Modular Arithmetic and RSA Encryption

Stuart Reges, Principal Lecturer, University of Washington

Some Basic Terminology

- Alice wants to send a secret message to Bob
- Eve is eavesdropping
- Cryptographers tell Alice and Bob how to encode their messages
- Cryptanalysts help Eve to break the code
- Historic battle between the cryptographers and the cryptanalysts

Public Key Encryption

- Proposed by Diffie, Hellman, Merkle
- First big idea: use a function that cannot be reversed (a humpty dumpty function): Bob tells Alice a function to apply using a public key, and Eve can't compute the inverse
- Second big idea: use asymmetric keys (sender and receiver use different keys): Bob has a private key to compute the inverse
- Primary benefit: doesn't require the sharing of a secret key

RSA Encryption

- Named for Ron Rivest, Adi Shamir, and Leonard Adleman
- Invented in 1977, still the premier approach
- Based on Fermat's Little Theorem:
  $a^{p-1} \equiv 1 \pmod{p}$ for prime p, gcd(a, p) = 1
- Slight variation:
  $a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$ for distinct primes p and q, gcd(a,pq) = 1
- Requires large primes (100+ digit primes)

Example of RSA

- Pick two primes p and q, compute $n = p \times q$
- Pick two numbers e and d, such that:
  $e \times d = (p-1)(q-1)k + 1$ (for some k)
- Publish n and e (public key), encode with:
  $(\text{original message})^e \bmod n$
- Keep d, p and q secret (private key), decode with:
  $(\text{encoded message})^d \bmod n$

Why does it work?

- Original message is carried to the e power, then to the d power:
  $(msg^e)^d = msg^{e \times d}$
- Remember how we picked e and d:
  $msg^{ed} = msg^{(p-1)(q-1)k + 1}$
- Apply some simple algebra:
  $msg^{ed} = (msg^{(p-1)(q-1)})^k \times msg^1$
- Applying Fermat's Little Theorem:
  $msg^{ed} = (1)^k \times msg^1 = msg$

Further exploration

- Longer description of this talk: http://www.cs.washington.edu/homes/reges/rsa/
- CryptoKids: http://www.nsa.gov/kids/
- Python programming: http://python.org/