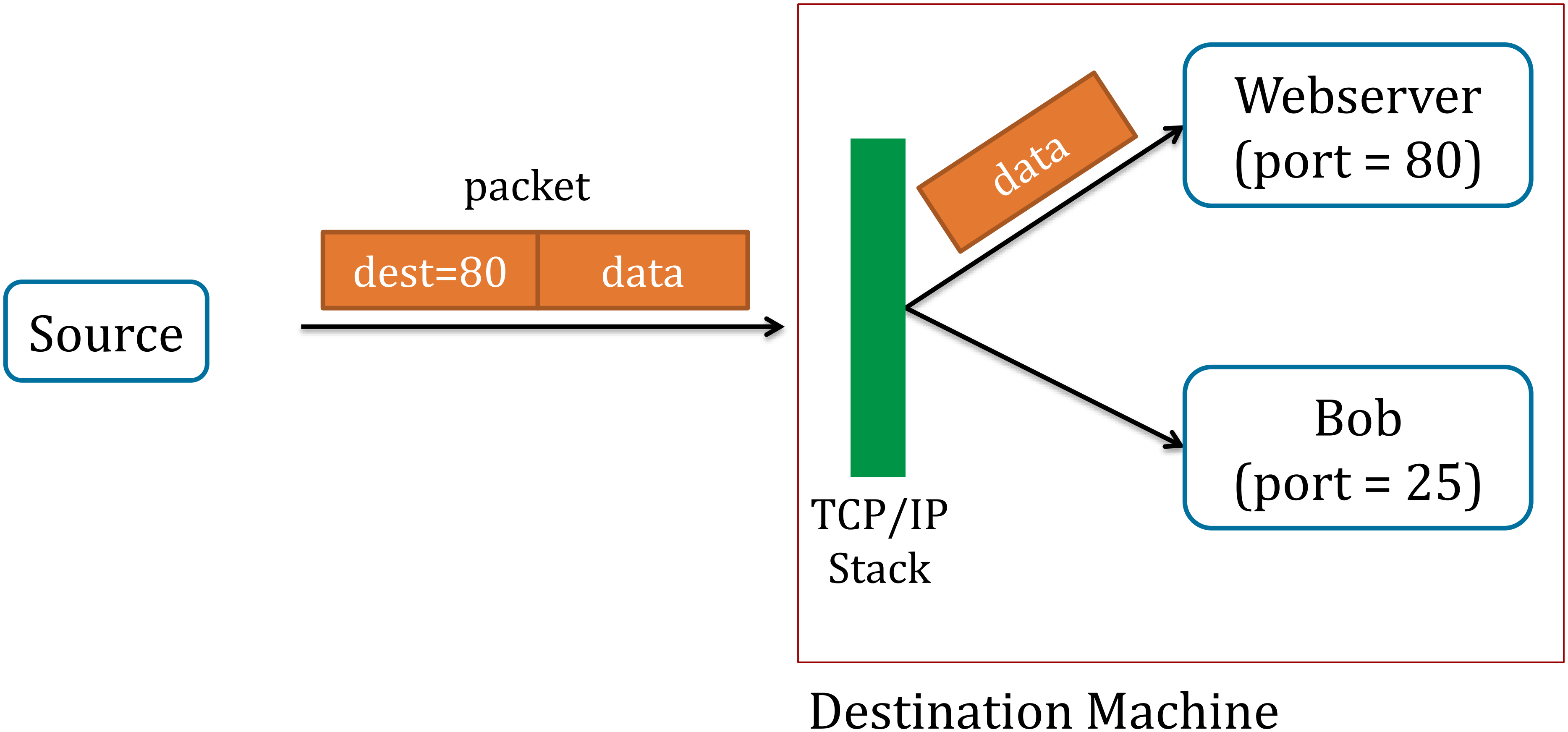
Authenticated Encryption and Cryptographic Network Protocols

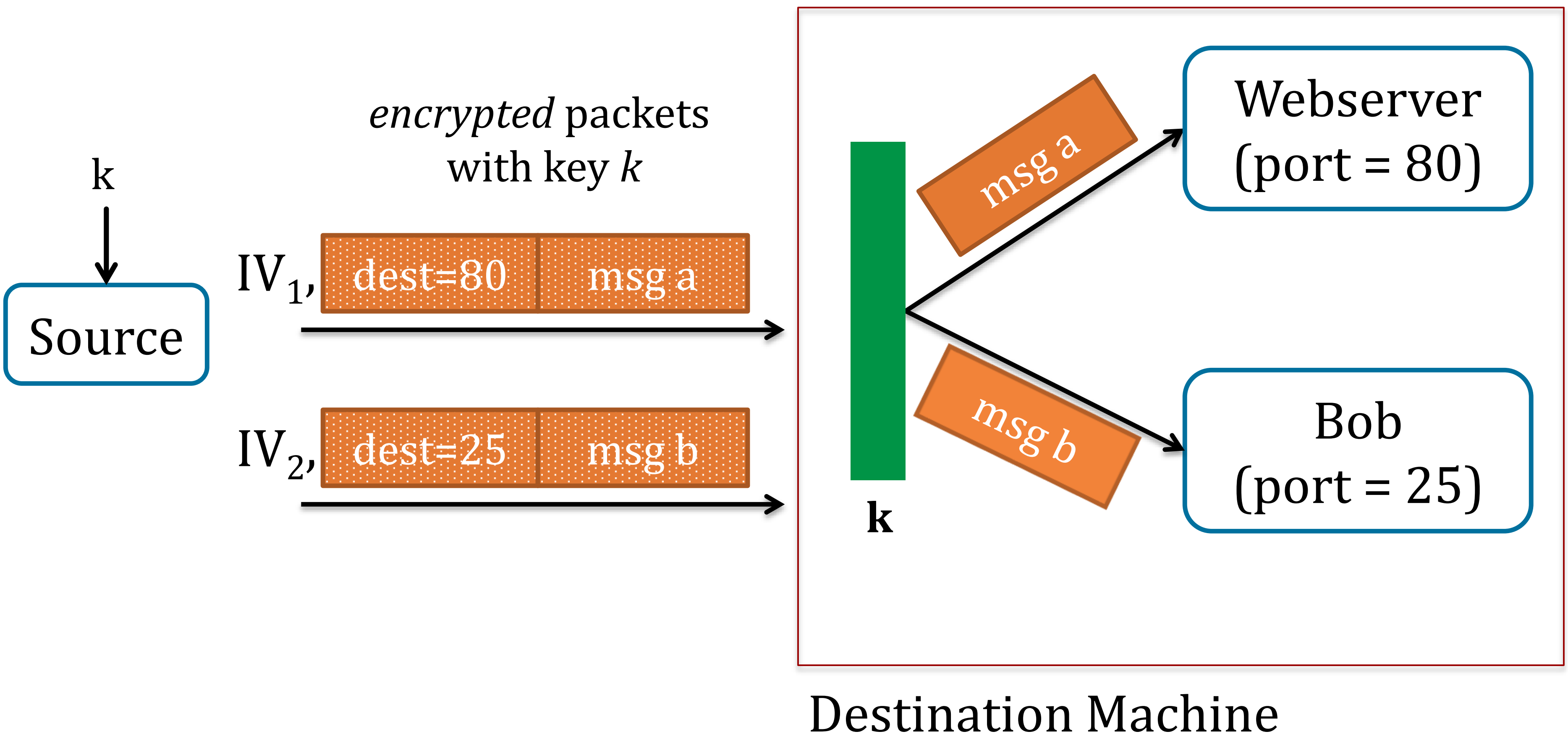
David Brumley
dbrumley@cmu.edu
Carnegie Mellon University

Some Straw Men

TCP/IP (highly abstracted)

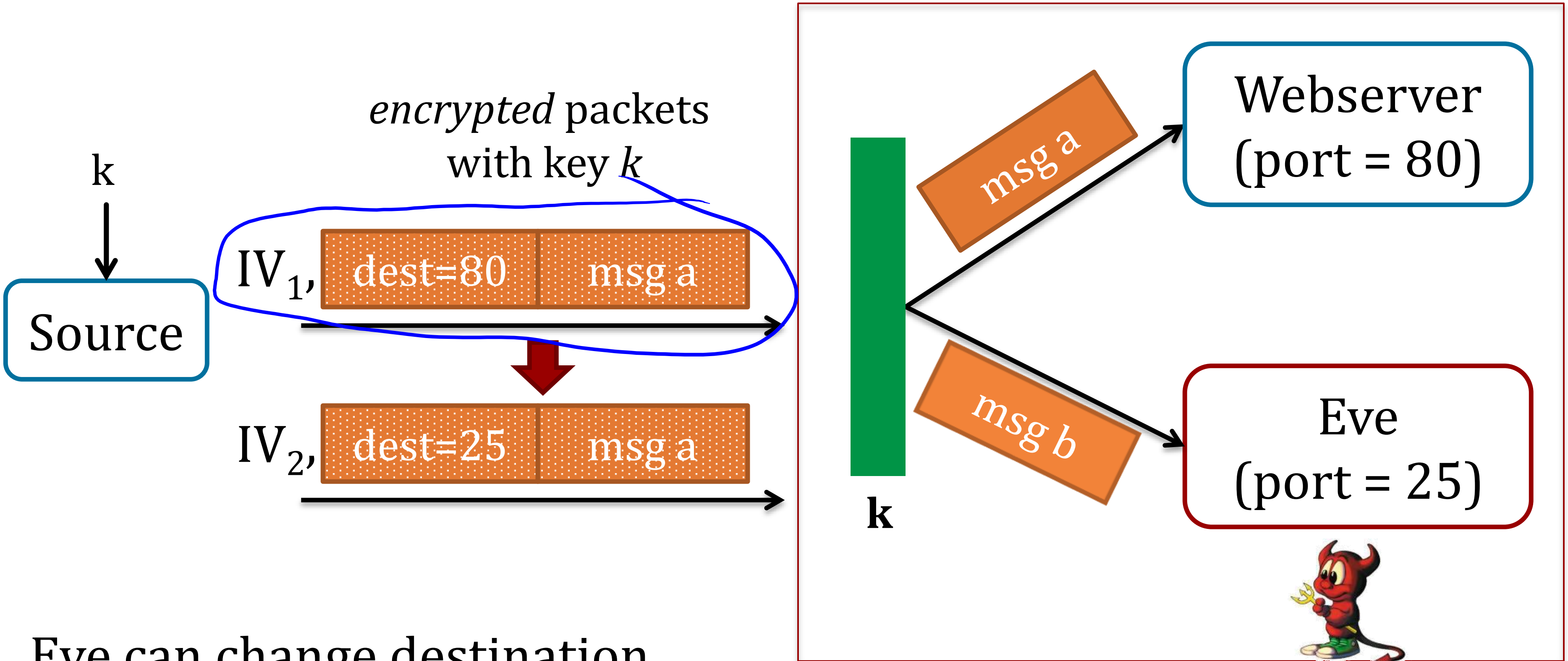


Encrypted with CBC and random IV



Example Tampering Attack

Encrypted with CBC and random IV



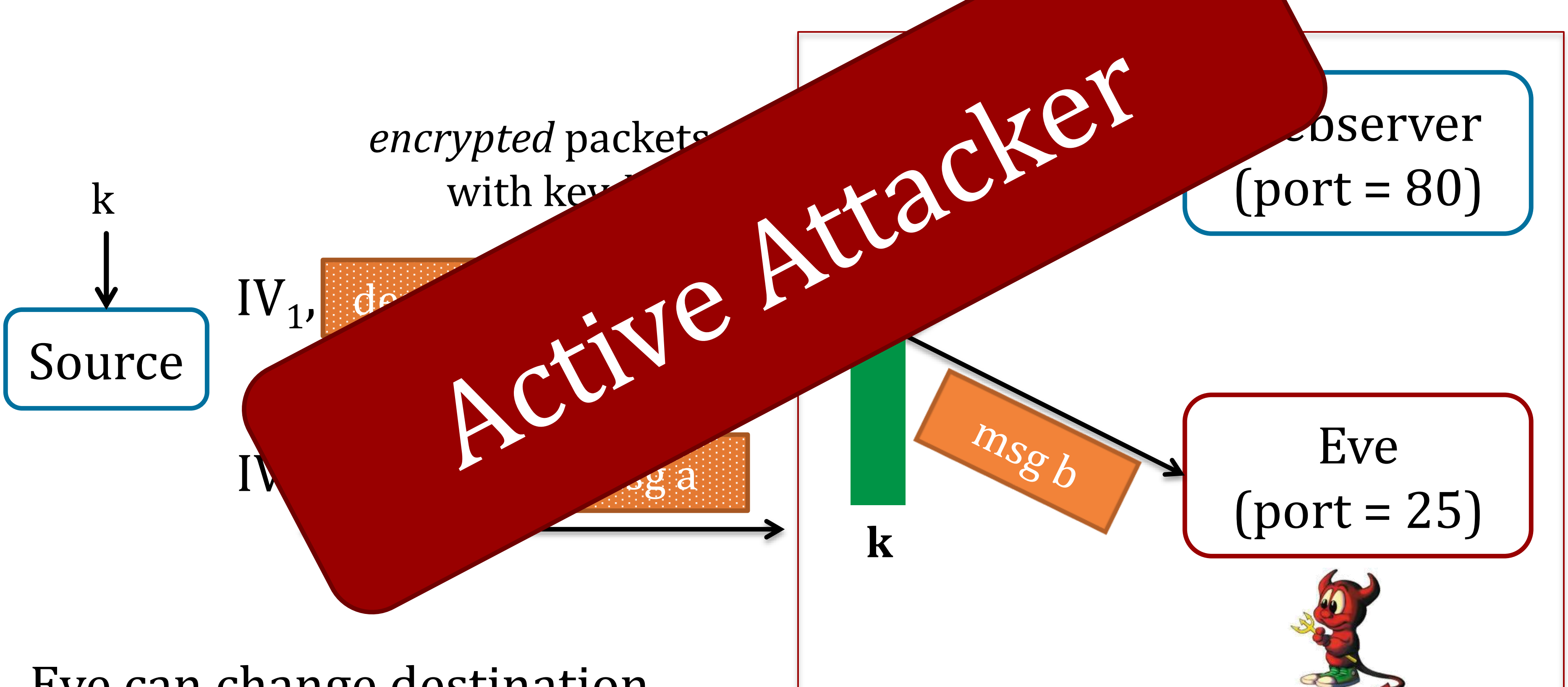
Eve can change destination
(easy with CBC and rand IV)

Destination Machine

Example Tampering Attack

Malleable

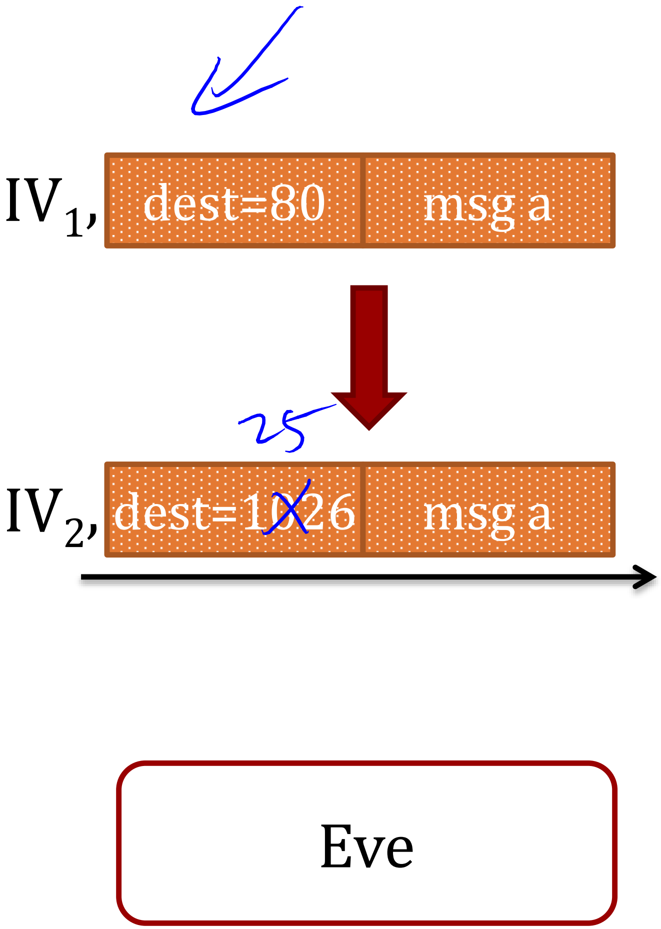
Encrypted with CBC and random



Destination Machine

Eve can change destination
(easy with CBC and rand IV)

How?



CBC encryption:

$D(k, c[0]) \oplus IV_1 = \text{“dest=80”}$

Attack:

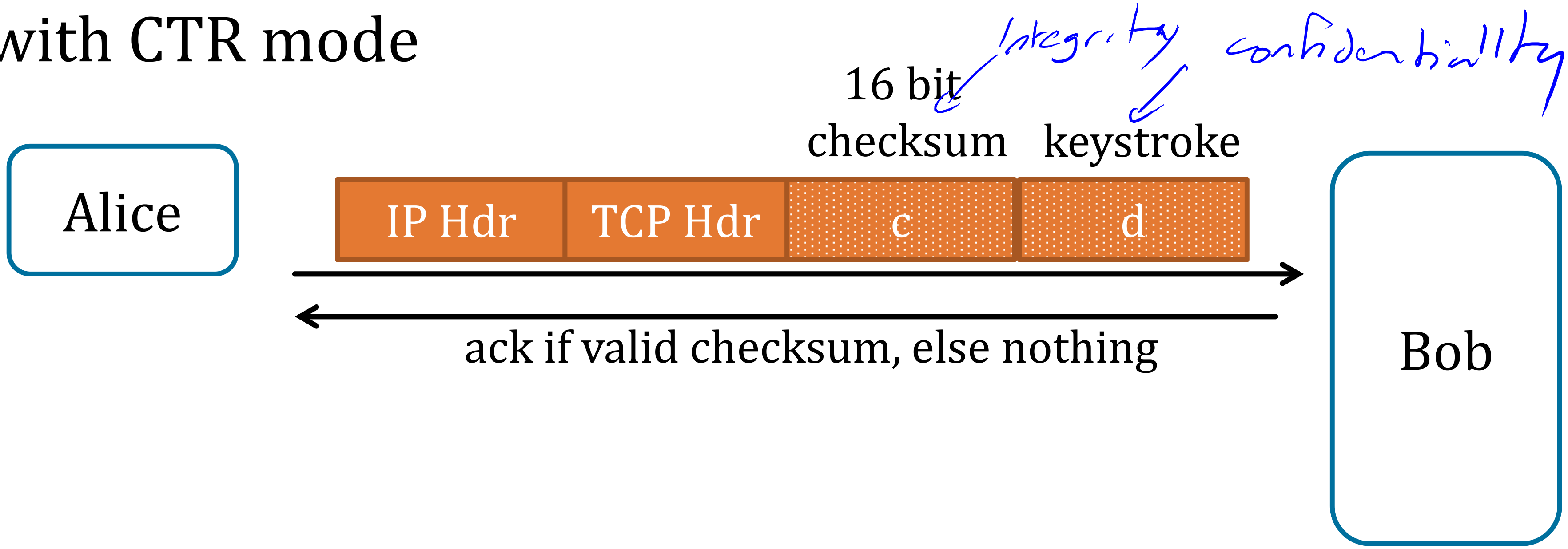
$IV_2 = IV_1 \oplus \underbrace{000...80 \oplus 000...25}$

xor out “80” and
xor in “1026”
25

An Attack Using Only Network Access

Example:

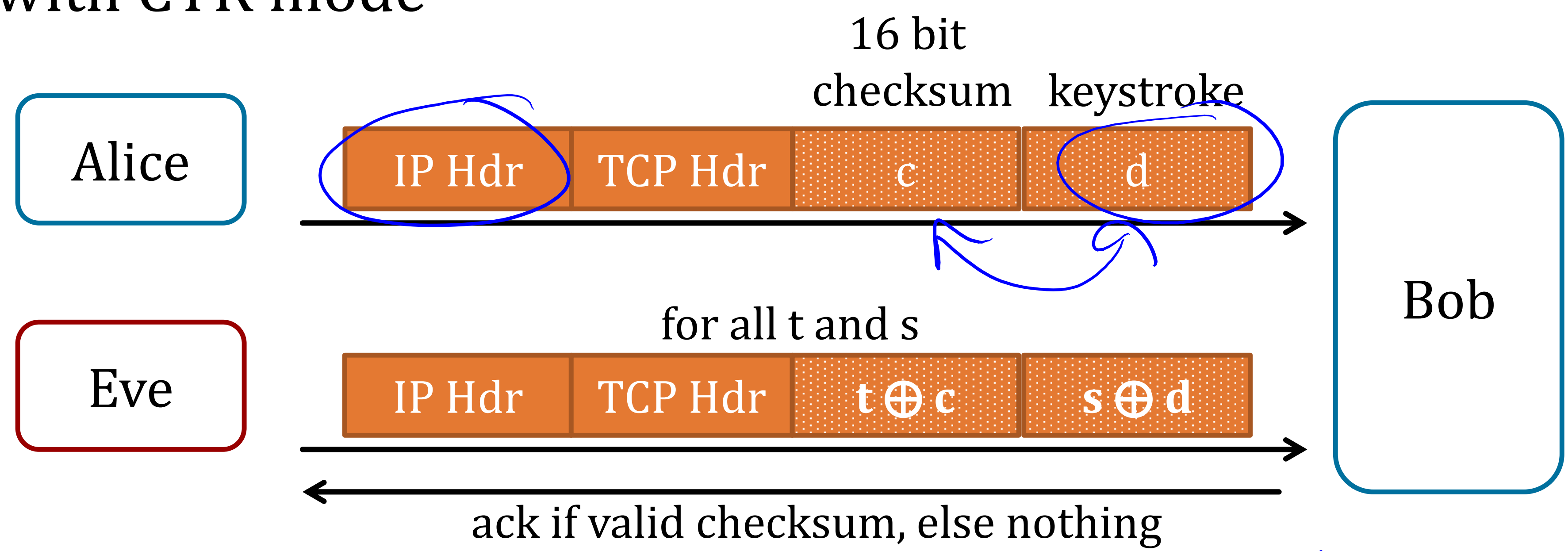
Remote terminal app where each keystroke encrypted with CTR mode



An Attack Using Only Network Access

Example:

Remote terminal app where each keystroke encrypted with CTR mode



$\{\text{checksum}(\text{hdr}, d) = t \oplus \text{checksum}(\text{hdr}, d \oplus s)\} \Rightarrow$

Even can find d for many realistic checksums*

* potentially not for TCP checksum

The Story So Far

Confidentiality: semantic security against a CPA attack

– Examples: Using CBC with a PRP, AES

Integrity: security against existential forgery

– Examples: CBC-MAC, NMAC, PMAC, HMAC

Now: security against *tampering*

– Integrity + Confidentiality!

The lesson

CPA security *cannot* guarantee secrecy under *active* attacks.

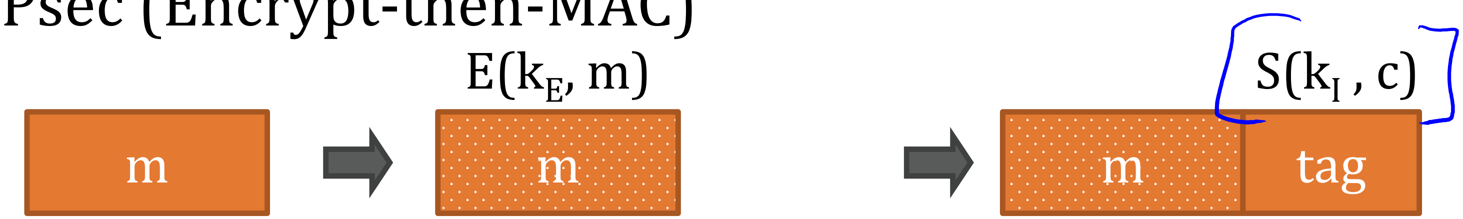
Integrity Only	→	✓ Secure MAC
Integrity + Secrecy	→	✗ Secure MAC + Secure Cipher
Integrity + Secrecy	→	✓ *Authenticated Encryption* 2003

Motivating Question: Which is Best?

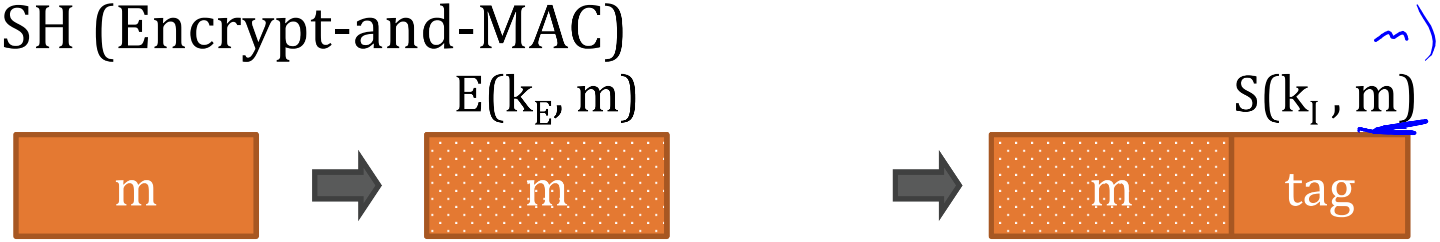
Encryption Key = K_E; MAC key = k_I

Option 1: SSL (MAC-then-encrypt)

Option 2: IPsec (Encrypt-then-MAC)



Option 3: SSH (Encrypt-and-MAC)



Authenticated Encryption

An *authenticated encryption* system (E,D) is a cipher where

As usual: $E: K \times M \times N \longrightarrow C$

but $D: K \times C \times N \longrightarrow M \cup \{\perp\}$

reject ciphertext as invalid

Security: the system must provide

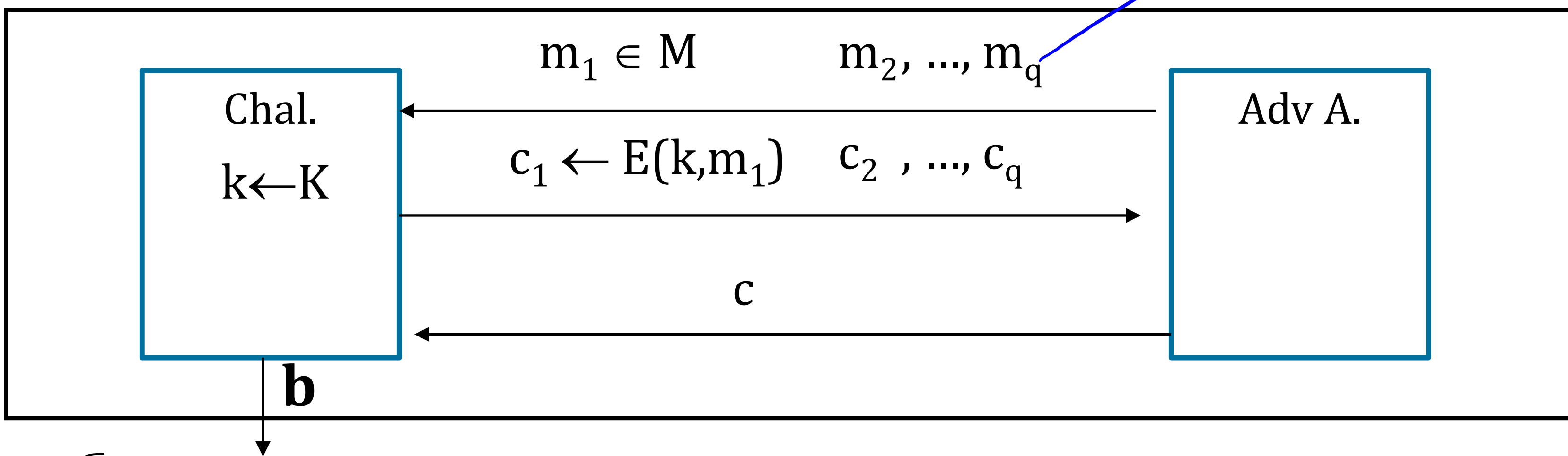
– Semantic security under CPA attack, ***and***

– *ciphertext integrity*. The attacker cannot create a new ciphertext that decrypts properly.

Ciphertext Integrity

(t, ε, q)
"computational"

For b ={0,1}, define EXP(0) and EXP(1) as:



$$\begin{cases} \mathbf{b}=1 & \text{if } D(k,c) \neq \perp \text{ and } c \notin \{c_1, \ldots, c_q\} \\ \mathbf{b}=0 & \text{otherwise} \end{cases}$$

$t =$

Def: (E,D) has *ciphertext integrity* iff for all "efficient" A:

$$\mathrm{Adv}_{\mathrm{CI}}[A,I] = \Pr[\text{Chal. outputs } 1] < \varepsilon$$

Authenticated Encryption

Def: cipher (E,D) provides ***authenticated encryption (AE)*** if it is

(1) semantically secure under CPA, and

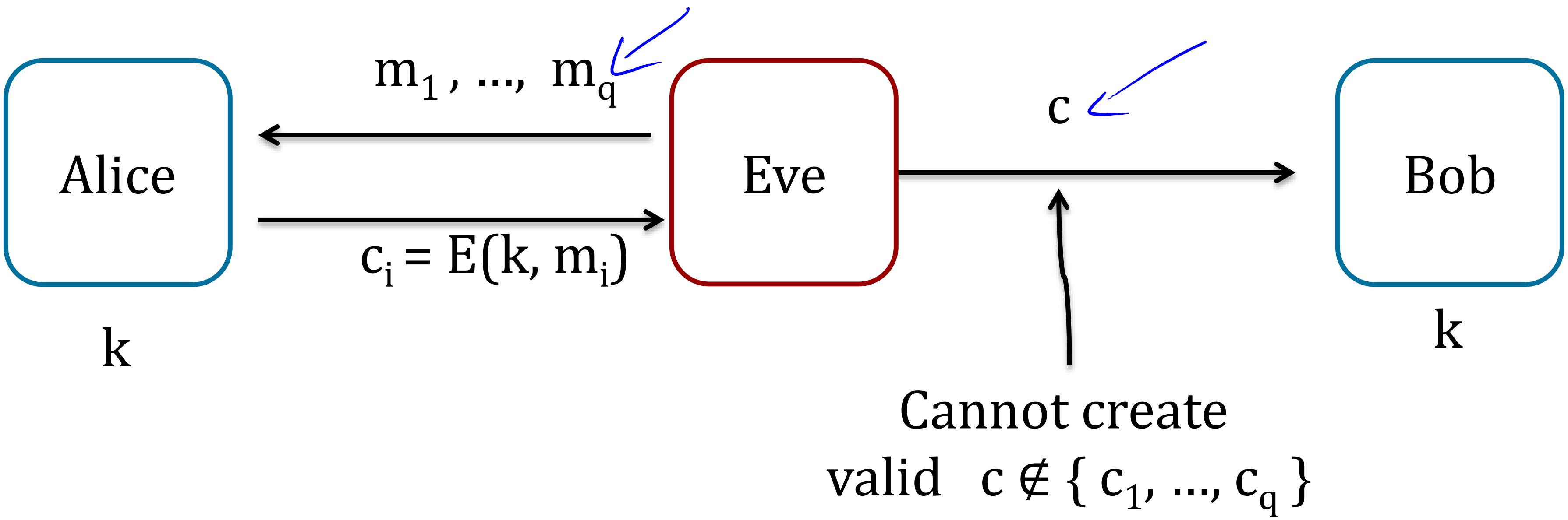
(2) has ciphertext integrity

Counter-example: CBC with rand. IV does not provide AE

– D(k, ·) never outputs ⊥, hence adv. always wins ciphertext integrity game

Implication 1: Authenticity

Attacker cannot fool Bob into thinking a message was sent from Alice



$\Rightarrow$ if $D(k,c) \neq \perp$ Bob *guaranteed* message is from someone who knows k (but could be a replay)

Implication 2

CPA

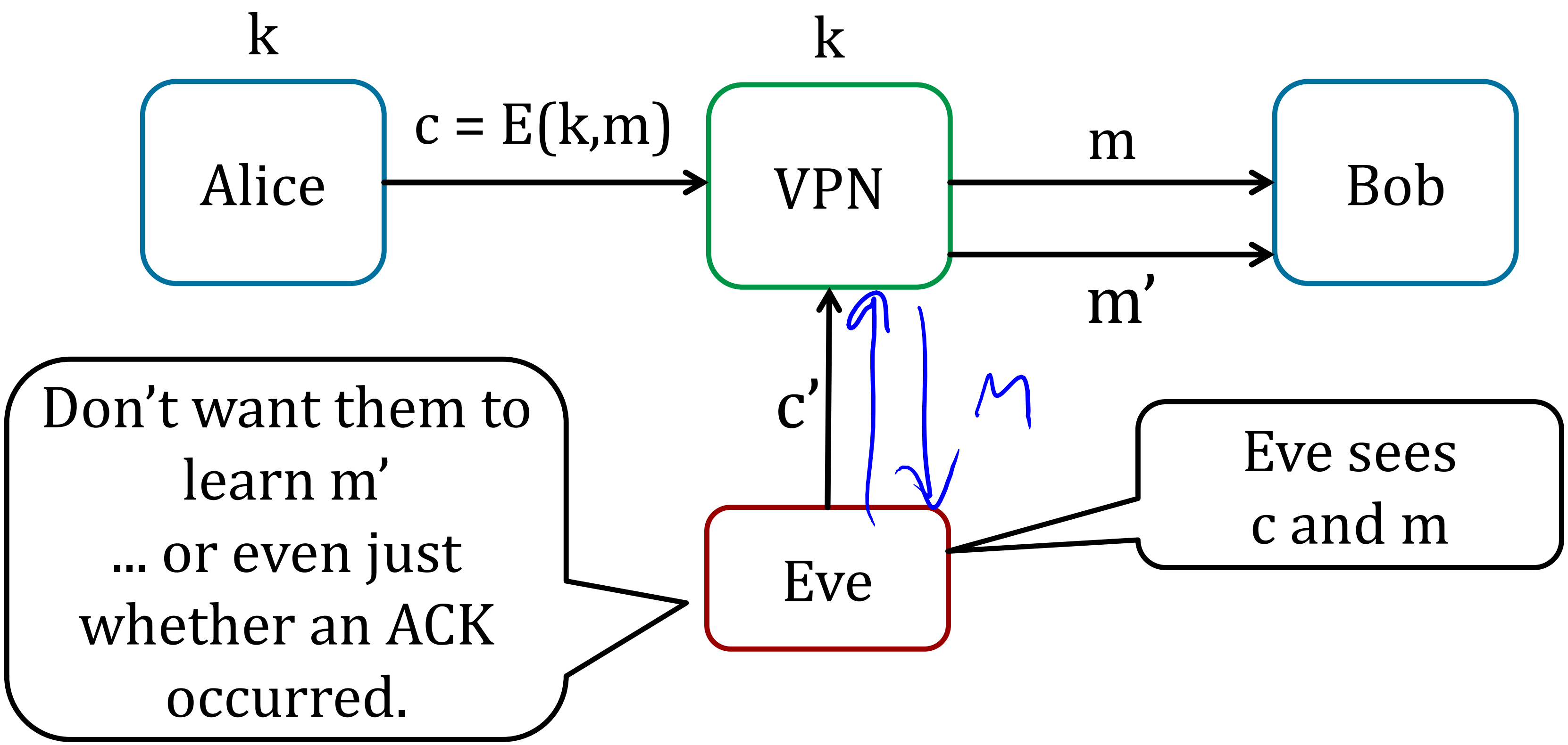
Authenticated encryption $\Rightarrow$

Security against ***chosen ciphertext attack***

Chosen Ciphertext Attacks

Chosen Ciphertext Attacks

<u>Def</u>: A CCA adversary has the capability to get ciphertexts of their choosing decrypted.



The Lunchtime CCA Attack

It's
Lunchtime!

Alice's Computer

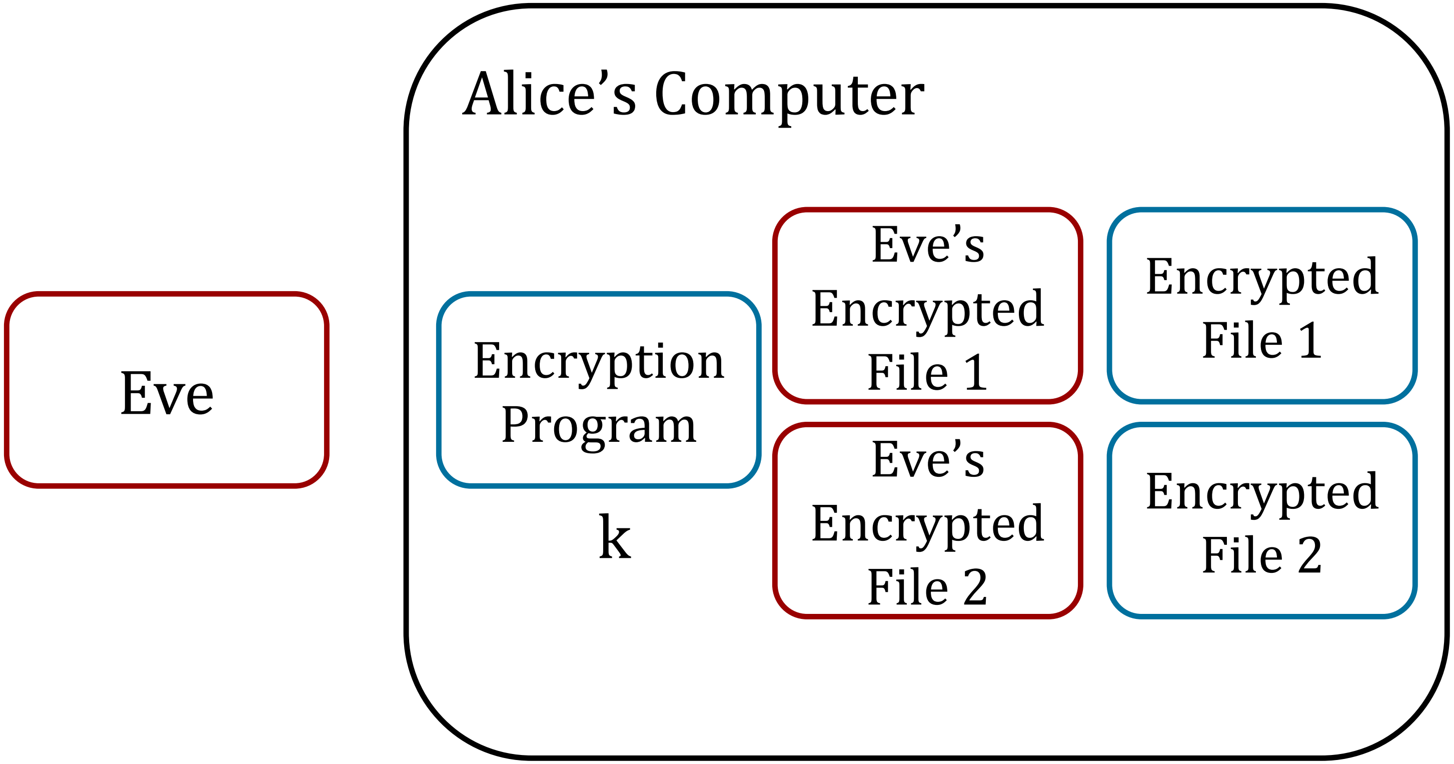
Encryption
Program

k

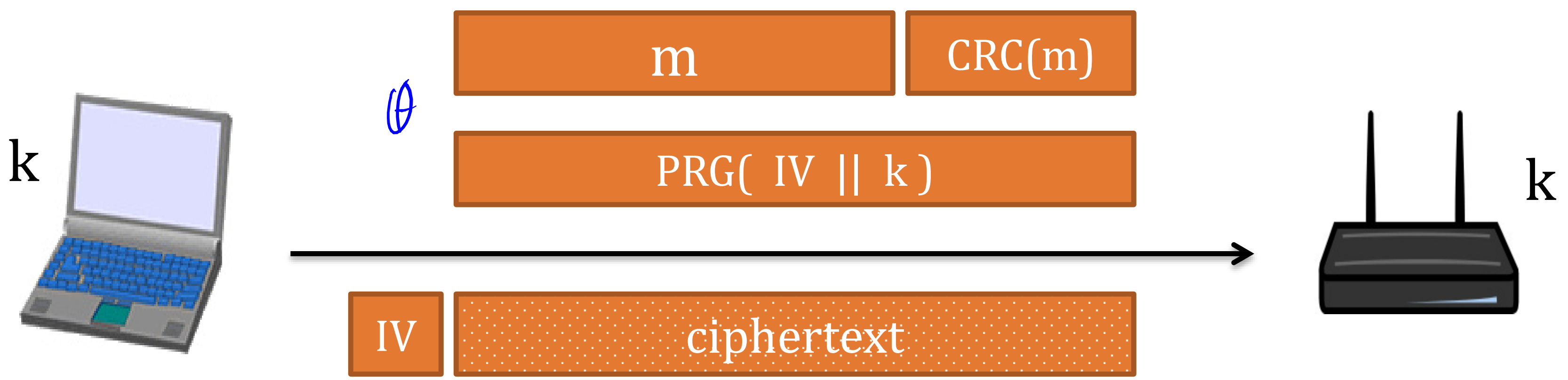
Encrypted
File 1

Encrypted
File 2

The Lunchtime CCA Attack



802.11b WEP: how not to do it



Active attacks

Fact: CRC is linear, i.e.

$\forall m,p$: $\mathbf{CRC(\ m \oplus p) = CRC(m) \oplus F(p)}$

WEP ciphertext: IV | dest-port = 80 data | CRC

$\oplus$

attacker: 000…00……… **XX**…..0000 | **F(XX)**

XX = 25$\oplus$80

IV | dest-port = 25 data | CRC'

Upon decryption CRC is valid,
but ciphertext is changed !!

Chosen Ciphertext Security

Adversaries Power: both CPA and CCA

- Can obtain the encryption of arbitrary messages
- Can decrypt ciphertexts of his choice

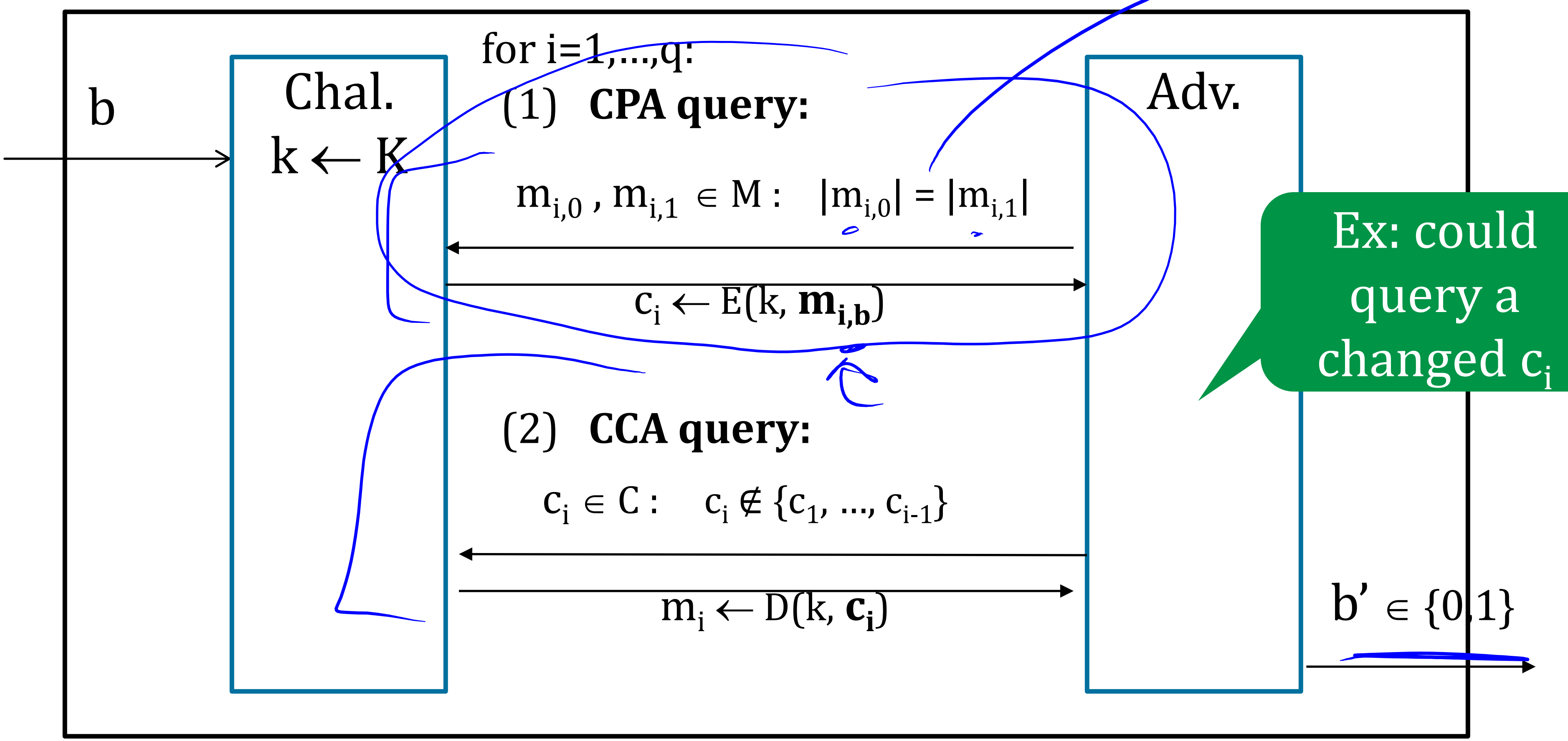
Adversaries Goal: break semantic security

CCA Game Definition

Let ENC = (E,D) over (K,M,C).

For b = {0,1}, define EXP(0) and EXP(1)

CPA



CCA Game Definition

Let ENC = (E,D) over (K,M,C).

For b = {0,1}, define EXP(0) and EXP(1)

b

for i=1,...,q:

(1) **CPA query:**

$m_{i,0}, m_{i,1} \in M$: $|m_{i,0}| = |m_{i,1}|$

ENC = (E,D) is CCA secure iff

Adv[A,ENC] = |Pr[Exp(0) = 1] – Pr[Exp(1) = 1]| < ε

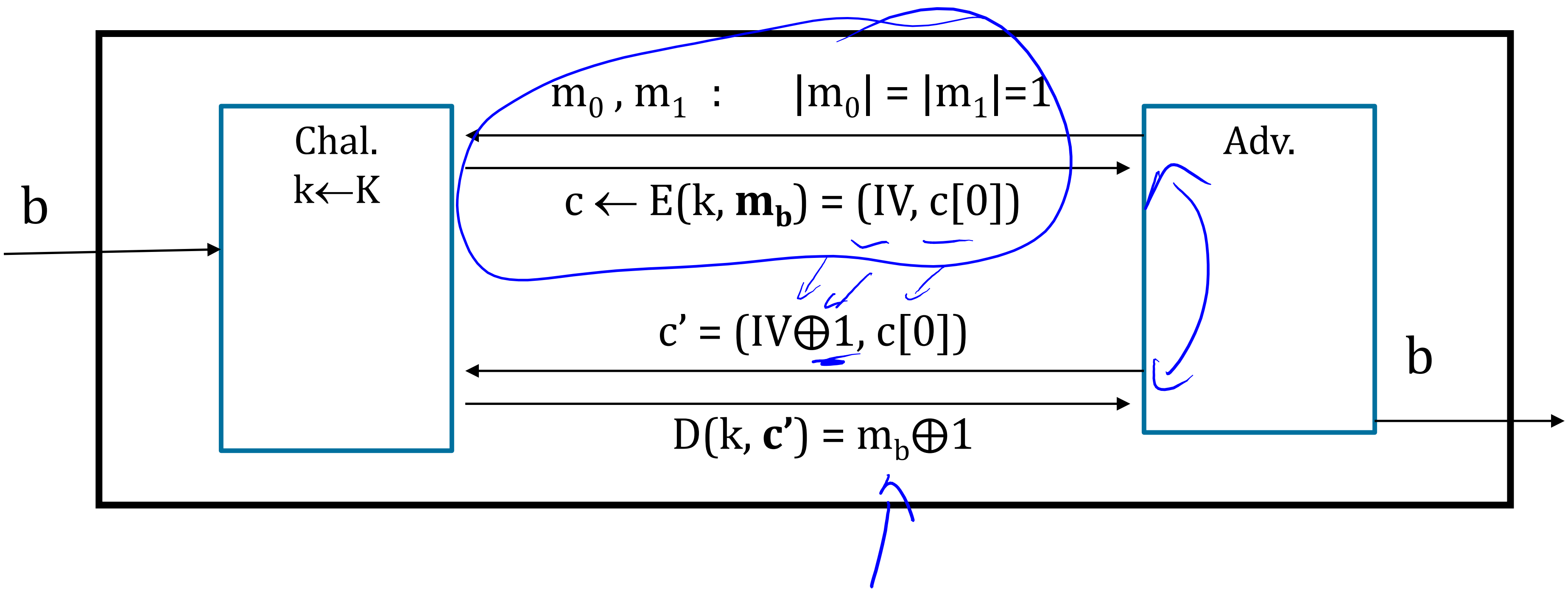
(2) **CCA query:**

$c_i \in C$: $c_i \notin \{c_1, \ldots, c_{i-1}\}$

$m_i \leftarrow D(k, \mathbf{c_i})$

$b' \in \{0,1\}$

Example: CBC is not CCA Secure



<u>Thm</u>: Let (E,D) be a cipher that provides AE.
Then (E,D) is CCA secure !

In particular, for any q-query eff. A there exist eff. B_1, B_2 s.t.

$$\mathrm{Adv}_{\mathrm{CCA}}[A,E] \le 2q \cdot \mathrm{Adv}_{\mathrm{CI}}[B_1,E] + \mathrm{Adv}_{\mathrm{CPA}}[B_2,E]$$

AE implies CCA security!

So What?

Authenticated encryption assures security against:

- A passive adversary (CPA security)
- An active adversary that can even decrypt some ciphertexts (CCA security)

Limitations:

- Does not protect against replay
- Assumes no other information other than message/ciphertext pairs can be learned.
  - Timing attacks out of scope
  - Power attacks out of scope
  - …

AE Constructions

Cipher + MAC = security

History

Android

Pre 2000: Crypto API's provide *separate* MAC and encrypt primitives

- Example: Microsoft Cryptographic Application Programming Interface (MS-CAPI) provided HMAC and CBC + IV
- Every project had to combine primitives in their own way

2000: Authenticated Encryption

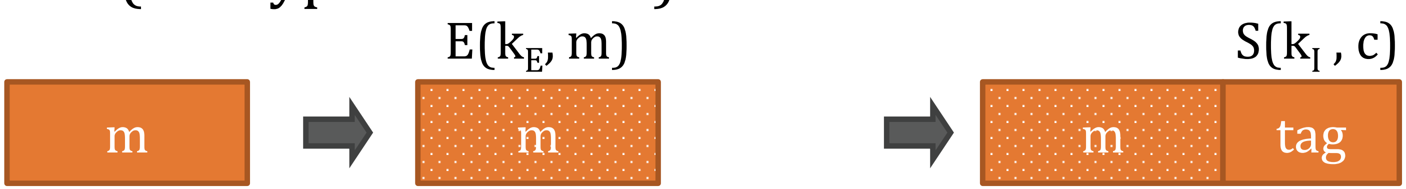
- Bellare and Namprempre in Crypto, 2000
- Katz and Yung in FSE, 2000

Motivating Question: Which is Best?

Encryption Key = K_E; MAC key = k_I

Option 1: SSL (MAC-then-encrypt)

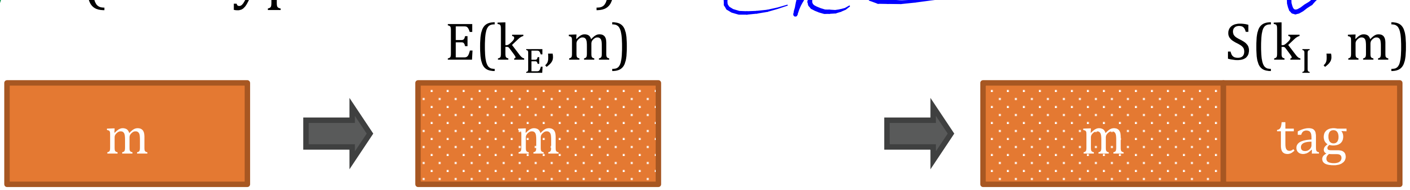
Option 2: IPsec (Encrypt-then-MAC)



Always Correct

SH (Encrypt-and-MAC)

CRC



Theorems

Let (E,D) by a CPA secure cipher and (S,V) a MAC secure against existential forgery. Then:

1. Encrypt-then-MAC *always* provides authenticated encryption
2. MAC-then-encrypt *may* be insecure against CCA attacks
   – however, when (E,D) is rand-CTR mode or rand-CBC, MAC-then-encrypt provides authenticated encryption

Standards

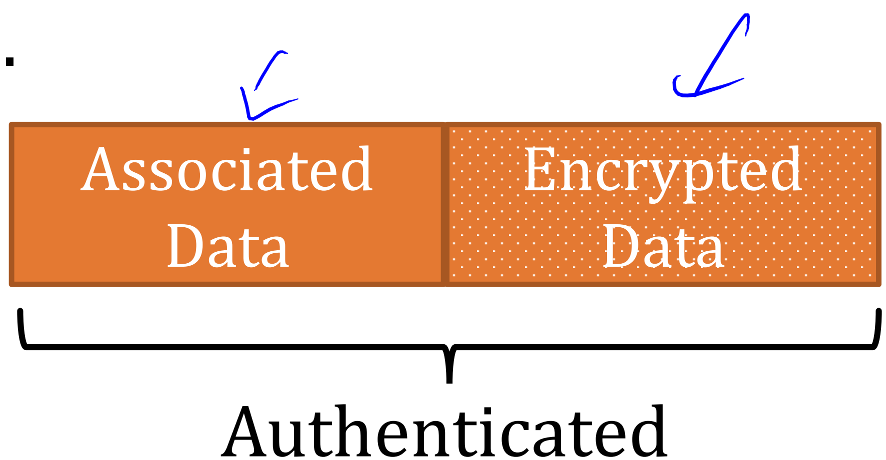
GCM: CTR mode encryption then CW-MAC

CCM: CBC-MAC then CTR mode (802.11i)

EAX: CTR mode encryption then CMAC

All are nonce-based.

All support *Authenticated Encryption with Associated Data (AEAD)*.



An example API (OpenSSL)

int **AES_GCM_Init**(AES_GCM_CTX *ain,
    unsigned char ***nonce**,   unsigned long noncelen,
    unsigned char ***key**,   unsigned int klen)

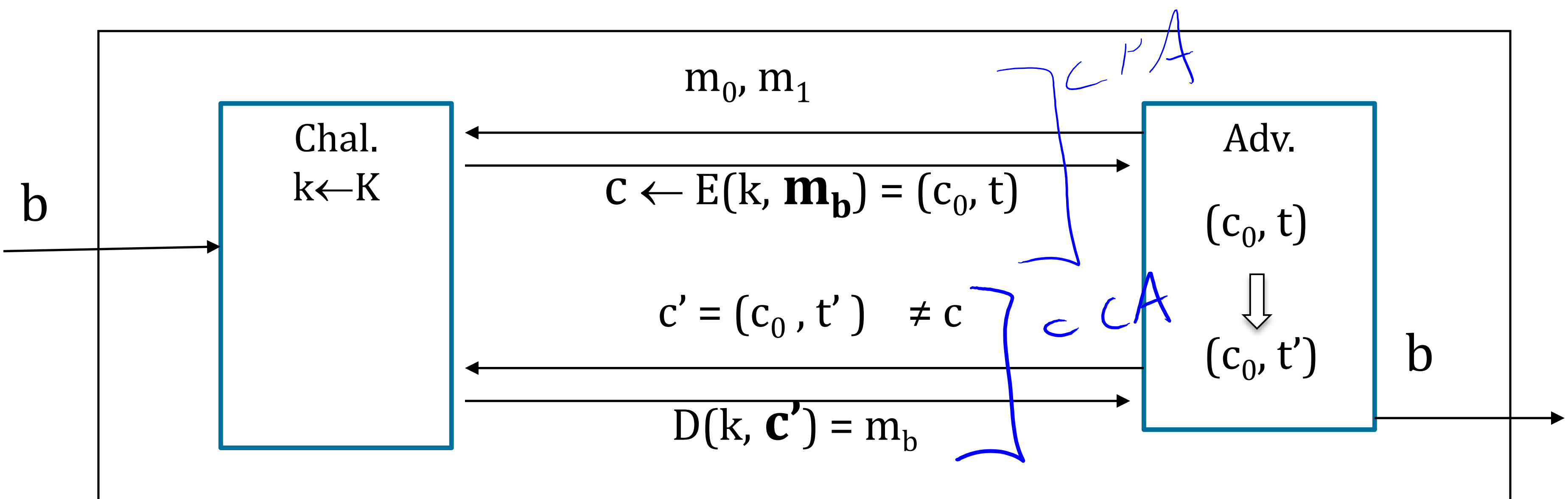
int **AES_GCM_EncryptUpdate**(AES_GCM_CTX *a,
    unsigned char ***aad**,   unsigned long aadlen,
    unsigned char ***data**,   unsigned long datalen,
    unsigned char ***out**,   unsigned long *outlen)

MAC Security -- an explanation

Recall: MAC security required an attacker given (m , t) couldn't find a different t' such that (m,t') is a valid MAC

Why? Suppose not: $(m , t) \longrightarrow (m , t')$

Then Encrypt-then-MAC would not have Ciphertext Integrity !!



Performance

From Crypto++ 5.6.0 [Wei Dai]

AE Cipher	Code Size	Speed (MB/sec)	Raw Cipher	Raw Speed
AES/GCM	Large	108	AES/CTR	139
AES/CCM	smaller	61	AES/CBC	109
AES/EAX	smaller	61	AES/CMAC	109
AES/OCB*	small	129	HMAC/SHA1	147

* OCB mode may have patent issues. Speed extrapolated from Ted Kravitz's results.

Summary

Encrypt-then-MAC

- Provides integrity of CT
- Plaintext integrity
- If cipher is malleable, we detect invalid CT
- MAC provides no information about PT since it's over the encryption

MAC-then-Encrypt

- No integrity of CT
- Plaintext integrity
- If cipher is malleable, can change message w/o detection
- MAC provides no information on PT since encrypted

Encrypt-and-MAC

- No integrity on CT
- Integrity of PT can be verified
- If cipher is malleable, contents of CT can be altered; should detect at PT level
- May reveal info about PT in the MAC (e.g., MAC of same messages are the same)

PT | M

M("David") → Da_

Wrapup

- Authenticated Encryption
  - Chosen Ciphertext Attack (CCA) and CCA-secure ciphers
  - AE game = CCA + CPA secure

2000

- Encrypt-then-MAC always right
  - Don’t roll your own


Questions?

END

Case Study: TLS

Certificates bind a public key to a user

Certificate Authority (CA) binds certificate to person

CA Signature

Certificate parameters

Alice

VeriSign®

Alice Sends:

User ID || public key || …

Alice

Alice Generates and Gives:

User ID || public key || …

CA Computes:

D=H(User ID || public key || …)

Sig = Sign(*D*, CA private key)

Gives Alice *Sig*

Alice

Certificate Authority (CA)

Alice Generates and Gives:

User ID || public key || …

CA Computes:

D=H(User ID || public key || …)

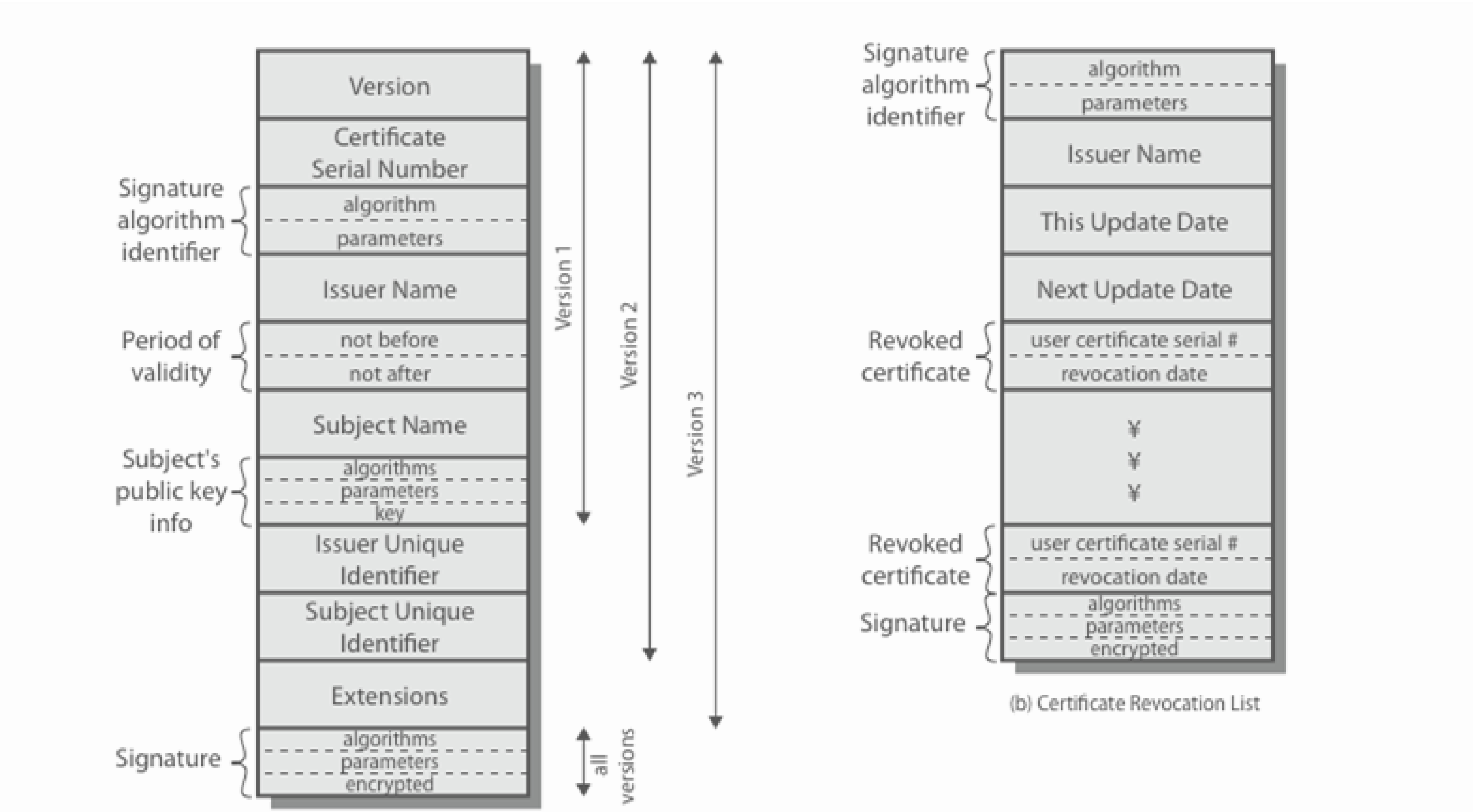
Sig = Sign(*D*, Serial, CA private key)

Gives Alice <*Sig, Serial*>

Alice's Certificate

[User ID || public key || …] || CA Name || Serial || Sig || <add. params>

X.509 Certificates



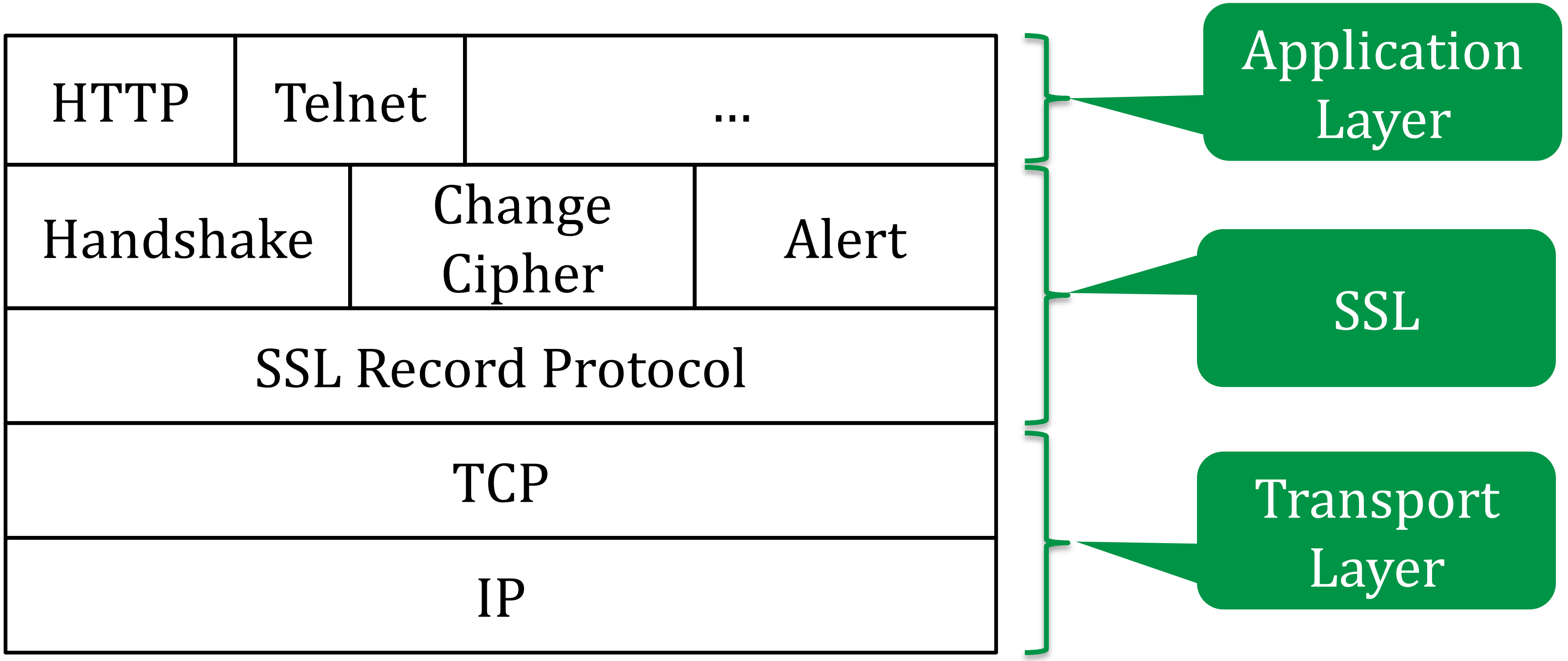
(b) Certificate Revocation List

TLS and SSL

- Transport Layer Security (TLS)
  - Secure socket layer (SSL) predecessor
  - originally developed by Netscape
  - version 3 designed with public input
  - RFC 2246

- Uses TCP to provide a reliable end-to-end service

Protocol Stack



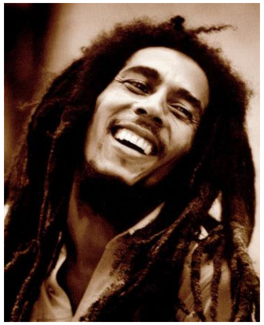
Session Establishment


Alice

1. ClientHello

2. ServerHello

3. ClientKeyExchange

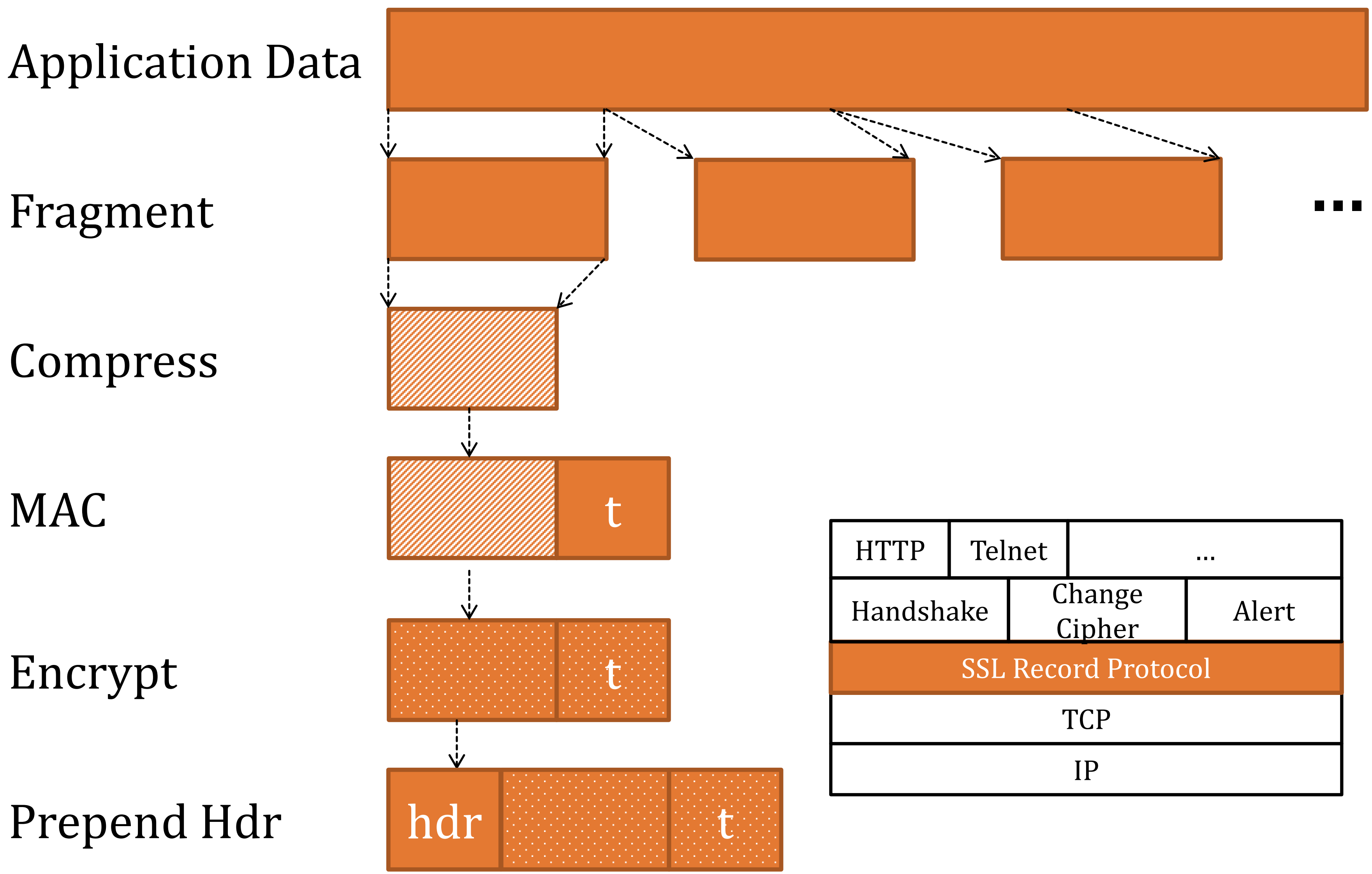

Bob.com

supported MAC's and ciphers

Encrypt with symmetric cipher using shared secret

HTTP	Telnet	...	
Handshake	Change Cipher	Alert	
SSL Record Protocol			
TCP			
IP			

Protocol Record

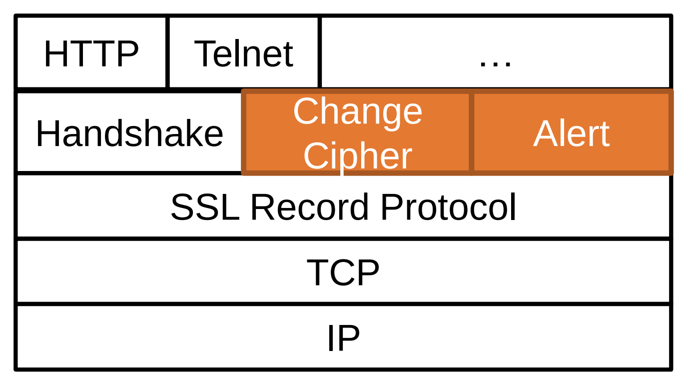


Other Fields

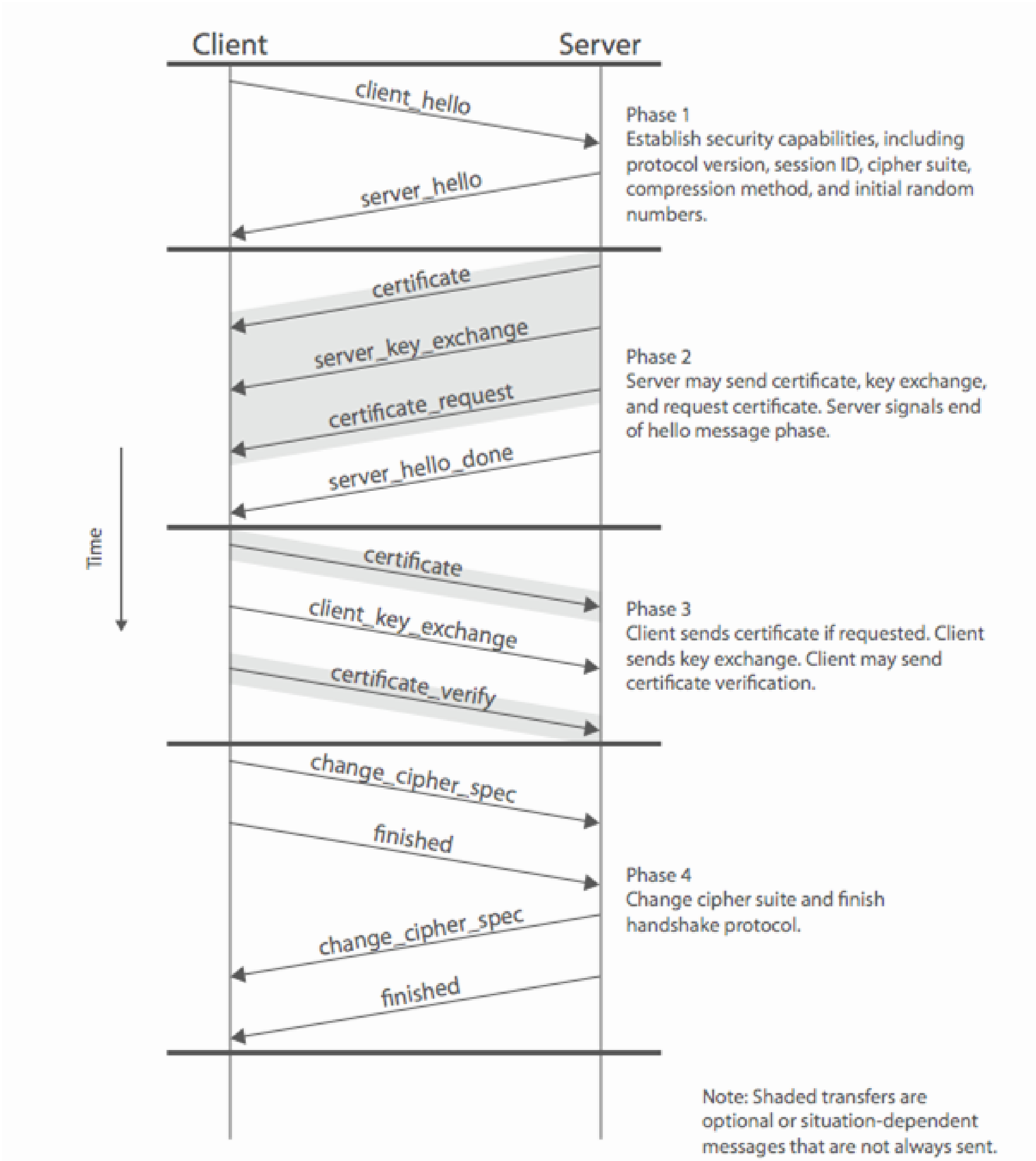
Change cipher: Re-initiate handshake protocol, e.g., to re-negotiate the keying material used for encryption

Alert: Signal warning or fatal problem

- Fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
- Warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown



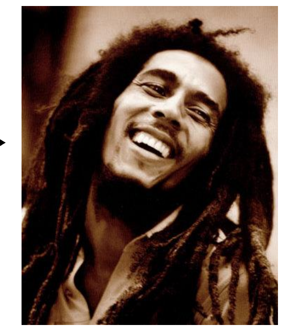
Detailed Protocol



TLS Crypto

Browser

$k_{b\to s}, k_{s\to b}$



Server

$k_{b\to s}, k_{s\to b}$

Unidirectional keys: $k_{b\to s}, k_{s\to b}$

Stateful encryption:

- Each side maintains two 64-bit counters: $ctr_{b\to s}$, $ctr_{s\to b}$
- Init. to 0 when session started. ctr++ for every record.
- Purpose: replay defense

TLS Record Encryption

(CBC AES-128, HMAC-SHA1)

TLS Record

Type	Version	Length
Data	…	…
…	…	…
Tag	Tag	Tag
Tag	Tag	Pad

$k_{b\to s} = (k_{mac}, k_{enc})$

Browser side **enc($k_{b\to s}$, data, $ctr_{b\to s}$) :**

step 1: tag $\leftarrow$ S(k_{mac} , [$++ctr_{b\to s}$ || header || data])

step 2: pad [header || data || tag] to AES block size

step 3: CBC encrypt with k_{enc} and new random IV

step 4: prepend header

TLS Record Decryption

(CBC AES-128, HMAC-SHA1)

Server side **dec($k_{b\to s}$, record, ctr$_{b\to s}$)** :

step 1: CBC decrypt record using k_{enc}

step 2: check pad format, send bad_record_mac if invalid

step 3: check tag on [++ctr$_{b\to s}$ || header || data]

send bad_record_mac if invalid

Provides authenticated encryption

(provided no other info. is leaked during decryption)

TLS Record Decryption

(CBC AES-128, HMAC-SHA1)

Server side **dec($k_{b\to s}$, record, $ctr_{b\to s}$)** :

step 1: CBC decrypt record using k_{enc}

step 2: check pad format, send decryption_failed if invalid

step 3: check tag on [$++ctr_{b\to s}$ || header || data]

send bad_record_mac if invalid

V1.1 Bug:
Only difference is error messages

Padding Oracles

Server side **dec($k_{b\to s}$, record, $ctr_{b\to s}$)** :

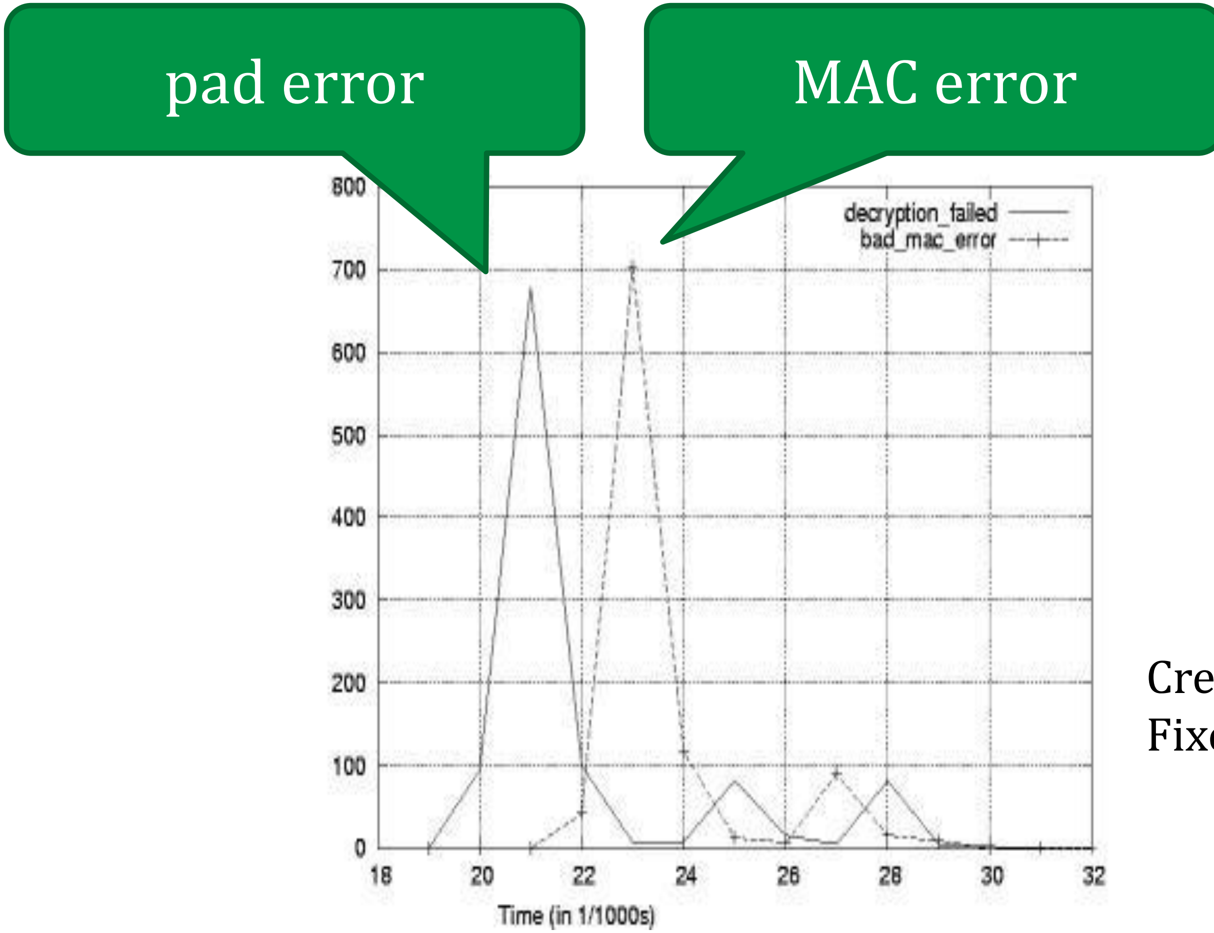
step 1: CBC decrypt record using k_{enc}

step 2: check pad format, abort if *invalid*

step 3: check tag, abort if *invalid*

Two different types of errors:
bad pad vs bad MAC

Padding Attack: Attacker submits ciphertext and learns if last byte of plaintext are a valid pad



Credit: Brice Canvel
Fixed in OpenSSL 0.9.7a

In older TLS 1.0:
padding oracle due to different alert messages.

TLS Padding

Type	Version	Length
Data	...	...
...	...	...
Tag	Tag	Tag
Tag	Tag	Pad

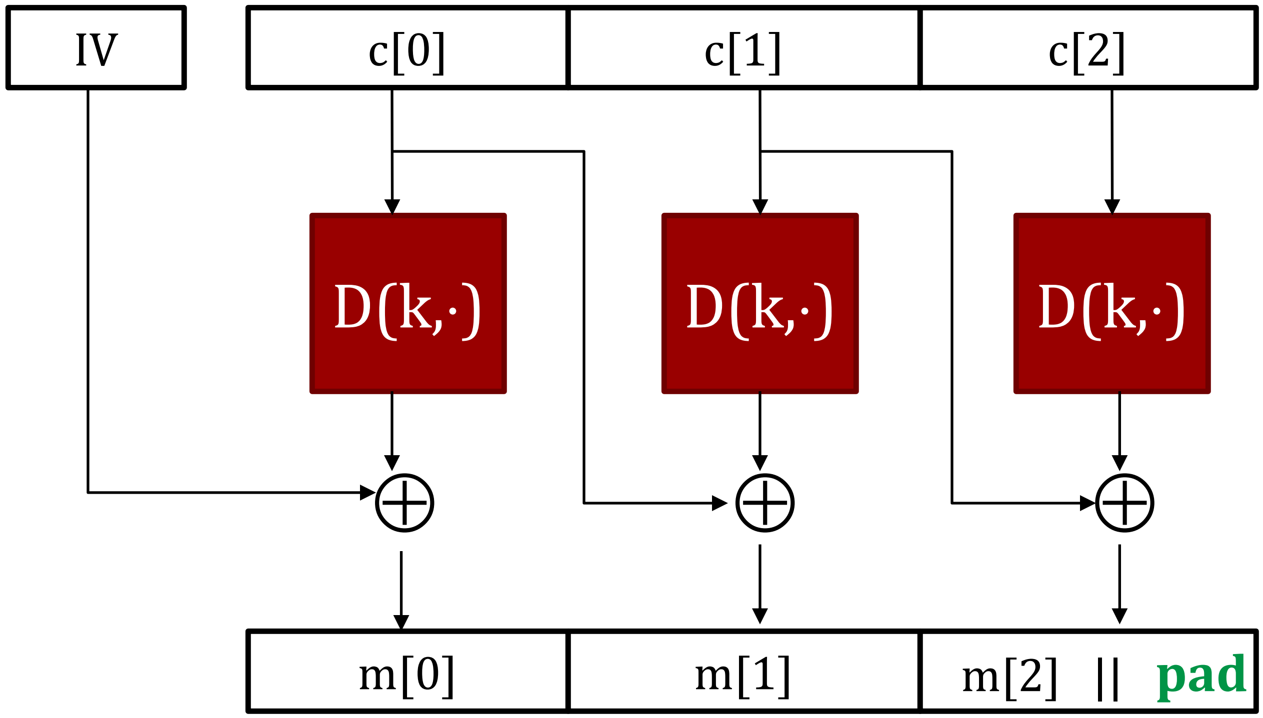
Valid paddings:

- 0x01 for 1 byte padding
- 0x02 0x02 for 2 byte padding
- 0x03 0x03 0x03 for 3 byte padding
- ....

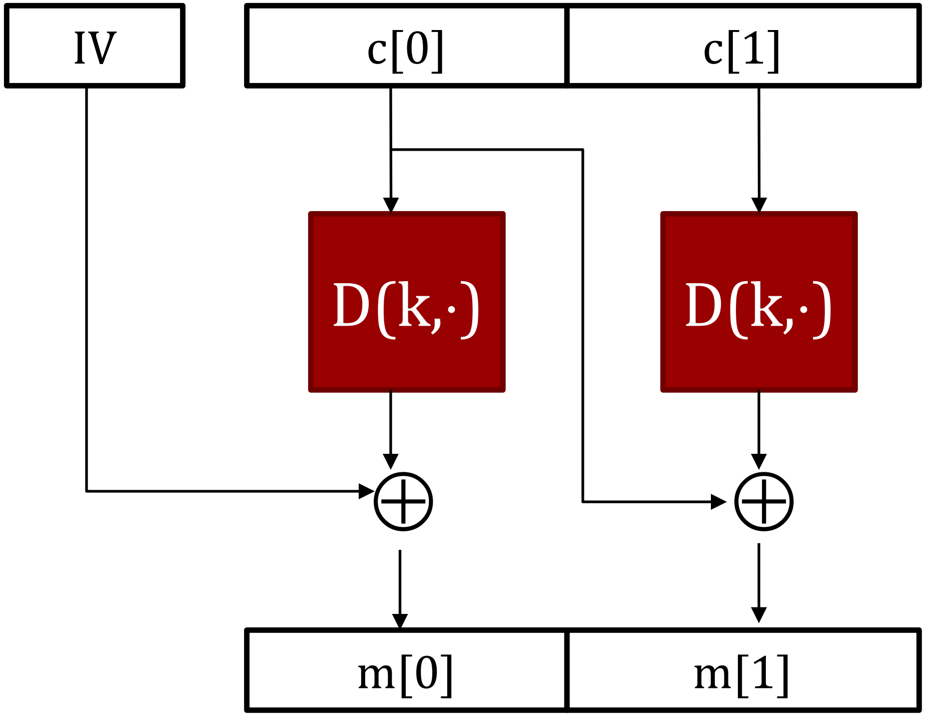
Using a Padding Oracle with CBC

Example:

Attacker has ciphertext c = (c[0], c[1], c[2]) and wants m[1]. We'll show you how to get last byte of m[1]. (Full break possible)

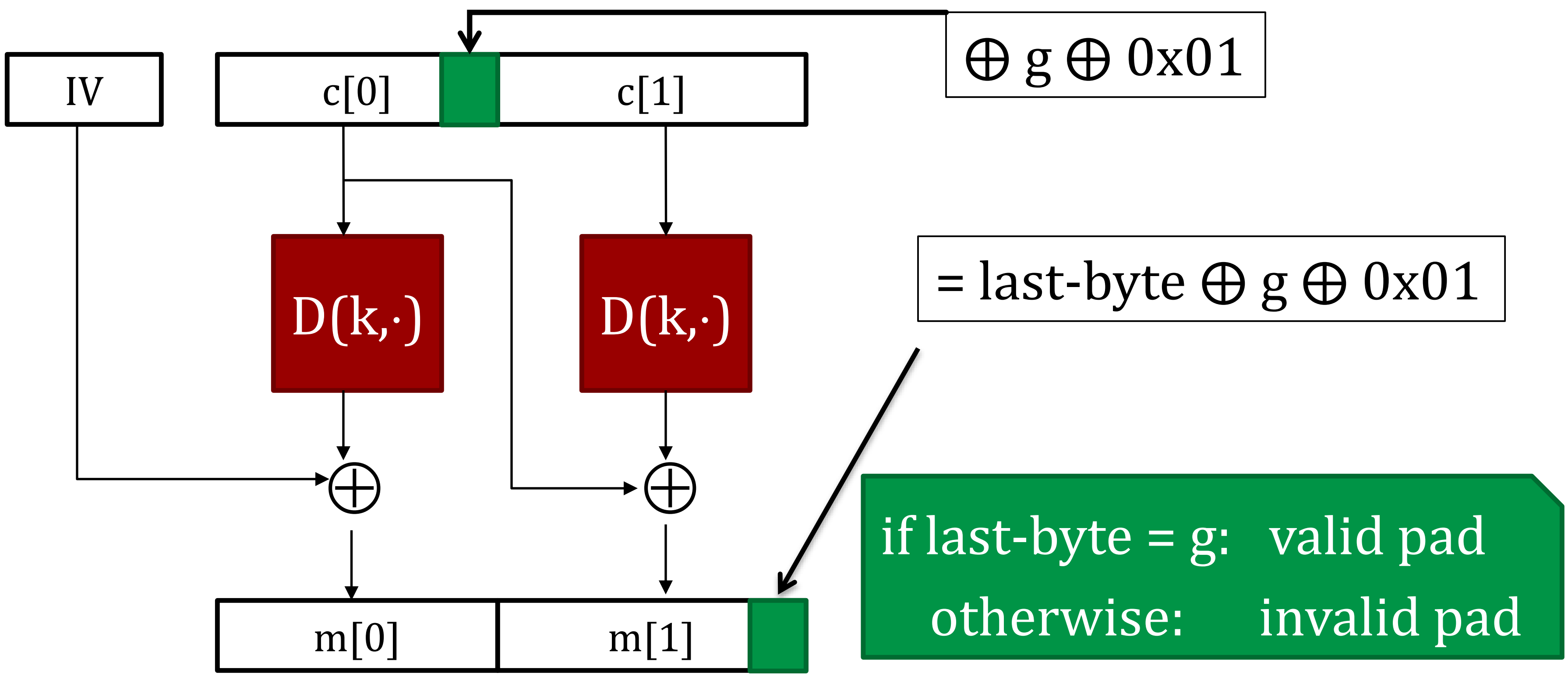


Step 1: Throw Away c[2]



Step 2: Guess and Check if Padding Valid

Let g be our guess for the last byte of m[1]



*note MAC will fail, but we get the byte.

Using a Padding Oracle

Attack: submit (IV, c'[0], c[1]) to padding oracle
⇒ attacker learns if last byte = g

Repeat with g = 0,1, …, 255 to learn last byte of m[1]

Then use a (0x02, 0x02) pad to learn the next byte
and so on …

Another TLS Bug Prior to 1.1

IV for CBC is predictable using chained IV

- IV for next record is last ciphertext block of current record.
- Not CPA secure (see block cipher lecture). BEAST attack is a practical implementation

Other Problems

The TLS header leaks the length of TLS records

- Lengths can also be inferred by observing network traffic

For many web applications, leaking lengths reveals sensitive info:

- In tax preparation sites, lengths indicate the type of return being filed which leaks information about the user’s income
- In healthcare sites, lengths leaks what page the user is viewing
- In Google maps, lengths leaks the location being requested

No easy solution

Lesson

1. Encrypt-then-MAC would completely avoid many problem.
   – MAC is checked first and ciphertext discarded if invalid
2. MAC-then-CBC provides Authenticated Encryption, but padding oracle destroys it

Certificate Revocation

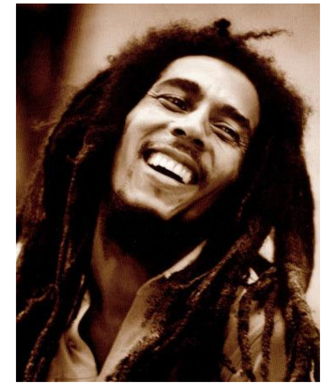
What to do if your keys are compromised.

Certificate Revocation

Alice

1. ClientHello

2. ServerHello
(send cert., e.g., pub key e)



Bob

1. Check CA signature on key
2. ....
3. Accept key

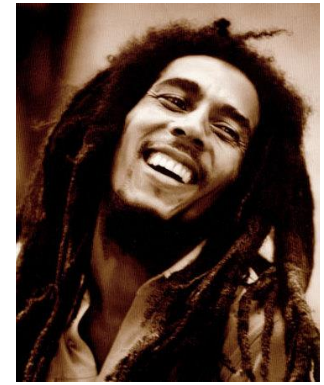
What needs to happen here?

Certificate Revocation

1. ClientHello

Alice

2. ServerHello
(send cert., e.g., pub key e)



Bob

Verification protocol

Verification Authority

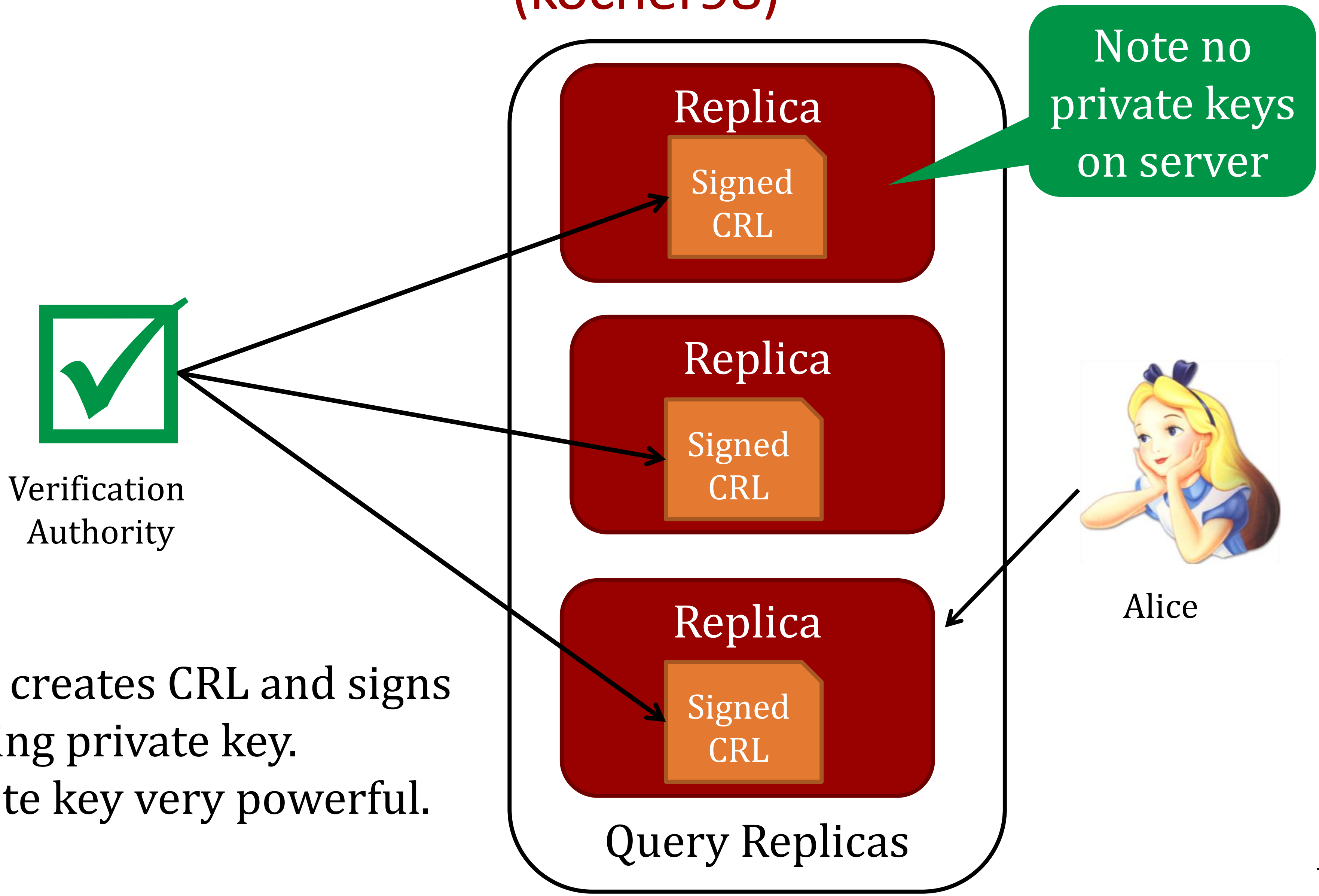
Certificate Verification Protocols

- Expiration Date

- Certificate Revocation Lists (CRL) and Certificate Revocation Trees (CRT)

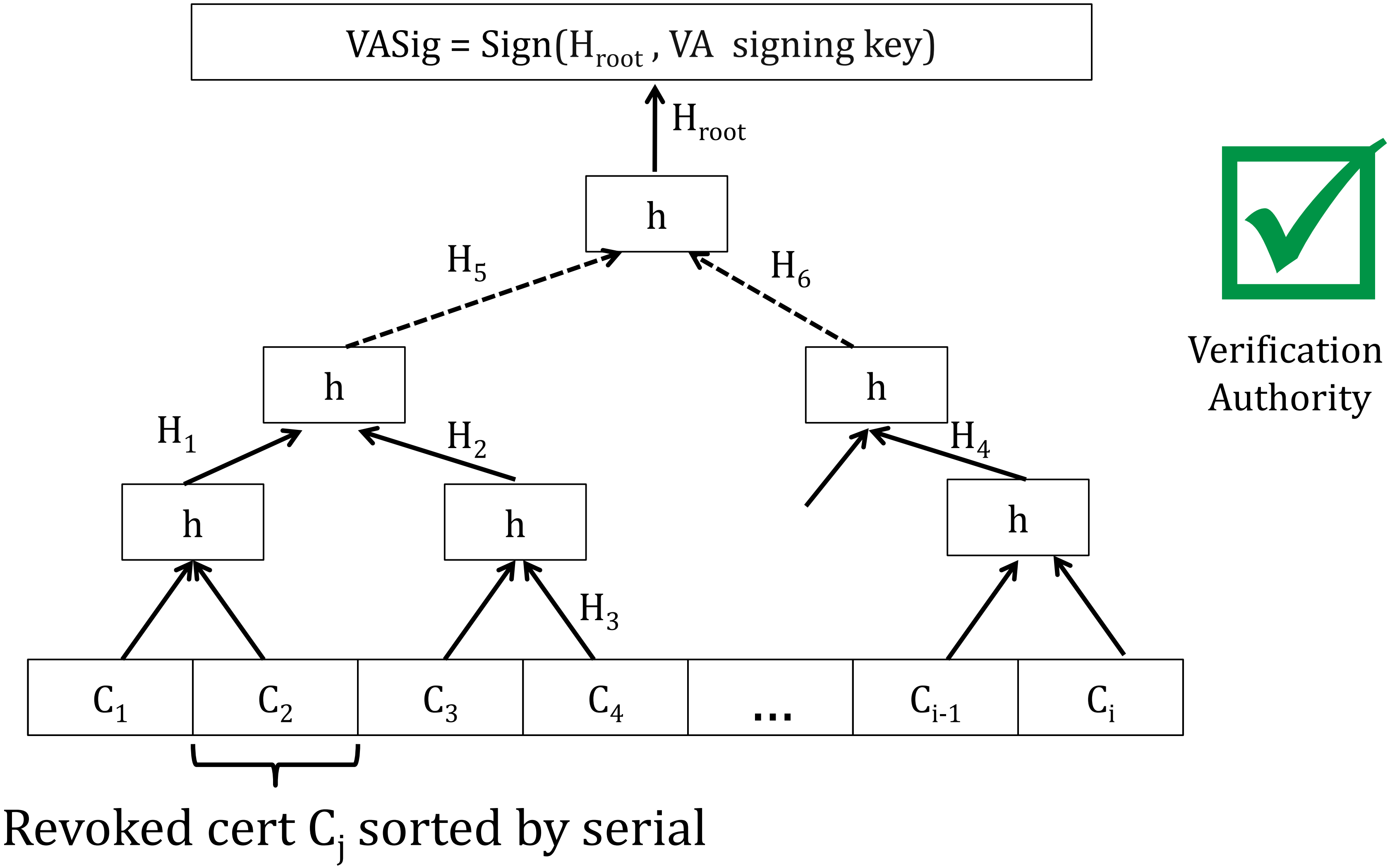
- OCSP – Online Cert Status Protocol

Efficient Certificate Revocation Lists (kocher98)



VA creates CRL and signs using private key.
Note key very powerful.

Certificate Revocation Tree Generation



1. Is Bob's Cert C_2 revoked

2. [C_1, H_2, H_6, VASig]

Alice

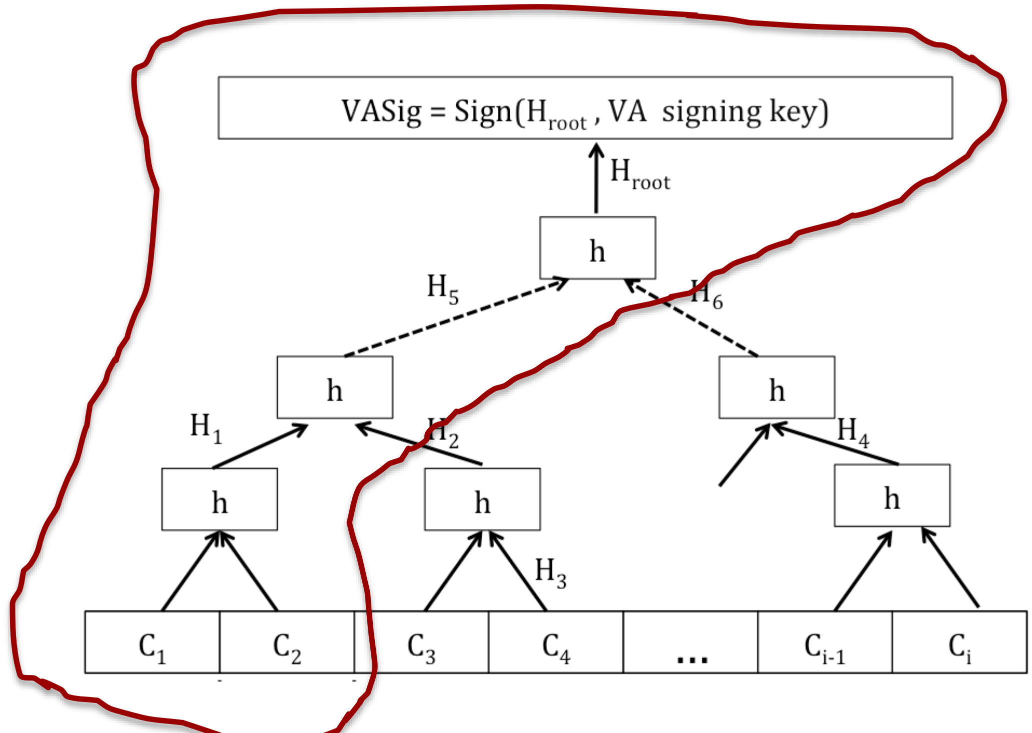
VA Replica

Signed CRL

3. Alice validates C_2 by:

- $H'_{root} = H(H(C_1, C_2), H_2, H_6)$
- $H' =?= H$
- VA Sig valid?

Size of Proof: O(log i)



Online Cert Status Protocol

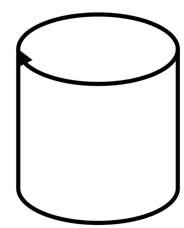
Alice

1. Request(Bob’s Cert)

3. Response(
Sign(Bob’s Cert {OK,BAD})
VA Signing Key
)

Verification Authority

2. Check DB



Implemented in IE7 (Vista+), Firefox, Safari (by default Lion+), Opera, Chrome