CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography:
Symmetric Encryption (finish), Hash Functions, Message Authentication Codes

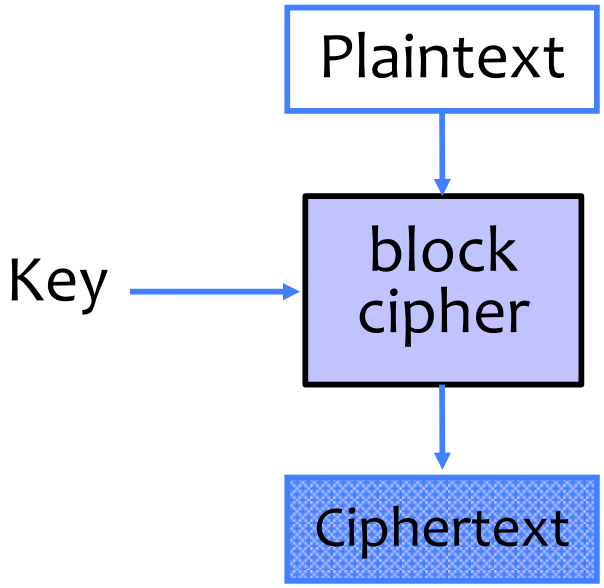
Spring 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

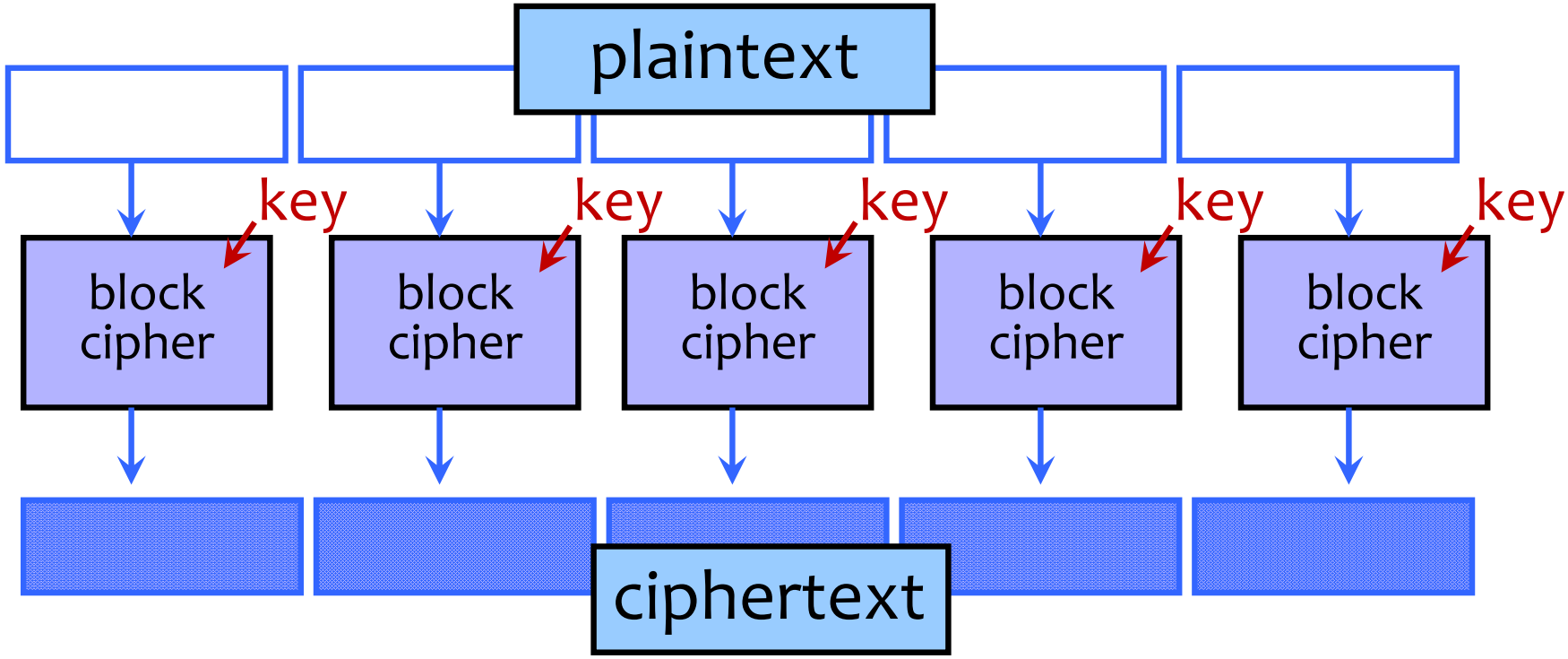
Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Recap: 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)

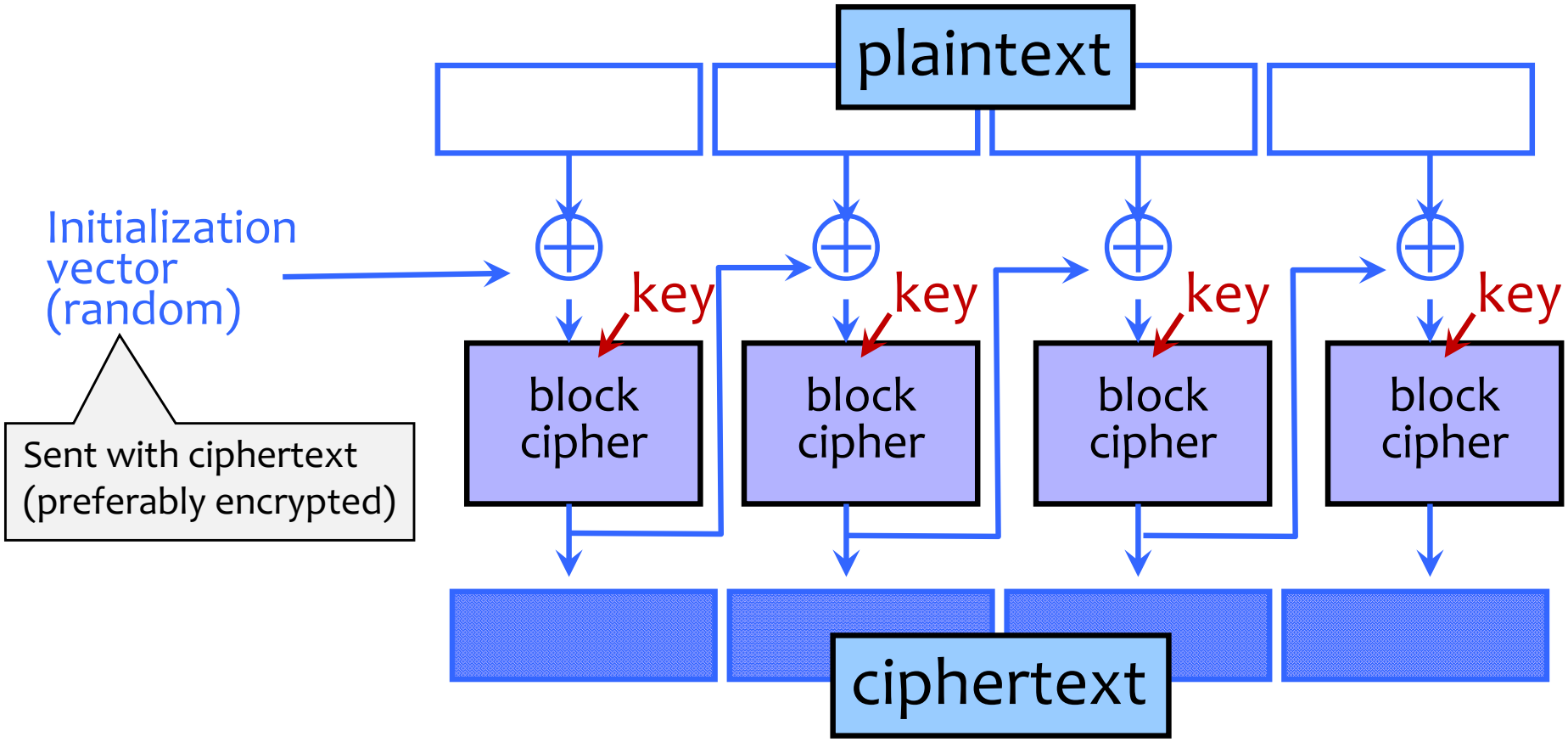


Electronic Code Book (ECB) Mode



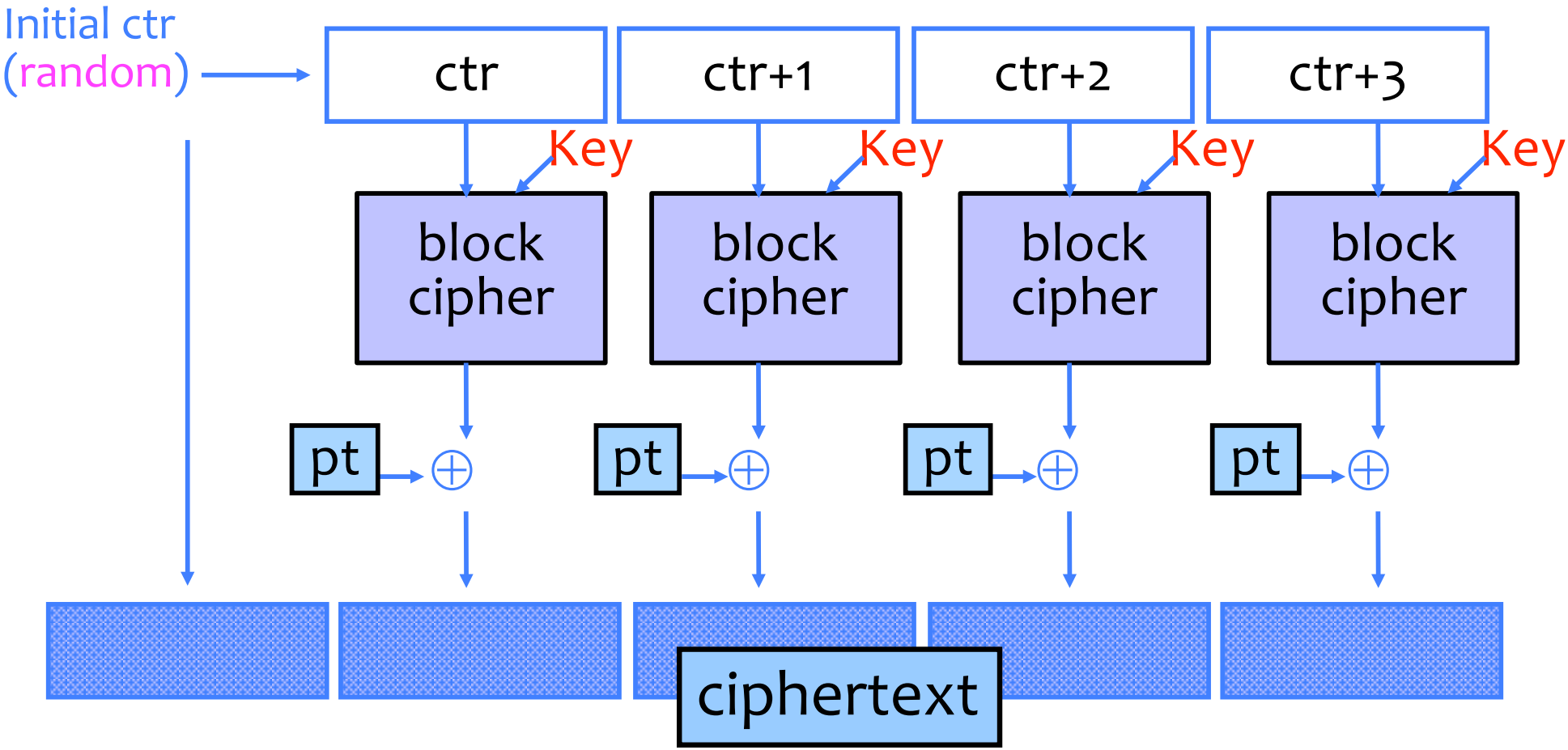
- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Cipher Block Chaining (CBC) Mode: Encryption



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

Counter Mode (CTR): Encryption



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

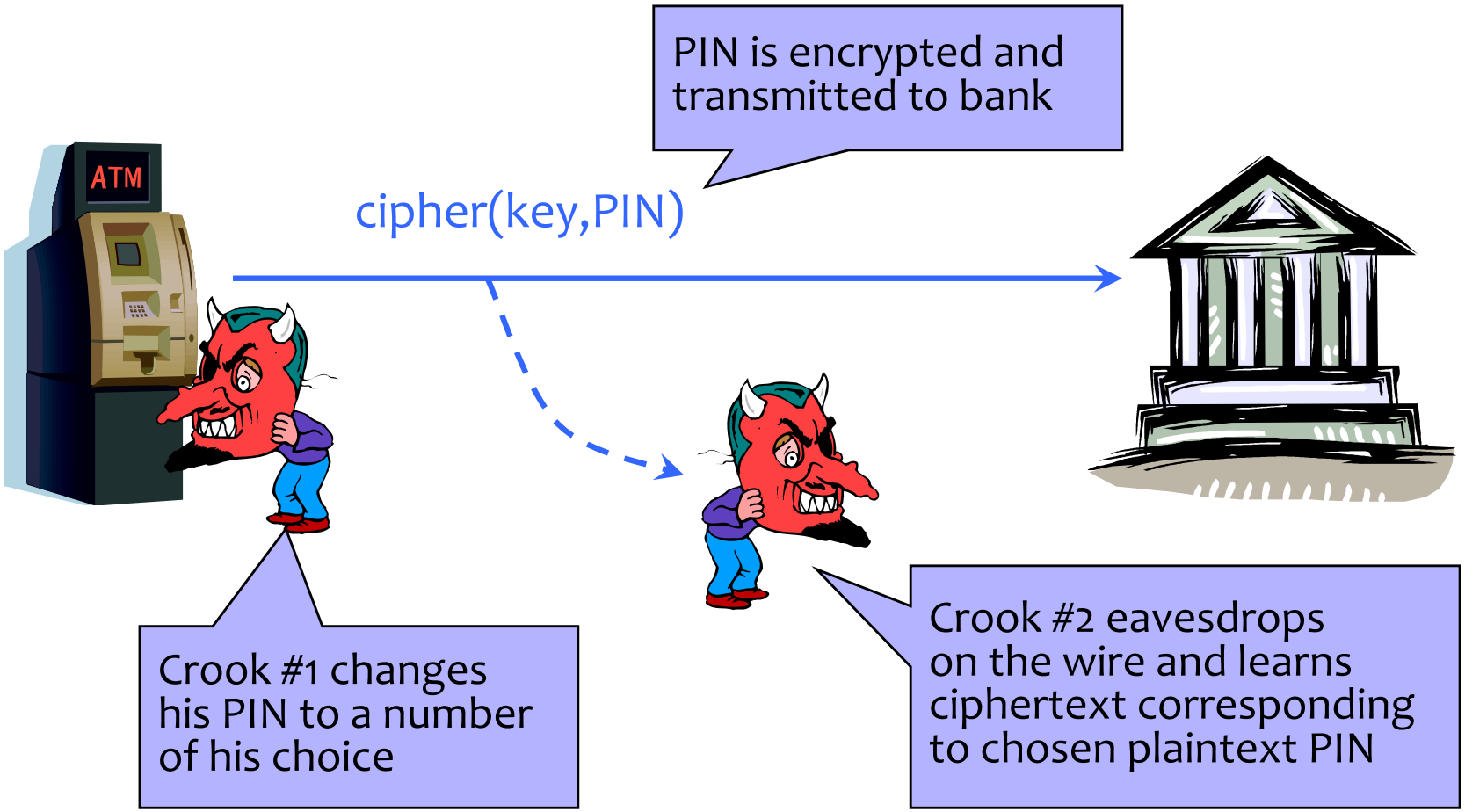
When is an Encryption Scheme "Secure"?

- 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?

- Attackers knows ciphertext and encryption algthm
  - What else does the attacker know? Depends on the application in which the cipher is used!
- Ciphertext-only attack
- KPA: Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- **CPA: Chosen-plaintext attack (even stronger)**
  - **Can obtain ciphertext for any plaintext of his choice**
- CCA: Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext except the target

Chosen Plaintext Attack (CPA)



… repeat for any PIN value

Chosen Plaintext Security Game

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

→ Any deterministic, stateless symmetric encryption scheme (such as ECB mode) is insecure against chosen plaintext attacks.

Very Informal Intuition

Minimum security requirement for a modern encryption scheme

- Security against chosen-plaintext attack (CPA)
  - Ciphertext leaks no information about the plaintext
  - Even if the attacker correctly guesses the plaintext, he cannot verify his guess
  - Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
- Security against chosen-ciphertext attack (CCA)
  - Integrity protection – it is not possible to change the plaintext by modifying the ciphertext

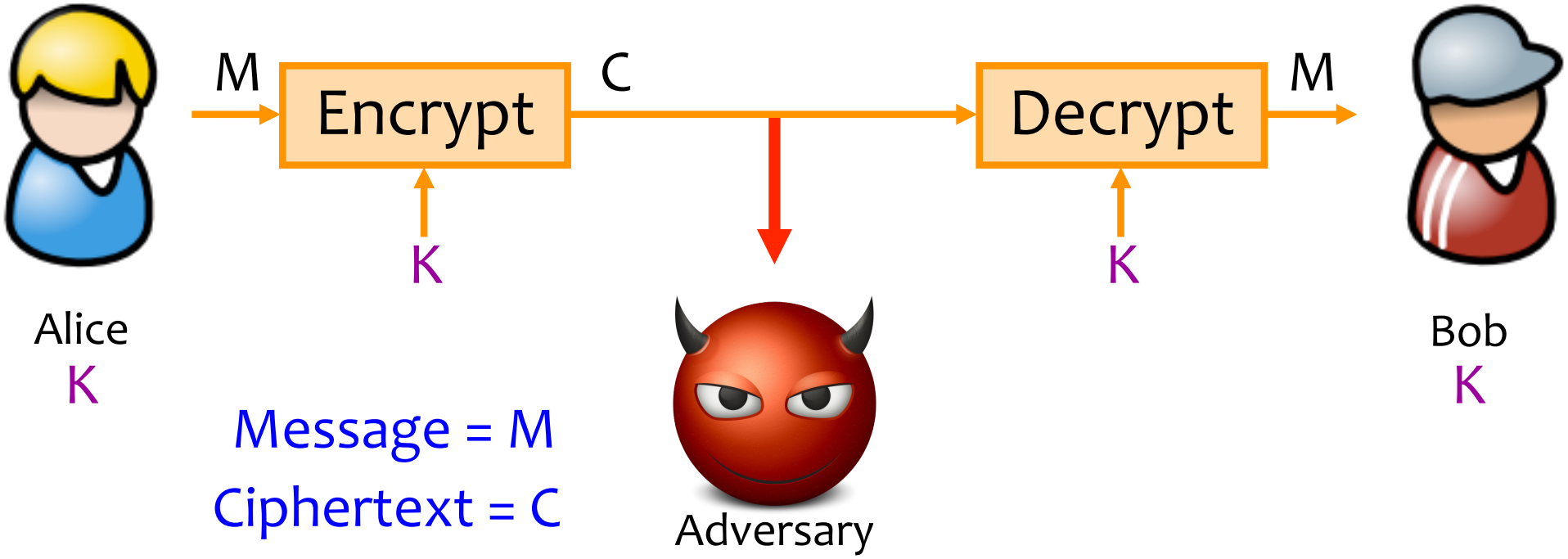
Why Hide Everything?

- 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)

Message Authentication Codes

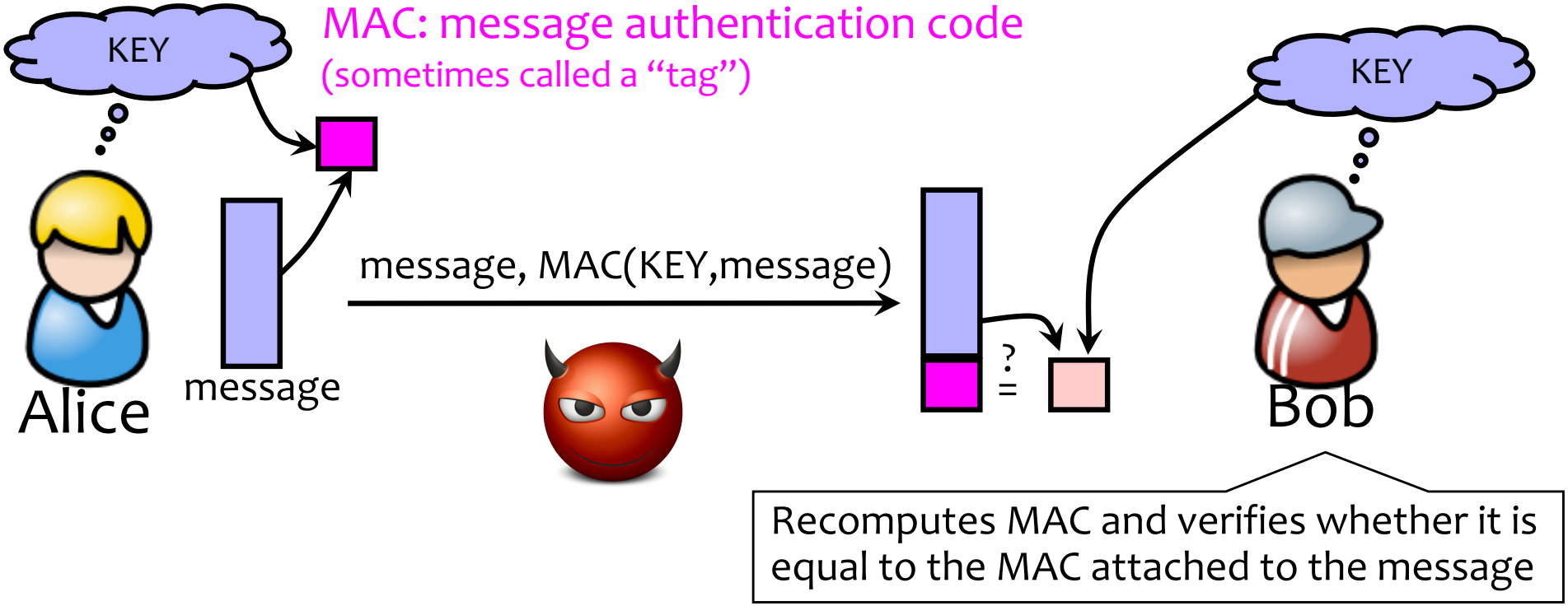
So Far: Achieving Privacy

Encryption schemes: A tool for protecting privacy.



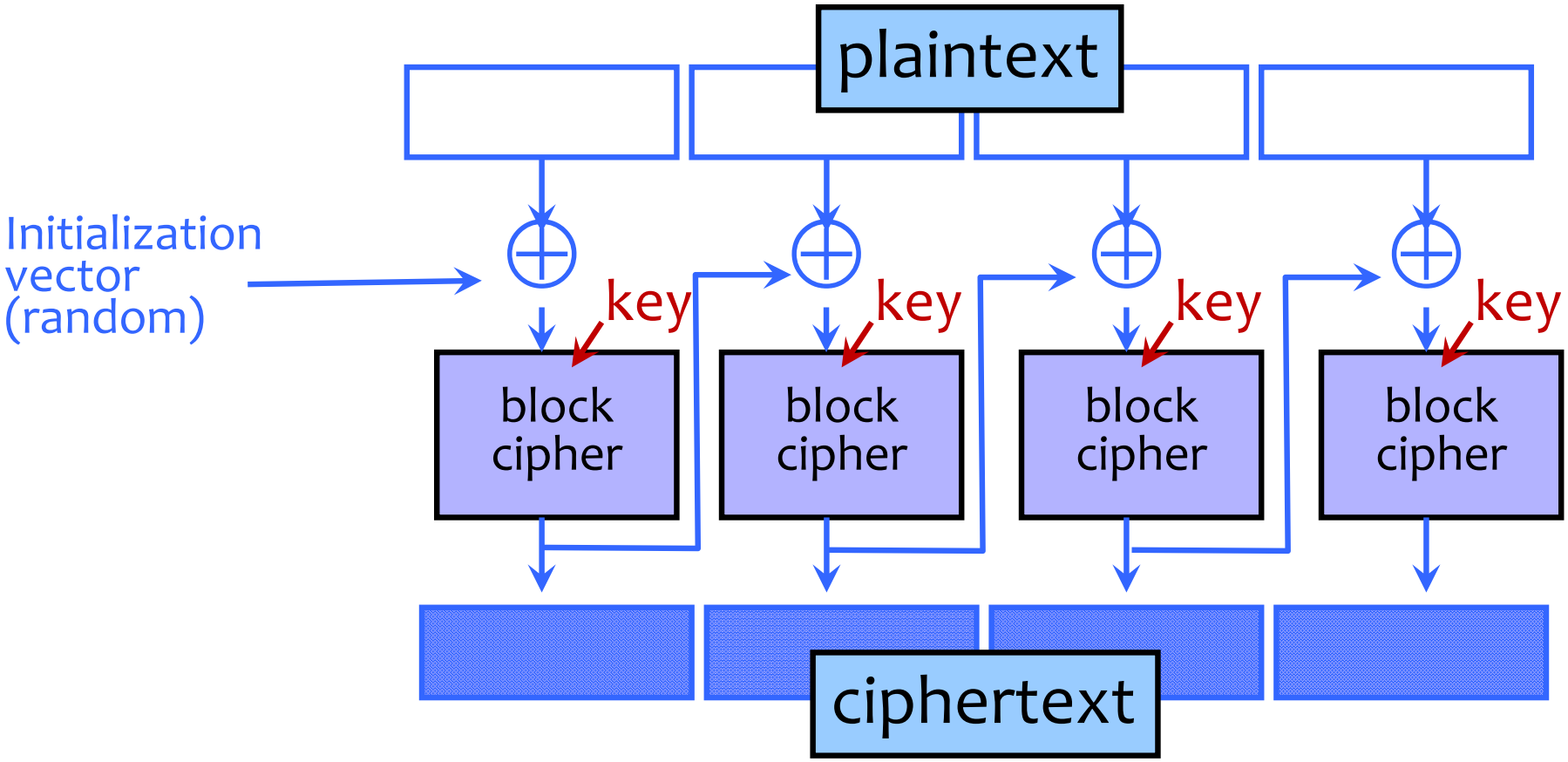
Now: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



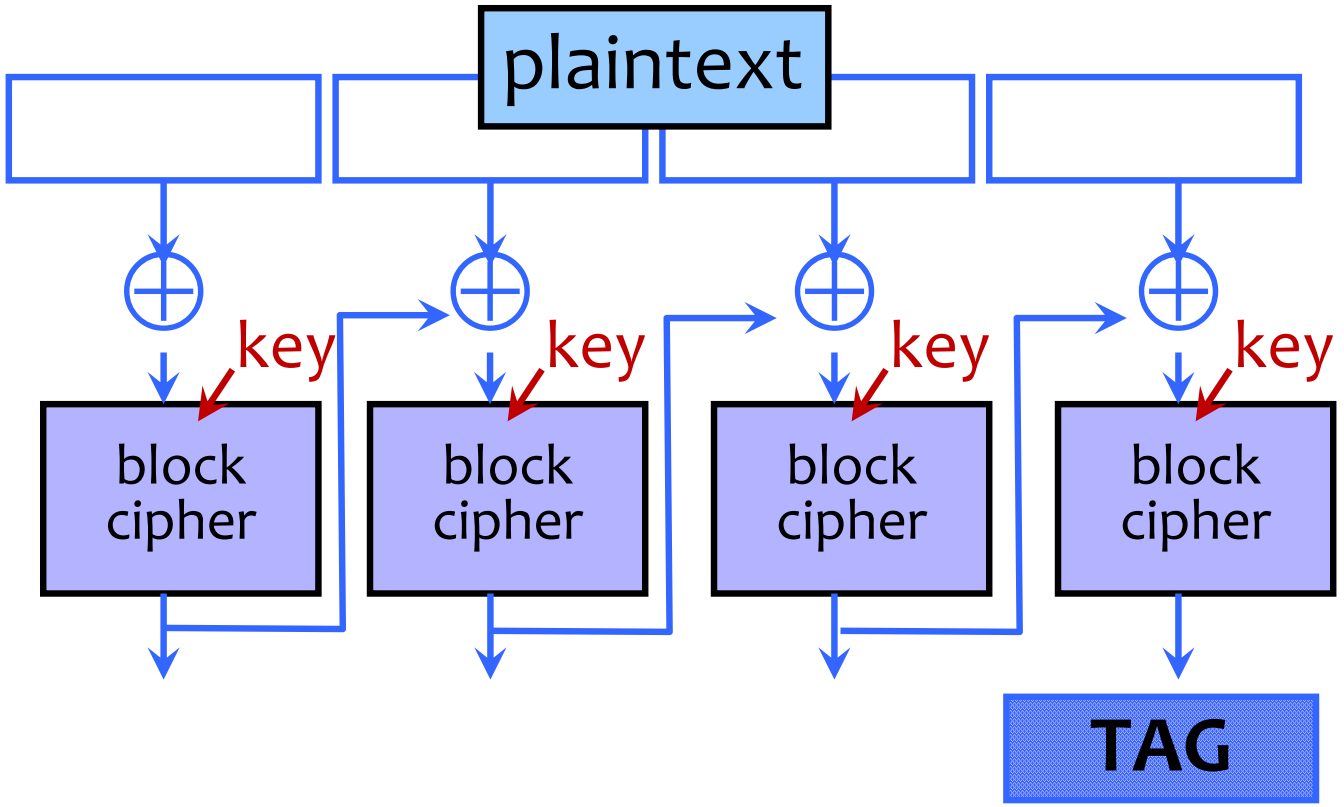
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

Reminder: CBC Mode Encryption



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

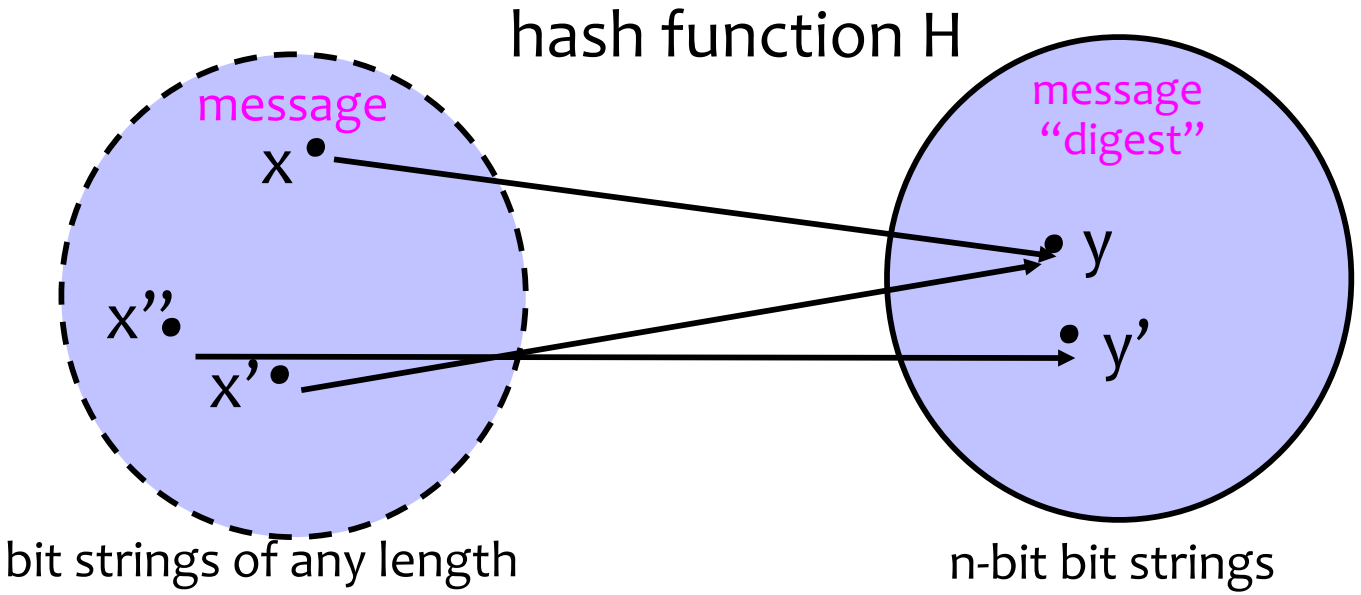
CBC-MAC



- Not secure when system may MAC messages of different lengths.
- NIST recommends a derivative called CMAC [FYI only]

Hash Functions

Hash Functions: Main Idea



- Hash function H is a lossy compression function
  - Collision: h(x)=h(x') for distinct inputs x, x'
- H(x) should look "random"
  - Every bit (almost) equally likely to be 0 or 1
- Cryptographic hash function needs a few properties…

Property 1: 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.

Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x)=h(x')$

Birthday Paradox

- Are there two people in the first 1/3 of this classroom that have the same birthday?
  - 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 birthday "collision" with a room of only 23 people.
    - For simplicity, approximate when we expect a collision as sqrt(365).
- Why is this important for cryptography?
  - 2^{128} different 128-bit values
    - Pick one value at random. To exhaustively search for this value requires trying on average 2^{127} values.
    - Expect "collision" after selecting approximately 2^{64} random values.
    - 64 bits of security against collision attacks, not 128 bits.

Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x)=h(x')$
- Birthday paradox means that brute-force collision search is only $O(2^{n/2})$, *not* $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$

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 y=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. ½

Property 3: 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
- Weak collision resistance does not imply collision resistance.

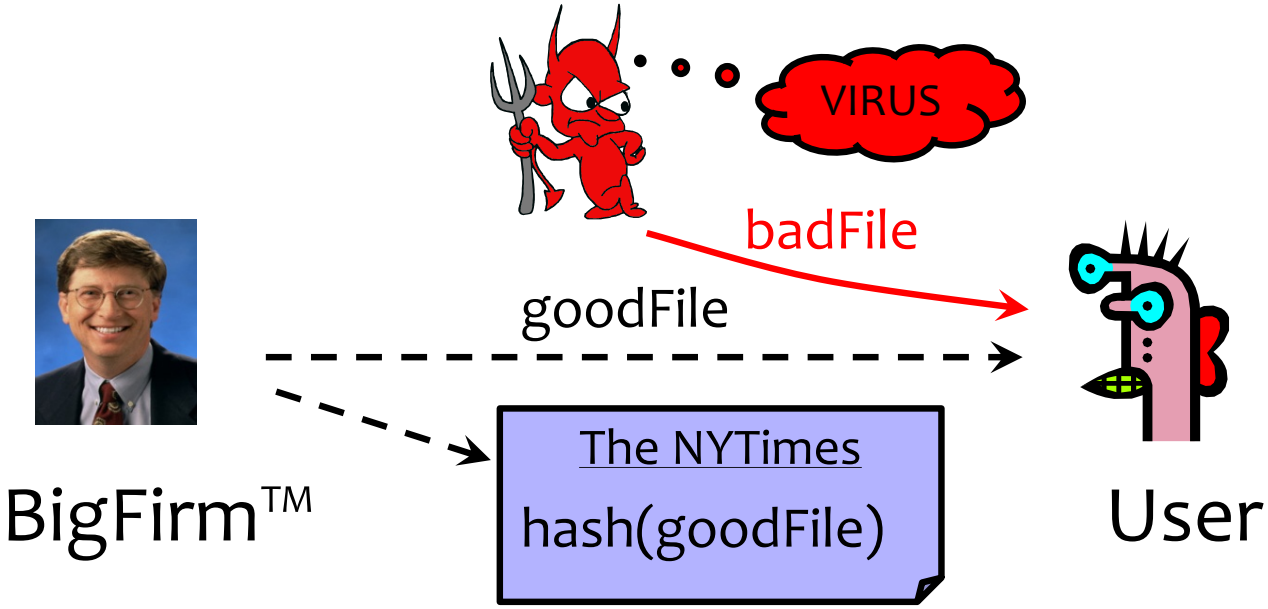
Hashing vs. Encryption

- Hashing is one-way. There is no "un-hashing"
  - A ciphertext can be decrypted with a decryption key… hashes have no equivalent of "decryption"
- Hash(x) looks "random" but can be compared for equality with Hash(x')
  - Hash the same input twice → same hash value
  - Encrypt the same input twice → different ciphertexts
- Crytographic hashes are also known as "cryptographic checksums" or "message digests"

Application: Password Hashing

- Instead of user password, store hash(password)
- When user enters a password, compute its hash and compare with the entry in the password file
  - System does not store actual passwords!
  - Cannot go from hash to password!
- Why is hashing better than encryption here?
- Does hashing protect weak, easily guessable passwords?

Application: Software Integrity



Goal: Software manufacturer wants to ensure file is received by users without modification.

Idea: given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)

Which Property Do We Need?

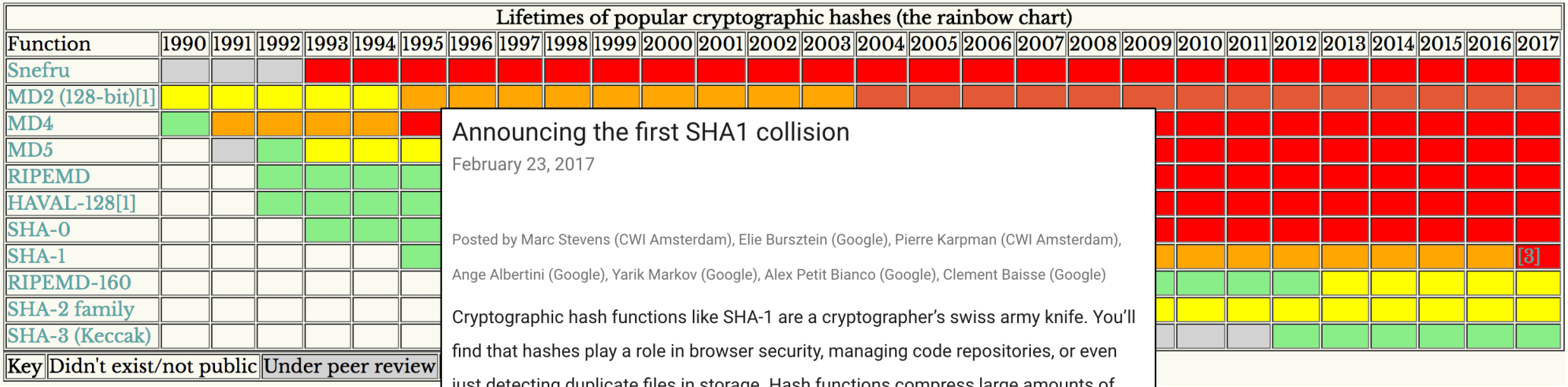
- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover the/a valid password
- Integrity of software distribution **(or lab 1 checkpoint!)**
  - Weak collision resistance
  - But software images are not really random… may need full collision resistance if considering malicious developers
- 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: standard released by NIST in August 2015

Lifetimes of Hash Functions

http://valerieaurora.org/hash.html



Lifetimes of popular cryptographic hashes (the rainbow chart)

Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Snefru																												
MD2 (128-bit)[1]																												
MD4																												
MD5																												
RIPEMD																												
HAVAL-128[1]																												
SHA-0																												
SHA-1																												[3]
RIPEMD-160																												
SHA-2 family																												
SHA-3 (Keccak)																												

Key: Didn't exist/not public | Under peer review

Announcing the first SHA1 collision

February 23, 2017

Posted by Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), Yarik Markov (Google), Alex Petit Bianco (Google), Clement Baisse (Google)

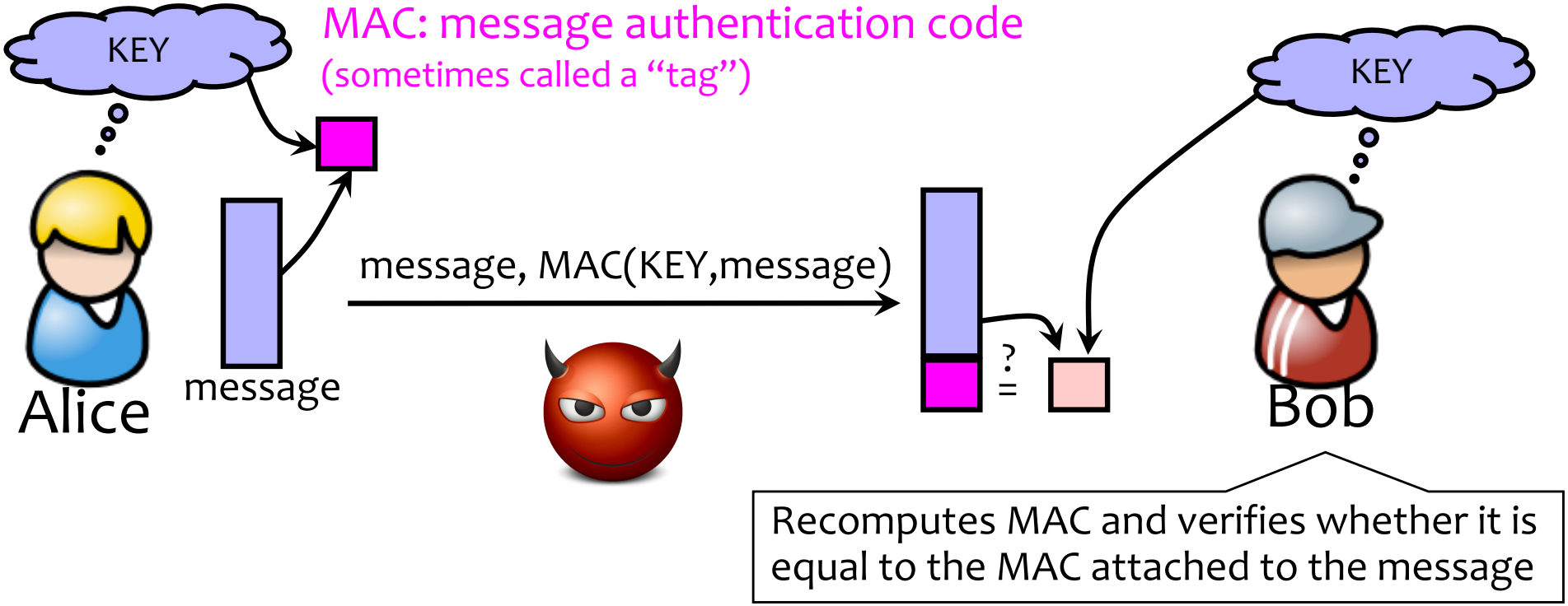
Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security, managing code repositories, or even just detecting duplicate files in storage. Hash functions compress large amounts of data into a small message digest. As a cryptographic requirement for wide-spread use, finding two messages that lead to the same digest should be computationally infeasible. Over time however, this requirement can fail due to attacks on the mathematical underpinnings of hash functions or to increases in computational power.

Today, more than 20 years after of SHA-1 was first introduced, we are announcing the first practical technique for generating a collision. This represents the culmination of two years of research that sprung from a collaboration between the CWI Institute in Amsterdam and Google. We've summarized how we went about generating a collision below. As a proof of the attack, we are releasing two PDFs that have identical SHA-1 hashes but different content.

https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



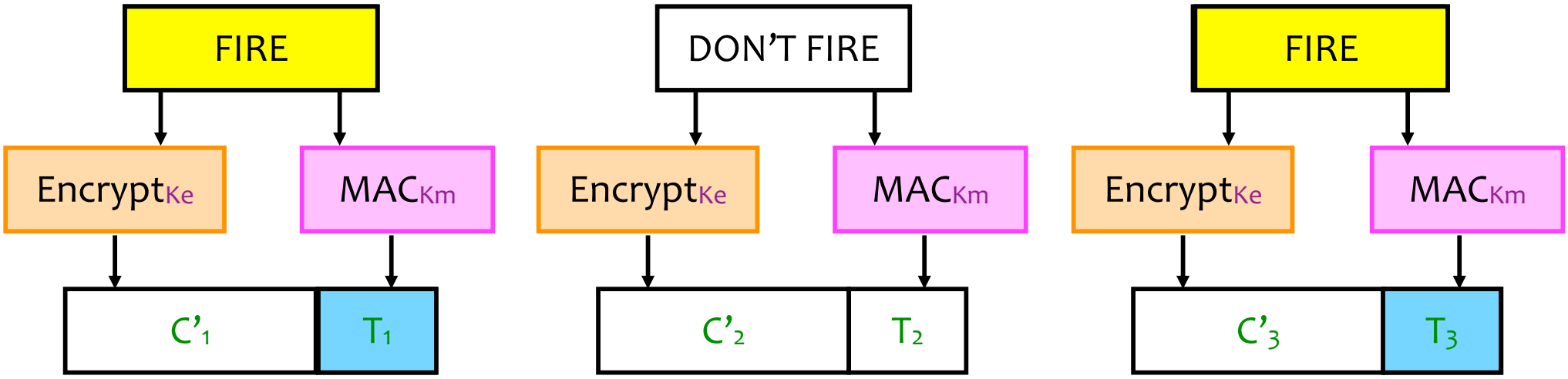
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

HMAC

- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec
- Why not encryption?
  - Hashing is faster than block ciphers in software
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

Authenticated Encryption

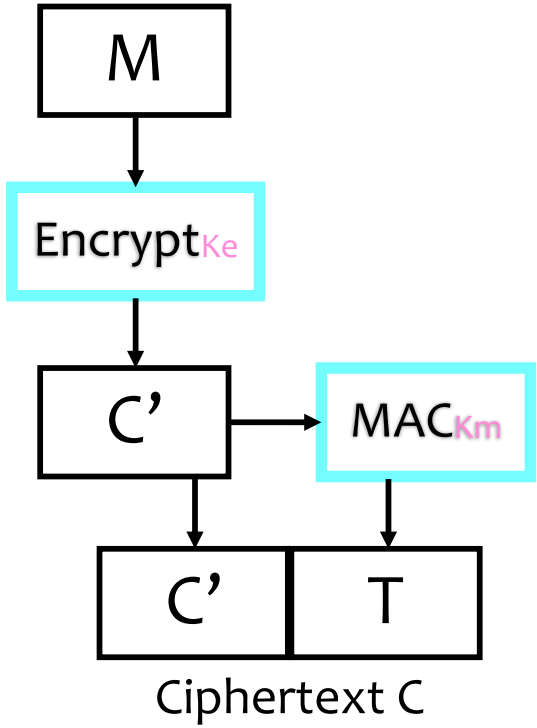
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.
- But be careful!
  - Obvious approach: Encrypt-and-MAC
  - Problem: MAC is deterministic! same plaintext → same MAC



Authenticated Encryption

- Instead:
  Encrypt *then* MAC.

- (Not as good:
  MAC-then-Encrypt)



Encrypt-then-MAC