Session 9: Security

Dr John Fawcett

Rock-Paper-Scissors

Rules of the game

- Two players
- Each picks one of rock, paper or scissors
- Tell each other what you chose …
    - ... at the same time
- Who wins?
    - Rock beats scissors
    - Scissors beats paper
    - Paper beats rock
    - Anything else is a draw.

Playing by email

How should we reveal choices simultaneously when there's communication delay?

- If my message to you arrived a few seconds late, how would you tell if it had been delayed in transit, or if I'd waited to receive your message before sending mine?

Could rely on email servers' clocks

- Communications systems may not add timestamp
- Can be forged anyway

Secure Commitment

You want me to commit to my choice before you tell me yours

- You don’t want me to be able to change my mind after hearing your choice
- But I don’t want to tell you my choice, so you cannot use that knowledge to generate your choice

Are we stuck?

Hash Functions [1]

Written $H(m) = v$

- ‘m’ is a *message*
  - The thing we want to hash
  - Variable size
  - Any sort of input data you like
- ‘v’ is the *hash value*
  - Output of the hash function
  - Fixed size, even if input size varies

Hash Functions [2]

Good hash functions have three properties

1. First pre-image resistance:
   given a specific ‘v_0’, it’s hard to find any ‘m_0’ such that $H(m_0)= v_0$
2. Second pre-image resistance:
   given a message m_1, it’s hard to find a different message m_2 such that $H(m_1)=H(m_2)$
3. (Strong) Collision resistance:
   it’s hard to find any two different messages, m_x and m_y, that have the same hash: $H(m_x)=H(m_y)$

Should be easy to compute H(x) given x.

A simple hash function

Let’s try the remainder after division by 10.

- Mathematically $H(x) = (x \bmod 10)$
- Examples:
  - $H(16777215) = 5$
  - $H(33) = 3$
  - $H(0) = 0$
- This accepts inputs (messages) of any length.
- The output is fixed size (one decimal digit).

But is it any good?

Evaluating our hash function

$H(x) = (x \bmod 10)$

It's rubbish! Remember…

- First pre-image resistance:
  given a specific 'v_0', it's hard to find any 'm_0' such that $H(m_0)= v_0$

Unfortunately, it's trivial to find a message, m_0 such that $H(m_0)=v_0$: $m_0 = v_0$ always works!

So does $m_0 = v_0 + 10n \quad n=0,1,2,...$

Evaluating our hash function

$H(x) = (x \bmod 10)$

It's *really* rubbish! Remember…

- Second pre-image resistance:
  given a message m_1, it's hard to find a different message m_2 such that $H(m_1)=H(m_2)$

Unfortunately, it's trivial to find a different message, m_2 such that $H(m_1)=H(m_2)$:
$m_2 = m_1 + 10n$ where $n=\ldots,-2,-1,1,2,\ldots$ (not 0 so m_1, m_2 differ)

Evaluating our hash function

$H(x) = (x \bmod 10)$

This is the worst hash function ever! Remember…

- (Strong) Collision resistance:
  it's hard to find any two different messages, m_x and m_y, that have the same hash:
  $H(m_x)=H(m_y)$

Unfortunately, it's trivial to find different messages, m_x and m_y, such that $H(m_x)=H(m_y)$:
$m_x = m_y + 10n$ where $n=\dots,-2,-1,1,2,\dots$ (not 0 so m_x, m_y differ)

Better Hash Functions

Old:

- MD2 (1989): first pre-image and collisions now cracked
- MD4 (1990): all three properties now defeated
- MD5 (1991): first pre-image and collisions now cracked

These output a 128-bit number given any size input message.

Cracking MD5 requires a (really good) supercomputer, but they exist!

Better Hash Functions

Recent:

- SHA-0 (1993): can find collisions with a supercomputer
- SHA-1 (1993): can find collisions with lots of supercomputers
- SHA-256 (2001), SHA-512 (2001): first pre-image + collisions (though you need an awesome supercomputer for this!)
- SHA-3 (2012): secure! For now...

SHA=Secure Hash Algorithm

Back to Rock-Paper-Scissors

You wanted me to commit to a choice but I didn’t want to tell you what the choice was.

Idea: I could hash my choice and reveal the hash value.

- First pre-image resistance means that you cannot go from the hash value back to the choice.
- Second pre-image resistance means you’re confident that I cannot tell you my hash value but then change my choice in a way that keeps the same hash value.
- Collisions resistance means I cannot borrow the University’s supercomputer for a month in advance, and find a hash collision.

Are we done?

No. We've got excited and rushed in without thinking hard enough. ☹️

There are only three choices: R, P, S.

So there are only three hash values!

You can calculate the hash of each of them and check whether my hashed choice is the first, second or third hash value.

But my hash function was super-duper, so what's gone wrong?

The hash function was good but the way we used it was dumb.

Doing this properly

The problem was that there are only three choices.

I need there to be so many choices that you cannot possibly compute all the possible hash values.

I append my choice, C, to a long random number, N (say, 256 bits). Use Physics to invent random junk...

Then I send you $H(N_{256} \,||\, C)$ || means concatenation

Properties of $H(N_{256} || C)$

This is a *salted hash*.

C is ‘R’, ‘P’ or ‘S’.

For each one, there are 2^{256} possible inputs to the hash function.

To tabulate all those at 1000 per second, you’d need 10^{67} years and a LOT of energy! You might also want a really, really big hard disk to write down all the answers.

The final protocol

Phase 1

- Each player makes their choice.
- Each player does some Physics to get a 256-bit random number: their own N_{256}.
- Each player sends $H(N_{256} \| C)$ to the other.

Phase 2

- On receipt, they send their N and C to each other.
- Check other player's hash.
- Decide who won.

A final horror

Think about that "append" operation: I appended a player's choice to the random number: $N \mid\mid C$.

If I'd encoded C as a number (say, 0 for Rock, 1 for Paper, 2 for Scissors) and added it to N, so the players send $H(N_{256} + C)$, then the protocol is insecure – I can arrange to never lose!

What do you do to guarantee that you never lose?

Now it's time for you to have a go

Exercise sheet!