CSE P 564 / 590 (Autumn 2012)

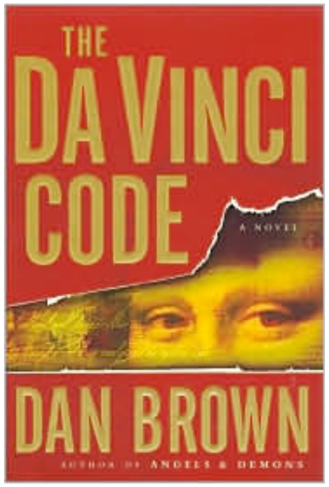
Cryptography

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell,
Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Cryptography and Security

- Art and science of *protecting* our *information*.
  - Keeping it private, if we want privacy
  - Protecting its integrity, if we want to avoid forgeries.



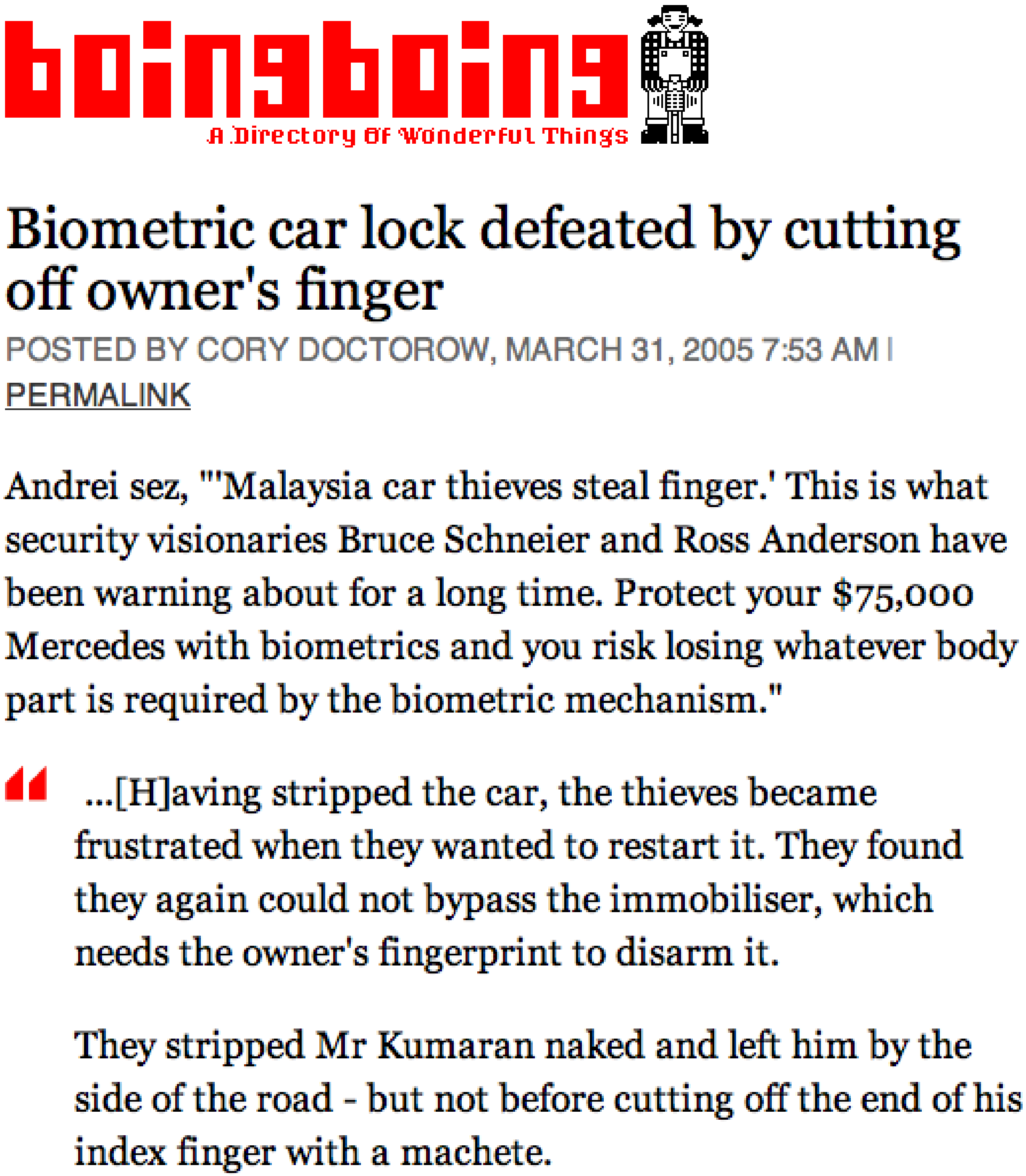
Images from Wikipedia and Barnes and Noble

Some thoughts about cryptography

- Cryptography only one small piece of a larger system
- Must protect entire system
  - Physical security
  - Operating system security
  - Network security
  - Users
  - **Cryptography** (following slides)
- "Security only as strong as the weakest link"
  - Need to secure weak links
  - But not always clear what the weakest link is (different adversaries and resources, different adversarial goals)
  - Crypto failures may not be (immediately) detected
- Cryptography helps after you've identified your threat model and goals

Improved se

- RFIDs in car ke
  - RFIDs in car ke
  - Result: Car ja



Biometric car lock defeated by cutting off owner's finger

POSTED BY CORY DOCTOROW, MARCH 31, 2005 7:53 AM | PERMALINK

Andrei sez, "'Malaysia car thieves steal finger.' This is what security visionaries Bruce Schneier and Ross Anderson have been warning about for a long time. Protect your $75,000 Mercedes with biometrics and you risk losing whatever body part is required by the biometric mechanism."

> ...[H]aving stripped the car, the thieves became frustrated when they wanted to restart it. They found they again could not bypass the immobiliser, which needs the owner's fingerprint to disarm it.
>
> They stripped Mr Kumaran naked and left him by the side of the road - but not before cutting off the end of his index finger with a machete.

Key Entry Pad (4-digit PIN)

Image from profmason.com

- This is the key pad on my office safe.
- Inside my safe is a copy of final exam.
- How long would it take a you to break in?

- Answer (combinatorics):
  - 10^4 tries *maximum*.
  - 10^4 / 2 tries on *average*.
- Answer (unit conversion):
  - 3 seconds per try --> 4 hours and 10 minutes on average

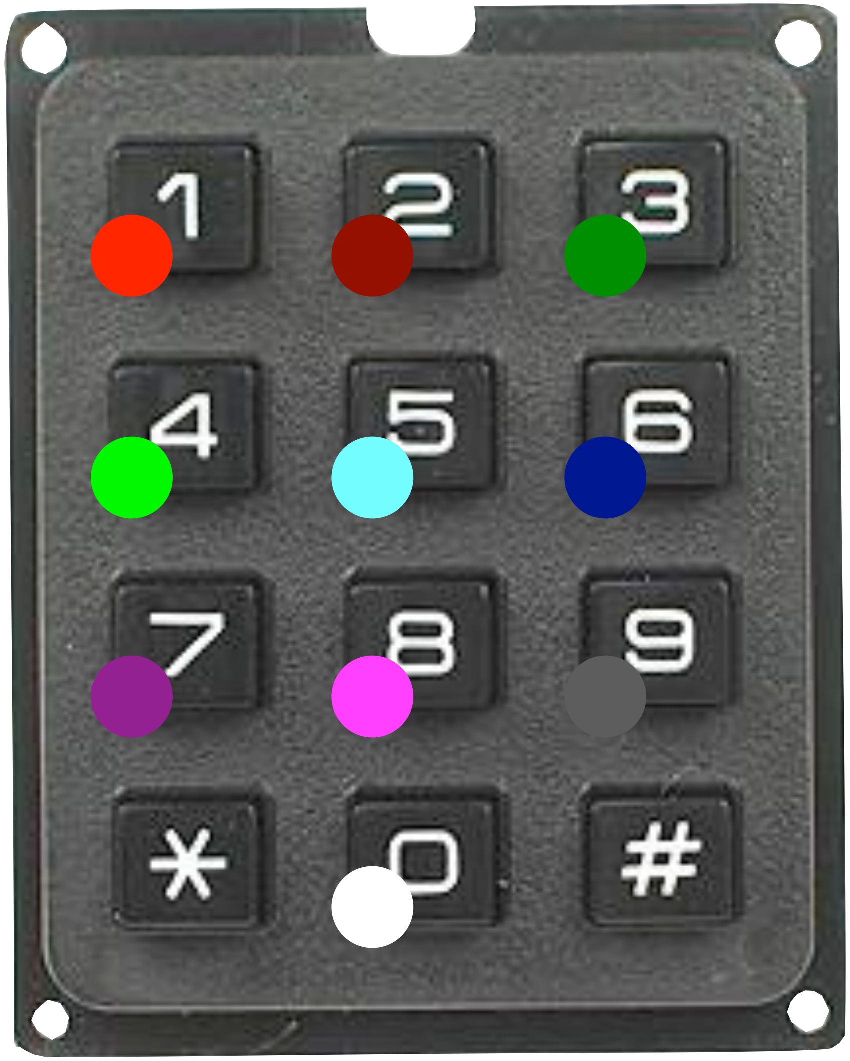
Key Entry Pad (4-digit PIN)

Image from profmason.com

- Now assume the safe automatically calls police after 3 failed attempts.
- What is the probability that you will guess the PIN within 3 tries?
- (Assume no repeat tries.)

- Answer (combinatorics):
  - 10000 choose 3 possible choices for the 3 guesses
  - 1 × (9999 choose 2) possible choices contain the correct PIN
  - So success probability is 3 / 10000

Key Entry Pad (4-digit PIN)



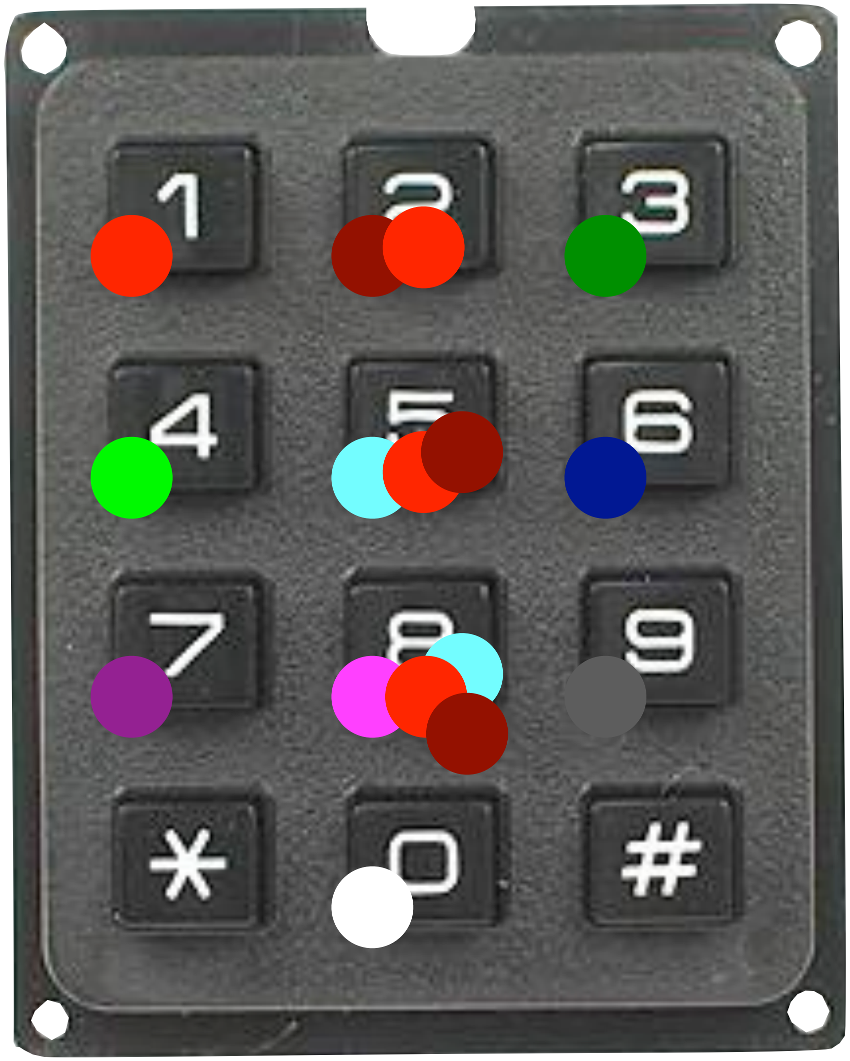
- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)



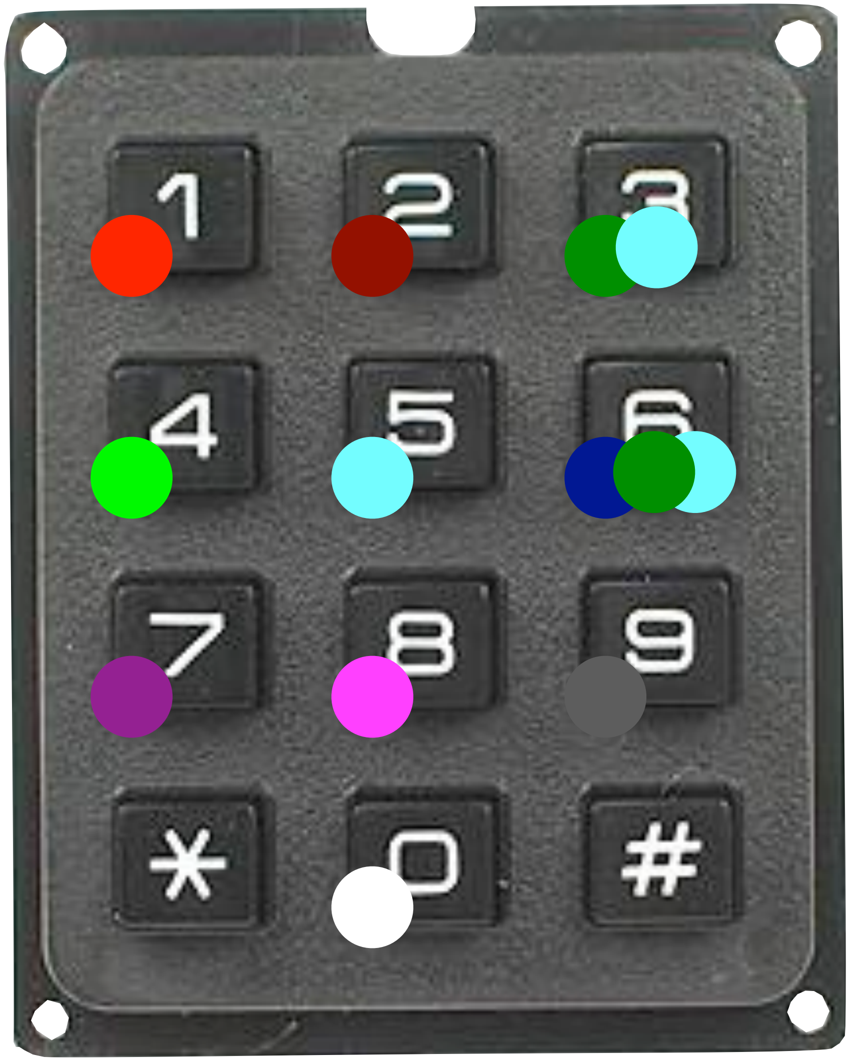
- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)



- Could you do better at guessing the PIN?

- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

Image from profmason.com

Idea from http://eprint.iacr.org/2003/217.ps

Key Entry Pad (4-digit PIN)

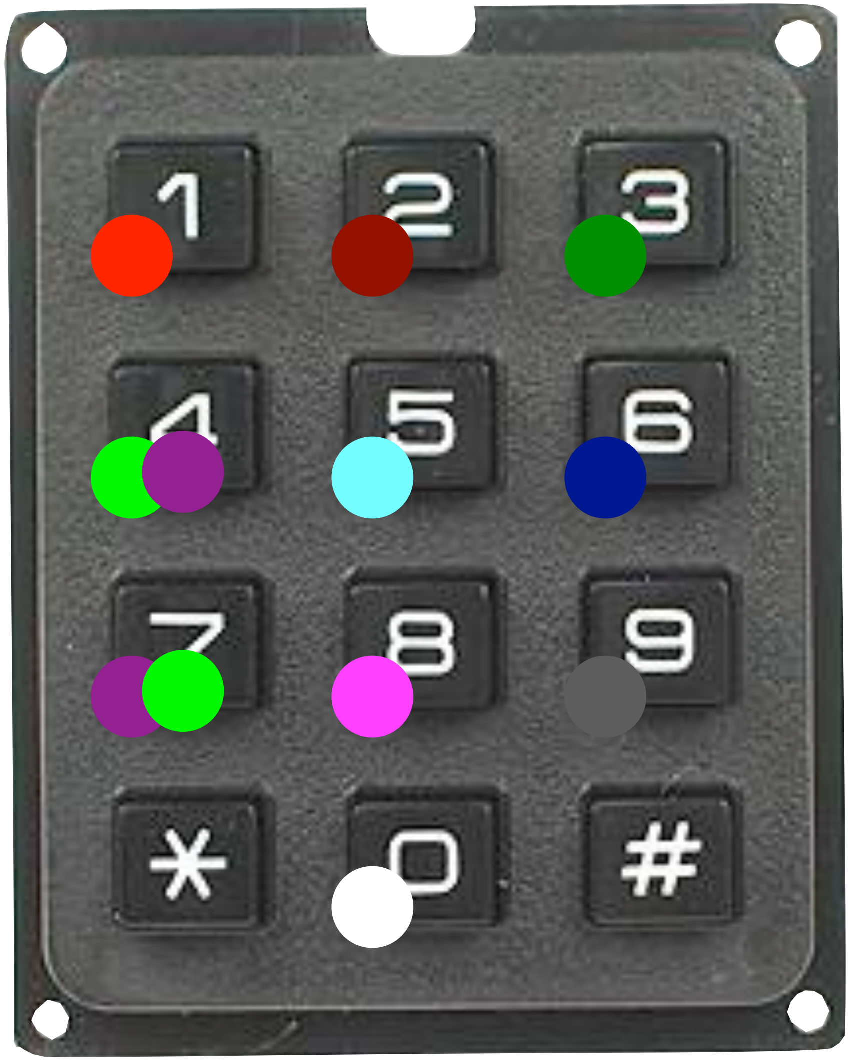


Image from profmason.com

- Could you do better at guessing the PIN?

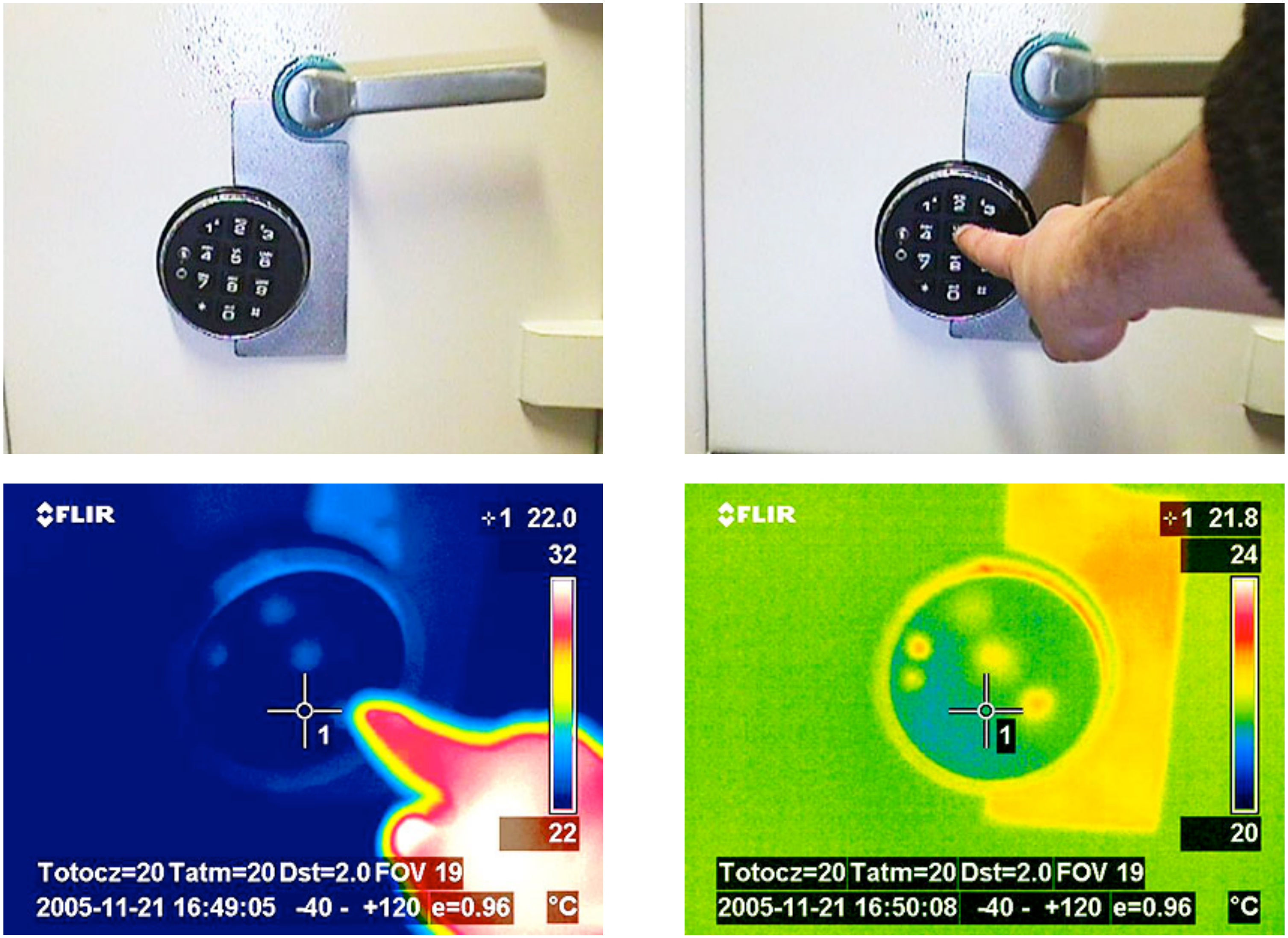
- Answer (*chemical* combinatorics):
  - Put different chemical on each key (NaCl, KCl, LiCl, ...)
  - Observe residual patterns after I access safe

Lesson: Consider the complete system, physical security, etc

Lesson: Think outside the box

Idea from http://eprint.iacr.org/2003/217.ps

Thermal Patterns

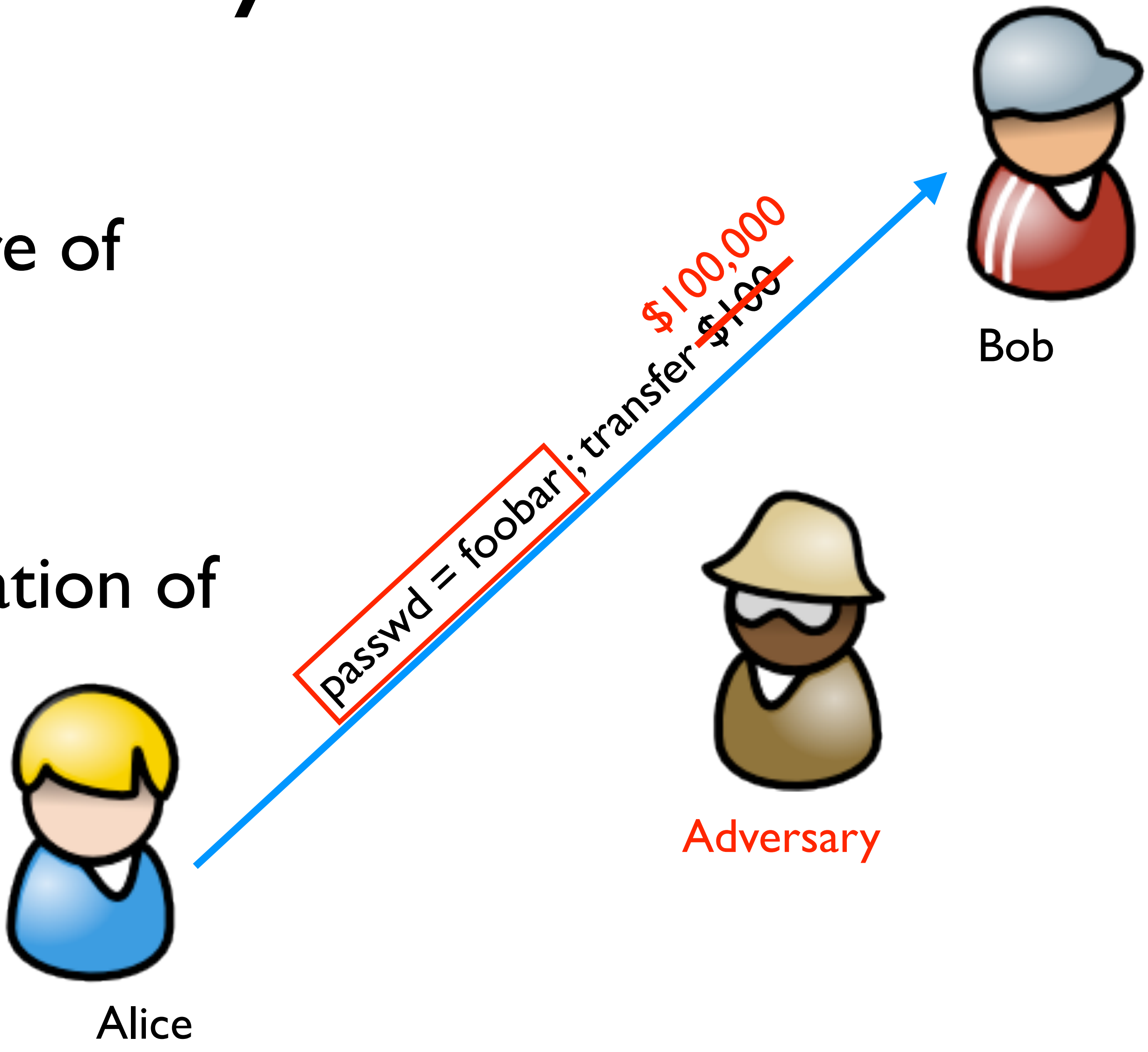

FLIR
1 22.0
32
1
22
Totocz=20 Tatm=20 Dst=2.0 FOV 19
2005-11-21 16:49:05 -40 - +120 e=0.96
°C
FLIR
1 21.8
24
1
20
Totocz=20 Tatm=20 Dst=2.0 FOV 19
2005-11-21 16:50:08 -40 - +120 e=0.96
°C

Images from http://lcamtuf.coredump.cx/tsafe/

Common Communication Security Goals

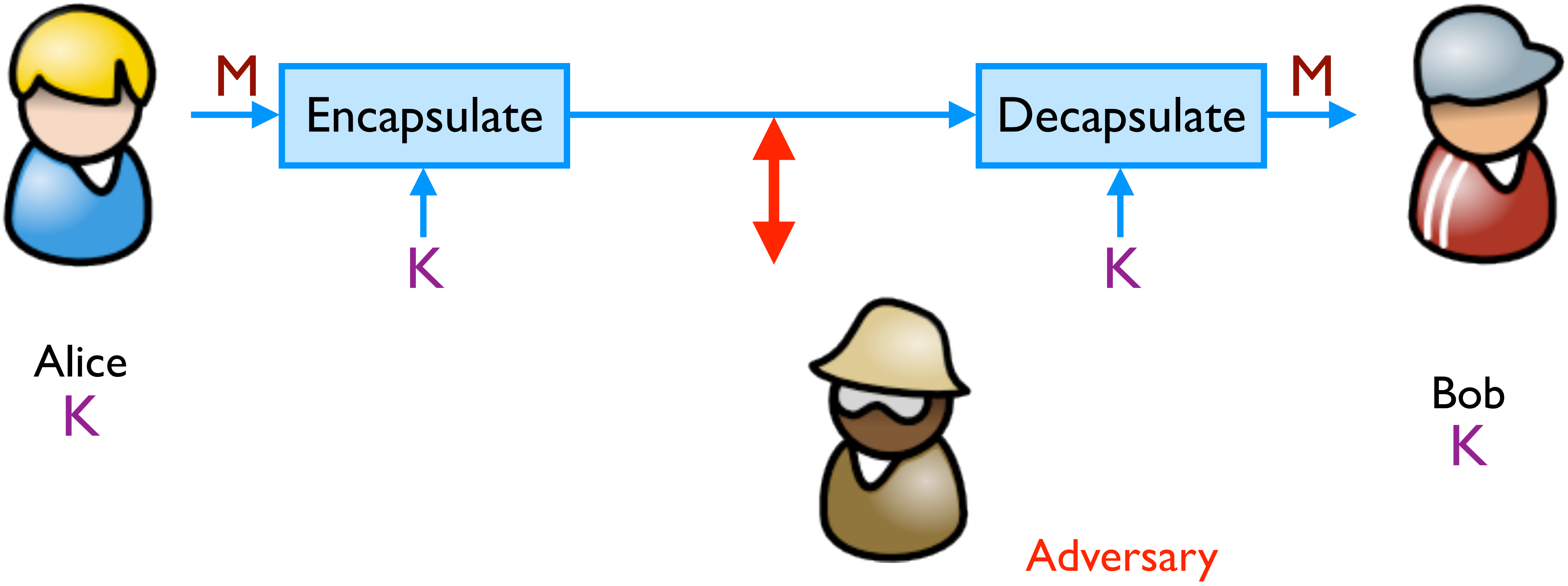
Privacy of data
Prevent exposure of information

Integrity of data
Prevent modification of information



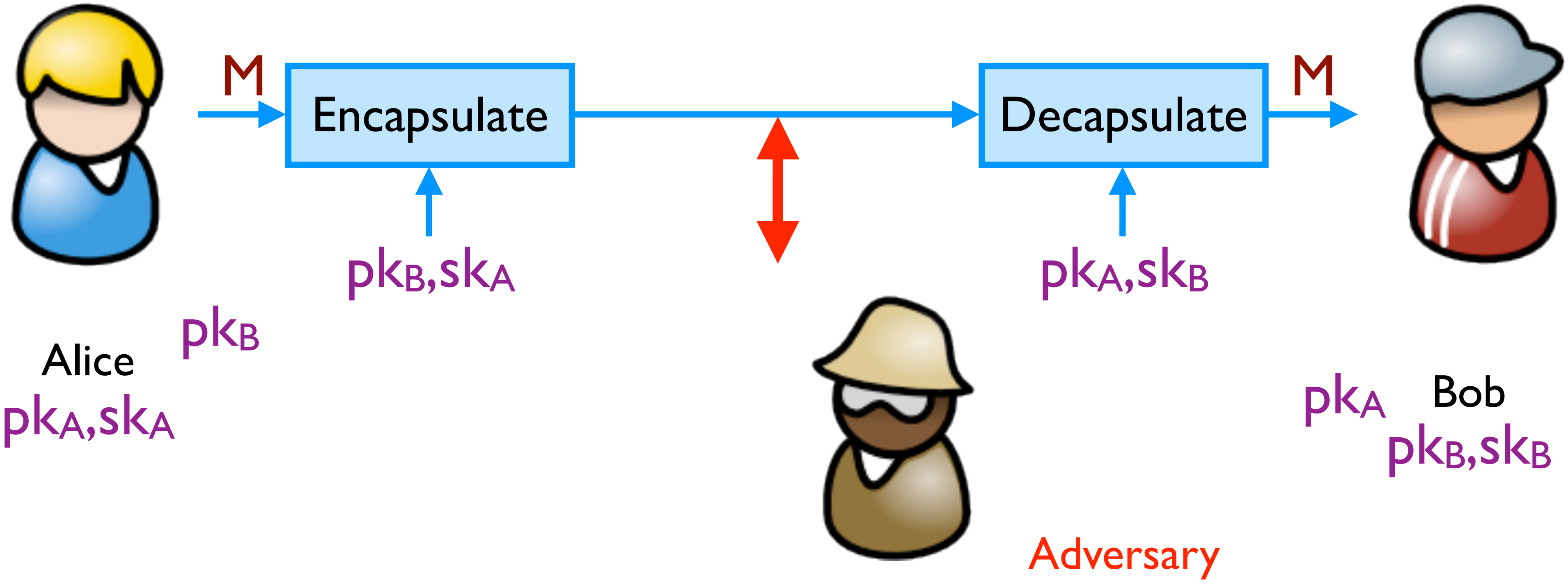
Symmetric Setting

Both communicating parties have access to a shared random string K, called the key.



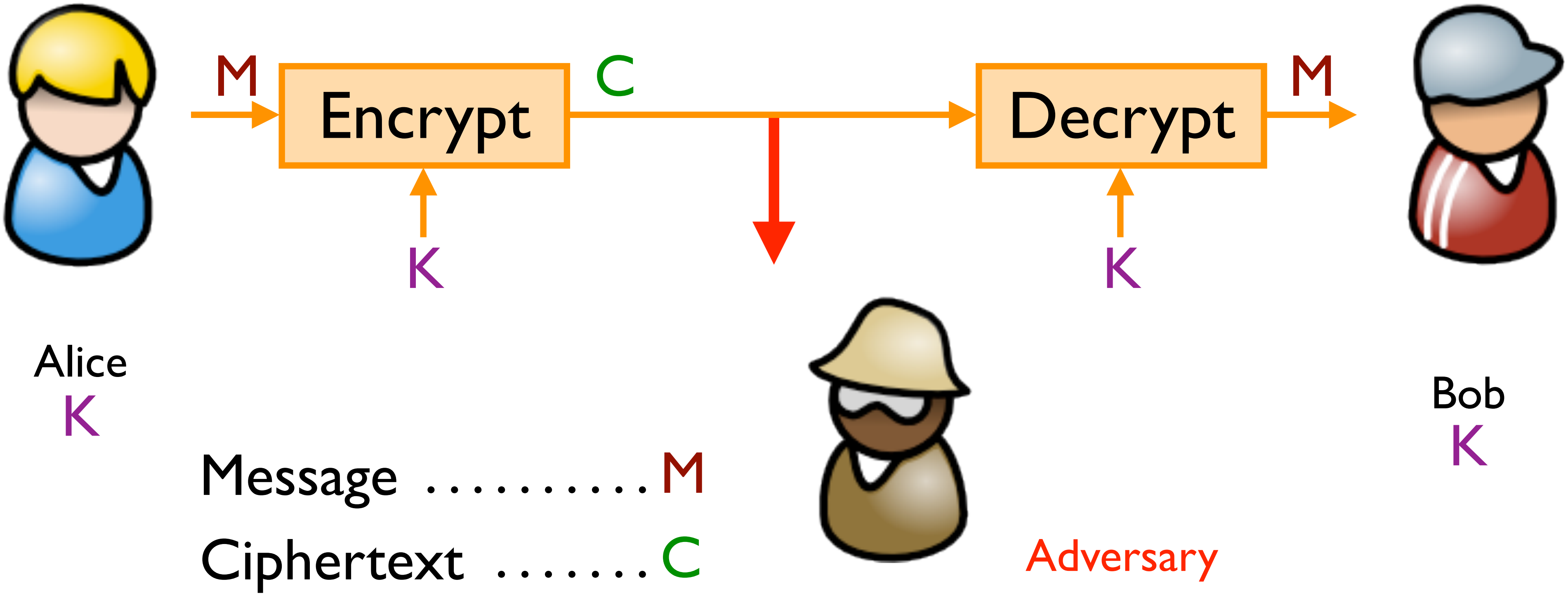
Asymmetric Setting

Each party creates a public key pk and a secret key sk.



Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.

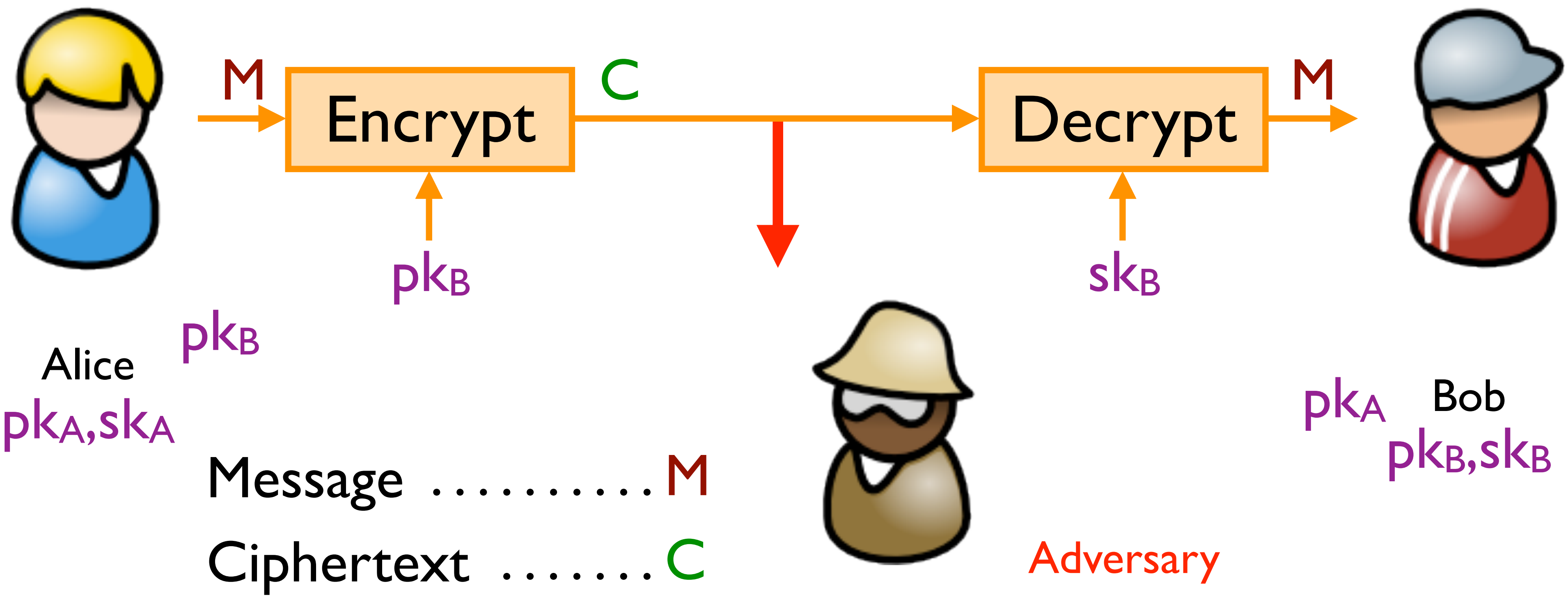


Message . . . . M

Ciphertext C

Achieving Privacy (Asymmetric)

Encryption schemes: A tool for protecting privacy.



M

Encrypt

C

Decrypt

M

pk_B

sk_B

pk_B

Alice

pk_A, sk_A

pk_A

Bob

pk_B, sk_B

Message . . . . M

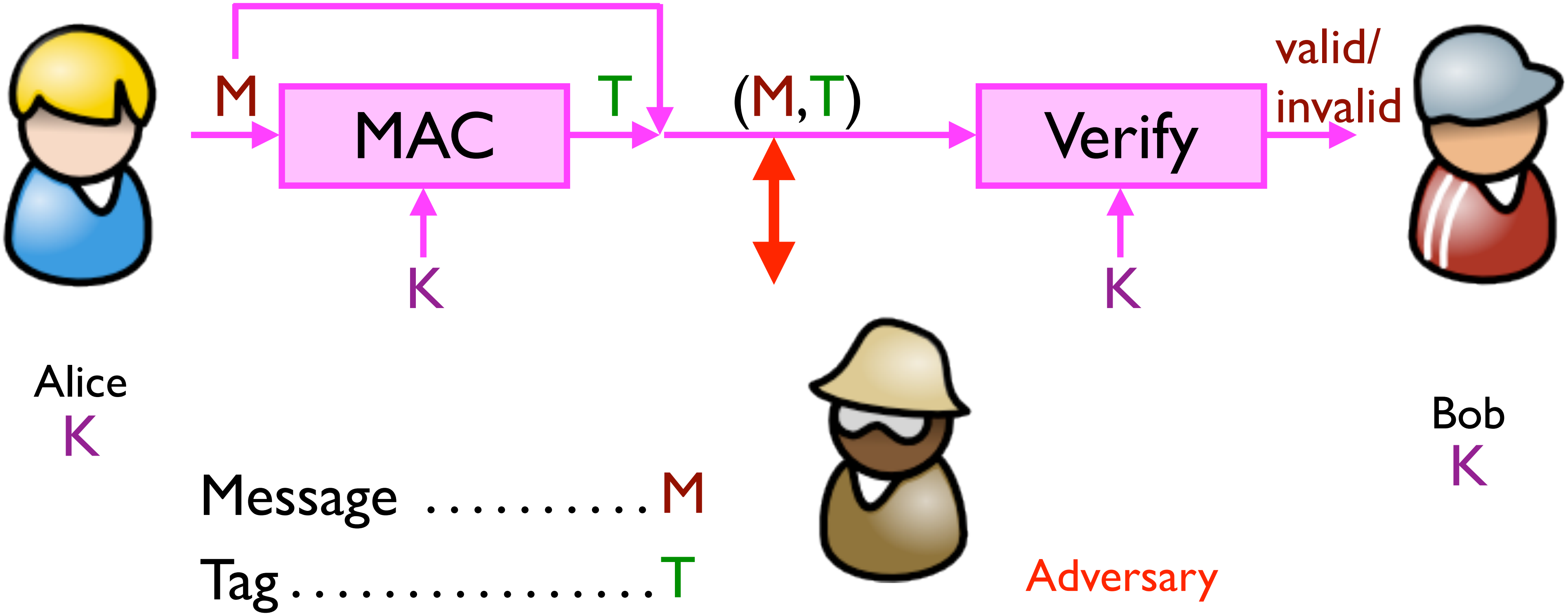
Ciphertext C

Adversary

Achieving Integrity (Symmetric)

Message authentication schemes: A tool for protecting integrity.

(Also called message authentication codes or MACs.)

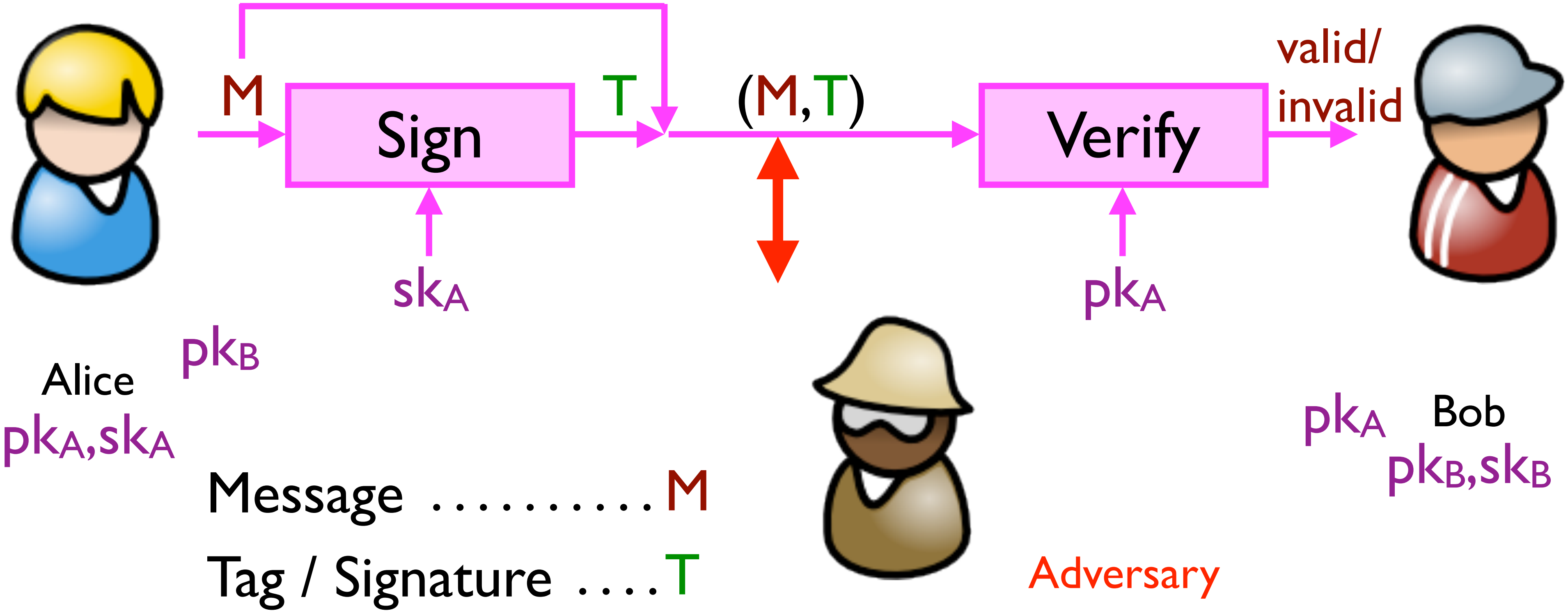


Message . . . . M

Tag . . . . T

Achieving Integrity (Asymmetric)

Digital signature schemes: A tool for protecting integrity and authenticity.

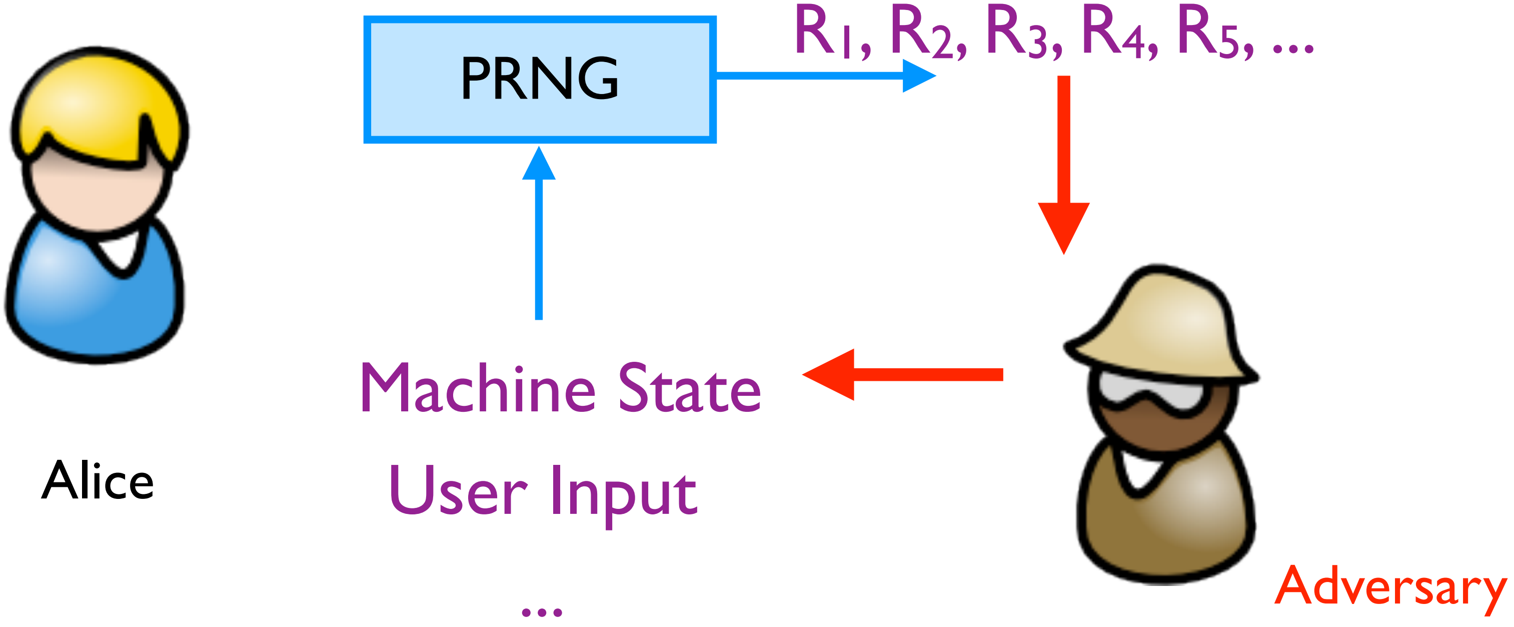


Message . . . . M

Tag / Signature . . . . T

“Random” Numbers

Pseudorandom Number Generators (PRNGs)



PRNG

$R_1, R_2, R_3, R_4, R_5, \ldots$

Alice

Machine State

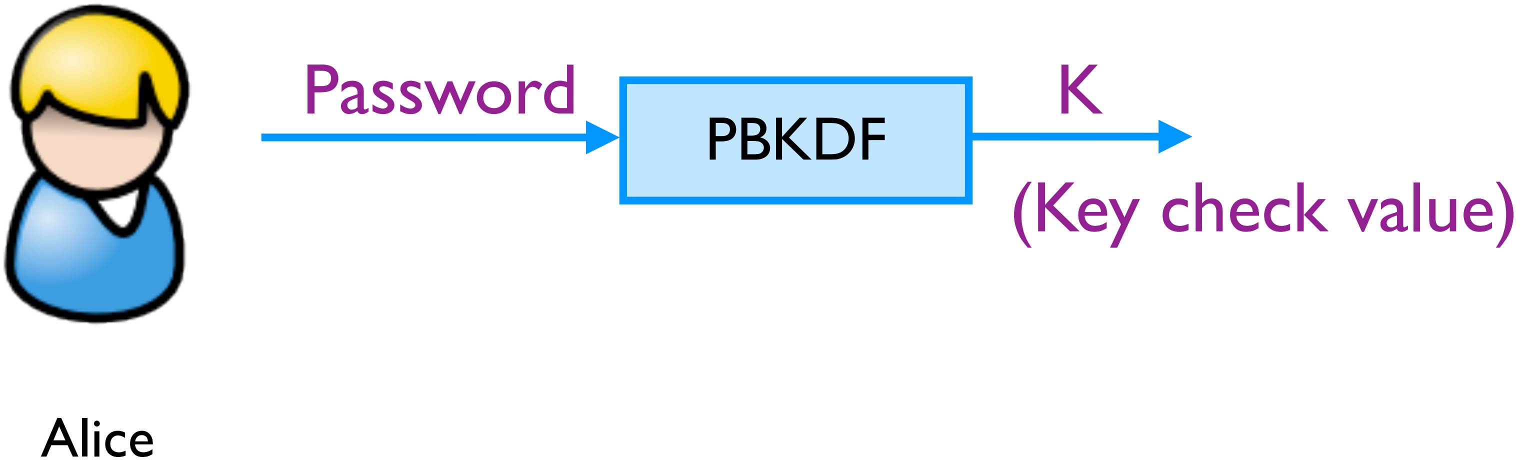
User Input

...

Adversary

Getting keys: PBKDF

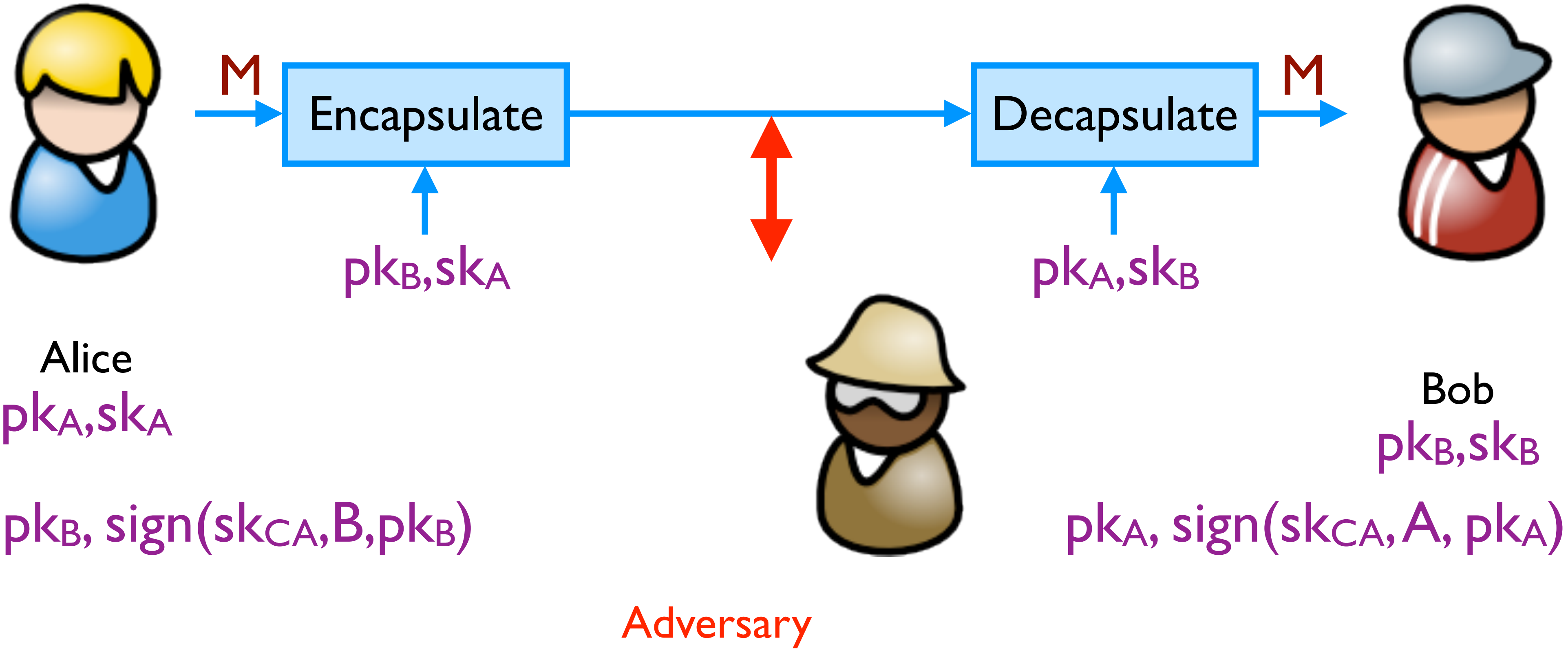
Password-based Key Derivation Functions



Getting keys: CAs

Each party creates a public key pk and a secret key sk.

(Public keys signed by a trusted third party: a certificate authority.)



M

Encapsulate

Decapsulate

M

pk_B, sk_A

pk_A, sk_B

Alice

pk_A, sk_A

pk_B, sign(sk_{CA}, B, pk_B)

Bob

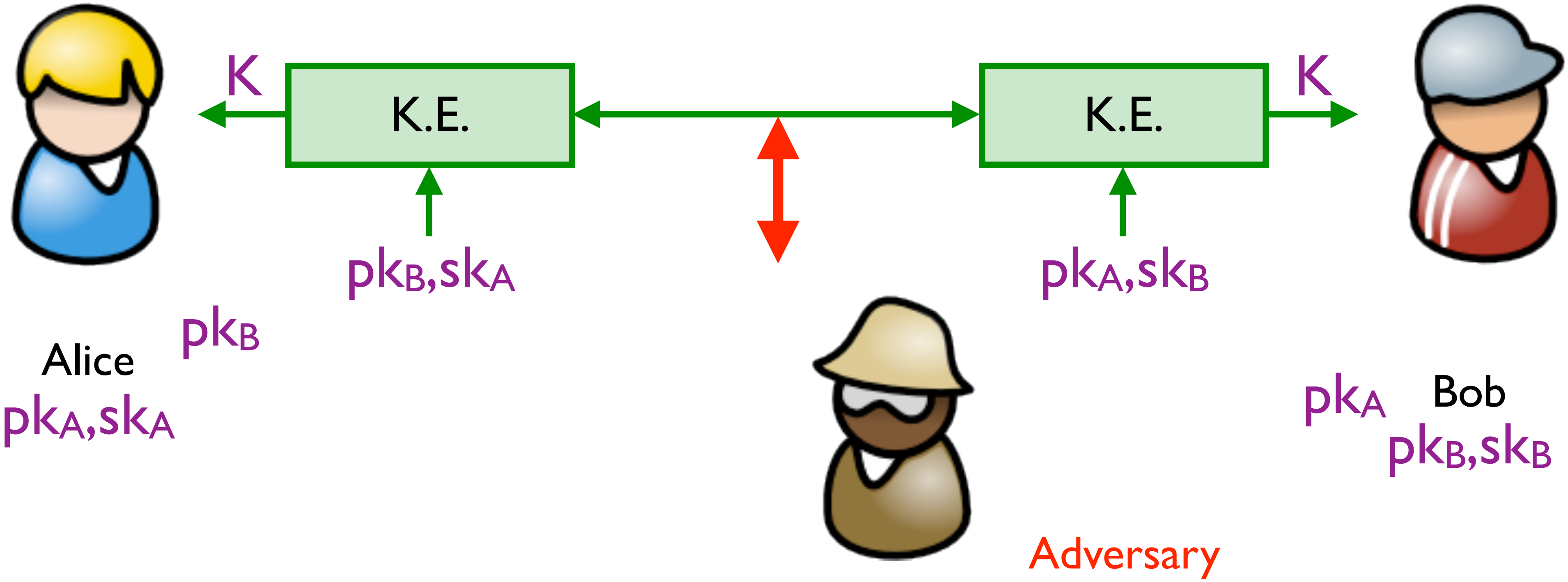
pk_B, sk_B

pk_A, sign(sk_{CA}, A, pk_A)

Adversary

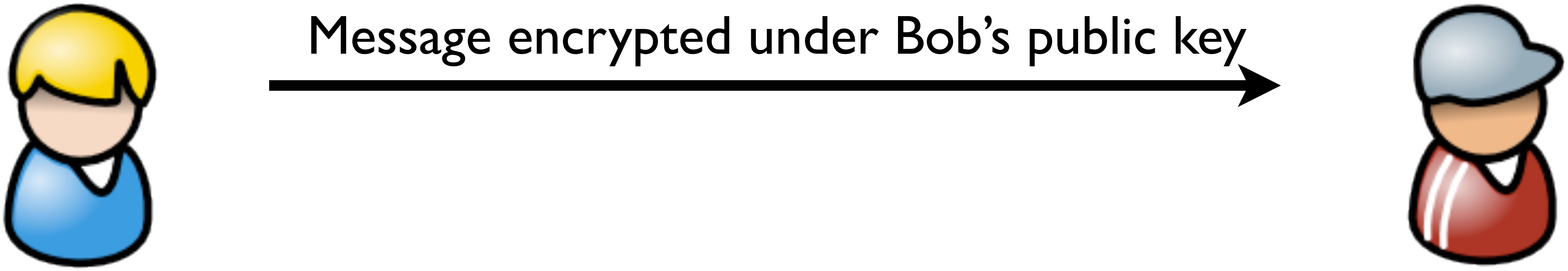
Getting keys: Key exchange

Key exchange protocols: A tool for establishing a shared symmetric key from public keys



One-way Communications

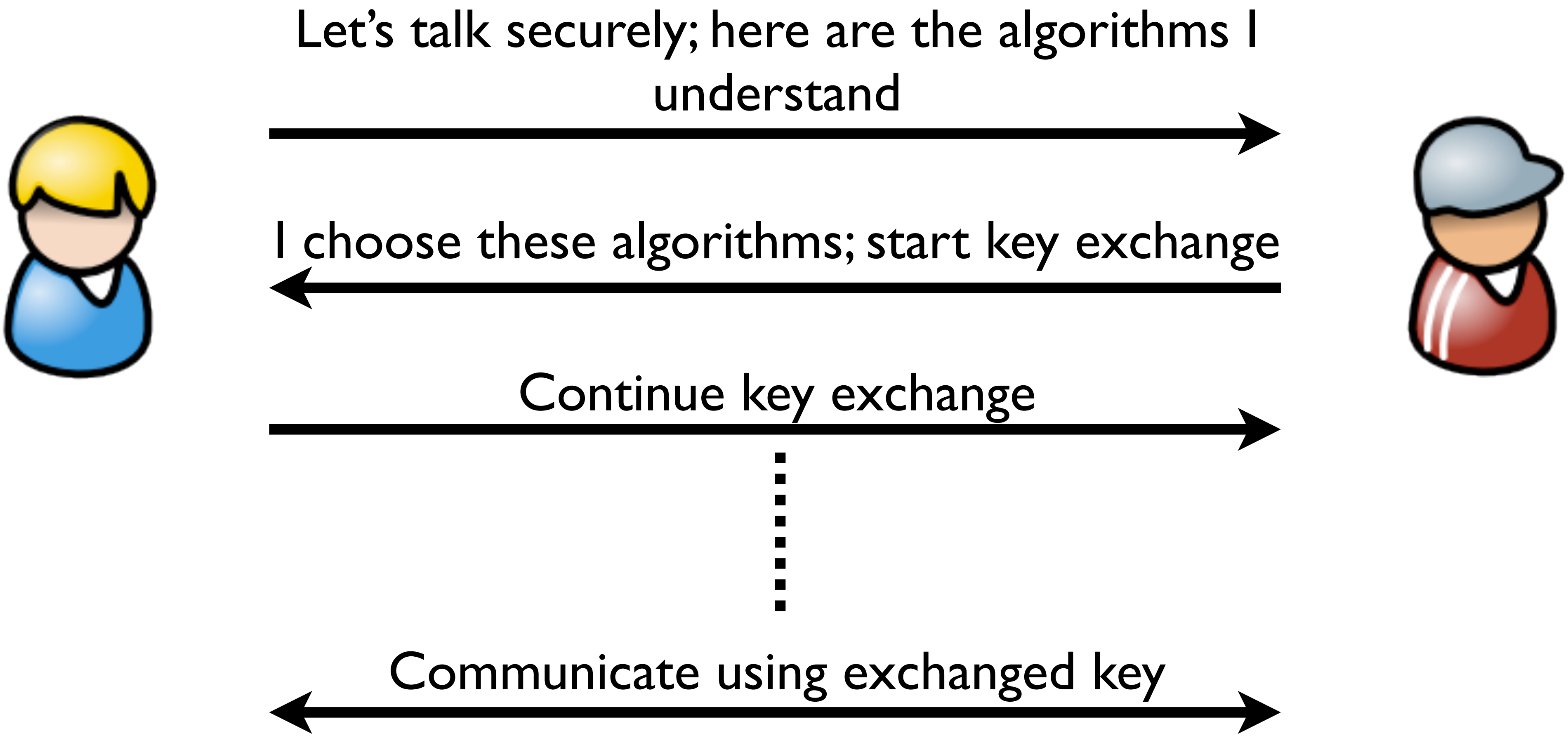
PGP is a good example



Message encrypted under Bob’s public key

Interactive Communications

In many cases, it’s probably a good idea to just use a standard protocol/system like SSH, SSL/TLS, etc...



Let’s talk securely; here are the algorithms I understand

I choose these algorithms; start key exchange

Continue key exchange

Communicate using exchanged key

Let’s Dive a Bit Deeper

One-way Communications

(*Informal* example; ignoring, e.g., signatures)

1. Alice gets Bob's public key; Alice *verifies* Bob's public key (e.g., via CA)
2. Alice generates random symmetric keys K1 and K2
3. Alice encrypts the message M the key K1; call result C
4. Alice authenticates (MACs) C with key K2; call the result T
5. Alice encrypts K1 and K2 with Bob's public key; call the result D

6. Send D, C, T

(Assume Bob's private key is encrypted on Bob's disk.)

7. Bob takes his password to derive key K3
8. Bob decrypts his private key with key K3
9. Bob uses private key to decrypt K1 and K2
10. Bob uses K2 to verify MAC tag T
11. Bob uses K1 to decrypt C

Interactive Communications

(*Informal* example; details omitted)

1. Alice and Bob exchange public keys and certificates
2. Alice and Bob use CA's public keys to verify certificates and each other's public keys
3. Alice and Bob take their passwords and derive symmetric keys
4. Alice and Bob use those symmetric keys to decrypt and recover their asymmetric private keys.
5. Alice and Bob use their asymmetric private keys and a *key exchange* algorithm to derive a shared symmetric key

(They key exchange process will require Alice and Bob to generate new pseudorandom numbers)

6. Alice and Bob use shared symmetric key to encrypt and authenticate messages

(Last step will probably also use random numbers; will need to rekey regularly; may need to avoid replay attacks,...)

What cryptosystems have you heard of? (Past or present)

History

- Substitution Ciphers
  - Caesar Cipher
- Transposition Ciphers
- Codebooks
- Machines

- Recommended Reading: **The Codebreakers** by David Kahn and **The Code Book** by Simon Singh.
  - Military uses
  - Rumrunners
  - ....

Classic Encryption

- Goal: To communicate a secret message
- Start with an *algorithm*
- Caesar cipher (substitution cipher):

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

GHIJKLMNOPQRSTUVWXYZABCDEF
```

Then add a secret key

- Both parties know that the secret word is "victory":

  ABCDEFGHIJKLMNOPQRSTUVWXYZ

  VICTORYABDEFGHJKLMNPQSUWXZ

- "state of the art" for thousands of years

Kerckhoff’s Principle

- Security of a cryptographic object should depend **only** on the secrecy of the secret (private) key

- Security should not depend on the secrecy of the algorithm itself.

Mid-way Summary

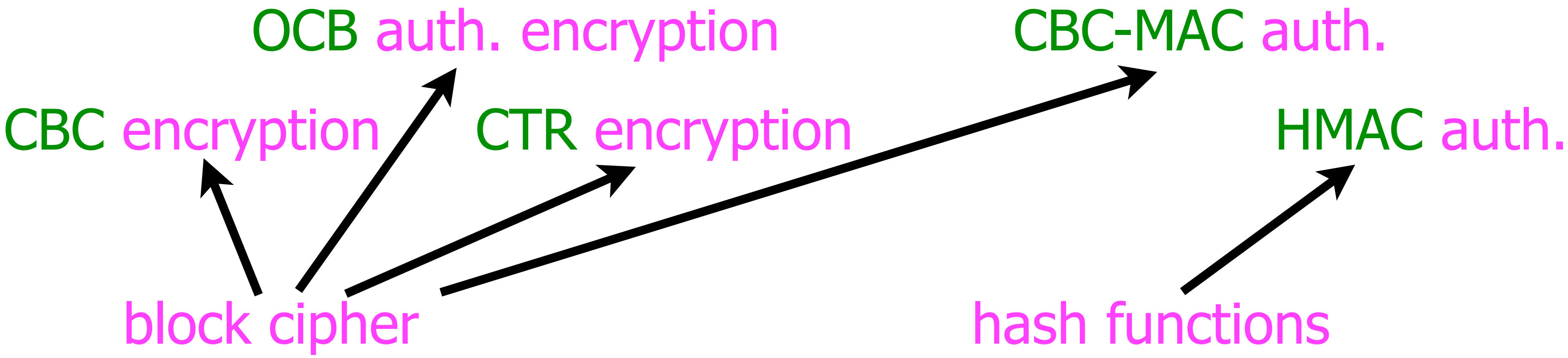
- **Symmetric cryptography**
  - Both sides know ***shared key***, no one else knows anything. Can ***encrypt, decrypt, sign/MAC, verify***
  - Computationally lightweight
  - ***Challenge:*** How do you privately share a key?
- **Asymmetric cryptography**
  - Everyone has a ***public*** *key* that everyone else knows; and a paired ***secret*** *key* that is private
  - Public key can ***encrypt***; only secret key can ***decrypt***
  - Secret key can ***sign/MAC***, public key can ***verify***
  - Computationally expensive
  - ***Challenge:*** How do you validate a public key?

Mid-way Summary

- **Where are public keys from?**
  - One solution: keys for ***Certificate Authorities*** *a priori* known by browser, OS, etc.
- **Where are shared keys from?**
  - In person exchange, snail mail, etc.
  - If we have verifiable public/private keys: ***key exchange*** protocol generates a shared key for symmetric cryptography

How cryptosystems work today

- Layered approach:
  - Cryptographic primitives, like block ciphers, stream ciphers, hash functions, and one-way trapdoor permutations
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication
- Public algorithms (Kerckhoff’s Principle)
- Security proofs based on assumptions (not this course)



OCB auth. encryption

CBC-MAC auth.

CBC encryption

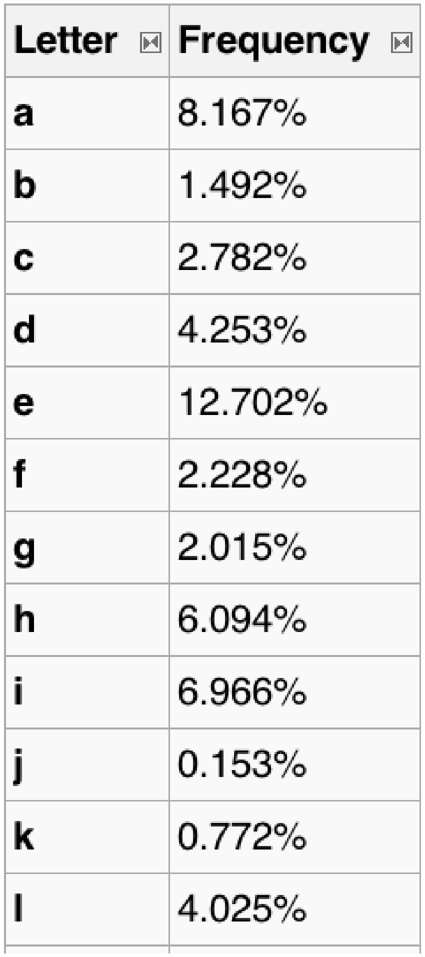
CTR encryption

HMAC auth.

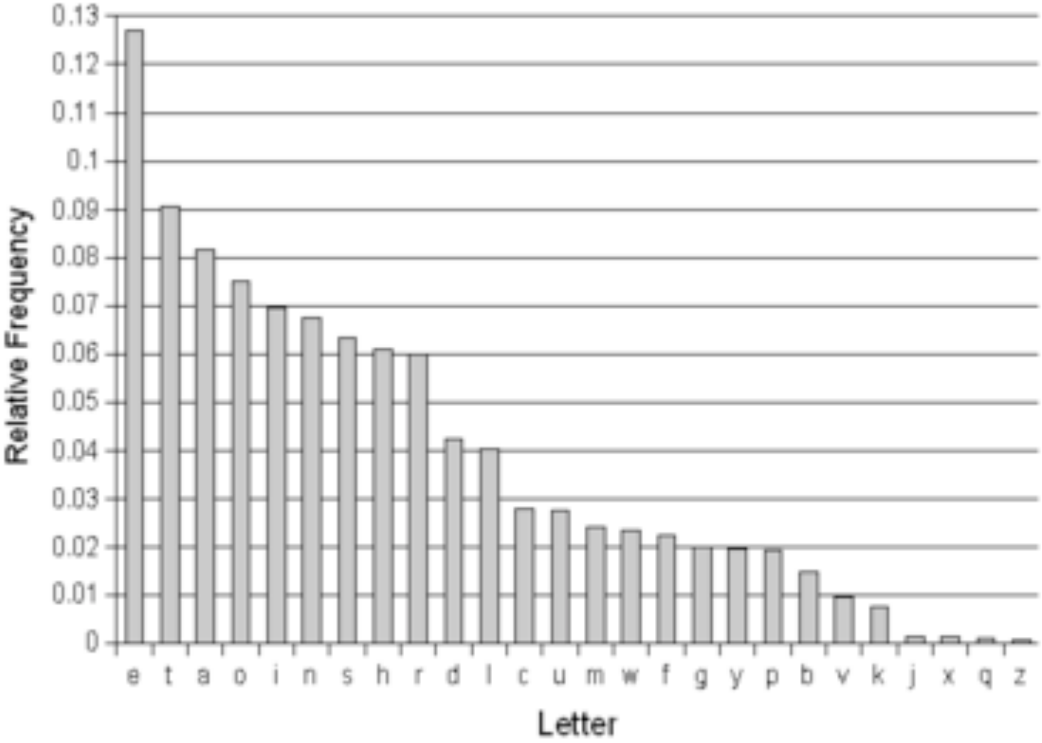
block cipher

hash functions

“Old Days” Cryptanalysis and Probabilities



Letter	Frequency
a	8.167%
b	1.492%
c	2.782%
d	4.253%
e	12.702%
f	2.228%
g	2.015%
h	6.094%
i	6.966%
j	0.153%
k	0.772%
l	4.025%



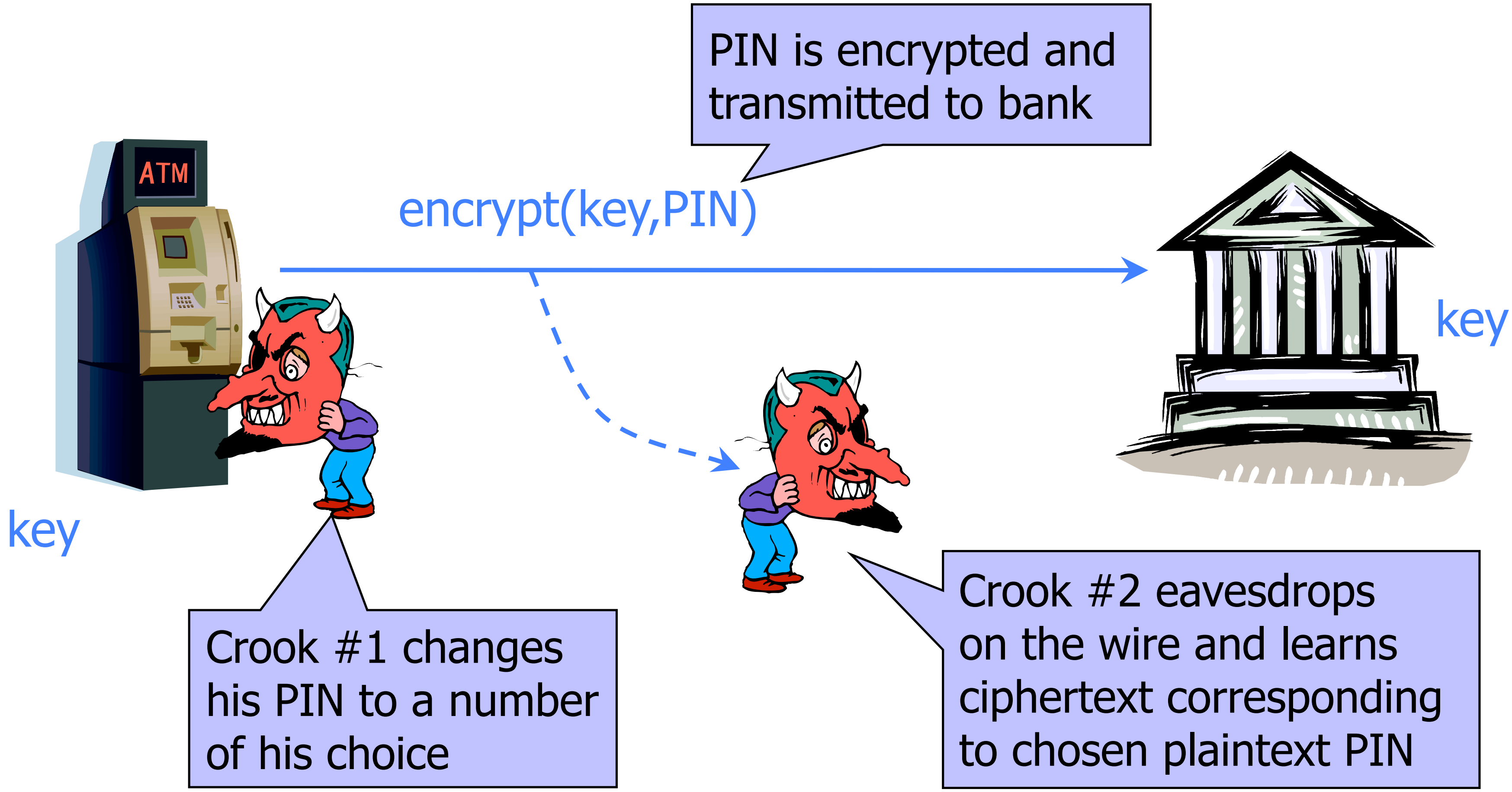
From http://en.wikipedia.org/wiki/Letter_frequencies

Attack Scenarios for Encryption

- Ciphertext-Only
- Known Plaintext
- Chosen Plaintext
- Chosen Ciphertext (and Chosen Plaintext)

- (General advice: Target strongest level of privacy possible -- even if not clear why -- for extra “safety”)

Chosen-Plaintext Attack



PIN is encrypted and transmitted to bank

encrypt(key,PIN)

key

key

Crook #1 changes his PIN to a number of his choice

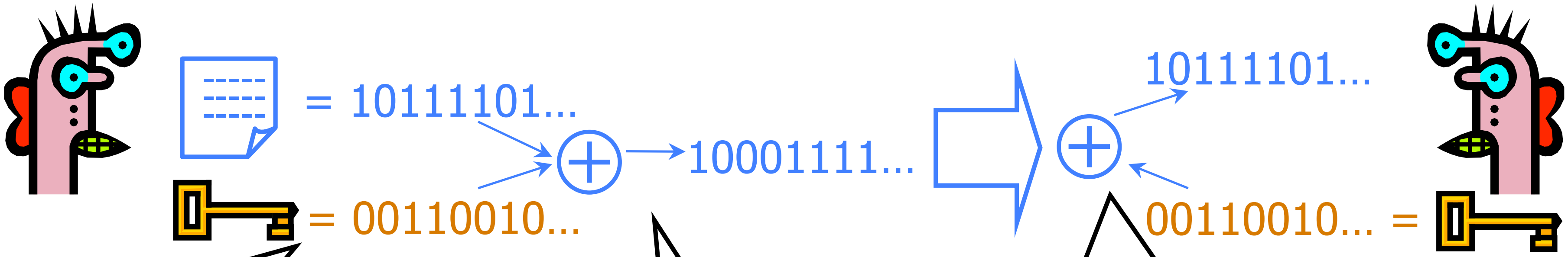
Crook #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

… repeat for any PIN value

Attack Scenarios for Integrity

- What do you think these scenarios should be?

One-Time Pad



= 10111101…

= 00110010…

⊕ → 10001111…

⊕

10111101…

00110010… =

Key is a random bit sequence
as long as the plaintext

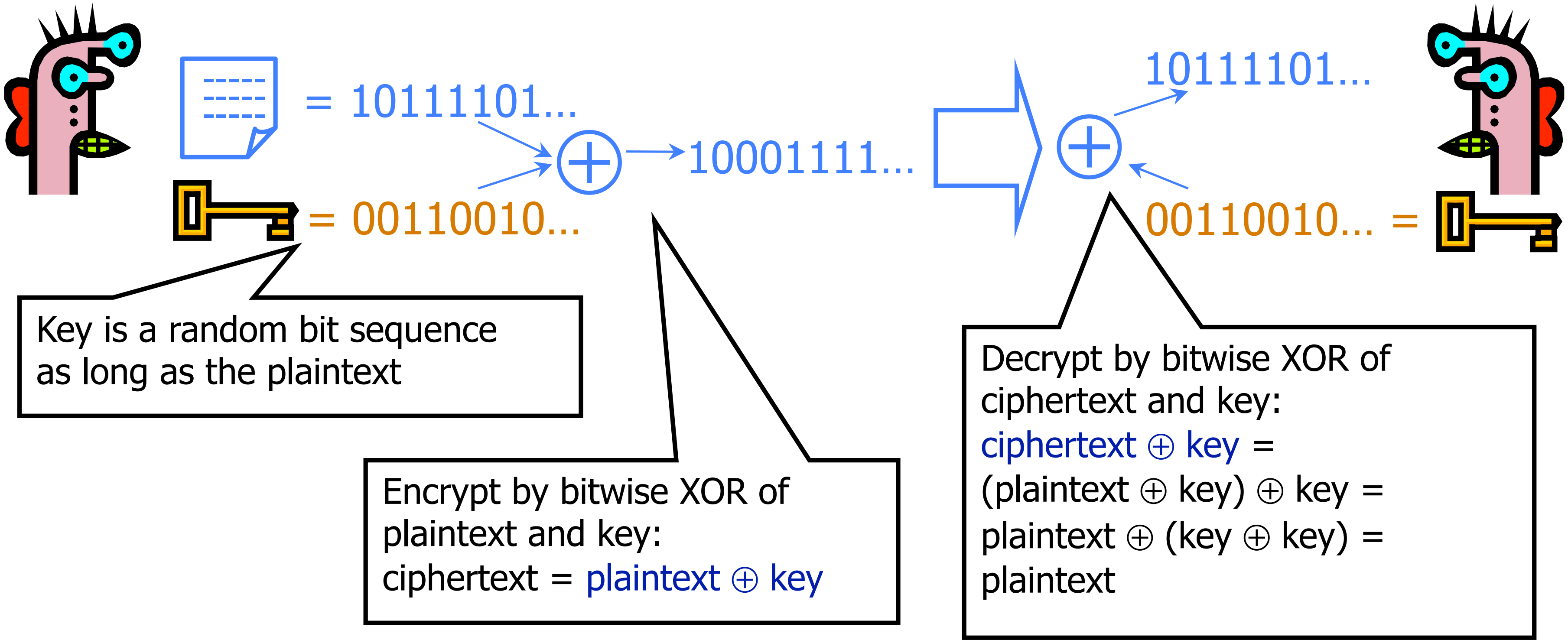
Encrypt by bitwise XOR of
plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of
ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Advantages of One-Time Pad

- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - …as long as the key sequence is truly random
    - True randomness is expensive to obtain in large quantities
  - …as long as each key is same length as plaintext
    - But how does the sender communicate the key to receiver?

Disadvantages



Key is a random bit sequence as long as the plaintext

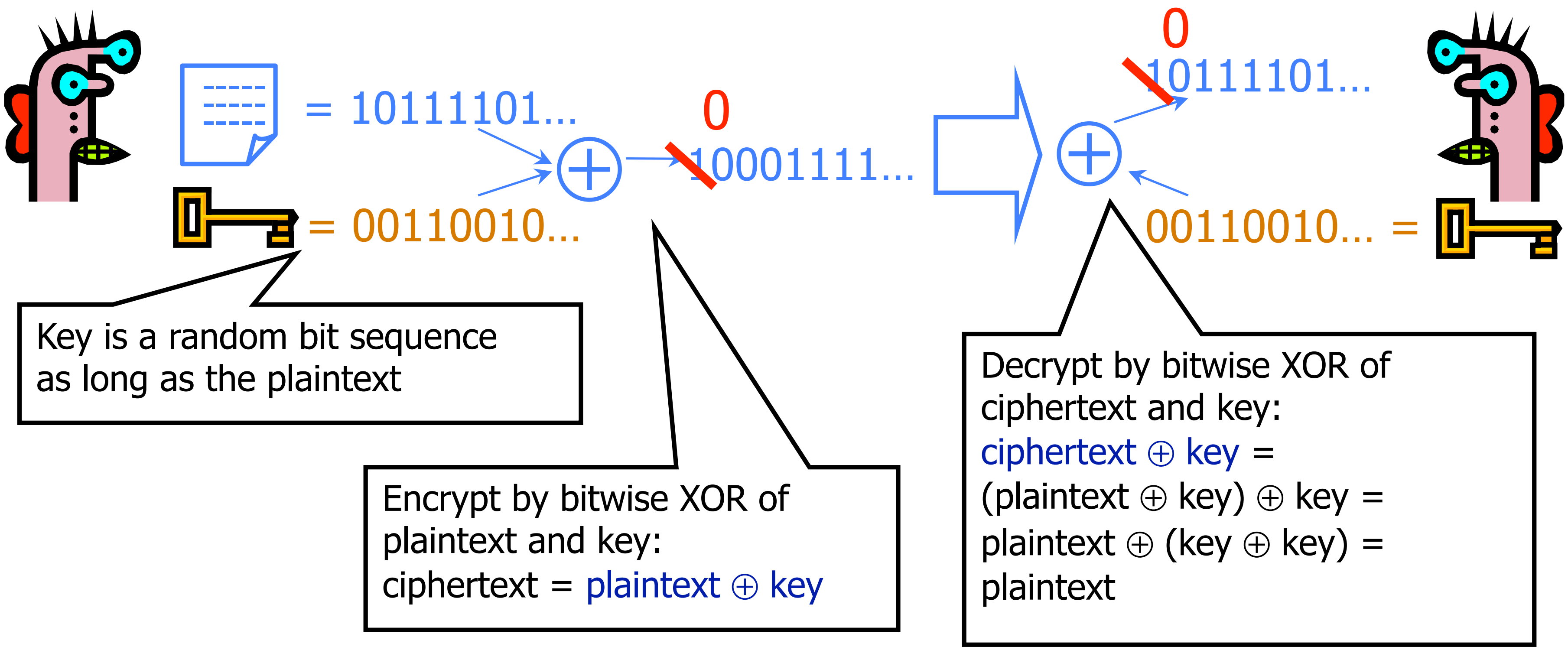
Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #1: Keys as long as messages.

- Impractical in most scenarios
- Still used by intelligence communities

Disadvantages



Key is a random bit sequence as long as the plaintext

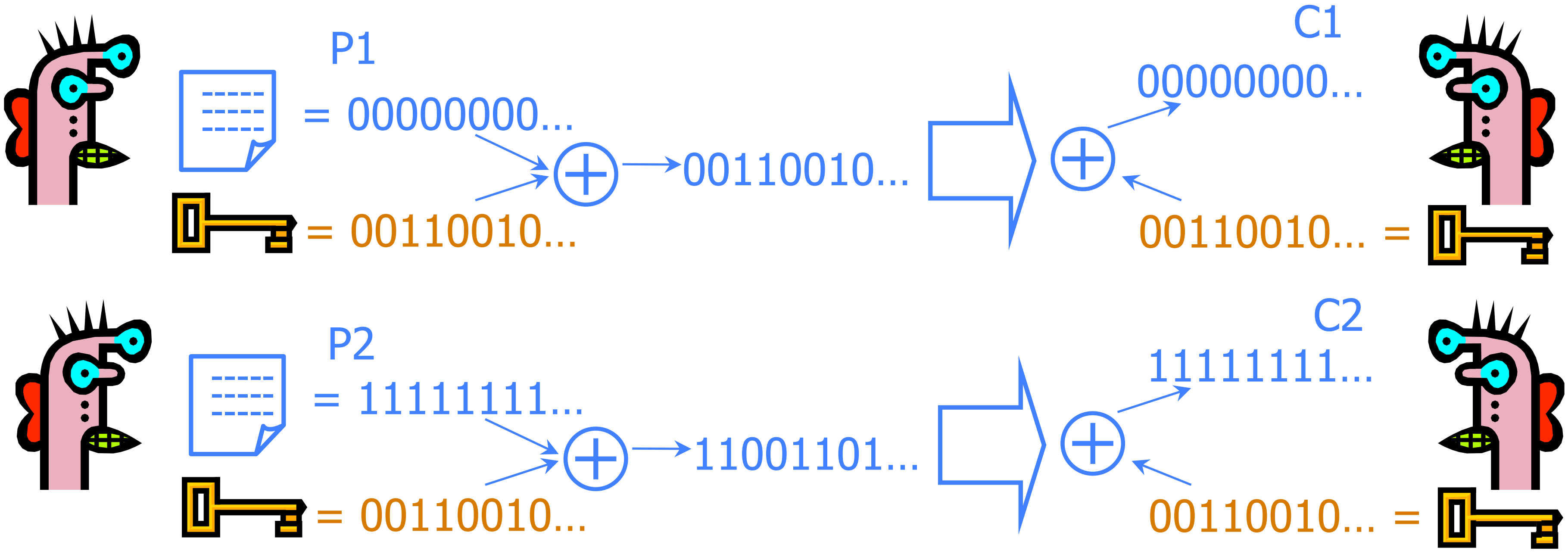
Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #2: No integrity protection

Disadvantages

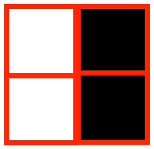
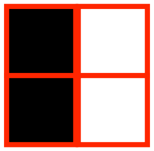
Disadvantage #3: Keys cannot be reused



Learn relationship between plaintexts:

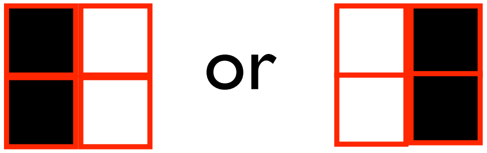
C1⊕C2 = (P1⊕K)⊕(P2⊕K) = (P1⊕P2)⊕(K⊕K) = P1⊕P2

Visual Cryptography

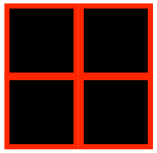
- Generate a random bitmap
- Encode 0 as: 
- Encode 1 as: 

Visual Cryptography

- Take a black and white bitmap image
- For a white pixel, send the same as the mask

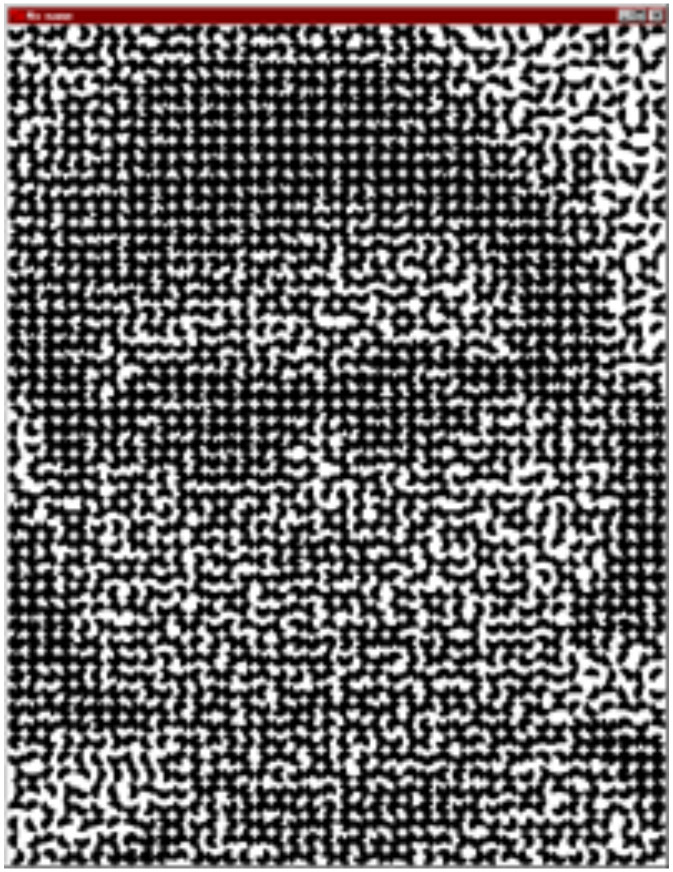


- For a black pixel, send the opposite of the mask



See also http://www.cs.washington.edu/homes/yoshi/cs4hs/cse-vc.html

Visual Cryptography



- http://www.cl.cam.ac.uk/~fms27/vck/face.gif

See also http://www.cs.washington.edu/homes/yoshi/cs4hs/cse-vc.html

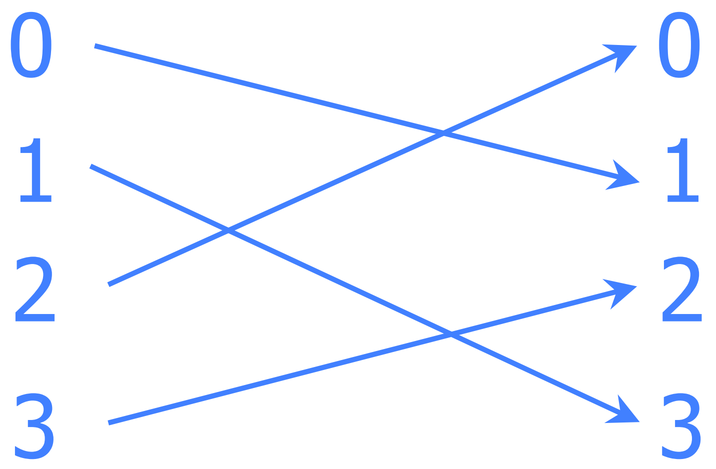
Reducing Keysize

- What do we do when we can’t pre-share huge keys?
  - When OTP is unrealistic

- We use special cryptographic primitives
  - Single key can be reused (with some restrictions)
  - But no longer provable secure (in the sense of the OTP)

- Examples: Block ciphers, stream ciphers

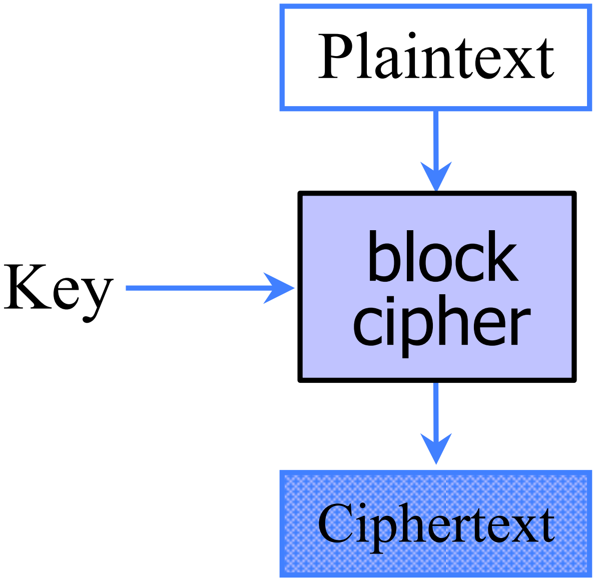
Background: Permutation



- For N-bit input, $2^N!$ possible permutations
- Idea for how to use a keyed permutation: split plaintext into blocks; for each block use secret key to pick a permutation
    - Without the key, permutation should "look random"

Block Ciphers

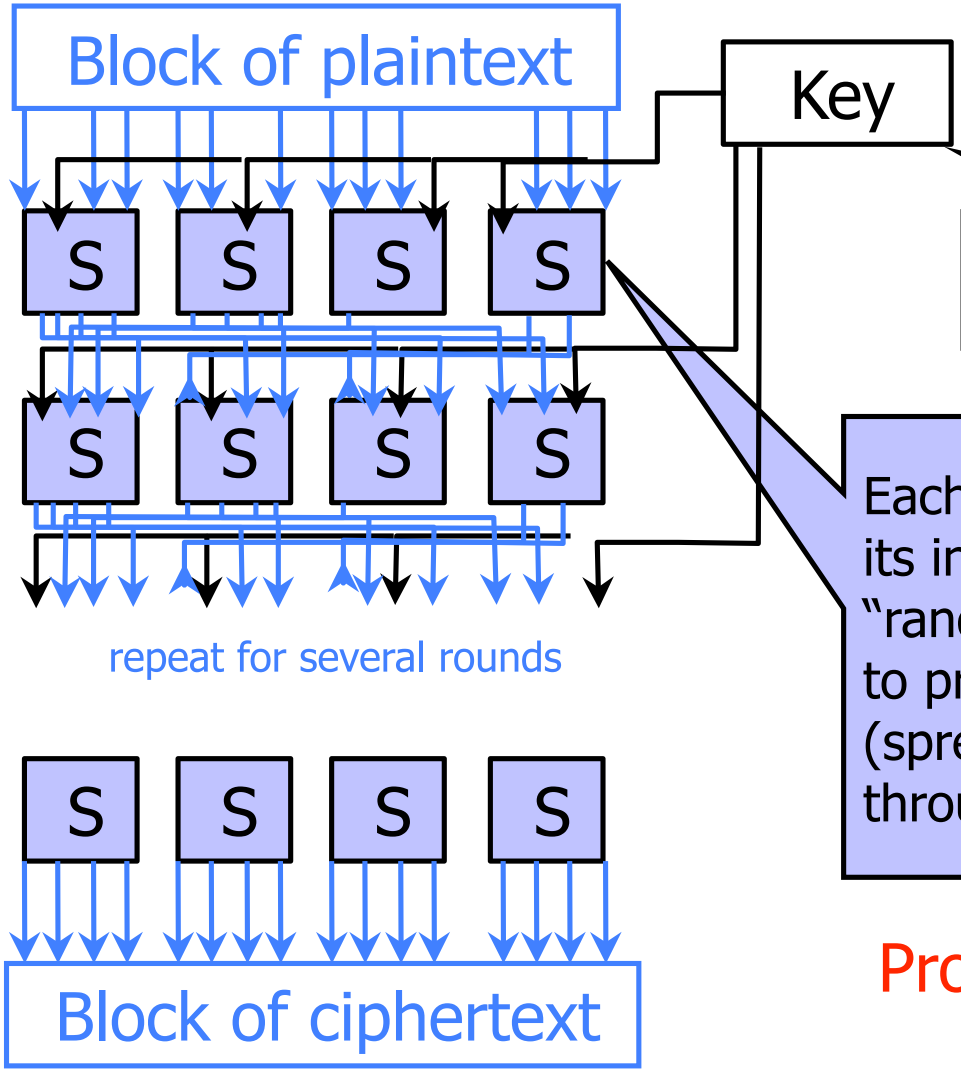
- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Each key defines a different permutation
  - Same key is reused for each block (can use short keys)



Block Cipher Security

- Result should look like a random permutation on the inputs
  - Recall: not just shuffling bits. N-bit block cipher permutes over 2^N inputs.
- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

Block Cipher Operation (Simplified)

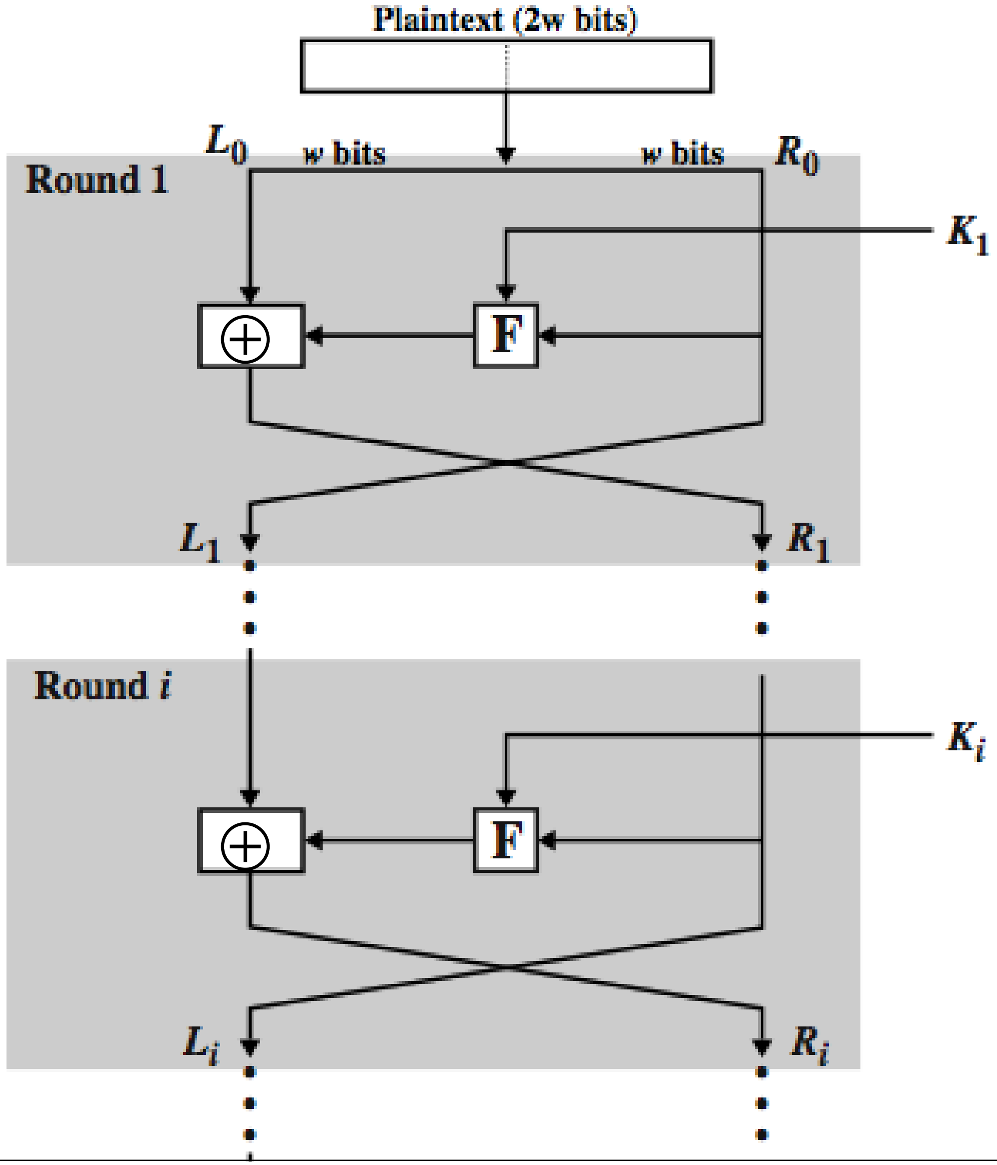


Add some secret key bits to provide confusion

Each S-box transforms its input bits in a “random-looking” way to provide diffusion (spread plaintext bits throughout ciphertext)

Procedure must be reversible (for decryption)

Feistel Structure (Stallings Fig 2.2)

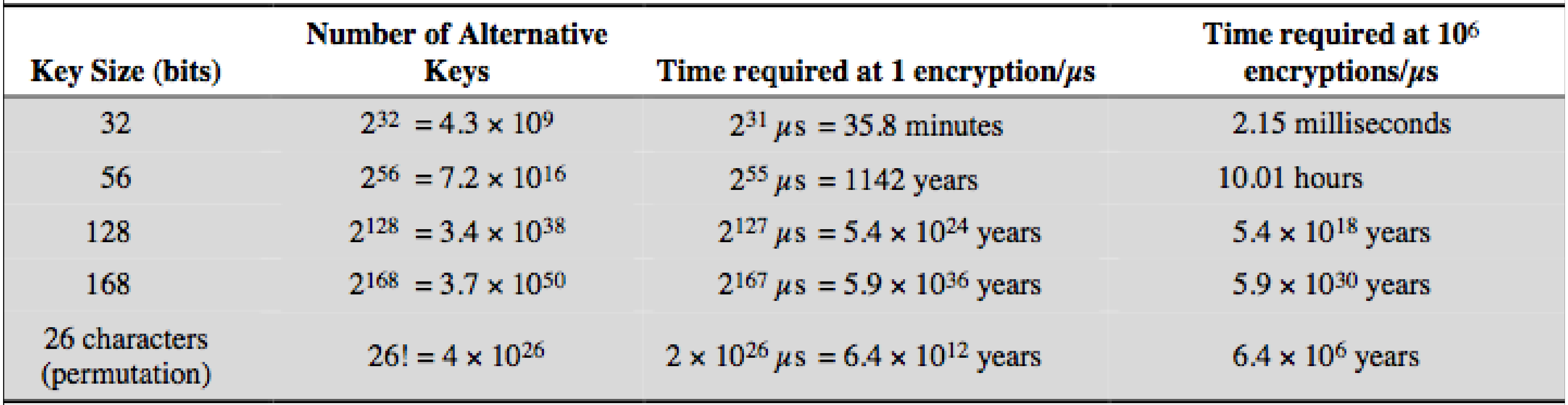


DES

- Feistel structure
  - "Ladder" structure: split input in half, put one half through the round and XOR with the other half
  - Theoretical support: After 3 random rounds, ciphertext indistinguishable from a random permutation if internal F function is a pseudorandom function (Luby & Rackoff)
- DES: Data Encryption Standard
  - Feistel structure
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

DES and 56 bit keys (Stallings Tab 2.2)

- 56 bit keys are quite short



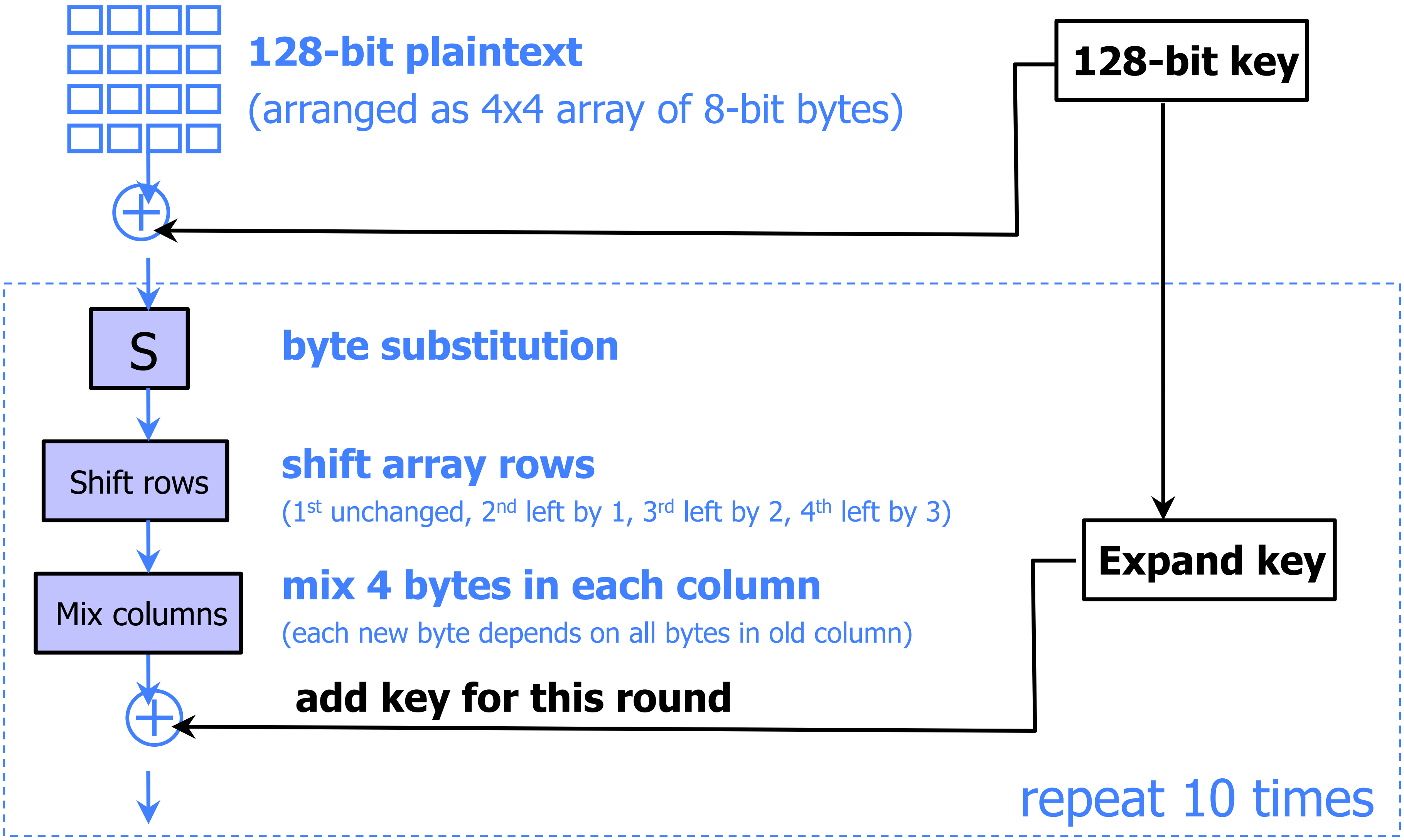
Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- 1999: EFF DES Crack + distibuted machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

Advanced Encryption Standard (AES)

- New federal standard as of 2001
- Based on the Rijndael algorithm
- 128-bit blocks, keys can be 128, 192 or 256 bits
- Unlike DES, does not use Feistel structure
  - The entire block is processed during each round
- Design uses some very nice mathematics

Basic Structure of Rijndael



128-bit plaintext

(arranged as 4x4 array of 8-bit bytes)

128-bit key

S

byte substitution

Shift rows

shift array rows

(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

Expand key

Mix columns

mix 4 bytes in each column

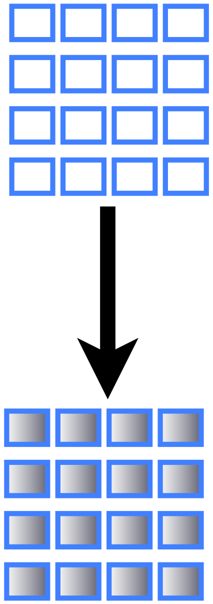
(each new byte depends on all bytes in old column)

add key for this round

repeat 10 times

Encrypting a Large Message

- So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size



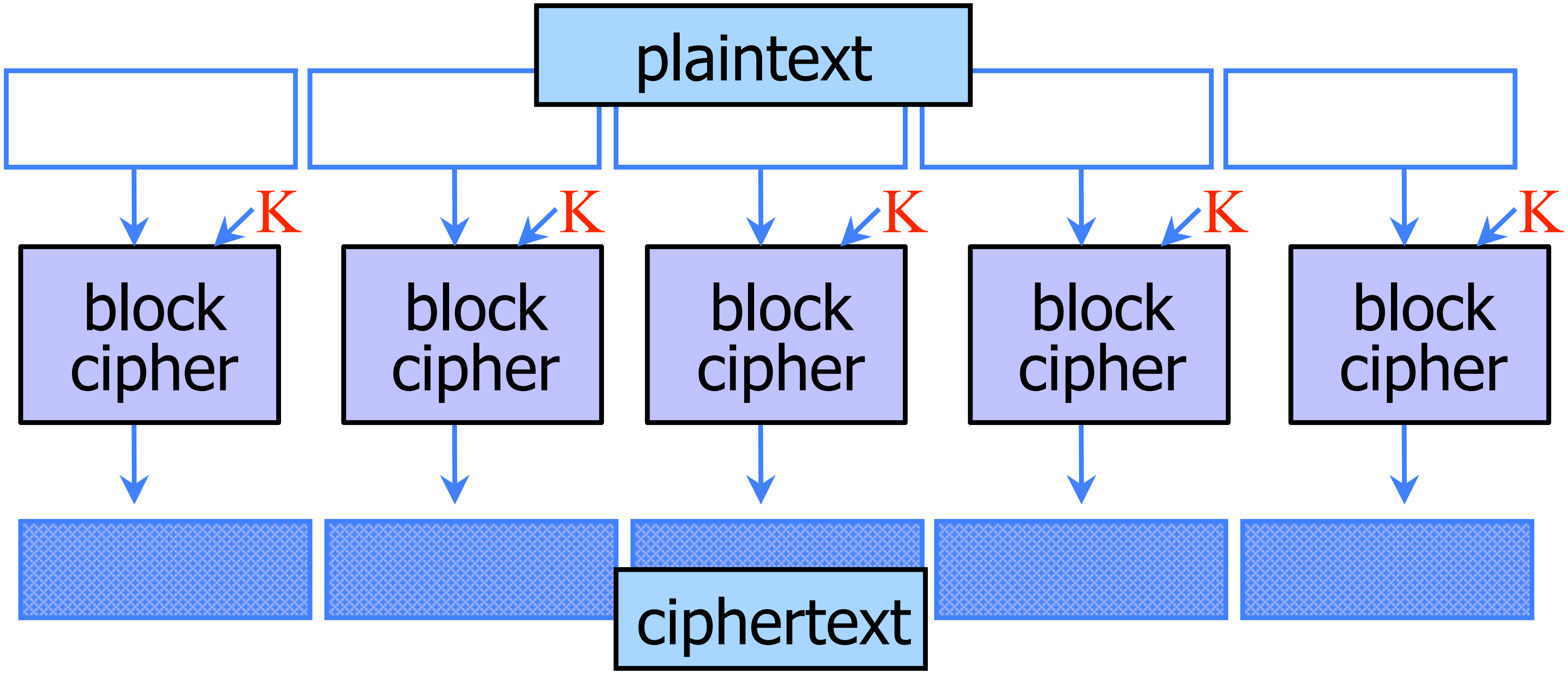
128-bit plaintext

(arranged as 4x4 array of 8-bit bytes)

128-bit ciphertext

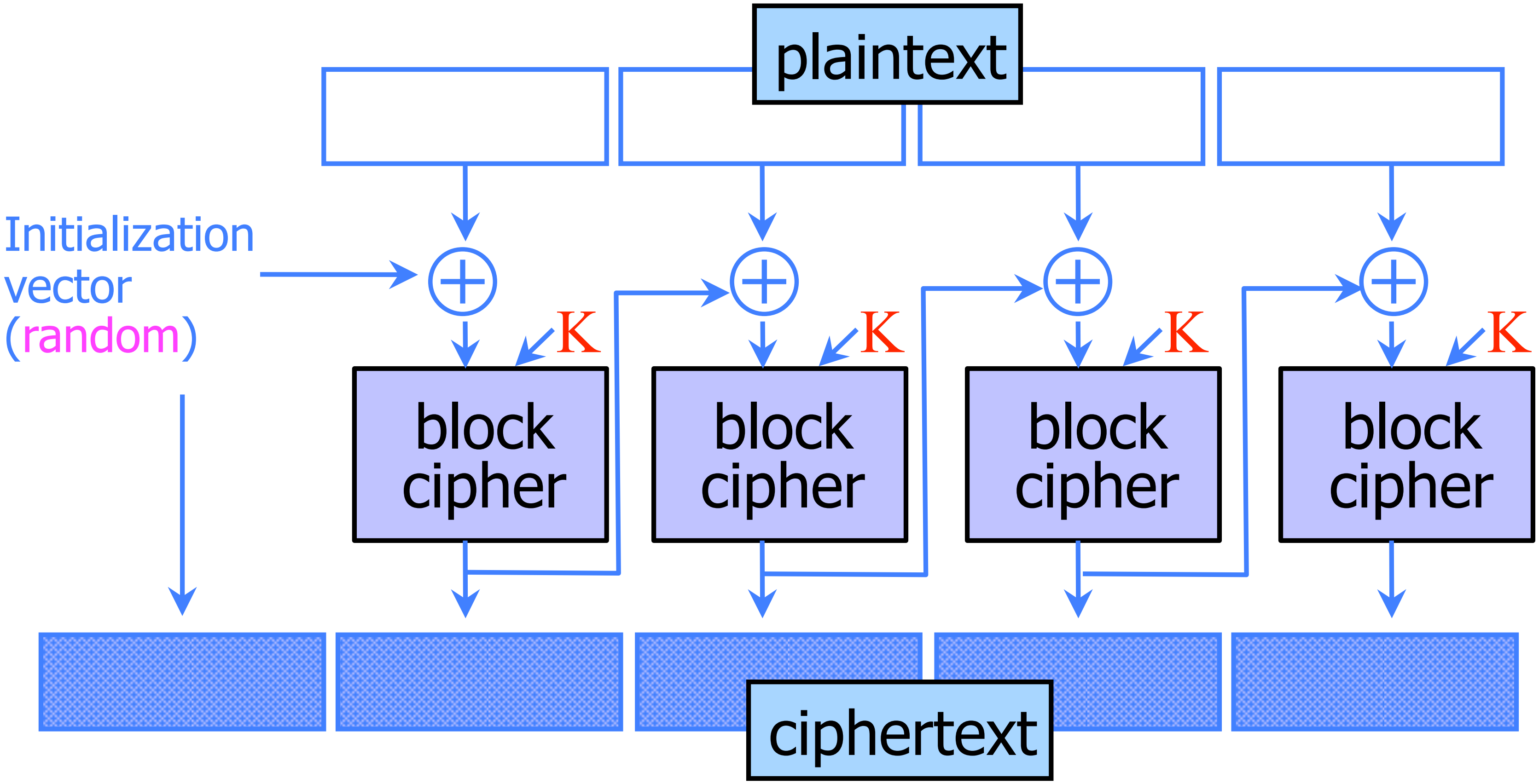
- What should we do?

Electronic Code Book (ECB) Mode



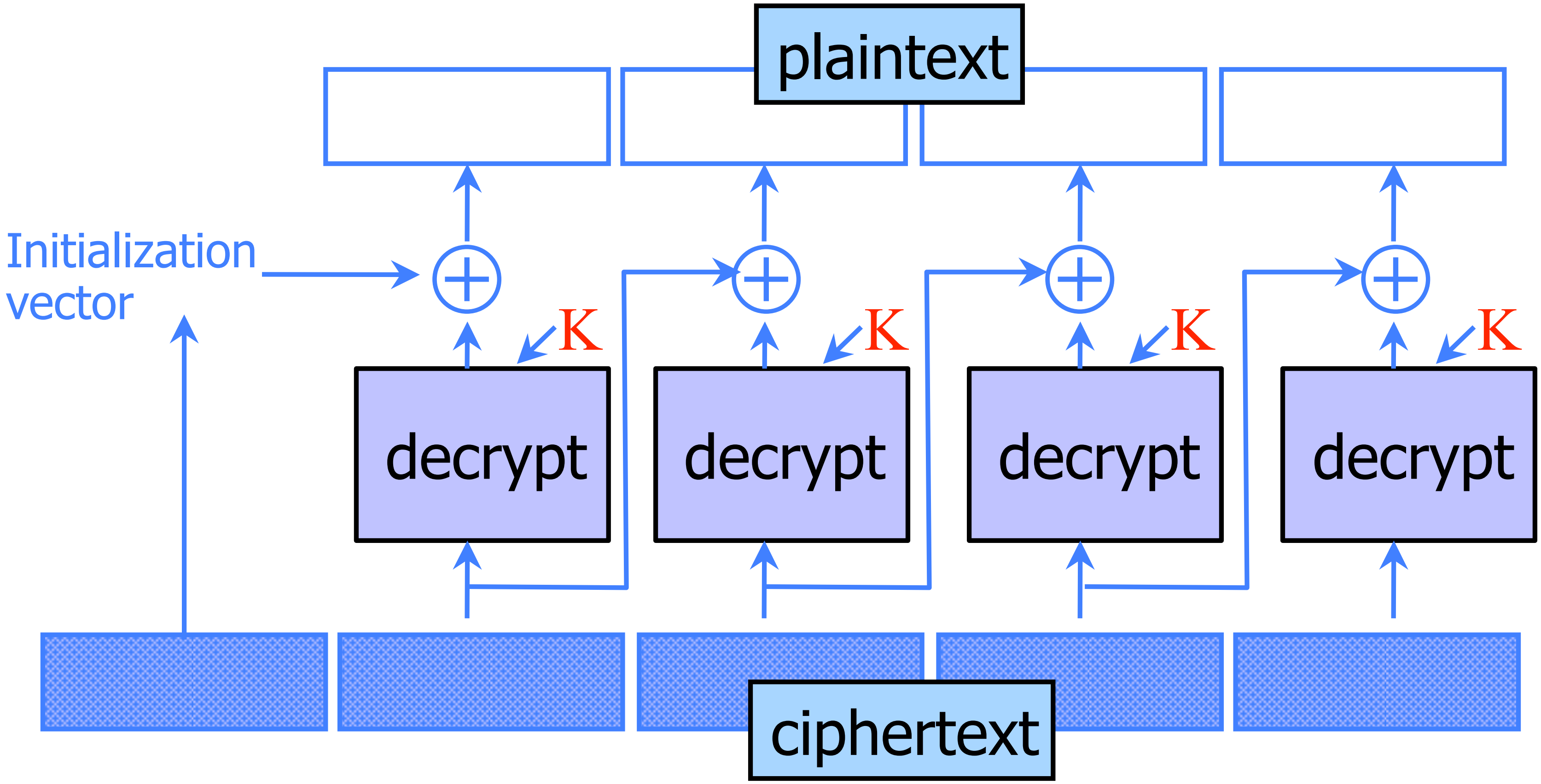
- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Cipher Block Chaining (CBC) Mode: Encryption



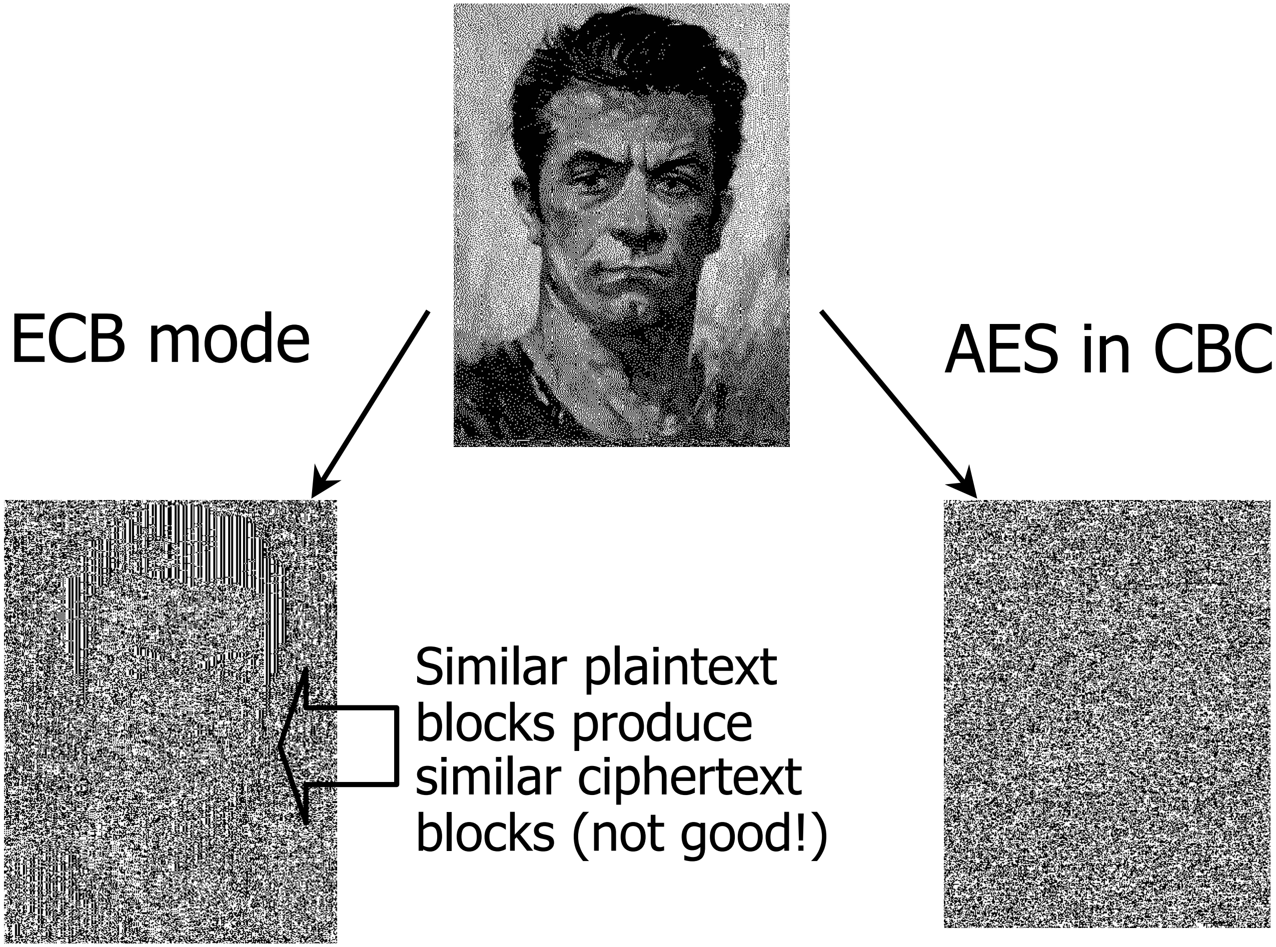
- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
    - Still does not guarantee integrity

CBC Mode: Decryption



ECB vs. CBC

[Picture due to Bart Preneel]



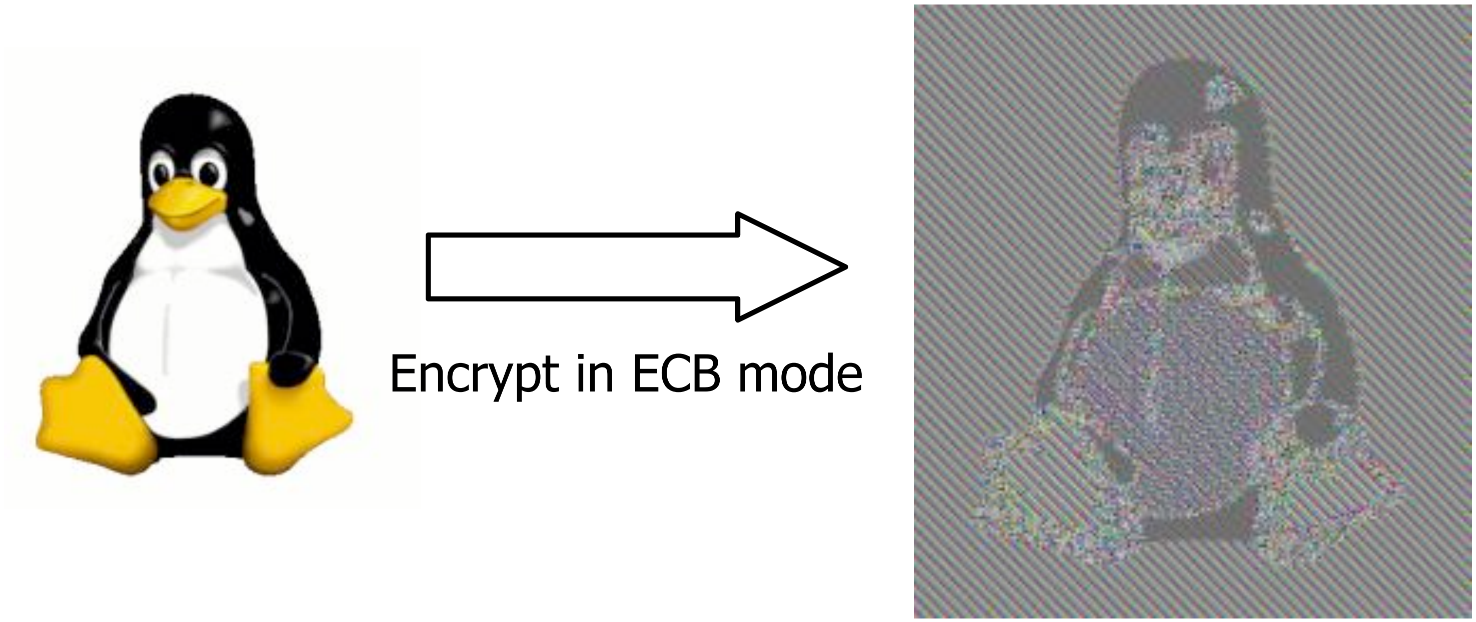
AES in ECB mode

AES in CBC mode

Similar plaintext
blocks produce
similar ciphertext
blocks (not good!)

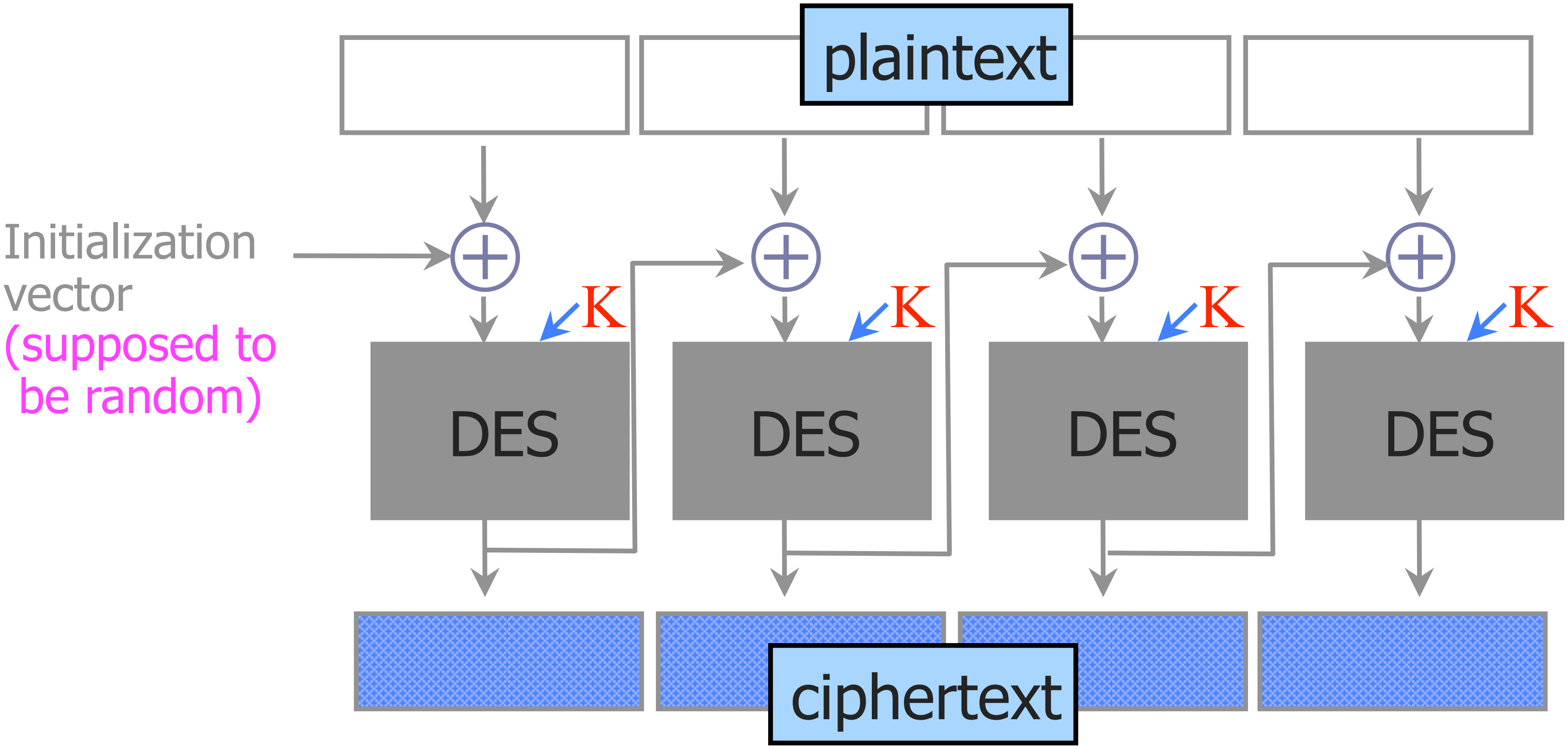
Information Leakage in ECB Mode

[Wikipedia]



Encrypt in ECB mode

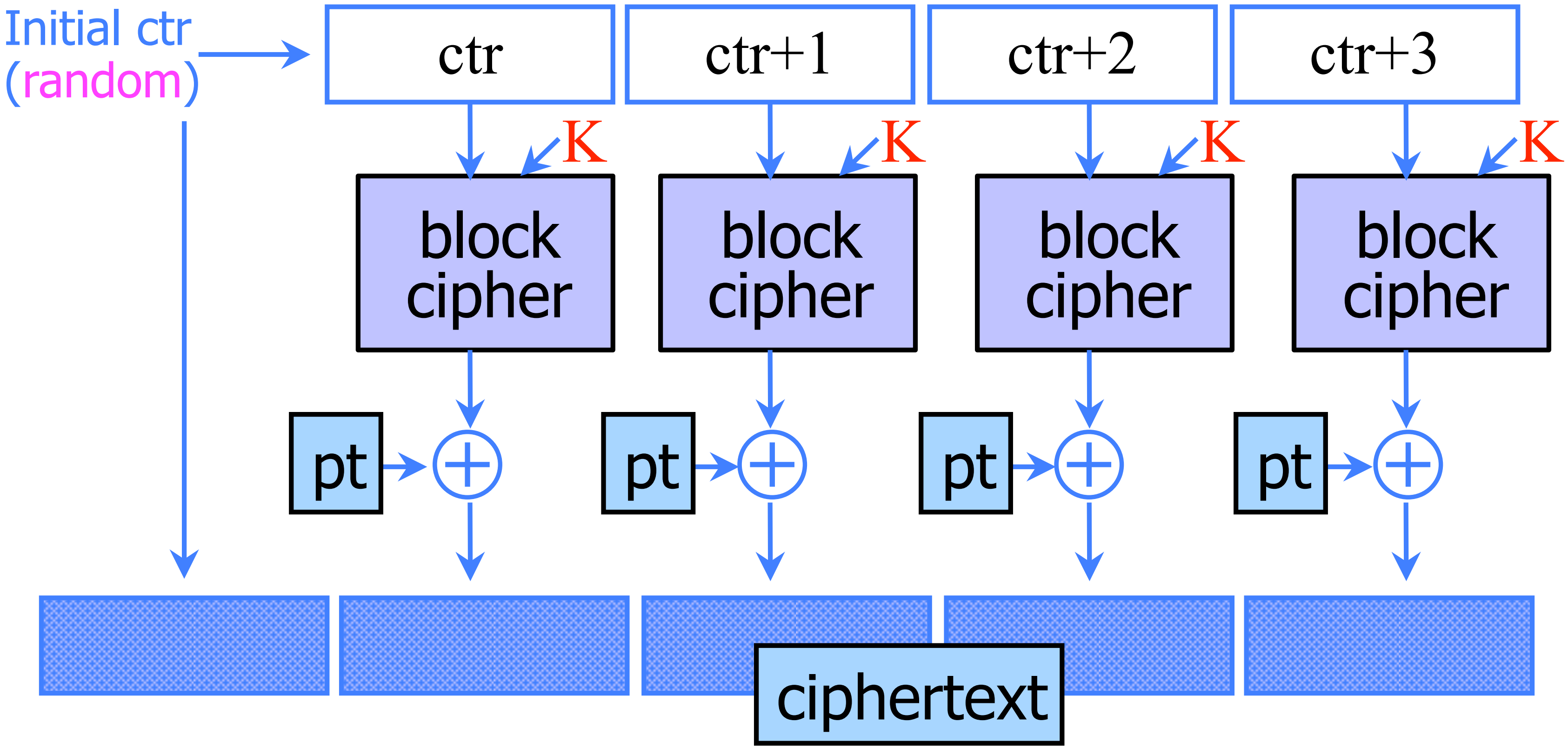
CBC and Electronic Voting



Found in the source code for Diebold voting machines:

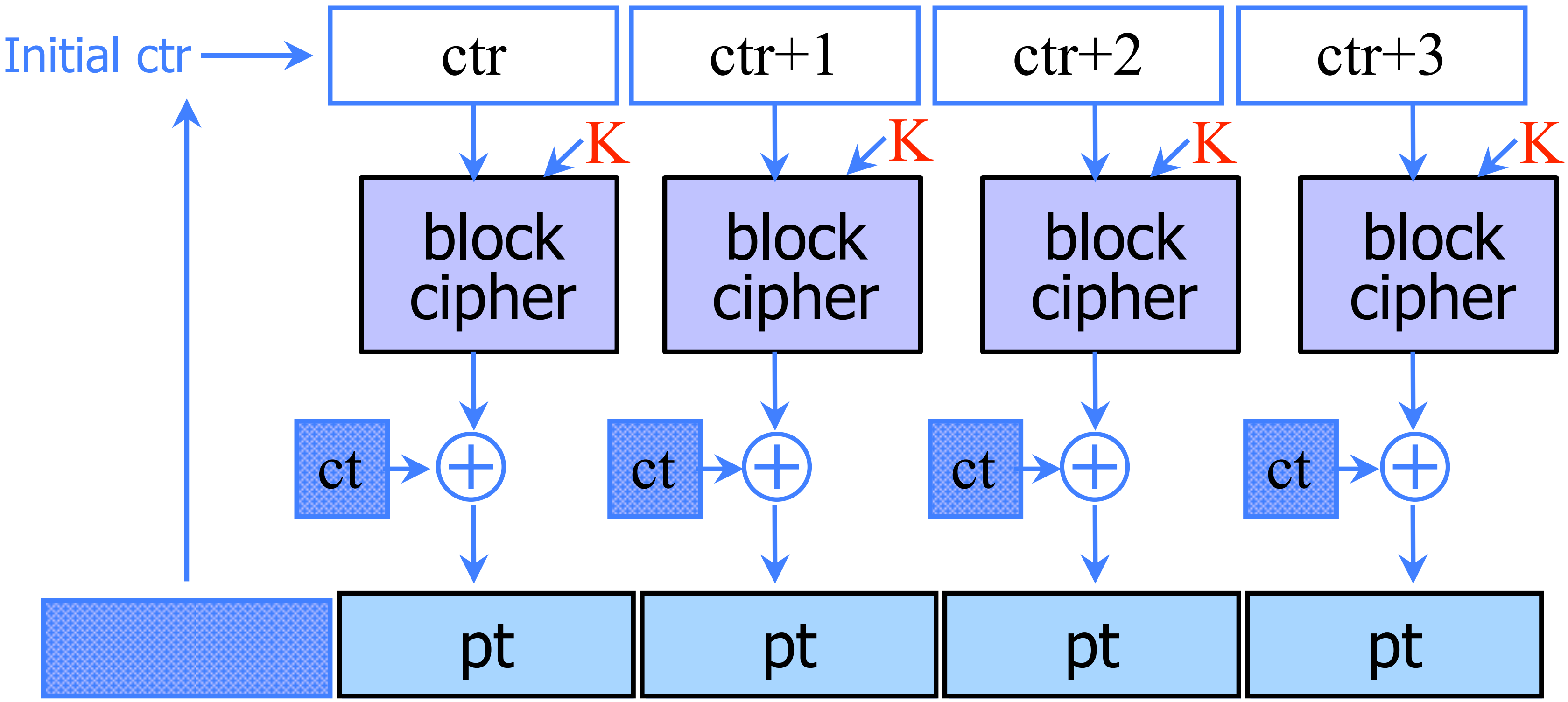
```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
          totalSize, DESKEY, NULL, DES_ENCRYPT)
```

Counter (CTR) Mode: Encryption



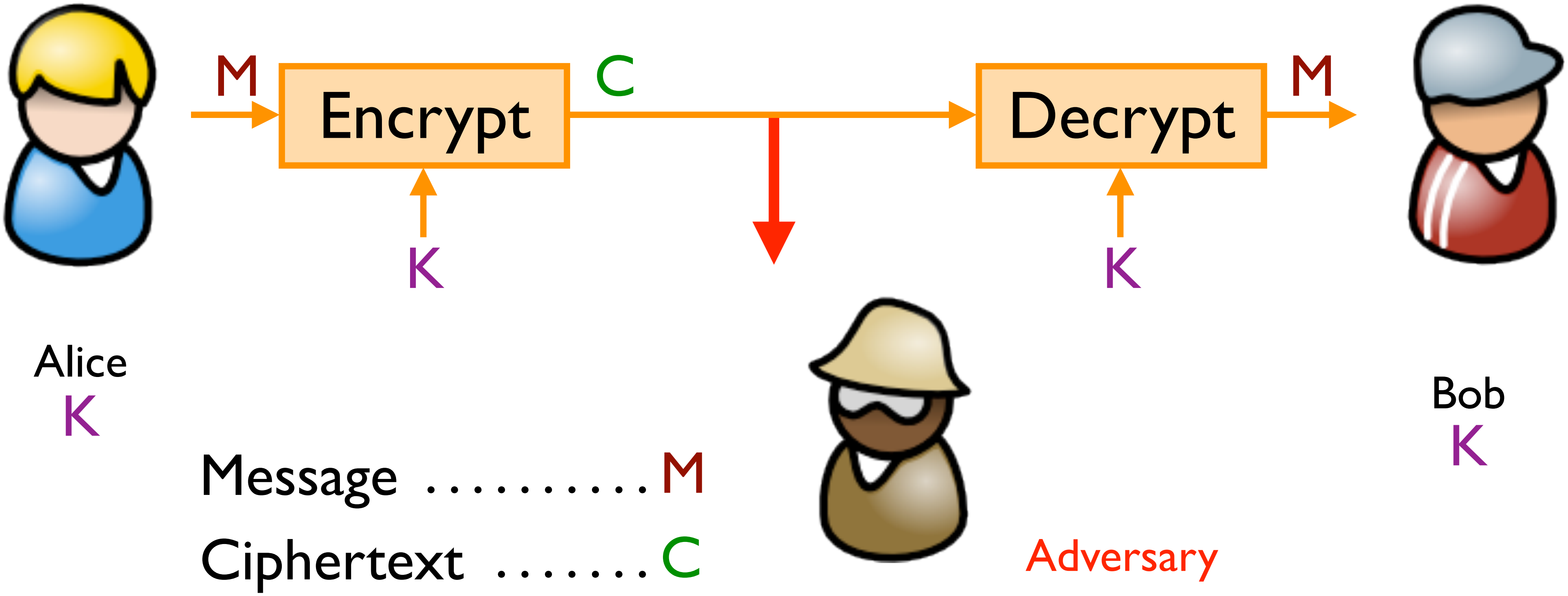
- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity
- Fragile if ctr repeats

CTR Mode: Decryption



Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.



Message . . . . M

Ciphertext C

When Is an Encryption Scheme "Secure"?

- Hard to recover the key?
  - What if attacker can learn plaintext without learning the key?
- Hard to recover plaintext from ciphertext?
  - What if attacker learns some bits or some function of bits?
- Fixed mapping from plaintexts to ciphertexts?
  - What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?
  - Implication: encryption must be randomized or stateful

How Can a Cipher Be Attacked?

- Assume that the attacker knows the encryption algorithm and wants to learn information about some ciphertext
- Main question: what else does attacker know?
  - Depends on the application in which cipher is used!
- Ciphertext-only attack
- Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of his choice
- Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext except the target
  - Sometimes very realistic model

Defining Security (Not Required)

- Attacker does not know the key
- He chooses as many plaintexts as he wants, and learns the corresponding ciphertexts
- When ready, he picks two plaintexts M_0 and M_1
  - He is even allowed to pick plaintexts for which he previously learned ciphertexts!
- He receives either a ciphertext of M_0, or a ciphertext of M_1
- He wins if he guesses correctly which one it is

Defining Security (Not Required)

- Idea: attacker should not be able to learn even a single bit of the encrypted plaintext
- Define Enc(M_0,M_1,b) to be a function that returns encrypted M_b

  0 or 1

  - Given two plaintexts, Enc returns a ciphertext of one or the other depending on the value of bit b
  - Think of Enc as a magic box that computes ciphertexts on attacker's demand. He can obtain a ciphertext of any plaintext M by submitting M_0=M_1=M, or he can try to learn even more by submitting $M_0 \neq M_1$.
- Attacker's goal is to learn just one bit b

Chosen-Plaintext Security (Not Required)

- Consider two experiments (A is the attacker)

Experiment 0

A interacts with Enc(-,-,0)
and outputs bit d

Experiment 1

A interacts with Enc(-,-,1)
and outputs bit d

  - Identical except for the value of the secret bit
  - d is attacker's guess of the secret bit

- Attacker's advantage is defined as

If A "knows" secret bit, he should be able to make his output depend on it

| Prob(A outputs 1 in Exp0) - Prob(A outputs 1 in Exp1)) |

- Encryption scheme is chosen-plaintext secure if this advantage is negligible for any efficient A

“Simple” Example (Not Required)

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

$C_1 \leftarrow$ Enc(X,Y,b); $C_2 \leftarrow$ Enc(Y,Y,b);

If $C_1=C_2$ then b=1 else say b=0

- The advantage of this attacker A is 1

Prob(A outputs 1 if b=0)=0 Prob(A outputs 1 if b=1)=1

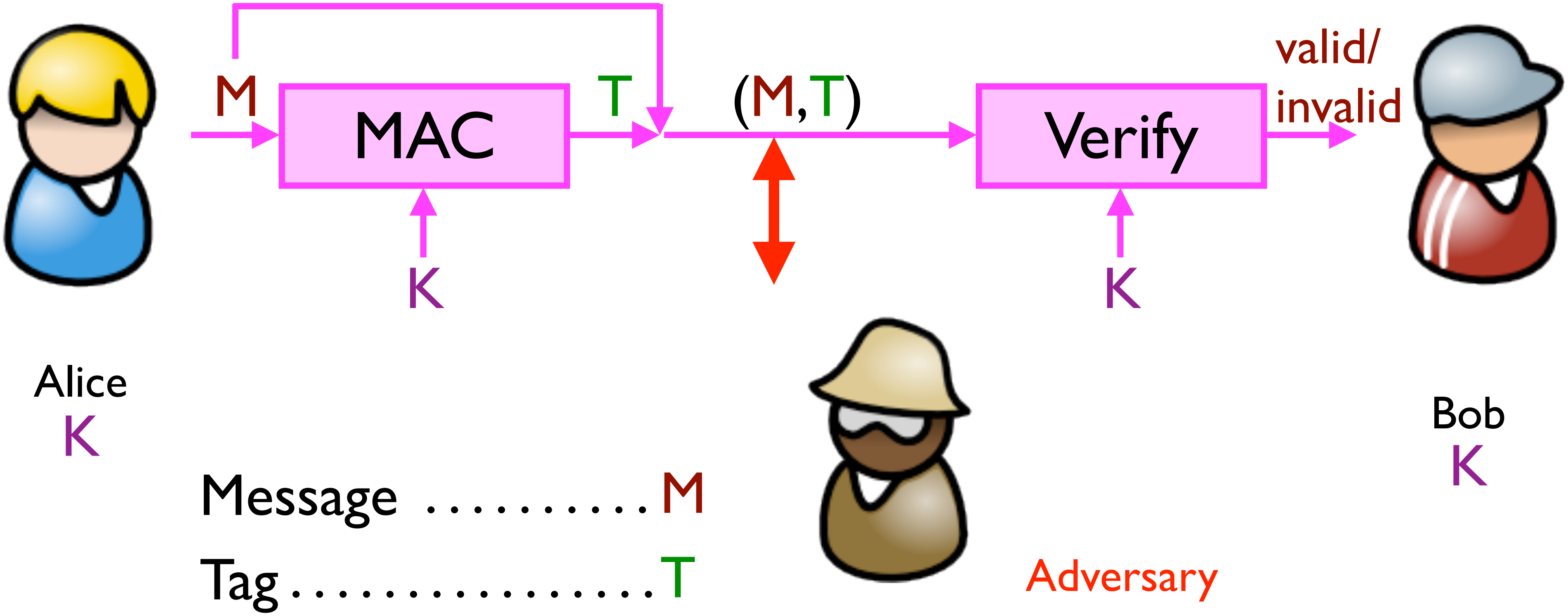
Why Hide Everything?

- Leaking even a little bit of information about the plaintext can be disastrous
- Electronic voting
  - 2 candidates on the ballot (1 bit to encode the vote)
  - If ciphertext leaks the parity bit of the encrypted plaintext, eavesdropper learns the entire vote
- Also, want a strong definition, that implies other definitions (like not being able to obtain key)

Achieving Integrity (Symmetric)

Message authentication schemes: A tool for protecting integrity.

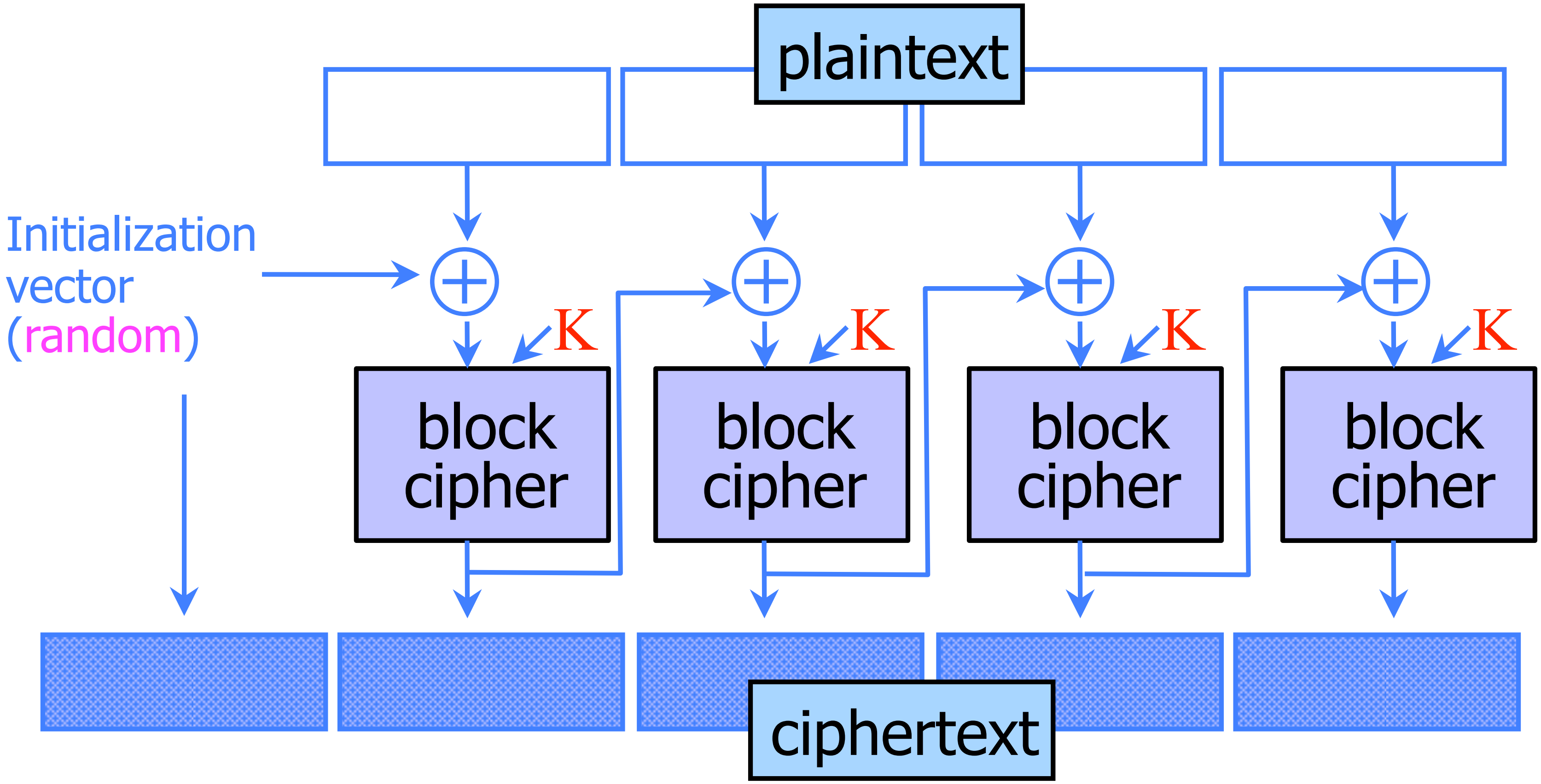
(Also called message authentication codes or MACs.)



Message . . . . M

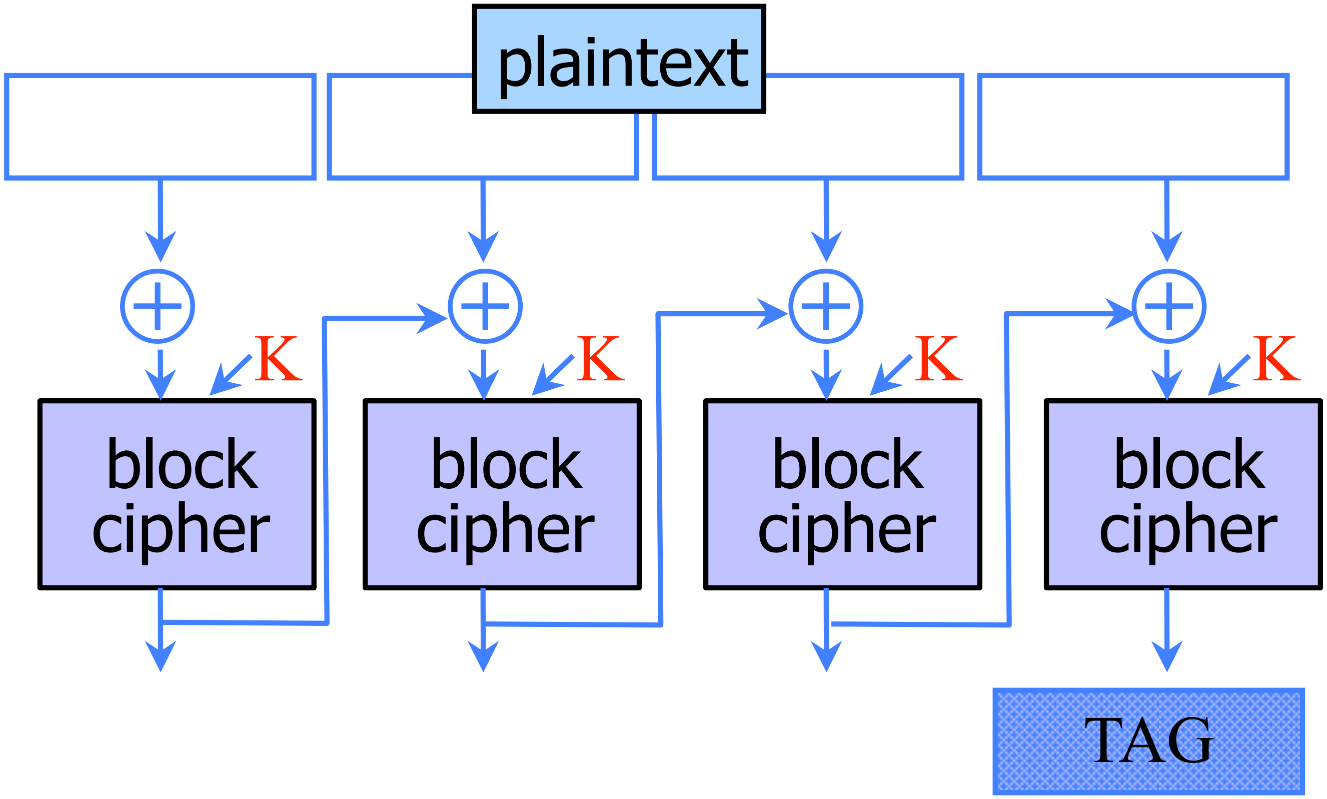
Tag . . . . T

CBC Mode: Encryption



- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

CBC-MAC



- Not secure when system may MAC messages of different lengths.
  - NIST recommends a derivative called CMAC (not required)

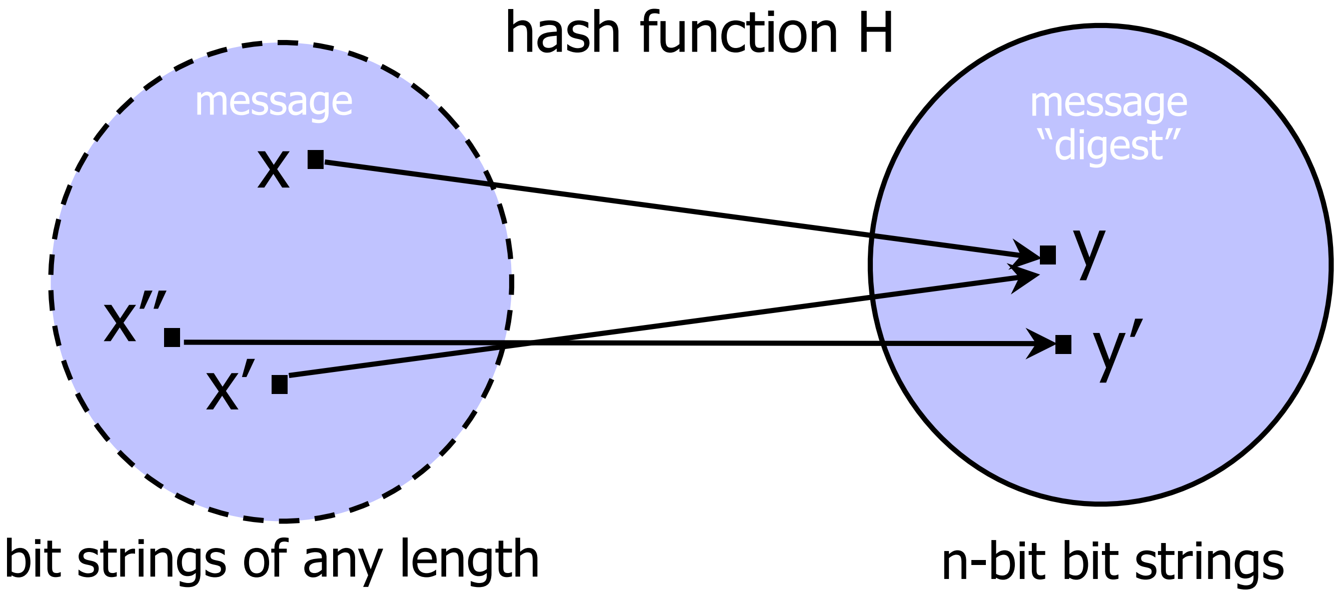
Birthday attacks

- Are there two people in the first 1/3 of this classroom that have the same birthday?
  - Yes?
  - No?

Birthday attacks

- Why is this important for cryptography?
  - 365 days in a year (366 some years)
    - Pick one person. To find another person with same birthday would take on the order of 365/2 = 182.5 people
    - Expect "collision" -- two people with same birthday -- with a room of only 23 people
    - For simplicity, approximate when we expect a collision as the square root of 365.
  - 2^{128} different 128-bit keys
    - Pick one key at random. To exhaustively search for this key requires trying on average 2^{127} keys.
    - Expect a "collision" after selecting approximately 2^{64} random keys.
    - 64 bits of security against collision attacks, not 128 bits.

Broad Class of Hash Functions



- H is a lossy compression function
  - Collisions: h(x)=h(x') for distinct inputs x, x'
  - Result of hashing should "look random" (make this precise later)
    - Intuition: half of digest bits are "1"; any bit in digest is "1" half the time
- Cryptographic hash function needs a few properties…

One-Way

- Intuition: hash should be hard to invert
  - "Preimage resistance"
  - Let $h(x')=y\in\{0,1\}^n$ for a random x'
  - Given y, it should be hard to find any x such that h(x)=y
- How hard?
  - Brute-force: try every possible x, see if h(x)=y
  - SHA-1 (common hash function) has 160-bit output
    - Expect to try 2^{159} inputs before finding one that hashes to y.

Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′? $1/2^n$
  - How many pairs would we need to look at before finding the first collision? $O(2^n)$
  - How many pairs x,x′ total? Choose(t,2)=t(t-1)/2 ~ $O(t^2)$
  - What is t? $2^{n/2}$

One-Way vs. Collision Resistance

- One-wayness does not imply collision resistance
  - Suppose g is one-way
  - Define h(x) as g(x’) where x’ is x except the last bit
    - h is one-way (to invert h, must invert g)
    - Collisions for h are easy to find: for any x, h(x0)=h(x1)
- Collision resistance does not imply one-wayness
  - Suppose g is collision-resistant
  - Define h(x) to be 0x if x is n-bit long, 1g(x) otherwise
    - Collisions for h are hard to find: if y starts with 0, then there are no collisions, if y starts with 1, then must find collisions in g
    - h is not one way: half of all y’s (those whose first bit is 0) are easy to invert (how?); random y is invertible with probab. 1/2

Weak Collision Resistance

- Given randomly chosen x, hard to find x′ such that h(x)=h(x′)
  - Attacker must find collision for a specific x. By contrast, to break collision resistance it is enough to find any collision.
  - Brute-force attack requires O(2^n) time
  - AKA second-preimage collision resistance
- Weak collision resistance does not imply collision resistance

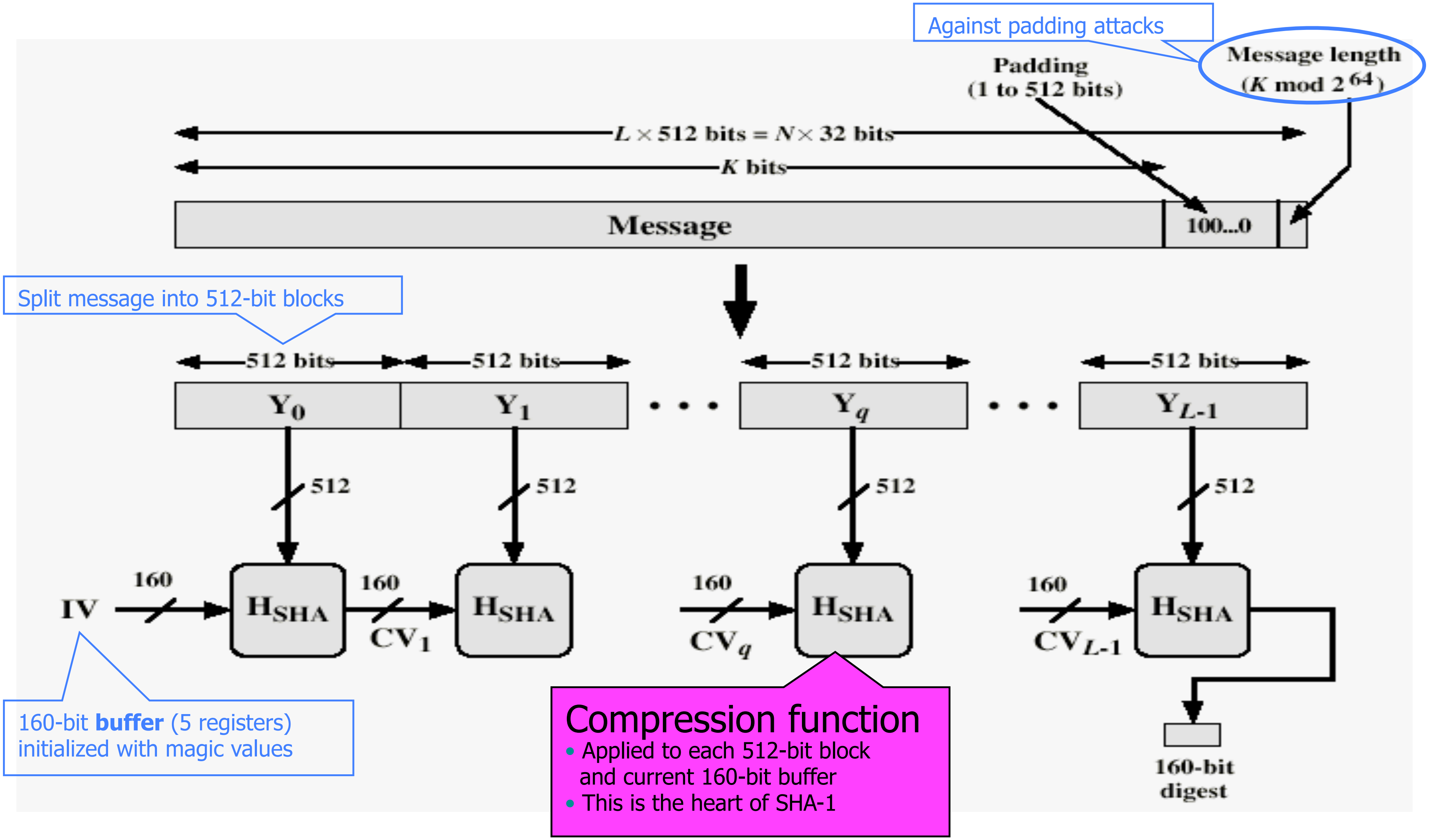
Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - Weak collision resistance: hard to recover the/a valid password
- Integrity of software distribution
  - Weak collision resistance (second-preimage resistance)
  - But software images are not really random…
  - Collision resistance if considering malicious developers
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B (this may mean that she needs to hash some randomness with B too)
  - Collision resistance: Alice should not be able to change her mind to bid B′ such that H(B)=H(B′)

Common Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Also recently broken! (Theoretically -- not practical.)
- SHA-256, SHA-512, SHA-224, SHA-384
- SHA-3: Just picked -- not an official standard yet

Basic Structure of SHA-1 (Not Required)



Against padding attacks

Split message into 512-bit blocks

160-bit **buffer** (5 registers)
initialized with magic values

Compression function

- Applied to each 512-bit block and current 160-bit buffer
- This is the heart of SHA-1

How Strong Is SHA-1?

- Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- Brute-force inversion requires 2^{160} ops, birthday attack on collision resistance requires 2^{80} ops
- Some weaknesses, e.g., collisions can be found in 2^{63} ops (2005)

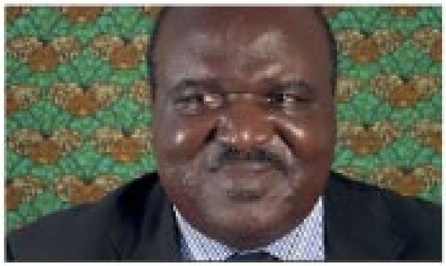
International Criminal Tribunal for Rwanda (Example Application)

- http://www.nytimes.com/2009/01/27/science/27arch.html?_r=1&ref=science

Adama Dieng
CB44-8847-D68D-8CD2-C2F5
22FE-177B-2C30-3549-C211

Angeline Djampou
EA39-EC39-A5D0-314D-04A6
5258-572C-9268-8CB7-6404

Avi Singh
CD69-2CB5-78CB-D8D7-7D81
F9B2-9CEA-5B79-DA4F-3806



Alfred Kwende
C690-FC5A-8EB7-0B83-B99D
2593-608A-F421-BEE4-16B2

Sir Dennis Byron
CA46-BE7A-B8F6-095A-C706
1C60-31E7-F9EA-AF96-E2CE

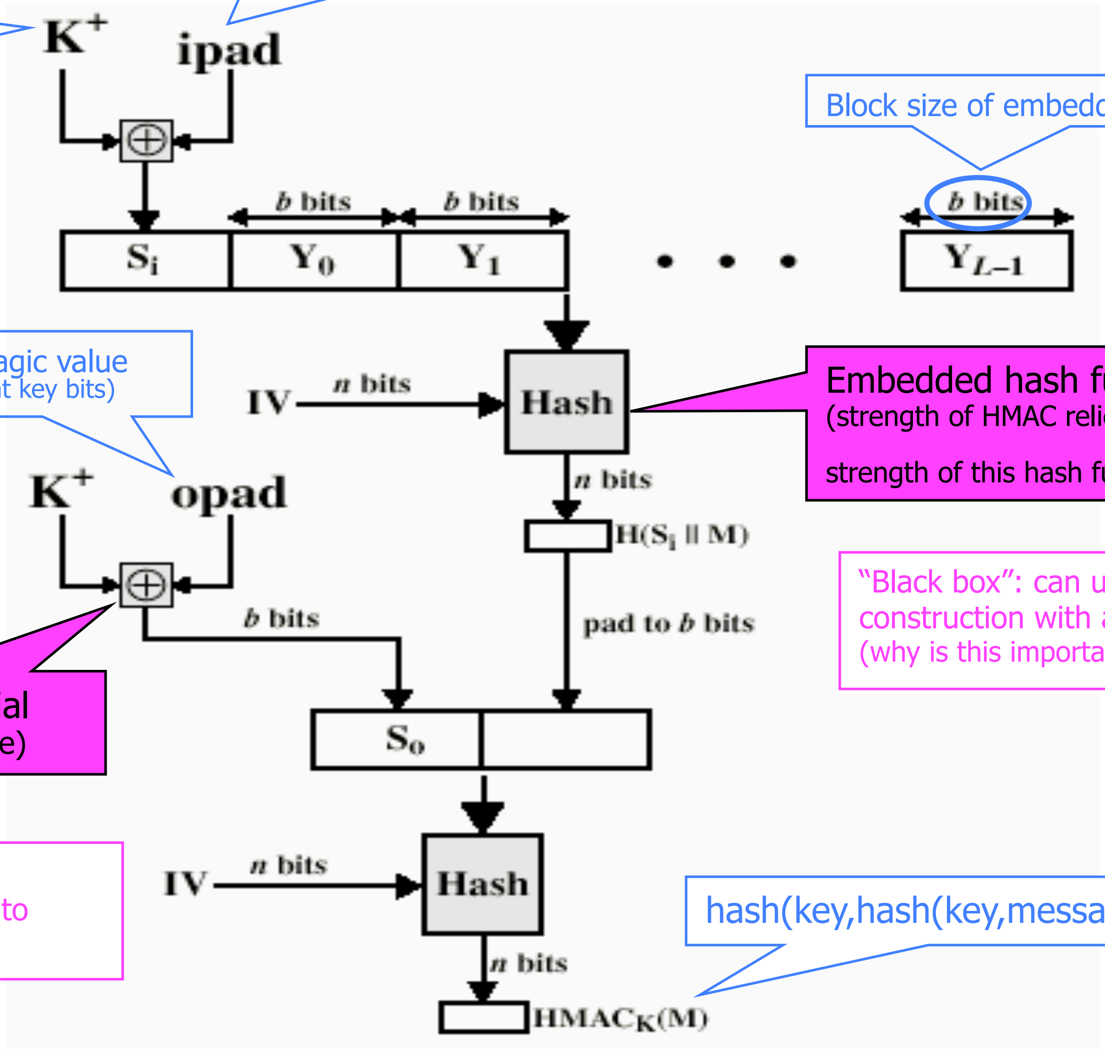
Everard O'Donnell
909F-86AB-C1B8-57A7-9CF6
5BCD-7F5E-F4F6-68CA-70D1

- Credits: Alexei Czeskis, Karl Koscher, Batya Friedman

HMAC

- Construct MAC by applying a cryptographic hash function to message and key
- Invented by Bellare, Canetti, and Krawczyk (1996)
- Mandatory for IP security, also used in SSL/TLS

Structure of HMAC



magic value (flips half of key bits)

Secret key padded to block size

Block size of embedded hash function

another magic value (flips different key bits)

Embedded hash function (strength of HMAC relies on

strength of this hash function)

“Black box”: can use this HMAC construction with any hash function (why is this important?)

“Amplify” key material (get two keys out of one)

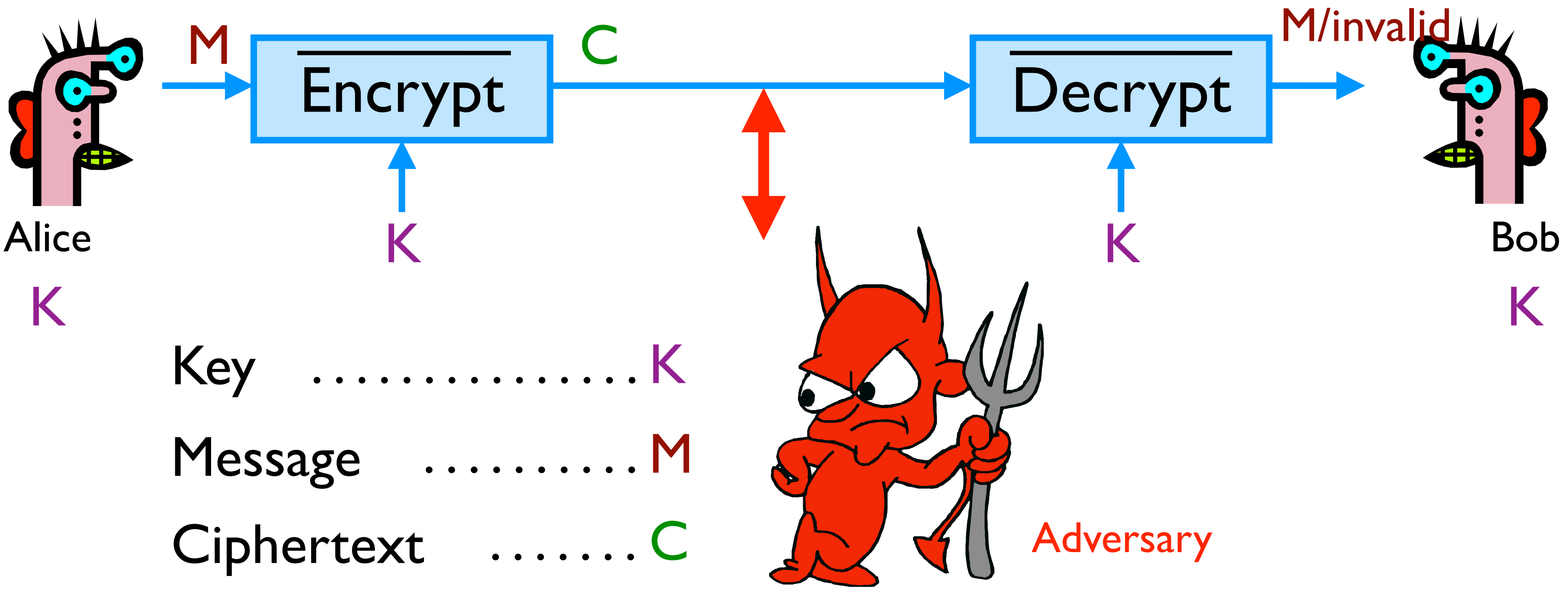
Very common problem: given a small secret, how to derive a lot of new keys?

hash(key,hash(key,message))

Achieving Both Privacy and Integrity

Authenticated encryption scheme

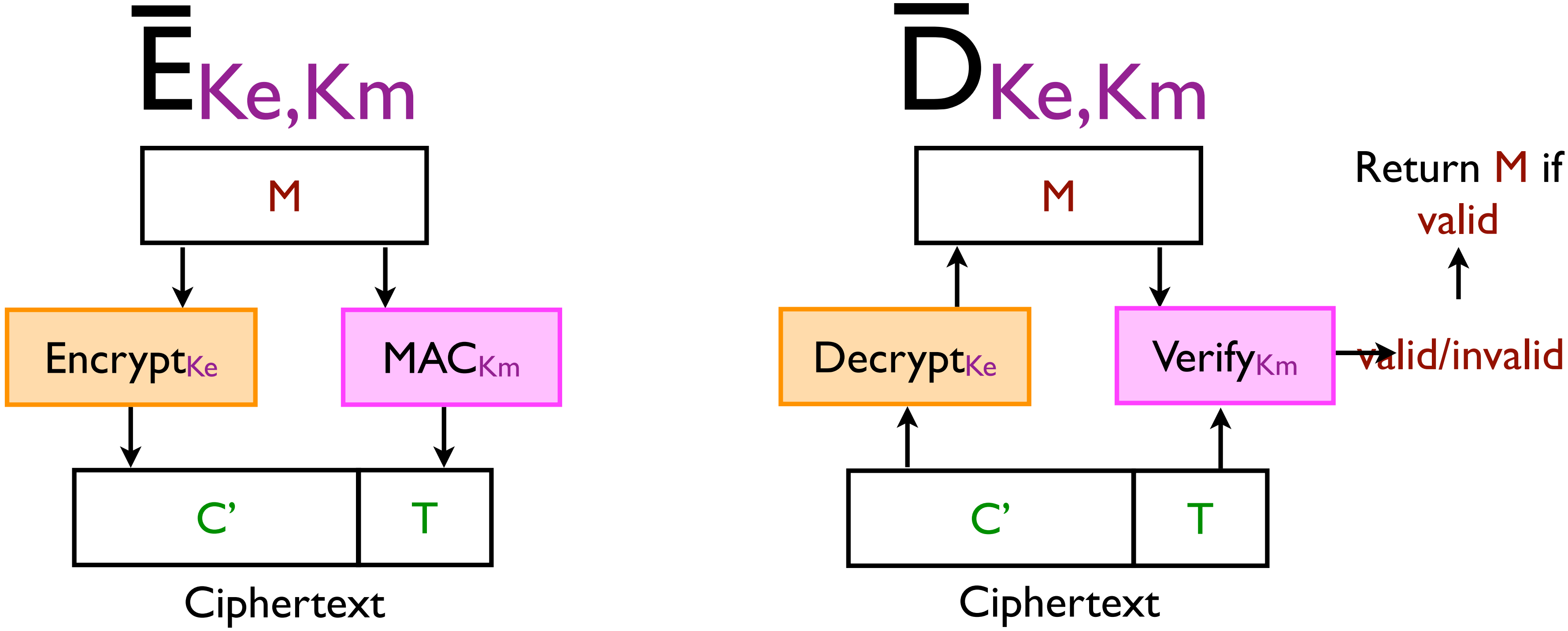
Recall: Often desire both privacy and integrity. (For SSH, SSL, IPsec, etc.)



Key	K
Message	M
Ciphertext	C

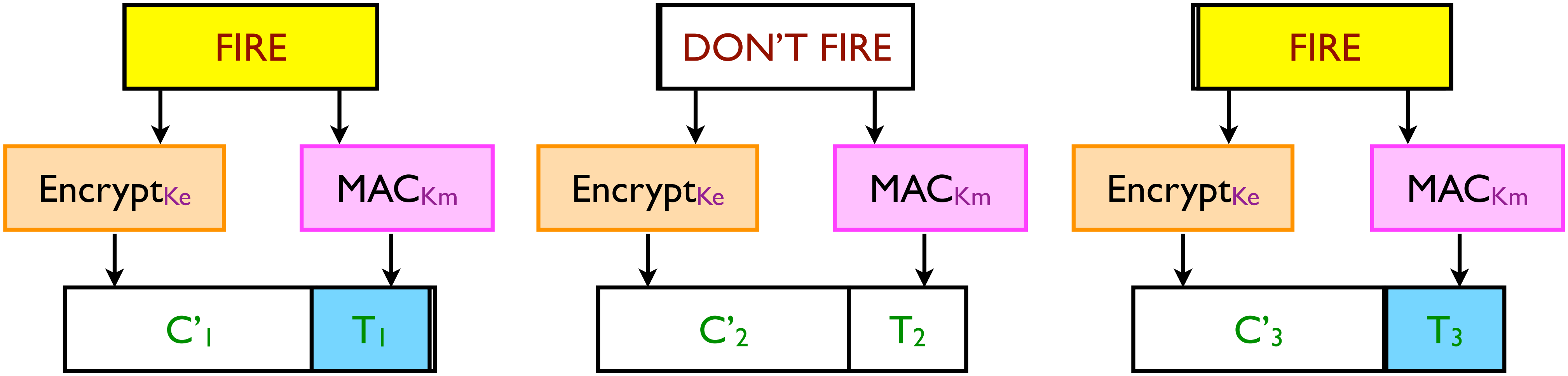
Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



But insecure! [BN, Kra]

Assume Alice sends messages:

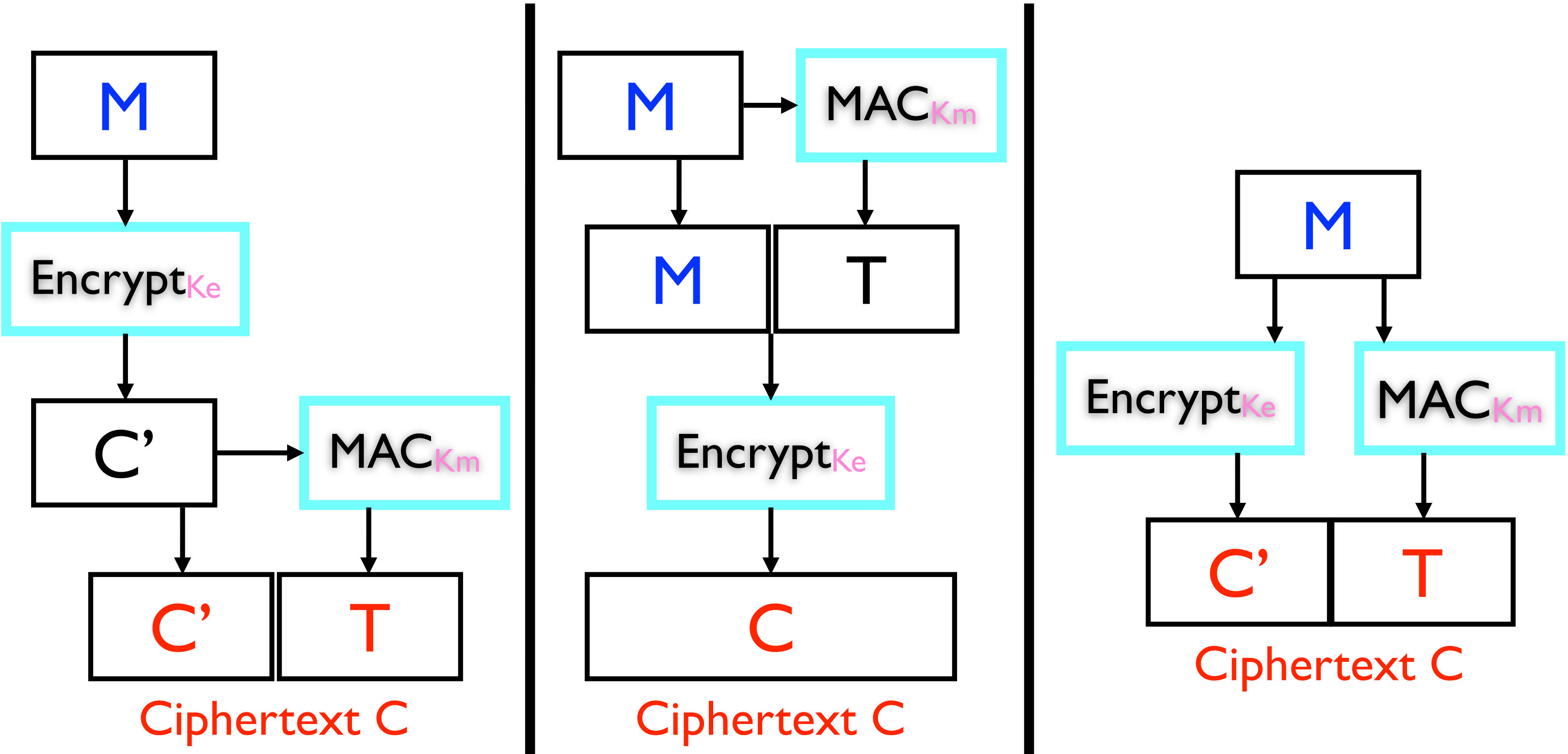


If $T_i = T_j$ then $M_i = M_j$

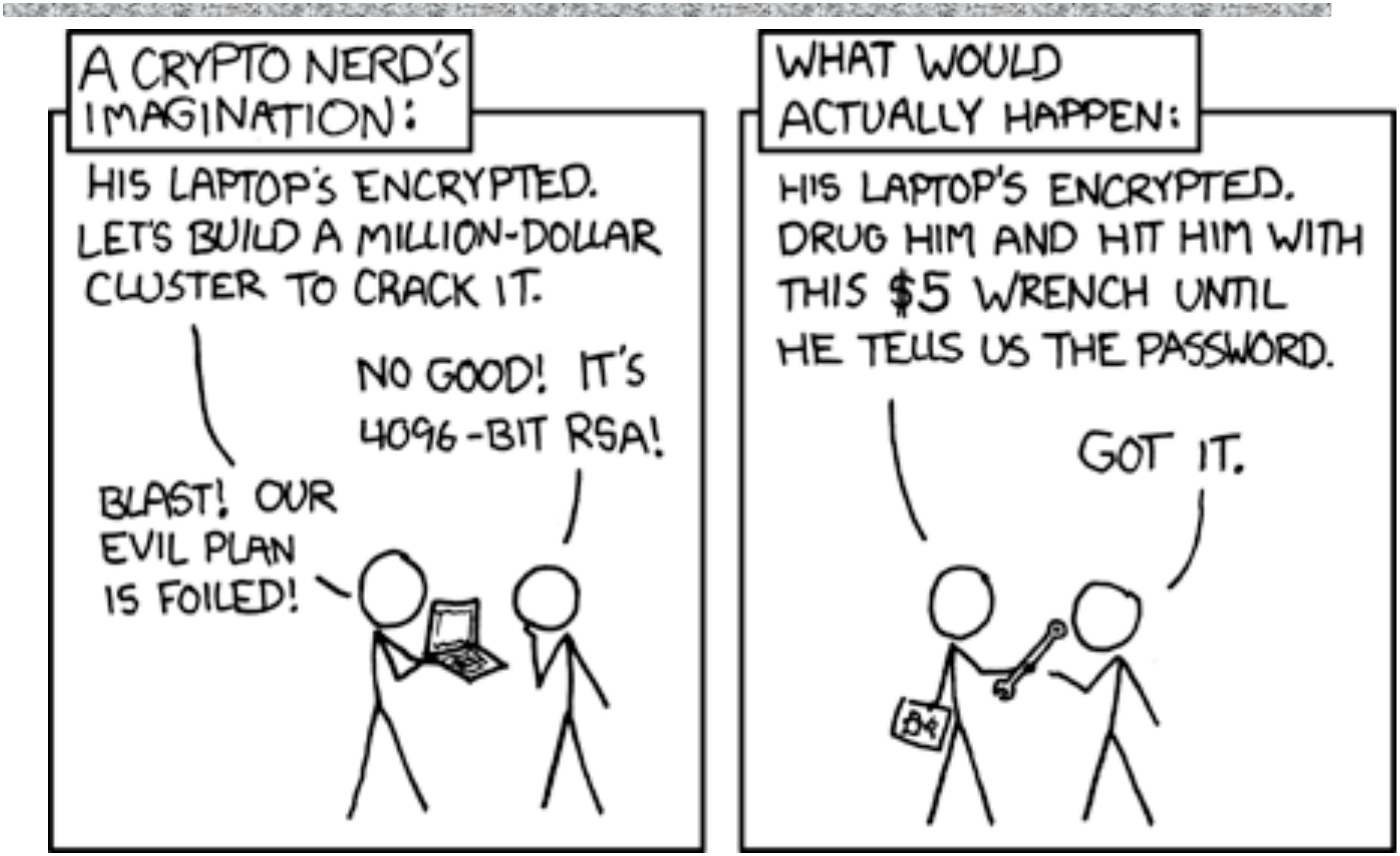
Adversary learns whether two plaintexts are equal.

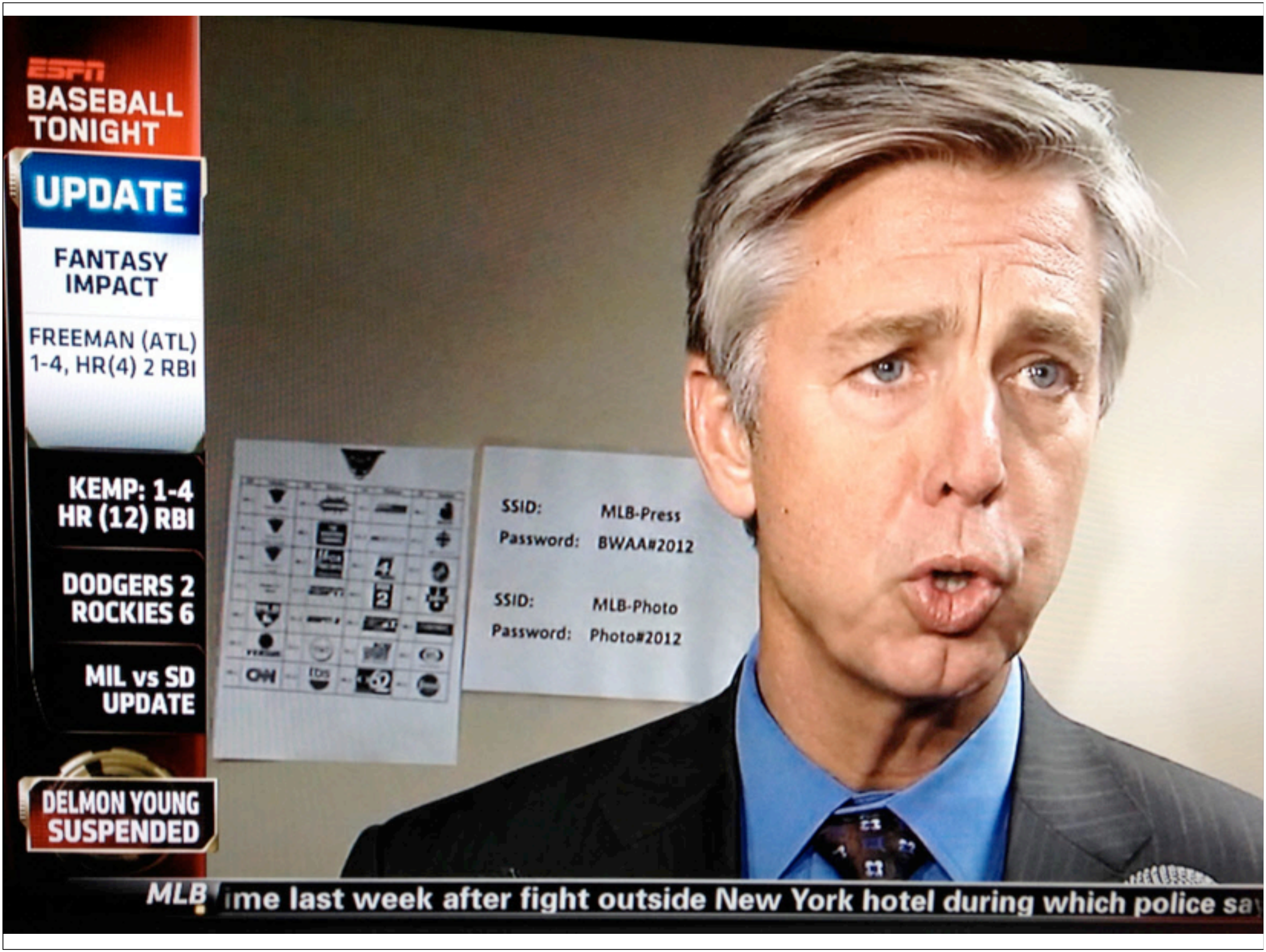
Especially problematic when M_1, M_2, ... take on only a small number of possible values.

Results of [BN00,Kra01]



	Encrypt-then-MAC	MAC-then-Encrypt	Encrypt-and-MAC
Privacy	Strong (CCA)	Weak (CPA)	Insecure
Integrity	Strong (CTXT)	Weak (PTXT)	Weak (PTXT)


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.



SSID: MLB-Press
Password: BWAA#2012

SSID: MLB-Photo
Password: Photo#2012

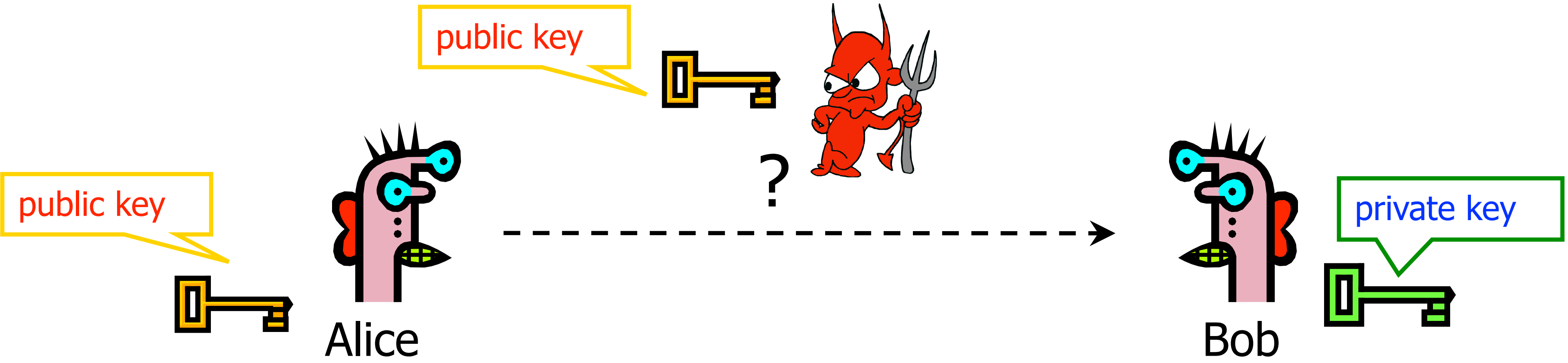
ime last week after fight outside New York hotel during which police sa

(Reminder:) Symmetric Cryptography

- **1 secret key**, shared between sender/receiver
- Repeat fast and simple operations lots of times (rounds) to mix up key and ciphertext
- **Why do we think it is secure?** (simplistic)
  - Lots of heuristic arguments
    - If we do lots and lots and lots of mixing, no simple formula (and reversible) describing the whole process (cryptographic weakness).
    - Mix in ways we think it's hard to short-circuit all the rounds. Especially non-linear mixing, e.g., S-boxes.
  - Some math gives us confidence in these assumptions

Public Key Cryptography

Basic Problem



Given: Everybody knows Bob’s public key

Only Bob knows the corresponding private key

Goals: 1. Alice wants to send a secret message to Bob

2. Bob wants to authenticate himself

Public-Key Cryptography

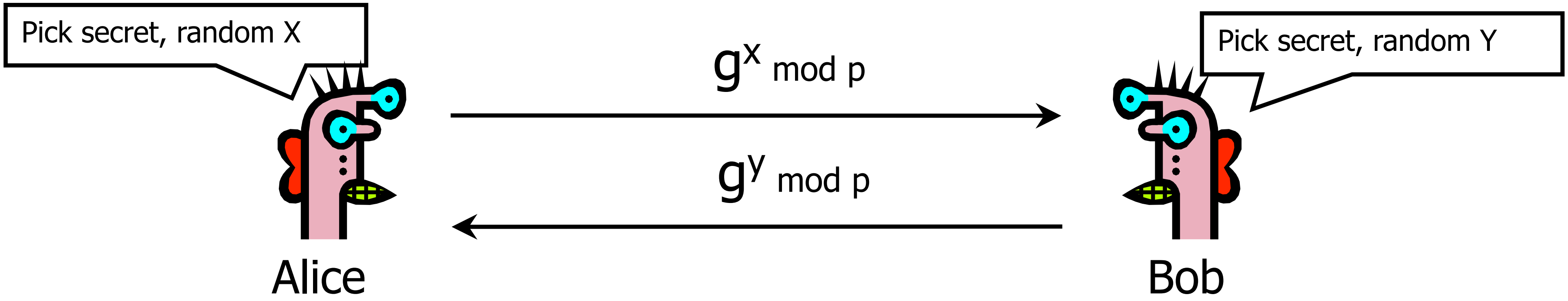
- Everyone has **1 private key and 1 public key**
  - **One for each security goal**
  - **Or 2 private and 2 public, when considering both encryption and authentication**
- Mathematical relationship between private and public keys
- **Why do we think it is secure?** (simplistic)
  - Relies entirely on **problems we believe are "hard"**

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (or at least different)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can “sign” a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime number, g is a generator of Z_p^*
    - $Z_p^*=\{1, 2 \ldots p-1\}$; $\forall a \in Z_p^*$ $\exists i$ such that $a=g^i \bmod p$
    - Modular arithmetic: numbers “wrap around” after they reach p



Pick secret, random X

Pick secret, random Y

$g^x \bmod p$

$g^y \bmod p$

Alice

Bob

Compute $k=(g^y)^x=g^{xy} \bmod p$

Compute $k=(g^x)^y=g^{xy} \bmod p$

http://www.wolframalpha.com/ and http://www.google.com

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:

  given g^x mod p, it's hard to extract x

  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!

- Computational Diffie-Hellman (CDH) problem:

  given g^x and g^y, it's hard to compute g^{xy} mod p

  - … unless you know x or y, in which case it's easy

- Decisional Diffie-Hellman (DDH) problem:

  given g^x and g^y, it's hard to tell the difference between g^{xy} mod p and g^r mod p where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication

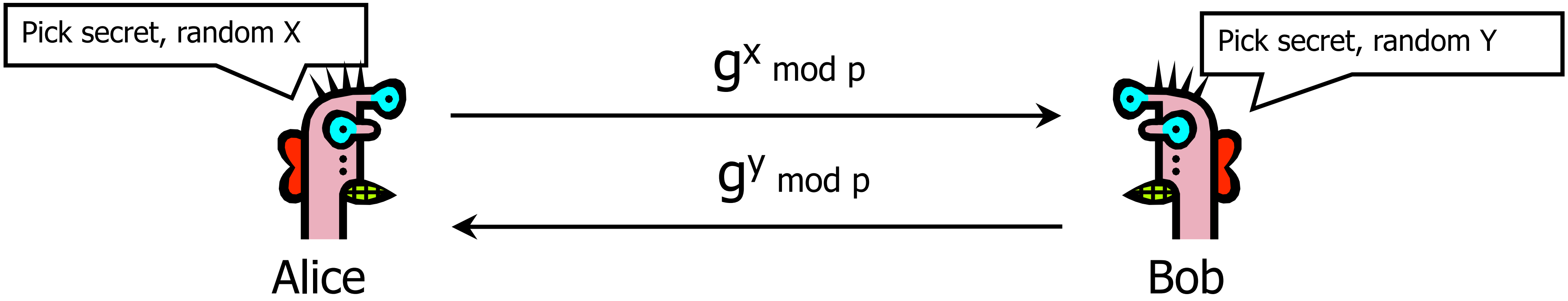
Properties of Diffie-Hellman

- DDH: not true for integers mod p, but true for other groups
- DL problem in p can be broken down into DL problems for subgroups, if factorization of p-1 is known.
- Common recommendation:
  - Choose p = 2q+1 where q is also a large prime.
  - Pick a g that generates a subgroup of order q in Z_p^*
    - DDH is hard for this group
    - (OK to not know all the details of why for this course.)

  - Hash output of DH key exchange to get the key

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p, q are large prime numbers, $p=2q+1$, g a generator for the subgroup of order q
    - Modular arithmetic: numbers “wrap around” after they reach p



Pick secret, random X

Pick secret, random Y

$g^x \bmod p$

$g^y \bmod p$

Alice

Bob

Compute $k=H((g^y)^x)=H(g^{xy} \bmod p)$ Compute $k=H((g^x)^y)=H(g^{xy} \bmod p)$

Requirements for Public-Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
  - Computationally infeasible to determine private key SK given only public key PK
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to compute M from C without SK
  - Even infeasible to learn partial information about M
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ is the number of integers in the $[1,n]$ interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- Euler's theorem:

  if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$

  Z_n^*: multiplicative group of integers mod n (integers relatively prime to n)
- Special case: <u>Fermat's Little Theorem</u>

  if p is prime and $\gcd(a,p)=1$, then $a^{p-1} = 1 \bmod p$

RSA Cryptosystem

[Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 bits each (need primality testing, too)
  - Compute n=pq and $\varphi(n)=(p-1)(q-1)$
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, e=3 or $e=2^{16}+1=65537$ (why?)
  - Compute unique d such that ed = 1 mod $\varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
  - Modular exponentiation by repeated squaring
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why RSA Decryption Works (Simplified)

- $e \cdot d = 1 \bmod \varphi(n)$, thus $e \cdot d = 1 + k \cdot \varphi(n)$ for some k

  Can rewrite: $e \cdot d = 1 + k(p-1)(q-1)$

- Let m be any integer in Z_n* (not all of Z_n)
- $c^d \bmod n = (m^e)^d \bmod n$
- $= m^{1+k(p-1)(q-1)} \bmod n$
- $= (m \bmod n) * (m^{k(p-1)(q-1)} \bmod n)$
- Recall: Euler's theorem:

  if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$

- $c^d \bmod n = (m \bmod n) * (1 \bmod n)$
- $= m \bmod n$
- But: True for all m in Z_n, not just m in Z_n*

Why RSA Decryption Works (skip)

- $e \cdot d = 1 \bmod \varphi(n)$, thus $e \cdot d = 1 + k \cdot \varphi(n)$ for some k
  Can rewrite: $e \cdot d = 1 + k(p-1)(q-1)$

- Let m be any integer in Z_n
- If $\gcd(m,p)=1$, then $m^{ed} = m \bmod p$
  - By Fermat's Little Theorem, $m^{p-1} = 1 \bmod p$
  - Raise both sides to the power $k(q-1)$ and multiply by m
  - $m^{1+k(p-1)(q-1)} = m \bmod p$, thus $m^{ed} = m \bmod p$
  - By the same argument, $m^{ed} = m \bmod q$
- Since p and q are distinct primes and $p \cdot q = n$,
  $m^{ed} = m \bmod n$ (using the Chinese Remainder Theorem)
- True for all m in Z_n, not just m in Z_n^*

Why Is RSA Secure?

- RSA problem: given n=pq, e such that gcd(e,(p-1)(q-1))=1 and c, find m such that m^e=c mod n
  - i.e., recover m from ciphertext c and public key (n,e) by taking eth root of c
  - There is no known efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes p_1, …, p_k such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$
- If factoring is easy, then RSA problem is easy (because knowing factors means you can compute d), but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring n -- but if it is, we don’t know how

On RSA encryption

- Encrypted message needs to be in interpreted as an integer less than n
  - Reason: Otherwise can't decrypt.
  - Message is very often a symmetric encryption key.
- But still not quite that simple

Caveats

- e =3 is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of c to recover m (i.e., no operations taken module n)
    - Even problems if "pad" m in some ways [Hastad]
  - Let $c_i = m^3 \bmod n_i$ - same message is encrypted to three people
    - Adversary can compute $m^3 \bmod n_1 n_2 n_3$ (using CRT)
    - Then take ordinary cube root to recover m

- Don't use RSA **directly** for privacy! Need to pre-process input in some way.

Sample Encryption

- 26 2 15 13 7 14 13 13 1 28 14 15 13 14
  20 9 6 31 25 26 14 16 23 15 26 2 6 13 1

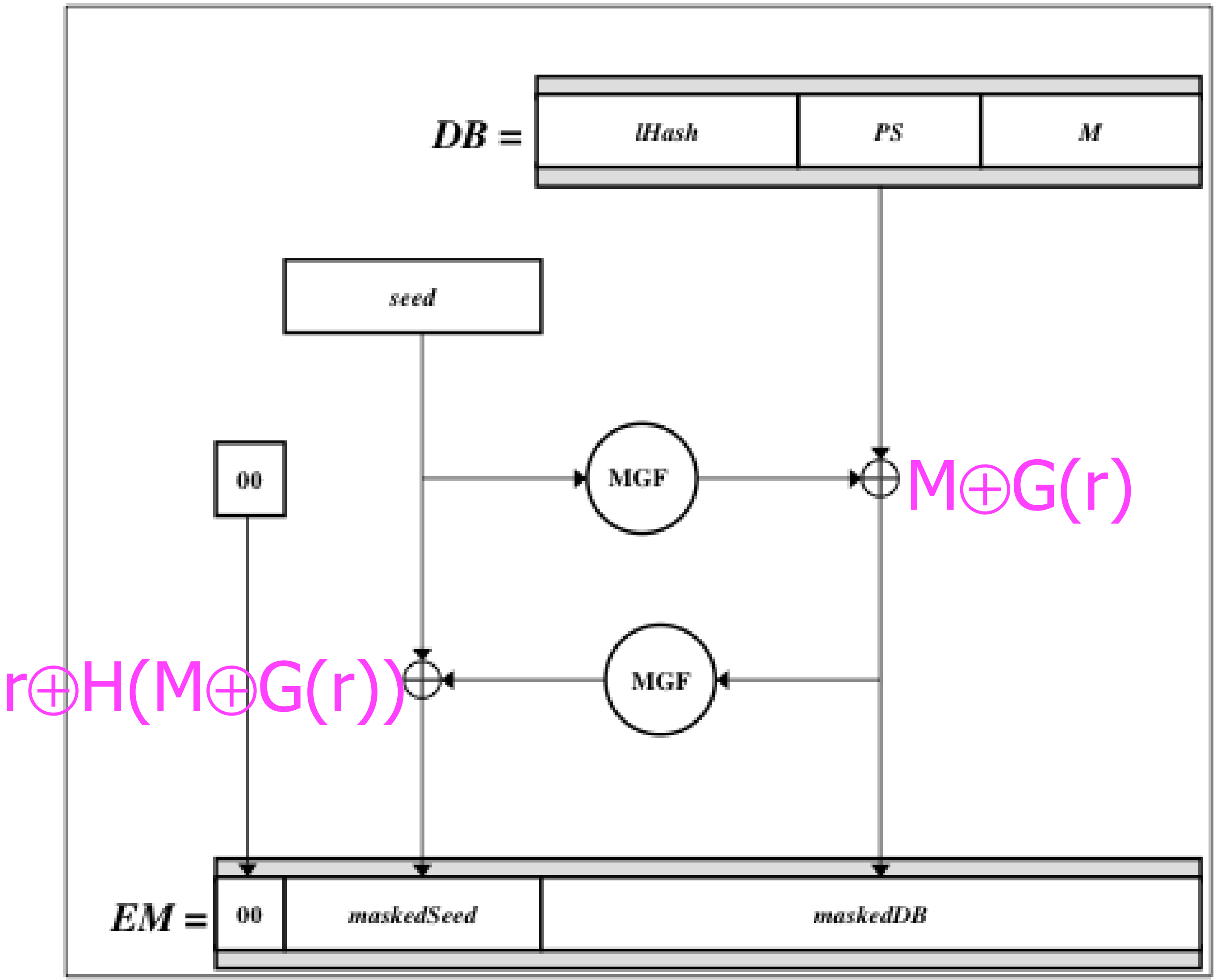
- P=3, Q=11, N=33, E=7, D=3
- ‘A’ converted to 1 before encryption; ‘B’ Converted to 2 before encryption; ...

- A-1 B-2 C-3 D-4 E-5 F-6 G-7 H-8 I-9 J-10 K-11 L-12 M-13 N-14 O-15 P-16 Q-17 R-18 S-19 T-20 U-21 V-22 W-23 X-24 Y-25 Z-26
- http://www.wolframalpha.com/ or http://www.google.com

Integrity in RSA Encryption

- Plain RSA does not provide integrity
  - Given encryptions of m_1 and m_2, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \bmod n = (m_1 \cdot m_2)^e \bmod n$
  - Attacker can convert m into m^k without decrypting
    - $(m_1^e)^k \bmod n = (m^k)^e \bmod n$
- In practice, OAEP is used: instead of encrypting M, encrypt $M \oplus G(r) ; r \oplus H(M \oplus G(r))$
  - r is random and fresh, G and H are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - … if hash functions are "good" and RSA problem is hard

OAEP (image from PKCS #1 v2.1)



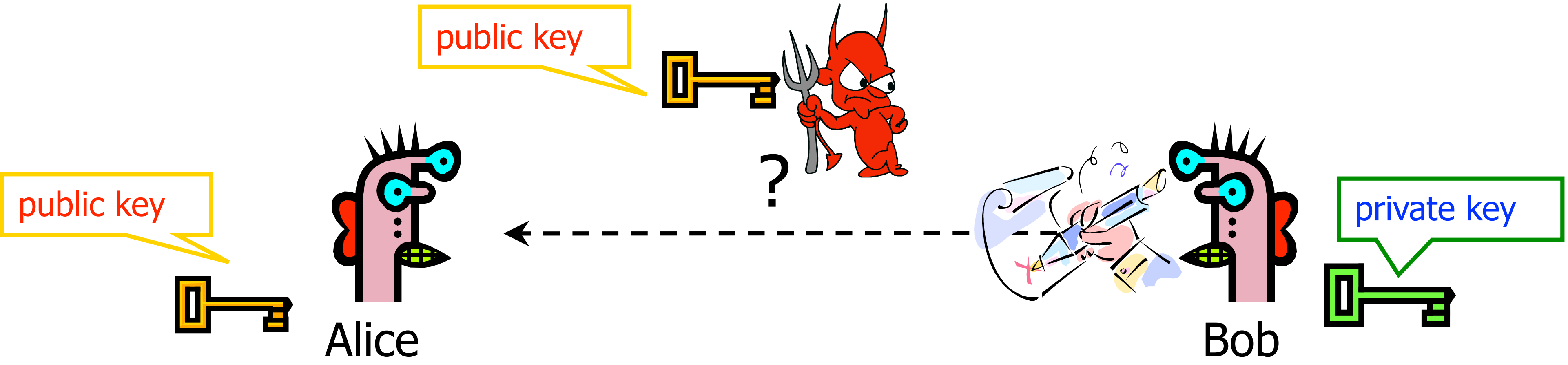
M⊕G(r)

r⊕H(M⊕G(r))

Summary of RSA

- Defined RSA primitives
  - Encryption and Decryption
  - Underlying number theory
  - Practical concerns, some mis-uses
  - OAEP

Digital Signatures: Basic Idea



<u>Given</u>: Everybody knows Bob's public key

Only Bob knows the corresponding private key

<u>Goal</u>: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key

RSA Signatures

- Public key is (n,e), private key is d
- To sign message m: $s = m^d \bmod n$
  - Signing and decryption are the same **underlying** operation in RSA
  - It's infeasible to compute s on m if you don't know d
- To verify signature s on message m:

  $s^e \bmod n = (m^d)^e \bmod n = m$

  - Just like encryption
  - Anyone who knows n and e (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures

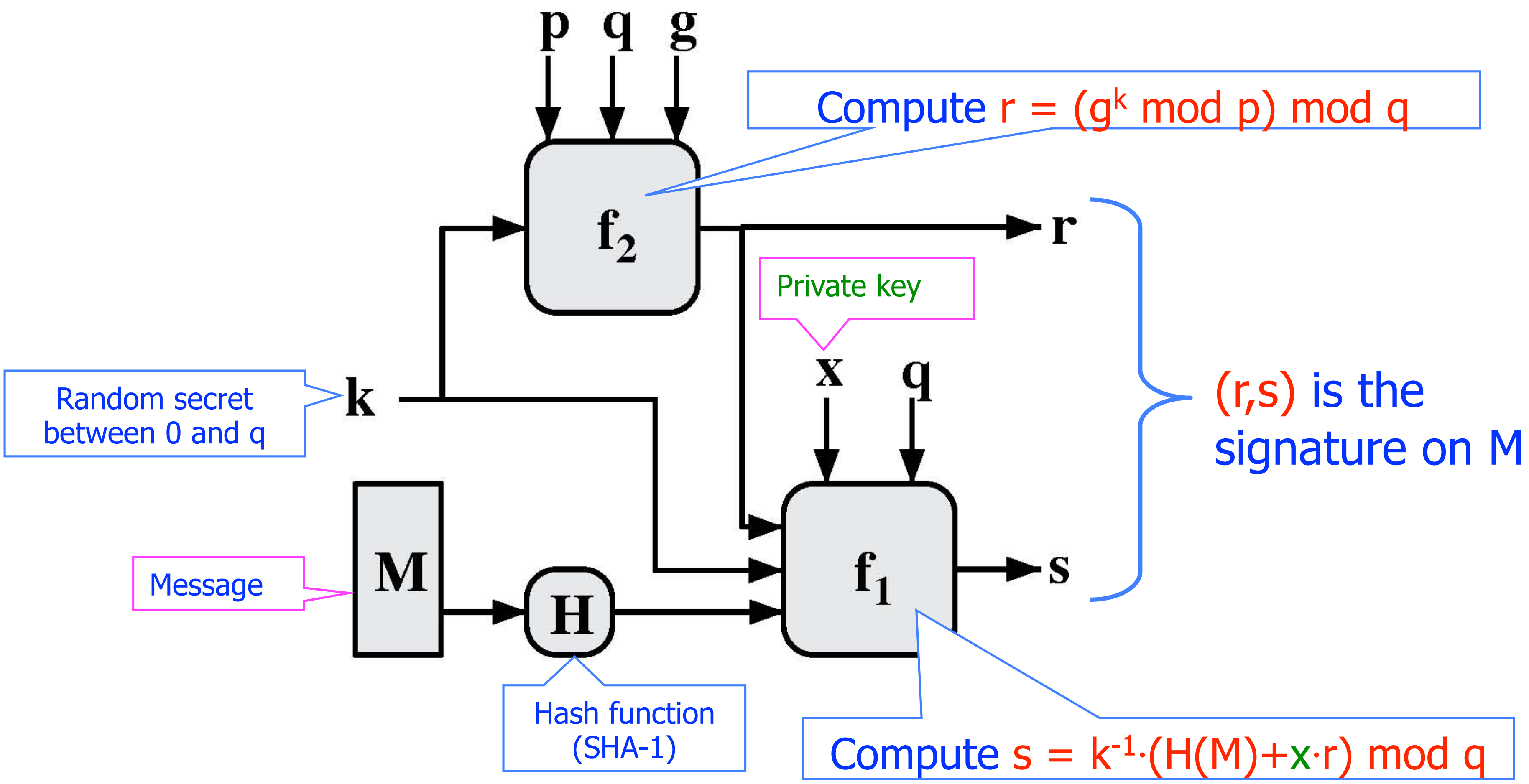
Encryption and Signatures

- Often people think: Encryption and decryption are inverses.
- That's a common view
  - True for the RSA **primitive (underlying component)**
- But not one we'll take
  - To really use RSA, we need padding
  - And there are many other decryption methods
  - And there are many other signing methods

Digital Signature Standard (DSS)

- U.S. government standard (1991-94)
  - Modification of the ElGamal signature scheme (1985)
- Key generation:
  - Generate large primes p, q such that q divides p-1
    - $2^{159} < q < 2^{160}$, $2^{511+64t} < p < 2^{512+64t}$ where $0 \leq t \leq 8$
  - Select $h \in Z_p^*$ and compute $g = h^{(p-1)/q} \bmod p$
  - Select random x such $1 \leq x \leq q-1$, compute $y = g^x \bmod p$
- Public key: $(p, q, g, y = g^x \bmod p)$, private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from $g^x \bmod p$ (public key)

DSS: Signing a Message (Skim)



Compute $r = (g^k \bmod p) \bmod q$

Private key

Random secret between 0 and q

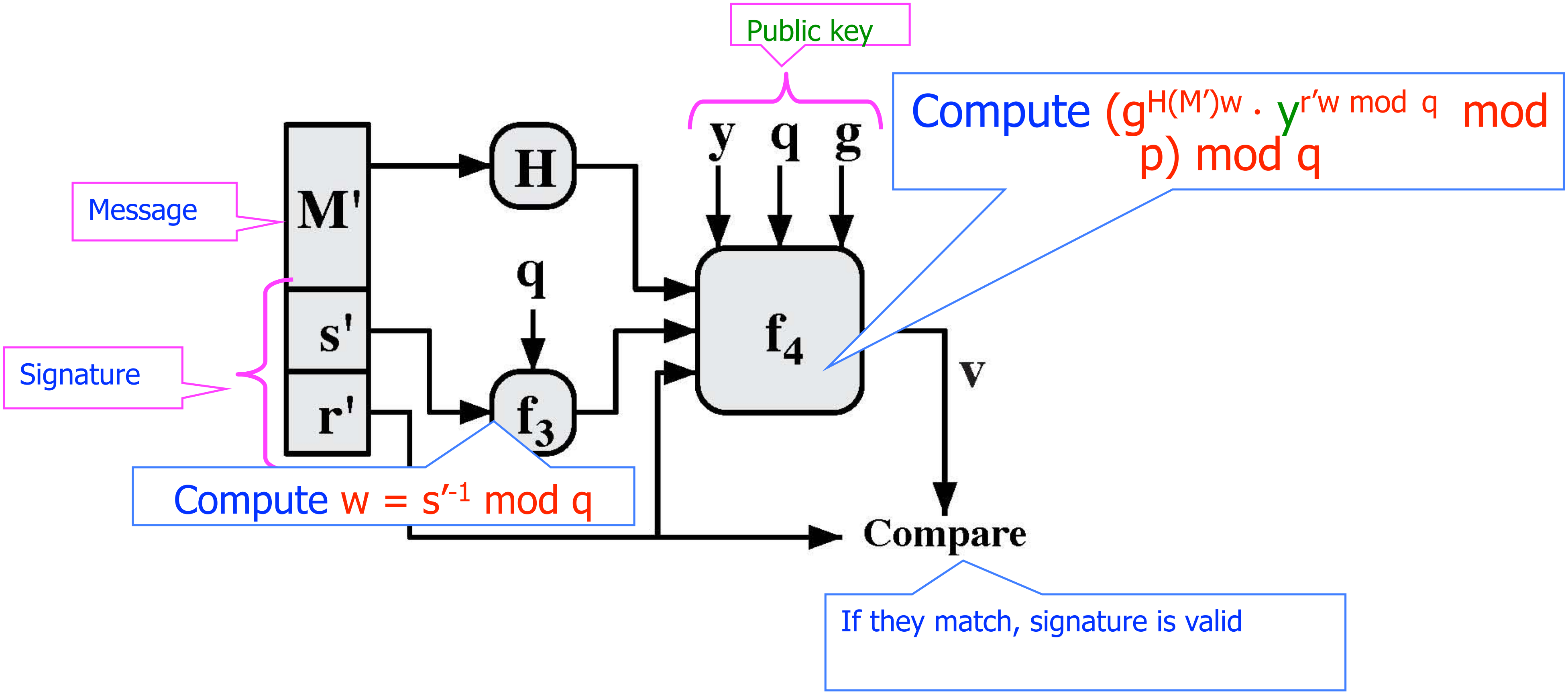
(r,s) is the signature on M

Message

Hash function (SHA-1)

Compute $s = k^{-1} \cdot (H(M)+x \cdot r) \bmod q$

DSS: Verifying a Signature (Skim)



Public key

Compute $(g^{H(M')w} \cdot y^{r'w \bmod q} \bmod p) \bmod q$

Message

Signature

Compute $w = s'^{-1} \bmod q$

If they match, signature is valid

Advantages of Public-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - **Fewer** “chicken-and-egg” key establishment problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
    - Caveats to come
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
- Reduce protection of information to protection of authenticity of public keys
  - No need to keep public keys secret, but must be sure that Alice’s public key is really her true public key

Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - E.g., IPsec, SSL, SSH, ...
- Keys are longer
  - 1024+ bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - What if factoring is easy?
    - Factoring is believed to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions)

Note: Optimizing Exponentiation

- How to compute M^x mod N? Say x=13
- Sums of power of 2, x = 8+4+1 = $2^3+2^2+2^0$
- Can also write x in binary, e.g., x = 1101
- Can solve by repeated squaring
  - y = 1;
  - $y = y^2 * M \bmod N$ // $y = M$
  - $y = y^2 * M \bmod N$ // $y = M^2 * M = M^{2+1} = M^3$
  - $y = y^2 \bmod N$ // $y = (M^{2+1})^2 = M^{4+2}$
  - $y = y^2 * M \bmod N$ // $y = (M^{4+2})^2 * M = M^{8+4+1}$
- Does anyone see a potential issue?

Timing attacks

Collect timings for exponentiation with a bunch of messages M1, M2, ... (e.g., RSA signing operations with a private exponent)

Assume (inductively) know b_3=1, b_2=1, guess b_1=1

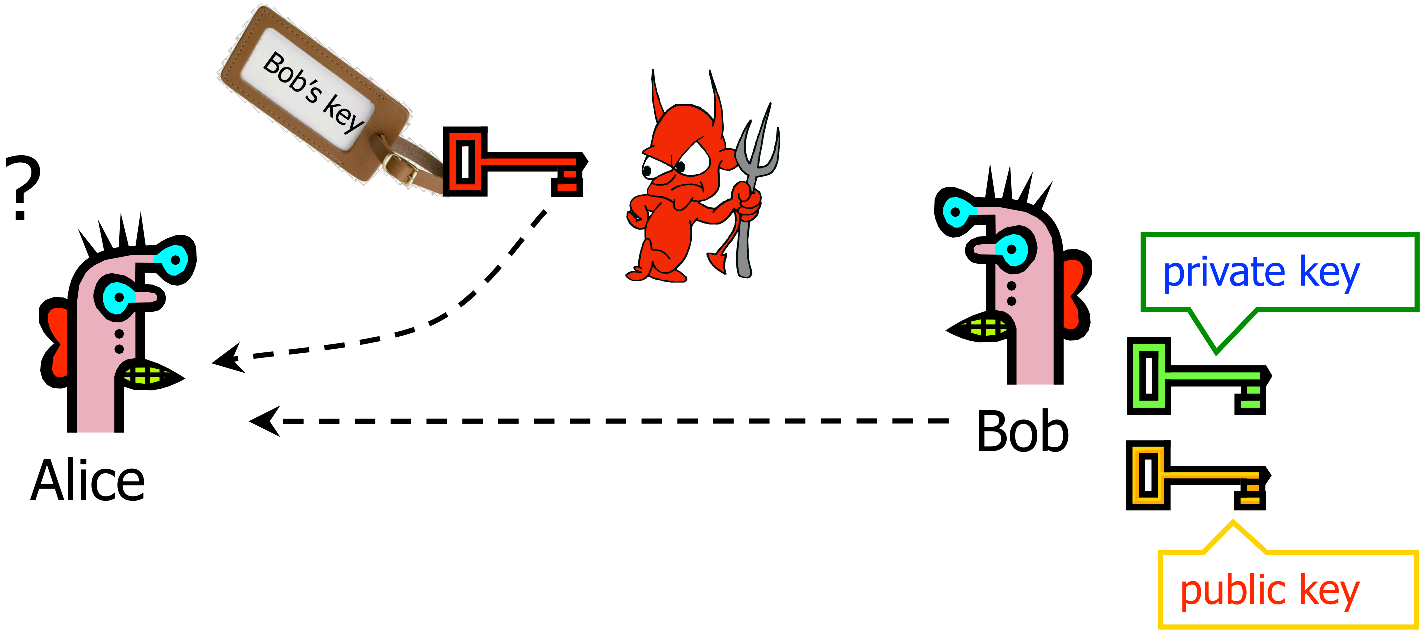
i	$b_i = 0$	$b_i = 1$	Comp	Meas
3	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		
2	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		
1	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$	X1 secs	
0	$y = y^2 \bmod N$	$y = y^2 * M1 \bmod N$		Y1 secs

i	$b_i = 0$	$b_i = 1$	Comp	Meas
3	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		
2	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		
1	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$	X2 secs	
0	$y = y^2 \bmod N$	$y = y^2 * M2 \bmod N$		Y2 secs

Timing attacks

- If b_1 = 1, then set of { Yj - Xj | j in {1,2, ..} } has distribution with "small" variance (due to time for final step, i=0)
  - "Guess" was correct when we computed X1, X2, ...
- If b_1 = 0, then set of { Yj - Xj | j in {1,2, ..} } has distribution with "large" variance (due to time for final step, i=0, and incorrect guess for b_1)
  - "Guess" was incorrect when we computed X1, X2, ...
  - So time computation wrong (Xj computed as large, but really small, ...)
- Strategy: Force user to sign large number of messages M1, M2, ... Record timings for signing.
- Iteratively learn bits of key by using above property.

Authenticity of Public Keys



Problem: How does Alice know that the public key she received is really Bob's public key?

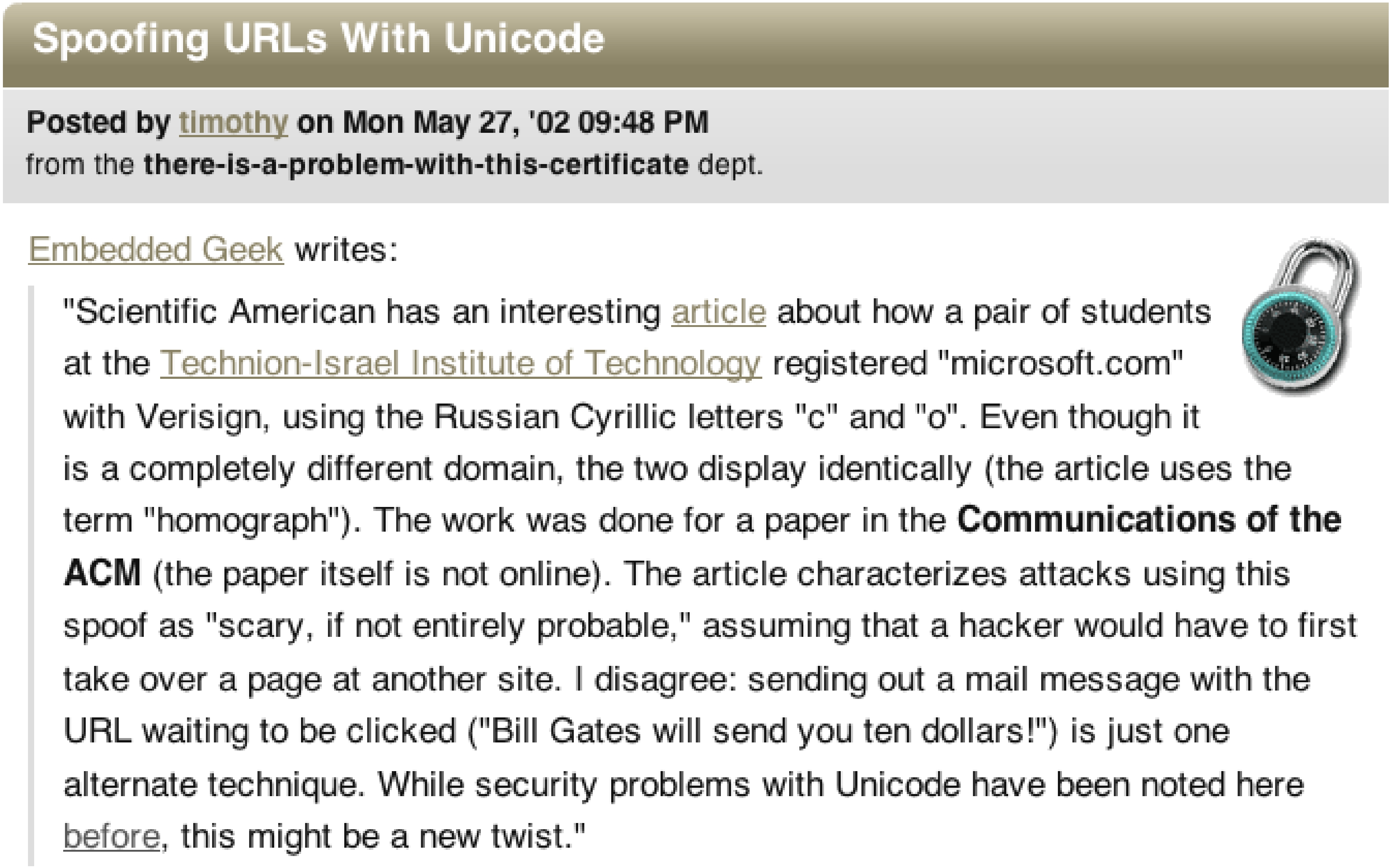
Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}(“Bob”, PK_B)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  - Every computer is pre-configured with CA’s public key

Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority’s signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $\mathrm{sig}_{\mathrm{Verisign}}$(“AnotherCA”, $\mathrm{PK}_{\mathrm{AnotherCA}}$), $\mathrm{sig}_{\mathrm{AnotherCA}}$(“Alice”, PK_{A})
  - What happens if root authority is ever compromised?

Many Challenges



Spoofing URLs With Unicode

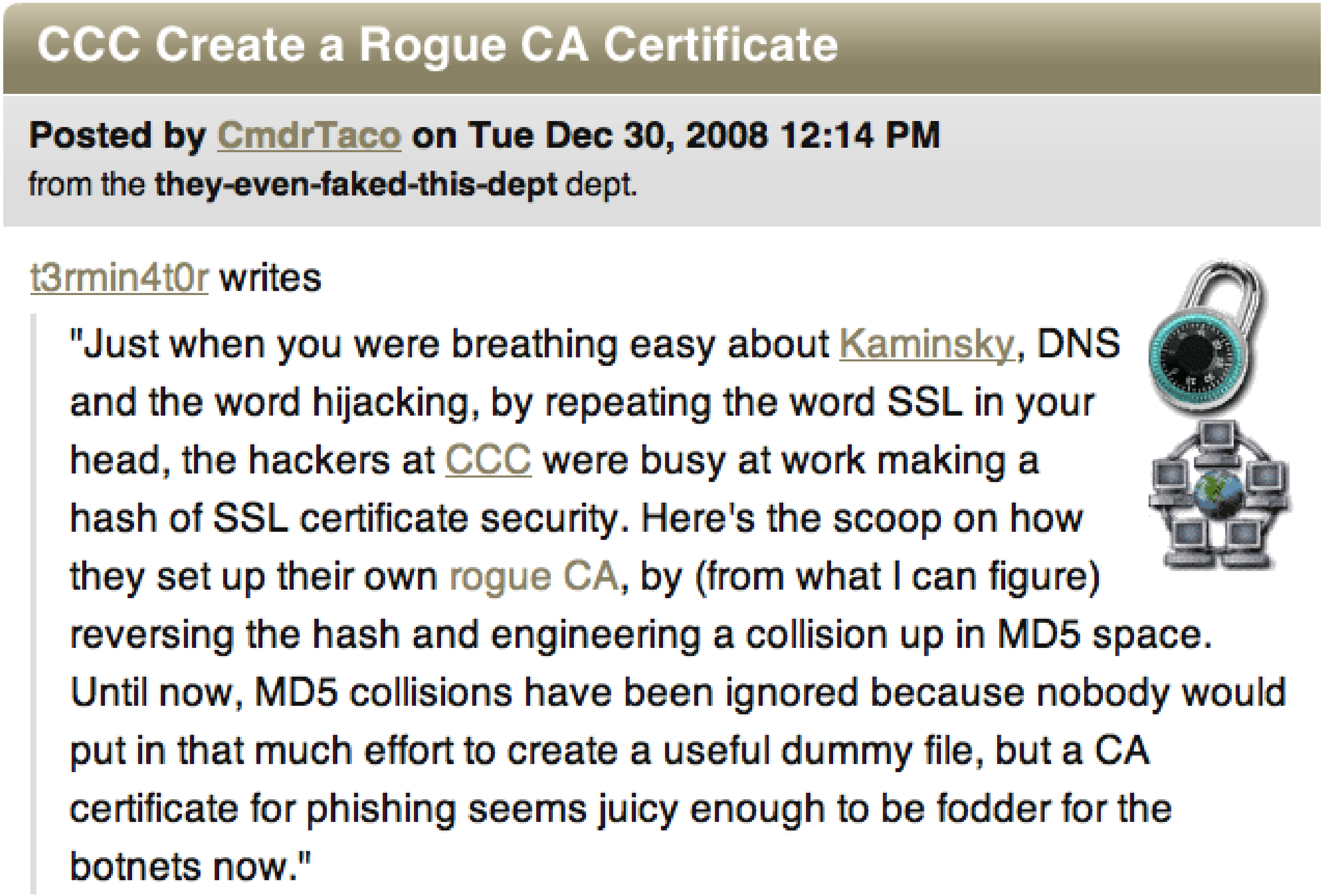
Posted by timothy on Mon May 27, '02 09:48 PM
from the **there-is-a-problem-with-this-certificate** dept.

Embedded Geek writes:

> "Scientific American has an interesting article about how a pair of students at the Technion-Israel Institute of Technology registered "microsoft.com" with Verisign, using the Russian Cyrillic letters "c" and "o". Even though it is a completely different domain, the two display identically (the article uses the term "homograph"). The work was done for a paper in the **Communications of the ACM** (the paper itself is not online). The article characterizes attacks using this spoof as "scary, if not entirely probable," assuming that a hacker would have to first take over a page at another site. I disagree: sending out a mail message with the URL waiting to be clicked ("Bill Gates will send you ten dollars!") is just one alternate technique. While security problems with Unicode have been noted here before, this might be a new twist."

http://it.slashdot.org/story/08/12/30/1655234/CCC-Create-a-Rogue-CA-Certificate
http://www.win.tue.nl/hashclash/rogue-ca/

Many Challenges

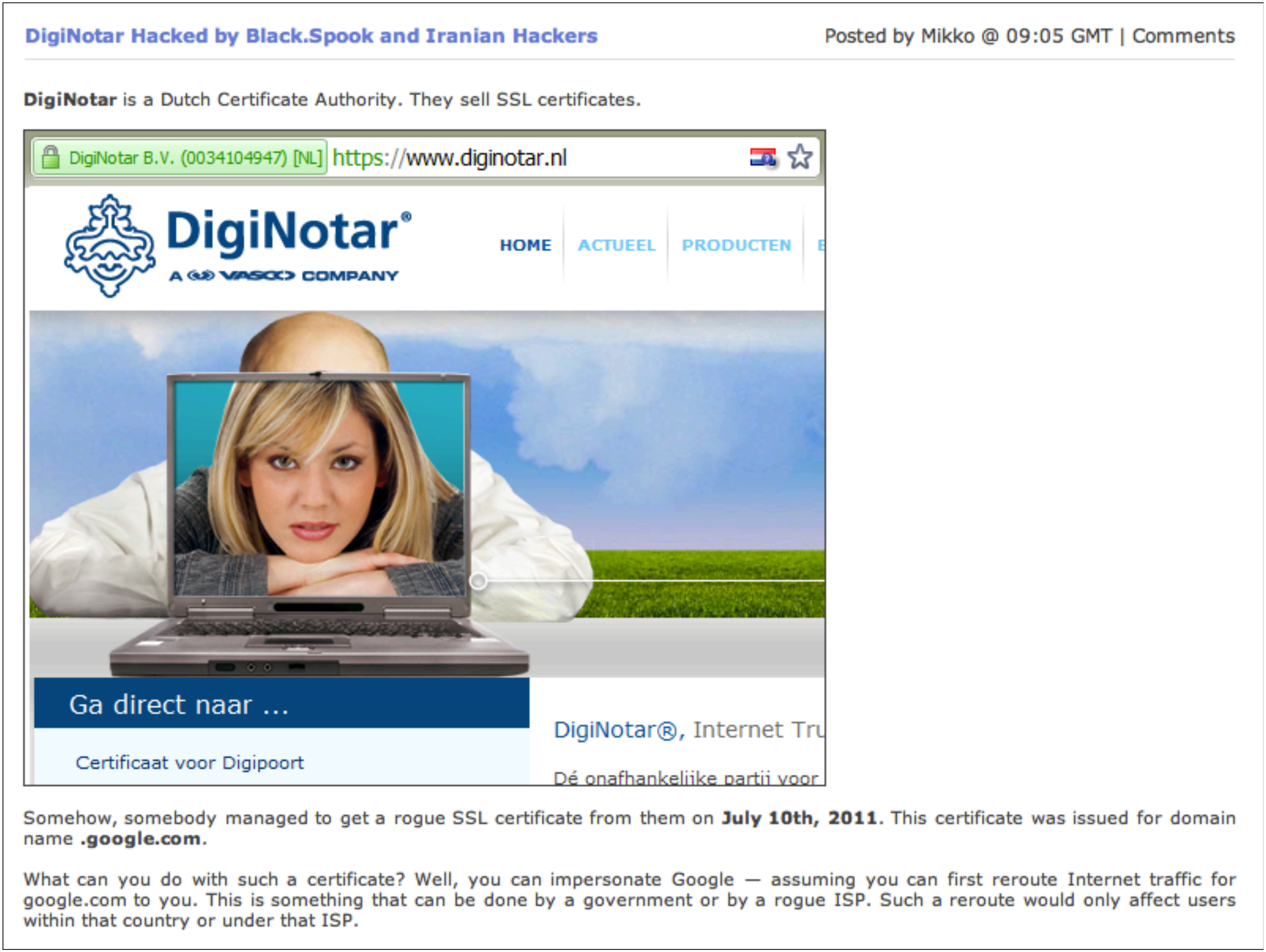


CCC Create a Rogue CA Certificate

Posted by CmdrTaco **on Tue Dec 30, 2008 12:14 PM**
from the **they-even-faked-this-dept** dept.

t3rmin4t0r writes

> "Just when you were breathing easy about Kaminsky, DNS and the word hijacking, by repeating the word SSL in your head, the hackers at CCC were busy at work making a hash of SSL certificate security. Here's the scoop on how they set up their own rogue CA, by (from what I can figure) reversing the hash and engineering a collision up in MD5 space. Until now, MD5 collisions have been ignored because nobody would put in that much effort to create a useful dummy file, but a CA certificate for phishing seems juicy enough to be fodder for the botnets now."



DigiNotar Hacked by Black.Spook and Iranian Hackers

Posted by Mikko @ 09:05 GMT | Comments

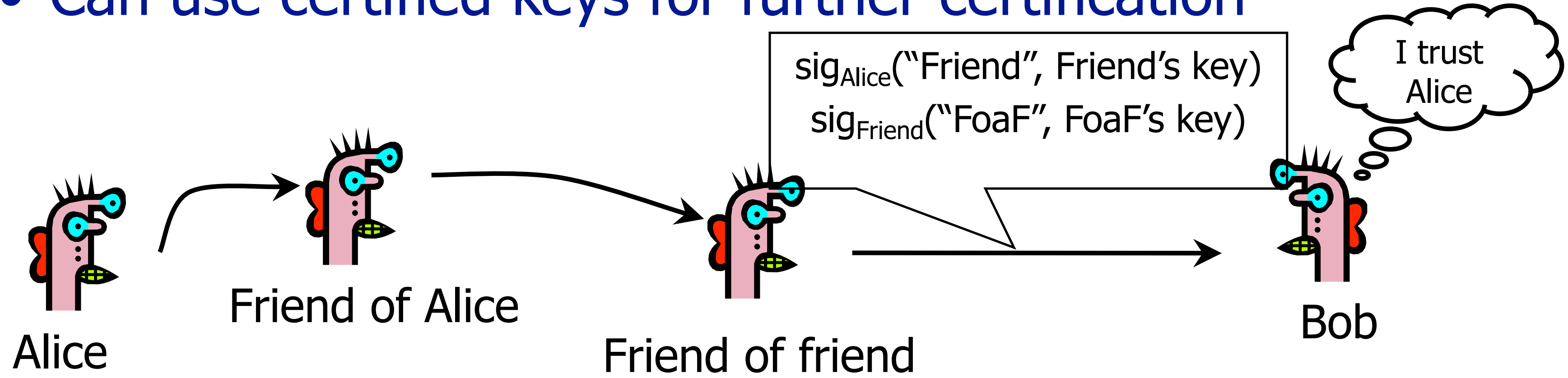
DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.

Somehow, somebody managed to get a rogue SSL certificate from them on **July 10th, 2011**. This certificate was issued for domain name **.google.com**.

What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.

Alternative: “Web of Trust”

- Used in PGP (Pretty Good Privacy)
- Instead of a single root certificate authority, each person has a set of keys they “trust”
  - If public-key certificate is signed by one of the “trusted” keys, the public key contained in it will be deemed valid
- Trust can be transitive
  - Can use certified keys for further certification



$\text{sig}_{\text{Alice}}$(“Friend”, Friend’s key)
$\text{sig}_{\text{Friend}}$(“FoaF”, FoaF’s key)

I trust Alice

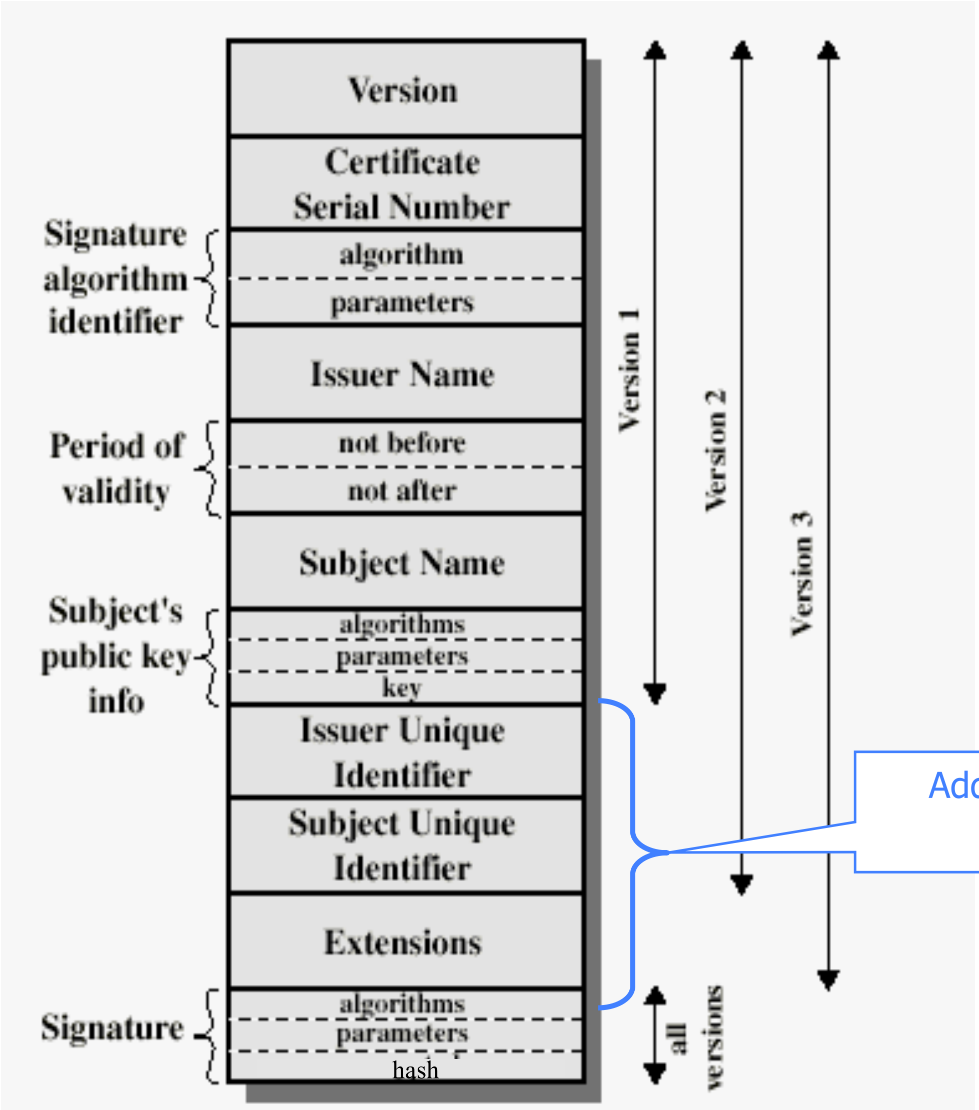
Alice

Friend of Alice

Friend of friend

Bob

X.509 Certificate



Added in X.509 versions 2 and 3 to address usability and security problems

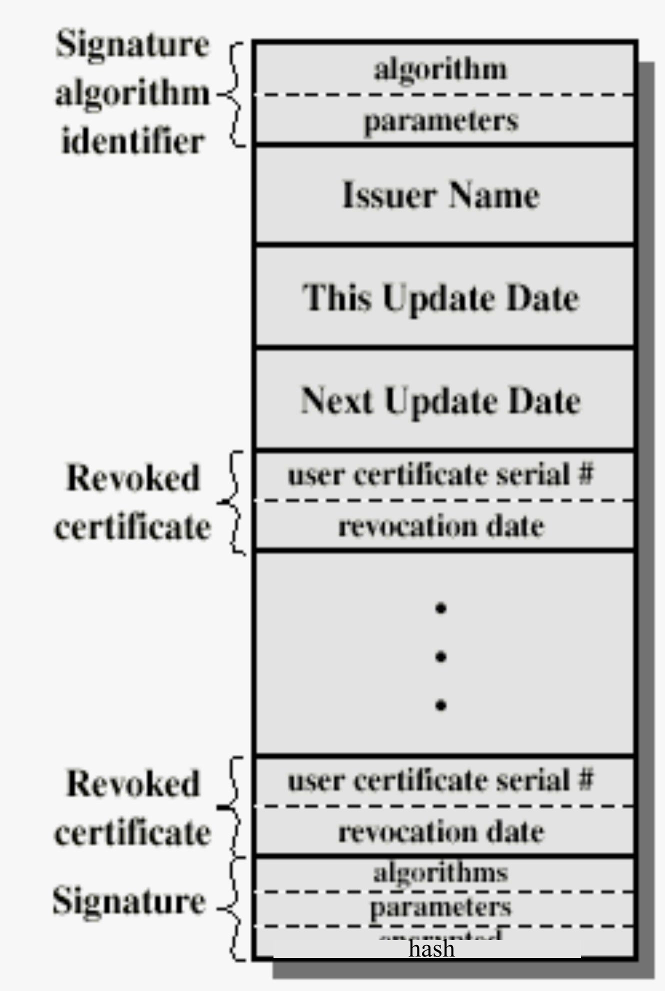
Certificate Revocation

- Revocation is <u>very</u> important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  - CA’s private key has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don’t bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities

Certificate Revocation Mechanisms

- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor
- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a “delta CRL” containing only updates

X.509 Certificate Revocation List



Because certificate serial numbers must be unique within each CA, this is enough to identify the certificate

Convergence