Crypto tricks:
Proof of work, Hash chaining

CS 161: Computer Security
Prof. David Wagner

April 13, 2016
A Tangent: How Can I Prove I Am Rich?
Math Puzzle – Proof of Work

• **Problem.** To prove to Bob I’m not a spammer, Bob wants me to do 10 seconds of computation before I can send him an email. How can I prove to Bob that I wasted 10 seconds of CPU time, in a way that he can verify in milliseconds?
Math Puzzle – Proof of Work

• **Problem.** To prove to Bob I’m not a spammer, Bob wants me to do 10 seconds of computation before I can send him an email. How can I prove to Bob that I wasted 10 seconds of CPU time, in a way that he can verify in milliseconds?

• **Hint:** Computing 1 billion SHA256 hashes might take 10 seconds.
Your Solution

- Bob provides a random challenge $r$
- I compute: find $x$ such that $H(r,x)$ starts with 33 0 bits
  - This will take me $2^{33}$ hash computations, on average
  - Geometric: coin flip, with $1 / 2^{33}$ chance of heads
- Bob verifies by: checking that $H(r,x)$ starts with 33 0 bits
- Problem: replay attacks
Your Solution

- Bob picks 50-bit primes $p,q$, sends me $n = pq$
- I have to factor $n$, send back $p$ and $q$
- Bob can verify by multiply $p \times q$
Solution

• To prove that I wasted 10 seconds of CPU time, in a way that he can verify quickly:

• Bob sends me: $r$
• I look for $x$ such that first30bits(SHA256($x || r$)) = 0
• I send Bob: $x$
• Bob can verify using a single hash.
Tamper-Evident Logging

• We work for the police Electronic Records office.
• To ensure that evidence can’t be questioned in court, we want to make sure that evidence can’t be tampered with, after it is logged with the office.
• In other words: a police officer can log an electronic file at any time; after it is logged, no back-dating or after-the-fact changes to evidence should be possible.
• How should we do it? What crypto or data structures could we use?
Design Problem for You

- Idea: Each day, collect all the files \((f_1, f_2, \ldots, f_n)\) that are logged that day. Then, publish something in the next day’s newspaper, to commit to these files.
- Question: What should we publish? Needs to be short, and ensure after-the-fact changes or backdating are detectable.
- When a file \(f_i\) is exhibited into evidence in a trial, how can judge verify it hasn’t been modified post-facto?
Your Solution

• Store in database: f1, .., fn
• Publish: H(f1), H(f2), .., H(fn)
• To verify $f_i$: reveal fi
Your Solution

• Store in database: $f_1, \ldots, f_n$
• Publish: $H(H(f_1), H(f_2), \ldots, H(f_n))$
• To verify $f_i$: reveal $f_i, H(f_1), H(f_2), \ldots, H(f_n)$
Your Solution

- Store in database: $f_1, \ldots, f_n$
- Publish: $\text{Sign}(f_1), \text{Sign}(f_2), \ldots, \text{Sign}(f_n)$, signed under judge’s key
- To verify $f_i$: reveal $f_i$
Candidate Solution

- Store in database: $f_1, \text{Sign}(f_1), f_2, \text{Sign}(f_2), \ldots, f_n, \text{Sign}(f_n)$
- Publish: public key
- To verify $f_i$: reveal $f_1, \text{Sign}(f_i)$

- Critique: Sysadmin can get a copy of the private key, modify database, update the signature, and thus modify old entries or create new backdated entries.
Candidate Solution

• Publish: $H(f_1, f_2, \ldots, f_n)$
• To verify $f_i$: reveal $f_1, f_2, \ldots, f_n$
Solution

• Each day, collect all the files \((f_1, f_2, \ldots, f_n)\) that are logged that day. Then, publish \(H(f_1, f_2, \ldots, f_n)\) in the next day’s newspaper, to commit to these files.

• When a file \(f_i\) is exhibited into evidence in a trial, reveal \(f_1, f_2, \ldots, f_n\) to judge. Judge can hash them, check that their hash was in the right day’s newspaper, and check that \(f_i\) is in the list.
Better Solution

• Each day, collect all the files \((f_1, f_2, \ldots, f_n)\) that are logged that day. Let \(f_0\) be the previous day’s hash. Publish \(H(f_0, f_1, f_2, \ldots, f_n)\) in the next day’s newspaper, to commit to these files.

• Note that exhibiting file \(f_i\) into evidence still requires revealing entire list of other files \((f_1, f_2, \ldots, f_n)\) that were logged that day. Can you think of any way to avoid that?
Tamper-evident Audit Logs

• $X_1 = H(X_0, \text{“opened vault”})$
• $X_2 = H(X_1, \text{“disabled alarm”})$
• $X_3 = H(X_2, \text{“closed alarm”})$
• $X_4 = H(X_3, \text{“front door locked”})$
• $X_5 = H(X_4, \text{“closed vault”})$

• Publishing any $X_i$ commits to all prior log entries.
Take-away

- Using hash chaining, we can provide tamper-evident audit logs that let us detect after-the-fact modifications and backdated entries.
Distributed Logging

• Let’s do distributed peer-to-peer logging of public data. We have $n$ computers; they all know each others’ public keys. Any computer can broadcast to all others (instantaneously, reliably). Any computer should be able to append a signed entry to the log, and to verify integrity of any previous log entry.

• Security goal: Malicious computers should not be able to back-date entries or modify past log entries. Assume $\leq 3$ computers are malicious.

• Problem 1. Describe a protocol for this. What does Alice do to append an entry? What do other computers need to do?
Your Solution

• To append log entry e:
• Other computers should:
Distributed Logging

• **Problem 2.** Let’s generalize. Suppose $m$ of the $n$ computers are malicious. If we make the obvious change to your protocol, for which $m$ can it be made secure?

  • (a): for all $m < n$.
  • (b): for all $m < n/2$.
  • (c): for all $m < n/3$.
  • (d): for all $m < \sqrt{n}$.
  • (e): for all $m < O(\lg n)$.
Problem 2. Let’s generalize. Suppose $m$ of the $n$ computers are malicious. If we make the obvious change to your protocol, for which $m$ can it be made secure?

- (a): for all $m < n$.
- (b): for all $m < n/2$.
- (c): for all $m < n/3$.
- (d): for all $m < \sqrt{n}$.
- (e): for all $m < O(lg n)$. 
Distributed Money

• Donna gets the brilliant idea to use this log to store financial transactions. Each person’s initial balance is public.

• To transfer $10 from Alice to Bob, Alice appends a signed log entry saying “I transfer $10 to Bob” and broadcasts it. Everyone can compute the updated balance for Alice and Bob.

• **Problem 3.** What are some ways that a malicious actor might try to attack this scheme? Is this a good scheme?
Your Answers

- Replay
- Denial of service attacks
- Broadcast doesn’t scale
- TOCTTOU vulnerability
Problems with This Scheme

- Initial balance is arbitrary
- Broadcasting is expensive and doesn’t scale
- A conspiracy of $n/2$ malicious computers can fork the audit log and steal all the money
- Sybil attacks: Anyone can set up millions of servers and thus have a 50% majority
Bitcoin

• Public, distributed, peer-to-peer audit log of all transactions.
• To append an entry to the log, the latest value must hash to something whose first 30 bits are zero; then broadcast it to everyone.
• Anyone who appends an entry to the log is given a small reward, in new money (a fraction of a Bitcoin).