Advanced Cryptography

（Provable Security）

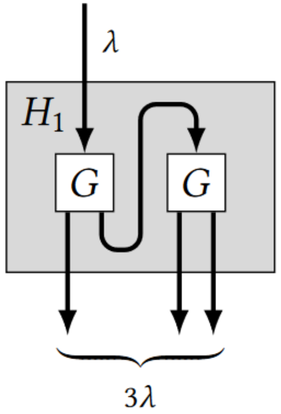
Yi LIU

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$H_1(s)$:
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$



$\mathcal{L}^{H_1}_{\text{prg-real}}$

$\text{QUERY}_{H_1}()$:
$s \leftarrow \{0,1\}^\lambda$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$

$\left.\begin{matrix} x \| y := G(s) \\ u \| v := G(y) \\ \text{return } x \| u \| v \end{matrix}\right\} H_1(s)$

$\text{QUERY}_{H_1}()$:
$x \| y := \text{QUERY}_G()$
$u \| v := G(y)$
return $x \| u \| v$

$\diamond$

$\mathcal{L}^{G}_{\text{prg-real}}$

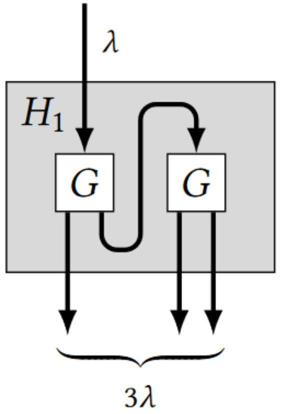
$\text{QUERY}_G()$:
$s \leftarrow \{0,1\}^\lambda$
return $G(s)$

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$H_1(s)$:
  $x \| y := G(s)$
  $u \| v := G(y)$
  return $x \| u \| v$



$\text{QUERY}_{H_1}()$: $x \| y := \text{QUERY}_G()$ $u \| v := G(y)$ return $x \| u \| v$	$\diamond$	$\mathcal{L}^G_{\text{prg-real}}$ $\text{QUERY}_G()$: $s \leftarrow \{0,1\}^\lambda$ return $G(s)$	

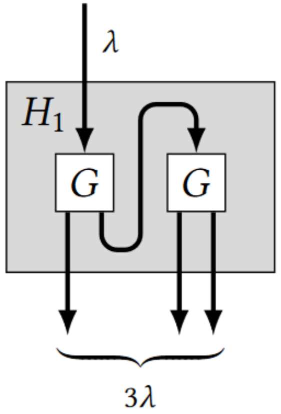
$\text{QUERY}_{H_1}()$: $x \| y := \text{QUERY}_G()$ $u \| v := G(y)$ return $x \| u \| v$	$\diamond$	$\mathcal{L}^G_{\text{prg-rand}}$ $\text{QUERY}_G()$: $r \leftarrow \{0,1\}^{2\lambda}$ return r

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$\underline{H_1(s):}$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$



$\underline{\text{QUERY}_{H_1}():}$
$x \| y := \text{QUERY}_G()$
$u \| v := G(y)$
return $x \| u \| v$

$\diamond$

$\mathcal{L}^G_{\text{prg-rand}}$
$\underline{\text{QUERY}_G():}$
$r \leftarrow \{0,1\}^{2\lambda}$
return r

$\Longrightarrow$

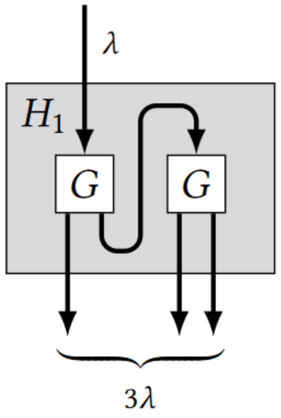
$\underline{\text{QUERY}_{H_1}():}$
$x \| y \leftarrow \{0,1\}^{2\lambda}$
$u \| v := G(y)$
return $x \| u \| v$

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$\underline{H_1(s):}$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$



$\underline{\text{QUERY}_{H_1}():}$
$x \| y \leftarrow \{0,1\}^{2\lambda}$
$u \| v := G(y)$
return $x \| u \| v$

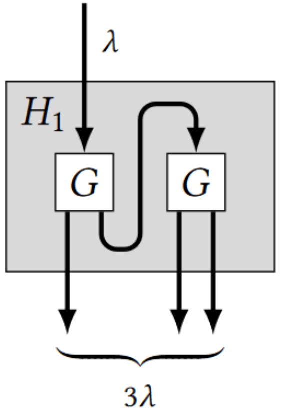
$\underline{\text{QUERY}_{H_1}():}$
$x \leftarrow \{0,1\}^{\lambda}$
$y \leftarrow \{0,1\}^{\lambda}$
$u \| v := G(y)$
return $x \| u \| v$

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$\underline{H_1(s):}$
$x\|y := G(s)$
$u\|v := G(y)$
return $x\|u\|v$



$\underline{\text{QUERY}_{H_1}():}$
$x \leftarrow \{0,1\}^\lambda$
$y \leftarrow \{0,1\}^\lambda$
$u\|v := G(y)$
return $x\|u\|v$

$\Rightarrow$

$\underline{\text{QUERY}_{H_1}():}$
$x \leftarrow \{0,1\}^\lambda$
$u\|v := \text{QUERY}_G()$
return $x\|u\|v$

$\diamond$

$\mathcal{L}^G_{\text{prg-real}}$

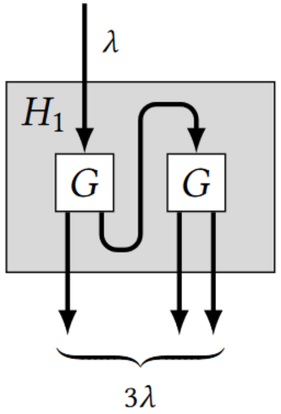
$\underline{\text{QUERY}_G():}$
$s \leftarrow \{0,1\}^\lambda$
return $G(s)$

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$H_1(s)$:
  $x \| y := G(s)$
  $u \| v := G(y)$
  return $x \| u \| v$



$\textsc{query}_{H_1}()$:
  $x \leftarrow \{0,1\}^\lambda$
  $u \| v := \textsc{query}_G()$
  return $x \| u \| v$

$\diamond$

$\mathcal{L}^G_{\text{prg-real}}$

$\textsc{query}_G()$:
  $s \leftarrow \{0,1\}^\lambda$
  return $G(s)$

$\Rightarrow$

$\textsc{query}_{H_1}()$:
  $x \leftarrow \{0,1\}^\lambda$
  $u \| v := \textsc{query}_G()$
  return $x \| u \| v$

$\diamond$

$\mathcal{L}^G_{\text{prg-rand}}$

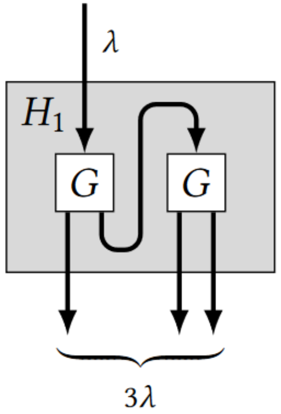
$\textsc{query}_G()$:
  $r \leftarrow \{0,1\}^{2\lambda}$
  return r

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$H_1(s)$:
- $x \| y := G(s)$
- $u \| v := G(y)$
- return $x \| u \| v$



$\text{QUERY}_{H_1}()$:
- $x \leftarrow \{0,1\}^\lambda$
- $u \| v := \text{QUERY}_G()$
- return $x \| u \| v$

$\diamond$

$\mathcal{L}^G_{\text{prg-rand}}$

$\text{QUERY}_G()$:
- $r \leftarrow \{0,1\}^{2\lambda}$
- return r

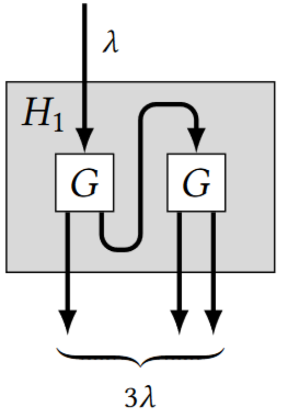
$\text{QUERY}_{H_1}()$:
- $x \leftarrow \{0,1\}^\lambda$
- $u \| v \leftarrow \{0,1\}^{2\lambda}$
- return $x \| u \| v$

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then H_1 is a secure (length-tripling) PRG.

Proof

$\underline{H_1(s):}$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$



$\underline{\text{QUERY}_{H_1}():}$
$x \leftarrow \{0,1\}^\lambda$
$u \| v \leftarrow \{0,1\}^{2\lambda}$
return $x \| u \| v$

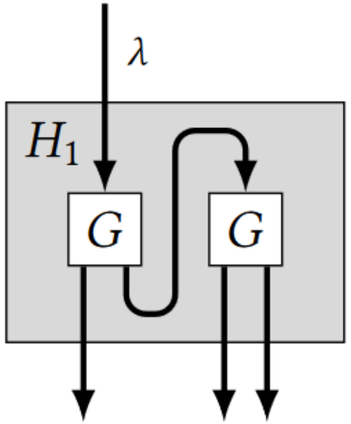
$\mathcal{L}^{H_1}_{\text{prg-rand}}$

$\underline{\text{QUERY}_{H_1}():}$
$r \leftarrow \{0,1\}^{3\lambda}$
return r

Extending the Stretch of a PRG

If G is a secure with stretch 1 PRG, then is H_1 a secure PRG with stretch 2?

$\underline{H_1(s):}$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| u \| v$



A Concrete Attack on H_2

Claim Construction H_2 is not a secure PRG, even if G is.

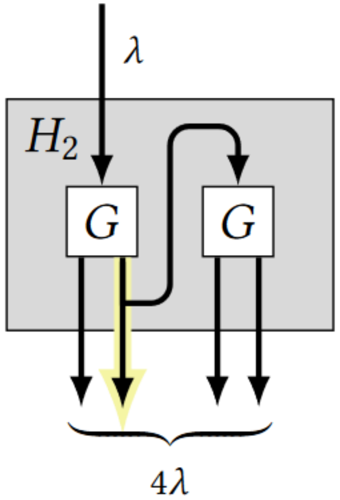
Proof

Consider the distinguisher $\mathcal{A}$

$$\boxed{\begin{array}{l} x\|y\|u\|v := \textsc{query}() \\ \text{return } G(y) \stackrel{?}{=} u\|v \end{array}}$$

$\underline{H_2(s):}$

$x\|y := G(s)$
$u\|v := G(y)$
return $x\|y\|u\|v$



$$\Pr\left[\mathcal{A} \diamond \mathcal{L}^{H_2}_{\text{prg-real}} \Rightarrow 1\right] = 1$$

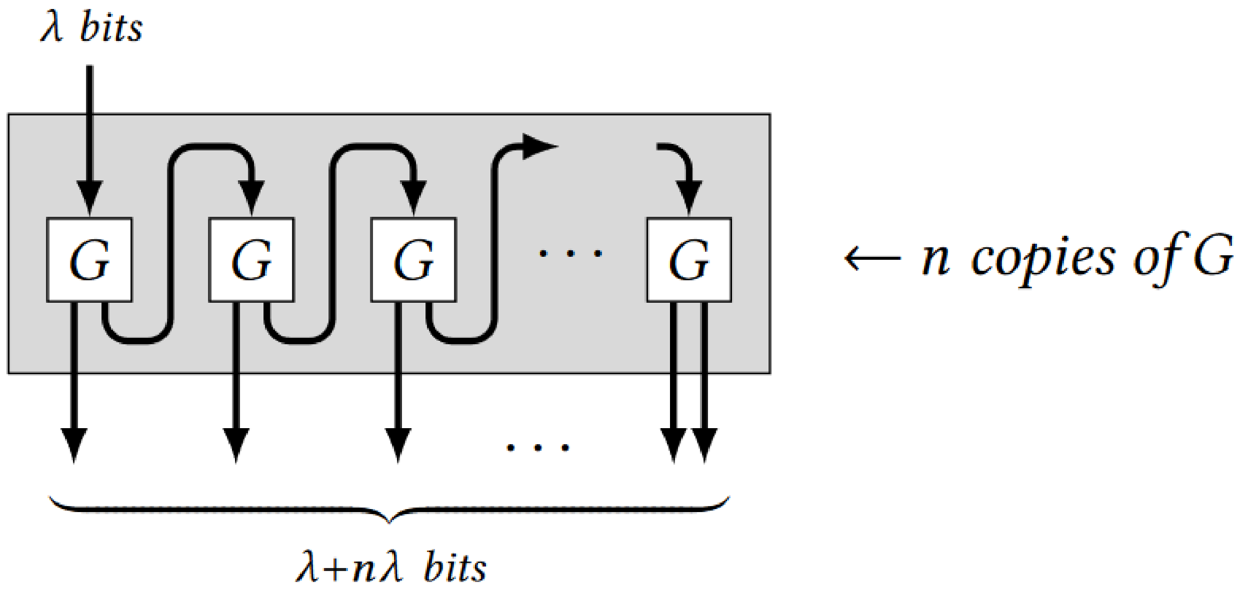
$$\Pr\left[\mathcal{A} \diamond \mathcal{L}^{H_2}_{\text{prg-rand}} \Rightarrow 1\right] = 1/2^{2\kappa}$$

Imagine that x and y being chosen before u and v. As soon as y is chosen, the value $G(y)$ is uniquely determined, since G is a deterministic algorithm.
Then $\mathcal{A}$ will output true if $u\|v$ is chosen exactly to equal this $G(y)$. Since u and v are chosen uniformly, and are a total of 2λ bits long, this event happens with probability $1/2^{2\lambda}$.

Extending the Stretch of a PRG

Claim If G is a secure length-doubling PRG, then for any n (polynomial function of λ) the following construction H_n is a secure PRG with stretch $n\lambda$:

$\underline{H_n(s):}$
  $s_0 := s$
  for $i = 1$ to n:
    $s_i \| t_i := G(s_{i-1})$
  return $t_1 \| \cdots \| t_n \| s_n$



Stream Cipher

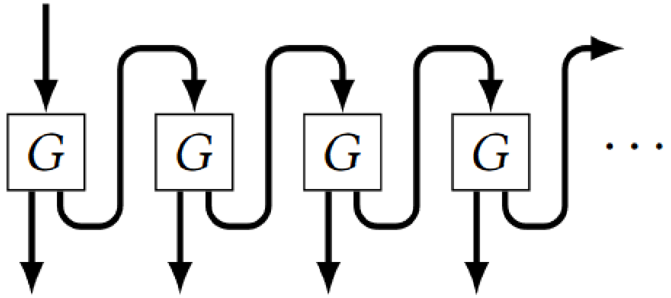
Definition A stream cipher is an algorithm G that takes a seed s and length ℓ as input, and outputs a string. It should satisfy the following requirements:

1. $G(s, \ell)$ is a string of length ℓ.

2. If $i < j$, then $G(s, i)$ is a prefix of $G(s, j)$.

3. For each n, the function $G(\cdot, n)$ is a secure PRG.

- The PRG-feedback construction can be used to construct a secure stream cipher



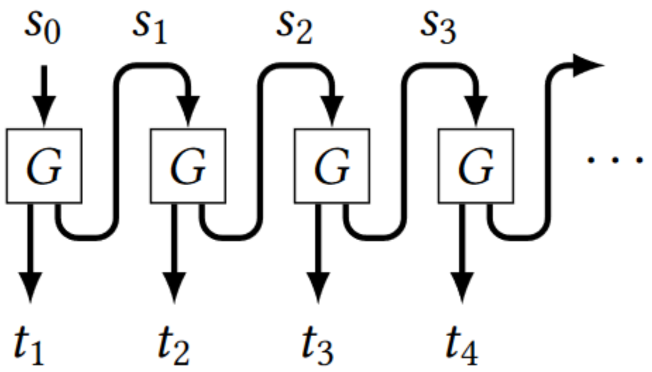
Symmetric Ratchet

- Suppose Alice & Bob share a symmetric key k and exchange messages over a long period of time.
- However, suppose Bob's device is compromised and an attacker learns k. Then the attacker can decrypt all past, present, and future ciphertexts that it saw!
- Alice & Bob can protect against such a key compromise by using the PRG-feedback stream cipher to constantly "update" their shared key.

Symmetric Ratchet

- Alice & Bob can protect against such a key compromise by using the PRG-feedback stream cipher to constantly "update" their shared key.
  - They use k to seed a chain of length-doubling PRGs, and both obtain the same stream of pseudorandom keys $t_1, t_2, \ldots$
  - They use t_i as a key to send/receive the ith message.
  - After making a call to the PRG, they erase the PRG input from memory, and only remember the PRG's output. After using t_i to send/receive a message, they also erase it from memory.
- This way of using and forgetting a sequence of keys is called a symmetric ratchet.

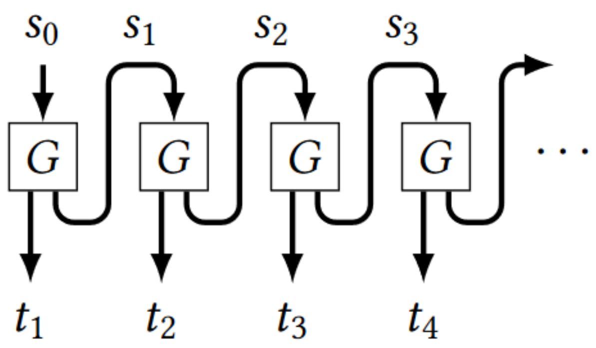
$s_0 = k$
for $i = 1$ to ∞:
  $s_i \| t_i := G(s_{i-1})$
  erase s_{i-1} from memory
  use t_i to encrypt/decrypt the ith message
  erase t_i from memory



Symmetric Ratchet

- Suppose that an attacker compromises Bob's device after n ciphertexts have been sent. The only value residing in memory is s_n , which the attacker learns.
- The attacker learns no information about $t_1, \ldots, t_n$ from s_n , which implies that the previous ciphertexts remain safe.
- The adversary only compromises the security of future messages, but not past messages.
  - **Forward secrecy**: messages in the present are protected against a key-compromise that happens in the future.

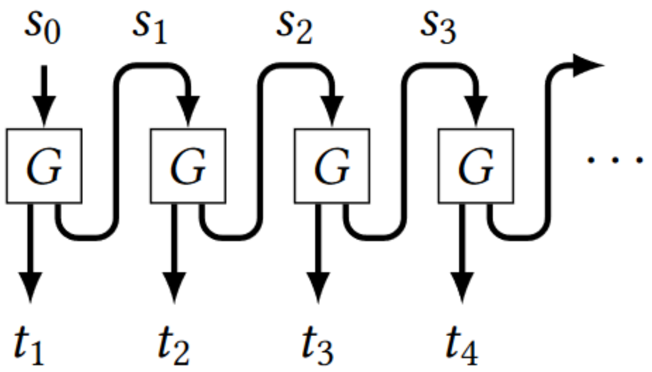
$s_0 = k$
for $i = 1$ to ∞:
  $s_i \| t_i := G(s_{i-1})$
  erase s_{i-1} from memory
  use t_i to encrypt/decrypt the ith message
  erase t_i from memory



Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

$s_0 = k$
for $i = 1$ to ∞:
  $s_i \| t_i := G(s_{i-1})$
  erase s_{i-1} from memory
  use t_i to encrypt/decrypt the ith message
  erase t_i from memory



Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation is captured by the following library

$\text{ATTACK}(m_1, \ldots, m_n)$:

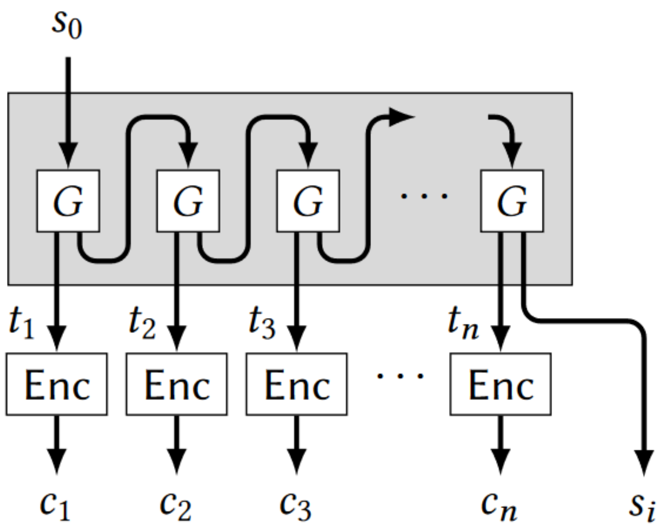
$\quad s_0 \leftarrow \{0,1\}^\lambda$

$\quad$ for $i = 1$ to n:

$\qquad s_i \| t_i := G(s_{i-1})$

$\qquad c_i \leftarrow \Sigma.\mathsf{Enc}(t_i, m_i)$

$\quad$ return $(c_1, \ldots, c_n, s_n)$



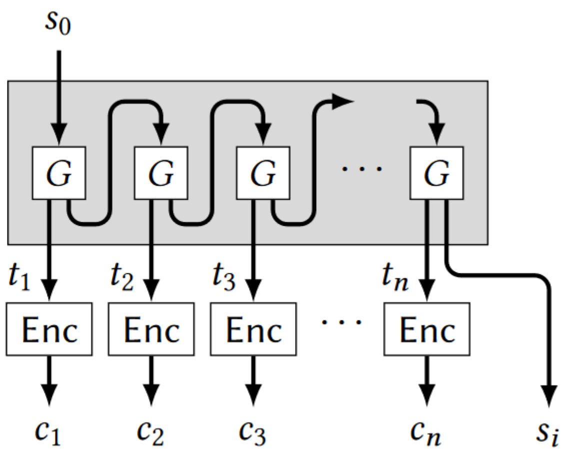
Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

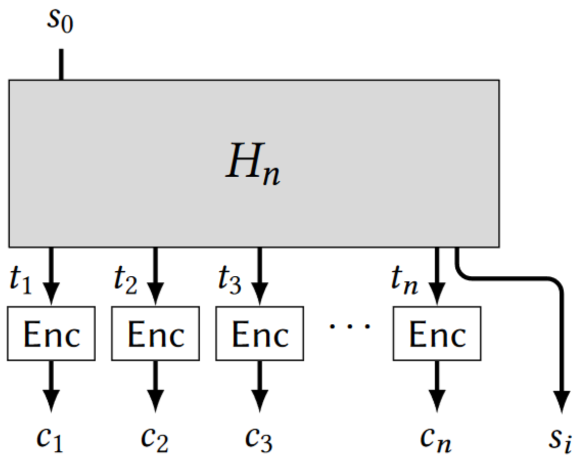
Proof

The situation is captured by the following library

$\underline{\text{ATTACK}(m_1, \ldots, m_n)}$:
- $s_0 \leftarrow \{0,1\}^\lambda$
- for $i = 1$ to n:
  - $s_i \| t_i := G(s_{i-1})$
  - $c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i)$
- return $(c_1, \ldots, c_n, s_n)$



$\underline{\text{ATTACK}(m_1, \ldots, m_n)}$:
- $s_0 \leftarrow \{0,1\}^\lambda$
- $t_1 \| \cdots \| t_n \| s_n := H_n(s_0)$
- for $i = 1$ to n:
  - $c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i)$
- return $(c_1, \ldots, c_n, s_n)$



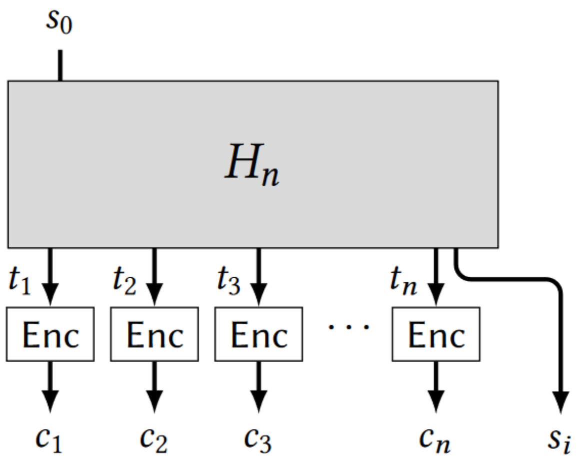
Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation is captured by the following library

$\underline{\textsc{attack}(m_1, \ldots, m_n):}$
$s_0 \leftarrow \{0,1\}^\lambda$
$t_1 \| \cdots \| t_n \| s_n := H_n(s_0)$
for $i = 1$ to n:
  $c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i)$
return $(c_1, \ldots, c_n, s_n)$



$\underline{\textsc{attack}(m_1, \ldots, m_n):}$
for $i = 1$ to n:
  $t_i \leftarrow \{0,1\}^\lambda$
$s_n \leftarrow \{0,1\}^\lambda$
for $i = 1$ to n:
  $c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i)$
return $(c_1, \ldots, c_n, s_n)$

Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation captured by the following library

$$
\begin{array}{l}
\underline{\text{ATTACK}(m_1, \ldots, m_n):} \\
\quad \text{for } i = 1 \text{ to } n: \\
\qquad t_i \leftarrow \{0,1\}^\lambda \\
\quad s_n \leftarrow \{0,1\}^\lambda \\
\quad \text{for } i = 1 \text{ to } n: \\
\qquad c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i) \\
\quad \text{return } (c_1, \ldots, c_n, s_n)
\end{array}
\Rightarrow
\begin{array}{l}
\underline{\text{ATTACK}(m_1, \ldots, m_n):} \\
\quad \text{for } i = 1 \text{ to } n: \\
\qquad t_i \leftarrow \{0,1\}^\lambda \\
\qquad c_i \leftarrow \Sigma.\text{Enc}(t_i, m_i) \\
\quad s_n \leftarrow \{0,1\}^\lambda \\
\quad \text{return } (c_1, \ldots, c_n, s_n)
\end{array}
$$

Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation is captured by the following library

$\underline{\textsc{attack}(m_1, \ldots, m_n):}$
  for $i = 1$ to n:
    $t_i \leftarrow \{0,1\}^\lambda$
    $c_i \leftarrow \Sigma.\mathsf{Enc}(t_i, m_i)$
  $s_n \leftarrow \{0,1\}^\lambda$
  return $(c_1, \ldots, c_n, s_n)$

$\underline{\textsc{attack}(m_1, \ldots, m_n):}$
  for $i = 1$ to n:
    $c_i \leftarrow \textsc{ctxt}(m_i)$
  $s_n \leftarrow \{0,1\}^\lambda$
  return $(c_1, \ldots, c_n, s_n)$

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

$\underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):}$
  $k \leftarrow \Sigma.\mathsf{KeyGen}$
  $c \leftarrow \Sigma.\mathsf{Enc}(k, m)$
  return c

Symmetric Ratchet

Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation is captured by the following library

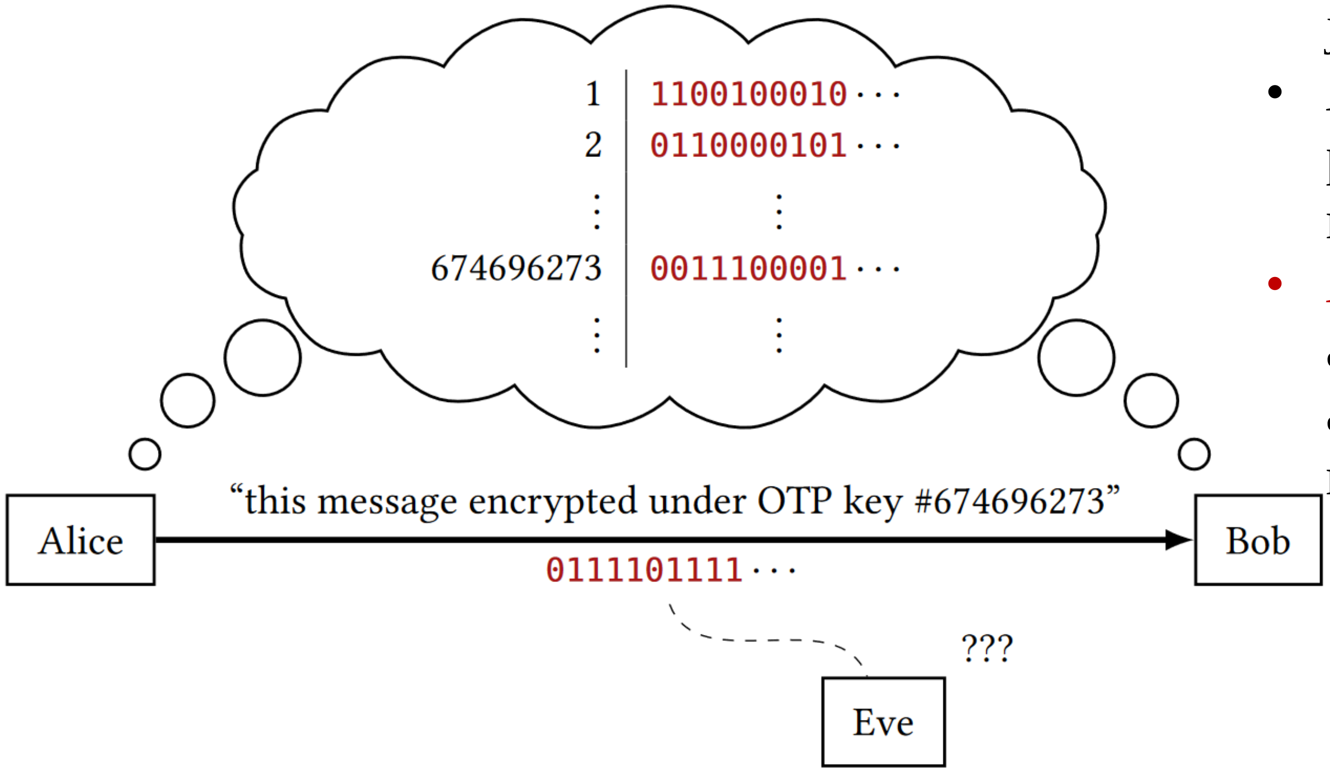
$$\begin{array}{|l|}\hline \underline{\textsc{attack}(m_1,\ldots,m_n):} \\ \quad \text{for } i = 1 \text{ to } n: \\ \qquad c_i \leftarrow \textsc{ctxt}(m_i) \\ \quad s_n \leftarrow \{0,1\}^\lambda \\ \quad \text{return } (c_1,\ldots,c_n,s_n) \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k,m) \\ \quad \text{return } c \\ \hline \end{array} \Rightarrow \begin{array}{|l|}\hline \underline{\textsc{attack}(m_1,\ldots,m_n):} \\ \quad \text{for } i = 1 \text{ to } n: \\ \qquad c_i \leftarrow \Sigma.C \\ \quad s_n \leftarrow \{0,1\}^\lambda \\ \quad \text{return } (c_1,\ldots,c_n,s_n) \\ \hline \end{array}$$

The attacker cannot distinguish the first n ciphertexts from random values, even when seeing s_n.

Pseudorandom Functions & Block Ciphers

Pseudorandom Functions

- Imagine if Alice & Bob had an infinite amount of shared randomness. They could split it up into λ-bit chunks and use each one as a one-time pad whenever they want to send an encrypted message of length λ.



- An exponential amount of something is often just as good as an infinite amount.
- A shared table containing "only" 2^{λ} one-time pad keys would be quite useful for encrypting as many messages as you could ever need.
- A pseudorandom function (PRF) is a tool that allows Alice & Bob to achieve the effect of such an exponentially large table of shared randomness in practice.

Pseudorandom Functions

- Imagine a huge table of shared data stored as an array T , so the ith item is referenced as $T[i]$. Instead of thinking of i as an integer, we can also think of i as a binary string. If the array has 2^{in} items, then i will be an in-bit string.
- A pseudorandom function emulates the functionality of a huge array. It is a function F that takes an input from $\{0,1\}^{in}$ and gives an output from $\{0,1\}^{out}$.
- However, F also takes an additional argument called the seed, which acts as a kind of secret key.

Pseudorandom Functions

- The goal of a pseudorandom function is to "look like" a uniformly chosen array / lookup table.
- We want to allow situations like $in \geq \lambda$
  - In those cases the first library runs in exponential time. It is generally convenient to build our security definitions with libraries that run in polynomial time. (The definition of indistinguishability requires all calling programs to run in polynomial time.)
  - Populate T in a lazy / on-demand way

for $x \in \{0,1\}^{in}$:
  $T[x] \leftarrow \{0,1\}^{out}$

$\underline{\text{LOOKUP}(x \in \{0,1\}^{in})\text{:}}$
  return $T[x]$

$k \leftarrow \{0,1\}^{\lambda}$

$\underline{\text{LOOKUP}(x \in \{0,1\}^{in})\text{:}}$
  return $F(k,x)$

Pseudorandom Functions

Definition Let $F : \{0,1\}^\lambda \times \{0,1\}^{in} \to \{0,1\}^{out}$ be a deterministic function. We say that F is a secure pseudorandom function (PRF) if $\mathcal{L}^F_{\text{prf-real}} \approx \mathcal{L}^F_{\text{prf-rand}}$, where:

$\mathcal{L}^F_{\text{prf-real}}$
$k \leftarrow \{0,1\}^\lambda$
$\underline{\textsc{lookup}(x \in \{0,1\}^{in}):}$
return $F(k,x)$

$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty assoc. array
$\underline{\textsc{lookup}(x \in \{0,1\}^{in}):}$
if $T[x]$ undefined:
$T[x] \leftarrow \{0,1\}^{out}$
return $T[x]$

Note that even in the case of a "random function" ($\mathcal{L}^F_{\text{prf-rand}}$), the function T itself is still deterministic!

Pseudorandom Functions

Q: How many functions for $\mathcal{L}^F_{\text{prf-rand}}$?

A: $2^{out \cdot 2^{in}}$

Q: How many function for $\mathcal{L}^F_{\text{prf-real}}$?

A: "only" 2^λ possible functions

$\mathcal{L}^F_{\text{prf-real}}$

$k \leftarrow \{0,1\}^\lambda$

$\underline{\textsc{lookup}(x \in \{0,1\}^{in})\text{:}}$
  return $F(k, x)$

$\mathcal{L}^F_{\text{prf-rand}}$

T := empty assoc. array

$\underline{\textsc{lookup}(x \in \{0,1\}^{in})\text{:}}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{out}$
  return $T[x]$

Pseudorandom Functions

Suppose we have a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and use it to construct a PRF F as follows:

$$\begin{array}{|l|}\hline \underline{F(k,x):} \\ \quad \text{return } G(k) \oplus x \\ \hline\end{array}$$

Is this a secure PRF?

$$\begin{array}{|l|}\hline \multicolumn{1}{|c|}{\mathcal{A}} \\ \hline \text{pick } x_1, x_2 \in \{0,1\}^{2\lambda} \textit{ arbitrarily} \text{ so that } x_1 \neq x_2 \\ z_1 := \textsc{lookup}(x_1) \\ z_2 := \textsc{lookup}(x_2) \\ \text{return } z_1 \oplus z_2 \stackrel{?}{=} x_1 \oplus x_2 \\ \hline\end{array}$$

Pseudorandom Functions

Suppose we have a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and use it to construct a PRF F as follows:

$$\frac{F(k,x):}{\quad \text{return } G(k) \oplus x}$$

$\mathcal{A}$
pick $x_1 \neq x_2 \in \{0,1\}^{2\lambda}$
$z_1 := \text{LOOKUP}(x_1)$
$z_2 := \text{LOOKUP}(x_2)$
return $z_1 \oplus z_2 \stackrel{?}{=} x_1 \oplus x_2$

$\diamond$

$\mathcal{L}^F_{\text{prf-real}}$
$k \leftarrow \{0,1\}^\lambda$
$\underline{\text{LOOKUP}(x):}$
return $G(k) \oplus x$ // $F(k,x)$

$\Pr[\mathcal{A} \diamond \mathcal{L}^F_{\text{prf-real}} \Rightarrow 1] = 1$

Pseudorandom Functions

Suppose we have a length-doubling PRG $G : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ and use it to construct a PRF F as follows:

$\underline{F(k,x)}$:
  return $G(k) \oplus x$

$\mathcal{A}$ will output 1 if and only if z_2 is chosen to be exactly the value $x_1 \oplus x_2 \oplus z_1$.

$\mathcal{A}$		$\mathcal{L}^F_{\text{prf-rand}}$
pick $x_1 \neq x_2 \in \{0,1\}^{2\lambda}$ $z_1 := \text{LOOKUP}(x_1)$ $z_2 := \text{LOOKUP}(x_2)$ return $z_1 \oplus z_2 \stackrel{?}{=} x_1 \oplus x_2$	$\diamond$	$T :=$ empty assoc. array $\underline{\text{LOOKUP}(x)}$: if $T[x]$ undefined: $T[x] \leftarrow \{0,1\}^{2\lambda}$ return $T[x]$

$$\Pr[\mathcal{A} \diamond \mathcal{L}^F_{\text{prf-real}} \Rightarrow 1] = 1$$

$$\Pr[\mathcal{A} \diamond \mathcal{L}^F_{\text{prf-rand}} \Rightarrow 1] = 1/2^{2\lambda}$$

PRFs vs PRGs

PRG can be used to construct a PRF, and vice-versa.

- The construction of a PRG from PRF is practical, and is one of the more common ways to obtain a PRG in practice.
- The construction of a PRF from PRG is more of theoretical interest and does not reflect how PRFs are designed in practice.

Constructing a PRG from a PRF

Suppose we have a PRF $F: \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ (i.e., $in = out = \lambda$). We can build a length-doubling PRG in the following way (Counter PRG):

$$
\begin{array}{l}
\underline{G(s):} \\
\quad x := F(s, \mathtt{0}\cdots\mathtt{00}) \\
\quad y := F(s, \mathtt{0}\cdots\mathtt{01}) \\
\quad \text{return } x \| y
\end{array}
$$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof

During the proof, we are allowed to use the fact that F is a secure PRF. That is, we can use the fact that the following two libraries are indistinguishable:

$\mathcal{L}^F_{\text{prf-real}}$
$k \leftarrow \{0,1\}^\lambda$
$\underline{\text{LOOKUP}(x \in \{0,1\}^{in})\text{:}}$
return $F(k,x)$

$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty assoc. array
$\underline{\text{LOOKUP}(x \in \{0,1\}^{in})\text{:}}$
if $T[x]$ undefined:
$T[x] \leftarrow \{0,1\}^{out}$
return $T[x]$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof

In order to prove that G is a secure PRG, we must prove that the following libraries are indistinguishable:

$\mathcal{L}^G_{\text{prg-real}}$

$\underline{\text{QUERY}():}$

$$\left.\begin{array}{l} s \leftarrow \{0,1\}^\lambda \\ x := F(s, 0\cdots00) \\ y := F(s, 0\cdots01) \\ \text{return } x \| y \end{array}\right\} // G(s)$$

$\mathcal{L}^G_{\text{prg-rand}}$

$\underline{\text{QUERY}():}$

$$\begin{array}{l} r \leftarrow \{0,1\}^{2\lambda} \\ \text{return } r \end{array}$$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof

We can only use $\mathcal{L}_{\mathtt{prf-real}}$ to deal with a single PRF seed at a time, but $\mathcal{L}_{\mathtt{prg-real}}$ deals with many PRG seeds at a time.

To address this, we will have to apply the security of F (i.e., replace $\mathcal{L}_{\mathtt{prf-real}}$ with $\mathcal{L}_{\mathtt{prf-rand}}$) many times during the proof.

This proof will have a variable number of hybrids that depends on the calling program.

$\mathcal{L}^{G}_{\text{prg-real}}$

QUERY():
- $s \leftarrow \{0,1\}^\lambda$
- $x := F(s, 0\cdots00)$
- $y := F(s, 0\cdots01)$
- return $x \| y$

// $G(s)$

$\mathcal{L}^{F}_{\text{prf-real}}$

$k \leftarrow \{0,1\}^\lambda$

LOOKUP($x \in \{0,1\}^{in}$):
- return $F(k, x)$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof

This proof will have a variable number of hybrids that depends on the calling program.

$\mathcal{L}_{\text{hyb-}i}$:

$count := 0$

QUERY():
- $count := count + 1$
- if $count \leqslant \boxed{i}$:
  - $r \leftarrow \{\texttt{0},\texttt{1}\}^{2\lambda}$
  - return r
- else:
  - $s \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda}$
  - $x := F(s, \texttt{0}\cdots\texttt{00})$
  - $y := F(s, \texttt{0}\cdots\texttt{01})$
  - return $x \| y$

- In $\mathcal{L}_{\text{hyb}-0}$, the if-branch is never taken ($count \leq 0$ is never true). This library behaves exactly like $\mathcal{L}_{\text{prg}-\text{real}}$.
- If q is the total number of times that the calling program calls QUERY, then in $\mathcal{L}_{\text{hyb}-q}$, the if-branch is always taken ($count \leq q$ is always true). This library behaves exactly like $\mathcal{L}_{\text{prg}-\text{rand}}$.
- Therefore, $\mathcal{L}_{\text{hyb}-0} \equiv \mathcal{L}_{\text{prg}-\text{real}}$, $\mathcal{L}_{\text{hyb}-q} \equiv \mathcal{L}_{\text{prg}-\text{rand}}$.
- To complete the proof, we must show that $\mathcal{L}_{\text{hyb}-(i-1)} \approx \mathcal{L}_{\text{hyb}-i}$ for all i.

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb}-(i-1)} \equiv \mathcal{L}_{\text{hyb}-i}$ for all i.

$\mathcal{L}_{\text{hyb-}i}$:

$count := 0$

<u>QUERY():</u>
- $count := count + 1$
- if $count \leqslant \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots 00)$
  - $y := F(s, 0\cdots 01)$
  - return $x\|y$

$count := 0$

<u>QUERY():</u>
- $count := count + 1$
- if $count < \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = \boxed{i}$:
  - $s^* \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s^*, 0\cdots 00)$
  - $y := F(s^*, 0\cdots 01)$
  - return $x\|y$
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots 00)$
  - $y := F(s, 0\cdots 01)$
  - return $x\|y$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb-}(i-1)} \equiv \mathcal{L}_{\text{hyb-}i}$ for all i.

$count := 0$

<u>QUERY():</u>
- $count := count + 1$
- if $count < \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = \boxed{i}$:
  - $s^* \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s^*, 0\cdots00)$
  - $y := F(s^*, 0\cdots01)$
  - return $x \| y$
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots00)$
  - $y := F(s, 0\cdots01)$
  - return $x \| y$

$\Rightarrow$

$count := 0$

<u>QUERY():</u>
- $count := count + 1$
- if $count < \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = \boxed{i}$:
  - $x :=$ LOOKUP$(0\cdots00)$
  - $y :=$ LOOKUP$(0\cdots01)$
  - return $x \| y$
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots00)$
  - $y := F(s, 0\cdots01)$
  - return $x \| y$

$\diamond$

$\mathcal{L}^{F}_{\text{prf-real}}$

$k \leftarrow \{0,1\}^{\lambda}$

<u>LOOKUP(x):</u>
- return $F(k, x)$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb-}(i-1)} \equiv \mathcal{L}_{\text{hyb-}i}$ for all i.

$count := 0$

$\textsc{query}()$:
  $count := count + 1$
  if $count < \boxed{i}$:
    $r \leftarrow \{0,1\}^{2\lambda}$
    return r
  elsif $count = \boxed{i}$:
    $x := \textsc{lookup}(0\cdots00)$
    $y := \textsc{lookup}(0\cdots01)$
    return $x \| y$
  else:
    $s \leftarrow \{0,1\}^{\lambda}$
    $x := F(s, 0\cdots00)$
    $y := F(s, 0\cdots01)$
    return $x \| y$

$\diamond$

$\mathcal{L}^F_{\text{prf-real}}$

$k \leftarrow \{0,1\}^{\lambda}$

$\textsc{lookup}(x)$:
  return $F(k, x)$

$\Rightarrow$

$count := 0$

$\textsc{query}()$:
  $count := count + 1$
  if $count < \boxed{i}$:
    $r \leftarrow \{0,1\}^{2\lambda}$
    return r
  elsif $count = \boxed{i}$:
    $x := \textsc{lookup}(0\cdots00)$
    $y := \textsc{lookup}(0\cdots01)$
    return $x \| y$
  else:
    $s \leftarrow \{0,1\}^{\lambda}$
    $x := F(s, 0\cdots00)$
    $y := F(s, 0\cdots01)$
    return $x \| y$

$\diamond$

$\mathcal{L}^F_{\text{prf-rand}}$

$T :=$ empty assoc. array

$\textsc{lookup}(x)$:
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{\lambda}$
  return $T[x]$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb}-(i-1)} \equiv \mathcal{L}_{\text{hyb}-i}$ for all i.

$count := 0$

QUERY():
- $count := count + 1$
- if $count < i$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = i$:
  - $x :=$ LOOKUP($0\cdots00$)
  - $y :=$ LOOKUP($0\cdots01$)
  - return $x \| y$
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots00)$
  - $y := F(s, 0\cdots01)$
  - return $x \| y$

$\diamond$

$\mathcal{L}^{F}_{\text{prf-rand}}$

$T :=$ empty assoc. array

LOOKUP(x):
- if $T[x]$ undefined:
  - $T[x] \leftarrow \{0,1\}^{\lambda}$
- return $T[x]$

$\Rightarrow$

$count := 0$

QUERY():
- $count := count + 1$
- if $count < i$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = i$:
  - $x :=$ LOOKUP($0\cdots00$)
  - $y :=$ LOOKUP($0\cdots01$)
  - return $x \| y$
- else:
  - $s \leftarrow \{0,1\}^{\lambda}$
  - $x := F(s, 0\cdots00)$
  - $y := F(s, 0\cdots01)$
  - return $x \| y$

$\diamond$

LOOKUP(x):
- $r \leftarrow \{0,1\}^{\lambda}$
- return r

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb-}(i-1)} \equiv \mathcal{L}_{\text{hyb-}i}$ for all i.

$$\boxed{\begin{array}{l} count := 0 \\ \\ \underline{\text{QUERY}():} \\ \quad count := count + 1 \\ \quad \text{if } count < \boxed{i}: \\ \qquad r \leftarrow \{0,1\}^{2\lambda} \\ \qquad \text{return } r \\ \quad \text{elsif } count = \boxed{i}: \\ \qquad x := \text{LOOKUP}(0\cdots00) \\ \qquad y := \text{LOOKUP}(0\cdots01) \\ \qquad \text{return } x\|y \\ \quad \text{else:} \\ \qquad s \leftarrow \{0,1\}^{\lambda} \\ \qquad x := F(s, 0\cdots00) \\ \qquad y := F(s, 0\cdots01) \\ \qquad \text{return } x\|y \end{array}} \diamond \boxed{\begin{array}{l} \underline{\text{LOOKUP}(x):} \\ \quad r \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } r \end{array}} \Rightarrow \boxed{\begin{array}{l} count := 0 \\ \\ \underline{\text{QUERY}():} \\ \quad count := count + 1 \\ \quad \text{if } count < \boxed{i}: \\ \qquad r \leftarrow \{0,1\}^{2\lambda} \\ \qquad \text{return } r \\ \quad \text{elsif } count = \boxed{i}: \\ \qquad x \leftarrow \{0,1\}^{\lambda} \\ \qquad y \leftarrow \{0,1\}^{\lambda} \\ \qquad \text{return } x\|y \\ \quad \text{else:} \\ \qquad s \leftarrow \{0,1\}^{\lambda} \\ \qquad x := F(s, 0\cdots00) \\ \qquad y := F(s, 0\cdots01) \\ \qquad \text{return } x\|y \end{array}}$$

Constructing a PRG from a PRF

Claim If F is a secure PRF, then the counter PRG construction G is a secure PRG.

proof To complete the proof, we must show that $\mathcal{L}_{\text{hyb}-(i-1)} \equiv \mathcal{L}_{\text{hyb}-i}$ for all i.

$$\mathcal{L}_{\text{prg-real}} \equiv \mathcal{L}_{\text{hyb}-0} \approx \mathcal{L}_{\text{hyb}-1} \approx \cdots \approx \mathcal{L}_{\text{hyb}-q} \equiv \mathcal{L}_{\text{prg-rand}}$$

$count := 0$

<u>QUERY():</u>

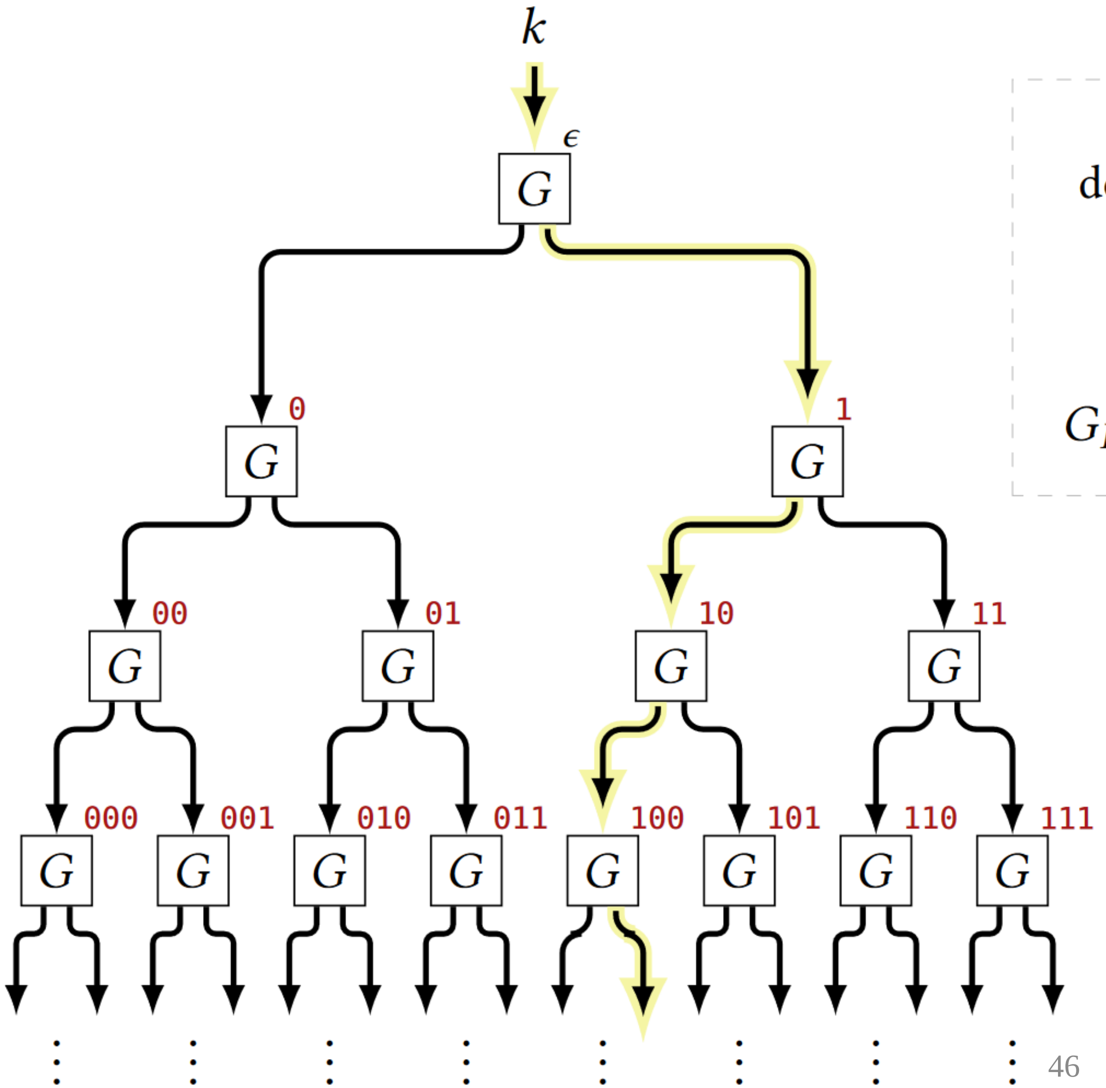
  $count := count + 1$
  if $count < \boxed{i}$:
    $r \leftarrow \{0,1\}^{2\lambda}$
    return r
  elsif $count = \boxed{i}$:
    $x \leftarrow \{0,1\}^{\lambda}$
    $y \leftarrow \{0,1\}^{\lambda}$
    return $x \| y$
  else:
    $s \leftarrow \{0,1\}^{\lambda}$
    $x := F(s, 0 \cdots 00)$
    $y := F(s, 0 \cdots 01)$
    return $x \| y$

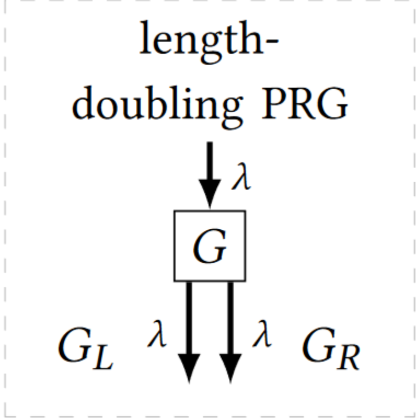
Constructing a PRF from a PRG

- To extend PRG's stretch
  - By making a long chain (like a linked list) of PRGs.
- Constructing a PRF from a PRG
  - By chaining PRGs together in a binary tree
  - The leaves of the tree correspond to final outputs of the PRF.
  - If we want a PRF with an exponentially large domain (e.g., $in = \lambda$), the binary tree itself is exponentially large!
  - However, it is still possible to compute any individual leaf efficiently by simply traversing the tree from root to leaf.

Constructing a PRF from a PRG

For convenience, we will write $G_L(k)$ and $G_R(k)$ to denote the first λ bits and last λ bits of $G(k)$, respectively.





- A complete binary tree of height *in* (the input length of the PRF).
- Every node has a position which can be written as a binary string.
  - 0 means "go left" and 1 means "go right." the root has position ϵ (the empty string).
- Running the PRF on some input does not involve computing labels for the entire tree, only along a single path from root to leaf.

Constructing a PRF from a PRG

$$in = arbitrary$$
$$out = \lambda$$

$\underline{F(k, x \in \{0,1\}^{in}):}$
$v := k$
for $i = 1$ to in:
  if $x_i = 0$ then $v := G_L(v)$
  if $x_i = 1$ then $v := G_R(v)$
return v

Claim If G is a secure PRG, then the construction is a secure PRF.

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof We prove the claim using a sequence of hybrids.

The number of hybrids depends on in.

$$\mathcal{L}^{F}_{\text{prf-real}} \equiv \mathcal{L}_{\text{hyb-}0} \approx \mathcal{L}_{\text{hyb-}1} \approx \cdots \approx \mathcal{L}_{\text{hyb-}q} \equiv \mathcal{L}^{F}_{\text{prf-rand}}$$

$\mathcal{L}_{\text{hyb-d}}$

T := empty assoc. array
 	
<u>QUERY(x):</u>
  p := first $\boxed{d}$ bits of x
  if $T[p]$ undefined:
    $T[p] \leftarrow \{0,1\}^\lambda$
  $v := T[p]$
  for $i = \boxed{d+1}$ to in:
    if $x_i = 0$ then $v := G_L(v)$
    if $x_i = 1$ then $v := G_R(v)$
  return v

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof We prove the claim using a sequence of hybrids.

$\mathcal{L}_{\text{hyb-0}}$

T := empty assoc. array	k := undefined
	// k is alias for $T[\epsilon]$
LOOKUP(x):	
p := first 0 bits of x	$p = \epsilon$
if $T[p]$ undefined:	if k undefined:
$T[p] \leftarrow \{0,1\}^\lambda$	$k \leftarrow \{0,1\}^\lambda$
$v := T[p]$	
for i = 1 to *in*:	$v := F(k, x)$
if $x_i = 0$ then $v := G_L(v)$	
if $x_i = 1$ then $v := G_R(v)$	
return v	return $F(k, x)$

The only difference is when the PRF seed k ($T[\epsilon]$) is sampled: eagerly at initialization time in $\mathcal{L}^F_{\text{prf-real}}$ vs. at the last possible minute in $\mathcal{L}_{\text{hyb-0}}$. Hence, $\mathcal{L}^F_{\text{prf-real}} \equiv \mathcal{L}_{\text{hyb-0}}$

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof We prove the claim using a sequence of hybrids.

$\mathcal{L}_{\text{hyb-}in}$	
$T :=$ empty assoc. array	
$\underline{\text{LOOKUP}(x)\text{:}}$	
$p :=$ first $\boxed{in}$ bits of x	$p = x$
if $T[p]$ undefined:	if $T[x]$ undefined:
$T[p] \leftarrow \{0,1\}^\lambda$	$T[x] \leftarrow \{0,1\}^\lambda$
$v := T[p]$	
for $i = \boxed{in + 1}$ to in:	
if $x_i = 0$ then $v := G_L(v)$	// *unreachable*
if $x_i = 1$ then $v := G_R(v)$	
return v	return $T[x]$

$$\mathcal{L}_{\text{hyb}-in} \equiv \mathcal{L}^{F}_{\text{prf}-\text{rand}}$$

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof To finish the proof, we show that $\mathcal{L}_{\text{hyb}-(d-1)}$ and $\mathcal{L}_{\text{hyb}-d}$ are indistinguishable.

$\mathcal{L}_{\text{hyb-d}}$

T := empty assoc. array

QUERY(x):
  p := first $d-1$ of x
  if $T[p]$ undefined:
    $T[p] \leftarrow \{0,1\}^\lambda$
  $v := T[p]$
  for $i = d$ to *in*:
    if $x_i = 0$ then $v := G_L(v)$
    if $x_i = 1$ then $v := G_R(v)$
  return v

$\Rightarrow$

T := empty assoc. array

LOOKUP(x):
  p := first $d-1$ bits of x
  if $T[p]$ undefined:
    $T[p] \leftarrow \{0,1\}^\lambda$
    $T[p\|0] := G_L(T[p])$
    $T[p\|1] := G_R(T[p])$
  p' := first d bits of x
  $v := T[p']$
  for $i = d+1$ to *in*:
    if $x_i = 0$ then $v := G_L(v)$
    if $x_i = 1$ then $v := G_R(v)$
  return v

It computes the labels of both of p' s children, even though only one is on the path to x.

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof To finish the proof, we show that $\mathcal{L}_{\text{hyb}-(d-1)}$ and $\mathcal{L}_{\text{hyb}-d}$ are indistinguishable.

T := empty assoc. array

$\underline{\text{LOOKUP}(x)\text{:}}$
$\quad p :=$ first $d-1$ bits of x
$\quad$ if $T[p]$ undefined:
$\quad\quad T[p] \leftarrow \{0,1\}^\lambda$
$\quad\quad T[p\|0] := G_L(T[p])$
$\quad\quad T[p\|1] := G_R(T[p])$
$\quad p' :=$ first d bits of x
$\quad v := T[p']$
$\quad$ for $i = d+1$ to in:
$\quad\quad$ if $x_i = 0$ then $v := G_L(v)$
$\quad\quad$ if $x_i = 1$ then $v := G_R(v)$
$\quad$ return v

$\Rightarrow$

T := empty assoc. array

$\underline{\text{LOOKUP}(x)\text{:}}$
$\quad p :=$ first $d-1$ bits of x
$\quad$ if $T[p]$ undefined:
$\quad\quad T[p\|0] \| T[p\|1] := \text{QUERY}()$
$\quad p' :=$ first d bits of x
$\quad v := T[p']$
$\quad$ for $i = d+1$ to in:
$\quad\quad$ if $x_i = 0$ then $v := G_L(v)$
$\quad\quad$ if $x_i = 1$ then $v := G_R(v)$
$\quad$ return v

$\diamond$

$\mathcal{L}^G_{\text{prg-real}}$

$\underline{\text{QUERY}()\text{:}}$
$\quad s \leftarrow \{0,1\}^\lambda$
$\quad$ return $G(s)$

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof To finish the proof, we show that $\mathcal{L}_{\text{hyb}-(d-1)}$ and $\mathcal{L}_{\text{hyb}-d}$ are indistinguishable.

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x):}$

$\quad p :=$ first $d-1$ bits of x
$\quad$ if $T[p]$ undefined:
$\quad\quad T[p\|0] \| T[p\|1] := \text{QUERY}()$
$\quad p' :=$ first d bits of x
$\quad v := T[p']$
$\quad$ for $i = d+1$ to *in*:
$\quad\quad$ if $x_i = 0$ then $v := G_L(v)$
$\quad\quad$ if $x_i = 1$ then $v := G_R(v)$
$\quad$ return v

$\diamond$

$\mathcal{L}^G_{\text{prg-real}}$

$\underline{\text{QUERY}():}$

$\quad s \leftarrow \{0,1\}^\lambda$
$\quad$ return $G(s)$

$\Rightarrow$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x):}$

$\quad p :=$ first $d-1$ bits of x
$\quad$ if $T[p]$ undefined:
$\quad\quad T[p\|0] \| T[p\|1] := \text{QUERY}()$
$\quad p' :=$ first d bits of x
$\quad v := T[p']$
$\quad$ for $i = d+1$ to *in*:
$\quad\quad$ if $x_i = 0$ then $v := G_L(v)$
$\quad\quad$ if $x_i = 1$ then $v := G_R(v)$
$\quad$ return v

$\diamond$

$\mathcal{L}^G_{\text{prg-rand}}$

$\underline{\text{QUERY}():}$

$\quad r \leftarrow \{0,1\}^{2\lambda}$
$\quad$ return r

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof To finish the proof, we show that $\mathcal{L}_{\text{hyb}-(d-1)}$ and $\mathcal{L}_{\text{hyb}-d}$ are indistinguishable.

$T :=$ empty assoc. array

$\underline{\textsc{lookup}(x)}$:
  $p :=$ first $d-1$ bits of x
  if $T[p]$ undefined:
    $T[p\|0] \| T[p\|1] := \textsc{query}()$
  $p' :=$ first d bits of x
  $v := T[p']$
  for $i = d+1$ to in:
    if $x_i = 0$ then $v := G_L(v)$
    if $x_i = 1$ then $v := G_R(v)$
  return v

$\diamond$

$\mathcal{L}^{G}_{\text{prg-rand}}$

$\underline{\textsc{query}()}$:
  $r \leftarrow \{0,1\}^{2\lambda}$
  return r

$\Rightarrow$

$T :=$ empty assoc. array

$\underline{\textsc{lookup}(x)}$:
  $p :=$ first $d-1$ bits of x
  if $T[p]$ undefined:
    $T[p\|0] \leftarrow \{0,1\}^{\lambda}$
    $T[p\|1] \leftarrow \{0,1\}^{\lambda}$
  $p' :=$ first d bits of x
  $v := T[p']$
  for $i = d+1$ to in:
    if $x_i = 0$ then $v := G_L(v)$
    if $x_i = 1$ then $v := G_R(v)$
  return v

Constructing a PRF from a PRG

Claim If G is a secure PRG, then the construction is a secure PRF.

proof To finish the proof, we show that $\mathcal{L}_{\text{hyb}-(d-1)}$ and $\mathcal{L}_{\text{hyb}-d}$ are indistinguishable.

```
T := empty assoc. array

LOOKUP(x):
  p := first  d − 1  bits of x
  if T[p] undefined:
    T[p‖0] ← {0,1}^λ
    T[p‖1] ← {0,1}^λ
  p′ := first  d   bits of x
  v := T[p′]
  for i =  d  + 1 to in:
    if x_i = 0 then v := G_L(v)
    if x_i = 1 then v := G_R(v)
  return v
```

Since we sample it uniformly, it doesn’t matter when (or if) that extra value is sampled. Hence, this library has identical behavior to $\mathcal{L}_{\text{hyb}-d}$.

$\mathcal{L}_{\text{hyb-d}}$

```
T := empty assoc. array

QUERY(x):
  p := first  d  bits of x
  if T[p] undefined:
    T[p] ← {0,1}^λ
  v := T[p]
  for i =  d + 1  to in:
    if x_i = 0 then v := G_L(v)
    if x_i = 1 then v := G_R(v)
  return v
```

Block Ciphers (Pseudorandom Permutations)

- Even in the case of $in = out$, A function from $\{0,1\}^{in}$ to $\{0,1\}^{out}$ chosen at random is unlikely to have an inverse, therefore a PRF instantiated with a random key is unlikely to have an inverse.
- A pseudorandom permutation (PRP) — also called a block cipher — is essentially a PRF that is guaranteed to be invertible for every choice of seed.

Block Ciphers (Pseudorandom Permutations)

Let $F\colon \{0,1\}^\lambda \times \{0,1\}^{blen} \to \{0,1\}^{blen}$ be a deterministic function. We refer to $blen$ as the blocklength of F and any element of $\{0,1\}^{blen}$ as a block. We call F a secure pseudorandom permutation (PRP) (block cipher) if the following two conditions hold:

- **Invertible given *k***: There is a function $F^{-1} : \{0,1\}^\lambda \times \{0,1\}^{blen} \to \{0,1\}^{blen}$ satisfying $F^{-1}(k, F(k,x)) = x$, for all $k \in \{0,1\}^\lambda$ and all $x \in \{0,1\}^{blen}$.
- **Security**: $\mathcal{L}^F_{\text{prp-real}} \approx \mathcal{L}^F_{\text{prp-rand}}$, where:

$\mathcal{L}^F_{\text{prp-real}}$

$k \leftarrow \{0,1\}^\lambda$

$\underline{\text{LOOKUP}(x \in \{0,1\}^{blen}):}$
  return $F(k,x)$

$\mathcal{L}^F_{\text{prp-rand}}$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x \in \{0,1\}^{blen}):}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
  return $T[x]$

"T .values" refers to $\{v \mid \exists x : T[x] = v\}$

Switching Lemma (PRPs are PRFs, Too!)

Lemma If $in = out = blen = \lambda$, $\mathcal{L}_{\text{prp-rand}} \approx \mathcal{L}_{\text{prf-rand}}$.

$\mathcal{L}^F_{\text{prf-rand}}$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x \in \{0,1\}^{in}):}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{out}$
  return $T[x]$

$\mathcal{L}^F_{\text{prp-rand}}$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x \in \{0,1\}^{blen}):}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
  return $T[x]$

Switching Lemma (PRPs are PRFs, Too!)

Lemma If $in = out = blen = \lambda$, $\mathcal{L}_{\text{prp-rand}} \approx \mathcal{L}_{\text{prf-rand}}$.

proof Recall the replacement-sampling lemma

$\mathcal{L}_{\text{samp-L}}$
$\underline{\text{SAMP}()\text{:}}$ $\quad r \leftarrow \{0,1\}^\lambda$ $\quad$return r

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$ $\underline{\text{SAMP}()\text{:}}$ $\quad r \leftarrow \{0,1\}^\lambda \setminus R$ $\quad R := R \cup \{r\}$ $\quad$return r

Switching Lemma (PRPs are PRFs, Too!)

Lemma If $in = out = blen = \lambda$, $\mathcal{L}_{\text{prp-rand}} \approx \mathcal{L}_{\text{prf-rand}}$.

proof Recall the replacement-sampling lemma

Now consider the following library

$\mathcal{L}^*$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x \in \{0,1\}^\lambda):}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \text{SAMP}()$
  return $T[x]$

$\diamond$

$\mathcal{L}_{\text{samp-L}}$

$\underline{\text{SAMP}():}$
  $r \leftarrow \{0,1\}^\lambda$
  return r

$\mathcal{L}^F_{\text{prf-rand}}$

$T :=$ empty assoc. array

$\underline{\text{LOOKUP}(x \in \{0,1\}^{in}):}$
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^{out}$
  return $T[x]$

Switching Lemma (PRPs are PRFs, Too!)

Lemma If $in = out = blen = \lambda$, $\mathcal{L}_{\text{prp-rand}} \approx \mathcal{L}_{\text{prf-rand}}$.

proof Recall the replacement-sampling lemma

Now consider the following library

$\mathcal{L}^*$
$T :=$ empty assoc. array
<u>LOOKUP($x \in \{0,1\}^\lambda$):</u>
if $T[x]$ undefined:
$T[x] \leftarrow$ SAMP()
return $T[x]$

$\diamond$

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
<u>SAMP():</u>
$r \leftarrow \{0,1\}^\lambda \setminus R$
$R := R \cup \{r\}$
return r

$\mathcal{L}^F_{\text{prp-rand}}$
$T :=$ empty assoc. array
<u>LOOKUP($x \in \{0,1\}^{blen}$):</u>
if $T[x]$ undefined:
$T[x] \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
return $T[x]$

Switching Lemma (PRPs are PRFs, Too!)

Lemma If $in = out = blen = \lambda$, $\mathcal{L}_{\text{prp-rand}} \approx \mathcal{L}_{\text{prf-rand}}$.

proof Recall the replacement-sampling lemma

$$\mathcal{L}_{\text{prf-rand}} \equiv \mathcal{L}^* \diamond \mathcal{L}_{\text{samp-L}} \approx \mathcal{L}^* \diamond \mathcal{L}_{\text{samp-R}} \equiv \mathcal{L}_{\text{prp-rand}}$$

Corollary Let $F: \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a secure PRP (with $blen = \lambda$). Then F is also a secure PRF.

proof $\mathcal{L}^F_{\text{prf-real}} \equiv \mathcal{L}^F_{\text{prp-real}} \approx \mathcal{L}^F_{\text{prp-rand}} \approx \mathcal{L}^F_{\text{prf-rand}}$

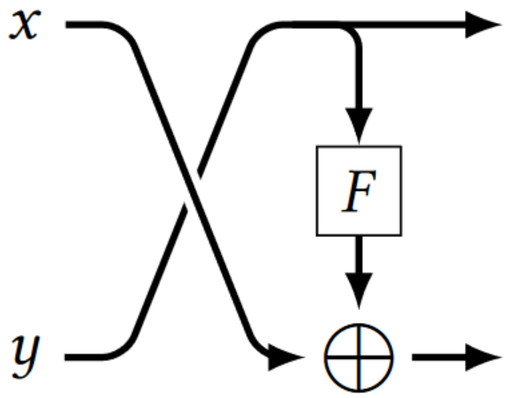
Constructing a PRP from a PRF

- Feistel construction: convert a not-necessarily-invertible function $F : \{0,1\}^n \to \{0,1\}^n$ into an invertible function $F^* : \{0,1\}^{2n} \to \{0,1\}^{2n}$. The function F^* is called the Feistel round with round function F

$\underline{F^*(x \| y)}$:

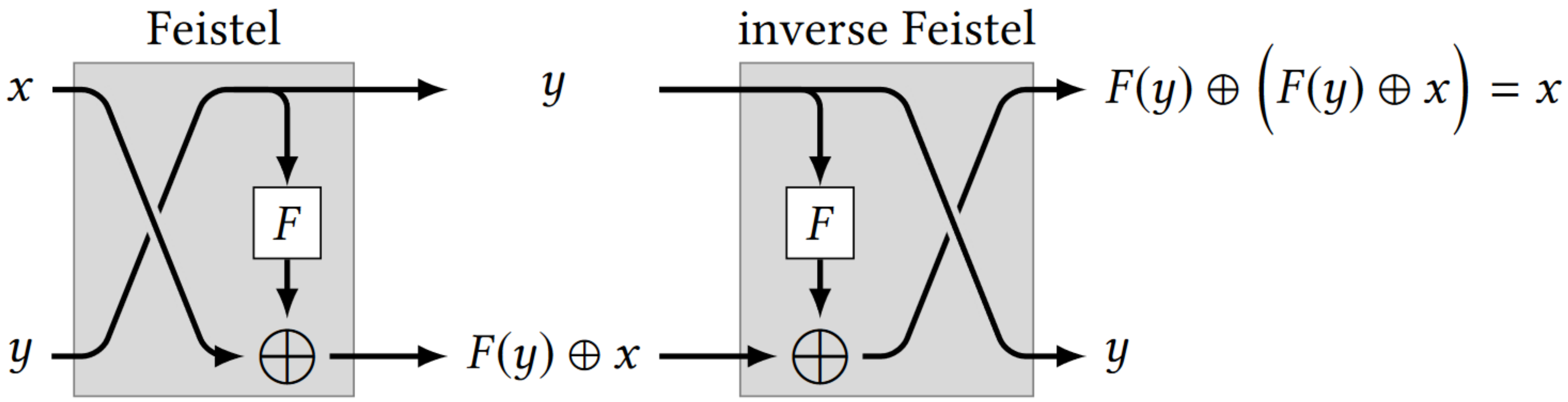
    // each of x, y are n bits

    return $y \| (F(y) \oplus x)$



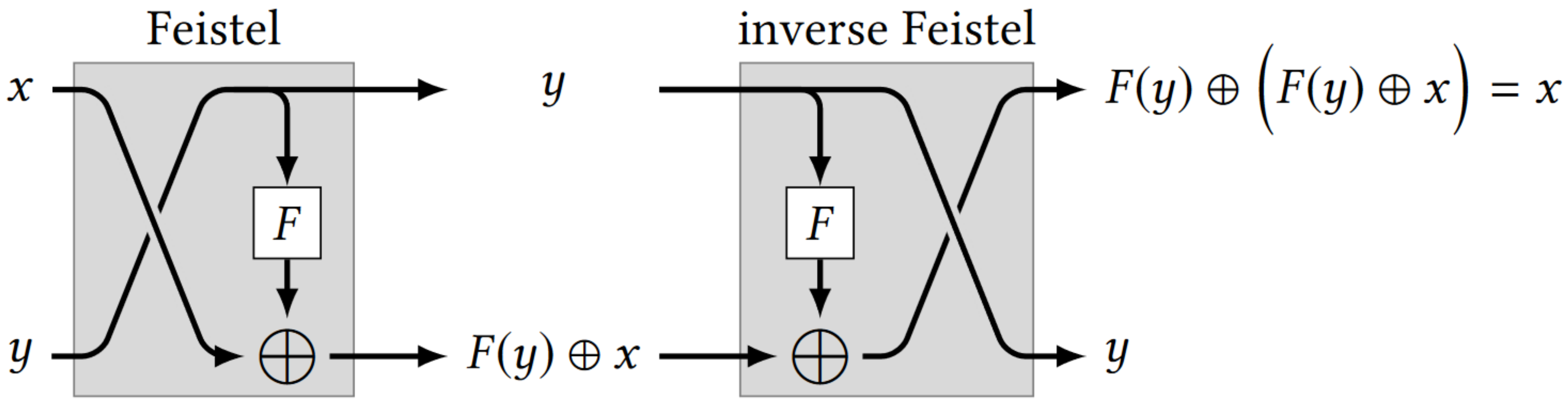
Constructing a PRP from a PRF

- Feistel construction: convert a not-necessarily-invertible function $F : \{0,1\}^n \to \{0,1\}^n$ into an invertible function $F^* : \{0,1\}^{2n} \to \{0,1\}^{2n}$. The function F^* is called the Feistel round with round function F



Feistel Construction

- If $F(y) = 0^n$, then $F^*(x\|y) \ = \ y\|(F\ (y) \oplus x) = y\|(0^n \oplus x) = y\|x$
- If $F(y) = y$, then $F^*(x\|y) = y\|(F(y) \oplus x) = y\|(y \oplus x)$, also invertible since we can solve for x.

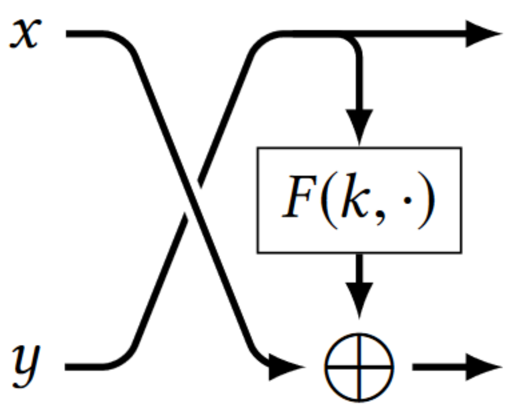


Feistel Cipher

- The output of F^* contains half of its input, making it quite trivial to break the PRP-security of F^*.
- We can avoid this trivial attack by performing several Feistel rounds in succession, resulting in a construction called a Feistel cipher.
- At each round, we can even use a different key to the round function. (key schedule)

$\underline{F^*(k, x \| y)}$:

$\quad$ return $y \| (F(k, y) \oplus x)$



Feistel Cipher

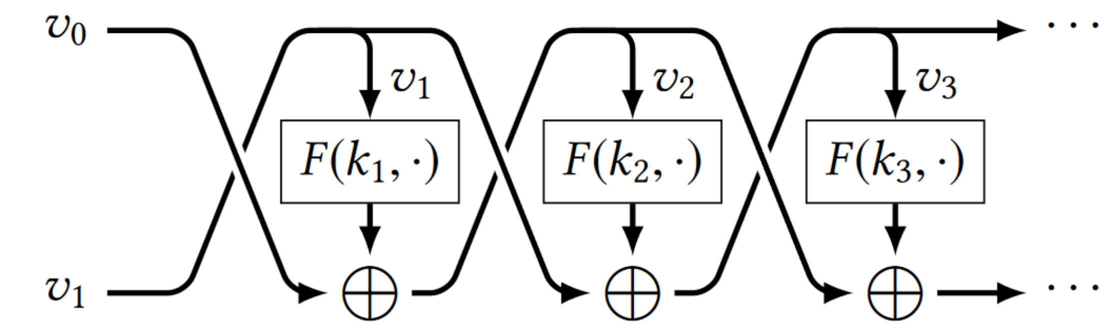
- We can avoid this trivial attack by performing several Feistel rounds in succession, resulting in a construction called a Feistel cipher.
- At each round, we can even use a different key to the round function. (key schedule)

$\mathbb{F}_r\big((k_1,\ldots,k_r), v_0 \| v_1\big)$:

for $i = 1$ to r:

$\quad v_{i+1} := F(k_i, v_i) \oplus v_{i-1}$

return $v_r \| v_{r+1}$

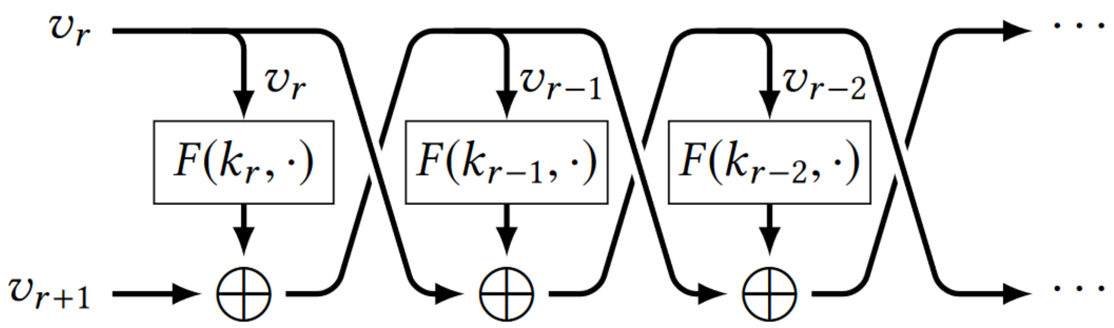


$\mathbb{F}_r^{-1}\big((k_1,\ldots,k_r), v_r \| v_{r+1}\big)$:

for $i = r$ downto 1:

$\quad v_{i-1} := F(k_i, v_i) \oplus v_{i+1}$

return $v_0 \| v_1$



Feistel Cipher

- **Theorem** (Luby-Rackoff) If $F\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a secure PRF, then the 3-round Feistel cipher is a secure PRP.

$\underline{\mathbb{F}_r\big((k_1,\ldots,k_r), v_0\|v_1\big):}$

for $i = 1$ to r:

$\quad v_{i+1} := F(k_i, v_i) \oplus v_{i-1}$

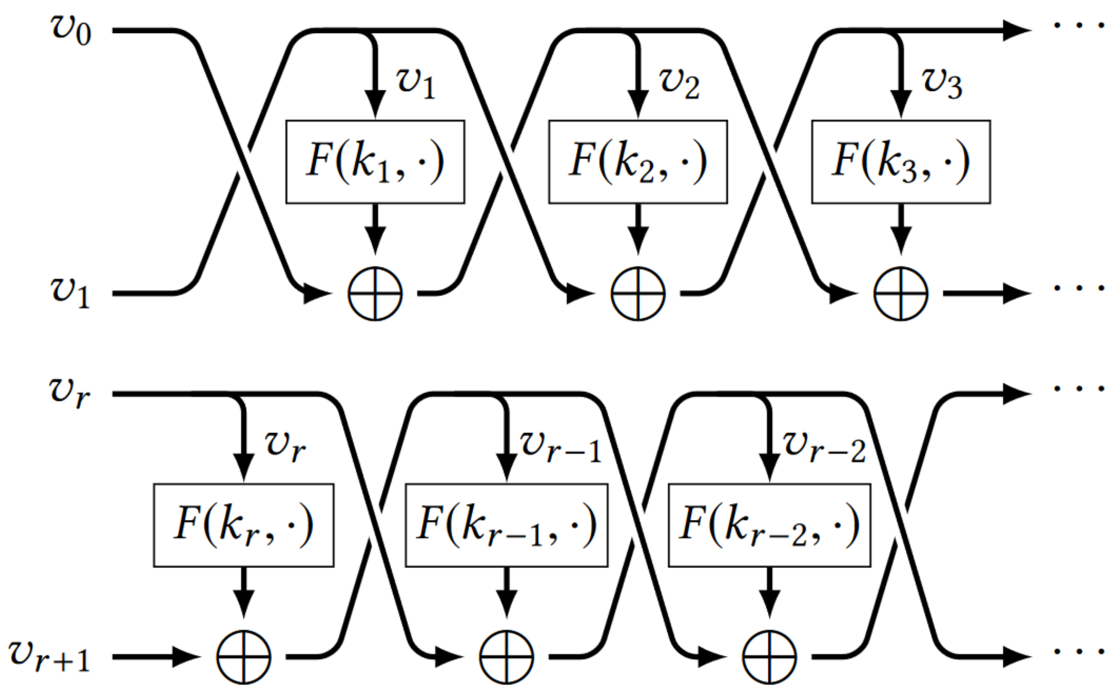
return $v_r\|v_{r+1}$

$\underline{\mathbb{F}_r^{-1}\big((k_1,\ldots,k_r), v_r\|v_{r+1}\big):}$

for $i = r$ downto 1:

$\quad v_{i-1} := F(k_i, v_i) \oplus v_{i+1}$

return $v_0\|v_1$



PRFs and Block Ciphers in Practice

- We currently have no proof that any secure PRP exists.
- Advanced Encryption Standard (AES) block cipher demonstrates the cryptographic community's best efforts at instilling such confidence.

- As we have seen, once you have access to a good block cipher, it can be used directly also as a secure PRF, and it can be used to construct a simple PRG.
  - The PRF-security and PRG-security of these constructions is guaranteed to be as good as the PRP-security of AES.

Strong Pseudorandom Permutations

- The PRP security definition only guarantees a security property for F and not its inverse.
- It is possible to construct F which is a secure PRP, whose inverse F^{-1} is not a secure PRP!

$$E_k(x) = \begin{cases} 0 & \text{if } x = k \\ \mathrm{AES}_k(k) & \text{if } x = \mathrm{AES}_k^{-1}(0) \\ \mathrm{AES}_k(x) & \text{otherwise.} \end{cases}$$

Strong Pseudorandom Permutations

Definition Let $F\colon \{0,1\}^{\lambda} \times \{0,1\}^{blen} \to \{0,1\}^{blen}$ be a deterministic function. We say that F is a secure strong pseudorandom permutation (SPRP) if $\mathcal{L}^F_{\text{sprp-real}} \approx \mathcal{L}^F_{\text{sprp-rand}}$.

$\mathcal{L}^F_{\text{sprp-real}}$

$k \leftarrow \{0,1\}^{\lambda}$

$\underline{\text{LOOKUP}(x \in \{0,1\}^{blen}):}$
  return $F(k,x)$

$\underline{\text{INVLOOKUP}(y \in \{0,1\}^{blen}):}$
  return $F^{-1}(k,y)$

$\mathcal{L}^F_{\text{sprp-rand}}$

T, T_{inv} := empty assoc. arrays

$\underline{\text{LOOKUP}(x \in \{0,1\}^{blen}):}$
  if $T[x]$ undefined:
    $y \leftarrow \{0,1\}^{blen} \setminus T.\text{values}$
    $T[x] := y; \quad T_{inv}[y] := x$
  return $T[x]$

$\underline{\text{INVLOOKUP}(y \in \{0,1\}^{blen}):}$
  if $T_{inv}[y]$ undefined:
    $x \leftarrow \{0,1\}^{blen} \setminus T_{inv}.\text{values}$
    $T_{inv}[y] := x; \quad T[x] := y$
  return $T_{inv}[y]$

Strong Pseudorandom Permutations

Definition Let $F: \{0,1\}^\lambda \times \{0,1\}^{blen} \to \{0,1\}^{blen}$ be a deterministic function. We say that F is a secure strong pseudorandom permutation (SPRP) if $\mathcal{L}^F_{\text{sprp-real}} \approx \mathcal{L}^F_{\text{sprp-rand}}$.

Theorem (Luby-Rackoff) If $F: \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a secure PRF, then the 4-round Feistel cipher is a secure SPRP.