2023

Computational Number Theory & Cryptography

B.Sc. IN CYBER SECURITY

Madhya Pradesh Bho□
Open University

PGDCS-103

Computational Number Theory and Cryptography

Block-1

Block-2

Block-3

Block-4:

o

Block-1

Unit 1: Computational Complexity

1

1.1 COMPLEXITY OF COMPUTATION & COMPLEXITYCLASSES

We will restrict ourselves to two types of Complexities:

Time Complexity

Space Complexity.

By time/space complexity we mean the time/space as a function of input size required by an algorithm to solve a problem.

Problems are categorized into 2 types

(i) DecisionProblem

(ii) OptimizationProblem.

For the purpose of present discussion we will concentrate on decision problems. This is defined as follows.

Definition 1: Let $\sum$ be a set of alphabets and let $L \subseteq \sum^*$ be a language. Given a string $x \in L$ or $x \notin L$ is decision problem.

Notation: Let p() denote a polynomial function.

We will define some complexity classes:

Definition 2: The class **P** comprises of all languages $L \subseteq \sum^*$ such that there exist a polynomial time algorithm A to decide L. In other words given a string $x \in \sum^*$ the algorithim A can determine in time $p(|x|)$ whether $x \in L$ or $x \notin L$.

Definition3: The class **NP** comprises of all language $L \subseteq \sum^*$ such that given a string $x \in L$ a proof of the membership of $x \in L$ can be found and verified in time $p(|x|)$.

Definition 4: The class **Co-NP** comprises of all language $L \subseteq \sum^*$ such that $\sum^* - L \in$ **NP**.

Note: We can easily verify **CO-P=P** and thus P $\subseteq$**NP** $\cap$ **CO-NP**.

Definition 5: The class PSPACE comprise of all languages $L \subseteq \sum^*$ such that there exists an algorithm A that uses polynomial working space with respect to the input size to decide L. In other words given a string $x \in \sum^*$ the algorithim A can determine using space, i.e., $p(|x|)$ whether $x \in L$ or $x \notin L$.

We will state without proof the following result that follows from Savitch'stheorem:
PSPACE=NSPACE

Polynomial Time reducibility:

A language $L_1 \subseteq \sum^*$ is said to be polynomial time reducible to $L_2 \subseteq \sum^*$ if there is a polynomial time computable function $f(\ )$ such that $\forall x \in \sum^*$, $x \in L_1$ if and only if $f(x) \in L_2$. We denote this by $L_1 \propto_p L_2$. we can clearly observe that polynomial time reductions are transitive.

Completeness:

A language $L \subseteq \sum^*$ is said to be complete with respect to any complexity class C if all problems in that complexity class C can be reduced to L.Thus we formally define the notion of NP-Completeness.

Definition 6:

A language $L \subseteq \sum^*$ is said to be **NP-Complete** if

(i) $L \in$ **NP**

(ii) $\forall L' \in$ NP, we have $L' \propto_p L$.

The above definition is not very suitable to prove a language L to be **NP-Complete** since we have infinitely many language in the class **NP** to be reduced to L. Hence for providing NP-Completeness we resort to the following equivalentdefinition.

Definition 7:

A language L is said to be **NP-Complete** if

(i) $L \in$ **NP**

(ii) $\exists\ L' \subseteq \sum^*$ that is **NP-Complete** and $L' \propto_p L$.

The previous two definitions are equivalent since:

L' is **NP-Complete** $\Rightarrow \forall L'' \in$ **NP**, $L'' \propto_p L'$ (from **definition 6**) $\Rightarrow \forall L'' \in$ **NP**, $L'' \propto_p L' \propto_p L \Rightarrow \forall L'' \in$ **NP**, $L'' \propto_p L$ (from the transitivity of polynomial time reductions.) $\Rightarrow$ L is **NP-Complete.**

Only catch in this approach is to prove the first problem to be NP-Complete for which we usually take as ***SATISFIABILITY*** problem.

1.2 ENCODINGSCHEME

In all the definition of computational complexity we assume the input string x is represented using some reasonable encoding scheme.

Input size will usually refer to the numbers of components of an instance. For example when we

consider the problem of sorting the input size usually refers to the number of data items to be sorted ignoring the fact each item would take more than 1 bit to represent on a computer

But when we talk about primarily testing, i.e., to test whether a given integer n is prime or composite the simple algorithm to test for all factors from 2,3,……, $\lfloor \sqrt{n} \rfloor$ is considered exponentialsincetheinputsizeI(n)is $\beta = log_2^n$ bitsandthetimecomplexityisO($\sqrt{n}$),i.e., O($2^{\frac{1}{2}\beta}$).

Again if n is represented in unary the same algorithm would be considered polynomial. For number theoretic algorithms used for cryptography we usually deal with large precision numbers. So while analyzing the time complexity of the algorithm we will consider the size of the operands under binary encoding as the input size. We will analyze most of our programs estimating the number of arithmetic operations as function of input size β. While converting this complexity to the number of bit operations we have to consider the time complexities of addition, subtraction, multiplication &division.

Addition & subtraction:

Clearly addition and subtraction of two β bit numbers can be carried out using O(β) bit operations.

Multiplication:

Let X and Y be two ß bit numbers

Let $X = 2^{\beta/2} X_1 + X_2$

$Y = 2^{\beta/2} Y_1 + Y_2$....

Each $X_1, X_2, Y_1 Y_2$ is a $\beta/2$ bit long numbers

Then $X \times Y = 2^{\beta} X_1 Y_1 + 2^{\beta/2} (X_1 Y_2 + X_2 Y_1) + X_2 Y_2$

∴ Thus the time complexity of the above multiplication

$T(\beta) = 4T(\beta/2) + C\beta$ → Time for addition

↓

4 multiplications to Compute $X_1 Y_1$, $X_1 Y_2$,

. $X_2 Y_1$ & $X_2 Y_2$

Unit 2: GCD Computation

2

2.1 ELEMENTARYNUMBER-THEORY

Brief review of notions from elementary number theory concerning the set

Z = {..., -2, -1, 0, 1, 2...} of integers and

N = {0, 1, 2 , ...} of natural numbers.

Z_n= {0, 1, 2… n-1}

Z_n^+= {1, 2… n-1.}

Common divisors and greatest common divisors (GCD):

Let a, b ∈**Z**

$d \in \mathbf{Z} \wedge d \mid a \wedge d \mid b \Rightarrow d \mid ax + by$ [$x, y \in \mathbf{Z}$]

Let d = gcd (a,b)

d' | a Λ d' | b ⇒ d' | d [d′is common divisor of a and b]

The following are elementary properties of the gcd function:

gcd(a , b) = gcd(b , a)

gcd(a , b) = gcd(- a , b)

gcd(a , b) = gcd(| a |, | b |)

gcd(a , 0) = | a |

gcd(a , ka) = | a | for any $k \in \mathbf{Z}$.

Theorem 1

If a and b are any integers then gcd(a,b) is the smallest positive element of the set {$ax + by : x, y \in \mathbf{Z}$ }

Proof:

Let s be the smallest positive element of the set:{ $ax + by : x , y \in \mathbf{Z}$}

Let $q = \lfloor a / s \rfloor$ and $s = ax + by$

$a \bmod s = a - qs = a - q (ax + by) = a (1 - qx) + b (- qy)$

$a \bmod s < s$ and $a \bmod s$ is a linear combination of a and b . Thus $a \bmod s = 0 \Rightarrow s \mid a$

Using analogous reasoning we can show s | b. Thus $s \leq \gcd(a,b)$.

Let $d = \gcd(a,b) \Rightarrow d \mid a$ and $d \mid b$. Thus $d \mid$ s and $s > 0 \Rightarrow d \leq s$. We have shown before $d \geq s$ and

thus we have established that d=s.

Corollary 1:

For any integers *a* and *b*, if $d \mid a$ and $d \mid b$ then $d \mid \gcd(a, b)$.

Relatively prime integers

Two integers a, b are said to be relatively prime if their only common divisor is 1, that is, if $\gcd(a, b) = 1$.

Theorem 2

For any integers a, b, and p, if both gcd(a, p) = 1 and gcd(b, p) = 1, then gcd(ab, p) = 1.

Proof :

gcd(a, p) = 1 $\Rightarrow \exists$ x, y $\in \mathbf{Z}$ such that ax + py = 1 gcd(b,

p) = 1 $\Rightarrow \exists$ x′ y′ $\in \mathbf{Z}$ such that bx′ + py′ = 1

Multiplying these equations and rearranging, we have

$ab(x\,x') + p(ybx' + y'ax + pyy') = 1.$

Thus linear combination of a, b and p is equal to 1

Thus we have gcd (ab, p) = 1

Theorem 3

For all primes p and all integers a, b if p | ab ⇒ p | a or p | b .

Proof:

Assume otherwise, i.e., p | a and p | b. Since p is prime only 2 factors are there for p i.e. 1 & p.

Therefore gcd(a, p) =1 and gcd(b, p) =1 then gcd (ab, p)=1 ⇒ p |ab. □

Unique factorization

A composite integer a can be written in exactly one way as a product of the form:

a = $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$

Where pi‘s are primes $\forall i \in (1..k)$ such that $p_1 < p_2 < p_3 \text{--------} < p_k$

and $e_i \in \mathbf{Z}^+ (i = 1,2, \text{-------} k)$

Theorem 4 (GCD Recursion theorem)

For any non negative integer a and positive integer b gcd (a, b)= gcd (b, a mod b)

Proof:

We will show gcd (b, a mod b) | gcd (a, b) and gcd (a, b) | gcd (b, a mod b). Let d = gcd (b, a mod b). Thus d | b and d | (a mod b).

Now a = $\left\lfloor \frac{a}{b} \right\rfloor$ b + a mod b. Hence a is a linear combination of b and a mod b and so d | a. Therefore d | a and d | b ⇒ d | gcd (a, b) from **Corollary1.**

Let d = gcd (a, b) ⇒ d | a and d | b. Now a mod b = a – $\left\lfloor \frac{a}{b} \right\rfloor$ b and that implies a mod b is a linear combination of a and b. Thus d | (a mod b) ⇒ d | b and d | (a mod b) ⇒ d | gcd (b , a mod b) from **Corollary 1**.

2.2 EUCLID'SALGORITHM

EUCLID (*a, b*)

1. if*b=0*

2. then return(*a*)

3. else return(EUCLID(*b, a* mod*b*))

Lemma: If a > b ≥ 1 and the invocation EUCLID(a, b) performs k ≥ 1 calls

then $a \geq F_{k+2}$ and $b \geq F_{k+1}$

Proof: (By induction)

Basis: Let *k*=1, we know *a* > *b* ≥ 1

⇒*b* ≥ F_2 = 1 (here *k*+1=2)

Since *a* > *b* ⇒*a* ≥ 2 ⇒*a* ≥ F_3 (here*k*+2=3)

If *a* >*b* initially then this property *a* >*b* is maintained at each recursive invocation in EUCLID (a*, b)* algorithm, since *b* >*a* mod *b*always.

NOTE: Since *a* mod *b* <*b* ⇒ The invariant 1st argument > 2nd argument of EUCLID's algorithm is maintained during each iteration.

Inductive Hypothesis: Assume the result holds for # of invocations ≤ *k* -1

Inductive proof: Let EUCLID (*a* , *b*) makes *k* invocations

⇒ EUCLID (*b* , *a* mod *b*) makes (*k* -1) invocation

From our inductive hypothesis:

b ≥ $F_{(k-1)+2}$, *a* mod *b* ≥ $F_{(k-1)+1}$

Therefore $b \geq F_{k+1}$, $a \bmod b \geq F_k$

We know, $a = \left\lfloor \frac{a}{b} \right\rfloor * b + a \bmod b$ (where $\lfloor a \rfloor$ = Floor (a))

$\left\lfloor \frac{a}{b} \right\rfloor \geq 1 \Rightarrow a \geq b + a \bmod b$.

Since $a \bmod b \geq F_k$ we have $a \geq F_{k+1} + F_k \Rightarrow a \geq F_{k+2}$.

Lame's Theorem: For any integer $k \geq 1$ if a $> b \geq 1$ and if $b < F_{k+1}$ then EUCLID (a, b) makes fewer than k recursive calls gcd (F_{k+1}, F_k) = gcd (F_k, F_{k-1}) = … = gcd (1,0) = 1

Therefore # of recursive invocation = k-1

This shows that the bound k-1 is tight.

$F_k / F_{k-1} \approx \Phi$[Golden Ratio $\Phi = \frac{1+\sqrt{5}}{2}$]

To represent F_k, # of bits required =k

Therefore for two β bit numbers running time complexity of EUCLID is O (β)

EXTENDED-EUCLID ALGORITHM

Goal: Given 2 integers a and b compute integers x and y such that gcd (a, b) = ax + by.

EXTENDED-EUCLID (a, b)

1. if b =0
2. then return (a, 1,0)
3. (d', x', y') ← EXTENDED-EUCLID (b, a modb)
4. (d, x, y) ← (d', y', x'– $\left\lfloor \frac{a}{b} \right\rfloor$y')
5. return (d, x, y)

For a=99 and b =78 the following table illustrates the values of variables d, x, y at different levels of recursion for the algorithm EXTENDED-EUCLID(99, 78). We can easily verify that gcd(99, 78) = 3 = -11(99) + 14(78).

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	--	3	1	0

The correctness of the algorithm is established from the following inductive argument.

Basis: Let d denote gcd(a, b). When EUCLID terminates b = 0 and d = a ⇒ x = 1, y = 0. Thus the arguments returned by EXTENDED-EUCLID iscorrect.

Inductive Hypothesis: Assume the values d', x', y' returned by EXTENDED-EUCLID(b, a mod b) is correct.

Induction Step: We have to show EXTENDED-EUCLID(a, b) correctly computes d, x, y.

d'= x'b + y' (a mod b) = x'b + y' (a - $\lfloor \frac{a}{b} \rfloor$b) = y'a + (x'- $\lfloor \frac{a}{b} \rfloor$y')b =d

⇒ x = y' and y = x'- $\lfloor \frac{a}{b} \rfloor$y'.

Reference:

1. Introduction to Algorithms , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India* .

Unit 3: Finite Groups 3

3.1 MODULAR ARITHMETICGROUPS

A ***group*** (S , $\oplus$) is a set S together with a binary operation $\oplus$ defined on S for which the following properties hold.

1. **Closure:** For all a , $b \in S$, we have $a \oplus b \in S$.
2. **Identity:** There is an element $e \in S$, called the ***identity*** of the group, such that $e \oplus a = a \oplus e = a$, for all $a \in S$.
3. **Associativity:** For all a , b , $c \in S$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
4. **Inverses:** For each $a \in S$, there exists a unique element $b \in S$, called the ***inverse*** of a , such that $a \oplus b = b \oplus a = e$.

As an example, consider the familiar group (**Z** , +) of the integers **Z** under the operation of addition: 0 is the identity, and the inverse of a is $-a$.

Abelian group :

If a group (S , $\oplus$) satisfies the ***commutative law*** $a \oplus b = b \oplus a$, for all a , $b \in S$, then it is an ***abelian group*** .

The groups defined by modular addition and multiplication

First we define the congruence notation ≡ as follows:

If a , $b \in \mathbf{Z}$ then we say $a \equiv b$ modulo n if $\exists p$, q , $r \in \mathbf{Z}$ such that $a = pn + r$ and $b = qn + r$.

We will denote a mod n as $[a]_n$

We can form two finite abelian groups by using addition and multiplication modulo n , where n is a positive integer. These groups are based on the equivalence classes of the integers modulo n

$a \equiv a'$ (mod n) and $b \equiv b'$ (mod n), then

$a + b \equiv a' + b'$ (mod n),

$ab \equiv a' b'$ (mod n) .

Thus, we define addition and multiplication modulo n , denoted $+_n$ and $*_n$, as follows:

$[a]_n +_n [b]_n = [a + b]_n$ (addition modulo n)

[a] $_n$ * $_n$ **[b]** $_n$ **= [a *b]** $_n$(multiplicative modulo n)

Using this definition of addition modulo n , we define the ***additive group modulo n*** as ($Z_{n,} +_n$). The size of the additive group modulo n is $|Z_n| = n$. Modular addition over the group ($Z_{6,} +_6$) is defined as follows:

Z_6	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Closure: If a $\in Z_n$ and b $\in Z_n$ then from the definition of addition modulo n a $+_n$ b = $[a + b]_n \in Z_n$

Identity: 0 is the identity element of

Z_nInverse: Inverse of [a] $_n$ is [-a] $_n \equiv$ [n-

a] $_n$

Using this definition of multiplication modulo n , we define the ***multiplicative group modulo n*** as($\mathbf{Z}^*_{n,} *_n$) where $\mathbf{Z}^*_n = \{[a] \varepsilon_n \mathbf{Z} \mid gcd(a, n)=1\}$. For e.g. when n=15,

$\mathbf{Z}^*_{15}$= {1, 2, 4, 7, 8, 11, 13, 14}. Modular multiplication over the group($\mathbf{Z}^*_{15,} *_{15}$) is defined as follows:

Z_n	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Identity: [1] $_n$

Inverse: Since gcd(a , n)=1 for every $a \in \mathbf{Z}^*_n$ from **Extended-Euclid** (a , n) we obtain x and y

such that $ax + ny = 1 \Rightarrow ax \equiv 1 \bmod n \Rightarrow x$ is the inverse of a .

Clearly both $+_n$ and $*_n$ are associative and commutative. Thus we have established the following theorem:

Theorem 1: Both $(\mathbf{Z}_n, +_n)$ and $(\mathbf{Z}^*_n, *_n)$ form finite Abelian groups.

$|\mathbf{Z}^*_n| = \Phi(n)$ where $\Phi(n)$ is the **Euler *phi* function** .

From ***unique factorization*** **theorem** n can be expressed in terms of prime factors as follows:

$n = p_1^{e1} p_2^{e2} \ldots p_k^{ek}$

$$\Phi(n) = n \prod_{i=1}^{k} (1-1/p_i)$$

In our example $n = 15$

$15 = 3*5$

$\Phi(15) = 15(1-1/3)(1-1/5) = 8$

For $n = 45 = 3^2 * 5$ we have $\Phi(45) = 45\ (1-1/3)(1-1/5) = 24$. Thus the group $(\mathbf{Z}^*_{45}, *_{45})$ contains $|\mathbf{Z}^*_{45}| = 24$ elements.

3.2 SUBGROUPS

Subgroups and its Properties :

Lecture no 4.

Let $(G, \oplus)$ be a group and $H \subseteq G$ is a subgroup if

1. H is closed.
2. $\forall a \in H$, $a^{-1} \in H$.

Proof: To show H is a group ∃

1. Closure [Follows from 1st condition]
2. Associativity [Follows from associativity of G]
3. Inverse $a \in H$, $a^{-1} \in H$ [2nd condition]

$a \oplus a^{-1} \in H$ [1st condition]

$\Rightarrow e \in H$

Theorem 1 : A non empty closed subset of a finite group is always a subgroup.

Proof : Let $(G, \oplus)$ be a finite group & H be a non empty closed subset of G. Pick an element $a \in H$ & generate the sequence a , a^2 , a^3 , ... where $a^2 = a \oplus a$, $a^3 = a^2 \oplus a$ and so on.

This is an infinite sequence all whose members belong to finite subset H and hence all elements in the sequence cannot be distinct. Thus there must be at least 2 elements that are identical.

$a^r = a^s \ (r \neq s)$

$\Rightarrow a^{r-s} = e$

$\Rightarrow a^{-1} = a^{r-s-1} \in H$

Thus H is a subgroup from our definition.

Definition 1: Let (G, $\oplus$) be a group and H is a subgroup of G. Between two elements a , b $\in$ G we define a congruence relation as follows:

a ≡ b mod H if a $\oplus$ b $^{-1} \in$ H

Lemma 1: Congruence relation is an equivalence relation.

Proof:

Reflexive:

We have to show a ≡ a mod H for all a $\in$ G

From the definition of congruence relation a $\oplus$ a $^{-1}$ = e $\in$ H $\Rightarrow$ a ≡ a mod H.

Symmetric:

Let a , b $\in$ G. Since $a \equiv b$ mod H

$a \oplus b^{-1} \in H$

$\Rightarrow (a \oplus b^{-1})^{-1} \in H$ [Since H is a subgroup]

$\Rightarrow (b^{-1})^{-1} \oplus a^{-1} \in H$

$\Rightarrow b \equiv a$ mod H

Transitive:

Let a , b , c $\in$ G Λ $a = b$ mod H Λ $b \equiv c$ mod H

$a \oplus b^{-1} \in H \ \Lambda \ b \oplus c^{-1} \in H$ So

$a \oplus b^{-1} \oplus b \oplus c^{-1} \in H$

$\Rightarrow a \oplus c^{-1} \in H$

$\Rightarrow a \equiv c$ modH

Cosets: Let (G, $\oplus$) be a group and H is a subgroup of G .Pick an element a belonging to G.

Let H a = { $h \oplus a \mid h \in H$} be Right coset

Let a H= { $a \oplus h \mid h \in H$} be Left coset

Lemma 2: H a = { $x \mid x \equiv a$ mod H} $\forall a \in$ G

Proof: Let [a] = { $x \mid x \equiv a$ mod H}. We have to show H a =[a].

To prove H a $\subseteq$ [a] let us pick an element $h \oplus a \in$ H a . Thus $\forall h \in$ H $a \oplus (h \oplus a)^{-1} = a \oplus a^{-1} \oplus h^{-1} =$ h $^{-1} \in$ H [Since H is a subgroup].

$\Rightarrow$ h $\oplus$ a $\in$ [a]

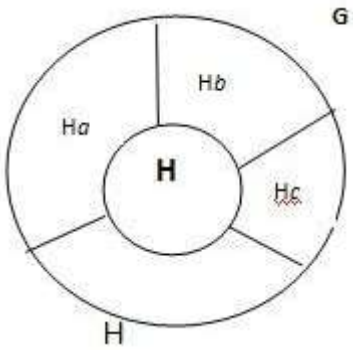
To prove that [a] $\subseteq$ H a let us pick an element $x \in$ [a]

$\Rightarrow a \oplus x^{-1} \in$ H

$\Rightarrow (a \oplus x^{-1})^{-1} \in$ H

$\Rightarrow x \oplus a^{-1} \in$ H

Therefore for some $h \in$ H we have $x = h \oplus a$. Thus $x \in$ H a and [a] $\subseteq$ Ha.



Lemma 3: There is a one to one correspondence between any 2 right cosets of G.

Proof: To establish one to one correspondence between two sets X and Y we have to exhibit a mapping f : $X \rightarrow Y$ such that $\forall a_1 , a_2 \in X , a_1 = a_2$ if and only if $f(a_1) = f(a_2)$ where $f(a_1), f(a_2) \in Y$. Also f has to be onto. As in our case we have if $h_1 \oplus a = h_2 \oplus a$ then $h_1 = h_2$ and thus $h_1 \oplus b = h_2 \oplus b$. The function is also onto because for every element $h \oplus b$ in the range there is an inverse element $h \oplus a$ *in thedomain.*

Theorem 2 [Lagrange]: Let (G, $\oplus$) be a finite group and H is a subgroup of G. Then o(H) | o(G).

Proof: Notation o(S) = |S|

Let k be the number of right cosets. Thus k * o(H) = o(G) and o(H) | o(G).

Order of an element

Let (G, $\oplus$) be a finite group. Let $a \in$ G. Then **order** (a) is defined as the smallest positive integer t such that $a^{(t)} =$ e. [$a^t = a \oplus a \oplus a$ ... $\oplus$ a t times].

Theorem 3 : For any finite group (G,*) and any $a \in$ G the order of the element is equal to the size of the subgroup it generates i.e., ord (a) =|<a>|.

Proof: <a > = e , a , a^2 , a^3 .....

Let t = **ord** (a). So $a^{(t)} = e$.

$\Rightarrow a^{(t)} * a^{(k)}$

$\Rightarrow e * a^{(k)} = a^{(k)}$

If $j > t$ $\exists i < j$ such that $a^{(i)} = a^{(j)}$

Thus no new elements are generated beyond $a^{(t)}$. Hence $|<a>| \leq t$.

Now we have to show that $|<a>| \geq t$ by proving all elements in $<a> = \{ a^1 , a^2 ,\ldots.., a^t\}$ are distinct. Assume otherwise $\Rightarrow \exists 1 \leq i < j \leq t$ suchthat $a^i = a^j$. Let t be $j + k$.Hence $a^{i+k} = a^{j+k} = a^{(t)} = e \Rightarrow a^{i+(t-j)} = e$ andweknow that $i + t - j < t$. Thuswearriveatacontradiction since t = ord (a) is the smallest power to which a has to be raised to become identity. Thus our assumption $\exists 1 \leq i < j \leq t$ such that $a^i = a^j$ is incorrect . Therefore, each element of the sequence $a^{(1)}$, $a^{(2)}$, ..., $a^{(t)}$ is distinct, and $|<a>| \geq t$. Thus we conclude that ord (a) = $|<a>|$ **Corollary 1:**

Let (G, $\oplus$) be a finite group with identity e then for all $a \in G$ *we have* $a^{ord(G)} = e$.

Proof : Consider the subgroup $<a>$ of G. From Theorem 3 $|<a>| = ord(a)$. From Lagrange's theorem $ord (a) \mid ord(G)$. Let $ord (G)$ *be* $k * ord(a)$. Thus $a^{ord(G)} = a^{k*ord(a)} = e^k = e$.

Consider the group $(Z^*_n, * n)$. We already know that $| Z_n^* | = \Phi(n)$.

Euler's Theorem :

For any integer $n > 1$

$a^{\Phi(n)} \equiv 1 \bmod n$ for all $a \in Z_n^*$. [Corollary 1]

Fermat's Theorem :

If p is a prime then $| Z_p^* | = \Phi (p) = p - 1$.

$a^{(p-1)} \equiv 1 \ mod\ p\ for\ all\ a \in Z_p^*$.[Corollary1]

Reference:

1. Introduction to Algorithms , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, Prentice Hall India.

2. Topics in Algebra , Second Edition, I. N.Herstein, John Wiley.

3.3 PRIMITIVEROOTS

Definition 1: In a group (G, $\oplus$) an element $g \in$ G is called a ***primitive root*** or the ***generator*** if $ord(g) = |G|$.

Definition 2: A group (G, $\oplus$) is said to be ***cyclic*** if there is a ***generator*** for G.

Now the natural question is if (Z_n^*, x_n) always cyclic?

The answer is no. The following theorem Niven and Zuckerman [..] characterizes for which n (Z_n^*, x_n) is cyclic.

Theorem 1: The value of n >1 for which (Z_n^*, x_n) is cyclic are 2, 4, $p^e$and $2p^e$, for all primes p > 2 and positive integer e .

We will not prove the entire theorem. But we will concentrate on the special case when n is a prime p and show that (Z_n^*, x_n) is always cyclic.

Lemma 1 : For any n>1, $\sum_{d|n} \Phi(d) = n$

Proof : Let $\forall g$ A_g= { x |1 $\le x \le n$ $\wedge$ gcd(x , n) = g }.

If g | n then the corresponding A_g's are non-empty. They partition the set {1, 2, 3 …, n } such that

$\sum_{g|n} |A_g| = n$.

Let $d = n / g$. [Assume | A_g| = Φ(n / g)]

$$\sum_{d|n} \Phi(d) = \sum_{g|n} \Phi(n/g) = \sum_{g|n} |A_g| = n$$

Let $x \in Z_d^*$ <=> gcd(xg , dg) = g *gcd(x , d) = g [since gcd(x , d)=1]

.....<=> gcd(xg , x) = g

.....<=>$xg \in A_g$

Therefore there is a one to one correspondence between elements of $Z_d^*$and A_g.

$\Rightarrow$| A_g| = | Z_d^*| = Φ(d) . □

Theorem 2 : Z_p^* is cyclic and has Φ(p -1) generators.

Proof : Z_p^*= {1, 2, …, p -1} **.** Let O_k= { $x \in Z_p^*$| ord(x)= k }

Clearly $\sum_{k|(p-1)} |O_k| = p-1$ ……..(1)

Assume | O_k| = 0 or Φ(k) (that is to be proved later in Lemma2)

Then $\sum_{k|(p-1)} |O_k| \le \sum_{k|(p-1)} \Phi(k) = p-1$ [From Lemma1]

For the equality (1) to hold, we must have $|O_k| = \Phi(k)$ for all $k \mid (p-1)$.

Put $k = p-1$ and there are $\Phi(p-1)$ generators of Z_p^*.Thus it must be cyclic.

Lemma 2: $|O_k| = 0$ or $\Phi(k)$

Proof: If $|O_k| \neq 0$ then $\exists$ an element $a \in O_k$.

Generate the sequence $\{ a , a^2 , a^3 \ldots, a^k = e \}$

To prove this lemma we first establish a claim as follows:

Claim: $\forall\, 1 \leq j \leq k$ ord$(a^j) = k$ if and only if gcd $(j , k) = 1$.

To prove the if part let us assume gcd $(j , k) = 1$. Thus there exists integers m and n such that $mj + nk = 1$. Let k'' be the order of $a^j$and assume $k'' < k$. Therefore $(a^j)^{k''} = e$. So we have:

$$(a^{k''})^j = e \wedge (a^{k''})^k = e \Rightarrow (a^{k''})^{mk+nj} = a^{k''} = e.$$

Thus ord$(a) < k$ which is a contradiction. Hence our assumption on the order of $a^j$to be less than k is incorrect.

To provethe onlyif part weassumegcd$(j , k) = k' > 1$.Then $(a^j)^{k/k'} = (a^j)^{j/k'} = (e)^{j/k'} = e$.Since $k' > 1$, $(k / k') < k \Rightarrow a^j$has order $< k$.

So we have established our claim.

Thus $O_k = \{ a^j \mid \gcd(j , k) = 1\} \Rightarrow |O_k| = \Phi(k)$. □

Theorem 2 provides a nice characterization of computing generators for the group (Z_p^*, x_p).

A randomized Las Vegas algorithm to compute the generator is to pick an element $a \in Z_p^*$at random and check if its order is $p-1$. We will describe the checking of order of an element later.

First we observe that since there are $\Phi(p-1)$ generators of $Z_p^*$the probability that an arbitrary element of $Z_p^*$is a generator equals $\Phi(p-1)/(p-1)$. Thus after expected $(p-1)/\Phi(p-1)$ trials we will be able to obtain a generator.

3.4 GENERATORCOMPUTATION

From the fundamental theorem of arithmetic, we have $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ where $p_1 > p_2 > \ldots\ldots p_k$ are prime numbers and each $\alpha_i > 0$.

Lemma 3: For each $i \geq 1$, $p_i > i$.

Proof: We will prove the above lemma using induction. Let us assume our inductive hypothesis holds $\forall i < m$. We must show that it holds for $i = m$. The basis is clearly verified from the fact

thatfori=1,$p_i \geq 2$.Thereforep_i>ifor$i$=1.Fromourinductivehypothesiswehavep_{m-1}>m-1. Since $p_{m-1}+1$ is an even number we have $p_m > p_{m-1}+1$. Therefore p_m> m and our inductive hypothesis holds for i=m .•

Theorem 3: If ω (n) denotes the number of distinct prime factors of n , then

$$\omega(n) \in O\left(\frac{\ln n}{\ln \ln n}\right).$$

Proof: From the fundamental theorem of arithmetic, we have $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ where $p_1 > p_2 > \dots p_k$ are prime numbers and each $\alpha_i > 0$.

Now to maximize k , the number of distinct prime factors, we restrict each α_i so that each $\alpha_i = 1$. Thus from Lemma 3 we have $n = p_1 > p_2 > \dots\dots p_k > k$!. Using Stirling's Formula we obtain:

$$\left(\frac{k}{e}\right)^k < n \Rightarrow \omega(n) \in O\left(\frac{\ln n}{\ln \ln n}\right)$$

Theorem4:
$$\forall n > 1, \Phi(n)/n \geq \Omega\left(\frac{\ln \ln n}{\ln n}\right)$$

Proof: We know F (n) = n $\Phi(n) = n\prod_{i=1}^{k}(1-\frac{1}{p_i})$ where$n$ = $p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$. From **Lemma 3** we have:

$$\forall i \geq 1, p_i > i \Rightarrow \frac{1}{p_i} < \frac{1}{i} \Rightarrow 1-\frac{1}{p_i} > 1-\frac{1}{i} \Rightarrow$$

$$\forall n > 1, \Phi(n)/n \geq \frac{1}{k} \geq \Omega\left(\frac{\ln \ln n}{\ln n}\right) \quad \square$$

Now the only thing that is left is to compute the order of the randomly picked element $a \in Z_p^*$. For this we have to assume the prime factorization of (p – 1) is available with us. Let p_1 ,p_2 ,.....$p_k$are distinct prime factors of (p – 1). The following theorem enables us to computethe generator in polynomial time.

Theorem 5: Let p be a prime, $a \in Z_p^*$ is a primitive root or the generator if and only if the congruencerelation $a^{\frac{p-1}{p_i}} \equiv 1 (\mathrm{mod} p)$ does not hold for each prime divisor p_i.

Proof: Let a be a generator of Z_p^*, then we have $a^{p-1} \equiv 1 \pmod{p}$ *and* $a^h \neq 1 \pmod{p}$ $\forall h \in (0 .. p-1)$. *Since* $\frac{p-1}{p_i} \in (0 .. p-1)$ $\forall p_i$ *, wecannot have* $a^{\frac{p-1}{p_i}} \equiv 1 (\mathrm{mod} p)$ *for any* p_i.

Let $a^{\frac{p-1}{p_i}} \equiv 1 (\mathrm{mod} p)$ does hold for some prime divisor p_i . Assume that a is not a primitive root, then its order should be less than $p - 1$. Let ord(a) = d . Then we have $d < p - 1 \wedge d \mid p - 1$. So (p – 1) / d is an integer and is therefore divisible by some prime factor p_i of p -1.Then $\frac{p-1}{p_i} = cd$ for some c . Then $a^{\frac{p-1}{p_i}} \equiv a^{cd} (\mathrm{mod} p) \equiv 1 (\mathrm{mod} p)$, this contradicts with our assumption. So our assumption is wrong and hence a cannot be a generator ofZ_p^*.

From our previous discussion the probability of finding a generator in a single trial is Φ (p -1)/(p -1). Thus with high probability after expected (p -1)/ Φ (p -1) trials, i.e., $O\left(\frac{\log p}{\log \log p}\right)$ trials we will obtain a generator. Since the maximum number of distinct prime factors of (p – 1) will be $O\left(\frac{\log p}{\log \log p}\right)$. So we have to test $O\left(\frac{\log p}{\log \log p}\right)$ p_i' ssuchthat $a^{\frac{p-1}{p_i}} \equiv 1 (\mathrm{mod} p)$ does not hold for each p_i.

For the time being we will assume time to perform modular exponentiation is O(log p). [We will elaborate this algorithm later.]

Thus we have established the following theorem:

Theorem 6: Given a prime p and the prime factorization of $p-1$, a generator of $(Z_{p,}^{*} \times_{p})$ can be computed by a randomized Las Vegas algorithm with expected running time $\boldsymbol{O}$

$$\left(\frac{\log^3 p}{(\log \log p)^2} \right).$$

Unit 4: Modular Arithmetic 4

4.1 SOLVING MODULAR LINEAREQUATIONS

Solve for the unknown x in the following equation:

$ax \equiv b \bmod n$

given a , b and n .

Consider the subgroup of (Z_n, $+_n$), i.e., $\{ a^x : x > 0 \} = \{ ax \bmod n : x > 0 \} = \langle a \rangle$. Thus the above equation has a solution if and only if $b \in \langle a \rangle$.

Theorem 1:

For any positive integers a and n , if d = gcd(a , n), then $\langle a \rangle = \langle d \rangle = \{0, d , 2 d , 3 d , …., ((n / d)-1)/ d \}$ in $Z_n$and thus $|\langle a \rangle| = n / d$.

Proof:

We have to show that $\langle a \rangle = \langle d \rangle$. First we show $\langle d \rangle \subseteq \langle a \rangle$. Since d = gcd (a , b) we have $x , y \in Z_n^+$ such that $d = ax + ny$. If either x or y returned by EXTENDED-EUCLID is negative we consider them as $[n + x]_n$or $[n + y]_n$respectively. Thus $ax \equiv d \bmod n \Rightarrow d \in \langle a \rangle \Rightarrow d$ is some multiple of a . All others members of $\langle d \rangle$ belongs to $\langle a \rangle$ since they are multiple of $d \Rightarrow$multiple of multiple of a .

Now we show $\langle a \rangle \subseteq \langle d \rangle$. Pick an arbitrary element $m \equiv ax \bmod n \in \langle a \rangle \Rightarrow m = ax + ny \Rightarrow d \mid m$ (since $d \mid a$ and $d \mid n$) $\Rightarrow m \in \langle d \rangle$. Combining these result $\langle a \rangle = \langle d \rangle$

Corollary 1:

The equation $ax \equiv b \pmod n$ is solvable for the unknown x if and only if gcd(a , n) | b .

Theorem 2: Let d = gcd (a , n) and suppose that $d = ax' + ny'$ for some integers x' and y' . If $d \mid b$ then the equation $ax \equiv b \bmod n$ has one of its solutions x_0 as:

$x_0 = x' (b / d) \bmod n$

Proof: We have to show $ax_0 \equiv b \bmod n$. From the given condition we know $ax' \equiv d \bmod n$. Thus $ax_0 \equiv ax' (b / d) \bmod n \equiv d (b / d) \bmod n \equiv b \bmod n$.

Theorem 3: Consider the modular linear equation $ax \equiv b \bmod n$. If $d = \gcd(a, n)$ and $d \mid b$ and that x_0 is any solution to this equation then this equation has d distinct solutions:

$x_i = x_0 + i\,(n / d)$ for $i = 0, 1, \ldots, d-1$

Proof: We have to show $ax_i \equiv b \bmod n \;\forall i \in (0 .. d-1)$. Since $d = \gcd(a, n)$, $d \mid a$. Hence $\exists$ an integer $k = a / d$. From the given condition the following must hold:

$ax_i \equiv a\,(x_0 + i\,(n / d)) \bmod n \equiv (ax_0 + ai\,(n / d)) \bmod n \equiv (ax_0 + kin) \bmod n \equiv ax_0 \bmod n \equiv b \bmod n$.

So x_i is a solution to the given equation. Thus we conclude there are d distinct solutions to the given equation.

The following procedure computes all solutions of the modular linear equation $ax \equiv b \bmod n$.

MODULAR-LINEAR-EQUATION-SOLVER (a , b , n)

1. $(d, x', y') \leftarrow$ EXTENDED-EUCLID(a , n)
2. **if** $d \mid b$
3. **then** $x_0 \leftarrow x'(b / d) \bmod n$
4. **for** $i = 0$ to $d-1$
5. **do print** $(x_0 + i\,(n / d)) \bmod n$
6. ***else print*** *"NoSolution."*

Exercise: Find all solutions to the equation $35x \equiv 10 \pmod{50}$

Solution: Here $a = 35$, $b = 10$ and $n = 50$. We know gcd(35, 50) = 5. Thus there are 5 solutions to the given equation.

Since 3 x 35 + (-2) x 50 = 5 we have $x' = 3$. Thus $x_0 = x'(b / d) \bmod n = 3 \times (10/5) \bmod 50 = 6$. Other solutions are $x_i = x_0 + i\,(n / d)$ [$i = 1, 2, \ldots, 4$] i.e., $x_1 = 16$, $x_2 = 26$, $x_3 = 36$, $x_4 = 46$.

Corollary 2: For any $n > 1$ if $\gcd(a, n) = 1$ then the equation $ax \equiv b \bmod n$ has exactly one solution.

Corollary 3: For any $n > 1$ if $\gcd(a, n) = 1$ then the equation $ax \equiv 1 \bmod n$ has exactly an unique solution, i.e., $a^{-1} \in Z_n^*$.

Reference:

1. *Introduction to Algorithms* , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India*.

2. *Introduction to Analytic Number Theory* , T. M. Apostol, *Springer International*.

4.2 MODULAREXPONENTIATION

A frequently occurring operation in number-theoretic computations is raising one number to a power modulo another number, also known as **modular exponentiation** . More precisely, we would like an efficient way to compute $a^b \bmod n$, where a and b are nonnegative integers and n is a positive integer. They all are β bit numbers.

To compute $a^b \pmod n$ we can adopt the following approach:

Perform b multiplications ($a \times a \times a \times \ldots \times a$) mod n .

There are some drawbacks of this approach that are asfollows:

1. 1. The intermediate result is too large to fit inmemory.
   2. Not polynomial-time with respect to input size, since we are performing$b$ multiplications where our input size is $\beta \cong \lceil \log b \rceil$.

Here we present a polynomial time algorithm to perform modular exponentiation using repeated squaring.

MODULAR-EXPONENTIATION (*a* , *b* , *n*)

1 $c \leftarrow 0$

2 $d \leftarrow 1$

3 let $<b_k b_{k-1} \ldots b_0>$ *be the binary representation of b . 4*

for $i = k$ ***down to*** *0*

5 ***do*** $c \leftarrow 2c$

6 $d \leftarrow (d \cdot d) \bmod n$

7 ***if*** $b_i = 1$

8 ***then*** $c \leftarrow c + 1$

9 $d \leftarrow (d \cdot a) \bmod n$

10 ***return****d*

Here we note that the above program will run perfectly even if we remove the variable c altogether from the program. The variable c is retained to describe the loop invariant with which we establish the correctness of the above algorithm.

Invariant: In each iteration the following invariant is maintained:

1. 1. Let the current bit being processed is b_i. The value of c is the same as the prefi x $< b_k b_{k-1} \ldots, b_{i+1}>$ of the binary representation of b.
   2. $d = a^c \bmod n$.

We use this loop invariant as follows:

Initialization:

Initially i = k , so that the prefi x < $b_k b_{k-1}$ ..., b $_{i+1}$> is empty, which corresponds to c = 0.

Moreover, d = 1 = a ^{c} mod n .

Maintenance:

Let c ' and d ' denote the values of c and d at the end of an iteration of the **for** loop and thus the values prior to the next iteration. Each iteration updates c ' ← 2 c (if b_i = 0) or c ' ← 2 c + 1 (if b_i = 1), so that c will be correct prior to the next iteration.

If b_i = 0 then d ' = d^2 mod n i.e., d ' = (a ^{c})^2 mod n and hence d ' = a ^{2c} mod n = $a^{c'}$ mod n .

If b_i = 1 then d' = d ^{2} a mod n i.e., d ' = (a ^{c})^2 a mod n and hence d ' = a $^{2c+1}$ mod n = $a^{c'}$ mod n .

in either case, d = a^cmod n prior to the next iteration.

Termination:

At termination, i = -1. Thus, c = b , since c has the value of the prefi x < $b_k b_k$-1 ... b_0> of b 's binary representation. Hence d = a^c mod n = a^b mod n .

Analysis of Time Complexity:

If the inputs a , b , and n are β -bit numbers then the total number of arithmetic operations required is O(β) since we are iterating β times. Since the time complexity of multiplying two β -bit numbers is O(β^2) the total number of bit operations required is O(β^3). Thus the algorithm is clearly polynomial with respect to input size.

Note: Modular exponentiation algorithm is an essential component used in several cryptographic algorithms.

One weakness of the algorithm is the different timing requirement of each iteration depending on the value of the bit b_i. If the bit b_i= 0 clearly the **for** loop take much less computation time than the bit b_i = 1. This weakness of modular exponentiation had been exploited to attack several cryptographic algorithms. This attack is known as **timing attack** .

There are several remedies to overcome the attack. One possible solution is to remove the difference in computation time for b_i= 0 or 1 by adding some delay in each iteration when b_i= 0 and making the loop execution time equal to that of b_i = 1. We will discuss other remedies later.

Reference:

1. Introduction to Algorithms , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, Prentice Hall India .

4.3 CHINESE REMAINDER THEOREM

Around A.D. 100, the Chinese mathematician Sun-Tsu posed the following problem:

Problem 1: Determine the numbers that leave remainders 2, 3 and 2 when divided by 3, 5 and 7 respectively.

One solution to the above problem is 23. The general solution is 23+105k for arbitrary integer k.

A system of two or more linear congruence need not have solution. Consider the system of congruence $x \equiv 0 \bmod 2$ and $x \equiv 1 \bmod 4$. This system is clearly unsolvable. Since the second congruence implies x is of the form 4k + 1 which makes it indivisible by 2 and thereby making the first congruence infeasible.

But the above argument doesn‘t hold when the system of congruence have pair-wise relatively prime moduli (for example 3, 5 and 7). We will prove that the system of congruence that can be solved individually can also be solved simultaneously provided they have pair-wise relatively prime moduli.

We will first prove the result for a system of 2 congruence relations and then generalize for arbitrary number of congruence relations.

Lemma 1: The system of congruence $x \equiv a \bmod n_1$ and $x \equiv b \bmod n_2$ has exactly one solution modulo the product n_1n_2 provided gcd $(n_1,n_2) =1$.

Proof: Since gcd $(n_1, n_2) = 1$ there are integers p and q such that $pn_1 + qn_2 = 1$. Thus $pn_1 \equiv 1 \pmod{n_2}$ and $qn_2 \equiv 1 \pmod{n_1}$. Let $x' = bpn_1 + apn_2$. Thus $x' \equiv a \bmod n_1$ and $x' \equiv b \bmod n_2$. Thus x′ is a solution to our given system of congruence. Let x″ denote another solution to the system. Thus $x' \equiv x''$ (mod n_1) and $x' \equiv x''$ (mod n_2). Since gcd $(n_1, n_2) = 1$ we have $x' \equiv x''(\bmod n)$ where $n = n_1n_2$.

Chinese Remainder Theorem (Generalized Version): Let $n_1, n_2, \ldots, n_k$ be pair-wise relatively prime integers with gcd $(n_i, n_j) = 1$ where $i \neq j$. Let $a_1, a_2, \ldots, a_k$ be arbitrary integers. Then there exists exactly one solution $x \pmod{n_1 x\ n_2 x \ldots\ x\ n_k}$ to the system of congruence:

$x \equiv a_1 \pmod{n_1}$, $x \equiv a_2 \pmod{n_2}$, …, $x \equiv a_k \pmod{n_k}$.

Proof: Let $n = n_1 \times n_2 \times \ldots \times n_k$. Let us define $m_i = n/n_i$ for $i = 1, 2, \ldots, k$. Thus $m_i = n_1 \times n_2 \times \ldots \times n_{i-1} \times n_{i+1} \times \ldots \times n_k$. We now define $c_i = m_i(m_i^{-1} \bmod n_i)$ for $i = 1, 2, \ldots, k$. We know $m_i^{-1} \bmod n_i$ exists since $\gcd(m_i, n_i) = 1$. Finally we define:

$$x \equiv (a_1 c_1 + a_2 c_2 + \ldots + a_k c_k) \pmod{n}$$

To prove that x satisfies every congruence we argue as follows. We know that $c_j \equiv m_j \equiv 0 \pmod{n_i}$ for $j \neq i$ and $c_i \equiv 1 \pmod{n_i}$ otherwise. Thus

$$\begin{aligned} x &\equiv a_i c_i \pmod{n_i} \\ &\equiv a_i m_i (m_i^{-1} \bmod n_i) \pmod{n_i} \\ &\equiv a_i \pmod{n_i} \end{aligned}$$

For all $i = 1, 2, \ldots, k$.

Problem 2: Find all solutions to the equations $x \equiv 4 \pmod 5$ and $x \equiv 5 \pmod{11}$.

Solution: $a_1 = 4$, $n_1 = m_2 = 5$, $a_2 = 5$ and $n_2 = m_1 = 11$. $n = 55$.

We know $11^{-1} \equiv 1 \pmod 5$ and $5^{-1} \equiv 9 \pmod{11}$. Thus we have:

$c_1 = 11 (1 \bmod 5) = 11$ and $c_2 = 5 (9 \bmod 11) = 45$

Thus $x = 4 \times 11 + 5 \times 45 \pmod{55} = 44 + 225 \pmod{55} = 269 \pmod{55} = 49 \pmod{55}$.

So the general solution to the given system of congruence is $49 + 55k$ where k is an arbitrary integer.

Corollary 1: If $n_1, n_2, \ldots, n_k$ are pair-wise relatively prime and $n = n_1 \times n_2 \times \ldots \times n_k$ then for all integer x and a

$x \equiv a \pmod{n_i}$

for $i = 1, 2, \ldots, k$ if and only if

$x \equiv a \pmod n$.

Proof: For the if part of the proof we assume $x \equiv a \pmod n$ and hence $(x-a) = kn$ for some integer k. Thus $n_i \mid (x-a)$ since $n = n_1 \times n_2 \times \ldots \times n_k$.

To prove the only if part we assume $x \equiv a \pmod{n_i}$ for all $i = 1, 2, \ldots, k$. We prove this part by induction on k.

Basis: When $k = 2$ we have $x \equiv a \pmod{n_1}$ and $x \equiv a \pmod{n_2}$. We have to prove $x \equiv a \pmod{n_1 n_2}$. From the given congruence we can infer there exists integers k_1 and k_2 such that $(x-a) = k_1 n_2 = k_2 n_2$. Since n_1 and n_2 are relatively prime we have integers l and m such that $ln_1 + mn_2 = 1$. Multiplying both sides by (x-a) we have lk $_2n$ $_2n$ $_1$ + m k $_1n$ $_1n_2$ = (x-a) and hence (x-a) = $k_3 n_1 n$ $_2$ where k_3 is an integer.

Inductive Hypothesis: Assume the hypothesis holds for pair-wise relatively prime integers n_1, n_2, …, n_{k-1}.

Induction step: We have to show the corollary holds for pair-wise relatively prime integers n_1, n_2, …, n_k. Let $n' = n_1 \times n_2 \times \ldots \times n_{k-1}$. We know that n' and n_k are relatively prime and $x \equiv a \pmod{n'}$ and $x \equiv a \pmod{n_k}$. Following similar argument used for the proof of the basis we can show $x \equiv a \pmod{n' n_k}$. We also know that $n = n_1 \times n_2 \times \ldots \times n_k$ and thus $n = n' \times n_k$. Hence we have proved that $x \equiv a \pmod{n}$.

Reference:

1. *Introduction to Algorithms* , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India*.
2. *Introduction to Analytic Number Theory* , T. M. Apostol, *Springer International*.

4.4 DISCRETE LOGARITHMPROBLEM

Theorem 1: If g is a generator of Z_n^* then the equation $g^x \equiv g^y \pmod{n}$ holds if and only if the equation $x \equiv y \pmod{\Phi(n)}$ holds

Proof : To prove the if part we assume $x \equiv y \pmod{\Phi(n)}$. Thus $x = y + k\Phi(n)$ for some integer k.

Therefore $= g^{y+k\Phi(n)}$

$g^x = g^y (g^{\Phi(n)k}) \pmod{n}$

$= g^y (1)^k \pmod{n}$

$= g^y \pmod{n}$

To prove the only if part we assume that $g^x \equiv g^y \pmod{n}$. The sequence of powers of g generates every element of $\langle g \rangle$ and $|\langle g \rangle| = \Phi(n)$. Thus the sequence of powers of g is periodic with period $\Phi(n)$. Therefore if $g^x \equiv g^y \pmod{n}$, then we must have $x \equiv y \pmod{\Phi(n)}$.

Discrete Logarithm: Let g be the generator of the group Z_n^* . Given an element $y = g^x \pmod{n}$ the discrete logarithm is defined as $\text{dlog}_{n,g}(y) = x$.

Let us consider and the group (Z_7^*, x_n) . Clearly the group is cyclic since n = 7 is a prime number. We can see that 3 is a generator of the group. Thus discrete logarithm according to the previous definition is defined by the following table:

x	1	2	3	4	5	6
$dlog_{7,3}(x)$	0	2	1	4	5	1

Table – 1

Given g, x and *n* it is easy to determine y. By the word easy we mean it is polynomial time computable. This clearly follows from the fact that we can perform modular exponentiation in polynomial time. But given g, y and n it is difficult to compute x. This problem is known as the discrete logarithm problem. Till to-date we are not aware of any polynomial time algorithm for this problem. Many cryptographic algorithms utilize the difficulty of solving the discrete logarithm problem.

Now we can clearly see that given n if we pre-compute the entire Table-1 by computing sequentially the indices $g^0 \bmod n$, $g^1 \bmod n$,…so on and storing the corresponding exponent of g in the indexed array location. Once we are done with this preprocessing given an arbitrary x we can compute $dlog_{n,g}(x)$ in polynomial time. But there comes the tradeoff between time and memory **.**

Note: Discrete Logarithm Problem $\in$ NP. This follows from the fact that given a guess of x clearly the verification whether $y = g^x \pmod n$ can be carried out in polynomial time using modular exponentiation algorithm.

Properties of Logarithms:

$\log_a 1 = 0$

$\log_a a = 1$

$\log_a xy = \log_a x + \log_a y$

$\log_a x^n = n\log_a x$

Properties of DiscreteLogarithms:

$dlog_{n,g}(1)=0$ $\quad g^0 = 1 \pmod n$

$dlog_{n,g}(g)=1$ $\quad g^1 = g \pmod n$

$dlog_{n,g}(xy) = (dlog_{n,g}(x)+dlog_{n,g}(y)) \pmod{\Phi(n)}$

[Proof is provided in the Explanation]

$\text{dlog}_{n,g}\, x^r = r\, \text{dlog}_{n,g}(x) \pmod{\Phi(n)}$

[Using repeated application of the earlier property]

Explanation :

$x = g^{\text{dlogn}_{n,g}(x)}$

$y = g^{\text{dlogn}_{n,g}(y)}$

$(xy) \bmod n = g^{(\text{dlog}_{n,g}(x)+\text{dlog}_{n,g}(y))} \bmod n$

$xy = g^{\text{dlog}_{n,g}x,y} (\bmod n)$

$g^{\text{dlog}(xy)_{n,g}} \equiv (g^{(\text{dlog}(x)_{n,g}+\text{dlog}_{n,g}(y)} \bmod n$

Applying Theorem 1 we have:

$\text{dlog}_{n,g}(xy) = (\text{dlog}_{n,g}(x)+\text{dlog}_{n,g}(y)) \bmod(\Phi(n)))$

Reference:

1. *Cryptography and Network Security* , William Stallings, *Prenctice Hall India*.
2. *Introduction to Cryptography with Coding Theory* , W. Trappe and L. C. Washington, *Pearson Education* .

4.5 QUADRATICRESIDUES

Definition 1: An element $a \in Z_p^*$ is called a **quadratic residue** if there are elements $\pm x \in Z^*n$ such that $x^2 \equiv a \bmod n$. Otherwise we call a to be a **non-quadratic residue** .

Lemma 1: Let p be an odd prime and g be the generator of Z^* . Then all even powers of g are quadratic residues and all odd powers of g are non-quadratic residues.

Proof: Let $l = 2k$ be an even number and clearly g^l is a quadratic residue since $(\pm g^k)^2 \equiv g^l \bmod n$. Thus here $x = g^k$. In contrast let $l = 2k + 1$ be an odd number. We will prove that in this case g^l is not a quadratic residue by contradiction. Assume otherwise, i.e., let g^l be a quadratic residue. Thus there exists $x = g^m \in Z^*_p$ such that $x^2 \equiv a \bmod n$. Thus $g^{2m} \equiv g^{2k+1} \bmod p$. Applying **Theorem 1** in lecture-4 (Module -3) we have . Thus we have $2m \equiv 2k + 1 \bmod \Phi(p)$. Thuswe

have 2 m≡ 2 k + 1 mod (p -1). This implies (p -1) | (2 m -2 k -1). Since p -1 is an even number and (2 m -2 k -1) is an odd number and an even number cannot divide an odd number we have arrived at a contradiction. Thus our assumption that g^l is a quadratic residue is not correct and hence it is a non-quadratic residue.

Theorem 1: Let p be an odd prim and e ≥ 1. Then the equation $x^2 \equiv 1 \bmod p^e$ has only 2 solutions namely ±1.

Proof: Let g be the generator of the cyclic group $Z_{p^e}^*$. Thus we can rewrite our modular equation as $g^{2d\log_{p^e,g} x} \equiv g^0 \pmod{p^e}$. Thus from **Theorem 1** in lecture-4 (Module-3)we have $2d \log_{p^e,g} x \equiv 0 \bmod \Phi(p^e)$. We know $\Phi(p^e) = p^{e-1}(p-1)$. Thus the givenmodular equation is solvable since gcd(2, p^{e-1} (p-1)) = 2 | 0 and it has exactly 2 solutions namely ±1(By Inspection).

Note that if n is an arbitrary composite number the equation $x^2 \equiv 1 \bmod n$ can have more than 2 solutions. For example if n = 15 then 4 and 11 are two non-trivial roots of the equation $x^2 \equiv 1 \bmod n$ besides 1 and 14. Later in **Theorem 3** we will estimate the number of roots of the equation $x^2 \equiv 1 \bmod n$ when n is an arbitrary composite number.

Theorem 2: [Euler] Let $a \in Z_p^*$ and it is a quadratic residue if and only if $a^{\frac{p-1}{2}} \equiv 1 \bmod p$.

Proof: To prove the if part we assume that $a \in Z_p^*$ is a quadratic residue. Thus it must an even power of g where g is ageneratorof Z_p^*. Let a be equal to g^{2k}.Thus

$$a^{\frac{p-1}{2}} \equiv (g^{2k})^{\frac{p-1}{2}} \equiv (1)^k \equiv 1 \bmod p.$$

For the only if part weassume that $a \in Z_p^*$ is a non-quadratic residue. Thus it must bean odd power of g where g is ageneratorof Z_p^* Let a be equal to g^{2k+1}.Thus

$$a^{\frac{p-1}{2}} \equiv (g^{2k+1})^{\frac{p-1}{2}} \equiv (g^{p-1})^{k} g^{\frac{p-1}{2}} \equiv (1)^{k} g^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \bmod p$$

Since $g^{p-1} \equiv 1 \bmod p$ and from **Theorem 1** $g^{\frac{p-1}{2}} \equiv \pm 1 \bmod p$. Since g is the generator its order cannot be less than (p-1). Thus $g^{\frac{p-1}{2}} \equiv -1 \bmod p$ and $a^{\frac{p-1}{2}} \equiv -1 \bmod p$.

Now the most natural question is what happens when n is composite. In other words how many roots are there of the equation $x^2 \equiv a \bmod n$ when n is a composite number. We have to consider two cases:

- n is even and of theform $2^{e_1} p_2^{e_2} p_3^{e_3} \dots p_k^{e_k}$
- n is odd and of theform $p_1^{e_1} p_2^{e_2} p_3^{e_3} \dots p_k^{e_k}$

In both cases p_i's are all primes. From **Theorem 1** we know that there are exactly 2 roots for each of the modular linear equation $x^2 \equiv 1 \bmod p_i^{e_i}$ $\forall$i (pi $\neq$ 2). Again we can easily prove the following for modular linearequations.

- $x^2 \equiv 1 \bmod 2$ has only 1root.
- $x^2 \equiv 1 \bmod 4$ has exactly 2roots.
- $x^2 \equiv 1 \bmod 2^e$ has exactly 4 roots for $\forall e \geq 2$.

With this knowledge if we lift the result from primes to composites using CRT (Chinese Remainder Theorem) we can observe that the equation $x^2 \equiv a \bmod n$ has 2^k roots when n is odd and when *n* is even it has 2^{k-1}, 2^k and 2^{k+1} roots respectively for $e_1=1$, $e_1=2$ and $e_1 \geq 2$.

Now we introduce to the notion of ***Legendre Symbol*** of an element $a \in Z_p^*$. It isdenoted

by $\left(\frac{a}{p}\right)$ and is defined asfollows:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} = \begin{cases} +1 \\ -1 \end{cases}$$

depending on whether *a* is a ***quadratic residue*** or ***non-quadratic residue*** from Euler's Criterion.

Theorem 3: For every odd prime *p*wehave $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = \begin{cases} +1 \\ -1 \end{cases}$ depending on $p \equiv 1 \bmod 4$ and $p \equiv 3 \bmod 4$respectively.

Theorem 4: For every odd prime *p* wehave $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$ = +1 (if $p \equiv \pm 1 \bmod 8$) and = -1 (if $p \equiv \pm 3 \bmod 8$).

Proof: Consider the following congruences :

$p\text{-}1 \equiv 1(-1)^1 \bmod p$, $2 \equiv 2(-1)^2 \bmod p$, $p\text{ -}3 \equiv 3(-1)^3 \bmod p$, $4 \equiv (-1)^4 \bmod p$, …, *r*

$$\frac{p-1}{2}(-1)^{\frac{p-1}{2}} \equiv 2 \bmod p.$$

Here *r* is either $p - (p\text{-}1)/2$ or $(p\text{-}1)/2$. If we multiply these congruences and observing the fact that the number on the left of each congruence is even,we obtain:

$$2.4.6 \ldots (p\text{-}1) \equiv \left(\frac{p-1}{2}\right)! (-1)^{1+2+3+\ldots+\frac{p-1}{2}} \bmod p.$$

Thus we have $2^{\frac{p-1}{2}}\left(\frac{p-1}{2}\right)! \equiv \left(\frac{p-1}{2}\right)!(-1)^{\frac{p^2-1}{8}} \bmod p$. Since $\left(\frac{p-1}{2}\right)! \not\equiv 0 \bmod p$ we have established the first equalitysince $\left(\frac{2}{p}\right) = 2^{\frac{p-1}{2}}$.

Theorem 5: If a prime $p \equiv 3 \bmod 4$, then $\equiv \forall a \in Z_p^*$ either is *a* or *–a* is anon-quadratic residue.

Proof: If $p \equiv 3 \bmod 4$, then p will be of the form $4l + 3$ for some integer l, i.e., $p = 2k+1$ for some odd number k where $k = 2l+1$. We prove the theorem using contradiction. Assume both a and $-a$ are quadratic residues modulo p. We then have $x^2 \equiv a \bmod p$ and $y^2 \equiv -a \bmod p$ for some $x, y \in Z_p^*$. From this we have $x^{2k} \equiv ak \bmod p$ and $y^{2k} \equiv (-1)^k a^k \bmod p$. Since k is odd $x^{2k} \bmod p$ and $y^{2k} \bmod p$ must have opposite signs. But from **Fermat's little theorem**, both x^{2k} and y^{2k} must be congruent to 1 mod p, which contradicts the assumption. Hence either a or $-a$ is a non-quadratic residue.

If $p \equiv 3 \bmod 4$ then either $p \equiv 3 \bmod 8$ or $p \equiv 7 \bmod 8$. Using the previous theorems we can easily show that the first case 2 is the generator and in the second case p-2 is the generator. Thus using this characterization we can compute the generator of any odd prime $p \equiv 3 \bmod 4$ in O(1) time.

Now we will prove an important theorem for finding out the square root of any quadratic residue.

Theorem 6: If a $\in Z_p^*$ is a quadratic residue then its square root is $a^{\frac{p+1}{4}} \bmod p$.

Proof: Clearly we can see that $\left(a^{\frac{p+1}{4}}\right)^2 \bmod p = \left(a^{\frac{p+1}{2}}\right) \bmod p = \left(a^{\frac{p-1}{2}}\right) a \bmod p = a \bmod p$. This is because **Legendre symbol** $\left(a^{\frac{p-1}{2}}\right) \bmod p = +1$, since a is a quadraticresidue.

We can use the above theorem to compute the square root of anyquadraticresidue $a \in Z_p^*$ deterministically using modular exponentiation when $p \equiv 3 \bmod 4$.

Reference:

1. *Introduction to Algorithms* , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India*.
2. *Introduction to Analytic Number Theory* , T. M. Apostol, *Springer International*.
3. *Randomized Algorithms* , R. Motwani & P. Raghavan, *Cambridge University Press*.

Block-2

Unit 1: Key Exchange 1

5.1 DIFFIE HELLMAN KEYEXCHANGE

This key exchange protocol is one of the earliest technique that illustrates the use ofnumber theory in public key cryptography. Here two parties say Alice and Bob want to agree on a common key **K** that will be used for encryption in a symmetric key cryptosystem. A simple example is asfollows:

Key : **K**, Message :***m***

Encryption by Alice:

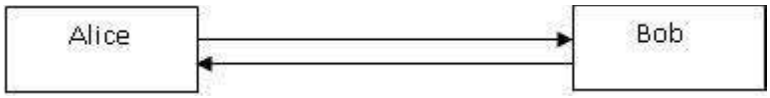
Cipher text produced is $\mathbf{C} = \boldsymbol{m} \oplus \mathbf{K}$

Decryption by Bob :

Original message is retrieved back as follows: $\boldsymbol{m} \oplus \mathbf{K} \oplus \mathbf{K} = \boldsymbol{m}$.

During the process of establishing agreement between Alice and Bob it is essential that no third party should be able to compute **K**. Let us first describe the process of establishing agreement:

$Y_A = g^{X_A} \bmod p$

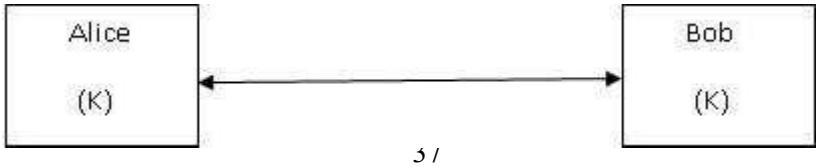


$Y_B = g^{X_B} \bmod p$

Publicly Available Information: prime p , generator g of the group (Z^*_p, x $_p$).

Step 1. Both Alice and Bob choose their private keys X_A and X_B respectively such that $1 < X_A < p - 1$ and $1 < X_B < p - 1$.

Step 2. Alice sends her public key $Y_A = g^{X_A} \bmod p$ and Bob sends his public key $Y_B = g^{X_B} \bmod p$.



Step 3. Both Alice and Bob agrees on the commonkey**K**= $(Y_B)^{X_A} \bmod p = (Y_A)^{X_B}$ mod p = $g^{X_A X_B} \bmod p$.

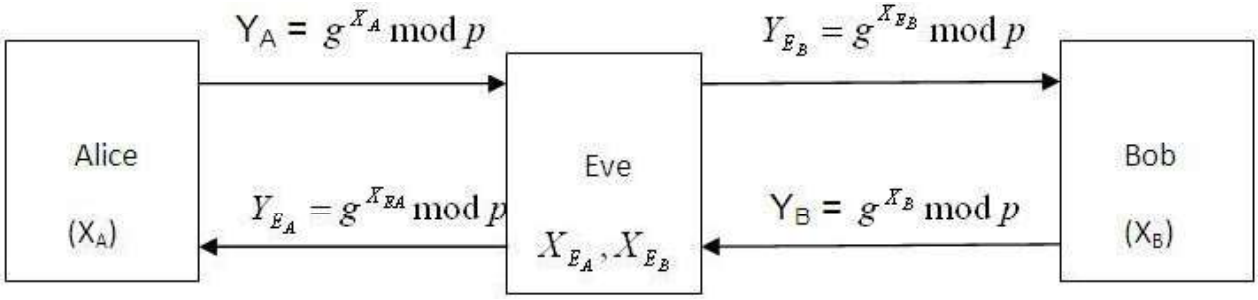
Information available to the eavesdropper are prime *p*, generator *g*, public key of Alice Y_A and public key of Bob Y_B. But to compute **K** from the available information requires computing either X_A, i.e., the secret key of Alice or X_B, i.e., the secret key of Bob. But this reduces to solving the ***discrete logarithm problem***. So the key exchange scheme is secured.

In this key exchange scheme many times it becomes computationally difficult to compute the generator *g* forthe group Z_p^*. So instead of using a generator the common practice is to pickup anelement from Z_p^* having large order to avoid ***small subgroup attack***. If the element picked has a small order then the cardinality of the sub-group generated by that element will be small and thus any brute-force algorithm will crack the ***discrete logarithm problem*** over that small subgroup.

To avoid this problem we introduce a class of primes called ***safe primes***. A safe prime *p* can always be expressed in the form 2*q*+1 where *q* is another prime. Clearly these primes are congruent to 3 mod 4. For any prime*p*,$|Z_p^*| = p - 1$. Hence if *p* is a safeprime$|Z_p^*| = 2q$. Hence there can be only subgroups of order 1, 2 and *q*. Now consider the set of allquadratic residuesin Z_p^*. The size of this set is (*p* -1)/2, i.e., *q*. It is easy to show that this set isclosed with respect to multiplication modulo *p*. Hence it is a asubgroupof Z_p^*. This subgroup is preferred in Diffie Hellman key exchange due to security issues related to the disclosure ofthe least significant bit information to the eavesdropper. Moreover since the cardinality of this subgroup is *q*, a prime number, any element of this subgroup would be thegenerator.

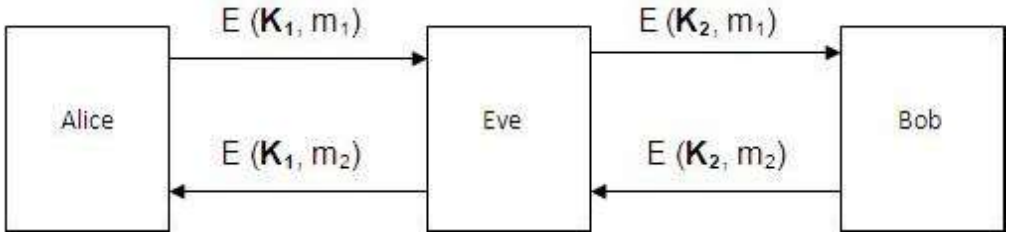
Attacks on Diffie Hellman Key Exchange Scheme:

The proposed Diffie Hellman key exchange scheme is susceptible to a type of attack known as ―Man-In-The-Middle‖ attack. Theattack proceedsasfollows:



Suppose Eve is in between Alice and Bob. Eve has 2 secret keys X_{E_A}, X_{E_B}. Eve intercepts Y_A from Alice and Y_B from Bobandsends $Y_{E_A} = g^{X_{EA}} \bmod p$ to Aliceand $Y_{E_B} = g^{X_B} \bmod p$ toBob.

Finally Alice and Eve agrees on a commonkey $\mathbf{K_1}= g^{X_A X_{E_A}} \bmod p$ and Eve and Bobagrees on a commonkey$\mathbf{K_2}= g^{X_B X_{E_B}} \bmod p$. Subsequent communication between Alice, Eve and Bob takes place as follows:



In the above diagram E(**K**, m) denotes the encrypted message m with the key **K**.

Suppose Alice wants to communicate message m1 to Bob. She sends E($\mathbf{K_1}$, m_1) to Bob, i.e., message m_1 encrypted by $\mathbf{K_1}$. Eve intercepts that and decrypts with $\mathbf{K_1}$and sends the encrypted message E($\mathbf{K_2}$, m_1) to Bob. Bob decrypts that with his key $\mathbf{K_2}$. Similarly if Bob wants to communicate a message m_2 to Alice he sends E($\mathbf{K_2}$, m_2) to Alice. That encrypted text is intercepted by Eve and decrypted with $\mathbf{K_2}$. Eve subsequently send the encrypted message E($\mathbf{K_1}$,

m_2) to Alice which she decrypts with $\mathbf{K_1}$to obtain m_2. Thus both Alice and Bob are completely unaware of the presence of Eve in the middle.

Remedy:

To overcome this type of attack every message should be authenticated by the sender. In other words the use of MAC or digital signature will eliminate this type of attack.

Reference:

1. *Introduction to Algorithms* , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India*.

2. *Introduction to Analytic Number Theory* , T. M. Apostol, *Springer International*.

3. *Randomized Algorithms* , R. Motwani & P. Raghavan, *Cambridge University Press*.

5.2 CRYPTO SYSTEMS BASED ON DISCRETE LOG

Massey Omura Cryptosystem:

Suppose Alice wants to send a message m to Bob. Alice locks the message $\boldsymbol{m}$ with a lock and sends it to Bob. Bob doesn't have the key to unlock. So what Bob does he puts an additional lock over it and sends it back to Alice. Alice unlocks her own lock and sends it back to Bob. Bob then unlocks his own lock and recovers the message $\boldsymbol{m}$.

Here a large prime p is publicly available. Also assume $\boldsymbol{m} < p$ .The communication is carried out through three following steps:

Step 1. Alice chooses a private key a and locks the message $\boldsymbol{m}$ by raising it to the power a to obtain $\boldsymbol{m}^{a} \bmod p$ and subsequently sends to Bob.

Step 2. Bob chooses another private key b and puts the additional lock by raising the received content to the power b to obtain $\boldsymbol{m}^{ab} \bmod p$ and subsequently sends back to Alice.

Step 3. Alice unlocks her own lock using a^{-1} ,i.e., shecomputesm $\left(m^{ab}\right)^{a^{-1}} \bmod p \equiv m^{aa^{-1}b} \bmod p \equiv \boldsymbol{m}^{[k(p-1)+1]b} \bmod p \equiv \boldsymbol{m}^{b} \bmod p$ and sends it again to Bob.

Step 4. Bob then recovers the message $\boldsymbol{m}$ using b^{-1}aftercomputing $m^{bb^{-1}} \bmod p \equiv \boldsymbol{m}^{k(p-1)+1} \bmod p \equiv \boldsymbol{m} \bmod p$.

The security of this cryptosystem is based on the difficulty of solving the discrete logarithm problem.

Like Diffie-Hellman key exchange scheme this cryptosystem is also susceptible to Man-In-The-Middle-Attack . Clearly we can see Alice has no way of distinguishing Bob from Eve. So to avoid this attack all message should be authenticated by the sender.

ElGamal Crypto system:

Here Bob chooses a large prime p and a generator g . Bob also chooses a secret integer a and computes $k_1 = g^a \bmod p$. The information (p , g , k_1) is made public and is Bob's public key. Suppose Alice now wants to send a message m to Bob where $0 \leq m < p$. The communication takes place as follows:

Encryption by Alice:

1. 1. Alice downloads the public key of Bob (p , g , k_1) from a publicdirectory.
   2. Alice chooses a secret random integer k and computes $C_1 \equiv g^k (\bmod p)$.
   3. Alice also computes $C_2 \equiv k^k_1$ m modp.
   4. Alice sends the pair (C_1 , C_2) to Bob.

In fact we can think C_2 as a masked message and C_1 contains the clue to unmask the message.

Decryption by Bob:

1. 1. Bob computes$k_2 \equiv C_1^{-a} \bmod p$.
   2. Bob decrypts by computing $C_2 k_2 \equiv k_1^k m (g^k)^{-a} \equiv (g^a)^k m (g^k)^{-a} \equiv$ m mod p.

If the eavesdropper knows about Bob's secret key a then he/she can easily decrypt the message using the same procedure adapted by Bob. But to compute a from g and k_1 requires solving the discrete logarithm problem.

Also computing the integer k from C_1 , g and p requires solving the discrete logarithm problem.It should be noted that k should be a random integer and should vary in each run. Otherwise if the same k is used in 2 sessions (for two distinct messages) it possible to break ElGamal Cryptosystem. The attack is explained as follows:

Assumption: The attacker Eve had somehow came to know the plaintext m in the session when Alice used the secret random integer k .

If Alice uses the same secret random integer k in another session for the message m' then shewill send the pair (C_1 , C_3) to Bob where $C_3 \equiv k_1^k m' \bmod p$.

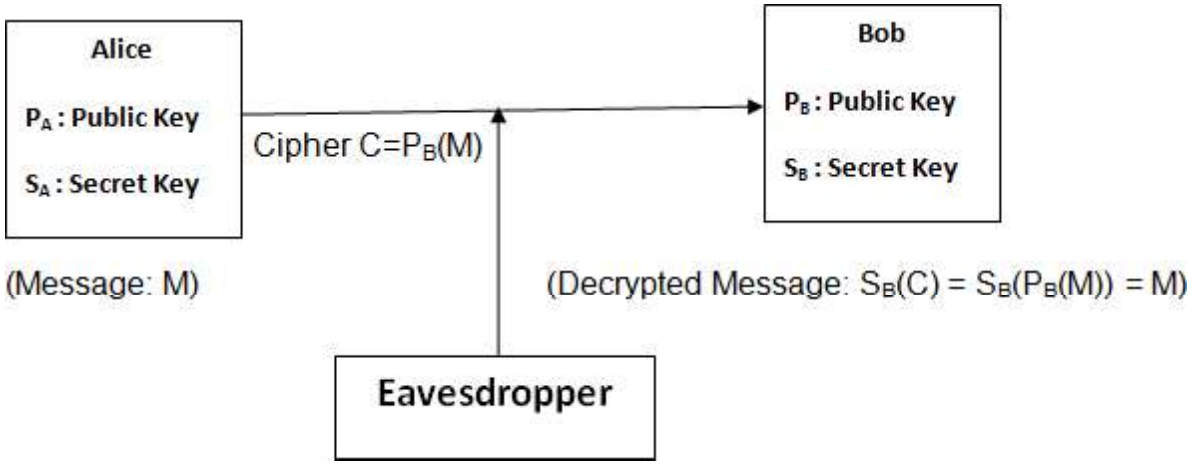
Now $\frac{C_2}{C_3} \equiv \frac{m}{m'} \bmod p \Rightarrow m' \equiv \frac{C_3}{C_2} m \bmod p \Rightarrow m' \equiv C_3 C_2^{-1} m \bmod p$. Since Eve knows all C_2, C_3 , m and p she can compute m' .

Thus the heart of the cryptosystem is based on discrete log.

Like Diffie-Hellman key exchange scheme this cryptosystem is also susceptible to Man-In-The-Middle-Attack. Clearly we can see Bob has no way of distinguishing Alice from Eve. So to avoid this attack all messages should be authenticated by the sender.

Unit 2: Public Key Cryptosystem 2

6.1 PUBLIC KEY CRYPTOSYSTEM &RSA



Suppose Alice wants to send some message M to Bob. But she cannot allow any other person to know the content of M. So she has to send her message in an encrypted format that can be decrypted only by Bob and not by any third party / eavesdropper. In public key cryptosystem this is achieved as follows:

Each party has a pair of public and secret key. So Bob had public key $P_B$and secret key S_B . All public keys of different parties are maintained in a public directory. So Alice first finds the public key P_B of Bob from the public directory. She then encrypts the message M with the public key P_B and obtains the encrypted message or cipher-text $C = P_B (M)$. This cipher-text C is sent to Bob across the communication channel. After Bob receives the cipher-text C he decrypts using his secret key S_B to get $S_B (C) = S_B (P_B (M)) = M$, the original message back.

So in this cryptosystem we have to ensure two things:

- **i.** S_B should be the inverse of P_B.

  ii. In spite of the knowledge of P_B it is computationally infeasible to an eavesdropper to determine S_B .

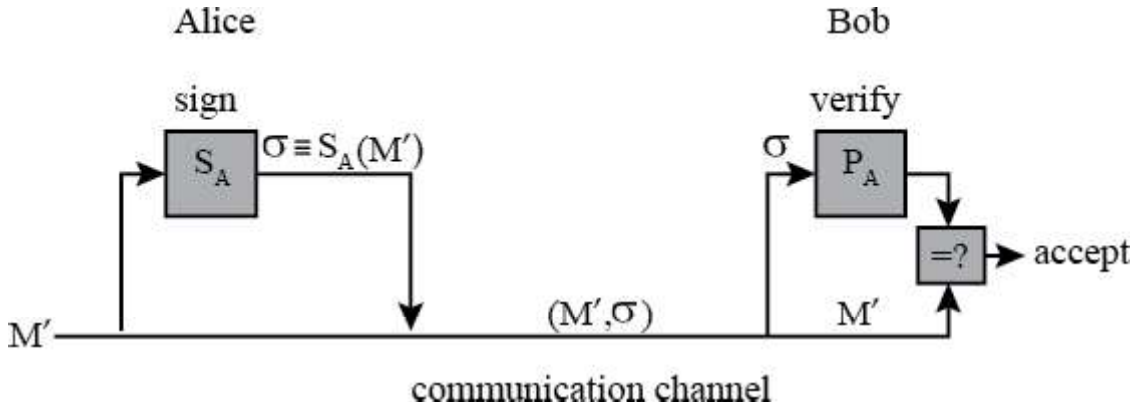
One important issue that is left is how a message M is represented. It is usually represented by an integer obtained as below.

M: –Iam fine.
In the above message there are 9 distinct alphabets including blank. So we can use a number system of base 9 and assign the code to each alphabet as follows:

Alphabet	Code
.	0
I	1
a	2
e	3
f	4
i	5
m	6
n	7
––	8

So the message string M is mapped to the integer: 9^{10} x 1 + 9^9 x 8 + 9^7 x 2 + 9^6 x 6 + 9^5 x 8 + 9^4 x 4 + 9^3 x 5 + 9^2 x 7 + 9^1 x 3 + 9^0 x 0. We can uniquely determine the string M back given this integer in base 9.

DIGITAL SIGNATURES



Alice signs the message M' by appending her digital signature $\sigma = S_A(M')$ to it. She transmits the message/signature pair (M', σ) to Bob, who verifies it by checking the equation $M' = P_A(\sigma)$. If the equation holds, he accepts (M', σ) as a message that has been signed by Alice.

• Alice computes her ***digital signature*** s for the message M' using her secret key $S_A$and the

equation σ = S_A(M').

• Alice sends the message/signature pair (M', σ) toBob.

• When Bob receives (M',σ), he can verify that it originated from Alice by using Alice's public key to verify the equation M' = P_A(σ). (Presumably, M' contains Alice's name, so Bob knows whose public key to use.) If the equation holds, then Bob concludes that the message M' was actually signed by Alice. If the equation doesn't hold, Bob concludes either that the message M' or the digital signature s was corrupted by transmission errors or that the pair (M',σ) is an attempted forgery. Digital signature provides both authentication of the signer's identity Sometimes a variation of the above approach is used for digital signatures. Here a one-wayhash function h () is used. The hash function h () is public. These hash functions are called cryptographic has functions. Given a message M it is easy to compute h (M) but it is computationally infeasible to find two messages M and M' such that h (M) = h (M') . So Alice applies her secret key $S_A$over

h (M') and not over M'. So the digital signature σ = S_A(h (M')). Now she sends the pair (M',σ) to Bob. Bob cannot compute h^{-1}(). So in the first step he applies Alice's public key $P_A$over s to obtain P_A(σ) = P_A(σ_A(h (M')))= h (M'). In the second step Bob applies the public hash function over the first component of the pair (M', σ), i.e., $M'$to obtain h (M'). Bob accepts the signature as valid if and only if the results obtained in the two steps are equal. Otherwise he rejects the signature. This may happen either due to error in transmission or due to tampering by an eavesdropper. So he will ask Alice to retransmit the message-signature pair again.

Exercise Question: Alice's signature can be verified by any person including Bob. What Alice must do to ensure that only Bob can verify her signature?

RSA Public Key Cryptosystem

The cryptosystem is set up as follows:

- 1. Choose two large random and distinct primes p and q 100 – 200 digit each roughly of the samesize.
  2. Compute n = pq
  3. Compute Φ(n) = n (1-1/ p)(1-1/ q) = (p -1)(q -1).
  4. Pick an integer e that is relatively prime to Φ (n), i.e., gcd (e , Φ (n)) =1.
  5. Compute d the multiplicative inverse of e modulo Φ (n), i.e., ed ≡ 1 mod Φ(n).
  6. Publish the pair (e , n) as the **RSA public key**.

7. Retain the pair (d , n) as the **RSA secret key**.

Suppose Alice wants to send a message M to Bob. Assume M<n . So Alice encrypts M with the public key d of Bob to obtain the cipher-text C = $M^d \bmod n$. She sends C to Bob. Bob decrypts the cipher C using his secret key d to get $C^d \bmod n \equiv M^{ed} \bmod n \equiv M \bmod n$. *Eve knows C, e and n . But to determine M she has to determine d for which she has to compute Φ(n). Since Φ(n) = n (1-1/ p)(1-1/ q) = (p -1)(q -1), for Bob it is easy to compute Φ(n) since he knows both p and q . But this computation for Eve requires factoring n . So it is computationally infeasible for Eve to determine d .*

Correctness of RSA is established via following argument:

We know M $\in \mathbb{Z}_n$ *since M* $\in \mathbb{Z}$ $\wedge M < n$.

Since e and d are multiplicative inverses we have ed ≡ 1 mod Φ (n), i.e., ed = 1 + k (p -1)(q -1) for some integer k .

Now if M ≠ 0 mod p, we have:

$$
\begin{aligned}
M^{ed} &\equiv M(M^{k(p-1)(q-1)}) && \bmod p \\
&\equiv M(M^{(p-1)k(q-1)} && \bmod p \\
&\equiv M(1)^{k(q-1)} && \bmod p \text{ [Applying Fermat's Theorem]} \\
&\equiv M && \bmod p
\end{aligned}
$$

Again if $M \equiv 0 \bmod p$ then trivially $M^{ed} \equiv M \bmod p$.

Thus for all $M \in \mathbb{Z}_n$ we have:

$M^{ed} \equiv M \bmod p$ ----------(1)

Similarly for all $M \in \mathbb{Z}_n$ we have:

$M^{ed} \equiv M \bmod q$ ----------(2)

Combining (1) and (2) using Chinese Remainder Theorem we have:

$M^{ed} \equiv M \bmod n$

for all M.

Computationally hard assumption for RSA algorithm is the difficulty of factoring the modulus n . If n can be factorized in polynomial time to obtain p and q then the attacker can break the cipher in polynomial time. This is because the attacker will know Φ (n) =(p -1)(q -1) and then by using **EXTENDED-EUCLID** algorithm d can be computed. Conversely if the attacker can figure out the decryption key d then the attacker can come to know k Φ (n) since $ed \equiv 1 \bmod \Phi$ (

n) $\Rightarrow$ ed = 1+ k Φ (n) for some integer k . Then using the randomized algorithm discussed in **Lecture-1 Module-5** the attacker can factorize n in polynomial time.

RSA is frequently used in hybrid mode with fast non-public key cryptosystem. It is combined with cryptosystems for which encryption and decryption keys are identical like DES or AES. RSA is used to transmit the key. But the original message is encrypted as a symmetric cipher. Suppose the key required for symmetric encryption is K. So the message M is encrypted with K to obtain the symmetric cipher E(K, M). But the receiving party Bob doesn't know K. So the sender Alice encrypts K in RSA with receiver's public key P_B to obtain P_B (K). She then sends to Bob E(K, M) || P_B (K). Bob after receiving applies his own secret key S_B over P_B (K) to obtain K. He then applies K over the first component for symmetric decryption to retrieve M.

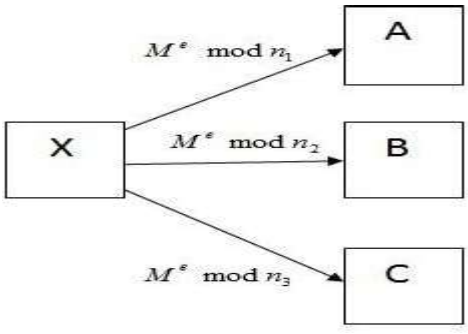
6.2 CHOICE OF THE PUBLICKEY

RSA Contd.

Choice of the Public Key: To speed up the modular exponentiation operation it is desirable that the public key has lot of 0 bits. Usual choice of public key is of the form 2^k+1 since this will have exactly two zeros. Common choice of public keys are 3, 17 and 65537 (= 2^{16}+1).

If the public key is very small then RSA is vulnerable to the following attack:

Suppose the encryption / public key is e = 3 used by three different users A, B and C having 3 distinct moduli namely n_1, n_2 and n_3. Suppose the sender X wants to send the same message M to A, B and C.So he encrypts all of them with the same public key e and computes the cipher texts $C_A = M^e \bmod n_1$, $C_B = M^e \bmod n_2$ and $C_C = M^e \bmod n_3$ respectively.



Suppose it happens to be n_1, n_2 and n_3 are pairwise relatively prime and $n_1 * n_2 * n_3 > M^e$. This can only happen if e is very small. In our case let us assume $M^3 < n_1 * n_2 * n_3$ since e = 3 though M^3 is larger than each n_1, n_2 and n_3. Then using Chinese Remainder Theorem the attacker can easily compute M^3 and thus can determine M after computing the cube root.

Operations using the secret key:

For the decryption operation we perform the following modular exponentiation operation to retrieve the original message M:

M = $C^{d} \bmod n$ where C is the cipher text, d is the secret key and $n = p \times q$ where p and q are two large primes. To speed up this operation we compute:

$V_p = C^{d} \bmod p$ and $V_q = C^{d} \bmod q$

From these using to compute $C^{d} \bmod n$ we have to use Chinese Remainder Theorem.

So we compute:

$X_p = q \times (q^{-1} \bmod p)$ and $X_q = p \times (p^{-1} \bmod q)$

Now we retrieve M as follows:

$M = (V_p X_p + V_q X_q) \bmod n$

To speed up the two modular exponentiation operations to compute $V_p$and $V_q$we can make use of Fermat's theorem as follows:

$a^{b} \bmod p = a^{y} \bmod p$

where $b = (p-1)x + y$ since $a^{p-1} \equiv 1 \bmod p$.

Attacks on RSA:

There are several attacks on RSA public key cryptosystem. They are categorized as follows:

1. Brute Force Attack: Here the attacker tries with different secretkeys.

2. Mathematical Attacks: Most of these approaches finally broil down to factoring RSA modulus.

3. Timing Attack: This attack uses the timing difference of modularexponentiation algorithm depending on the number of 0 bits and 1 bits in the secret key. We will elaborate on thislater.

4. Chosen Cipher Text Attack (CCA).

Mathematical Attack:

Here we prove that if the attacker can figure out the secret key d in polynomial time then we have a randomized polynomial time algorithm to factor n.

Choose a random number $r \in \mathbb{Z}_n^*$. Since both e and d are known we know $ed - 1 = k.\Phi(n)$. Thus from Euler's theorem $r^{ed-1} \equiv 1 \bmod n$.The goal here is to obtain a non-trivial square root of 1.For this we keepon computing $r^{\frac{ed-1}{2}}$, $r^{\frac{ed-1}{4}}$, … and so on till we get either -1 or a non-trivialsquare

root of 1 or $\frac{ed-1}{2^i}$ is no longer divisible by 2. If we obtain -1 or $\frac{ed-1}{2^i}$ is no longer divisible by2 we repeat the above procedure selecting a new random number *r*. Otherwise if we get a non-trivial square root of 1, i.e., *x* such that $x^2 \equiv 1 \bmod n$ and $x \neq \pm 1$ then gcd (*x*+1, *n*) or gcd (*x*-1,*n*) will give a non-trivial factor of *n* (i.e., 1 or *n*). Thus we have a randomized polynomial time algorithm to factorize *n*.

Chosen Cipher Text Attack:

Here the attacker Eve gets holds of a cipher text *C* that was sent by Alice to Bob. Let *M* be the corresponding plaintext. Thus $M = Ce \bmod n$. The attack proceeds as follows:

1. 1. Eve selects a random number *r*, such that $1 < r < n-1$ and $\gcd(r,n) = 1$.
1. 2. Eve computes $X = r^e C \bmod n$ and submits to Bob as a chosen cipher text.
1. 3. Eve receives back the signed message from Bob $Y = X^d \bmod n = rM \bmod n$.
1. 4. Since Eve know r^{-1} she retrieves the message $M = r^{-1} Y \bmod n$.

Remedies:

To overcome this attack the plaintext is usually padded prior to encryption. Method like optimal asymmetric encryption scheme (***OAEP***) has been proposed to overcome such attacks.

Reference:

1. *Cryptography and Network Security* , William Stallings, *Prenctice Hall India* .

6.3 ATTACKS ON RSA & REMEDIES

Timing Attack:

This attack was first suggested by Paul Kocher in 1995. He showed that it is possible to find out the secret key by careful examination of the computation times in a series of decryption procedure. The method uses the weakness of modular exponentiation algorithm and can be used to attack not only RSA but also any other cryptographic algorithms that use modular exponentiation that includes algorithms based on discrete log computation.

Suppose Eve sends to Bob several ciphertexts *y*. After decryption of each ciphertext Bob sends the acknowledgement back to Eve. Thus Eve comes to know the decryption time of each ciphertext. From this timing information Eve has to figure out the decryption exponent *d*.

We need to assume that Eve knows the hardware being used to to calculate $y^d \pmod n$. Eve can use this information to calculate the computation time of various steps that occur in this process.

Let $d=b_1b_2\ldots b_w$ be written in binary. Let y and n be integers. We perform the modular exponentiation using the following algorithm:

1. Start with $k=1$ *and* $s_1=1$.

2. If $b_k==1$, *let* $r_k \equiv s_k\, y \pmod n$. *If* $b_k == 0$, *let* $r_k=s_k$.

3. Let $s_{k+1} \equiv r_k^2 \pmod n$.

4. If $k==w$, *stop. If* $k<w$, *add 1 to k and go to (3).*

Finally $r_w \equiv y^d \pmod n$.

Here we note that the multiplication $s_k\, y$ occurs only when the bit $b_k==1$. In practice there is a large variation in timing of this multiplication operation.

Now we need to introduce few notations from Probability & Statistics. Let t denote the random variable for the time taken for the decryption of a ciphertext y.

Let $t_1, t_2, \ldots, t_n$ denote the decryption time of ciphertexts $y_1, y_2, \ldots, y_n$. The mean is denoted by :

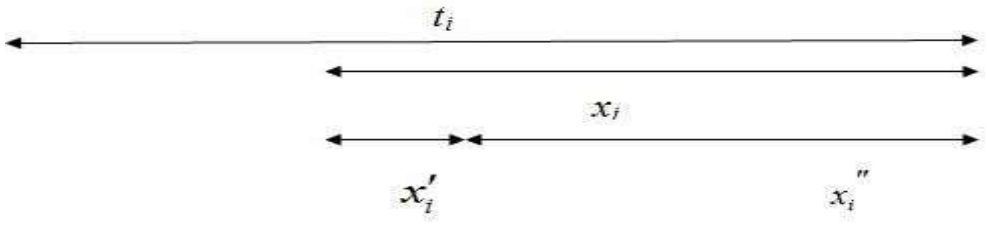
$$Mean(t) = m = \frac{t_1 + t_2 + \ldots + t_n}{n}$$

The variance of the random variable t is denoted by:

$$\mathrm{Var}(t) = \frac{(t_1 - m)^2 + (t_2 - m)^2 + \ldots + (t_n - m)^2}{n}$$

If we break up the computation time t_i for the decryption of the ciphertext y_i into two independent random processes with computation times t_i' and t_i'' respectively such that $t_i = t_i' + t_i''$, then $\mathrm{Var}(t_i) = \mathrm{Var}(t_i') + \mathrm{Var}(t_i'')$.

Eve knows $t_1, t_2, \ldots, t_n$. Suppose she knows bits $b_1b_2\ldots b_{k-1}$ *of the secret key d. Since she knows the hardware she can figure out the time required for computing* $r_1, r_2, \ldots r_{k-1}$ *in the modular exponentiation algorithm. Also she can determine the time to calculate* $s_{k+1} \equiv r_k^2 \pmod n$ *when* $b_k== 0$ *since* $r_k = s_k$. *Thus she knows the remaining computation time* x_i *for each ciphertext* y_i *to compute* $r_k, \ldots, r_w$.



Let x_i' *be the computation time for* $s_k\, y \pmod n$ *if the bit* $b_k==1$. *Eve still doesn't know* b_k.

Let $x_i'' = x_i - x_i'$. *Evecomputes* $Var(x_i)$ *and* $Var(x_i'')$. *If* $Var(x_i) > Var(x_i'')$ *Eve concludes* $b_k = 1$ *else* $b_k = 0$. *After determining* b_k *Eve proceeds in the same manner to determine the remaining bits of the secret key.*

Correctness Proof: *If* $b_k = 1$ *then the multiplication* $s_k\, y \pmod n$ *indeed occurs. It is reasonable toassume* x_i' *and* x_i'' *are independent andthus:*

$Var(x_i) = Var(x_i') + Var(x_i'') > Var(x_i'')$.

If $b_k = 0$ *then the multiplication does not occur and* $= Var(x_i') \cong 0$. *Thus*

$Var(x_i) = Var(x_i') + Var(x_i'') \cong Var(x_i'')$.

Remedies:

i) Constant Exponentiation Time: Timing attack can be avoided if Bob sends the acknowledgement back after the same fixed amount of time for each ciphertext. This solves the problem but the performance isdegraded.

ii) Random Delay: Here Bob sends back the acknowledgement after adding a random delay after the end of each modular exponentiation computation. This is susceptible to attack since the attacker can compensate the added random delay considering it as fluctuation over the d.c (average) component.

iii) Blinding: This proceeds asfollows:

1. 1. Bob selects a random number r, such that $1 < r < n-1$ and $\gcd(r,n)=1$.
   2. Bob computes $X = r^e C \bmod n$, where e is the publickey.
   3. Bob Computes $Y = X^d \bmod n = rM \bmod n$.
   4. Since Bob knows r^{-1} he retrieves the message $M = r^{-1} Y \bmod n$.

This process prevents the attacker in knowing what cipher text bits are being processed and prevents bit by bit analysis that is essential for the timing attack.

Reference:

1. *Introduction to Cryptography with Coding Theory* , W. Trappe and L. C. Washington, *Pearson Education* .
2. *Cryptography and Network Security* , William Stallings, *Prenctice Hall India*.

6.4 RABIN CRYPTO SYSTEM

Rabin Cryptosystem:

Rabin cryptosystem is described as follows:

Let n be the product of two distinct primes p and q, $p, q \equiv 3 \pmod 4$

Let P, C $\in Z_n$, where P is the plaintext and C is the cipher text.

Define

$K = \{(n, p, q, B) : 0 \le B \le n\text{-}1\}$

For K = (n, p, q, B), define

$e_K(x) = x(x+B) \bmod n$

and

$$d_K(y) = \left(\sqrt{\frac{B^2}{4} + y} - \frac{B}{2}\right) \bmod n$$

The values n and B are public, while p and q are secret.

The encryption function e_K is not an injection, so decryption cannot be done in an unambiguous fashion. In fact, there are four possible plaintexts that could be the encryption of any givenciphertext.

$$x, \quad -x, \quad \omega\left(x + \frac{B}{2}\right) - \frac{B}{2}, \quad -\omega\left(x + \frac{B}{2}\right) - \frac{B}{2}$$

It is easy to verify thatif ω is a nontrivial square root of 1 modulo n, then there are four decryptions of $e_K(x)$ for any $x \in Z_n$:

$$e\left(\omega\left(x + \frac{B}{2}\right) - \frac{B}{2}\right) = \left(\omega\left(x + \frac{B}{2}\right) - \frac{B}{2}\right)\left(\omega\left(x + \frac{B}{2}\right) + \frac{B}{2}\right) = \omega^2\left(x + \frac{B}{2}\right)^2 - \left(\frac{B}{2}\right)^2 = x^2 + Bx = e$$

Example:

So the decryption process won‘t be unique unless the plaintext contains sufficient redundancy to eliminate three pf these four values.

The decryption process is analyzed as follows:

Given a ciphertext y, the plaintext x is determined by the solving the equation

$x^2 + Bx \equiv y \pmod n$

Substituting $x = x_1 - B/2$, the above equation reduces to

$x_1^2 - B\,x_1 + B^2/\,4 + Bx_1 - B^2/\,2 - y \equiv 0 \pmod n$

or

$x_1^2 \equiv B^2\,/\,4 + y \pmod n$

Let $C = B^2 / 4 + y$, then we can rewrite the congruence as

$$x_1^2 \equiv C \pmod{n}$$

So, decryption reduces to extracting square roots modulo n. This is equivalent to solving the two congruences

$x_1^2 \equiv C \pmod{p}$

and

$x_1^2 \equiv C \pmod{q}$

Now there are two square roots of C modulo p and two square roots of C modulo q. Using the Chinese remainder theorem, these can be combined to yield four solutions modulo n. Also it can be determined by Euler's criterion if C is a quadratic residue modulo p (and modulo q). Infact, C will be a quadratic residue modulo p (and modulo q) if encryption is performed correctly.

When $p \equiv 3 \pmod 4$, there is a simple formula to compute square roots of quadratic residues modulo p. Suppose C is a quadratic residue and $p \equiv 3 \pmod 4$. Then we have that

$$\begin{aligned} \pm C^{(p+1)/4} \bmod p &\equiv C^{(p+1)/2} \pmod p \\ &\equiv C^{(p-1)/2} C \pmod p \\ &\equiv C \pmod p \end{aligned}$$

Here, we again make use of Euler's criterion, which says that if C is a quadratic residue modulo p, then $C^{(p-1)/2} \equiv 1 \pmod p$. Hence the two square roots of C modulo p are $\pm C^{(p+1)/4} \bmod p$. In a similar fashion, the two square roots of C modulo q are $\pm C^{(q+1)/4} \bmod q$. One can then obtain the four square roots x_1 of C modulo n using the Chinese RemainderTheorem

Example:

Let us illustrate the encryption and decryption procedures for the Rabin cryptosystem with a toy example. Suppose $n = 77 = 7 \times 11$ and $B = 9$. Then the encryption function is

$e_K(x) = x^2 + 9x \bmod 77$

and the decryption function is

$$d_K(y) = \left(\sqrt{1 + y} - 43\right) \bmod 77$$

Suppose the ciphertext $y = 22$. Compute the square roots of 23 modulo 7 and modulo 11. Since 7 and 11 are both congruent to 3 mod 4, using the formula derived above, we have

$$23^{(7+1)/4} \equiv 22 \equiv 4 \bmod 7$$

$$23^{(11+1)/4} \equiv 13 \equiv 1 \bmod 11$$

Using Chinese Remainder Theorem, we compute the four square roots of 23 modulo 77 to be ±10 and ±32 mod77.

Finally, the four possible plaintexts are

10 – 63 mod 77 = 44

67 – 43 mod 77 = 24

32 – 43 mod 77 = 66

45 – 43 mod 77 = 2

The computationally hard problem in this cryptosystem is the difficulty of factoring themodulus *n*. In contrary let us assume that the adversary can figure out the square roots modulo *n* . Since *n* is the product of two primes there will be 4 square roots x_1 , x_2 , x_3 and x_4 such that $x_1^2 \equiv x_2^2 \equiv x_3^2 \equiv x_4^2 \bmod n$. Among these 4 square roots there will be a pair such that $x_i \bmod n \neq \pm x_j \bmod n$ for some $i , j \in [1..4]$. Then $gcd (x_i + x_j, n)$ or $gcd (x_i - x_j, n)$ will give a non trivial factor of *n*. Thus if we can break the cryptosystem in polynomial time we will be able to factor *n* in polynomial time.

Reference:

1. *Cryptography Theory and Practice* , D. R. Stinson, *CRC Press*.

Unit 3: Factorization 3

7.1 CURRENT STATE OF THEART

Current state of the art

Factorization of integer in polynomial time is still to date an unresolved problem. Cryptographic algorithms like RSA, Rabin all rely upon the difficulty of integer factorization problem. Even to date factoring large integers with very fast computers require a lot of computing time. There are some efficient pseudo polynomial time algorithms known for the factoring problem.

Difficulty and complexity

If a large, b-bit number is the product of two primes that are roughly the same size, then no algorithm is published that can factor in polynomial time. That means there is no widely known algorithm that can factor it in time $O(b^k)$ for any constant k. In other words, there are algorithms which are super-polynomial but sub- exponential. In particular, the best published asymptotic running time is for the general number field sieve (GNFS) algorithm, which, for a b-bit number n, is:

$$O\left(\exp\left(\left(\tfrac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right)$$

For an ordinary computer, GNFS is the best published algorithm for large n (more than about 100 digits). For a quantum computer, however, Peter Shor discovered an algorithm in 1994 that solves it in polynomial time. This will have significant implications for cryptography if a large quantum computer is ever built. Shor's algorithm takes only $O(b^3)$ time and O(b) space on b-bit number inputs. In 2001, the first 7-qubit quantum computer became the first to run Shor's algorithm. It factored the number 15.

It is not known exactly which complexity classes contain the integer factorization problem. The decision-problem form of it ("does N have a factor less than M?") is known to be in both NP and co-NP. This is because both YES and NO answers can be trivially verified given the prime factors (whose correctness can be verified using the AKS primality test). It is known to be in BQP because of Shor's algorithm. It is suspected to be outside of all three of the complexity

classes P, NP-Complete, and co-NP-Complete. If it could be proved that it is in either NP-Complete or co-NP-Complete, that would imply NP = co-NP. That would be a very surprising result, and therefore integer factorization is widely suspected to be outside both of those classes. Many people have tried to find classical polynomial-time algorithms for it and failed, and therefore it is widely suspected to be outside P.

Interestingly, the decision problem "is N a composite number?" (or equivalently: "is N a prime number?") appears to be much easier than the problem of actually finding the factors of N. Specifically, the former can be solved in polynomial time (in the number n of digits of N) with the AKS primality test. In addition, there are a number of probabilistic algorithms that can test primality very quickly if one is willing to accept the small possibility of error. The easiness of primality testing is a crucial part of the RSA algorithm, as it is necessary to find large prime numbers to start with.

Trial division

Trial division is the simplest and easiest to understand of the integer factorization algorithms.

Given an odd composite integer n there must be a prime factor less than $\sqrt{n}$. Thus we need to test for all primes $p \le \lfloor \sqrt{n} \rfloor$ that divides n . Let π(n) denote the number of primes less than n . From the prime number theorem we have thefollowing:

$$Limit_{n\to\infty}\pi(n) = \frac{n}{\ln n}$$

Thus we have to test for all $\pi(\lfloor \sqrt{n} \rfloor)$ prime factors of n . From the previous theorem wehave:

$$\pi(\sqrt{n}) \approx \frac{2\sqrt{n}}{\ln n}$$

If a v a riant is used without pri m ality testing, but si m ply dividing by every odd nu m ber less than the square root of *n* , pri m e or not, it can take up to about

$$\frac{\sqrt{n}}{2}$$

trial divisions which for large n is worse.

If n has small prime factors then this algorithm performs quite well. This means that for n with large p rime factors of similar size (like those used in public key cryptography), trial division is computation a lly infeasible. For most significant factoring concerns, however, other algorithms are more efficient and therefore feasible.

Given a composite integer n(throughout this article, n means "the integer to be factored"),trial division consists of trial-dividing n by every prime number less than or equal to $\sqrt{n}$. If a number is found which divides evenly into n, that number is a factor ofn.

A definite bound on the prime factors is possible. Suppose P(i) is the i'th prime, so that $P(1) = 2$, $P(2) = 3$, etc. Then the last prime number worth testing as a possible factor o n is P(i) where $P(i + 1)^2 > n$; equality here would mean that P(i + 1) was a factor. This is all very well, but usually inconvenient to apply for the inspection of a single n since determining the correct value for i is more effort than simply trying the one unneeded candidate P(i + 1) that would be involved in testing with all P(i) such that

$P(i) <= \sqrt{n}$. Should the square root of n be integral, then it is a factor and n is a Perfect square, not that this is a good way of findingthem.

Trial division is guaranteed to find a factor of n, since it checks all possible prime factors of n. Thus, if the algorithm finds no factor, it is proof that n is prime.

In the worst case, trial division is a laborious algorithm. If it starts from 2 and works up to the square root of n, the algorithm requires

trial divisions, where $\pi(x)$ denotes the prime counting function, the number of primes less than x. This does not take into account the overhead of primality testing to obtain the prime numbers as candidate factors. If a variant is used without primality testing, but simply dividing by every odd number less than the square root of n, prime or not, it can take up to about

$$\frac{\sqrt{n}}{2}$$

trial divisions which for large n is worse.

This means that for n with large prime factors of similar size (like those used in public key

cryptography), trial division is computationally infeasible.

However, for n with at least one small factor, trial division can be a quick way to find that small factor. It is worthwhile to note that for random n, there is a 50% chance that 2 is a

factor of n, and a 33% chance that 3 is a factor, and so on. It can be shown that 88% of all positive integers have a factor under 100, and that 91% have a factor under 1000.

For most significant factoring concerns, however, other algorithms are more efficient and therefore feasible.

Pollard's p-1 algorithm [4]

Pollard's p − 1 algorithm is a number theoretic integer factorization algorithm, invented by John Pollard in 1974. It is a special-purpose algorithm, meaning that it is only suitable for integers with specific types of factors.

The algorithm is based on the insight that numbers of the form $a^b - 1$ tend to be highly composite when b is itself composite. Since it is computationally simple to evaluate numbers of this form in modular arithmetic, the algorithm allows one to quickly check many potential factors with great efficiency. In particular, the method will find a factor p if b is divisible by p− 1, hence the name. When p − 1 is smooth (the product of only small integers) then thisalgorithm is well-suited to discovering the factorp.

Base concepts

Let n be a composite integer with prime factor p. By Fermat's little theorem, weknowthat

$a^{p-1} \equiv 1 \pmod{p}$ for a coprime top

Let us assume that p − 1 is B-powersmooth for some reasonably sized B (more on the selection of this value later). Recall that a positive integer m is called B-**smooth** if all prime factors p_i of m are such that $p_i \leq B$. m is called B-**powersmooth** if all prime powers

$_i$ dividing m are such that $p_{ii} \leq B$.

Let $p_1, ..., p_L$ be the primes less than B and let $e_1, ..., e_L$ be the exponents such that

$$p_i^{e_i} \leq B < p_i^{e_i+1}$$

Let

$$M = \prod_{i=1}^{L} p_i^{e_i}$$

As a shortcut, M = lcm{1, ..., B}. As a consequence of this, (p− 1) divides M, and also if p^e divides M this implies that $p^e \leq B$. Since (p − 1) divides M we know that $a^M \equiv 1 \pmod p$, and because p divides n this means $\gcd(a^M - 1, n) > 1$.

Therefore if $\gcd(a^M - 1, n) \neq n$, then the gcd is a non-trivial factor of n.

If p − 1 is not B-power-smooth, then $a^M \not\equiv 1 \pmod p$ for at least half of all a.

Pollard concepts

Let *n* = pqr, where p and *q* are distinct primes and r is an integer, such that $p - 1$ is B-powersmooth and $q - 1$ is not B-powersmooth. Now, $gcd(a^M - 1, n)$ yields a proper factor of n. In the case where q − 1 is B-powersmooth, the gcd may yield a trivial factor because q divides $a^M - 1$. This is what makes the algorithm specialized. For example, 172189 = 421 × 409. 421 − 1 = $2^2 \times 3 \times 5 \times 7$ and 409 − 1 = $2^3 \times 3 \times 17$. So, an appropriate value of B would be from 7 to 16. If B was selected less than 7 the gcd would have been 1 and if B was selected higher than 16 the gcd would have been n. Of course, we do not know what value of B is appropriate in advance, so this will factor into thealgorithm.

To speed up calculations, we also know that when taking the gcd we can reduce one part modulo the other, so $\gcd(a^M - 1, n) = \gcd(a^M - 1 \bmod n, n)$. This can be efficiently calculated using modular exponentiation and the Euclidean algorithm.

Algorithm and running time

The basic algorithm can be written as follows:

Inputs: n: a composite integer

Output: a non-trivial factor of n or failure

1. select a smoothness bound B
2. randomly pick a coprime to n (note: we can actually fix a, random selection here is not imperative)
3. for each prime q ≤B

$$e \leftarrow \left\lfloor \frac{\log B}{\log q} \right\rfloor$$

a ← a^{qe} mode n (note: this is a^M)

4. g ← gcd(a − 1,n)
5. if 1 < g < n then returng
6. if g = 1 then select a higher B and go to step 2 or returnfailure
7. if g = n then go to step 2 or returnfailure

If g = 1 in step 6, this indicates that for all p − 1 that none were B-powersmooth. If g = n in step 7, this usually indicates that all factors were B-powersmooth, but in rare cases it could indicate that a had a small order modulo p.

The running time of this algorithm is O(B × log B × $\log^2$n), so it is advantageous to pick a small value of B.

7.2 LARGE PRIMEVARIANT

Large prime variant

A variant of the basic algorithm is sometimes used. Statistically, there is often a factor p of n such that p − 1 = fq such that f is B-powersmooth and B < q≤ B', where q is a prime and B' is called a semi-smoothness bound.

As a starting point, this would work into the basic algorithm at step 6 if we encountered gcd = 1 but didn't want to increase B. For all primes B < q_1, ..., q_L ≤ B', we check if

$$\gcd\left(a^{q_i m} - 1, n\right) \neq 1$$

to obtain a non-trivial factor of n. This is quickly accomplished, because if we let c = a^M, and d_1 = q_1 and d_i = q_i − q_i − 1, then we can compute

$$a^{q_1 m} = c^{d_1}, a^{q_2 m} = c^{d_1} c^{d_2} = a^{q_1 m} c^{d_2}, \ldots$$

The running time of the algorithm with this variant then becomes O(B' × log B' × $\log^2$n).

Additional information

Because of this algorithm's effectiveness on certain types of numbers the RSAspecifications require that the primes, p and q, be such that p-1 and q-1 are non-B- power-smooth for small values of B.

Williams' p plus 1 algorithm[5]

In computational number theory, **Williams' p + 1 algorithm** is an integer factorization algorithm invented by H. C. Williams.

It works well if the number N to be factored contains one or more prime factors p such that $p + 1$ is smooth, i.e. $p + 1$ contains only small factors. It uses Lucas sequences. It is analogous to Pollard's p-1 algorithm.

Algorithm

Choose some integer A greater than 2 which characterizes the sequence:

$V_0 = 2, V_1 = A, V_j = AV_{j-1} - V_{j-2}$

where all operations are performed modulo N.

Then any odd prime p divides $\gcd(N, V_M - 2)$ whenever M is a multiple of $p - (D / p)$, where $D = A^2 - 4$ and (D / p) is the Jacobi symbol.

We require that $(D / p) = -1$, that is, D should be a quadratic non-residue modulo p. But as we don't know p beforehand, more than one value of A may be required before finding a solution. If $(D / p) = +1$, this algorithm degenerates into a slow version of Pollard's p-1 algorithm.

So, for different values of M we calculate $\gcd(N, V_M - 2)$, and when the result is not equal to 1 or to N, we have found a non-trivial factor of N. The values of M used are successive factorials, and V_M is the M-th value of the sequence characterized by V_{M-1}.

To find the M-th element V of the sequence characterized by B, we proceed in a manner similar to left-to-right exponentiation:

x=B

y=(B^2-2) mod N

for each bit of M to the right of the most significant bit if the bit is 1

    x=(x*y-B) mod N

    y=(y^2-2) mod N

else

    y=(x*y-B) modN

    x=(x^2-2) modN

V=x

Example

With N=112729 and A=5, successive values of V_M are: V_1

of seq(5) = V_1! of seq(5) = 5

V_2 of seq(5) = V_2! of seq(5) = 23

V_3 of seq(23) = V_3! of seq(5) = 12098

V_4 of seq(12098) =V_4! of seq(5) =87680

V_5 of seq(87680) = V_5! of seq(5) = 53242

V_6 of seq(53242) = V_6! of seq(5) = 27666

V_7 of seq(27666) = V_7! of seq(5) = 110229

At this point, gcd(110229-2,112729) = 139, so 139 is a non-trivial factor of 112729. Notice that p+1 = 140 = 2 × 5 × 7. The number 7! is the lowest factorial which is multiple of 140, so the proper factor 139 is found in this step.

Lenstra elliptic curve factorization[6]

Lenstra elliptic curve factorization or the **elliptic curve factorization method** (**ECM**) is a fast, sub-exponential running time algorithm for integer factorization which employs elliptic curves. Technically, the ECM is classified as a deterministic algorithm as all "random" steps (such as the choice of curves) used can be de-randomized and done in a deterministic way. (This is not to say that the algorithm can't be implemented in a probabilistic way, if one so chooses, provided one has a true source of randomness.)

For factoring ECM is the third-fastest known factoring method. The second fastest is the multiple polynomial quadratic sieve and the fastest is the general number field sieve; both are probabilistic algorithms.

Practically speaking, ECM is considered a special purpose factoring algorithm as it is most suitable for finding small factors. Currently, it is still the best algorithm for divisors not greatly exceeding 20 to 25 digits (64 to 83 bits or so), as its running time is

dominated by the size of the smallest factor *p* rather than by the size of the number *n* to be factored. The largest factor found using ECM so far was discovered on August 24, 2006 by B. Dodson and has 67 digits[7]. Increasing the number of curves tested improves the chances of finding a factor, but they are not linear with the increase in the number of digits.

Derivation

ECM is at its core an improvement of the older p-1 algorithm. The p-1 algorithm finds prime factors p such that p-1 is B-powersmooth for small values of b. For any e, a multiple of p-1, and any a relatively prime to p, by Fermat's little theorem we have $a^{e} \equiv 1 \pmod{p}$. Then gcd(a^{e}-1, n) is likely to produce a factor of n. However, the algorithm fails when $p-1$ has large prime factors, as is the case for numbers containing strong primes,for example.

ECM gets around this obstacle by considering the group of a random elliptic curve over the finite field $\mathbf{Z}_p$, rather than considering the multiplicative group of $\mathbf{Z}_p$which always has order p-1.

The order of the group of an elliptic curve over $\mathbf{Z}_p$varies (randomly) between $p + 1 - 2\sqrt{p}$ and $p + 1 + 2\sqrt{p}$ by Hasse's theorem, and is likely to be smooth for some elliptic curves. Although there is no proof that a smooth group order will be found in the Hasse-interval, by using heuristic probabilistic methods, the Canfield-Erdös-Pomerance theorem with suitably optimized parameter choices, and the L-notation, we can expect to try $\mathbf{L}[\sqrt{2} / 2, \sqrt{2}]$ curves before getting a smooth group order. This heuristic estimate is very reliable in practice.

Lenstra's elliptic curve factorization

The Lenstra elliptic curve factorization method to find a factor of the given number n works as follows:

- • Pick a random elliptic curve over $\mathbf{Z}$ with a point A on it. Then, we consider the group law on this curve mod n — this is possible since almost all residues mod n have inverses, which can be found using the Euclidean algorithm, and finding a noninvertible residue is tantamount to factoring$n$.

  • Compute eA in this group, where e is product of small primes raised to small powers,as in the p-1 algorithm. This can be done one prime at a time, thusefficiently.

  • Hopefully, eA is a zero element of the elliptic curve group in $\mathbf{Z}_p$, but not in $\mathbf{Z}_q$for another prime divisor q of n (as in the p-1 method, it is unlikely that both groups will have an order which is a divisor of e). Then we can find a factor of n by finding the greatest common divisor of the first coordinate of A and n, since this coordinate will be zero in $\mathbf{Z}_p$.

• If it does not work, we can try again with some other curve and starting point.

The complexity depends on the size of the factor and can be represented by, where *p* is the smallest factor ofn.

7.3 DIXON'S FACTORIZATION METHOD

Dixon's factorization method

In number theory, **Dixon's factorization method** (also **Dixon's algorithm**) is a general- purpose it is the prototypical factor base method, and the only factor base method for which a run-time bound not reliant on conjectures about the smoothness properties of values of a polynomial is known. The algorithm was designed by John D. Dixon, a mathematician at Carleton University, and was published in 1981.

Basic idea

Dixon's method is based on finding a congruence of squares modulo the integer N which we intend to factor. Fermat's factorization algorithm finds such a congruence by selecting random or pseudo-random *x*values and hoping that the integer $x^2 \bmod N$ is the square of an integer . :

$$x^2 \equiv y^2 \pmod{N}, \qquad x \not\equiv \pm y \pmod{N}.$$

For example, if *N*=84923, we notice (by starting at 292, the first number greater than $\sqrt{N}$ and counting up) that 505^2 mod 84923 is 256, the square of 16. So(505-16)(505+16)=0 mod *N*. Computing the GCD of 505-16 and N using Euclid's algorithm gives us 163, which is a factor of *N*.

In practice, selecting random *x* values will take an impractically long time to find a congruence of squares, since there are so few squares less than N.

Dixon's method replaces the condition 'is the square of an integer' with the much weaker one 'has only small prime factors'; for example, there are 292 squares less than 84923,

662 numbers whose prime factors are only 2,3,5 or 7, and 4767 whose prime factors are all less than 30.

If we have lots of numbers $a_1 \ldots a_n$ whose squares can be factorisedas

$a_i^2 \mod n = \prod_{j=1}^{m} b_i^{e_{ij}}$ for afixed set $b_1 \cdots b_m$ of small primes, linear algebra modulo 2 on the matrix e_{ij}will give us a subset of the $a_i$whose squares combine to a product of small primes to an even power -- that is, a subset of the $a_i$whose squares combine to asquare.

Method

Firstly, a set of primes less than some bound B is chosen. This set of primes is called the factor base. Then, using the polynomial

$p(x) = x^2 (\mod n)$

many values of x are tested to see if $p(x)$ factors completely over the factor base. If it does, the pair (x, $p(x)$) is stored. Such a pair is called a relation. Then, once the number of relations collected exceeds the size of the factor base, we can enter the next stage.

The $p(x)$ values are factorized (this is easy since we are certain they factorize completely over the factor base) and the exponents of the prime factors are converted into an exponent vector mod 2. For example, if the factor base is {2, 3, 5, 7} and the $p(x)$ value is

30870, we have:

$30870 = 2^1.3^2.5^1.7^3$

This gives an exponent vector of:

$$\mathbf{v}_i = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \mod 2$$

If we can find some way to add these exponent vectors together (equivalent to multiplying the corresponding relations together) to produce the zero vector (mod 2), then we can get a congruence of squares. Thus we can put the exponent vectors together into a matrix, and formulate an equation:

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n = 0 \pmod 2$$

This can be converted into a matrix equation:

$$\begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nn} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \pmod 2$$

This matrix equation is then solved (using, for example, Gaussian elimination) to find the vector **c**. Then:

$$\prod_k x_k^2 \equiv \prod_k p(x_k) \pmod n$$

where the products are taken over all k for which C_k= 1. At least one of the $C_k$must be one. Because of the way we have solved for **c**, the right-hand side of the above congruence is a square. We then have a congruence of squares.

Example

Considering the factor base {2,3,5,7}, we will try to factor 84923.

513^2 mod 84923 = 8400 = 2^4*3*5^2*7

517^2 mod 84923 = 33600 = 2^6*3*5^2*7

so

$(513.537)^2$ mod 84923 = $2^{10}.3^2$ 5^4 7^2

513 times 537 is 20712 (mod 84923).

That is,

20712^2 mod 84923 = $(2^5.3.55^2.7)^2$ mod 84923 = 16800^2 mod 84923

We then look at 20712-16800 = 3912 and 20712+16800 = 37512, and compute their greatest common divisors with 84923 by using Euclid's algorithm. This is 163 in the case of 3912, and 521 in the case of 37512; and, indeed, 84923 = 521 * 163.

REFERENCES

1. http://en.wikipedia.org/wiki/Trial_division

2. Richard P. Brent. *An Improved Monte Carlo Factorization Algorithm*, BIT 20, 1980,pp.176-184

3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and CliffordStein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-

03293-7. Section 31.9: Integer factorization, pp.896–901

4. http://en.wikipedia.org/wiki/Pollard%27s_p-1_algorithm

5. http://www.mersennewiki.org/index.php/P_Plus_1 MersenneWiki article about p+1 factorizationmethod.

6. Lenstra Jr., H. W. "Factoring integers with elliptic curves." *Annalsof Mathematics* (2) 126 (1987), 649-673. MR89g:11125.

7. Brent, Richard P. "Factorization of the tenth Fermat number." *Mathematics of Computation* 68 (1999),429-451.

8. http://en.wikipedia.org/wiki/Fermat%27s_factorization_method

9. J. D. Dixon, "Asymptotically fast factorization of integers," *Math. Comput.*, 36(1981), p.255-260.

7.4 QUADRATIC-SIEVE FACTORING

The pseudo code for the MD5 algorithm is as follows:

// *Note: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating*

var *int* [64] r, k

// *r specifies the per-round shift amounts*

```
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}
```

// *Use binary integer part of the sines of integers as constants:*

for i **from** 0 **to** 63

  k[i] := floor(abs(sin(i + 1)) × (2 **pow** 32))

// *Initialize variables:*

h0 := 0x67452301

h1 := 0xEFCDAB89

h2 := 0x98BADCFE

h3 := 0x10325476

//*Pre-processing:*

append "1" bit **to** message

```
append "0" bits until message length in bits = 448 (mod 512)
append bit (bit, not byte) length of unpadded message as 64-bit little-endian integer to message
// Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
break chunk into sixteen 32-bit little-endian words w[i], 0 = i = 15
// Initialize hash value for this chunk:
var int a := h0
var int b := h1
var int c := h2
var int d :=h3

// Mainloop:

for i from 0 to 63
    if 0 ≤ i ≤ 15 then
        f := (b and c) or (( not b) and d)
        g := i
    else if 16 ≤ i ≤ 31
        f := (d and b) or (( not d) and c)
        g := (5×i + 1) mod 16
else if 32 ≤ i ≤ 47
        f := b xor c xor d
        g := (3×i + 5) mod 16
else if 48 ≤ i ≤ 63
        f := c xor (b or ( not d))
        g := (7×i) mod 16

temp := d
d := c
c :=b
```

b := b + **leftrotate** ((a + f + k[i] + w[g]) , r[i])

a := temp

// *Add this chunk's hash to result so far:*

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

var *int* digest := h0 **append** h1 **append** h2 **append** h3

// *(expressed as little-endian)*

// *leftrotate function definition*

leftrotate (x, c)

return (x << c) **or** (x >> (32-c));

Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two message having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

Differences Between MD4 and MD5

The following are the differences between MD4 and MD5

1. A fourth round has beenadded.
2. Each step now has a unique additiveconstant.
3. The function g in round 2 was changed from (XY v XZ v YZ) to(XZ v Y not(Z)) to makeg lesssymmetric.
4. Each step now adds in the result of the previous step.This

.....a. promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2and

.....a. 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have beenapproximately

.....a. optimized, to yield a faster "avalanche effect." The shifts in

.....b.different rounds are distinct.

SHA hash functions

The **SHA hash functions** are five cryptographic hash functions designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard . SHA stands for Secure Hash Algorithm.

The five algorithms are denoted *SHA-1* , *SHA-224* , *SHA-256* , *SHA-384* , and *SHA-512* . The latter four variants are sometimes collectively referred to as *SHA-2* . SHA-1 produces a message digest that is 160 bits long; the number in the other four algorithms' names denote the bit length of the digest they produce.

SHA-1 is employed in several widely used security applications and protocols, including TLS and SSL , PGP , SSH , S/MIME , and IPsec . It was considered to be the successor to MD5 , an earlier, widely-used hash function.

SHA-1 algorithm

Initialize variables:

h0 := 0x67452301

h1 := 0xEFCDAB89

h2 := 0x98BADCFE

h3 := 0x10325476

h4 := 0xC3D2E1F0

Pre-processing:

append the bit '1' to the message

append k bits '0', where k is the minimum number >= 0 such that the resulting message

.....length (in *bits*) is congruent to 448 (mod 512)

append length of message (before pre-processing), in *bits* , as 64-bit big-endian integer

Process the message in successive 512-bit chunks:

break message into 512-bit chunks

for each chunk

.....break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15

Extend the sixteen 32-bit words into eighty 32-bit words:

for i **from** 16 to 79

.....w[i] := (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16]) **leftrotate** 1

Initialize hash value for this chunk:

a := h0

b := h1

c := h2

d := h3

e := h4

Main loop:

for i from 0 to 79

    if 0 ≤ i ≤ 19 then

        f := (b and c) or ((not b) and d)

        k := 0x5A827999

    else if 20 ≤ i ≤ 39

        f := b xor c xor d

        k := 0x6ED9EBA1

    else if 40 ≤ i ≤ 59

        f := (b and c) or (b and d) or (c and d)

        k := 0x8F1BBCDC

    else if 60 ≤ i ≤ 79

        f := b xor c xor d

        k := 0xCA62C1D6

temp := (a **leftrotate** 5) + f + e + k + w[i]

e :=d

d := c

c := b **leftrotate** 30

b := a

a := temp

Add this chunk's hash to result so far:

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

h4 := h4 + e

Produce the final hash value (big-endian):

digest = hash = h0 **append** h1 **append** h2 **append** h3 **append** h4

Reference:

Hans Delfs and Helmut Knebl, Introduction to Cryptography: Principles and Applications, 2 nd Edition, Springer Verlag.

1. *Introduction to Cryptography with Coding Theory* , W. Trappe and L. C. Washington, *Pearson Education* .
2. *Cryptography and Network Security* , William Stallings, *Prenctice Hall India*.
3. *Cryptography Theory and Practice* , D. R. Stinson, *CRC Press*.

7.5 POLLARD-RHOMETHOD

Fermat Factorization

This method of factorization a number ***n*** is based on the fact that every odd number can be expressed as the difference of two squares.

Let $$n = ab = {}^{1}/_{4}\,((a+b)^2 - (a-b)^2) = (\tfrac{(a+b)}{2})^2 - (\tfrac{(a-b)}{2})^2$$

Notice that, if n is odd, then so is a and b and hence $\frac{(a+b)}{2}$ and $\frac{(a-b)}{2}$ areintegers.

Therefore, $n = t^2 - s^2$ where $t = \frac{(a+b)}{2}, s = \frac{(a-b)}{2}$

$$\Rightarrow n = (t+s)(t-s), a = (t+s) \; and \; b = (t-s)$$

Factorization technique

One tries various values of *t*,hopingthat $t^2 - n = s^2$ is a square.

FermatFactor (n): // n is odd

$$t \leftarrow \lceil \sqrt{n} \rceil$$

$$s\_sq \leftarrow t * t - N$$

while s_sq isn't a square:

$$t \leftarrow t + 1$$

$$s\_sq \leftarrow t * t - N$$

endwhile

return $t - \sqrt{s\_sq}$ // or $t + \sqrt{s\_sq}$

Run time*:* let n = ab ,then $t = {}^{(a+b)}/_{2}$

Number of stepsrequired: $${}^{(a+b)}/_{2} - \sqrt{n} = (\sqrt{a} - \sqrt{b})^2 = {}^{(\sqrt{n} - a)^2}/_{2a}$$

If n is prime (so that a = 1), one needs O(n) steps! But if n has a factor close to its square-root the method works quickly.

E.g. factorize 200819

Soln: $\lceil \sqrt{n} \rceil = 449$

449^2 - 200819 = 782 ≠ perfect square

450^2 - 200819 = 41^2

⇒200819 = (450 + 41)(450 - 41) = 491* 409

Factor Bases

Let $t^2 = s^2 = \text{mod } n \wedge t \neq \pm s \text{ mod } n$ then gcd (t + s,n) and gcd (t - s,n) gives nontrivial factors of n.

Proof: $t^2 = s^2$ mod n ⇒ n| $t^2 - s^2$ ⇒ n | (t + s)(t-s)

But $t \neq \pm s \, mod \, n \Rightarrow n \nmid (t+s) \wedge n \nmid (t-s)$

⇒gcd (t + s,n) and gcd (t - s,n) give nontrivial factors of n

Pollard's rho heuristic

As the procedure is only a heuristic, neither its running time nor its success is guaranteed. Given n it can factorize in $\sqrt[4]{n}$ time.

Pollard-Rho (n**)**

1. $i \leftarrow 1$
2. $x_1 \leftarrow RANDOM\ (0, n-1)$
3. $y \leftarrow x_1$
4. $k \leftarrow 2$
5. **while** TRUE
6. **do** $i \leftarrow i+1$
7. $x_i \leftarrow ({x_{i-1}}^2 - 1) mod\ n$

8. $d \leftarrow \gcd(y - x_i, n)$
9. **if** $d \neq 1$ *and* $d \neq n$
10. **then** print d **end if**
11. **if** $i = k$
12. **then** $y \leftarrow x_i$ **end if**
13. $k \leftarrow 2k$

14. **Endwhile**

The following sequences may be noted from the procedure above:

$x_i \leftarrow ({x_{i-1}}^2 - 1) mod\, n$

$k \leftarrow 2,4,8,16 \ldots$

$y \leftarrow x_1, x_2, x_4, x_8, \ldots$

Notice that as x_1 was initialized with a value in Z_n and all succeeding values in the sequence depend on the previous value, the sequence is bound to repeat after some values. Moreover, the dependence is a random function given by:

$f_n(x) = (x^2 - 1) mod\, n$

7.6 POLLARD RHOANALYSIS

Pollard Rho Analysis:

By the ***birthday-paradox,*** the sequence then must repeat after $\Theta(\sqrt{n})$steps in expectation. As will be shown below, a similar sequence of ${x_i}'s$for a prime factor of n will alsorepeatin $\Theta(\sqrt{p})$steps or $\Theta(\sqrt[4]{n})$steps in expectation because the greatest value of the smallest prime factor of n is less than$\sqrt{n}$.

Let p be a nontrivial factor of n, then the sequence $\langle x_i \rangle$induces acorrespondingsequence $\langle {x_i}' \rangle$ modulo pwhere

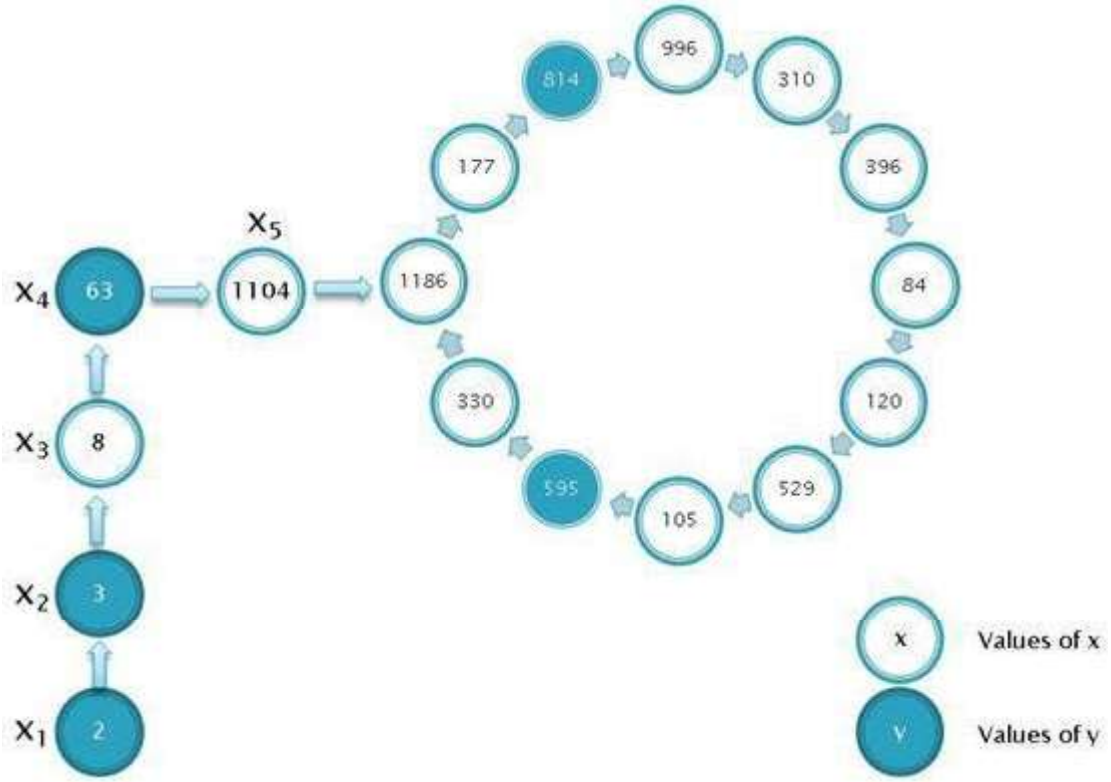
$x_i' = x_i mod\ p$ for all i

Now $x_{i+1}' = x_{i+1} mod\ p = f_n(x_i) mod\ p = \big((x_i{}^2 - 1) mod n\big) mod\ p$

$\Rightarrow x_{i+1}' = (x_i{}^2 - 1) mod\ p = ((x_i mod\ p)^2 - 1) mod\ p = ((x_i')^2 - 1) mod\ p$

$= f_p(x_i')$

Thus, althoughthesequence $\langle x_i' \rangle$ is not being computed explicitly, it is well defined and obeys the same recurrence as the sequence $\langle x_i \rangle$. By similar reasoning as for the original sequence, the sequence $\langle x_i' \rangle$ repeats in $\Theta(\sqrt{p})$. Consider the figure below for theillustration.



Let t denote the index of the first repeated value in the $\langle x_i' \rangle$ sequence, and let $u > 0$ denote the length of the cycle that has been produced.

i.e. t and $u > 0$ are the smallest values such that $x'_{t+i} = x'_{t+u+i}$ for all $i \geq 0$. Clearly the length t of the tail of the _Rho' and the length u of the cycle take the value $\Theta(\sqrt{p})$ inexpectation.

Also, if $x'_{t+i} = x'_{t+u+i}$ then $p \mid (x_{t+u+i} - x_{t+i})$ and $\gcd(x'_{t+u+i} - x_{t+i}) > 1$

When **Pollard-Rho** saves as y any value x_k such that k≥ t then y mod p will always remain on the cycle modulo p because future values will always be ones already on the cycle. Then, to ensure that line 8 of **Pollard-Rho** computes a nontrivial factor, all that is required is that $y \bmod p = x'_i$. This happens when k is set to a value greater than u which causes x_i to loop around all values in the cycle modulo p without a change in y. A factor of n is then discovered when x_i _runs into' the previously stored value of y modulop.

Since the expected values of both t and u are $\Theta(\sqrt{p})$, the expected number of steps to produce the factor p is $\Theta(\sqrt{p})$. For the smallest factor of n, p is less than $\sqrt{n}$ and hence the overall run time is $\sqrt{n}$ inexpectation.

Two reasons why the algorithm may not perform as expected:

- The heuristic analysis of the run time may result in the the cycle of values modulo p to be much larger than $\sqrt{p}$, in which case the algorithm performs correctly but slower thandesired.
- The divisors of n produced may not always be a trivial one like 1 orn.

Both these problem are found to be insignificant in practice.

Reference:

1. *Introduction to Algorithms* , Second Edition, T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein, *Prentice Hall India*.

2. A course in Number Theory and Cryptography, Neal Koblitz, Springer.

Block-3

Unit 1: Primality Testing 1

8.1 PRIMALITY TESTING

Primality Testing

Mathematicians have tried in vain to this day to discover some order in the sequence ofprime numbers, and we have reason to believe that it is a mystery into which the human mind will neverpenetrate. L.EULER.

Abstract: Our objective is to find a polynomial-time foolproof algorithm to determine whether a given integer is prime. Everyone knows trial division, in which we try to divide n by every integer m intherange $2 \leq m \leq \sqrt{n}$.The number of steps in this algorithm will be at least the number of integers m we consider, which issomethinglike $\sqrt{n}$ in the worst case (when nis prime).Notethat $\sqrt{n}$ isroughly $2^{\frac{d}{2}}$ where d is the number of digits of n when written in binary (and d is roughly log{n} where, here and throughout, we will take logarithms in base 2). We first present few basic algorithms for primality testing and then proceed with AKSalgorithm[2].

Introduction:

There are few better known or more easily understood problems in pure mathematics than the question of rapidly determining whether a given integer is prime. The problem of distinguishing prime numbers from composite numbers, and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length. Nevertheless we must confess that all methods that have been proposed thus far are either restricted to very special cases or are so laborious and difficult that even for numbers that do not exceed the limits of tables constructed by estimable men, they try the patience of even the practiced calculator. And these methods do not apply at all to larger numbers.

Primes come up in many different places in the mathematical literature, and some of these suggest ways to distinguish primes from composites. Those of us who are interested in primality testing always look at anything new with one eye open to this application, and yet finding a fast

primality testing algorithm has remained remarkably elusive. See [4] and [5] for probabilistic and randomized primality testing methods. The advent of the AKS algorithm makes us wonder whether we have missed some such algorithm, something that one could perform in a few minutes, by hand, on any enormous number.

The ultimate goal of this line of research has been, of course, to obtain an unconditional deterministic polynomial-time algorithm for primality testing. This is achieved by the AKS algorithm. Next we give the mathematical background to understand the algorithms for Primality testing.

Introduction to Jacobi Symbol:

Suppose we want to determine whether or not $x^2 \equiv a \pmod p$ has a solution, where p is prime, If p is small, we could square all of the numbers mod p and see if a is on the list. When p is large, this is impractical. If $p \equiv 3 \bmod 4$ we can find out the by using a technique in which we compute $s \equiv a^{(p+1)/4} \pmod p$. If a has a square root, then s is one of them, so we simply have to square s and see if we get a. If not, then a has no square root mod p. The following proposition gives a method for deciding whether a is a square mod p that works for arbitrary odd p.

Proposition: let *p* be a odd prime and let *a* be an integer with $a \not\equiv 0 \pmod p$. Then $a^{(p-1)/2} \equiv \pm 1 \pmod p$. The congruence $x^2 \equiv a \pmod p$ has a solution if and only if $a^{(p-1)/2} \equiv 1 \pmod p$.

Proof: Let $y \equiv a^{(p-1)/2} \pmod p$. Then $y^2 \equiv a^{p-1} \equiv 1 \pmod p$, by Fermat's theorem. Therefore, $y \equiv \pm 1 \pmod p$.

If $a \equiv x^2$, then $a^{(p-1)/2} \equiv x^{p-1} \equiv 1 \pmod p$. The hard part is showing the converse. Let g be a primitive root mod p. Then $a \equiv g^j$ for some j. If $a^{(p-1)/2} \equiv 1 \pmod p$, then

$$g^{j(p-1)/2} \equiv a^{(p-1)/2} \equiv 1 \pmod p.$$

Which implies $j.(p-1)/2 \equiv 0 \pmod{p-1}$. This implies that j must be even: $j=2k$. Therefore, $a \equiv g^j \equiv g^{(k)2} \pmod p$, so a is a square mod p.

Although the above proposition is easy to implement by a computer , it is rather difficult to use by hand. In the following we introduce the Legendre and Jacobi symbols, which gave us an easy way to determine whether or not a number is a square mod p. they are also very useful in Primality testing.

Let p be an odd prime and let a (not) $\equiv 0 \pmod p$. Define the Legendre symbol

$\left(\frac{a}{p}\right)$ = { 1 if $x^2 \equiv a \pmod p$ has a solution.

- 1 if $x^2 \equiv a \pmod{p}$ has a solution.

Some important properties of the Legendre symbol are given in the following.

Properties: Let p be an odd prime.

1. If $a \equiv b \not\equiv 0 \pmod{p}$, then

$$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$$

2. If $a \not\equiv 0 \pmod{p}$, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p}.$$

3. If $ab \not\equiv 0 \pmod{p}$, then

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right), \left(\frac{a}{pq}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

4. $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$.

5. $$\left(\frac{a}{p}\right) = \left(\frac{a \bmod p}{p}\right)$$

6. If m,n are both odd positive numbers then
$$\left(\frac{m}{n}\right) = (-1)^{(m-1)(n-1)/4}\left(\frac{n}{m}\right)$$

7. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}, \left(\frac{1}{n}\right) = 1, \left(\frac{0}{n}\right) = 0$

The properties above can be used to build a recursive algorithm to compute the Jacobi symbol efficiently. In fact, the algorithm is strongly reminiscent of Euclid's algorithm for the gcd.Here is how the algorithm applies to compute $\left(\frac{m}{n}\right)$:

- If m > n then use the invariance property: return $\left(\frac{m \bmod n}{n}\right)$.
- If m=0 or m=1, then use(7) : return 0 or1
- Factor m as $2^k l$, where l is odd. If k >0 use formulas (7) and (3) : return $\left(\frac{2}{n}\right)^{k \bmod 2}\left(\frac{l}{n}\right)$.
- Use reciprocity : if m=n=3 mod 4 then return $-\left(\frac{n}{m}\right)$; otherwise return $\left(\frac{n}{m}\right)$.

As this method is similar to Euclidean GCD algorithm, its complexity too isO ($\log^2(n)$).

The Jacobi symbol extends the Legendre symbol from primes p to composite odd integers n. One might define the symbol to be +1 if a is a square mod n and -1 if not. However, this would cause the property (3) to fail.

In order to preserve property (3), we define the Jacobi symbol as follows. Let n be an odd positive integer and let a be a nonzero integer with gcd (a, n) =1. Let

$$n = p_1^a p_2^b p_3^c \ldots\ldots p_r^q$$

be prime factorization of n. Then

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p1}\right)^a \left(\frac{a}{p2}\right)^b \left(\frac{a}{p3}\right)^c \ldots\ldots \left(\frac{a}{pr}\right)^q$$

The symbols on the right side are Legendre symbols introduced earlier. Note that if n=p, the right side is simply one Legendre symbol, so the Jacobi symbol reduces to the Legendre symbol.

Properties:

1. If $a \equiv b \pmod n$ and $\gcd(a, n) = 1$, then

$$\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$$

2. If $\gcd(ab, n) = 1$ then

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right).$$

3. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$.

4. $$\left(\frac{2}{n}\right) = \begin{cases} +1 \text{ if } n \equiv 1 \text{ or } 7 \pmod 8. \\ -1 \text{ if } n \equiv 3 \text{ or } 5 \pmod 8. \end{cases}$$

8. Let m be odd with $\gcd(m, n)=1$. Then

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{if } m \equiv n \equiv 3 \pmod 4. \\ +\left(\frac{n}{m}\right) & \text{otherwise.} \end{cases}$$

Before going into any of the primality tests we give a basic principle on which the tests depend upon.

Basic principle: let n be an integer and suppose there exist integers x and y with x2≡y2 (mod n), but x ≢ ±y (mod n). Then n is composite. Morover, gcd(x-y,n) gives a nontrivial factor of n.

Proof: Let d = gcd(x-y, n). if d= n then x ≡ y (mod n), which is assumed not to happen. Suppose d=1. A basic result on divisibility is that if a| bc and gcd(a,b) =1, then a |c.In our case, since n divides x2– y2 = (x-y) (x+y) and d=1, we must have that n divides x+y, which contradicts the assumption that x ≢ -y (mod n). Therefore d ≠ 1, n so d is nontrivial factor of n.

8.2 FERMAT PRIMALITY TEST

Fermat Primality Test: *Let n* >1 *be an integer. Choose a random integer a with* 1 < *a* < *n*-1. *If* $a^{n-1} \not\equiv 1$*(mod n) then n is composite. If* $a^{n-1} \equiv 1$ *(mod n), then n is probably prime.*

If we are careful about how we do this successive squaring, the Fermat test can be combined with the basic principle to yield the following stronger result.

Miller- Rabin Primality test:

Let n >1 *be an odd integer. Write a*-1 =$2^k m$ *with m odd. Choose a random integer a with* 1< *a*< *n*-1. *Compute* $b_0 \equiv a^m$ *(mod n). If* $b_0 \equiv \pm 1$ *(mod n), then stop and declare that n is probably prime. Otherwise, let* $b_1 \equiv b_0^2$*(mod n). If* $b_1 \equiv 1$ *(mod n), then n is composite (and gcd (*b_0 -1,*n) gives a nontrivial factor of n). If* $b_1 \equiv -1$ *(mod n), then stop and declare that n is probably prime. Otherwise, let* $b_2 \equiv b_1^2$*(mod n). If* $b_2 \equiv 1$ *(mod n), then n is composite. If* $b_2 \equiv -1$ *(mod n), then stop and declare that n is probably prime. Continue in this way until stopping and reaching* b_{k-1}. *If* $b_{k-1} \not\equiv -1$ *(mod n), then n is composite.*

The reason why the test works is- suppose, for example that $b_3 \equiv 1$ (mod n). This means that $b_2^2 \equiv 1$ (mod n). This means that $b_2^2 \equiv 1^2$ (mod n). Apply the basic principle from before. Either $b_2 \equiv \pm 1$ (mod n), or $b_2 \not\equiv \pm 1$ (mod n) and n is composite. In the latter case, gcd (b_2-1, n) give a nontrivial factor of n. In the former case, the algorithm would have stopped by the previous step.

MILLER-RABIN (n, s)

1. For j← 1 tos
2. do a ←RANDOM(1,n-1)
3. If WITNESS(a,n)

4. then returnCOMPOSITE
5. returnPRIME

WITNESS (a, n)

- Let <b_k,b_{k-1}…b_0> be the binary representation ofn-1.
- d←1
- for I ← k down to0
- do x ←d
- d←(d. d) modn
- if d=1 and x≠1 andx≠n-1
- then returnTRUE
- if(b_i=1)then
- d← (d. a) modn
- endfor
- ifd≠1
- then returnTRUE
- returnFALSE

If we reach b_{k-1} , we computed $b_{k-1} \equiv a^{(n-1)/2} \pmod n$. The square of this is a^{n-1}, which must be 1 (mod n) if n is prime, by Fermat's Theorem. Therefore, if n is prime, $b_{k-1} \equiv \pm 1 \pmod n$. All other choices mean that n is composite. Moreover, if $b_{k-1} \equiv 1$ then, if we didn't stop at an earlierstep, $b_{k-2}^2 \equiv 1^2 \pmod n$with $b_{k-2} \not\equiv \pm 1 \pmod n$. This means that n is composite (and we can factorn).

Although all prime numbers will be detected through this test, however the converse is not true. There are numbers which pass this test but are composite, i.e n is composite and $a^{n-1} \equiv 1 \pmod n$for all possible bases a. Such numbers are called Carmichael numbers. For example 561 is a Carmichael number. Carmichael numbers are usually of the form ($p_1.p_2.p_3$) where the number is product of primes.

An alternative and equivalent definition of Carmichael numbers is given by **Korselt's criterion**.

Theorem : A positive composite integer n is a Carmichael number if and only if n is square-free, and for all prime divisors p of n, it is true that $p - 1 \mid n - 1$.

For example:

561= 3.11.17 is square-free and 2 |560, 10|560, 16 |560.

1105= 5.13.17 is square-free and 4 |1104, 12|1104, 16 |1104.

Solovay-Strassen Primality test: *let n be an odd integer. Choose several random integers a with 1<a<n-1.if*

$$(\frac{a}{n}) \not\equiv a^{(n-1)/2} \pmod{n}$$

For some a, then n is composite. If

$$(\frac{a}{n}) \equiv a^{(n-1)/2} \pmod{n}$$

For all a, then n is probably prime.

Running time: $O((\log n)^3)$. This follows from running times of separate parts of the algorithm: finding gcd, computing of Jacobi symbol, and finally computing powers of *a*.

Respectively, $O((\log n)^2) + O((\log n)^2) + O((\log n)^3)$.

Definition 1. *For odd n > 3, wedefine* $E(n) = \{a \in Z_n^* \mid (\frac{a}{n}) = a^{\frac{(n-1)}{2}} \bmod n\}$

We will use the following lemma.

Lemma 2.1. *For odd n > 3, n is prime if and only if* $E(n) = Z_n^*$

For the proof of the lemma, refer to the book Randomized Algorithms [1], Lemma 14.30.

Theorem 2.2. *If n is an odd prime, and* a $\in \{1, \ldots, n-1\}$*, the probability that the algorithm returns "prime" is* $Pr_{a \in \{1,2,\ldots n-1\}}[Solovay-Strassen(n) = \text{“prime”}] = 1$.

If n is an odd composite, the probability that algorithm returns "composite" is

$Pr_{a \in \{1,2,\ldots n-1\}}$ $[Solovay-Strassen(n) = \text{“composite”}] \geq 1/2$

Proof: If *n* is an odd prime, then the algorithm will obviously always output‖prime‖. Let us now prove the second part of the theorem. Assume that *n* is an odd composite. We will show that the probability of the algorithm returning ‖prime‖ is ≤ 1/2

$Pr_{a \in \{1,2,\ldots n-1\}}$ $[Solovay-Strassen(n) = \text{“prime”}] =$

$Pr_{a \in \{1,2,\ldots n-1\}}$ $[\{gcd(a, n) = 1\} \in \{= a^{(n-1)/2} \bmod n\}] \frac{E(n)}{n-1}$

From Lemma 2.1 it follows that $E(n) \neq Z_n^*$

Now it is easy to show that $E(n)$ is a **subgroup** of the multiplicative group Z_n^*

$a, b \in E(n) \in (ab \bmod n) \in E(n)$

$a \in E(n) \Rightarrow a^{-1} \in E(n)$.

$E(n)$ is thus a proper subgroup of Z_n^*

and, from elementary group theory, we conclude that

$$|E(n)| \leq \frac{|Z_n^*|}{2} \leq \frac{(n-1)}{2}.$$

Thus $Pr_{a \in \{1,2,\ldots n-1\}}$ [*Solovay–Strassen*(*n*) =–*prime*‖]≤ ½

8.3 AKS PRIMALITY TEST

AKS PRIMALITY TEST:

First we describe a characterization of prime numbers that will provide the conceptual mathematical foundation for our polynomial time algorithm.

Lemma 3.1: Let a∈ Z, n ∈N, such that (a, n) = 1. Then n is prime iff $(x+a)^n \equiv x^n + a \pmod n$.

Proof:

By the Binomial theorem we have:

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

If n isprimethen $\binom{n}{k}$ is divisible by n according to the binomial theorem. By Fermat'slittle theorem, we have $a^n \equiv a \pmod n$ and hence the equivalence in the above equationholds.

If n is composite, then let q be a prime divisor of n with $q^s \mid n$. The coefficient of x^{n-q} in the binomial expansion of $(x+a)^n$ is $\frac{n(n-1)\ldots(n-q+1)}{q!} a^q$. The numerator is divisible by q^s but not by q^{s+1}. The denominator is divisible byq.Hence $\frac{n(n-1)\ldots(n-q+1)}{q!} a^q \neq 0 \pmod n$. Since (a,n) =1, implies $(a,q^s) = 1$, implies $(a^q, q^s) = 1$,implies $\frac{n(n-1)\ldots(n-q+1)}{q!} a^q \neq 0 \pmod n$.

Therefore $(x+a)^n \neq x^n + a \pmod n$

The above identity suggests a simple method for testing the primality of an integer n. We can choose an integer a such that (a, n) = 1 and calculate $f(x) = (x + a)^n - (x^n + a)$. If this function is equal to *0 (mod n)* then n is prime, else n is composite. Although this is certainly a valid

primality test, it is horribly inefficient as it involves the computation of n coefficients. The trick however is in choosing a suitable integer a. The simplest method for reducing the number of coefficients that need to be computed is to evaluate f(x) modulo n and modulo some polynomial of small degree, say (x^r - 1).

Although it is clear that all primes p satisfy $(x + a)^p - (x^p + a) \equiv 0 \bmod (p, x^r - 1)$, some composite numbers may satisfy this equation for all values of a and r. It turns out that for a judiciously chosen r, if the above identity is satisfied for several values of a, then n can be shown to be a prime power. The number of a's and the appropriate value of r are bounded by log(n). Therefore we have just described a deterministic polynomial time primality testing algorithm.

Algorithm:

INPUT: $n \geq 1$

STEP 1: If $a, b > 1 \in N$ such that $n = a^b$, then output COMPOSITE. STEP2: Find the minimal $r \in N$ such that $o_r(n) > \log^2(n)$

STEP3 : For a=1to r do

If $1 < (a,n) < n$, then output COMPOSITE

STEP4: if $r \geq n$, then output PRIME

STEP5: For a=1 to $\lfloor \sqrt{\varphi(r)} . \log n \rfloor$ do

If $(x+a)^n - (x^n + a) \neq 0 \bmod (n, x^r - 1)$, then output COMPOSITE.

STEP 6: output PRIME.

Proof: If n is prime, STEP 1 cannot return COMPOSITE. Similarly, STEP 3 cannot return COMPOSITE. Hence, the AKS algorithm will always return PRIME if n is prime.

Conversely, if the AKS algorithm returns PRIME, we will prove that n is indeed prime. If the algorithm returns PRIME in STEP 4, n must be prime because otherwise a non trivial factor a would have been found in STEP 3. The only case which remains is that if the algorithm returns PRIME in STEP 6.

Lemma: There exists an integer $r \in N$ with the following properties:

1. $r \leq \max\{3, \lceil \log^5(n) \rceil\}$
2. $O_r(n) > \log^2(n)$
3. $(r, n) = 1$

Proof:

For n=2, r=3 satisfies all the conditions.

For $n > 2,\ \lceil log^5(n) \rceil > 10.$

We know that for $n \geq 7$, $lcm(n) \geq 2^n$ where lcm(m) denote the LCM of first m numbers.

So we get the following:

$$lcm\left(\lceil log^5(n) \rceil\right) \geq 2^{\lceil log^5(n) \rceil}$$

Now consider: $N = n.\prod_{i=1}^{\left\lfloor log^2(n) \right\rfloor}(n^i - 1).$

Let r be the smallest integer not dividing N. then condition (2) is obviously satisfies as r is not divisor

(n^i-1) for $i \leq \lfloor log^2(n) \rfloor$. Condition (1) is also satisfiesbecause

$$N \leq n^{1+2+\cdots log^2(n)} = n^{\frac{1(log^4(n)+log^2(n))}{2}} < n^{log^4(n)} < 2^{log^5(n)}$$

Thus $N < lcm\left(\lceil log^5(n) \rceil\right)$ and hence $r < \lceil log^5(n) \rceil$

Now we prove (3). It is clear that (r,*n*) <r,as otherwise r would divide n and hence N. Thus

$\frac{r}{(r,n)}$is an integer lessthanmax $\{3, \lceil log^5(n) \rceil\}$ not dividing N. Because r was chosen to be minimal,it must be case that $\{3, \lceil log^5(n) \rceil\}$.hence we have found ther.

Because $O_r(n) > 1$, n must have some prime divisor p such that $O_r(p) > 1$. STEP 3 did not output COMPOSITE, so we know that (n, r)=(p, r) =1. Additionally, we know that p > r as otherwise STEP 3 or STEP 4 would have returned a decision regarding the primality n.

Hypothesis:

1. Natural number n has p as a prime divisor.
2. $r < \lceil log^5(n) \rceil$ and $O_r(n) > log^2(n)$
3. (r,n) =1
4. $l := \lfloor \sqrt{\varphi(r)}.\log n \rfloor$

We now focus our attention on STEP 5 of the algorithm. Let us define an introspective. For polynomial *f*(X) and number $m \in N$, we say that m is introspective for*f*(X) if

$$f(X)^m = f(X^m) \pmod{X^r - 1, p}.$$

Lemma: let n ∈ N haveprime divisor p and let a ∈ N with 0 ≤ a ≤ lif n,p are introspective for (x+a),then $\frac{n}{p}$ is introspective for (x+a) aswell.

Proof: As p and n are both introspective for (x+a), we have

$$(x^{\frac{n}{p}} + a)^p \equiv (x^n + a) \equiv (x+a)^n \equiv (x+a)^{p\frac{n}{p}} \pmod{x^r - 1, p}$$

$$(x^{\frac{n}{p}} + a)^p \equiv ((x+a)^{\frac{n}{p}})^p \text{ implies } (x^{\frac{n}{p}} + a)^p + (-((x+a)^{\frac{n}{p}})^p) \equiv 0 \pmod{p}$$

$$\text{Therefore, } ((x^{\frac{n}{p}} + a) - (x+a)^{\frac{n}{p}})^p \equiv 0 \pmod{p}$$

$$\text{Let } ((x^{\frac{n}{p}} + a) - \left(x+a)^{\frac{n}{p}}\right) = h.$$

We must show h ≡ 0(mod x^r-1).Because (r,p) =1, x^r-1 factors into distinct irreducible $h_i(x)$ over Z_p. Using the Chinese Remainder theorem , we get

$$h^p \in \prod_i \frac{Z(x)_p}{h(x)_i}$$

As x^r-1 divides h^p, each of the irreducible factors $h_i(x)$ divide h. Hence x^r-1 divides h. Hence the proof.

It is easy to see the introspective numbers are closed under multiplication and that the set of functions for which a given integer is introspective is closed under multiplication.

We can now state a fact as a consequence of the above results.

Every element if the set $I = \{\left(\frac{n}{p}\right)^i . p^j : i,j \geq 0\}$ is introspective for every polynomial in the set $\{\prod_{a=0}^{l}(x+a)^{e_a} \mid e_a \geq 0\}$. We now define two groups based on these sets that will play a crucial role in theproof.

1. $I_r = \{i(mod\ r) : i \in I\}$ This is a subgroup of Z^*_r since (n,r) =(p,r)=1. Let G be this group and |G|=t.G is generated by n and p modulo r and since $O_r(n) > \log^2(n)$, $t > \log^2(n)$.
2. $G = \{f(mod\ h(x), p) : f \in P\}$ Let $Q_r(X)$ be r^{th} cyclotomic polynomial over F_p . Polynomial $Q_r(X)$ divides X^r-1 and factors into irreducible factors of degree $o_r(p)$. Let h(X) be one such irreducible factor. Since $o_r(p) > 1$, the degree of h(X) is greater than one. The second group is the set of all residues of polynomials in P modulo h(X) andp. Let G be this group. This group is generated by elements X, X+1,X+2,…, X+l in the field F = F_pX/ (h(X)) and is a subgroup of the multiplicative group of F.

Lemma : $|G| \geq \binom{t+1}{t-1}$

Proof: Note that because h(x) is a factor of Q_r(X) , x is a primitive rth root of unity in F. We now show that if f, g ∈ P are distinct polynomials with degrees less than t, then they map to distinct elements in G.

Suppose, that $f(x)$ =g(x) in F. Let m ∈ I. Then m is introspective for f and g, so$f(x^m)$=g(x^m) within F. Then x^m is a root of j(z)=f(z)-g(z) for every m ∈I_r. We know,(m,r)=1, so each such x^m is a primitive rth root of unity. Hencethereare$|I_r| = t$ distinct roots of j(z) in F. But the degree j(z) < t by the choice of f and g. This contradiction (a polynomial cannot have more roots ina field than its degree) implies that $f(x)$≠ g(x) inF.

Notice that i≠ j in F_p whenever 1≤ i, j≤ l since $l = \lfloor \sqrt{\varphi(r)}.\log n \rfloor < \sqrt{r}.\log n < r < p$ Then by above ,x,x+1,x+2,x+3...x+l are a. Since the degree of h(x) is greater than 1, all of theselinear polynomials are nonzero in F. therefore there are atleast, l+1 distinct polynomials of degree 1 in G. hence there atleast $\binom{l+s}{s}$polynomials of degree s in G. Then the order of G is atleast $\binom{t+1}{t-1}$ hence theproof.

Lemma : If n is not a power ofpthen $|G| \leq n^{\sqrt{t}}$.

Proof: Consider the following subset of I:

$$I' = \left\{ \left(\frac{n}{p}\right)^i . p^j \;\middle|\; 0 \leq i,j \leq \lfloor \sqrt{t} \rfloor \right\}$$

If n is not a power of p,then $|I'| \geq (1 + \lfloor \sqrt{t} \rfloor)^2 > t$Since $|I_r| = t$, there are at least two elements of I‘ that are equivalent modulo r. Label these elements m_1,m_2 where m_1>m_2.

Then $x^{m_1} \equiv x^{m_2} (mod\, x^r - 1)$

Let f(x)∈P Then becausem_1,m_2 are introspective $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2}) = f(x)^{m_2} \;\; (mod\, x^r - 1, p).$

Thus $f(x)^{m_1} = f(x)^{m_2}$ in the field F. Thereforethepolynomial $Q(y) = y^{m_1} - y^{m_2}$ has atleast |G| roots in F (since f(x) ∈ P was arbitrary).Thenbecause $fracnp)^{\lfloor \sqrt{t} \rfloor}.p)^{\lfloor \sqrt{t} \rfloor}$ is the largest element ofI‘.

$$\deg(Q(y)) = m_1 \leq fracnp.p)^{\lfloor\sqrt{t}\rfloor} = n^{\lfloor\sqrt{t}\rfloor}$$

Itfollowsthat $|G| \leq n^{\sqrt{t}}$. Hence theproof.

Lemma*:* *If AKS algorithm return PRIME then n is prime.*

Proof: Assume that the algorithm return prime.Recallthat $|I_r| = t$ and is generated by n and p, therefore t≥ Or(n) > $\log^2(n)$or $t > \lfloor\sqrt{t}.\log(n)\rfloor$.

We know that $\lfloor\sqrt{\varphi(r).\log(n)}\rfloor$ *and* $|G| \geq \binom{t+1}{t-1}$

$$|G| \geq \binom{l+1+\lfloor\sqrt{t}\log(n)\rfloor}{\lfloor\sqrt{t}.\log(n)\rfloor}$$

$$|G| \geq \binom{2.\lfloor\sqrt{t}\log(n)\rfloor}{\lfloor\sqrt{t}.\log(n)\rfloor}$$

$$|G| \geq 2^{\lfloor\sqrt{t}\log(n)\rfloor+1}$$

$$|G| \geq 2^{\lfloor\sqrt{t}\log(n)\rfloor}$$

$$|G| \geq n^{\sqrt{t}}$$

Also by lemma, $|G| \leq n^{\sqrt{t}}$if p is not a power of p. Therefore it must be the case that n= p^k for some k>0 . But STEP 1 did not output COMPOSIT, so k=1, proving that n is indeed prime.This completes our proof of theorem.

Time Complexity:

The overall complexity of AKS algorithm is O ($log^{10.5}$(n)).

Conclusion: In this report we have presented the three important Primality testing algorithms, Miller- Rabin Test, Solovay –Strassen test, AKS algorithm. We also gave an introduction to the Jacobi symbol. The AKS algorithm is an unconditional deterministic polynomial time algorithm for Primality testing. It was first of its kind. The algorithm was a major breakthrough for Primality testing and in general for mathematics. The authors received many accolades, including the 2006 Godel prize and the 2006 Fulkerson Prize, for this work.

References:

1. R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge UniversityPress,1995.

2. M. Agarwal, N.Kayal and N.saxena , primes in P,Department of Computer Science Engineering, Indian Institute of technology Kanpur. Available from the World wide webhttp:// www.cse.iitk.ac.in/news/primality.pdf

3. Trappen and Washington, Introduction to Cryptography with CodingTheory.

4. G.L. Miller riemanns hypothesis and tests for Primality.

5. M.O. Rabin . Probabilistic algorithm for testingprimality.

Acknowledgement: Mr. Sai Sheshank Burra (B. Tech, CSE) [Scribe for Last Three Lectures]

Unit 2: Elliptic Curve Cryptosystem 2

9.1 ELLIPTICCURVES

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field.

Note: Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse.

Definition: Let K be a fieldofcharacteristic $\neq 2,3$, and let x^3+ax+b (where $a,b \in K$) be a cubic polynomial with no multiple roots. An ellipticcurveover K is the set ofpoints (x,y) with $x,y \in K$ which satisfy theequation

$$y^2 = x^3 + ax + b \tag{1}$$

together with asingle element O and called the–point at infinity‖.

If K is a field of characteristic 2, then an ellipticcurveover K is the set of points satisfying an equation of typeeither

$$y^2 + cy = x^3 + ax + b \tag{2}$$

or else,

$$y^2 + xy = x^3 + ax^2 + b \tag{3}$$

If K is a field of characteristic 3, then an ellipticcurveover is the set of points satisfying the equation

$$y^3 = x^3 + ax^2 + bx + c \tag{4}$$

Figure 1 shows two examples of elliptic curves. Now, consider the set of pointsE (a,b) consisting of all of the points (x,y) that satisfy Equation (1) together withtheelement O. Using a different value of the pair (a,b) results in a different set E(a,b).

Using this terminology, the two curves in Figure 1 depict the sets $E(-1,0)$ and $E(1,1)$, respectively.

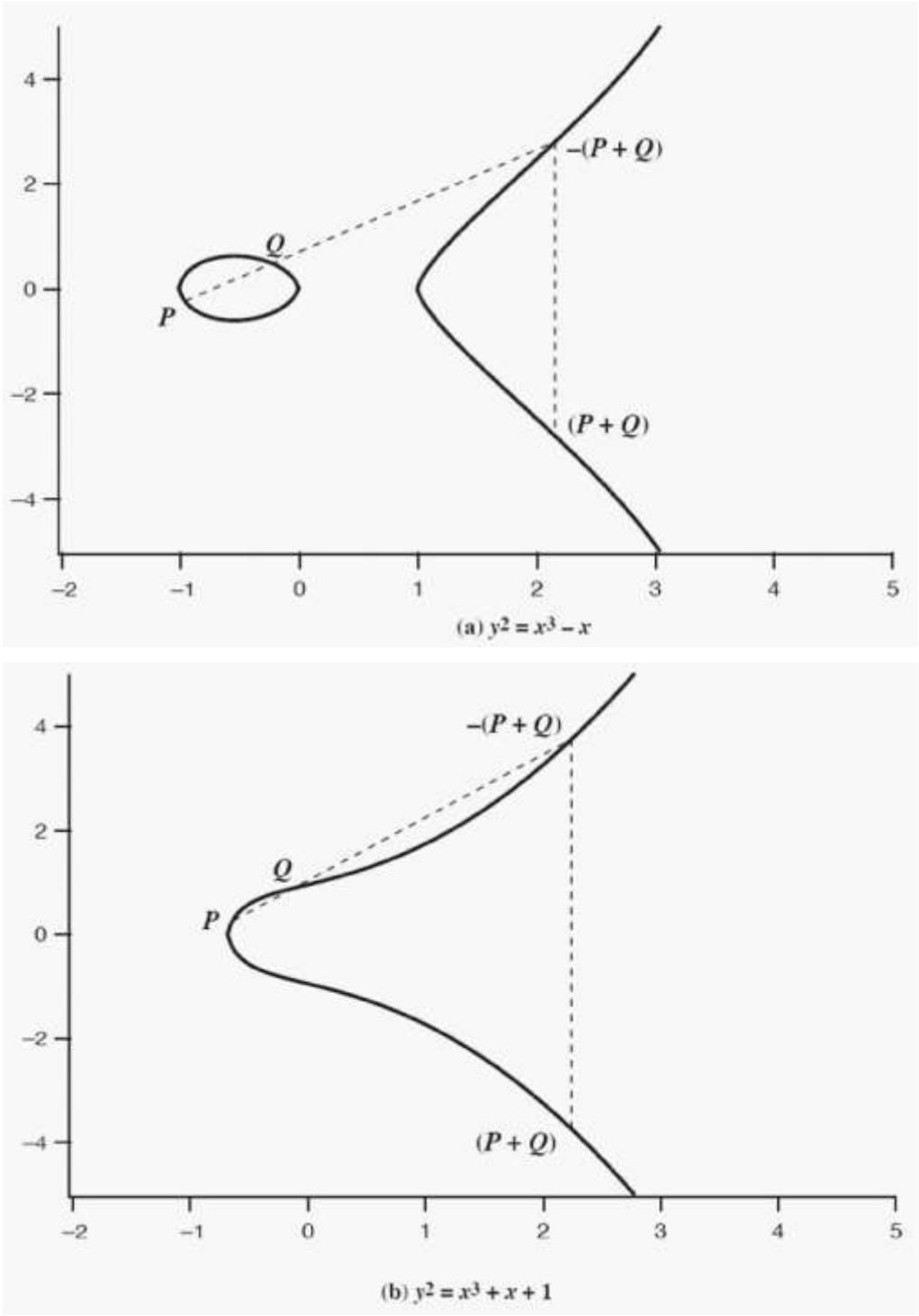


Figure 1 Examples of Elliptic Curves

Geometric Description of Addition :

A group can be defined based on the set E $(a,b)$for specificvaluesof $a$and $b$in Equation (1), provided the following condition ismet:

$$4a^3 + 27b^2 \neq 0 \quad (5)$$

To define a group, we define an operation, called addition and denoted by +, for the set $E^{(a,b)}$, where a and b satisfy Equation (5). In geometric terms, the rules for addition can be stated as follows: If three points on an elliptic curve lie on a straight line, theirsumis O.

From this definition, we can define the rules of addition over an elliptic curve:

i. O serves as the additive identity. Thus $O = -O$ and for any point P on the elliptic curve, $P + O = P$. In what follows, we assume $P \neq O$ and $Q \neq O$.

ii. The negative of a point P is the point with the same x coordinate but the negative of the y coordinate; that is; if $P = (x, y)$, then $-P = (x, -y)$. Note that these two points can be joined by a vertical line. Note that $P + (-P) = P - P = O$.

iii. To add two points P and Q with different x coordinates, draw a straight line between them and find the third point of intersection R that is the point of intersection (unless the line is tangent to the curve at either P or Q, in which case we take $R = P$ or $R = Q$, respectively). To form a group structure, we need to define addition on these three points as follows: $P + Q = -R$. That is, we define $P + Q$ to be the mirror image (with respect to the x axis) of the third point of intersection. Figure 1 illustrates this construction.

iv. The geometric interpretation of the preceding item also applies to two points, P and $-P$, with the same x coordinate. The points are joined by a vertical line, which can be viewed as also intersecting the curve at the infinity point. We therefore have $P + -P = O$.

v. To double a point Q, draw the tangent line and find the other point of intersection S. Then $Q + Q = 2Q = -S$.

Let (x_1, y_1), $(x_2, y_2)$and $(x_3, -y_3)$denote thecoordinates of P,Q,and Rrespectively. We want to express $x_3$and y_3 interms of x_1, y_1, x_2, y_2.

Let $y = \alpha x + \beta$be the equation of the linepassingthrough $P$and Q.

$$\therefore \alpha = \frac{y_2 - y_1}{x_2 - x_1}$$

The equation of the elliptic curve$y^2 = x^3 + ax + b$is

$$(\alpha x + \beta)^2 = x^3 + ax + b$$

$$\Rightarrow x^3 - (\alpha x + \beta)^2 + ax + b = 0$$

Roots of theequation are x_1, x_2, x_3.

$$(x-\alpha)(x-\beta)(x-\gamma)=0$$

$$\Rightarrow\ x^3-(\alpha+\beta+\gamma)x^2+(\alpha\beta+\beta\gamma+\gamma\alpha)x-\alpha\beta\gamma=0$$

Sum of the roots = $(\alpha+\beta+\gamma)$

Addition of two points:

$$x_1+x_2+x_3=\alpha^2$$

$$\therefore x_3=\alpha^2-x_1-x_2$$

$$y_3=\alpha x_3+\beta$$

$$y_1=\alpha x_1+\beta$$

$$\therefore \beta=y_1-\alpha x_1$$

$$y_3=\alpha x_3+y_1-\alpha x_1$$

$$\therefore y_3=\alpha(x_3-x_1)+y_1$$

$$\therefore -y_3=\alpha(x_1-x_3)-y_1$$

$$-R=(x_3,y_3)$$

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2$$

$$y_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1$$

9.2 ELLIPTIC CURVES (CONTD.) AND FINITEFIELDS

$$y^2=x^3+ax+b \quad \ldots\ldots \quad P=Q=(x_1,y_1)$$

$$\therefore 2y\frac{dy}{dx}=3x^2+a$$

$$\therefore \frac{dy}{dx}=\frac{3x^2+a}{2y}=\alpha$$

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

$R = (x_3, -y_3)$ is the point of intersection of the tangent at P and the elliptic curve.

Example : On the elliptic curve $y^2 = x^3 - 36x$ let $P = (-3,9)$ and $Q = (-2,8)$. Find $P + Q$ and $2P$.

Solution .

$P = (-3,9)$.... $Q = (-2,8)$

For finding $P + Q$,

$$\alpha = \frac{8-9}{-2+3} = \frac{-1}{1} = -1$$

$$x_3 = \alpha^2 - x_1 - x_2$$

$$= 1 + 3 + 2$$

$$= 6$$

$$y_3 = \alpha(x_3 - x_1) - y_1$$

$$= -1(-3-6) - 9$$

$$= 9 - 9$$

$$= 0$$

$$\therefore P + Q = (6,0)$$

For finding $2P$,

$$\alpha = \frac{3x_1^2 - 36}{2y_1}$$

$$= \frac{3(9) - 36}{2(9)}$$

$$= -\frac{1}{2}$$

$$x_3 = \alpha^2 - 2x_1$$

$$= \frac{1}{4} - 2(-3)$$

$$= \frac{25}{4}$$

$$y_3 = -y_1 + \alpha(x_1 - x_3)$$

$$= -9 + \left(-\frac{1}{2}\right)\left(-3 - \frac{25}{4}\right)$$

$$= -\frac{35}{8}$$

$$\therefore 2P = \left(\frac{25}{4}, -\frac{35}{8}\right)$$

Elliptic curves over Z_p:

For elliptic curves over Z_p, wehave

$$y^2 = (x^3 + ax + b) \bmod p \tag{6}$$

Now consider the set $E_p(a,b)$consisting of all pairsofintegers (x,y)that satisfy Equation (6), together with a pointat infinity O.Thecoefficients $a$and $b$and thevariables $x$and $y$are all elements ofZ_p.

It can be shown that a finite abelian group can be defined based on the set$E_p(a,b)$providedthat $(x^3 + ax + b) \bmod p$has no repeated factors. This is equivalent to thecondition

$$(4a^3 + 27b^2) \bmod p \neq 0 \bmod p \tag{7}$$

For example, let $a = 1,\ b = 1$ and $p = 23$, that is, the ellipticcurve $E_{23}(1,1)$: $y^2 = x^3 + x + 1 \ (\text{mod } 23)$. Forthe set $E_{23}(1,1)$, we are only interested in thenonnegative integers in the quadrant from $(0,0)$through $(p-1, p-1)$that satisfy theequation mod P. Table 1 lists the points (other than$O$) that arepart of $E_{23}(1,1)$. Figure 2 plots thepoints of $E_{23}(1,1)$.

In case of the finite group $E_p(a,b)$, the numberofpoints $N$is bounded by

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

Table 1 Points on theEllipticcurve $E_{23}(1,1)$otherthan O

(0,1)	(6,4)	(12,19)
(0,22)	(6,19)	(13,7)
(1,7)	(7,11)	(13,16)
(1,16)	(7,12)	(17,3)
(3,10)	(9,7)	(17,20)
(3,13)	(9,16)	(18,3)
(4,0)	(11,3)	(18,20)
(5,4)	(11,20)	(19,5)
(5,19)	(12,4)	(19,18)

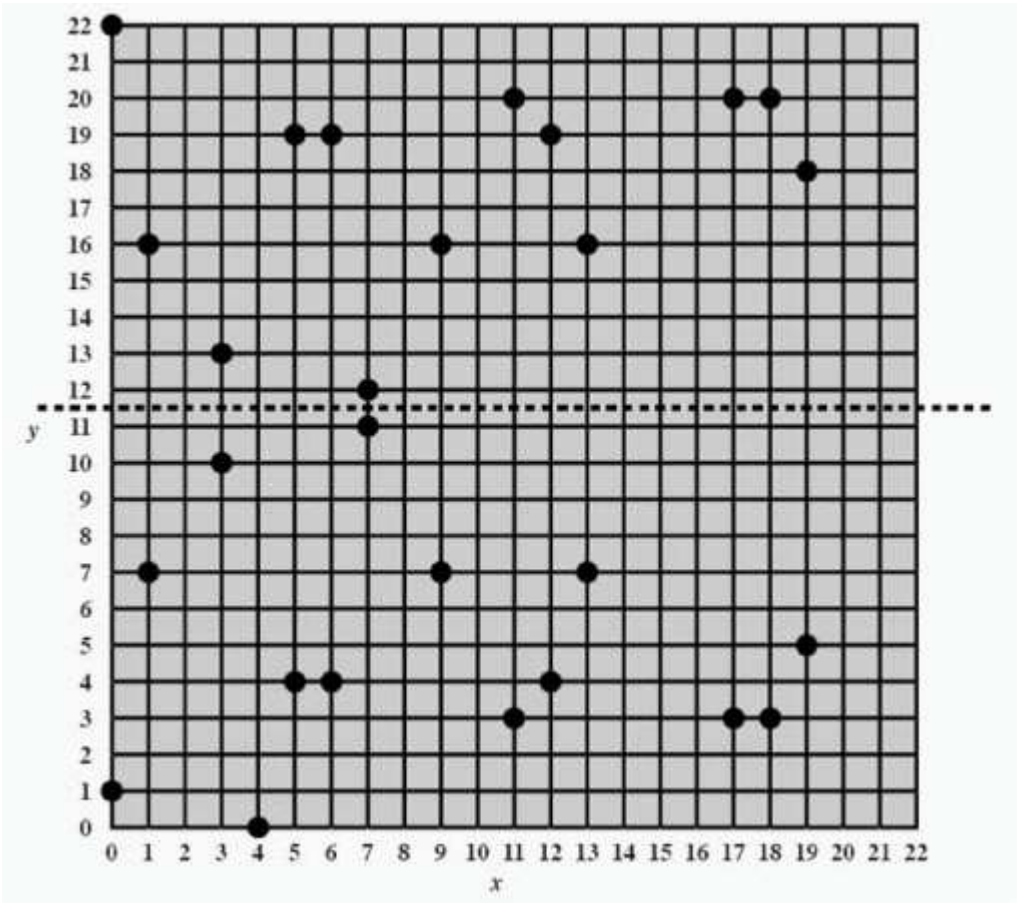


Figure 2 The Elliptic Curve $E_{23}(1,1)$

9.3 ECDLP

The Elliptic Curve cryptosystem (***ECC***) have the potential to provide relatively small block size, high security public key schemes that can be efficiently implemented. The Elliptic Curve Discrete Logarithm problem (***ECDL P***) is based on the fact that given *m.P* for some integer *m* and some point *P* on the Elliptic Curve where *P* is known, we have to find v alue of *m* . The smaller key size of Elliptic Curve Cryptosystem makes possible much more compact implementations for a given level of security , which means faster cryptographic operations, running on smaller chips or more compact software. We mainly concentrate on the Elliptic Curve whose equation is given by $y^2 = x^3 + Ax + B$ defined over a finite field $F_p$for prime *p* for *A* , *B* in the field. The ECC transforms data into some point representation of the Elliptic Curve. It relies on calculating the multiple of a point *P* as *m.P* which is public and it is difficult to find integer *m* from *P* and *m.P* . This is the Elliptic Curve Discrete Logarithm Problem (***ECDL P***). It basically defines a group by the operator addition on the points found on the EllipticCurve.

Informally a zero-knowledge proof system allows one person to convince another person of some fact without revealing any information about the proof. There are usually two participants, the prover and the verifier. The prover would like to prevent the verifier from gaining any useful information while participating in the protocol. For some details refer [3] and [8].

An Elliptic Curve is defined on a field. The field may be finite or infinite. We will draw our attention towards finite fields. It is denoted by $F_q$having q elements where $q = p^r$having p as the characteristic of the field $F_q$and r as any positive integer. We will mainly consider for the curve where $q = p$ i.e. $r = 1$. The points on the curve whose x and y values are in the field are taken into account. The ECC transforms the data into some point representation. The points form an Abelian Group w.r.t. the operator addition. There is one point indicated by O called the identity element.

Definition 2.1. The *Order* of a point is defined as the number of times the point must be added in order to give the identity element i.e. the point O .

Definition 2.2. The *Generator* of the group is a point whose Order is equal to the number of points that are in the group.

The basis of ***ECC*** is The Elliptic Curve Discrete Logarithm Problem i.e. the ***ECDL P*** .

Definition 2.3. The *Elliptic Curve Discrete Logarithm problem or* ***ECDLP*** is defined as follows: Given points P and Q on $E_p(A, B)$ such that the equation $m.P = Q$ holds. Compute k given P and Q .

Definition 2.4. The *Zero Knowledge Proof* is defined as follows:

There are usually two participants, the prover and the verifier. The prover knows some fact and wishes to prove that to the verifier. The prover and the verifier will be allowed to perform alternatively the following computations:

1. Receive message from the other party.
2. Perform a privatecomputation.
3. Send a message to the other party.

A typical *round* of the protocol will consist of a *challenge* by the verifier and a *response* by the prover. At the end the verifier either accepts or rejects.

Definition 2.5. The *Birthday Paradox* is defined as follows :

How many people must there be in a room before there is a 50% chance that two of them were born on the same day of the year.

The above problem can be stated in a different way as follows :

Given a random variable that is an integer with uniform distributions between 1 and n and a selection of k instances ($k = n$) of the random variable, what is the probability $p(n, k)$ that there is at least one duplicate ? The *Birthday Paradox* is a special case where $n = 365$ and asks for the value of k such that $p(n, k)>0 . 5$. The answer to this problem is

$k \approx O(\sqrt{n})$. [6]

Definition 2.6. The *M odular Linear Equation* is stated as $ax = b$ (*mod n*) where $a >0$ and $n >$ 0.

Review of Existing Results

Let E be an Elliptic Curve defined over a finite field with $F p$ having equation $y^2= x^3+ Ax + B$, where A &B satisfies the inequality $4 A^3+ 27 B^2= 0$. We can find the number of points on the curve by checking the Legendre Symbol for $y^2$for each value of x . T he number of points will be denoted by $\#E(F_p)$.

The Hasses's theorem provides some limit on the number of points on an Elliptic Curve defined over a finite field. It states that $|p +1-\#E (F_p) |\leq 2\sqrt{p}$. [5]

Theorem 3.1. *The Equation* $ax \equiv b$ (*mod n*) *is solvable for the unknown x if and only if* $gcd(a, n)|b$. *[4]*

Theorem 3.2. *The Equation* $ax \equiv b$ (*mod n*) *either has d distinct solutions modulo n, where* $d = gcd (a, n)$ *,or it has no solutions. [4]*

Theorem 3.3. *Let* $d = gcd (a, n)$ *, and suppose that* $d = ax^f+ ny^f$ *for some int e gers* x^f *and* y^f. *If* $d|$ *b, then the e quation* $ax \equiv b$ (*mod n*) *has as one of it's solutions the value* x_0, *where* $x_0= x^f(b/d)$ *mod n.[4]*

Theorem 3.4. *Sup p ose that the e quation* $ax\equiv b$ (*mod n*) *is solvable (that is,* $d|b$ *, where* $d =$ *gcd* (*a,n*)) *and that* x_0*is any solution to this e quation. Then, this e quation has exactly ddistinct solutions, modulo n, given by* $x_i= x_0+ i(n/d)$ *for* $i = 0 , 1 , 2 ,3 ,..... , d - 1$ *.[4]*

Corollary 3.5. For any $n >1$, if $gcd (a, n) = 1$, then the equation $ax \equiv b(mod\ n)$ has a unique solution, modulo n . [4] In particular if $b = 1$ then $x = a^{-1}n \in \mathbb{Z}^*$.

Theorem 3.6. *In a coin toss, if the probability of obtaining a head is p then it is expected that after* 1 */p tosses the first head is obtained. [2]*

Theorem 3.7. $\forall n>1, \varphi(n)/n= \Omega(\log \log n/ \log n)$. *[2]*

First we will provide a *Zero Knowledge Proof* for *Elliptic Curve Discrete Logarithm Problem* (ECDL P) and explain the properties. In the next section we will present an attack over the Zero Knowledge Protocol.

Properties of Zero Knowledge Interactive Proof

A *Zero Knowledge Interactive Proof (ZKIP)* or *Zero Knowledge Protocol* is an iteracti ve method for one party to prove to another that a (usually mathematical) statement is true without revealing anything other than the veracity of the statement. A *Zero Knowledge Interactive Proof* must satisfy three properties :

1. *Completeness* : If the statement is true, the honest verifier (that is, one following theprotocol property) will be convinced of this fact by an honestprover.
2. *Soundness* : If the statement is false, no cheating prover can convince the honest verifier that it is true except with small probability.
3. *Zero-Knowledge* : If the statement is true, no cheating verifier learns anything other thanthis fact.

9.4 ZERO KNOWLEDGEPROOF

Now we will give the *Zero Knowledge Proof* for *Elliptic Curve Discrete Logarithm Problem(ECDLP)* and prove the properties.Our proof has some resemblance with ElGamal signature scheme [1] descri bed in [5] in details . Let the prover be *Alice* and the verifier be *Bob* . Let the Elliptic Curve be denoted by $E_p(A,B)$ and let n be the number of points on the Elliptic Curve. Let $P \in E_p(A,B)$ be a generator of the group. So Alice wants to convince Bob that she knows the value of m where $Q = mP$ without disclosing m . It can be achieved by following steps :

1. Alice picks random integer k with $1 = k = p$-1 where p is the characteristic of the fieldand sends $R = kP$toBob.
2. Bob picks random integer r with $1 = r = p$-1 and sends it toAlice.
3. Alice computes $Y = (k\text{-}mr)\ mod\ n$ where n is the number of points on the curve i.e. $\# E(F_p) = n$, and sends it to Alice.
4. Bob verifies if $R == YP + rQ$.

If *step* 4 is satisfied then Bob accepts else rejects. Now we will verify the three properties stated previously for the protocol as follows:

1. *Completeness* : Given $Q = mP$. We have to show that if Alice knows value of m , then Bob is convinced that Alice knowsit.

Since Alice knows value of m , all four steps in the protocol can be carried out. At *step* 3 Alice computes $Y = (k - mr) \bmod n$ and sends it to Bob. At *step* 4, Bob verifies $YP + rQ = R$ or not.Now

$YP + rQ = (k - mr)\, P + rQ = kP - rmP + rQ = kP - rQ + rQ = kP = R$

(verified).

So Bob is convinced that Alice knows$m$.

2. *Soundness* : Here we have to show that if Alice does not know value of m then shecan't convince Bob that she knows it or succeeds with a very small probability.

Now suppose Alice doesn't know value of m and wants to convince Bob that she knows it. The only way that Alice can convince Bob is in *step* 3 of the protocol Alice should send such a value for Y such that $Y P$ should have value $R- rQ$, so that after

adding rQ Bob will get R .

i.e. $YP = R-rQ$

i.e. $YP = kP -rmP$

i.e. $YP = (k -mr)P$

i.e. $Y= (k -mr) \bmod n$

Now Alice has values of k , r but she doesn't have the value of m . So it can't find value of $k-mr$. So she can't cheat.

3. *Zero-Knowledge* : Here we have to show that no information is released in the pro-tocol.

Now in one session of the protocol Bob/Eavesdropper E has the followinginformation:

P ,Q , $R = kP$,r ,$Y = (k - mr) \bmod n$.

Now from $Y = (k - mr) \bmod n$, in order to find out value of m it knows value of r . So the only thing left is to know k . But to find k the only way is to solve the *ECDLP* , $R = kP$ for k . So Bob/Eavesdropper can't know value of m . So the proof is a *Perfect Zero-Knowledge* .

Attack on the Zero Knowledge Protocol

During the whole protocol the Eavesdropper E has the following information :

point P (known)

point $Q = mP$ (known)

point $R = k\, P$ (known) (k unknown)

number r (known)

number $Y = (k - mr) \bmod n$ (known, m unknown)

From it the Eavsdropper can't find any useful information. But the attack is possible if the attacker uses information from multiple sessions of the challenge-response protocol. Now suppose in one session

$$Y_1 = (k - mr_1) \bmod n \quad (1)$$

In another session Alice use the same k to compute R and thus

$$Y_2 = (k - mr_2) \bmod n \quad (2)$$

So (1) - (2) ? $Y_1 - Y_2 = m(r_2 - r_1) \bmod n$

$$\Rightarrow m(r_2 - r_1) = (Y_1 - Y_2) \bmod n \quad (3)$$

So $r_2 - r_1$ is known, and $Y_1 - Y_2$ is known. So we can solve for m by using *Theorem* 3.4. Here in the *Modular Linear Equation* $ax \equiv b \bmod n$, $a = (r_2 - r_1)$, $b = (Y_1 - Y_2)$, $x = m$ and the number of solutions = $gcd(a,n)$. The attack proceeds as follows :

In *step* 1 of the protocol Eavesdropper E gets the value of $R_i = k_i P$($i = 1 , 2 , 3, ...$) where i denotes the session numbers of the challenge-response protocol. Suppose at some session j , E disc overs $R_j = R_l$, for some $l < j$. Thus we have :

$k_j P = k_l P \Rightarrow (k_j - k_l)P = O$. We will assume P is either the generator or a point on the Elliptic Curve with high order. Otherwise *ECDLP* can be easily solved by any brute force method. Thus we can safely assume without loss of generality $O(P) >> k_j - k_l$. Thus the only way the equality holds if $k_j = k_l$. Thus the entire problem reduces to solving the *Modular Linear Equation* (3). From *Corollary* 3 . 1 of *Modular Linear Equation* we hence

$m = (Y_1 - Y_2)(r_2 - r_1) - 1 \bmod n$.

As stated in *Corollary*3.1,$(r_2 - r_1)$-1 would be uniquely defined if $gcd (r_2 - r_1, n) = 1$.

Let$\Delta = r_2 - r_1$. Thus $gcd(\Delta , n) = 1$. W e can adopt the following randomized algorithm to compute Δ and thus $r2$ from r_1 .

Algorithm 1 RAND (n)

1: Pick a random number x from (2 ,3 , ..., n -1).

2: Compute $gcd(x,n)$.

3: if $gcd (x,n) = 1$ then

4: Set $\Delta \leftarrow x$.

5: else

6: goto *step* 1.

7: end if

8: Return Δ.

We know that $|Zn^*| = \Phi(n)$. Thus the tptal number of integers less than n and relativelyprime w.r.t n is $\Phi(n)$. Thus the probability that the selected number $x \in Z n^*$ in RAND step 1 is $\Phi(n)/n$. Thus from Theorem 3.6 after expected $n / \Phi(n) \in O(\log n / \log\log n)$ iterations we will get $x \in Zn^*$. Thus the expected time complexity of RAND is $O(\log n / \log\log n)$ assuming the time complexity to compute $gcd(x,n)$ is $O(l\ og\ n)$. Thus in sessions i and j attacker will use a random number r_1 and $r_2 = r_1 + \Delta$. Now we can clearly see that this attack will fail if Alice chooses different values of k at each session. But in *step* 1 of the protocol Alice picks up k with $1 \leq k \leq p\text{-}1$ at random. Thus from *Birthday Paradox* after $O(\sqrt{p})$ sessions Alice will pick up k used in some earlier session with high probability. Thus after $O(\sqrt{p})$ sessions of the challenge response protocol with high prpbability an Eavesdropper can compute the value m for ***ECDLP*** .

6 Solution to Overcome the Above Attack

In this section we will provide a solution i.e., a modified Zero Knowledge Proof for the ECDLP that overcomes the above attack and prove the required properties i.e., Com- pleteness, Soundness, and Zero-Knowledge, as explained previously . We also provide an explanation of how it overcomes the above attack.

Let the proverbe *Alice* and the verifier be *Bob* . Let the Elliptic Curve be denoted by $E_p(A, B)$ and let n be the number of points on the Elliptic Curve. Let $P \in E_p(A, B)$ be a generator of the group. So Alice wants to convince Bob that she knows the value of m where $Q = mP$ without disclosing m . It can be achieved by following steps :

1. Alice picks random integers $k_1$and $k_2$with $1 \leq k_1, k_2 \leq p - 1$ where p is the c haracteristic ofthe field and sends $R_1 = k_1P$ and $R_2 = k_2Q$ toBob.

2. Bob picks random integer r with $1 \leq r \leq p - 1$ and sends it toAlice.

3. Alice computes $Y = (mrk_2 - k_1) \bmod n$ where n is the number of points on the curvei.e. $\#E(F_p) = n$, and sends it toAlice.

4. Bob verifies if $YP + R_1 == rR_2$.

If *step* 4 is satisfied then Bob accepts else rejects. Now we will verify the three propertiesstated previously for the protocol as follows:

1. *Completeness* : Given $Q = mP$. We have to show that if Alice knows value of m , then Bob is convinced that Alice knowsit.

Since Alice knows value of m , all four steps in the protocol can be carried out. At *step* 3 Alice computes $Y = (mrk_2\text{-}k_1) \ mod \ n$ and sends it to Bob. At *step* 4, Bob verifies $YP + R_1 = rR_2$or not.

Now $YP + R_1 = (mrk_2 - k_1)P + k_1P$

$= mrk_2P - k_1P + k_1P$

$= mrk_2P$

$= rk_2Q$ (Replacing mP by Q)

$= rR_2$(Replacing k_2Q by R_2) (Verified). So Bob is convinced that Alice knows m .

2. *Soundness* : Here we have to show that if Alice does not know value of m then shecan't convince Bob that she knows it or succeeds with a very small probability.

Now suppose Alice doesn't know value of m and wants to convince Bob that she knows it. The only way that Alice can convince Bob is in *step* 3 of the protocol Alice should send such a value for Y such that YP should have value $rR_2 - R_1$, so that after adding $R1$ Bob will get rR_2.

i.e. $YP = rR_2 - R_1$ i.e. $YP = k_2rQ - k_1P$ i.e. $YP = mrk_2P - k_1P$

i.e. $Y = (mrk_2 - k_1) \ mod \ n$

Now Alice has values of r ,k_2,$k_1$but she doesn't have the value of m . So it can't find value of $(mrk_2 - k_1)$. So she can't cheat.

3. *Zero-Knowledge* : Here we have to show that no information is released in the pro-tocol.

Now in one session of the protocol Bob/Eavesdropper E has the following information : $P,Q,R_1 = k_1P$, $R_2 = k_2Q$,$Y = (mrk_2 - k_1) \ mod \ n$.

Now $Y = (mrk_2 - k_1) \ mod \ n$. From this modular equation to find out value of m the known quantities are r and Y. In this modular linear equation $Y = (mrk_2 - k_1) mod \ n$ we have 3 unknowns $m,k_1,$ and k_2. Thus 2 ECDLPs $R_1 = k_1P$ and $R_2 = k_2Q$ reduces to solving $Y = (mrk_2 - k_1) \ mod \ n$. Thus in other words if there is an efficient way of obtaining $k_1$and $k_2$from the modular linear equation $Y = (mrk_2 - k_1) \ mod \ n$ then there is an efficient solution to 2ECDLPs $R_1 = k_1P$ and $R_2 = k_2Q$. Hence solving the modular linear equation $Y = (mrk_2 - k_1) \ mod \ n$is at least as hard as

solving ECDLPs $R_1 = k_1 P$ and $R_2 = k_2 Q$. So Bob/Eavesdropper can't know value of m . So the proof is a *Perfect Zero-Knowledge* .

Now we will explain how the attack is avoided. Now suppose as earlier *Bob / E* gets Y_1 and Y_2 as follows :

$Y_1 = (mr_1k_{12} - k) \bmod n$ and $Y_2 = (mr_2k_{22} - k) \bmod n$ i.e., in both sessions R 1 values are same. Here k_{12} and k_{22} indicate the k_2 values in both sessions.Now subtracting as previously we will get $Y_1 - Y_2 = m(r_1k_{12} - r_2k_{22}) \bmod n$. But as it doesn't know the value of k_{12} and k_{22}, so it can't solve for the *Modular Linear Equation* .Even if R_2 is same in b oth cases with $R_2 = k_2Q$ then it will get the final subtraction result as $Y_1 - Y_2 = mk_2(r_1 - r_2) \bmod n$. So solving it will give the value of mk_2. Again if we can obtain m efficiently we have an efficient solution to the ECDLP $R_2 = k_2Q$. Thus again we have a reduction from ECDLP to the problem of computing m from mk_2. So this proof system is not susceptible to the previous attack.

Conclusion

The Elliptic Curve cryptosystem (***ECC***) can play an important role in asymmetric cryptography . ***ECC*** is a stronger option than the **RSA** and Discrete Logarithm systems for the future. Here we have presented a Zero Knowledge In teractive Proof for ***ECDLP*** where the elliptic curve is of the form $E_p(A,B)$ where p is a prime. The re- sult can be easily generalized to $E_q(A,B)$ for composite q where $q = p^r$. Given a guess of m for ***ECDLP*** we can easily verify in polynomial time whether $P = m.Q$. This shows $\boldsymbol{ECDLP \in NP \subseteq PSPACE = IP}$ [7]. This confirms with our result that shows $\boldsymbol{ECDLP \in IP}$. Subsequently we have also presented an attack on the Zero Knowledge Pro-tocol using *Birthday Paradox* . Lastly we modified the Zero Knowledge Proof to overcome thisattack.

References

[1] T.ElGamal,‖APublicKeyCryptosystemandaSignatureSchemebasedon DiscreteLogarithm‖,IEEETransactionsonInformationTheory,July1985.

[2] Pinaki Mitra, M.Durgaprasad Rao, M. Kranthi Kumar, ‖Algorithms to Compute a Generator of the Group (Z_p^*, x_p) and Safe Primes‖, International Journal of Information Processing (Accepted forPublication).

[3] Steven G. Krantz, ‖Zero Knowledge Proofs‖, July2007.

[4] T.H. Coreman, C.E. Leiserson, R.L. Rivest, C. Stein,‖Introduction to Algo - rithms(Second Edition)‖', pp. 869-872.

[5] Lawrence C. Washington,‖Elliptic Curves - Number Theory and Cryptography‖, pp - (164-168), CHAPMAN & HALL/C R C,2003.

[6] William Stallings, ‖Cryptography and Network Security‖, Prentice Hall of India,2003.

[7] Adi Shamir, ‖IP = PS P A CE‖, Journal of the ACM (JACM), Volume 39 , Issue 4, pp869 - 877, October1992.

[8] PinakiMitraandSantoshSwain,—ZeroKnowledgeInteractiveProofforEllipticCurve Discrete Logarithm Problem‖, IJAC, Vol. 1(2010), pp. 1 –5.

Reference:

1. *A course in Number Theory and Cryptography,* Neal Koblitz ,*Springer.*

2. *Introduction to Cryptography with Coding Theory* , W. Trappe and L. C.Washington, *Pearson Education* .

3. *Cryptography and Network Security* , William Stallings, *Prenctice Hall India*.

9.5 ELLIPTIC CURVECRYPTOGRAPHY

Elliptic curve cryptosystem is based Elliptic Curve Discrete Logarithm Problem ,i.e., **ECDLP** . The problem is defined as follows:

Given points P and Q on $E_p(a,b)$ such that the equation $kP = Q$ holds. Compute k given P and Q .

Representing Plaintext Message by a Point on the Elliptic Curve

Suppose the plaintext message is an integer m. We have to represent this by a point on the elliptic curve $y^2 = x^3+ax+b \pmod p$. We choose the x -coordinate of the representative point by m . But it may so happen that $m^3+am+b \pmod p$ is not a quadratic residue and thus the ordinate value is undefined. So we adopt the following randomized procedure described in [1].

Let K be the largest integer such that the failure probability $1/2^k$ is acceptable. We also assume that $(m+1)K<p$. the message m will be represented by a point with the abscissa value $x = mK + j$,where0 $\leq j < K$. Also we assumethat$p \equiv 3 \bmod 4$. This assumption will help us in computing the square root deterministically. For j=0,1,2,…,K -1 check if $z=x^3+ax+b \pmod p$ is a quadratic residue or not. If it is a quadratic residue we compute the vale of y as $z^{\frac{p+1}{4}} \bmod p$.Nowwe

represent the message by P_m= (x,y). If the test fails for all values of j then we fail to map the message to a point. Clearly the failure probability is $1/2^k$.

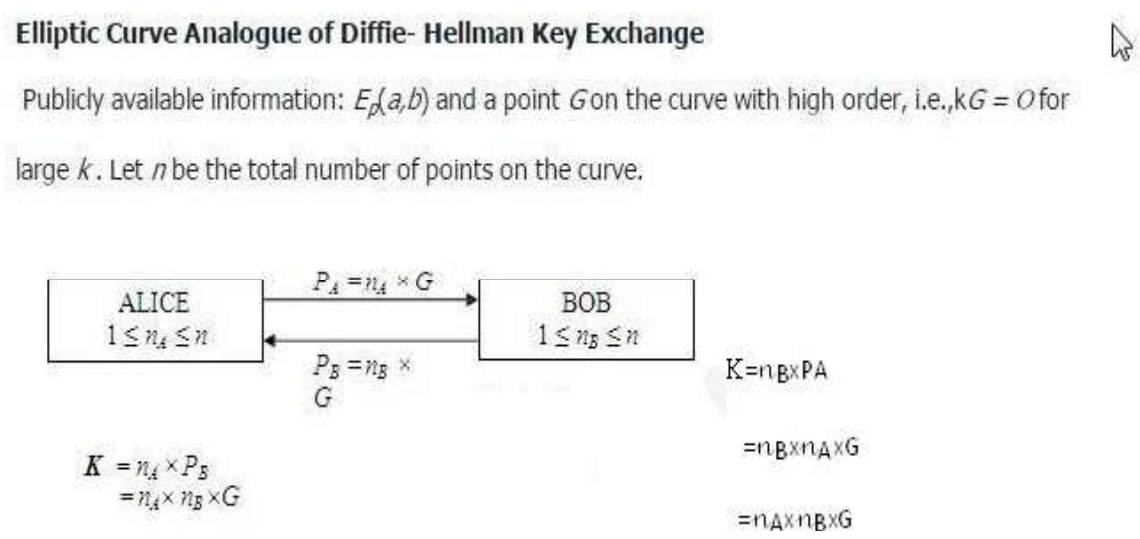
At the time of decryption we recover the message m from P_m= (x,y) as follows:

$m = \left\lfloor \frac{x}{K} \right\rfloor$.

Elliptic Curve Analogue of Diffie- Hellman Key Exchange

Publicly available information: $E_p(a,b)$ and a point G on the curve with high order, i.e.,k$G = O$ for

large k . Let n be the total number of points on the curve.



1. Alice chooses her private key $n_A$suchthat 1 $\le n_A \le n$ and computes the public key P_A=n_AxG.

2. Bob chooses his private key $n_B$suchthat1 $\le n_B \le n$ and computes the public key P_B=n_BxG.

3. Alice and Bob simultaneously compute the shared key $\boldsymbol{K} = n_A x n_B x G$ after computingn_AxP_B and n_Bx$P_A$respectively.

This key exchange scheme as mentioned earlier is susceptible to intruder-in-the-middle attack. To overcome this all messages should be authenticated by its sender.

Elliptic Curve Analogue of ElGamal Cryptosystem

Bob's Public Key: P_B

Bob's Secret Key: a where P_B=αG .

Other Publicly Available Information: Elliptic Curve $E_p(a,b)$ and a point G of large order on the elliptic curve and the prime p .

Encryption (Sender: Alice)

Let $P_m$be the point on the elliptic curve corresponding to the plaintext message m .

- Alice chooses a random number k , suchthat $1 \leq k \leq p$ -1.
- She computes the cipher text $C = \{C_1, C_2\} = \{ kG, P_m + kP_B \}$.
- She sends the cipher text $C = \{C_1, C_2\}$ toBob.

Decryption (Receiver: Bob)

After receiving the cipher text $C = \{C_1, C_2\}$

- Bob computes $\alpha C_1 = \alpha kG = k\alpha G = kP_B$
- Then Bob subtracts the result obtained in Step1. from C_2. Thus Bob computes $C_2 - kP_B = P_m$ and recovers the plaintext.

Reference:

1. *A course in Number Theory and Cryptography,* Neal Koblitz ,*Springer.*
2. *Introduction to Cryptography with Coding Theory* , W. Trappe and L. C. Washington, *Pearson Education* .
3. *Cryptography Theory and Practice* , D. R. Stinson, *CRC Press*.

Unit 3: Hash Function Digital Signatures

3

10.1 CRYPTOGRAPHIC HASHFUNCTIONS

In cryptography, a **cryptographic hash function** is a transformation that takes an input and returns a fixed-size string, which is called the hash value. Hash functions with this property are used for a variety of computational purposes, including cryptography. The hash value is a concise representation of the longer message or document from which it was computed. The message digest is a sort of "digital fingerprint" of the larger document. Cryptographic hash functions are used to do message integrity checks and digital signatures in various information security applications, such as authentication and message integrity.

There is no formal definition which captures all of the properties considered desirable for a cryptographic hash function.

A cryptographic hash function $h : M \rightarrow Z$ is a mapping from the set of messages of arbitray length i.e., the domain M to a set of fixed length (approx. 160 bits) message digests i.e., the range Z .

These properties below are generally considered prerequisites:

- *Preimage resistant* (See *one way function* for a related but slightly differentproperty): given *h(m)* it should be hard to find any m' such that $h\ (\ m'\ ) = h\ (\ m)$.
- *Second preimage resistant* : given an input m_1 , it should be hard to find another input,m_2 (not equal to m_1) such that $h\ (\ m_1\ ) = h\ (\ m_2\ )$.

This property is implied by collision-resistance. Second preimage resistance is sometimes referred to as weak collision resistance .

- *Collision-resistant* : it should be hard to find two different messages m_1 and m_2 suchthat $h\ (\ m_1\ ) = h\ (\ m_2\ )$. This property is sometimes referred to as strong collision resistance.

Birthday Paradox: If there are n people having m possible birthdays and if $n > \sqrt{m}$ (approx.)

then with high probability (i.e., probability > $\frac{1}{2}$) there will be a pair of people having the same birthday.

Proof: The probability that all people having distinct birthday (assuming $m > n$) is as follows:

$$= \frac{m(m-1)(m-2)\ldots(m-n+1)}{m^n} \quad 1\left(1-\frac{1}{m}\right)\left(1-\frac{2}{m}\right)\ldots\left(1-\frac{n-1}{m}\right)$$

$$\leq e^{-\frac{1}{m}} e^{-\frac{2}{m}} \ldots e^{-\frac{n-1}{m}} = e^{-\frac{n(n-1)}{2m}} \pounds \frac{1}{2} \Rightarrow \text{The probability that there is a pair ofpeople}$$

having the same birthday $\geq \frac{1}{2}$.

Thus $\frac{n(n-1)}{2m} \geq ln\ (2) \Rightarrow n(n-1) \geq 2\ ln\ (2)m \Rightarrow n \geq \sqrt{2\ln(2)m}$ (approx.)**Q.E.D**

Thus to check the strong collision resistance property of a hash function $h : M \rightarrow Z$ wherethe output is β bits, i.e., $|Z| = 2^{\beta}$ we have to test an arbitrary subset of M with cardinality β/2 for collision. So to make this computation difficult for an hacker β is usually set to 160bits.

It is however, a common misconception that "one-wayness" of a cryptographic hash function means irreversibility of processing of the hash state, and that it somehow contradicts the principles used to construct block ciphers. Such "irreversibility" in fact means presence of local collisions that could facilitate attacks. The hash function must be a permutation processing its state bijectively to be cryptographically secure. It must be irreversible regarding the data block just like any block cipher must be irreversible regarding the key (it should be impossible to find the key that can encrypt a block A into a block B faster than the brute-force). This makes iterated block ciphers and hash functions processing blocks of the same size as secret keys of those block ciphers virtually identical, except the roles of key and data blocks are swapped. All the attacks against the MDx and SHA families of hash functions exploit local collisions in the processing of the data block. The local collisions caused by the final addition operation can also be exploited by these attacks.

MDx Hash Function Family

The family of **MDx** hash function started from **MD4** and subsequently extended to **MD5** and **MD7** . We first explain the principle of **MD4** . **MD4** converts a message block whose length is

modulo 512 bit long to a message digest of 128 bits concatenating contents of 4 registers after 3 rounds. First given a bit string x of arbitrary length it converts it a message M whose length is modulo 512 bits. This is done as follows:

1. $d = (447-|x|) \bmod 512$
2. Let l denote the binary representation of $|x| \bmod 264$. $|l|=64$
3. $M = x \| 1 \| 0^d \| l$.

In the above algorithm $|x|$ denote the length of the bit string x . Thus we see that $|x \| 1 \| 0^d| =$ 448 mod 512. Concatenating l we get $|M|$ as a multiple of 512.

Then M is broken up into words of length 32 bits as follows:

$M = M[0] M[1] \ldots M[N-1]$

Where each $M[i]$ is 32 bit long and N º 0 mod 16. The overall algorithm proceeds as follows:

1. 1. $A = 67452301_{hex}$
   2. B = $efcdab89_{hex}$
   3. C = $98badcfe_{hex}$
   4. D = 10325476_{hex}
   5. ***for i = 0 to N/16 -1 do***
   6. ***for j = 0 to 15 do***
   7. $X[jj] = M[16i + j]$
   8. $AA = A$
   9. $BB = B$
   10. $CC = C$
   11. $DD = D$
   12. **Round1**
   13. **Round2**
   14. **Round3**
   15. $A = A + AA$
   16. $B = B + BB$
   17. $C = C + CC$
   18. $D = D + DD$

We maintain 4 registers A , B , C , D each of length 32 bits. In each iteration of the outer for loop we process a message block $X[0] X[1] \ldots X[15]$ of length 512 bits to produce a message digest

of length 128 bits formed by concatenating the contents of those 4 register A , B , C , D .

The above algorithm of **MD4** was subsequently extended to **MD5** that works in 4 rounds instead of 3 rounds.

10.2 ELGAMAL DIGITALSIGNATURES

1. Introduction:-

Traditionally signature with a message is used to give evidence of **identity** and **intention** with regard to that message. For years people have been using various types of signature to associate their identity and intention to the messages. Wax imprint, seal, and handwritten signature are the common examples. But when someone need to sign a digital message, things turn different. In case of signing a digital document one cannot use any classical approach of signing, because it can be forged easily. Forger just need to cut the signature and paste it with any other message.
For signing a digital document one uses **digital signature** [1][2][3].
Therefore, digital signature are required not to be separated from the message and attached to another. That is a digital signature is required to be both message and signer dependent. For validating the signature anyone can verify the signature, so digital signature are suppose to be verified easily.
A digital signature scheme typically consist of three distinct steps:

1. **Key generation:-** User compute their public key and corresponding privatekey.
2. **Signing:-** In this step user sign a given message with his/her privatekey.
3. **Verification:-** In this step user verify a signature for given message and publickey.

So the functionality provided by digital signature can be stated asfollows:
Authentication:- Digital signature provides authentication of the source of the messages as a message is signed by the private key of the sender which is only known to him/her.
Authentication is highly desirable in many applications.
Integrity:- Digital signature provides integrity as digital signature uniquely associate with corresponding message. i.e. After signing a message a message cannot be altered if someone do it will invalidate the signature. There is no efficient method to change message and its signature to produce a new message and valid signature without having private key. So both sender and receiver don‘t have to worry about in transit alteration.
Non- repudiation:- For a valid signature sender of message cannot deny having signed it.

In this report we are going to discuss different variation of digital signature. First we will describe **RSA digital signature scheme** and **Elgamal signature scheme**, along with their elliptic curve version. After covering above signature scheme we will talk about **digital signature standards**, and then we will cover **proxy signature scheme**, **blind signature scheme** and then we will finally talk about **short signature scheme**.

2. RSA Digital SignatureScheme

Suppose Alice want to send a *message**(m)** to* Bob. She can generate digital signature using RSA digital signature scheme [4] as follow:

Key Generation:-

She can generate key for RSA signature scheme:

1. Choose two distinct large prime numbers p and q.
2. Compute $n = pq$.
3. n is used as the modulus for both the public and privatekeys.
4. Compute $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totientfunction.
5. Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$.
6. Compute $d = e - 1 \bmod \varphi(n)$.

Then the public key and private key of user will be ***(e, n)*** and ***(d, n)*** respectively.

Now she have her public and private key. Now she can generate the signature of a message by encrypting it by her private key.

So she can generate signature corresponding to message***(m)*** as follow:

Signing:-

1. Represent the message ***m*** as an integer between ***0*** and ***n −1***.
2. Sign message by raising it to the ***dth*** power modulo***n***.

$$S \equiv m^d \pmod{n}$$

So ***S*** is the signature corresponding to message m. Now she can send message ***m***along with the signature ***S*** to Bob.

Upon receiving the message and signature ***(m, S)***, Bob can verify the signature by decrypting it by Alice public key as follow:

Verification:-

1. Verify signature by raising it to the ***eth*** power modulo***n***.

$m' \equiv S^e \pmod n$

2. If $\boldsymbol{m' = m \pmod n}$ then signature is valid otherwise not.

For a valid signature both ***m***and ***m'*** will be equalbecause:

$S \equiv m^d \pmod n$

$m' \equiv m^{de} \pmod n$

and

e is inverse of ***d,*** i.e. $\boldsymbol{ed \equiv 1 (mod\ \Phi(n))}$.

So, by using above algorithm Alice can generate a valid signature ***S*** for her message ***m,*** but there is a problem in above define scheme that is the length of the signature is equal to the length of the message. This is a disadvantage when message is long.

There is a modification in the above scheme. The signature scheme is applied to the **hash of the message,** rather than to the message itself.Now Alice have a message signature pair ***(m, S)***. So, the signature ***S*** is a valid signature for message ***m***. So a forger (lets say Eve) cannot forge Alice signature. i.e. She cannot use signature ***S*** with another message lets say $\boldsymbol{m_1}$, because $\boldsymbol{S^e}$is not equal to $\mathbf{m_1}$**.** Even when the signature scheme is applied to the hash of the message it is infeasible to forge the signature, because it is infeasible to produce two message ***m,*** $\boldsymbol{m_1}$with same hash value.

In practice, the public key in RSA digital signature scheme is much smaller than the private key. This enable a user to verify the message easily. This is a desired because a message may be verified more than once, so the verification process should be faster than signing process.

The RSA Digital Signature Algorithm:-

Additional instructions for RSA signature algorithm is as follows:

An RSA digital signature key pair consists of an RSA private key, which is used to compute a digital signature, and an RSA public key, which is used to verify a digital signature. An RSA digital signature key pair shall not be used for other purposes (e.g. key establishment).

An RSA public key consists of a modulus *n*, which is the product of two positive prime integers ***p*** and ***q*** (i.e., $\boldsymbol{n = pq}$), and a public key exponent ***e***. Thus, the RSA public key is the pair of values ***(n, e)*** and is used to verify digital signatures. The size of an RSA key pair is commonly

considered to be the length of the modulus ***n*** in bits (***nlen***). The corresponding RSA private key consists of the same modulus ***n*** and a private key exponent ***d*** that depends on ***n*** and the public key exponent ***e***. Thus, the RSA private key is the pair of values ***(n, d)*** and is used to generate digital signatures. In order to provide security for the digital signature process, the two integers ***p*** and ***q***, and the private key exponent ***d*** shall be kept secret. The modulus n and the public key exponent ***e*** may be made known to anyone.

The Standard specifies three choices for the length of the modulus (i.e., *nlen*): 1024, 2048 and 3072 bits.

An approved hash function, as specified in [7], shall be used during the generation of key pairs and digital signatures. When used during the generation of an RSA key pair, the length in bits of the hash function output block shall meet or exceed the security strength associated with the bit length of the modulus ***n***. The security strength associated with the RSA digital signature process is no greater than the minimum of the security strength associated with the bit length of the modulus and the security strength of the hash function that is employed. Both the security strength of the hash function used and the security strength associated with the bit length of the modulus ***n*** shall meet or exceed the security strength required for the digital signature process.

10.3 BLIND & PRONY SIGNATURE

Elgamal digital signature scheme[5] is proposed by Elgamal in 1985. This is based on Diffe-Hellman key exchange. This signature scheme is quite different from RSA signature scheme in terms of validity of signatures corresponding to a message. i.e. there are many valid signatures for a message. Suppose Alice want to sign a message using Elgamal digital signature scheme, she can generate signature ***S*** corresponding to message ***m*** as follow:

Key generation:-

She can generate key for Elgamal signature scheme as follow:

1. Choose ***p*** be a largeprime.
2. Choose ***g*** be a randomly chosen generator of the multiplicative group of integers ***Zp***.
3. Choose a secret key ***x*** such that ***1 < x < p −1***.
4. Compute $y = g^x \ (modp)$.

Then the public key and private key of user will be ***(p, g, y)*** and ***(p, g, x)*** respectively.

Signing:-

Now Alice has her public and private key so she can sign a message ***m*** by using following steps:

1. Choose a random number ***k*** such that $0 < k < p - 1$ and $gcd(k, p - 1) = 1$.
2. Compute $r \equiv g^k (mod p)$.
3. Compute $s \equiv (H(m) - xr)k^{-1} (mod\ p - 1)$. Where ***H(m)*** is hash of message.

Then the pair ***(r, s)*** is the signature of the message***m***.

Verification:-

Bob can verify the signature *(r, s)* of message *m* as follow:

1. Download Alice's public key ***(p, g,y).***
2. Compute $v_1 \equiv g^{H(m)} (mod\ p)$ and $v_2 \equiv y^r r^s (mod p).$
3. The signature is declared valid if and only if $v_1 \equiv v_2 (mod p).$

For a valid signature ***(r, s),*** $v_1 \equiv v_2 (mod\ p)$ since

$$s \equiv (H(m) - xr)k^{-1} (mod\ p - 1)$$

$$sk \equiv (H(m) - xr)(mod\ p - 1)$$

$$H(m) \equiv (sk + xr)(mod\ p - 1)$$

$$v_1 \equiv g^{H(m)} (mod\ p)$$

$$v_1 \equiv g^{(sk+xr)} (mod p)$$

$$v_1 \equiv g^{(sk)} g^{(xr)} (mod\ p)$$

$$v_1 \equiv (g^k)^s (g^x)^r (mod\ p)$$

$$v_1 \equiv y^r r^s (mod\ p)$$

$$v_1 \equiv v_2 (mod\ p).$$

The security of Elgamal digital signature scheme relies on the difficulty of computing discrete logarithms. The security of the system follows from the fact that since ***x*** is kept private for forging Elgamal digital signature one do need to solve discrete logarithm problem.s

Suppose Eve want to forge Alice signature for a message $m_1$and she doesn't know ***x*** (as ***x*** kept private by Alice), then she cannot compute ***s(*** as $s \equiv (H(m_1) - xr)k^{-1} (mod\ p - 1))$. Now the only option left is to choose ***s*** which satisfies the verification. Thus ***s*** should satisfy equation $y^r r^s \equiv g^{H(m)} (mod\ p)$ as Eve knows ***(p, g, y)*** so she can compute ***r***. So the equation can be rearrange as $r^s \equiv y^{-r} g^{H(m)} (mod\ p)$, which is again a discrete logarithm problem. So Elgamalsignature

scheme is secure, as long as discrete logarithm are difficult to compute.

Digital Signature Standards

Digital signature standards [6] define some standards to be followed. A digital signature scheme includes a signature generation and a signature verification. Each user has a public and private key and is the owner of that key pair.

For both the signature generation and verification processes, the message (i.e., the signed data) is converted to a fixed-length representation of the message by means of an approved hash function. Both the original message and the digital signature are made available to a verifier.

A verifier requires assurance that the public key to be used to verify a signature belongs to the entity that claims to have generated a digital signature (i.e., the claimed signatory). That is, a verifier requires assurance that the signatory is the actual owner of the public/private key pair used to generate and verify a digital signature. A binding of an owners identity and the owners public key shall be effected in order to provide this assurance.

A verifier also requires assurance that the key pair owner actually possesses the private key associated with the public key, and that the public key is a mathematically correct key. By obtaining these assurances, the verifier has assurance that if the digital signature can be correctly verified using the public key, the digital signature is valid (i.e., the key pair owner really signed the message). Digital signature validation includes both the (mathematical) verification of the digital signature and obtaining the appropriate assurances.

Technically, a key pair used by a digital signature algorithm could also be used for purposes other than digital signatures (e.g., for key establishment). However, a key pair used for digital signature generation and verification as specified in this Standard shall not be used for any other purpose. A number of steps are required to enable a digital signature generation or verification capability in accordance with Standards.

Initial Setup:-

Each intended signatory shall obtain a digital signature key pair that is generated as specified for the appropriate digital signature algorithm, either by generating the key pair itself or by obtaining the key pair from a trusted party. The intended signatory is authorized to use the key pair and is

the owner of that key pair. Note that if a trusted party generates the key pair, that party needs to be trusted not to masquerade as the owner, even though the trusted party knows the private key.

After obtaining the key pair, the intended signatory (now the key pair owner) shall obtain assurance of the validity of the public key and assurance that he/she actually possesses the associated private key.

Digital Signature Generation:-

Prior to the generation of a digital signature, a message digest shall be generated onthe information to be signed using an appropriate approved hashfunction.

Using the selected digital signature algorithm, the signature private key, the messagedigest, and any other information required by the digital signature process, a digital signature shall be generated according to theStandard.

The signatory may optionally verify the digital signature using the signature verification process and the associated public key. This optional verification serves as a final check to detect otherwise undetected signature generation computation errors; this verification may be prudent when signing a high-value message, when multiple users are expected to verify the signature, or if the verifier will be verifying the signature at a much later time.

Digital Signature Verification and Validation:-

In order to verify a digital signature, the verifier shall obtain the public key of the claimed signatory,(usually) based on the claimed identity. A message digest shall be generated on the data whose signature is to be verified (i.e., not on the received digital signature) using the same hash function that was used during the digital signature generation process. Using theappropriate digital signature algorithm, the domain parameters (if appropriate), the public key and the newly computed message digest, the received digital signature is verified in accordance with this Standard. If the verification process fails, no inference can be made as to whether the data is correct, only that in using the specified public key and the specified signature format, the digital signature cannot be verified for thatdata.

Before accepting the verified digital signature as valid, the verifier shall have

1. assurance of the signatory claimedidentity,
2. assurance of the validity of the public key,and

3. assurance that the claimed signatory actually possessed the private key that was usedto generate the digital signature at the time that the signature wasgenerated.

If the verification and assurance processes are successful, the digital signature and signed data shall be considered valid. However, if a verification or assurance process fails, the digital signature should be considered invalid.

10.4 SHORT SIGNATURE SCHEMEI

Suppose Alice want her message to be sign by Bob without letting him know the content of the message, she can got it done using Blind signature scheme [8]. Blind signatures scheme, proposed by Chaum, allow a signer to interactively sign messages for users such that the messages are hidden from the signer. Blind signature typically have two basic security properties: blindness says that a malicious signer cannot decide upon the order in which two messages have been signed in two executions with an honest user, and unforgeability demands that no adversarial user can create more signatures than interactions with the honest signer took place.

Blind signatures are typically employed in privacy-related protocols where the signer and message author are different parties. Blind signature schemes see a great deal of use in applications where sender privacy is important, some of them are:

1. Cryptographic election systems(e-Vote).
2. Digital cash schemes (e-Cash)

Blind signature scheme can be used with RSA signature algorithm. In RSA signature scheme a signature is computed by encrypting the message by the private key. In case of the blind signature there is one additional step **Blinding the message**. Alice can blind her message and get is signed by Bob, and remove the blinding factor after getting it signed. Suppose ***(e, N)*** and ***(d, N)*** is the public key and private key of Bob respectively then Alice can blind her messageas follows:

Blinding the message:-

1. Alice choose a random value ***r***, such that *r* is relatively prime to ***N (i.e. gcd(r, N) =1)***.

2. Calculate blinding factor by raising ***r*** to the public key ***e*** *(mod **N**)* (i.e. blinding factoris equal to $r^e \pmod N$).
3. Blind the message by computing the product of the message and blinding factor,i.e.

$$m' \equiv mr^e \pmod N$$

Now Alice can send blinded message ***m'*** to Bob. Now ***m'*** does not leak any information about ***m***, as ***r*** is private to Alice. Any malicious user need to solve discrete logarithm problem for recovering original ***m*** from ***m'***.

Signing:-

When Bob (signing authority) receive a blinded message from Alice (user) he will sign the message by his private key

$$S' \equiv (m')^d \pmod N$$

S' is the signature corresponding to message ***m'***. Bob send ***S'*** to Alice. Alice removes the blinding factor from the signature by dividing it ***r*** and revel the original RSA signature ***S*** as follow:

$$S \equiv S'r^{-1} \pmod N$$

Now Alice message ***m*** with signature ***S***, signature can be verified using Bob's public key.

Verification:-

Now signature can be verified as usual RSA signature.

1. Verify signature by raising it to the ***eth*** power module***N***.

$$m' \equiv S^e \pmod N$$

2. If $m' = m \pmod N$ then signature is valid otherwisenot.

The above scheme will work fine. i.e. ***(S, m)*** is a valid signature message tuple corresponding to Bob. Since

$$\begin{aligned} S &\equiv S'r^{-1} \pmod N \\ &\equiv (m')^d r^{-1} \pmod N \\ &\equiv (mr^e)^d r^{-1} \pmod N \\ &\equiv m^d r^{ed} r^{-1} \pmod N \\ &\equiv m^d r r^{-1} \pmod N \\ &\equiv m^d \pmod N \end{aligned}$$

5. Proxy Signature:-

In proxy signature scheme a user Alice (original signer) delegates her signing capability to another user, Bob(proxy signer), so that Bob can sign messages on behalf of Alice. Proxy signature can be validate for its correctness and can be distinguished between a normal signature and a proxy signature. So the verifier can be convinced of the original signer‘s agreement on the signed message. Proxy signature is used in a number of applications, including electronic commerce, mobile agents, distributed shared object systems,and many more. For example, the president of a company delegates a signing right to his/her secretary before a vacation. The secretary can make a signature on behalf of the president, and a verifier can be confident that the signature has been made by the authorized secretary. The verifier can also be convinced of the president‘s agreement on the signed message. Typically, a proxy signature scheme is as follows. The original signer Alice sends the proxy signer Bob a signature that is associated with a specific message. Bob makes a proxy private key using this information. Bob can then sign on a message with the proxy private key using a normal signature scheme. After the message and signature have been sent to the verifier, he/she recovers a proxy public key using public information and verifies the proxy signature using a normal signature scheme.

Proxy Signature scheme is introduced by Mambo [9]. Proxy signature scheme is based on a discrete logarithm problem. The original signer has the private key x and public key $y \equiv g^x (mod\ p)$. Proxy signature scheme is as follow:

System Parameters:-

The original signer choose k randomly and computes $r = g^k \bmod p$, and $s = x + kr \bmod p$. Now original signer send these system parameters to the proxy signer.

i.e. original signer sends (r, s) to the proxy signer. The proxy signer checks the validity of (r, s) as follows:

$$g^s = yr^r \bmod p$$

If this equality holds, the proxy signer accepts (r, s)as the valid proxy secret key.

Signing

The proxy signer signs a message m, then its signature $S_p$is generated. After that, the proxy signer sends the message and its signature, which are (m, S_p , r), to the verifier.

Verification

Upon receiving (m, S_p, r), the verifier recovers y' by $y' = yr^r \bmod p$ and substitute y' for y. After that, the verifier proceeds the verification phase of normal signature scheme.

10.5 SHORT SIGNATURE SCHEMEII

Short signature scheme[10] give the shortest signature among all discussed signature schemes. This signature scheme use elliptic curve and bilinear pairing. We will discuss this signature scheme starting from the basic signature scheme and then type of bilinear pairing it uses, after that security multiplier and finally types of elliptic curve used in this scheme.

Short signature scheme is in three parts, KeyGen, Sign, and Verify. It makes use of a hash function $h : \{0, 1\}^* \rightarrow G^*$. Where G is the base group and g is generator. G, g are system parameters.

1. **Key Generation:-** Choose a random $x \in Z^*_p$, and compute $v \leftarrow g^x$. x is the secret key and v is thepublic key.

2. Signing:- For a message $M \in \{0, 1\}^*$, and secret key x, Compute $h \leftarrow h(M)$, and $\sigma \leftarrow h^x$. The signatureis $\sigma \in G^*$.

3. **Verification:-** For a given public key v, a message M , and a signature , compute $h \leftarrow h(M)$ and verifythat (g, v, h, σ) is a valid Diffie-Hellmantuple.

So short signature scheme use bilinear pairing in verification of the signature.

Bilinear pairing:-

Let $G_1$and $G_T$be two cyclic groups of prime order q. Let G_2be a group and each element of G_2 has order dividing q. A bilinear pairing e is $e : G_1 \times G_2 \rightarrow G_T$such that

1. $e(g_1 , g_2) = 1_{GT}$for all $g_2 \in G_2$if and only if $g_1 = 1_{G1}$, and similarly $e(g_1 , g_2) = 1_{GT}$for all $g_1 \in G_1$if and only if $g_2 = 1_{G2}$.
2. for all $g_1 \in G1$ and $g_2 \in G_2$, $e(g_1 , g_2) = e(g_1^a , g_2^b)^{ab}$for all $a, b \in Z$.

Security Multiplier: - Let a finite field $F_p^l$where p is a prime and l is a positive integer, and an elliptic curve E over F_p^lhave m points. Let, point P of elliptic curve has order q, where $q^2 ! | m$. Then subgroup P has a security multiplier $\alpha > 0$, if order of p^lin $F^*_q$is α. We will discuss different families of elliptic curve Which are classified by the value of security multiplier.

Type 1

Let p be a prime where $p = 2(mod\ 3)$. Let E be the elliptic curve defined over F_p, and equation of the curve is $y^2 = x^3 + b$, Typically $b = \pm 1$. Then $E(F_p)$ is supersingular curve, and number of points, $\#E(F_p) = p + 1$, and $\#E(F_p^2) = (p + 1)^2$. For any odd $j \mid p + 1$, $G = E(F_p)[j]$ is cyclic and has security multiplier $\alpha = 2$. Let ι be the cube root of unity. Consider the following map, sometimes referred to as a distortion map:

$\Phi(x, y) = (\iota x, y)$

Then Φ maps points of $E(F_p)$ to points of $E(F_{p2})\backslash E(F_p)$. Thus if f denotes the bilinear pairing, then defining $e : G \times G \rightarrow F_{q2}$ by $e(P, Q) = f(P, \Phi(Q))$ gives a bilinear non-degenerate map.

Type 2

Unlike above discussed curve this type of curve have low characteristic field. Let F is a finite field defined over 3^l where l is a positive exponent. Let curve $E^+ : y^2 = x^3 + 2x + 1$, and $E : y^2 = x^3 + 2x - 1$, over F_3^l.

when $l = \pm 1 mod 12$

$\#E^+(F_3^l) = 3^l + 1 + 3^{(l+1)/2}$

when $l = \pm 5 mod 12$

$\#E^+(F_3^l) = 3^l + 1 - 3^{(l+1)/2}$

when $l = \pm 1 mod 12$

$\#E^-(F_3^l) = 3^l + 1 - 3^{(l+1)/2}$

when $l = \pm 5 mod 12$

$\#E^-(F_3^l) = 3^l + 1 + 3^{(l+1)/2}$

Type 3

Let p be a prime where $p \equiv 3(mod\ 4)$. Let E be the elliptic curve defined over F_p, and equation of the curve is $y^2 = x^3 + ax$, where $a \in Z(mod\ p)$. Then $E(F_q)$ is supersingular curve, and number of point, $\#E(F_p) = p + 1$, and $\#E(F_{p2}) = (p + 1)^2$. For any odd $j|p + 1$, Group $G = E(F_p)[j]$ is cyclic and has security multiplier $\alpha = 2$.

Type 4

Type 4 curves are non-supersingular. By considering cyclotomic polynomials, elliptic curve with security multiplier 12 can be generated. Let $q(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$. Let $t(x) = 6x^2 + 1$. If $D = 3$, then solution of CM equation will always be V = $6x^2 + 4x + 1$. It turns out $q(x) + 1 -$

$t(x) \mid q(x)12 - 1$. So the value of security multiplier is 12. Following algorithm is used to generate curves:

1. Pick an integer x of a desired magnitude. It may benegative.
2. Check if $q(x)$ isprime.
3. Check if $n = q(x) - t(x) + 1$ has a large prime factor r. (Ideally it should beprime.)
4. Try different values of k until a random point of $y^2 = x^3 + k$ has order n.

Type 5

Type 5 curve are also non-supersingular curve. Type 6 curve are ordinary curves with security multiplier 6. Order of type 6 curves is a prime or a prime multiplied by a small constant. Let a finite field F defined over some p where $p \approx q$. Where s is a small constant and q is a prime. When type 5 curve is defined over field F_{p6}, its order is a multiple of q^2.

References for Last 4 Lectures:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and CliffordStein –Introduction to Algorithms‖. Third Edition.
2. WadeTrappe,LawrenceC. Washington –Introduction to Cryptographywith Coding Theory‖ SecondEdition.
3. William Stallings —Cryptography and Network Security‖. FourthEdition.
4. R.L. Rivest, A.Shamir,andL.Adleman –A Method for ObtainingDigital Signatures

Unit 4: Stream Ciphers 4

11.1 VIDEO DATA CIPHERS

Ciphers:

1. **Block Cipher**
2. **StreamCipher**

Block Cipher: The same function is used to encrypt successive blocks (memory less).

Stream Cipher: This processes plan text as small as single bit. It has memory.

One – Time – Pad (corresponding cipher is called ***Vernam cipher***)

$$c_i = m_i \oplus k_i$$

m_i: plain text

k_i: keystream

c_i: ciphertext

Decryption :

$$\begin{aligned} m_i &= c_i \oplus k_i \\ &= m_i \oplus k_i \oplus k_i \\ &= m_i \end{aligned}$$

Assumption: is truly random.

Synchronous Stream Ciphers:

{There is a clock which is same at both the ends}

Definition: a synchronous stream cipher is one in which the key stream is generated independently of the plain text and cipher text.

Properties of Synchronous stream cipher:

• Synchronization requirement: In a synchronous stream cipher, both the sender and receiver must be synchronized using the same key. If synchronization is lost due to cipher text digits being inserted or deleted during transmission, then decryption fails and can only be restored through additional techniques for re-synchronization. This involves either re-initialization or

placing special marker at regular intervals or redundancy in plaintext.

• No error propagation: A cipher text digit that is modified during transmission doesn'teffect decryption of other cipher textdigits.

Active attacks: As a consequence of properly (i), the insertion, deletion or replay of cipher text digits by an active adversary causes immediate loss of synchronization and hence might possibly be detected by decryptors.

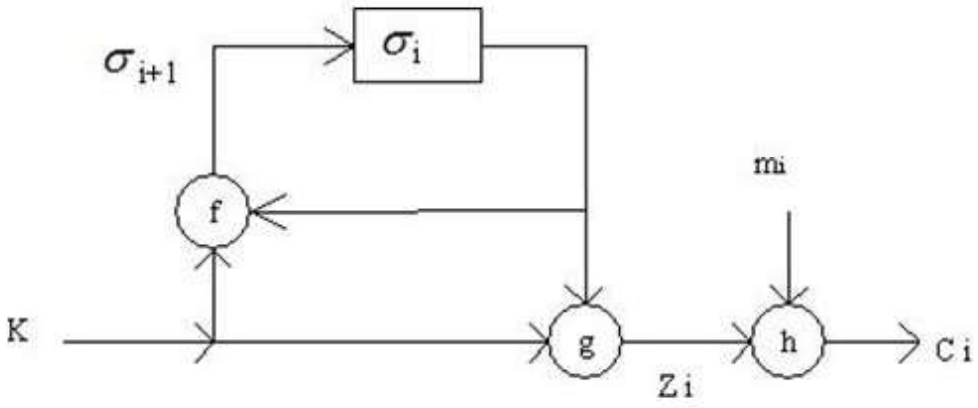
Application: Stream ciphers are used for video data stream.

Reference:

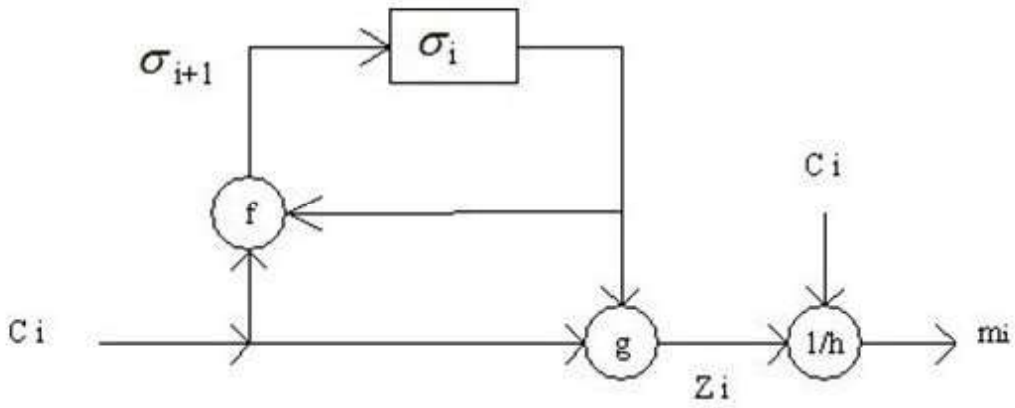
1. *Handbook of Applied Cryptography* , A. Menzes, P. van Oorschot and S. Vanstone.

Scribes: Rakesh Yarlagadda, Ravi Ranjan

Encryption:



Decryption:



Properties of Synchronous stream cipher:

• Synchronization requirement: In a synchronous stream cipher, both the sender and receiver must be synchronized using the same key. If synchronization is lost due to cipher text digits being inserted or deleted during transmission, then decryption fails and can only be restored through additional techniques for re-synchronization. This involves either re-initialization or placing special marker at regular intervals or redundancy in plaintext.

• No error propagation: A cipher text digit that is modified during transmission doesn't effect decryption of other cipher text digits.

Active attacks: As a consequence of properly (i), the insertion, deletion or replay of cipher text digits by an active adversary causes immediate loss of synchronization and hence might possibly be detected by decryptors.

Application: Stream ciphers are used for video data stream.

Reference:

1. *Handbook of Applied Cryptography* , A. Menzes, P. van Oorschot and S. Vanstone.

Scribes: Rakesh Yarlagadda, Ravi Ranjan

Block-4

Unit 1: Entity Authentication 1

12.1 ElGamal Signature Scheme

The ElGamal signature scheme is a digital signature scheme based on the algebraic properties of modular exponentiation, together with the discrete logarithm problem. The algorithm uses a key pair consisting of a public key and a private key. The private key is used to generate a digital signature for a message, and such a signature can be verified by using the signer's corresponding public key. The digital signature provides message authentication (the receiver can verify the origin of the message), integrity (the receiver can verify that the message has not been modified since it was signed) and non-repudiation (the sender cannot falsely claim that they have not signed the message).

Operations

The scheme involves four operations: key generation (which creates the key pair), key distribution, signing and signature verification.

Key generation

Key generation has two phases. The first phase is a choice of algorithm parameters which may be shared between different users of the system, while the second phase computes a single key pair for one user.

Parameter generation

- Choose a key length N. .
- Choose a N-bit prime number
- Choose a cryptographic hash function with output length L bits. If L > N, only the leftmost N bits of the hash output are used.
- Choose a generator g < p of the multiplicative group of integers modulo p, $Z^*_{p.}$

The algorithm parameters are. These parameters may be shared between users of the system.

Per-user keys

Given a set of parameters, the second phase computes the key pair for a single user:

- Choose an integer x randomly from {1…p – 2}
- Compute y: = g^x mod p.

x is the private key and y is the public key.

Key distribution

The signer should send the public key y to the receiver via a reliable, but not necessarily secret, mechanism. The signer should keep the private key x secret.

Signing

A message m is signed as follows:

- Choose an integer k randomly from {2… p – 2 }with k relatively prime to p - 1.
- Compute r := g^k mod p.
- Compute s := (H(m)-x r)k^{-1} mod (p-1).
- In the unlikely event that s=0 start again with a different random k.

The signature is (r,s)

Verifying a signature

One can verify that a signature is (r, s) a valid signature for a message m as follows:

- Verify that 0 < r < pand0<s<p-1.
- The signature is valid if and only if $g^{H(m)} \equiv y^{\wedge r} r^{\wedge s} \pmod p$

Correctness

The algorithm is correct in the sense that a signature generated with the signing algorithm will always be accepted by the verifier.

The computation of s during signature generation implies

$$H(m) \equiv x\, r + s\, k \bmod \{p-1\}$$

Since g is relatively prime to p,

$$g^{H(m)} \equiv g^{\{xr+sk\}} \pmod p$$
$$\equiv (g^{x)r}\ (g\wedge\{k\})\wedge s \pmod p$$
$$\equiv (y)^r\ (r)^s \pmod p$$

Security

A third party can forge signatures either by finding the signer's secret key *x* or by finding collisions in the hash function H(m) ≡H(M)(mod p-1). Both problems are believed to be difficult. However, as of 2011 no tight reduction to a computational hardness assumption is known.

The signer must be careful to choose a different *k* uniformly at random for each signature and to be certain that *k*, or even partial information about *k*, is not leaked. Otherwise, an attacker may be able to deduce the secret key *x* with reduced difficulty, perhaps enough to allow a practical attack. In particular, if two messages are sent using the same value of *k* and the same key, then an attacker can compute *x* directly.

Existential forgery

The original paper did not include a hash function as a system parameter. The message *m* was used directly in the algorithm instead of *H(m)*. This enables an attack called existential forgery, as described in section IV of the paper. Pointcheval and Stern generalized that case and described two levels of forgeries:

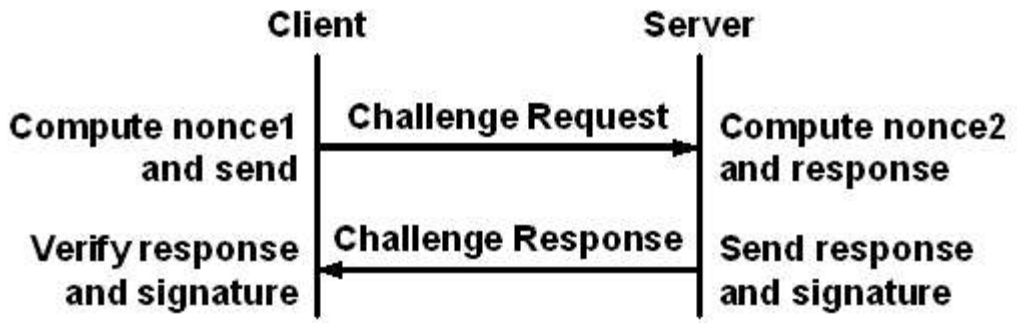
1. **The one-parameter forgery.** Select an e such that $1 < e < p - 1$. Setr := g^e y mod pand s := -r mod(p-1). Then the tuple(r, s) is a valid signature for the message m = es mod(p-1).
2. **The two-parameter forgery.** Select 1 < e,v < p-1, and gcd (v,p-1)=1. Set r := $g^e y^v$ mod p and s := -rv^{-1} mod{(p-1). Then the tuple (r, s)is a valid signature for the messagem = es mod{(p-1). The one-parameter forgery is a special case of the two-parameter forgery, when v = 1.

12.2 Autokey Identity Scheme

While the identity scheme described in RFC-2875 is based on a ubiquitous Diffie-Hellman infrastructure, it is expensive to generate and use when compared to others described here. There are five schemes now implemented in Autokey to prove identity: (1) trusted certificates (TC), (2) private certificates (PC), (3) a modified Schnorr algorithm (IFF aka Identify Friendly or Foe), (4) a modified Guillou-Quisquater algorithm (GQ), and (5) a modified Mu-Varadharajan algorithm (MV). The TC scheme, which involves a certificate trail to a trusted host, is discussed on the Autokey Protocol page. Each of the others involves generating parameters specific to the scheme, together with public and private values used by the scheme.

In order to simplify implementation, each scheme uses existing structures in the OpenSSL library, including those for RSA and DSA cryptography. As these structures are sometimes used in ways far

different than their original purpose, they are called cuckoo structures in the descriptions that follow.



In the challenge-response schemes client Alice asks server Bob to prove identity relative to a secret group key b provided by a trusted authority (TA). As shown in the figure above, client Alice rolls random nonce1 and sends to server Bob. Bob rolls random nonce2, performs some mathematical function and returns the response along with the hash of some private value to Alice. Alice performs another mathematical function and verifies the result matches the hash in Bob's message.

Each of the five schemes is intended for specific use. There are three kinds of keys, trusted agent, server and client. Servers can be clients of other servers, but clients cannot cannot be servers for dependent clients. In general, the goals of the schemes are that clients cannot masquerade as a servers and a servers cannot masquerade as a trusted agents (TAs), but they differ somewhat on how to achieve these goals. To the extent that identity can be verified without revealing the group key, the schemes are properly described as zero-knowledge proofs.

The four identity schemes described here have different design goals and are intended for specific application. The PC scheme is intended for one-way broadcast configurations where clients cannot run a duplex protocol. It is essentially a symmetric key cryptosystem where the certificate itself is the key.

The IFF scheme is intended for servers operated by national laboratories. The servers share a private group key and provide the public client parameters on request. The clients cannot masquerade as servers.

The GQ scheme is intended for exceptionally hostile scenarios where it is necessary to change the client key at relatively frequent intervals. As in the IFF scheme the servers share a private group key and provide the public client parameters on request. In this scheme clients requre a public key to complete the exchange. This is conveyed in the server certificate in an extension field. The certificate

can be changed while retaining the same group key.

The MV scheme is intended for the most challenging scenarios where it is neccesary to protect against both server and client masquerade. The private values used by the TA to generate the cryptosystem are not available to the servers and the private values used by the servers to encrypt data are not available to the clients. Thus, a client cannot masquerade as a server and a server cannot masquerade as a TA. However, a client can verify a server has the correct group key even though neither the client nor server know the group key, nor can either manufacture a client key acceptable to any other client. A further feature of this scheme is that the TA can collaborate with the servers to revoke client keys.

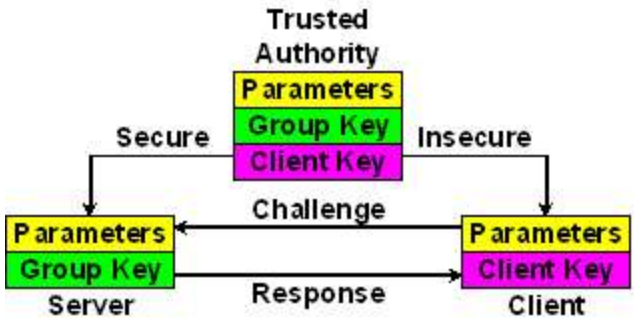
Private Certificate (PC) Cryptosystem

The PC scheme shown above uses a private certificate as the group key. A certificate is designated private by a X509 Version 3 extension field when generated by the ntp-keygen program in the NTP software distribution. In the Autokey context it functions as a symmetric key. The private certificate is generated by a TA and distributed to all group members by secure means and is never revealed outside the group. A client is marked trusted in the (optional) Parameter Exchange and authentic when the first signature is verified. This scheme is cryptographically strong as long as the private certificate is protected; however, it can be very awkward to refresh the keys or certificate, since new values must be securely distributed to a possibly large population and activated simultaneously

Schnorr *(IFF) Cryptosystem*

The Schnorr (IFF) identity scheme can be used when certificates are generated by utilities other than the ntp-keygen program in the NTP software distribution. Certificates can be generated by the OpenSSL library or an external public certificate authority, but conveying an arbitrary public value in a certificate extension field might not be possible. The TA generates the IFF parameters, private key and public key, then sends these values to the servers and the parameters and public key to the clients. Without the private key a client cannot masquerade as a legitimate server.

The DSA parameters are generated by routines in the OpenSSL library. The IFF values hide in a DSA cuckoo structure which uses the same parameters. The p is a 512-bit prime, g a generator of the multiplicative group Zp* and q a 160-bit prime that divides p - 1 and is a qth root of 1 mod p; that is, gq = 1 mod p. The TA rolls a private random group key b (0 < b < q) and computes the public key v = gb, then distributes private (p, q, g, b) to the servers using secure means and public (p, q, g, v) to the clients not necessarily using secure means.



The TA generates a DSA parameter structure for use as IFF parameters. The IFF parameters are identical to the DSA parameters, so the OpenSSL library DSA parameter generation routine can be used directly. The DSA parameter structure is written to a file as an encrypted DSA key encoded in PEM. Unused structure members are set to one.

IFF	DSA	Item	Include
p	p	modulus	all
q	q	modulus	all
g	g	generator	all
b	priv_key	group key	server
v	pub_key	client key	client

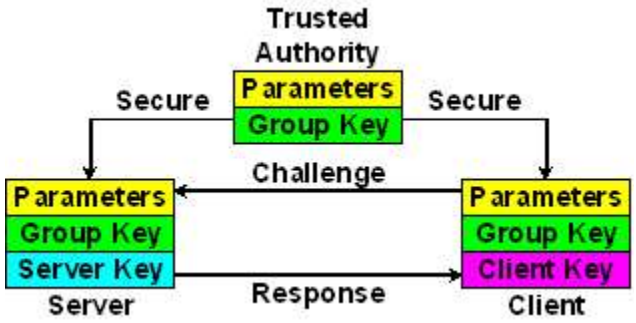
Alice challenges Bob to confirm identity using the following protocol exchange.

1. Alice rolls random r (0 < r < q) and sends to Bob.
2. Bob rolls random k (0 < k < q), computes y = k + b r mod q and x = gk mod p, then sends (y, hash(x)) to Alice.
3. Alice computes z = gy vr mod p and verifies hash(z) equals hash(x).

Guillou-*Quisquater (GQ) Cryptosystem*

The Guillou-Quisquater (GQ) identity scheme is useful when certificates are generated by the ntp-keygen program in the NTP software distribution. The TA generates the GQ parameters, private key and public key, then sends these values to the servers and the parameters to the clients. The public key is inserted in an X.509 extension field when the certificate is generated. Without the private key a client cannot masquerade as a legitimate server.

The RSA parameters are generated by routines in the OpenSSL libbrary. The GQ values hide in a RSA cuckoo structure which uses the same parameters. The values are used in an identity scheme based on RSA cryptography and described in [1] and [5] p. 300 (with errors). The 512-bit public modulus n = p q, where p and q are secret large primes. The TA rolls random group key b (0 < b < n) and sends (n, b) to the servers using secure means. The private key and public key are constructed later.



The TA generates a RSA parameter structure for use as GQ parameters. The RSA parameter structure is written to a file as an encrypted RSA key encoded in PEM. Unused structure members are set to one.

When generating new certificates, the server rolls new random private key u (0 < u < n) and public key its inverse u-1 obscured by the group key v = u-1 b. These values replace the private and public keys normally generated by the RSA scheme. In addition, the public key v is conveyed in a X.509 certificate extension.

GQ	RSA	Item	Include
n	n	modulus	all
b	e	group key	all
u	p	server key	server
v	q	client key	client

Alice challenges Bob to confirm identity using the following exchange**.**

1 Alice rolls random r (0 < r < n) and sends to Bob.

2 Bob rolls random k (1 < k < n) and computes y = k ur mod n and x = kb mod n, then sends (y, hash(x)) to Alice.

3 Alice computes z = vr yb mod n and verifies hash(z) equals hash(x).

Mu-Varadharajan (MV) Cryptosystem

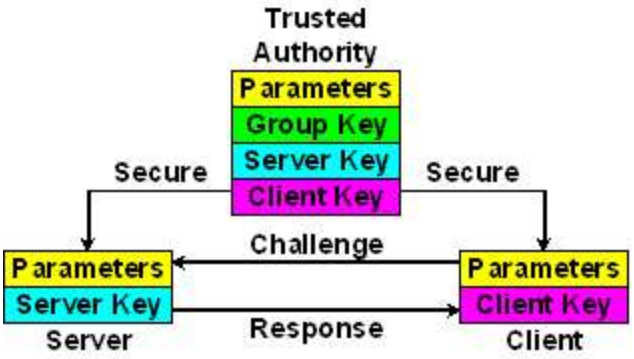
The Mu-Varadharajan (MV) scheme was originally intended to encrypt broadcast transmissiions to receivers which do not transmit. There is one encryption key for the broadcaster and a separate decryption key for each receiver. It operates something like a pay-per-view satellite broadcasting system where the session key is encrypted by the broadcaster and the decryption keys are held in a tamperproof set-top box. We don't use it this way, but read on.

In the MV scheme the TA constructs an intricate cryptosystem involving a number of activation keys known only to the TA. The TA decides which keys to activate and provides to the servers an encryption key E and server decryption keys gbar and ghat which depend on the activated keys. The servers have no additional information and, in particular, cannot masquerade as a TA. In addition, the TA provides for each activation key j individual client decryption keys xbar and xhat, which do not need to be changed if the TA enables or disables an activation key. The clients have no further information and, in particular, cannot masquerade as a server or TA.

Clients are assigned one of the activation keys and are provided with the corresponding client key. There can be any number of clients sharing the same activation key according to policy. While the machinery to enable and disable ativation keys is included in the current implementation, specific means and interfaces to do this are not yet available, so only one client key is provided.

The scheme is designed so that clients can construct the inverse of E from the server gbar and ghat and client xbar and xhat. In the scheme both E and its inverse are exponentiated by a server nonce, so the product is always one and the secrecy depends on the descrete log problem.

The MV values hide in a DSA cuckoo structure which uses the same parameters, but generated in a different way. The values are used in an encryption scheme similar to El Gamal cryptography and use a polynomial formed from the expansion of product terms (x - xj), as described in [3]. The paper has significant errors and serious omissions.



The TA generates the modulus, encryption key and server decryption keys as an encrypted DSA key encoded in PEM. Unused structure members are set to one.

MV	DSA	Item	Include
p	p	modulus	all
q	q	modulus	server
E	g	private encrypt key	server
gbar	priv_key	server decrypt key	server
ghat	pub_key	server decrypt key	server

The TA generates the modulus and client decryption keys as a nonencrypted DSA key encoded in PEM. It is used only by designated recipient(s) who pay a suitably outrageous fee for its use. Unused structure members are set to one.

MV	DSA	Item	Include
p	p	modulus	all
xbar	priv_key	client decrypt key	client
xhat	pub_key	client decrypt key	client

The devil is in the details. Let q be the product of n distinct primes s1j (j = 1...n), where each s1j, also called an activation key, has m significant bits. Let prime p = 2q + 1, so that q and each s1j divide p - 1 and p has M = nm + 1 significant bits. Let g be a generator of the multiplicative group Zp*; that is, gcd(g, p - 1) = 1 and gq = 1 mod p. We do modular arithmetic over Zq and then project into Zp* as powers of g. Sometimes we have to compute an inverse b-1 of random b in Zq, but for that purpose we require gcd(b, q) = 1. We expect M to be in the 500-bit range and n relatively small, like 30. The TA uses a nasty probabilistic algorithm to generate the cryptosystem. In the following let the number ofbits

in the modulus m = 512.

1 The object is to generate a multiplicative group Zp* modulo a prime p and a subset Zq mod q, where q is the product of n distinct m-bit primes s1j (j = 1...n) and q divides p - 1. As a practical matter, it is tough to find more than 31 distinct primes for mn = 512 or 61 primes for mn = 1024. The latter can take several hundred iterations and several minutes on a Sun Blade 1000.

2 Compute the modulus q as the product of the primes. Compute the modulus p as 2q + 1 and test p for primality. If p is composite, replace one of the primes with a new distinct prime and try again. Note that q will hardly be a secret since we have to reveal p to servers and clients. However, factoring q to find the primes should be adequately hard, as this is the same problem considered hard in RSA. Question: is it as hard to find n small prime factors totalling n bits as it is to find two large prime factors totalling n bits? Remember, the bad guy doesn't know n.

3 Associate with each s1j an element sj such that sj s1j = s1j mod q. One way to find an sj is to compute the quotient (q + s1j) / s1j mod p. The student should prove the remainder is always zero.

4 Compute the generator g of Zp using a random roll such that gcd(g, p - 1) = 1 and gq = 1. If not, roll again.

The cryptosystem parameters n, p, q, g, s1j, sj (j = 1...n) have been determined. The TA sets up a specific instance of the scheme as follows.

Roll random roots xj mod q (j = 1...n) for a polynomial of order n. While it may not be strictly necessary, Make sure each root has no factors in common with q.

Generate polynomial coefficients ai (i = 0...n) from the expansion of root products (x - xi) mod q in powers of x using a fast method contributed by Charlie Boncelet.

Generate gi = gai mod p for all i and the generator g. Verify prod(giai xji) = 1 for all i, j. Note the ai xji exponent is computed mod q, but the gi is computed mod p. Also note the expression given in the paper cited is incorrect.

1 Make master encryption key A = Prod(gixj) (i = 0...n, j = 1...n - 1). Keep it around for awhile, since it is expensive to compute.

2 Roll private random group key b mod q (0 < b < q), where gcd(b, q) = 1 to guarantee the inverse exists, then compute b-1 mod q. If b is changed, all keys must be recomputed.

3 Make private client keys xbarj = b-1 Sum(xin mod q) (i = 1...n, i != j) and xhatj = sjxjn for all j. Note that the keys for the jth client involve only sj and that s1j remain secret. The TA sends (p, xbarj, xhatj) to the jth client(s) using nonsecure means.

4 The activation key is initially q by construction. The TA revokes client j by dividing q by s1j. The quotient becomes the activation key s. Note we always have to revoke one key; otherwise, the plaintext and cryptotext would be identical. The TA computes private server encryption key E = As and server decryption keys gbar = gbars and ghat = ghatsb mod p and sends (p, E, gbar, ghat) to the servers using secure means. These values must be recomputed if an activation key is changed.

Alice challenges Bob to confirm identity using the following exchange.

1 Alice rolls random r (0 < r < q) and sends to Bob.

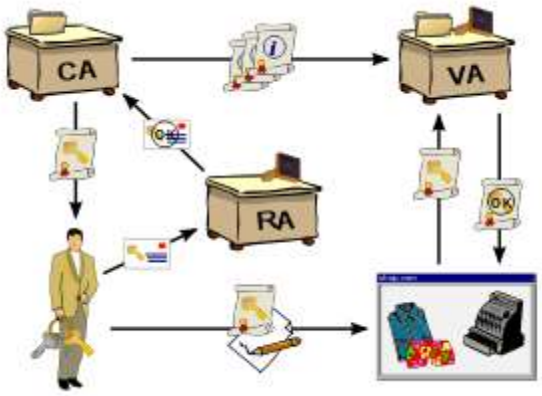
2 Bob rolls random k (0 < k < q), computes y = rEk, ybar = gbark and yhat = ghatk, then returns (y, ybar, yhat) to Alice.

3 Alice computes the session decryption key (Ek)-1 = ybarxhatj yhatxbarj, then verifies that y = r.

Unit 2: Public Key Infrastructure 2

12.2 Public key Infrastructure

A public key infrastructure (PKI) is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption. The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as e-commerce, internet banking and confidential email. It is required for activities where simple passwords are an inadequate authentication method and more rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred.



In cryptography, a PKI is an arrangement that *binds* public keys with respective identities of entities (like people and organizations). The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA). Depending on the assurance level of the binding, this may be carried out by an automated process or under human supervision.

The PKI role that may be delegated by a CA to assure valid and correct registration is called a *registration authority* (RA). Basically, an RA is responsible for accepting requests for digital certificates and authenticating the entity making the request. The Internet Engineering Task Force's RFC 3647 defines an RA as "An entity that is responsible for one or more of the following functions: the identification and authentication of certificate applicants, the approval or rejection of certificate applications, initiating certificate revocations or suspensions under certain circumstances, processing

subscriber requests to revoke or suspend their certificates, and approving or rejecting requests by subscribers to renew or re-key their certificates. RAs, however, do not sign or issue certificates (i.e., an RA is delegated certain tasks on behalf of a CA). While Microsoft may have referred to a subordinate CA as an RA, this is incorrect according to the X.509 PKI standards. RAs do not have the signing authority of a CA and only manage the vetting and provisioning of certificates. So in the Microsoft PKI case, the RA functionality is provided either by the Microsoft Certificate Services web site or through Active Directory Certificate Services which enforces Microsoft Enterprise CA and certificate policy through certificate templates and manages certificate enrollment (manual or auto-enrollment). In the case of Microsoft Standalone CAs, the function of RA does not exist since all of the procedures controlling the CA are based on the administration and access procedure associate with the system hosting the CA and the CA itself rather than Active Directory. Most non-Microsoft commercial PKI solutions offer a stand-alone RA component.

An entity must be uniquely identifiable within each CA domain on the basis of information about that entity. A third-party validation authority (VA) can provide this entity information on behalf of the CA.

The X.509 standard defines the most commonly used format for public key certificates.

Design

Public key cryptography is a cryptographic technique that enables entities to securely communicate on an insecure public network, and reliably verify the identity of an entity via digital signatures.

A public key infrastructure (PKI) is a system for the creation, storage, and distribution of digital certificates which are used to verify that a particular public key belongs to a certain entity. The PKI creates digital certificates which map public keys to entities, securely stores these certificates in a central repository and revokes them if needed.

A PKI consists of:

- A *certificate authority* (CA) that stores, issues and signs the digital certificates;
- A *registration authority* (RA) which verifies the identity of entities requesting their digital certificates to be stored at the CA;
- A *central directory*—i.e., a secure location in which keys are stored and indexed;
- A *certificate management system* managing things like the access to stored certificates or the delivery of the certificates to be issued;

- A *certificate policy* stating the PKI's requirements concerning its procedures. Its purpose is to allow outsiders to analyze the PKI's trustworthiness.

Method of Clarification

Broadly speaking, there have traditionally been three approaches to getting this trust: certificate authorities (CAs), web of trust (WoT), and simple public key infrastructure (SPKI).

Certificate authorities

The primary role of the CA is to digitally sign and publish the public key bound to a given user. This is done using the CA's own private key, so that trust in the user key relies on one's trust in the validity of the CA's key. When the CA is a third party separate from the user and the system, then it is called the Registration Authority (RA), which may or may not be separate from the CA. The key-to-user binding is established, depending on the level of assurance the binding has, by software or under human supervision.

The term trusted third party (TTP) may also be used for certificate authority (CA). Moreover, PKI is itself often used as a synonym for a CA implementation.

Issuer market share

- In this model of trust relationships, a CA is a trusted third party – trusted both by the subject (owner) of the certificate and by the party relying upon the certificate.
- According to NetCraft report from 2015, the industry standard for monitoring active Transport Layer Security (TLS) certificates, states that "Although the global [TLS] ecosystem is competitive, it is dominated by a handful of major CAs — three certificate authorities (Symantec, Sectigo, GoDaddy) account for three-quarters of all issued [TLS] certificates on public-facing web servers. The top spot has been held by Symantec (or VeriSign before it was purchased by Symantec) ever since [our] survey began, with it currently accounting for just under a third of all certificates. To illustrate the effect of differing methodologies, amongst the million busiest sites Symantec issued 44% of the valid, trusted certificates in use — significantly more than its overall market share."
- Following to major issues in how certificate issuing were managed, all major players gradually distrusted Symantec issued certificates starting from 2017.

Temporary certificates and single sign-on

This approach involves a server that acts as an offline certificate authority within a single sign-on system. A single sign-on server will issue digital certificates into the client system, but never stores them. Users can execute programs, etc. with the temporary certificate. It is common to find this solution variety with X.509-based certificates.

Starting Sep 2020, TLS Certificate Validity reduced to 13 Months.

Web of Trust

An alternative approach to the problem of public authentication of public key information is the web-of-trust scheme, which uses self-signed certificates and third-party attestations of those certificates. The singular term "web of trust" does not imply the existence of a single web of trust, or common point of trust, but rather one of any number of potentially disjoint "webs of trust". Examples of implementations of this approach are PGP (Pretty Good Privacy) and GnuPG (an implementation of OpenPGP, the standardized specification of PGP). Because PGP and implementations allow the use of e-mail digital signatures for self-publication of public key information, it is relatively easy to implement one's own web of trust.

One of the benefits of the web of trust, such as in PGP, is that it can interoperate with a PKI CA fully trusted by all parties in a domain (such as an internal CA in a company) that is willing to guarantee certificates, as a trusted introducer. If the "web of trust" is completely trusted then, because of the nature of a web of trust, trusting one certificate is granting trust to all the certificates in that web. A PKI is only as valuable as the standards and practices that control the issuance of certificates and including PGP or a personally instituted web of trust could significantly degrade the trustworthiness of that enterprise's or domain's implementation of PKI.[18]

The web of trust concept was first put forth by PGP creator Phil Zimmermann in 1992 in the manual for PGP version 2.0:

As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.

Simple public key Infrastructure

Another alternative, which does not deal with public authentication of public key information, is the simple public key infrastructure (SPKI) that grew out of three independent efforts to overcome the complexities of X.509 and PGP's web of trust. SPKI does not associate users with persons, since the key is what is trusted, rather than the person. SPKI does not use any notion of trust, as the verifier is also the issuer. This is called an "authorization loop" in SPKI terminology, where authorization is integral to its design.[citation needed] This type of PKI is specially useful for making integrations of PKI that do not rely on third parties for certificate authorization, certificate information, etc.; A good example of this is an Air-gapped network in an office.

Decentralized PKI

Decentralized identifiers (DIDs) eliminate dependence on centralized registries for identifiers as well as centralized certificate authorities for key management, which is the standard in hierarchical PKI. In cases where the DID registry is a distributed ledger, each entity can serve as its own root authority. This architecture is referred to as decentralized PKI (DPKI).

BlockChain-based PKI

An emerging approach for PKI is to use the blockchain technology commonly associated with modern cryptocurrency.[21][22] Since blockchain technology aims to provide a distributed and unalterable ledger of information, it has qualities considered highly suitable for the storage and management of public keys. Some cryptocurrencies support the storage of different public key types (SSH, GPG, RFC 2230, etc.) and provides open source software that directly supports PKI for OpenSSH servers.[citation needed] While blockchain technology can approximate the proof of work often underpinning the confidence in trust that relying parties have in a PKI, issues remain such as administrative conformance to policy, operational security and software implementation quality. A certificate authority paradigm has these issues regardless of the underlying cryptographic methods and algorithms employed, and PKI that seeks to endow certificates with trustworthy properties must also address these issues.

Here is a list of known blockchain-based PKI:

- CertCoin
- FlyClient
- BlockQuick

Uses

PKIs of one type or another, and from any of several vendors, have many uses, including providing public keys and bindings to user identities which are used for:

- Encryption and/or sender authentication of e-mail messages (e.g., using OpenPGP or S/MIME);
- Encryption and/or authentication of documents (e.g., the XML Signature or XML Encryption standards if documents are encoded as XML);
- Authentication of users to applications (e.g., smart card logon, client authentication with SSL). There's experimental usage for digitally signed HTTP authentication in the Enigform and mod_openpgp projects;
- Bootstrapping secure communication protocols, such as Internet key exchange (IKE) and SSL. In both of these, initial set-up of a secure channel (a "security association") uses asymmetric key—i.e., public key—methods, whereas actual communication uses faster symmetric key—i.e., secret key—methods;
- Mobile signatures are electronic signatures that are created using a mobile device and rely on signature or certification services in a location independent telecommunication environment;
- Internet of things requires secure communication between mutually trusted devices. A public key infrastructure enables devices to obtain and renew X509 certificates which are used to establish trust between devices and encrypt communications using TLS.

Unit 3: Classical Cryptography 3

13.1 Introduction

In this unit, we will learn to describe and analyze the following classical ciphers: ADFGVX, Affine, Beaufort, Bifid, Caesar, Columnar Transposition, Four-Square, Hill, Playfair, Polybius Square, Rail-fence, Simple Substitution, Straddle Checkerboard, Vigenere, Autokey, Enigma, and Lorenz ciphers.nz ciphers. These ciphers are intuitively easy to understand and seem to encrypt the message well, but they have many shortcomings, which we will discuss as we work through this unit. By studying these classical ciphers, you will learn to avoid poor cipher design.

13.2 Learning Objectives

Upon successful completion of this unit, students will be able to:

- Define, use, and effectively attack classical ciphers such as the ADFGVX, Affine, Beaufort, Bifid, Caesar, Columnar Transposition, Four-Square, Hill, Playfair, Polybius Square, Rail-fence, Simple Substitution, Straddle Checkerboard, Vigenere, and Autokey ciphers.
- Explain the workings of mechanical ciphers Enigma and Lorenz.

13.3 ADFGVX Cipher

In cryptography, the ADFGVX cipher was a field cipher used by the German Army during World War I. ADFGVX was in fact an extension of an earlier cipher called the ADFGX cipher. Invented by Colonel Fritz Nebel and introduced in March 1918, the cipher was a fractionating transposition cipher which combined a modified Polybius square with a single columnar transposition. The cipher is named after the six possible letters used in the ciphertext: A, D, F, G, V and X. These letters were chosen deliberately because they sound very different from each other when transmitted via morse code. The intention was to reduce the possibility of operator error.

From Kahn's 'The CodeBreakers':

"It was no less clear to the Allies that Germany planned to launch a climactic offensive in the spring. There were many signs—the new cipher itself was one. The question was: Where and when would

the actual blow fall? The German high command, recognizing the incalculable military value of surprise, shrouded its plans in the tightest secrecy. Artillery was brought up in concealment; feints were flung out here and there along the entire front to keep the Allies off balance; the ADFGVX cipher, which had reportedly been chosen from among many candidates by a conference of German ciphe specialists, constituted an element in this overall security, as did the new Schliis-selheft. The Allies bent every effort and tapped every source of information to find out the time and place of the real assault."

Georges Painvin was the French cryptanalyst tasked with cracking the ADFGVX cipher. The intelligence he provided was vital to the French war effort, particularly in saving Paris in 1918:

"At midnight on June 9 the front from Montdidier to Compiegne erupted in a fierce, pelting hurricane of high-explosive, shrapnel, and gas shells. For three hours a German artillery concentration that averaged one gun for no more than ten yards of front poured a continual stream of fire onto the French positions—and Ludendorff's urgent demand for ammunition became clear. But this time, for the first time since Ludendorff began his stupendous series of triumphs, there was no surprise. Painvin's manna had saved the French."

The Algorithm

The 'key' for a ADFGVX cipher is a 'key square' and a key word. e.g.

```
p h 0 q g 6

4 m e a 1 y

l 2 n o f d

x k r 3 c v

s 5 z w 7 b

j 9 u t i 8
```

The key square is a 6 by 6 square containing all the letters and the numbers 0 - 9. The key word is any word e.g.

GERMAN

There are a number of steps involved:

1. Build a table like the following with the key square. This is known as a polybius square.

```
A D F G V X
A | p h 0 q g 6
```

```
D | 4 m e a l y
F | l 2 n o f d
G | x k r 3 c v
V | s 5 z w 7 b
X | j 9 u t i 8
```

2. Encode the plaintext using this matrix, to encode the letter 'a', locate it in the matrix and read off the letter on the far left side on the same row, followed by the letter at the top in the same column. In this way each plaintext letter is replaced by two cipher text letters. E.g. 'attack' -> 'DG XG XG DG GV GD'. The cipher text is now twice as long as the original plaintext. Note that so far, it is just a simple substitution cipher, and trivial to break.

3. Write the code word with the enciphered plaintext underneath e.g.

```
G E R M A N
D G X G X G
D G G V G D
```

4. Perform a columnar transposition. Sort the code word alphabetically, moving the columns as you go. Note that the letter pairs that make up each letter get split apart during this step, this is called fractionating.

```
A E G M N R
X G D G G X
G G D V D G
```

5. Read the final cipher text off in columns.

-> XG GG DD GV GD XG

A Short Example

We will now encipher:

```
DEFEND THE EAST WALL OF THE CASTLE
```

Using the same key as above, after the substitution step we get:

```
FXDFFVDFFFFXXGADDFDFDGVAXGVGDGFAFAFGFVXGADDFGVDGVAXGFADF
```

We now write this out with the keyword above (table on the left), and sort the columns (table on the right):

```
G E R M A N
F X D F F V
D F F F F X
X G A D D F
D F D G V A
```

```
A E G M N R
F X F F V D
F F D F X F
D G X D F A
V F D G A D
```

X G V G D G	D G X G G V
F A F A F G	F A F A G F
F V X G A D	A V F G D X
D F G V D G	D F D V G G
V A X G F A	F A V G A X
D F	F D

We now read off the columns to get the final cipher:

```
FFDVDFADFXFGFGAVFAFFDXDXFFDVDFFDGGAGVGVXFAGGDGADFADVFXGX
```

13.4 Affine Cipher

Introduction

The Affine cipher is a special case of the more general monoalphabetic substitution cipher.The cipher is *less* secure than a substitution cipher as it is vulnerable to all of the attacks that work against substitution ciphers, in *addition* to other attacks. The cipher's primary weakness comes from the fact that if the cryptanalyst can discover (by means of frequency analysis, brute force, guessing or otherwise) the plaintext of two cipher text characters, then the key can be obtained by solving a simultaneous equation.

The Algorithm

The 'key' for the Affine cipher consists of 2 numbers, we'll call them a and b. The following discussion assumes the use of a 26 character alphabet ($m = 26$). a should be chosen to be relatively prime to m (i.e. a should have no factors in common with m). For example 15 and 26 have no factors in common, so 15 is an acceptable value for a, however 12 and 26 have factors in common (e.g. 2) so 12 cannot be used for a value of a. When encrypting, we first convert all the letters to numbers ('a'=0, 'b'=1, 'z'=25). The ciphertext letter c, for any given letter p is (remember p is the number representing a letter):

$$c = ap + b \pmod{m}, \quad 1 \leq a \leq m, \quad 1 \leq b \leq m$$

The decryption function is:

$$p = a^{-1}(c - b) \pmod{m}$$

where $a{-}1$ is the multiplicative inverse of a in the group of integers modulo m.

To find a multiplicative inverse, we need to find a number x such that:

$$ax = 1 \pmod{m}$$

If we find the number x such that the equation is true, then x is the inverse of a, and we call it $a-1$. The easiest way to solve this equation is to search each of the numbers 1 to 25, and see which one satisfies the equation. If you want a more rigorous solution, you can use matlab to find x:

```
> [g,x,d] = gcd(a,m);    % we can ignore g and d, we dont need them
> x = mod(x,m);
```

If you now multiply x and a and reduce the result (mod 26), you will get the answer 1. Remember, this is just the definition of an inverse i.e. if $a*x = 1 \pmod{26}$, then x is an inverse of a (and a is an inverse of x).

We now use the value of x we calculated as a-1. This allows us to perform the decryption step.

Note: As stated above, m does not have to be 26, it is simply the number of characters in the alphabet you choose to use. If upper case characters, lowercase characters and spaces are used, then m will be 53. Digits and punctuation could also be incorporated (which again would change the value of m).

Assume we discard all non alphabetical characters including spaces. Let the key be a=5 and b= 7. The encryption function is then (5*p + 7)(mod 26). To encode:

'defend the east wall of the castle',

We would take the first letter, 'd', convert it to a number, 3 ('a'=0, 'b'=1, ..., 'z'=25) and plug it into the equation:

$$c = (5 * p + 7) \pmod{26}$$
$$c = (5 * 3 + 7) \pmod{26} = 22 \pmod{26} = 22$$

since 'w' = 22, 'd' is transformed into 'w' using the values a=5 and b= 7. If we continue with all the other letters we would have:

'wbgbuwyqbbhtynhkkzgyqbrhtykb'

Now to decode, the inverse of 5 modulo 26 is 21, i.e. 5*21 = 1 (mod 26). The decoding function is

$$p = 21 * (c - 7) \pmod{26}$$
$$p = 21 * (22 - 7) \pmod{26} = 315 \pmod{26} = 3$$

so we have recovered d=3 as the first plaintext character.

'defendtheeastwallofthecastle'

13.5 Bifid Cipher

Introduction

Bifid is a cipher which combines the Polybius square with transposition, and uses fractionation to achieve diffusion. It was invented by Felix Delastelle. Delastelle was a Frenchman who

invented several ciphers including the bifid, trifid, and four-square ciphers. The first presentation of the bifid appeared in the *French Revue du Génie civil* in 1895 under the name of *cryptographie nouvelle*.
It has never been used by a military or government organization, only ever by amateur cryptographers. Be wary of the Wikipedia page on bifid, it is almost entirely incorrect.

The Algorithm

Keys for the Bifid cipher consist of a 25 letter 'key square'. e.g.

```
  1 2 3 4 5
1| p h q g m
2| e a y l n
3| o f d x k
4| r c v s z
5| w b u t i
```

Note that there is no 'j' in the key-square, it is merged with the letter 'i'. The example below will encipher 'defend the east wall of the castle' using the key shown above.

When enciphering a plaintext, each letter is replaced by the numbers on the left hand side and top of the key square. These are then written on top of one another as shown in step 1 (below). E.g. 'd' is in row 3, column 3 of the key square so 3 is written in the top row and 3 is written in the second row. This is done for all plaintext letters. Step 2: The numbers are then grouped into blocks of a certain size (this is called the period, and forms part of the key). In this example the period is 5. The groups are then read off left to right (this is the fractionating step that makes bifid slightly more difficult to crack than a simple substitution cipher). Step 3 shows the new sequence of numbers after reading the groups left to right, first the top row of the group followed by the bottom row. The entire string is then re-enciphered using the original keysquare (shown in step 4) e.g. 'row 3, col 2' is 'f' in the original key square.

An example encryption using the above key:

Plaintext: defend the east wall of the castle

step 1: row 323223 512 2245 5222 33 512 424522

col 312153 421 1244 1244 12 421 224441

step 2: 32322 35122 24552 22335 12424 522

31215 34211 24412 44124 21224 441

step 3: 3232231215 3512234211 2455224412 2233544124 1242421224 522441

step 4: f f y h m k h y c p l i a s h a d t r l h c c h l b l r

Thus 'defendtheeastwallofthecastle' becomes 'ffyhmkhycpliashadtrlhcchlblr' using the key square shown above and a period of 5 during the enciphering step.

13.4 Vignere Gronsfeld Cipher

Introduction

The Vigenère Cipher is a polyalphabetic substitution cipher. The method was originally described by Giovan Battista Bellaso in his 1553 book La cifra del. Sig. Giovan Battista Bellaso; however, the scheme was later misattributed to Blaise de Vigenère in the 19th century, and is now widely known as the Vigenère cipher.

Blaise de Vigenère actually invented the stronger Autokey cipher in 1586.

The Vigenère Cipher was considered le chiffre ind hiffrable (French for the unbreakable cipher) for 300 years, until in 1863 Friedrich Kasiski published a successful attack on the Vigenère cipher. Charles Babbage had, however, already developed the same test in 1854. Gilbert Vernam worked on the vigenere cipher in the early 1900s, and his work eventually led to the one-time pad, which is a provably unbreakable cipher.

The Algorithm

The 'key' for a vigenere cipher is a key word. e.g. 'FORTIFICATION'

The Vigenere Cipher uses the following tableau (the 'tabula recta') to encipher the plaintext:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

B B C D E F G H I J K L M N O P Q R S T U V W X Y Z A

C C D E F G H I J K L M N O P Q R S T U V W X Y Z A B

D D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

E E F G H I J K L M N O P Q R S T U V W X Y Z A B C D

F F G H I J K L M N O P Q R S T U V W X Y Z A B C D E

G G H I J K L M N O P Q R S T U V W X Y Z A B C D E F

H H I J K L M N O P Q R S T U V W X Y Z A B C D E F G

I I J K L M N O P Q R S T U V W X Y Z A B C D E F G H

J J K L M N O P Q R S T U V W X Y Z A B C D E F G H I

K K L M N O P Q R S T U V W X Y Z A B C D E F G H I J

L L M N O P Q R S T U V W X Y Z A B C D E F G H I J K

M M N O P Q R S T U V W X Y Z A B C D E F G H I J K L

N N O P Q R S T U V W X Y Z A B C D E F G H I J K L M

O O P Q R S T U V W X Y Z A B C D E F G H I J K L M N

P P Q R S T U V W X Y Z A B C D E F G H I J K L M N O

Q Q R S T U V W X Y Z A B C D E F G H I J K L M N O P

R R S T U V W X Y Z A B C D E F G H I J K L M N O P Q

S S T U V W X Y Z A B C D E F G H I J K L M N O P Q R

T T U V W X Y Z A B C D E F G H I J K L M N O P Q R S

U U V W X Y Z A B C D E F G H I J K L M N O P Q R S T

V V W X Y Z A B C D E F G H I J K L M N O P Q R S T U

W W X Y Z A B C D E F G H I J K L M N O P Q R S T U V

X X Y Z A B C D E F G H I J K L M N O P Q R S T U V W

Y Y Z A B C D E F G H I J K L M N O P Q R S T U V W X

Z Z A B C D E F G H I J K L M N O P Q R S T U V W X Y

To encipher a message, repeat the keyword above the plaintext:

FORTIFICATIONFORTIFICATIONFO

DEFENDTHEEASTWALLOFTHECASTLE

Now we take the letter we will be encoding, 'D', and find it on the first column on the tableau. Then, we move along the 'D' row of the tableau until we come to the column with the 'F' at the top (The 'F' is the keyword letter for the first 'D'), the intersection is our cipher text character, 'I'.

So, the cipher text for the above plaintext is:

FORTIFICATIONFORTIFICATIONFO

DEFENDTHEEASTWALLOFTHECASTLE

ISWXVIBJEXIGGBOCEWKBJEVIGGQS

Variants

There are several ciphers that are very similar to the vigenere cipher.

The Gronsfeld cipher is exactly the same as the vigenere cipher, except numbers are used as the key instead of letters. There is no other difference. The numbers may be picked from a sequence, e.g. the Fibonacci series, or some other pseudo-random sequence.

The gronsfeld cipher is cryptanalysed in the same way as the vigenere algorithm, however the autokey cipher will not be broken using the kasiski method since the key does not repeat. The best way to break the autokey cipher is to try and guess portions of the plaintext or key from the cipher text, knowing they must both follow the frequency distribution of English text. Guessing how the plaintext begins is the easiest way of cracking the cipher.