Holden Rohrer, Holden Watson, and Nithya Jayakumar
2019 Oct 21
Applied Combinatorics–Math 3012
Livshyts

Part 3: Final Report

Topic: Distributed Systems Integrity and Correctness

1 Distributed Networks' History

Distributed networking describes a number of protocols, systems, and technologies. The most popular amongst them is "the internet," to the extent it can be described as a single entity. It originates in very centralized institutions: MIT and DARPA, who invented packet switching, a way of transferring data across a group of nodes [1][2]. Packet switching was the birth of the internet, and as such is a central theme of our project.

The way that packets (any information transferred across any protocol) maintain their correctness, or proving that the data is untampered, and the way that computers connect to eachother will be explored in this paper. Additionally, we have included some Python programs as proofs-of-concept for some concepts discussed, such as RSA and routing. Execution instructions for those and source code has been included in the appendix.

2 The RSA Algorithm

In determining correctness, a major concern is determining that the message hasn't been tampered with by an intelligent intermediate. Public key cryptography tries to answer this problem by providing proof of authorship and, as an extension of "normal" encryption, preventing interception. RSA (Rivest-Shamir-Adleman, named after its MIT faculty creators) is one such algorithm. It works by providing a set of public keys to all parties, and corresponding secret private keys.

One of the simpler algorithms, it applies the NP-hard nature of factorizing a semiprime, Eulers theorem, and the Euclidean Algorithm to encrypt communication. Because it is simple to devise, it has been included as a sample, in the form of a Python script which encrypts and decrypts messages, given a small RSA key (compared to those used in real applications). There are several optimizations (such as applying the Chinese Remainder Theorem) which can be used, but none have been applied to maintain the code's simplicity.

2.1 Methodology

The encryption process begins with the selection of two large primes, p and q, their product $n = pq$, and a fourth number e relatively prime to $\phi(n)$. n is public, whereas p and q are secret.[3]
Encryption is accomplished through the following three steps:

1. Convert message to a number (like `a` becomes 1 and `ab` becomes 130, assuming a 128-character language)
2. Break the converted message into blocks of size less than n.
3. For each block B, an encrypted block C is created such that

$$C \equiv B^e \,(mod\, n)$$

. To decrypt that message:

1. Calculate an integer d such that $de \equiv 1 \,(mod\, \phi(n))$ using the Euclidean algorithm. Note that $\phi(n)$ is the totient function, or the number of non-coprime integers with n less than n.
2. Convert back using $B \equiv C^d \,(mod\, n)$.

The decryption process described above makes use of Eulers theorem. Some decryption algorithms make use of other mathematical theorems of relation, including the Chinese Remainder Theorem.

[1] *https://networkencyclopedia.com/packet-switching/*
[2] *https://www.internetsociety.org/internet/history-internet/brief-history-internet/*
[3] *https://primes.utm.edu/glossary/page.php?sort=RSA*

The RSA Algorithm, while nearly unbreakable, isnt as untouchable as originally thought, shown by the example number $n = pq$ that Rivest, Shamir, and Adleman published as a challenge in 77 was broken in 94. This proves that as computing power grows, the best cryptographers can do is increase the size of the secrets to make prime factorization as difficult as possible, or its analogue in more arcane algorithms.

2.2 The Code

The code for RSA encryption and decryption can be found in this folder at `rsa-encrypt.py` and `rsa-decrypt.py`. `rsa-encrypt` relies completely on user input, allowing the user to input a semiprime of arbitrary size (larger is more secure) and a value e which must be coprime with one less both divisors of the semiprime ($p-1$ and $q-1$). However, other than basic input and type conversion (string to list of integers to list of integers, for example), the "heavy-lifting" it does is very limited.

```
def decrypt_block(blk):
    return blk**d % n
```

defines the majority of it, specifically the application of Euler's theorem.

Similarly, decryption relies on the basic principle of Euler's theorem to develop the decryption value d (and the fact that that value can exist). While efficiency was not absolutely necessary, it could be improved by using a speedier (Euclidean algorithm-based) decision algorithm for d than simply checking all values. This was neglected to focus on the real interesting component of RSA. Once that value d is available, the decryption can be known easily In this case,

```
def encrypt_block(blk):
    return (blk ** e) % n
```

defines the heavy lifting of "undoing" the RSA encryption, and shows how RSA shines in its simplicity—in stark contrast with its convoluted comrades.

3 Algorithms for Graph Connectedness and Bridges

A graph is connected if there exists a path between any pair of vertices. Beearliest is a simple algorithm that uses a depth-first search to determine whether a given graph is connected, using Python3 and the NetworkX library.

```
import networkx as nx
def is_graph_connected(G):
    VISITED = []
    def dfs_connectedness(v):
        nonlocal VISITED
        for node in G.adj[v].keys():
            if node not in VISITED:
                VISITED.append(node)
                dfs_connectedness(node)
        return len(VISITED) == len(G.nodes)
    return dfs_connectedness(0)
```

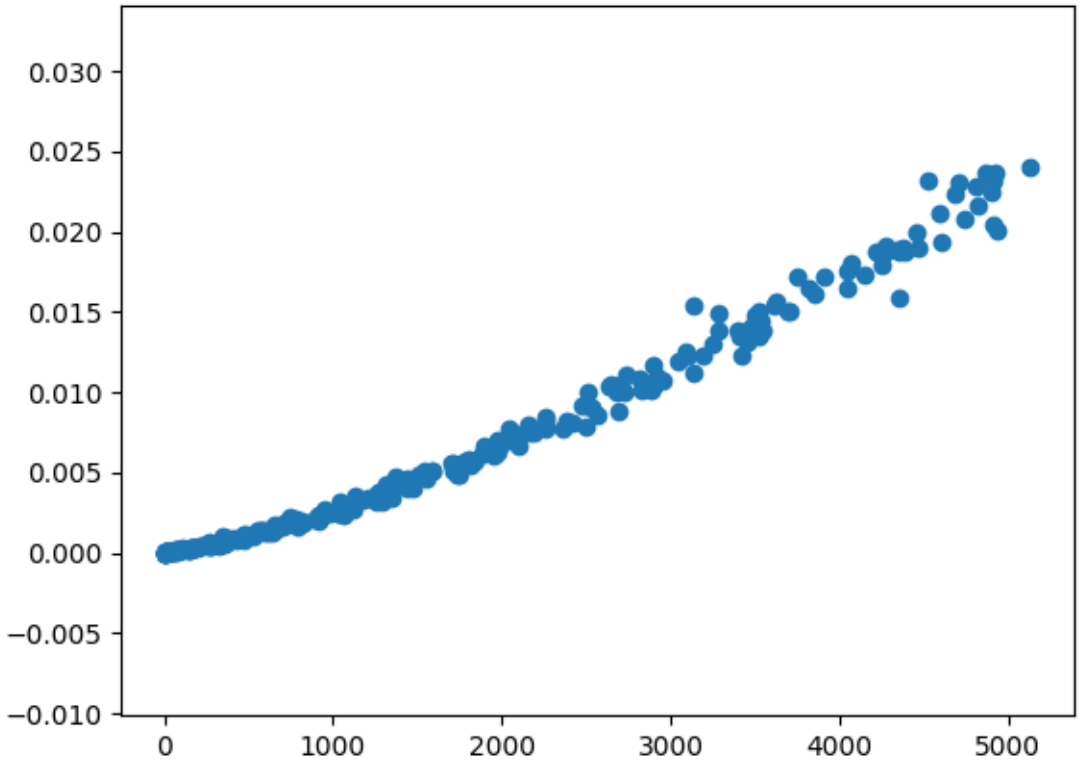
This algorithm compiles a list of the visited nodes of the graph G, and then determines if that list contains every node after the search is complete. The search can start at an arbitrary node (here, it chooses the node labelled 0). The algorithm has a time complexity of $O(V + E)$, where a given graph G has V vertices and E edges.

This same algorithm can be used to create an algorithm of complexity $O(E \cdot (V + E))$ to find all bridges of a connected graph (i.e. all edges which, when removed, would result in the graph becoming unconnected). The algorithm simply iterates through each edge of the original graph, removing it and then determining its connectedness using the method above. This quadratic algorithm is, however, not the most efficient algorithm for locating bridges. An $O(V + E)$ algorithm can be developed as shown below:

```
def bridges_2(G):
    BRIDGES = []
    tick = 0
    def bridges_2_recursion(v):
        nonlocal BRIDGES, VISITED, tick, discoveries, earliest, parents
```

```
            VISITED.append(v)
            discoveries[v] = tick
            earliest[v] = tick
            tick += 1
            for v2 in G.adj[v]:
                if v2 not in VISITED:
                    parents[v2] = v
                    bridges_2_recursion(v2)
                    earliest[v] = min(earliest[v], earliest[v2])
                    if earliest[v2] > discoveries[v]:
                        BRIDGES.append((v, v2))
                elif parents[v] != v2:
                    earliest[v] = min(earliest[v], discoveries[v2])
        n = len(G.nodes)
        VISITED = []
        discoveries = [float("Inf")] * n
        earliest = [float("Inf")] * n
        parents = [-1] * n
        for v in range(n):
            if v not in VISITED:
                bridges_2_recursion(v)
        return BRIDGES
```

The diagram shown below plots the time taken to calculate the number of bridges on the y-axis, with the value of V+E on the x-axis.



4 Appendix

These programs, need to be run with Python3, so install that as suggested by *https://www.python.org/downloads/*. The package manager `pip` is also necessary for installation of third party graphics libraries such as `NetworkX`. Install that as described here: *https://pip.pypa.io/en/stable/installing/*.

Now that those tools are available, run the following shell commands to install relevant libraries:

```
$ pip install networkx
$ pip install numpy
```

Each file should be executable with "`python3 $filename`", preferably in its local directory. The files can be obtained from the internet with "`git clone https://github.com/feynmansfedora/appcomb-proj.git`", assuming `git` is installed. If it is not, it can be cloned from *https://github.com/feynmansfedora/appcomb-proj* directly.