

Cryptography: HW2

Trevor Bramwell

Professor Mike Rosulek
January 23, 2015

1 Negligible Functions

Unless otherwise stated, all the following functions are negligible.

$$\frac{1}{2^{n/2}} = \frac{1}{2^{1/2}} * \frac{1}{2^n}$$

We know $\frac{1}{2^n}$ is negligible. Since this takes the form of $p(x)f(x)$ where $p(x)$ is a polynomial and $f(x)$ is negligible, $p(x)f(x)$ is negligible as well.

$$\frac{1}{2^{\log(n^2)}} < \frac{1}{2^{\log(n)}} < \frac{1}{n^c}$$

Because $(\log n)$ grows at a quicker rate than the constant c , $\frac{1}{2^{\log(n)}}$ will asymptotically approach zero faster than $\frac{1}{n^c}$.

$$\frac{1}{n^{\log(n)}} < \frac{1}{n^c}$$

$$\frac{1}{2^{(\log n)^2}} < \frac{1}{2^{(\log n)}} < \frac{1}{n^c}$$

$$\frac{1}{(\log n)^2} > \frac{1}{\log n} > \frac{1}{n^c} - \text{Not Negligible.}$$

$$\left(\frac{1}{n^{1/n}} = \frac{1}{n^{-n}} = n^n \right) > \frac{1}{n^c} - \text{Not Negligible.}$$

$$\left(\frac{1}{\sqrt{n}} = \frac{1}{n^{1/2}} \right) > \frac{1}{n^c} - \text{Not Negligible.}$$

$$\frac{1}{2^{\sqrt{n}}} < \frac{1}{2^n} < \frac{1}{n^c}$$

$$\frac{1}{n^{\sqrt{n}}} < \frac{1}{n^{\log n}} < \frac{1}{n^c}$$

2 Distinguisher for the injective PRG: G .

a $\text{bias}(A, \mathcal{L}_{\text{prg-real}}^G, \mathcal{L}_{\text{prg-rand}}^G) = \frac{1}{2^{n+\ell}}$

This distinguisher is no better than a random guess, except that it randomly guesses all possible outputs.

b This does not contradict G being a PRG. The security property of $\mathcal{L}_{\text{prg-real}}^G$ and $\mathcal{L}_{\text{prg-rand}}^G$ still holds.

c When the bias of G is not injective, the bias has the possibility of increasing a non-negligible amount, thus rendering $\mathcal{L}_{\text{prg-real}}$ insecure.

3 Secure Length-Tripling PRG

(a)

$A \diamond$

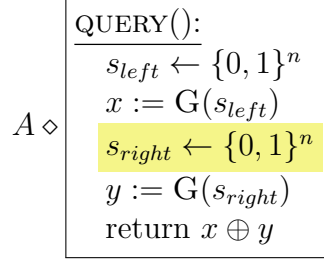
QUERY(): $s \leftarrow \{0, 1\}^{2n}$ $x := G(s_{\text{left}})$ $y := G(s_{\text{right}})$ return $x \oplus y$
--

(b)

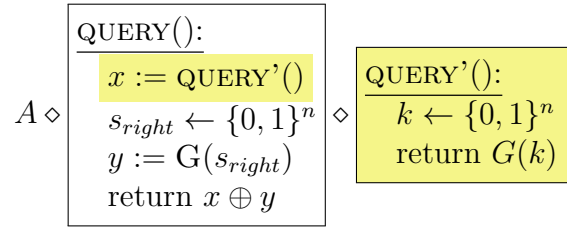
$A \diamond$

QUERY(): $s_{\text{left}} \leftarrow \{0, 1\}^n$ $s_{\text{right}} \leftarrow \{0, 1\}^n$ $x := G(s_{\text{left}})$ $y := G(s_{\text{right}})$ return $x \oplus y$

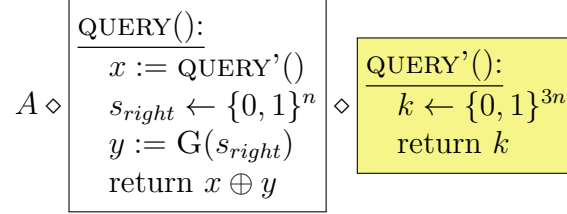
(c)



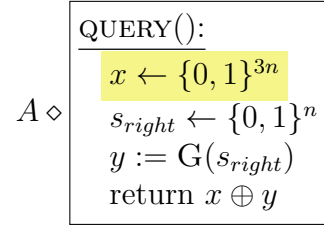
(d)



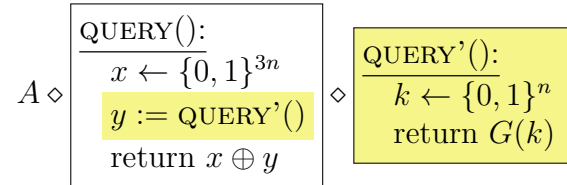
(e)



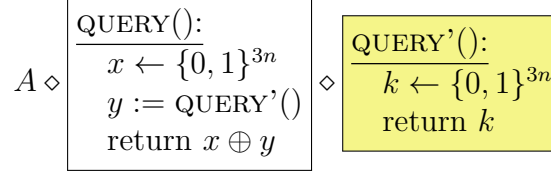
(f)



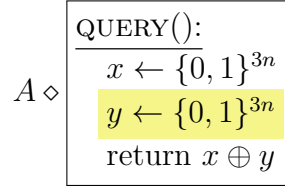
(g)



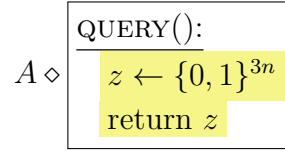
(h)



(i)



(j)



Justification of steps:

- (a) This is just $A \diamond \mathcal{L}_{\text{prg-real}}^G$ with the details of H filled in.
- (a) \Rightarrow (b) We have split s into its *left* and *right* parts. No effect on the program.
- (c) \Rightarrow (d) Factored out s_{left} and first call to G in its own subroutine $\text{QUERY}'()$. No effect on the program.
- (e) We swap the call to the length-tripling PRG G . Assuming G is a secure PRG, we can replace its call with the generation and return of $3n$ random-bits.
- (e) \Rightarrow (f) Inlining of $\text{QUERY}'()$. No effect on the program.
- (g) \Rightarrow (i) The process of (d) through (f) is repeated for y .
- (j) Because we know that XOR produces a random distribution at uniform, it is simple to state that this process is the same as generating $3n$ random-bits \square

4 Length-Quadrupling PRG Distinguisher

A():
 $s \leftarrow \{0, 1\}^n$
 $x := H(s)$
 return $x_{left} = x_{right}$

To show that H is **not** a PRG the above distinguisher is given. Because H uses G multiple times with the same s , x and y equal. Thus the bias for A is:

$$\begin{aligned}
 \text{bias}(A, \mathcal{L}_{\text{prg-real}}^H, \mathcal{L}_{\text{prg-rand}}^H) &= \\
 |Pr[A \diamond \mathcal{L}_{\text{prg-real}}^H \text{ outputs } 1] - Pr[A \diamond \mathcal{L}_{\text{prg-rand}}^H \text{ outputs } 1]| &= \\
 |1 - 0| &= \\
 1
 \end{aligned}$$