Cryptanalysis

Lecture 7: Discrete Log Based Systems

John Manferdelli
jmanfer@microsoft.com
JohnManferdelli@hotmail.com

Public Key (Asymmetric) Cryptosystems

- An asymmetric cipher is a pair of key dependant maps, (E(PK,-),D(pK,-)), based on related keys (PK, pK).
- D(pK,(E(PK,x))=x, for all x.
- PK is called the public key. pK is called the private key.
- Given PK it is infeasible to compute pK and infeasible to compute x given y=E(PK,x).

Idea from Diffie, Hellman, Ellis, Cocks, Williamson. Diffie and Hellman, "New Directions in Cryptography", IEEE Trans on IT 11/1976. CESG work in 1/70-74.

Algorithm Timings

- Adding two m-bit numbers takes O(m) time.
- Multiplying two m-bit numbers takes $<O(m^2)$.
- Multiplying a 2m-bit number and reducing modulo and m-bit number takes $O(m^2)$.
- Computing (a, b) for a, b< n takes $O(\ln^2(n))$ time (i.e.- fast). This is Euclid’s Algorithm and it started Knuth, Euclid and everyone else off on computational complexity. If n has m bits this is $O(m^2)$.
- Testing an number n for primality takes $O(n^{c\lg(\lg(n))})=O(2^{cm\lg(m)})$.
- Best known factoring: $O(n^{c(\lg(n)^\wedge(1/3)(\lg(\lg(n))^\wedge(2/3))})=O(2^{cm(m^\wedge(1/3)(\lg(m)^\wedge(2/3))})$ [a lot longer].

Representing Large Integers

- Numbers are represented in base 2^{ws} where ws is the number of bits in the “standard” unsigned integer (e.g. – 32 on IA32, 64 on AMD-64)
- Each number has three components:
  - Sign
  - Size in 2^{ws} words
  - 2^{ws} words where $n= i[ws-1]2^{ws(size-1)} + \ldots + i[1]2^{ws} + i[0]$
  - Assembly is often used in inner loops to take advantage of special arithmetic instructions like “add with carry”

Classical Algorithms Speed

- For two numbers of size s_1 and s_2 (in bits)
  - Addition/Subtraction: $O(s_1)+ O(s_2)$ time and $\max(s_1, s_2)+1$ space
  - Multiplication/Squaring: $O(s_1) \times O(s_2)$ time and space (you can save roughly half the multiplies on squaring)
  - Division: $O(s_1) \times O(s_2)$ time and space
    - Uses heuristic for estimating iterative single digit divisor: less than 1 high after normalization
  - Extended GCD: $O(s_1) \times O(s_2)$
  - Modular versions use same time (plus time for one division by modulus) but smaller space
  - Modular Exponentiation ($a^e \pmod n$): $O((\text{size } e)(\text{size } n)^2)$ using repeated squaring
  - Solve simultaneous linear congruence's (using CRT): $O(m^2)$ x time to solve 1 where m = number of prime power factors of n

Primitive roots in F_p

- F_p*= F_p – {0} is the finite field with p elements with the zero element. It is a cyclic multiplicative group.
- Each element, □, that generates F_p*is called a primitive root and each such primitive root is the a zero of a primitive polynomial.
- There are □(p-1) such primitive roots.
- Example:
- p=193. □=5 is a primitive root so <□>= F_p*.
- There are □(192) such primitive roots.
- Since 192= 8 x 24= 2^6 x 3, there are 192 x 1/3 = 64.

Irreducibility polynomials in $F_p[x]$

- Is f(x) irreducible?

```
u(x)= x;
for(i=1; i<(m+1)/2; i++) {
   u(x)= u(x)^p (mod f(x));
   d(x)= gcd(u(x)-x, f(x));
   if(d(x)!=1)
      return “irreducible”;
}
```

Finding generators (Gauss)

- Find a generator, *g*, for F_p*, $n = (p-1) = p_1^{e1} p_2^{e2} \ldots p_k^{ek}$.

```
while () {
        choose a random g∈G
        for(i=1; i<=k; k++) {
           b= g^{n/pi}
           if (b==1)
           break;
           }
         if(i>k)
        return g
  }
```

- G has $\phi(n)$ generators. Using the lower bound for $\phi(n)$, the probability that g in line 2 is a generator is at least $1/(6 \ln \ln n)$

Discrete Log

- If $□= □^x$, then $L_□(□)=x$. $L_□()$ is the discrete log function.
- If $□= □^x$, then $L_□(□)=xL_□(□)$. $L_□(□_1□_2)= L_□(□_1) + L_□(□_2)$

- **Discrete Log Problem (DLP):** Given p, prime, □ □<□>=F_p*. □ (mod p), a, unknown, find $L_□(□)$.
- **Computational Diffie Hellman Problem (CDHP):** Given p, prime, <□>=F_p*. $□^a$ (mod p), $□^b$ (mod p), find $□^{ab}$ (mod p).
- Theorem: CDHP $\leq_P$ DLP. If the factorization of p-1 is known and □(p-1) is $O((\ln(p))^c)$ smooth then DLP and CDHP are equivalent.
- Why is this different from computing continuous logs?

- Moral: Exponentiation is a one way function.

El Gamal cryptosystem

- Alice, the private keyholder, picks a large prime, p, where p-1 also has large prime divisors (say, p= 2rq+1) and a generator, g, for F_p^*. $\langle g \rangle = F_p^*$. Alice also picks a random number, a (secret), and computes $A=g^a$ (mod p). Alice's public key is <A, g, p>.
- To send a message, m, Bob picks a random b (his secret) and computes $B= g^b$ (mod p). Bob transmits $(B, mA^b)= (B, C)$.
- Alice decodes the message by computing $CB^{-a}=m$.
- Without knowing a, an adversary has to solve the Computational Diffie Hellman Problem to get m.

- Note: b must be random and never reused!

Timing

- Finding g takes about $O(\lg(p)^3)$ operations, so does primality testing and raising g to the a power mod p.
- Encryption is also $O(\lg(p)^3)$ and so is decryption.
- Note that key generation is cheap but for safety, $p>w^2$, where w is the “computational power” of the adversary.

Attack on reused nonce

- Suppose Bob reuses b for two different messages m_1 and m_2.
- An adversary, Eve, can see <B, C_1> and <B, C_2> where $C_i = Bm_i \pmod p$.
- Suppose Eve discovers m_1.
- She can compute $m_2 = m_1 C_2 C_1^{-1} \pmod p$.

- Don’t reuse b’s!

El Gamal Example

- Alice chooses
  – $p=919$. $g=7$.
  – $a=111$, $A= 7^{111}= 461 \pmod{919}$.
  – Alice's Public key is <919, 7, 461>

- Bob wants to send $m=45$, picks $b= 29$.
  – $B=7^{29} =788 \pmod{919}$, $461^{29}= 902 \pmod{919}$,
  – $C= (45)(902)= 154 \pmod{919}$.
  – Bob transmits (788, 154).

- Alice computes $(788)^{-111}= 902^{-1} \pmod{919}$.
  – $(54)(902)+(-53)(919)=1$. $54= 902^{-1} \pmod{919}$
  – Calculates $m= (154)(54)=45 \pmod{919}$.

El Gamal Signature

- $<g>= F_q$*. A picks a random as in encryption.
- Signing: Signer picks k: 1 □k □p-2 with (k, p-1)= 1 and publishes g^k. k is secret.
- $Sig_K(M,k)= (t,d)$
  - $t= g^k$ (mod p)
  - $d=(M-gt)k^{-1}$ (mod p-1)
- $Ver_K(M,t,d)$ iff $g^{kt}t^d=g^M$ (mod p)

- Notes: It’s important that M is a hash otherwise there is an existential forgery attack. It’s important that k be different for every message otherwise adversary can solve for key.

DSA

- Alice
  - $2^{159}<q<2^{160}$, $2^{511+64t}<p<2^{512+64t}$, 1 □t □8, q|p-1
  - Select primitive root x (mod p); compute: $g=x^{(p-1)/q}$ (mod p)
  - Picks a random, 1 □a □q-1. A= g^a (mod p)
  - Public Key: (p, q, g, A). Private Key: a.
- Signature Generation
  - Pick random k, r= (g^k (mod p)) (mod q). Note : **k must be different for each signature**.
  - s= k^{-1}(h(M)+ar) (mod q). Signature is (r,s)
- Verification
  - u= s^{-1}h(x)(mod q), v= (rs^{-1})(mod q)
  - Is $g^u A^v$= r (mod p)?
- Advantages over straight El Gamal
  - Verification is more efficient (2 exponentiations rather than 3)
  - Exponent is 160 bits not 768

Baby Step Giant Step --- Shanks

- $g^x = y \pmod p$.
- $m \sim \sqrt{p}$.
- Compute g^{mj}, $0 \leq j < m$.
- Sort (j, g^{mj}) by second coordinate.
- Pick i at random, compute $yg^{-i} \pmod p$).
- If there is a match in the tables $yg^{-i} = g^{mj} \pmod p$).
- $x = mj+i$ is the discrete log.

Baby Step Giant Step Example

- p=193. $\lfloor\sqrt{(p)}\rfloor$=13. m= 14. □= 5. □=41.
- 2 x 193 + (-77) x 5 = 1, $□^{-1}$= 116. $□^{-14}$= 189 (mod 193).

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$□^{j}$	5	25	125	46	37	185	153	186	158	18	90	64	127	56
$□□^{-mj}$	26	77	78	74	90	26	89	30	73	94	10	153	160	132

- So $□□^{-(14 \times 5)}$= 90 = $□^{11}$ (mod 193).
- Thus □□ $□□□^{14x5+11}$□ $□□□^{81}$ (mod 193).
- L_5(41)= 193.

Discrete log Pollard □

- $x_{i+1} = f(x_i)$
  - $f(x_i) = \square x_i$, if $x_i \ \square \ S_1$.
  - $f(x_i) = x_i^2$, if $x_i \ \square \ S_2$.
  - $f(x_i) = \square x_i$, if $x_i \ \square \ S_3$.
- $x_i = \square^{a[i]}\square^{b[i]}$.
  - $a[i] = a[i]$, if $x_i \ \square \ S_1$.
  - $a[i] = 2a[i]$, if $x_i \ \square \ S_2$.
  - $a[i] = a[i]+1$, if $x_i \ \square \ S_3$.
  - $b[i] = b[i]+1$, if $x_i \ \square \ S_1$.
  - $b[i] = 2b[i]$, if $x_i \ \square \ S_2$.
  - $b[i] = b[i]$, if $x_i \ \square \ S_3$.
- $x_{2i} = x_i \rightarrow a_{2i} - a_i = L_{\square}(\square)\ (b_{2i} - b_i)$

Pollard □□example

- p=229, n=191, □=228, □=2. $L_2(228)=110$

i	x_i	a_i	b_i
1	228	0	1
2	279	0	2
3	92	0	4
4	184	1	4
5	205	1	5
6	14	1	6
7	28	2	6
8	256	2	7
9	152	2	8
10	304	3	8
11	372	3	9
12	121	6	18
13	12	6	19
14	144	12	38

i	x_{2i}	a_{2i}	b_{2i}
1	279	0	2
2	184	1	4
3	14	1	6
4	256	2	7
5	304	3	8
6	121	6	38
7	144	12	152
8	235	48	154
9	72	48	118
10	14	96	119
11	256	97	120
12	304	98	51
13	121	5	104
14	144	10	163

- $x_{14}= x_{28}$, $(b_{14}-b_{28})= 125 \pmod{191}$, $L_2(228)=125^{-1} (a_{28}-a_{14})= 110$.

Pohlig-Hellman

- $p\text{-}1= \square\ q_i^{r[i]}$.
- Solve $\square^{x}= y \pmod{p}$ for $x \pmod{q_i^{r[i]}}$ and use Chinese Remainder Theorem.
- $x= x_0 + x_1\, q + x_2\, q^2 + \ldots + x^{r[i]-1}\, q^{r[i]-1}$.
- $x\, (p\text{-}1)/q= x_0\, (p\text{-}1)/q + (p\text{-}1)\, (\ldots)$
- So $\square^{(p-1)/q}= \square^{x[0](p-1)/q}$. Solve for x_0.
- The put $\square=\square\square^{-x[0]}$ and solve $\square^{(p-1)/(q\ x\ q)} = \square^{x[1](p-1)/q}$.
- This costs $O(\square_{i=1}^{r}\ e_i(\lg(n)+ \sqrt{}\ q_i)$.

Pohlig-Hellman example

- p=251. □= 71, □=210, <□>=F_{251}*. n=250= 2 x 5^3.
- $L_{71}(210)$= 1 (mod 2).
- x= $x_0 + x_1\ 5 + x_2\ 5^2$.
- So $□^{n/5}$= 71^{20}. $□^{n/5}$= 210^{20}= 149.
  - x_0= $L_{20}(149)$=2.
  - x_1= 4
  - x_2= 2
- x= 2+ 4 x 5 + 2 x 25= 72 (mod 125)
- Applying CRT: $L_{71}(210)$= 197.

Index Calculus

- $g^x = y \pmod p$. B= $(p_1, p_2, \ldots, p_k)$.
- Precompute
  - $g^{x_j} = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$
  - $x_j = a_{1j} \log_g (p_1) + a_{2j} \log_g (p_2) + \ldots + a_{kj} \log_g (p_k)$
  - If you get enough of these, you can solve for the $\log_g(p_i)$
- Solve
  - Pick s at random and compute $y\, g^s = p_1^{c_1} p_2^{c_2} \ldots p_k^{c_k}$ then
  - $\log_g (y)+s = c_1 \log_g (p_1) + c_2 \log_g (p_2) + \ldots + c_k \log_g (p_k)$

- This takes $O(e^{(1+\ln(p)\ln(\ln(p))})$ time.

- LaMacchia and Odlyzko used Gaussian integer index calculus variant to attack discrete log.

Index Calculus Example

- p=229. □=6. <□>= F_{229}*. n=228. □=13. S={2,3,5,7,11}.
- Step 1
  1. 6^{100} (mod 229)= 180= $2^2 \times 3^2 \times 5^1 \times 7^0 \times 11^0$.
  2. 6^{18} (mod 229)= 176= $2^4 \times 3^0 \times 5^0 \times 7^0 \times 11^1$.
  3. 6^{12} (mod 229)= 165= $2^0 \times 3^1 \times 5^1 \times 7^0 \times 11^0$.
  4. 6^{62} (mod 229)= 154= $2^1 \times 3^0 \times 5^0 \times 7^1 \times 11^1$.
  5. 6^{143} (mod 229)= 198= $2^1 \times 3^2 \times 5^0 \times 7^0 \times 11^1$.
  6. 6^{206} (mod 229)= 210= $2^1 \times 3^1 \times 5^1 \times 7^1 \times 11^0$.
- Taking $L_{□}$() of both sides, we get:
  1. 100= 2 $L_{□}$(2)+2$L_{□}$(3)+$L_{□}$(5) (mod 228)
  2. 18= 4$L_{□}$(2)+$L_{□}$(11) (mod 228)
  3. 12= $L_{□}$(3)+$L_{□}$(5)+$L_{□}$(11) (mod 228)
  4. 62= $L_{□}$(2)+$L_{□}$(7)+$L_{□}$(11) (mod 228)
  5. 143=$L_{□}$(2)+$L_{□}$(3)+$L_{□}$(11) (mod 228)
  6. 206= $L_{□}$(2)+$L_{□}$(3)+$L_{□}$(5)+$L_{□}$(11) (mod 228)

Index Calculus example - continued

- Review
  - $p=229$. □$=6$. $<$□$>= F_{229}^*$. $n=228$. Solving, we got:
  - $L_{□}(2)= 21 \pmod{228}$
  - $L_{□}(3)= 208 \pmod{228}$
  - $L_{□}(5) = 98 \pmod{228}$
  - $L_{□}(7)= 107 \pmod{228}$
  - $L_{□}(11)= 162 \pmod{228}$
- Step 2:
  - Recall □$=13$. Pick $k=77$
  - $13 \times 6^{77}= 147 = 3 \times 7^2 \pmod{229}$
  - $L_6(13)= (L_6(3)+2L_6(7)-77)= 117 \pmod{228}$

Diffie Hellman key exchange

Alice

Bob

A1: s= min(p size), N_a in {0, … 2^{256}-1}

$\xrightarrow{s,N_a}$

$\xleftarrow{(p,q,g),\ X=g^x,\ Auth_B}$

B1: Choose (p,q,g), x in {0, … 2^{256}-1}

A2: Check (p,q,g) X, $Auth_B$, pick y in {0,…q-1}

$\xrightarrow{Y= g^y,\ Auth_A}$

B2: Check Y, $Auth_A$

K= X^y

K= Y^x

DH key exchange example

- p=3547, g=2.
- Alice: a= 7.
- Bob: b=17.
- A→B_1: A=128 (=2^7), Sign_A(SHA-2(128 || r_1))
- B→A_1: B=3380(=2^{17}), Sign_B(SHA-2(3380 || r_2))
- K= 128^{17}=3380^7= 362.

Square roots mod p -- general comments

- We want x: $x^2 = a \pmod{p}$.
- Remember, we can check to see if a is a quadratic residue by computing (a/p).
- If we know a generator of F_p^*, g and $g^n=a$, then $g^{n/2}=x \pmod{p}$.
- Of course, this requires solving the discrete log problem so it does not offer a practical computational method.
- Since there is no order relation, approximations (e.g.- Newton’s method) don’t help much.

- Reference: Cohn, Computational Number Theory.

Square roots mod p --- simple cases

- We want x: $x^2 = a \pmod{p}$. First check $(a/p)=1$.
- $p = 3 \pmod{4}$:
  - $x = a^{(p+1)/4} \pmod{p}$
  - Example: $x^2 = 7 \pmod{31}$, $x = 7^8 \pmod{31} = 10$. $100 = 7 \pmod{31}$.
- $p = 5 \pmod{8}$
  - $b = a^{(p-1)/4} = \pm 1 \pmod{p}$.
  - If $b=1$, $x = a^{(p+3)/8} \pmod{p}$.
  - If $b = -1$, $x = (2a)(4a)^{(p-5)/8} \pmod{p}$.
  - Example 1: $p=13$. $a = 9$. $b = 9^3 = 1 \pmod{p}$. $x = 9^2 = 3$ (surprise!).
  - Example 2: $p=29$. $a = 6$. $6^7 = -1 \pmod{p}$. $x = (12)(24)^3 = 8 \pmod{29}$. $8^2 = 6 \pmod{29}$.
- This leaves the hard case, $p=1 \pmod{8}$.

General case - Tonelli-Shanks

- We want x: $x^2 = a \pmod p$
- $p-1=2^e \times q$, q, odd.

Square-Root(a)

```
1.  Choose n: $(n/p)= -1$; $z= n^q \pmod p$; $Q=(q-1)/2$.
2.  $y=z$; $r=e$;  $x=a^Q \pmod p$; $b=ax^2 \pmod p$; $x= ax \pmod p$;
3.  // Now if $R=2^{r-1}$, $ab=x^2$, $y^R=-1$, $b^R=1$;
    if(b==1)
        return(x);
    $M=2^m$;  for smallest $m>0$: $b^M= 1 \pmod p$
    if(m=r)
        return “non-residue”
4.  $TT= 2^{r-m-1}$; $t= y^{TT} \pmod p$; $y= t^2 \pmod p$; r=m; x=xt; b=by; goto 3;
```

Tonelli-Shanks example

- We want x: x^2= a (mod p). p=41, a=5, g=7.
- p-1=2^3 x 5. Note 6^{20}= -1 (41) so 6 is a non-residue.
- a= 5; n=6; z= 6^5 = 27 (mod 41).

Step	m	t	y	r	x	b
0	3		27	3	2	9
1	2	2	32	2	13	1

- x=13. 13^2 (mod 41)= 5.

Berlekamp factorization

- $f(x)= \square_{i=1}^{t} f_i(x)$ over F_p, $\deg(f(x))=n$. $f_i(x)$ irreducible.

```
F={f(x)};
for(i=1; i<n;i++)
    $x^{iq} = = \square_{j=0}^{n-1} q_{ij} x^j$  (mod f(x)), $q_{ij} \square F_p$.
Find basis $<v_1, \ldots, v_t>$ of null space of $(Q-I_n)$;
// $w= w_0, \ldots, w_{n-1}$. $w(x) = w_0 + w_1 x + \ldots + w_{n-1} x^{n-1}$
for(i=1; $i \leq t$;i++) {
    for (h(x) $\square$ F, deg (h)>1;) {
        Compute $(h(x), v_i(x)-\square)$, $\square \square F_p$;
        Replace h(x) in F with these;
    }
return (F);
```

- $O(n^3+tpn^2)$, t= # irreducible factors. Can be reduced to $O(n^3+t \lg(p)n^2)$.

Berlekamp factorization example

- Factor x^7-1 over F^2.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{pmatrix} = \begin{pmatrix} 1 \\ x^2 \\ x^4 \\ x^6 \\ x^1 \\ x^3 \\ x^5 \end{pmatrix}$$

- Adding I and solving get:
  - 1
  - $x^4+x^2+x = x(x^3+x+1)$
  - $x^6+x^5+x^3= x^3(x^3+x^2+1)$
  - Dividing into x^7-1, we get:
  - $(x+1)$

End