Announcements

- HW #3
  - Due now

- Project #1
  - Early: today by midnight – go for the bonus!
  - Due: Friday by midnight

Objectives

- Understand the big picture for cryptography
- Introduce Public Key Cryptography
- Apply all of our favorite algorithms to define RSA
- Understand why RSA is secure

Punch-Line

- RSA
  - Named after Rivest, Shamir, Adleman
  - Gives strong guarantees of security

  Exploits:
  - Polynomial time computability of:
    - Modular Exponentiation – modexp()
    - Greatest Common Divisor – extended-Euclid()
    - Fermat Primality Testing – primality2()
  - Intractability of:
    - Factoring
    - Modular root finding

Cryptography

- Alice and Bob want to communicate in private
- Eve is an eavesdropper
- Alice wants to send message x to Bob
- She encrypts clear-text x with e(): y = e(x)
- She sends the cypher-text y to Bob
- Bob decrypts y with d(): d(y) = x
- Goal: If Eve intercepts y, without d( ) she can do nothing.

\[\text{Alice: } x \xrightarrow{e(\cdot)} y \xrightarrow{d(\cdot)} x \quad \text{Bob}\]

\[\text{Eve}\]

Past: private key protocols
- Exchange codebook
  - E.g., One-time Pad
  - E.g., AES (Advanced Encryption Standard)

Present: public key protocols
- Never need to meet
  - e_{pub}(\cdot) is publicly available
  - Only Bob possesses d_{priv}(\cdot)
  - Alice can quickly run e_{pub}(\cdot)
  - Bob can quickly run d_{priv}(\cdot)
  - Without d_{priv}(\cdot), Eve must perform operations like factoring large numbers. Have fun!
  - E.g., RSA
Public Key Cryptography

Bob
Publishes his public key

Thanks to Diffie, Hellman, and Merkle

Alice

Public Key Cryptography

Bob
Get's Bob's public key

Thanks to Diffie, Hellman, and Merkle

Alice

Public Key Cryptography

Bob
Uses the key to encrypt her message

Alice

Public Key Cryptography

Bob
Sends the encrypted message over an open channel

Alice

Public Key Cryptography

Bob
Uses private knowledge to decrypt the message

Alice

Sends the encrypted message over an open channel

RSA

- Messages from Alice and Bob are numbers modulo $N$
- Messages larger than $N$ are broken into blocks, each represented as a number modulo $N$
- Encryption is a bijection on $\{0, 1, \ldots, N - 1\}$
  - i.e., a permutation
- Decryption is its inverse

Function that is both one-to-one and onto
Number Theory

- Let \( p \) and \( q \) be any two primes
- Let \( N = p \cdot q \)
- The "totient" function \( \phi(N) = (p - 1)(q - 1) \)
- Let \( e \) be a number relatively prime to \( \phi(N) \)
- Then (theorem)
  - \( f: x \rightarrow x^e \mod N \) is a bijection on \( \{0,1,\ldots,N-1\} \)
- Let \( d = \) the multiplicative inverse of \( e \mod \phi(N) \)
- Then (theorem)
  - \( g: y \rightarrow y^d \mod N \) is also a bijection on \( \{0,1,\ldots,N-1\} \)
  - Furthermore, for all \( x \in \{0,1,\ldots,N-1\} \),
    - \( (x^e)^d = x \mod N \)

Key Generation

Bob needs to generate his public and private keys.
- He picks two large \( n \)-bit random primes \( p \) and \( q \) (how?)
  - What role should primality tester play?
  - Test random \( n \)-bit numbers: \( O(n) \) to find one
- Public key is \( (N,e) \)
  - Where \( N = p \cdot q \)
  - Where \( e \) is an (at most) \( 2^{n/2} \)-bit number relatively prime to \( \phi(N) = (p-1)(q-1) \)
  - Often \( e = 3 \), which permits fast encoding
- Private key is \( d \), the multiplicative inverse of \( e \mod \phi(N) \)
  - How to compute?
    - extended-Euclid\((p-1)(q-1),e)\)
    - That should help with exercise 1.27 on HW#4

Sending Messages

Alice wants to send \( x \) to Bob
- Alice looks up his public key \( (N,e) \)
- She encodes \( x \) as cypher-text \( y = x^e \mod N \)
  - How to compute?
- Bob receives \( y \) and decodes it: \( x = y^d \mod N \)
  - How to compute?

That's it!

Example

- Key generation:
  - Let \( p = 5, q = 11 \)
  - \( N = p \cdot q = 55 \)
  - Let \( e = 3 \)
  - Check \( \text{gcd}(p-1)(q-1),e) = \text{Euclid}(40,3) = 1 \)
    - Therefore, relatively prime.
  - Public key: \( (N,e) = (55,3) \)
    - Private key: \( d = \) multiplicative inverse of 3 mod 40 = 27
    - Computed using extended-Euclid()
- Encryption:
  - Clear-text message \( x = 13 \)
  - \( y = x^e \mod 55 \)
  - Cypher-text message \( y = 13^3 \mod 55 = 52 \)
- Decryption:
  - \( x = y^d \mod 55 \)
  - Decrypted clear-text message \( x = 52^{27} \mod 55 = 13 \)

How Safe is RSA?

- There are two main principled attacks.
  - The first:
    - Factor the public key \( N \) into its primes
    - Compute multiplicative inverse of \( e \mod \phi(N) \) to get \( d \)
      - Given \( d \) and \( y \) (ciphertext), compute \( x = y^d \mod N \)
        using modular exponentiation.
    - Factoring is hard, but nobody knows how hard.
      - It is unlikely to be in either \( P \) or \( NP \)-complete.
  - The second attack involves computing \( y^{\frac{1}{2}} \equiv \sqrt{y} \mod N \)
    - Reasoning:
      - \( y^{\frac{1}{2}} \equiv (x^e)^{\frac{1}{2}} \equiv x \mod N \)
    - However, there is no known efficient algorithm for finding modular roots.
How Safe is RSA?

- **Third: Brute Force Attacks**
  - Try all values of $d$ – harder than factoring
  - Try all primes from 1 to $\sqrt{N}$

- Use $k$ computers: require $\frac{1}{k}$ time to crack

---

The Punch-line

- The crux of the security behind RSA
  - Efficient algorithms / Polynomial time computability of:
    - Modular Exponentiation – modexp()
    - Greatest Common Divisor – extended-Euclid()
    - Primality Testing – primality2()
  - Absence of efficient algorithms / Intractability of:
    - Factoring
    - Modular root finding

---

Using Public Key Authentication

- **PGP (Pretty Good Privacy)** uses the **Fermat** primality test.
- **SSH** uses ssh-keygen to generate a suitable pair of keys.
  - The prime number is tested using two methods.
    - The second of which is Miller-Rabin and some filtering based on known composites
    - (see source code)
  - From **man ssh**
    - Put your public key on the server (presumably because anyone can see your server)
    - Put your private key on your client (because you control it).
    - ssh can use public key authentication to verify that you are who you say you are without a password.
      - The server checks if this key is permitted
      - First, send the user (actually the ssh program running on behalf of the user) a challenge, a random number encrypted by the user's public key
      - The challenge can only be decrypted using the proper private key.
      - The user's client then decrypts the challenge using the private key, proving that he/she knows the private key but without disclosing it to the server.

---

RSA and SSH

**Client [private key]**

- Login
- Decrypt the random number using private key

**Server [public key]**

- Encrypt a random number using public key
- If numbers match, then you must have the private key

---

Thoughts

- Security of this scheme remains unproven
  - Factoring large numbers into their primes may turn out to be easy or unnecessary to break the code
  - Can you break it, or prove it is hard to break?
- Are there other “one-way” functions to do the job?
- Can we approximately break the code—derive a message $m$ within provable bounds of the clear-text $x$?
- Are there alternative handshaking schemes to facilitate private communication without prior coordination?
- Need creative minds on this problem (cool jobs)

---

Assignment

- **HW #4:**
  - 1.27
  - 2.1
  - 2.5(a-e) using the Master Theorem

- Due after the long weekend
  - Remember proj. #1 is due on Friday