Password Security

CSCE 499 Capstone Presentation

Daniel Case: BA Computer Science/BS Mathematics
Faculty Adviser: Dr. George Hauser

Overview of Presentation

- Introduction (context of research)
- Cryptographic Hash Functions
  - Properties
  - Design
  - Example: SHA-1
- Attacks:
  - Dictionary
  - Brute Force
  - Rainbow Table
- Conclusion

Password Security: Introduction

- Passwords are used for authentication/authorization
- Guarding sensitive information by restricting access

Password Security: Introduction

- Passwords are defined as *plaintext*
- Plaintext is unaltered, *unencrypted* text
- In good security schemes, passwords are *never* stored as plaintext
    - Stored elsewhere as a *hash value*

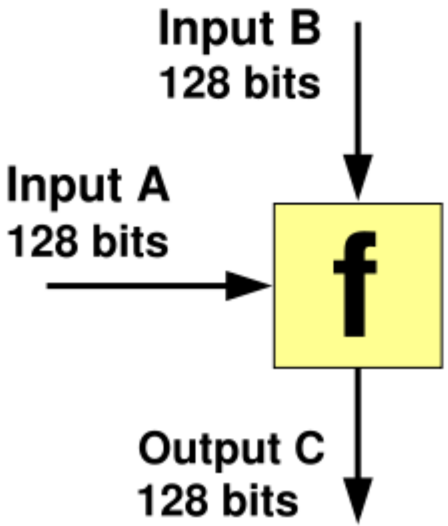
Cryptographic Hash Function

- Algorithm that maps strings of *any length* to strings of *fixed length*

- Several properties:
  - *Pre-image resistance*: Given a hash *h* it is infeasible to find an input *m such that h* = *hash*(m)
  - *Second preimage resistance*: Given an input $m1$, it is infeasible to find an $m2$ such that $m1 \mathrel{!=} m2$ and $hash(m1) = hash(m2)$
  - *Collision Resistance*: It is infeasible to find $m1$, $m2$ such that $hash(m1) = hash(m2)$

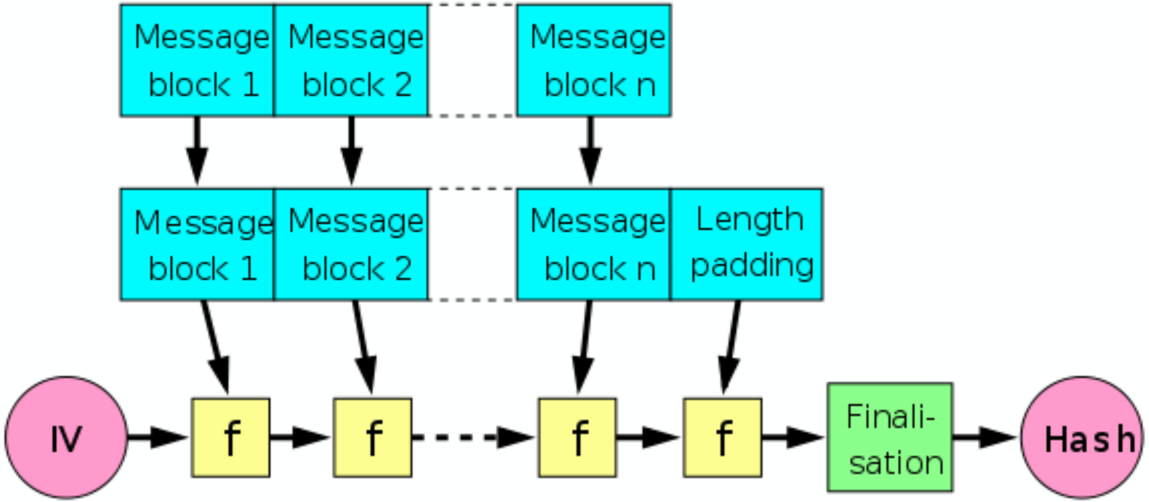
Merkle-Damgard Construction

- Uses a *Compression function*
- Transforms two fixed length inputs into a fixed length output



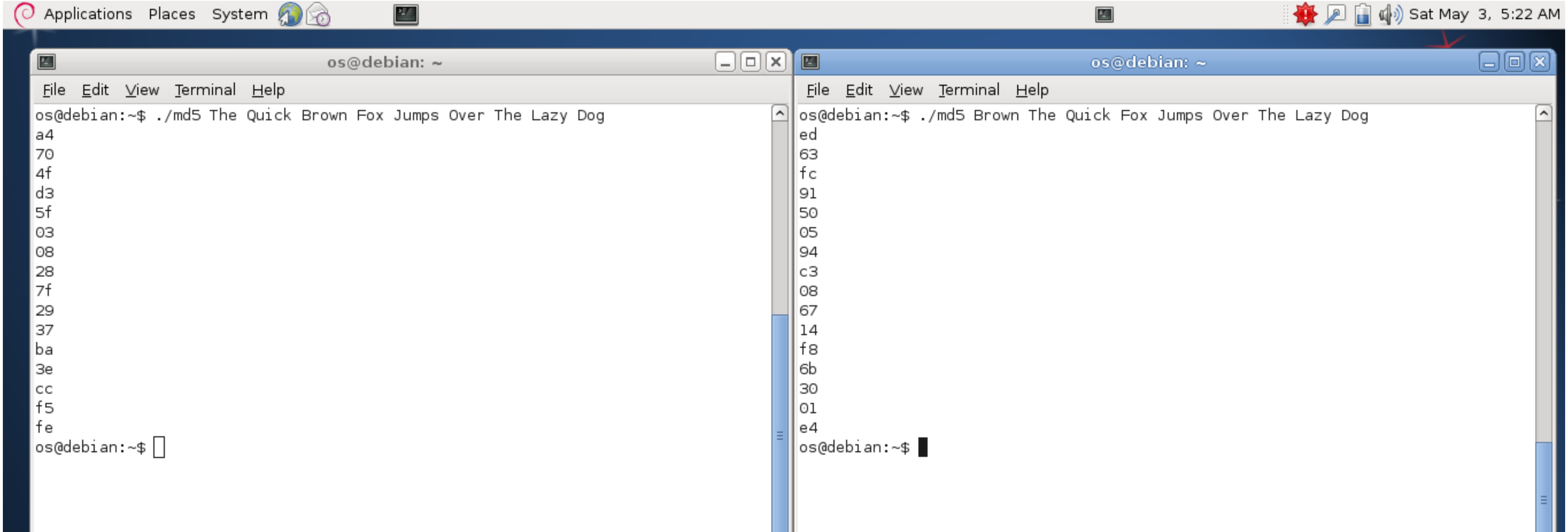
Merkle-Damgard Construction

- Compression function is a for-loop



Avalanche Effect

- Every change in input drastically alters the next output



SHA-1 Algorithm

- Pseudocode Example

Attacking Passwords

- Focus of research on three common password cracking techniques:
    - Dictionary Attack
    - Brute-Force Attack
    - Rainbow Table

- Assumptions:
    - Have access to list of password hashes
    - “Offline” attacks
    - Need to know which hash function is used

Dictionary Attack

- *Very Large text file* with common passwords
- Each entry is hashed and then compared

Dictionary Attack

- *Not guaranteed to work*
- All depends on the text file
- Shortcomings usually avoided by using two or more text files

Brute-Force Attack

- Uses a pre-defined charset or *alphabet* and a string size as parameters
- Sequentially generates and hashes characters/combinations
- Compares each to the target hash value

Brute-Force Attack

- Pros: It is *guaranteed* to work
- Cons: It can take a *very* long time

- Exponential time: will at best search through 2^*n*\2 combinations, where *n* is the size of the output hash

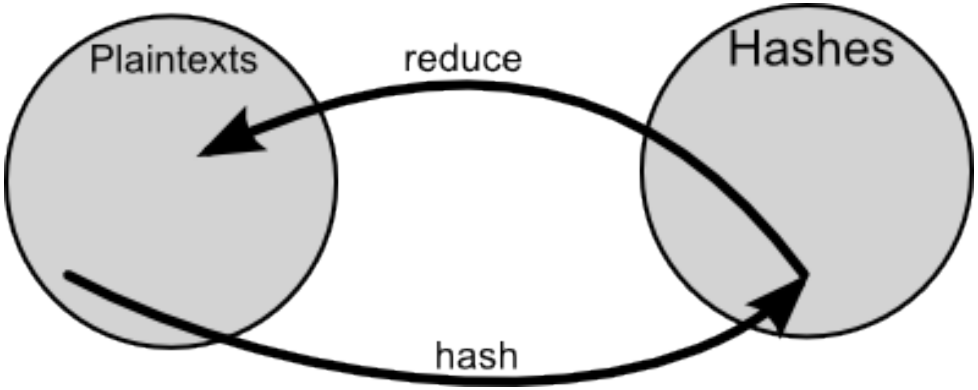
Rainbow Table

- Time/Space trade-off
- Large table of precomputed hash values
- Constructed with a large *input file*

- Input file is either
  - Dictionary based
  - Built ground-up from a character set (alphabet)

- Rainbow Tables *always* based on chosen hashing algorithm

Reduction Functions



- Reduction Functions map *hashes* to *plaintext* based on the *alphabet of the input file*
- Forms a Hash Chain

Hash Chains

- Hash chains start with plaintext from the input file
- Starting column is plaintext, next column is the previous entry hashed row by row

p1	**h1=H(p1)**
1	**1318**
2	**2636**
3	**3955**

Hash Chains

- Build the hash chains across the columns via Reduction Functions

p1	h1=H(p1)	p2=R1(h1)	h2=H(p2)	p3=R2(h2)	h3=H(p3)
1	1318	18	5191	19	0329
2	2636	26	3378	37	5520
3	3955	35	2884	88	4779

R2 = Middle two digits, R1 = Outer two digits

Properties of Rainbow Tables

- Hash chains are calculated - not all are stored
- Table size < number of lookups and calculations
- Rainbow Table stores only the first column of plaintext and the last column of the hash chain

p1	h3
1	0329
2	5520
3	4779

Rainbow Table Algorithm

1) Search for hashed value in Rainbow Table

    If found: goto Step 2. If not:

A) Starting with the last reduction function, *reduce* the hashed value to gain a new plaintext. (When Step 1 is repeated, use next lowest reduction function)

B) Hash new plaintext and repeat Step 1 with new hash value

2) Take the corresponding plaintext value in the RT and hash it

3) Compare the target hash

    If they match - then stop

    If not: goto Step 4

4) Apply the current reduction function, get plaintext, goto Step 2

Worked Example

- Try with a target hash: 2884

p1	h1=H(p1)	p2=R1(h1)	h2=H(p2)	p3=R2(h2)	h3=H(p3)
1	1318	18	5191	19	0329
2	2636	26	3378	37	5520
3	3955	35	2884	88	4779

Advantages and Disadvantages

- Very powerful, and very fast
- Able to be configured for any hash function

- Salts make the computation expensive
- Large tables: upwards of 300 GB for “smaller” tables

Concluding Thoughts

MD5/SHA-1 code courtesy of stackoverflow.com and wikipedia.com

Rainbow Table Algorithm and example explanation courtesy of stitchintime.wordpress.com

Reduction function image from kestas.kuliukas.com

All other images courtesy of google.com and wikipedia.com

Questions?

Comments?