2016-05-23

William A. Stein

5/23/2016

Contents

1 Math 480: Open Source Mathematical Software

1.0.1 2016-05-23

1.0.2 William Stein

1.1 *Lectures 25: Public-key crypto (part 1 of 3) *

This week we will talk about public key crypto, as a way to learn some **computational number theory**.

- Modular exponentation
- Diffie-Hellman
- RSA

Today:

1. Homework was collected, peer grading available (no guidelines available yet).
2. New homework that is due Friday at 6pm is now available.

TODAY: Some background so you can start working on your homework.

1.2 1. Modular Arithmetic

Let n be a positive integer. Then

$$/n = \{0, 1, 2, \ldots, n-1\}$$

is a ring with addition and multiplication modulo n.

```
R = IntegerModRing(12)
R
```

```
Ring of integers modulo 12
```

```
list(R)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
R(7) + R(9)
```

```
4
```

```
7 + 9
```

```
16
```

```
16 % 12
```

```
4
```

```
R(16)
```

```
4
```

```
# shorthand way to make a "number modulo 12"
a = Mod(7, 12)
a
```

```
7
```

```
parent(a)
```

```
Ring of integers modulo 12
```

```
type(a)
```

```
<type 'sage.rings.finite_rings.integer_mod.IntegerMod_int'>
```

Application: What are the last 3 digits of $n = 123456789^{123456789}$?
Solution: compute n modulo 1000:

```
Mod(123456789, 1000)^123456789
```

```
709
```

How does this work.

1. Write $a = 123456789$ in binary.
2. View $a^{123456789}$ as multiplying together a bunch of numbers of the form a^{2^i}, which we get by repeating squaring, using that $a^{2^i} = ((a^2)^2 \ldots)^2$.

```
123456789.bits()
```

```
[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1]
```

1.3 2. Diffie-Hellman

This is a very widely used algorithm that allows two programs A and B that communicate over an unsecured channel to agree on a shared secret. Here's how it works.

- Step 1. A and B agree on a prime number p and a number g modulo p.
- Step 2. A generates a random number $a < p$ and sends $g^a \pmod{p}$.
- Step 3. B generates a random number $b < p$ and sends $g^b \pmod{p}$.
- Step 4. A computes the shared secret $s = (g^a)^b \pmod{p}$.
- Step 5. B computes the shared secret $s = (g^b)^a \pmod{p}$.

A and B agree on the same secret because $(g^a)^b = g^{ab} = (g^b)^a$.

1.3.1 Security

The only information transmitted publicly is g, p, $g^a \pmod{p}$, and $g^b \pmod{p}$.

An adversary (say C) can do a few things:

- 1. Since C knows g, p, $g^a \pmod{p}$, it can **in theory** compute a! Just try powers of g until $g^t \pmod{p} = g^a \pmod{p}$. Also, there are many much more sophisticated algorithms to solve this **discrete logarithm problem** $a = \log_g(g^a \pmod{p})$.
- 1. Maybe C can trick A and B into thinking they are talking to each other, but they are really talking to C. (Major government agencies are well positioned to do this.)
- 1. Trick people into thinking p is prime when it isn't, and use this to create a backdoor. See homework problem 5.

Let's try it out!

```
set_random_seed(0)
# - Step 1. A and B agree on a prime number $p$ and a number $g$ \
  modulo $p$.
p = next_prime(10^100); g = Mod(2,p)
print "p=",p," g=",g
```

```
p= 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000267  g= 2
```

```
#  - Step 2. A generates a random number $a<p$ and sends $g^a\pmod{p\
   }$.
a = ZZ.random_element(p)
A_sends = g^a
A_sends
```

```
8961546464985269483822041230404389831420657654799728216338186719614899911445736678190040845169987389
```

```
# - Step 3. B generates a random number $b<p$ and sends $g^b\pmod{p}\
   $.
b = ZZ.random_element(p)
B_sends = g^b
B_sends
```

```
# - Step 4. A computes the shared secret $s = (g^a)^b\pmod{p}$.
A_computes = B_sends^a
A_computes
```

```
2518846831889549883379731685666706951656662994225272047639545644667304086899633914709004318923164641
```

```
# - Step 5. B computes the shared secret $s = (g^b)^a\pmod{p}$.
B_computes = A_sends^b
B_computes
```

```
2518846831889549883379731685666706951656662994225272047639545644667304086899633914709004318923164641
```