CSC 580
Cryptography and Computer Security

Math for Public Key Crypto, RSA, and Diffie-Hellman
(Sections 2.4-2.6, 2.8, 9.2, 10.1-10.2)

March 21, 2017

Overview

Today:

- Math needed for basic public-key crypto algorithms
- RSA and Diffie-Hellman

Next:

- Read Chapter 11 (skip SHA-512 logic and SHA3 iteration function)
- Project phase 3 due in one week (March 28) - finish it!

Background / Context

Recall example “trapdoor” function from last time: *Given a number n, how many positive integers divide evenly into n*?

- If you know the prime factorization of *n*, this is easy.
- If you don’t know the factorization, don’t know efficient solution

How does this fit into the public key crypto model?

- Pick two large (e.g., 1024-bit) prime numbers p and q
- Compute the product $n = p * q$
- Public key is *n* (hard to find *p* and *q*!), private is the pair (*p*,*q*)

Questions:

- How do we pick (or detect) large prime numbers?
- How do we use this trapdoor knowledge to encrypt?

Prime Numbers

A prime number is a number *p* for which its only positive divisors are 1 and *p*

Question: How common are prime numbers?

- The Prime Number Theorem states that there are approximately $n / \ln n$ prime numbers less than *n*.
- Picking a random *b*-bit number, probability that it is prime is approximately $1/\ln(2^b) = (1/\ln 2)*(1/b) \approx 1.44 * (1/b)$
  - For 1024-bit numbers this is about 1/710
  - "Pick random 1024-bit numbers until one is prime" takes on average 710 trials
  - This is efficient - if we can tell when a number is prime!

Primality Testing

Problem: Given a number n, is it prime?

Basic algorithm: Try dividing all numbers 2,..,sqrt(*n*) into *n*

Question: How long does this take if *n* is 1024 bits?

[redacted]

Fermat's Little Theorem

To do better, we need to understand some properties of prime numbers, such as…

Fermat's Little Theorem: If *p* is prime and *a* is a positive integer not divisible by *p*, then

$$a^{p-1} \equiv 1 \pmod{p} .$$

Proof is on page 46 of the textbook (not difficult!).

Fermat's Little Theorem - cont'd

Explore this formula for different values of *n* and random *a*'s:

a	$a^{n-1} \bmod n$ ($n = 221$)	$a^{n-1} \bmod n$ ($n = 331$)	$a^{n-1} \bmod n$ ($n = 441$)	$a^{n-1} \bmod n$ ($n = 541$)
64	1	1	379	1
189	152	1	0	1
82	191	1	46	1
147	217	1	0	1
113	217	1	232	1
198	81	1	270	1

Question 1: What conclusion can be drawn about the primality of 221?

Question 2: What conclusion can be drawn about the primality of 331?

Primality Testing - First Attempt

Tempting (but incorrect) primality testing algorithm for *n*:

```
Pick random a ∈ {2, ... , n-2}
if a^(n-1) mod n ≠ 1 then return "not prime"
else return "probably prime"
```

Why doesn't this work?

Primality Testing - First Attempt

Tempting (but incorrect) primality testing algorithm for *n*:

```
Pick random a ∈ {2, ... , n-2}
if a^(n-1) mod n ≠ 1 then return "not prime"
else return "probably prime"
```


Why doesn't this work? *Carmichael numbers*….

Example: 2465 is obviously not prime, but →

a	$a^{n-1} \bmod n$ ($n = 2465$)
64	1
189	1
82	1
147	1
113	1
198	1

Note: Not just for these *a*'s, but $a^{n-1} \bmod n = 1$ for **all** *a*'s that are relatively prime to *n*.

Primality Testing - Miller-Rabin

The previous idea is good, with some modifications
(Note: This corrects a couple of typos in the textbook):

```
MILLER-RABIN-TEST(n)   // Assume n is odd
    Find k>0 and q odd such that n-1 = 2^k q
    Pick random a ∈ {2, ... , n-2}
    x = a^q mod n
    if x = 1 or x = n-1 then return "possible prime"
    for j = 1 to k-1 do
        x = x^2 mod n
        if x = n-1 then return "possible prime"
    return "composite"
```

If n is prime, always returns "possible prime"
If n is composite, says "possible prime" with probability < ¼

Idea: Run 50 times, and accept as prime iff all say "possible prime"
Question: What is the error probability?

Euler's Totient Function and Theorem

Euler's totient function: $\phi(n)$ = number of integers from 1..n-1 that are relatively prime to n.

- If $s(n)$ is count of 1..n-1 that share a factor with n, $\phi(n) = n - 1 - s(n)$
  - $s(n)$ was our "trapdoor function" example
  - $\phi(n)$ easy to compute if factorization of n known
  - Don't know how to efficiently compute otherwise
- If n is product of two primes, $n=p*q$, then $s(n)=(p-1)+(q-1)=p+q-2$
  - So $\phi(p*q) = p*q - 1 - (p+q-2) = p*q - p - q + 1 = (p-1)*(q-1)$

Euler generalized Fermat's Little Theorem to composite moduli:

Euler's Theorem: For every a and n that are relatively prime (i.e., gcd(a,n)=1),

$$a^{\phi(n)} \equiv 1 \pmod{n} .$$

Question: How does this simplify if n is prime?

RSA Algorithm

Key Generation:

Pick two large primes p and q
Calculate $n=p*q$ and $\phi(n)=(p-1)*(q-1)$
Pick a random e such that gcd(e, $\phi(n)$)
Compute $d = e^{-1} \pmod{\phi(n)}$ [*Use extended GCD algorithm!*]
Public key is $PU=(n,e)$; Private key is $PR=(n,d)$

Encryption of message $M \in \{0,..,n-1\}$:

E(PU,M) = M^e mod n

Decryption of ciphertext $C \in \{0,..,n-1\}$:

D(PR,C) = C^d mod n

RSA Algorithm

Key Generation:

Pick two large primes p and q
Calculate $n=p*q$ and $\phi(n)=(p-1)*(q-1)$
Pick a random e such that gcd(e, $\phi(n)$)
Compute $d = e^{-1} \pmod{\phi(n)}$ *[Use extended GCD algorithm!]*
Public key is $PU=(n,e)$; Private key is $PR=(n,d)$

Encryption of message $M \in \{0,..,n-1\}$:

$E(PU,M) = M^e \bmod n$

Decryption of ciphertext $C \in \{0,..,n-1\}$:

$D(PR,C) = C^d \bmod n$

Correctness - easy when gcd(M,n)=1:

$D(PR,E(PU,M)) = (M^e)^d \bmod n$
$= M^{ed} \bmod n$
$= M^{k\phi(n)+1} \bmod n$
$= (M^{\phi(n)})^k M \bmod n$
$= M$

Also works when gcd(M,n)≠1, but slightly harder to show...

RSA Example

Simple example:

p = 73, q = 89
n = p*q = 73*89 = 6497
$\phi(n)$ = (p-1)*(q-1) = 72*88 = 6336
e = 5
d = 5069 [Note: 5*5069 = 25,345 = 4*6336 + 1]

Encrypting message M=1234:

$1234^5 \bmod 6497 = 1881$

Decrypting:

$1881^{5069} \bmod 6497 = 1234$

Note: If time allows in class, more examples using Python!

The Discrete Log Problem

For every prime number p, there exists a primitive root (or "generator") g such that

$$g^1, g^2, g^3, g^4, \ldots, g^{p-2}, g^{p-1} \quad \text{(all taken mod } p\text{)}$$

are all distinct values (so a permutation of 1, 2, 3, ..., p-1).

Example: 3 is a primitive root of 17, with powers:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$3^i \bmod 17$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

$f_{g,p}(i) = g^i \bmod p$ is a bijective mapping on $\{1,.., p-1\}$

$f_{g,p}(i)$ is easy to compute (modular powering algorithm)

Inverse, written $\mathrm{dlog}_{g,p}(x) = f_{g,p}^{-1}(x)$, is believed to be difficult to compute

g and p are global public parameters

Diffie-Hellman Key Exchange

Assume g and p are known, public parameters

Alice

$a \leftarrow$ random value from $\{1, \ldots, p\text{-}1\}$
$A \leftarrow g^a \bmod p$

Bob

$b \leftarrow$ random value from $\{1, \ldots, p\text{-}1\}$
$B \leftarrow g^b \bmod p$

Send A to Bob →

← *Send B to Alice*

$S_a \leftarrow B^a \bmod p$

$S_b \leftarrow A^b \bmod p$

In the end, Alice's secret (S_a) is the same as Bob's secret (S_b):

$$S_a = B^a = g^{ba} = g^{ab} = A^b = S_b$$

Eavesdropper knows A and B, but to get a or b requires solving the discrete logarithm problem!

Abstracting the Problem

There are many sets over which we can define powering.

Example: Can look at powers of $n \times n$ matrices (A^2, A^3, etc.)

Any finite set S with an element g such that $f_g : S \rightarrow S$ is a bijection (where $f_g(x) = g^x$ for all $x \in S$) is called a *cyclic group*

- Very cool math here - see Chapter 5 for more info (optional)

If f_g is easy to compute and f_g^{-1} is difficult, then can do Diffie-Hellman

"Elliptic Curves" are a mathematical object with this property

In fact: f_g^{-1} seems to be harder to compute for Elliptic Curves than $\mathbf{Z}_p$

- Consequence: Elliptic Curves can use shorter numbers/keys than standard Diffie-Hellman - so faster and less communication required!

Revisiting Key Sizes

From NIST publication 800-57a

Issue: PK algorithms based on mathematical relationships, and can be broken with algorithms that are faster than brute force.

We spent time getting a feel for how big symmetric cipher\ keys needed to be
→ How big do keys in a public key system need to be?

From NIST pub 800-57a:

Table 2: Comparable strengths

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$