CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

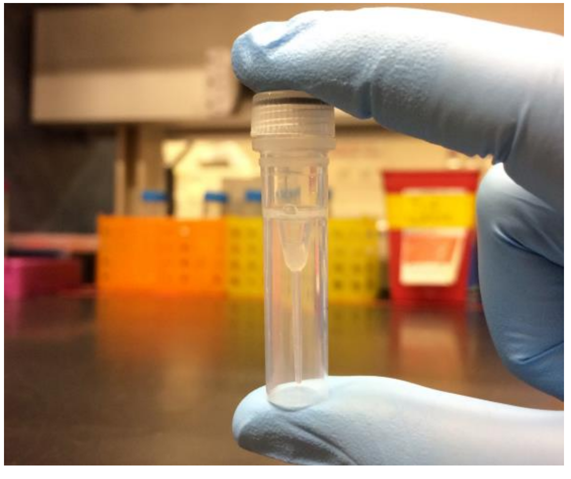
Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Franzi Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

- My office hours next week: Wed 12:30pm, CSE1 403
- HW 2 available
- Lab 1 due
- Book suggestions: Mitnick on Social Engineering; Chris Hadnagy on Social Engineering; "No Tech Hacking" book

Research Discussions

- Monday (10/14): Peter Ney on Bio-Cyber Security and Cell Site Simulators
- Monday (10/21): Karl Koscher on Automotive Cyber Security
- Wednesday (10/23): Ivan Evtimov on Adversarial Machine Learning
- Monday (10/28): Emily McReynolds on Law and Policy



Begin Crypto Review

- Also quiz section yesterday
- Also good anyway, recalling "spiral learning" process

Flavors of Cryptography

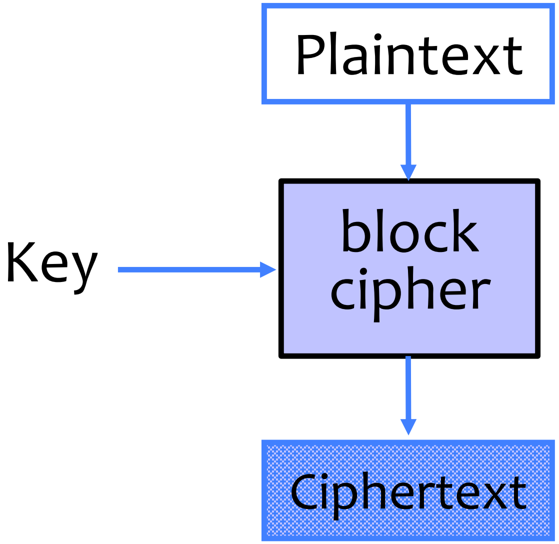
- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.
- Asymmetric cryptography
  - Each party creates a public key pk and a secret key sk.

How Cryptosystems are Made

- Primitives first (like block ciphers or RSA)
- Schemes second (like ECB, CTR mode)
- Protocols third (like SSL/TLS, SSH)

Block Ciphers and Keyed Permutation

- Not just shuffling of input bits!
  - Suppose plaintext = "111". Then "111" is not the only possible ciphertext!
- Instead:
  - Permutation of possible outputs
  - For N-bit input, $2^N!$ possible permutations
  - Use secret key to pick a permutation



Question from Last Time

- Question written on a worksheet: If/how you would use block ciphers with keys that are larger than blocks
- The way I think about this:
  - Think about keys and blocks separately
  - Keys determine *which* permutation to use
  - Blocks are the inputs/outputs to the keyed permutations

Example: With 3-bit Blocks

Key = 0000000

Input	Output
000	111
001	101
010	001
011	000
100	110
101	010
110	100
111	011

Key = 0000001

Input	Output
000	000
001	101
010	010
011	001
100	100
101	011
110	111
111	110

Key = 0000010

Input	Output
000	001
001	000
010	010
011	011
100	111
101	101
110	100
111	110

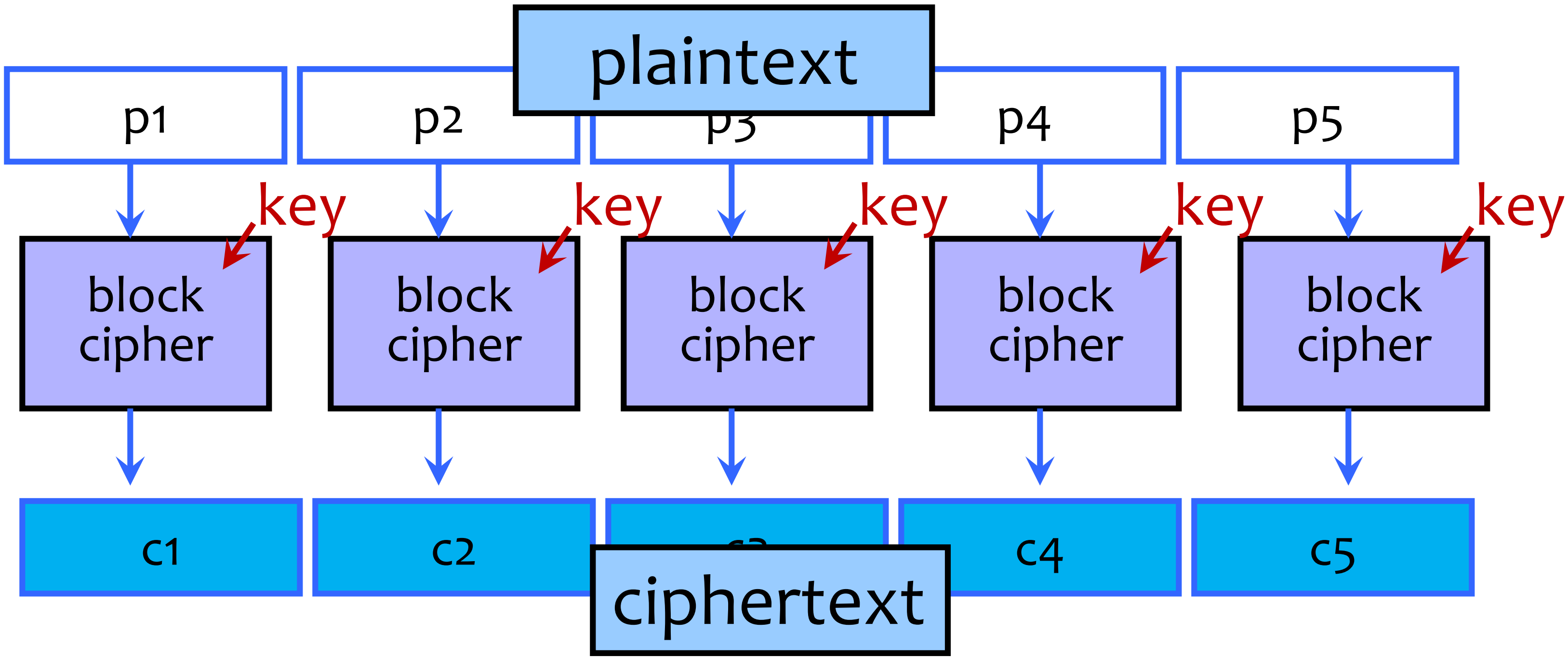
…

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

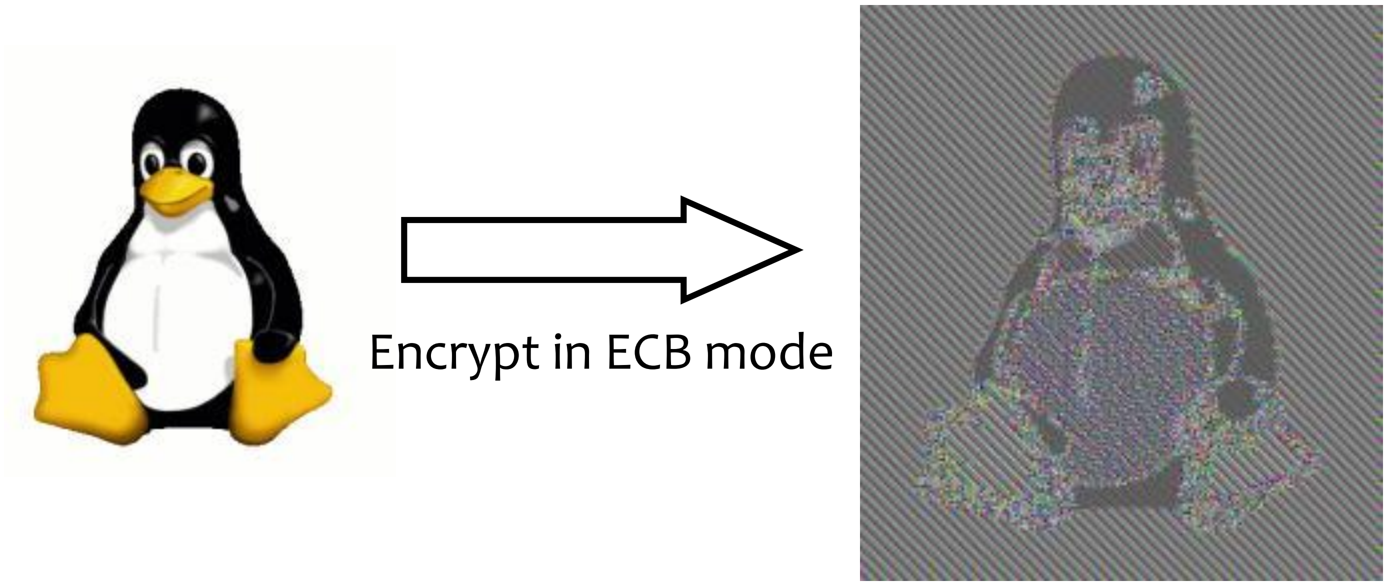
- **AES: Advanced Encryption Standard**
  - New federal standard as of 2001
    - NIST: National Institute of Standards & Technology
  - Based on the Rijndael algorithm
    - Selected via an open process
  - 128-bit blocks, keys can be 128, 192 or 256 bits

Electronic Code Book (ECB) Mode



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

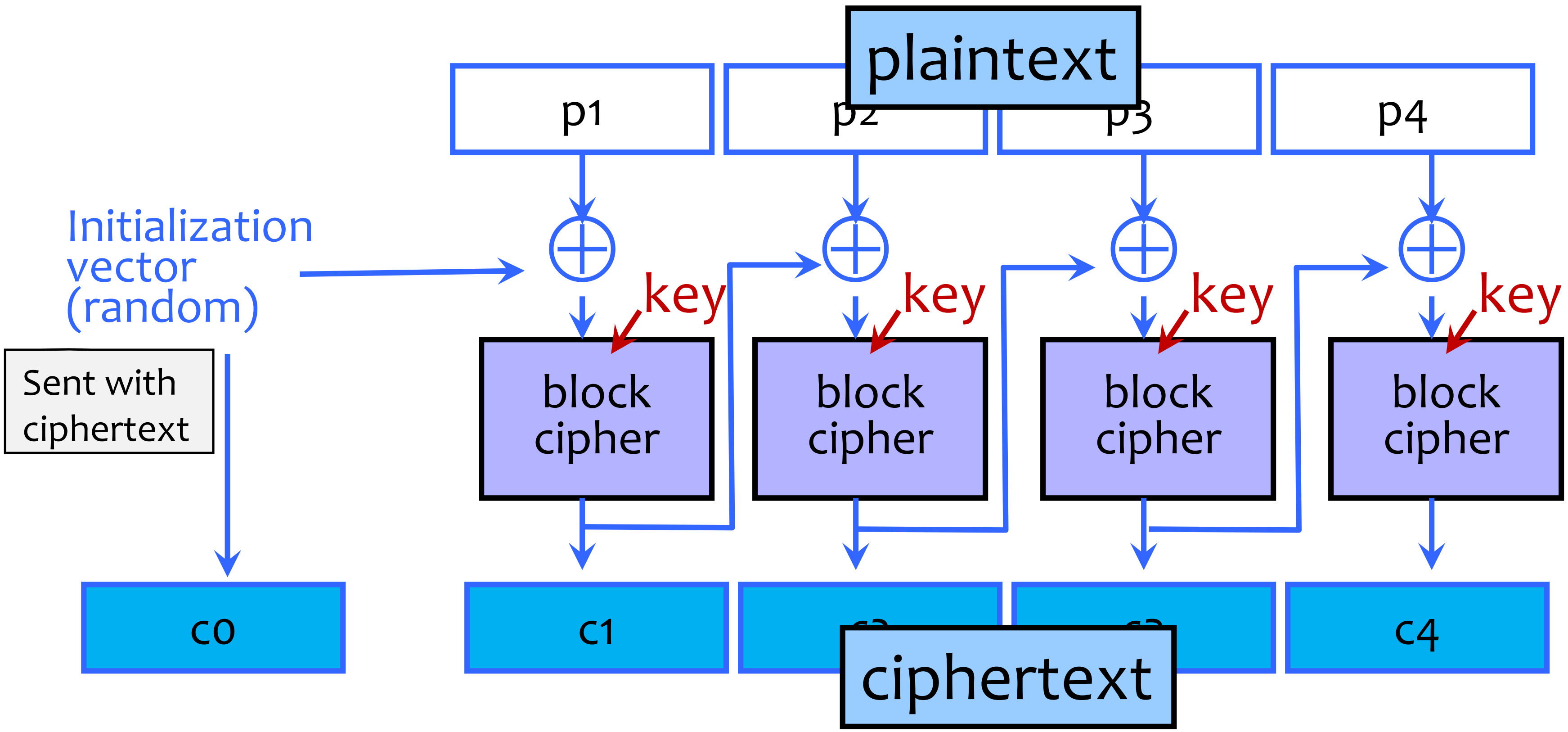
Information Leakage in ECB Mode



[Wikipedia]

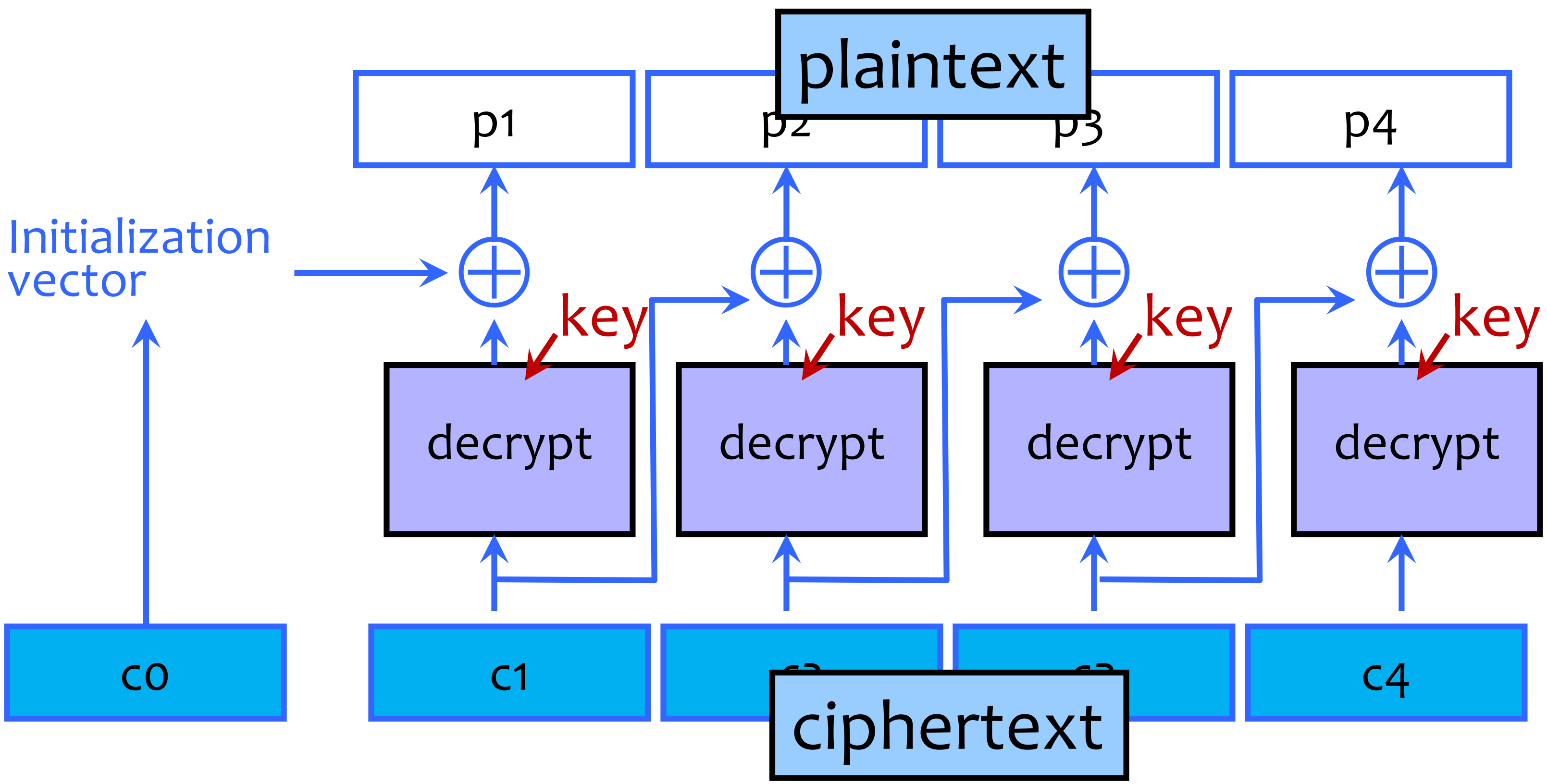
End Review

Cipher Block Chaining (CBC) Mode: Encryption

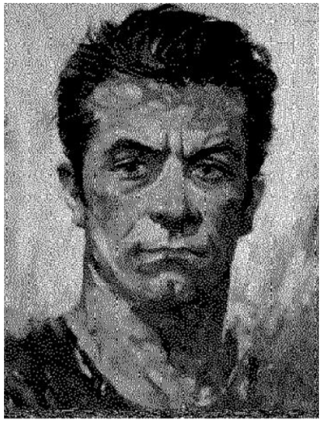


- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

CBC Mode: Decryption

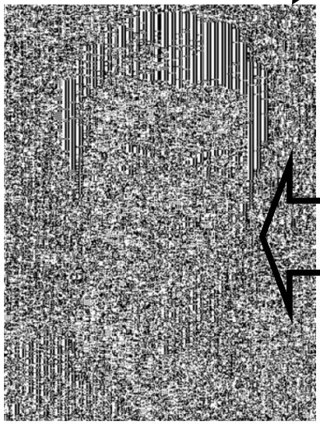


ECB vs. CBC

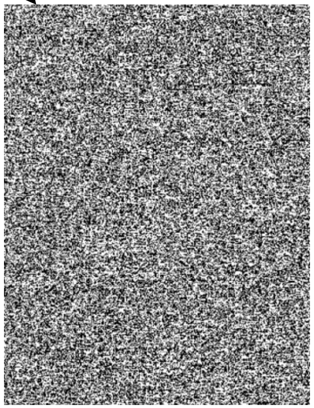


AES in ECB mode

AES in CBC mode

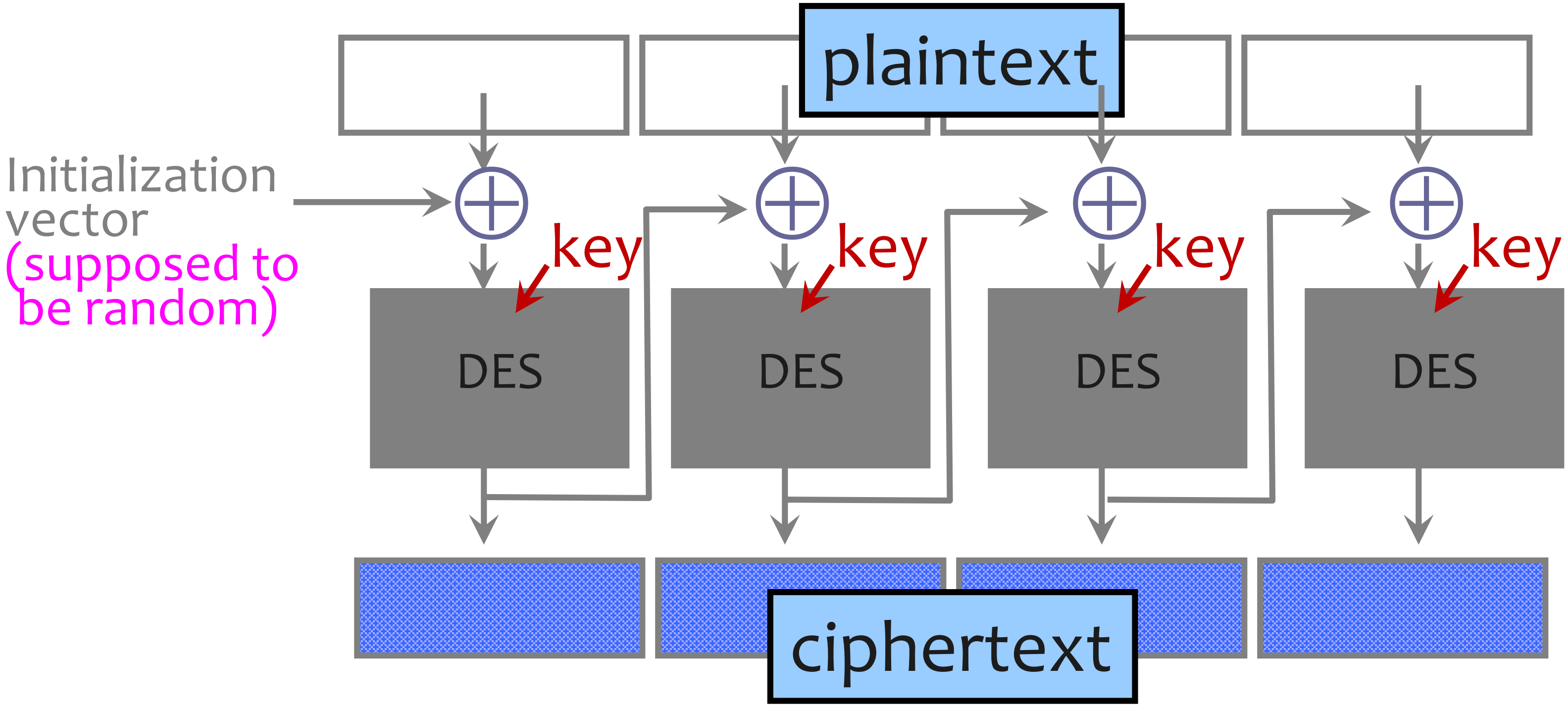


Similar plaintext blocks produce similar ciphertext blocks (not good!)



[Picture due to Bart Preneel]

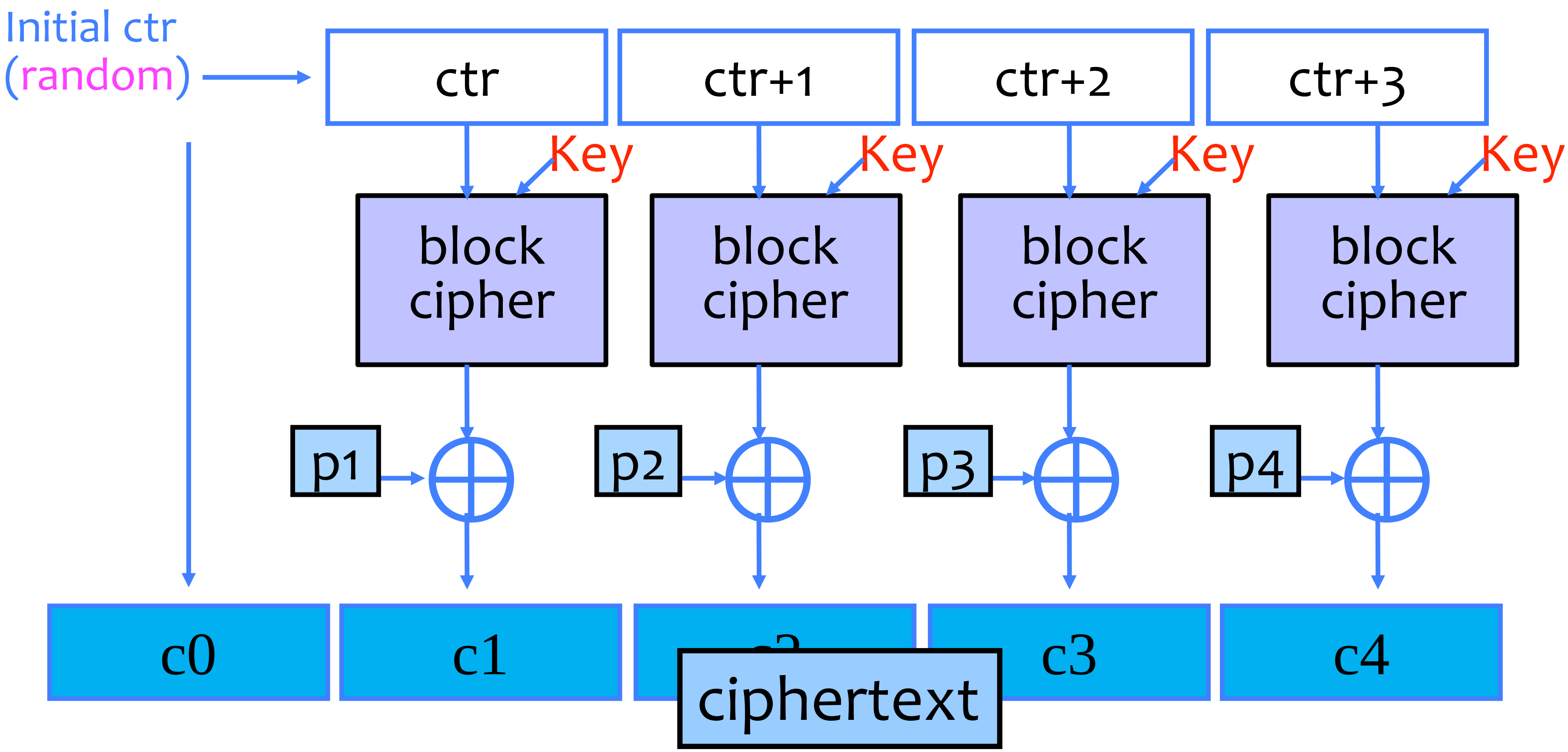
CBC and Electronic Voting



Found in the source code for Diebold voting machines:

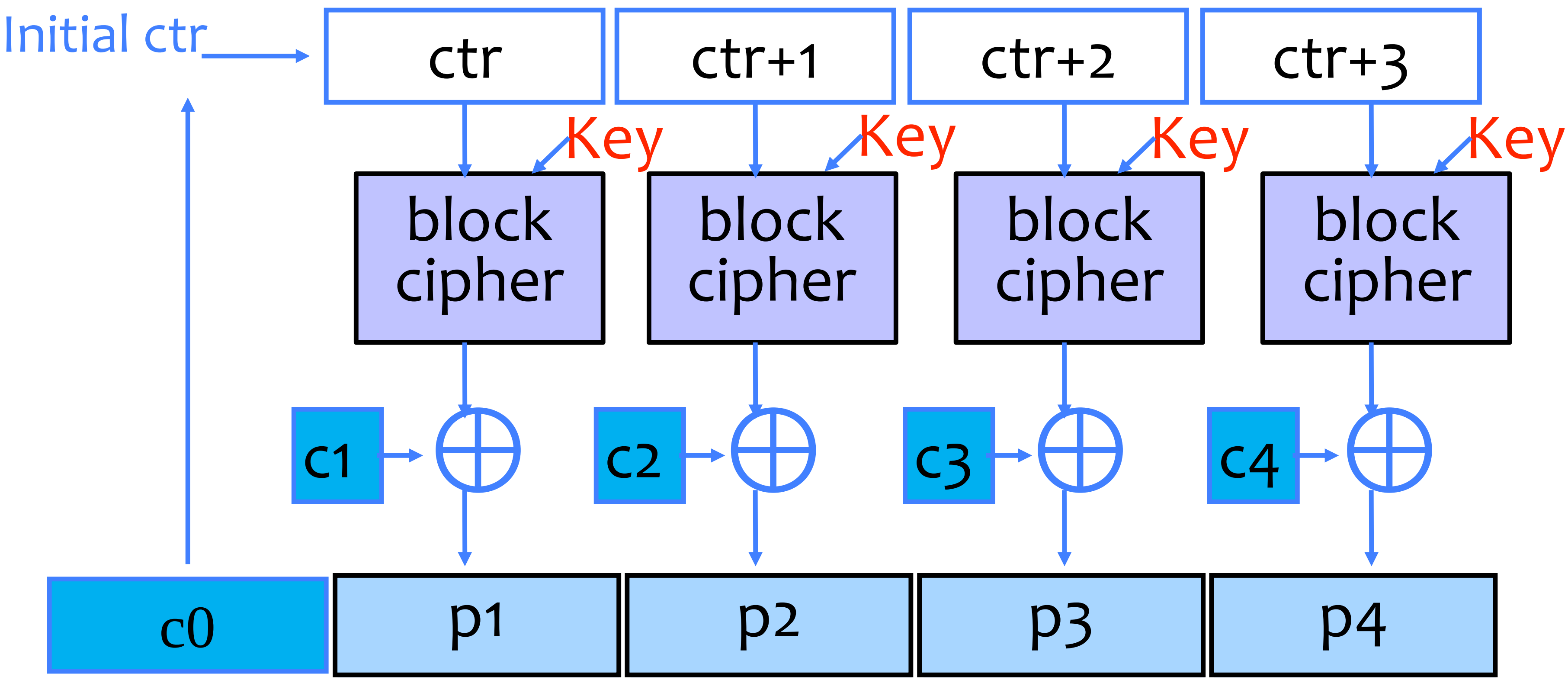
```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
        totalSize, DESKEY, NULL, DES_ENCRYPT)
```

Counter Mode (CTR): Encryption



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

Counter Mode (CTR): Decryption



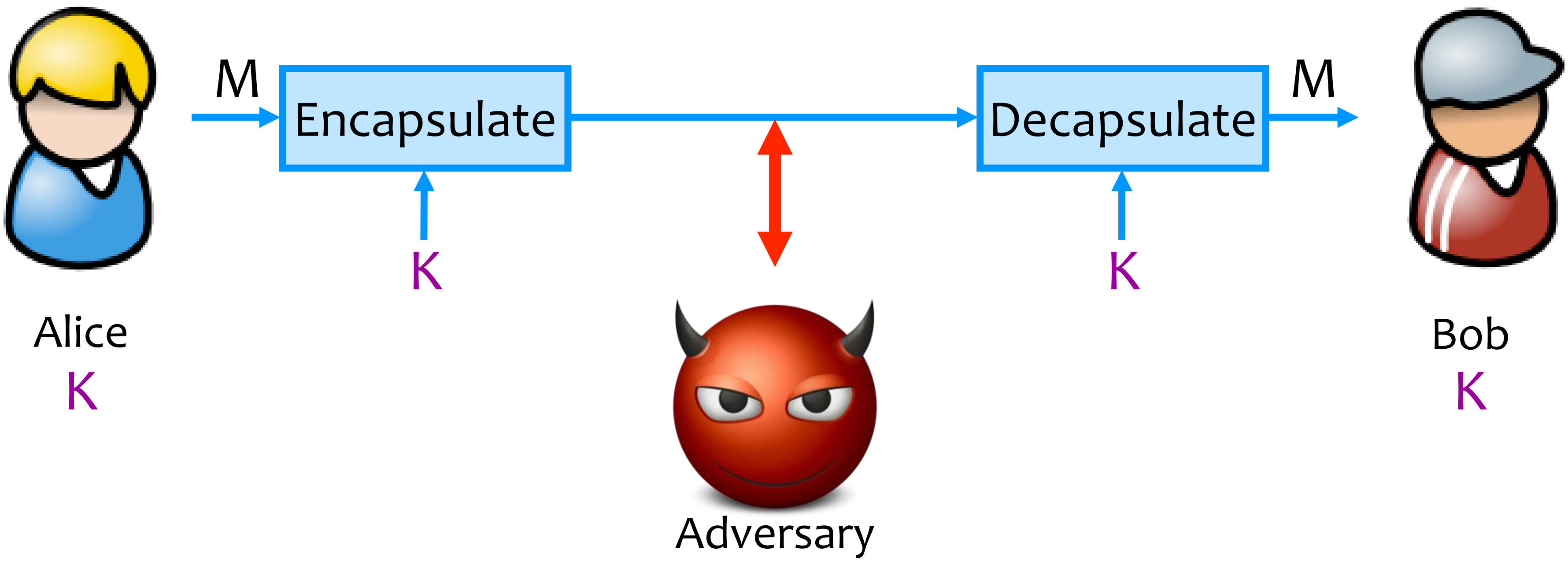
Stepping Back: Flavors of Cryptography

- 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.

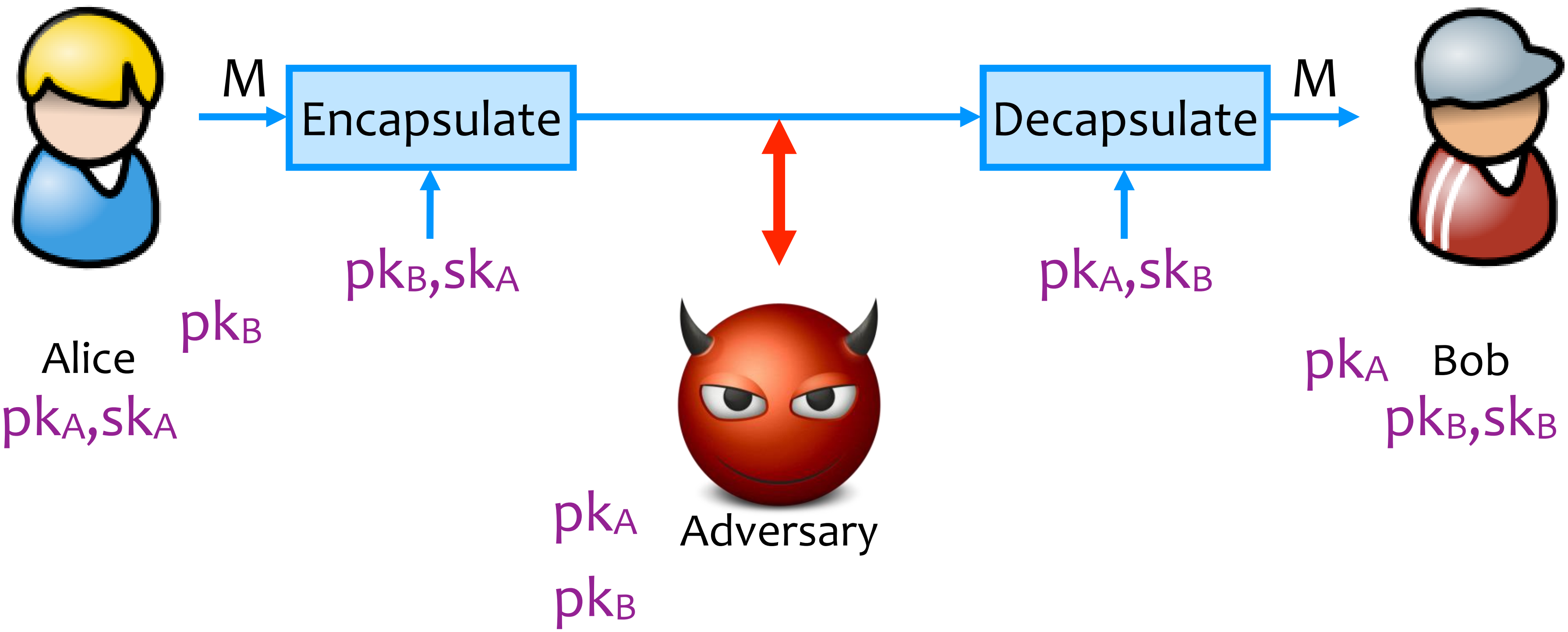
Symmetric Setting

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.



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.

Asymmetric (Public Key) Encryption

- Let's now look at an asymmetric building block: RSA
- Don't need to memorize details (for HW2, you can always look up details)
- Should try to understand "API-level" details (I'll clarify this as we go through slides)

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 $\varphi(ab) = \varphi(a)\,\varphi(b)$
- Main thing to "remember":
  - Easy to compute $\varphi(ab)$ if know a and b, for two primes a and b
  - Not known how to efficiently compute $\varphi(ab)$ if a and b unknown, for two primes a and b

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: $d \equiv e^{-1} \bmod \varphi(n)$

      How to compute?
  - Public key = (e,n); private key = (d,n)
- Encryption of m (m a number between 0 and n-1): $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e \bmod n)^d \bmod n = m$

Why Decryption Works (FYI)

- Decryption of c: $c^d \bmod n = (m^e \bmod n)^d \bmod n = (m^e)^d \bmod n = m$
- Recall $\mathbf{n}=pq$ and $\varphi(\mathbf{n})=(p-1)(q-1)$ and $ed \equiv 1 \bmod \varphi(n)$

- Chinese Remainer Theorem: To show $m^{ed} \bmod n \equiv m \bmod n$, sufficient to show:
  - $m^{ed} \bmod p \equiv m \bmod p$
  - $m^{ed} \bmod q \equiv m \bmod q$

- If $m \equiv 0 \bmod p \rightarrow m^{ed} \equiv 0 \bmod p$

- 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: $m^{(p-1)h}m \equiv 1^h m \bmod p \equiv m \bmod p$

Why is RSA "Secure"?

- RSA problem: given c, n=pq, and e such that gcd(e, φ(n))=1, find m such that $m^e=c \bmod n$
  - 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
- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$

- If factoring is easy, then RSA problem is easy (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

Caveats and Why RSA is "Insecure"

- Encrypted message needs to be an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
  - Recall ECB mode privacy concerns
- Plain RSA also does not provide integrity
  - Can tamper with encrypted messages
  - Suppose adversary sees two ciphertext $c1$ and $c2$, and then sends $c3 = c1 * c2 \bmod n$ to the recipient. What would that decrypt to?

How to Use RSA to Encrypt

- In practice, 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

- We will return to this after discussing hash functions

Some notes on modular arithmetic

- Can take modulus at any time in operation
- Try online tools, like https://www.wolframalpha.com/