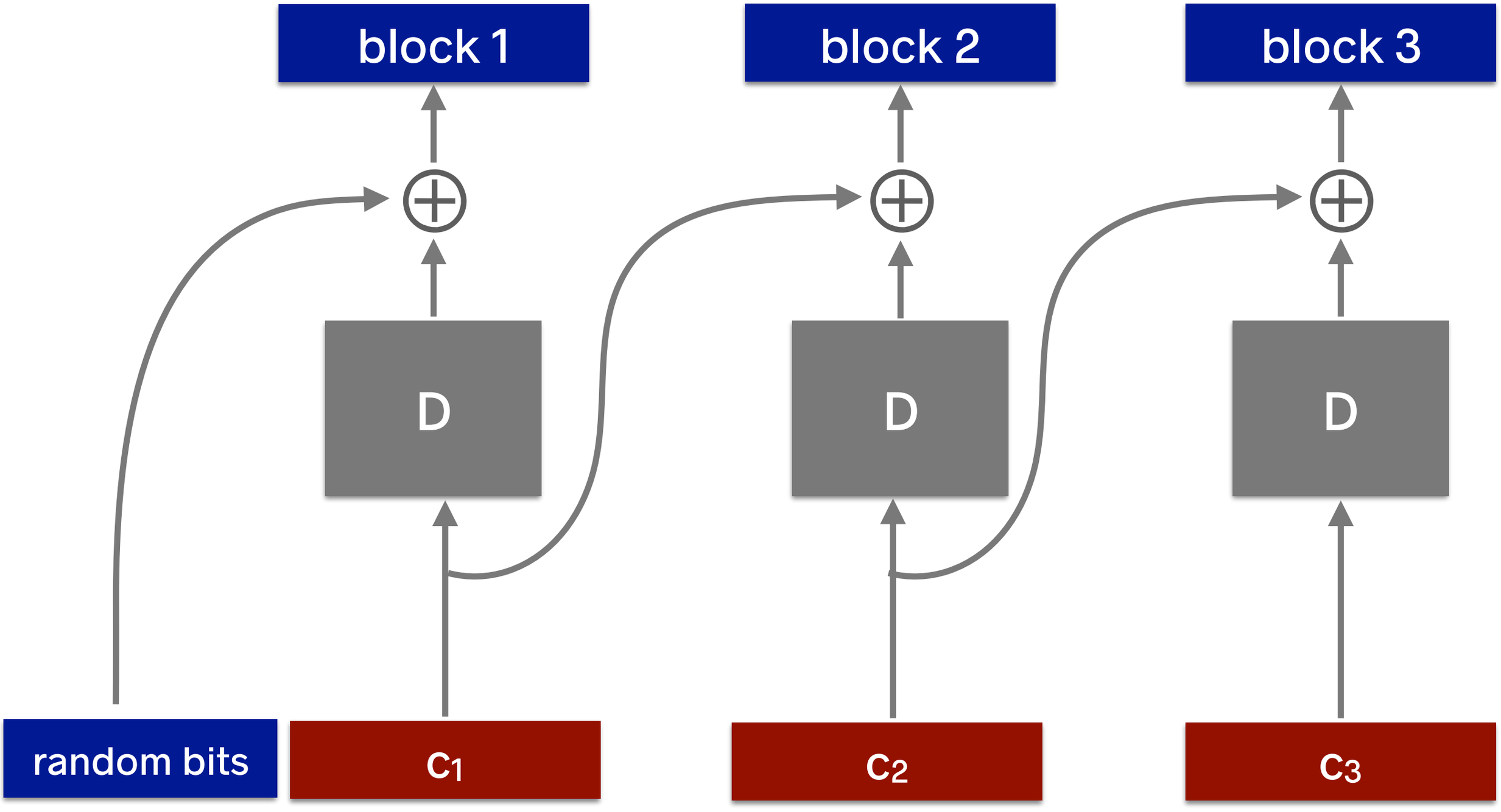
CBC decryption



CBC decryption



CBC decryption



Why is CBC mode inconvenient?

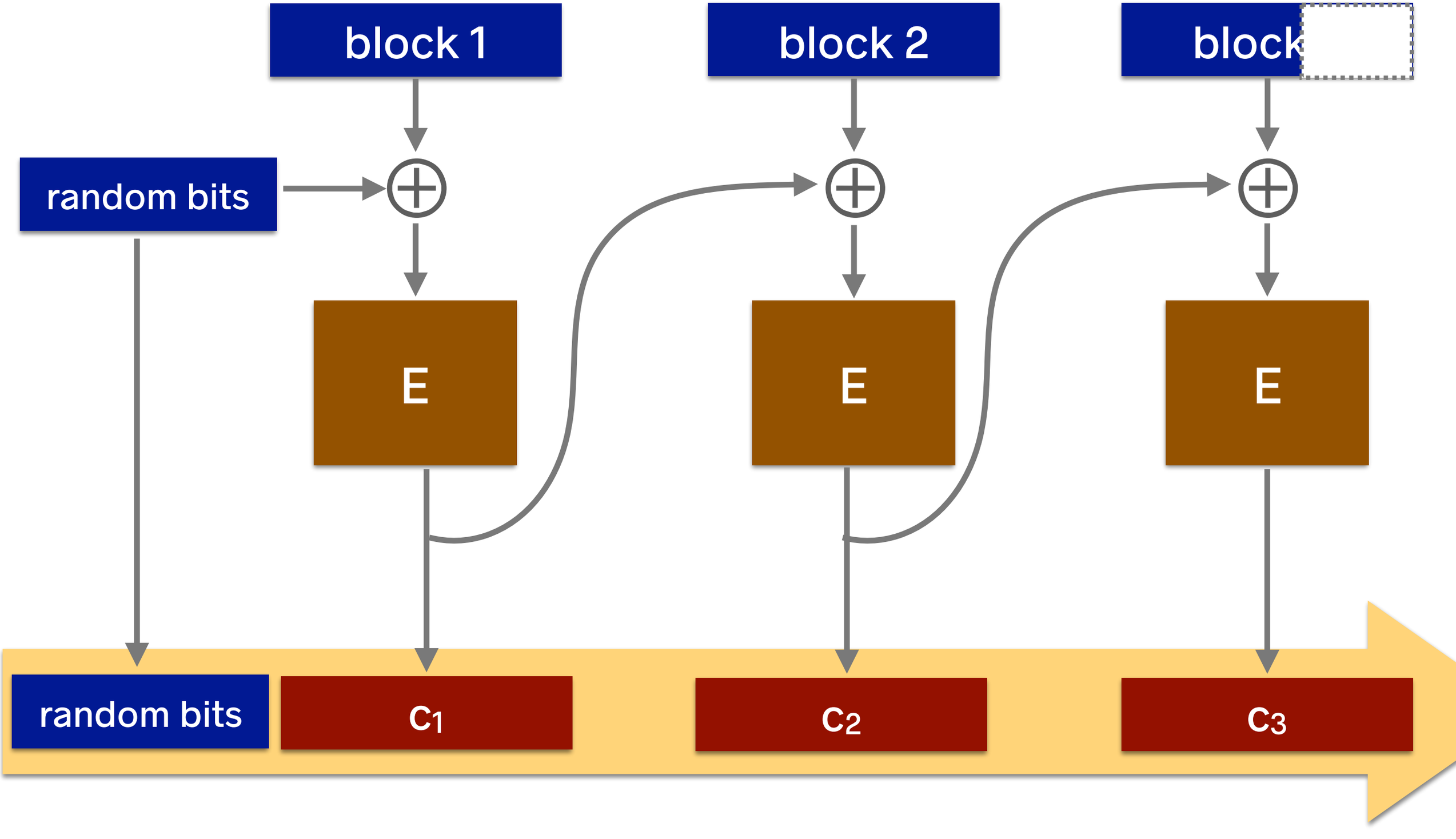
Why is CBC mode inconvenient?

1. Computers now have many cores
   - CBC encryption is a sequential operation
   - Can't make effective use of multicore hardware
     -> Modern ciphers focus on parallelism

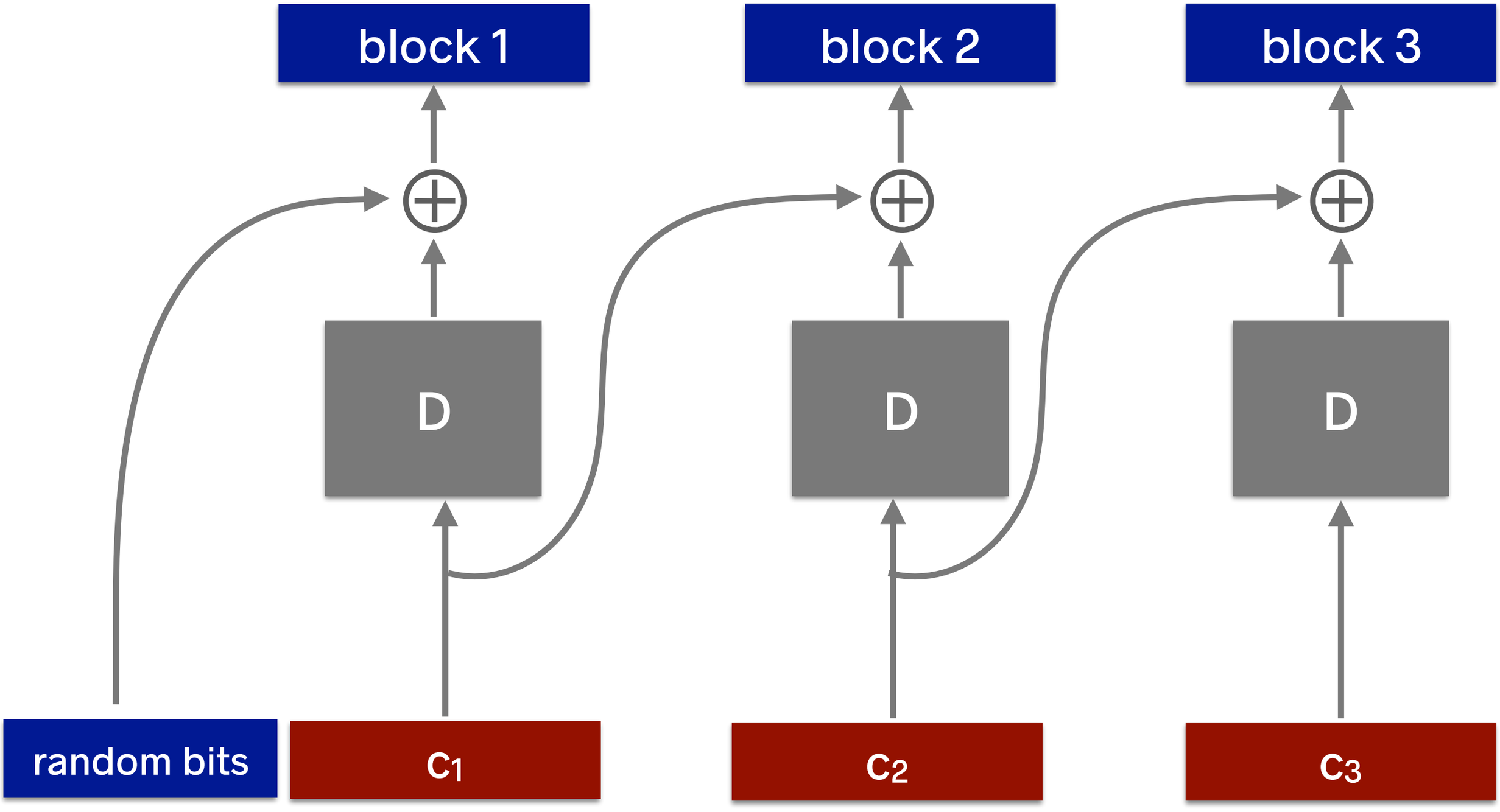
Why is CBC mode inconvenient?

1. Computers now have many cores
   - CBC encryption is a sequential operation
   - Can't make effective use of multicore hardware
     -> Modern ciphers focus on parallelism
2. CBC can only encrypt messages whose bitlength is a multiple of 128

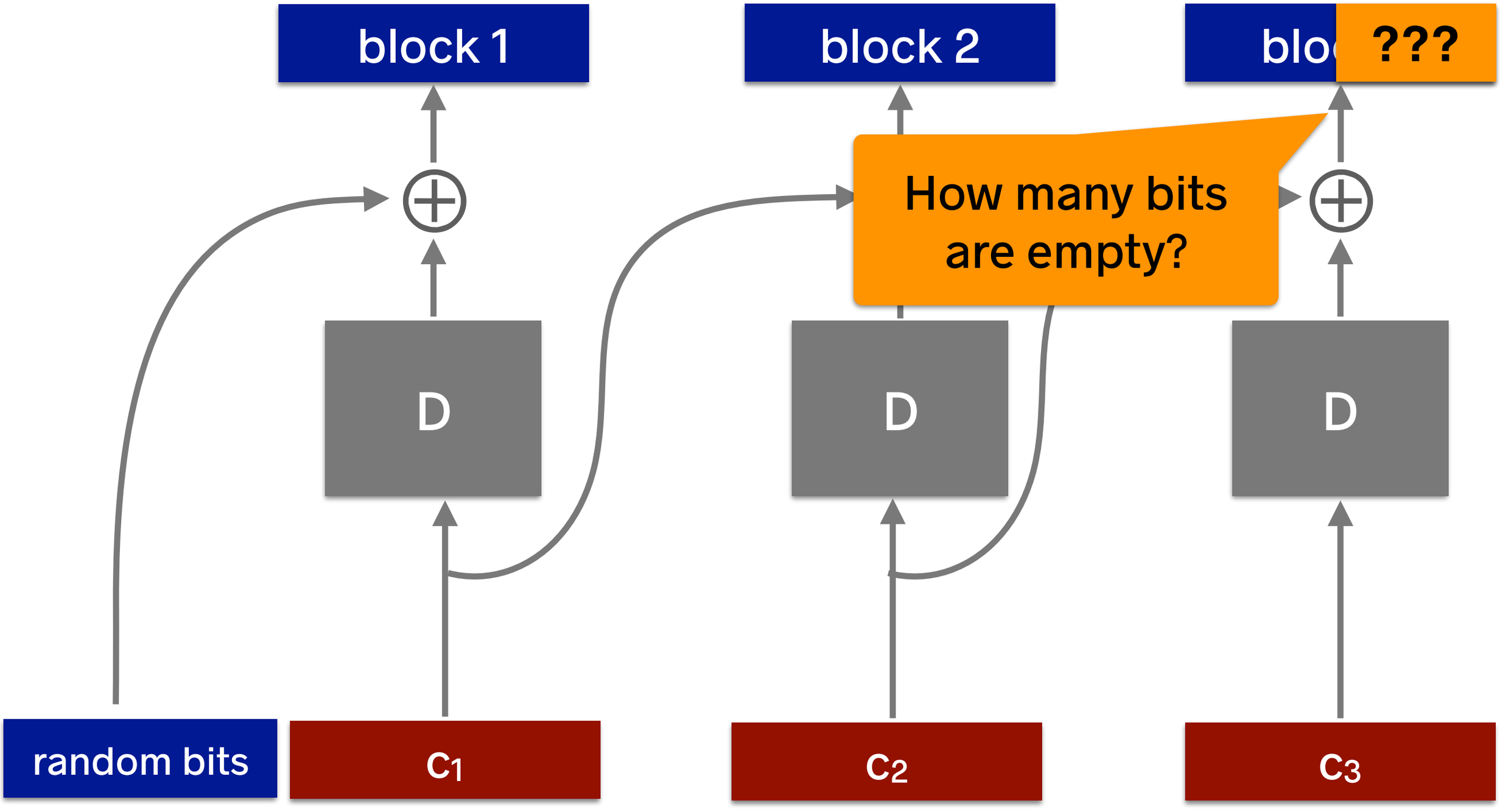
Message must be multiple of block size



Message must be multiple of block size



Message must be multiple of block size

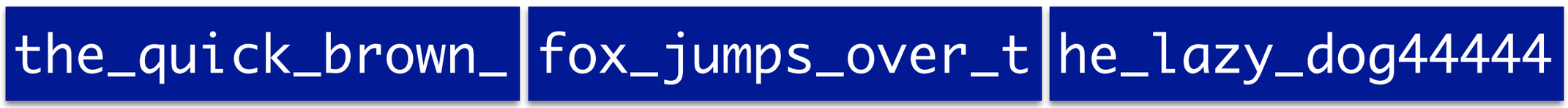


CBC Padding

the_quick_brown_	fox_jumps_over_t	he_lazy_dog44444

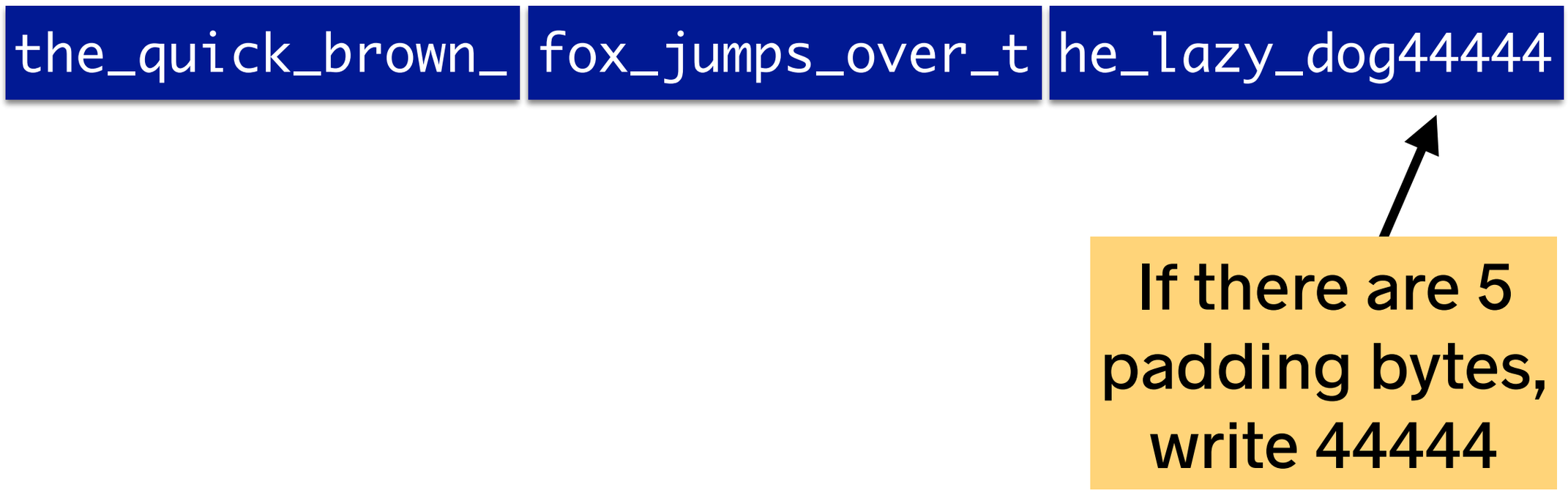
CBC Padding

Obvious fix: add some padding to the message



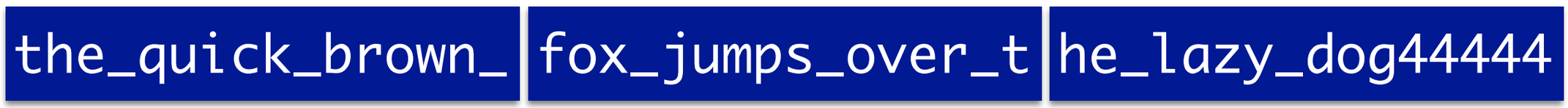
CBC Padding

Obvious fix: add some padding to the message



CBC Padding

Obvious fix: add some padding to the message



CBC Padding

Obvious fix: add some padding to the message

the_quick_brown_	fox_jumps_over_t	he_lazy_dog44444

Modified CBC decryption

1. Decrypt padded message as in normal CBC
2. Read last byte of message – say it has value N
3. Chop off the last N bytes of the message
   – Make sure the last N bytes are valid padding: NN…NN
   – If not, output “invalid padding”
4. Output message

Padding-oracle attack

This padding scheme creates big security problems
-> A “man-in-the-middle” attacker can learn pieces of the encrypted message [Vaudenay’02]

This has led to real-world vulnerabilities in TLS
- Lucky13 [AlFardan&Patterson ’13]
- POODLE [Möller&others’14]
- also in loads of other software (Ruby on Rails, etc.)

Attack set-up

- Your mail client (Outlook) sends your email username and password to the mail server over TLS.
  - This happens every ~2 minutes.
  - The mail client retries upon failure.
- The mail client sends the same secret message many times

Attack set-up

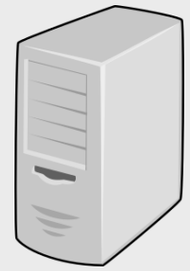
- Your mail client (Outlook) sends your email username and password to the mail server over TLS.
  - This happens every ~2 minutes.
  - The mail client retries upon failure.
- The mail client sends the same secret message many times

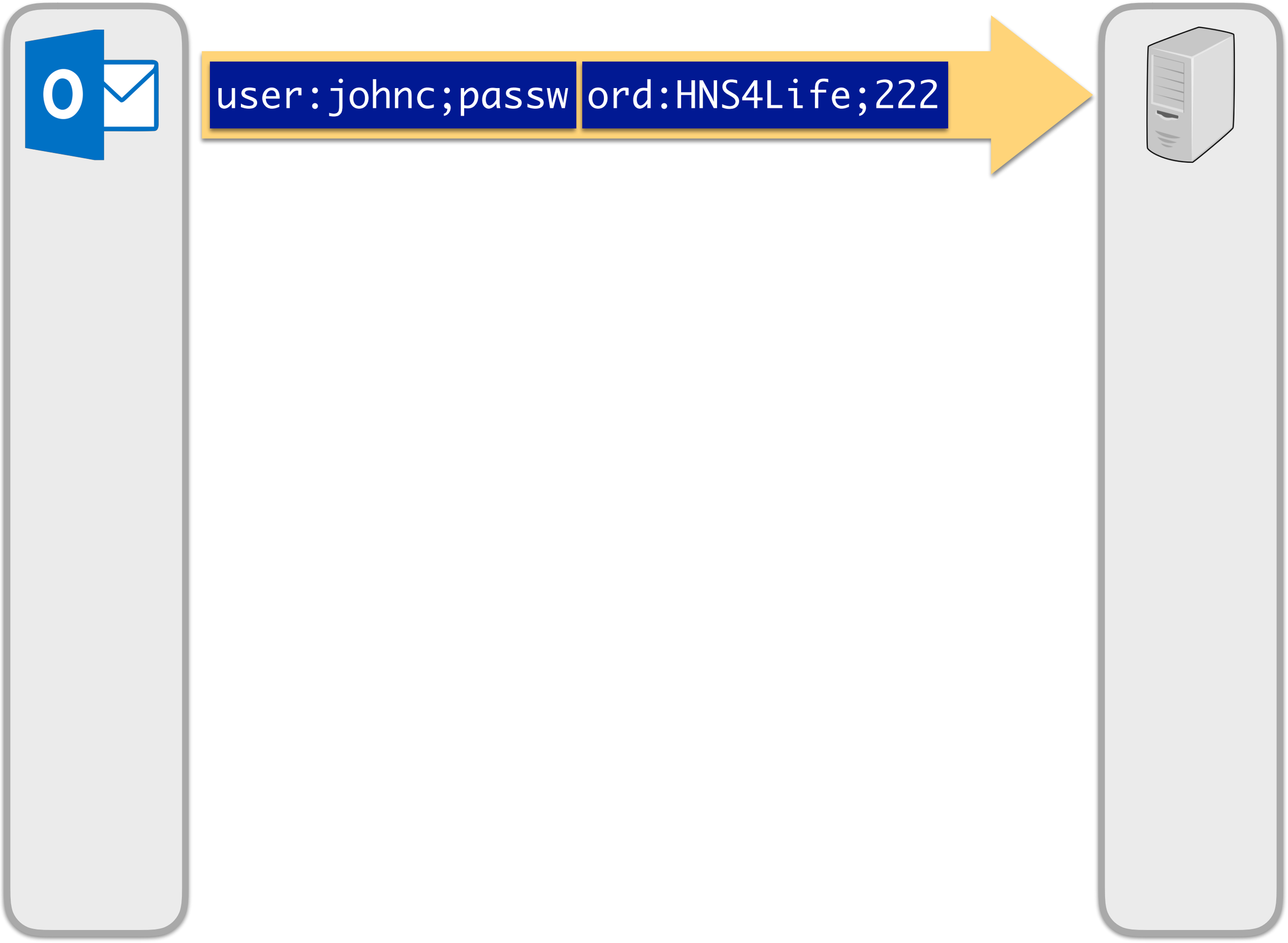
Attack set-up

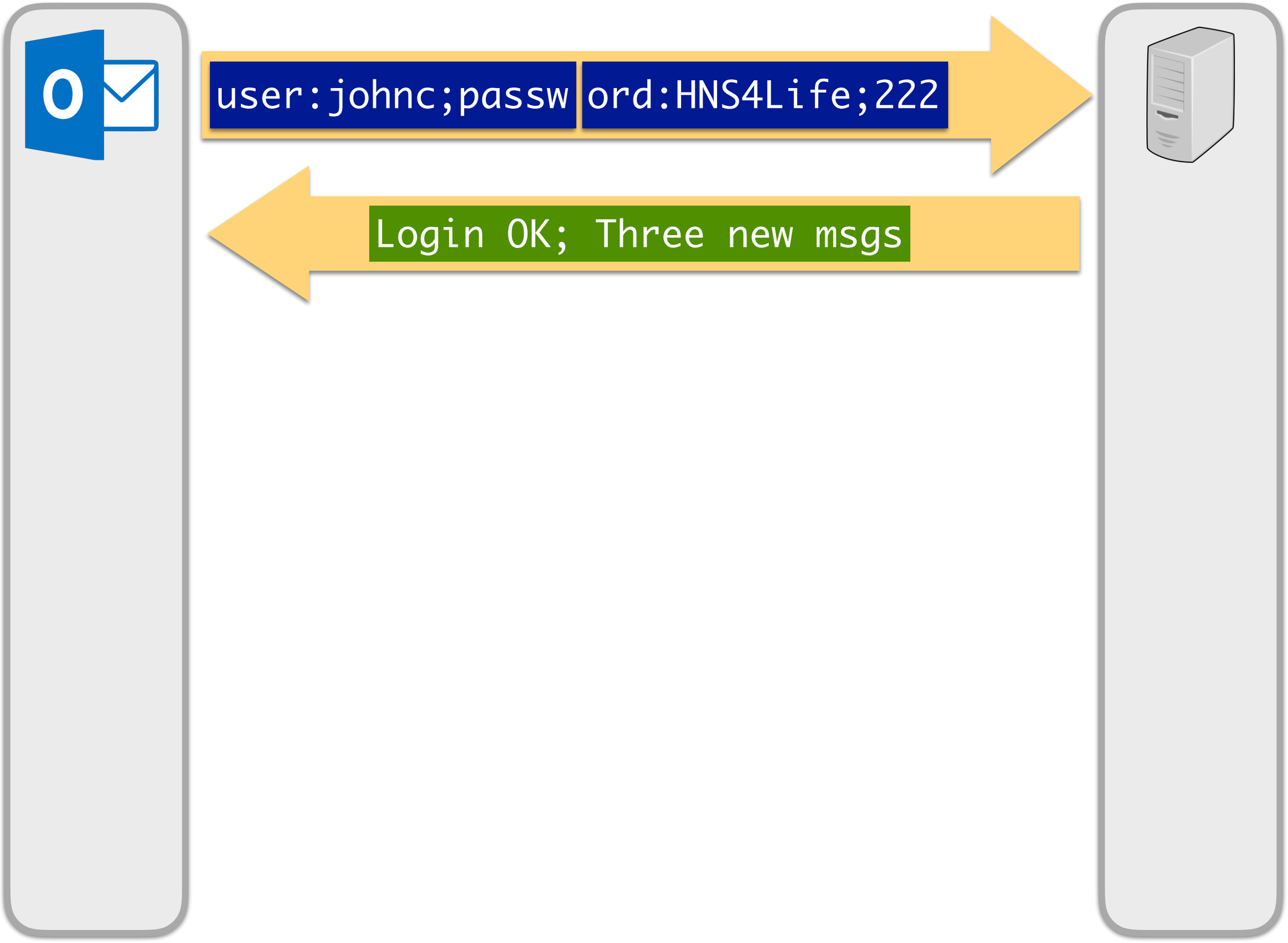
- Your mail client (Outlook) sends your email username and password to the mail server over TLS.
  - This happens every ~2 minutes.
  - The mail client retries upon failure.
- The mail client sends the same secret message many times

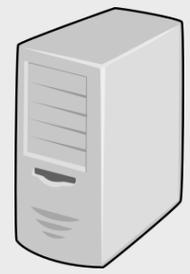
Attack set-up

- Your mail client (Outlook) sends your email username and password to the mail server over TLS.
  - This happens every ~2 minutes.
  - The mail client retries upon failure.
- The mail client sends the same secret message many times




user:johnc;passw
ord:HNS4Life;222

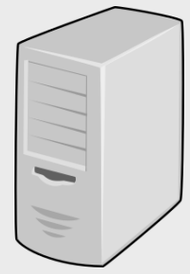

user:johnc;passw
ord:HNS4Life;222
Login OK; Three new msgs



user:johnc;passw ord:HNS4Life;222

Login OK; Three new msgs

n\<)0;q<SPav_oTo ord:HNS4Life;222

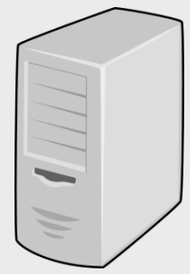


user:johnc;passw | ord:HNS4Life;222

Login OK; Three new msgs

n\<)O;q<SPav_oTo | ord:HNS4Life;222

Invalid request; | Please try again.



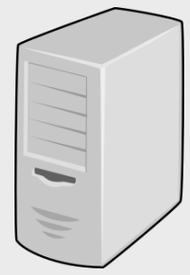
user:johnc;passw | ord:HNS4Life;222

Login OK; Three new msgs

n\<)0;q<SPav_oTo | ord:HNS4Life;222

Invalid request; | Please try again.

n\<)0;q<SPav_oTo | ord:HNS4Life;**9X2**



user:johnc;passw | ord:HNS4Life;222

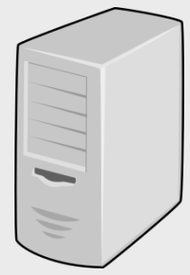
Login OK; Three new msgs

n\<)0;q<SPav_oTo | ord:HNS4Life;222

Invalid request; | Please try again.

n\<)0;q<SPav_oTo | ord:HNS4Life;**9X2**

Padding error.



user:johnc;passw | ord:HNS4Life;222

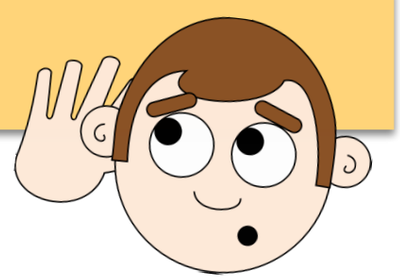
Login OK; Three new msgs

n\<)0;q<SPav_oTo | ord:HNS4Life;222

Invalid request; | Please try again.

n\<)0;q<SPav_oTo | ord:HNS4Life;**9X2**

Padding error.



Attack idea

Attack idea

- The server behaves differently when
  (a) request is invalid, versus
  (b) padding is incorrect.
- This leaks a bit of information about the plaintext!

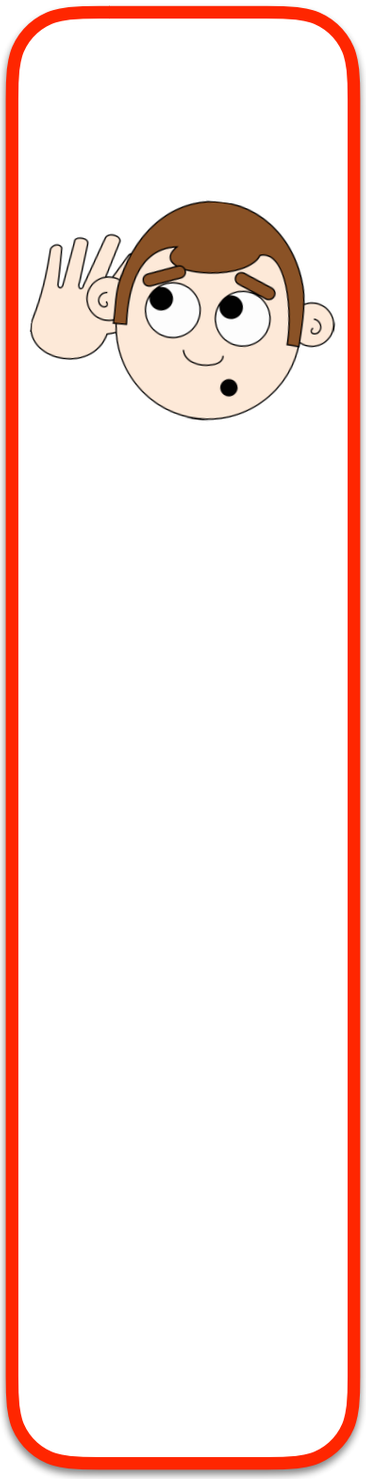
Attack idea

- The server behaves differently when
  (a) request is invalid, versus
  (b) padding is incorrect.
- This leaks a bit of information about the plaintext!

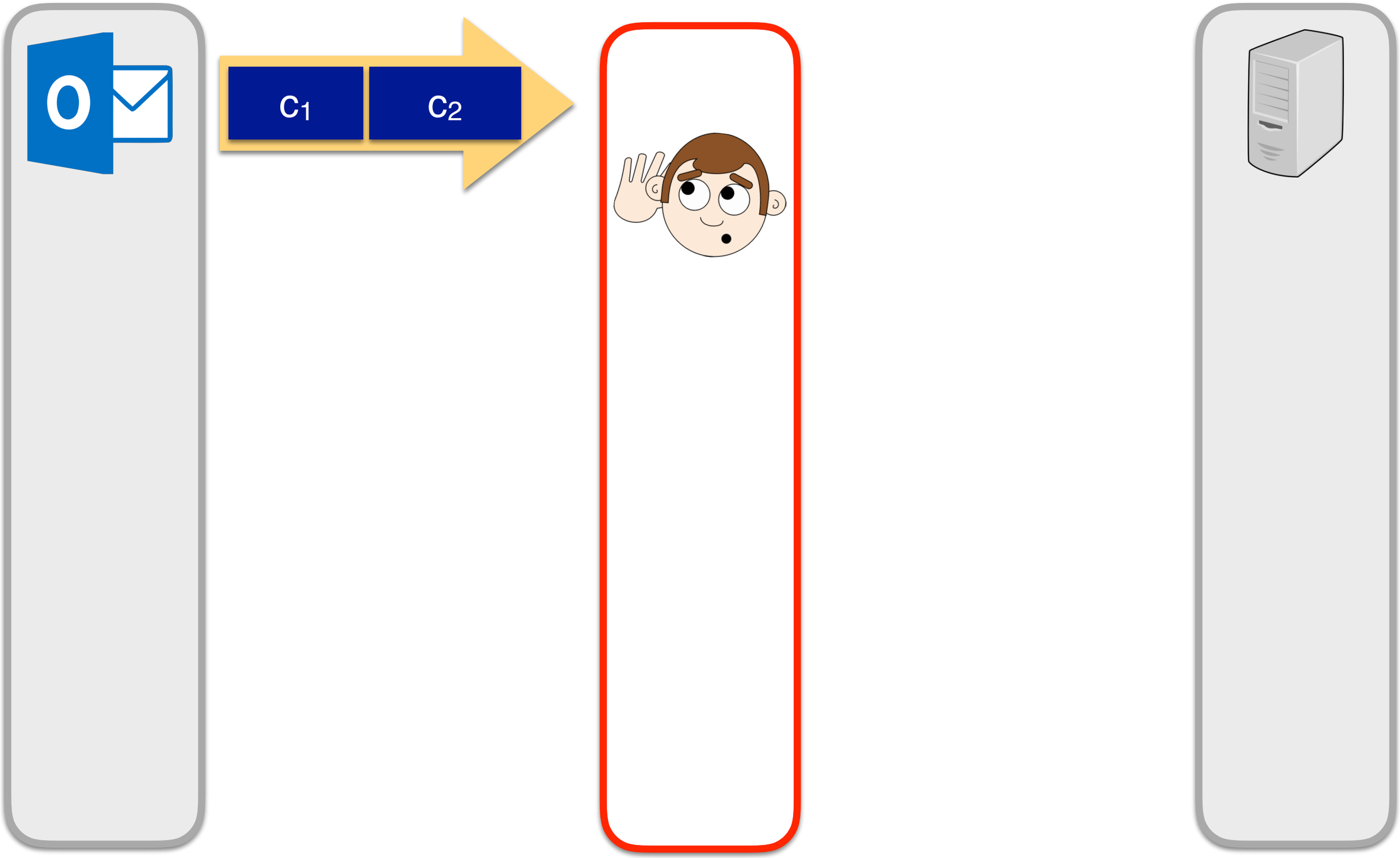
Attack idea

1. Tweak a few bits of the client's message.
2. Observe the server's response to learn 1 bit of information about the encrypted password.
3. Repeat until learn entire password.

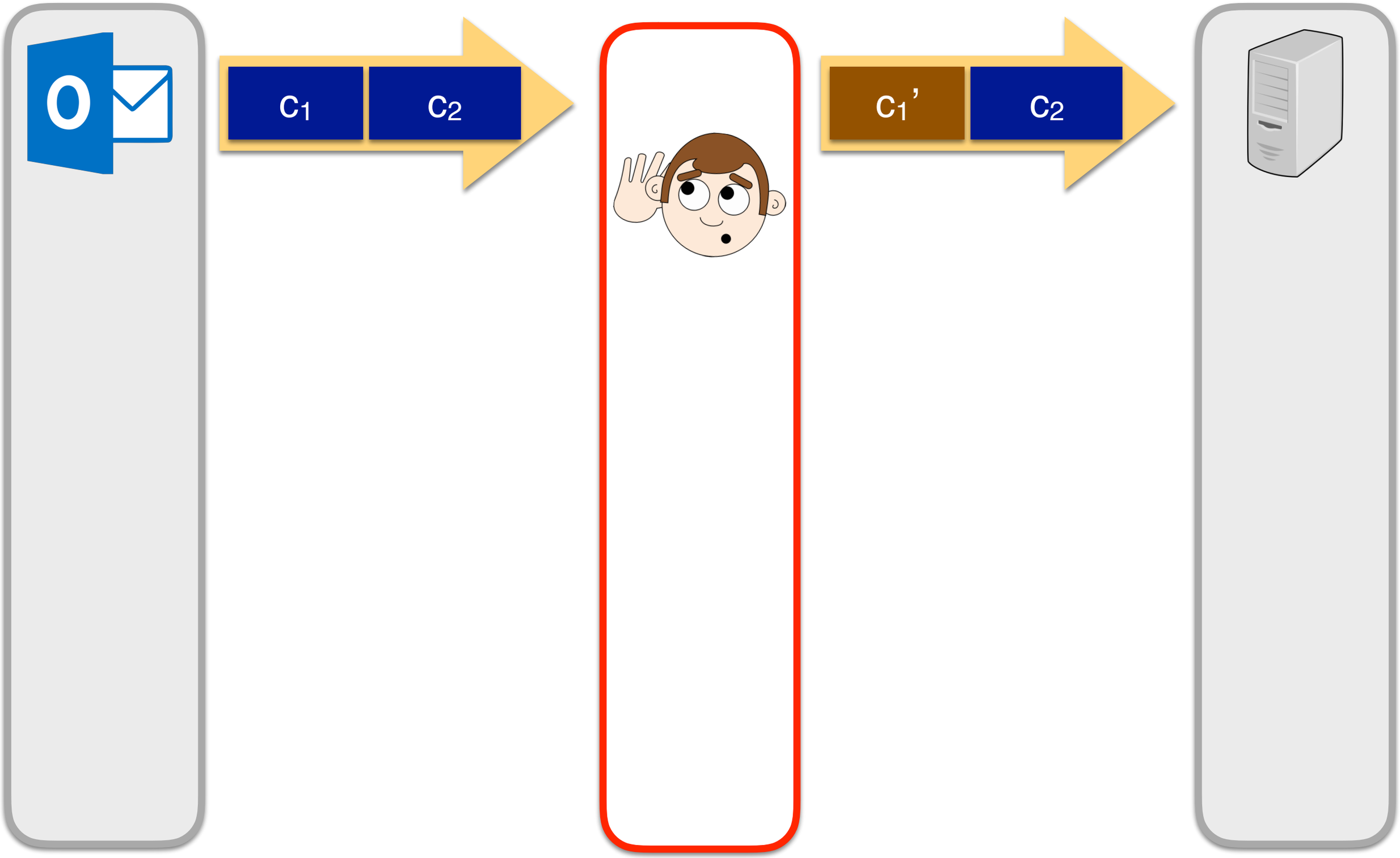
Attack idea



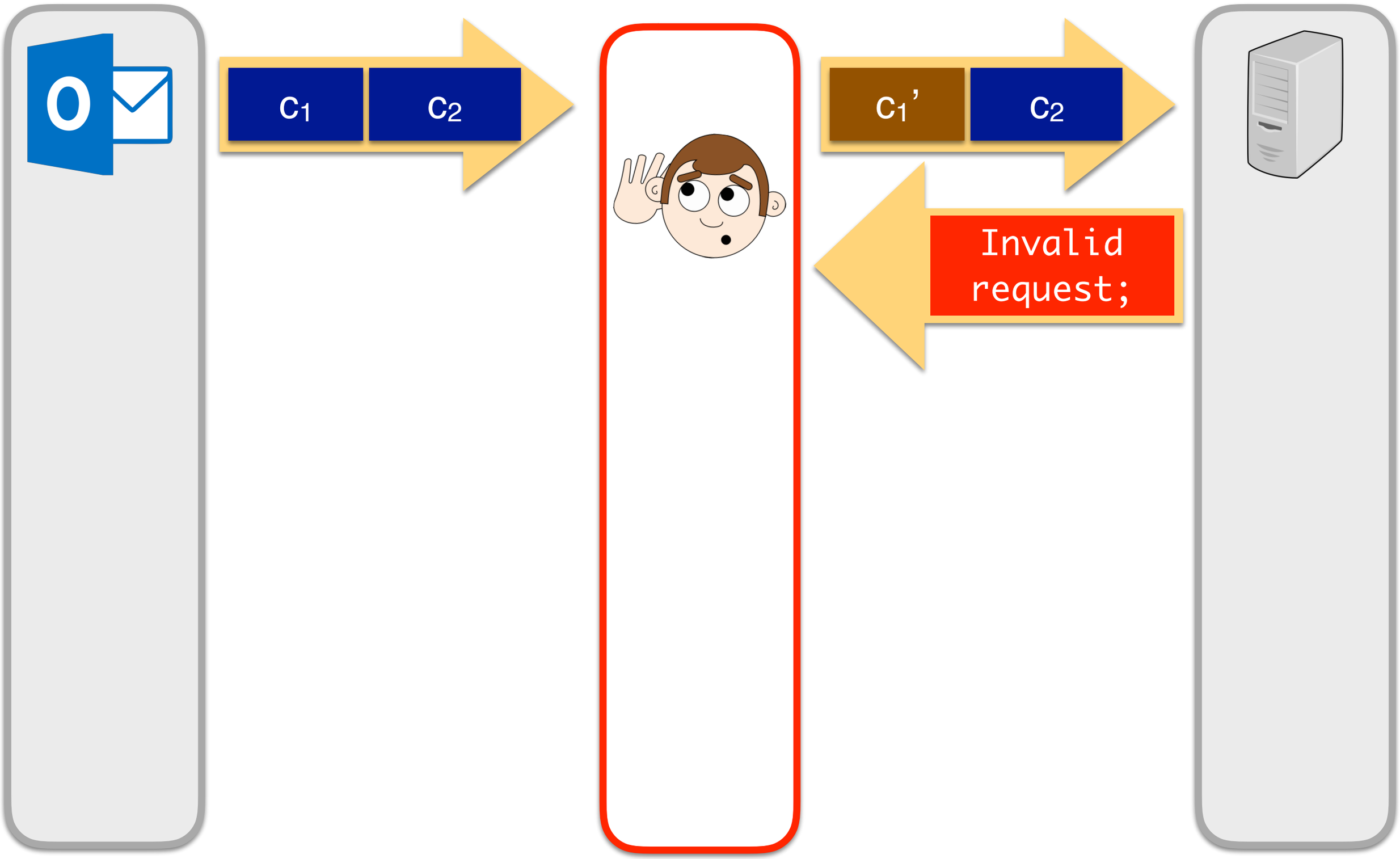
Attack idea



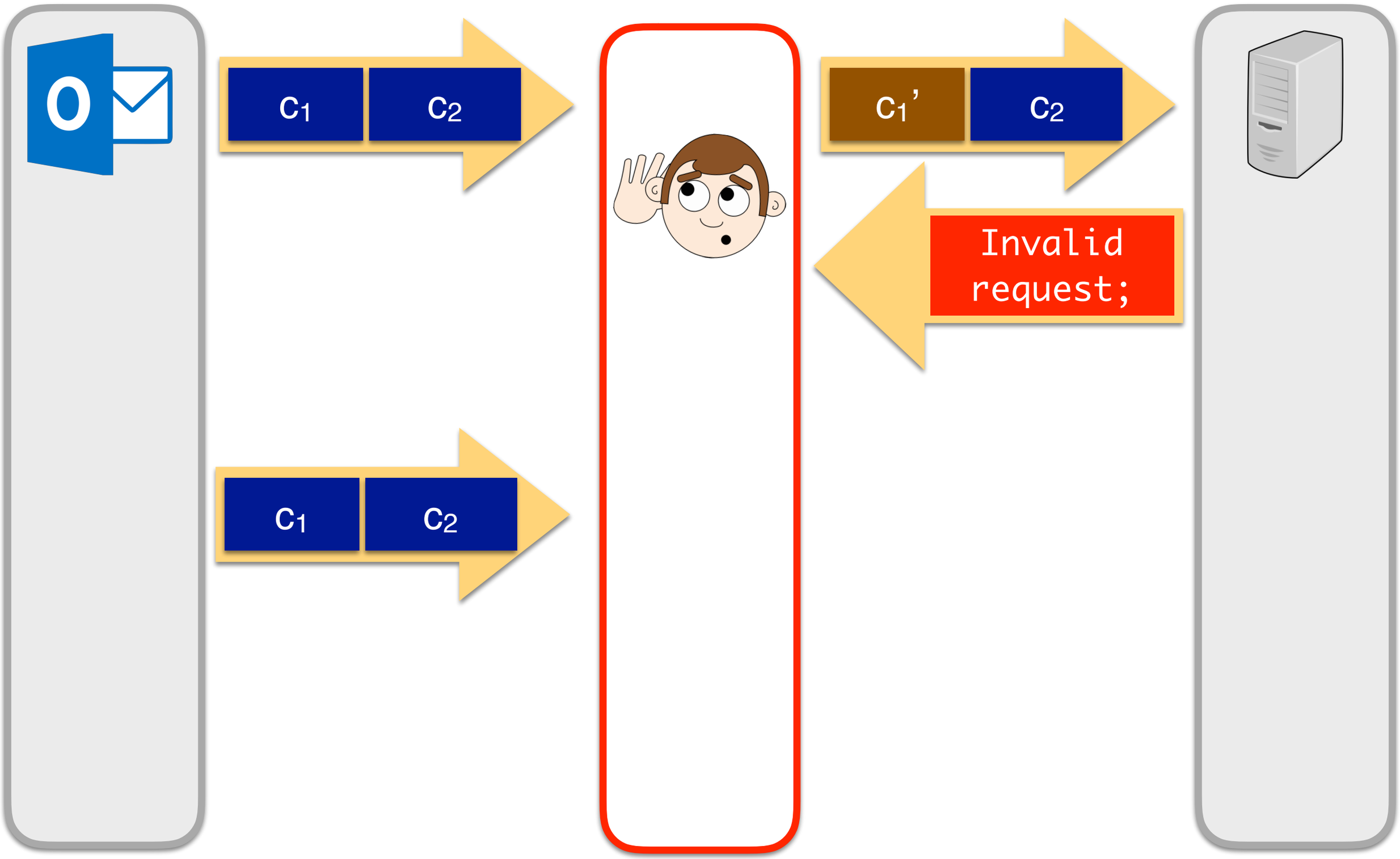
Attack idea



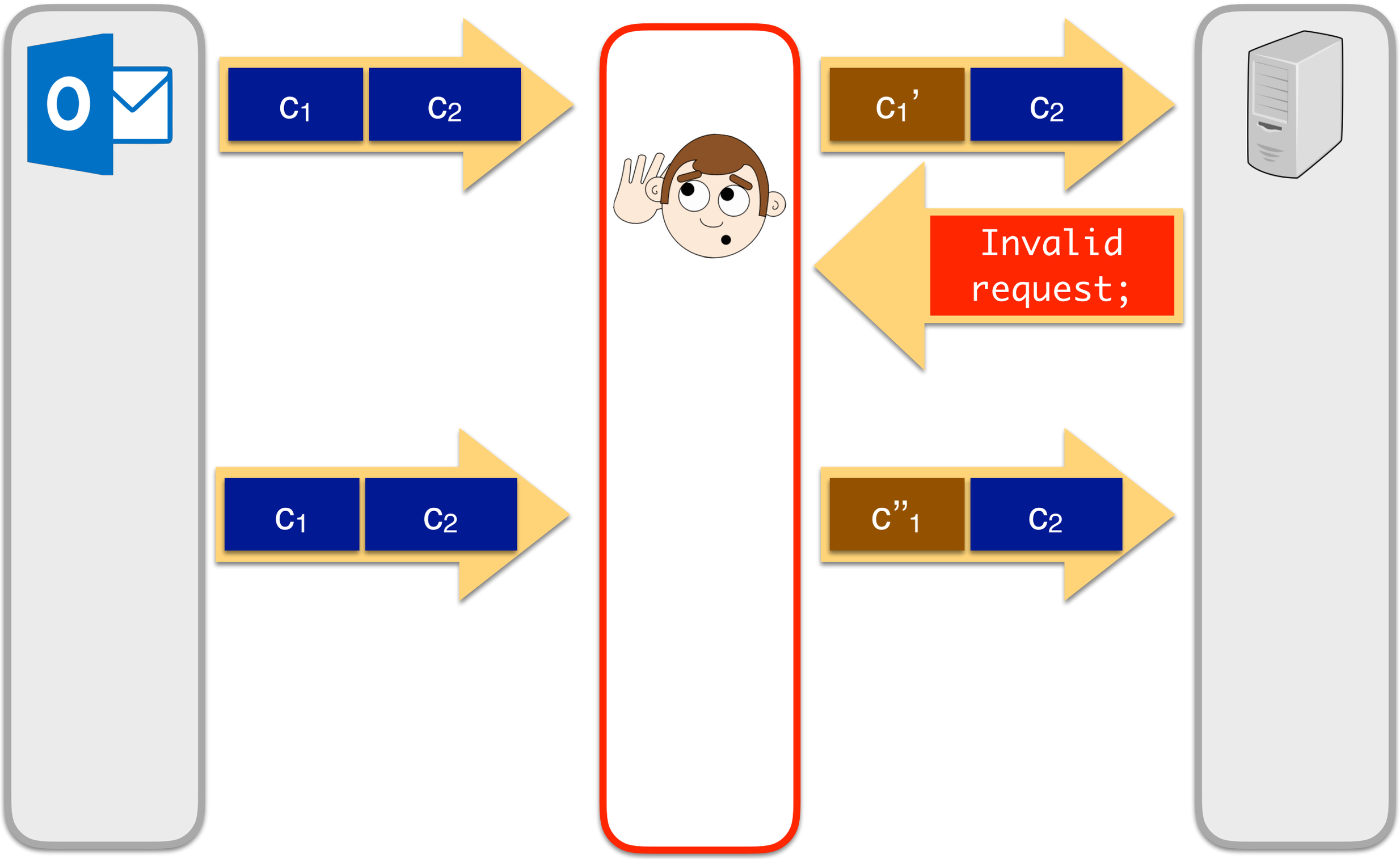
Attack idea



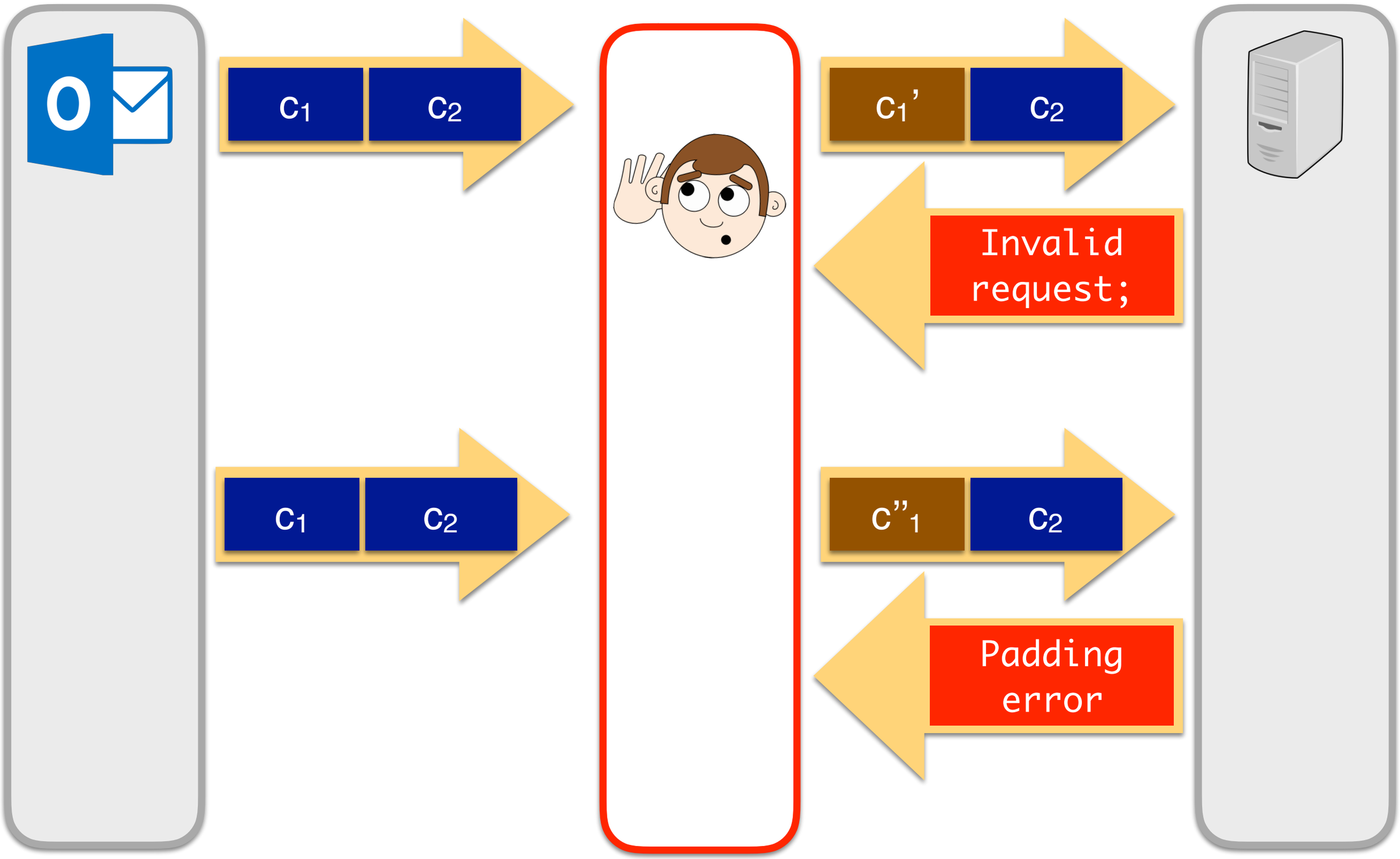
Attack idea



Attack idea

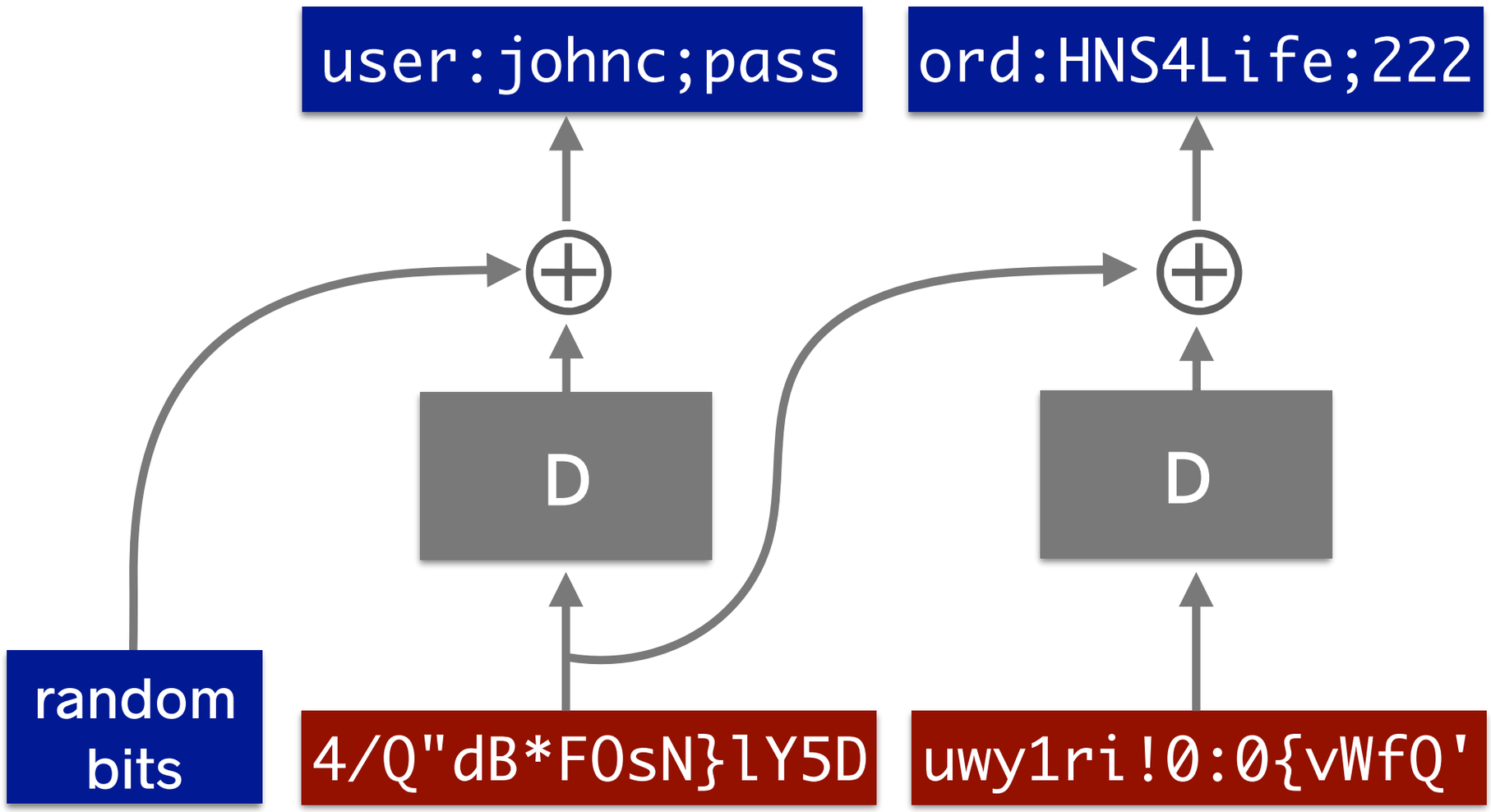


Attack idea



Recall CBC decryption

Message

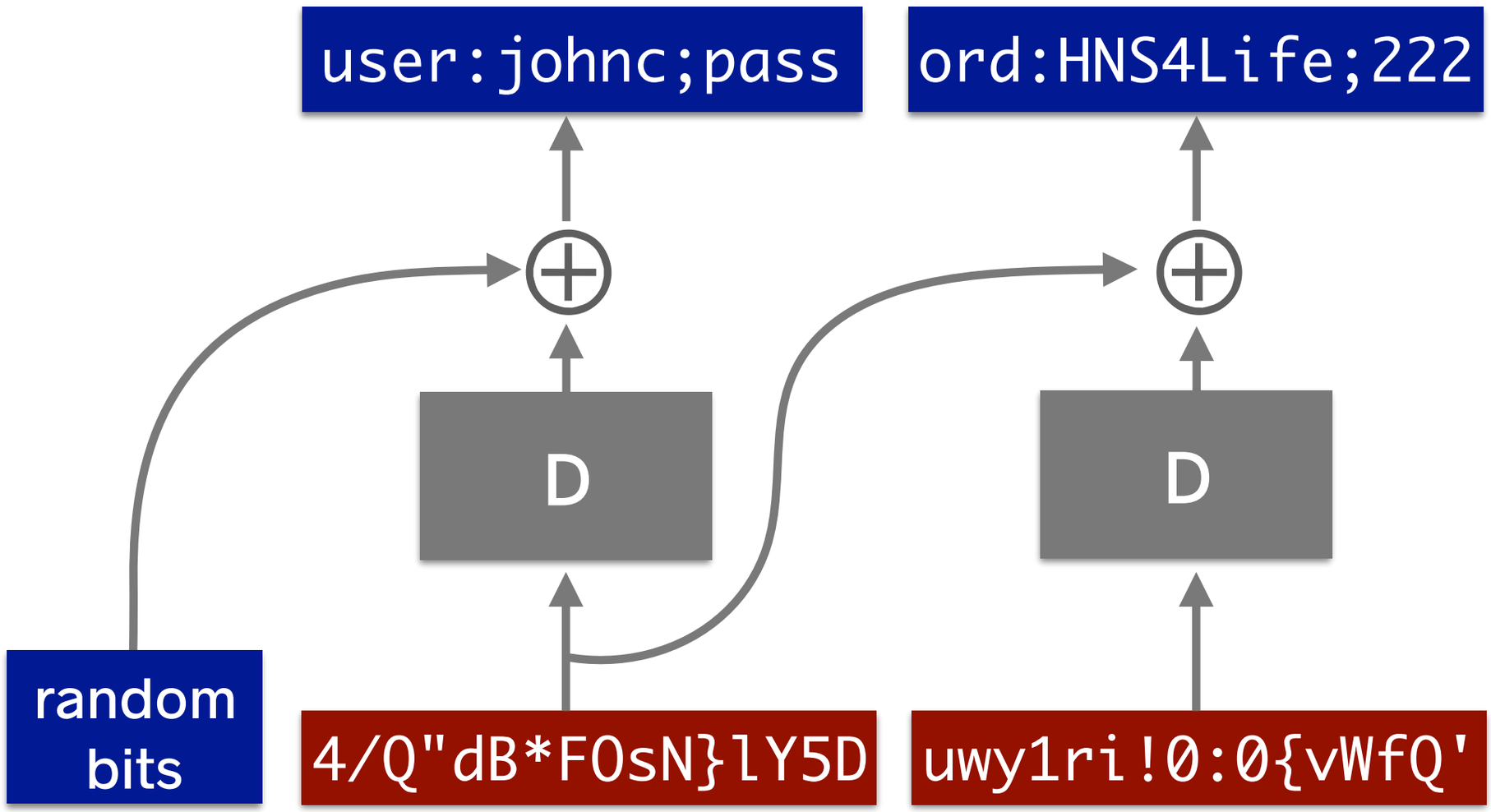


Ciphertext

Server response

Recall CBC decryption

Message



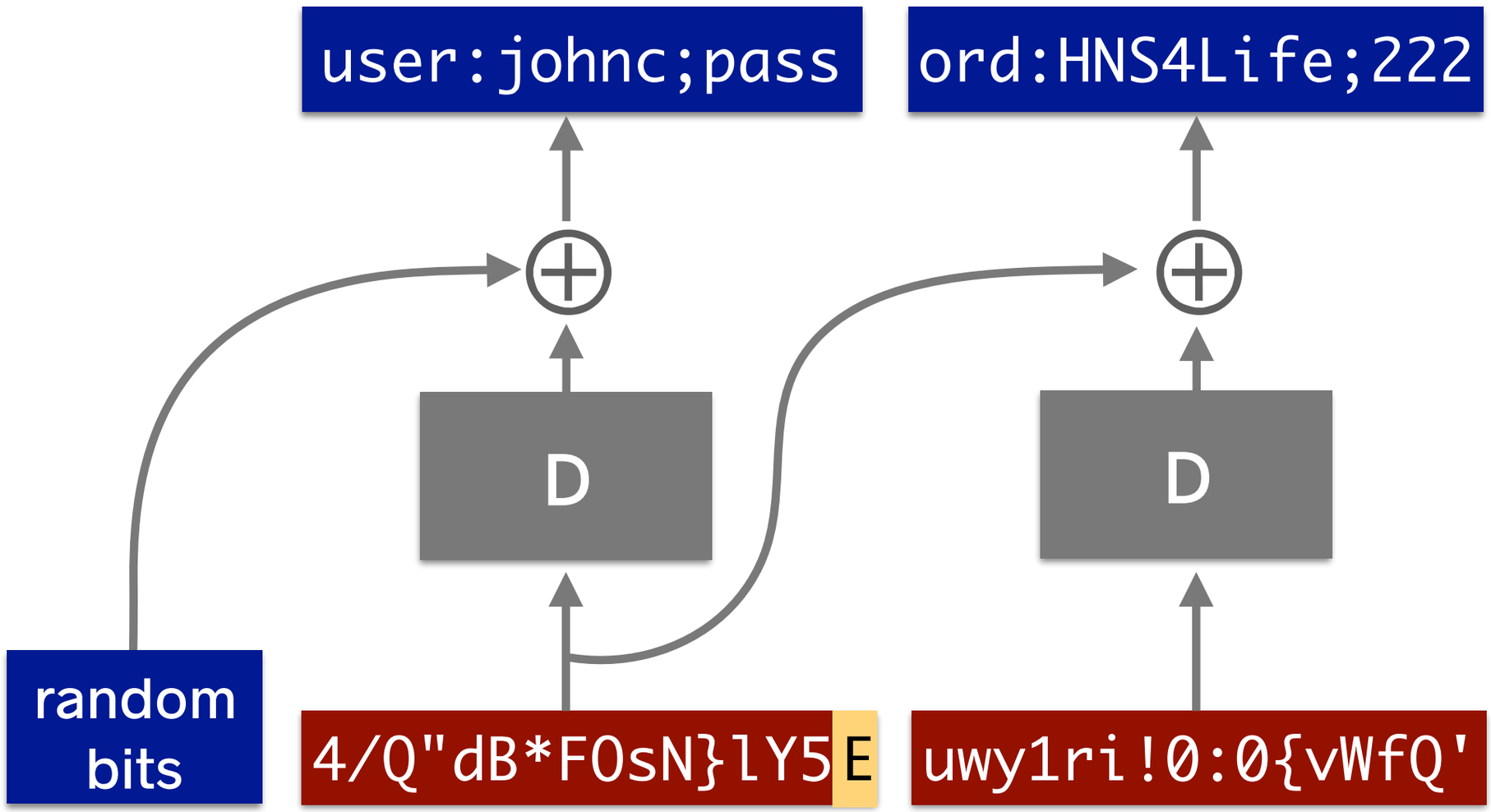
Ciphertext

Server response

```
Login OK; Three new msgs
```

Recall CBC decryption

Message

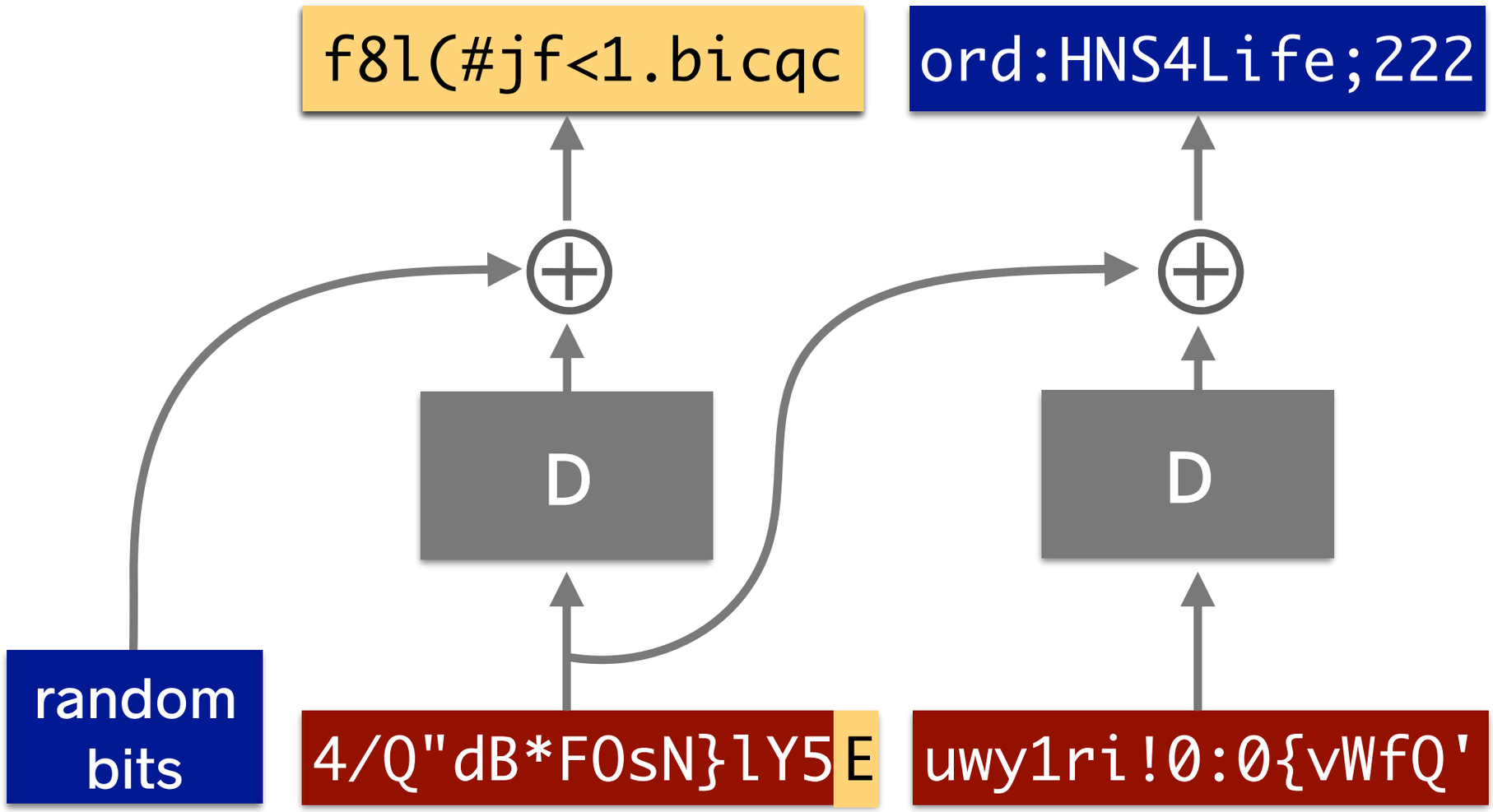


Ciphertext

Server response

Recall CBC decryption

Message

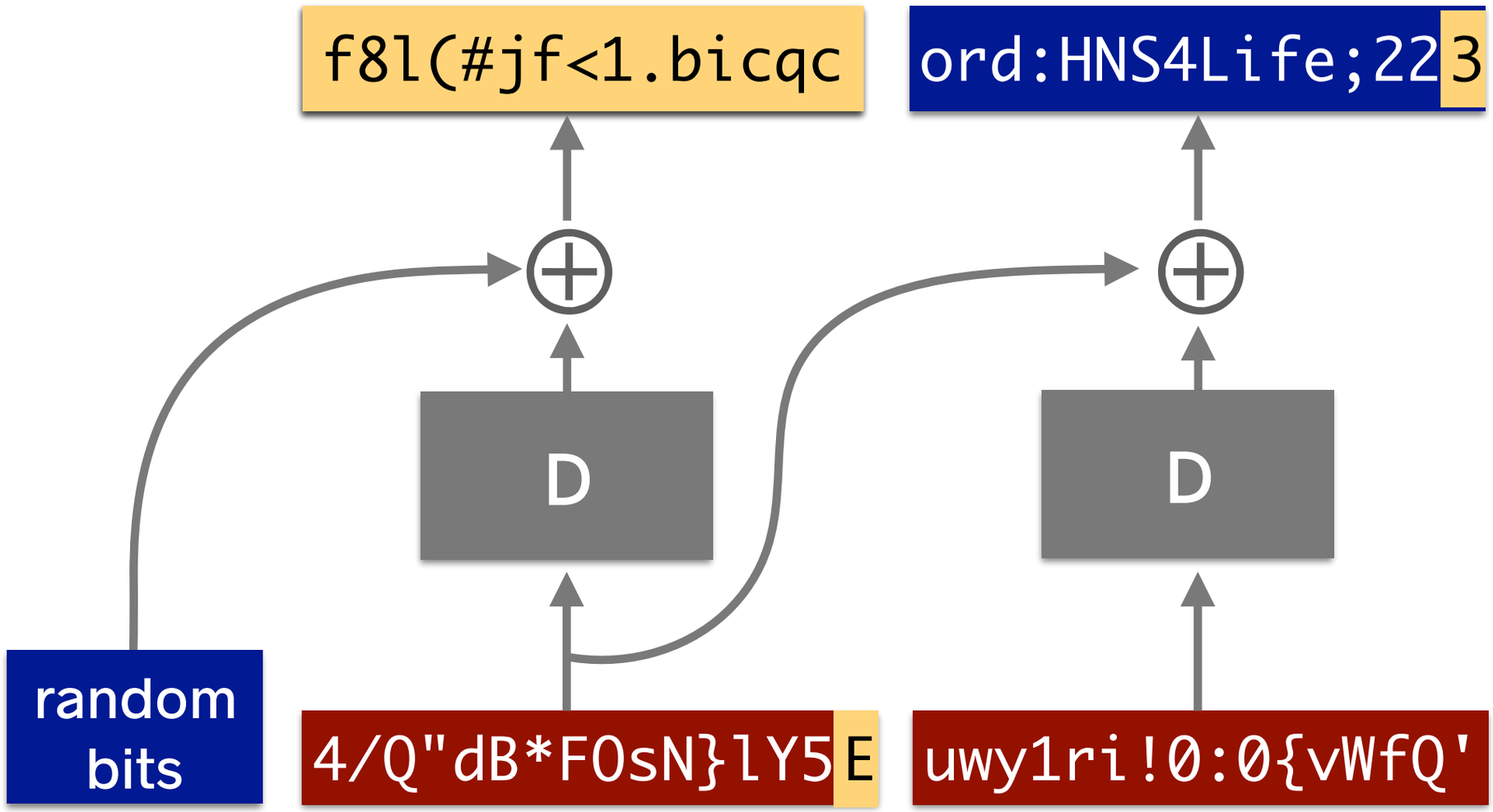


Ciphertext

Server response

Recall CBC decryption

Message

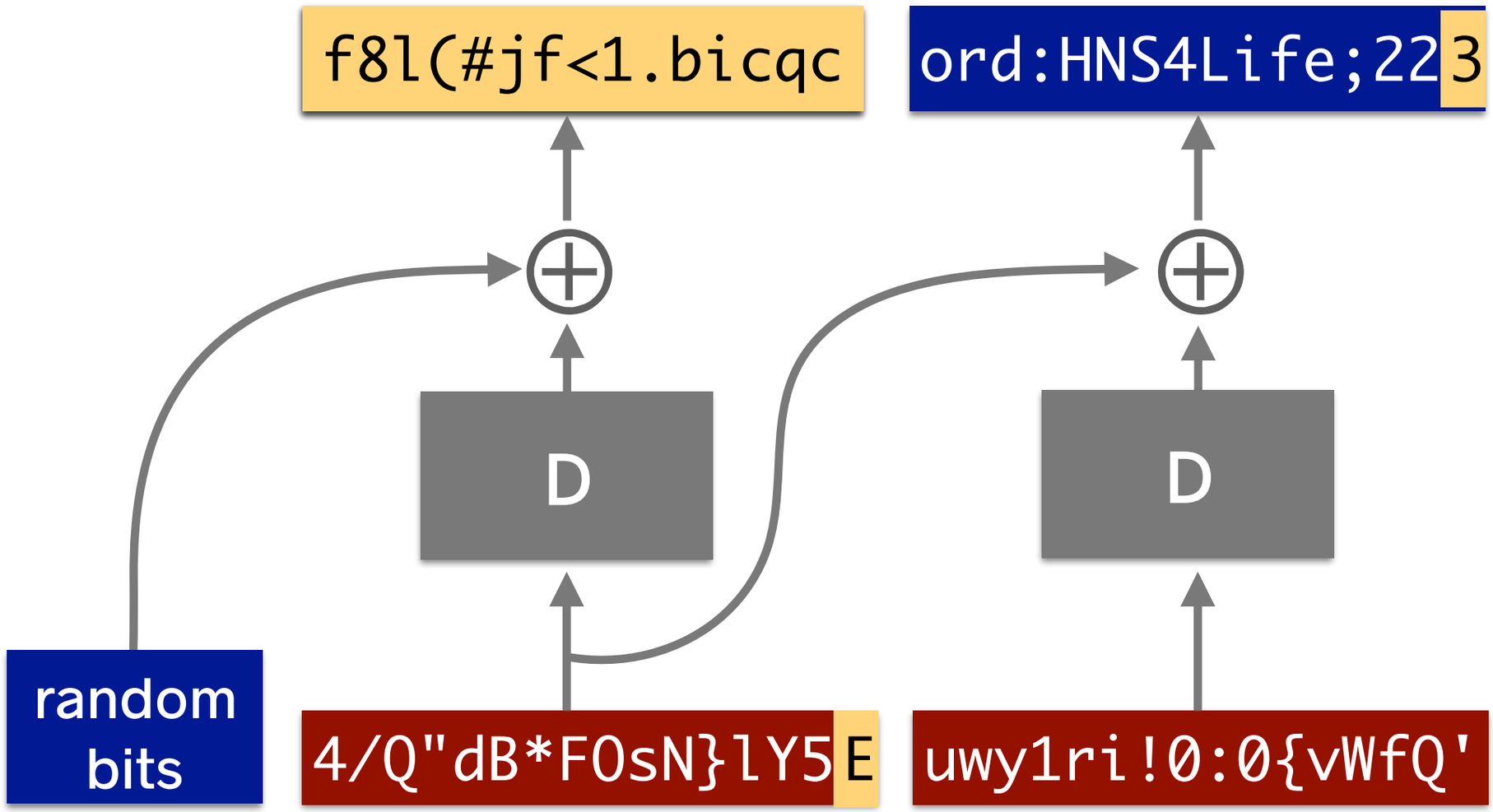


Ciphertext

Server response

Recall CBC decryption

Message



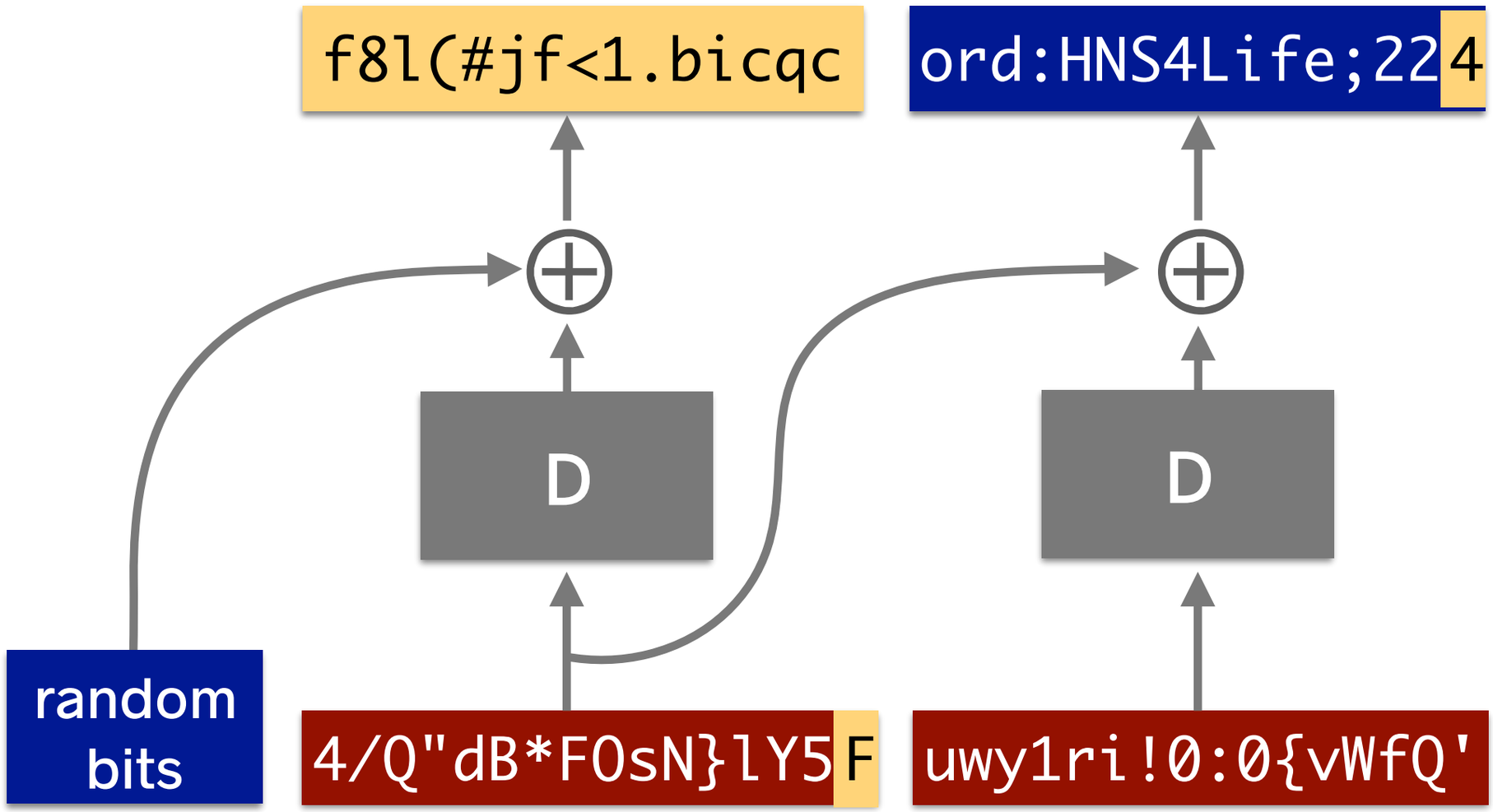
Ciphertext

Server response

```
Padding error.
```

1) Guess the last byte

Message

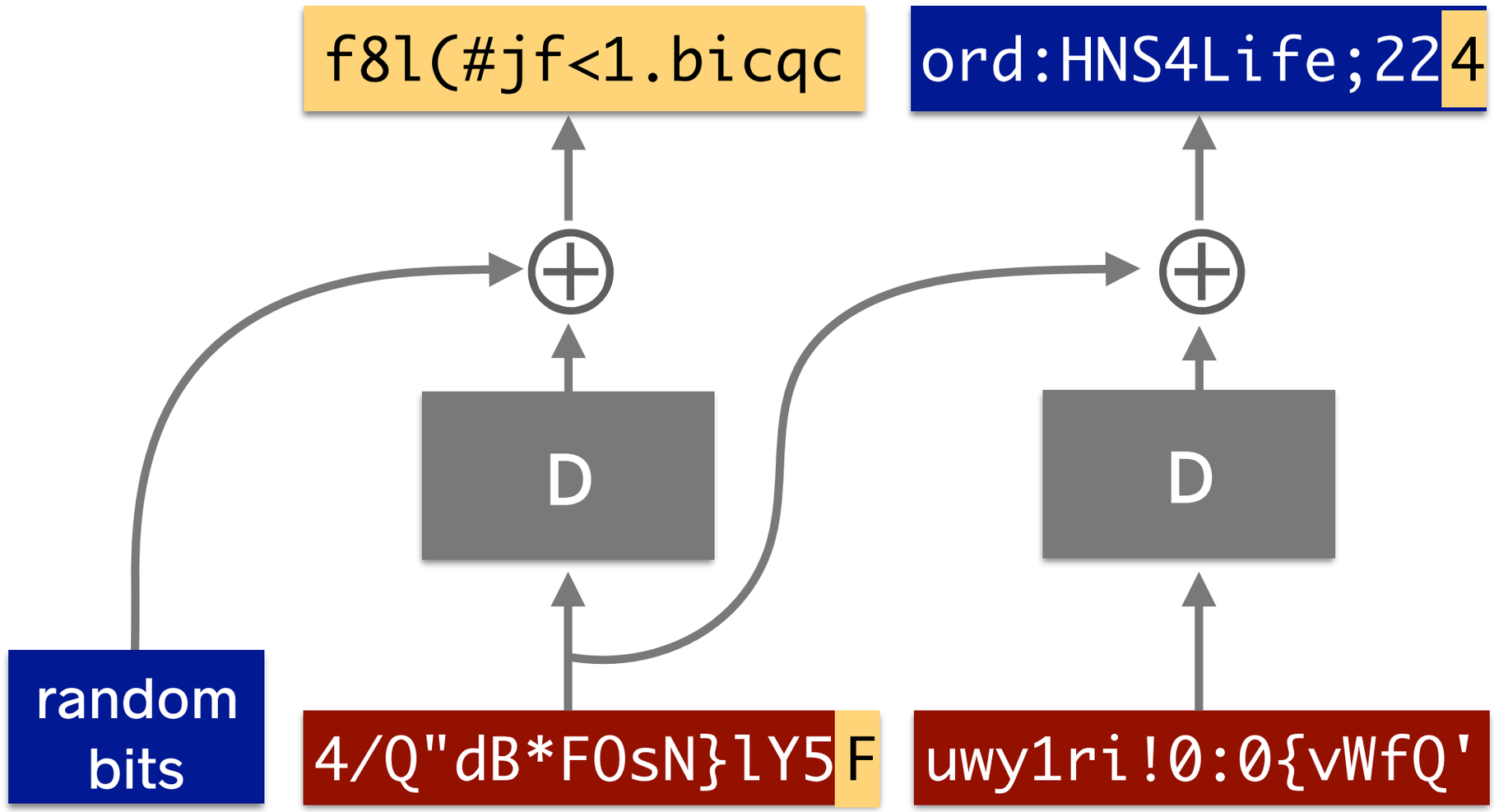


Ciphertext

Server response

1) Guess the last byte

Message



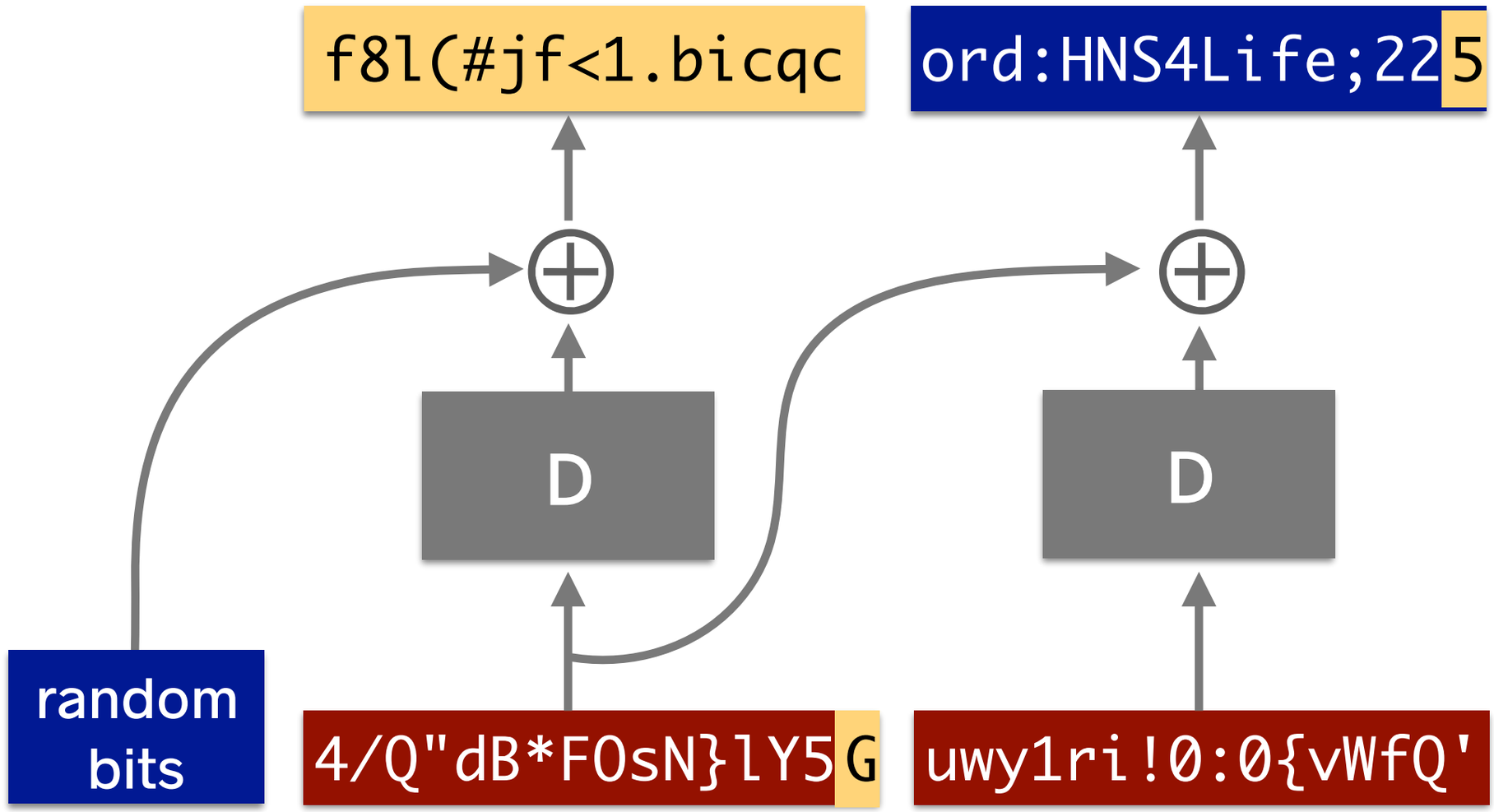
Ciphertext

Server response

```
Padding error.
```

1) Guess the last byte

Message

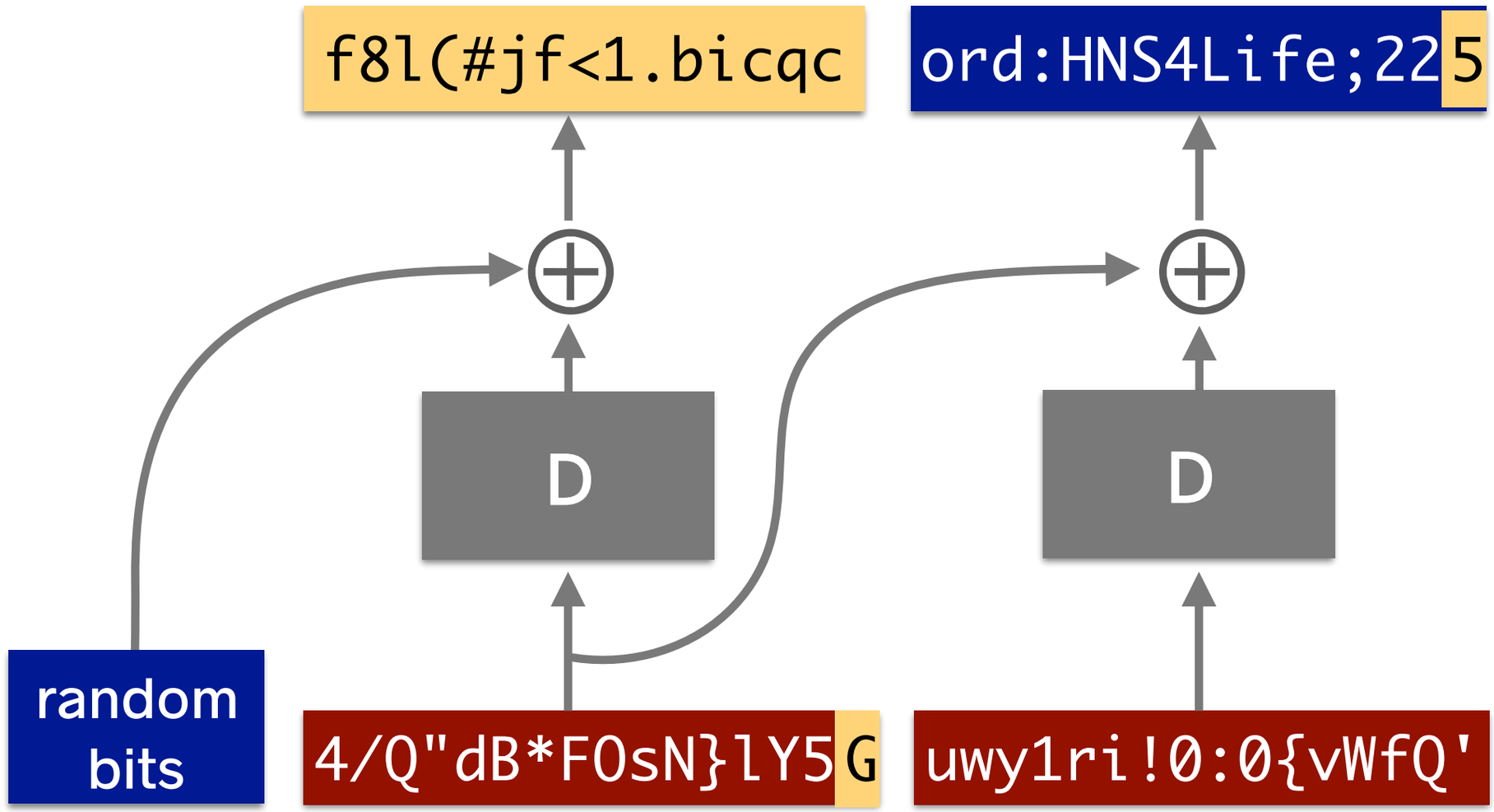


Ciphertext

Server response

1) Guess the last byte

Message



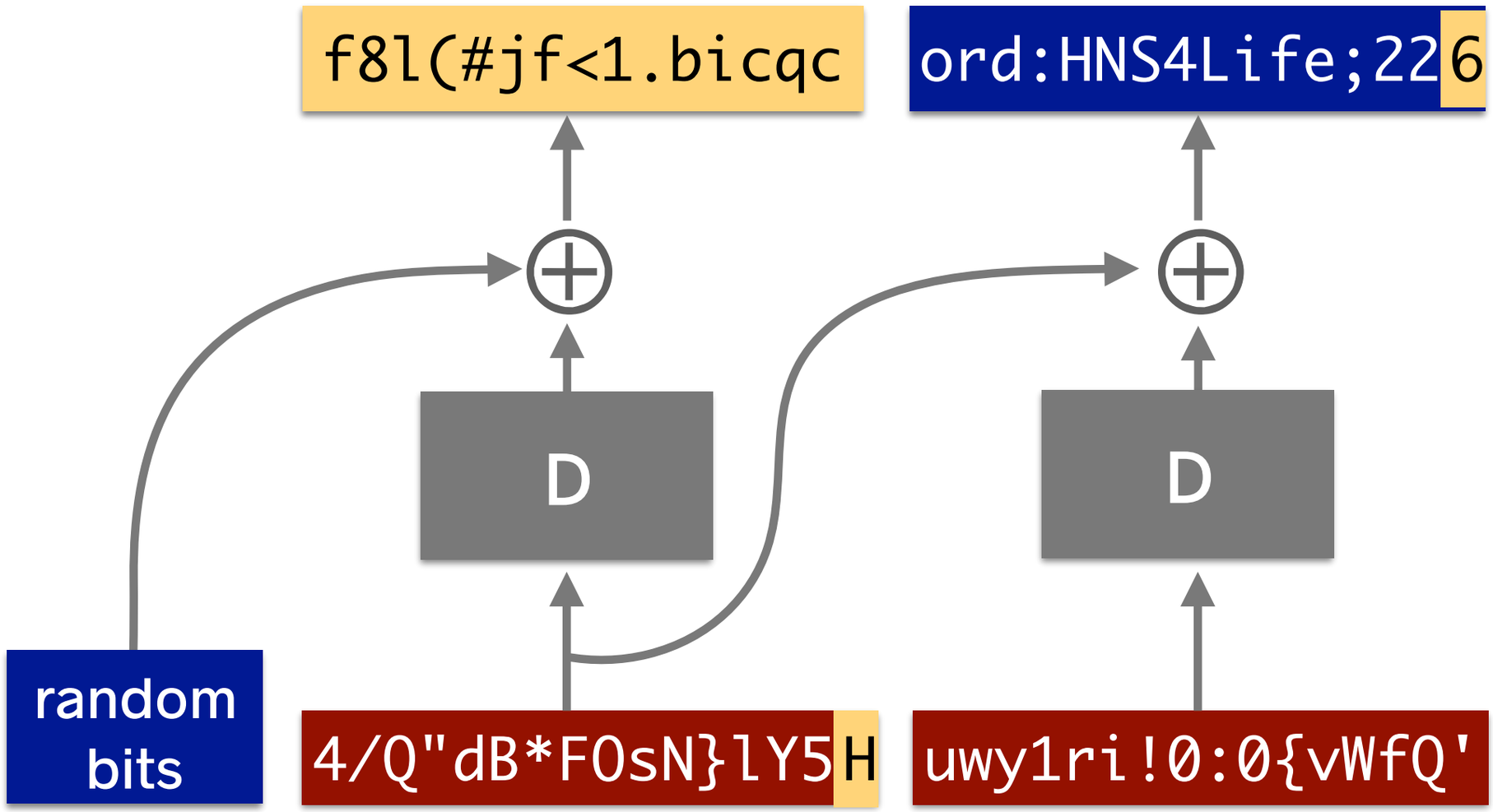
Ciphertext

Server response

```
Padding error.
```

1) Guess the last byte

Message

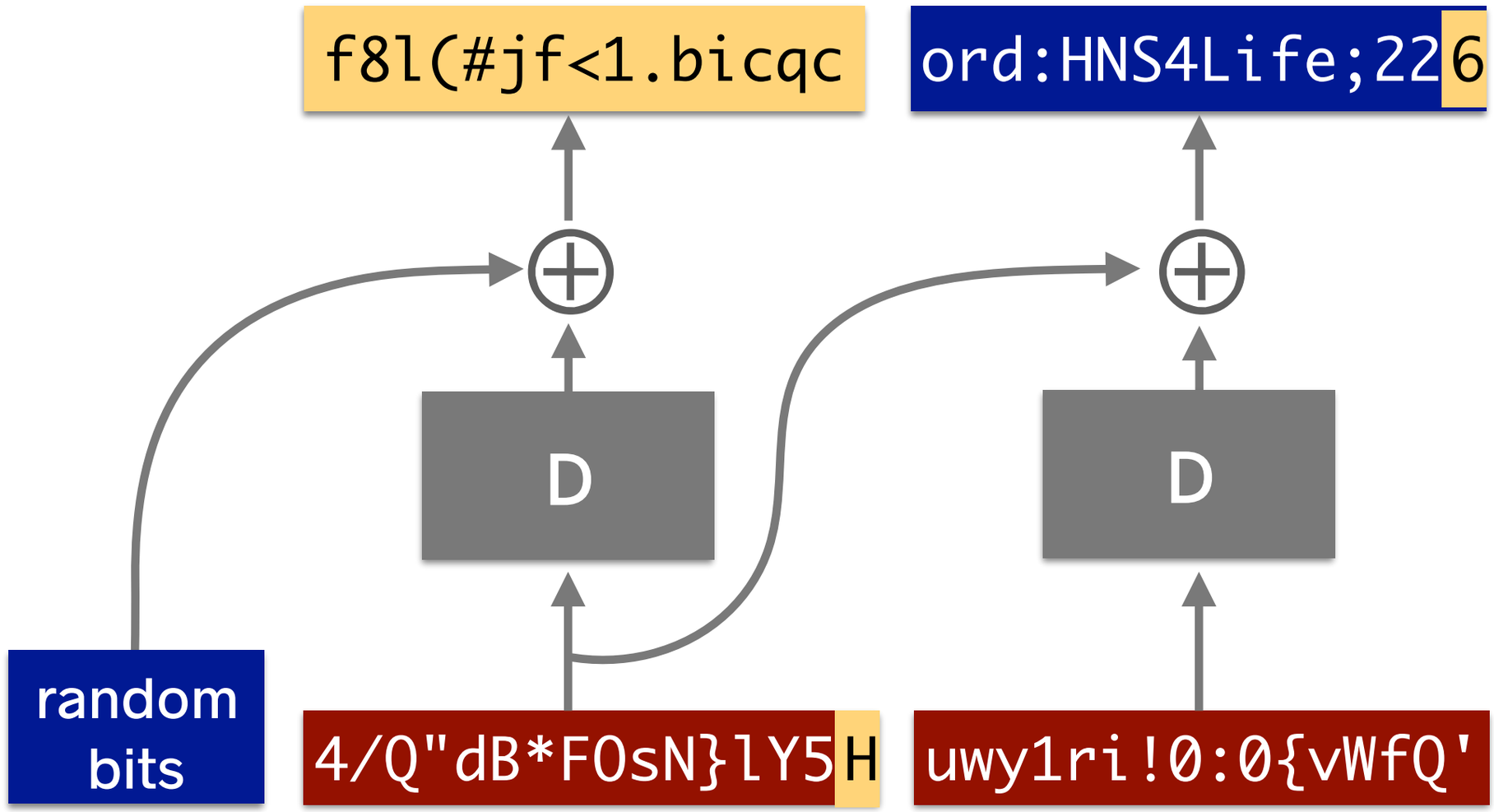


Ciphertext

Server response

1) Guess the last byte

Message



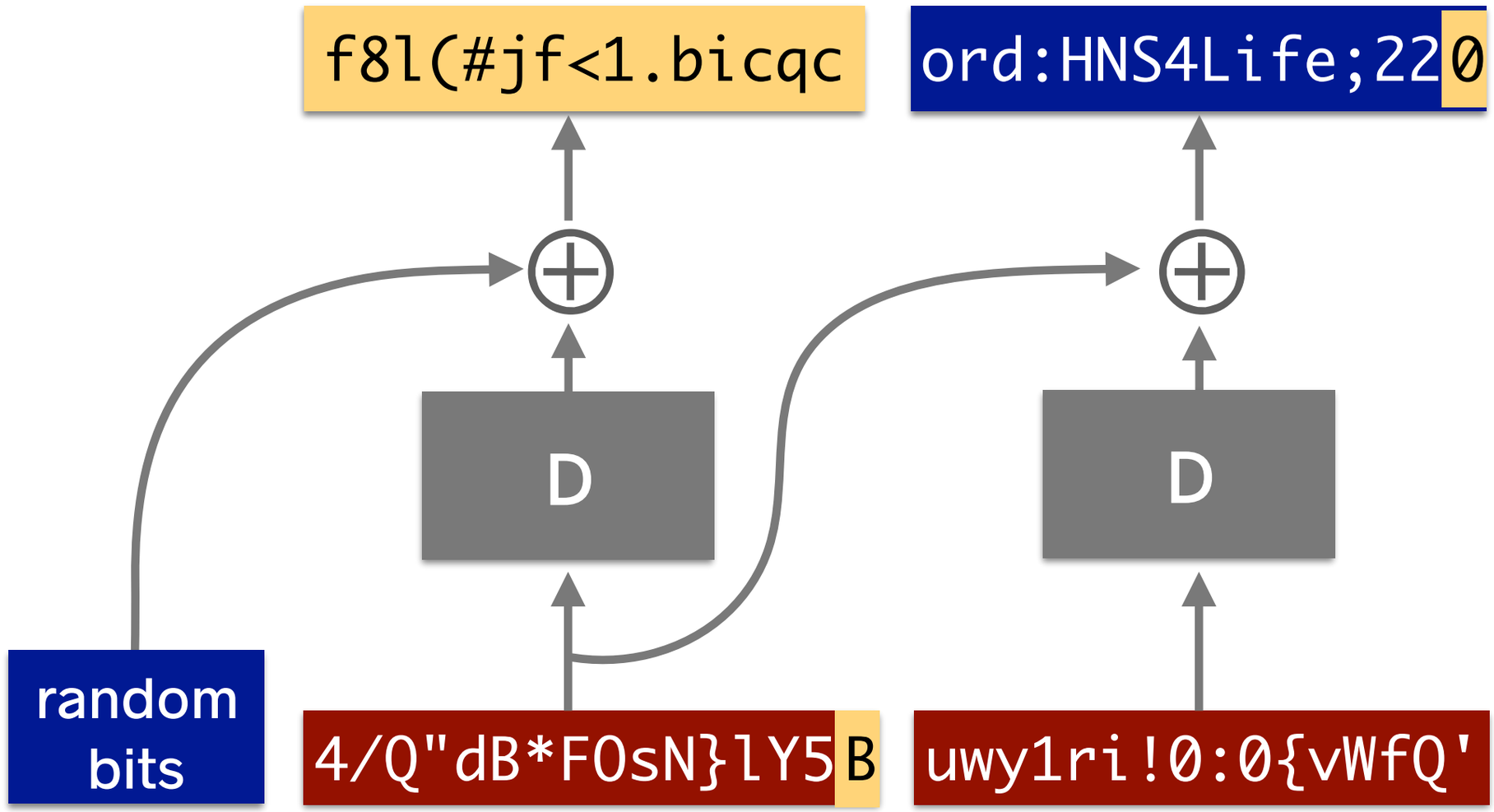
Ciphertext

Server response

```
Padding error.
```

1) Guess the last byte

Message

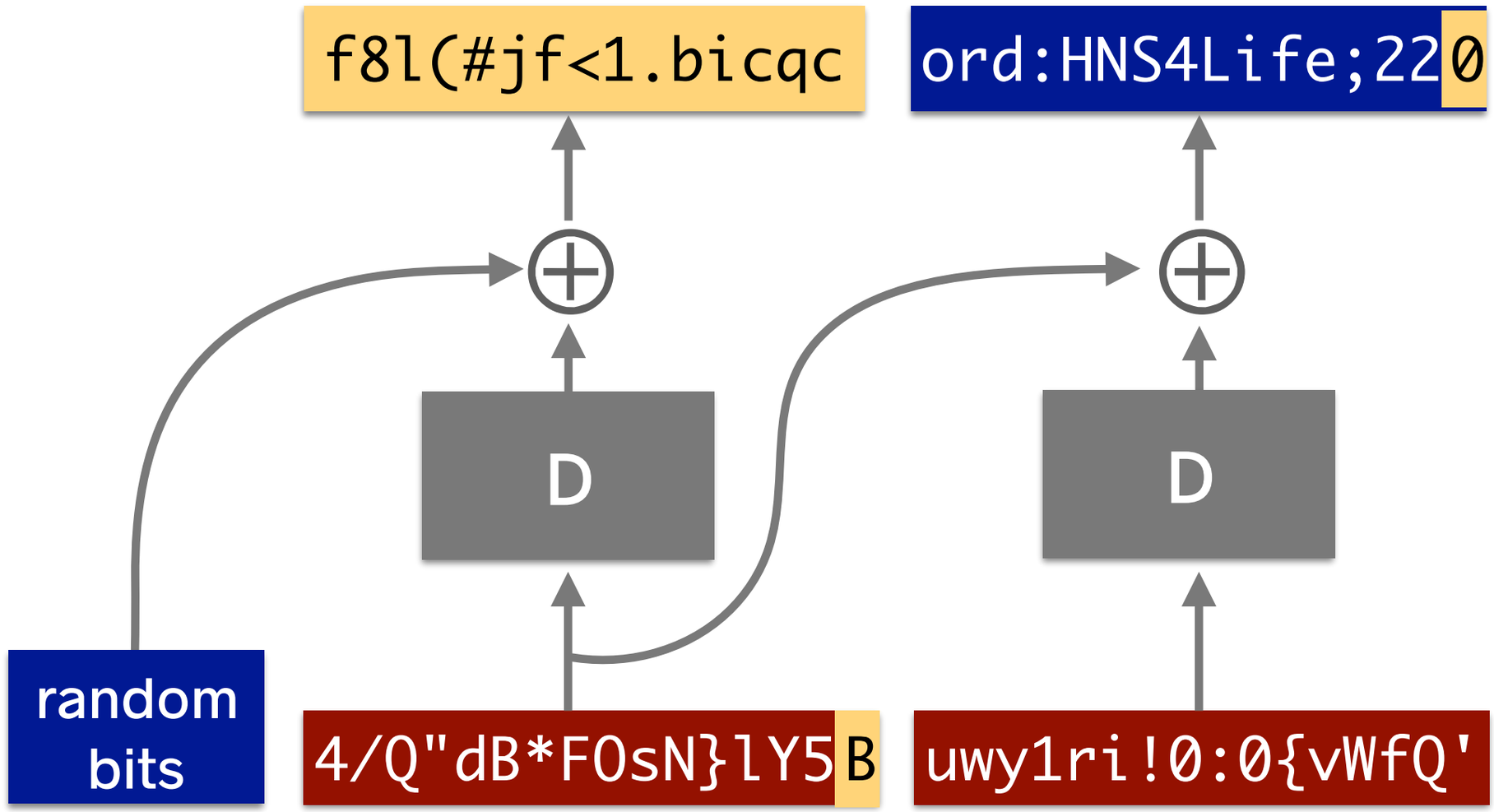


Ciphertext

Server response

1) Guess the last byte

Message

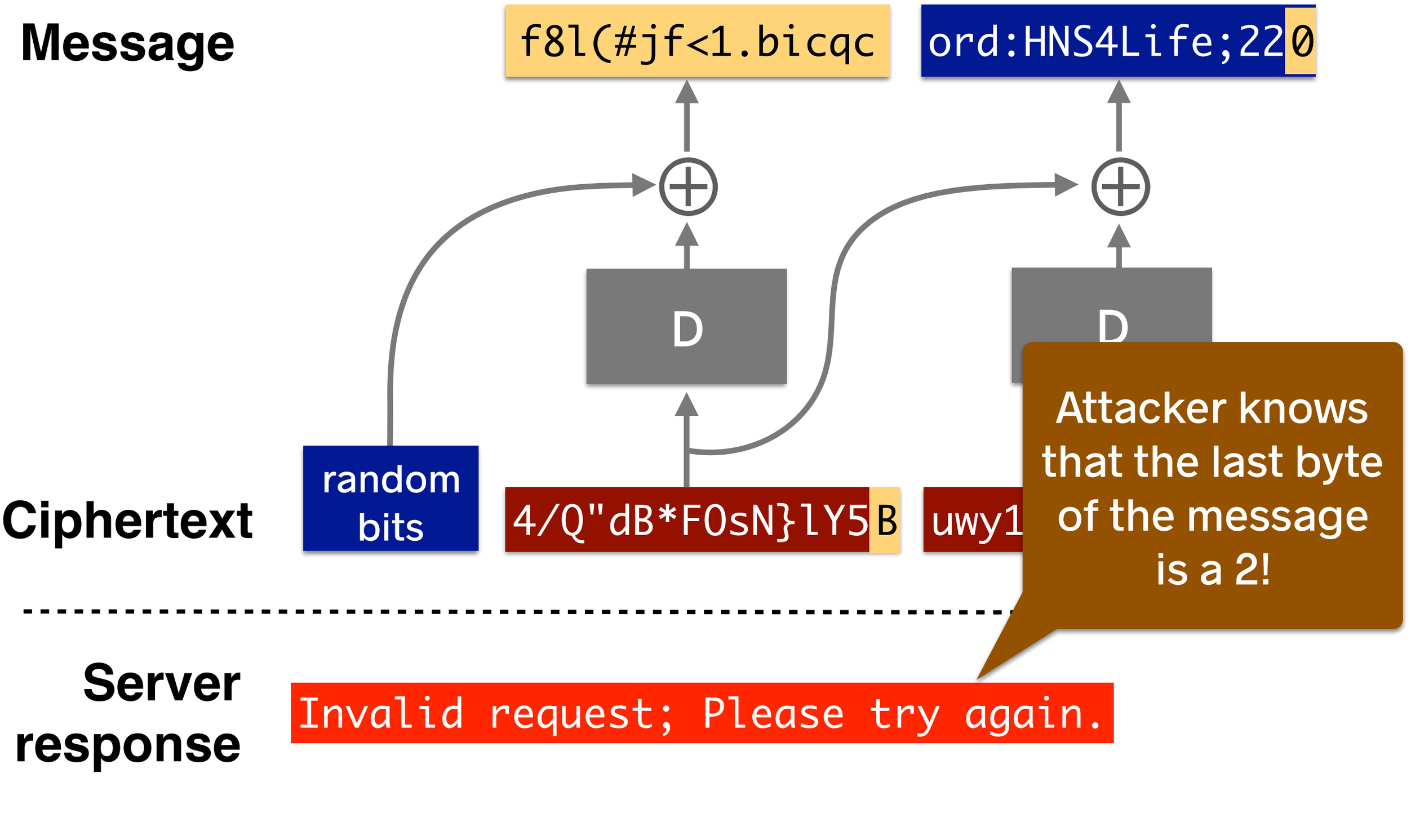


Ciphertext

Server response

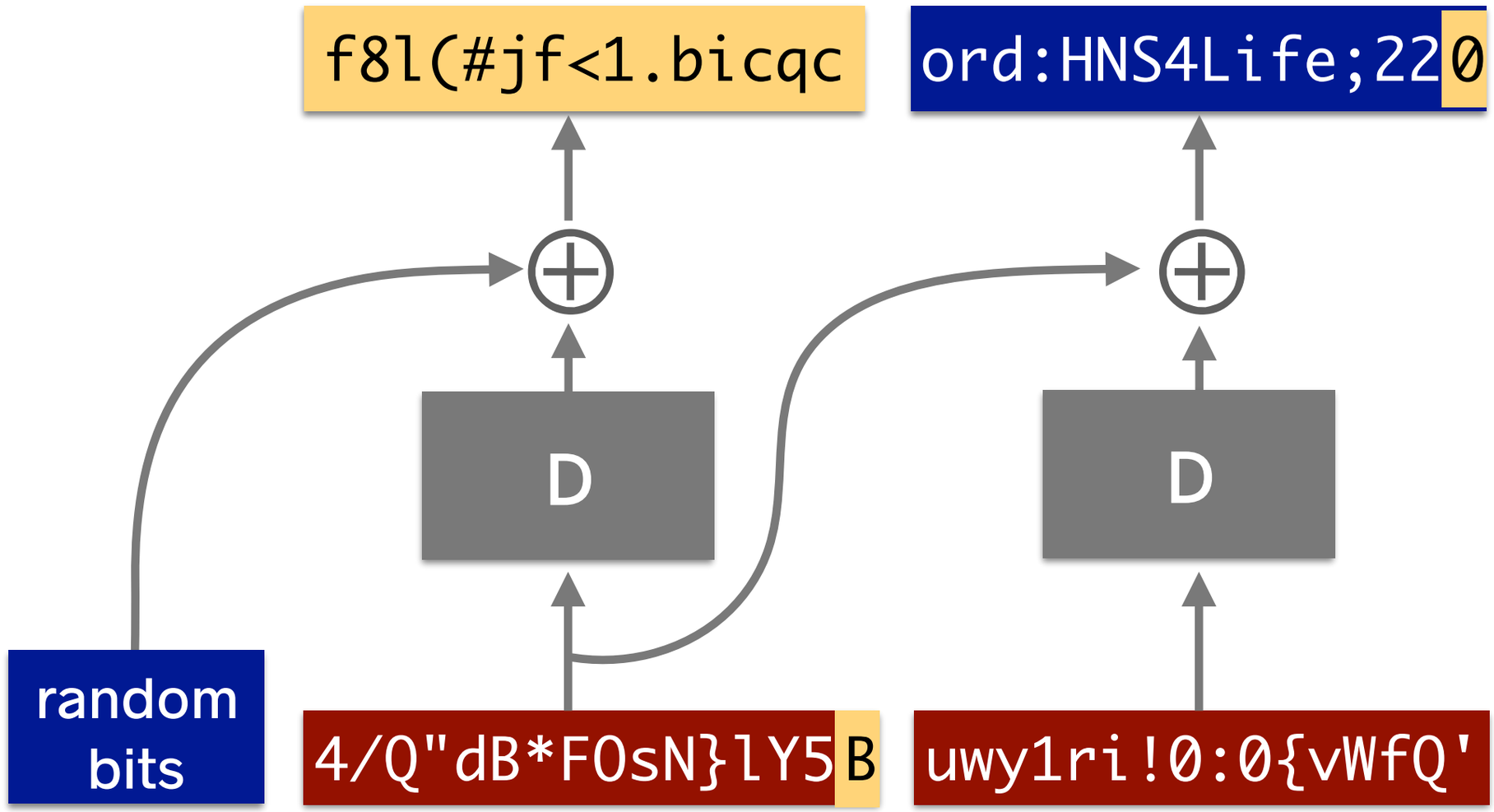
```
Invalid request; Please try again.
```

1) Guess the last byte



1) Guess the last byte

Message



Ciphertext

Server response

```
Invalid request; Please try again.
```

2) Learn the last byte

The attacker learns that the last byte of the plaintext, when decremented by two, equals ‘0’

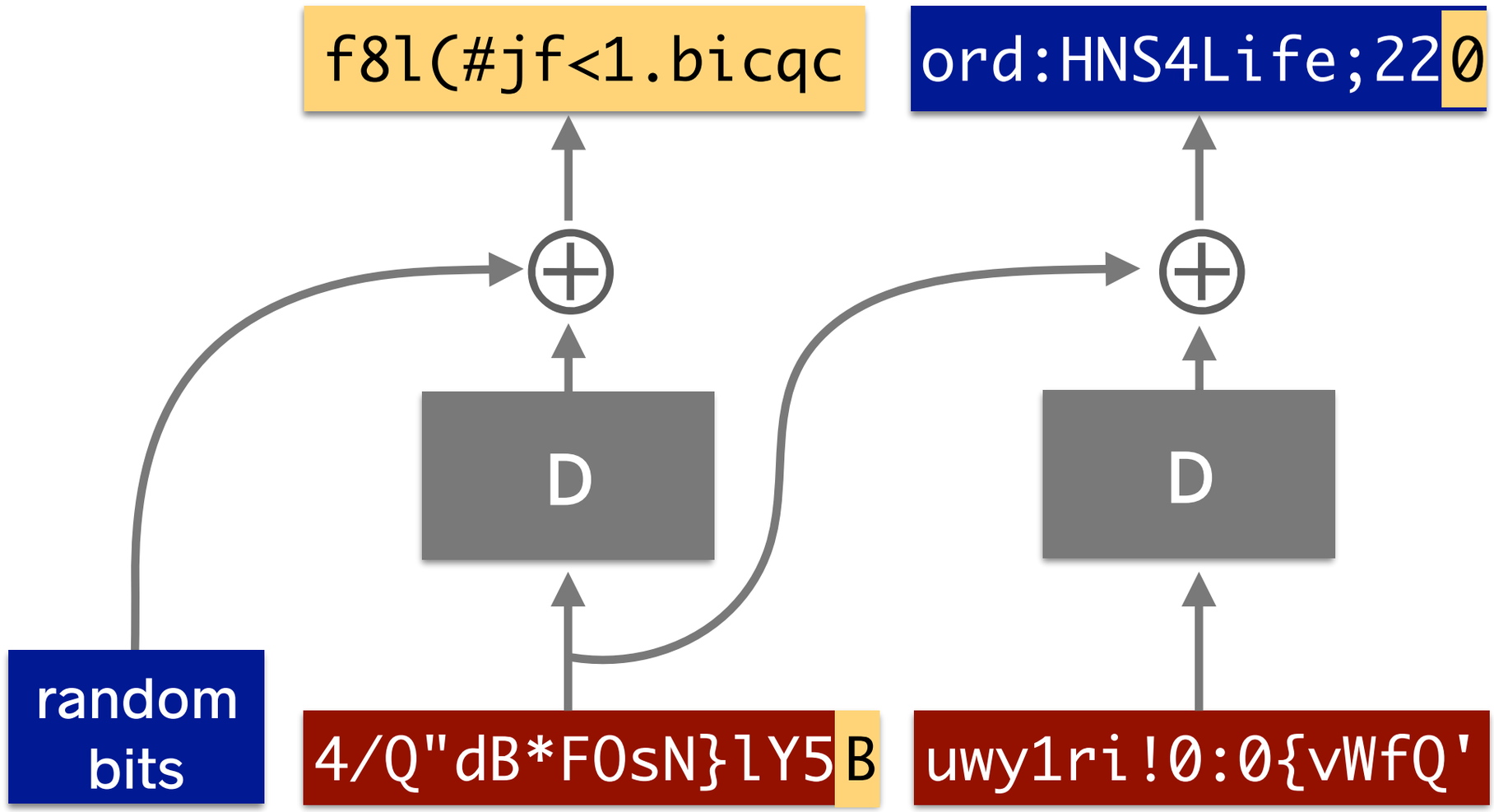
$$x - 2 = 0$$

implies

$$x = 2$$

2) Learn the last byte

Message

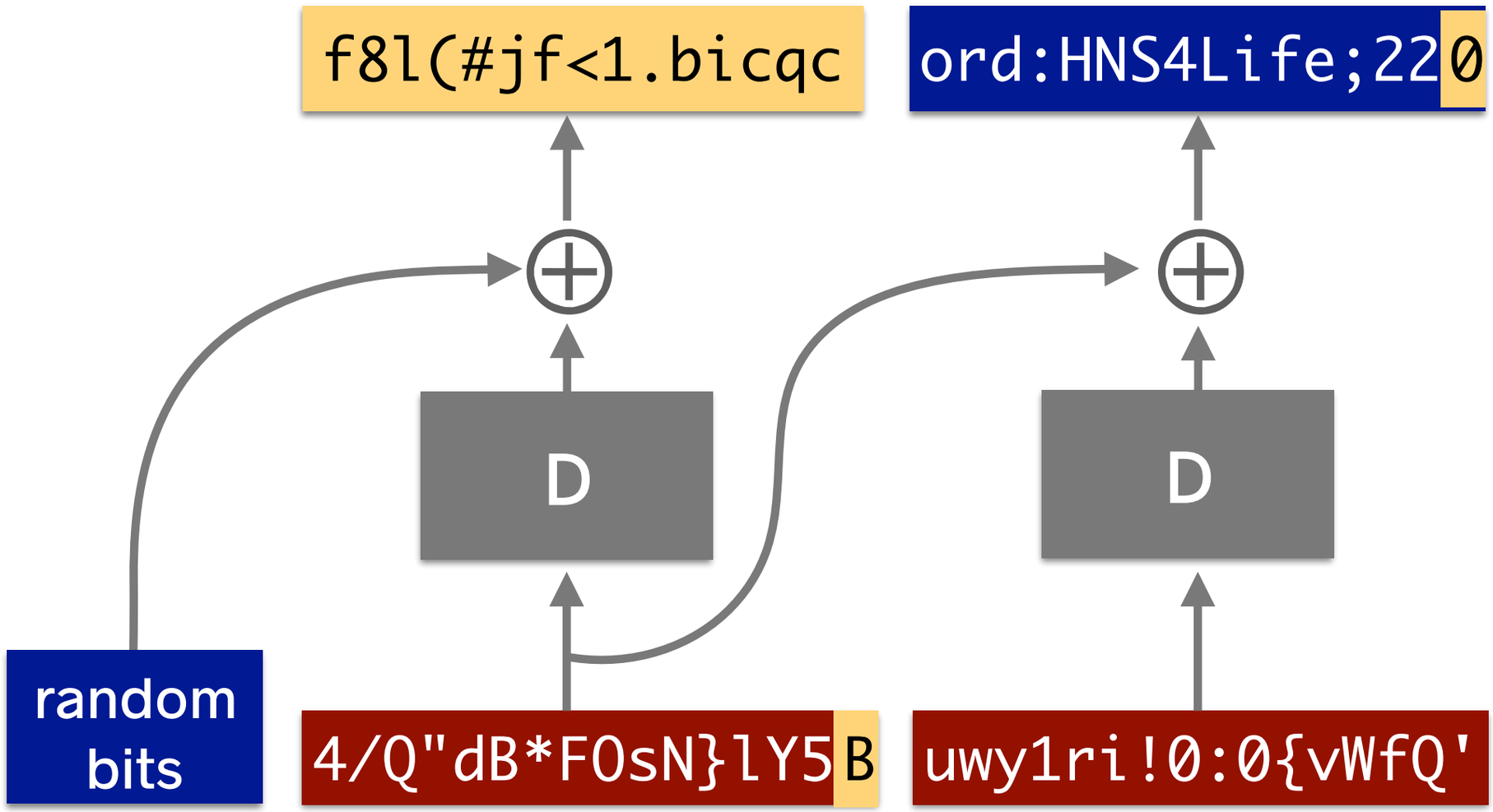


Ciphertext

Server response

2) Learn the last byte

Message

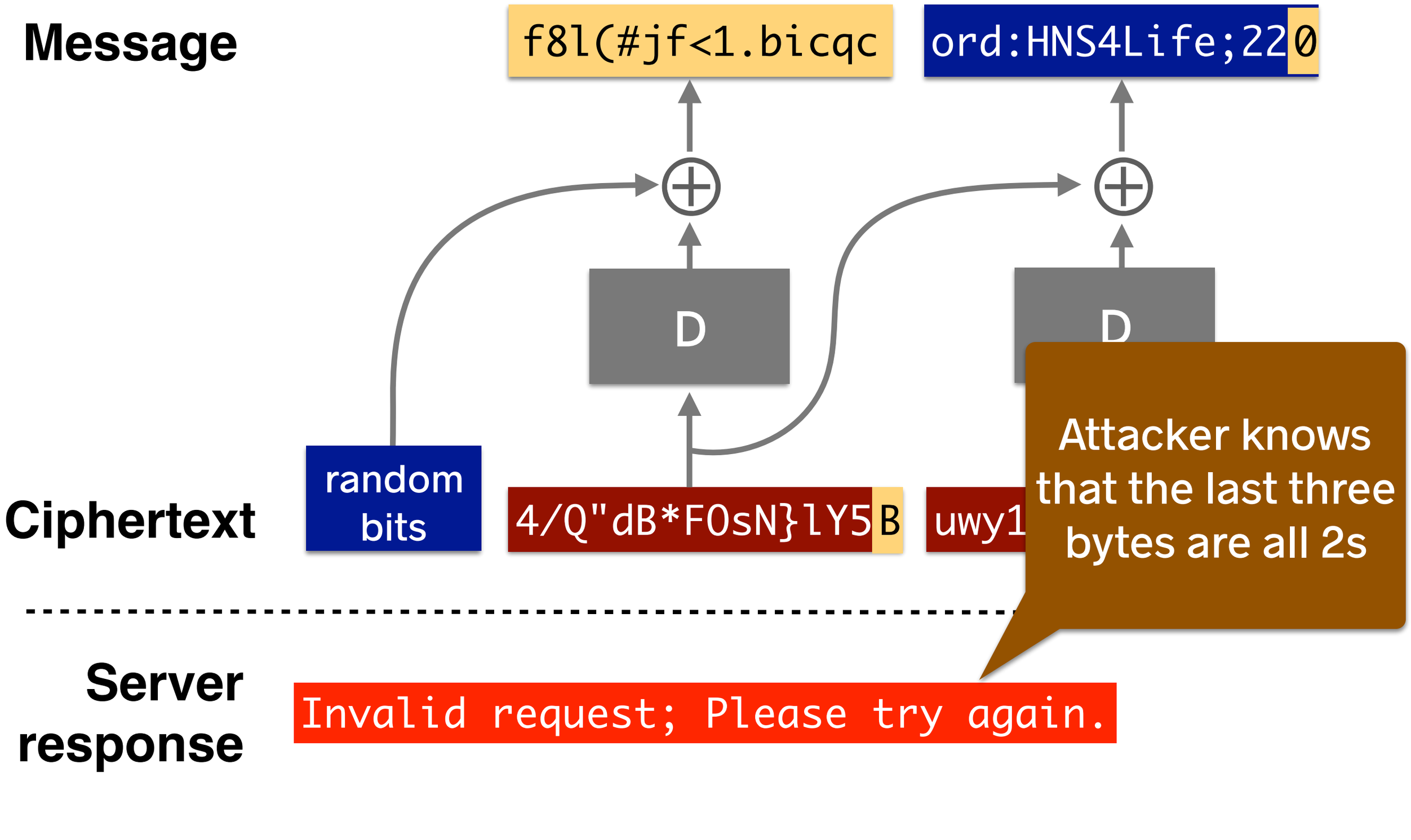


Ciphertext

Server response

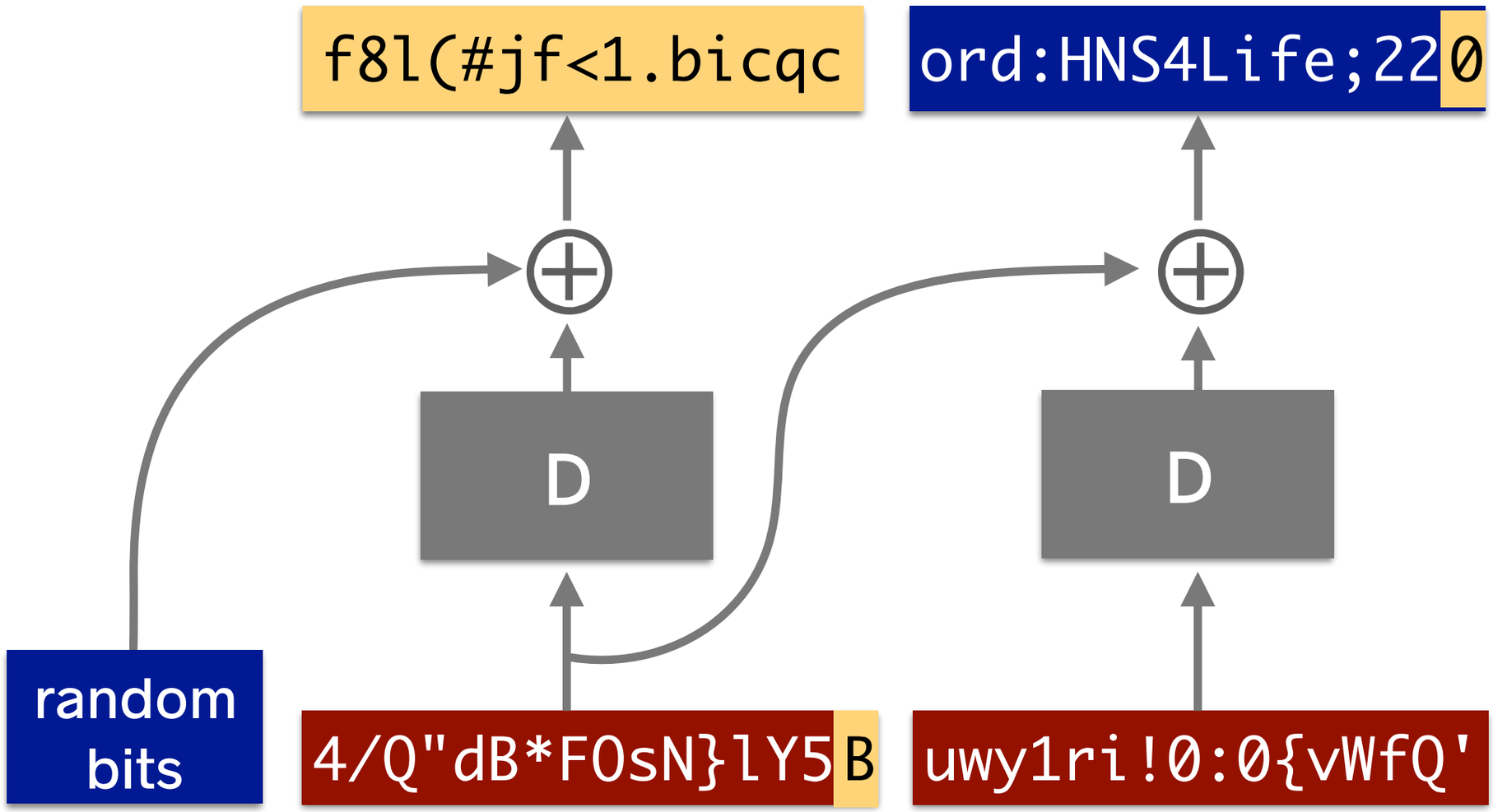
```
Invalid request; Please try again.
```

2) Learn the last byte



2) Learn the last byte

Message

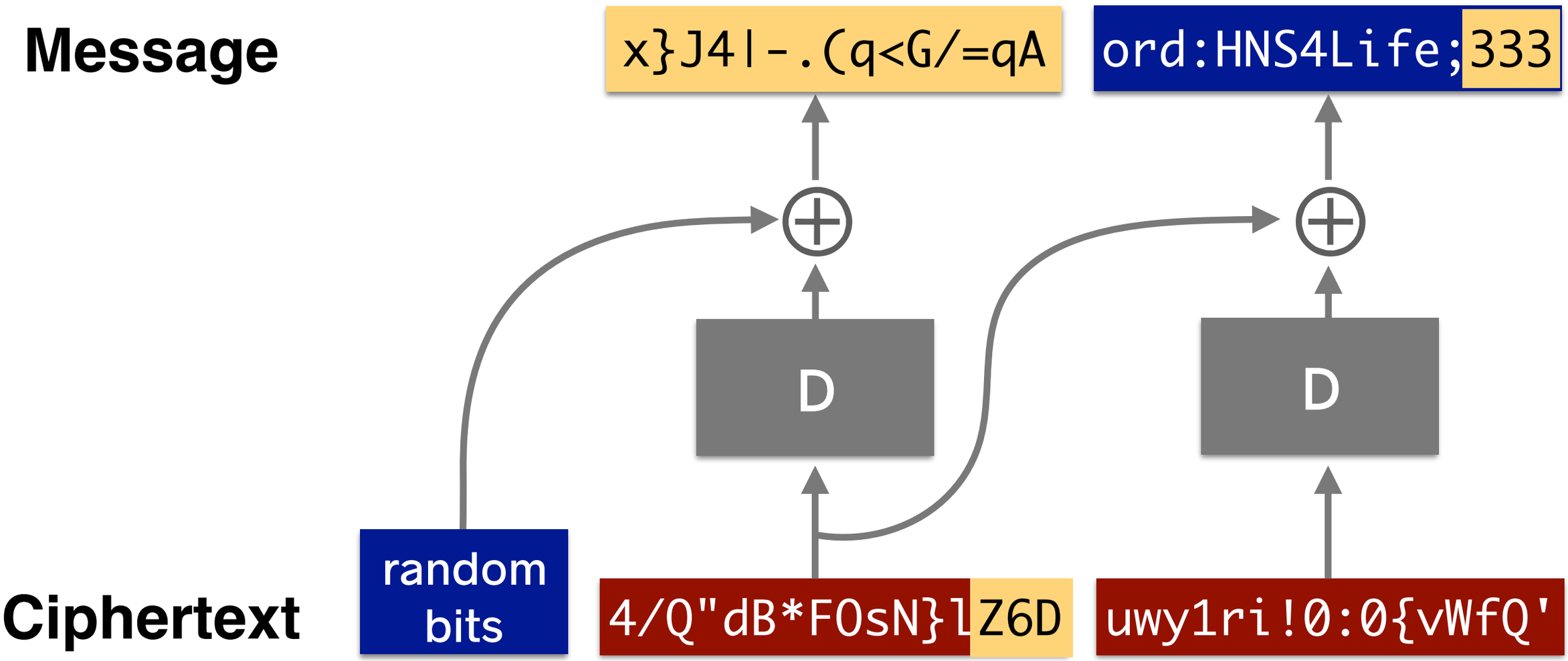


Ciphertext

Server response

```
Invalid request; Please try again.
```

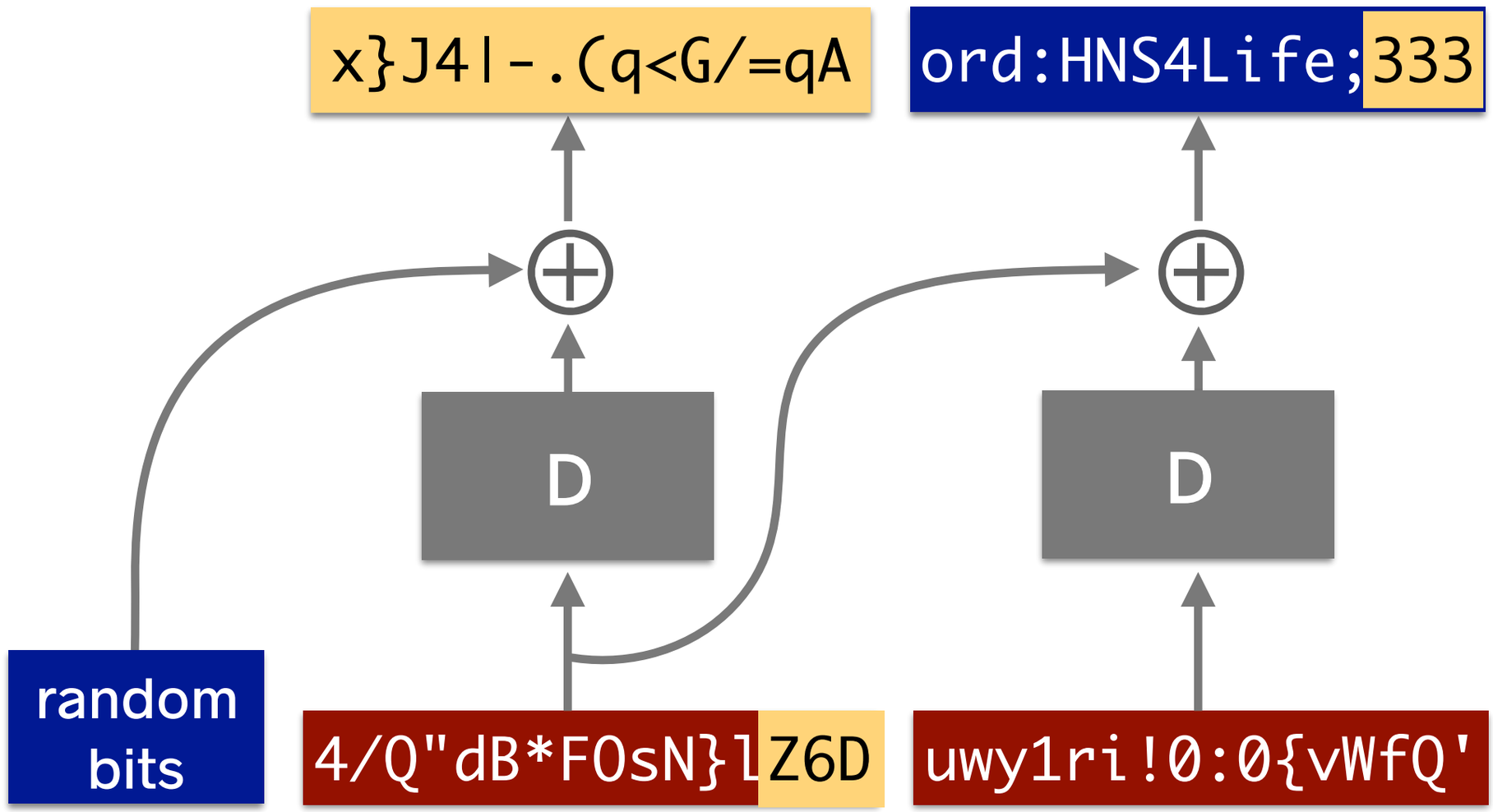
3) Guess the last non-padding byte



Server response

3) Guess the last non-padding byte

Message



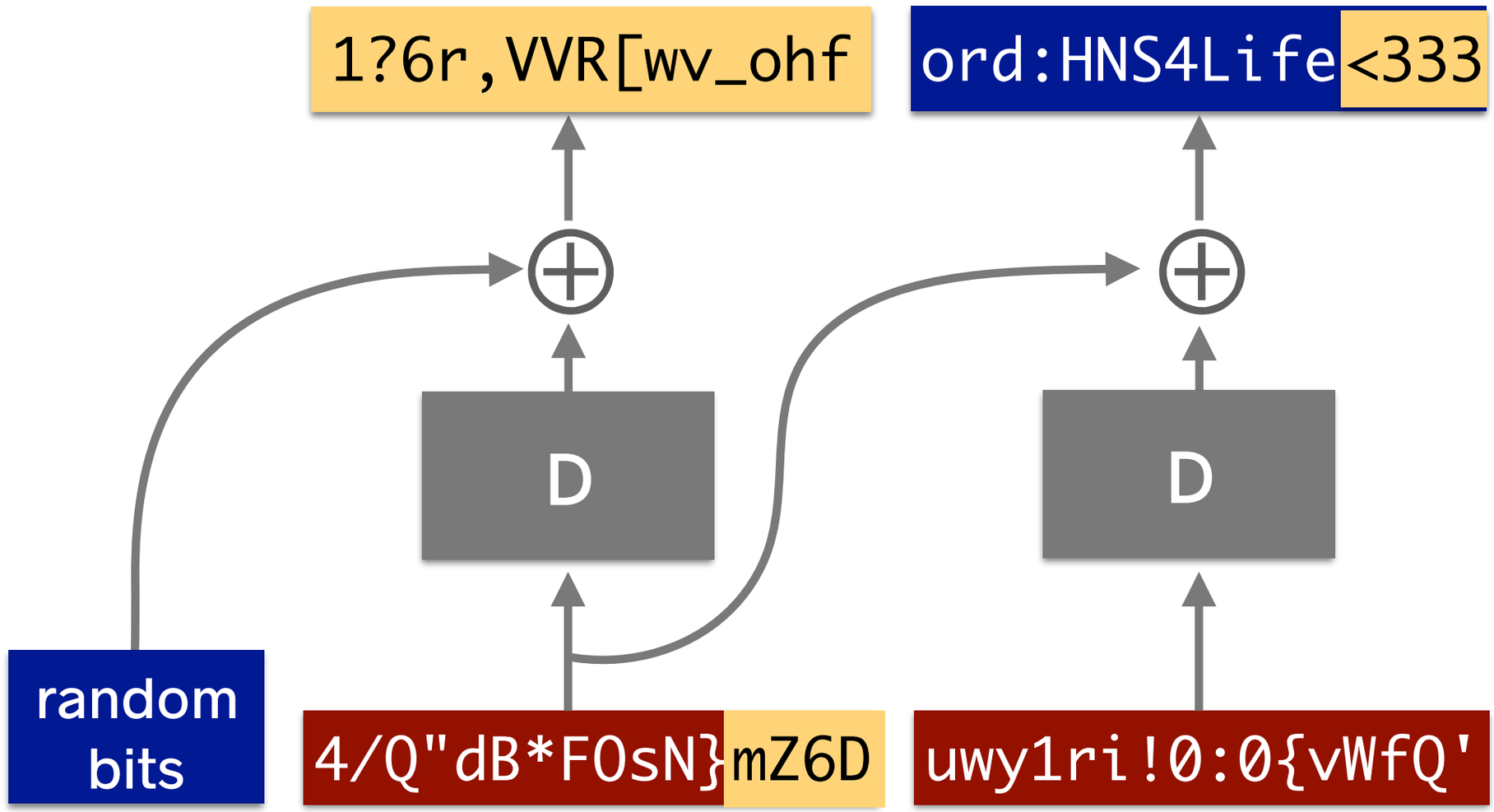
Ciphertext

Server response

`Padding error.`

3) Guess the last non-padding byte

Message

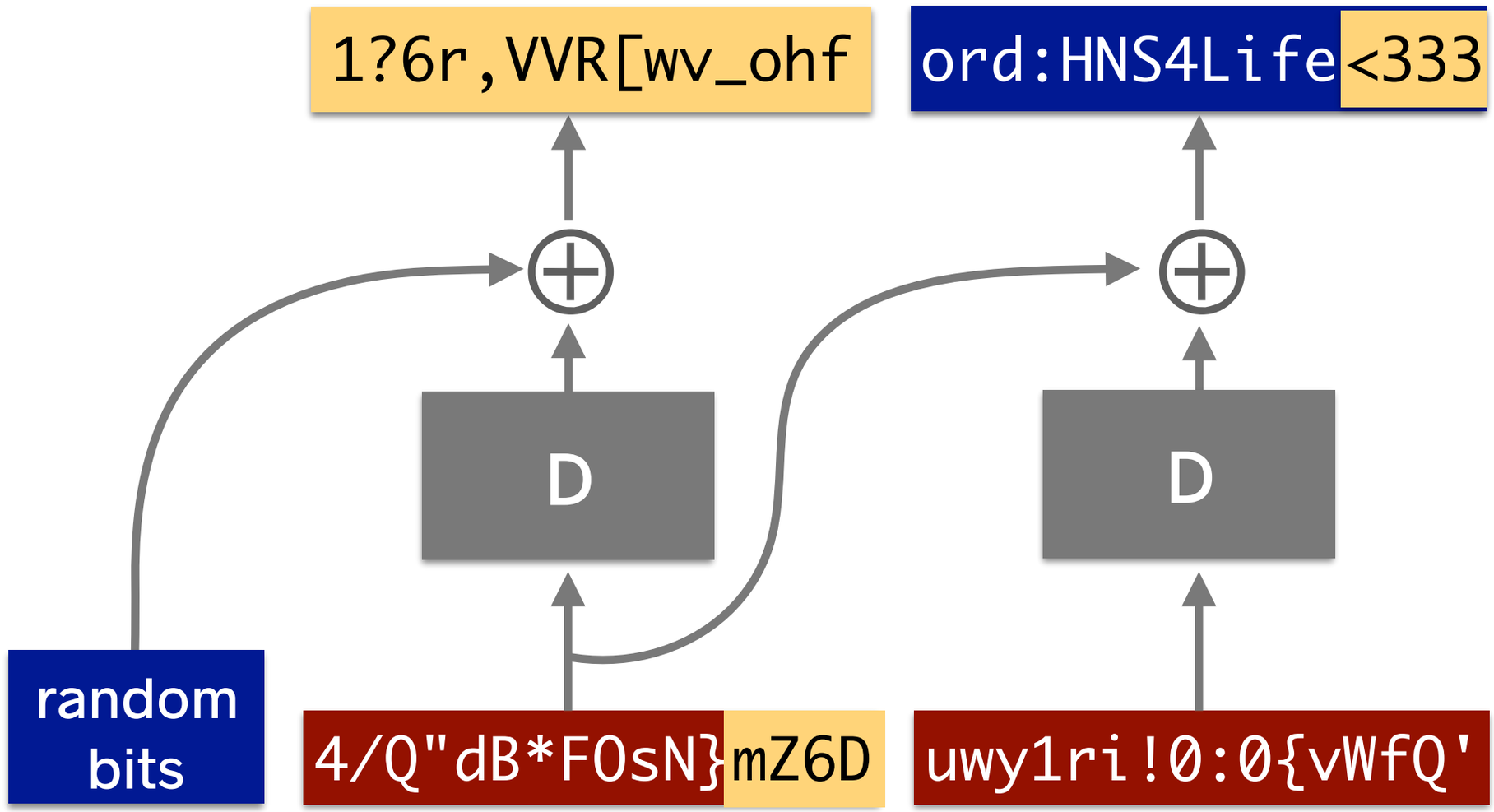


Ciphertext

Server response

3) Guess the last non-padding byte

Message

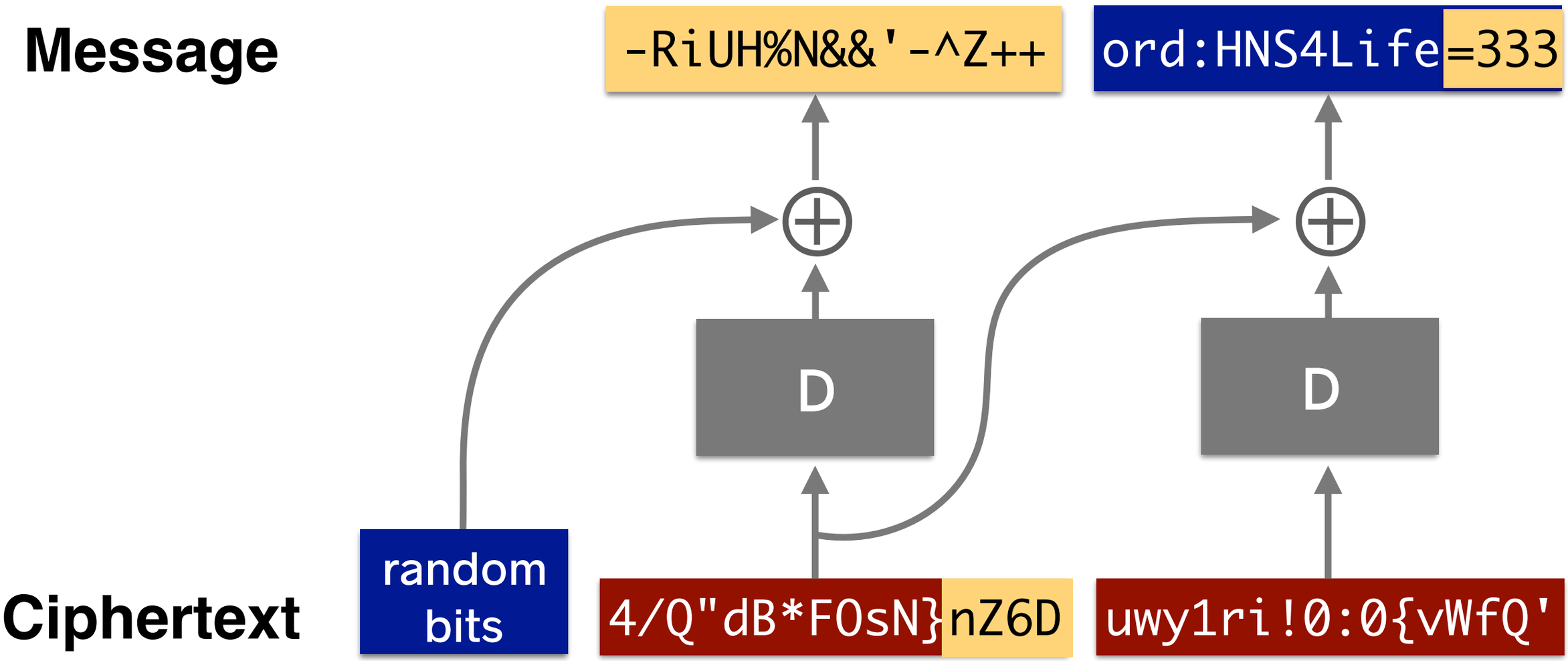


Ciphertext

Server response

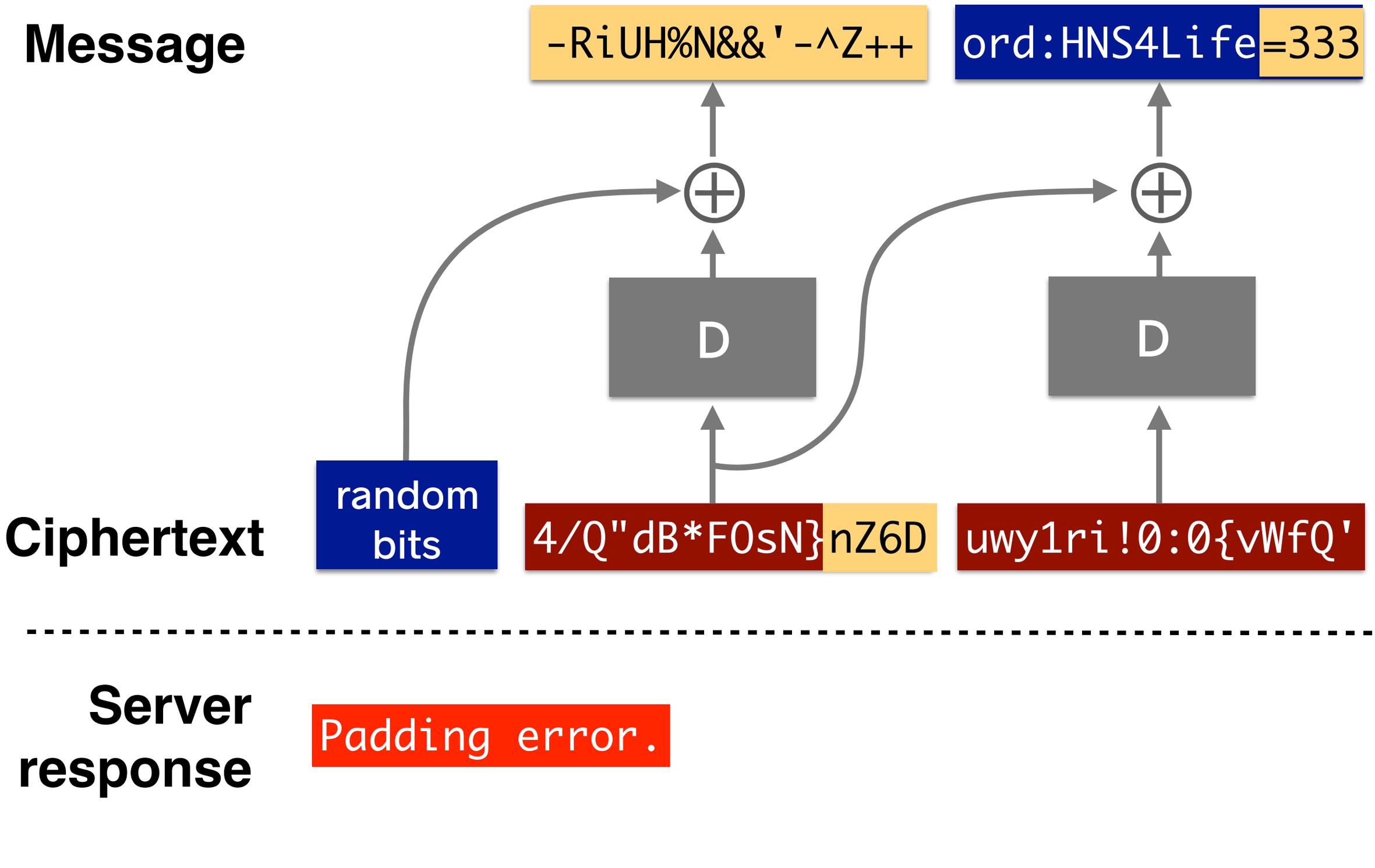
```
Padding error.
```

3) Guess the last non-padding byte



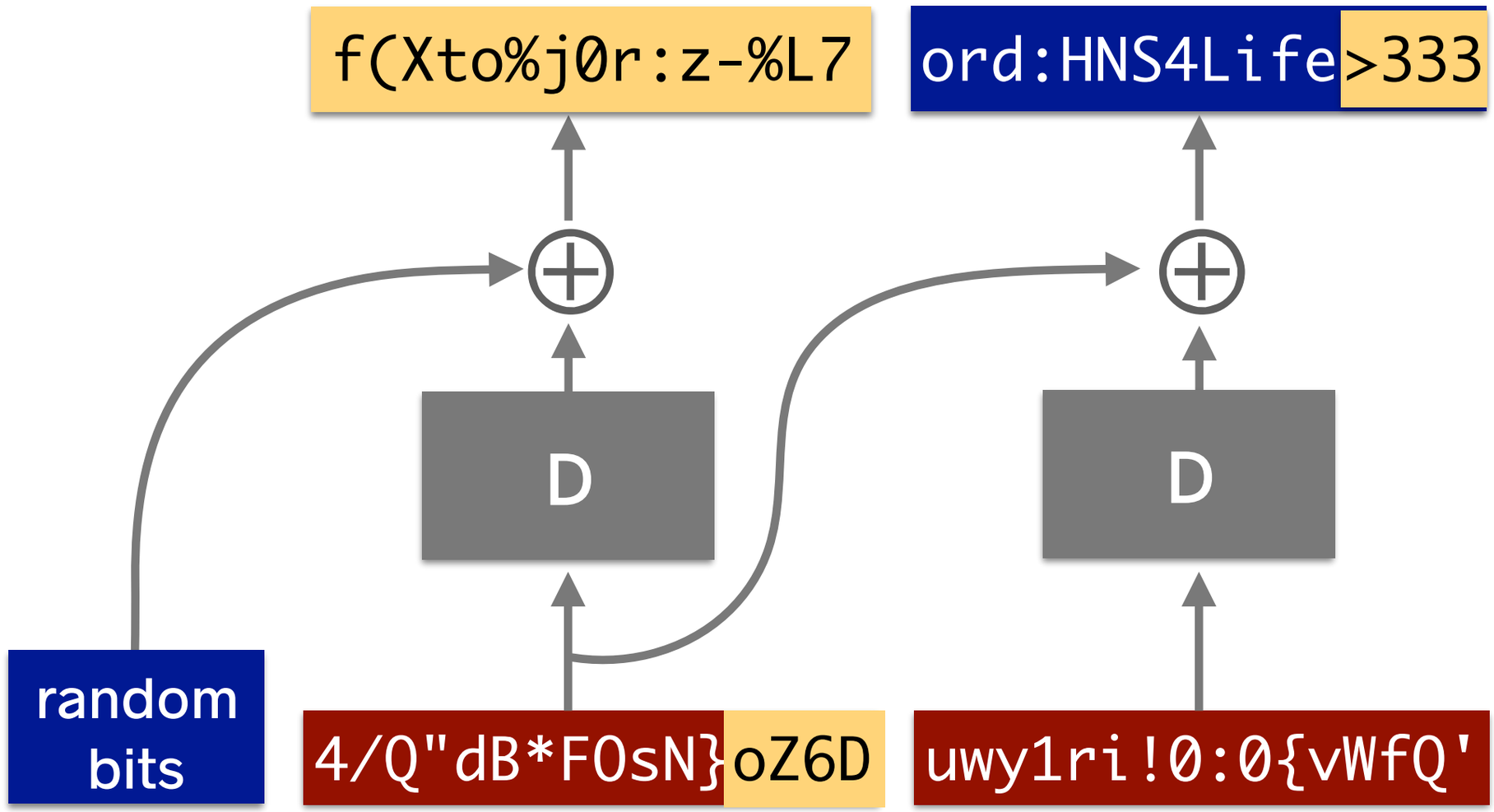
Server response

3) Guess the last non-padding byte



3) Guess the last non-padding byte

Message

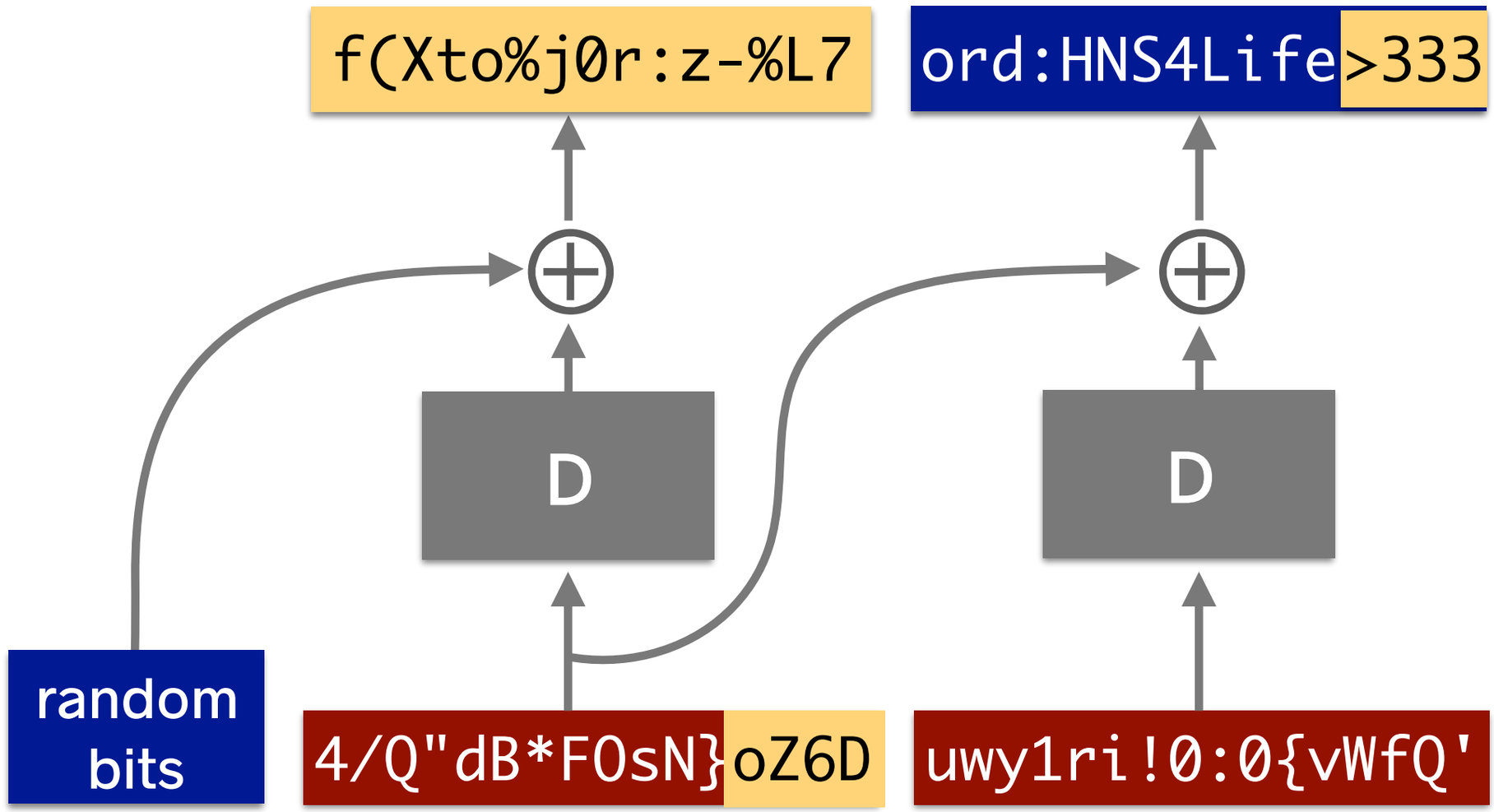


Ciphertext

Server response

3) Guess the last non-padding byte

Message

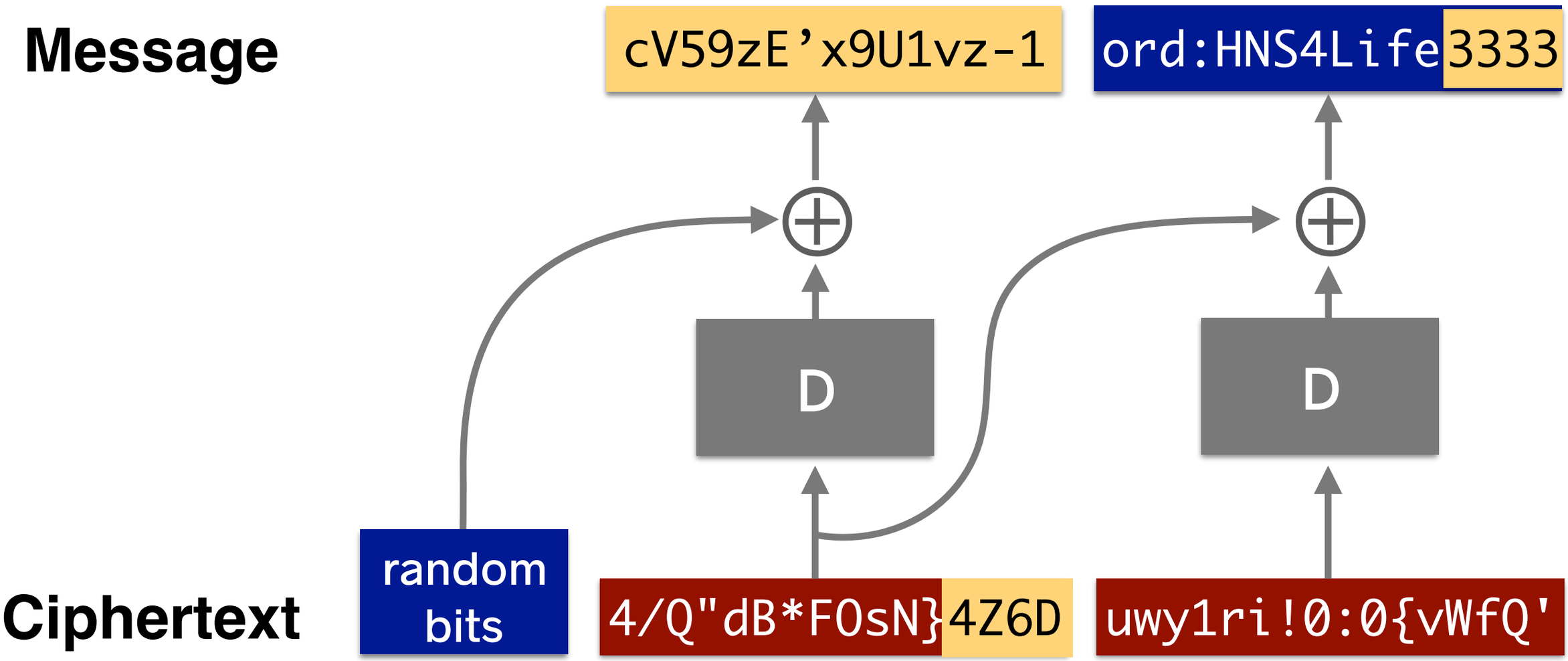


Ciphertext

Server response

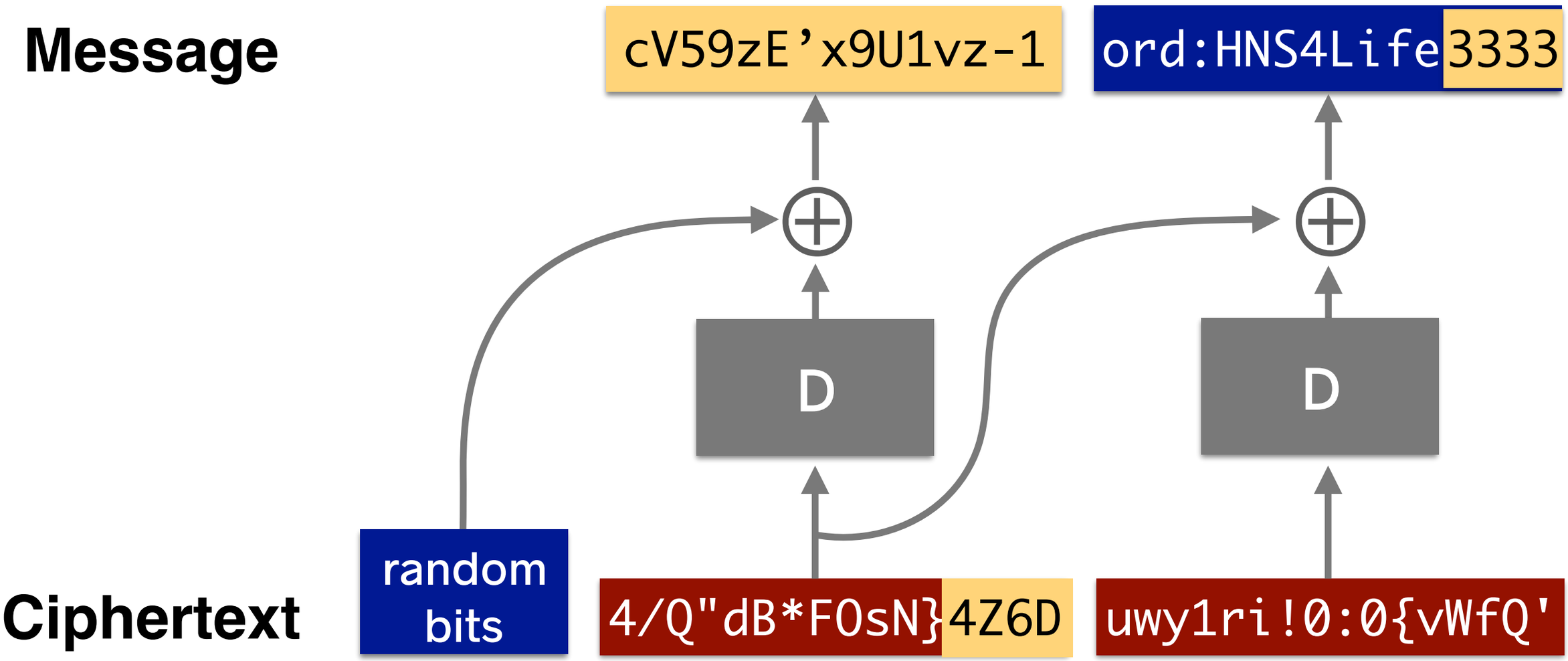
```
Padding error.
```

4) Learn the last non-padding byte



Server response

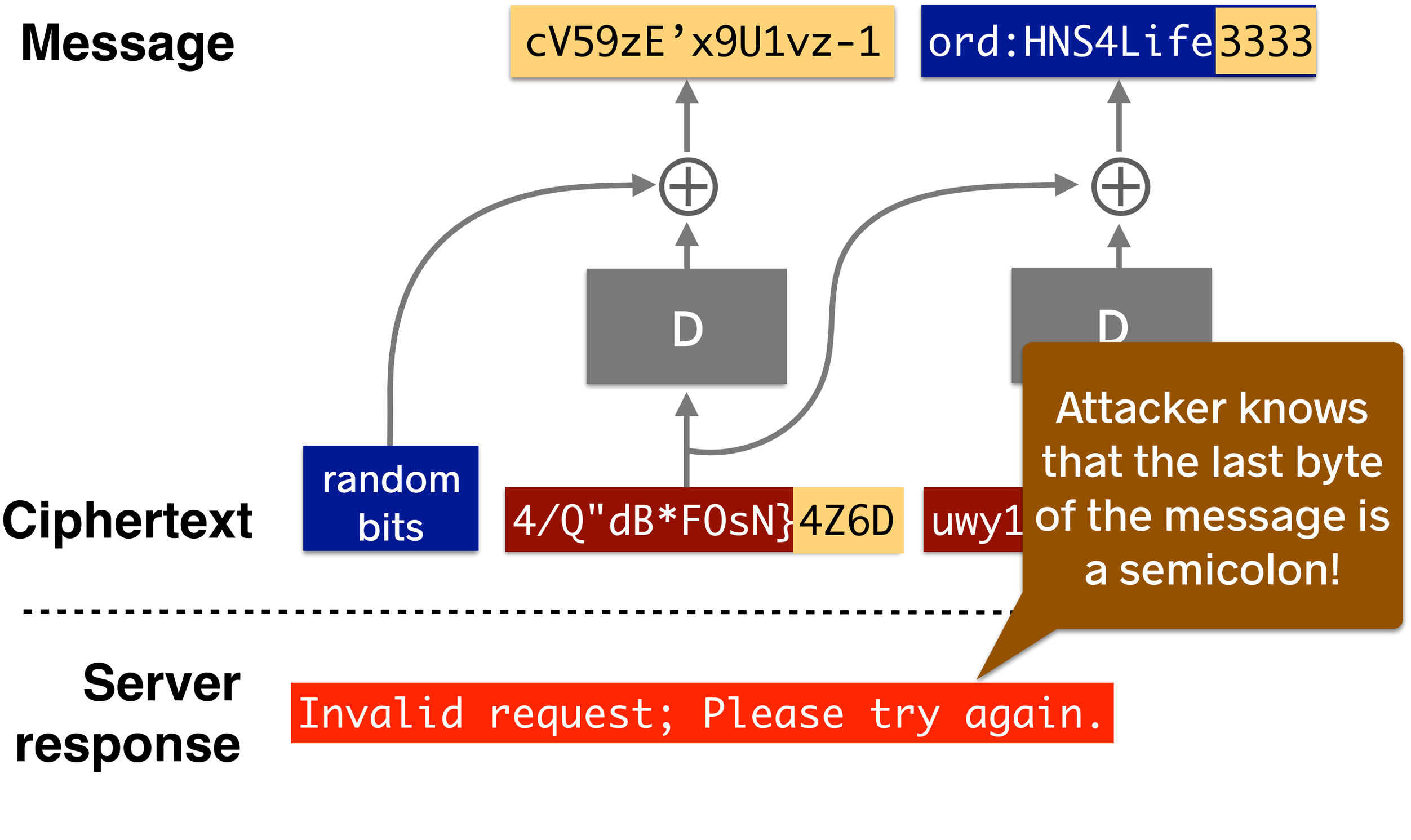
4) Learn the last non-padding byte



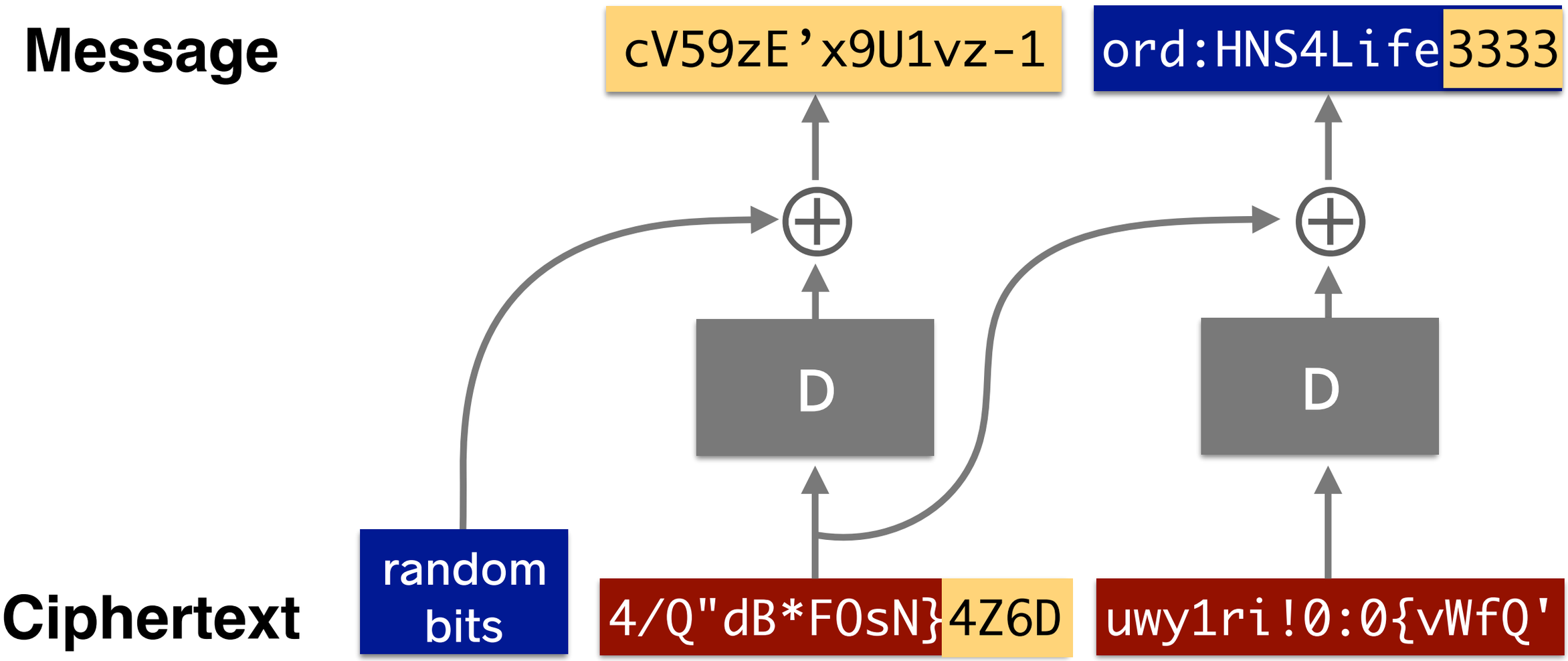
Server response

```
Invalid request; Please try again.
```

4) Learn the last non-padding byte



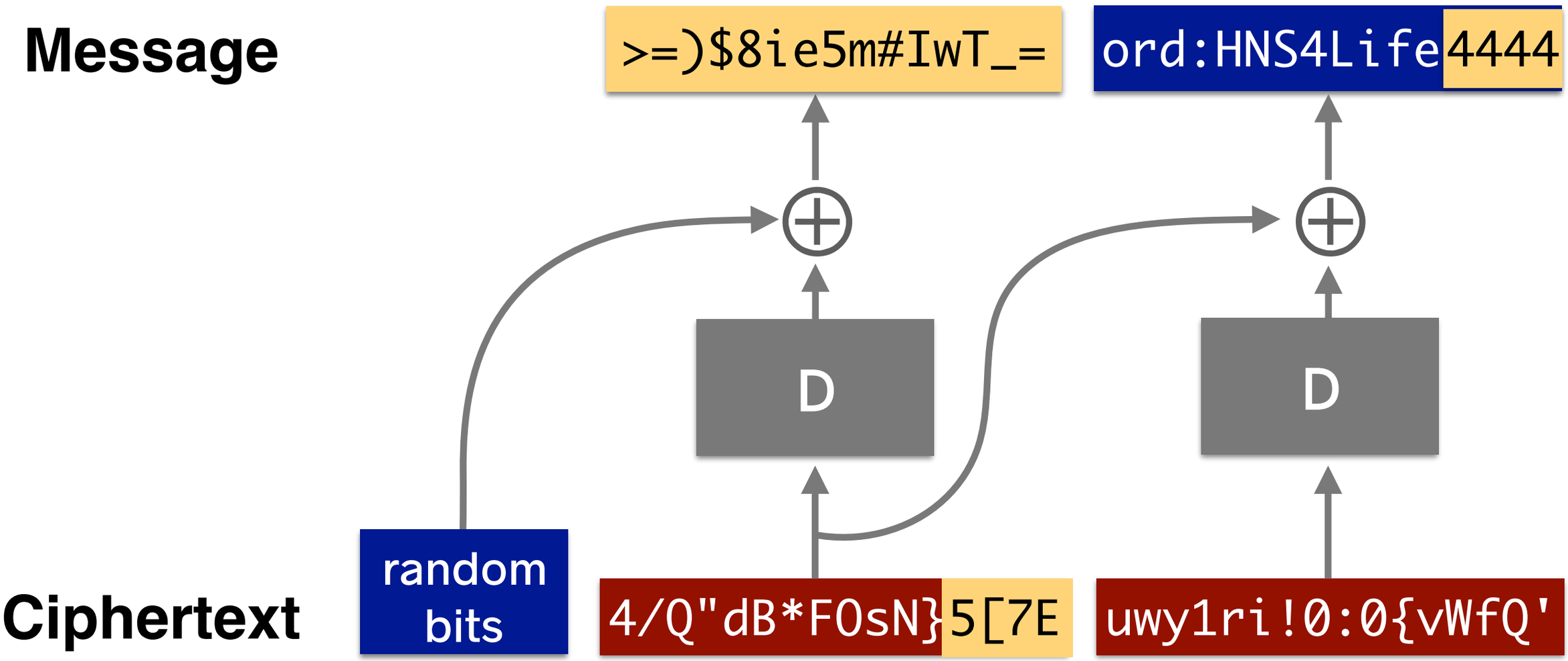
4) Learn the last non-padding byte



Server response

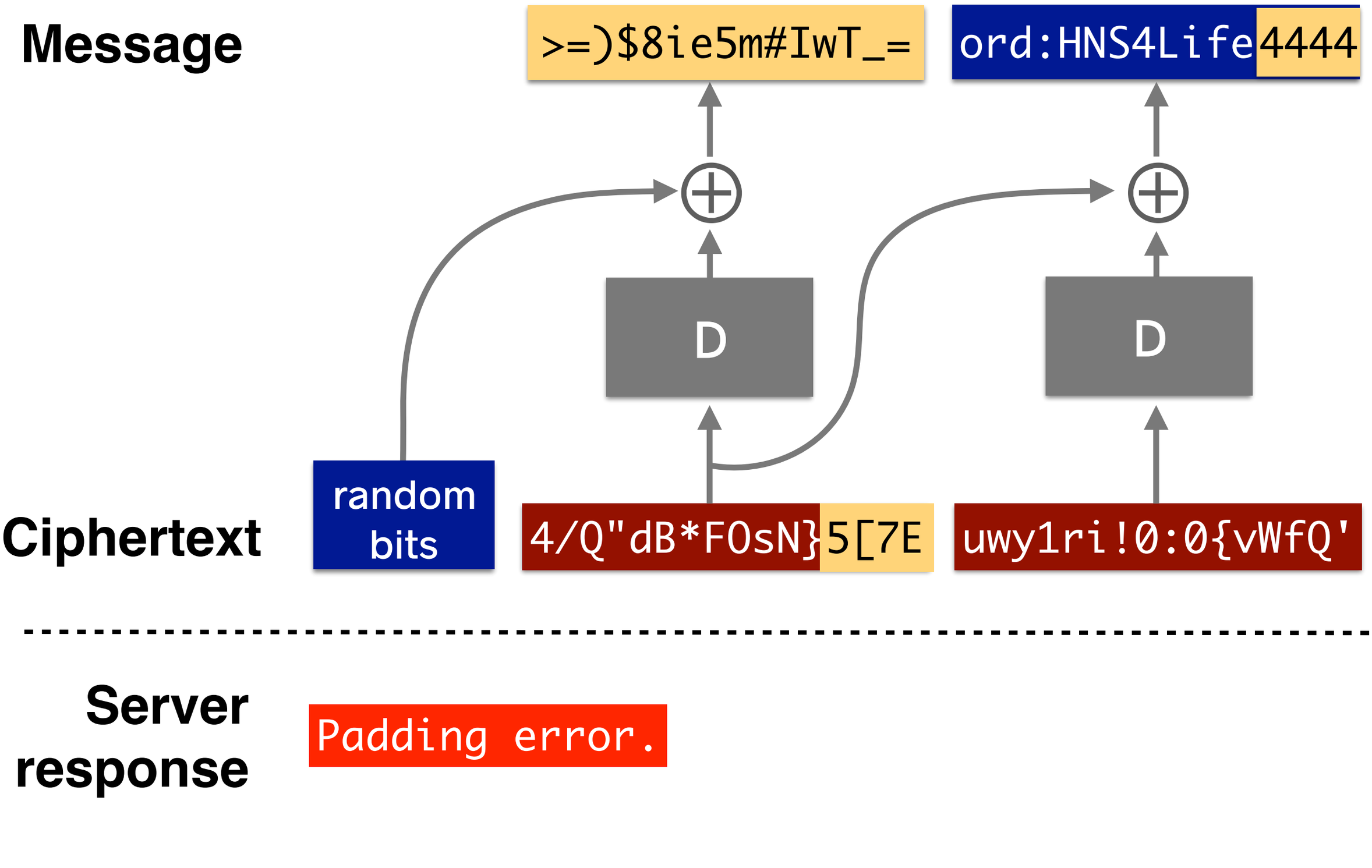
```
Invalid request; Please try again.
```

5) Continue attack to learn remaining bytes



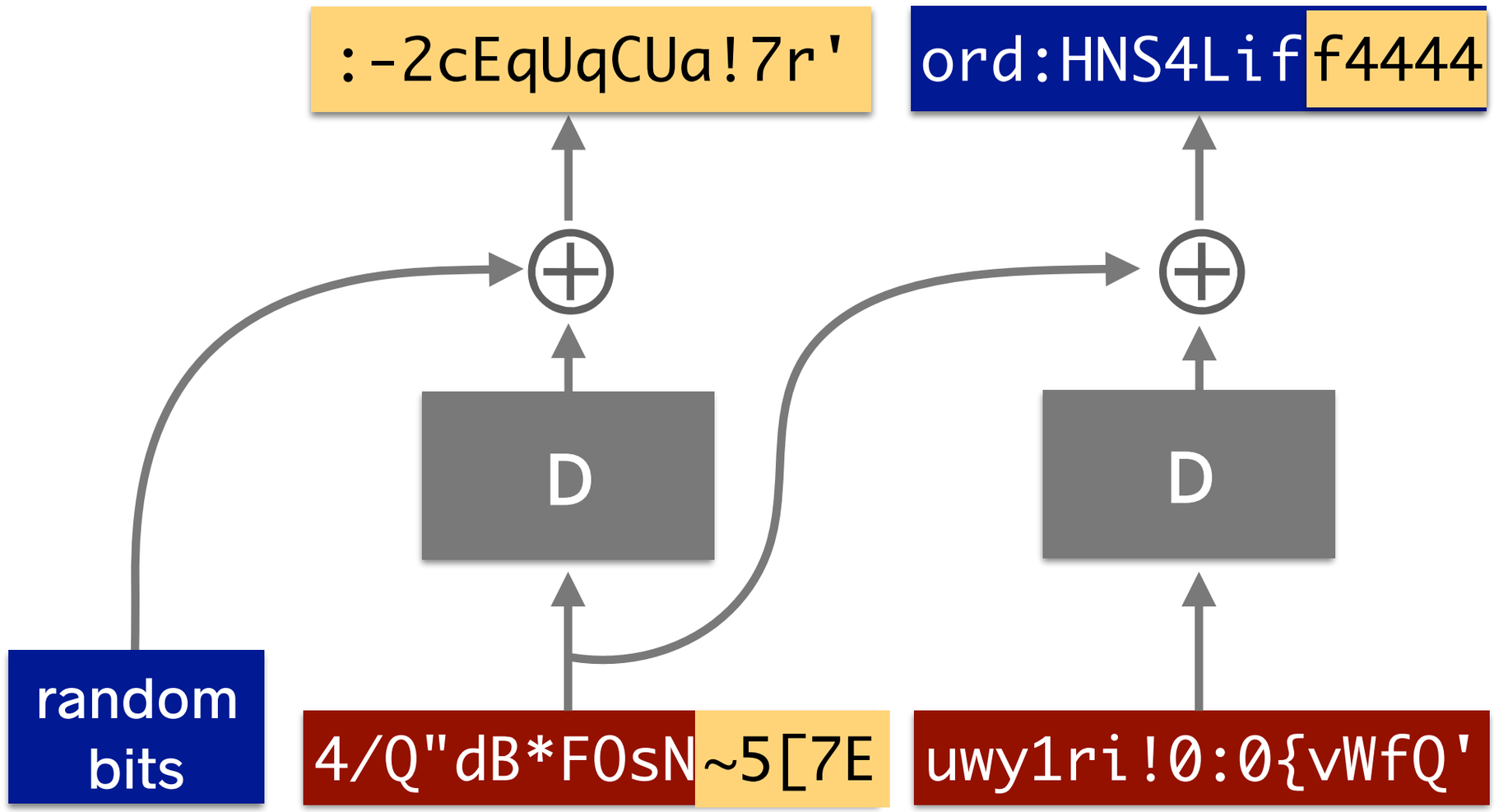
Server response

5) Continue attack to learn remaining bytes



5) Continue attack to learn remaining bytes

Message

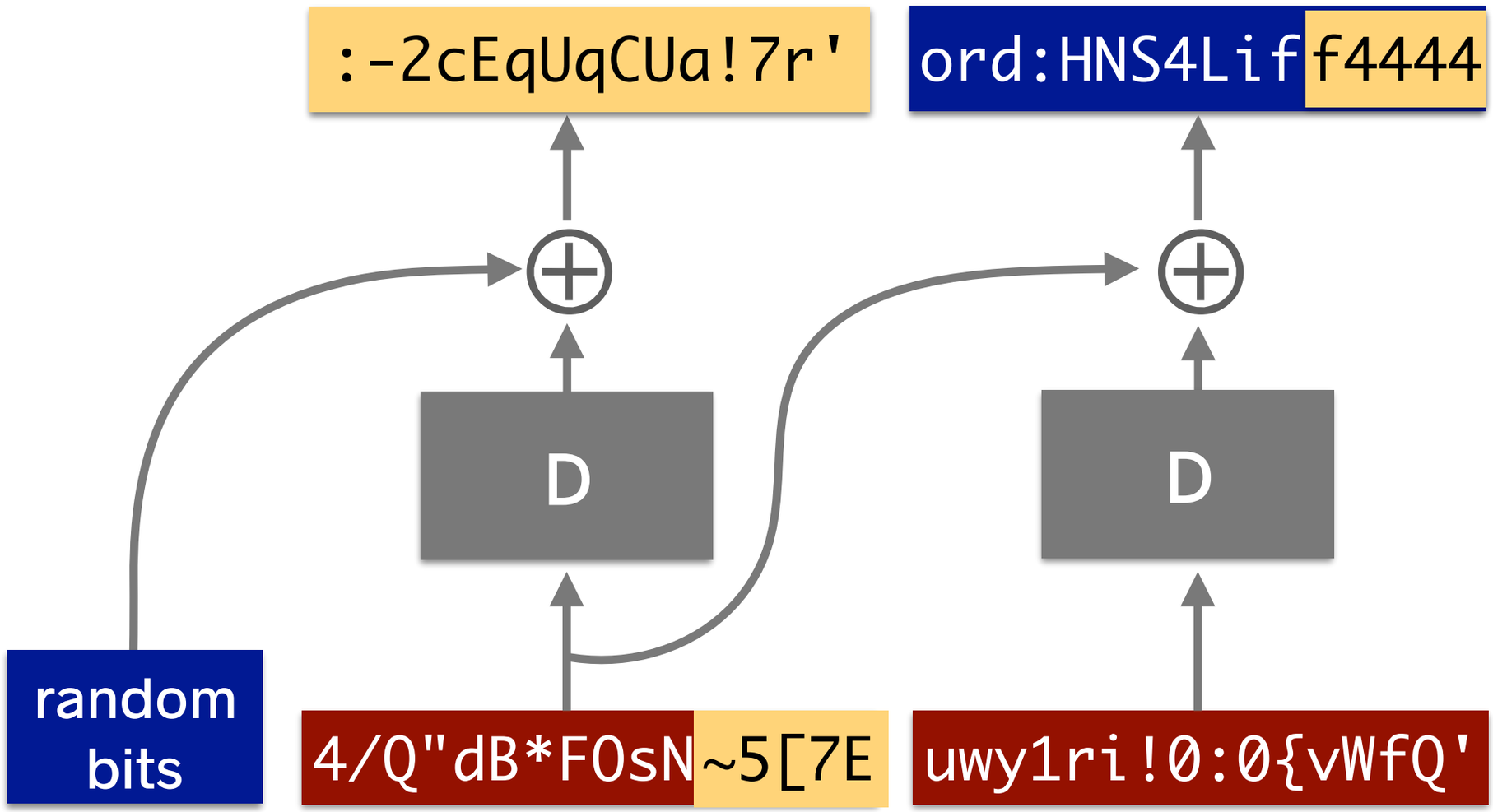


Ciphertext

Server response

5) Continue attack to learn remaining bytes

Message

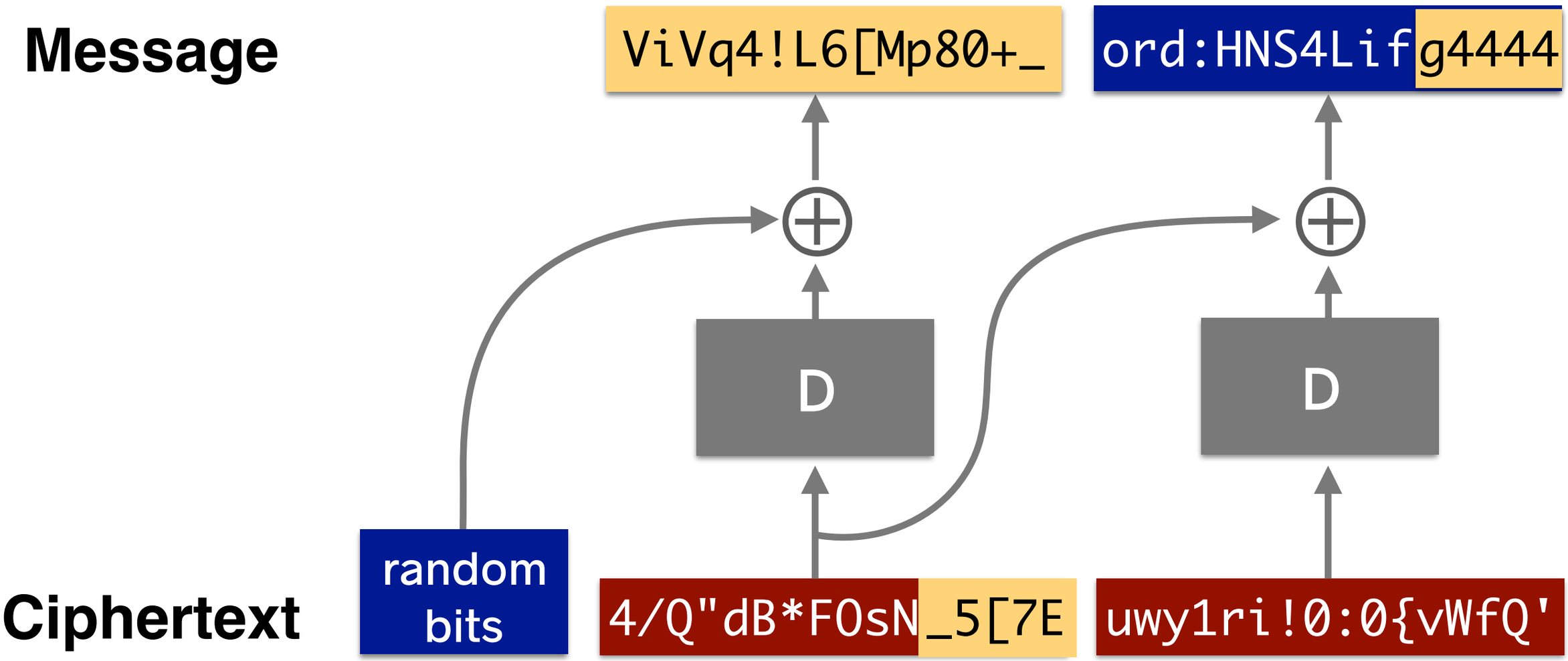


Ciphertext

Server response

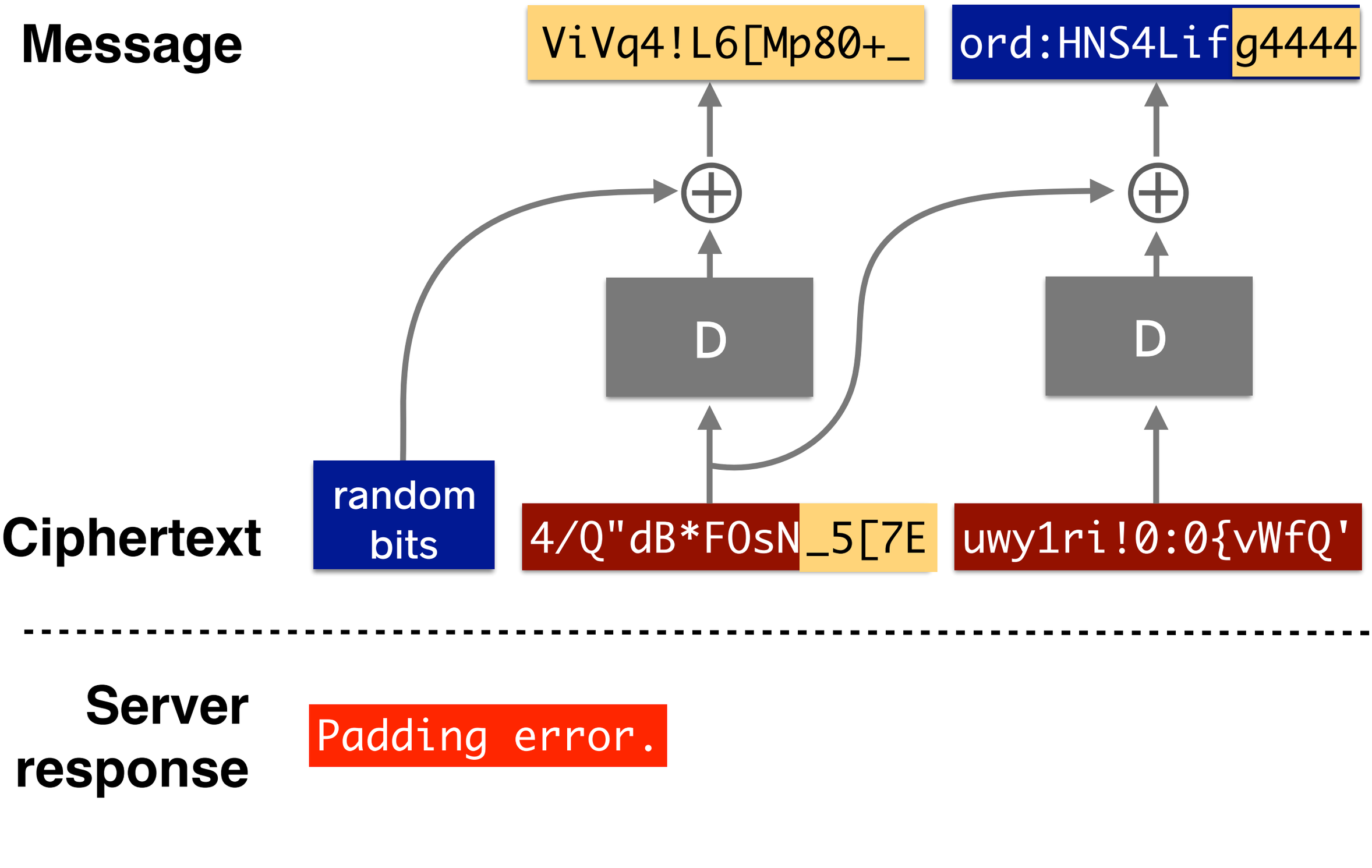
`Padding error.`

5) Continue attack to learn remaining bytes

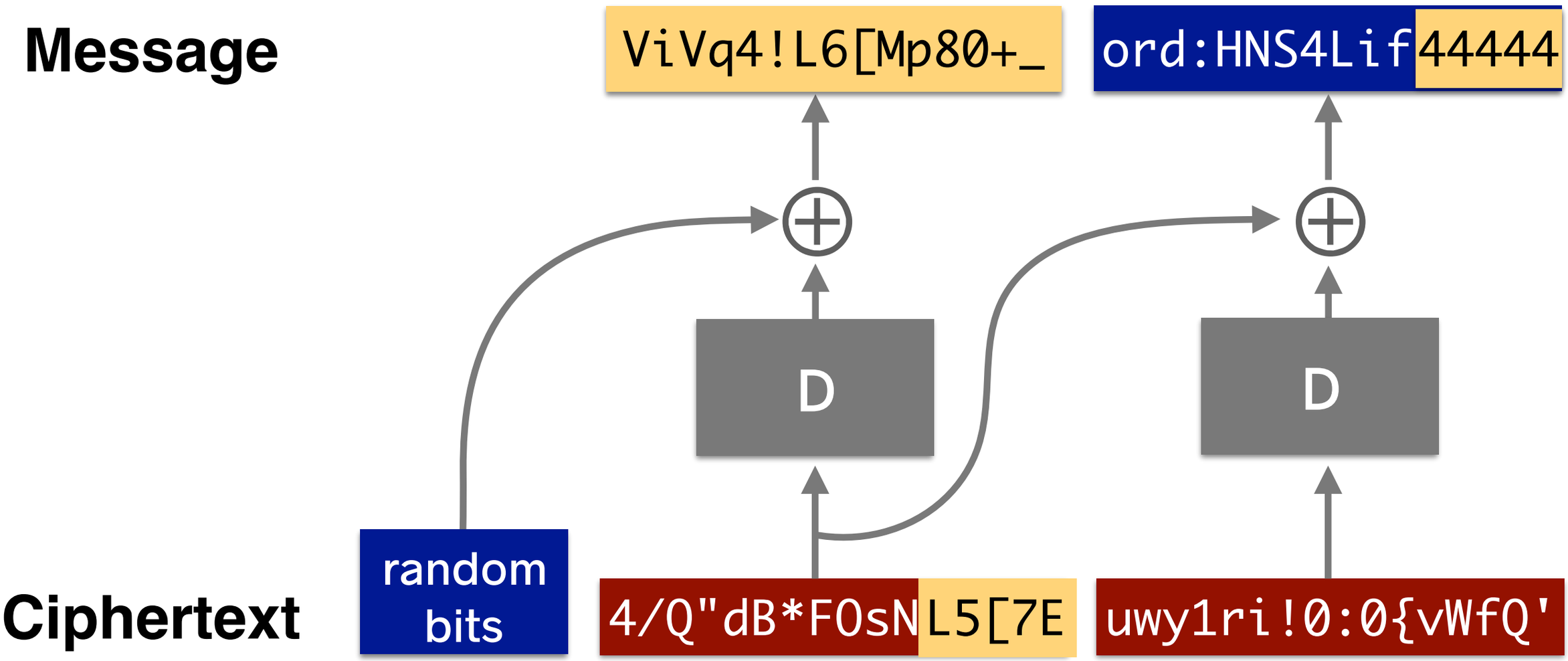


Server response

5) Continue attack to learn remaining bytes



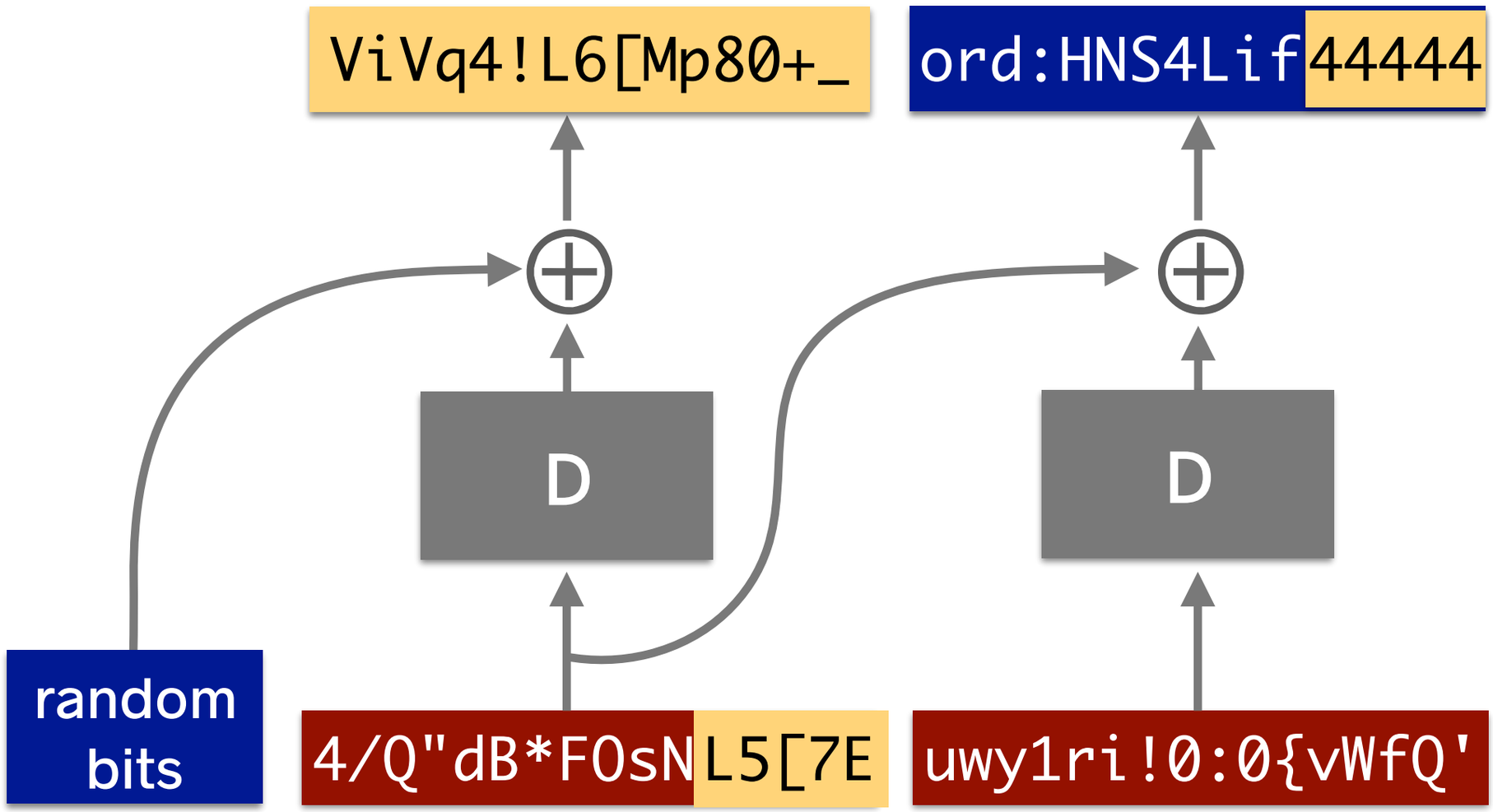
5) Continue attack to learn remaining bytes



Server response

5) Continue attack to learn remaining bytes

Message

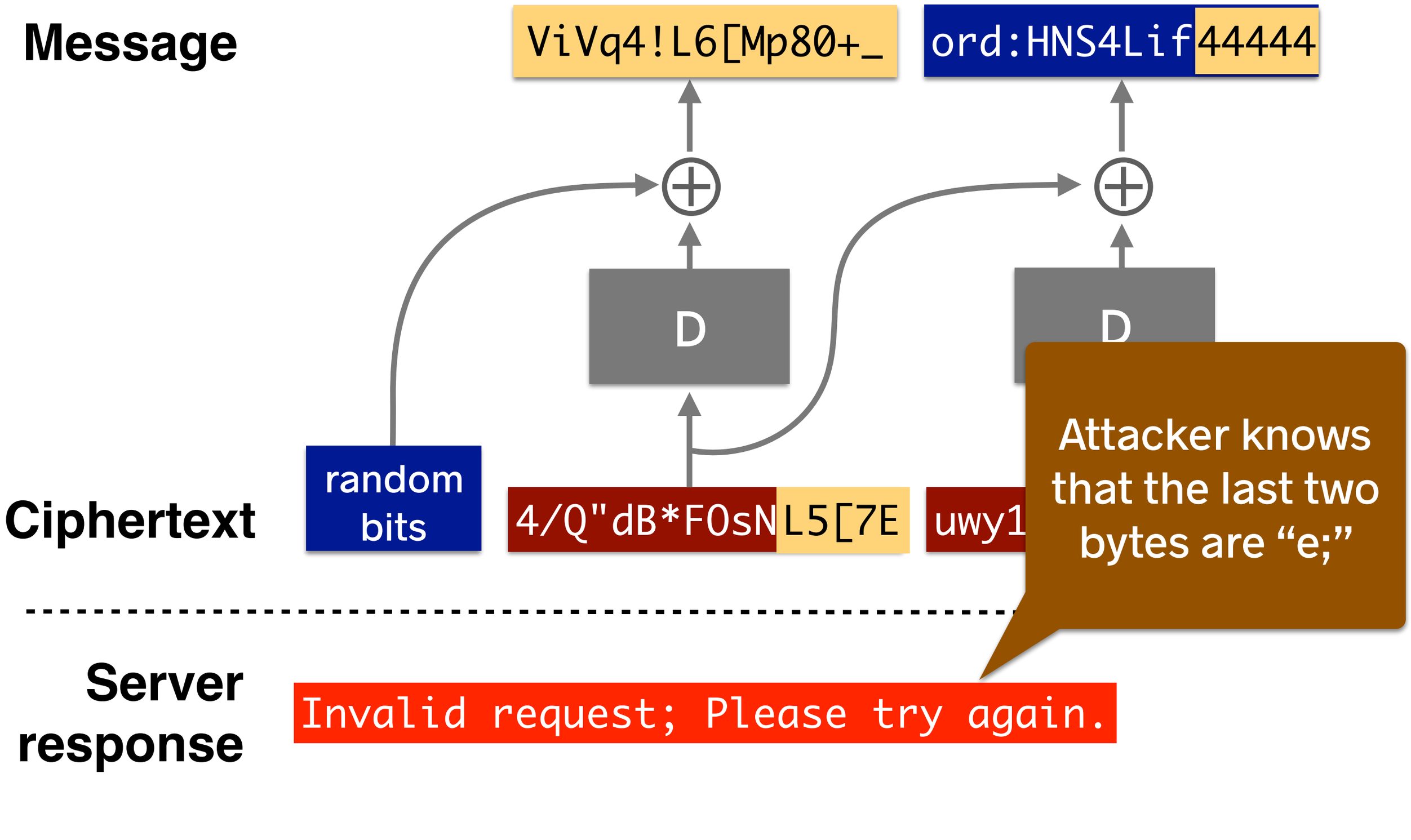


Ciphertext

Server response

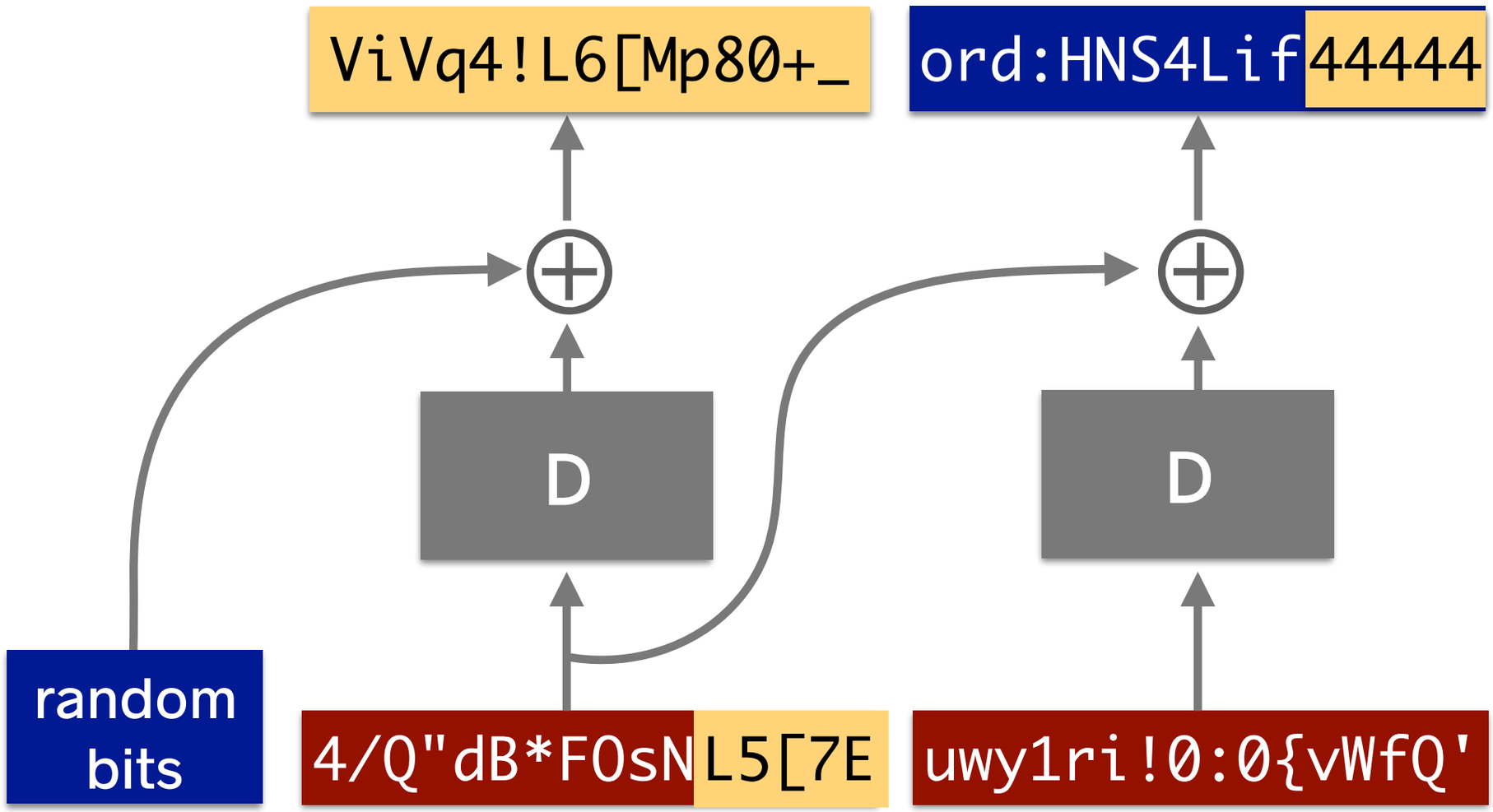
```
Invalid request; Please try again.
```

5) Continue attack to learn remaining bytes



5) Continue attack to learn remaining bytes

Message



Ciphertext

Server response

```
Invalid request; Please try again.
```

CBC “Padding-oracle” attack

- The attacker learns the rest of the plaintext in a byte-by-byte fashion

- Needs 16*256 = 4,096 guesses to recover the last 16 bytes of the encrypted message
  -> This is completely practical in the right setting

- After watching your mail client for a few hours, the attacker recovers your mail password!

- Modern encryption modes avoid padding
  – GCM: “Galois counter mode”

Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Goals of this talk

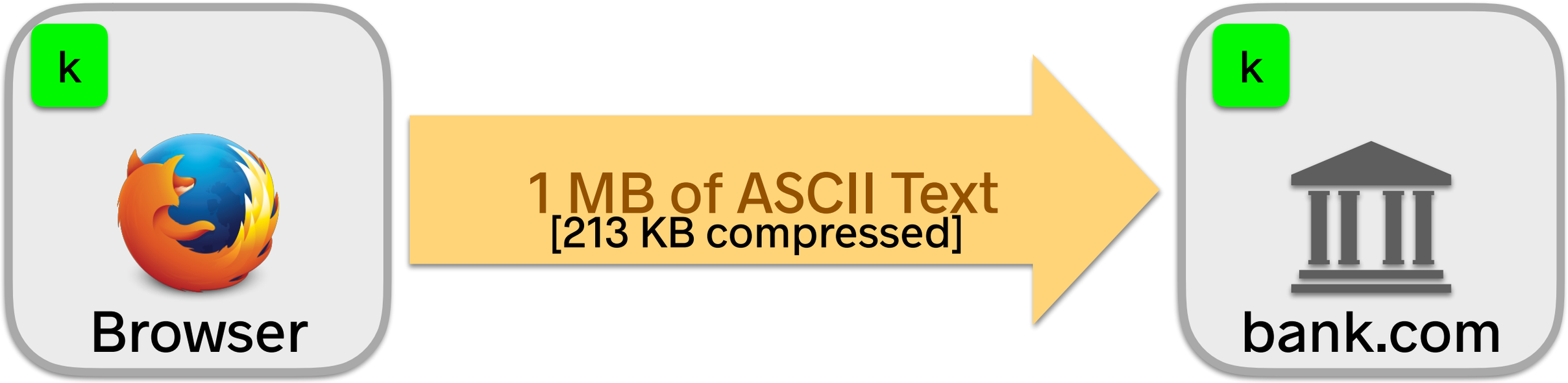
1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Compression and encryption

- We often send compressible data (e.g., ASCII text)
- Compressed data -> better apparent network throughput



Compression and TLS

Encrypt then compress? Or, compress then encrypt?

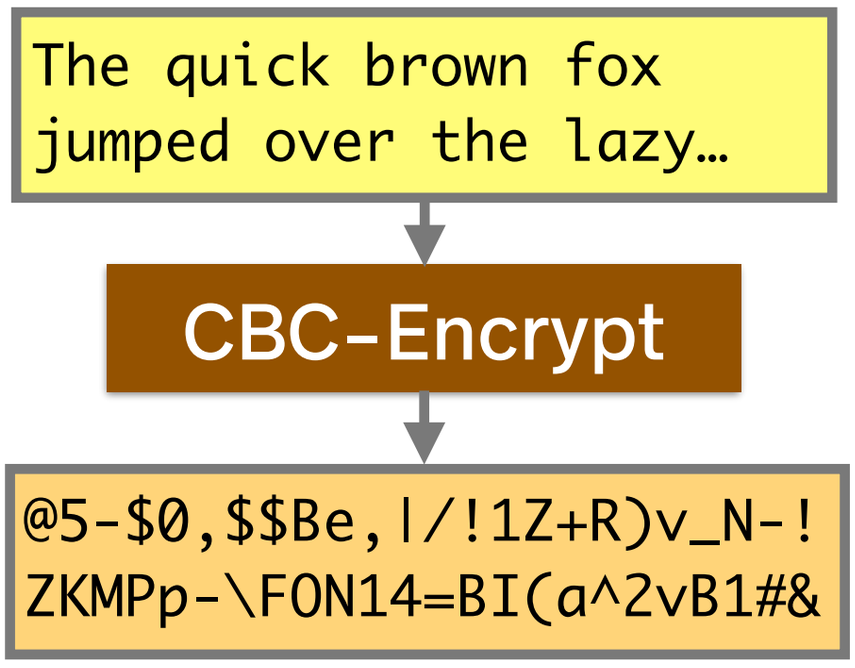
Compression and TLS

Encrypt then compress? Or, compress then encrypt?

```
The quick brown fox
jumped over the lazy…
```

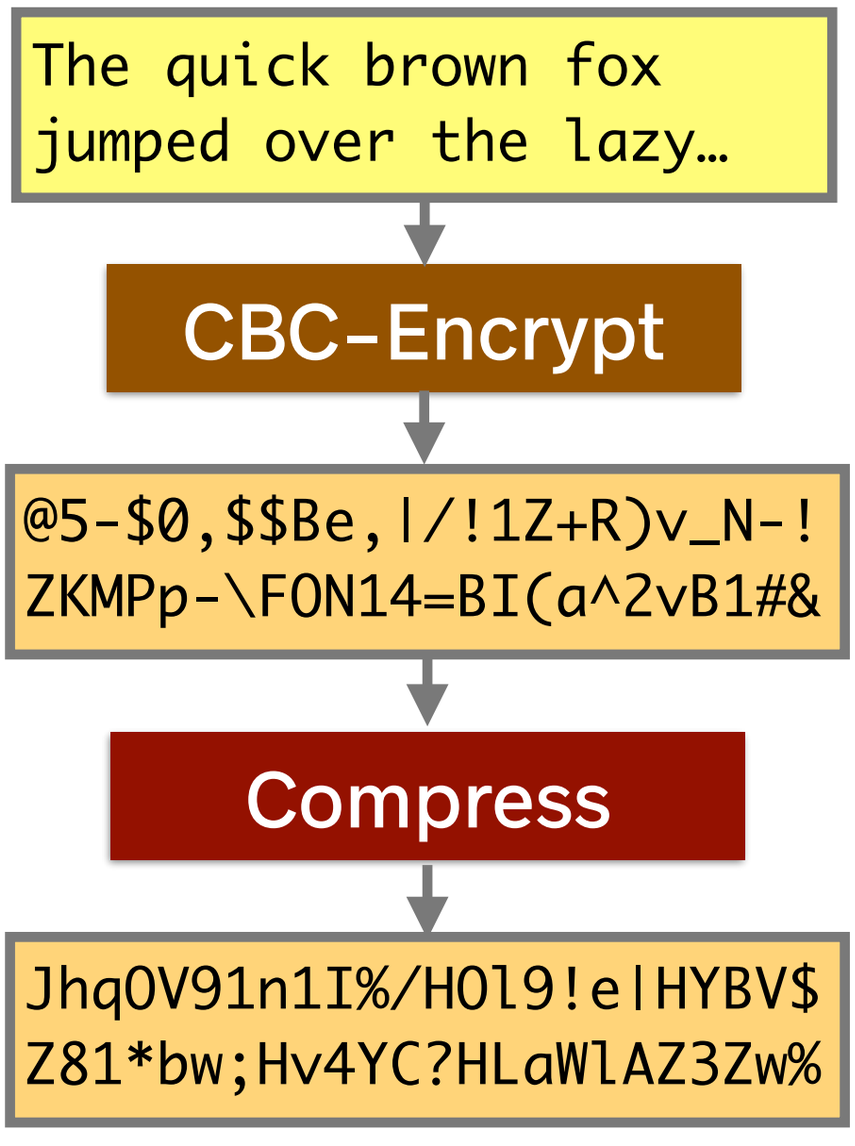
Compression and TLS

Encrypt then compress? Or, compress then encrypt?



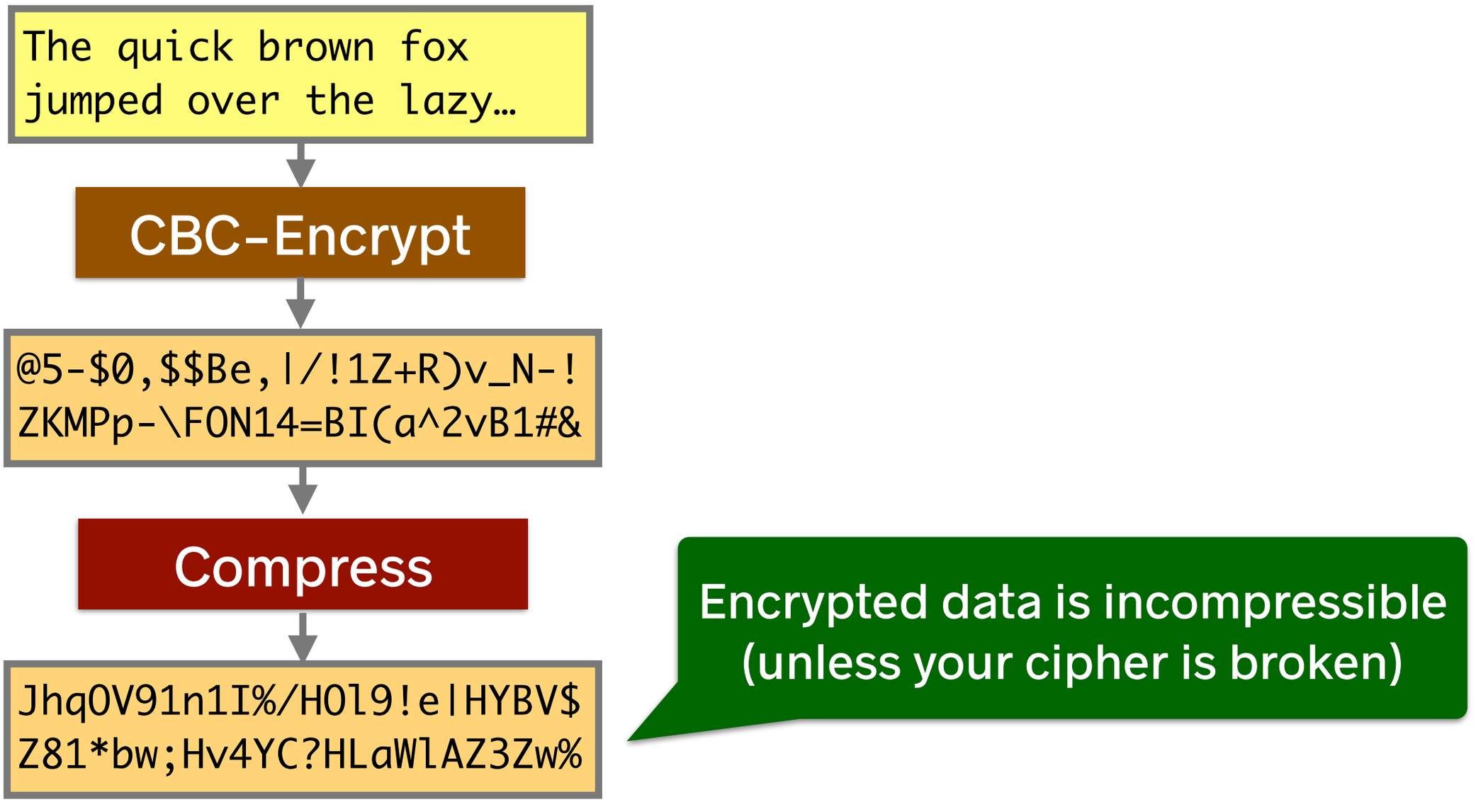
Compression and TLS

Encrypt then compress? Or, compress then encrypt?



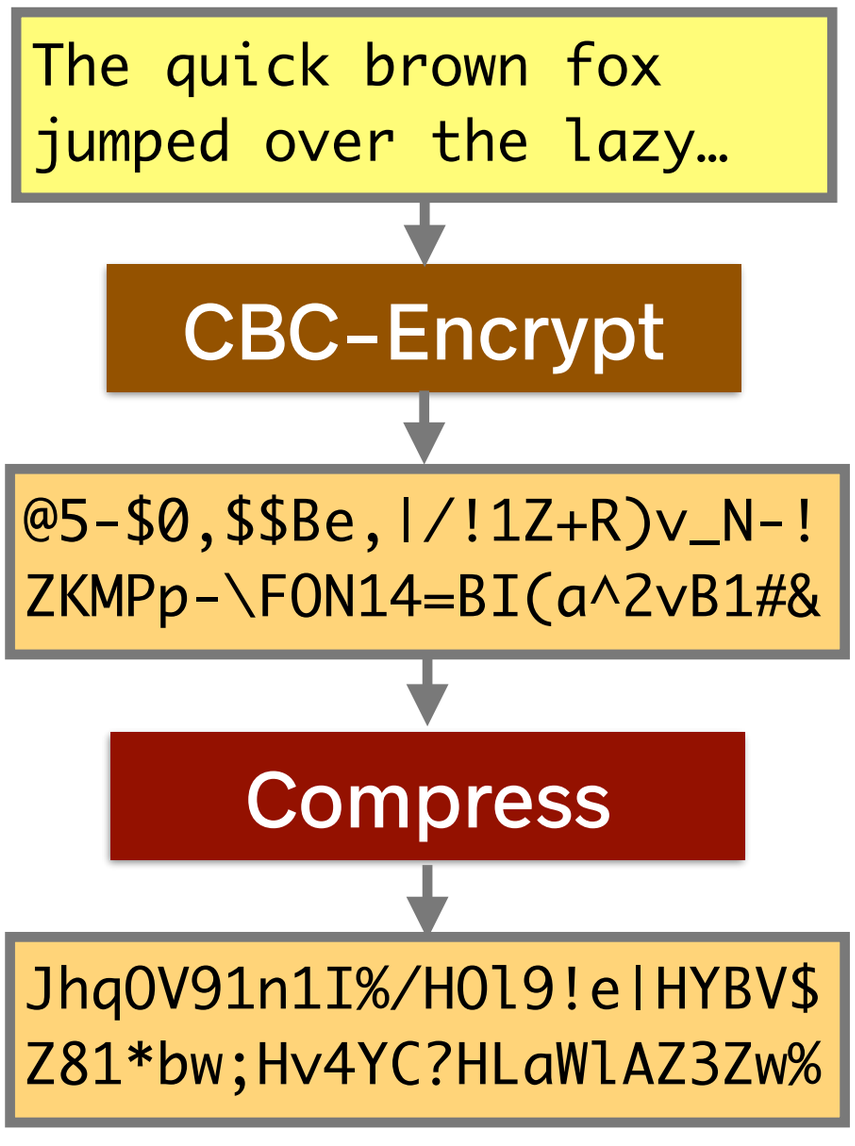
Compression and TLS

Encrypt then compress? Or, compress then encrypt?



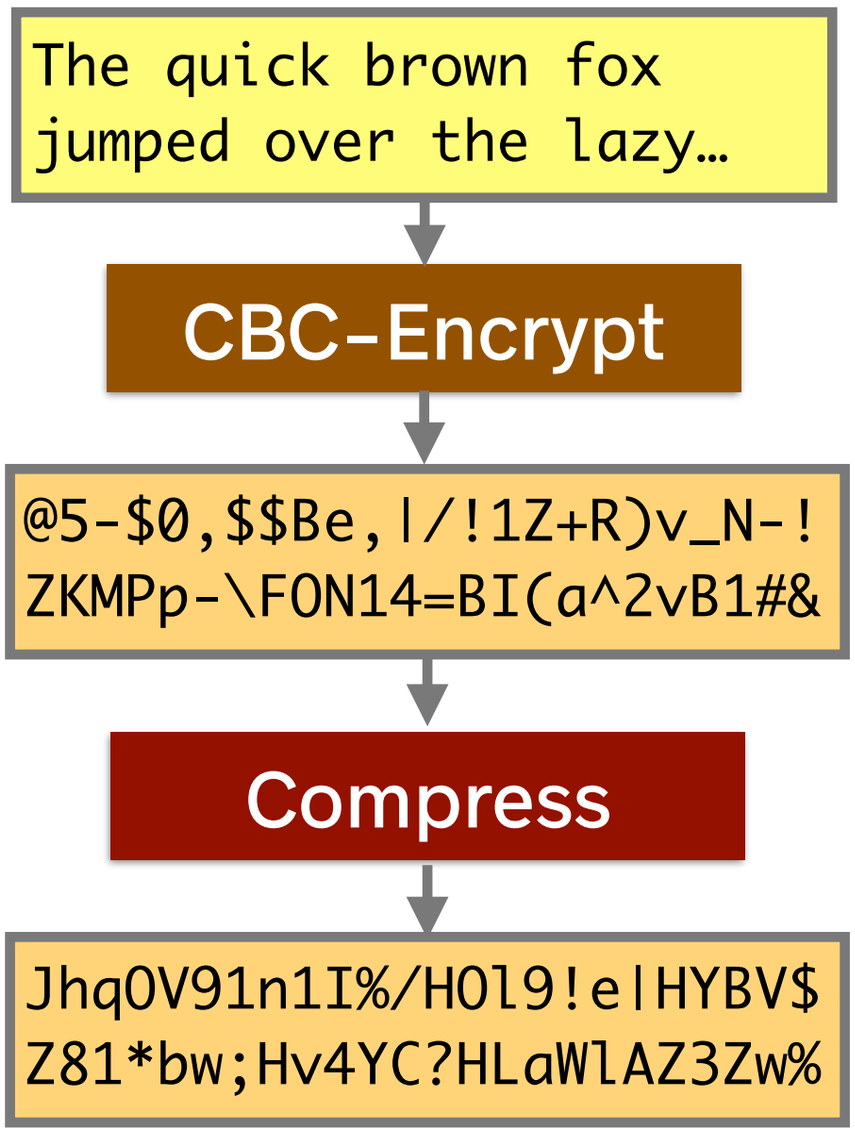
Compression and TLS

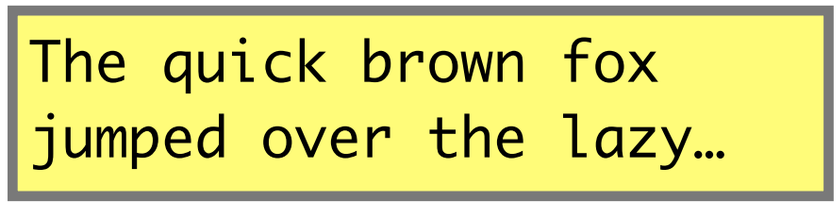
Encrypt then compress? Or, compress then encrypt?



Compression and TLS

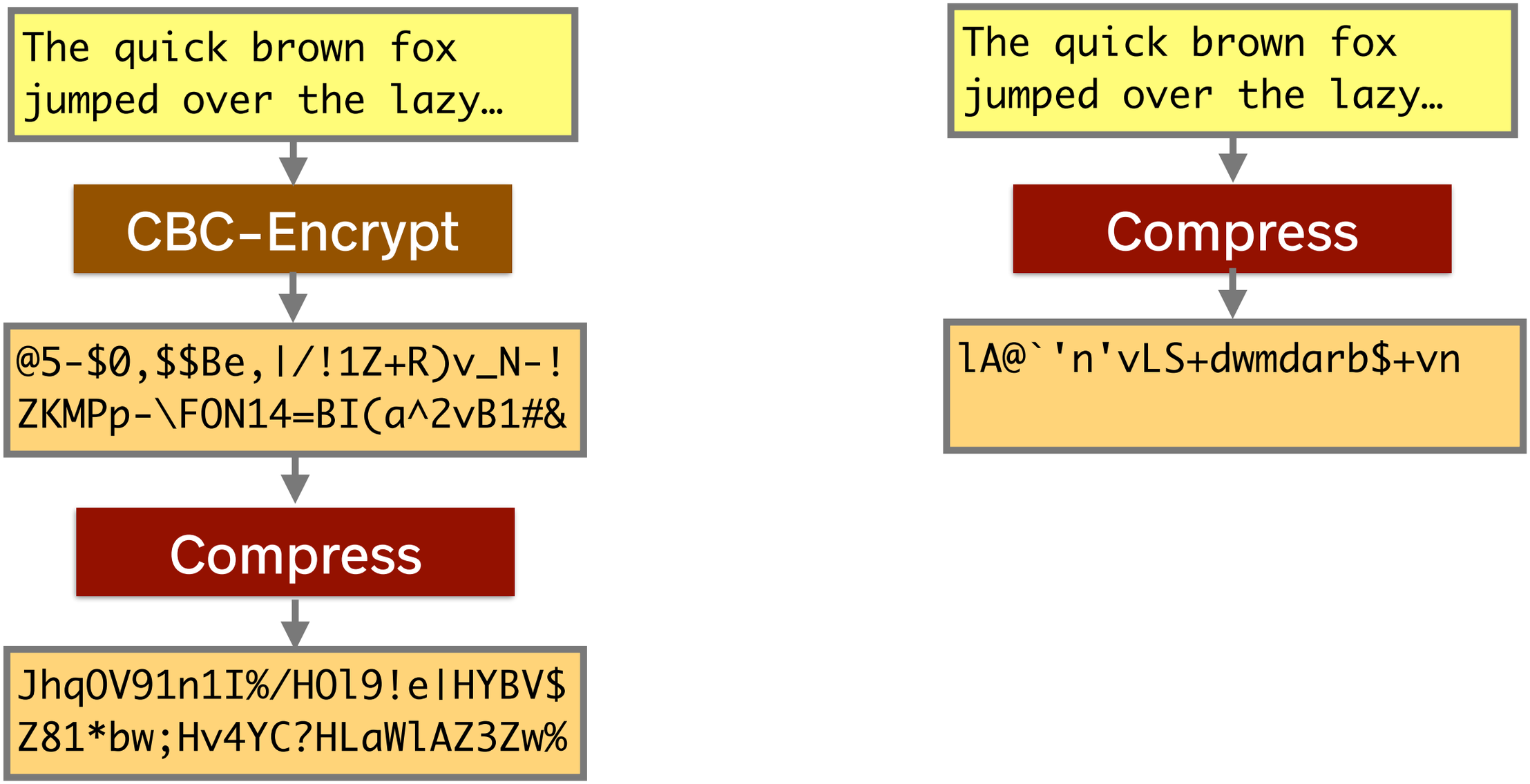
Encrypt then compress? Or, compress then encrypt?





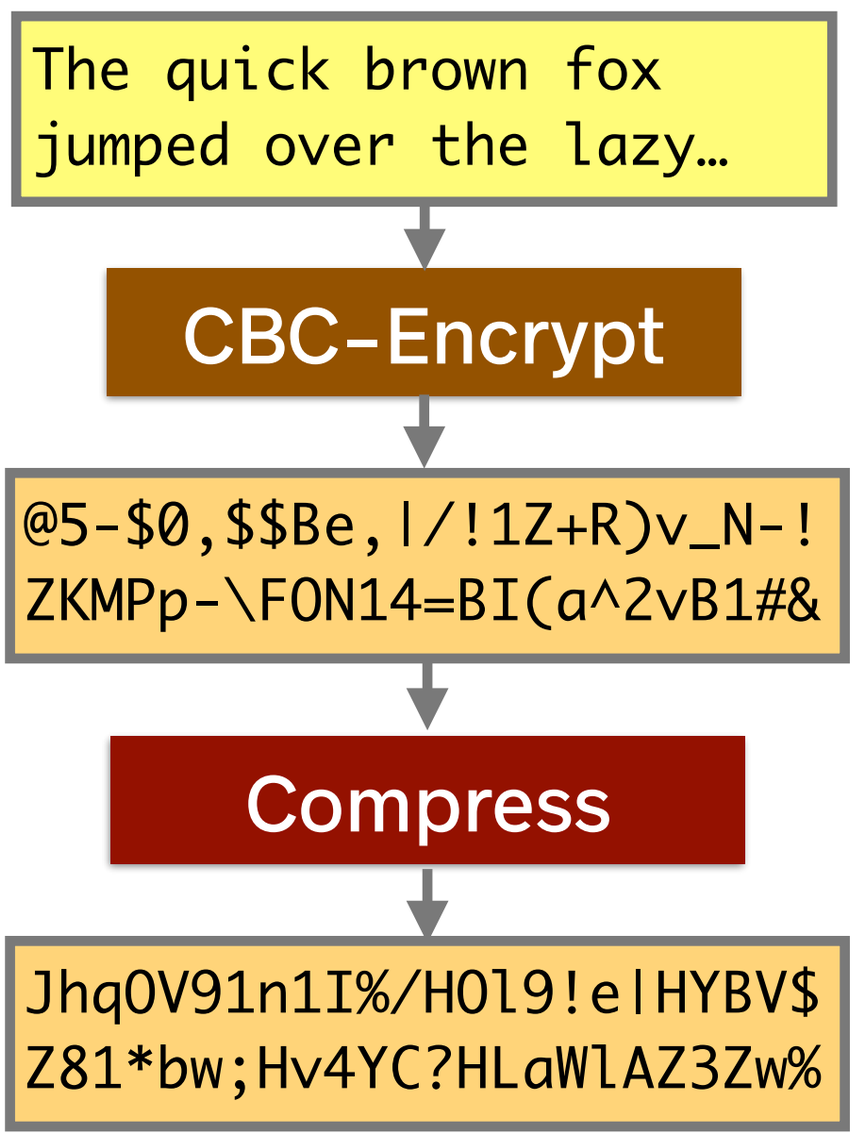
Compression and TLS

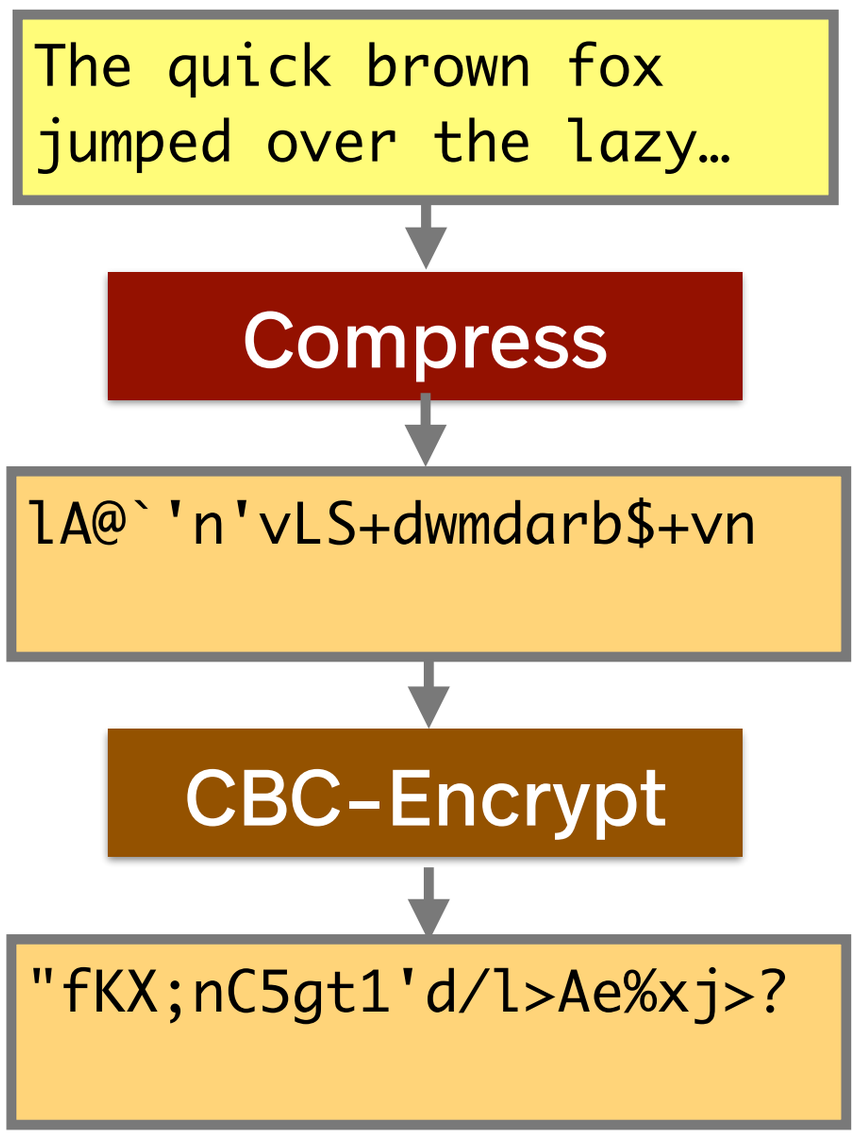
Encrypt then compress? Or, compress then encrypt?



Compression and TLS

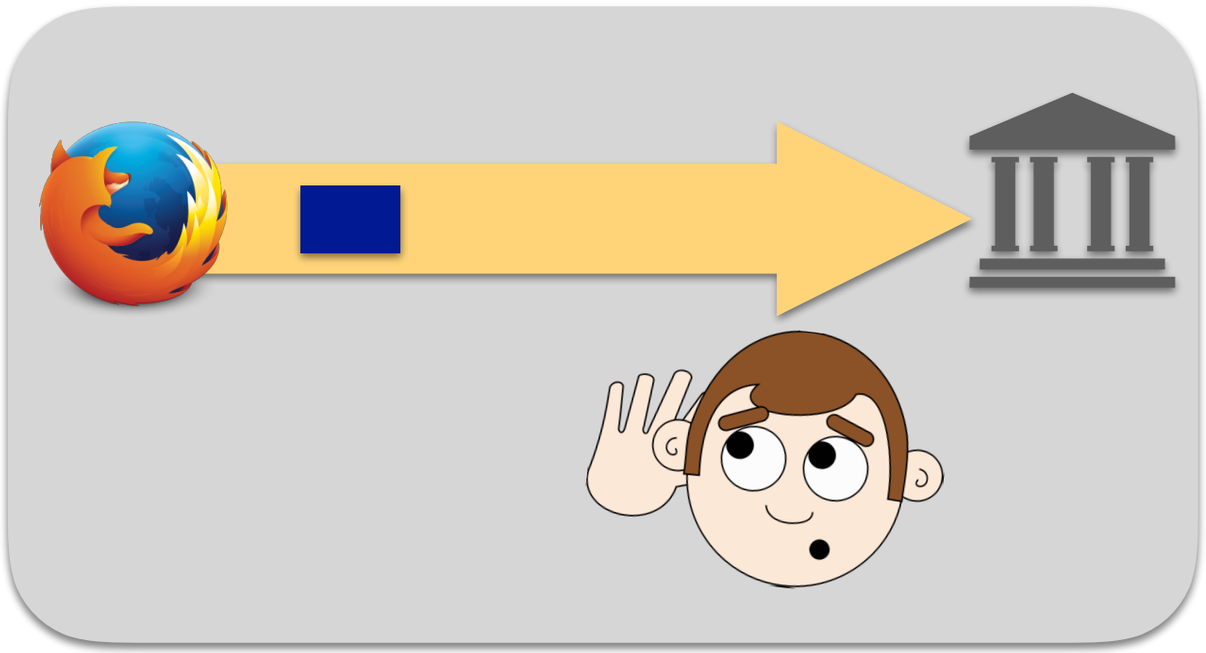
Encrypt then compress? Or, compress then encrypt?



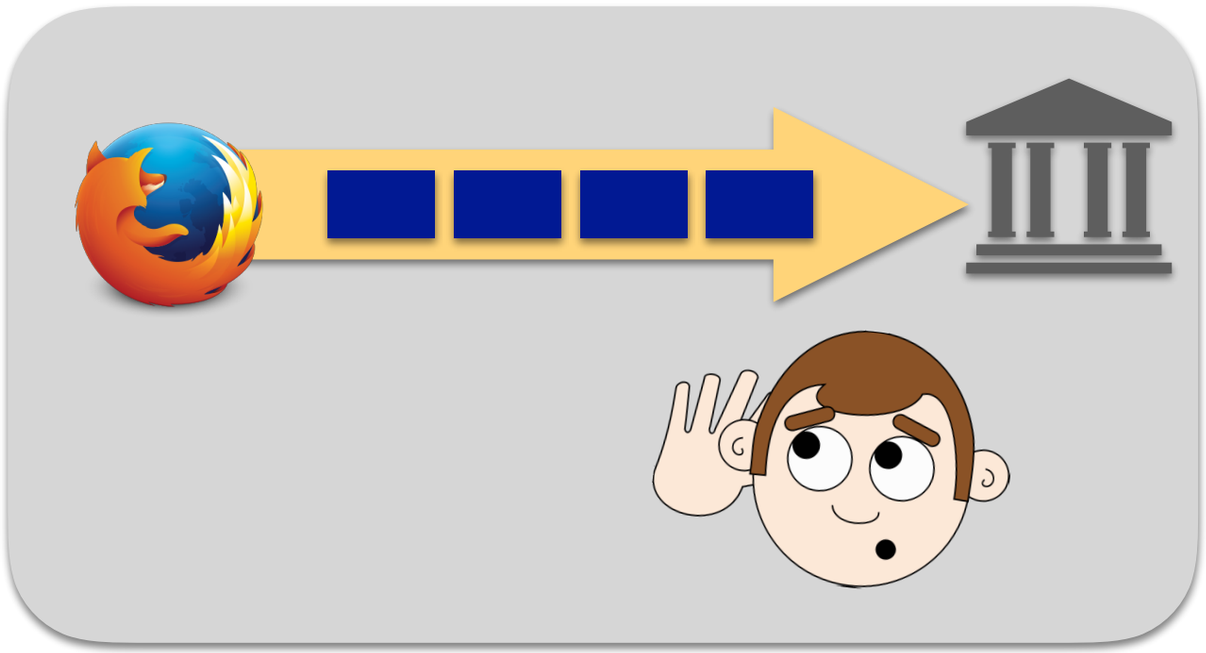


Compress-then-encrypt

PROBLEM
Ciphertext length leaks information to attacker.



Message is compressible.



Message not compressible.

Compress-then-encrypt

Compress-then-encrypt

PROBLEM

Ciphertext length leaks information to attacker.

Compress-then-encrypt

PROBLEM
Ciphertext length leaks information to attacker.

- Simple example: attacker can tell if you're sending the string 000000...00000 versus a random string

Compress-then-encrypt

PROBLEM
Ciphertext length leaks information to attacker.

- Simple example: attacker can tell if you're sending the string 000000...00000 versus a random string
- More interesting: attacker can tell if you're streaming "Mad Men" or "Sopranos" from Netflix

Compress-then-encrypt

PROBLEM
Ciphertext length leaks information to attacker.

- Simple example: attacker can tell if you're sending the string 000000...00000 versus a random string
- More interesting: attacker can tell if you're streaming "Mad Men" or "Sopranos" from Netflix
- Does this really matter?

Attacking compress-then-encrypt

Attacking compress-then-encrypt

- When you log into bank.com, your bank stores a “cookie” in your browser
  - Cookie could be a 128-bit random string
  - Anyone with your bank cookie can access your account

Attacking compress-then-encrypt

- When you log into bank.com, your bank stores a “cookie” in your browser
  - Cookie could be a 128-bit random string
  - Anyone with your bank cookie can access your account

- If you visit evil.com, an attacker can trick your browser into sending many encryptions of:

“`attacker-chosen-string || Cookie:secret-bank-token`”

Attacking compress-then-encrypt

- When you log into bank.com, your bank stores a “cookie” in your browser
  - Cookie could be a 128-bit random string
  - Anyone with your bank cookie can access your account

- If you visit evil.com, an attacker can trick your browser into sending many encryptions of:

“attacker-chosen-string || Cookie:secret-bank-token”

```
<html><title>Evil.com</title>
<img src="https://bank.com/?attacker-chosen-string1">
<img src="https://bank.com/?attacker-chosen-string2">
<img src="https://bank.com/?attacker-chosen-string3">
...
```

Attacking compress-then-encrypt

- When you log into bank.com, your bank stores a “cookie” in your browser
  - Cookie could be a 128-bit random string
  - Anyone with your bank cookie can access your account

- If you visit evil.com, an attacker can trick your browser into sending many encryptions of:

“attacker-chosen-string || Cookie:secret-bank-token”

```
<html><title>Evil.com</title>
<img src="https://bank.com/?attacker-chosen-stri
<img src="https://bank.com/?attacker-chosen-string2">
<img src="https://bank.com/?attacker-chosen-string3">
...
```

Cookie sent with these requests

Attacking compress-then-encrypt

- When you log into bank.com, your bank stores a “cookie” in your browser
  - Cookie could be a 128-bit random string
  - Anyone with your bank cookie can access your account

- If you visit evil.com, an attacker can trick your browser into sending many encryptions of:

“attacker-chosen-string || Cookie:secret-bank-token”

```
<html><title>Evil.com</title>
<img src="https://bank.com/?attacker-chosen-string1">
<img src="https://bank.com/?attacker-chosen-string2">
<img src="https://bank.com/?attacker-chosen-string3">
...
```

Attacking compress-then-encrypt

https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf

Attacking compress-then-encrypt

Attack idea (CRIME and BREACH TLS attacks)

[Rizzo&Duong'12] [Prado&others'13]

1. For many choices of s, trick user into sending encryptions of
   `<s || Cookie:secret-bank-token>`

https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf

Attacking compress-then-encrypt

Attack idea (CRIME and BREACH TLS attacks)

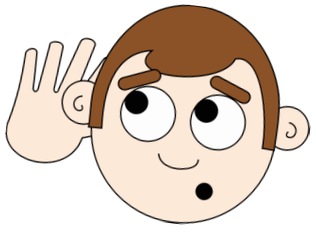
[Rizzo&Duong'12] [Prado&others'13]

1. For many choices of s, trick user into sending encryptions of
   `<s || Cookie:secret-bank-token>`
2. Observe how the length of the encrypted string varies.
3. Recover the password.

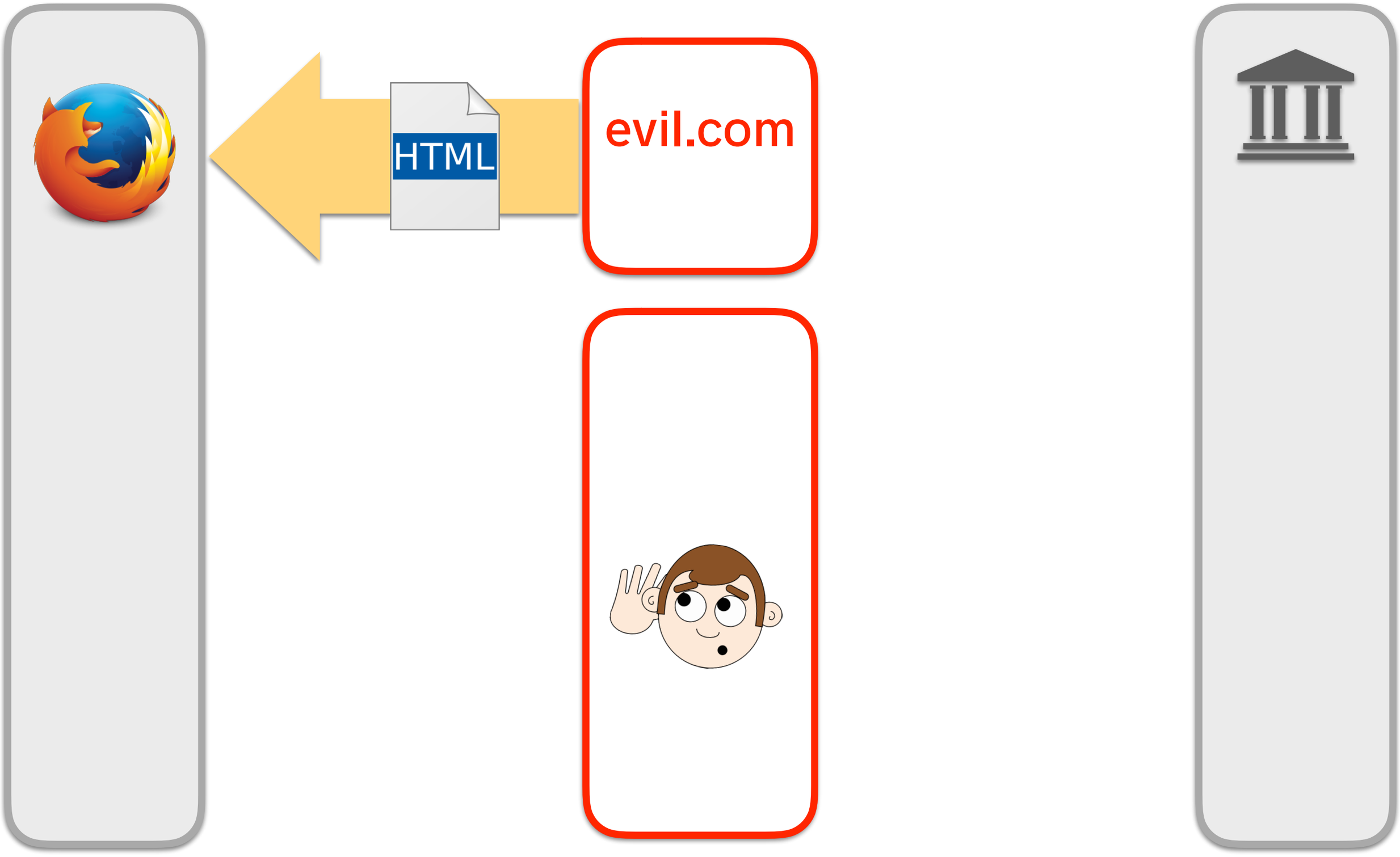
https://www.ietf.org/proceedings/85/slides/slides-85-saag-1.pdf

Attack idea

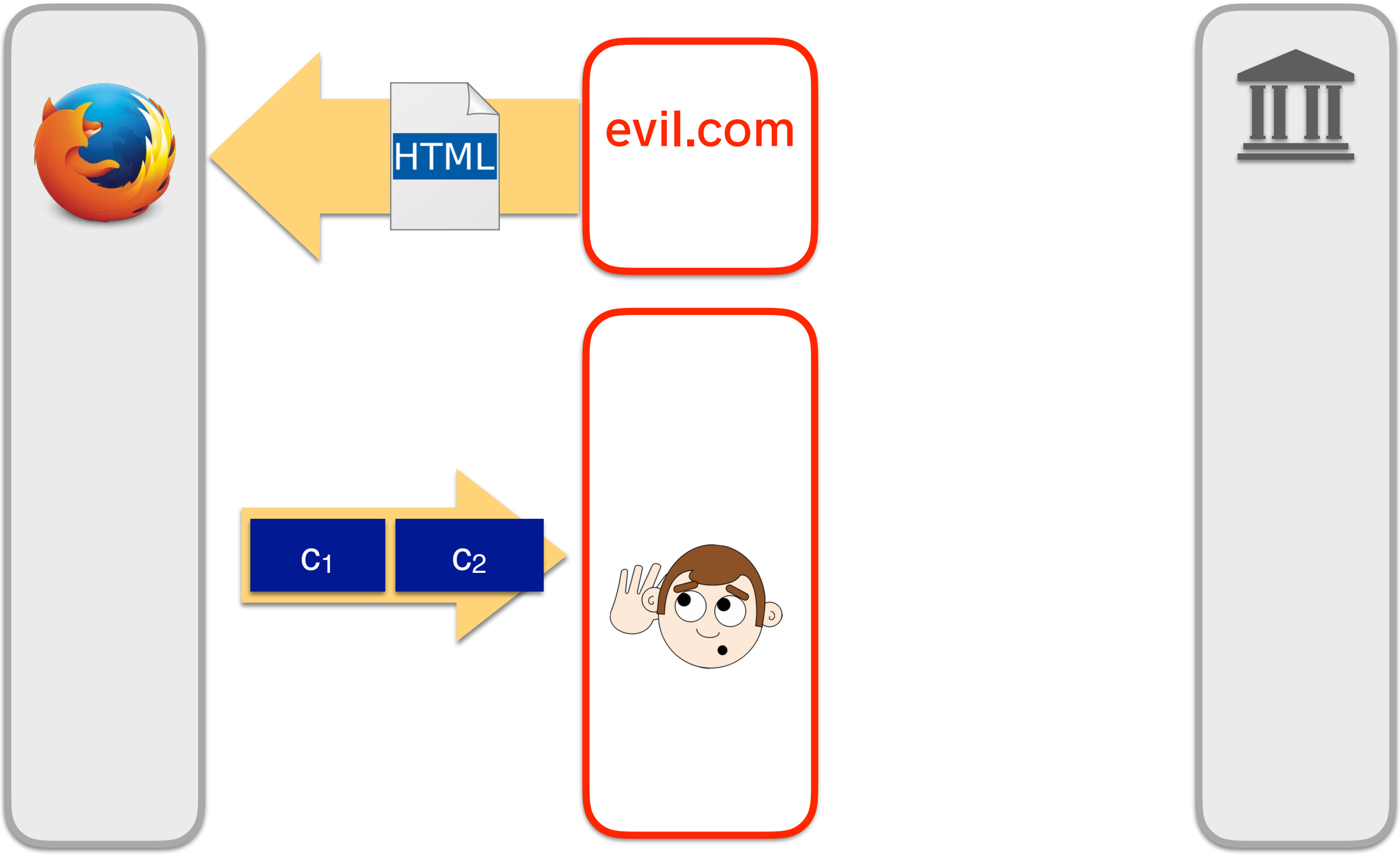
evil.com



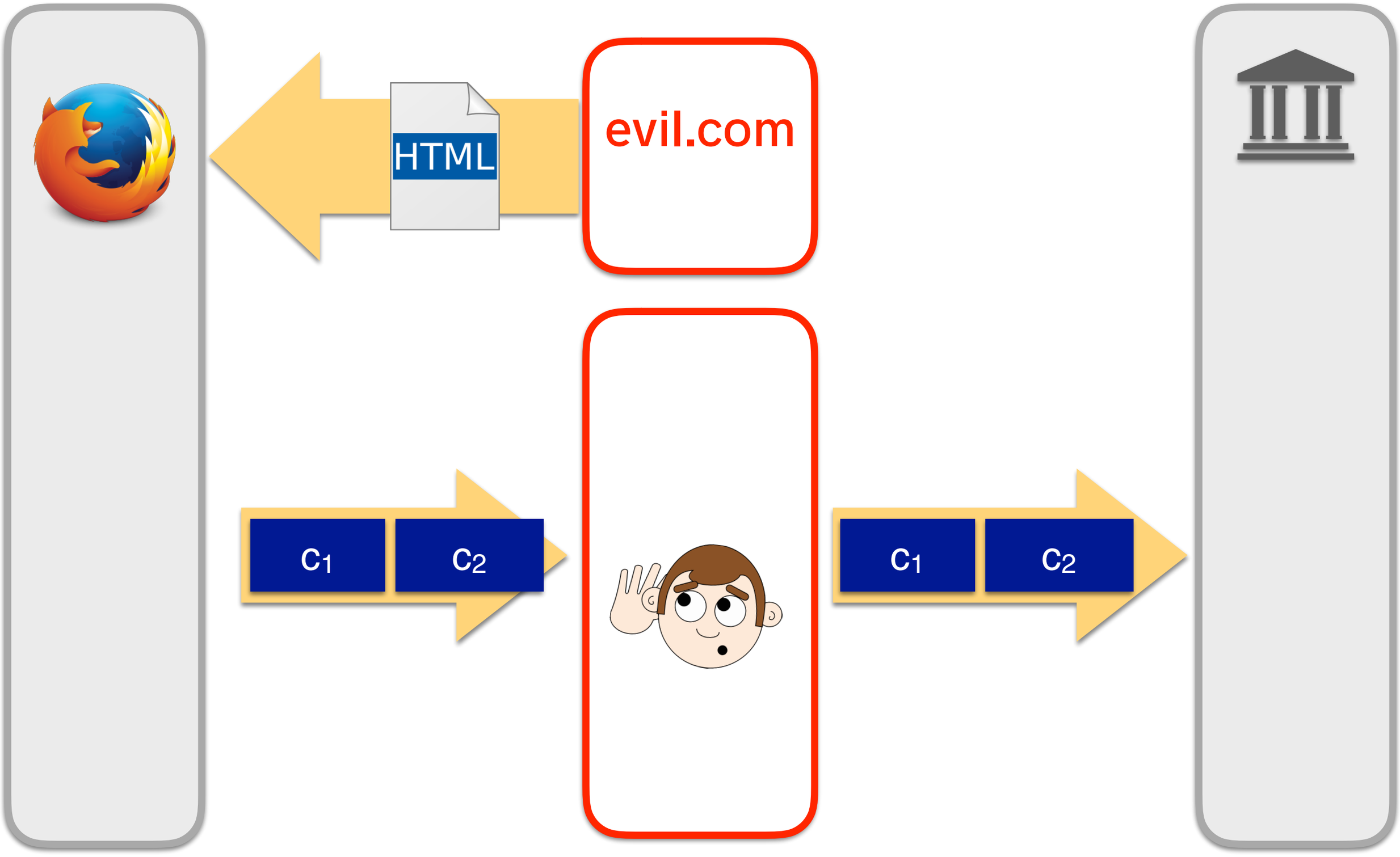
Attack idea



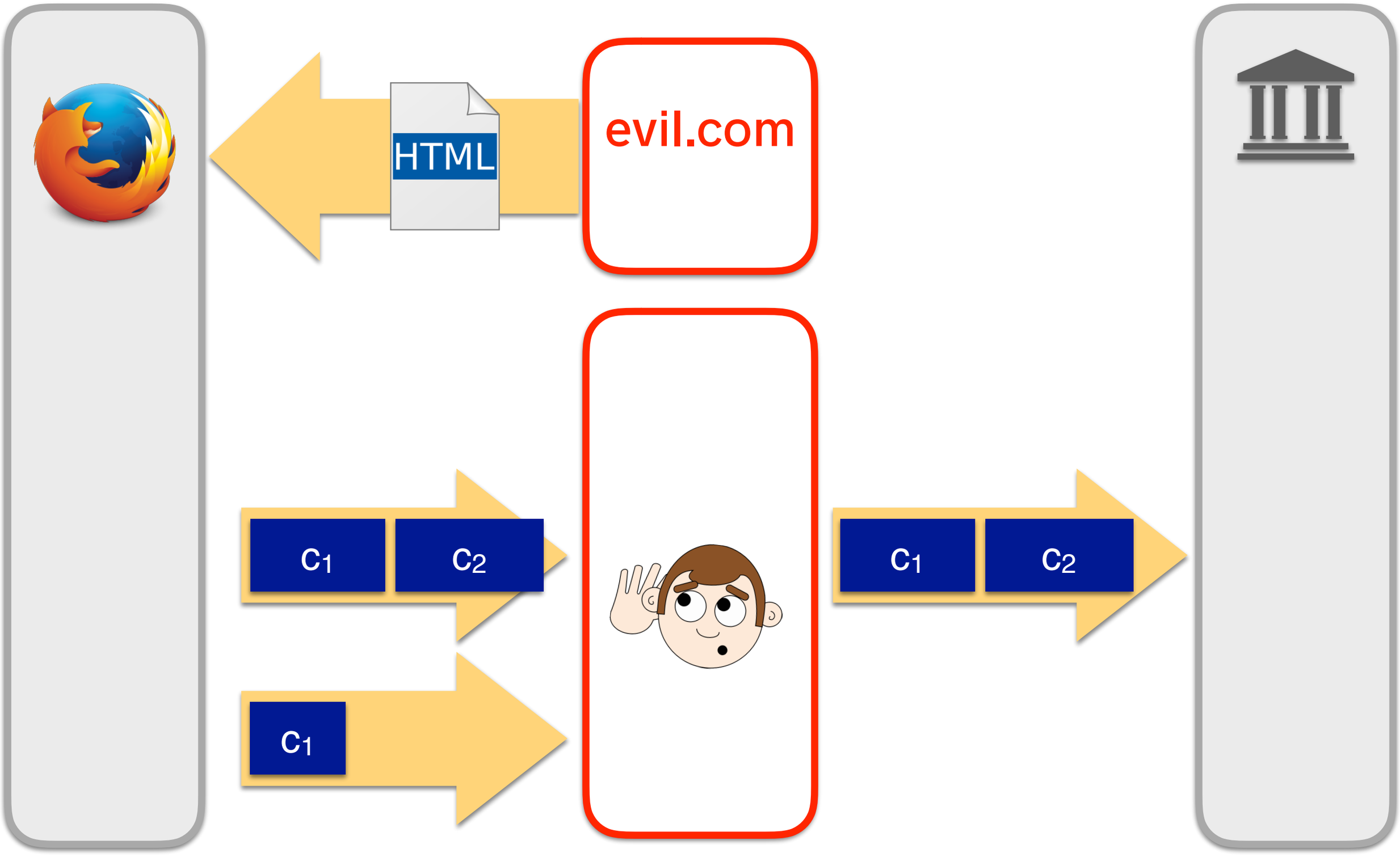
Attack idea



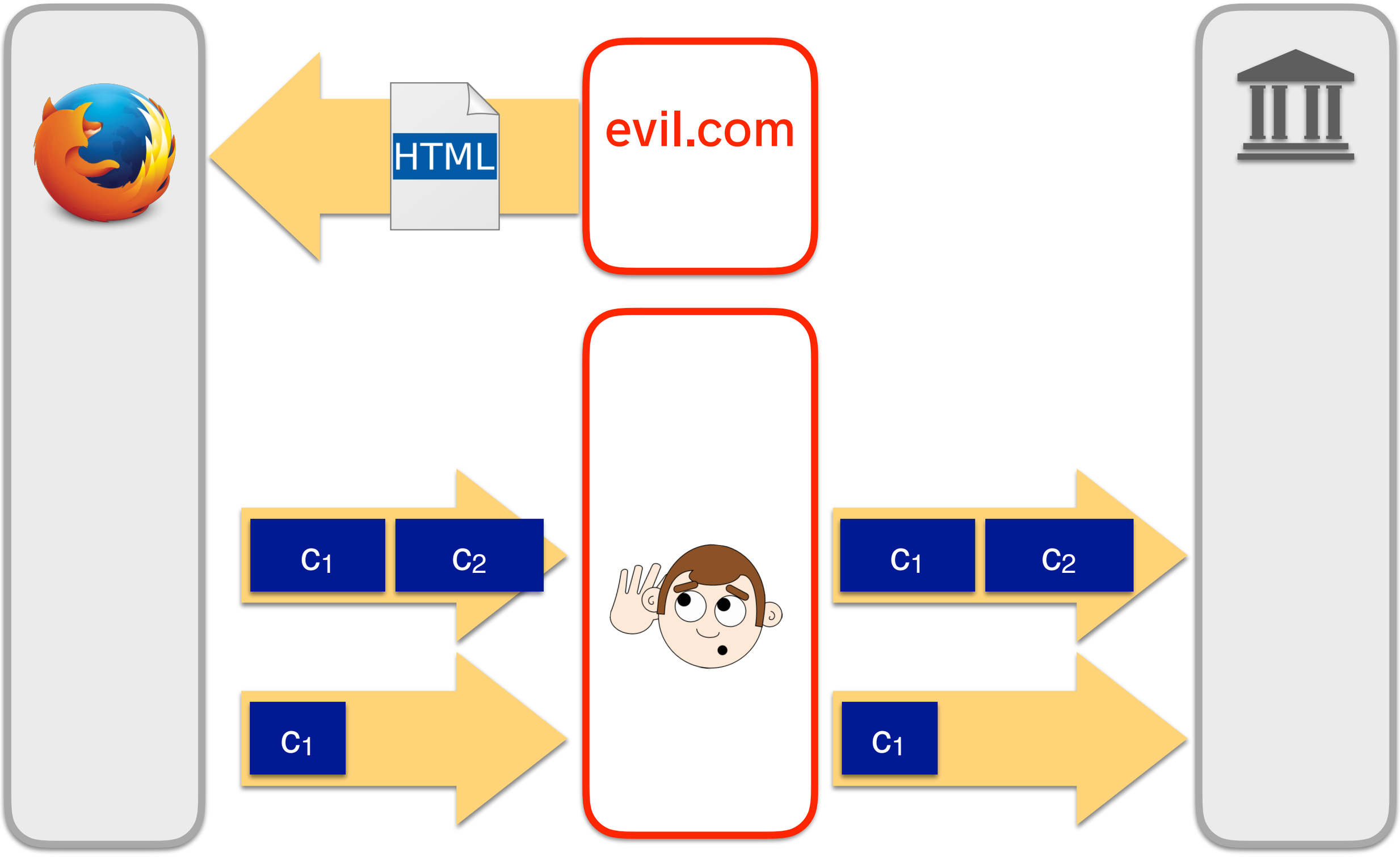
Attack idea



Attack idea



Attack idea



Attack idea

Attack idea

Say that the cookie is HX8f$Q.
Then

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q` compresses to 24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
...		

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes

Attacker knows that cookie starts with “H”

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:HX8f$Q`		24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:HX8f$Q`		24 bytes
`Cookie:HCCookie:HX8f$Q`		24 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:HX8f$Q`		24 bytes
`Cookie:HCCookie:HX8f$Q`		24 bytes
…		

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:HX8f$Q`		24 bytes
`Cookie:HCCookie:HX8f$Q`		24 bytes
…		
`Cookie:HXCookie:HX8f$Q`		23 bytes

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:H`		24 bytes
`Cookie:HCCookie:H`		24 bytes
…		
`Cookie:HXCookie:HX8f$Q`		23 bytes

Attacker knows that cookie starts with “HX”

Attack idea

Say that the cookie is HX8f$Q.
Then

`Cookie:ACookie:HX8f$Q`	compresses to	24 bytes
`Cookie:BCookie:HX8f$Q`		24 bytes
`Cookie:CCookie:HX8f$Q`		24 bytes
…		
`Cookie:HCookie:HX8f$Q`		23 bytes
`Cookie:HACookie:HX8f$Q`		24 bytes
`Cookie:HBCookie:HX8f$Q`		24 bytes
`Cookie:HCCookie:HX8f$Q`		24 bytes
…		
`Cookie:HXCookie:HX8f$Q`		23 bytes

Compress-then-encrypt

- For cookie of length L:
  - Cost of this attack: 256^{L} work
  - Brute-force guessing: 256^{L} work

- These attacks are practical!

- Mitigation? Turn off compression. :(
  - Can still use application-layer compression

Compress-then-encrypt

- For cookie of length L:
  - Cost of this attack: $256L$ work
  - Brute-force guessing: 256^L work

Exponential speed-up!

- These attacks are practical!

- Mitigation? Turn off compression. :(
  - Can still use application-layer compression

Compress-then-encrypt

- For cookie of length L:
  - Cost of this attack: 256^{L} work
  - Brute-force guessing: 256^{L} work

- These attacks are practical!

- Mitigation? Turn off compression. :(
  - Can still use application-layer compression

Moral of the story

Moral of the story

For end users:

- Demand encryption
  - Even weak encryption is better than none
- Use a modern browser and keep it up to date
  - Once found, these bugs get patched quickly

Moral of the story

For end users:

- Demand encryption
  – Even weak encryption is better than none
- Use a modern browser and keep it up to date
  – Once found, these bugs get patched quickly

For engineers:

- Encrypt everywhere
  – It costs almost nothing; you'll need it later anyways
- Never design a new crypto protocol
  – Off–the–shelf schemes suffice almost always
- Pay attention to small information leaks
  – Almost always: Small leak –> Large leak

Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Goals of this talk

1. Explain ideas behind link encryption (TLS).

2. Show why it's hard to get right.
   - Warm-up: ECB mode
   - Padding-oracle attack
   - Compress-then-encrypt attack

3. Highlight open problems and research directions.

Challenge 1: Secure-by-design protocols

https://eprint.iacr.org/2015/914.pdf

https://eprint.iacr.org/2015/978.pdf

https://tls13tamarin.github.io/TLS13Tamarin/

http://prosecco.gforge.inria.fr/personal/bblanche/publications/BhargavanBlanchetKobeissiSP2017.pdf

Challenge 1: Secure-by-design protocols

- TLS with “provable” protection against certain attacks?

- TLS v1.3 incorporates research along these lines
  - Specify the protocol in machine-readable language
  - Define formal security properties (e.g., confidentiality)
  - Prove that specification satisfies security properties

- Challenges:
  - Complexity of specification & of security goals
  - Implementation =? Specification
  - Effort required

https://eprint.iacr.org/2015/914.pdf

https://eprint.iacr.org/2015/978.pdf

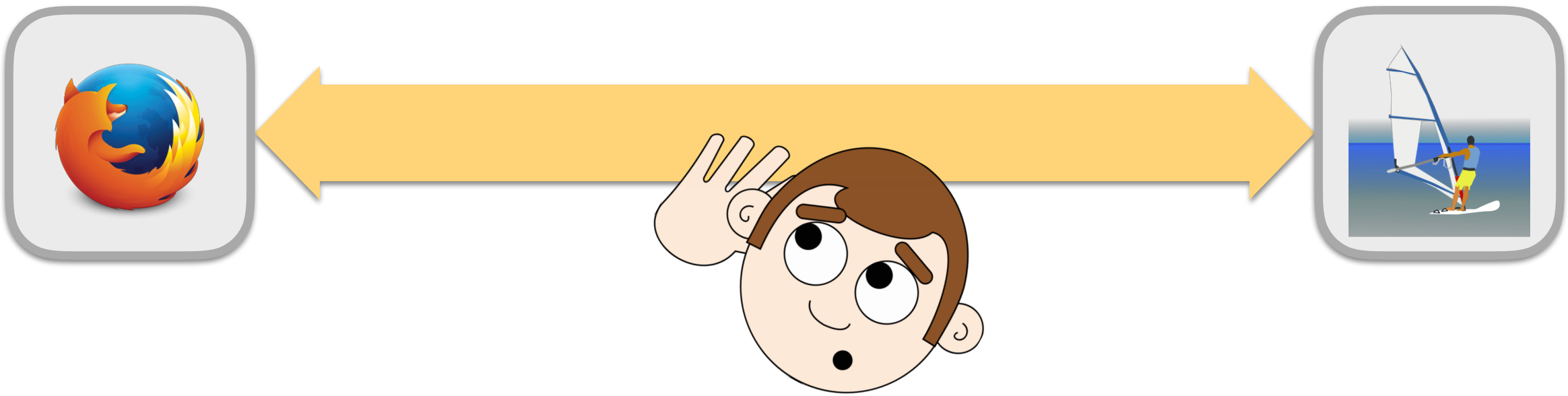
https://tls13tamarin.github.io/TLS13Tamarin/

http://prosecco.gforge.inria.fr/personal/bblanche/publications/BhargavanBlanchetKobeissiSP2017.pdf

Challenge 2: Hiding the metadata

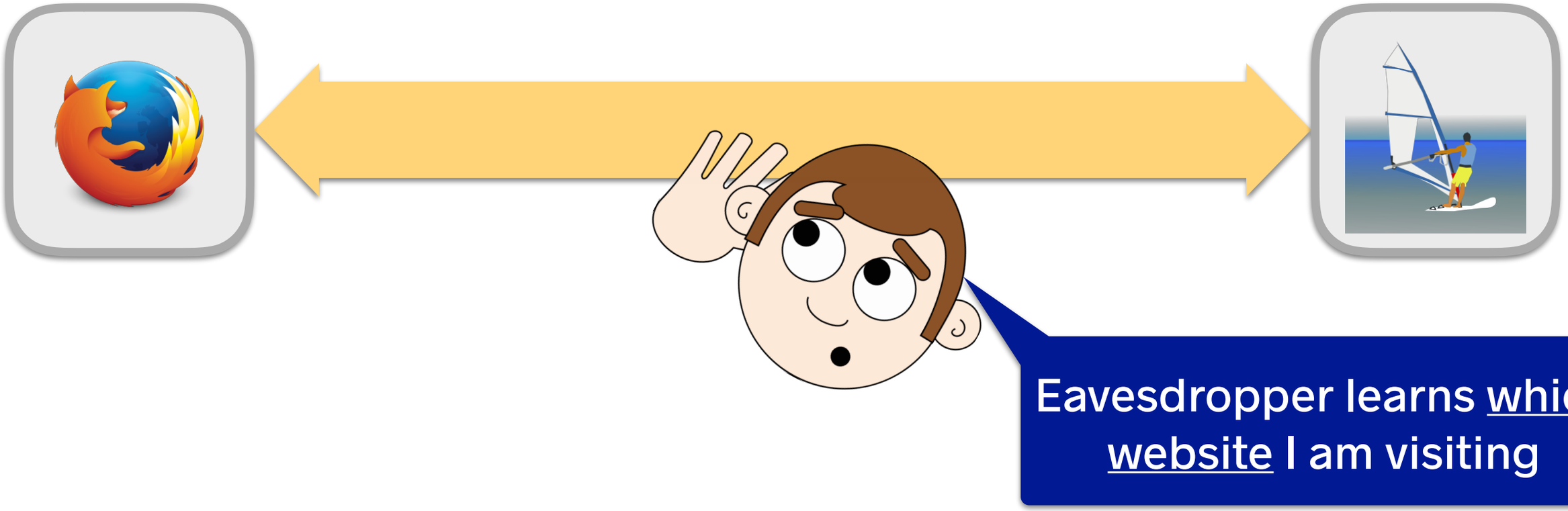
- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

Challenge 2: Hiding the metadata



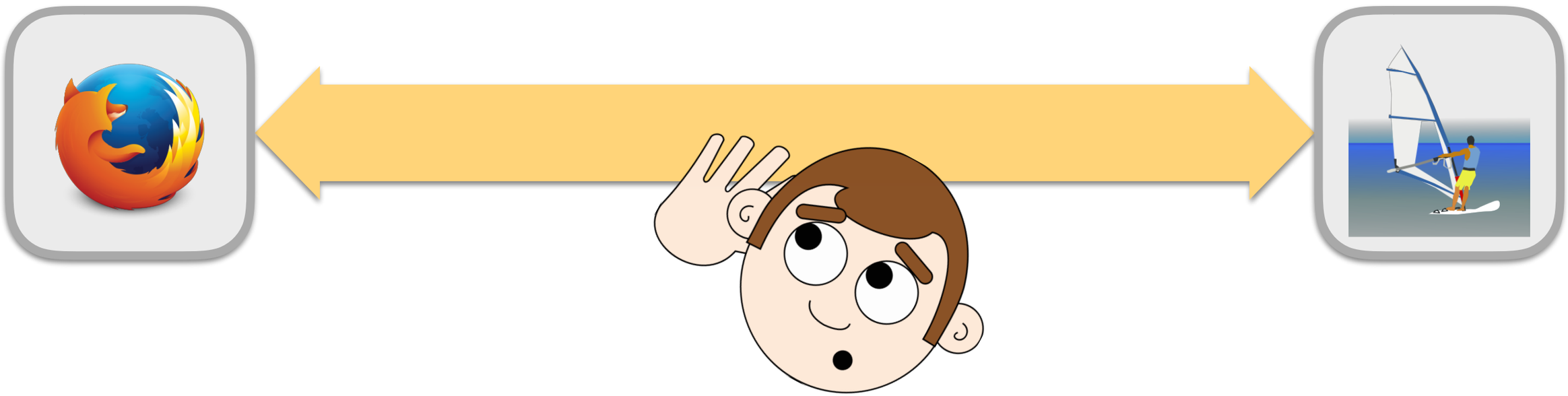
- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

Challenge 2: Hiding the metadata



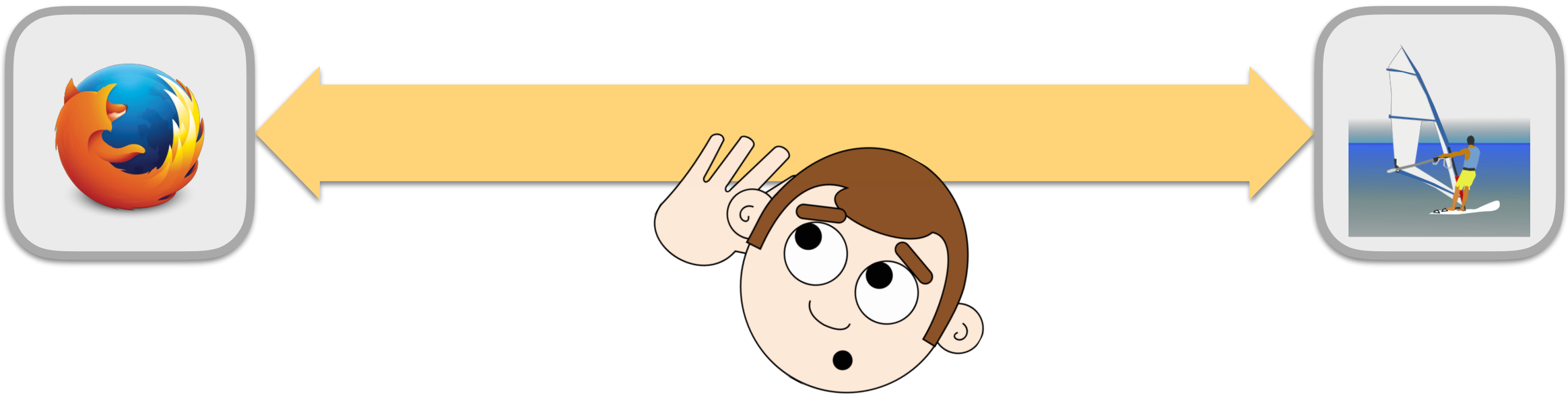
- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

Challenge 2: Hiding the metadata



- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

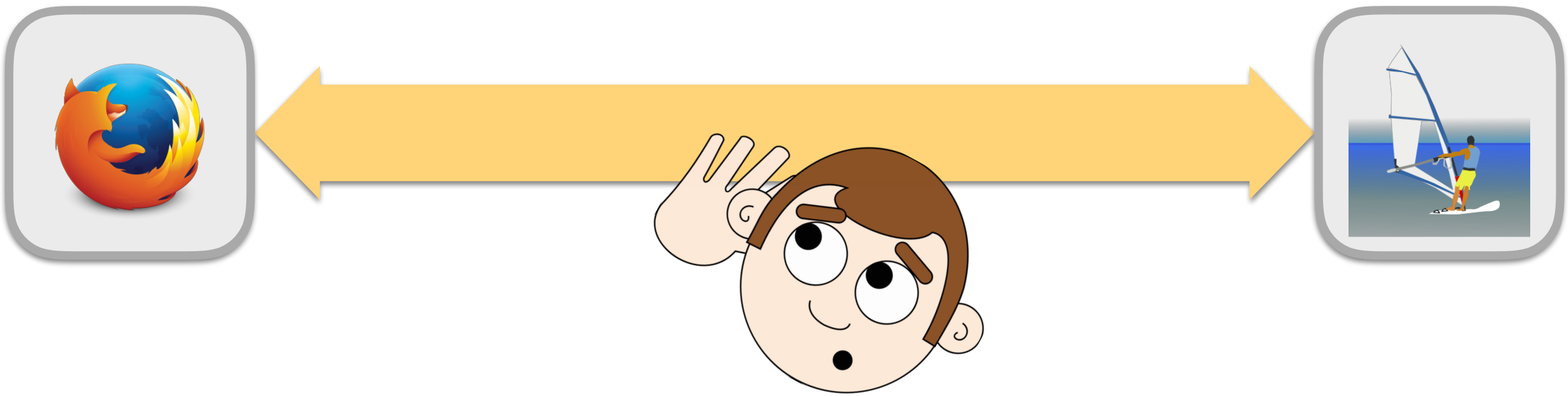
Challenge 2: Hiding the metadata



- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

Could be yours!

Challenge 2: Hiding the metadata



- TLS leaks website names and traffic patterns
- Repeated visits to cookie-enthusiasts.com reveal something about me
- This problem is a focus of my research

Challenge 3: “Post-quantum” cryptography

Challenge 3: “Post-quantum” cryptography

- Most crypto systems in use today rely on the hardness of one of these computational problems:
  - Integer factorization problem
  - Discrete logarithm problem

Challenge 3: “Post-quantum” cryptography

- Most crypto systems in use today rely on the hardness of one of these computational problems:
  - Integer factorization problem
  - Discrete logarithm problem

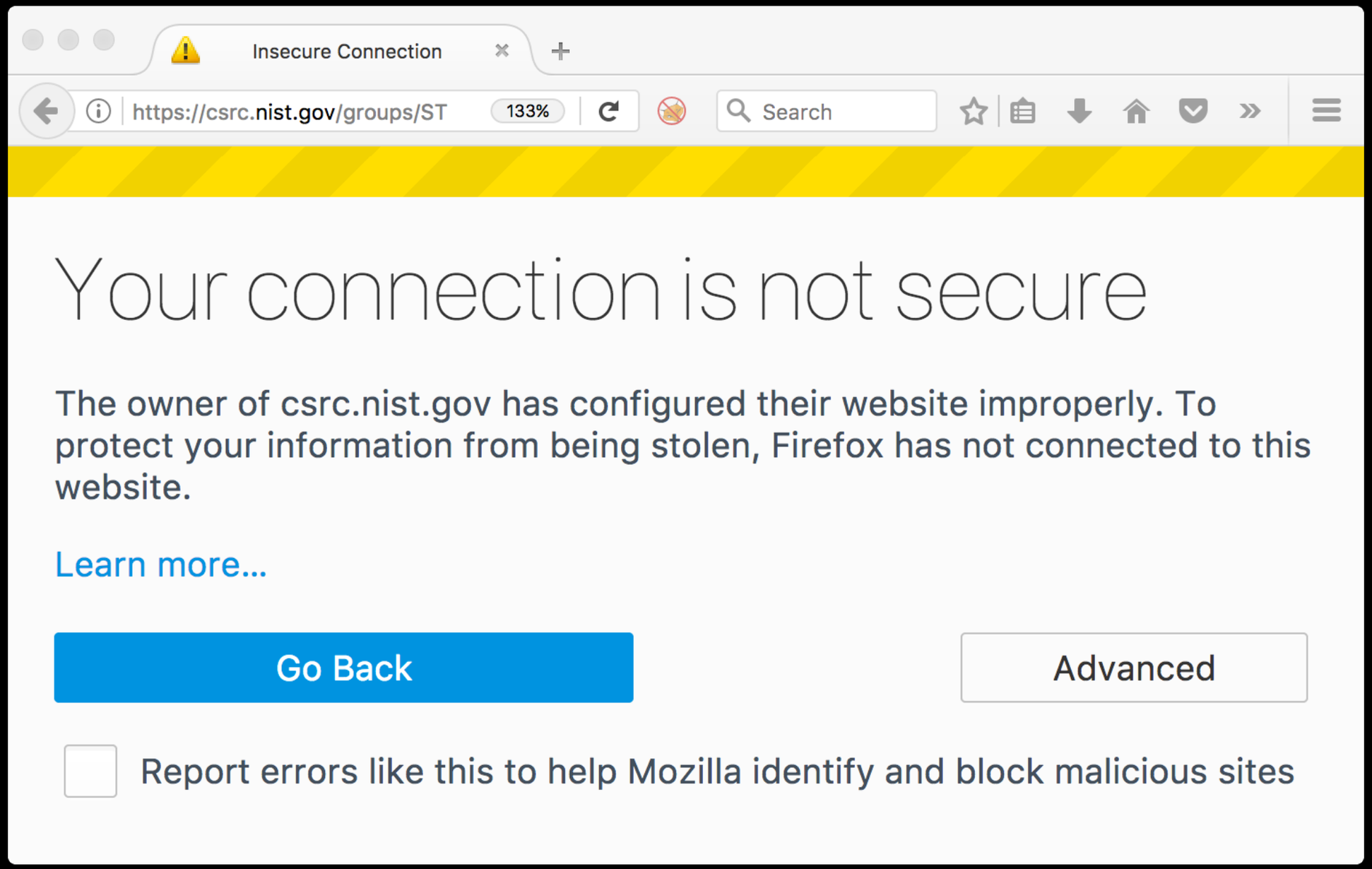
- “Quantum algorithms” solve these problems efficiently

Challenge 3: “Post-quantum” cryptography

- Most crypto systems in use today rely on the hardness of one of these computational problems:
  - Integer factorization problem
  - Discrete logarithm problem

- “Quantum algorithms” solve these problems efficiently
  - Large-scale quantum computers could break [Shor’94] much of today’s crypto

- The search for new cryptosystems is on!
  - NIST’s non-competition competition
    http://csrc.nist.gov/groups/ST/post-quantum-crypto/


Insecure Connection
https://csrc.nist.gov/groups/ST
133%
Search
Your connection is not secure
The owner of csrc.nist.gov has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.
Learn more...
Go Back
Advanced
Report errors like this to help Mozilla identify and block malicious sites

Concluding thoughts

- Cryptographic protocols are everywhere
  - Computers, phones, cars, satellites, toasters, etc.

- These protocols are devilishly hard to get right
  - Especially in systems with many moving parts (e.g., browser)

- And yet, the security of our computers, cars, planes, and satellites rely on us getting it right.
  - Lot's of room for research, experimentation, improvement.

Henry Corrigan-Gibbs
henrycg@cs.stanford.edu
henrycg.com

Want to learn more?

Books

- “The Codebreakers” by David Kahn
- “The Puzzle Palace” by James Bamford

Courses

- Free crypto course – www.crypto-class.org
- Stanford Center for Prof. Dev. – scpd.stanford.edu

Questions?

References

- BEAST: http://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art027

- Vaudenay: https://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf