Week 6 - Friday

COMP 4290

Last time

- What did we talk about last time?
- Exam 1!

- Before that:
  - Review
  - Attacks against hash functions
  - Digital signatures

Questions?

Project 2

Olivia Crespo Presents

Quantum Cryptography

Quantum cryptography

- When people talk about quantum computers in the context of cryptography, they're usually talking about one of two very different things:
  - Breaking cryptography with quantum computers
  - Using quantum mechanics to send secret messages

Quantum computers

- A quantum computer is built using **qubits** instead of classical bits for memory
- A qubit is not necessarily 1 or 0
  - Instead, it can be a superposition of them (both at the same time)
- A quantum computer may be able to explore many possible answers at the same time
- After computation is complete, the qubits are measured
- Measuring collapses their quantum states into either 1's or 0's

Shor's algorithm

- Shor's algorithm is an algorithm invented by Peter Shor in 1994 to factor integers
- With enough qubits, Shor's algorithm can factor an integer in $\mathbf{O}((\log N)^2(\log\log N)(\log\log\log N))$ time
  - In other words, polynomial in the length of the number N that is being factored
- RSA depends on the difficulty of factoring
- Shor's algorithm can be adapted to solve the discrete log problem as well

What a quantum computer could do

- Break all popular public key cryptography
  - The RSA, El Gamal, and Elliptic Curve public key systems can all be broken by Shor's algorithm
  - But there are some other systems out there that are not known to be vulnerable to quantum algorithms
- Get a quadratic speedup when trying to brute force symmetric ciphers like AES
  - Meaning that $\sqrt{2^n} = 2^{\frac{n}{2}}$ attempts might be needed instead of 2^n
  - Suggests that key lengths should be doubled

What quantum computers have done

- 2001: Factor 15 into 3 × 5
- 2012: Factor 21 into 3 × 7
- 2012: Factor 143 into 11 × 13
- 2012: Factor 56153 into 241 × 233 (although not realized until 2014)
- 2019: A Google quantum computer sampled a random quantum circuit 1,000,000 times in just over 3 minutes when simulating it with a supercomputer would have taken 10,000 years …
- 2022: Factor 261980999226229 into 15538213 × 16860433 (but using a kind of quantum computer that probably won't work for much larger numbers)
- 2024: Google quantum computer did *something* in 5 minutes they claim would take a supercomputer 10 septillion years to solve …
- There is some progress!
- The bigger factoring ones are using a different approach to quantum computing than Shor's algorithm
- A breakthrough could be soon …

Technical problems

- There are lots of approaches for making quantum computers
  - But none of them look very good yet
- It's hard to make qubits that behave right
- It's hard to make algorithms that efficiently query the qubits
- A phenomenon called quantum decoherence causes qubits to lose their superposition
  - Some quantum computers have to be cooled to almost absolute zero to reduce decoherence
- D-Wave is the best known quantum computer manufacturer
  - So far, none of their computers outperform classical computers on tasks that most people care about

Quantum resistant algorithms

- Because of potential quantum advances, cryptographers have worked on quantum-resistant algorithms, also called post-quantum cryptography (PQC)
  - Kyber is a key encapsulation mechanism with 512, 768, and 1024 bit keys that is supposed to be as strong as AES with 128, 192, and 256 bit keys, in a quantum environment
  - It shares symmetric keys (like AES), instead of something like RSA or Diffie-Hellman
  - It uses lattice theory, a deep math thing
  - There are also hash-based approaches to PQC
- AES using 256 bit keys is considered quantum secure

Quantum communication

- The other (and possibly more useful) way to use quantum mechanics is as a way to send secret information
- To do so, Sam (the sender) sends Ruth (the receiver) photons
- What's really remarkable about this kind of quantum cryptography is that no one can eavesdrop on it

The setup

- Light travels with a certain orientation, called its **polarization**
- Real polarization can be between 0° and 180° (the book says 360 °, but the wave goes up and down, so it doesn't really make sense to say that)
- We can break it down into 4 directions by rounding: ↑ → ↗ ↘
- It's critically important that it's hard to distinguish ↑ and ↗ and hard to distinguish → and ↘

Sending

- For each bit Sam wants to send, he randomly decides if he's sending straight or diagonal

Basis	Filter	0	1
Straight	+	↑	→
Diagonal	X	↗	↘

- Ruth also randomly decides if she's receiving straight or diagonal
  - If she uses the wrong filter, ↑ will be confused with ↗ or → will be confused with↘

Receiving

- Ruth receives symbols, but she isn't sure what she got
- On a public channel, she tells Sam which basis she was using for each bit
- Then Sam knows which ones she would have gotten right and which ones she would have gotten wrong
- He tells her which ones she would have gotten right
- Those are the ones that they keep

Why this works

- It's not a great way to send a message
- But it is a great way to agree on random bits
  - In other words, a session key for regular communication
- If Eve is eavesdropping, quantum properties say that she'll disturb the polarization of the photons
- Thus, Ruth will get garbage
- To make sure that no one is eavesdropping, Sam and Ruth share some bits publically, to see if they agree

Technical problems

- Most importantly, the process is inefficient
  - On average, half of the bits are bad, since Ruth had the wrong filter
  - Even more bits have to be ignored in order to do error checking and testing to see if there was an eavesdropper
- Making the actual photon guns isn't easy, although a kind of pulsed laser has been successfully used
- Another problem is that you have to have an optical quantum channel with anyone you want to exchange a key with
  - Either through the atmosphere or through fiber optics

Successes

- This technology is moving from theory into practice
- Government labs in the US and the UK have tested devices working through the atmosphere at up to 45 km
- Businesses that want to send high security traffic may negotiate keys using quantum key distribution in our lifetimes

Program Security

Secure programs

- For now, we will be pretty broad in our definition of programs
  - OS
  - Applications
  - Databases
  - Almost any other software
- What is a secure program?
- How do we know?
- How do we keep programs free from flaws?
- How do we protect computing resources against programs that contain flaws?

Fixing faults

- We can judge security by the number of faults found in a program
- Count the number of faults found and fixed
  - Program is good if it has few faults to begin with, right?
  - But isn't the program good if we've fixed a lot of faults?
  - Which is more meaningful?

Penetrate and patch

- In the early days, security was shown by finding faults and patching them
- Unfortunately, patching a fault often led to creating another one
- Why?
  - The patch fixed a narrow problem, but the cause was more general
  - The fault had non-obvious side effects
  - Fixing one problem caused a problem somewhere else
  - The fault was poorly fixed because a proper fix might impact functionality or performance

Terminology

- We talk about software **bugs**, but the term is vague
- The IEEE favors the following:
  - **Error:** A human mistake in developing software (bad design, bad implementation, typo…)
  - **Fault:** An incorrect step inside of a program (many faults can be caused by a single error)
  - **Failure:** A system departing from its required behavior (a failure might not happen if a particular fault is never executed)

Unexpected behavior

- Unexpected behavior is called a **program security flaw**
- The IEEE terminology is for software engineering and doesn't match exactly
- A program security flaw could be a fault or a failure
- Intentional security incidents are called **cyber attacks**
- Cyber attacks are not as common as the problems caused by unintentional flaws

Why is life so hard?

- It's very difficult to eliminate program security flaws for two reasons:

1. Programs should do a long list of operations correctly and shouldn't do another (possibly infinite) list of operations
   - The number of combinations to test is staggering
2. Software engineering develops faster than computer security
   - We're always playing catch-up

Non-malicious program errors

Flaws

- Landwehr et al. divided flaws into intentional and inadvertent
- The inadvertent flaws were divided into six categories
  1. **Validation:** Incomplete or inconsistent permission checks
  2. **Domain:** Poorly controlled access to data
  3. **Serialization and aliasing:** Mistakes in program flow order
  4. **Identification and authentication:** Incorrect basis for authentication
  5. **Boundary condition:** Failure on the first or last case
  6. **Logic:** Any mistakes in logic not already covered
- Other lists have been made, but this one is representative
- The next slides will cover some common types

Buffer overflows

- A **buffer overflow** happens when data is written past the end (or beginning) of an array
- Consider the following Java code:

```
char[] buffer = new char[10];

for(int i = 0; i < 10; ++i) {
     buffer[i] = 'A';
}
buffer[10] = 'B';
```

- In Java, this code will throw an **`ArrayIndexOutOfBoundsException`**, but it will not write memory where it shouldn't
- In C/C++, it might

Buffer overflow

- It could overwrite:
  - User data
  - User code
  - System data
  - System code

Buffer overflow security

- Without the presence of malicious attackers, buffer overflows can corrupt your data (or the system's) or crash your program
- A malicious attacker can exploit buffer overflows
  - By inserting data into system data or code so that the system does what he or she wants
  - By overwriting the stack pointer to cause arbitrary code in the attacker's memory to be executed
- Memory segmentation makes these attacks less common but still possible

Upcoming

Next time…

- Malicious code
- Start countermeasures
- Ashley Gutierrez presents

Reminders

- Read sections 3.1, 3.2, and 3.3
- Start on Project 2