6.857 Computer and Network Security
Lecture 12

Admin: Project proposals due this week.

Project Ideas:

- Electronic auctions
- See: recent CACM article http://www.cs.ut.ee/~lipmaa/crypto/link/protocols/auctions.php (good high-level set of links)

Today:

- Group theory review
- Diffie-Hellman key exchange
- Five crypto groups: Zp*, Qp, Zn*, Qn, elliptic curves

Group Theory review:

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

- identity: $\exists$ identity 1 s.t. $(\forall a \in G)\ a \cdot 1 = 1 \cdot a = a$
- inverses: $(\forall a \in G)(\exists b \in G)\ a \cdot b = 1 \qquad (b = a^{-1})$
- associativity: $(\forall a,b,c \in G)\ a\cdot(b \cdot c) = (a \cdot b)\cdot c$
- commutativity: $(\forall a,b \in G)\ a \cdot b = b \cdot a$

order: Let $\text{order}(a) =$ least $u>0$ s.t. $a^u = 1$ (in G).

Theorem: In a finite abelian group of size t

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

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, $(\forall a \in G)\ [a \neq 1] \Rightarrow [a$ 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.

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

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

- Computing discrete logarithms (the DL problem) is commonly assumed to be hard/infeasible for well-chosen groups G. [E.g. 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 $p > 2^k$ then we can identify k-bit messages with the integers $1, 2, \dots, 2^k$ mod p (in Z_p^*).

In some groups this can be a little tricky.

TOPIC:

DATE:

FILE UNDER:

PAGE:

API for a group

① G ← create_group(...)

② G.identity()

③ product $x \cdot y$ [sometimes written as "+"]

↓

④ power x^k $x \in G,\ k \in \mathbb{Z}$ [$k \cdot x$]

↓

⑤ inverse x^{-1} [$-x$]

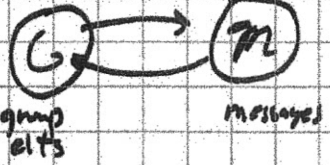
⑥ random elt G.random()

⑦ size G.order() $|G|$ [not always]

⑧ list elements G.elements() [not always]

⑨ represent msg: as elt of group G.rep(M) $0 \leq M < |G|$

& inverse G.unrep(x)



⑩ generator G.generator()

⑪ discrete log G.discretelog(g, y) $= x$ s.t. $g^x = y$

[usually not possible]

...

TOPIC:

DATE:

FILE UNDER:

PAGE: L14.10

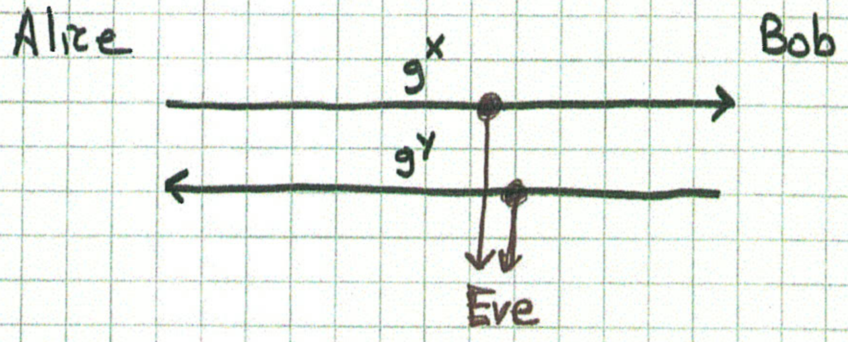
Diffie-Hellman key exchange (1976)

- How to establish shared secret in presence of eavesdropper? (Eve is passive - only listens)
- Precursor to true public-key

Merkle puzzles →

- Let G be a cyclic group with generator g. G and g are fixed & public.
- Alice chooses secret x randomly from $[0,..,|G|-1]$ Alice computes g^x as her "public key" (transient or permanent). g^x is a random element of G s.t. Alice knows discrete log.
- Bob similarly chooses secret y & computes "public key" g^y.
- Assumed difficulty of DL in G means even if Eve learns g^x or g^y she doesn't learn x or y.

- 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 ⇒ DH key exchange is secure
  (i.e. Eve doesn't learn K, except with negl. probability)
  Proof: Duh! (≈ 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.

① $Z_p^* = \{a : 1 \leq a < p\}$ where p is prime

Z_p^* is always cyclic.

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

② Q_p = quadratic residues (squares) mod prime p

$$= \{a^2 : 1 \leq a < p\}$$

$$\subsetneq Z_p^*$$

$|Q_p| = \frac{1}{2} |Z_p^*| = (p-1)/2$ ("half of Z_p^* are squares"

Q_p is cyclic: If $\langle g \rangle = Z_p^*$, then $\langle g^2 \rangle = Q_p$.

$Q_p = \{g^{2i} : 0 \leq i < (p-1)/2\}$ if $\langle g \rangle = Z_p^*$.

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 $a \notin \{1, p-1\}$.]

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

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

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

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

$\mathbb{Z}_n^* \approx \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ (Chinere remainder thm)

④ $Q_n = \{a^2 : 1 \leq a < n \ \& \ \gcd(a,n) = 1\}$

= "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

$|Q_n| = r \cdot s$

& 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.

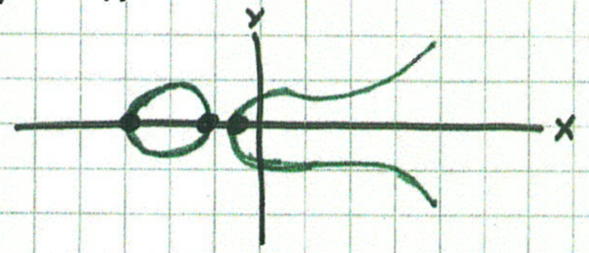
Let a, b be elements of $\mathbb{Z}_p$ such that

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

Consider equation (in variables x, y mod p)

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

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.)

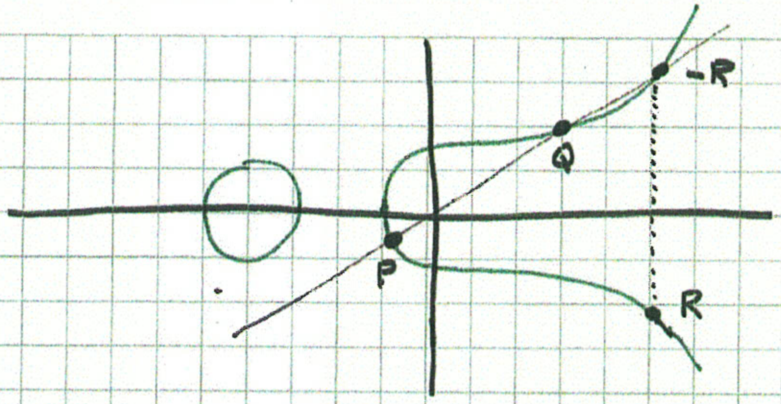
Fact: $|E|$ can be computed "efficiently".

(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 $P=Q$ & $y_1 \neq 0$: $m = 3x_1^2 + a$ (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 $\mathbb{F}_p$.

TOPIC:

DATE:

FILE UNDER:

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
....:
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 : 38 : 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:
```