

Admin:

Project proposals due Friday!

Project ideas:

Look at Lipmaa's site for ideas:
www.cs.ut.ee/~lipmaa/crypto
(e.g. auctions)

Today:

- Group theory review
- Diffie-Hellman (DH) key exchange
- Five crypto groups: \mathbb{Z}_p^* , \mathbb{Q}_p , \mathbb{Z}_n^* , \mathbb{Q}_n , elliptic curves

(multiplicative group)

identity

inverses

associativity

commutativity

order

Group Theory review:If $(G, *)$ is a finite abelian group of size t :

- \exists identity 1 s.t. $(\forall a \in G) a \cdot 1 = 1 \cdot a = a$
- $(\forall a \in G)(\exists b \in G) a \cdot b = 1 \quad (b = a^{-1})$
- $(\forall a, b, c \in G) a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- $(\forall a, b \in G) a \cdot b = b \cdot a$

[Multiplicative
or
additive
notation]Let $\text{order}(a) = \text{least } u > 0 \text{ s.t. } a^u = 1 \text{ (in } G\text{)}$.Theorem: In a finite abelian group of size t

$$(\forall a \in G) \text{order}(a) \mid t$$

[Lagrange's Theorem]

Theorem: In a finite abelian group of size t

$$(\forall a \in G) a^t = 1$$

Example: $(\forall a \in \mathbb{Z}_p^*) a^{p-1} = 1 \pmod{p}$ since $|\mathbb{Z}_p^*| = p-1$.Def: $\langle a \rangle = \{a^i : i \geq 0\}$ = subgroup generated by a Def: If $\langle a \rangle = G$ then G is cyclic and a is a generator of G .Note: $|\langle a \rangle| = \text{order}(a)$ Exercise: In a finite abelian group G of order t , where t is prime, $(\forall a \in G) [a \neq 1] \Rightarrow [a \text{ is a generator of } G]$.Fact: \mathbb{Z}_p^* is always cyclic.

- Fact: IF G is a cyclic group of order t and generator g , then the relation $x \longleftrightarrow g^x$ is one-to-one between $[0, 1, \dots, t-1]$ and G .

$x \mapsto g^x$: exponentiation, "powering-up"

$g^x \mapsto x$: discrete logarithm (DL)

- Computing discrete logarithms (the DL problem) is commonly assumed to be hard/infeasible for well-chosen groups G . [E.g. \mathbb{Z}_p^* for p a large randomly chosen prime]
- We often need to be able to represent messages as group elements: if M is a message space & G a group, we need an injective (one-to-one) map

$$f: M \rightarrow G$$

such that $f(m)$ can be chosen to "represent" message m .

E.g. if $p \gg 2^k$ then we can identify k -bit messages with the integers $1, 2, \dots, 2^k \bmod p$ (in \mathbb{Z}_p^*).

In some groups this can be a little tricky.

API for a group

- ① $G \leftarrow \text{create_group}(...)$
 - ② $G.\text{identity}()$
 - ③ product $x \cdot y$ [sometimes writes "+"]
 - ④ power x^k $x \in G, k \in \mathbb{Z}$ [\downarrow
 $k \cdot x$
 \downarrow
 $-x$]
 - ⑤ inverse x^{-1}
 - ⑥ random elt $G.\text{random}()$
 - ⑦ size $G.\text{order}()$ $|G|$ [not always]
 - ⑧ list elements $G.\text{elements}()$ [not always]
 - ⑨ represent msg as elt of group & inverse $G.\text{rep}(M)$ $0 \leq M < |G|$
-
- ⑩ generator $G.\text{generator}()$
 - ⑪ discrete log $G.\text{discrete_log}(g, y) = x \text{ s.t. } g^x = y$ [usually not possible]

...

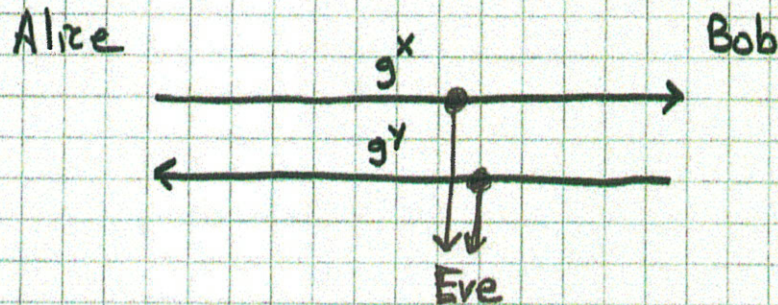
Diffie-Hellman key exchange (1976)

- How to establish shared secret in presence of eavesdropper? (Eve is passive - only listens)
- Precursor to true public-key
- Let G be a cyclic group with generator g .
 G and g are fixed & public.
- Alice chooses secret x randomly from $[0, \dots, |G|-1]$
Alice computes g^x as her "public key"
(transient or permanent). g^x is a
random element of G st. Alice knows
discrete log.
- Bob similarly chooses secret y & computes
"public key" g^y .
- Assumed difficulty of DL in G means even
if ^{Eve} ~~she~~ learns g^x or g^y she doesn't learn
 x or y .

Merkle
puzzles



- Alice sends g^x to Bob. Bob sends g^y to Alice



- Alice computes $K = (g^y)^x = g^{xy}$
- Bob computes $K = (g^x)^y = g^{xy}$
- IF DL hard, Eve can't compute x or y .
(But that doesn't nec. mean she can't compute K)
- CDH (Computational Diffie-Hellman Assumption):
Given g^x & g^y (for random x & y) it
is hard to compute g^{xy} [i.e. negl. chance of success]
- Theorem: $CDH \Rightarrow DH$ key exchange is secure
(i.e. Eve doesn't learn K , except with negl. probability)
Proof: Duh! (\approx assuming desired result!)
- Use K to encrypt and/or MAC later traffic.
(IF both, use PRF to derive encryption key
& MAC key from K ; don't use same key for both!)
- note: keys g^x and g^y are uncertified (OK if Eve is passive)

We look at five commonly used finite groups.

$$\textcircled{1} \mathbb{Z}_p^* = \{a : 1 \leq a < p\} \quad \text{where } p \text{ is prime}$$

\mathbb{Z}_p^* is always cyclic. (Proofs are nonconstructive!)

If $p = 2q + 1$ (q prime), then p is a "safe prime" and half of \mathbb{Z}_p^* are generators, and the other half are squares (Q_p).

$$\textcircled{2} Q_p = \text{quadratic residues (squares) mod prime } p \\ = \{a^2 : 1 \leq a < p\} \\ \subseteq \mathbb{Z}_p^*$$

$$|Q_p| = \frac{1}{2} |\mathbb{Z}_p^*| = (p-1)/2 \quad (\text{"half of } \mathbb{Z}_p^* \text{ are squares"})$$

Q_p is cyclic: If $\langle g \rangle = \mathbb{Z}_p^*$, then $\langle g^2 \rangle = Q_p$.

$$Q_p = \{g^{2i} : 0 \leq i < (p-1)/2\} \quad \text{if } \langle g \rangle = \mathbb{Z}_p^*$$

If $p = 2q + 1$ (p is a "safe prime") then

$$|Q_p| = q$$

and any element of Q_p (other than 1)

generates Q_p . [To find a generator,

take the square of any element $a \notin \{1, p-1\}$.]

$$\textcircled{3} \quad \mathbb{Z}_n^* = \{a : \gcd(a, n) = 1 \ \& \ 1 \leq a < n\} \quad (\text{RSA})$$

$$|\mathbb{Z}_n^*| = \varphi(n) \quad [\text{by defn}]$$

If $n = p \cdot q$ where p, q distinct odd primes,

then \mathbb{Z}_n^* is not cyclic

$$\mathbb{Z}_n^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^* \quad (\text{Chinese remainder thm})$$

$$\textcircled{4} \quad \mathbb{Q}_n = \{a^2 : 1 \leq a < n \ \& \ \gcd(a, n) = 1\}$$

= "squares mod n "

= "quadratic residues mod n "

If $n = p \cdot q$ where

$p = 2r + 1$ is a safe prime (r prime)

$q = 2s + 1$ is a safe prime (s prime)

then

$$|\mathbb{Q}_n| = r \cdot s$$

& \mathbb{Q}_n is cyclic.

⑤ Elliptic curve groups

Quite different, many nice properties, widely used.

Much deep mathematics related to elliptic curves.

Here is a very brief intro.

Let p be a prime.

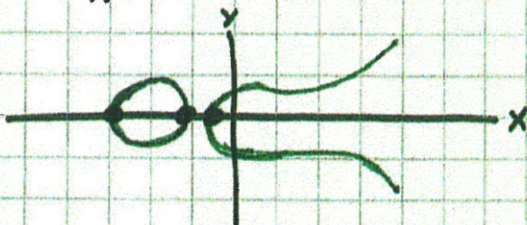
Let a, b be elements of \mathbb{Z}_p such that

$$4a^3 + 27b^2 \neq 0 \pmod{p} \quad (*)$$

Consider equation (in variables $x, y \pmod{p}$)

$$y^2 = x^3 + ax + b \pmod{p} \quad (**)$$

Graphically, something like this



Note that if (x, y) on curve, so is $(x, -y)$.

If roots are r_1, r_2, r_3 then

$$((r_1 - r_2)(r_1 - r_3)(r_2 - r_3))^2 = -(4a^3 + 27b^2)$$

so $(*)$ means roots are distinct.

Def: The points on the curve (***) are

$$E = \{(x, y) : y^2 = x^3 + ax + b \pmod{p}\} \cup \{\infty\}$$

Here " ∞ " denotes the "point at infinity" (e.g. $y = \infty$)

Fact: $|E| = p + 1 + t$ where $|t| \leq 2\sqrt{p}$

(This is about what you'd expect if $x^3 + ax + b$ acted "randomly": about half the values are squares, each of which has two square roots.)

Fact: $|E|$ can be computed "efficiently".

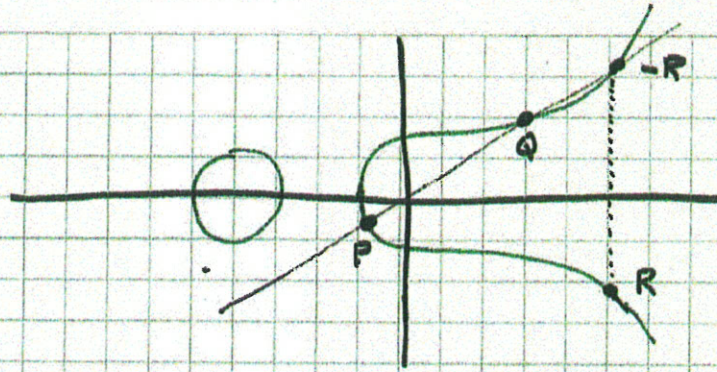
(Surprising) Fact: A binary operation (written additively as "+" can be defined on E s.t.

$(E, +)$ is a finite abelian group.

[∞ is the identity: $P + \infty = \infty + P = P$

[The inverse of (x, y) is $(x, -y)$ [also on curve].

[The inverse of ∞ is ∞ .



Let $P = (x_1, y_1)$ $Q = (x_2, y_2)$ $R = P + Q = (x_3, y_3)$.

Roughly: PQ defines a line.

Find "other point" on this line (call it $-R$)
return R as $P + Q$

Code: If $x_1 \neq x_2$: $m = (y_2 - y_1) / (x_2 - x_1)$ ("slope")

$$x_3 = m^2 - x_1 - x_2$$

$$y_3 = m(x_1 - x_3) - y_1$$

If $x_1 = x_2$ & $y_1 \neq y_2$: $P + Q = \infty$ (vertical line)

If $P = Q$ & $y_1 = 0$: $P + Q = \infty$ (vertical tangent)

If $P = Q$ & $y_1 \neq 0$: $m = (3x_1^2 + a) / 2y_1$ (tangent)

$$x_3 = m^2 - 2x_1$$

$$y_3 = m(x_1 - x_3) - y_1$$

Theorem: "+" is associative binary operation on E . (!)

Cor: $(E, +)$ is a finite abelian group.

Fact: $(E, +)$ may or may not be cyclic.

Fact: Can use other finite fields (e.g. $GF(2^k)$) instead of \mathbb{F}_p .

Why are elliptic curves interesting?

- The discrete logarithm problem seems to be quite hard (requiring $\approx |E|^{1/2}$ steps) for well-chosen E . (See "NIST standard curves") Thus, the groups can be smaller than \mathbb{Z}_p^* of the same security level. This yields both compactness and efficiency.
- Some elliptic curves admit "bilinear maps" enabling all sorts of really wonderful crypto operations. (More on this later.)

```
09:14:47 notes $ /Applications/sage/sage
Detected SAGE64 flag
Building Sage on OS X in 64-bit mode
```

```
-----
| Sage Version 4.6.2, Release Date: 2011-02-25
| Type notebook() for the GUI, and license() for information.
|-----
```

```
sage: # some experiments with elliptic curves with sage
sage: # First define a field mod 101
sage: F = Zmod(101)
sage: F
Ring of integers modulo 101
sage: # example of multiplication in F
sage: F(10)*F(11)
9
sage: # Define elliptic curve over F
sage: E = EllipticCurve(F,[0,1])
sage: E
Elliptic Curve defined by  $y^2 = x^3 + 1$  over Ring of integers modulo 101
sage: P = E.random_point()
sage: P
(96 : 49 : 1)
sage: # note coordinates are in projective form (X : Y : Z) representing
sage: # point  $x = X/Y$ ,  $y = Y/Z$ , with  $Z = 0$  for point at infinity.
sage: # get another point
sage: Q = E.random_point()
sage: Q
(29 : 94 : 1)
sage: P+Q
(21 : 77 : 1)
sage: # check commutativity
sage: Q+P
(21 : 77 : 1)
sage: # get third point
sage: R = E.random_point()
sage: R
(76 : 58 : -1)
sage: # check associativity
sage: P + (Q+R)
(53 : 2 : 1)
sage: (P+Q)+R
(53 : 2 : 1)
sage: # find size of this group
sage: E.order()
102
sage: # what are factors of 102?
sage: factor(102)
2 * 3 * 17
sage: # so possible orders of elements are 1,2,3,6,17,34,51,102
sage: P.order()
51
sage: Q.order()
51
sage: R.order()
51
sage: # none of P,Q,R are a generator (i.e. have order 102)
sage: # let's find one
sage: R = E.random_point()
```

```

sage: R.order()
102
sage: # bingo
sage: # what does identity look like?
sage: P-P
(0 : 1 : 0)
sage: I = P-P
sage: I
(0 : 1 : 0)
sage: I+P
(96 : 49 : 1)
sage: P+I
(96 : 49 : 1)
sage: -P
(96 : 52 : 1)
sage: # note that inverses just negate Y component, modulo 101
sage: # look at some small powers of generator R
sage: for i in range(15): print i, i*R
1....1
0 (0 : 1 : 0)
1 (72 : 85 : 1)
2 (15 : 89 : 1)
3 (9 : 86 : 1)
4 (84 : 21 : 1)
5 (52 : 44 : 1)
6 (87 : 61 : 1)
7 (90 : 65 : 1)
8 (35 : 31 : 1)
9 (10 : 71 : 1)
10 (18 : 51 : 1)
11 (93 : 14 : 1)
12 (4 : 41 : 1)
13 (38 : 39 : 1)
14 (76 : 58 : 1)
sage: # find discrete log of P, base R
sage: R.discrete_log(P)
80
sage: 80*R
(96 : 49 : 1)
sage: 80*R==P
True
sage: # find discrete log of Q, base R
sage: R.discrete_log(Q)
58
sage: # find elements of each possible order
sage: R.order()
102
sage: S = 2*R
sage: S.order()
51
sage: S = 3*R
sage: S.order()
34
sage: S = 6*R
sage: S.order()
17
sage: S = 17*R
sage: S.order()

```

```
6
sage: S = 34*R
sage: S.order()
3
sage: S = 51*R
sage: S.order()
2
sage: S
(100 : 0 : 1)
sage:
```