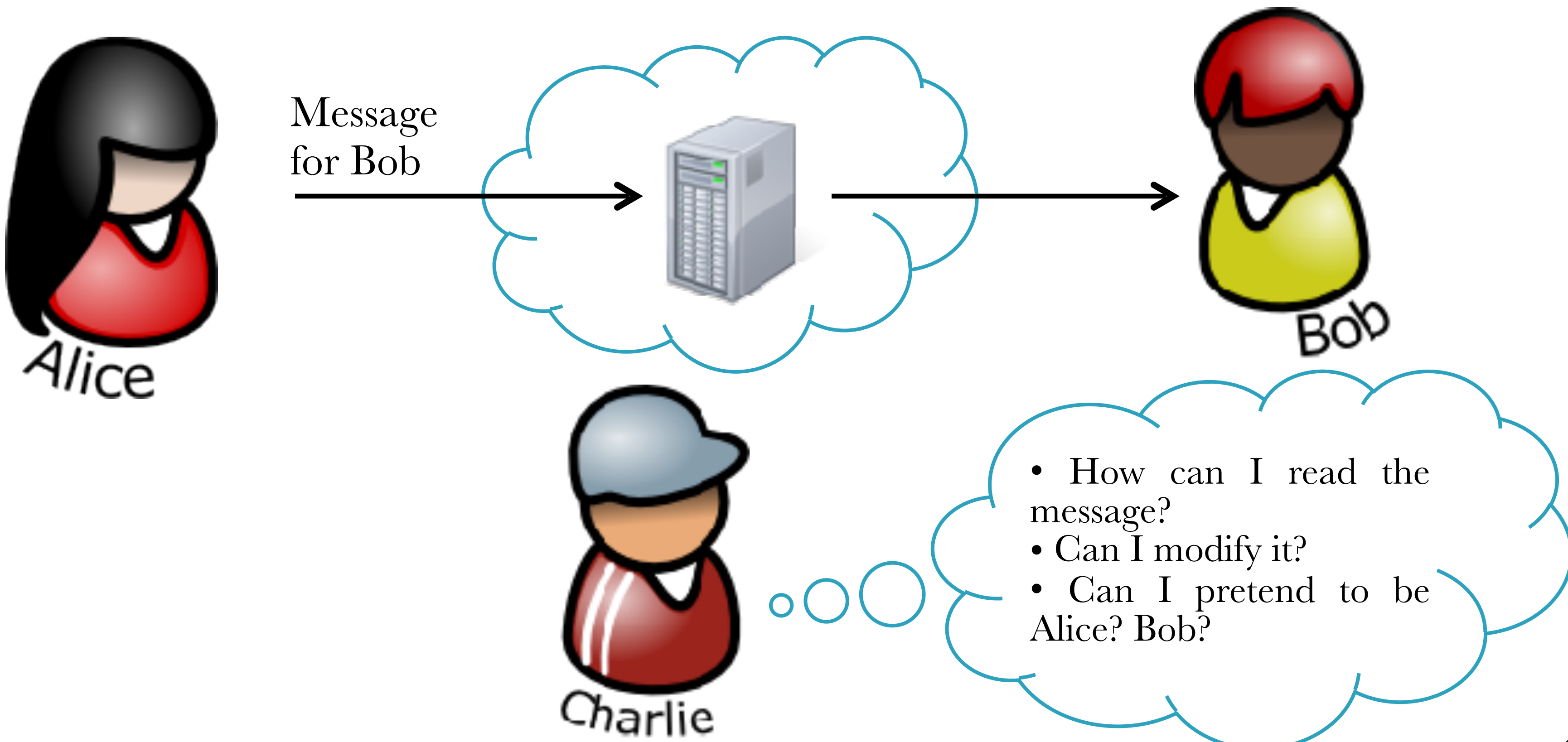
Overview of Cryptography

Brief intro to crypto

Cryptography: Introduction

- What is it: Cryptography is about protecting data



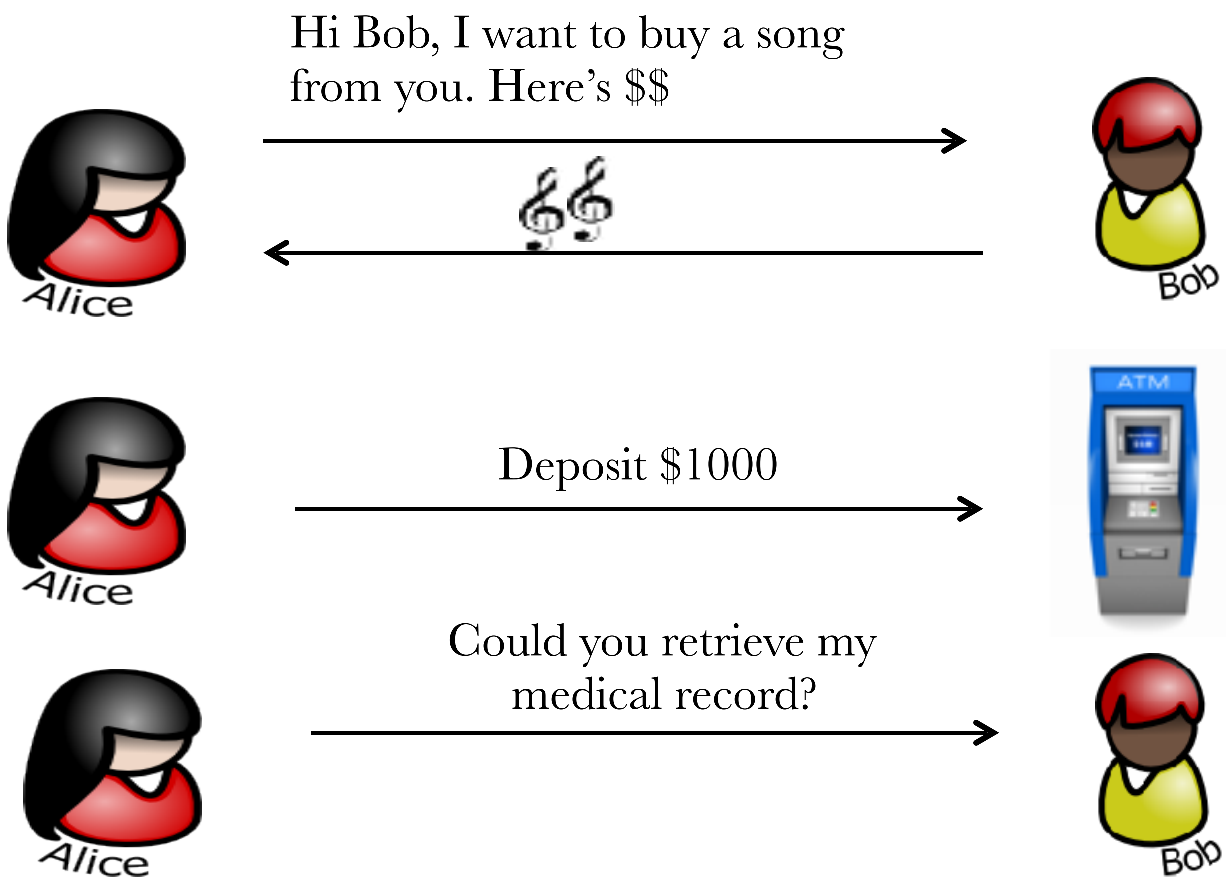
Who's Charlie? What does he want?

- Charlie is an "adversary"

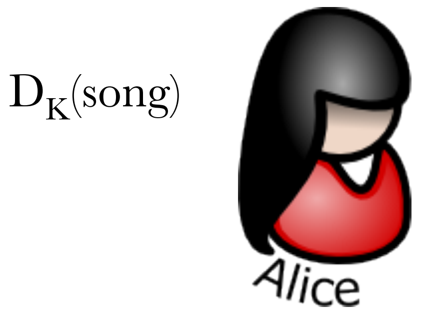
- Charlie's goals:
- Attack **Confidentiality** – Reading private message sent from Alice to Bob
- Attack **Integrity** – Modify private message
- Attack **Authenticity** – Impersonate Alice or Bob

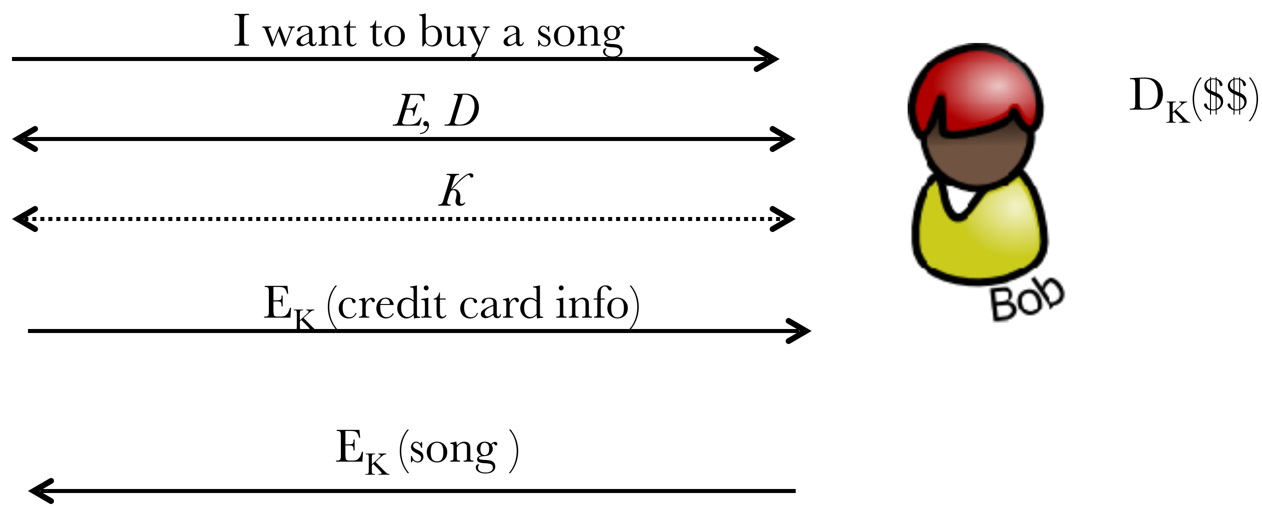
- "Protecting" = Stop Charlie!

Why Bother? Examples



Towards a Solution





- Selects encryption algorithm *E*
- Selects decryption algorithm *D*
- Generates key *K*

Some points:

1. If E_K and D_K are same, **symmetric** crypto
2. If E_K and D_K are different, e.g., AliceE_K, AliceD_K, and BobE_K, BobD_K, **public key** crypto

Goals of Cryptography

- Mainly ensure *CIA*: **C**onfidentiality, **I**ntegrity, **A**uthenticity of data
- Additional worthy goals: Non-repudiation, Fairness, Forward security…

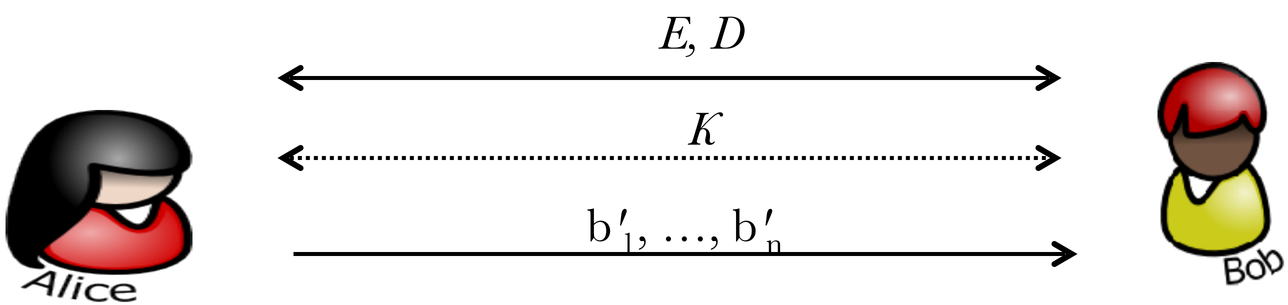
- Is it impossible to break crypto algorithms?
- Certainly not! Everything is breakable, given enough time and computing power.
- The point is: make it so that it takes too much time (non-polynomial)

Overview of Encryption

- Block ciphers: message divided into blocks
- Stream ciphers: continuous stream of bits

- <u>Block cipher</u>: Message divided into groups of bits – "blocks". Each block encrypted by same key

$M = b_1 || b_2 || \ldots || b_n$
$E_K(b_1) = b'_1$
$\ldots$
$E_K(b_n) = b'_n$

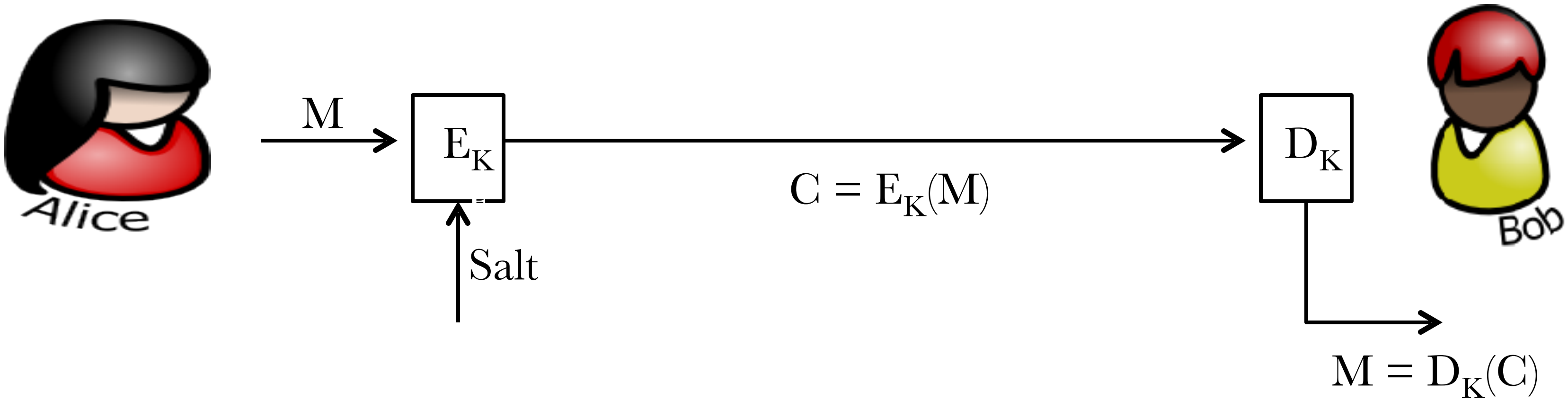


$D_K(x) = E_K^{-1}(x)!$

$D_K(b'_1) = b_1$
$\ldots$
$D_K(b'_n) = b_n$

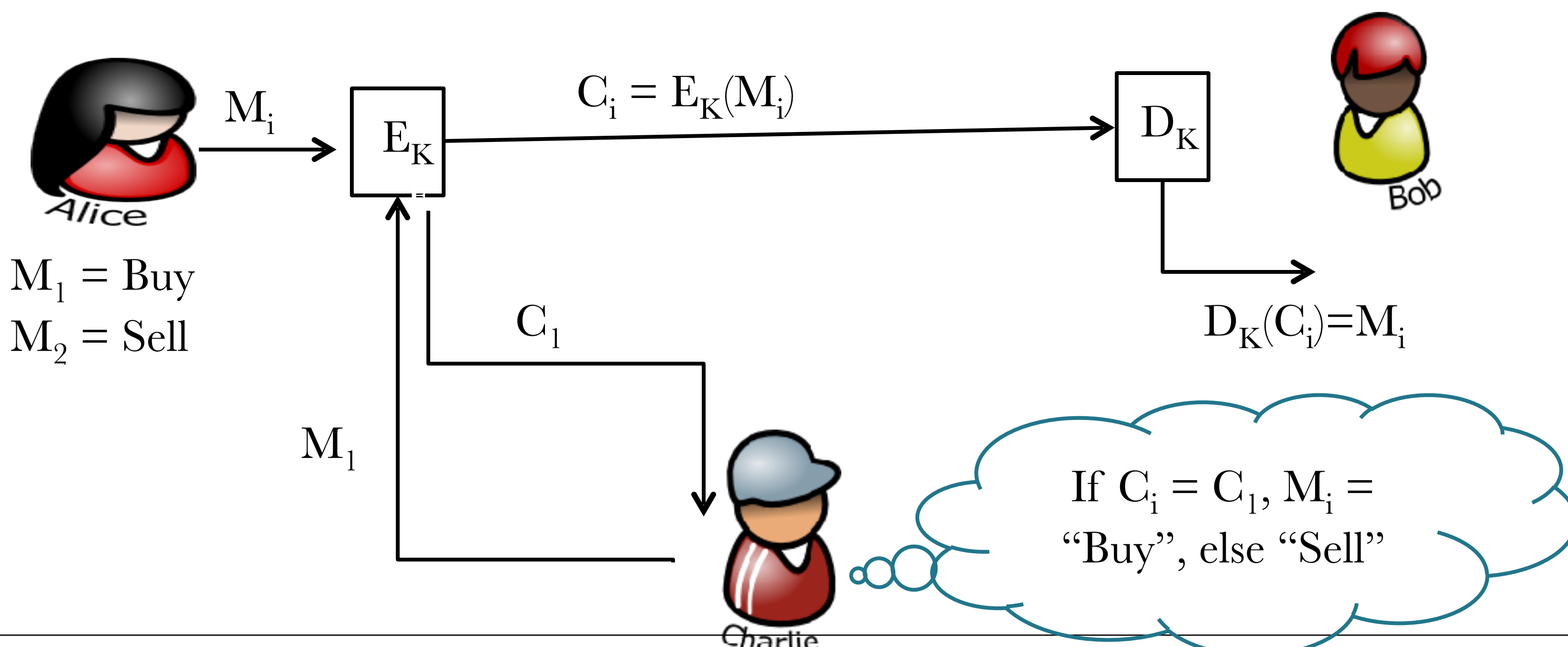
Symmetric Encryption

- Also called "shared-key encryption"
- Denoted by tuple (E,D,K)
- E and D known to all, K is secret
- Assume K is given to Alice and Bob, for now



Symmetric Encryption…cont'd

- Why salt/randomness?
- An encryption algorithm *must* be randomized
  - Why?
  - Encryptions of messages shouldn't be "predictable"
  - 2-option messages: Buy/Sell, Attack/Retreat, Pass/Fail, …



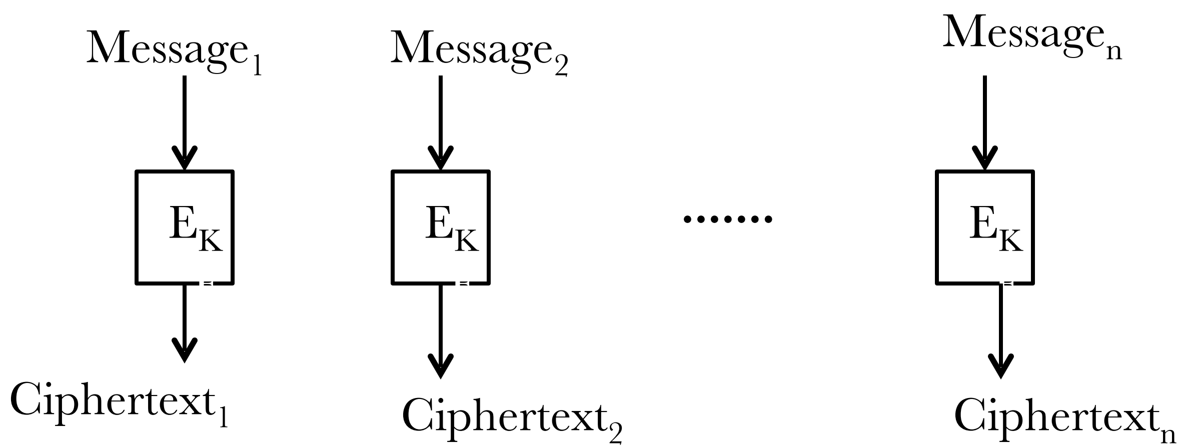
Why didn't the "Encryption" Work?

- It didn't work because:
- Every encryption of any given message comes out the same way
- Encryptions become guessable…

- Solution: randomize the plaintext
- Random bitstring (salt) added to each plaintext *before* it gets encrypted

Modes of Symmetric Encryption

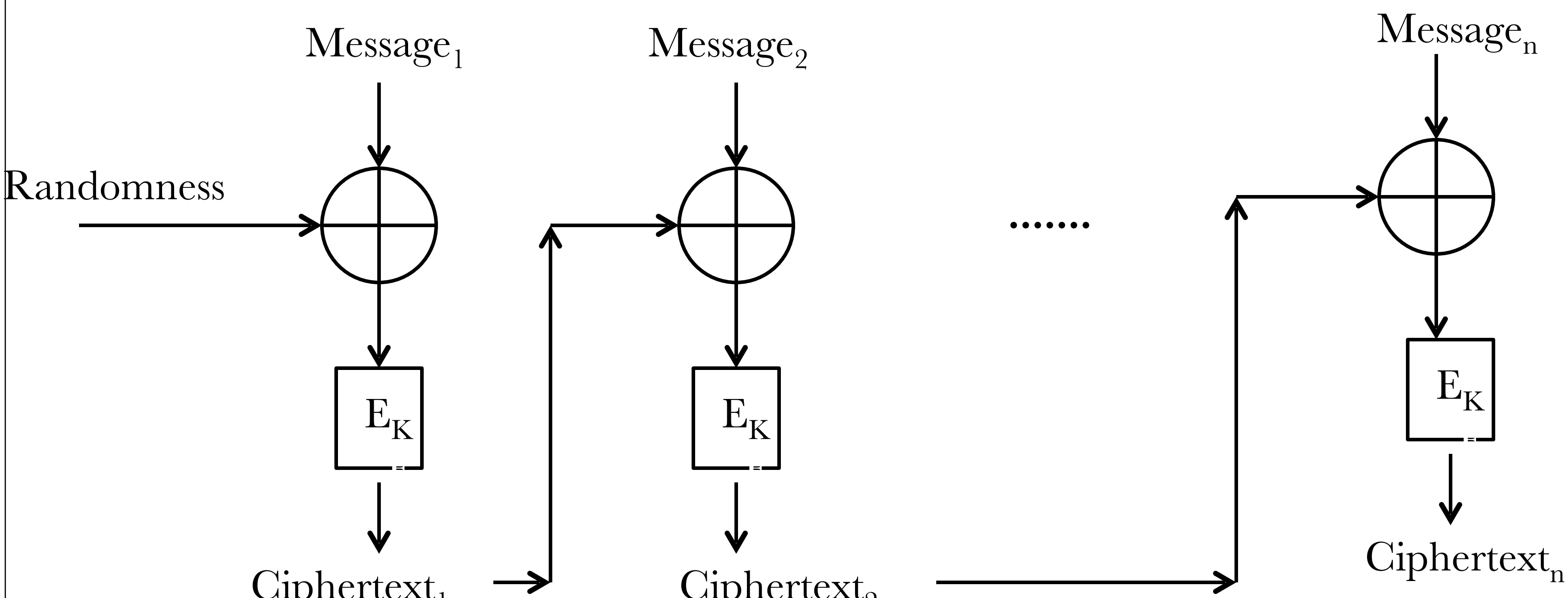
- Mode 1: Electronic Code Book (ECB)



- Can't be used for real-world stuff
- Main weakness: If Message$_1$ = Message$_2$, then Ciphertext$_1$ = Ciphertext$_2$

Mode 2: Cipher Block Chaining (CBC)

- Corrects weakness of Mode 1 (ECB)

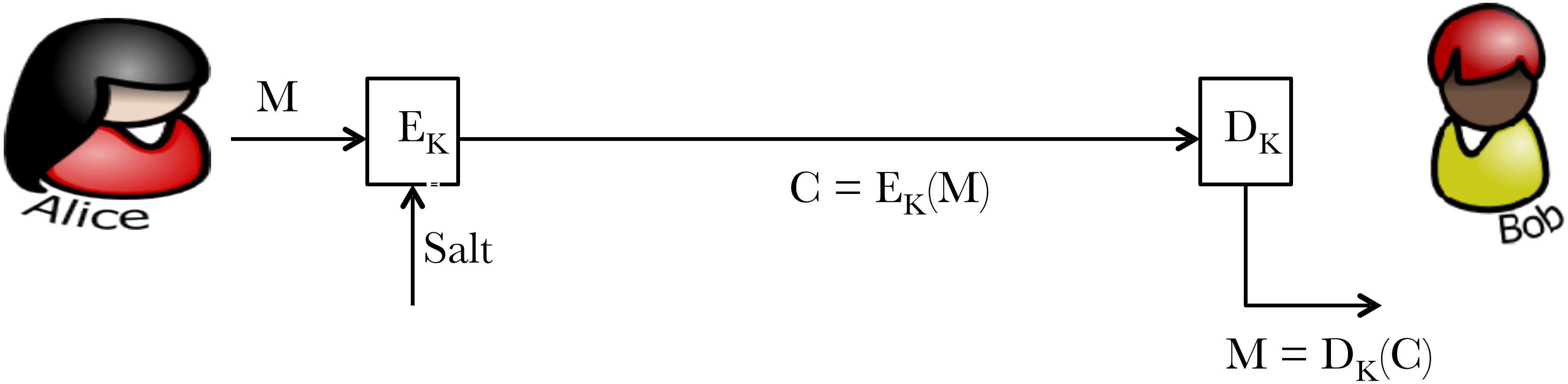


- Randomness -– "Initialization vector"

Curious?
Try `$openssl speed --help`

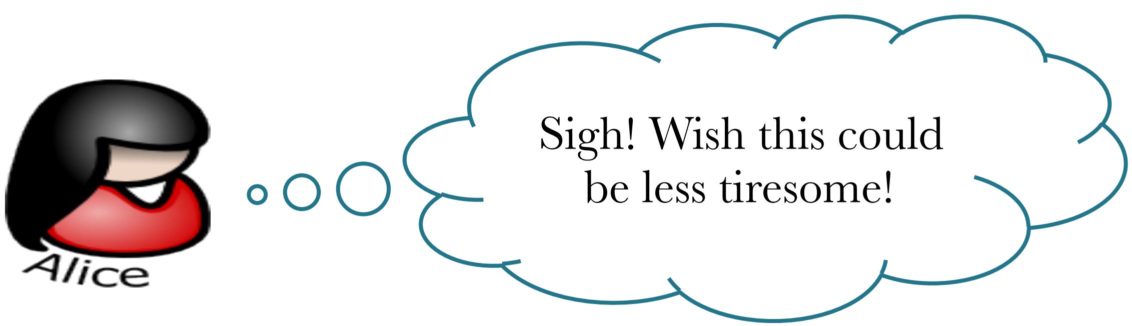
Public-key Cryptography

- Why do we need it?
- Lets go back to Alice and Bob
- But where does K come from? Magic?
- Alice and Bob need to setup a K *a-priori*



Public-key Cryptography

- Is setting up a K prior to encryption inconvenient?
- Yes! For multiple reasons
  - Alice and Bob need to have a secure connection to setup K
    - Need to avoid Eavesdropping Eve!
  - What if Alice and Charlie need to talk next?
    - Need to setup separate key

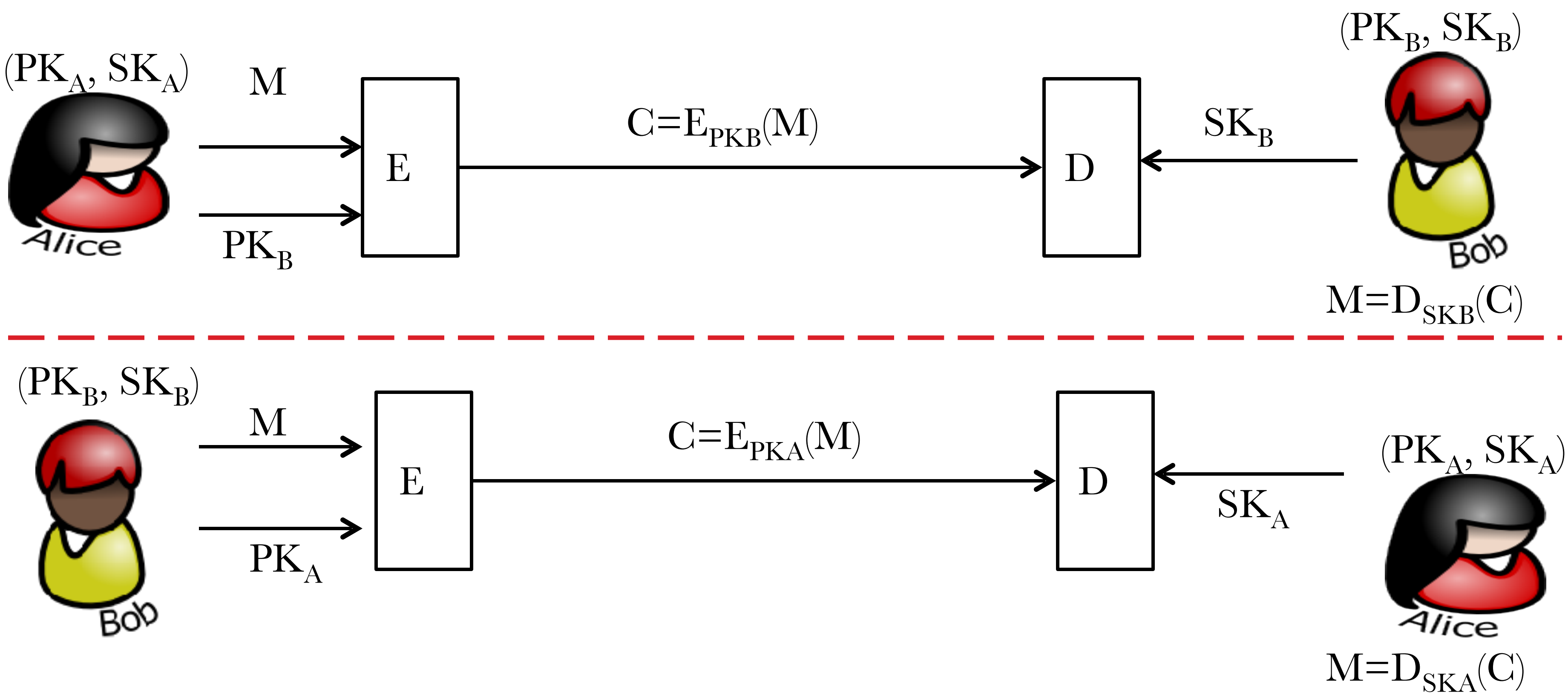


Public-key Cryptography

- Can we help Alice?
- Yes!

- Let Alice have 2 keys (PK_A,SK_A)
- PK_A – public, e.g., an e-mail address
- SK_A- secret, e.g., e-mail password
- No key setup
- No secure connection for setup required
- Similarly for Bob (PK_B, SK_B), Charlie (PK_C, SK_C),…

Public-key Cryptography



Public-key Cryptography

- But where do PKA and PKB come from?
- Is PKA really Alice's key? PKB Bob's key? Stolen keys? How can we be sure?

- Get public keys certified by a trusted authority
    - Certification authority
    - E.g., VeriSign (most popular), Entrust, …
    - Exercise for the curious: Who certifies NMSU's public key?

Cryptographic Strength

- Public-key crypto is based on mathematically hard problems: factorization, discrete-log
  - E.g., given primes p, q, easy to compute N=p•q
  - Given N, hard to find factors p, q – no *efficient* algorithm exists
- Alice computing (PK_A,SK_A), encrypting with PK_A – easy
- Attacker given PK_A, trying to compute SK_A – mathematically hard

- Symmetric key crypto based on series of substitutions, permutations – more latitude

Plaintext, Key → Juggle bits using key, until un-intelligible → Encrypted text

The Best of Both

- So, if PKC is all that great, do we even need symmetric encryption?
  - Yes, because public-key crypto is orders of magnitude slower than symmetric crypto
  - Convenient and secure, but inefficient

- Symmetric key crypto – very fast. Why? Bit-wise ops.

- Hybrid encryption
  - Use (PK,SK) to establish shared key K
  - Use K for everything thereafter

Key Lengths – Symmetric Crypto

- Only way of breaking symmetric key crypto – brute force the key
  - 10-bit key: 2^{10} possible values of key
  - 64-bit key (DES): 2^{64} possible values – (breakable)
- Legend: NSA reduced 64-bit to 56-bit in 1975 based on their then max. computing abilities!
- 128-bit key: 2^{128} combinations – secure for foreseeable future (AES, state-of-the-art)
- 256-bit key: 2^{256} combinations – secure for hundreds of years
- 512-bit key: 2^{512} combinations - until the sun freezes over and/or aliens take over the planet

Key Lengths – Public-key Crypto

- Public-key crypto: RSA (de-facto standard)
- Good key size in bits: 2048-recommended, 4096-if paranoid
- “Breaking” = factoring N =p•q; try out primes from [2..⌈N⌉]
- Broken till date: RSA-768 (N =768 bits, ~232 decimal digits)
- RSA factoring challenge – cash prizes for factoring large numbers (discontinued in 2007)

- But very easy to write bad implementations e.g., WEP
- Botched up implementations of good algorithms all too common

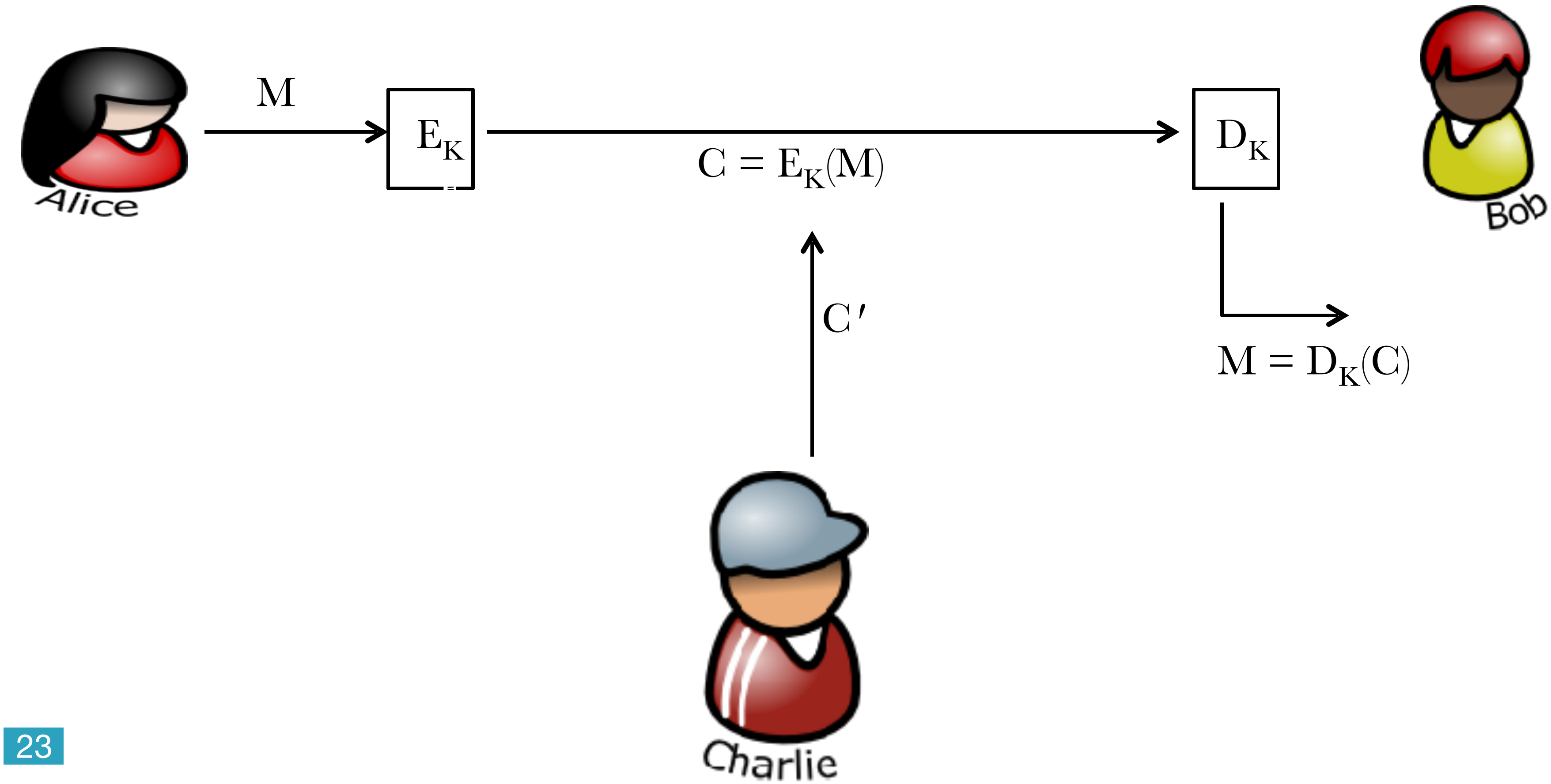
A Few more Primitives

- Encryption just provides confidentiality
- What about integrity, authenticity?
- Cost? Huge message = huge ciphertext ☹️

- More tools:
- Message authentication code (MAC)
- Hash functions
- Digital Signatures

Message Authentication Code (MAC)

- Provide (data) authenticity and integrity



Message Authentication Code (MAC)

- Provide authenticity and integrity

$C=E_K(M)$, Tag = MAC (K,M) →

↓

- M = DK(C)
- MAC (K,M) → Tag′
- Is Tag = Tag′?

- Error correction codes e.g., CRC, are a form of MAC

Hash Functions

- Provide message integrity
- Data Compression: Variable size input → Fixed size output

User name: Alice

~~Password: myPass~~ H(myPass) = 02d3af

User that wants to login to her account

Bank server

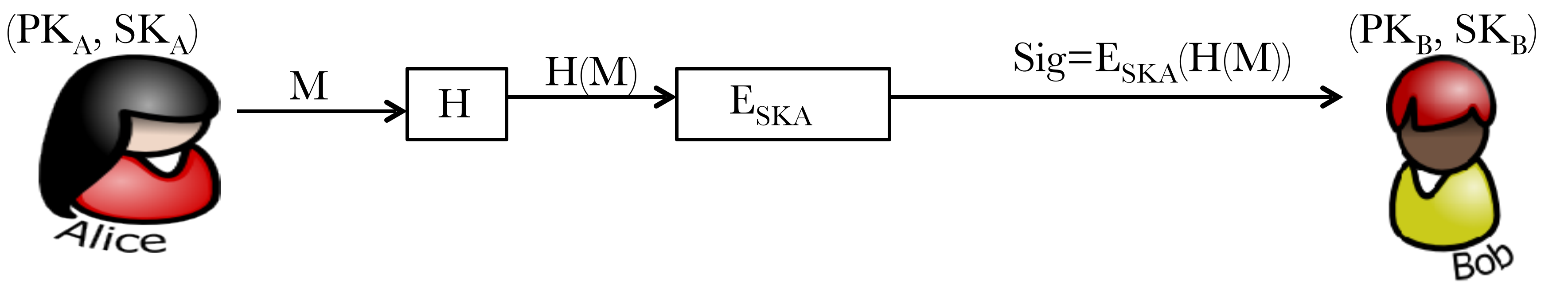
Is this safe: NO!
• What if Bob gets hacked into?
• What if Charlie intercepts Alice's password ?

Curious?
See /etc/shadow file on *nix systems!

Idea:
• Use a one-way "Hash" function
• H(password) = 02d3af (an unintelligible hexd. value)

Digital Signatures

- Based on public-key cryptography



- What is M? Known to both parties, e.g., a contract
- Why hash?
- Hashing prevents replay attacks
- Compactness

- Verify: D_{PKA}(Sig)) = X
- Bob knows M;
- Is H(M) = X ?
- If yes, signature is valid

Wrap-up and Recap

- Symmetric (shared key) crypto – efficient, but requires a shared secret to be setup
  - Modes of operations: ECB, CBC, …
  - E.g., DES, 3DES, AES
- Public-key crypto – easy to use, no shared secrets, but expensive (w.r.t. time)
  - E.g., RSA
- Encryption merely provides confidentiality
- Integrity, authenticity require MACs, Hash functions, Digital signatures
- Security metric – key length
  - Symmetric crypto – good length 128 bits and more
  - Public-key crypto – good length 2048 bits and more

Take-away Points

- Nothing is impossible to break, just that it takes ridiculously to do so
- More technically "can't be done in polynomial time"
- Randomness plays an important role in crypto!
- Good algorithms should hold up to serious scrutiny
  - All well-regarded algorithms are public
  - Security through obscurity doesn't work!