

# Probabilistic algorithms

Mathematical Cryptography,  
Lectures 17 - 18

- 1 Probabilistic algorithms
  - Probabilistic algorithms
  - Discrete probability distributions
  - Generate and test algorithm
  
- 2 Generating random numbers
  - Generating random numbers
  - Generating random primes
  - Generating random factorized numbers

## Probabilistic algorithms

### Generating random bits

Suppose we have an algorithm that generates a random bit (we'll leave aside how). So we have a new instruction, on the same level as the arithmetic instructions "sum of two bits" and "product of two bits",

$$\gamma \leftarrow \text{RAND},$$

which randomly assigns into the variable  $\gamma$  zero or one so that

- $P[\gamma = 1] = P[\gamma = 0] = \frac{1}{2}$ ,
- the result of the *RAND* instruction does not depend on its previous calls.

We will assume that one call of the *RAND* instruction takes constant time  $O(1)$ .

## Probabilistic algorithms

### Definition

- *Probabilistic algorithms* are algorithms that use the *RAND* instruction.
- *Deterministic algorithms* do not use the *RAND* instruction.

In algorithm, we will denote random generation as follows:

- $y \xleftarrow{\$} \{0, 1\}$  generates a random bit at time  $O(1)$
- $y \xleftarrow{\$} \{0, 1\}^{\times l}$  generates a string of length  $l$  of random bits at time  $O(l)$

## Probabilistic algorithms

For a given probabilistic algorithm  $A$  and an input  $x$ , we introduce random variables:

- $LOOPS$  = number of loop iterations executed in a given run of the algorithm
- $LOOPTIME$  = time for executing one loop
- $TIME$  = total running time of the algorithm
- $OUTPUT$  = value of the output at a given execution of the algorithm

The values of these random variables will depend on results of the  $RAND$  instructions in a given run of the algorithm  $A$  with input  $x$ .

We are interested in the expected running time of the algorithm  $A$  with input  $x$  (the expectation of the random variable  $TIME$ ), and in the probability distribution of the random variable  $OUTPUT$ .

## Probability space

More precisely, we create a countable probability space simulating a behavior of the probabilistic algorithm  $A$  with input  $x$  under different outcomes of the  $RAND$  instructions.

- $\Omega = \{\omega \in \{0, 1\}^{\times I}, I \in \mathbb{N}; \omega = \text{exact execution path}\}$   
An exact execution path is a sequence 0 a 1 in which each member corresponds to one algorithm instruction, so the last member corresponds to the  $HALT$  instruction. Furthermore, if the  $i$ -th instruction is  $RAND$ , then the  $i$ -th member is used as the result of the  $RAND$  instruction.
- $P(\omega) = \frac{1}{2^{|\omega|}}$ , where  $|\omega| = I$  is the length of the sequence  $\omega$ .
- It can be shown that  $\sum_{\omega \in \Omega} 2^{-|\omega|} = \alpha \leq 1$ . We say that the algorithm  $A$  stops at input  $x$  with probability  $\alpha$ . If  $\alpha = 1$ , then  $P : \Omega \mapsto \langle 0, 1 \rangle$  is a probability function on  $\Omega$ .

## Random variable

The random variables will then be mappings defined on  $\Omega$ :

- $TIME(\omega) = |\omega|$
- $OUTPUT(\omega)$  = the output of the algorithm  $A$  with input  $x$ , if the sequence  $\omega$  simulates the algorithm execution (it has results of the  $RAND$  instructions in the corresponding places).

Then we calculate the probability that the running time equals to  $l$  as follows:

- $P[TIME = l] = P(\{\omega \in \Omega, TIME(\omega) = l\}) = \frac{s}{2^l}$ , where  $s$  = the number of exact execution paths of length  $l$ .

We will treat random variables on an intuitive level, without going into the details of the computation in the relevant probability space.

## Discrete probability distributions

### Discrete probability distributions

Let the random variable  $X$  take countably many values from the set  $M = \{x_i, i \in I\}$ , where  $I \subseteq \mathbb{N}$ .

A probability function  $P$  is any nonnegative function from  $M$  to  $R$  that satisfies  $\sum_{i \in I} P[X = x_i] = 1$ .

A probability function defines a discrete probability distribution of the random variable on the set  $M$ .

### Expectation of a random variable

The *expectation* of the random variable  $X$  is defined as a weighted average:  $E(X) = \sum_{i \in I} x_i P[X = x_i]$ .

It is a linear function,  $E(aX + bY) = aE(X) + bE(Y)$  for any random variables  $X, Y$  on the set  $M$  and  $a, b \in R$ .

## Discrete probability distributions

- A random variable  $X$  has a *uniform distribution* on the set  $\{1, 2, \dots, m\}$  in case  $P[X = i] = \frac{1}{m}$  for every  $1 \leq i \leq m$ . The expectation is  $E(X) = \frac{m+1}{2}$ .
- A random variable  $X$  has an *alternative distribution* with parameter  $p$  on the set  $\{0, 1\}$  in case  $P[X = 1] = p$ ,  $P[X = 0] = 1 - p$ . The expectation is  $E(X) = p$ .
- The random variable  $X$  has a *geometric distribution* with parameter  $p$  on the set  $\{1, 2, 3, \dots\} = \mathbb{N}^+$  in case  $P[X = i] = (1 - p)^{i-1}p$  for every  $i \geq 1$ . The expectation is  $E(X) = \frac{1}{p}$ .  
(If an experiment has an alternative distribution with the probability of success  $p$ , then the geometric distribution gives the probability that the first success occurs in the  $i$ -th iteration of the experiment.)

## Probabilistic algorithms

### Algorithm - coin toss until heads comes up

- repeat  $y \stackrel{\mathcal{C}}{\leftarrow} \{0, 1\}$
- until  $y = 1$

### Algorithm analysis

The probability that the algorithm stops after one cycle is  $\frac{1}{2}$ . The random variable  $LOOPS$  has a geometric distribution with parameter  $p = \frac{1}{2}$ , the expected number of loops is  $E(LOOPS) = 2$ . The probability that the number of loops is at least  $k$  is equal to  $P[LOOPS \geq k] = \frac{1}{2^{k-1}}$ , so  $\lim_{k \rightarrow \infty} P[LOOPS \geq k] = 0$ . The algorithm stops with probability 1, even though the number of algorithm steps is not bounded. The situation that the algorithm does not stop has zero probability.

## Probabilistic algorithms

### Algorithm GT (= Generate and test)

We have two probabilistic algorithms  $A(x)$  and  $B(x, y)$ , where  $B$  returns *true* or *false*. The algorithm  $GT(x)$  combines them both:

- repeat  $y \leftarrow A(x)$
- until  $B(x, y)$
- output  $y$

### Analysis of the GT algorithm

- If  $A$  stops with probability 1 on input  $x$  and for each output  $y$  it is true that  $B$  stops with probability 1 on input  $(x, y)$ , while for some  $y$  the probability that  $B(x, y)$  returns *true* is positive, then  $GT$  also stops with probability 1 on input  $x$ .

## Probabilistic algorithms

### Analysis of the GT algorithm

Let  $\mathcal{H}_1$  be the event that the algorithm halts after the first loop iteration, and  $T$  be the set of all possible outputs of the algorithm  $A(x)$ .

- The random variable  $LOOPS$  has a geometric distribution with parameter  $p = P[\mathcal{H}_1]$ .
- $E(TIME) = E(LOOPS) E(LOOPTIME) = \frac{1}{p} E(LOOPTIME)$
- For each  $t \in T$ ,  $P[OUTPUT = t] = P[OUTPUT = t | \mathcal{H}_1]$

## Generating random numbers

### Algorithm RN (= Random number)

Input: a natural number  $m \geq 1$

Output: a random natural number less than  $m$

- $l \leftarrow \lceil \log_2(m) \rceil$  (So  $2^{l-1} < m \leq 2^l$ )
- repeat
  - $y \xleftarrow{\mathcal{U}} \{0, 1\}^{\times l}$  (String type)
  - $n \leftarrow \sum_{i=0}^{l-1} y_i 2^i$  (Integer type)
- until  $n < m$
- output  $n$

## Generating random numbers

### Analysis of the RN algorithm

- Algorithm stops with probability 1.
- *LOOPS* has a geometric distribution with parameter  $p = \frac{m}{2^l} > \frac{1}{2}$ , so the expected number of loop iterations is  $E(\text{LOOPS}) < 2$ .
- The time required for one loop is  $O(l)$ , the expected time is  $E(\text{TIME}) \in O(2l) = O(l)$ , where  $l = \text{len}(m)$ .
- The output is uniformly distributed over the set  $\{0, \dots, m-1\}$ , so  $P[\text{OUTPUT} = n] = \frac{1}{m}$  for each  $0 \leq n \leq m-1$ .

## Generating random numbers

### Algorithm RN (= Random number)

Input: a natural numbers  $1 \leq m_1 < m_2$

Output: a random natural number from the interval  $\{m_1, \dots, m_2\}$

- $l \leftarrow \lceil \log_2(m_2 + 1) \rceil$
- repeat
  - $y \xleftarrow{\mathcal{U}} \{0, 1\}^{\times l}$  (String type)
  - $n \leftarrow \sum_{i=0}^{l-1} y_i 2^i$  (Integer type)
- until  $m_1 \leq n \leq m_2$
- output  $n$

The expected time is  $O(l)$  and the output is uniformly distributed over the set  $\{m_1, \dots, m_2\}$ .

## Generating random numbers

### Algorithm GT (= Generate and test) more specifically

Input: a finite set  $T$  and its non-empty subset  $T'$

Output: a random element of  $T'$

- repeat  $y \xleftarrow{\mathcal{U}} T$
- until  $y \in T'$
- output  $y$

We assume that we can randomly generate an element from  $T$  in the expected time  $O(f)$  and that the output of this algorithm is uniformly distributed over the set of  $T$ .

We further assume that we can efficiently test that  $y \in T'$  in the expected time  $O(g)$ , and that  $T' \neq \emptyset$ .

Moreover in doing so, both algorithms stop with probability 1.

## Generating random numbers

### Analysis of the GT algorithm

- The GT algorithm stops with probability 1.
- *LOOPS* has a geometric distribution with parameter  $p = \frac{|T'|}{|T|} > 0$ , so the expected number of loops is  $E(\text{LOOPS}) = \frac{|T|}{|T'|}$ .
- The expected time of one loop is  $E(\text{LOOPTIME}) \in O(f + g)$ , so the total expected time is  $E(\text{TIME}) \in O\left(\frac{|T|}{|T'|}(f + g)\right)$ .
- The output is uniformly distributed over the set  $T'$ , so  $P[\text{OUTPUT} = t] = \frac{1}{|T'|}$  for each  $t \in T'$ ,  $P[\text{OUTPUT} = t] = 0$  for each  $t \in T \setminus T'$ .

## Generating random primes

### Algorithm RP (= Random prime)

Input: a natural number  $m \geq 2$

Output: a random prime between 2 and  $m$   
(or a random  $l$ -bit prime)

- repeat  $n \xleftarrow{\$} \{2, \dots, m\}$   
(resp.  $n \xleftarrow{\$} \{2^{l-1}, \dots, 2^l - 1\}$ )
- until *IsPrime*( $n$ )
- output  $n$

An *IsPrime*( $n$ ) algorithm for testing primality is a "black box" for now, which returns *true* or *false* for every  $n \in \mathbb{N}$ .

## Density of prime numbers

For the time analysis of the algorithm RP, we need to estimate how many primes there are and what is their "density" among the natural numbers.

### Euclidean theorem

There are infinitely many primes.

### Proposition

It can be found  $n$  consecutive composite numbers for every  $n \in \mathbb{N}$ . (There are arbitrarily large "holes" between primes.)

## Density of prime numbers

Let  $\pi(m)$  denote the number of primes between 1 and  $m$ , including  $m$ .

### Chebyshev's theorem

For every natural number  $m \geq 2$ ,  $\pi(m) \in \Theta\left(\frac{m}{\ln(m)}\right)$ .

### Proposition

For each natural number  $m \geq 2$ :  $\pi(m) \geq \frac{\ln(2)}{2} \cdot \frac{m}{\ln(m)} \doteq 0.35 \cdot \frac{m}{\ln(m)}$

## Density of prime numbers

### Consequence

Since  $\ln(m) = \frac{\log_2(m)}{\log_2(e)}$ , it is also true that  $\pi(m) \in \Theta(\frac{m}{\ln(m)})$ , or there exist  $c_1, c_2 > 0$  such that for all  $m \geq m_0$  holds:

$$c_1 \frac{1}{\ln(m)} < \frac{\pi(m)}{m} < c_2 \frac{1}{\ln(m)}$$

### Note

There is together 168 primes up to  $m = 1000$ .

The Chebyshev's estimate is  $\frac{m}{\ln(m)} = \frac{1000}{\ln(1000)} \doteq 145$ .

While  $\ln(1000) \doteq 10$ , our estimate  $\frac{m}{\ln(m)} = \frac{1000}{\ln(1000)} \doteq 100$  is slightly more inaccurate.

## Density of prime numbers

### Bertrand's postulate

For every natural number  $m \geq 1$ ,  $\pi(2m) - \pi(m) > \frac{m}{3 \ln(2m)}$ .  
Or, there is  $\Omega(\frac{m}{\ln(m)})$  primes between  $m$  and  $2m$ .

### Consequence

There exists  $c > 0$  such that for all  $m \geq m_0$  the following holds:

$$c \frac{1}{\ln(m)} < \frac{\pi(2m) - \pi(m)}{m}$$

## Generating random primes

### Analysis of the RP algorithm for deterministic IsPrime(n)

- We assume that the algorithm  $IsPrime(n)$  works in time  $O(\tau(l))$  for all  $n \leq m$ , where  $l = \ln(m)$ , and that  $\tau(l) > l$ . Then the time required for one loop is  $O(\tau(l))$ .
- *LOOPS* has a geometric distribution with parameter  $p = \frac{\pi(m)}{m-1}$ , or with parameter  $p = \frac{\pi(2^l) - \pi(2^{l-1})}{2^l - 1}$  for an  $l$ -bit prime. In both cases we have an estimate  $p > \frac{c}{l}$  for an appropriate constant  $c \doteq \frac{1}{3}$ , thanks to the Chebyshev's and Bertrand's theorems.
- The expected time is  $E(TIME) \in O(l\tau(l))$ .
- The output is uniformly distributed over the set of all primes.

## Generating random primes

### Analysis of the RP algorithm for probabilistic IsPrime(n)

Let the algorithm  $IsPrime(n)$  be a probabilistic algorithm that is burdened with a one-sided error: for a prime  $n$  the answer *true* is certain, for a composite number  $n$ , the answer *true* is also possible with probability at most  $\epsilon$ .

- We will assume that the algorithm  $IsPrime(n)$  works in expected time  $O(\tilde{\tau}(l))$  for all  $n \leq m$ , where  $l = \ln(m)$ , and that  $\tilde{\tau}(l) > l$ . Then the expected time for one loop is  $O(\tilde{\tau}(l))$ .
- *LOOPS* has a geometric distribution with parameter  $p > \frac{\pi(m)}{m-1}$ , since the algorithm may terminate even for a composite number  $n$ . Thanks to Chebyshev's theorem, we get  $p > \frac{c}{l}$  for suitable  $c \doteq \frac{1}{3}$ .
- The expected time is  $E(TIME) = E(LOOPTIME)E(LOOPS) \in O(l\tilde{\tau}(l))$ .

## Generating random primes

### Analysis of the RP algorithm for probabilistic $IsPrime(n)$

- $P[OUTPUT = n] = \beta < \frac{1}{\pi(m)}$  is the same for every prime  $n \leq m$ .
- $P[OUTPUT = n] > 0$  also for every composite  $n \leq m$ .
- We estimate the total probability of the event that the output is a composite number:

$$P[n \text{ composite} | IsPrime(n)] = \frac{P[IsPrime(n) | n \text{ composite}] P[n \text{ composite}]}{P[IsPrime(n)]}$$
$$< \frac{\epsilon}{\pi(m)} < \frac{\epsilon l}{c}$$

The probability that the output is composite is  $O(\epsilon l)$ .  
(The estimate is very rough, but sufficient in order to answer the question how small  $\epsilon$  should we choose.)

## Generating random factorized numbers

### Algorithm RS (= Random non-increasing sequence)

Input: a natural number  $m \geq 2$

Output: a non-increasing sequence of numbers between 1 and  $m$

- $n_0 \leftarrow m, k \leftarrow 0$
- repeat
  - $k \leftarrow k + 1$
  - $n_k \leftarrow^{\mathcal{U}} \{1, \dots, n_{k-1}\}$  (the same number can be chosen again)
- until  $n_k = 1$
- output  $(n_1, \dots, n_k)$

The expected time is  $O(l^2)$ , where  $l = \text{len}(m)$  (the intervals will roughly halve, we expect  $\log_2(m)$  random selections).

## Generating random factorized numbers

### Algorithm RFN (= Random factorized number)

Input: a natural number  $m \geq 2$ , (or number of bits  $l$ )

Output: a random factorized number  $n \leq m$ , (or  $l$ -bit  $n$ )

- repeat
  - generate a non-increasing sequence of numbers up to  $m$  using the RS algorithm, we get the sequence  $(n_1, \dots, n_k)$  in which numbers can repeat
  - select a subsequence of all primes using  $IsPrime(n_i)$ , we get the sequence  $(p_1, \dots, p_s)$  and we keep all duplicates
  - $n \leftarrow \prod_{i=1}^s p_i$  (once multiplication exceeds  $m$ , we will not multiply further)
  - $x \leftarrow^{\mathcal{U}} \{1, \dots, m\}$  (to ensure that  $n$  is randomly large enough)
- until  $x \leq n \leq m$  (or  $2^l \leq n < 2^{l+1}$ )
- output  $n, (p_1, \dots, p_s)$

## Generating random factorized numbers

### Analysis of the RFN algorithm

For a deterministic  $IsPrime(n)$  operating in time  $O(\tau(l))$ :

- The expected time is  $E(TIME) \in O(l^2 \tau(l))$ .
- The output is uniformly distributed over the set  $\{1, \dots, m\}$ .

For a probabilistic  $IsPrime(n)$  operating in expected time  $O(\tilde{\tau}(l))$ , which can return *true* for a composite number with probability at most  $\epsilon$ :

- The expected time is  $E(TIME) \in O(l^2 \tilde{\tau}(l))$ .
- All correctly factorized outputs have the same probability.
- The total probability that the output is incorrectly factorized is  $O(\epsilon l^2)$  for sufficiently small  $\epsilon$ , roughly  $\epsilon l \leq \frac{1}{2}$ .

## Generating a random prime $p$ with factorized $p - 1$

### Algorithm RPF

Input: a natural number  $m \geq 2$ , (or number of bits  $l$ )

Output: a random prime number  $p \leq m + 1$  (or  $l$ -bit  $p$ ), together with a factorization of  $p - 1$

- repeat
  - generate a random factorized number less than  $m$  (or  $l$ -bit) using the RFN algorithm, we get  $n, (p_1, \dots, p_s)$
- until  $IsPrime(n + 1)$
- $p \leftarrow n + 1$
- output  $p, (p_1, \dots, p_s)$  is the factorization of  $p - 1$

## Generating a random prime $p$ with factorized $p - 1$

### Analysis of the RPF algorithm

For probabilistic  $IsPrime(n)$  operating in expected time  $O(\tilde{\tau}(l))$ , which can return *true* for a composite number with probability at most  $\epsilon$ :

- The expected time is  $O(l^3 \tilde{\tau}(l))$ .
- Every prime  $p$  with correctly factorized  $p - 1$  is chosen with equal probability.
- The probability that the output  $p$  is not prime or  $p - 1$  is not correctly factorized is  $O(\epsilon l^2)$  for sufficiently small  $\epsilon$ , roughly  $\epsilon l \leq \frac{1}{2}$ .

## Probabilistic algorithms

### Literature

- Shoup: A Computational Introduction to Number Theory and Algebra. Chapter 9.
- For the basics of probability theory, see the same book, chapter 8, paragraphs 1-4 and 10.
- Theorems about primes can be found there in the chapter 5. <http://shoup.net/ntb/>