PROBABILISTIC POLYNOMIAL TIME ALGORITHMS & CRYPTOGRAPHY

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Problems and Algorithms

The problem of cryptography is how to prove security against an unknown adversary. The model of the adversary must therefore be a wide as possible, and then reasonable restrictions be placed on the adversary to have practical results. A pure theory of cryptography, based on information theory, shows that perfect encryption is possible, but the condition under which the result holds is impractical. What is believed about the adversary, is that it is a logical machine, a computer, that has a cost for its resources. The adversary is an algorithm. We need to learn to measure and categorize algorithms, and play games which match one algorithm against another, in very general ways.

Date: October 19, 2019. *Updated:* October 27, 2019.

Problems are also arranged by the size. In the case of sorting, it is both the size of the numbers to sort and the number of numbers to sort.

1.1. **Problem Instances.** Algorithms solve problems. In the theory we are developing, a problem is actually a family of problem instances. For instance, the problem of sorting has as an instance a specific list of n integers. If the problem family is finite, a table can list all problem-solution pairs. In the case of a very large but finite set of problem instances, an algorithm, implemented by a program that takes many fewer lines than the table, is a preferable way to encode the table. When the problem set is infinite however an algorithm is synonymous with solving the problem. It can produce any solution, on demand, from a small number of lines of code.

1.2. **Algorithms.** The study of algorithms is concerned with solving problems, and finding efficient solution to problems. The cost of a solution is the number of steps taken to produce the solution, and also the number of memory cells used during the run of the algorithm. The first measure is the time complexity of the algorithm, the second measure is the space complexity of the algorithm.

An algorithm is made up of a sequence of steps, each step considered reasonable for a unit of computational work. For instance, an addition of two integers of bounded size, or a decision to continue with the next step, or jump forward steps, based on the outcome of a short logical calculation. The steps can be arranged in blocks that are run repeatedly, so that the number of steps executed far exceed the number of lines written.

Algorithms also use data, which is abstracted as a potentially infinite series of labelled cells, each cell capable of being marked with a symbol from a finite set of memory marks. In practice — memory is an array of bytes, each byte containing an integer 0 through 255.

A good example for this discussion is pseudo-code description in the book *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein. The pseudo code is a sort of platform neutral, syntax permissive, programming language for the purpose of identifying and counting the number of steps taken during the run of the algorithm. While one can reference that book for more details, if one has programming experience, then it is generally correct to understand a step as a statement. The number of statements run can be larger than the number of statements written when loops or recursive calls cause the same statements to be run multiple times.

```
# see -  github.com/burtr/Workbook/crypto

def is_natural(N):
    """
    N is a finite string over 0,1
    """
    t1 = False
    t2 = False
    for n in N:
        if not t1:
            t1 = True
        else:
            t2 = True
        last_n =n
    if t1:
        return (not t2) or (last_n==1)
    # reject the empty string
    return False
```

Figure 1: Recognizing a natural number

1.3. **Formalization.** The theoretical apparatus for stating and reasoning about problems and algorithms, is to abstract a problem as a language, and the algorithm that solves the problem, as a device that recognizes the language.

Example: The language of addition of natural numbers $\mathcal{D}$, would be a subset of the space of triples of digit strings, in which the first element, is equal to the sum of the second and third elements,

$$\mathcal{D} = \{\, x, y, z \in \mathcal{N} \times \mathcal{N} \times \mathcal{N} \,|\, x \leq y+ \,\}.$$

We could further formalize the set of naturals $\mathcal{N}$ as any finite string of zero's and one's either ending with a 1 or is the length one string 0,

$$\mathcal{N} = \{\, 0 \,\} \cup \{\, (0|1)^* 1 \,\}.$$

Note that this is a "little endian" representation of the naturals, where the least significant digit is the first (leftmost) digit. So, 6 would be the string 011, not 110.

An algorithm to recognize the set $\mathcal{N}$ is given in Figure 1. An algorithm to recognize the set $\mathcal{D}$ is given in Figure 2.

Typically, an algorithm is thought of as something that computes a result, rather than which recognizes a result when given. If a result

```
# for addition_rules and carry_rules and
# full program see -
#     github.com/burtr/Workbook/crypto

def is_sum(D,E,F):
    """
    D, E and F are a finite strings over 0,1
    """
    if not all([is_natural(D),
                is_natural(E), is_natural(F)]):
        return False
    carry = False
    D, E, F = extend_longer(D,E,F)
    for d, e, f in zip( D, E, F):
        if (d,e,f,carry) not in addition_rules:
            return False
        else:
            carry = (e,f,carry) in carry_rules
    return True
```

Figure 2: Recognizing the addtion of natural numbers,

can be computed, say $z = f(x, y)$, then it can be recognized,

$$\hat{f}(z, x, y) = (z == f(x, y)).$$

The looseness this gives in the other direction, whether a recognizer can be used to calculate, will be used in discussing non-deterministic algorithms. Since the space of the answer is enumerable, if there is a stopping rule, at worse one can enumerate all possible answers until either the stopping rule applies or the recognizer accepts some result.

In cases such as over the naturals, where the search for the answer is on a totally ordered set, the recognizer might be slightly modified, for instance, recognizing $(z \leq x + y)$, and binary search for z used, rather than blind enumeration, see Figure 3.

1.4. **Asymptotic Efficiency.** It is not convenient, necessary, or desirable, to step count precisely. In a code such as Figure 4, the run time is

$$T(n) = T_A + nT_B + T_C$$

as a function of n, where T_A is the number of steps in the code block A, etc.. '

```
def do_sum(D,E,F,):
    """
    D is a recognizer for d <= e+f
    E and F are a finite strings over 0,1
    """
    if D(0,e,f):
        return True
   d = 1
   while True:
       if not D(d,e,f):
           d = concatenate(0,d)
   c = d
   d = remove_left(d)
   while d:
       if D(merge(c,d),e,f):
           c = merge(c,d)
   return c
```

Figure 3: Adding using recognition

Overestimating all these times and factoring them together, we get that the number of steps taken by the algorithm in Figure 4 is proportional to n, the number of items in array `a`. It maybe be possible to modify the code blocks to vary the exact number of steps, but what asymptotic efficiency focuses on is the growth of work as a function of n. For findmax it is linear.

The big-Oh notation summarizes this by saying the algorithm runs in time $O(n)$. The definition of $O(f)$ is a set of functions which are bounded above by f, allowing for the linear scale of the range and ignoring any finite behavior in the domain,

$$O(n) = \{\, f : \mathcal{N} \rightarrow \mathcal{R} \,|\, \exists N > 0, \exists c > 0, \forall n \geq N, c\,n \geq f(n) \geq 0 \,\}$$

If this notation is unfamiliar then it should be reviewed. It is of crucial importance in the definition of PPT algorithms.

2. Tractable Algorithms

In the movie *The Imitation Game*, there is an exciting race between Alan Turing's computations to break the Enigma and Commander Alastair Denniston wishing to shut down the Bombe that was doing the computation. Although a problem might be theoretically solvable by and algorithm, if the algorithm requires too much resource, either

```
find_max(a[], n):
   {
   // code block A
   i = 1
   biggest = a[0]
   }
   while (i++<n) {
       // code block B
       if (biggest < a[i])
           biggest = a[i]
   }
   {
   // code block C
    print(biggest)
   }
```

Figure 4: A simpleO(n) algorithm

memory space or computation steps, then the solution remains theoretical. Problems that require so much resource are called *intractable*. Our adversary will have to be realized, and therefore its algorithm must be *tractable*, meaning the time and space can be provided.

In code breaking one is up against a natively intractable problem, to find a needle in an exponentially large haystack. However, besides the depiction in the film being historically bogus, Turing and others were finding avenues of tractability that allowed Enigma messages to be read. This is what is means to break a cipher — to provide a tractable method to find that needle.

In this section we describe both deterministic and probabilistic polynomial time algorithms, and identify such algorithms with tractable computation.

2.1. **Polynomial Time Algorithms.** In practice, the distinction between linear algorithms, running in time $O(n)$, and (say) quadratic algorithms running in time $O(n^2)$ is significant. In the first case the algorithm runs as fast as the data can be read; in the second case, there is a definite increase in work as the problem size grows. For each factor of k increase in problem size, the number of steps in the solution increases by k^2. However, cryptography is indifferent to such refinement. Either we are resistant to an computational adversary or we are not. With this in mind we focus of algorithms with run time $O(n^k)$ for any

k,

$$P = \bigcup_{k>1} O(n^k)$$

Definition 2.1 (P time). An algorithm with efficiency in P is called a polynomial time (P-time) algorithm.

The focus on P-time algorithms is because,

(1) The class is robust. Changes in computational model, and the inability to precisely identify the complexity of the problem generally do not make problems enter or leave this class.
(2) Algorithms not in this class consume additional resource with almost exponentially increasing hunger with increasing n, they are only solutions in principle, not in fact, as sufficient resources cannot be supplied.

2.2. **Probabilistic Polynomial Time Algorithms.** So far all algorithms we are considering are *deterministic*. On a given problem instance, the steps taken are always the same on each run. They also have a uniform time bound — not a single problem instance is permitted to exceed the time allocated for a problem of that size.

However, we need a wider variety of algorithms at our disposal — algorithms that include the element of chance. Algorithms solving cryptographic puzzles will often make guesses and hope for the best. The effectiveness of an adversary will less be how quickly it works but how often it works. We need to include *non-deterministic algorithms* that can have various outcomes on any given problem instance. Variations include how often the algorithm succeeds or how often the given output is a correct answer.

For a probabilistic algorithm $\mathcal{A}$, along with the problem instance x, the algorithm is supplied with a sufficiently long random tape $\omega \in \Omega$. At points of decision, the algorithm can consult ω for the next random bit in sequence, and branch its computation according to the bit. Laid out this way, on a given x the computation is not linear sequence, but a branching tree, each path caused by differing bits in ω.

The time bound on $\mathcal{A}$ is uniform for all problems of size n, and for all computation paths ω. If the time bound is some polynomial $p(n)$, the ω is in fact chosen uniformly at random from the space $\Omega = \{0,1\}^{p(n)}$, and any event $S \subseteq \Omega$ has probability,

$$P[S] = |\, S \,| / 2^{p(n)}.$$

Different computation paths can lead to different results. In a decision problem, the possibilities are T, F, and possibly $\perp$ to signify either an error, the algorithm being halted for time, or no conclusive result. The algorithm is a sort of random variable,

$$\begin{array}{rccc} \mathcal{A}_*(x): & \Omega & \longrightarrow & \{\, T, F, \perp \,\} \\ & \omega & \longmapsto & \mathcal{A}_\omega(x) \end{array}$$

with probability the measure on Ω of those random strings leading to the result,

$$Pr[\,\mathcal{A}_\omega(x) == 1\,] = Pr[\,\{\,\omega \in \Omega \,|\, P_\omega(x) == 1\,\}\,].$$

Definition 2.2 (**PPT Algorithm**)**.** A *Probabilistic Polynomial Time Algorithm* is a P-time algorithm $\mathcal{A}_\omega(x)$; where the polynomial bound in input length n applies uniformly for all inputs x of length n and for all random tapes ω; and where $\omega \in \Omega$ is drawn independently, uniformly at random for each run of the algorithm; and with outcomes stated as a probability.

3. Adversarial Indistinguishability Games

3.1. **Oracle Machines.**

About schemes of interacting machines, and in particular, the oracle scheme, in which the main algorithm can seek answers by querying another machine, called *the oracle*. An oracle query counts only as one computational step, and the computational power of the oracle will be much greater than that given the main algorithm, or else contains secret information that is difficult to extract, else the algorithm could answer the query itself, without help of the oracle.

A typical notation for an oracle $\mathcal{E}$ provided to a PPT algorithm $\mathcal{A}$ is $\mathcal{A}^{\mathcal{E}}$.

The oracle scheme will be used in the next section, where we define a competition between an encryption algorithm, provided as an oracle $\mathcal{E}$, a PPT algorithm called *the Adversary* $\mathcal{A}$ that is mediated by an algorithm called *the Protocol*, Π,

$$Eav^{\mathcal{E}}_{\mathcal{A},\Pi}(n)$$

The notation is a bit intense.

(1) The n is essentially the number of bits in the encryption. The definition intends that the encryption be defined for arbitrary n, which will be a security parameter. The larger the n, the greater the security.
(2) The encryption is superscripted as an oracle, and it is the subject of the game, it is what is being tested for security.
(3) The adversary and protocol interact, and are placed in the subscript.

(4) The protocol is fixed and is part of the eavesdropping model. However the adversary can be any PPT algorithm.

3.2. **Eavesdropping Model of Security.** We have introduced the PPT algorithm to model what in reality will be our adversary. We now place that adversary against an encryption to see if it can win. The protocol models our beliefs about the sort of attack that will be undertaken.

Definition 3.1. Adversary-Protocol games: Adversarial Indistinguishability under an Eavesdropping model for encryption $\mathcal{E}$ is comprised of an encryption scheme,

$$\mathcal{E} = (Gen(n), Enc_k, Dec_k),$$

and an Adversary-Protocol complex $[\mathcal{A}, \Pi]$, that has the following interaction,

(1) A security parameter n is announced, $n \in \mathcal{N}$;
(2) Π chooses a bit $b \in_U \{0,1\}$, and chooses a key $k \leftarrow Gen(n)$, generated uniformly at random;
(3) Independently, $\mathcal{A}$ chooses two message, m_0 and m_1 of equal length and delivers them to Π;
(4) Π responds with $c = Enc_k(m_b)$;
(5) $\mathcal{A}$ decides and announces a bit $\tilde{b} \in \{0,1\}$.

The value of $Eav^{\mathcal{E}}_{\mathcal{A},\Pi}(n)$ is $(b == \tilde{b})$.

Definition 3.2. Adversarial Indistinguishability: An encryption scheme $\mathcal{E}$ has *Adversarial Indistinguishability* if, for any PPT adversary $\mathcal{A}$,

$$Pr[\, Eav^{\mathcal{E}}_{\mathcal{A},\Pi}(n)\,] \leq 1/2 + g(n)$$

for a negligible function $g(n)$.

Because the coin b is fair, any adversary can achieve success with probability 1/2 by guessing randomly, or even by guessing always 0 or 1. Hence there can be no weakness in $\mathcal{E}$ as there is no reference to $\mathcal{E}$.

Also, a small advantage is allowed. For instance, the advantage that the adversary achieves by a *brute force attack* shown in Figure 5.

3.3. **Perfect Security.** The one time pad, or Vernon cipher is an example of a secure cipher in the eavesdropper model. The security of the cipher is perfect, the negligible function can be $g(n) = 0$,

The cipher works on a plaintext of 0's and 1's of length m, $\langle p_i \rangle$ by creating m random bits $\langle r_i \rangle$ exclusively or'ing plaintext and the random bits,

$$\langle c_i \rangle = \langle p_i \oplus r_i \rangle.$$

```
    def brute_force_attack(m0, m1, c):
        k0 = k1 = None
        for k in Keys:
            if c = E(k,m0):
                k0 = k
            if c = E(k,m1):
                k1 = k
        if k0 == None:
            return 1
        if k1 == None:
              return 0
        return None
```

Figure 5: A brute force attack for the adversary

The random string is then provided as the key to decode,

$$\langle c_i \oplus r_i \rangle = \langle p_i \oplus r_i \oplus r_i \rangle = \langle p_i \rangle.$$

The security is perfect but with the disadvantage that the key is large and it can only be used once.

That the security is perfect can be seen from this argument. Suppose rather than encrypt the plaintext, the cipher ignored the plaintext and just set the ciphertext to a completely random string. Then the key is calculated by the encipherer as,

$$\langle r_i \rangle = \langle p_i \oplus c_i \rangle.$$

All that has happened is that the key and the ciphertext have changed places in the order they are created. However, there is no way exterior to the encipherment that this can be known.

That is to say, sending the ciphertext of a Vernon cipher is equivalent to sending a random string of bits, without any reference to the plaintext.

3.4. **CPA and CCA Models of Security.** The model of Eavesdropper security is too weak. Even the Vernon cipher can be broken by a chosen plaintext attack, even though this cipher is Eavesdropping secure. The attacker sends a bit string of zero's to be encrypted, and the encryption returns the key. True, this key should never be reused, but if somehow it does get reused, then the attacker can immediately decrypt the message.

A model that resists chosen plaintext attacks, CPA, is also preferred over EAV because it is safe for multiple messages.

The game for $CPA^{\mathcal{E}}_{\mathcal{A},\Pi}$ is like for the eavesdropper, except that the adversary is given oracle access to the encryption. The adversary is in fact $\mathcal{A}^{\mathcal{E}}$. The protocol Π draws a key at random, loads it into the encryption oracle, and the adversary can then run whatever test encryption it wishes.

While CPA security is a very good model, and one that is in use, it still has a weakness when confronted with a chosen ciphertext attack. In these attacks, we will allow the adversary to have oracle access to both an encryption oracle and a decryption oracle.

The game for $CCA^{\mathcal{E}}_{\mathcal{A},\Pi}$ is like the CPA game except the adversary is actually $\mathcal{A}^{\mathcal{E},\mathcal{D}}$. A rule has to be made, however, that the adversary cannot query the decryption oracle for the challenge ciphertext.

We described a CPA secure cipher that generates a random sequence $\langle r_i \rangle$ and encrypts,

$$\langle c_i \rangle = \langle p_i \oplus r_i \rangle.$$

The CCA attacker can generate and sequence $\langle q_i \rangle$ and ask for a decryption of $\langle c_i \oplus q_i \rangle$, which the protocol will permit, and the attacker receives back $\langle p_i \oplus q_i \rangle$. $\mathcal{A}$ then reapplies the q_i sequence and now has recovered the plaintext.

4. References

Introduction to Modern Cryptography, by Jonathan Katz and Yehuda Lindell. First edition was 2007.

Introduction to Algorithms, Cormen, Leiserson, Rivest and Stein.

Probabilistic Encryption, S. Goldwasser and S. Micali, J. of Computer and System Sciences, Vol. 28, April 1984, pp 270–299.

A Concrete Security Treatement of Symmetric Encryption, M. Bellare, A. Desai, E. Jokipii, P. Rogaway. Extended abstract in the Proceedings of the 38th Symposium of Foundations of Computer Science, IEEE, 1997. Full paper September 2000.