Quiz 4
(Deadline April 18, 2025)

Problem 1

Description (3 pts)

Public Key Infrastructure (PKI) is widely used in various aspects of the world. One of its most important functions is authorization management. Therefore, understanding how to create certificates and manage the certification chain is crucial. Now, your task is to assist our NYCU-CE (Cryptography Engineering) company in generating several certificates.

Requirements

1. **Certificate Generation:**
   - Go to this repository github.com/Alonza0314/cert-go
   - Download the `cert-go` executable file from github.com/Alonza0314/cert-go/releases
   - Read the description github.com/Alonza0314/cert-go/blob/main/cmd/README.md
   - Use this tool to generate the following four certificates, ensuring that the organization field is set to "NYCU-CE":
     (a) root certificate
     (b) intermediate certificate
     (c) server certificate
     (d) client certificate
   - (Hint keywords: private key, csr, cert)
2. **Description of Commands:**
   - Describe the commands you used in the previous section and list them in the order they were executed, using numeric order.
3. **Certificate Chain Description:**
   - Describe your certificate chain structure.
4. **(Extra) Repository Improvement:**
   - Check for any areas of improvement in the `cert-go` repository.
   - If you find any, try to create a pull request.

Evaluation

- **Certificate Generation: 2 points**
  - After generating the certificates, there will be additional files (private keys, CSRs). Please include all the generated files in your submitted ZIP file, and describe what files are generated.
  - (You're not required to include the screenshots in PDF files)
- **Description of Commands: 0.5 point**
  - Provide a complete description in correct order.
- **Certificate Chain Description: 0.5 point**
  - (Both in graph and just describe in text are accepted)
- **(Extra) Repository Improvement:**
  - Propose improvements: **0.5 point**
  - Create a pull request and have it accepted (merged) by the repo owner: **1 point**

Problem 2

Description (5 pts)

RC4's vulnerability mainly arises from its inadequate randomization of inputs, particularly the initialization vector (IV) and key integration, due to its reliance on the initial setup by its Key Scheduling Algorithm (KSA). The cipher operates through two phases: KSA, which shuffles a 256-byte state vector based on the key to ensure dependency and randomization, and the Pseudo-Random Generation Algorithm (PRGA), where it further manipulates this state to produce a seemingly random output stream.

To help you understand the importance of randomization algorithms, here we provide the pseudocode for two slightly different shuffle algorithms.

Naïve algorithm:

```
For i from 0 to length(cards)-1
    Generate a random number n between 0 and length(cards)-1
    Swap the elements at indices i and n
EndFor
```

Fisher–Yates shuffle (Knuth shuffle):

```
For i from length(cards)-1 down to 1
    Generate a random number n between 0 and i
    Swap the elements at indices i and n
EndFor
```

Requirements

1. **Simulation Implementation:**
   - Write a Go/Python program to simulate two algorithms with a set of 4 cards, shuffling each a million times.
   - Collect the count of all combinations and output.
   - We focus on "distribution", so the card sets' order does not need to be in lexicographical order and is accepted as is.

   For example:

```
$ go run main.go // or python3 main.py
Naive algorithm:
  [1 2 3 4]: 41633
  [1 2 4 3]: 41234
  ... and so on
Fisher-Yates shuffle:
  [1 2 3 4]: 41234
  [1 2 4 3]: 41555
  ... and so on
```

2. **Algorithm Comparison:**
   - Based on your analysis, which one is better, and why?

3. **Drawback Analysis:**
    - What are the drawbacks of the other algorithm?
4. **RC4 Improvement:**
    - After learning these two shuffling algorithms, please propose improvement methods for "RC4."
    - (Note: This section does not cover the Naïve or Fisher-Yates (FY) algorithms mentioned above. Instead, it focuses on discussing "RC4," specifically the "KSA" and "PRGA".)

Evaluation

- **Simulation Implementation: 2 points**
    - Successfully print the output with reasonable distribution.
        - (Please provide the screenshots of your code and output in your PDF file)
        - (Please include your code in the submitted ZIP file)
- **Algorithm Comparison: 1 point**
    - Provide a complete description.
- **Drawback Analysis: 1 point**
    - Provide a complete description.
- **RC4 Improvement: 1 point**
    - Provide a complete and reasonable improvement method.

Problem 3

Description (2 pts)

The Miller-Rabin test is an algorithm based on probability that is employed to ascertain the primality of a given number. It operates by selecting random bases multiple times and examining if these bases offer substantial indications that the number is not prime.

The procedure consists of the following steps:

1. Given a number n, find an integer s and an odd number q such that $n - 1 = 2^s q$.
2. Choose a random number a from the range $[1, n - 1]$.
3. Compute $a^q \mod n$. If the result is 1 or $n - 1$, then n passes.
4. For i from 0 to $s - 1$, compute $a^{2^i q} \mod n$. If one of these is $n - 1$, n again passes.
5. If none of the above conditions are met, n is composite.

It is typical to carry out trial divisions by small primes before conducting the Miller-Rabin test in order to promptly identify obvious composites.

Requirements

1. **Miller-Rabin on RSA Moduli:**
   - What happens when we apply the Miller-Rabin test to numbers in the format pq, where p and q are large prime numbers?
2. **RSA Security Analysis:**
   - Can we break RSA with it? Please provide a reasonable explanation.

Evaluation

- **Miller-Rabin on RSA Moduli: 1 point**
  - Provide a complete explanation of what happens.
- **RSA Security Analysis: 1 point**
  - Give a correct and reasonable explanation.

File Structure

There are **ONE** zip file you need to upload, and it should be structured as follows:

- `<student_id>.zip`
  - `<student_id>.pdf`
  - `problem1/`
    - `client/`
      - `client.cert.pem`
      - (required additional pems...)
    - `intermediate/`
      - `intermediate.cert.pem`
      - (required additional pems...)
    - `root/`
      - `root.cert.pem`
      - (required additional pems...)
    - `server/`
      - `server.cert.pem`
      - (required additional pems...)
  - `problem2/`
    - `main.go` or `main.py`
    - (additional codes...)

(You may include any additional code files, as long as they follow this file structure.)
(**Incorrect file structure or incorrect file names will result in a 2 points deduction.**)

Grading Policy

1. This quiz comprises three problems:
   - Problem 1: 3 points + 1.5 points(extra)
   - Problem 2: 5 points
   - Problem 3: 2 points
2. **Late Submission Penalty:** A penalty of **0.5 points per day** will be applied for late submissions, up to a maximum of 20 days. Beyond 20 days, late submissions will be assigned zero.
3. All quizzes are **mandatory**. Failure to submit any quiz will result in an automatic failing grade for the course.