Fundamentals of Computer Security

Spring 2015

Radu Sion

Intro

Encryption

Hash Functions

A Message From Our Sponsors

- **Fundamentals**
  - System/Network Security, crypto
- **How** do things work
- **Why**
- How to **design** secure stuff

What we are not

- How to **install** XXX
- **Command line options** of XXX
- Latest **iexplorer buffer overflow bug**
- Latest McAfee/XXX **products**
- Network **administration**
- How to break your gf/bf email account

Ground Rules

- Dates are listed online now
- Zero tolerance to academic dishonesty
- Informal class, ask questions anytime
- **Read your assigned readings !**
- There may be quizzes
- Call me Radu
- Questions: office hours, or email to schedule appt.
- Email: sion@cs
- Have fun !

Evaluation

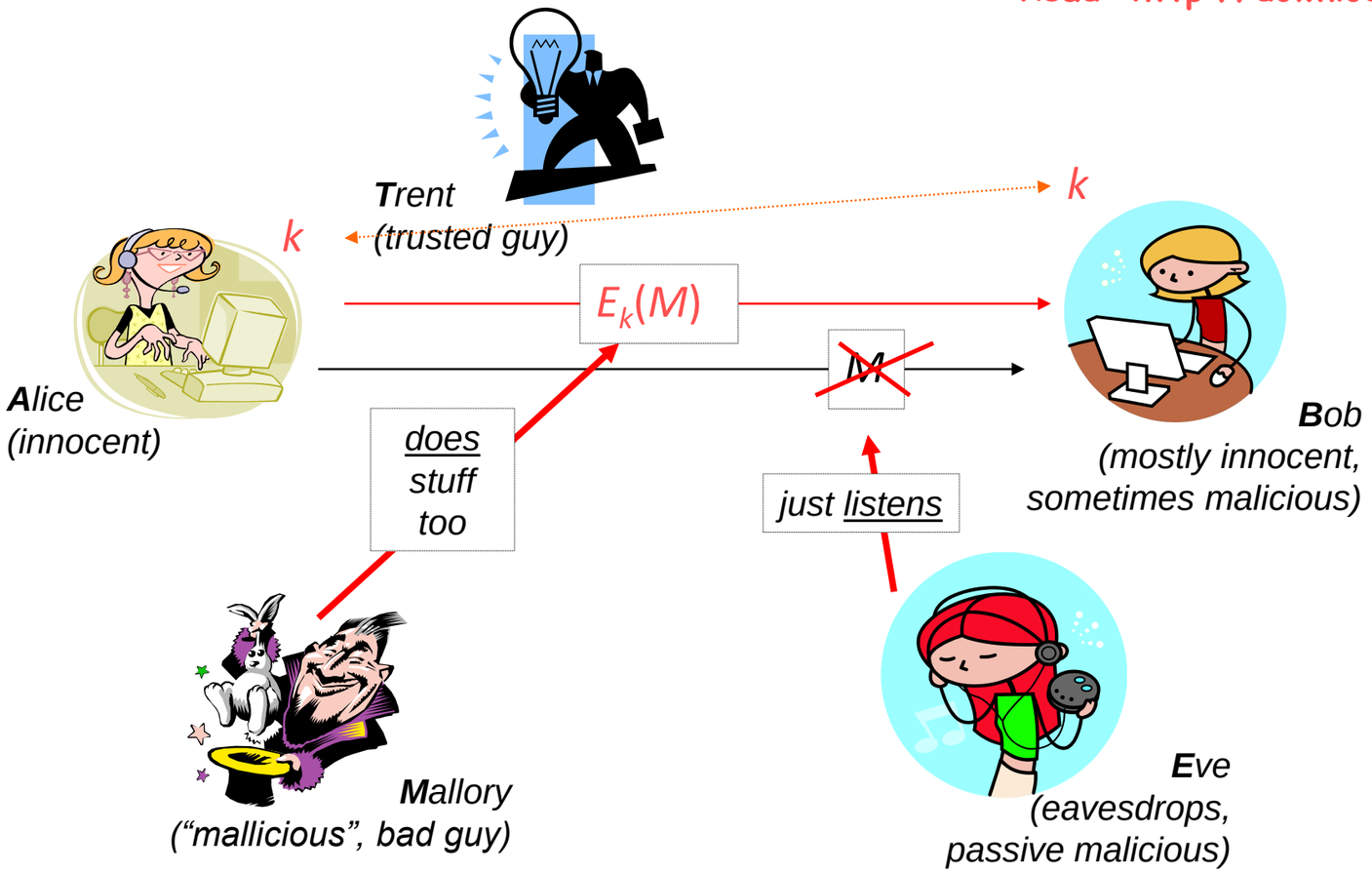
- Homeworks (0-10%)
- Midterm (30-40%)
- Activity and pop quizzes (0-10%)
- Final (40-50%)
- Course website: **check link in your email**

Classical Crypto

- Single/Symmetric Key Encryption
- Cryptographic Hash Functions

Meet the Cast

Read: http://downlode.org/etext/alicebob.html !



An inconvenient truth

- Where does *k* come from ? (“key distribution”)
- Can Eve distinguish between $E_k(M_1)$ and $E_k(M_2)$ if she knows M_1 and M_2 ? Should not be able to !!! (“semantic security”)
- Make sure that $E_k(M_1) \neq E_k(M_2)$ if $M_1 \neq M_2$ (maybe not ?)
- Can Mallory modify $E_k(M)$ into an $E_k(M_{mallory})$? (”malleability”)
- etc (! lots of stuff !)
- Danger: things seem trivial and they are not – result: super weak systems !

Caesar Cipher

- Example: Cæsar cipher
  - $\mathcal{M}$ = { sequences of letters }
  - $\mathcal{K}$ = { i | i is an integer and $0 \le i \le 25$ }
  - $\mathcal{E}$ = { E_k | $k \in \mathcal{K}$ and for all letters m,
    $E_k(m) = (m + k) \bmod 26$ }
  - $\mathcal{D}$ = { D_k | $k \in \mathcal{K}$ and for all letters c,
    $D_k(c) = (26 + c - k) \bmod 26$ }
  - $C = \mathcal{M}$

Attacks

- Opponent whose goal is to break cryptosystem is the *adversary*
  - Assume adversary knows algorithm used, but not key
- Many types of attacks:
  - *ciphertext only*: adversary has only ciphertext; goal is to find plaintext, possibly key
  - *known plaintext*: adversary has ciphertext, corresponding plaintext; goal is to find key
  - *chosen plaintext*: adversary may supply plaintext and obtain corresponding ciphertext; goal is to find key
  - *chosen ciphertext*: adversary may supply ciphertext and obtain corresponding plaintext; goal is to find key
  - etc

How to attack?

- Mathematical attacks
  - Based on analysis of underlying mathematics
- Statistical attacks
  - Make assumptions about the distribution of letters, pairs of letters (digrams), triplets of letters (trigrams), *etc.*
    - Called *models of the language*
  - Examine ciphertext, correlate properties with the assumptions.

Statistics

- Compute frequency of each letter in ciphertext:

  G 0.1 H 0.1 K 0.1 O 0.3

  R 0.2 U 0.1 Z 0.1

- Apply 1-gram model of English
- Correlate and invert encryption

Caesar has a Problem ☺

- Key is too short
  - Can be found by exhaustive search
  - Statistical frequencies not concealed well
    - They look too much like regular English letters
- So make it longer
  - Multiple letters in key
  - Idea is to smooth the statistical frequencies to make cryptanalysis harder

Vigènere Cipher

- Like Cæsar cipher, but use a phrase
- Documented by Blaise de Vigenere (court of Henry III of France) in Paris, 1586 – actually a variant of a cipher by a J.B. Porter
- Example
  - Message `THE BOY HAS THE BALL`
  - Key `VIG`
  - Encipher using Cæsar cipher for each letter:

```
key    VIGVIGVIGVIGVIGV
plain  THEBOYHASTHEBALL
cipher OPKWWECIYOPKWIRG
```

Holy Grail: One-Time Pad

- A Vigenère cipher with a random key at least as long as the message
  - Provably unbreakable
  - Why? Look at ciphertext `DXQR`. Equally likely to correspond to plaintext `DOIT` (key `AJIY`) and to plaintext `DONT` (key `AJDY`) and any other 4 letters
  - Warning: keys *must* be random, or you can attack the cipher by trying to regenerate the key
    - Approximations, such as using pseudorandom number generators to generate keys, are *not* random

Crypto Hash Functions

- Mathematical function to generate a set of *k* bits from a set of *n* bits (where $k \leq n$).
  - *k* is usually smaller then *n*
- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is “parity”
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits

Sample Use

- Bob receives “10111101” as bits.
  - Sender is using even parity; 6 1 bits, so character was received correctly
    - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
  - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Definition

Cryptographic hash h: $A \rightarrow B$:

1. For any $x \in A$, $h(x)$ is **easy to compute**
2. *h(x)* is of fixed length for any x (**compression**)
3. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$. (**pre-image resistance**)
4. It is computationally infeasible to find <u>any</u> two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$ (**collision resistance**)
5. Alternate form of 3 (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$. (**second pre-image resistance**)

Collisions

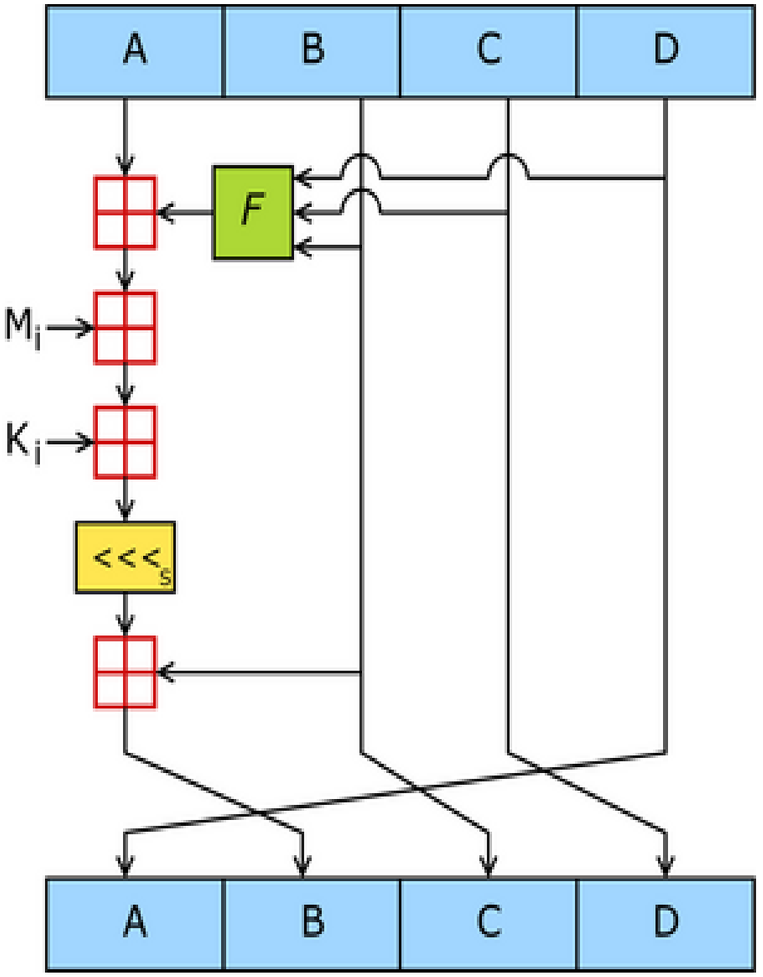
- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
  - Pigeonhole principle: if there are n containers for n+1 objects, then at least one container will have 2 objects in it.
  - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files

Intuition

- A hash is a **one-way, non-invertible** function of that produces **unique** (with *high likely-hood*), **fixed-size** outputs for different inputs.
- The probability of any bit flipping in the output bit-string should be always ½ for any change (even one bit) in the input (“randomness”).

Sample Cipher: MD5

- Basic idea: Continuously update hash value with 512 bit blocks of message
  - 128 bit initial value for hash
  - Bit operations to “compress”
- Compression function: Update 128 bit hash with 512 bit block
  - Pass 1: Based on bits in first word, select bits in second or third word
  - Pass 2: Repeat, selecting based on last word
  - Pass 3: xor bits in words
  - Pass 4: y ⊕ (x or ~z)



Example MD5 Digest

md5_digest("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6

md5_digest("The quick brown fox jumps over the lazy cog") =
1055d3e698d289f2af8663725127bd4b

Hashes to (not) use

- Do not use at all the following:
  - **MD5, SHA-0/1, any other obscure "secret" ones**
- For use in civilian/.com setting (until 2025):
  - **SHA-256/512, SHA3**

Cool Application: Keyed Hashes

Message Authentication Code (MAC)

- MAC(msg)=H(H(key,msg,key),msg)
- Usage: append this to message to allow authentication

Purpose of MACs?

- Want to enable only a certain party to verify authenticity of data for which it has a MAC (for example).
- Want to prevent Mallory to alter message and simply replace MAC (cannot do it now – doesn't know the secret key)

Optional for next week

For **+5% credit** in final exam.

Install openssl and decrypt **any** of the following ciphertexts:

```
U2FsdGVkX18Avp0s9oaA8I2HeaLoCG1gZyRmoLWWBFZXcrm/1ZsXSjxc2XTpbPZw

U2FsdGVkX18KRUFApfRXdayMo8sYd96zEAdPXyA4hzMBdWxqVigJGsLs4okBhwje

U2FsdGVkX1/DUTj3FPMhUWb/hgxIchBN6LWoRbLm2L/CARN/VSAYlg==

U2FsdGVkX1/+vE2czERZciAIJteLkzndHwW9QrdibZ/Z6q8=
```