Elements of Public-Key Cryptography and Primality Testing and Factoring

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Spring 2012 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

In the era of the Internet there are many electronic activities that require the exchange of information between two Internet based computers. Electronic mail, shopping, banking transactions are such activities that occasionally require secure communication. Given the nature of the Internet, information transmitted from computer A to computer B goes through an intermediate computer C, called the router. A router like C is a computer that can inspect the input, copy and forward the information to another computer and can also modify information.

Therefore secure information exchange requires methods that allow the secure exchange of information over insecure communication channels such as the Internet and the telephone network.

Research activities in the cryptographic field have developed a number of techniques to facilitate these activities that can be summarized in the following cryptographic applications.

a. **Encryption and Decryption** A message M that a computer is to transmit is called the plaintext. Encryption is the operation that transforms the plaintext M into an sequence of apparently indistinguishable characters called the ciphertext C. Decryption is the operation that transforms C back into M. The decryption normally relies on additional secret information that is available to the decryptor but not (hopefully) to third parties. Although encryption and decryption should be computationally feasible operations themselves, obtaining M from C should be computationally infeasible for someone not possessing the secret information that the decryptor has. Note that "computationally infeasible" does not mean that a third party can not recover M from C. It just means that a third party MAY NOT recover M from C in a reasonable amount of time and for a reasonable cost.

b. **Digital Signature processing** The creator A of a message M computes a message S derived from M and some secret information known to the creator. S is the digital signature of M if a third party can verify in a reasonable amount of time that the creator A is indeed the person who created M and S is his/her signature for M.

Computationally infeasible problems can be used to create secure encryption and decryption methods. Both sender and receiver will also have some additional pieces of information that allow them to solve the infeasible problems quickly, whereas an eavesdropper would need to spend a huge amount of time. The infeasible problems one can use can be provably intractable problems but also apparently intractable problems (i.e. NP-complete problems, problems like factoring etc.)

(c) Copyright A. Gerbessiotis. All rights reserved.

A public-key cryptosystem is a system that allows the encryption of messages sent between two parties in such a way than an eavesdropper will not be able to decrypt them. Such a system also allows a party to append an unforgeable digital signature to a an electronic piece of information (e.g. document, message).

According to Diffie and Hellman (1976) a public-key cryptosystem with encryption method P (to be known as the "public" key) and decryption method S (to be known as the "secret key") must be able to perform the following tasks.

1. For any message M , the following must always be true $S(P(M)) = M$.
2. It is computationally infeasible to derive S from P.
3. Both P and S are easy to compute.
4. $P(S(M)) = M$ for any M.

In a public-key cryptosystem, each participant has a **public key** and a **secret key**. For a party A (call it Alice), the public and secret keys are denoted respectively by P_A and S_A. For another party B (call it Bob), the keys are P_B and S_B respectively.

In such cases an encryption or digital signature of a message can be performed as follows.

```
ciphertext BobSendsAlice(message M)
    //Alice's public-key is P_A
1.  Bob gets Alice's P_A;
2.  C=P_A (M);
3.  return(C)
```

```
plaintext AliceReceivesBob (ciphertext C)
  // Alice's secret key is S_A
1. M = S_A (C);
2. return(M);
```

Since Alice's encryption and decryption methods $P_A()$ and $S_A()$ have property 1 of a public-key cryptosystem, we have that $S_A(P_A(M)) = S_A(C) = M$.

Suppose now that Alice wants to send Bob a signed message M'.

```
   AliceSigns (message M')
1. g = S_A (M');
2. return(<M',g>);
```

```
   Bobverifies(<message M', signature g>)
1. if ( M' == P_A (g))
2.  return("Alice sent M'; signature OK");
3. else
4.  return("Signature not valid");
```

Introduction

The public-key encryption system must be carefully chosen. The one used by Merkle-Hellman for example was weak. The most widely known PKC is the RSA cryptosystem of Rivest-Shamir-Adleman. This works as follows.

A participant created his public and private keys with the following procedure. Suppose it is Alice who runs this setup.

- Alice selects two large random prime numbers p_A and q_A of say 128-256 digits.
- The product $n_A = p_A q_A$ is then computed. Since n_A is the product of two primes $\phi(n_A) = (p_A - 1)(q_A - 1)$ is also computed.
- An **odd integer** e_A is selected that is relatively prime to $\phi(n_A)$. In practice e_A can be as little as 3.
- The multiplicative inverse d_A of e_A modulo $\phi(n_A)$ is then computed using the extended GCD algorithm.
- Alice publicizes her **RSA public-key** $P_A = (e_A, n_A)$.
- Alice stores in a secure place her **RSA secret key** $S_A = (d_A, n_A)$.
- The message domain is Z_n.
- The encryption method is $C = P(M) = M^{e_A} \bmod n_A$
- The decryption method is $M = S(C) = C^{d_A} \bmod n_A$.

(c) Copyright A. Gerbessiotis. All rights reserved.

```
   // Public key (e,n)        //Secret key (d,n)
   RSAEncryption(M)              RSADecryption(C)
1. C = M**e mod n;          1. M = C**d mod n;
2. return(C);               2. return(M);


   RSASignature(M)               RSAVerification(<M,S>)
1. S = M**d mod n;          1. temp = S**e mod n;
2. retun(<M,S>);            2. if  (temp == M) return ("Verification OK");
                            3.   else return("Verification failed!");
```

Theorem RSA. RSA works and defines inverse transformations.

Proof. $P(S(M)) = M^{e_A d_A} \bmod n_A = S(P(M))$. Since d_A and e_A are multiplicative inverses modulo $\phi(n_A)$ we have that $e_A d_A = 1 + k\phi(n_A) = 1 + k\phi(p_A)\phi(q_A)$. The latter is because $n_A = p_A q_A$ and thus $\phi(n_A) = (p_A - 1)(q_A - 1) = \phi(p_A)\phi(q_A)$. The message M is assumed to be a number between 0 and n_A, i.e. $0 < M < n_A$. This is one of the limitations of RSA if used for message encryption; because of this RSA is most frequently used as a key-exchange protocol. To show that RSA works we need to show that $M^{e_A d_A} \equiv 1 \pmod{n_A}$. One way to prove this is through the Chinese remaining theorem (proof in the textbook). An alternative method follows.

Case 1. Let $(M, n_A) = 1$ i.e. M and n_A are relatively prime. Then by Euler's Theorem we have that $M^{\phi(n_A)} \equiv 1 \pmod{n_A}$. The proof then easily follows.

$$\begin{aligned} M^{e_A d_A} &\equiv M^{1+k\phi(n_A)} \pmod{n_A} \\ &\equiv M M^{k\phi(n_A)} \pmod{n_A} \\ &\equiv M(M^{\phi(n_A)})^k \pmod{n_A} \\ &\equiv M(1)^k \pmod{n_A} \\ &\equiv M \pmod{n_A} \end{aligned}$$

Proof of correctness continued

Case 2. Let $(M, n_A) = d > 1$ i.e. M and n_A are not relatively prime.

The only non-trivial divisors of n_A are the prime numbers p_A, q_A. Since M has a common non-trivial divisor with n_A, d must be one of p_A and q_A. Therefore M can only be a multiple of one or the other; if M is a multiple of both p_A, q_A, then $M \geq n_A$ which is impossible since $M < n_A$.

Without loss of generality let us assume that M is a multiple of p_A i.e. that $M = mp_A$. Then $(M, q_A) = 1$. The latter implies that $M^{\phi(q_A)} \equiv 1 \pmod{q_A}$ after an application of Euler's theorem.

If we raise the latter to the power $k\phi(p_A)$ we obtain that $M^{k\phi(p_A)\phi(q_A)} \equiv 1 \pmod{q_A}$. Therefore $M^{k\phi(p_A)\phi(q_A)} = 1 + qq_A$ for some integer q. Multiplying both sides by M gives $M^{k\phi(p_A)\phi(q_A)+1} = M + Mqq_A$. Sime $M = mp_A$ this means that $M^{k\phi(p_A)\phi(q_A)+1} = M + mp_Aqq_A$. Sime $n_A = p_Aq_A$ this means that $M^{k\phi(p_A)\phi(q_A)+1} = M + mqn_A$. Set integer t the product of $t = mq$. $M^{k\phi(p_A)\phi(q_A)+1} = M + tn_A$ implies $M^{k\phi(p_A)\phi(q_A)+1} \equiv M \pmod{n_A}$ i.e. $M^{e_Ad_A} \equiv M \pmod{n_A}$. This proves Case 2.

Theorem RSA has thus been proven as a consequence of Cases 1 and 2.

(c) Copyright A. Gerbessiotis. All rights reserved.

Theorem RSA-Break. In order to break a cryptosystem like RSA we need to find d_A given e_A. This requires knowledge of $\phi(n_A)$. However, it is a **common belief** that knowledge of $\phi(n_A)$ requires knowledge of the factors p_A, q_A of n_A.

Because RSA is in practice quite slow, it is mainly used for key-management, i.e. key exchanges for non public-key cryptosystems. In such an approach Alice works as follows.

```
  AliceEncrypts (M,B) //Alice sends encrypted message to Bob
1. choose random key K;
2. S = Encrypt(M,K); //Encrypt (non  public-key) message M with key K
3. Get public-key P_B of Bob;
4. C= P_B (K);
5. return(<S,C>);


  BodDecrypts(<S,C>);
1. K= S_B (C);
2. M = Decrypt(S,K);
3. return(M);
```

Introduction

Public-key cryptosystems like RSA are often used with a public one-way hash function. For example RSA can be used in such a way for digital signature generation.

A one-way hash function is also referred to as a fingerprint or a message digest.

A one-way hash function h maps a message M of arbitrary length to an integer Y (such that $Y = h(M)$) that has a fixed number of bits, that is often called the digest of M, that satisfies the following properties: (a) given M it is easy to compute $h(M)$, (b) it is difficult to find M and N such that $h(M) = h(N)$ (collision-resistance property) or given $h(M)$ it is infeasible to find M (inversion-resistance property).

The MD5 hash function has 128-bit digests/fingerprints and the SHA-1 hash function has 160-bit fingerprints.

Under the RSA cryptosystem it is difficult to sign M if M is larger than n_A. What one can do however, is first hash the message and then sign the hashed message. In such a case Alice transmits not $< M, S(M)$ but $< M, S(h(M)) >$.

(c) Copyright A. Gerbessiotis. All rights reserved.

Introduction

If Alice and Bob want to exchange a secret key they can perform the following key exchange operation.

```
  Diffie-Hellman-Key-Exchange()
1. Alice and Bob exchange prime p and generator g
   of Z_p.
2. Alice chooses random a and sends Bob   B= g**a mod p;
3. Bob   chooses random b and sends Alice A= g**b mod p;
4. Alice computes K= (g**b)**a mod p;
5. Bod   computes K= (g**a)**b mod p;
6. Alice and Bob have agreed on K.
```

(c) Copyright A. Gerbessiotis. All rights reserved.

1. Alice picks a number a and hashes it B= h(a) and sends B to Bob.
2. Bob receives B and sends Alice a guess (i.e. 0 or 1).
3. Alice checks a and concludes whether it is odd (1) or even (0).
4. If Alice's conclusion agrees with Bob's guess Alice declares "H" else Alice declares "T" as result of coin flip.
5. Alice sends Bob result of coin flip and guess a.
6. Bob checks that Alice did not cheat (i.e. Bob compares oddness or evenness of a with the announced result.

(c) Copyright A. Gerbessiotis. All rights reserved.

Someone claims he can count the leaves of any tree almost instantly but he is not willing to reveal his method. How can you test him that he is telling the truth or that he is a cheater?

Introduction

One of the requirements of the RSA cryptosystem is the ability of finding large-size prime numbers quickly and the intractability of factoring composite numbers.

One of the problems that is of interest is how easy it is to find a prime number and of course how easy is it to verify whether a number is prime.

In order to answer the first question the Prime Number Theorem comes to assistance.

Theorem Density of Primes. The density of prime numbers less than n is $\ln n$, i.e. thare are $\pi(n)$ prime numbers less than n, where $\pi(n) = \Theta(n/\ln n)$.

The other problem we will be dealing with next is the following one.

Problem. Given a $\lg p$-bit integer p, is p a prime?

Some elementary algorithms for primality testing test all integers up to $\sqrt{p}$ as potential factors for p. Such methods however are too slow. The resulting algorithms are not polynomial in the number of bits of p, i.e. their running time is not $O(\lg^{\Theta(1)} p)$ but rather exponential in this number i.e. $\Omega(p^{\Theta(1)}) = \Omega(2^{\Theta(\lg p)})$.

The test for primality testing that we will introduce is based on Fermat's little theorem.

Fermat's Little Theorem. For any prime number p, and any $0 \leq a < n$.

$$a^{p-1} \equiv 1 \pmod{p}$$

(c) Copyright A. Gerbessiotis. All rights reserved.

```
   FirstPrimalityTest (p)
1. while ( 1 ) {
2.  Choose at random a number a between 1 and p-1.
3.  c= gcd(a,p);
4   if (c != 1)  return("p is not prime");
}
```

The number of divisors of p that are less than p is $p - \phi(p)$. If p is large and has a single big factor, i.e. $p = f^2$, then the running time of the test will be non-polynomial.

```
   SecondPrimalityTest (p)
1. while ( 1 ) {
2.  Choose at random a number a between 1 and p-1.
3.  Compute  c= a**(p-1) mod p.
4   if (c != 1)  return("p is not prime");
    // Lines 5-6 repeat the test of FirstPrimalityTest
5.  if  (gcd(a,p) != 1)
6.    retun("A factor of p is gcd(a,p));
}
```

Algorithm SecondPrimalityTest is a primality testing algorithm based on Fermat's Little Theorem. If a number a is found such that $a^{p-1} \neq 1 \bmod p$, then we know that p is not a prime number. If such an a however does not exist then p may or may not be a prime. In fact there are numbers n called **Carmichael numbers** that have the Fermat's Little theorem property. 561, 1105 and 1729 are Carmichael numbers and composites.

We give below an outline of the probabilistic algorithm that we will use for primality testing. The algorithms we are going to present in the next few sections are randomized algorithms, i.e. they rely on the existence of a random number generator that they use. A randomized algorithm would recognize elements of a language L. In our case, L would contain all prime numbers (or all composite numbers depending on the context).

Monte Carlo, Las Vegas and Atlantic City

Let L be a language (e.g. set of prime numbers) accepting a recognition algorithm A, i.e. for every string $x \in L$, A on input x, A accepts x, while for every string $x \notin L$, A rejects that x.

Class RP (Monte Carlo Algorithms). The class RP (Randomized Polynomial time) consists of all languages L that have a randomized algorithm A running in **worst-case polynomial time** such that for any input x,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is at least $1/2$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is 0 (i.e. $A(x)$ rejects).

An RP algorithm is a Monte Carlo algorithm that can err only for an $x \in L$. This is what we call an one-sided error. co-RP is the set of languages L whose complement $\bar{L}$ is in RP. The class co-RP errs only for $x \notin L$. A problem belonging to both RP and co-RP can be solved by a randomized algorithm with zero-sided error i.e. a Las-Vegas algorithm. A Las-Vegas algorithm is a Monte-Carlo algorithm for $x \notin L$ and a Monte-Carlo algorithm for $x \in L$.

Class ZPP (Las Vegas Algorithms). The class ZPP (Zero-error Probabilistic Polynomial time) is the class of languages that have a Las Vegas algorithm A running in **expected polynomial time**.

1. for any $x \in L$ then, $A(x)$ accepts with probability 1.
2. for any $x \notin L$ then, $A(x)$ accepts with probability 0 (i.e. $A(x)$ rejects).

Class PP. The class PP (Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in **worst-case polynomial time** such that for any input x,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is greater than $1/2$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is less than $1/2$.

(c) Copyright A. Gerbessiotis. All rights reserved.

The casinos (continued)

Class BPP (Atlantic-City Algorithms). The class BPP (Bounded-error Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in **worst-case polynomial time** such that for any input x,

1. for any $x \in L$ then, the probability that $A(x)$ accepts is greater than $3/4$, and
2. for any $x \notin L$ then, the probability that $A(x)$ accepts is less than $1/4$.

The $3/4$ and $1/4$ bounds are arbitrary and we can have $1/2 + \epsilon$ and $1/2 - \epsilon$ as long as ϵ is bounded below by the inverse of a polynomial in input size $\mid x \mid$.

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

The Miller-Rabin Algorithm

```
  MillerRabin(n, confidence)
  // Run the algorithm with error rate at most 1-confidence
1. rounds =  lg(-(1-confidence));
2. for(i=0; i < rounds; i++) {
3.  a = RANDOM(1..n-1);
4.  if (Witness(a,n) == TRUE)
5.   return(COMPOSITE);
6. }
7. return(PRIME);

  Witness(a,n)
1. n-1 = 2**t u; // u odd
2. x[0]= a**u mod n;
3. for(i=1;i<=t;i++){
4.  x[i]=(x[i-1]*x[i-1]) mod n;
5.  if (x[i]==1) && (x[i-1] != 1) && (x[i-1] != (n-1))
6.   return(TRUE);
7. }
8. if (x[t] != 1)
9.   return(TRUE);
10.return(FALSE);
```

(c) Copyright A. Gerbessiotis. All rights reserved.

The performance of the Miller-Rabin primality testing algorithm is based on the following proposition.

Proposition. If there is a non-trivial square root of 1 mod n, then n is composite.

The algorithm always responds correctly when it says "n is composite". If however, the algorithm responds with an "n is prime", it may occassionally be wrong (i.e n may be composite but is reported to be prime).

The algorithm tries to detect "witnesses" i.e. non-trivial square roots of 1 mod n. A z attests the compositeness of n, if $z^2 \equiv 1 \bmod n$, and $z \neq \pm 1$. If such a root is detected, the algorithm returns "n is composite" (TRUE). The algorithm is wrong when the "witness" a is not among the at least $(n-1)/2$ numbers between 1 and $n-1$ that can attest the "compositeness" of n.

The compositeness of even numbers can be detected immediately. In any other case n must be odd. In such a case $n-1$ is of the form $n-1 = 2^t u$, for some $t \geq 1$. The binary representation of n establishes whether $x_i \equiv a^{2^i u} \bmod n$ is a non-trivial square root of n, i.e. $x_i^2 \bmod n \equiv 1 \pmod n$. Line 5 of the algorithm checks whether x_i is such a square root by making sure that x_i is neither 1 nor $-1 \bmod n = (n-1) \bmod n$. If the loop of lines 3-7 completes, this means that $x_t = a^{n-1} \not\equiv 1 \pmod n$ and thus n is not prime by FERMAT's little theorem (line 9). Otherwise the test for witness a is inconclusive and a FALSE is returned in line 10.

Note that FALSE does not necessarily mean that n is prime. It just means that a is not good enough to prove the compositeness of n.

Error rate in Miller-Rabin. Out of the $n-1$ possible witnesses, at least $(n-1)/2$ return TRUE either through line 6 or through line 9. This means that the probability that a is not good enough is at most 1/2. The probability that Miller-Rabin exits and returns PRIME (when it is not) is less than $1/2^{\text{rounds}}$ and therefore the confidence in its report is at least confidence$= 1 - 1/2^{\text{rounds}}$.

A Polynomial Time Algorithm

Until August 5, 2002, there was no algorithm for testing the primality of n in polynomial time. On August 6, M. Agrawal, N. Kayal, and N. Saxena, all of the Indian Institute of Technology in Kanpur announced a primality testing algorithm whose worst-case running time is about $O(\lg^{12} n)$ but in practice the algorithm has complexity about $O(\lg^6 n)$.

```
Input: Integer n > 1

 if (n has the form n= a**b with b > 1) then output COMPOSITE

 Find the smallest r such that o_r (n) > 4*logn *logn
 where o_r (n) is the smallest k such that  n**k = 1 mod r.

 If 1 < (a,n) < n for some a<= r output COMPOSITE

 If n <=r output PRIME

 For a=1 to floor ( 2 sqrt(phi(r)) logn ) do
   if (( X+a)**n != X**n +a (mod X**r -1 , n) ) output COMPOSITE

 output PRIME;
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Pratt's witnesses

In this section we solve a similar problem to primality testing. We show how to prove that a number p is a prime number. We would like to achieve this in an efficient way.

Let p be a prime number. Then $p-1$ is a composite. If it is a composite, then the number of divisors of $p-1$ can not be more than $\lg p$ as each divisor is at least equal to 2.

Then $p-1 = \prod p_i^{a_i} \geq \prod 2^{a_i} = 2^s$ and as a result $s \leq \lg p$.

In order to convince say, Alice that p is a prime, we will present her with polynomial in $\lg p$ number of "witnesses" to the primality of p. We will utilize the following lemma

Lemma. Let p be a positive integer and x be an integer such that $x^{p-1} \equiv 1 \pmod{p}$, and for any divisor d of $p-1$ we have that $x^d \not\equiv 1 \pmod{p}$. Then p is a prime number.

Proof. We note that if p is a prime number by Fermat's Little Theorem we must have that $a^{p-1} \equiv 1 \pmod{p}$ for all a such that $1 \leq a \leq p-1$.

We show first that $gcd(x,p) = 1$. Let otherwise be $t = gcd(x,p)$. Then $x = tx_1$ and $p = tp_1$. Since $x^{p-1} \equiv 1 \pmod{p}$ we obtain that $x^{p-1} = 1 + kp$ for some integer k. Then $x^{p-1} - kp = (tx_1)^{p-1} - ktp_1 = 1$. The left hand side of the second inequality is multiple of t. So must be the right hand side. This is only possible if $t = 1$.

Thus $gcd(x,p) = 1$ as required. This also shows that $x \in Z_p^*$.

The order of x in Z_p^* must be $p-1$. Since otherwise if the order of x it r we would have $x^r \equiv 1 \pmod{p}$ which would imply that r divides $p-1$. The latter however is impossible by the properties of any divisor d of $p-1$. The order of Z_p^* is $\phi(p)$. We just proved that the order of x is $p-1$ and therefore $p-1/\phi(p)$ and also $p-1 \leq \phi(p)$. The maximum value of $\phi(p)$ is $p-1$ i.e. $\phi(p) \leq p-1$. This implies that $\phi(p) = p-1$ i.e. p is a prime.

(c) Copyright A. Gerbessiotis. All rights reserved.

Pratt's witnesses (continued)

Normally a certificate to the primeness of p denoted by $W(p)$ is a tuple defined recursively for all prime divisors of $p-1$.

$$W(p) = \langle x, W(p_1), W(p_2), \ldots W(p_k) \rangle$$

```
   PrattPrimeVerifyWitness(p,W(p))
   // Prove that p is a prime number.
1. assert(p-1 == p1**a1 p2**a2 ... pk**ak);
2. for(i=1 ; i <= k ; i++) {
3.  PrattPrimeVerifyWitness( pi );
   }
4. for(i=1 ; i <= k ; i++) {
5.    assert(x**((p-1)/pi) != 1 (mod p));
   }
6. assert(x**(p-1) == 1 (mod p));
   }
```

Let the running time of the procedure be $T(p)$
Then,

$$T(p) = 1 + \sum_i T(p_i) + k + 1$$

One step for the running time means 1 step of an operation in line 1 or 5 or 6; in practice this is not constant time. By changing variables $T(p) + 1 = P(p)$ we have that

$$P(p) = 3 + \sum_i P(p_i)$$

we conclude by using the substitution method that $P(p) = O(\lg p)$.

(c) Copyright A. Gerbessiotis. All rights reserved.

Pollard's Rho Method

Problem. Given an integer n, decompose n into a product of primes.

The naive way to find the factors of n is to use trial division and try all integers up to $\sqrt{n}$.

In other words, trial division of all numbers up to N can factor all numbers up to N^2. Then, all numbers up to N^4 can also be factored. Pollard's rho method that is described below, uses trial division for numbers up to N to factor number up to N^4. This method is a heuristic and no guarantee of success is given.

The idea behind Pollard's rho method is as follows. If n is not prime, then by picking at random two x, y integers less than n, then, if $gcd(x-y, n) \neq 1$ a factor of n has been established. Instead of picking random x and y's the algorithm starts with a random x, and generates numbers according to the sequence $x_i = x_{i-1}^2 - 1 \bmod n$.

```
      PollardRho (n)
1. i=1;
2. x[1] = RANDOM(0..n-1); // Choose a random x
3. y=x[1];                // Every 2**k trials y is updated
4. k=2;
5. while (1) {
6.  i++;
7.  x[i] = x[i-1]**2 -1 mod n; //  Use x*x -1 mod n
8.  d = gcd(y-x[i],n);         // Do (y- ((x*x-1) mod n)) and  n
9.  if (d !=1) && (d != n ) {  // have non trivial divisors
10    output(d);
11. }
12. if (i == k ) {
13.  y=x[i];
14.  k=2*k;
15. }
16.}
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Pollard's Rho Method: The analysis

Pollard's method works as follows. At iteration i, $x_i = x_{i-1}^2 - 1 \pmod{n}$ following the assignment of line 7. An infinite sequence $x_1, x_2, \ldots$ is then generated. From time to time an x_k is saved into y. The first time a save is performed at $k = 2$, the next time at $k = 4$, then at $k = 8$, and in general the k-th save is at $i = 2^k$. Note that as soon as $x_i = y = x_k$ a cycle has been found, and the x's that follow just retrace the cycle. Since x_1 is a random number, if we "assume" that all intermediate x_i's are also random, the birthday paradox tells us that $O(\sqrt{n})$ values are sufficient to obtain a cycle i.e. two values that are the same.

If p is a prime divisor of n then the same recurrence gives the following sequence mod$p$

$$y_{i+1} = y_i^2 - 1 \pmod{p}$$

where $y_i = x_i \bmod p$. Reasoning as before we conclude that the sequence y_i repeats after $O(\sqrt{p})$ of steps.

In general for a cycle of length c, and t the index of the first repeated value we have that $y_{t+i} = y_{t+i+c}$, with $t, c = O(\sqrt{p})$. The former implies that p is such that $p/x_{t+i+c} - x_{t+i}$. This is equivalent to $gcd(x_{t+i+c} - x_{t+i}, n) > 1$. (If the value saved in y is an x_k with $k \geq t$, the x_k is on the cycle and it is bound to be identified.) Thus p will be output through line 10 in $O(\sqrt{p})$ steps. Since $p = O(\sqrt{n})$, this implies that a factor can be found on the average in $O(n^{1/4})$ steps.

(c) Copyright A. Gerbessiotis. All rights reserved.