CSE P564: Computer Security and Privacy

Cryptography Part 2

Autumn 2024

David Kohlbrenner
dkohlbre@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

Paper discussion

Four Attacks and a Proof for Telegram

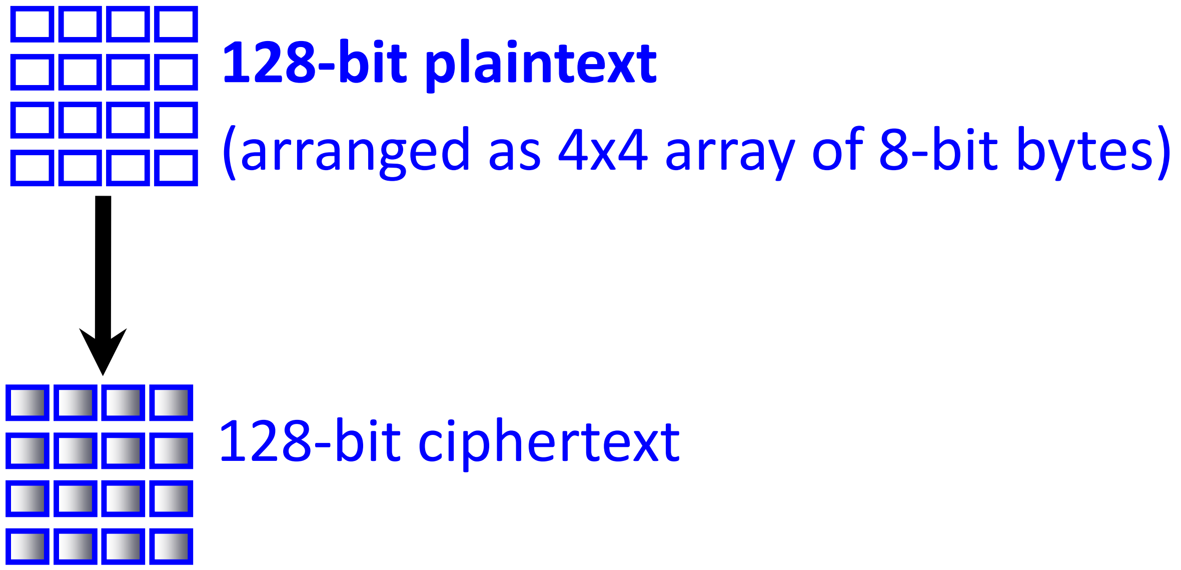
Pick one/more of the following to discuss

- The authors give an unusually specific motivation for the paper. How did it affect your understanding of the work?
- What was one of the attacks? What did it break about Telegram?
- Did the paper adjust your perception of Telegram?
- The authors describe several core security principles they wanted the protocol to uphold. Identify one and discuss what it would mean to the end user.

Cryptography!

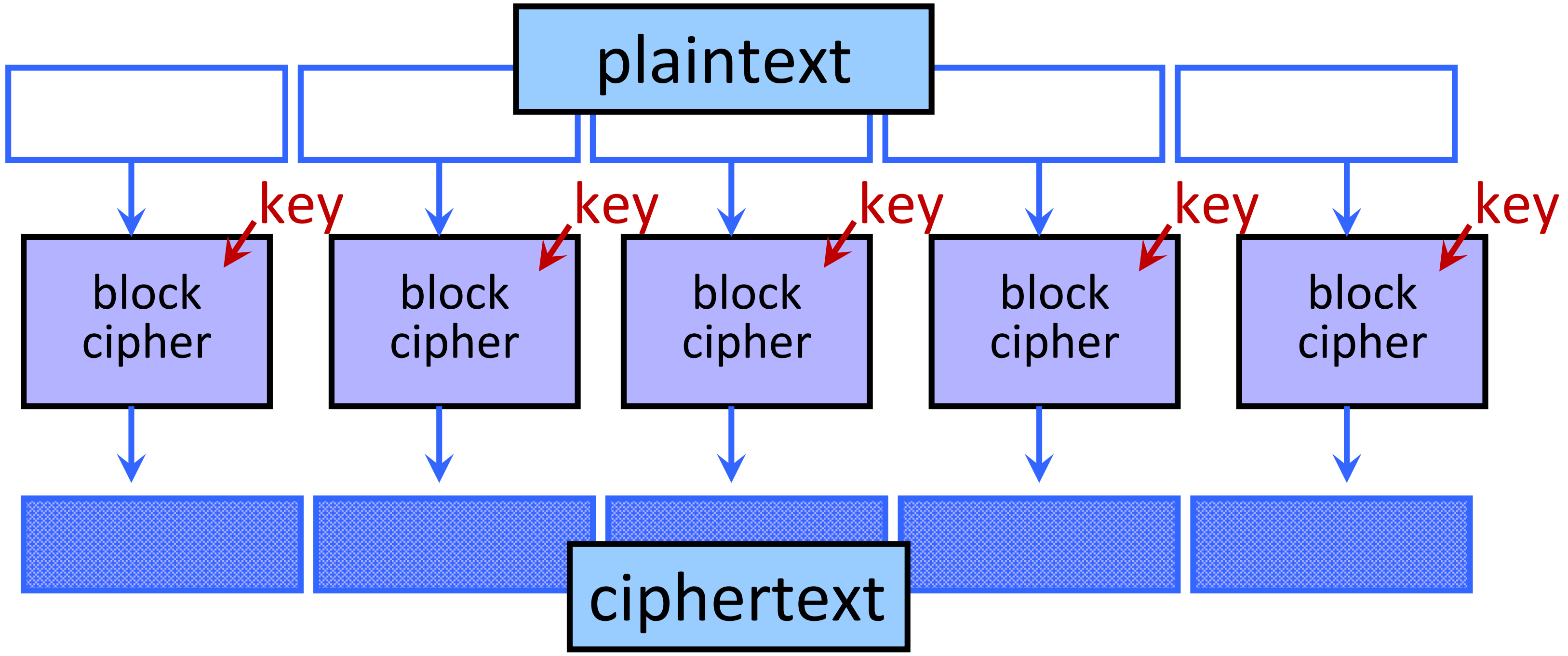
Encrypting a Large Message

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

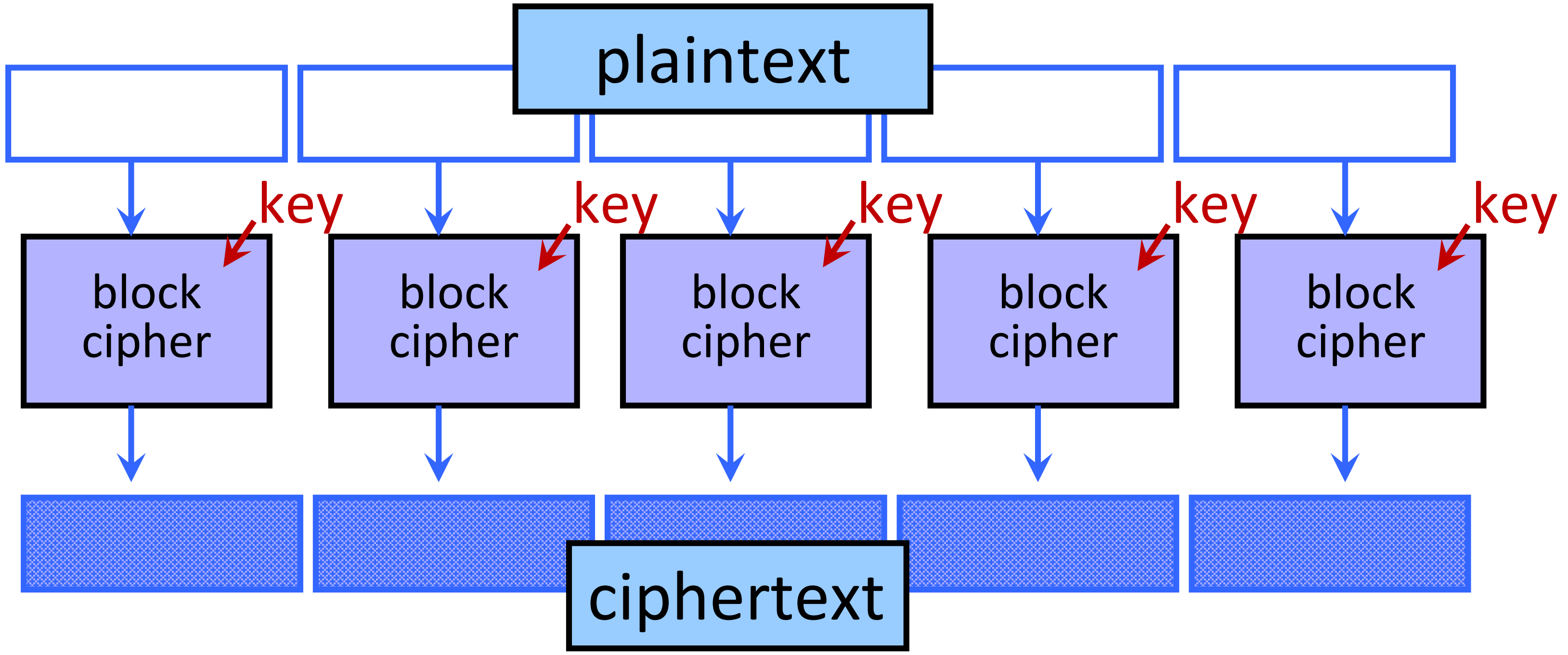


- What should we do?

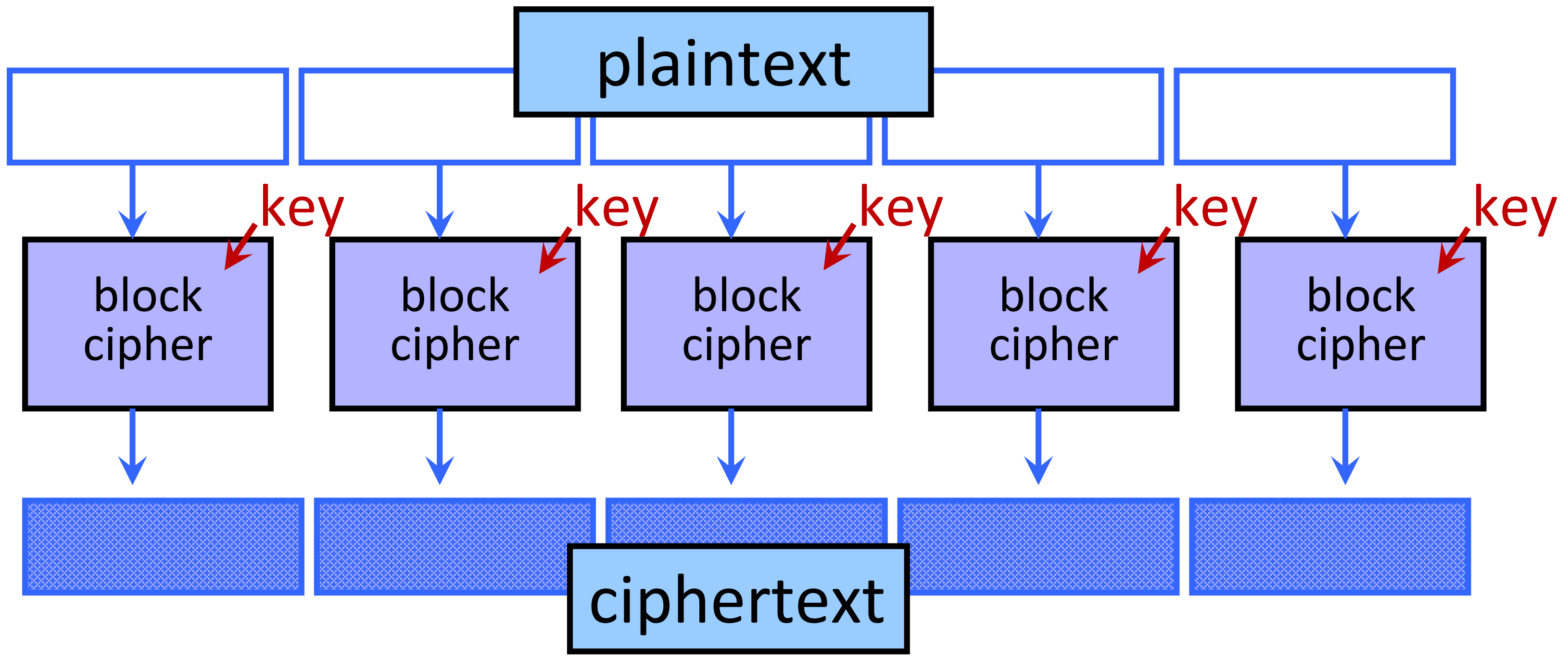
Electronic Code Book (ECB) Mode



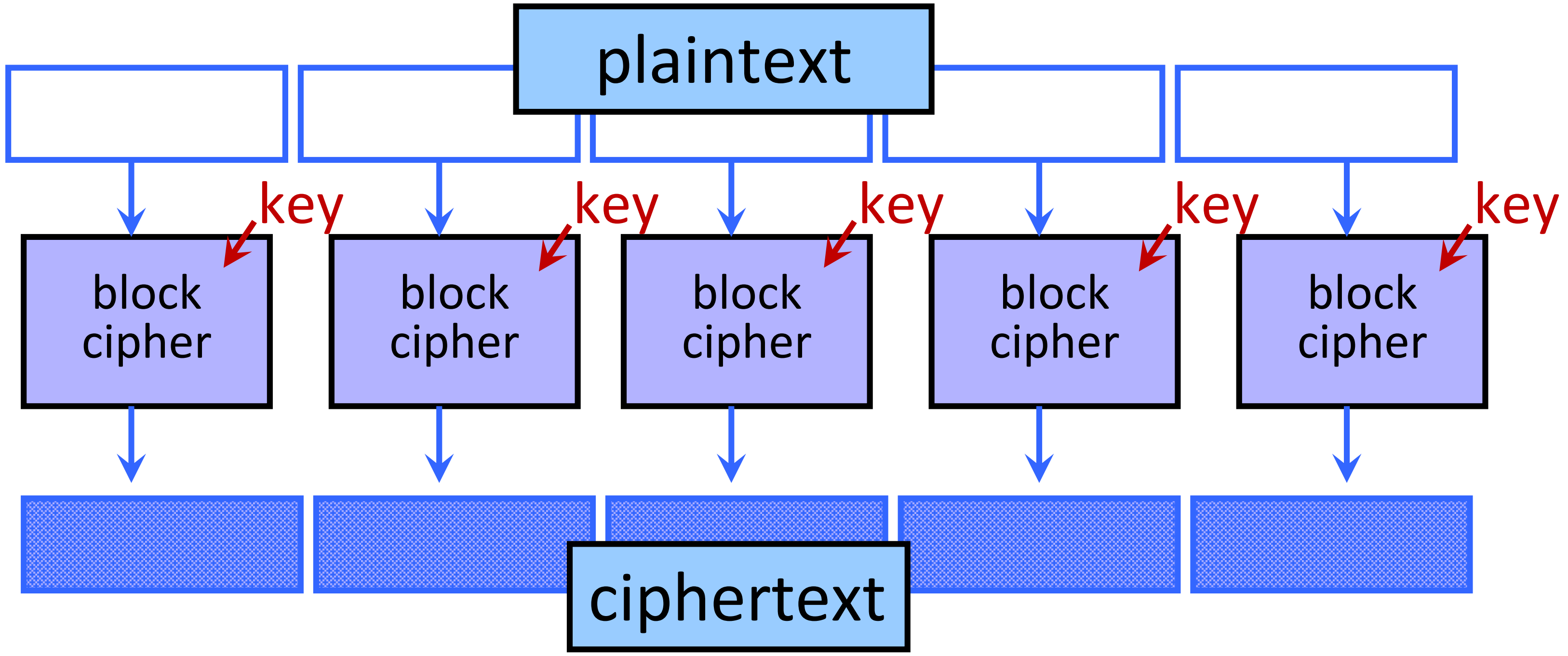
Electronic Code Book (ECB) Mode



Canvas: What properties of ECB aren't great?

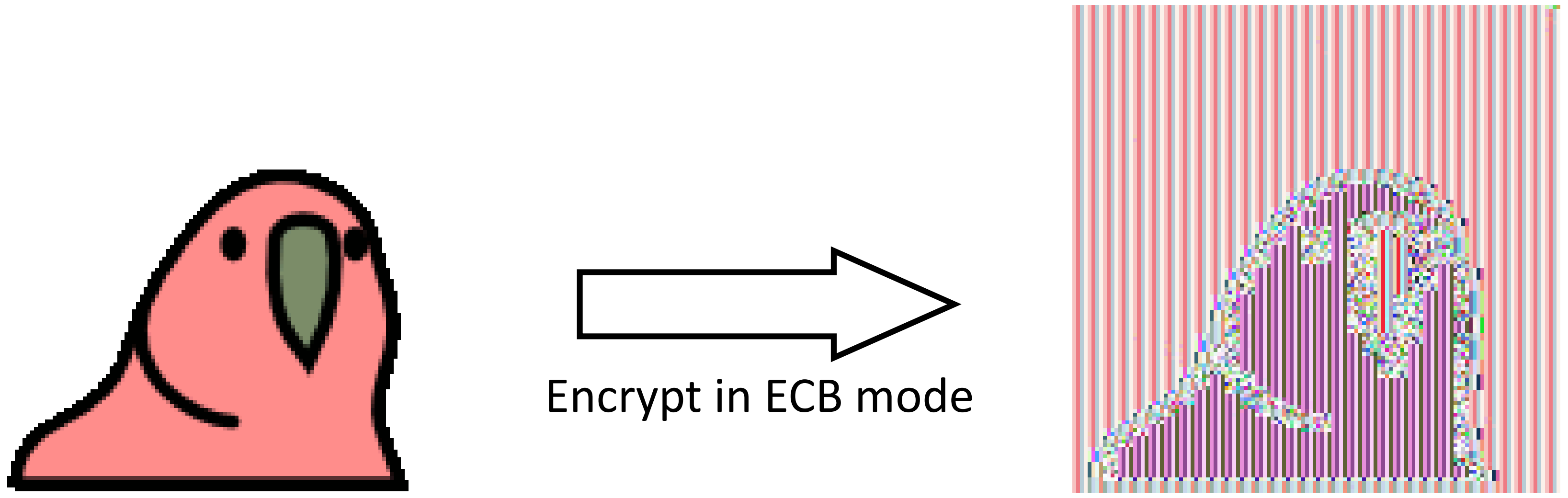


Electronic Code Book (ECB) Mode

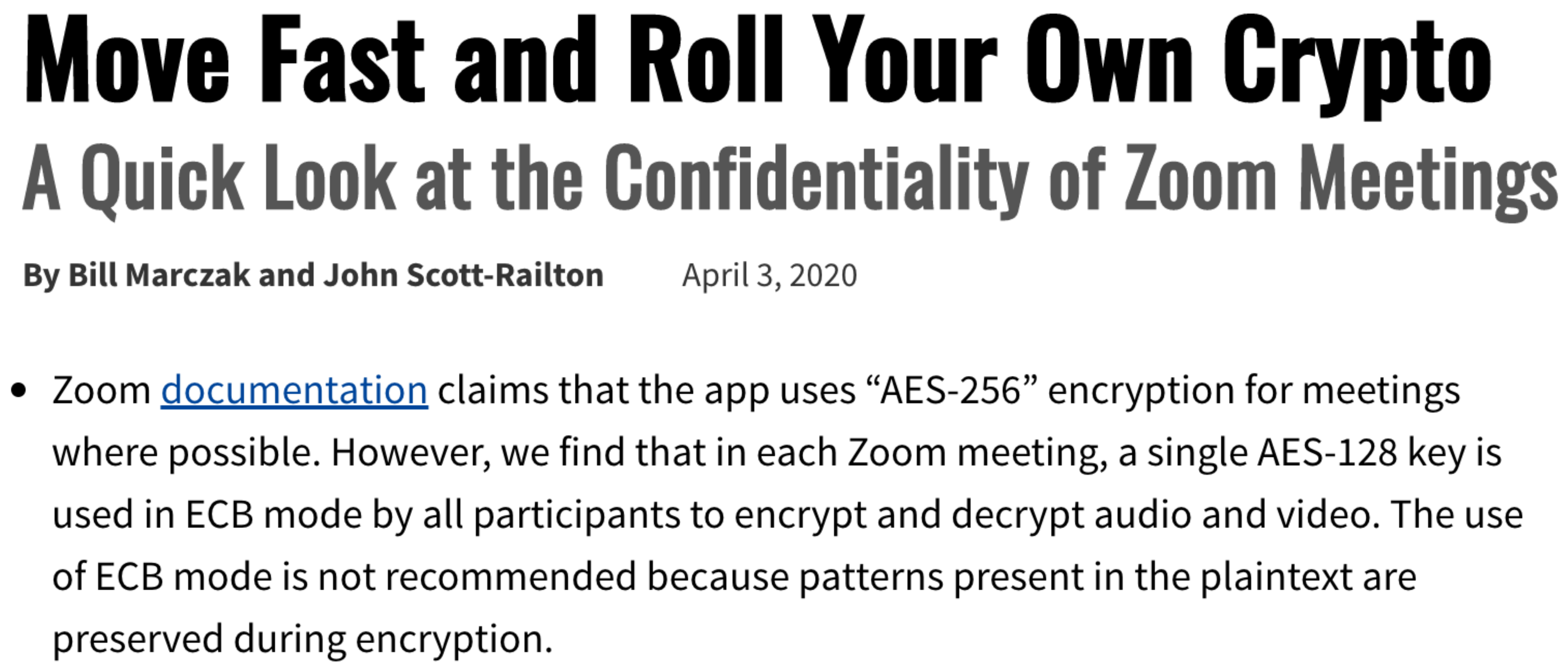


- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Information Leakage in ECB Mode



Oops



Move Fast and Roll Your Own Crypto

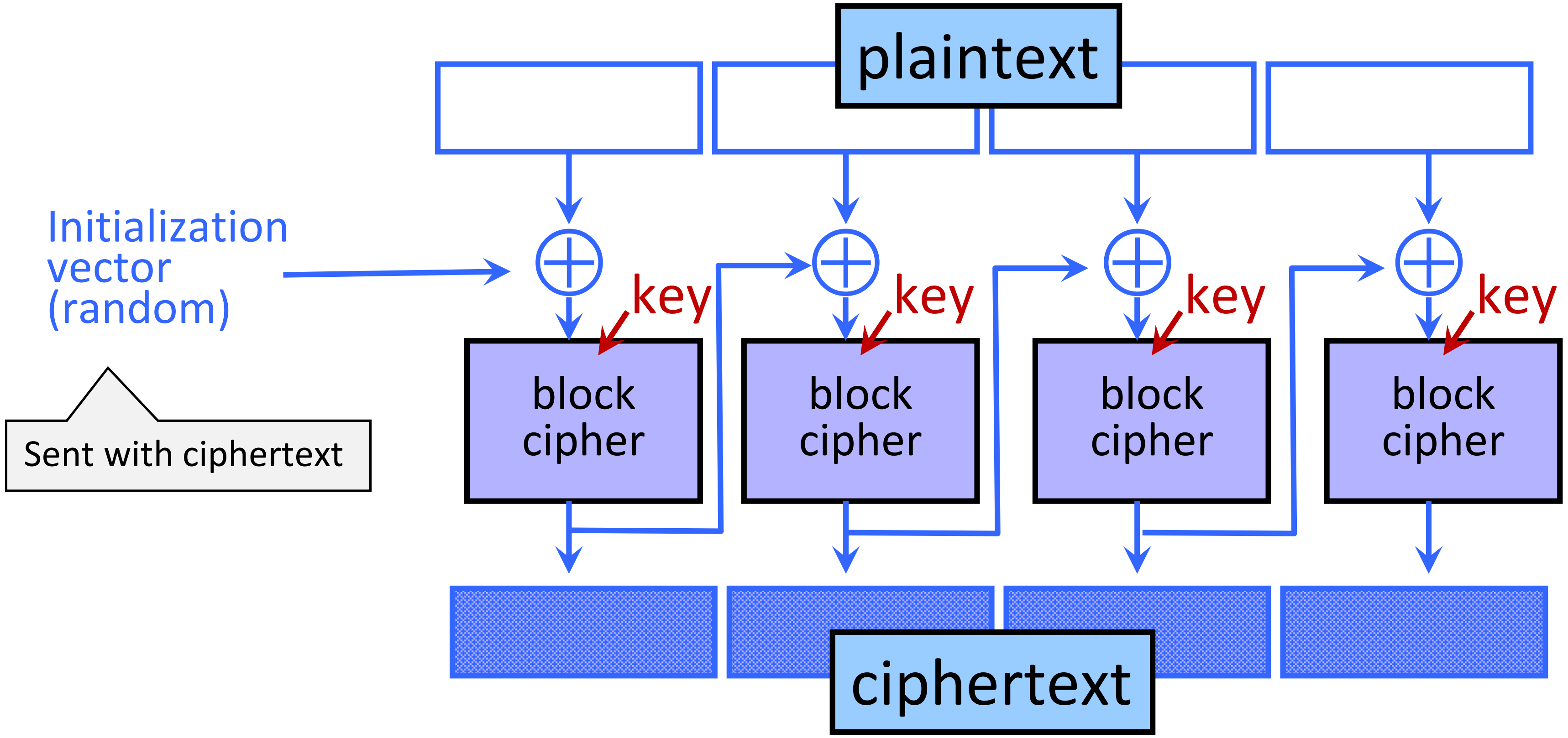
A Quick Look at the Confidentiality of Zoom Meetings

By Bill Marczak and John Scott-Railton April 3, 2020

- Zoom documentation claims that the app uses "AES-256" encryption for meetings where possible. However, we find that in each Zoom meeting, a single AES-128 key is used in ECB mode by all participants to encrypt and decrypt audio and video. The use of ECB mode is not recommended because patterns present in the plaintext are preserved during encryption.

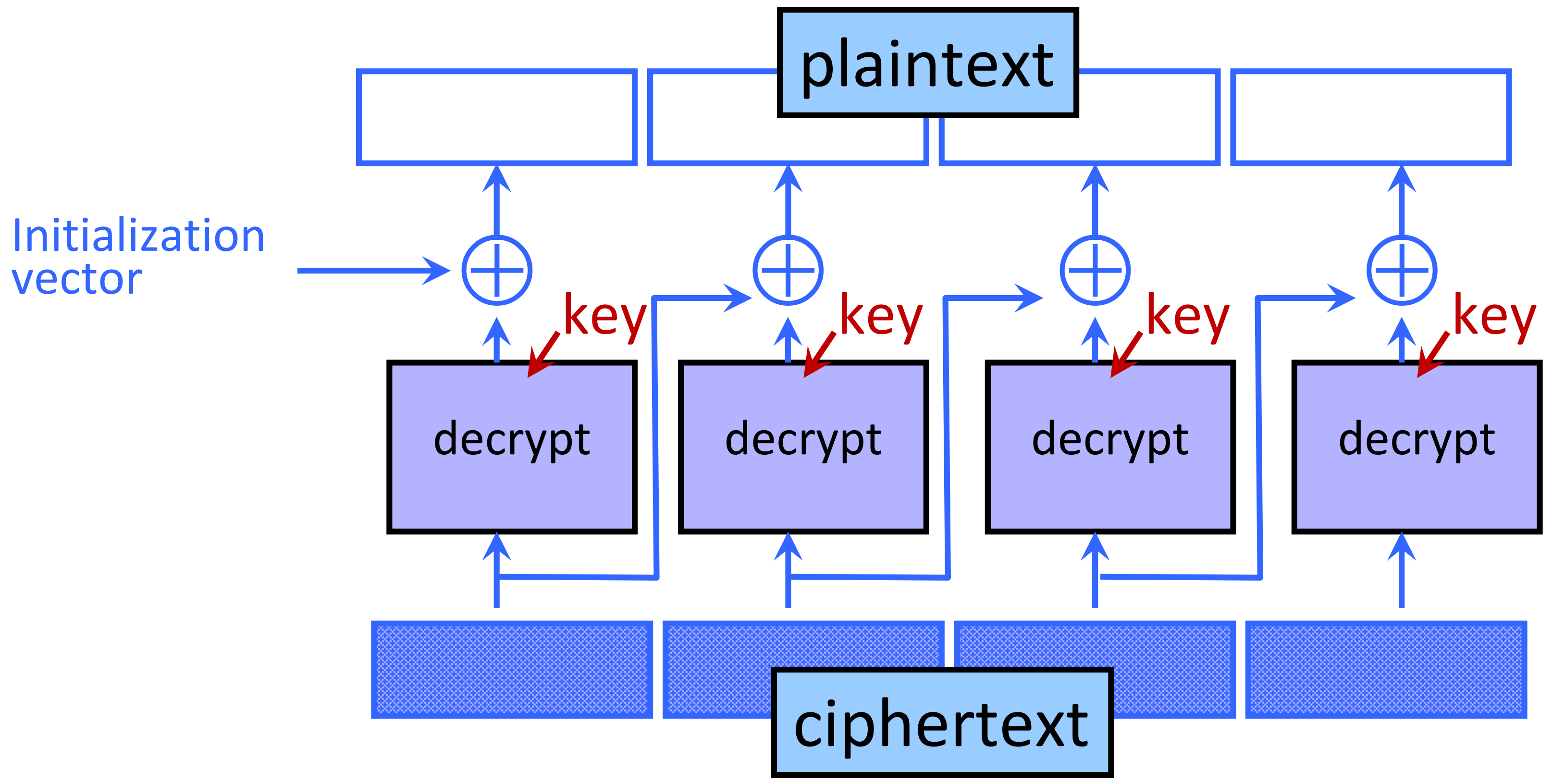
https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/

Cipher Block Chaining (CBC) Mode: Encryption

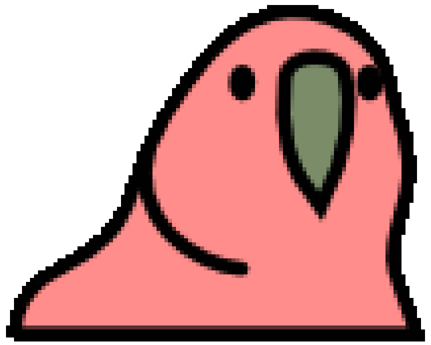


- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

CBC Mode: Decryption

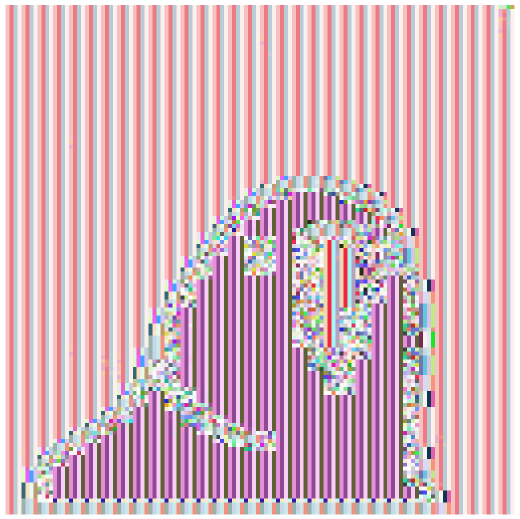


ECB vs. CBC

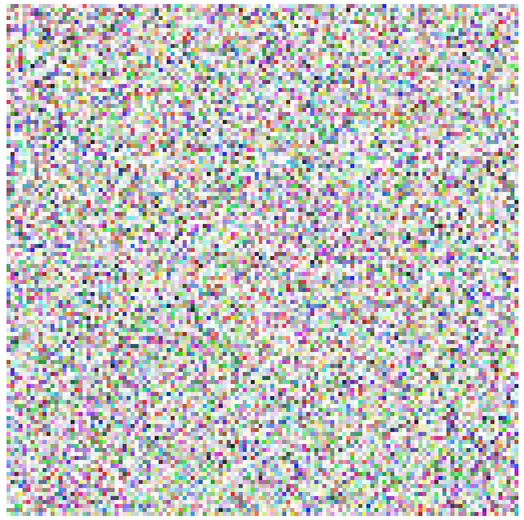


AES in ECB mode

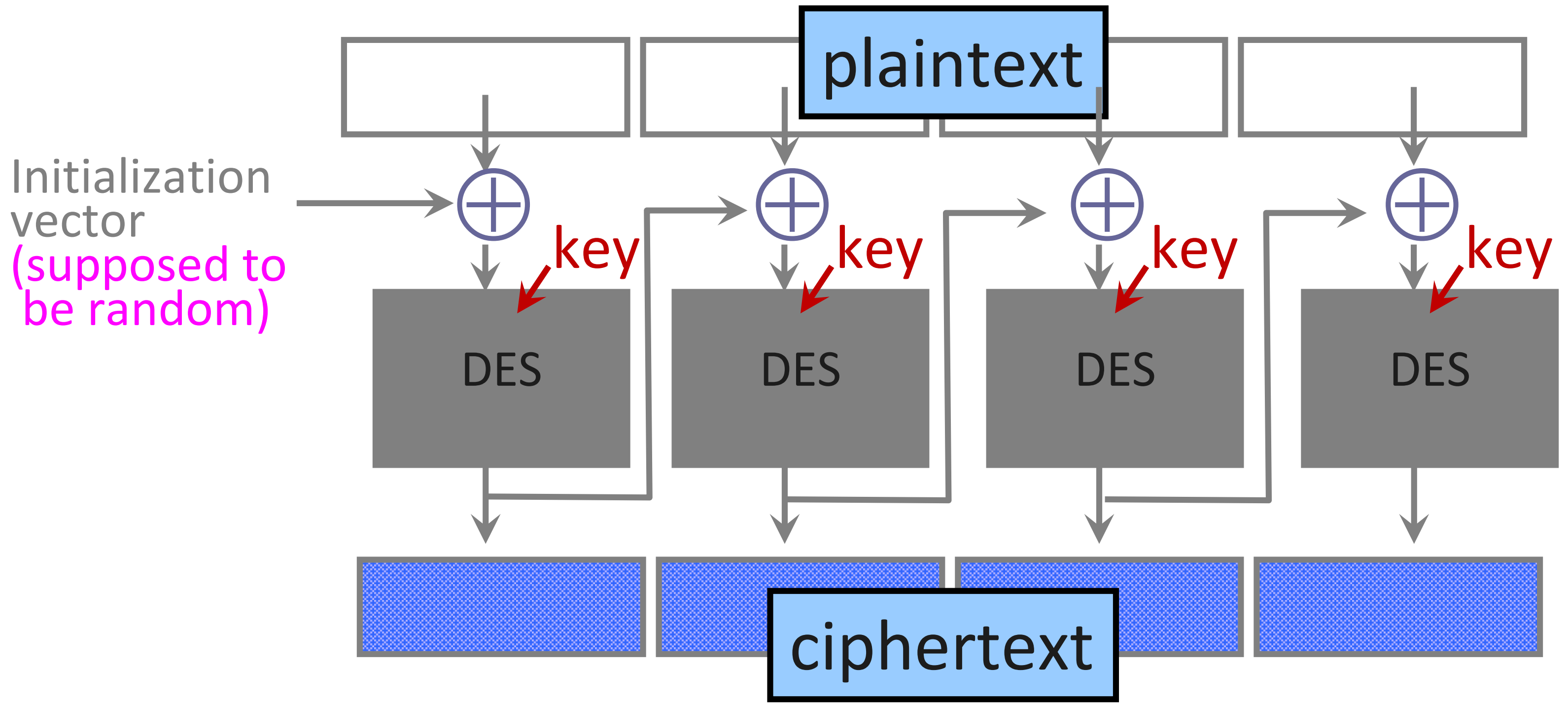
AES in CBC mode



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



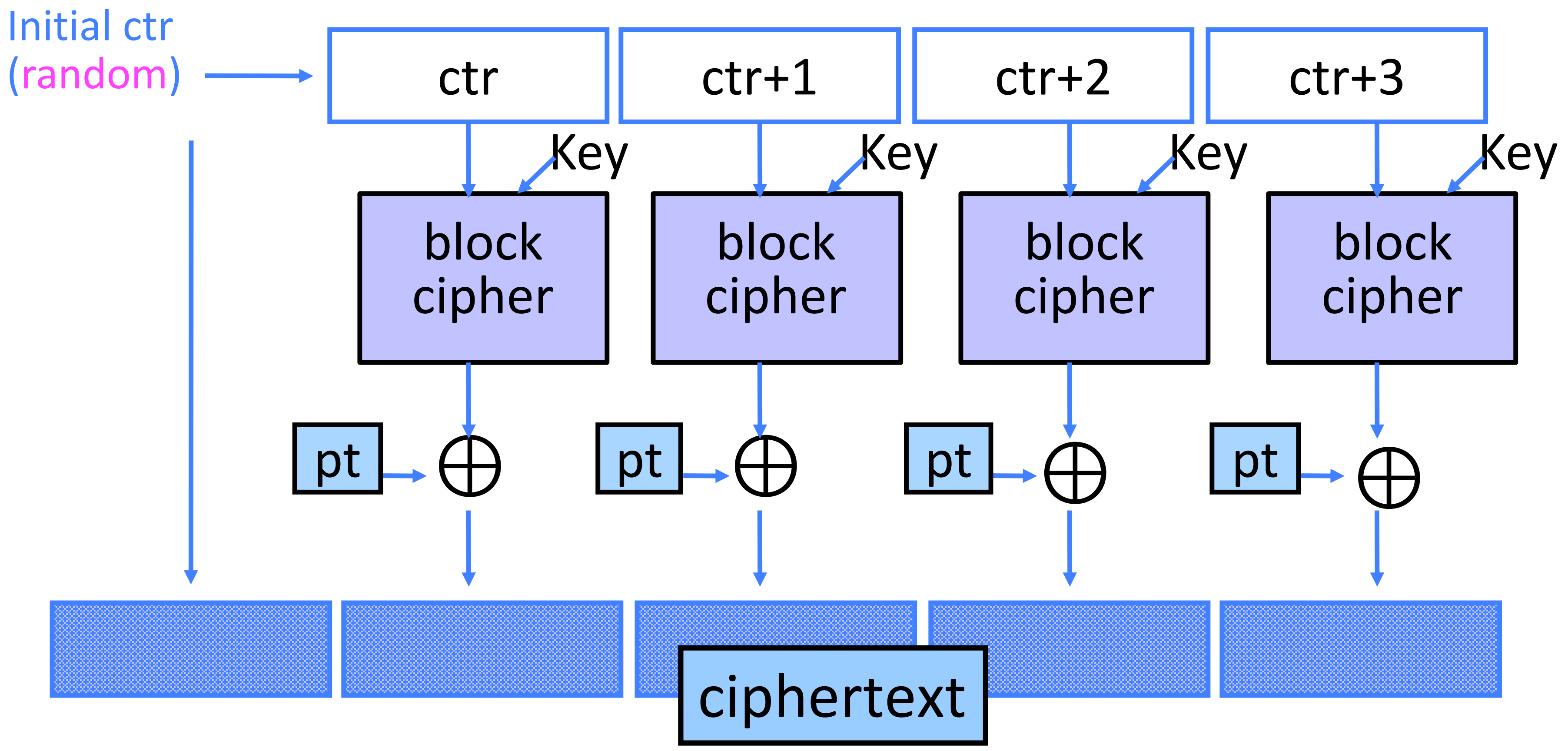
Initialization Vector Dangers



Found in the source code for Diebold voting machines:

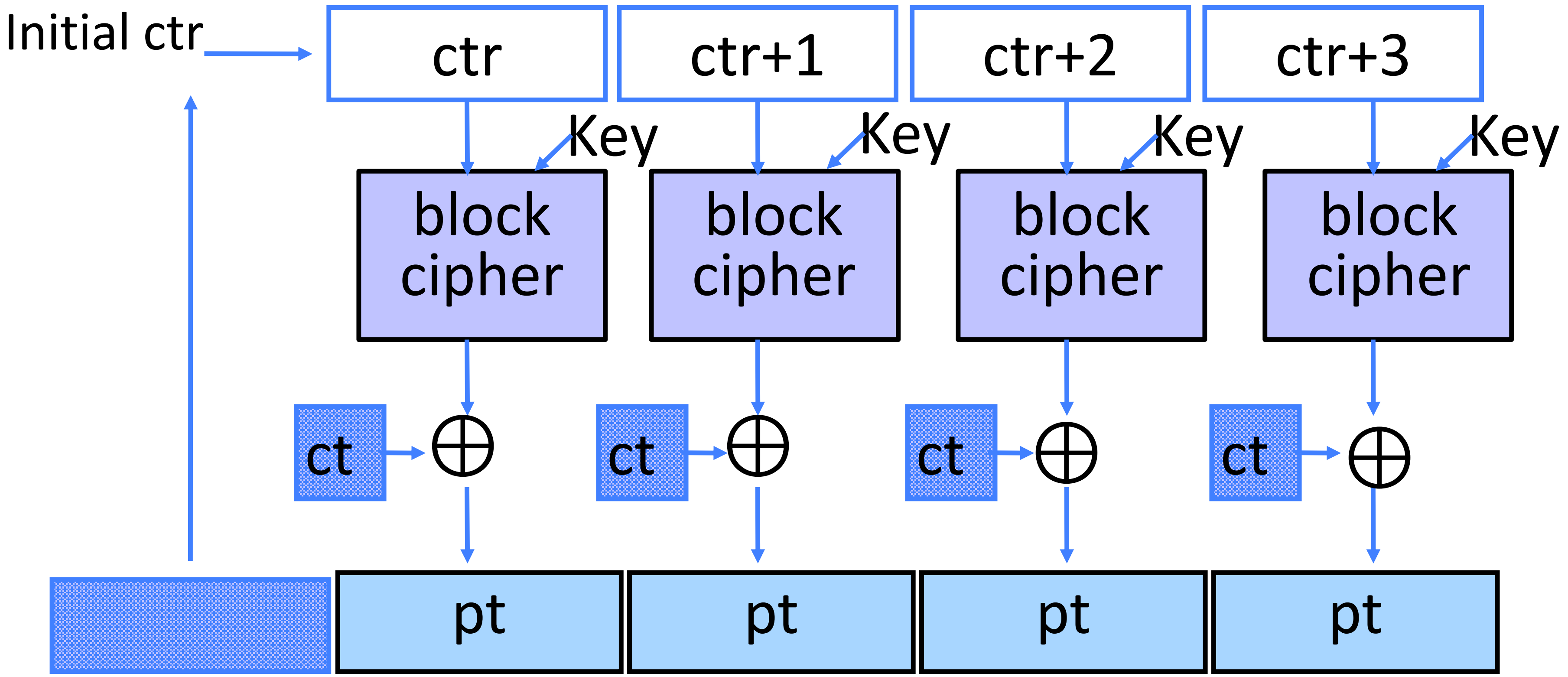
```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
        totalSize, DESKEY, NULL, DES_ENCRYPT)
```

Counter Mode (CTR): Encryption

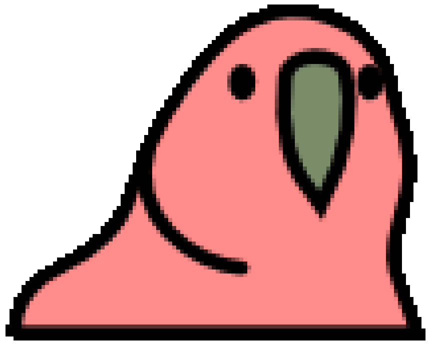


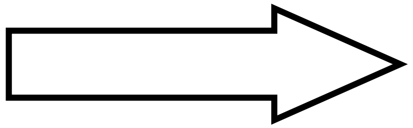
- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity; Fragile if ctr repeats

Counter Mode (CTR): Decryption



Information Leakage in CTR Mode (poorly)





Encrypt in CTR mode:
But with the same counter for each frame!

Ok, so what mode do I use?

- Don't choose a mode, use established libraries ☺

- Modes that might be good:
  - GCM-SIV - Galois/Counter Mode + more
  - CCM – CTR + CBC-MAC (we'll get to that next time)
  - CTR (sometimes its fine!)

- AES-128 is standard (AES-256 is… fine)
  - Be concerned if something says "AES 1024"…

https://research.kudelskisecurity.com/2022/05/11/practical-bruteforce-of-aes-1024-military-grade-encryption/

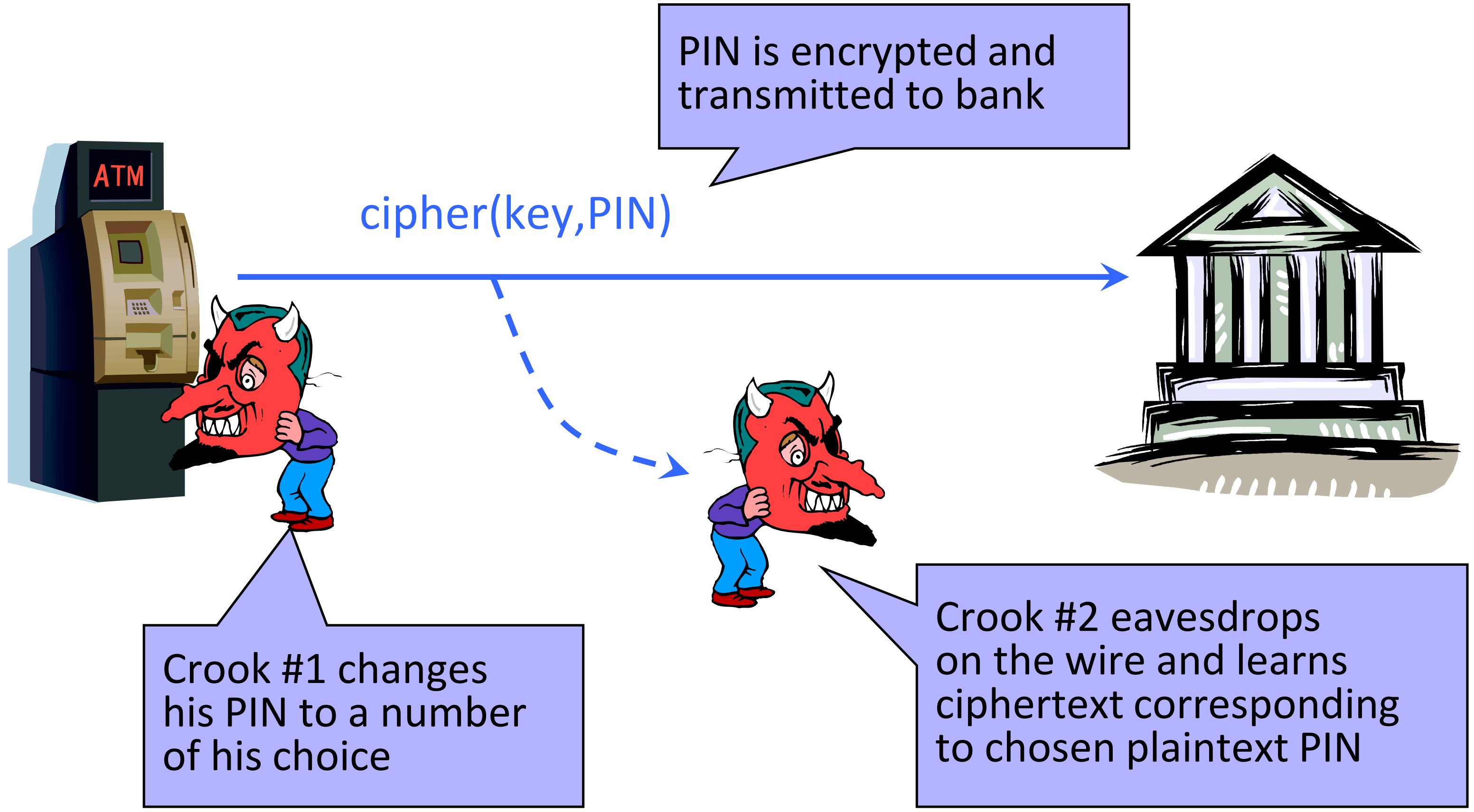
When is an Encryption Scheme “Secure”?

- 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?

How Can a Cipher Be Attacked?

- Attackers knows ciphertext and encryption algorithm
  - 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 choice
- CCA: Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext <u>except</u> the target

Chosen Plaintext Attack


PIN is encrypted and transmitted to bank
ATM
cipher(key,PIN)
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

The shape of the formal approach

- INDistinguishability under Chosen Plaintext Attack
  - IND-CPA
- Formalized *cryptographic game*

- Adversary submits pairs of *plaintexts* (M_a, M_b)
  - Gets back ONE of the *ciphertexts* (C_x)

- Adversary must guess which ciphertext this is (C_a or C_b)
  - If they can do better than 50/50, they win

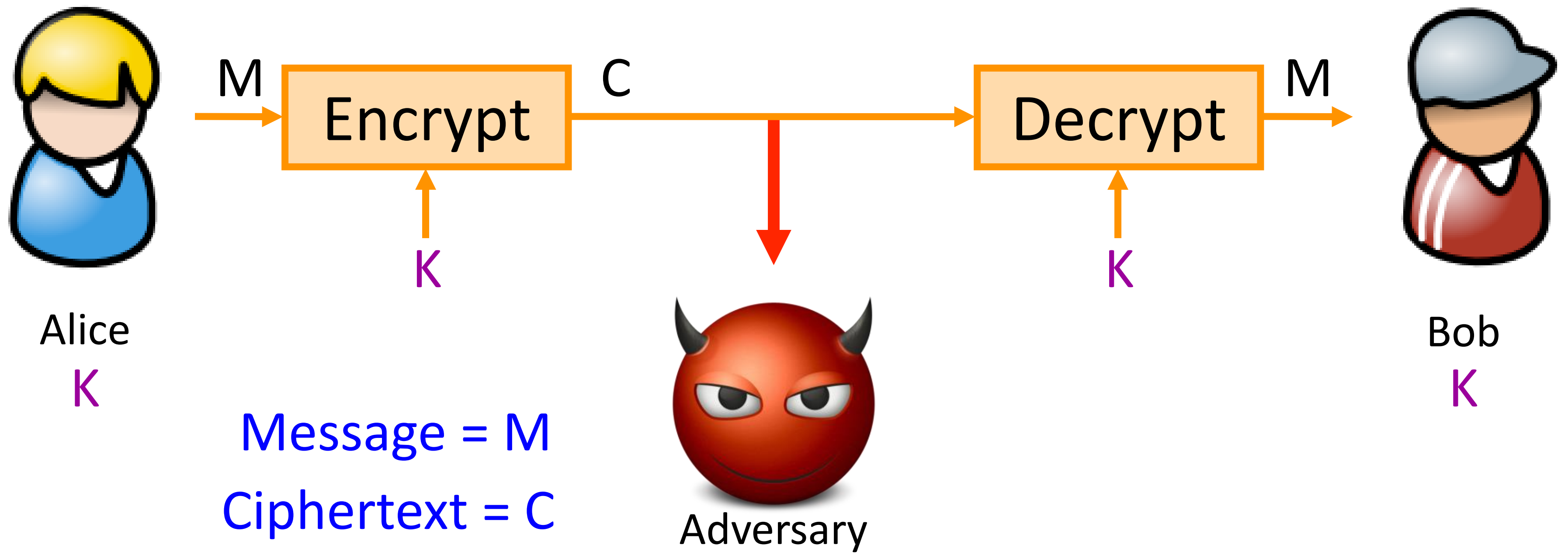
Very Informal Intuition

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
    - Implication: encryption must be randomized or stateful
- Security against chosen-ciphertext attack (CCA)
  - Integrity protection – it is not possible to change the plaintext by modifying the ciphertext

So Far: Achieving Privacy

Encryption schemes: A tool for protecting privacy.

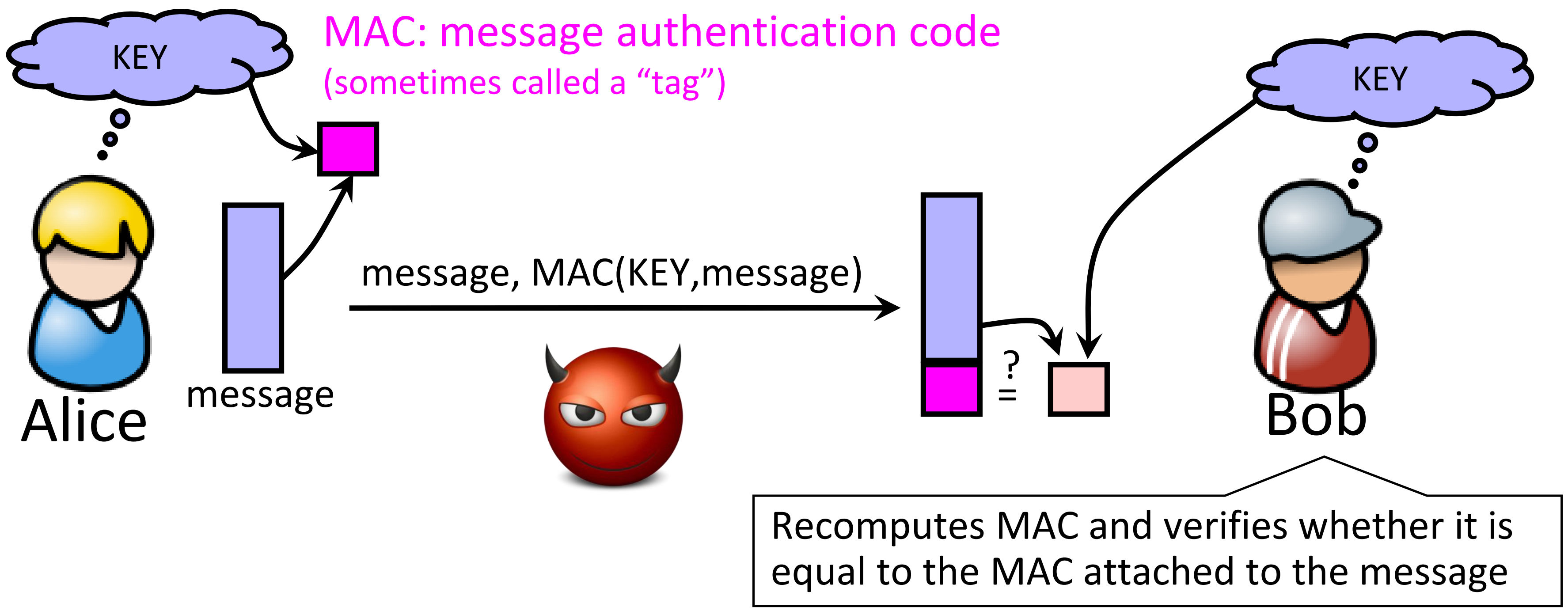


Message = M

Ciphertext = C

Now: Achieving Integrity

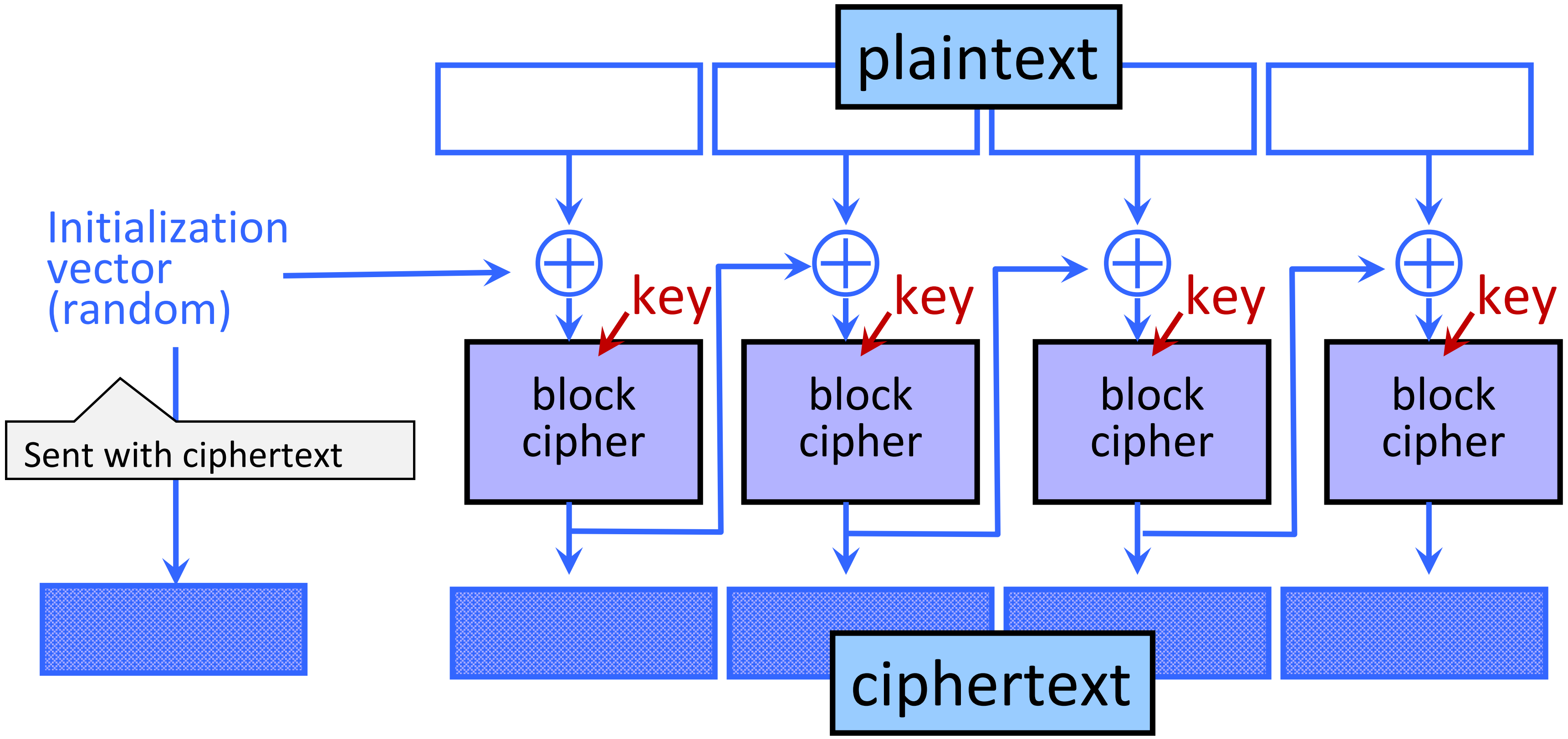
Message authentication schemes: A tool for protecting integrity.


KEY
MAC: message authentication code
(sometimes called a "tag")
KEY
message, MAC(KEY,message)
Alice
message
?
=
Bob

Recomputes MAC and verifies whether it is equal to the MAC attached to the message

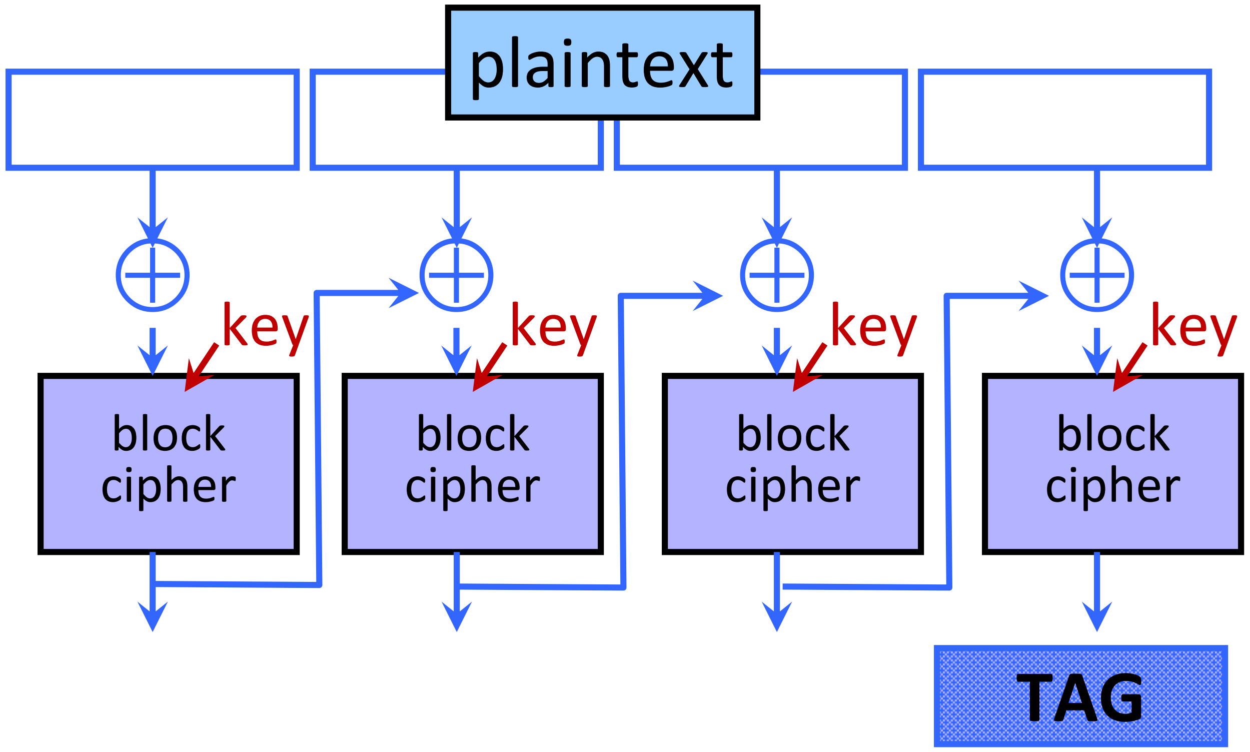
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

Reminder: CBC Mode Encryption



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

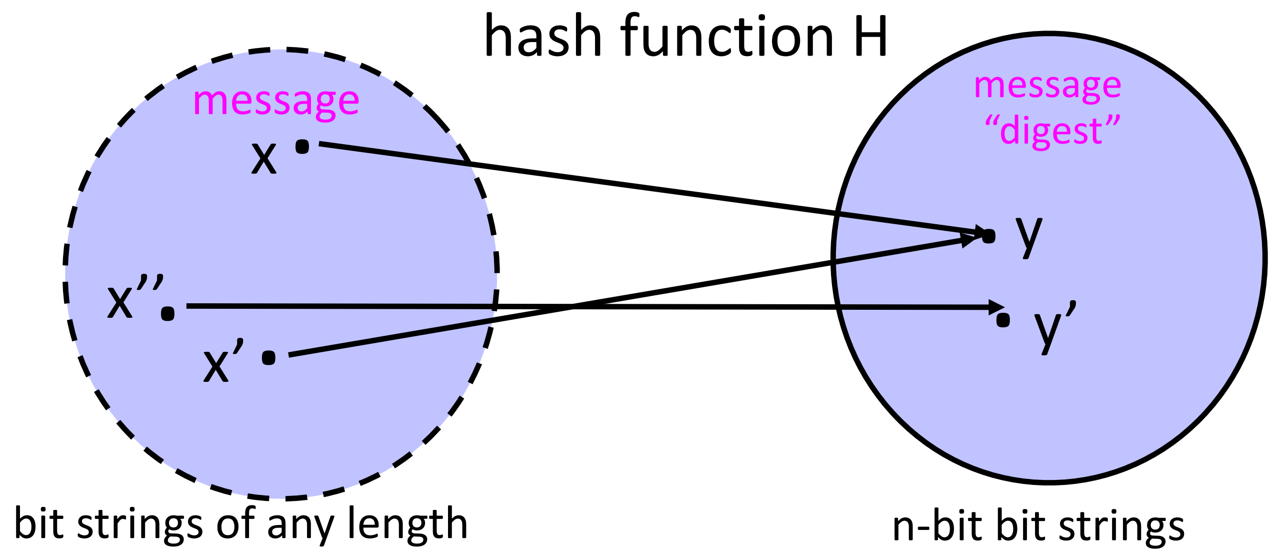
CBC-MAC



- Not secure when system may MAC messages of different lengths
- Use a different key – not encryption key
- NIST recommends a derivative called CMAC [FYI only]

Another Tool: 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 your part of the 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.

Collision resistance does **not** imply one-wayness.

You can prove this by constructing a function that has one property but not the other.

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.
  - … Does it imply one-way-ness/preimage resistance?

One-Way vs. Collision Resistance
(Details here mainly FYI)

- One-wayness does <u>not</u> imply collision resistance
  - Suppose g is one-way
  - Define h(x) as g(x') where x' is x except drop 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 <u>not</u> 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 ½

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

- Gradesceope!

- 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?
- How might you migrate a system from encryption to hashing?

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!
- Don’t need to worry about where to store the key!
- Cannot go from hash to password!

Application: Password Hashing

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?

Application: Password Hashing + Salting

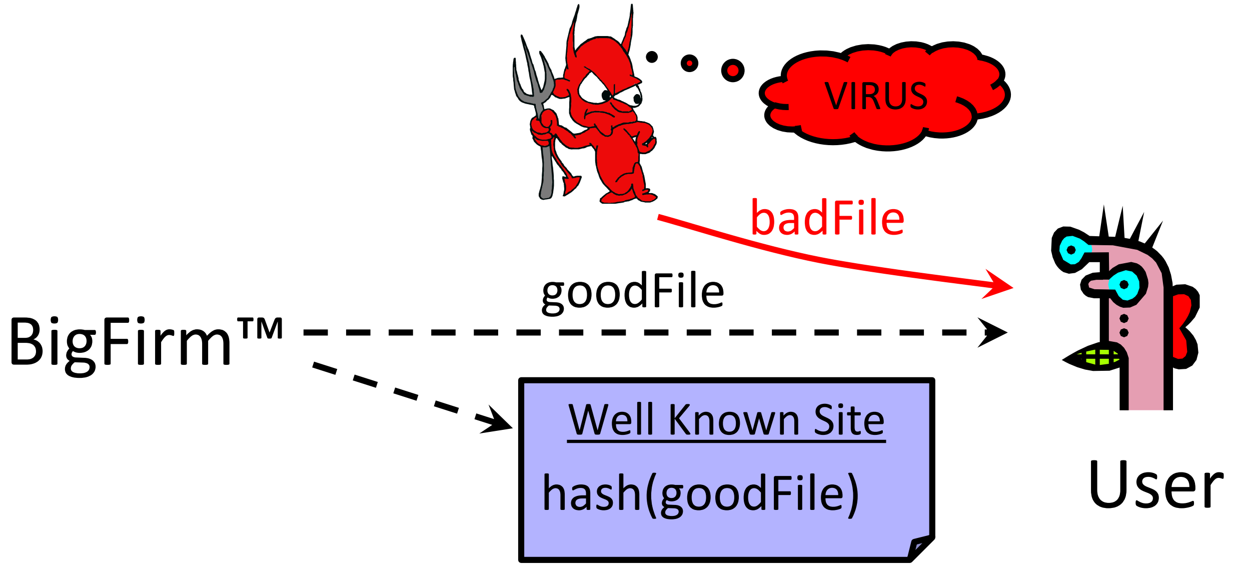
- Salting
  - We ‘salt’ hashes for password by adding a randomized suffix to the password
    - E.g. Hash(“coolpassword”+”35B67C2A”)
  - We then store the salt with the hashed password!
  - Server generates the salt

- The goal is to prevent *precomputation attacks*
  - If the adversary doesn’t know the salt, they can’t *precompute* common passwords

Hash Functions Review

- Map large domain to small range (e.g., range of all 160- or 256-bit values)
- Properties:
  - Collision Resistance: Hard to find two distinct inputs that map to same output
  - One-wayness: Given a point in the range (that was computed as the hash of a random domain element), hard to find a preimage
  - Weak Collision Resistance: Given a point in the domain and its hash in the range, hard to find a new domain element that maps to the same range element

Application: Software Integrity



<u>Goal:</u> Software manufacturer wants to ensure file is received by users without modification.

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

Application: Software Integrity

- Which property do we need?
    - One-wayness?
    - (At least weak) Collision resistance?
    - Both?

Which Property Do We Need?

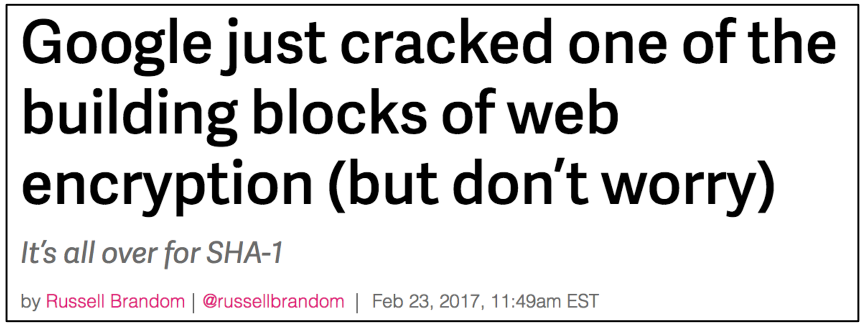
- Passwords stored as hash(password)
  - **One-wayness:** hard to recover the/a valid password
- Integrity of software distribution
  - **Weak collision resistance**
  - But software images are not really random… may need **full collision resistance** if considering malicious developers
- Commitments (e.g. auctions)
  - Alice wants to bid B, sends H(B), later reveals B
  - **One-wayness:** rival bidders should not recover B (this may mean that they need to hash some randomness with B too)
  - **Collision resistance:** Alice should not be able to change their mind to bid B' such that H(B)=H(B')

Commitments

Common Hash Functions

- **SHA-2: SHA-256, SHA-512, SHA-224, SHA-384**
- **SHA-3: standard released by NIST in August 2015**
- MD5 – Don’t Use!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- 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!
- Plenty of others, some OK some not
  - E.g. The BLAKE family seems fine

SHA-1 Broken in Practice (2017)

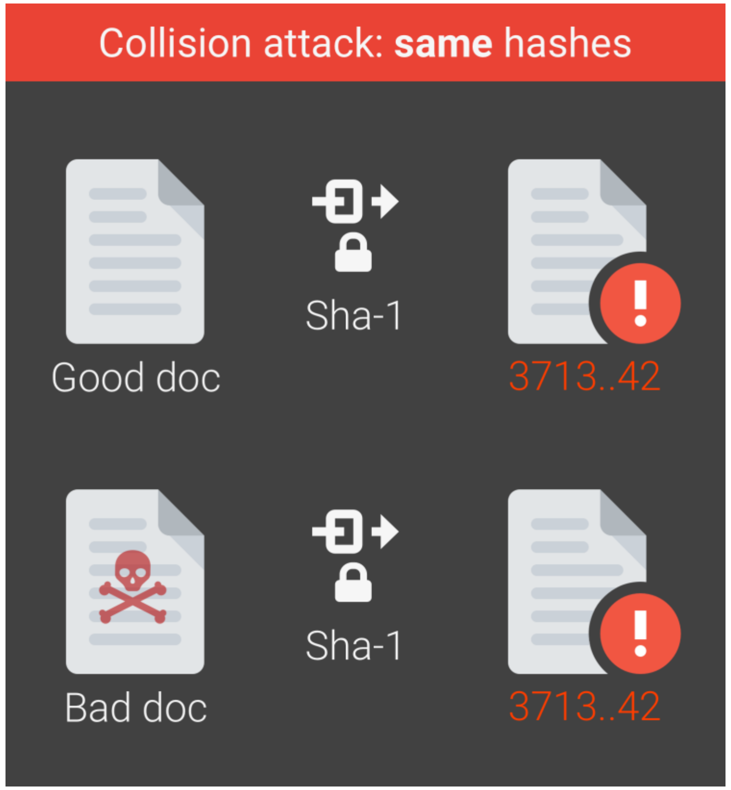


Google just cracked one of the building blocks of web encryption (but don't worry)

It's all over for SHA-1

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

https://shattered.io

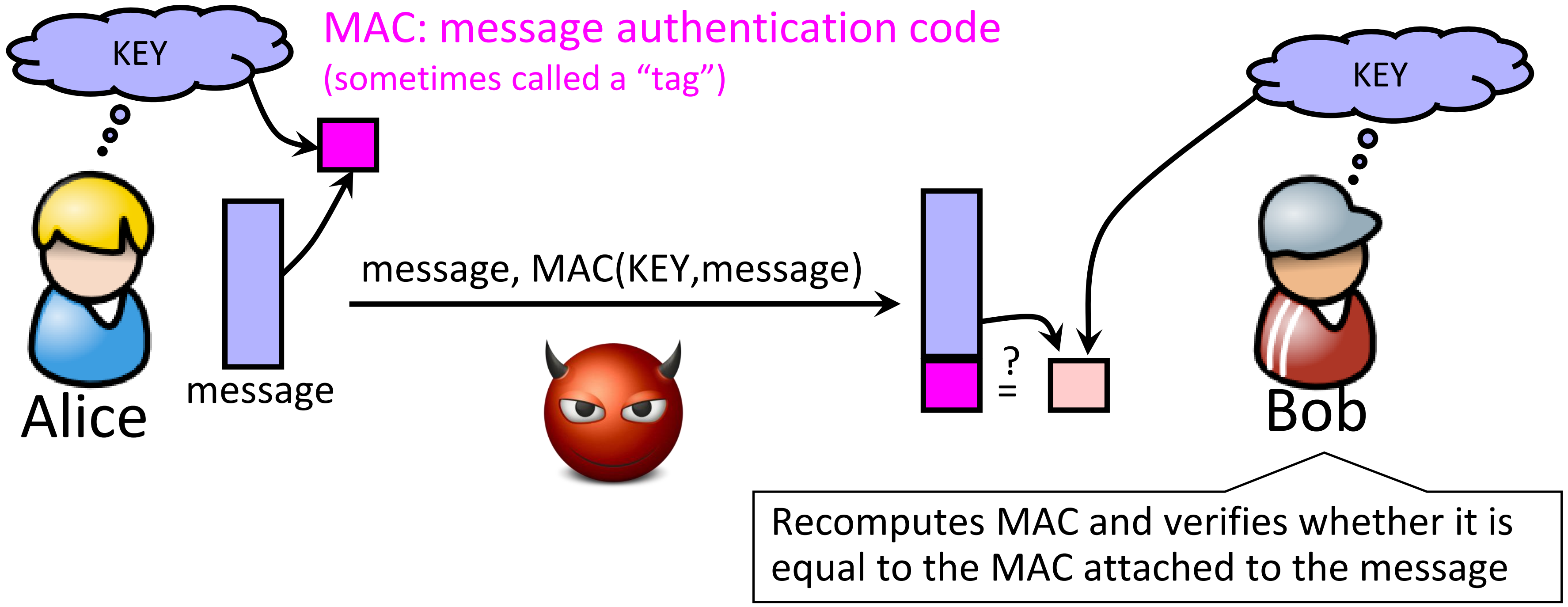


Aside: How we evaluate hash functions

- Speed
  - Is it amenable to hardware implementations?
- Diffusion
  - Does changing 1 bit in the input affect all output bits?
- Resistance to attack approaches
  - Collisions?
  - Length extensions?
  - etc

Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.


KEY
MAC: message authentication code
(sometimes called a "tag")
KEY
message, MAC(KEY,message)
Alice
message
?
=
Bob

Recomputes MAC and verifies whether it is equal to the MAC attached to the message

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? (Historical reasons)
  - 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

MAC with SHA3

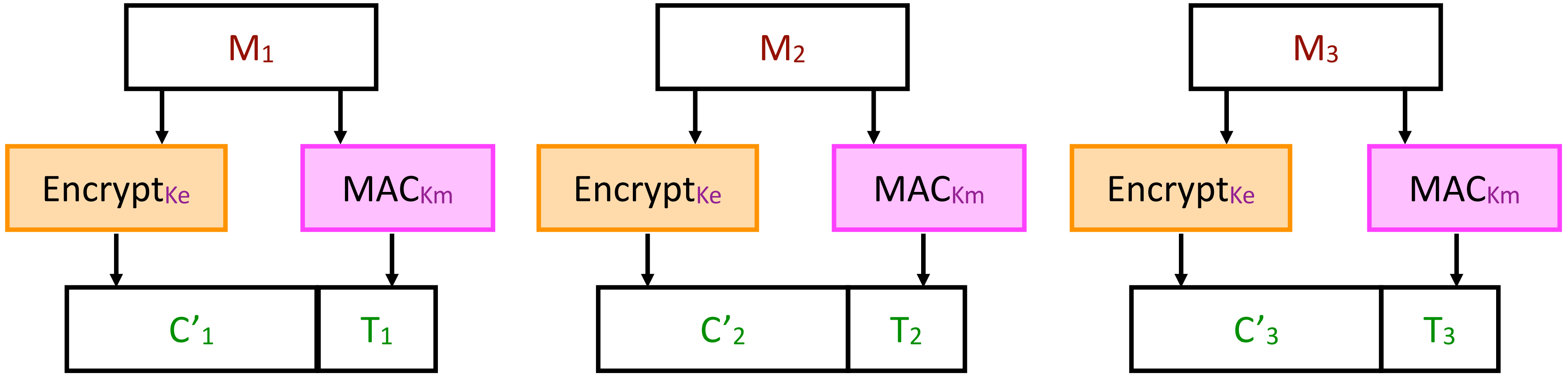
- SHA3(Key || Message)

- SHA3 is designed to get the same safety properties as HMAC constructions

Authenticated Encryption

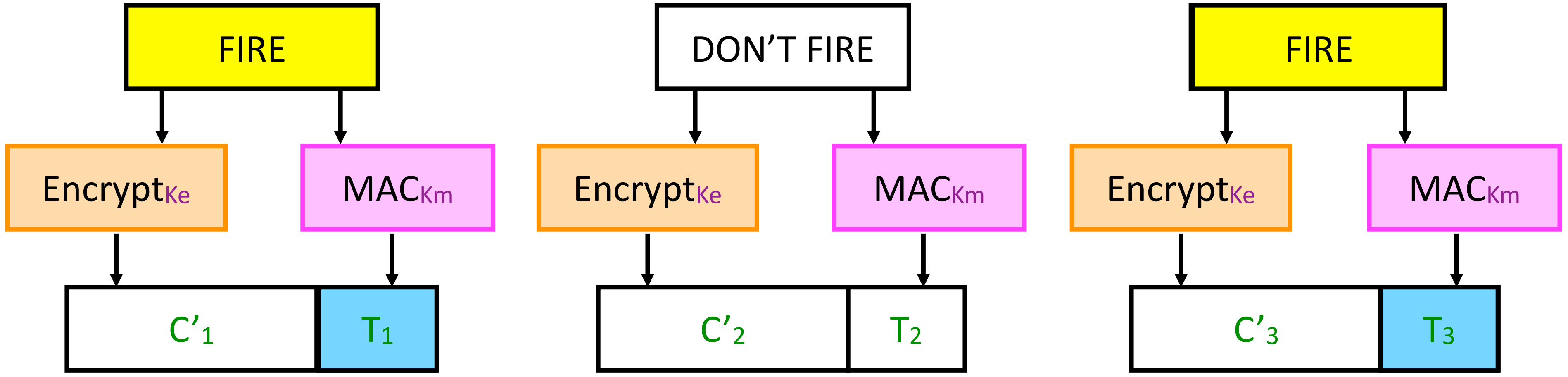
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.

- Is this fine? (Gradescope!)



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

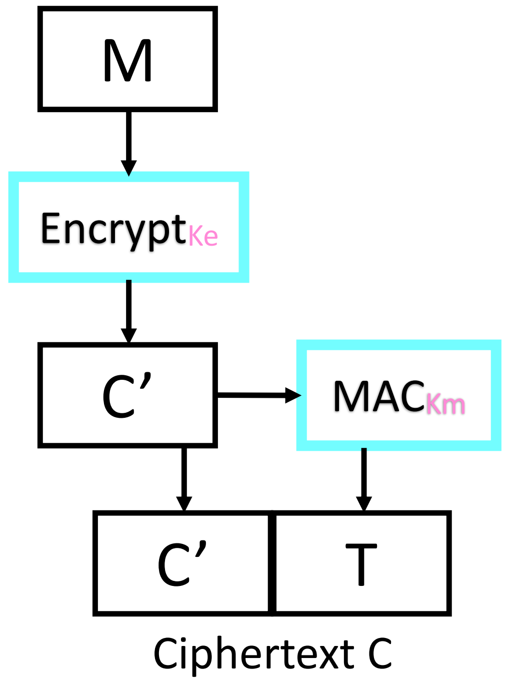


Authenticated Encryption

- Instead:

  Encrypt *then* MAC.

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



Encrypt-then-MAC

Back to asymmetric cryptography

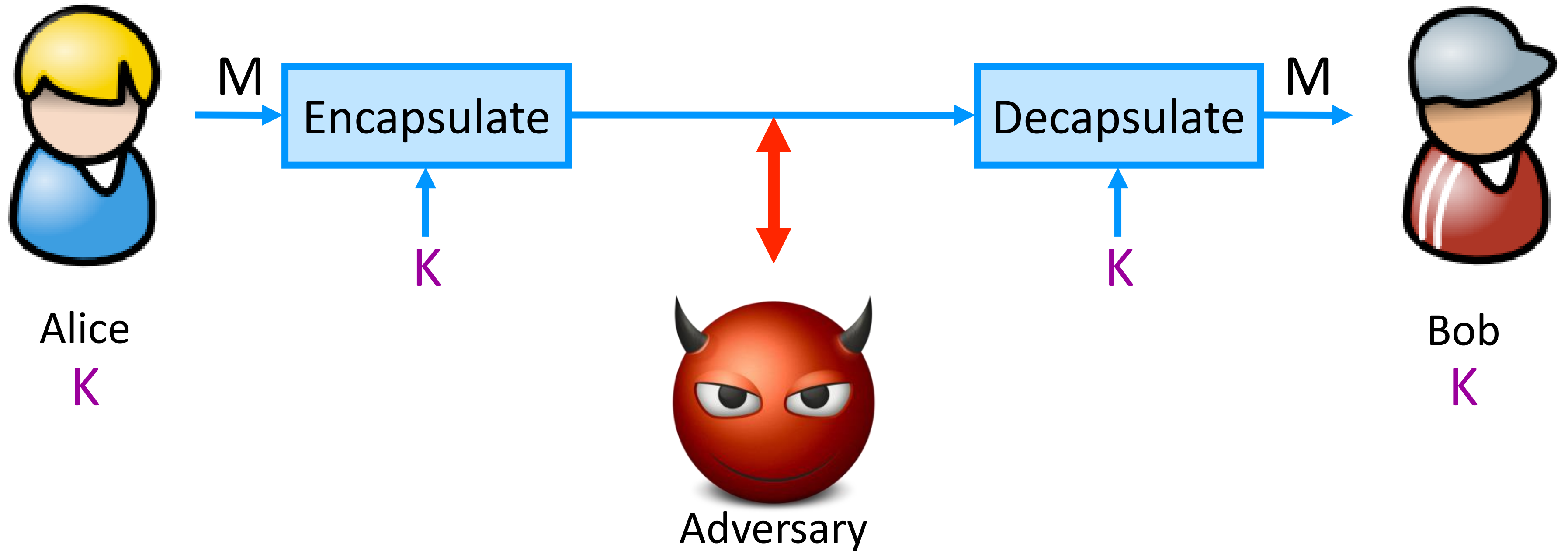
Stepping Back: Flavors of Cryptography

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

- Asymmetric cryptography
  - Each party creates a public key pk and a secret key sk.

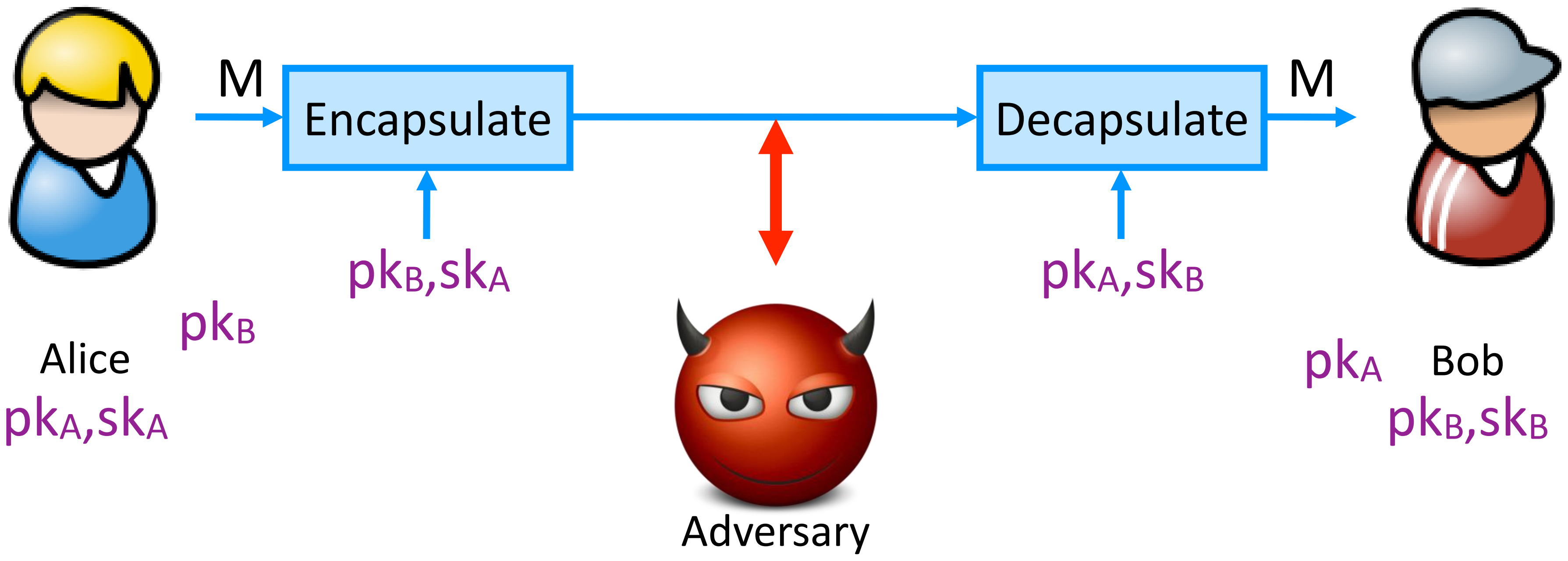
Symmetric Setting

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

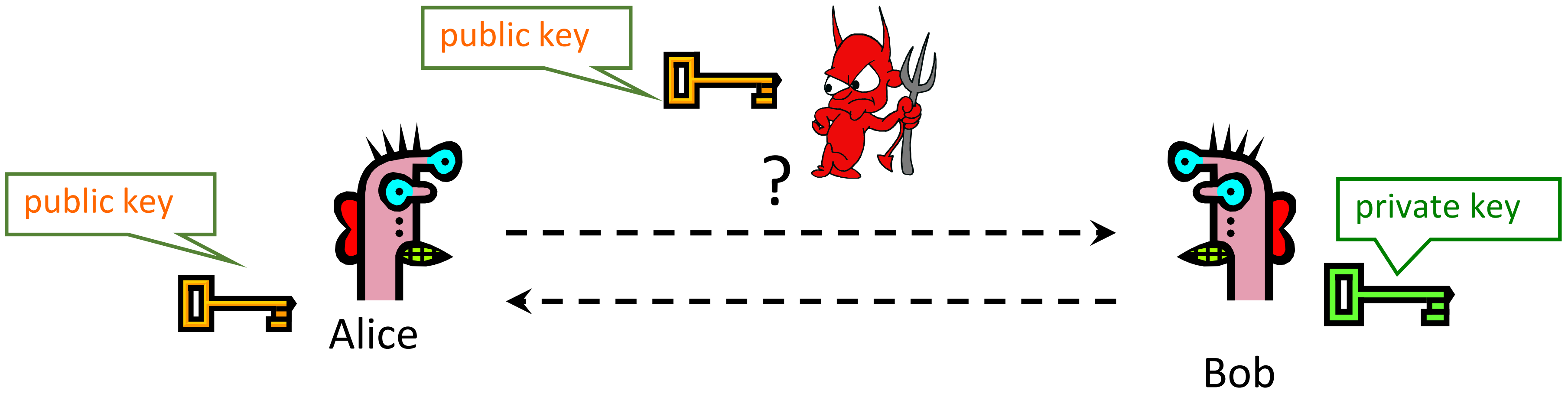


Asymmetric Setting

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



Public Key Crypto: Basic Problem



Ignore for now: How do we know it's REALLY Bob's??

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 themself

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?)

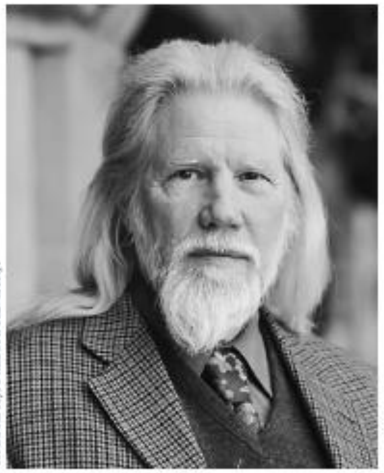
Session Key Establishment

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}*

Diffie-Hellman Protocol (1976)

Diffie and Hellman Receive 2015 Turing Award


Rod Searcey/Stanford University.

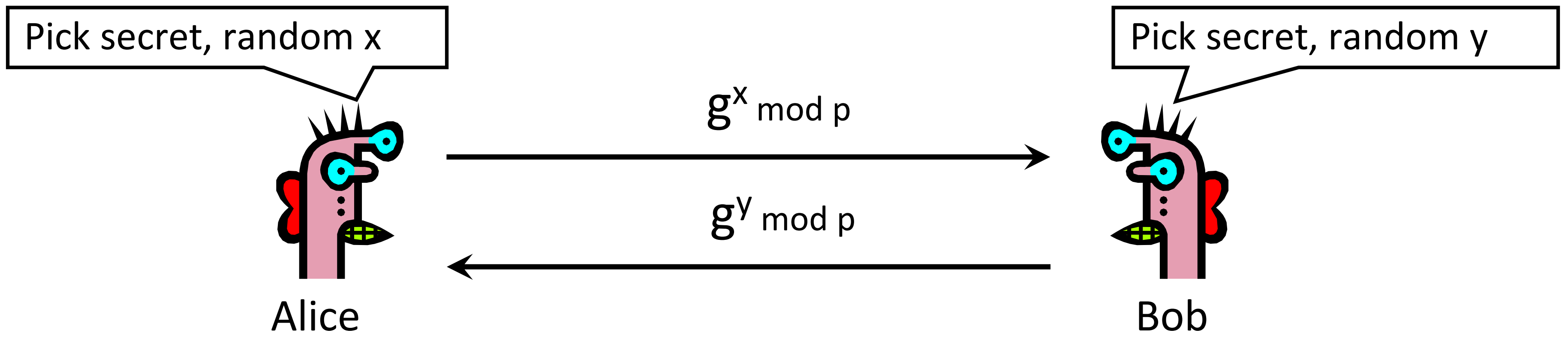
Whitfield Diffie

Linda A. Cicero/Stanford News Service.

Martin E. Hellman

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}; a Z_p* i such that a=g^i mod p
    - Modular arithmetic: numbers "wrap around" after they reach p



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

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

Example Diffie Hellman Computation

Why is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:

  given $g^x \bmod p$, it's hard to extract x

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

- Computational Diffie-Hellman (CDH) problem:

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

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

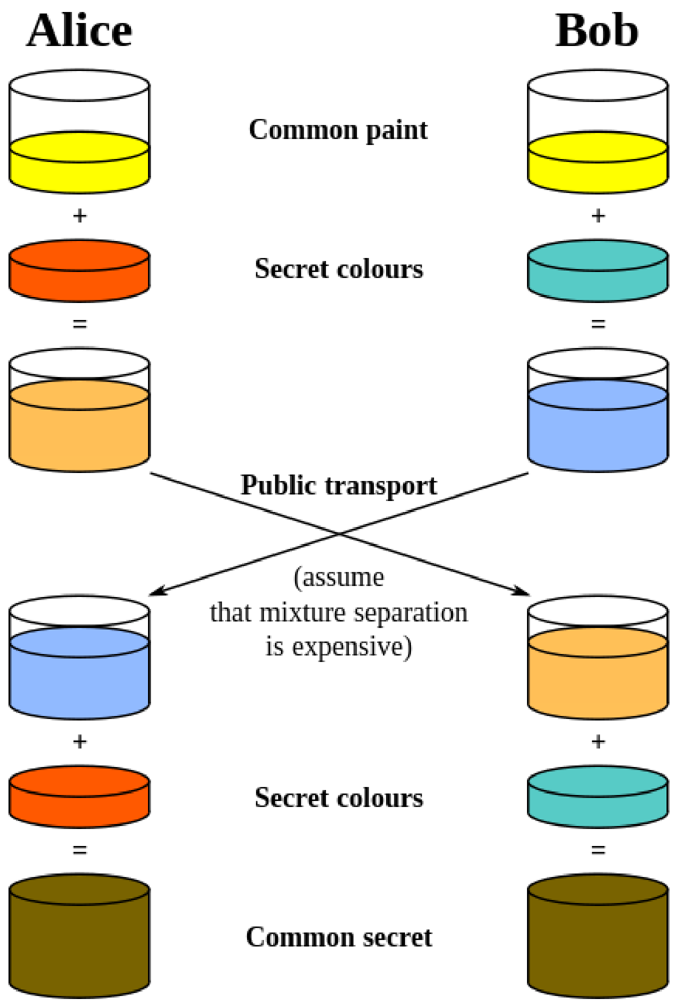
- Decisional Diffie-Hellman (DDH) problem:

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

More on Diffie-Hellman Key Exchange

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

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:
g^x mod p
g^y mod p

Common secret: g^{xy} mod p

[from Wikipedia]

Diffie-Hellman Caveats

- 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*
    - DDH is hard in this group
  - Eavesdropper can't tell the difference between the established key and a random value
  - In practice, 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 (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Example from Earlier

- 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) (5 is a prime)
    - g=3 generates a group of prime order

Stepping Back: Asymmetric Crypto

- We’ve just seen session key establishment
  - Can then use shared key for symmetric crypto
- Next: public key encryption
  - For confidentiality
- Then: digital signatures
  - For authenticity

Requirements for Public Key Encryption

- Key generation: computationally easy to generate a pair (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 $\varphi(n)$ $(n \geq 1)$ 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
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that $\varphi(ab) = \varphi(a)\ \varphi(b)$ if a & b are relatively prime

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
  - 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);

Secret key = (d,n)

Encryption of m: c = m^e mod n

Decryption of c: c^d mod n = $(m^e)^d$ mod n = m

Why is RSA Secure?

- RSA problem:
  - Given c, $n=pq$, and e such that $\gcd(e, \varphi(n))=1$
  - Find m such that $m^e=c \bmod n$

- 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 *without* knowing p and q

Why is RSA Secure?

- There is no known efficient algorithm for doing this *without* knowing p and q

- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ 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

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

RSA OAEP $M \oplus G(r) \,||\, r \oplus H(M \oplus G(r))$

Review: RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 2048 bits each (need primality testing, too)
  - 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: $d \equiv e^{-1} \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Actually, RSA is bad and stop using it

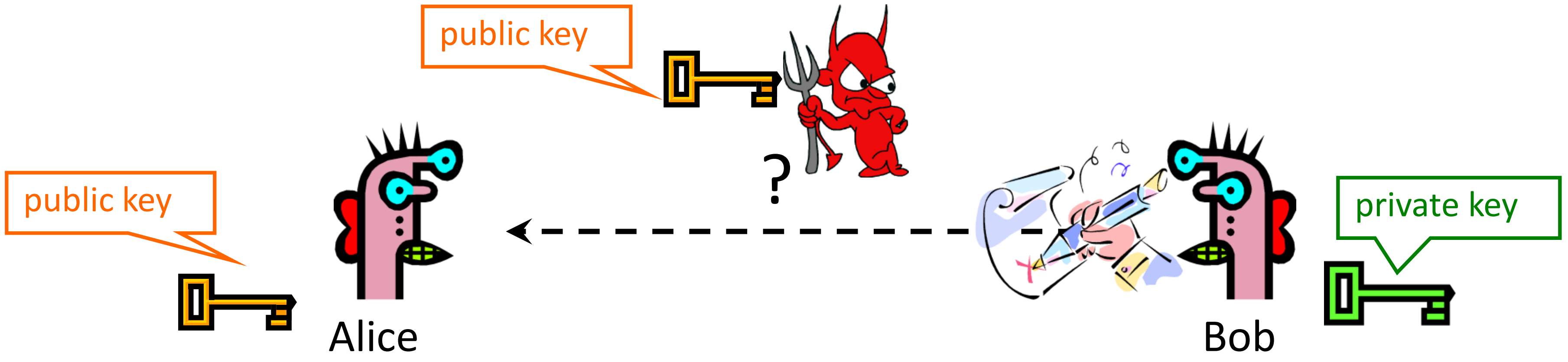
- Math is OK, implementation isn’t
  - Yes, all the implementations

- https://blog.trailofbits.com/2019/07/08/fuck-rsa/

- Sorry I just spent time teaching it to you
  - Maybe you would’ve preferred projected coordinate math on elliptic curves?

Using public key cryptography… backwards

Digital Signatures: Basic Idea



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. 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: $s = m^d \bmod n$
  - 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 $s^e \bmod n = (m^d)^e \bmod n = m$
  - Just like encryption (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Without padding and hashing: Consider multiplying two signatures together
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

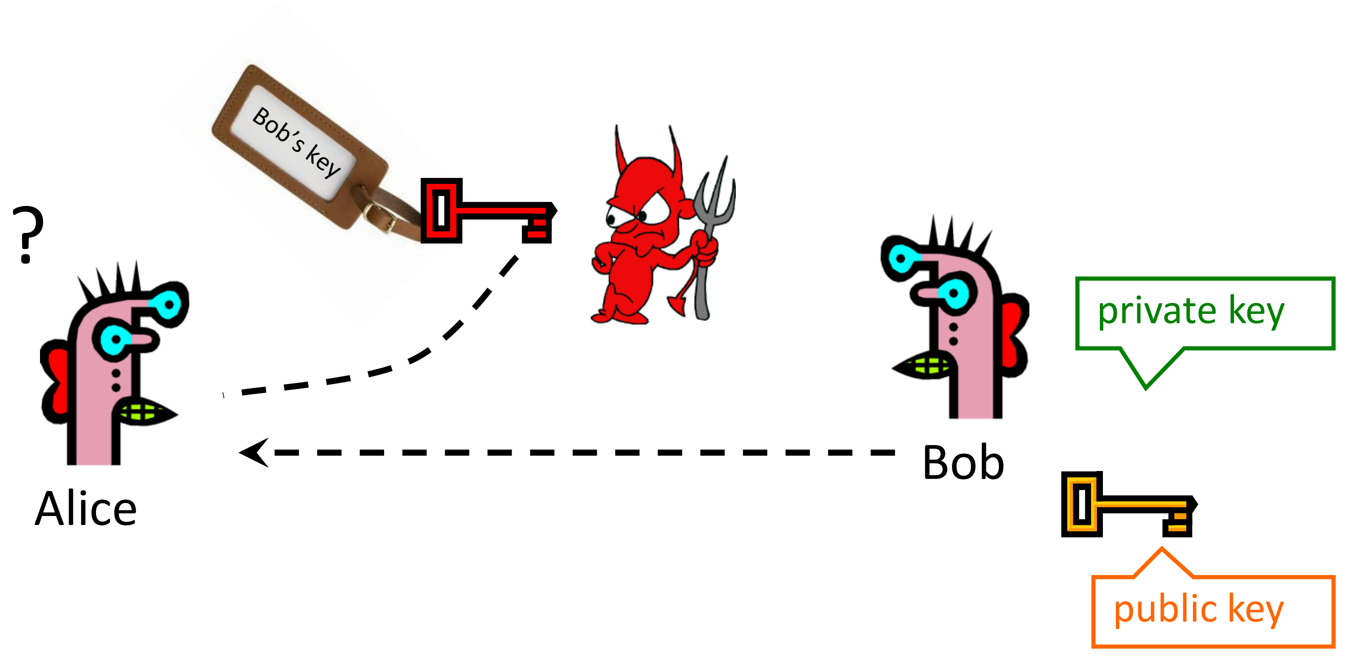
- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- Public key: (p, q, g, $y=g^x \bmod p$), private key: x
- Each signing operation picks a new random value, to use during signing. Security breaks if two messages are signed with that same value.
- Security 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)
- Again: We've discussed discrete logs modulo integers; significant advantages to using elliptic curve groups instead.

Post-Quantum

- If quantum computer become a reality
  - It becomes much more efficient to break conventional asymmetric encryption schemes (e.g., factoring becomes “easy”)
  - Easy is a very relative term, but Shor’s Algorithm is compelling.

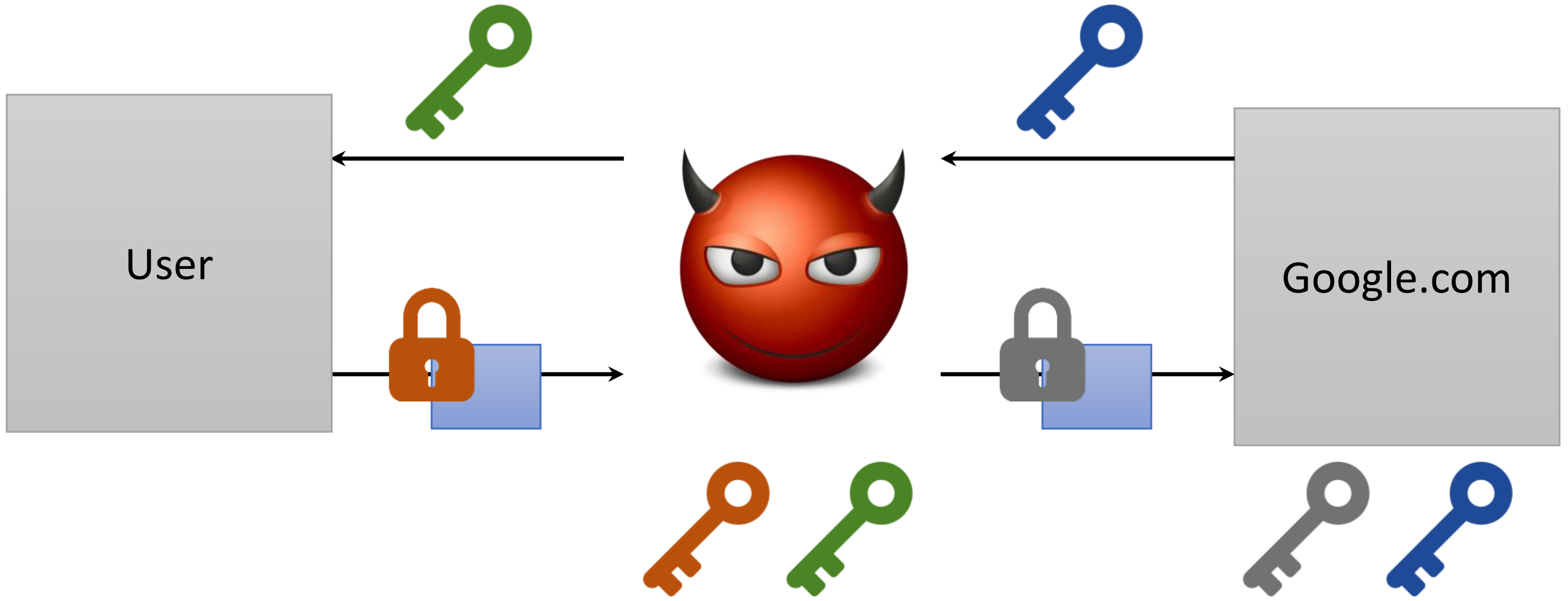
- There exists efforts to make quantum-resilient asymmetric encryption schemes
  - (Check out NIST’s PQC competition!)
  - Current likely winner for most things is Kyber aka ML-KEM

Authenticity of Public Keys



<u>Problem</u>: How does Alice know that the public key they received is really Bob's public key?

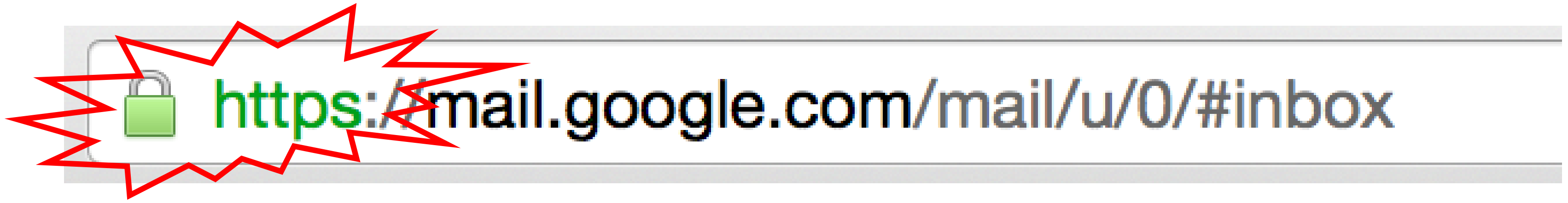
Person-in-the Middle/On-path-attacker



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)
    - Additional information often signed as well (e.g., expiration date)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves their 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

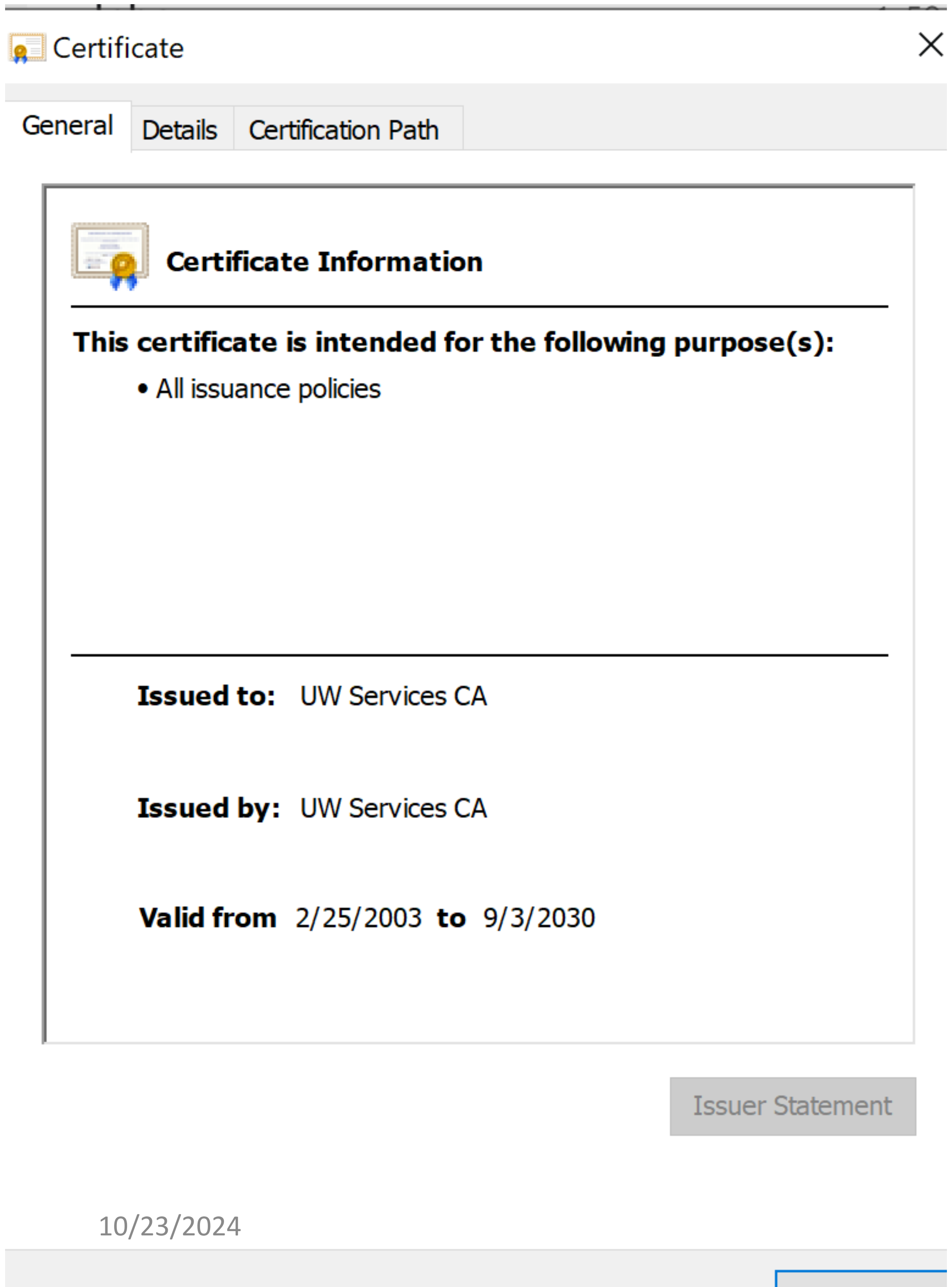
You encounter this every day…


https://mail.google.com/mail/u/0/#inbox

SSL/TLS: Encryption & authentication for connections

SSL/TLS High Level

- SSL/TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret symmetric key established in the handshake protocol to protect communication between the client and the server


Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

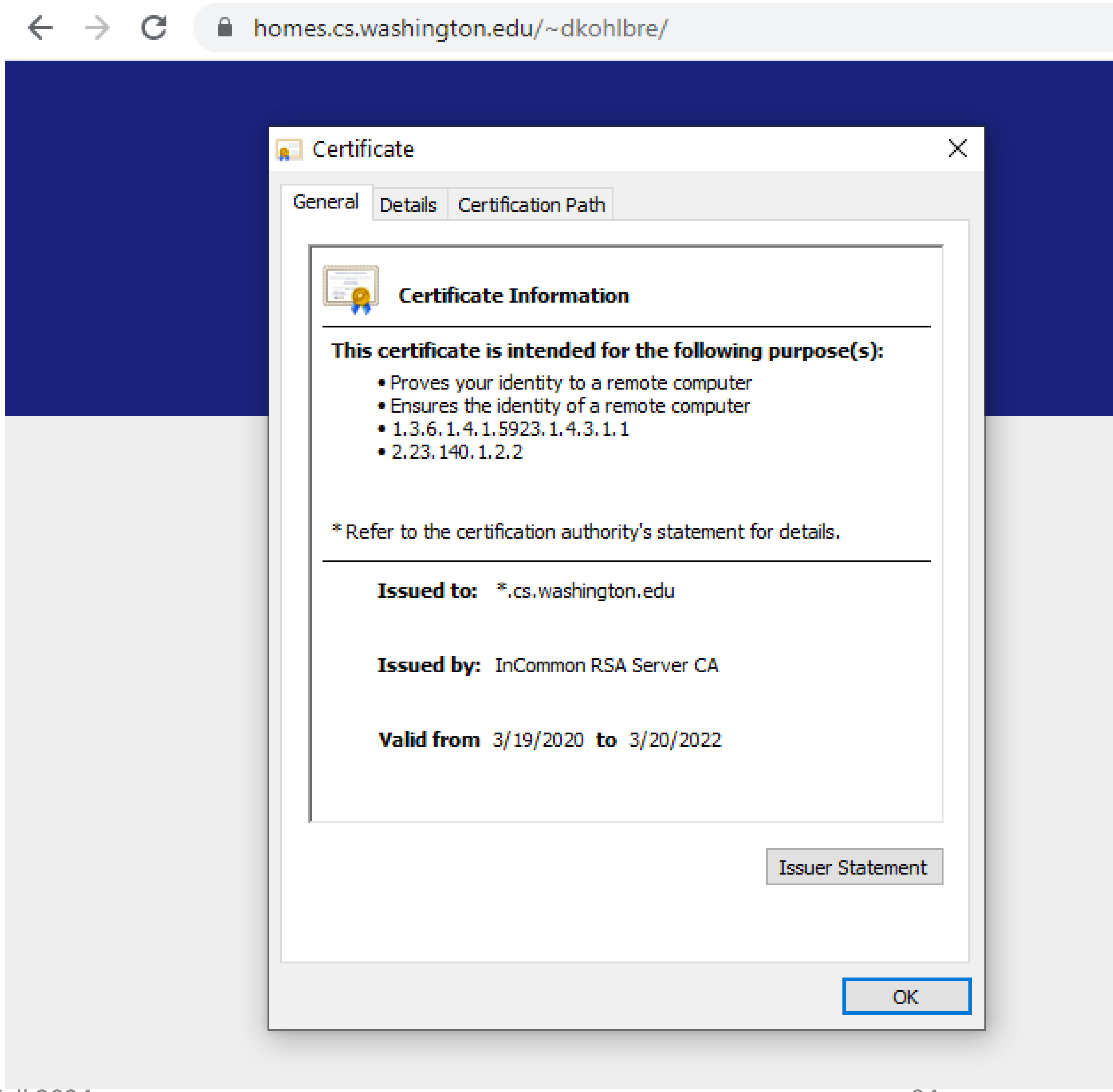
- All issuance policies

Issued to: UW Services CA

Issued by: UW Services CA

Valid from 2/25/2003 **to** 9/3/2030

Issuer Statement


homes.cs.washington.edu/~dkohlbre/

Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

- Proves your identity to a remote computer
- Ensures the identity of a remote computer
- 1.3.6.1.4.1.5923.1.4.3.1.1
- 2.23.140.1.2.2

* Refer to the certification authority's statement for details.

Issued to: *.cs.washington.edu

Issued by: InCommon RSA Server CA

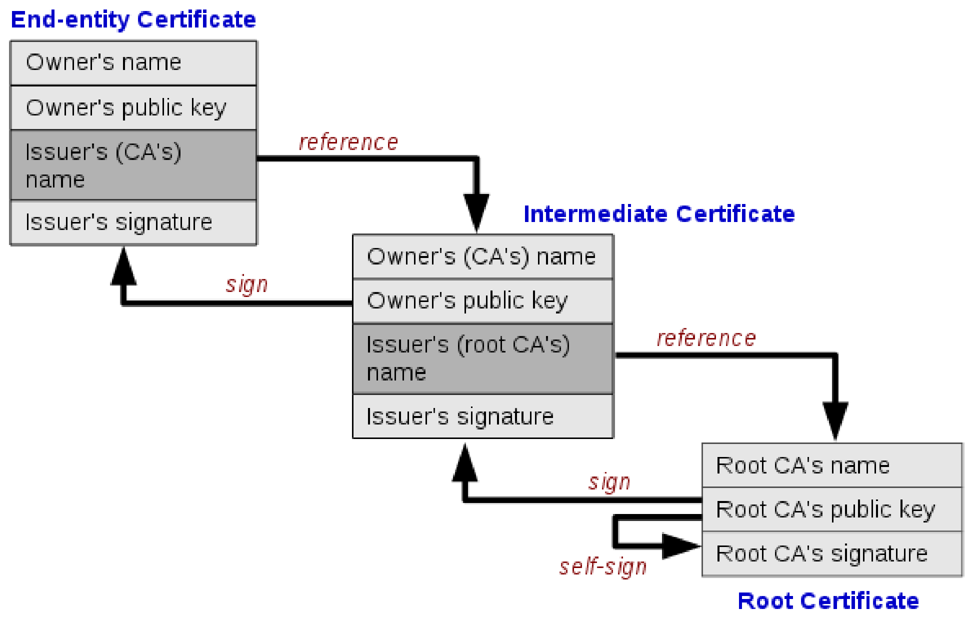
Valid from 3/19/2020 **to** 3/20/2022

Issuer Statement

OK

Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority (e.g., Verisign)
  - Everybody must know the root's public key
  - Instead of single cert, use a certificate chain
    - $\mathrm{sig}_{\mathrm{Verisign}}$("AnotherCA", $PK_{\mathrm{AnotherCA}}$), $\mathrm{sig}_{\mathrm{AnotherCA}}$("Alice", PK_A)
  - Not shown in figure but important:
    - Signed as part of each cert is whether party is a CA or not



- What happens if root authority is ever compromised?