Pseudo random permutations

What is a Pseudo Random Permutation?

A family of functions $P_k$ is a $(t, q, \varepsilon)$-pseudo-random permutation if $\forall A$ running in time $\leq t$ and making $\leq q$ oracle queries then:

\[
Adv A = |\Pr[A^{P_k} = 0 | k \leftarrow U_1] - \Pr[A^{\pi} = 0 | \pi \leftarrow Perms]| \leq \varepsilon
\]

\[
P_k: \{0,1\}^n \rightarrow \{0,1\}^n, k \leftarrow \{0,1\}^l
\]

Remarks:

- $P_k$ is injective
  - Therefore, since domain of $P_k$ = range of $P_k$ $\Rightarrow$ $P_k$ is surjective
- This is not a bitwise-permutation. You could have $P_k(000) = 011$
- $P_k$ is injective and surjective $\Rightarrow$ $P_k$ is bijective $\iff$ $P_k$ is invertible

From an attacker’s point of view PRPs and PRFs look almost identical.

What is the size of the set of all PRPs $P_k: \{0,1\}^n \rightarrow \{0,1\}^n$?

\[
|\text{Perms}_{\{0,1\}^n}| = 2^n!
\]

There are $2^n$ elements in the domain, and there are $2^n!$ ways of mapping them to each other

PRPs versus PRFs theorem

PRPs and PRFs are $(\infty, q, \frac{q^2}{2^n+1})$-computationally indistinguishable.

\[
\begin{align*}
\text{Funcs}_{\{0,1\}^n}^{\infty, q} & \sim \frac{q^2}{2^n+1} \text{Perms}_{\{0,1\}^n}\\
\end{align*}
\]

Proof

Note that permutations are one-to-one, but functions might not be, and this is the only property you can use to distinguish between them.

Program $R$ (random function)

if $T[x]$ is not undefined then
  return $T[x]$
else
  $T[x] \leftarrow U_n$
  return $T[x]$

Program $\pi$ (random permutation)

if $T[x]$ is not undefined then
  return $T[x]$
else
  $y \leftarrow U_n$
  if $y \in \text{Range}(T)$
    bad = true
    $y \leftarrow U_n - \text{Range}(T)$
  return $T[x]$
\[ \text{Adv} A = |\Pr[A^R = 0|R \leftarrow \text{funcs}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms}|] = \]

Splitting them into cases, based on whether \textit{bad} is true or false...

\[ \text{Adv} A = |\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{false}] \Pr[\text{bad} = \text{false}] \\
+ \Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{true}] \Pr[\text{bad} = \text{true}] \\
- \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{false}] \Pr[\text{bad} = \text{false}] \\
- \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{true}] \Pr[\text{bad} = \text{true}] | \]

Grouping them based on whether \textit{bad} is true or false...

\[ \text{Adv} A = |(\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{false}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{false}] \Pr[\text{bad} = \text{false}] \\
+ (\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{true}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{true}] \Pr[\text{bad} = \text{true}] |) \Pr[\text{bad} = \text{true}] | \]

Applying \(|a + b| \leq |a| + |b|\)...

\[ \text{Adv} A \leq \Pr[\text{bad} = \text{false}] |(\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{false}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{false}] |) \Pr[\text{bad} = \text{false}] \\
+ \Pr[\text{bad} = \text{true}] |(\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{true}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{true}] |) \Pr[\text{bad} = \text{true}] | \]

When \textit{bad} is false, then the \(R\) and \(\pi\) programs behave the same and are indistinguishable. The advantage of the attacker in this case will be 0.

\[ \Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{false}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{false}] = 0 \]

When \textit{bad} is true, then we will bound advantage of the attacker by 1, which is the maximum he can have, assuming he has some very good method of distinguishing between

\[ \text{Adv} A \leq \Pr[\text{bad} = \text{true}] |(\Pr[A^R = 0|R \leftarrow \text{funcs} \cap \text{bad} = \text{true}] - \Pr[A^\pi = 0|\pi \leftarrow \text{perms} \cap \text{bad} = \text{true}] |) \leq \Pr[\text{bad} = \text{true}] | \]

\[ \text{Adv} A \leq \Pr[\text{bad} = \text{true}] \leq \frac{q^2}{2^{n+1}} \]

\textbf{Why:} What is the probability of getting the same number twice after picking \(q\) \(n\)-bit random numbers? (The birthday problem)

\textbf{Answer:} 0.3 \(q^2 \frac{2^n}{2^n} \leq p \leq 0.5 q^2 \frac{2^n}{2^n} \) (you can find a proof for this if you look up the “birthday problem”)

\textbf{Examples of PRPs}

\textbf{AES}

AES is a \((t, q, \frac{t}{2^{128}})\)-secure PRP

\textbf{Linear transformations (bad example)}

Let \(A = \{n \times n\ \text{invertible matrices over GF}(2)\} \)
This is a bad example since you can feed it a special kind of input which will reveal the columns of the matrix. For instance, if \( n = 3 \), and \( A_k = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \) then you can compute \( P \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \text{col}_1 \), \( P \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \text{col}_2 \), and \( P \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \text{col}_3 \) effectively obtaining the matrix and thus getting knowledge of the full behavior of \( P_k \).

**Real or random security (Real versus ideal world)**

**Ideal world:**

- Whenever Alice sends a message to Bob, she sends him \( E_k(m) \) and she also sends \( E_k($m) \) to Eve, where $ replaces \( m \) with random bits.
  
  o That’s to say Eve will always know that a message of a particular length was sent.

**Real world:**

- Whenever Alice sends a message \( E_k(m) \) to Bob, Eve gets a copy of \( E_k(m) \).
- Even can also provide a message \( m \) to Alice for her to send it encrypted as \( E_k(m) \) to Bob.
  
  o Eve can now see \( E_k(m) \).

In the real world, we need “indistinguishability under chosen plaintext attack”, a.k.a. IND-CPA.