ECE 455: CYBERSECURITY

Lecture #5

Daniel Gitzel

Announcement

- **Read papers for quiz next week.**
- **Start thinking about final project.**
  - Assignment will be posted on google group.
  - Proposals due 10/21.
- **Midterm will be on 10/21.**
  - Covers all material from lectures and papers up to and including cryptography (Lectures 0 through 6).

In the news...

- **“The Rise of One-Time Password Interception Bots”**
  - OTP interception service — Otp[.]agency — advertised a web-based bot designed to trick targets into giving up OTP tokens.
  - See Krebs on Security article:
  - https://krebsonsecurity.com/2021/09/the-rise-of-one-time-password-interception-bots/

CRYPTOGRAPHY

Introduction

- **The art and science of protecting information**
  - Keeping it confidential, if we want privacy
  - Protecting its integrity, if we want to prevent tampering
  - Assuring authenticity, if we don’t want forgeries
- **“Secure communication over an insecure channel”**

Cautions and Thoughts

- **Cryptography only one small piece of a larger system**
- **Must protect entire system**
  - Physical security
  - Operating system security
  - Network security
  - Users
  - Cryptography (following slides)
- **Recall the weakest link**

- **“Those who think that cryptography can solve their problems don’t understand cryptography and don’t understand their problems.”**

More Security, More Problems

- **RFIDs in car keys:**
  - RFIDs in car keys make it harder to hot-wire a car
  - Result: Car-jackings increased
- **Biometric Locks:**
  - Thieves cutoff fingers!

boingboing
A Directory Of Wonderful Things

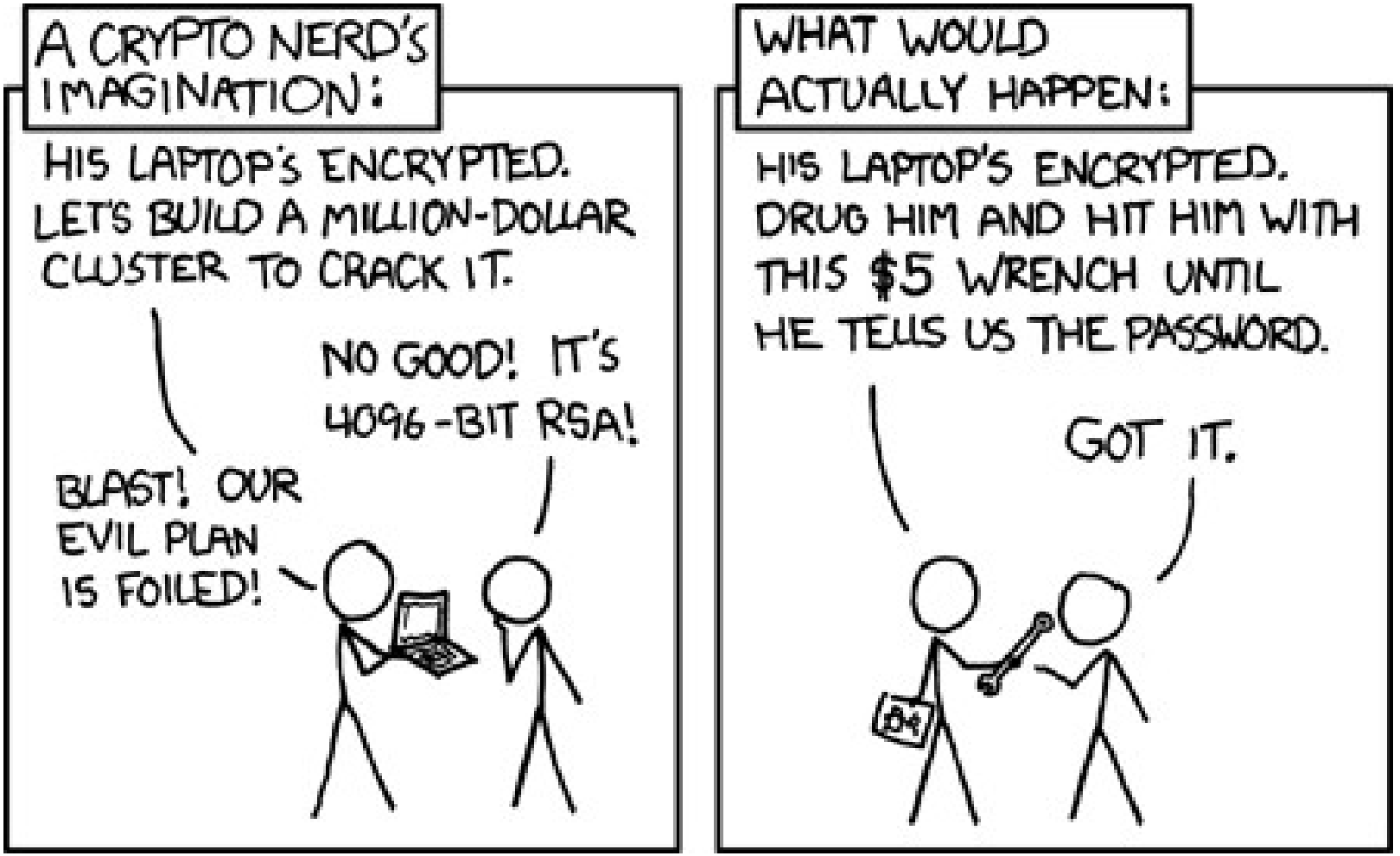
Biometric car lock defeated by cutting off owner's finger

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.

More Security, More Problems



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**

Ingredient: Randomness

- **Many applications (especially security ones) require randomness**
- **Explicit uses:**
  - Generate secret cryptographic keys
  - Generate random initialization vectors for encryption
- **Other “non-obvious” uses:**
  - Generate passwords for new users
  - Shuffle the order of votes (in an electronic voting machine)
  - Shuffle cards (for an online gambling site)

C’s rand() Function

- **C has a built-in random function: rand()**
- **Problem: don’t use rand() for security-critical applications!**
  - Given a few sample outputs, you can predict subsequent ones

```
unsigned long int next = 1;
/* rand:  return pseudo-random integer on 0..32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
/* srand:  set seed for rand() */
void srand(unsigned int seed) {
    next = seed;
}
```

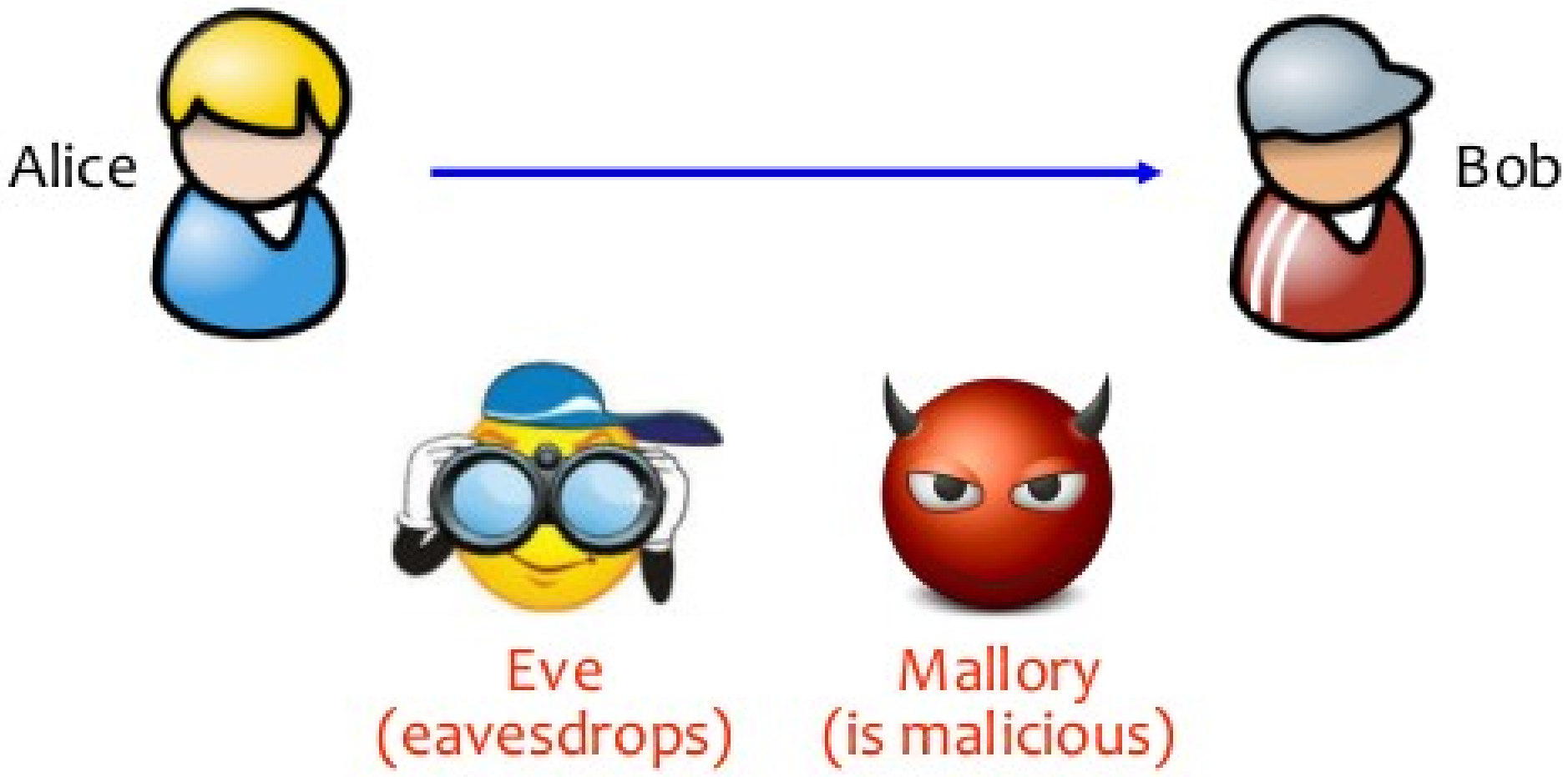
PS3 and Randomness

- **2010/2011: Hackers found/released private root key for Sony's PS3**
- **Key used to sign software – now can load any software on PS3 and it will execute as "trusted"**
- **Due to bad random number: same "random" value used to sign all system updates**

Obtaining Pseudorandom Numbers

- **For security applications, want “cryptographically secure pseudorandom numbers”**
- **Libraries include cryptographically secure pseudorandom number generators (CSPRNG)**
- **Linux:**
  - /dev/random
  - /dev/urandom - nonblocking, possibly less entropy
- **Internally:**
  - Entropy pool gathered from multiple sources
  - e.g., mouse/keyboard timings
- **Challenges with embedded systems, saved VMs**

Alice, Bob, and the Usual Suspects



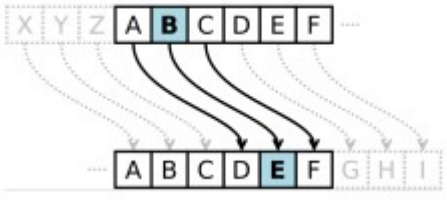
CIPHERS

History

- **Substitution Ciphers**
  - Caesar Cipher
- **Transposition Ciphers**
- **Codebooks**
- **Machines**
- **Recommended Reading: The Codebreakers by David Kahn and The Code Book by Simon Singh.**

History: Caesar Cipher (Shift Cipher)

- **Plaintext letters are replaced with letters a fixed shift away in the alphabet.**
- **Example:**
  - Plaintext: The quick brown fox jumps over the lazy dog
  - Key: Shift 3
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - DEFGHIJKLMNOPQRSTUVWXYZABC
  - Ciphertext: WKHTX LFNEU RZQIR AMXPS VRYHU WKHOD CBGRJ



History: Caesar Cipher (Shift Cipher)

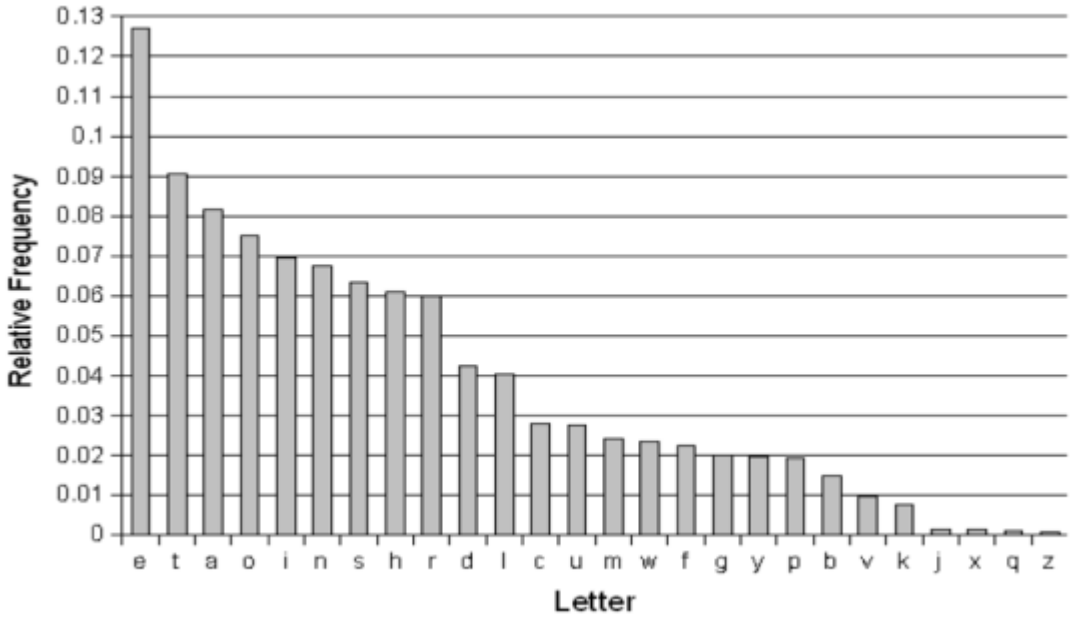
- **ROT13: shift 13 (encryption and decryption are symmetric)**
- **What is the key space?**
  - 26 possible shifts.
- **How to attack shift ciphers?**
  - Brute force.

History: Substitution Cipher

- **Super-set of shift ciphers: each letter is substituted for another one.**
  - Add a secret key
- **Example:**
  - Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Cipher: ZEBRASCDFGHIJKLMNOPQTUVWXY
- **“State of the art” for thousands of years**
  - Now found in newspaper “puzzles pages”

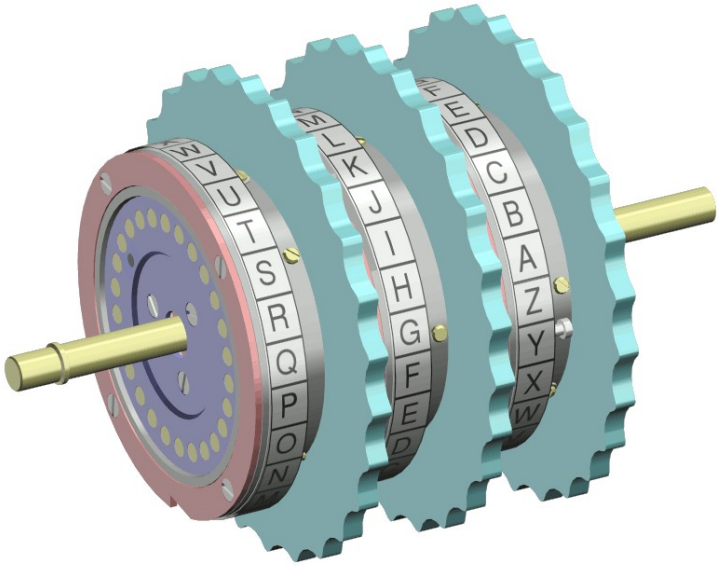
History: Substitution Cipher

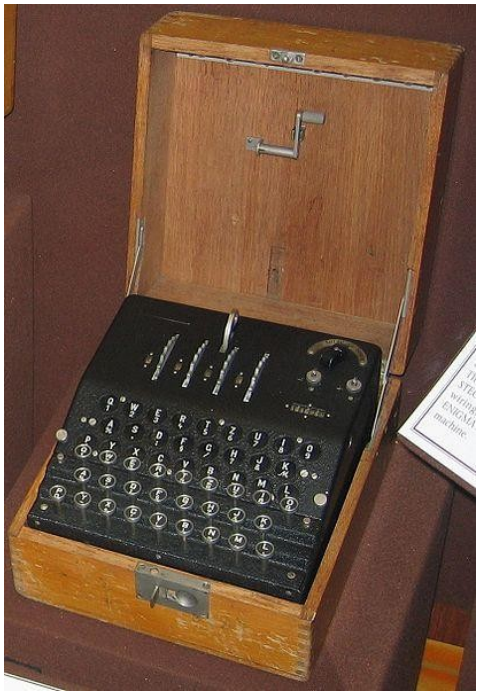
- **What is the key space? 26! ~= 2^88**
- **How to attack?**
  - Frequency analysis



History: Enigma Machine

- **Uses rotors (substitution cipher) that change position after each key.**
- **Key = initial setting of rotors**
- **Key space = 26^n for n rotors**





Primer: Modular Arithmetic

- **Basis for many modern cryptographic algorithms.**
- **Let m be an integer (the modulus)**
  - define an equivalence relation *mod* m on the set of integers
  - a = b *mod* m if and only if
  - a - b = λ · m for some integer λ.
- **We say “a is equivalent to b modulo m”.**
- **This equivalence relation divides the set of integers into m equivalence classes**
  - $(a)_m = \{ b \mid a = b \bmod m\}, 0 \leq a < m$
  - we write a mod m for $(a)_m$.

Primer: Modular Arithmetic

- **The following properties hold:**
  - (a mod m) + (b mod m) = (a+b) mod m,
  - (a mod m) · (b mod m) = (a · b) mod m,
  - for every a ≠ 0 mod p, p prime, there exists an integer a^-1 so that a·a^-1 = 1 mod p.
- **Multiplicative order modulo p:**
  - Let p be a prime and a an arbitrary integer
  - the multiplicative order of a modulo p is the smallest integer n so that a^n = 1 mod p.

Fermat's Little Theorem

- **for p prime and a ≠ 0 mod p**
- **we have a^(p-1) = 1 mod p**
- **Example: p = 5,**
  - 2^4 = 16 ≡ 1 mod 5
  - 3^4 = 81 ≡ 1 mod 5
  - 4^4 = 256 ≡ 1 mod 5
- **Note: when computing a^x mod p, you are working modulo p-1 in the exponent**
- **Corollary for n = p·q, e·d = 1 mod lcm(p-1,q-1):**
- **for a, 0 < a < n, we have a^(e·d) = a mod n.**

Difficult Problems

- **Discrete Logarithm Problem (DLP):**
  - Given a prime modulus p, a basis a, and a value y, find the discrete logarithm of y, i.e. an integer x so that y = a^x mod p.
- **n-th Root Problem:**
  - Given integers m, n and a, find an integer b so that b^n = a mod m; the solution b is the n-th root of a modulo n.
- **Factorisation:**
  - Find the prime factors of an integer n.
- **With suitable parameters, these problems are a basis for many cryptographic algorithms.**
- **However, not all instances of these problems are difficult to solve.**

How Cryptosystems Work Today

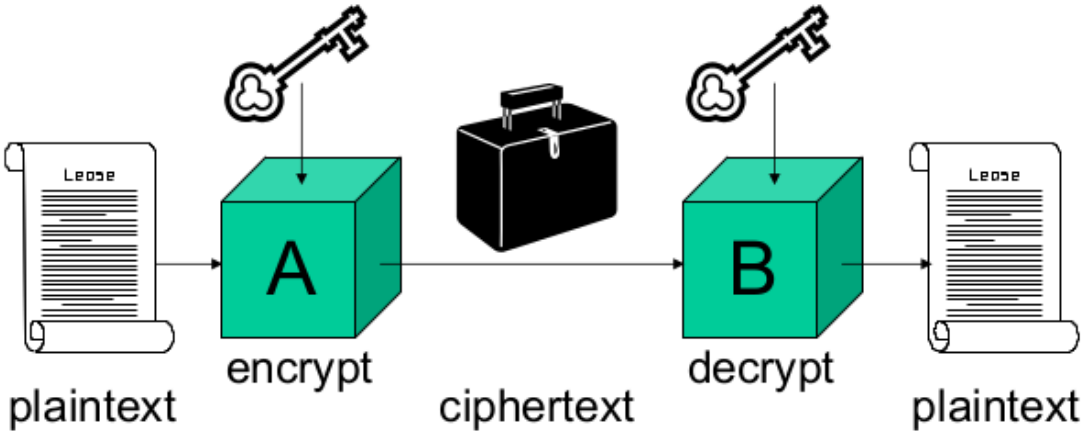
- **Layered approach:**
  - Cryptographic primitives, like block ciphers, stream ciphers, hash functions, and one-way trapdoor permutations
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication
- **Public algorithms (Kerckhoff's Principle)**
- **Security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

“Flavors” of Cryptography

- **Symmetric cryptography**
  - Both communicating parties have access to a shared random string **K**, called the **key**.
  - Classical approach (one and only one key per “lock”)
- **Asymmetric cryptography**
  - Each party creates a public key **PK** and a secret key **SK**.
  - Modern cryptographic approach, no real physical analogy

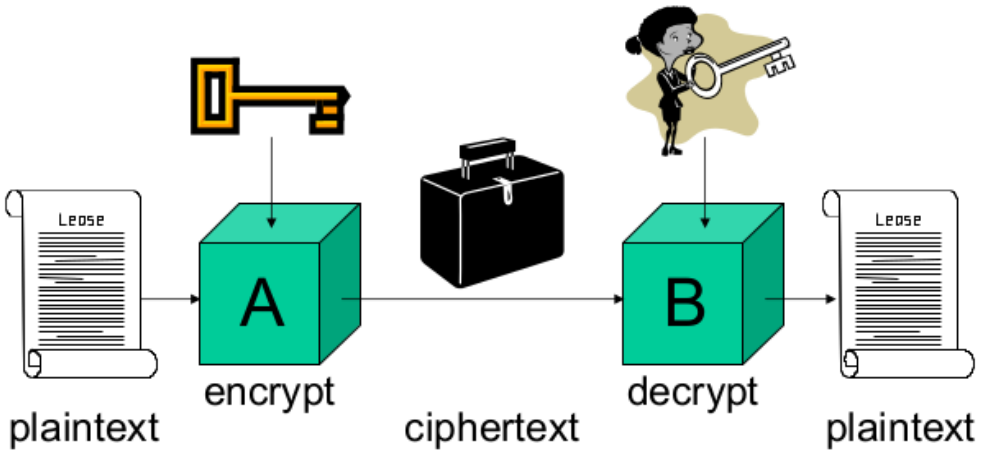
Symmetric Setting

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



Asymmetric Setting

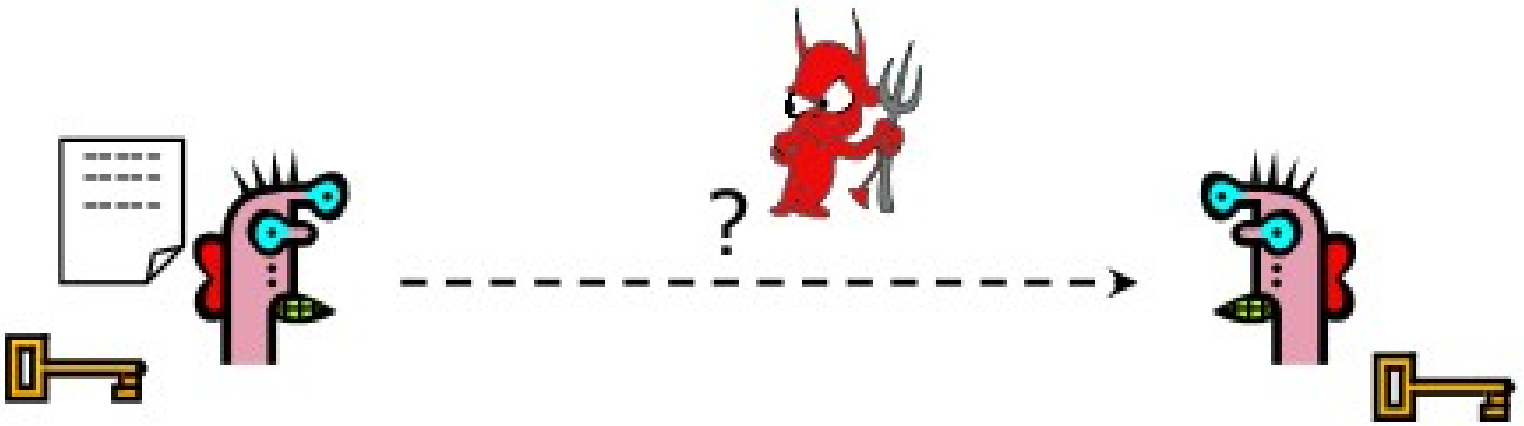
Each party creates a public key PK and a secret key SK.



“Flavors” of Cryptography

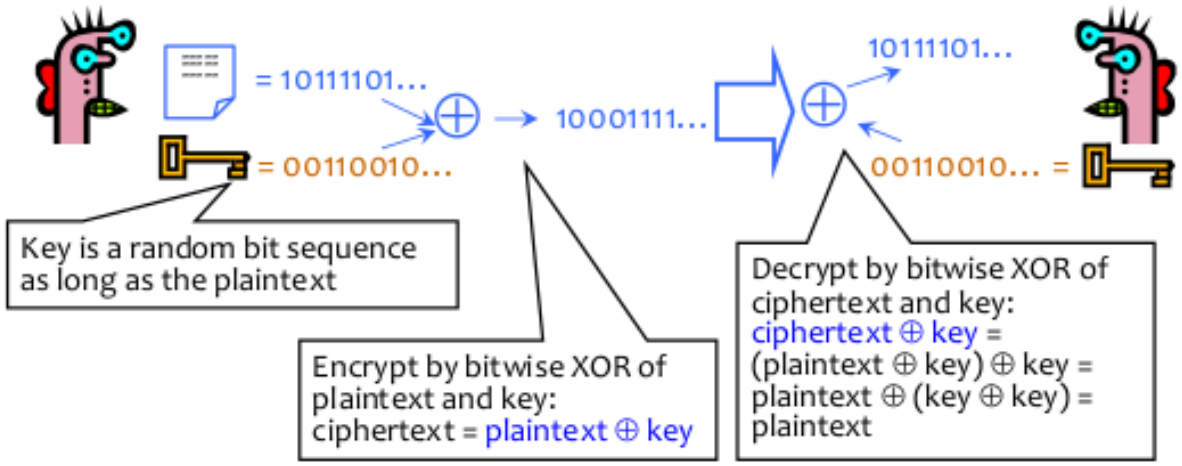
- **Symmetric cryptography**
  - Both communicating parties have access to a shared random string **K**, called the **key**.
  - **Challenge: How do you privately share a key?**
- **Asymmetric cryptography**
  - Each party creates a public key **PK** and a secret key **SK**.
  - **Challenge: How do you validate a public key?**

Confidentiality: Basic Problem



- **Given (Symmetric Crypto): both parties know the same secret.**
- **Goal: send a message confidentially.**

One-Time Pad



- **Cipher achieves perfect secrecy if and only if:**
  - there are as many possible keys as possible plain texts
  - **and** every key is equally likely
- **(Claude Shannon, 1949)**

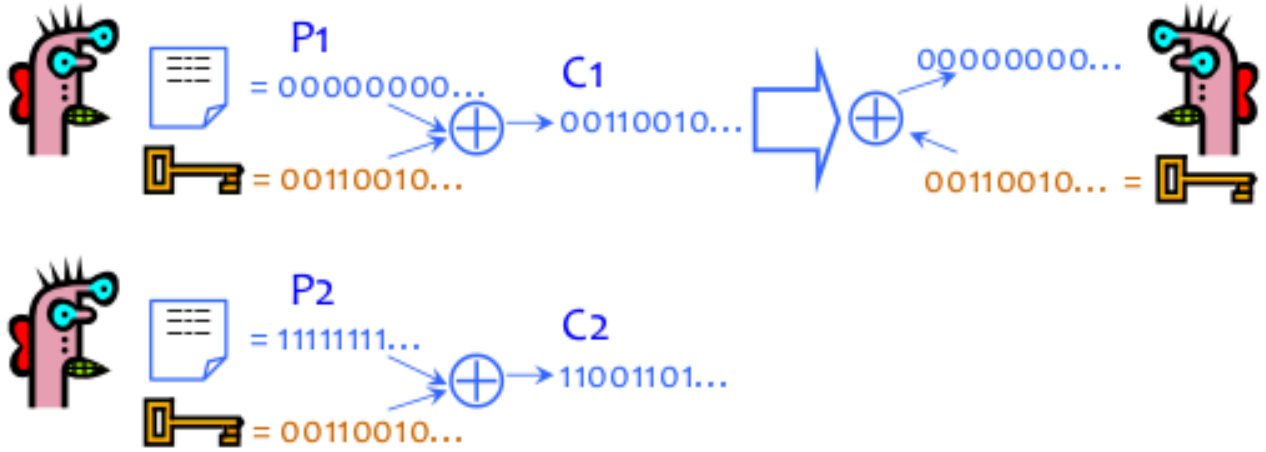
Advantages of One-Time Pad

- **Easy to compute**
  - Encryption and decryption are the same operation
  - Bit-wise XOR is very cheap to compute
- **As secure as theoretically possible**
  - Given a cipher-text, all plain-texts 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 plain-text
    - **But how does sender communicate the key to receiver?**

Problems with One-Time Pad

- **Key must be as long as the plaintext**
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- **Insecure if keys are reused**

Dangers of Reuse



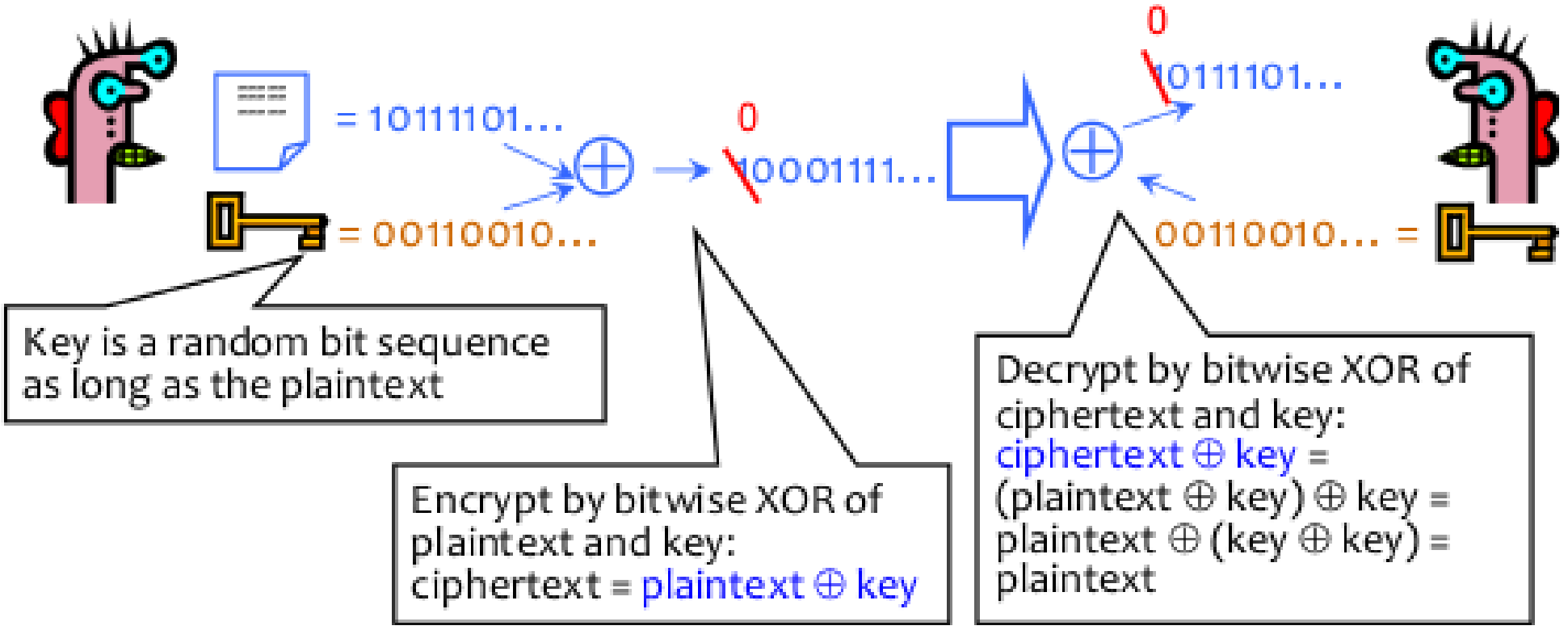
Learn relationship between plain-texts

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

Problems with One-Time Pad

- **Key must be as long as the plain-text**
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- **Insecure if keys are reused**
  - Attacker can obtain XOR of plain-texts

Integrity?


= 10111101...
= 00110010...
0
10001111...
0
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

Problems with One-Time Pad

- **Key must be as long as the plain-text**
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- **Insecure if keys are reused**
  - Attacker can obtain XOR of plain-texts
- **Does not guarantee integrity**
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plain-text, but can easily change it to something else

Reducing Key Size

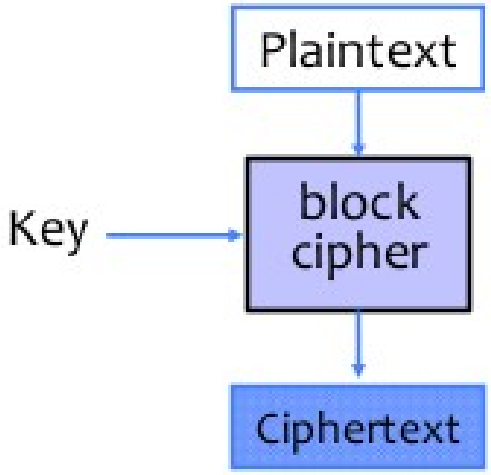
- **What to do when it is infeasible to pre-share huge random keys?**
  - When one-time pad is unrealistic...
- **Use special cryptographic primitives:**
  - block ciphers, stream ciphers
  - Single key can be re-used (with some restrictions)
  - **Not as theoretically secure as one-time pad**

Stream Ciphers

- **One-time pad:**
- **Ciphertext(Key,Message)=Message⊕Key**
  - Key must be a random bit sequence as long as message
- **Idea: replace “random” with “pseudo-random”**
  - Use a pseudo-random number generator (PRNG)
  - PRNG takes a short, truly random secret seed and expands it into a long “random-looking” sequence
    - E.g., 128-bit seed into a 10^6 -bit pseudo-random sequence
- **Ciphertext(Key,Msg)=Msg⊕PRNG(Key)**
  - Message processed bit by bit (unlike block cipher)

Block Ciphers

- **Operates on a single chunk (“block”) of plain-text**
  - 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)



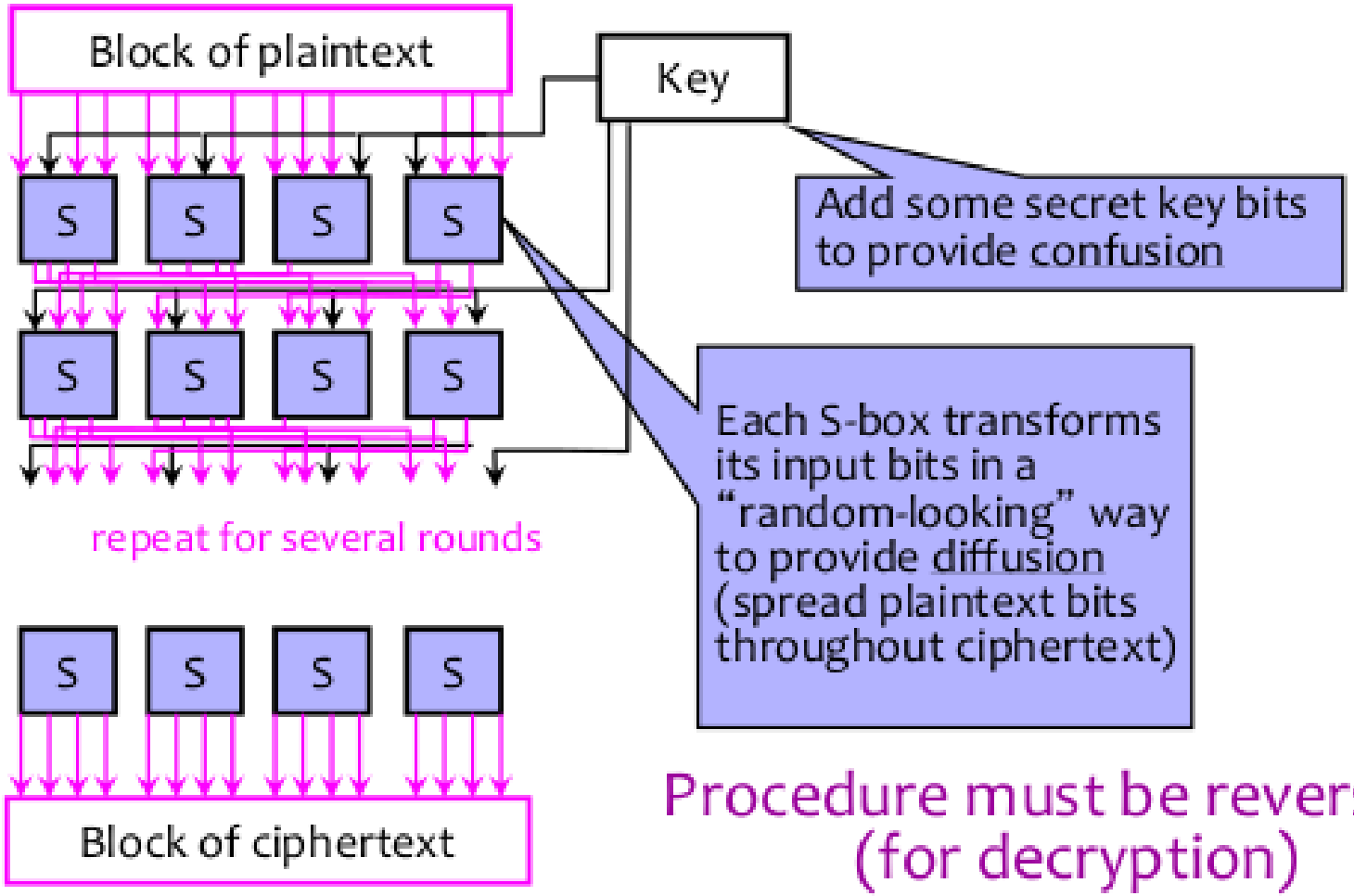
Keyed Permutation

- **Not just shuffling of input bits!**
  - Suppose plain-text = “111”.
  - Then “111” is not the only possible cipher-text!
- **Instead:**
  - Permutation of possible outputs
  - For N-bit input, (2^N)! possible permutations
  - Use secret key to pick a permutation

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)

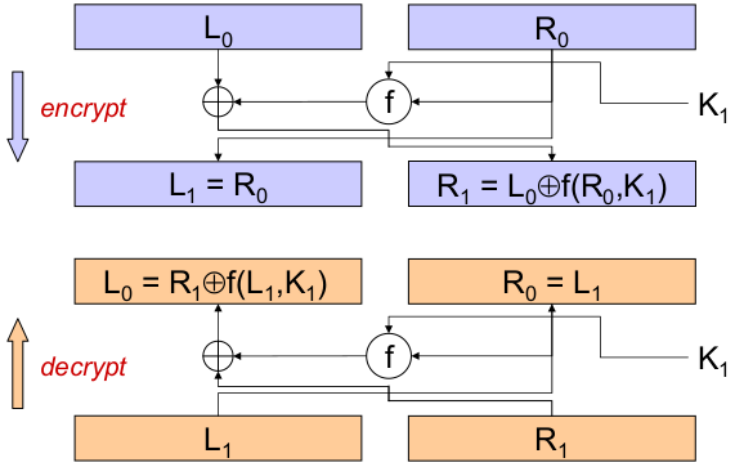


Standard Block Ciphers

- **DES: Data Encryption Standard**
  - **Feistel structure**: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

Feistel Structure

- **Efficient block ciphers usually have a round structure.**
- **Each round depends on a sub-key; each round in itself is not very “secure”.**
- **Security through iteration:**
  - How many rounds do you want?
- **Use the same structures for encryption and decryption.**



DES and 56 bit keys

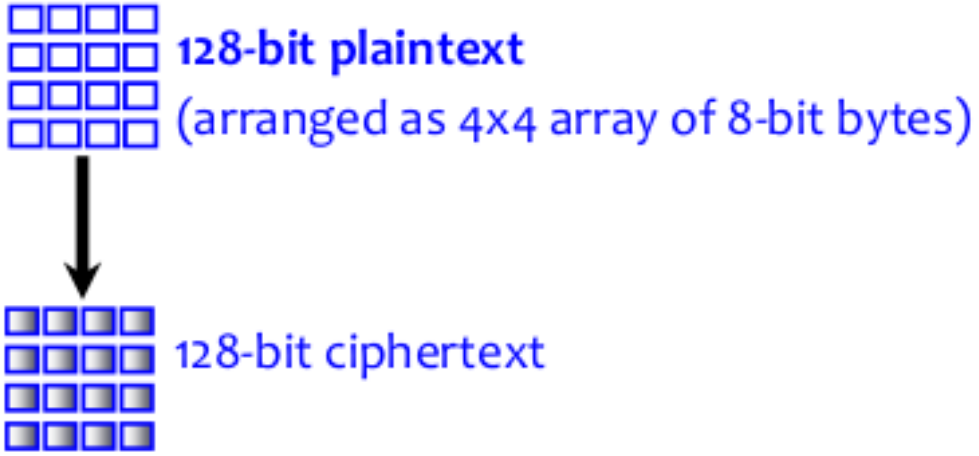
- **56 bit keys are quite short**
- **1999: EFF DES Crack + distributed machines**
  - < 24 hours to find DES key
  - DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity
- **AES: Advanced Encryption Standard**
  - New federal standard as of 2001
- **NIST: National Institute of Standards & Technology**
  - Based on the Rijndael algorithm
- **• Selected via an open process**
  - 128-bit blocks, keys can be 128, 192, or 256 bits

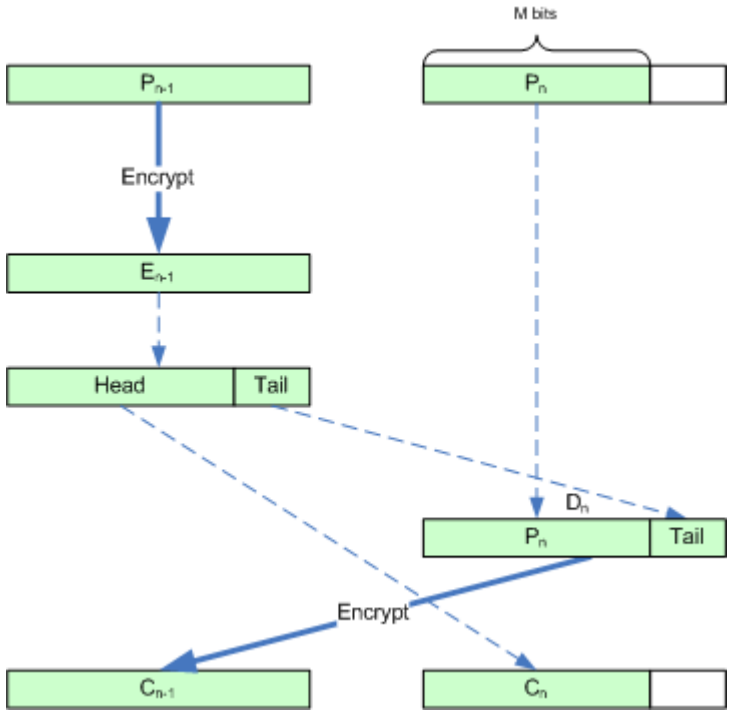
Encrypting a Large Message

- **Our plain-text is larger than 128-bit block size!**
- **What should we do?**



Cipher Text Stealing

- **Block cipher with block size k bytes**
- **n blocks**
  - Block *n-1* is the last full block
  - Block *n* is m < k bytes
- **Encrypt P_{n-1} -> E_{n-1}**
- **Take last k - m bytes of E_{n-1}**
- **Append this “tail” to P_n**
  - This block is now k bytes (full size)
- **Encrypt the augmented P_n -> C_{n-1}**
- **Transmit C_{n-1} and C_n (the “head”)**
- **Cipher text is the same size as the message!**

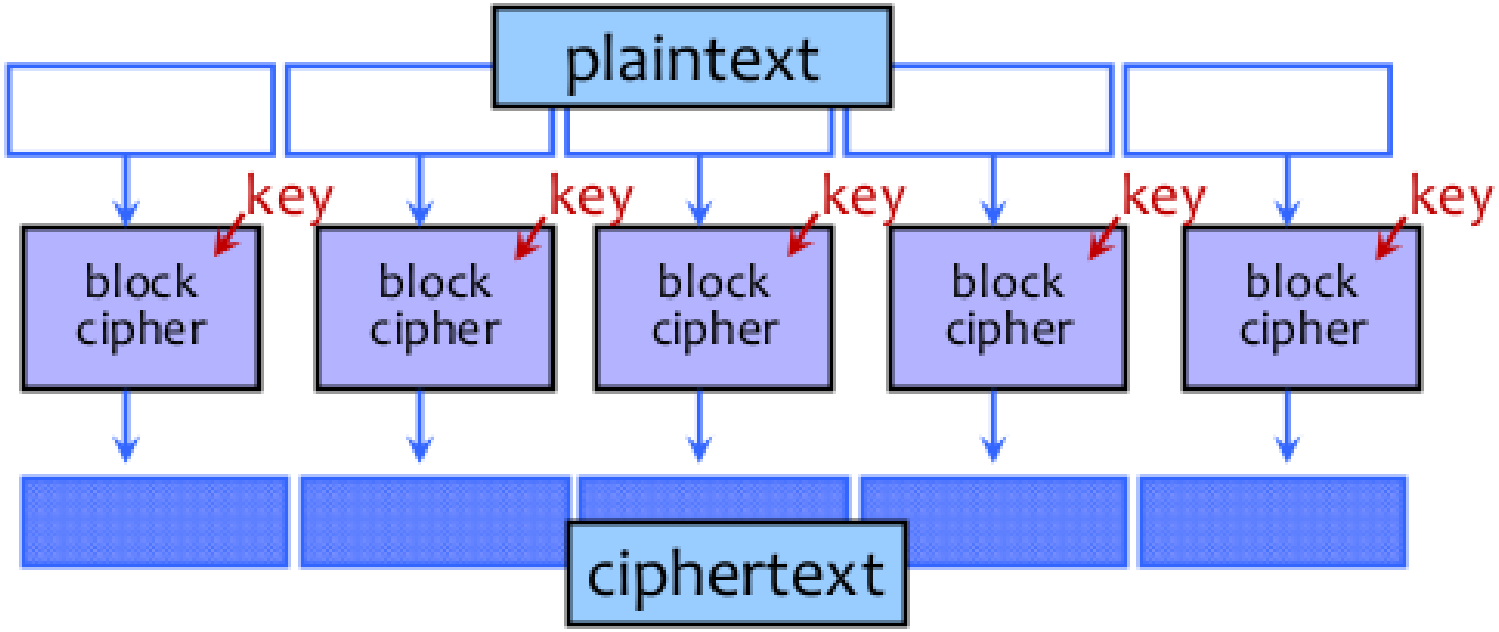


Block Cipher Modes

- **Cipher can operate in different modes**
- **Modes can change a cipher's behavior**
  - Improve computational performance
  - Prevent cipher text manipulation
  - Improve security

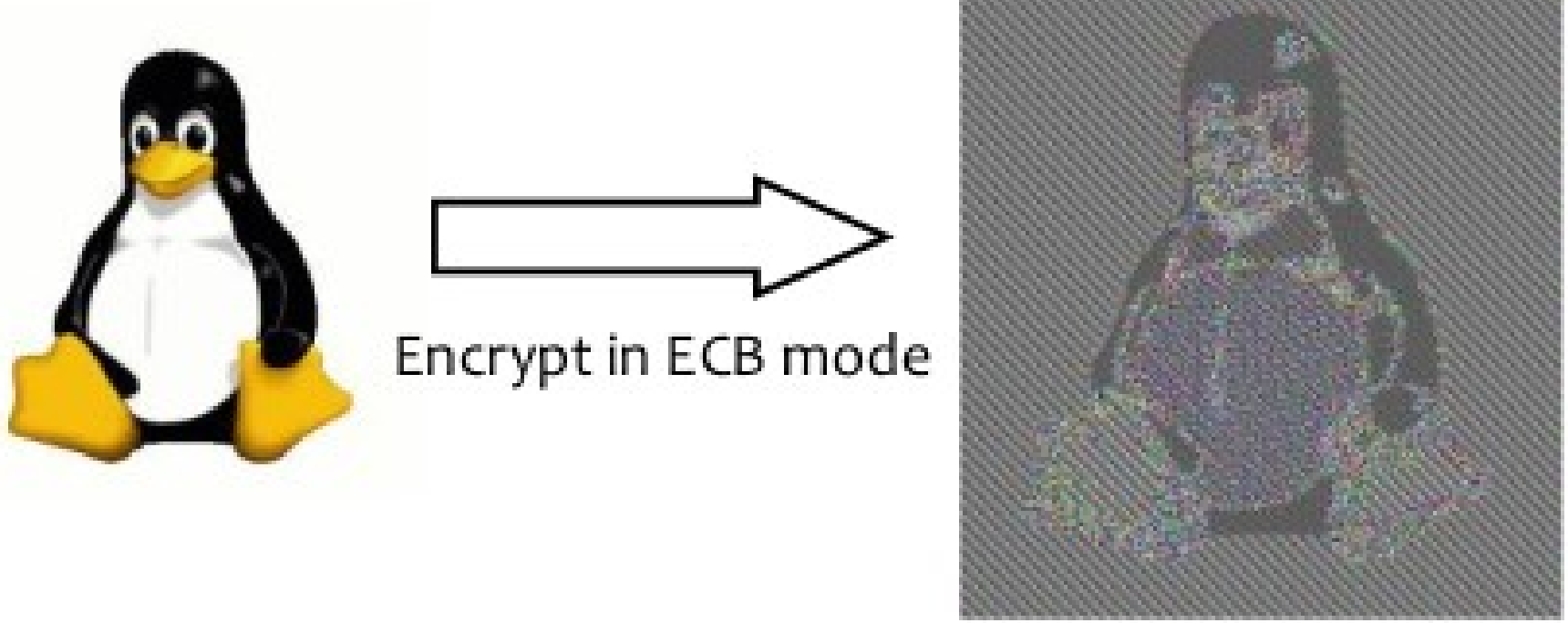
Electronic Code Book (ECB) Mode

- **Identical blocks of plain-text produce identical blocks of cipher-text**
- **No integrity checks: can mix and match blocks**



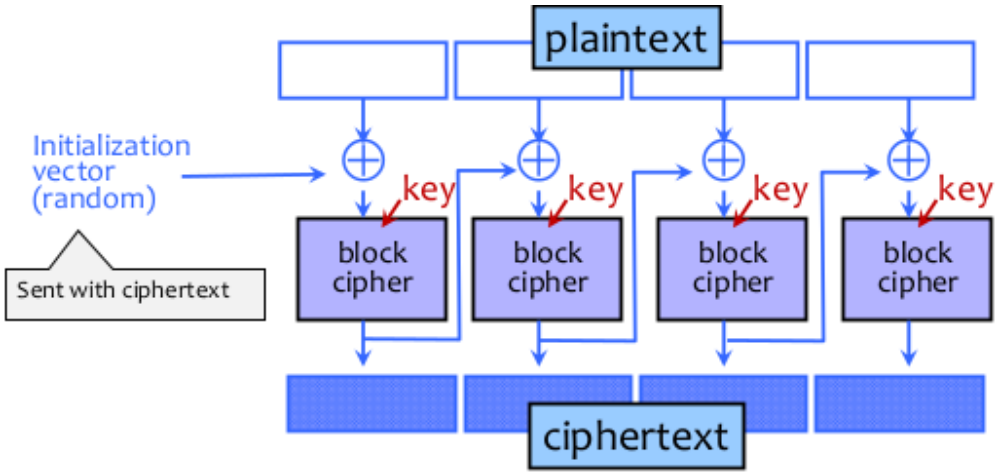
Information Leakage in ECB Mode

Can detect some patterns!

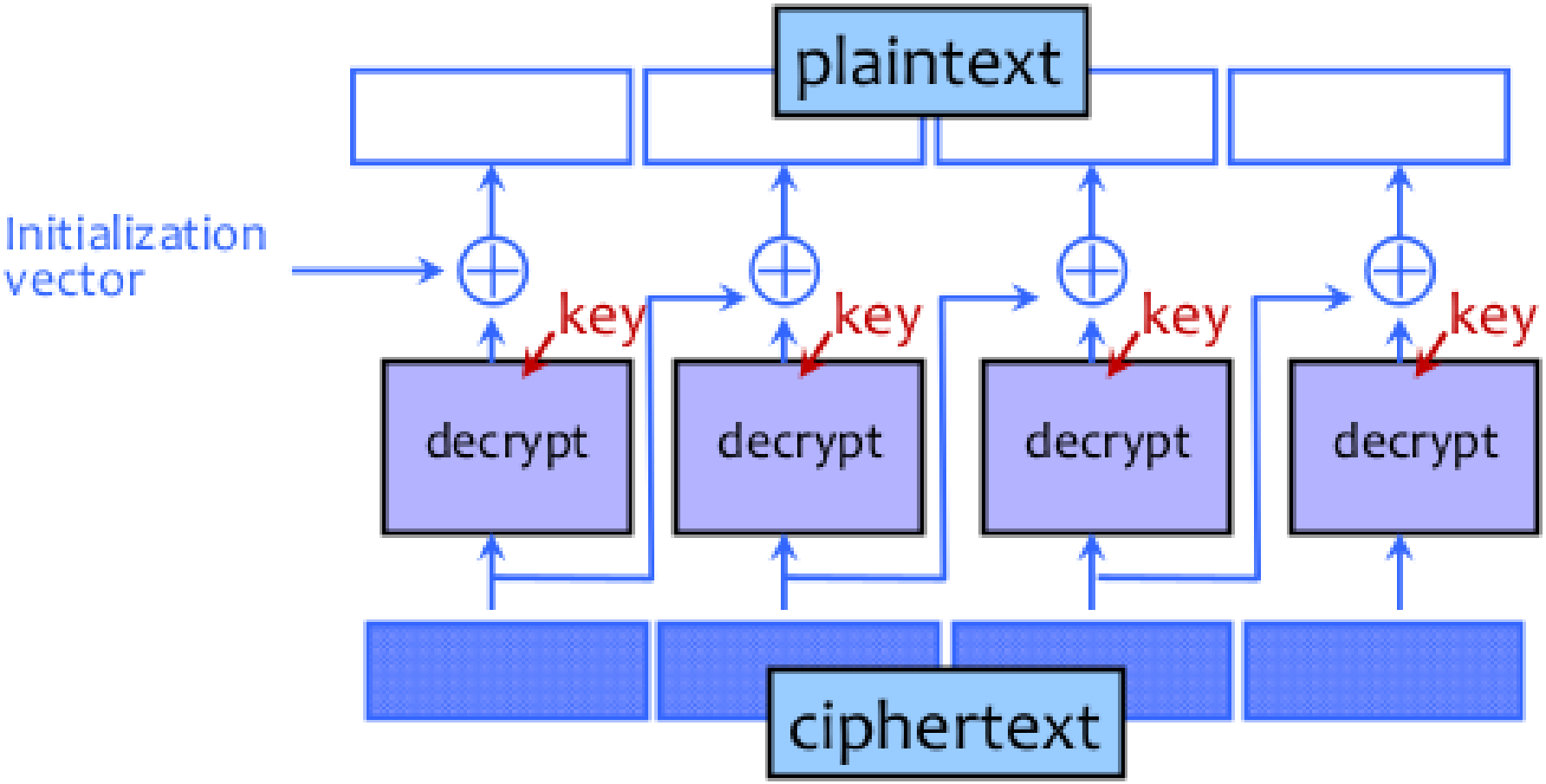


Cipher Block Chaining (CBC) Mode: Encryption

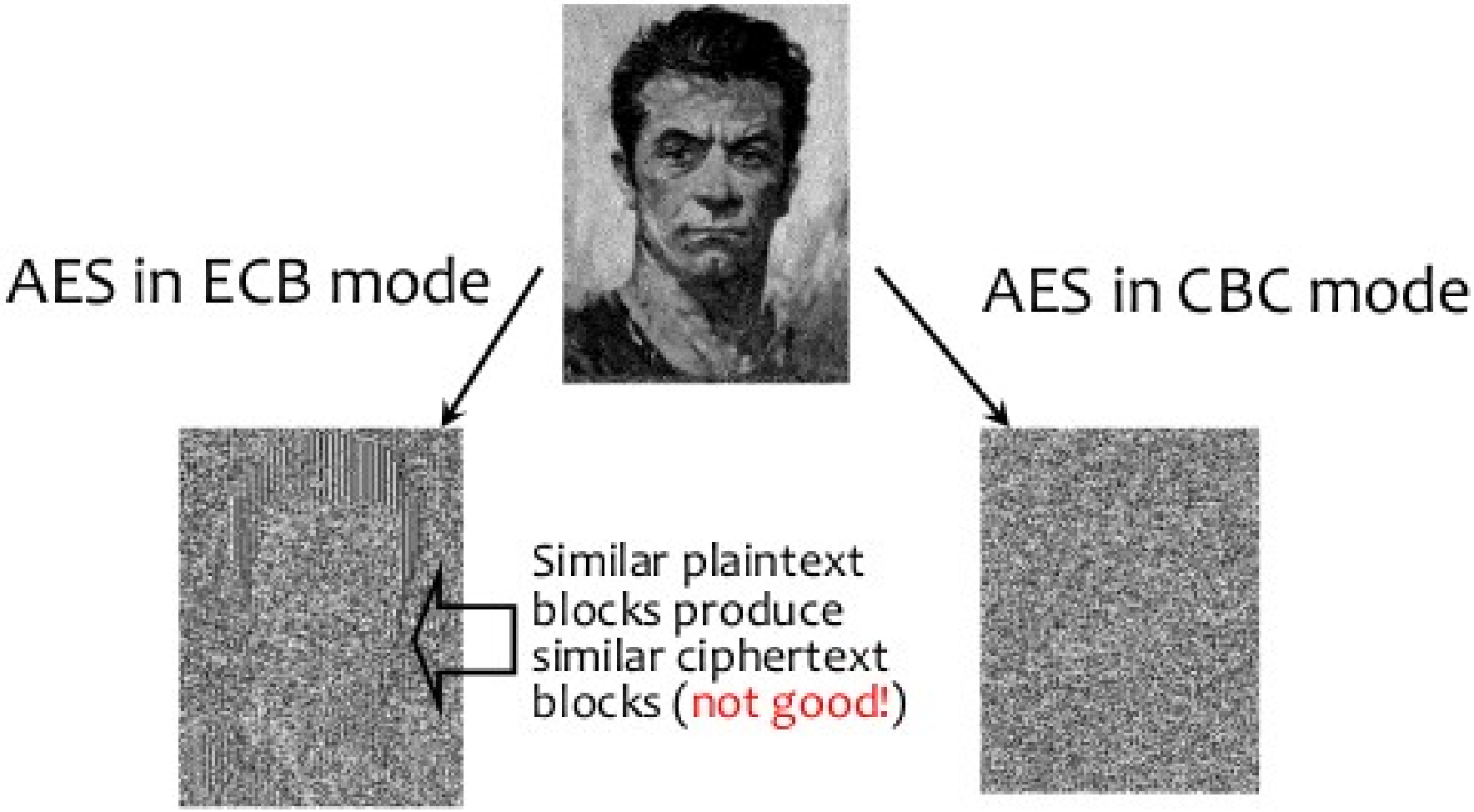
- **Identical blocks of plain-text encrypted differently**
- **Last cipher-block depends on entire plain-text**
- **Still does not guarantee integrity**



CBC Mode: Decryption

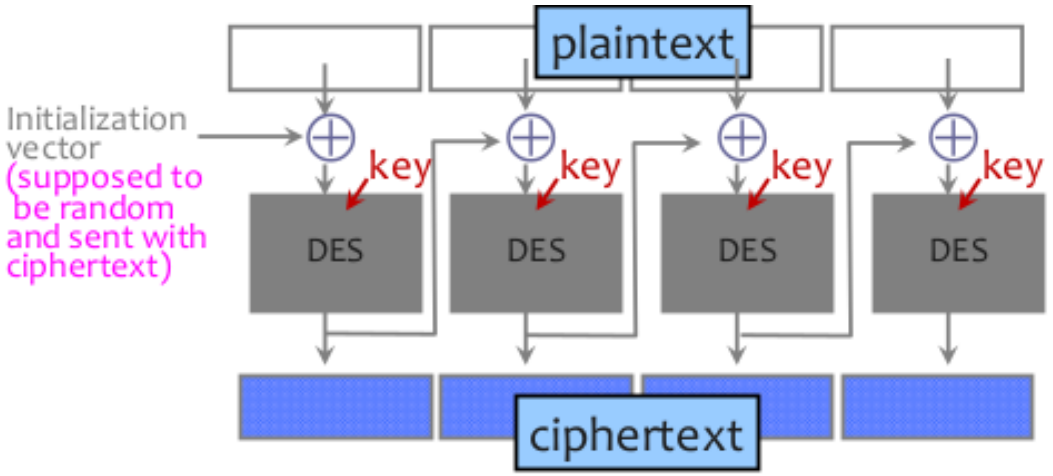


ECB vs. CBC


AES in ECB mode
AES in CBC mode
Similar plaintext blocks produce similar ciphertext blocks (not good!)

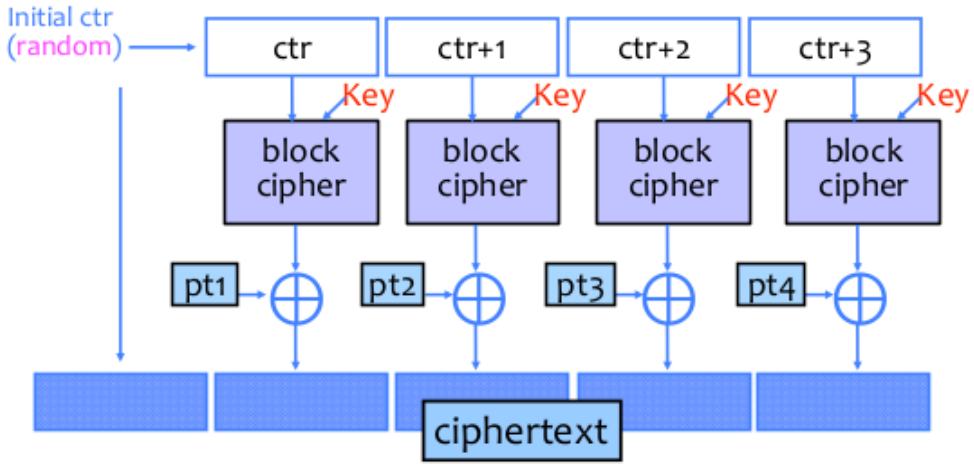
CBC and Electronic Voting

- **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)`

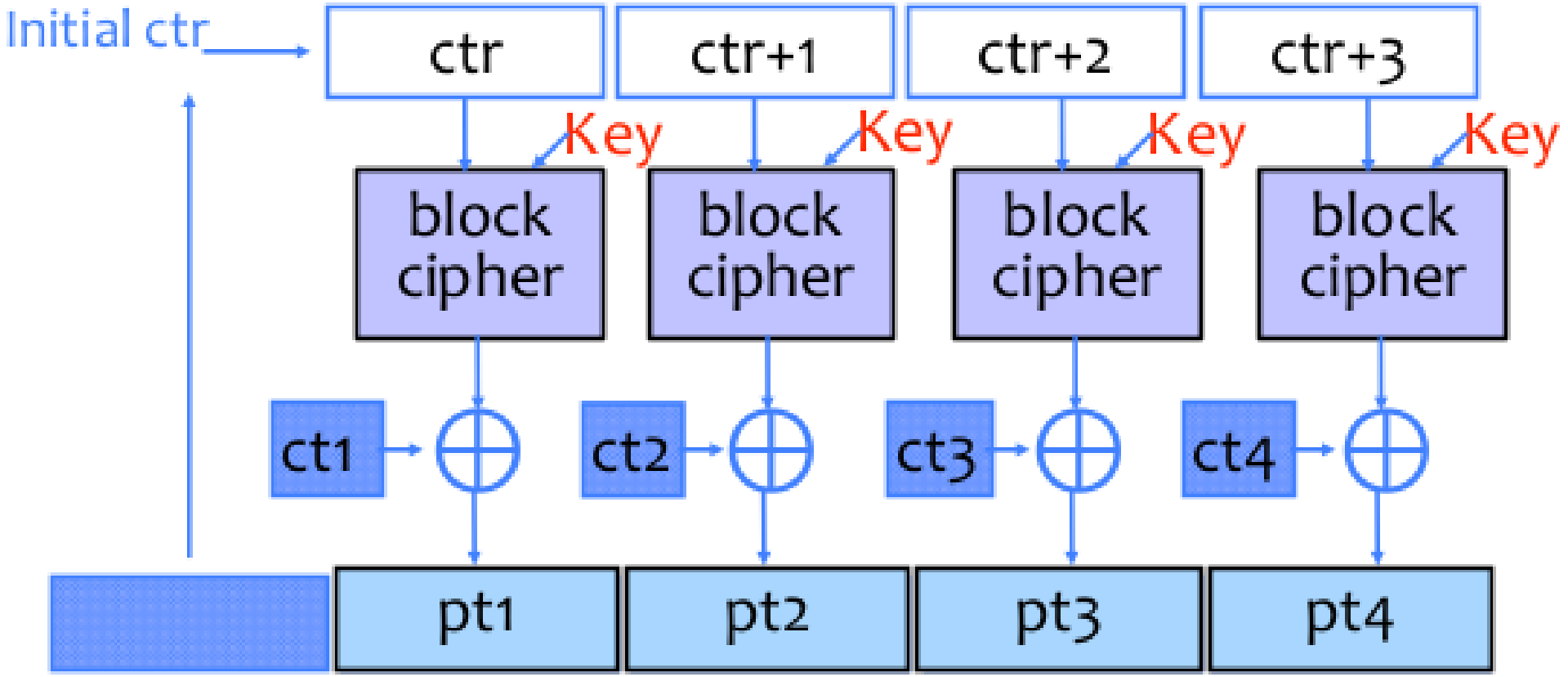


Counter Mode (CTR): Encryption

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



Counter Mode (CTR): Decryption



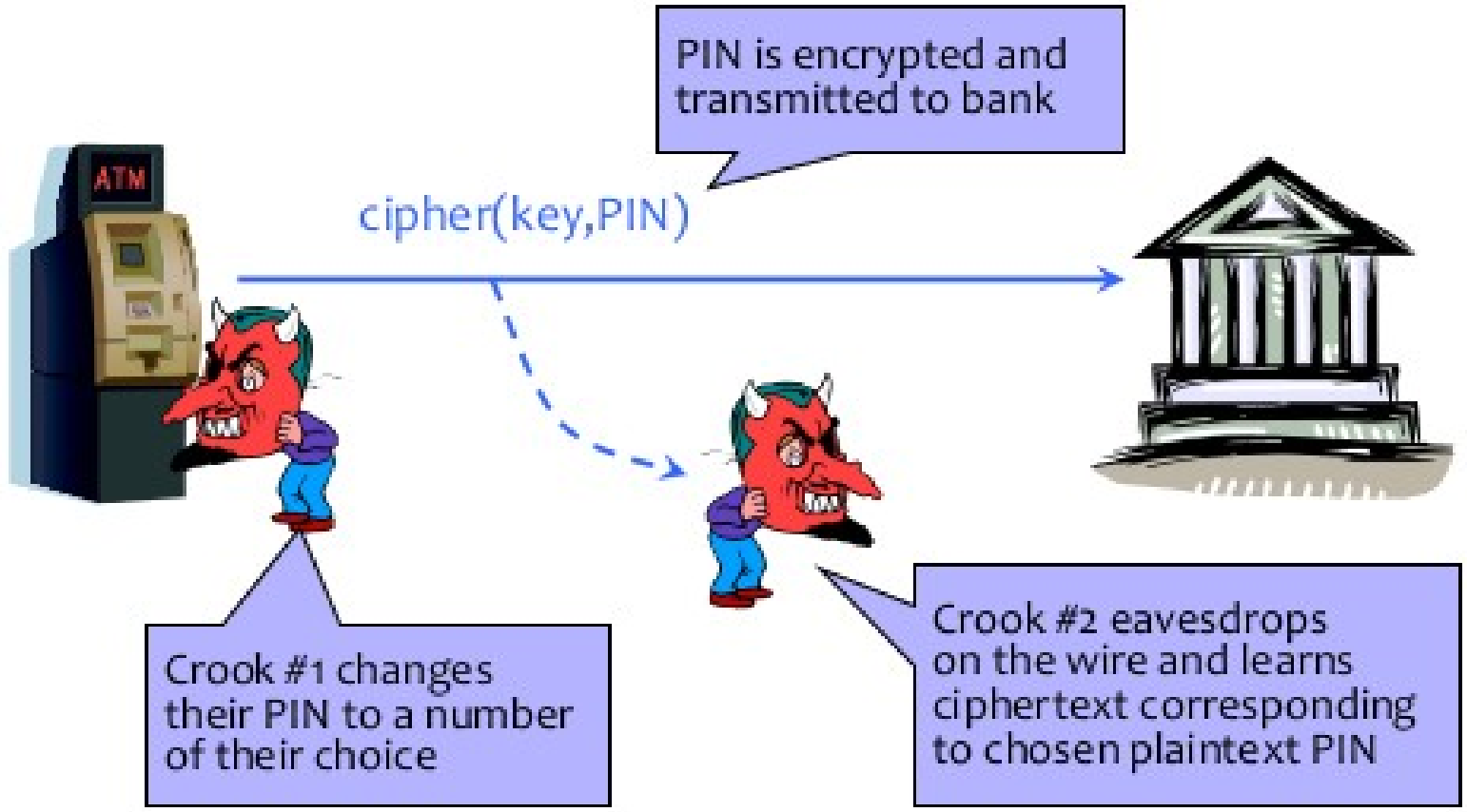
When is an Encryption Scheme "Secure"?

- **Hard to recover the key?**
  - What if attacker can learn plain-text without learning the key?
- **Hard to recover plain-text from cipher-text?**
  - What if attacker learns some bits or some function of bits?

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 their choice
- **CCA: Chosen-ciphertext attack (very strong)**
  - Can decrypt any ciphertext except the target

Chosen Plaintext Attack



... repeat for any PIN value

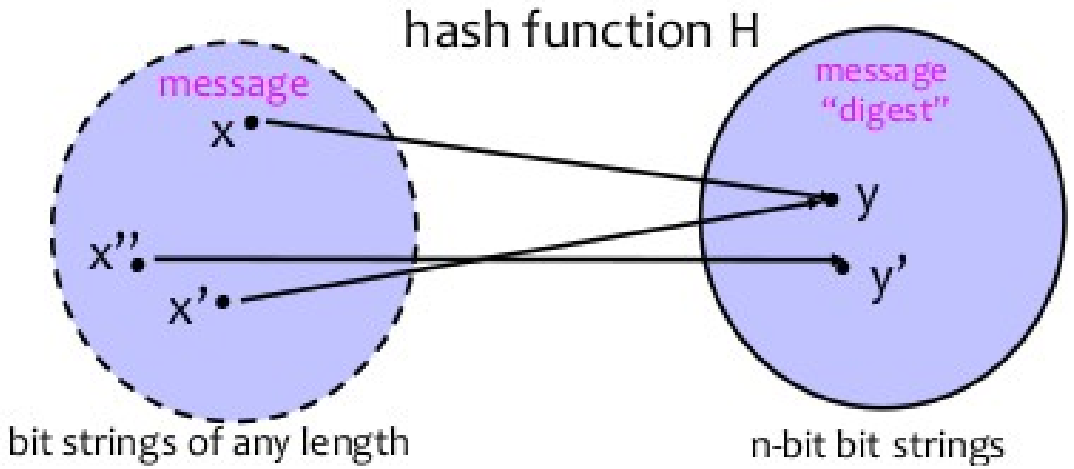
Very Informal Intuition

- **Security against chosen-plain-text attack (CPA)**
  - Cipher-text leaks no information about the plain-text
  - Even if the attacker correctly guesses the plain-text, they cannot verify their guess
  - Every cipher-text is unique, encrypting same message twice produces completely different cipher-texts
    - Implication: encryption must be randomized or stateful
- **Security against chosen-cipher-text attack (CCA)**
  - Integrity protection – it is not possible to change the plain-text by modifying the cipher-text

HASH FUNCTIONS

Basic Principle

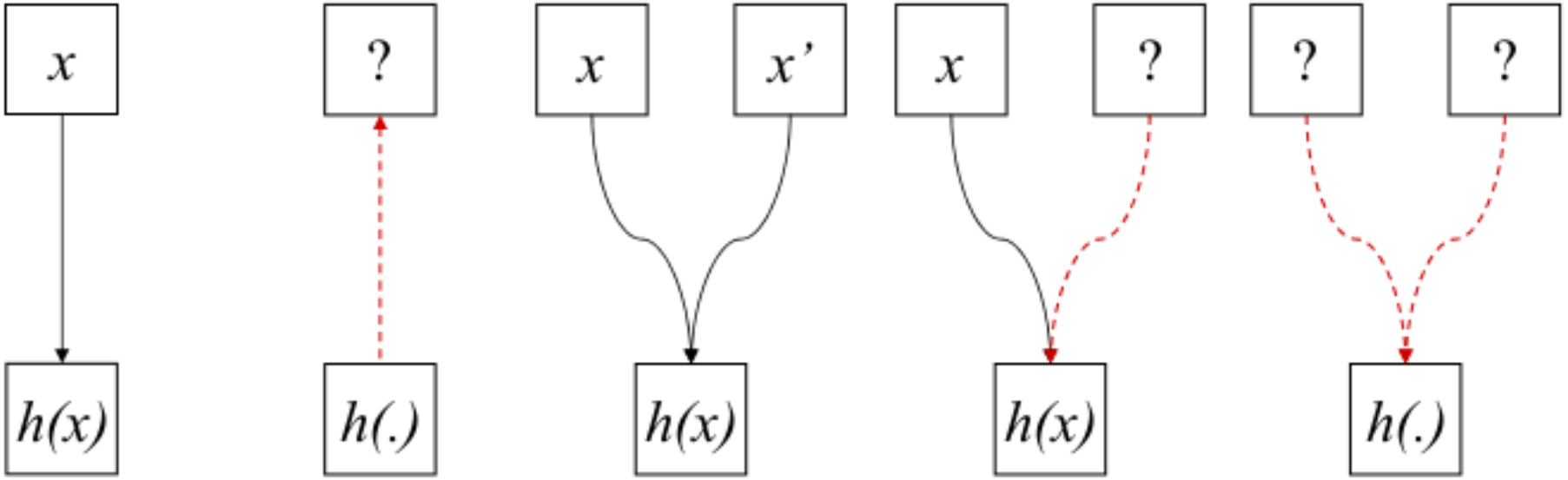
- **Hash function H is a lossy compression function**
  - Collision: h(x)=h(x’) for distinct inputs x, x’
- **H(x) should look “random”**
  - Every bit (almost) equally likely to be 0 or 1
- **Cryptographic hash function needs a few properties…**



Basic Properties

- **Compression**
  - h(x) maps x of arbitrary bit length to outputs of fixed length n.
- **Pre-image resistance (one-way)**
  - Given y, it is infeasible to find an input x, s.t. h(x) = y
- **2nd Pre-image resistance (weak collision resistance)**
  - Given x, and h(x), it is infeasible to find another x', x' ≠ x s.t. h(x) = h(x')
- **Strong collision resistance**
  - Infeasible to find any two inputs x, x', x ≠ x', s.t. h(x) = h(x')

Basic Properties



ease of computation

pre-image resistance

collision

2nd pre-image resistance

collision resistance

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.**

Birthday Paradox

- **Are there two people in the first few rows of this classroom that have the same birthday?**
  - Pick one person.
  - To find another person with same birthday would take O(365/2) people
  - Expect birthday “collision” with a room of only 23 people.
  - We expect a collision as sqrt(pi/2 * 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≠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 probability 0.5.

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

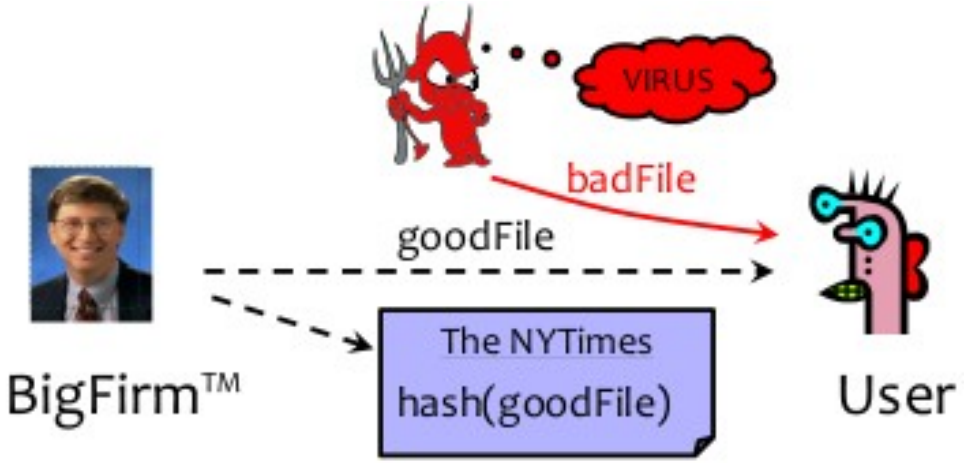
- **Anyone can compute a hash! There's no secret.**
- **Hashing is one-way. There is no "un-hashing"**
  - A cipher-text can be decrypted with a decryption key… hashes have no equivalent of "decryption"
- **Hash(x) looks "random" but can be compared for equality with Hash(x')**
  - Hash the same input twice -> same hash value
  - Encrypt the same input twice -> different cipher-texts
- **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**
- **Why is hashing better than encryption here?**
- **System does not store actual passwords!**
- **Cannot go from hash to password!**

Application: Software Integrity

- **Goal: Software manufacturer wants to ensure file is received by users without modification.**
  - 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**
  - Weak collision resistance
  - But software images are not really random... may need full collision resistance if considering malicious developers

Common Hash Functions

- **MD5 – Don’t use!**
  - 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) – Don’t use!**
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005; practical attack 2017!
- **SHA-256, SHA-512, SHA-224, SHA-384**
- **SHA-3: standard released by NIST in August 2015**

SHA-1 Broken in Practice (2017)

Google just cracked one of the building blocks of web encryption (but don't worry)

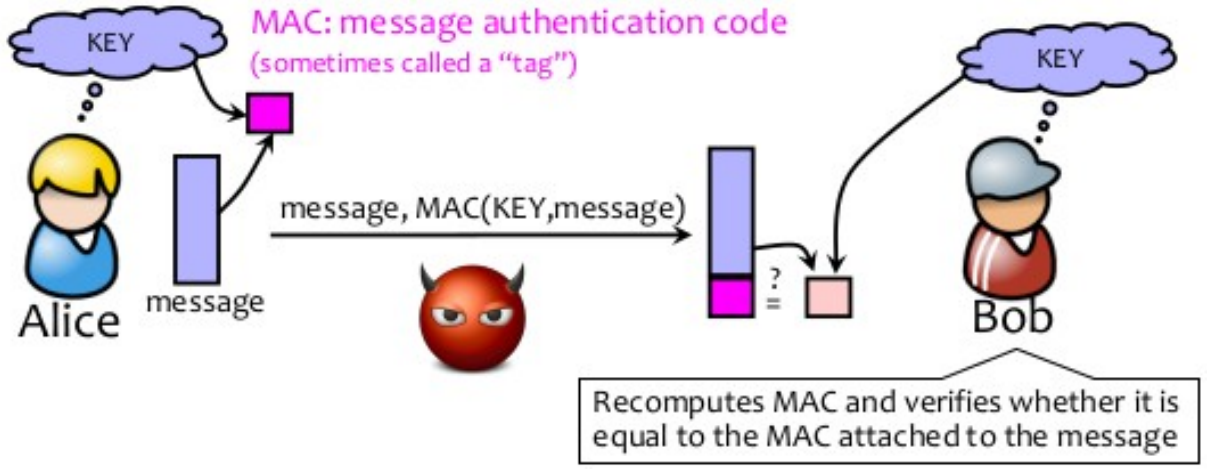
It's all over for SHA-1

by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST

https://shattered.io

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

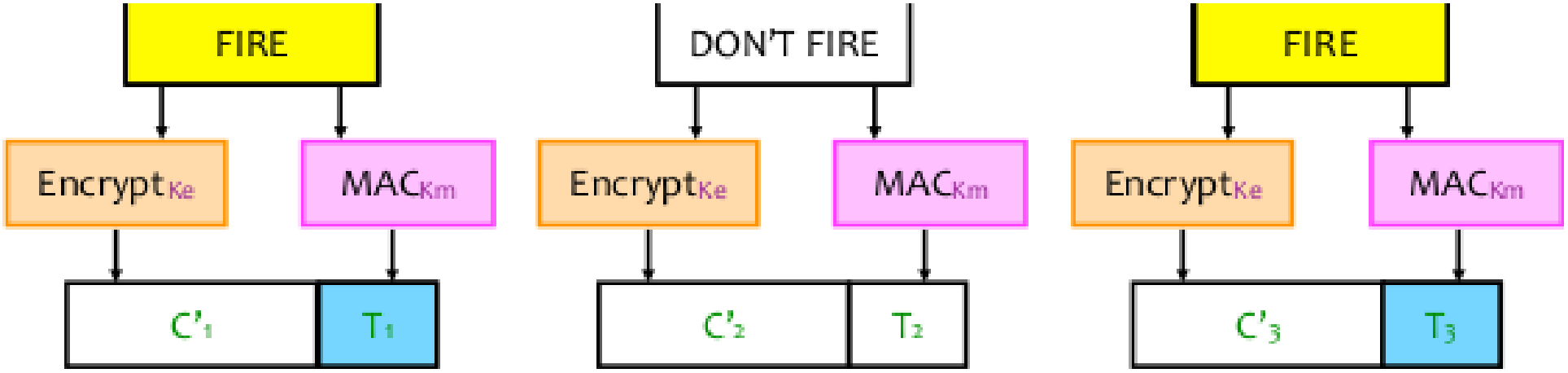
Authenticated Encryption

What if we want both privacy and integrity?

- **Natural approach: combine encryption scheme and a MAC.**
- **But be careful!**

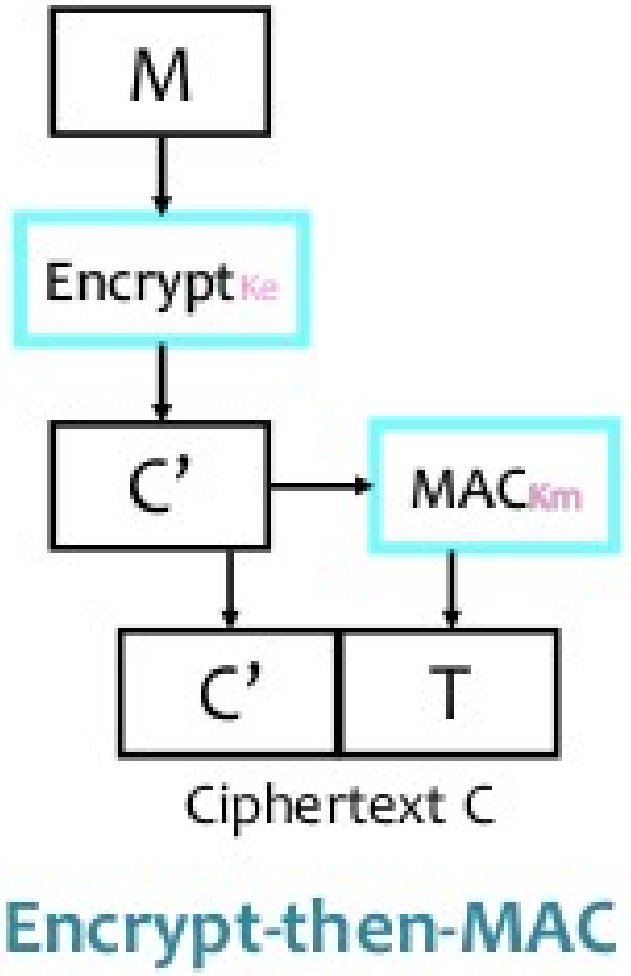
– Obvious approach: Encrypt-and-MAC

– Problem: MAC is deterministic! same plain-text -> same MAC



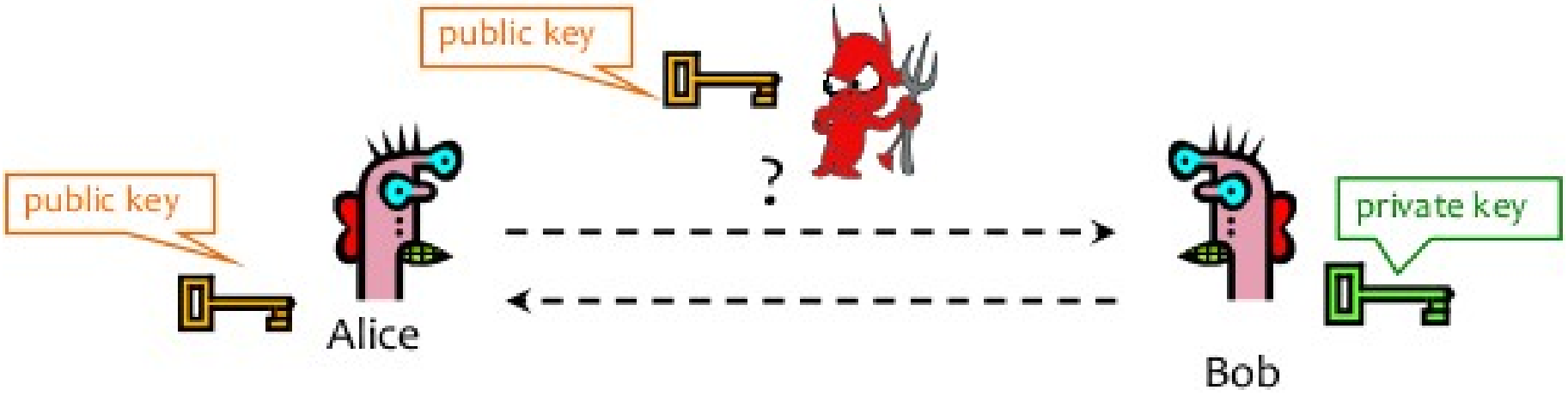
Authenticated Encryption

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



Public Key Crypto: Basic Problem

- **Everybody knows Bob's public key**
- **Only Bob knows the corresponding private key**
- **Alice wants to send a secret message to Bob**
- **Bob wants to authenticate himself**



Applications of Public Key Crypto

- **Session key establishment**
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)
- **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

Modular Arithmetic

- **Given g and prime p, compute:**
  - g^1 mod p, g^2 mod p, ... g^100 mod p
- **For p=11, g= 10**
  - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, ...
  - Produces cyclic group {10, 1} (order=2)
- **For p=11, g=7**
  - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, ...
  - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
- **g=7 is a “generator” of Z_11***
- **For p=11, g=3**
  - 3^1 mod 11 = 3, 3^2 mod 11 = 9, 3^3 mod 11 = 5, ...
  - Produces cyclic group {3,9,5,4,1} (order = 5)

Diffie-Hellman Protocol (1976)

- **Alice and Bob never met and share no secrets**
- **Public info: p and g**
  - p is a large prime, g is a generator of Z_p*
  - Z_p*={1, 2 … p-1}
  - for all a in Z_p* there exists i s.t. a=g^i mod p
- **Modular arithmetic: numbers "wrap around" after they reach p**
- **Compute k=(g^y mod p) x =g^xy mod p**
- **Compute k=(g^x mod p) y =g^xy mod p**

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 mod p and g y mod p, 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 mod p and g y mod p, 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 (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers**
  - Common recommendation:
  - Choose p=2q+1, where q is also a large prime
  - Choose g that generates a subgroup of order q in Z_p*
  - Eavesdropper can’t tell the difference between the established key and a random value
  - Often hash g xy mod p, and use the hash as the key
  - Can use the new key for symmetric cryptography
- **Diffie-Hellman protocol (by itself) does not provide authentication**
  - Party in the middle attack (often called “man in the middle attack”)

More on Diffie-Hellman Key Exchange

- **We have discussed discrete logs modulo integers.**
- **Significant advantages in using elliptic curve groups**
  - Some similar mathematical properties (i.e., are “groups”) but have better security and performance (size) properties

Requirements for Public Key Encryption

- **Key generation**
  - computationally easy to generate (public key PK, private key SK)
- **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 learn anything about M from C without SK
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- **Euler totient function φ(n) (n≥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
  - Easy to compute for primes: φ(p) = p-1
  - Note that φ(ab) = φ(a) φ(b)

RSA Cryptosystem

[Rivest, Shamir, Adleman 1977]

- **Key generation:**
  - Generate large primes p, q (need primality testing)
  - Compute n=pq and φ(n)=(p-1)(q-1)
  - Choose small e, relatively prime to φ(n)
    - Typically, e=3 or e=2^16 +1=65537
  - Compute unique d such that ed ≡ 1 mod φ(n)
    - Modular inverse: d ≡ e^-1 mod φ(n)
  - Public key = (e,n); private key = (d,n)
- **Encryption of m (m a number between 0 and n-1):**
  - c = m^e mod n
- **Decryption of c:**
  - c^d mod n = (m^e mod n)^d mod n = m

Why Decryption Works (FYI)

- **Decryption of c:**
  - c^d mod n = (m^e mod n)^d mod n = (m^e)^d mod n = m
- **Recall n=pq and φ(n)=(p-1)(q-1) and ed ≡ 1 mod φ(n)**
- **Chinese Remainder Theorem: To show m^ed mod n ≡ m mod n, sufficient to show:**
  - m^ed mod p ≡ m mod p
  - m^ed mod q ≡ m mod q
- **If m ≡ 0 mod p -> m^ed ≡ 0 mod p**
- **Else m^ed = m^(ed-1) m = m^[k(q-1)(p-1)] m =m^[h(p-1)] m for some k, and h=k(q-1).**
- **Why? Recall how d was chosen and the definition of mod.**
- **Fermat Little Theorem:**
  - m^[(p-1)h] m ≡ 1^h m mod p ≡ m mod p

Why is RSA Secure?

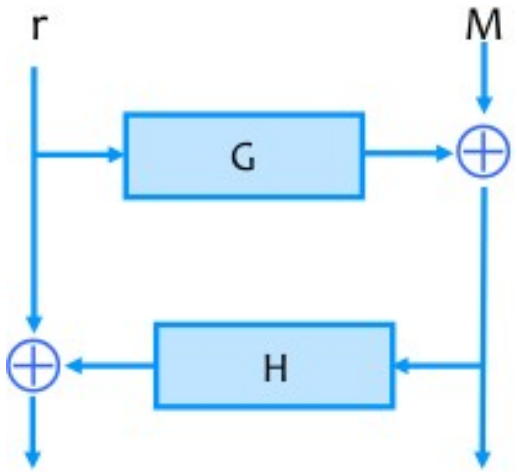
- **RSA problem: given c, n=pq, and e such that**
- **gcd(e, φ(n))=1, find m such that m^e = c mod n**
  - In other words, recover m from ciphertext c and public key (n,e) by taking e^{th} root of c modulo n
  - There is no *known* efficient algorithm for doing this
- **Factoring problem:**
  - given positive integer n, find primes p_1, …, p_k such that $n=p_1^{e1} p_2^{e2} \ldots p_k^{ek}$
- **If factoring is easy, then RSA problem is easy (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))**
  - It may be possible to break RSA without factoring n -- but if it is, we don’t know how

RSA Encryption Caveats

- **Encrypted message needs to be interpreted as an integer less than n**
- **Don’t use RSA directly for privacy – output is deterministic! Need to pre-process input somehow**
- **Plain RSA also does not provide integrity**
  - Can tamper with encrypted messages
- **Optimal asymmetric encryption padding (OAEP) is used:**
  - instead of encrypting M, encrypt M xor G(r) ; r xor H(M xor G(r))
  - r is random and fresh, G and H are hash functions

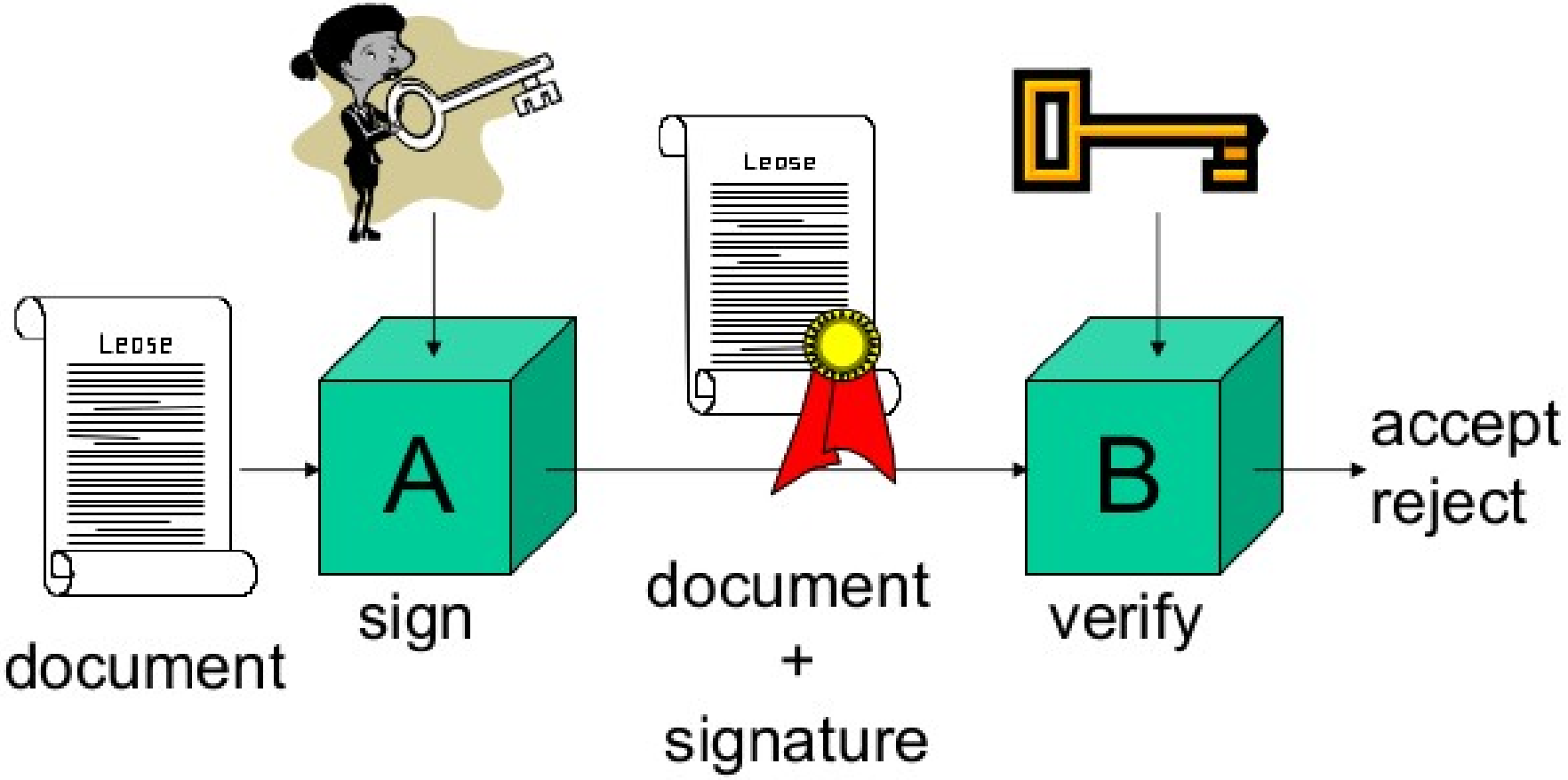
OAEP as a Figure

- **M xor G(r) ; r xor H(M xor G(r))**



DIGITAL SIGNATURES

Basic Idea


Lease
A
sign
document
Lease
document
+
signature
B
verify
accept
reject

Digital Signatures: Basic Idea

- **Everybody knows Bob's public key**
  - Only Bob knows the corresponding private key
- **Bob sends a "digitally signed" message**
  - To compute a signature, must know the private key
  - To verify a signature, only the public key is needed

RSA Signatures

- **Public key is (n,e), private key is (n,d)**
- **To sign message m: s = m^d mod n**
  - Signing & decryption are 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:**
  - verify that s e mod n = (m^d)^e mod n = m
  - "Just like encryption" (for RSA primitive)
  - Anyone who knows n and e (public key) can verify signatures produced with d (private key)
- **"Just like encryption" in quotes!**
  - In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

- **Digital Signature Standard (DSS)**
  - U.S. government standard (1991, most recent rev. 2013)
- **Public key: (p, q, g, y=g^x mod 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 mod p (public key)

Cryptography Summary

- **Goal: Privacy**
  - Symmetric keys:
    - One-time pad, Stream ciphers
    - Block ciphers (e.g., DES, AES) and modes: EBC, CBC, CTR
  - Public key crypto (e.g., Diffie-Hellman, RSA)
- **Goal: Integrity**
  - MACs, often using hash functions (e.g, MD5, SHA-256)
- **Goal: Privacy and Integrity**
  - Encrypt-then-MAC
- **Goal: Authenticity (and Integrity)**
  - Digital signatures (e.g., RSA, DSS)