CS/ECE407 Cryptography – Homework 1
Perfect Secrecy and Pseudorandomness

Due: Thursday, February 20, 10:30am CT

Please get started on the programming portion of this assignment early, to ensure you understand how to properly build/run the code.

Remember, you are free to collaborate with up to one classmate. See the course webpage for more details. You are expected to write out and submit your own solutions! Your collaboration is for discussing problems at a high level, not plagiarizing answers.

The non-coding portion of your homework should be typed or carefully handwritten. We provide a LaTeX template for this document, if you would like to use it as a starting point. **If we cannot read your handwritten answers, they will not receive credit.**

The coding portion of the homework should be submitted as a single C++ file named `hw1.cpp`. Simply fill in the provided function definitions. You should not need to `#include` additional files.

Typed solutions and C++ code should be submitted through Gradescope (see course webpage). Hand-written solutions should be scanned and turned in through Gradescope.

Problem 1 (Breaking traditional schemes)**.**

1. **(2 points)** The following ciphertext was created using a Caesar Cipher, which is a specialization of a substitution cipher:

   RDCVGPIJAPIXDCHNDJPGTPEGDETGRGNEIPCPANHICDL

   What does the corresponding plaintext say? Hint: *The cipher operates only on the 26 capital letters of the English alphabet.*

Definition 1 (Perfect Secrecy)**.** Let Enc, Dec be a cipher with message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$. The cipher achieves **perfect secrecy** if for every pair of messages $m_0, m_1 \in \mathcal{M}$ and for every ciphertext $c \in \mathcal{C}$, the following holds:

$$\Pr_{k \leftarrow_{\$} \mathcal{K}}[\ E(k, m_0) = c\ ] = \Pr_{k \leftarrow_{\$} \mathcal{K}}[\ E(k, m_1) = c\ ]$$

Problem 2 (Perfect Secrecy, Part 1)**.** Alice and Bob need to communicate confidentially, and they decide to use a one-time-pad-based cipher. Alice and Bob understand that to safely encrypt this message and to achieve perfect secrecy, they need a key with (at least) n uniformly random bits, and they decide to flip a fair coin n times to choose these bits. However, they fear that all n coin flips might come up tails! If this happens, their key will be the all zero key, and so when they use it as a one-time-pad to encrypt their message m, they will be sending m in the clear!

Thus, Alice and Bob agree ahead of time on a contingency plan: if they happen to flip n tails, they will start over and flip the coin n more times. They will repeat this until they get some key that is not all zeros.

1. **(2 points)** Do Alice and Bob need to adopt this contigency plan? Why/why not?
2. **(2 points)** Is it secure to use this contingency plan? Does the answer depend on n?

Problem 3 (Perfect Secrecy, Part 2)**.** Recall the definition of perfect secrecy (Definition 1). Let's construct a cipher with integer message space $\{0, 1, ..., 7\}$ and integer key space $\{0, 1, ..., 7\}$. Here is a proposed cipher:

```
Enc(k, x):
  return $x + k$
```

```
Dec(k, c):
  return $c - k$
```

Here, '+' denotes integer addition.

1. **(2 points**) Is the scheme **correct**? If so, prove it; if not, show a key/message pair that does not correctly decrypt.
2. **(3 points**) This scheme does not satisfy perfect secrecy. Prove it.
3. **(2 points**) Propose a simple modification to the scheme such that it satisfies perfect secrecy.

Definition 2 (Negligible)**.** Let $\mu : \mathbb{N} \to \mathbb{R}$ denote a function. We say that μ is **negligible** if for every positive polynomial f, there exists some natural number x_0 such that for all $x > x_0$:

$$|\mu(x)| < \frac{1}{f(x)}$$

Problem 4 (Negligible Functions)**.** Recall the definition of a negligible function (Definition 2).

1. **(2 points)** Which of the following are negligible (in x)?
   (a) $f_0(x) = 1/x^{100}$
   (b) $f_1(x) = 1/2^x$
   (c) $f_2(x) = \sin(x)$
   (d) $f_3(x) = 1/n!$
   (e) $f_4(x) = 1/(n^{\log n})$

2. **(3 points)** One reason cryptographers like to work with negligible functions is that they are nicely *composable*. Let $\mu_0(x)$ and $\mu_1(x)$ be two negligible functions. Prove that $\nu(x) = \mu_0(x) + \mu_1(x)$ is also negligible.

Definition 3 (Pseudorandom Generator)**.** Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a polytime function. G is a **length doubling pseudorandom generator** if the following indistinguishability holds:

```
Real() :                      Ideal() :
  s ←$ {0,1}^λ        ≈c        x ←$ {0,1}^{2λ}
  return G(s)                   return x
```

$$\boxed{\begin{array}{l} \mathit{Real}() : \\ \quad s \leftarrow_{\$} \{0,1\}^\lambda \\ \quad \mathsf{return}\ G(s) \end{array}} \overset{c}{\approx} \boxed{\begin{array}{l} \mathit{Ideal}() : \\ \quad x \leftarrow_{\$} \{0,1\}^{2\lambda} \\ \quad \mathsf{return}\ x \end{array}}$$

Definition 4 (Pseudorandom Function Family)**.** Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a polytime function.[1] F is a **pseudorandom function family** (PRF) if the following indistinguishability holds:

$$\boxed{\begin{array}{l} k \leftarrow_{\$} \{0,1\}^\lambda \\ \mathit{Query}(x : \{0,1\}^\lambda) : \\ \quad \mathsf{return}\ F(k,x) \end{array}} \overset{c}{\approx} \boxed{\begin{array}{l} D \leftarrow \mathit{EmptyDictionary} \\ \mathit{Query}(x : \{0,1\}^\lambda) : \\ \quad \mathsf{if}\ x \notin D : \\ \qquad D[x] \leftarrow_{\$} \{0,1\}^\lambda \\ \quad \mathsf{return}\ D[x] \end{array}}$$

Problem 5 (PRF to PRG)**.** In class, we saw the definition of a PRG and of a PRF; see Definitions 3 and 4.

1. **(2 points)** Assume you have a PRF F. Construct a PRG G. *Hint. Your pseudocode should have the signature of a PRG:*

```
G(x : {0,1}^λ) :
  // your code goes here
  return ...
```

2. **(3 points)** Prove that your G satisfies the security requirement of Definition 3 by reducing PRG security of G to PRF security of F (Definition 4).

[1] For simplicity, we define PRF F where the key length, the input length, and the output length are each λ bits. In general, a PRF could have different values for each of these three lengths.

Problem 6 (Attacking "PRFs" and "PRGs"). Our cryptographic definitions of PRFs and PRGs insist that there exists *no* polytime program that can distinguish two particular programs. In this problem, you will show that certain programs are *not* PRFs/PRGs. To do this, you will write polytime programs (in C++) that win a distinguishing game.

Notation:

- Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a length-doubling PRG.
- Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a PRF.
- Let *halves* denote a helper program that splits a string at its middle.
- Let $x \mid y$ denote the concatenation of strings x and y.
- Let $x \oplus y$ denote the bitwise XOR of strings x and y.

Problems:

1. **(2 points)** Break the following "PRG":

```
$bad(s : \{0,1\}^\lambda)$ :
  $w, x \leftarrow halves(G(s))$
  $y, z \leftarrow halves(G(x))$
  return $w \mid x \mid y \mid z$
```

2. **(2 points)** One might attempt to fix the above PRG by mixing some bits together. Break the following "PRG":

```
$bad(s : \{0,1\}^\lambda)$ :
  $w, x \leftarrow halves(G(s))$
  $y, z \leftarrow halves(G(x))$
  return $w \mid (w \oplus x) \mid y \mid z$
```

3. **(3 points)** Break the following "PRF" which takes 2λ bits, splits them in half, runs F on each half, then bitwise XORs the result:

```
$bad(k : \{0,1\}^\lambda, x : \{0,1\}^{2\lambda})$ :
  $x_0, x_1 \leftarrow halves(x)$
  return $F(k, x_0) \oplus F(k, x_1)$
```

4. **(3 points)** Break the following "PRF" which takes λ bits and computes:

```
$bad(k : \{0,1\}^\lambda, x : \{0,1\}^\lambda)$ :
  return $F(k, x) \mid F(k, F(k, x))$
```

Instructions: The following files are included as part of the assignment:

- `hw1.cpp`: *This is the only file you will edit and submit.* Fill in the empty procedure definitions and win each game.
- `hw1.h`: This file sets up the games for each of the above problems. We recommend reading this code.
- `main.cpp`: This file includes the `main` procedure that we will run to grade your code. Feel free to run it yourself to check your solutions work! For each game, if you win at least 200 out of 256 times, you will get full credit.
- `util.h`, `util.cpp`, `aes.h`, and `aes.cpp`: These files include helper code needed to set up the problems. If you are struggling, we recommend reading the documentation in `util.h`. The most important detail of `util.h` is its `bitstring` class. Useful utility functions include:
  - `halves` splits a bitstring in half.
  - `|` concatenates two bitstrings into one bitstring.
  - `^` XORs two bitstrings.
  - `==`, `!=` compare bitstrings for equality/inequality.
  - `random` takes as input an integer `n` and outputs a randomly sampled bitstring of length `n`. E.g., the following snippet computes a uniform length 128 bitstring:

    ```
    bitstring x = random(128);
    ```

  There is no need to read `aes.h` or `aes.cpp`, unless you are just curious.
- `Makefile`: Use `make` to build your code. You should be able to use a terminal to build and run. You should see something like the following:

```
> make
g++ -std=c++20 main.cpp hw1.cpp util.cpp aes.cpp -o hw1
> ./hw1
Game 1:  You won 132 out of 256 rounds.
You're not there yet, keep trying!
Game 2:  You won 121 out of 256 rounds.
You're not there yet, keep trying!
Game 3:  You won 128 out of 256 rounds.
You're not there yet, keep trying!
Game 4:  You won 124 out of 256 rounds.
You're not there yet, keep trying!
```

Note that in this problem, the security parameter λ is set to `128`.

Feedback: Feel free to leave feedback with respect to this homework and the course! Did you find the homework too easy/too hard/just right? How is the pace of the course so far? Please add any feedback that would help improve the course.