CS/ECE407 Cryptography – Homework 2
PRGs, PRFs, and Computational Security

Due: Thursday, February 19, 3:30pm CT

Remember, you may 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 `hw2.cpp`. Simply fill in the provided function definitions. Do not `#include` additional files.

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

Definition 1 (Indistinguishability)**.** Let X and Y denote two probability ensembles over strings. Let $\mathcal{A}$ denote an arbitrary probabilistic program that outputs either 0 or 1. We denote $\mathcal{A}$'s **advantage** in distinguishing X from Y by the following function of λ:

$$Advantage_{\mathcal{A}}(\lambda) = \left| \Pr\left[b = 1 \;\middle|\; \begin{array}{l} x \leftarrow X_\lambda \\ b \leftarrow \mathcal{A}(1^\lambda, x) \end{array} \right] - \Pr\left[b = 1 \;\middle|\; \begin{array}{l} y \leftarrow Y_\lambda \\ b \leftarrow \mathcal{A}(1^\lambda, y) \end{array} \right] \right|$$

If X and Y are probability ensembles over functions, we instead write the following:

$$Advantage_{\mathcal{A}}(\lambda) = \left| \Pr\left[b = 1 \;\middle|\; \begin{array}{l} x \leftarrow X_\lambda \\ b \leftarrow \mathcal{A}^x(1^\lambda) \end{array} \right] - \Pr\left[b = 1 \;\middle|\; \begin{array}{l} y \leftarrow Y_\lambda \\ b \leftarrow \mathcal{A}^y(1^\lambda) \end{array} \right] \right|$$

Here, $\mathcal{A}^x$ denotes giving $\mathcal{A}$ oracle (i.e., black-box) access to the function x. We say that ensembles X and Y are **indistinguishable**, written $X \approx Y$, if for any polynomial-time $\mathcal{A}$, the following holds:

$$Advantage_{\mathcal{A}} \text{ is a negligible function}$$

Definition 2 ((Length-Doubling) Pseudorandom Generator)**.** Let G be a polytime-computable function that on input a λ-bit string outputs a 2λ-bit string. G is a **length doubling pseudorandom generator** if the following probability ensembles are indistinguishable (in λ):

$$\left\{ G(s) \;\middle|\; s \leftarrow \{0,1\}^\lambda \right\} \approx \left\{ r \;\middle|\; r \leftarrow \{0,1\}^{2\lambda} \right\}$$

Definition 3 (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:

$$\left\{ F(k, \cdot) \;\middle|\; k \leftarrow \{0,1\}^\lambda \right\} \approx \left\{ f \;\middle|\; f \text{ is a uniform function from } \{0,1\}^\lambda \text{ to } \{0,1\}^\lambda \right\}$$

Problem 1 (Transitivity of Indistinguishability)**.** Consider three probability ensembles X, Y, Z. Suppose that $X \approx Y$ and $Y \approx Z$.

- **(3 points)** Use Definition 1 to prove that $X \approx Z$.
- **(2 points)** We can consider a version of indistinguishability that makes the adversary's advantage more explicit. In particular, we can define the following notation:

  $$A \approx_\alpha B$$

  This means that any polytime adversary $\mathcal{A}$ trying to distinguish ensembles A and B has advantage bounded by some function α:

  $$Advantage_{\mathcal{A}}(\lambda) \leq \alpha(\lambda) \qquad\qquad \text{for all } \lambda$$

  Suppose that $X \approx_\alpha Y$ and $Y \approx_\beta Z$. Prove that $X \approx_\gamma Z$, where $\gamma(\lambda) = \alpha(\lambda) + \beta(\lambda)$.

[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 2 (PRF to PRG)**.** In class, we saw the definition of a PRG and of a PRF; see Definitions 2 and 3.

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

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

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

Problem 3 (Bit Flipping)**.** For a string $x \in \{0,1\}^n$, let $\bar{x}$ denote the bitwise negation of x. For example, if $x = 0100$, then $\bar{x} = 1011$. Let G denote a length-doubling secure PRG.

1. **(2 points)** Let G' be a length-doubling function defined as follows:

$$G'(x) = \overline{G(x)}$$

   That is, G' calls G on x, then bitwise negates the output. Is G' a PRG? If not, prove it is not by constructing a distinguisher. If it is, prove it is by demonstrating a security reduction.

2. **(2 points)** Let G' be a length-quadrupling function defined as follows:

$$G'(x) = G(x) \mid \overline{G(x)}$$

   That is, G' calls G twice, negates the output of the second call, and returns the concatenation of the two resulting strings. Is G' a PRG? If not, prove it is not by constructing a distinguisher. If it is, prove it is by demonstrating a security reduction.

Problem 4 (Attacking "PRFs" and "PRGs")**.** Remember, 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.

Coding 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:

- `hw2.cpp`: **This is the only file you will edit and submit.** Fill in the empty procedure definitions and win each game.
- `hw2.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 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 hw2.cpp util.cpp aes.cpp -o hw2
> ./hw2
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.