Advanced Cryptography

（Provable Security）

Yi LIU

Asymptotic Bounds on the Birthday Probability

We are most interested in the case where q is relatively small compared to N (e.g., when q is a polynomial function of λ but N is exponential).

Lemma if $q \leq \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \leq \text{BirthdayProb}(q, N) \leq \frac{q(q-1)}{2N}$. This means that $\text{BirthdayProb}(q, N) = \Theta\left(\frac{q^2}{N}\right)$

Asymptotic Bounds on the Birthday Probability

Lemma if $q \leq \sqrt{2N}$, then $0.632\,\frac{q(q-1)}{2N} \leq \mathrm{BirthdayProb}(q,N) \leq \frac{q(q-1)}{2N}$. This means that $\mathrm{BirthdayProb}(q,N) = \Theta(\frac{q^2}{N})$

Proof

Upper bound:

For positive x and y: $(1-x)(1-y) = 1-(x+y)+xy \geq 1-(x+y)$

More generally, when all x_i are positive, $\prod_i (1-x_i) \geq 1-\sum_i x_i$. Hence,

$$1-\prod_i (1-x_i) \leq 1-\left(1-\sum_i x_i\right) = \sum_i x_i$$

Then $\mathrm{BirthdayProb}(q,N) = 1-\prod_{i=1}^{q-1}\left(1-\frac{i}{N}\right) \leq \sum_{i=1}^{q-1}\frac{i}{N} = \frac{\sum_{i=1}^{q-1} i}{N} = \frac{q(q-1)}{2N}$

Asymptotic Bounds on the Birthday Probability

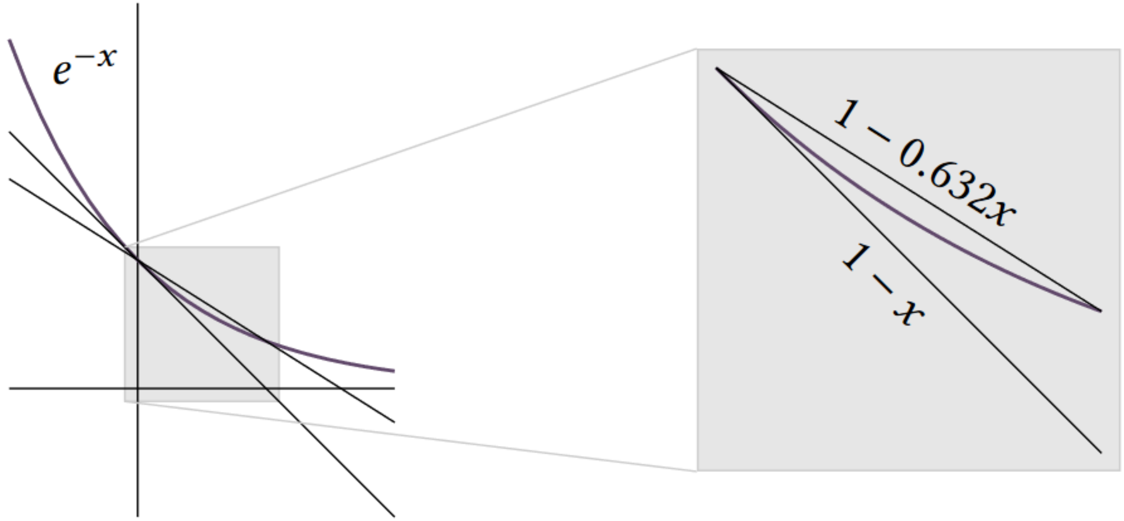
Lemma if $q \leq \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \leq \text{BirthdayProb}(q, N) \leq \frac{q(q-1)}{2N}$. This means that $\text{BirthdayProb}(q, N) = \Theta(\frac{q^2}{N})$

Proof

Lower bound: Use the fact that when $0 \leq x \leq 1$,

$$1 - x \leq e^{-x} \leq 1 - 0.632x$$

0.632 from $1 - \frac{1}{e} = 0.632112$ ...



Asymptotic Bounds on the Birthday Probability

Lemma if $q \le \sqrt{2N}$, then $0.632\,\frac{q(q-1)}{2N} \le \mathrm{BirthdayProb}(q, N) \le \frac{q(q-1)}{2N}$. This means that $\mathrm{BirthdayProb}(q, N) = \Theta(\frac{q^2}{N})$

Proof

Lower bound: Use the fact that when $0 \le x \le 1$,

$$1 - x \le e^{-x} \le 1 - 0.632x$$

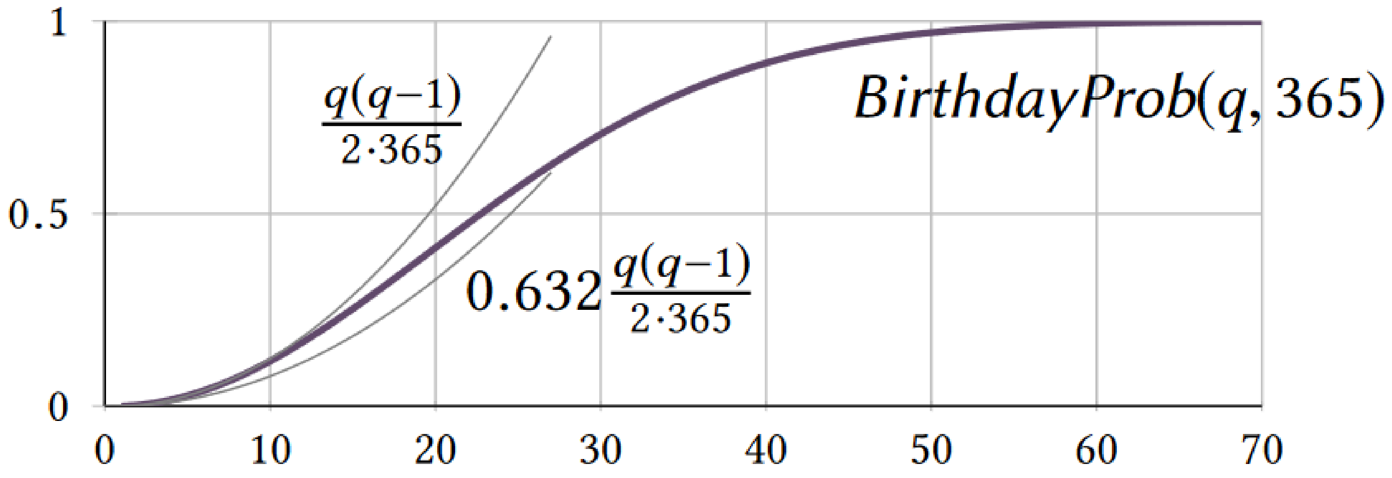
We have $\prod_{i=1}^{q-1}(1 - \frac{i}{N}) \le \prod_{i=1}^{q-1} e^{-\frac{i}{N}} = e^{-\sum_{i=1}^{q-1}\frac{i}{N}} = e^{-\frac{q(q-1)}{2N}} \le 1 - 0.632\,\frac{q(q-1)}{2N}$

The last inequality uses the fact $q \le \sqrt{2N}$ and thus $\frac{q(q-1)}{2N} \le 1$.

Hence, $\mathrm{BirthdayProb}(q, N) = 1 - \prod_{i=1}^{q-1}\left(1 - \frac{i}{N}\right) \ge 1 - \left(1 - 0.632\,\frac{q(q-1)}{2N}\right) = 0.632\,\frac{q(q-1)}{2N}$

Asymptotic Bounds on the Birthday Probability

Lemma if $q \leq \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \leq \text{BirthdayProb}(q, N) \leq \frac{q(q-1)}{2N}$. This means that $\text{BirthdayProb}(q, N) = \Theta(\frac{q^2}{N})$



The Birthday Problem in Terms of Indistinguishable Libraries

- **with replacement** vs. **without replacement**

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

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

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

$R := \emptyset$

$\underline{\textsc{samp}():}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus R$
$\quad R := R \cup \{r\}$
$\quad$ return r

- A natural way to distinguish them: call SAMP many times
  - If ever see a repeated output, then must be linked to $\mathcal{L}_{\mathtt{samp-L}}$. After some number of calls to SAMP, if still don't see any repeated outputs, you might eventually stop and guess that you are linked to $\mathcal{L}_{\mathtt{samp-R}}$.

The Birthday Problem in Terms of Indistinguishable Libraries

- A natural way to distinguish them: call SAMP many times
  - If ever see a repeated output, then must be linked to $\mathcal{L}_{\text{samp-L}}$. After some number of calls to SAMP, if still don't see any repeated outputs, you might eventually stop and guess that you are linked to $\mathcal{L}_{\text{samp-R}}$.
  - Return 1 if see a repeated value.
    - When linked to $\mathcal{L}_{\text{samp-R}}$, can never return 1
    - When linked to $\mathcal{L}_{\text{samp-L}}$, return 1 with prob. $\text{BirthdayProb}(q, 2^\lambda)$
    - The advantage is $\text{BirthdayProb}(q, 2^\lambda) = \Theta(\frac{q^2}{2^\lambda})$, which is negligible.

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

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

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

$R := \emptyset$

SAMP():
- $r \leftarrow \{0, 1\}^\lambda \setminus R$
- $R := R \cup \{r\}$
- return r

The Birthday Problem in Terms of Indistinguishable Libraries

Lemma For all calling programs $\mathcal{A}$ that make q queries to the SAMP subroutine, the advantage of $\mathcal{A}$ in distinguishing the libraries is at most $\mathrm{BirthdayProb}(q, 2^{\lambda})$.

In particular, when $\mathcal{A}$ is polynomial-time (in λ), q grows as a polynomial in the security parameter. Hence, $\mathcal{A}$ has negligible advantage. We have $\mathcal{L}_{\text{samp}-\text{L}} \approx \mathcal{L}_{\text{samp}-R}$

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

<u>SAMP()</u>:
  $r \leftarrow \{0,1\}^{\lambda}$
  return r

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

$R := \emptyset$

<u>SAMP()</u>:
  $r \leftarrow \{0,1\}^{\lambda} \setminus R$
  $R := R \cup \{r\}$
  return r

The Birthday Problem in Terms of Indistinguishable Libraries

Lemma For all calling programs $\mathcal{A}$ that make q queries to the SAMP subroutine, the advantage of $\mathcal{A}$ in distinguishing the libraries is at most $\text{BirthdayProb}(q, 2^\lambda)$.

In particular, when $\mathcal{A}$ is polynomial-time (in λ), q grows as a polynomial in the security parameter. Hence, $\mathcal{A}$ has negligible advantage. We have $\mathcal{L}_{\text{samp-L}} \approx \mathcal{L}_{\text{samp-}R}$

proof

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

$\mathcal{L}_{\text{hyb-}R} \equiv \mathcal{L}_{\text{samp-R}}$

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

$R := \emptyset$
bad := 0

SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  if $r \in R$ then:
    bad := 1
  $R := R \cup \{r\}$
  return r

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

$R := \emptyset$
bad := 0

SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  if $r \in R$ then:
    bad := 1
    $r \leftarrow \{0,1\}^\lambda \setminus R$
  $R := R \cup \{r\}$
  return r

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

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

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

$R := \emptyset$

SAMP():
  $r \leftarrow \{0,1\}^\lambda \setminus R$
  $R := R \cup \{r\}$
  return r

The Birthday Problem in Terms of Indistinguishable Libraries

Lemma For all calling programs $\mathcal{A}$ that make q queries to the SAMP subroutine, the advantage of $\mathcal{A}$ in distinguishing the libraries is at most $\text{BirthdayProb}(q, 2^\lambda)$.

In particular, when $\mathcal{A}$ is polynomial-time (in λ), q grows as a polynomial in the security parameter. Hence, $\mathcal{A}$ has negligible advantage. We have $\mathcal{L}_{\text{samp-L}} \approx \mathcal{L}_{\text{samp-}R}$

proof

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

$\mathcal{L}_{\text{hyb-}R} \equiv \mathcal{L}_{\text{samp-R}}$

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

$R := \emptyset$
bad := 0
 	
SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  if $r \in R$ then:
    bad := 1
 	
  $R := R \cup \{r\}$
  return r

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

$R := \emptyset$
bad := 0
 	
SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  if $r \in R$ then:
    bad := 1
    $r \leftarrow \{0,1\}^\lambda \setminus R$
  $R := R \cup \{r\}$
  return r

We can bound the advantage of the calling program:

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{samp-L}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{samp-R}} \Rightarrow 1]\right| = \left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-L}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-R}} \Rightarrow 1]\right|$$

$$\leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-L}} \text{ sets bad} = 1] = \text{BirthdayProb}(q, 2^\lambda)$$

The Birthday Problem in Terms of Indistinguishable Libraries

- Birthday problem in terms of indistinguishable libraries makes it a useful tool in future security proofs.
  - We can replace a uniform sampling step with a sampling-without-replacement step.
  - This change has only a negligible effect, but now the rest of the proof can take advantage of the fact that samples are never repeated.
- If a security proof does use the indistinguishability of the birthday libraries, the scheme can likely be broken when a uniformly sampled value is repeated.
  - Since this becomes inevitable as the number of samples approaches $\sqrt{2^{\lambda+1}} \sim 2^{\frac{\lambda}{2}}$, it means the scheme only offers $\lambda/2$ bits of security. When a scheme has this property, we say that it has birthday bound security.

$$0.632\frac{q(q-1)}{2N} \leqslant BirthdayProb(q, N) \leqslant \frac{q(q-1)}{2N}.$$

A Generalization

- The following two libraries are indistinguishable, provided that the argument $\mathcal{R}$ to SAMP is passed as an explicit list of items (i.e., take the time to "write down" the elements of $\mathcal{R}$).

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

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

- Suppose the calling program makes q calls to SAMP, and in the ith call it uses an argument $\mathcal{R}$ with n_i items. Then the advantage of the calling program is at most:

$$1 - \prod_{i=1}^{q} \left(1 - \frac{n_i}{2^\lambda}\right)$$

The birthday scenario: $n_i = i - 1$

Pseudorandom Generators

Recall One-Time Pad

- One-time pad
  - KeyGen: $k \leftarrow \{0,1\}^{\lambda}$
  - $\mathrm{Enc}(k, m)$: $m \oplus k$
  - $\mathrm{Dec}(k, c)$: $k \oplus c$
- To encrypt $m \in \{0,1\}^{2\lambda}$
- If we have a function $G: \{0,1\}^{\lambda} \rightarrow \{0,1\}^{2\lambda}$
  - $\mathrm{Enc}(k, m) = G(k) \oplus m$
  - If the output of G is “close enough” to uniform?

Pseudorandom Generators

- A pseudorandom generator (PRG) is a deterministic function G whose outputs are longer than its inputs.
- When the input to G is chosen uniformly at random, it induces a certain distribution over the possible output.
  - This output distribution cannot be uniform. However, the distribution is pseudorandom if it is indistinguishable from the uniform distribution.

Pseudorandom Generators

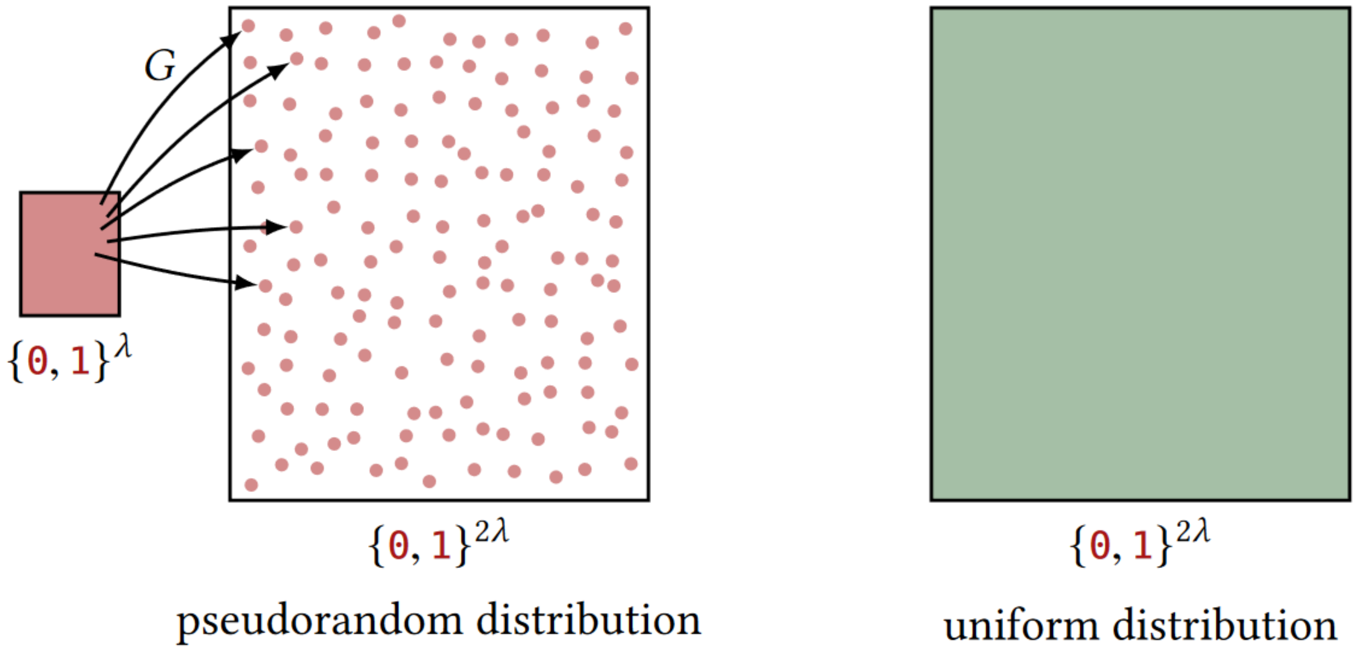
Definition Let $G: \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be a deterministic function with $\ell > 0$. We say that G is a secure pseudorandom generator (PRG) if $\mathcal{L}^G_{\text{prg-real}} \approx \mathcal{L}^G_{\text{prg-rand}}$, where:

$\mathcal{L}^G_{\text{prg-real}}$	$\mathcal{L}^G_{\text{prg-rand}}$
QUERY():	QUERY():
$s \leftarrow \{0,1\}^\lambda$	$r \leftarrow \{0,1\}^{\lambda+\ell}$
return $G(s)$	return r

The value ℓ is called the stretch of the PRG. The input to the PRG is called a seed.

Pseudorandom Generators

For a length-doubling ($\ell = \lambda$) PRG (not drawn to scale)



Pseudorandom Generators

For a length-doubling ($\ell = \lambda$) PRG

- From a relative perspective, the PRG's output distribution is tiny. Out of the $2^{2\lambda}$ strings in $\{0,1\}^{2\lambda}$, only 2^{λ} are possible outputs of G. These strings make up a $2^{\lambda}/2^{2\lambda} = 1/2^{\lambda}$ fraction of $\{0,1\}^{2\lambda}$ — a negligible fraction!
  - A polynomial time calling program cannot notice this
- From a absolute perspective, the PRG's output distribution is huge. There are 2^{λ} possible outputs of G, which is an exponential amount!

Pseudorandom Generators

- A PRG is indeed an algorithm into which you can feed any string you like. However, security is only guaranteed when the PRG is being used exactly as described in the security libraries
  - In particular, when the seed is chosen uniformly/secretly and not used for anything else.

$\mathcal{L}^G_{\text{prg-real}}$	$\mathcal{L}^G_{\text{prg-rand}}$
$\underline{\text{QUERY}():}$ $s \leftarrow \{0,1\}^\lambda$ return $G(s)$	$\underline{\text{QUERY}():}$ $r \leftarrow \{0,1\}^{\lambda+\ell}$ return r

Pseudorandom Generators in Practice

- We have no examples of secure PRGs!
- If it were possible to prove that some function G is a secure PRG, it would resolve the famous P vs NP problem.
- Cryptographic research can offer are candidate PRGs, which are conjectured to be secure.
- Modern crypto: provable security claims are conditional — if X is secure then Y is secure.
- When you really need a PRG in practice, you would actually use a PRG that is built from something called a block cipher.
  - A block cipher is conceptually more complicated than a PRG, and can even be built from a PRG (in principle).
  - More useful object, so implemented with specialized CPU instructions in most CPUs

A Construction

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

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

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

QUERY():
$r \leftarrow \{0,1\}^{\lambda+\ell}$
return r

- If the input s is random, then $s\|s$ is also random, too, right?

$G(s)$:
return $s\|s$

- No!

$\mathcal{A}$

$x\|y :=$ QUERY()
return $x \stackrel{?}{=} y$

A Construction

$$\mathcal{L}^G_{\text{prg-real}}$$
$\text{QUERY}()$:
  $s \leftarrow \{0,1\}^\lambda$
  return $G(s)$

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

- If the input s is random, then $s\|s$ is also random, too, right?

$G(s)$:
  return $s\|s$

- No!

$$\mathcal{A}$$
$x\|y := \text{QUERY}()$
return $x \stackrel{?}{=} y$

$\diamond$

$$\mathcal{L}^G_{\text{prg-real}}$$
$\text{QUERY}()$:
  $s \leftarrow \{0,1\}^\lambda$
  return $s\|s$

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

A Construction

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

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

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

QUERY():
  $r \leftarrow \{0,1\}^{\lambda+\ell}$
  return r

- If the input s is random, then $s \| s$ is also random, too, right?

$G(s)$:
  return $s \| s$

- No!

$\mathcal{A}$

$x \| y :=$ QUERY()
return $x \stackrel{?}{=} y$

$\diamond$

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

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

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

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

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

This scheme will not have (perfect) one-time secrecy. That is, encryptions of m_L and m_R will not be identically distributed in general. However, the distributions will be indistinguishable if G is a secure PRG.

(Computational) One-Time Secrecy

Definition Let Σ be an encryption scheme, and let $\mathcal{L}^{\Sigma}_{\text{ots}-\text{L}}$ and $\mathcal{L}^{\Sigma}_{\text{ots}-R}$ be defined as below. Then Σ has (computational) one-time secrecy if $\mathcal{L}^{\Sigma}_{\text{ots}-\text{L}} \approx \mathcal{L}^{\Sigma}_{\text{ots}-R}$. That is, if for all polynomial-time distinguishers $\mathcal{A}$, we have $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots}-\text{L}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots}-\text{R}} \Rightarrow 1]$.

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$
$\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M})$:
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m_L)$
return c

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
$\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M})$:
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m_R)$
return c

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme is instantiated using a secure PRG G, then it has computational one-time secrecy.

Pseudo OTP

$\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\mathcal{L}_{\text{ots-L}}^{\text{pOTP}}$		$\mathcal{L}_{\text{ots-L}}^{\text{pOTP}}$
EAVESDROP$(m_L, m_R \in \{0,1\}^{\lambda+\ell})$: $k \leftarrow \{0,1\}^\lambda$ $c := G(k) \oplus m_L$ return c	$\approx$	EAVESDROP$(m_L, m_R \in \{0,1\}^{\lambda+\ell})$: $k \leftarrow \{0,1\}^\lambda$ $c := G(k) \oplus m_R$ return c

Pseudo OTP

$\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\mathcal{L}_{\text{ots-L}}^{\text{pOTP}}$

EAVESDROP$(m_L, m_R \in \{0,1\}^{\lambda+\ell})$:
  $k \leftarrow \{0,1\}^\lambda$
  $c := G(k) \oplus m_L$
  return c

$\Rightarrow$

EAVESDROP(m_L, m_R):
  $z \leftarrow$ QUERY()
  $c := z \oplus m_L$
  return c

$\diamond$

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

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

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

EAVESDROP(m_L, m_R):		$\mathcal{L}^{G}_{\text{prg-real}}$		EAVESDROP(m_L, m_R):		$\mathcal{L}^{G}_{\text{prg-rand}}$
$z \leftarrow$ QUERY()	$\diamond$	QUERY():	$\Rightarrow$	$z \leftarrow$ QUERY()	$\diamond$	QUERY():
$c := z \oplus m_L$		$s \leftarrow \{0, 1\}^{\lambda}$		$c := z \oplus m_L$		$r \leftarrow \{0, 1\}^{\lambda+\ell}$
return c		return $G(s)$		return c		return r

Pseudo OTP

$\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

EAVESDROP(m_L, m_R):
$z \leftarrow$ QUERY$()$
$c := z \oplus m_L$
return c

$\diamond$

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

QUERY$()$:
$r \leftarrow \{0,1\}^{\lambda+\ell}$
return r

$\Rightarrow$

$\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$

EAVESDROP(m_L, m_R):
$z \leftarrow \{0,1\}^{\lambda+\ell}$
$c := z \oplus m_L$
return c

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$
EAVESDROP(m_L, m_R):
$z \leftarrow \{0,1\}^{\lambda+\ell}$
$c := z \oplus m_L$
return c

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$
EAVESDROP(m_L, m_R):
$z \leftarrow \{0,1\}^{\lambda+\ell}$
$c := z \oplus m_R$
return c

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$

$\underline{\text{EAVESDROP}(m_L, m_R):}$
$z \leftarrow \{0, 1\}^{\lambda+\ell}$
$c := z \oplus m_R$
return c

$\Rightarrow$

$\underline{\text{EAVESDROP}(m_L, m_R):}$
$z \leftarrow \text{QUERY}()$
$c := z \oplus m_R$
return c

$\diamond$

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

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

Pseudo OTP

$\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\textsc{eavesdrop}(m_L, m_R)$:
$z \leftarrow \textsc{query}()$
$c := z \oplus m_R$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-rand}}$
$\textsc{query}()$:
$r \leftarrow \{0,1\}^{\lambda+\ell}$
return r

$\Rightarrow$

$\textsc{eavesdrop}(m_L, m_R)$:
$z \leftarrow \textsc{query}()$
$c := z \oplus m_R$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-real}}$
$\textsc{query}()$:
$s \leftarrow \{0,1\}^{\lambda}$
return $G(s)$

Pseudo OTP

$\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{M} = \{0,1\}^{\lambda+\ell}$, $\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen: $k \leftarrow \mathcal{K}$, return k.

Enc(k, m): return $G(k) \oplus m$

Dec(k, c): return $G(k) \oplus c$

Claim If this scheme (called pOTP) is instantiated using a secure PRG G, then it has computational one-time secrecy.

Proof

$\underline{\text{EAVESDROP}(m_L, m_R):}$
$z \leftarrow \text{QUERY}()$
$c := z \oplus m_R$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-real}}$
$\underline{\text{QUERY}():}$
$s \leftarrow \{0,1\}^\lambda$
return $G(s)$

$\Rightarrow$

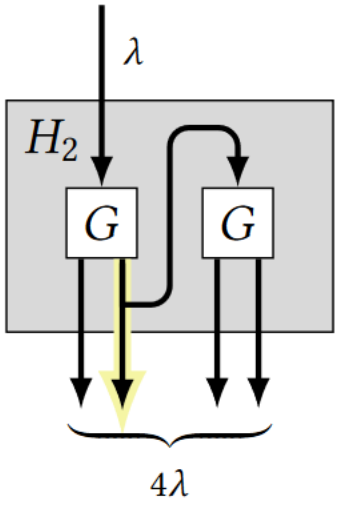
$\mathcal{L}^{\text{pOTP}}_{\text{ots-R}}$
$\underline{\text{EAVESDROP}(m_L, m_R):}$
$k \leftarrow \{0,1\}^\lambda$
$c := G(k) \oplus m_R$
return c

Extending the Stretch of a PRG

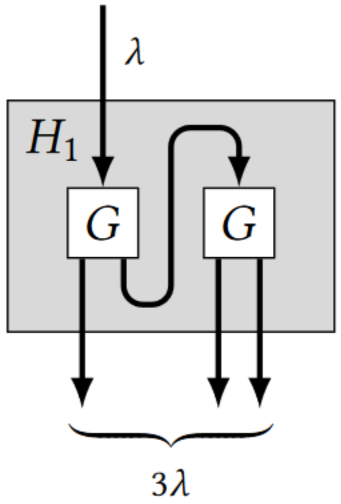
- Suppose $G: \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ is a length-doubling PRG.

- Are the following constructions secure?

$\underline{H_2(s):}$
$x \| y := G(s)$
$u \| v := G(y)$
return $x \| y \| u \| v$



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

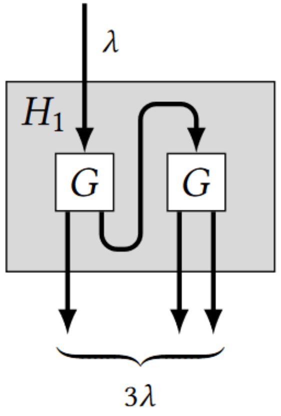


Extending the Stretch of a PRG

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

Proof

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

$\diamond$

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

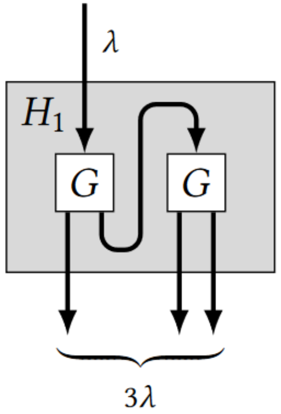
$\text{QUERY}_G()$:
  $s \leftarrow \{0,1\}^\lambda$
  return $G(s)$

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 $x\|u\|v$



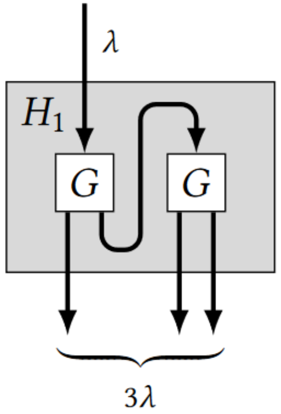
$\text{QUERY}_{H_1}()$: $x\|y := \text{QUERY}_G()$ $u\|v := G(y)$ return $x\|u\|v$	$\diamond$	$\mathcal{L}^G_{\text{prg-real}}$ $\text{QUERY}_G()$: $s \leftarrow \{0,1\}^\lambda$ return $G(s)$	$\Rightarrow$	$\text{QUERY}_{H_1}()$: $x\|y := \text{QUERY}_G()$ $u\|v := G(y)$ return $x\|u\|v$	$\diamond$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

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



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

$\diamond$

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

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

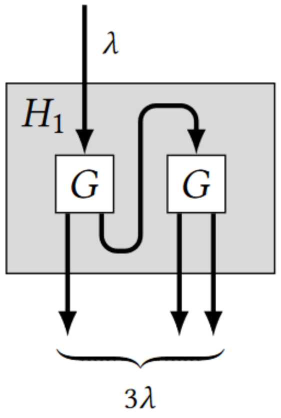
$\text{QUERY}_{H_1}()$:
$x \| y \leftarrow \{0,1\}^{2\lambda}$
$u \| v := G(y)$
return $x \| u \| v$

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 $x\|u\|v$

$\Rightarrow$

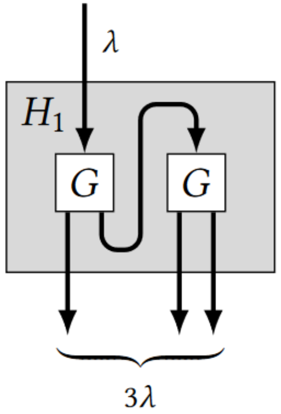
$\underline{\text{QUERY}_{H_1}():}$
$x \leftarrow \{0,1\}^{\lambda}$
$y \leftarrow \{0,1\}^{\lambda}$
$u\|v := G(y)$
return $x\|u\|v$

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 $x \| u \| v$

$\diamond$

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

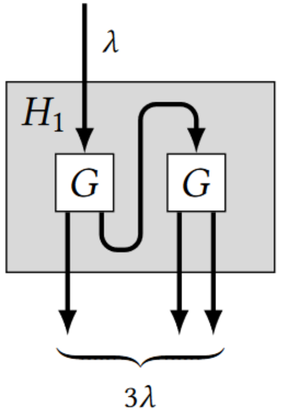
$\underline{\text{QUERY}_G():}$
$s \leftarrow \{0,1\}^\lambda$
return $G(s)$

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 $x \| u \| v$



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

$\Rightarrow$

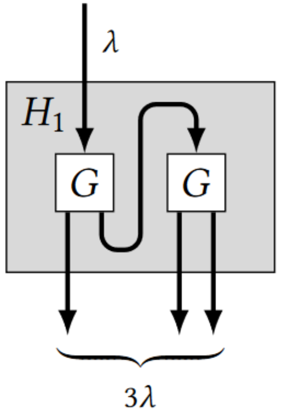
$\text{QUERY}_{H_1}()$:		$\mathcal{L}^G_{\text{prg-rand}}$
$x \leftarrow \{0,1\}^\lambda$	$\diamond$	$\text{QUERY}_G()$:
$u \| v := \text{QUERY}_G()$		$r \leftarrow \{0,1\}^{2\lambda}$
return $x \| u \| v$		return r

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 $x \| u \| v$



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

$\diamond$

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

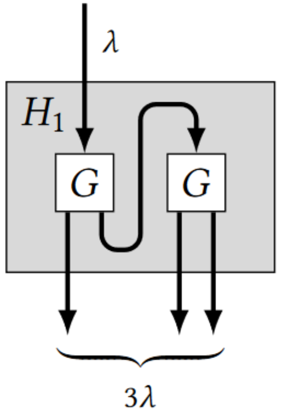
$\textsc{query}_{H_1}()$:
$x \leftarrow \{0,1\}^\lambda$
$u \| v \leftarrow \{0,1\}^{2\lambda}$
return $x \| u \| v$

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

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



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

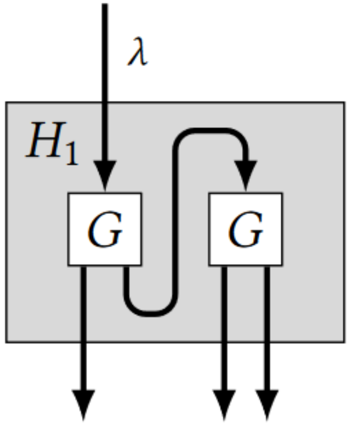
$\mathcal{L}^{H_1}_{\text{prg-rand}}$

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

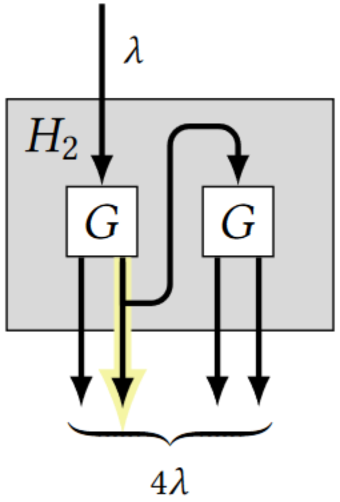
Claim Construction H_2 is not a secure PRG, even if G is.

Proof

Consider the distinguisher $\mathcal{A}$

$x\|y\|u\|v := \text{QUERY}()$
return $G(y) \stackrel{?}{=} u\|v$

$H_2(s)$:
$x\|y := G(s)$
$u\|v := G(y)$
return $x\|y\|u\|v$



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

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

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 $n\lambda$:

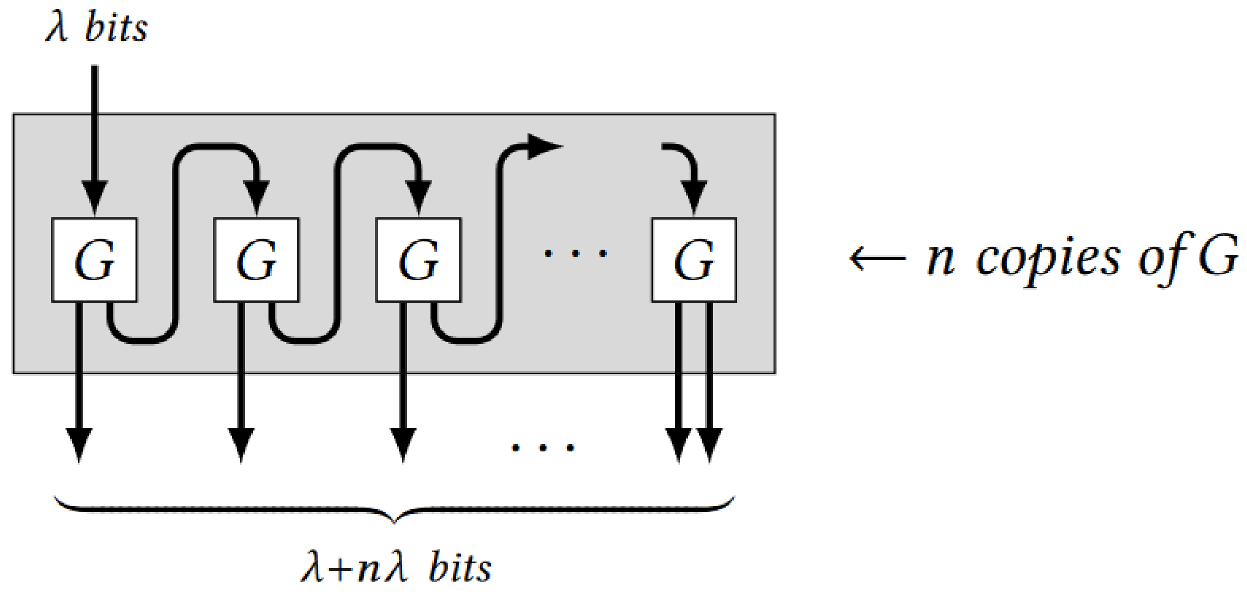
$\underline{H_n(s)\text{:}}$

$s_0 := s$

for $i = 1$ to n:

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

return $t_1 \| \cdots \| t_n \| s_n$



Stream Cipher

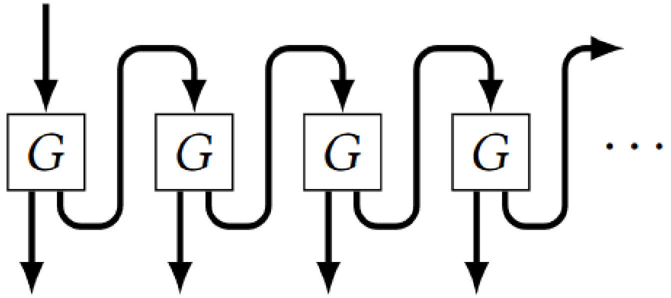
Definition A stream cipher is an algorithm G that takes a seed s and length ℓ as input, and outputs a string. It should satisfy the following requirements:

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

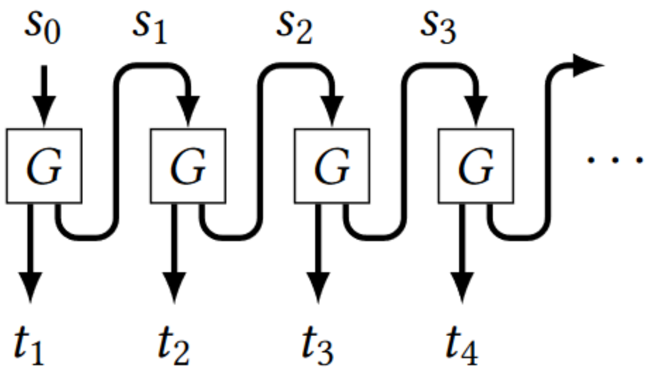
- Suppose Alice & Bob share a symmetric key k and exchange messages over a long period of time.
- However, suppose Bob's device is compromised and an attacker learns k. Then the attacker can decrypt all past, present, and future ciphertexts that it saw!

- 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.

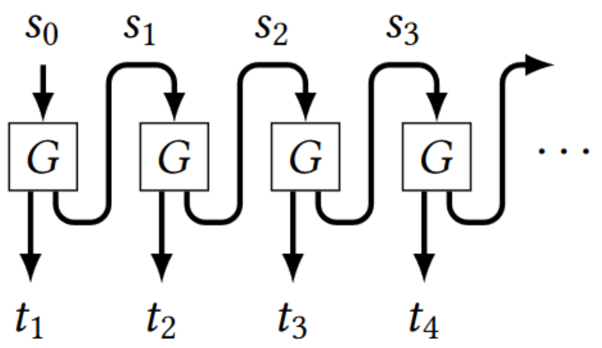
$s_0 = k$
for $i = 1$ to ∞:
  $s_i \| t_i := G(s_{i-1})$
  erase s_{i-1} from memory
  use t_i to encrypt/decrypt the ith message
  erase t_i from memory



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.

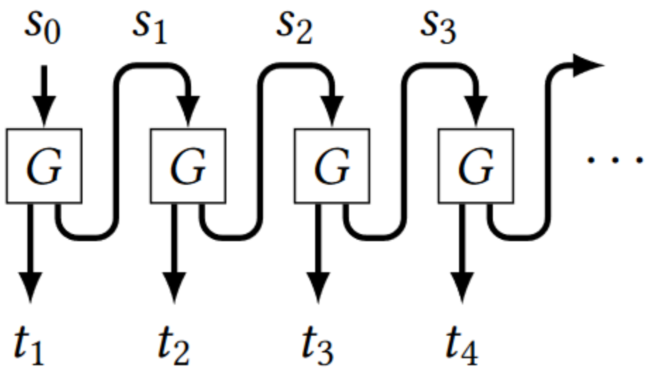
$s_0 = k$
for $i = 1$ to ∞:
  $s_i \| t_i := G(s_{i-1})$
  erase s_{i-1} from memory
  use t_i to encrypt/decrypt the ith message
  erase t_i from memory



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



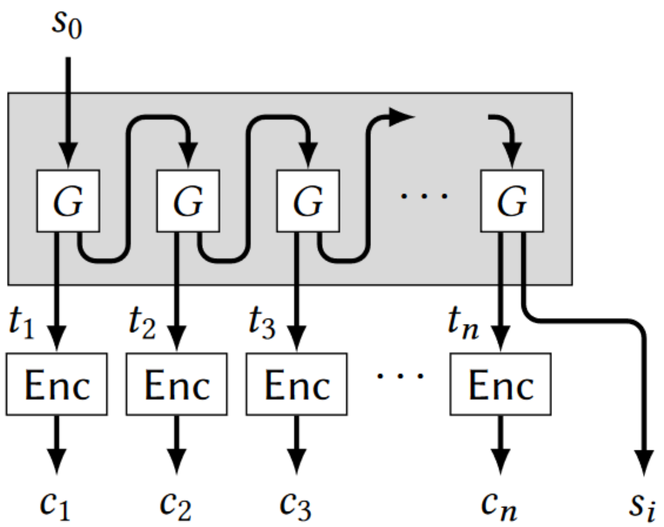
Symmetric Ratchet

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

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



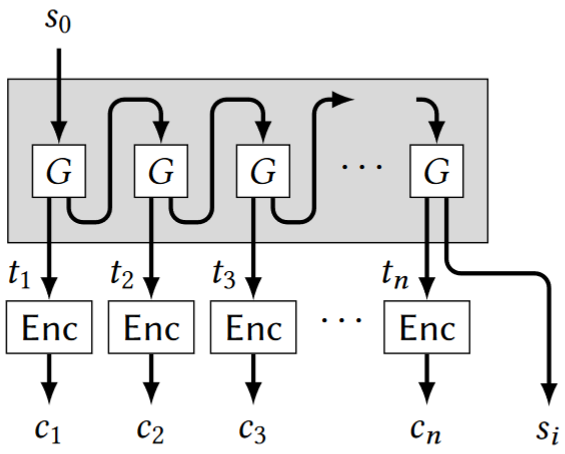
Symmetric Ratchet

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.

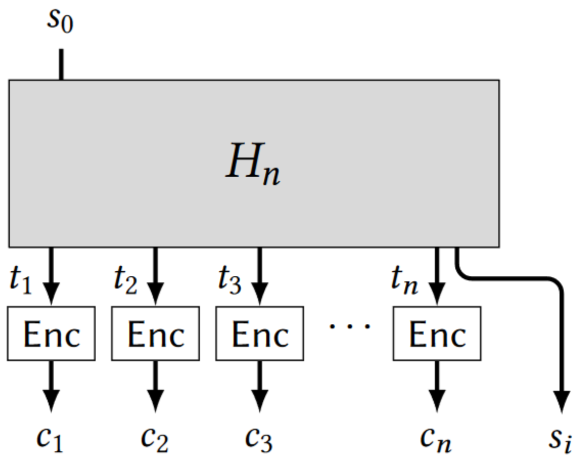
Proof

The situation is captured by the following library

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



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



Symmetric Ratchet

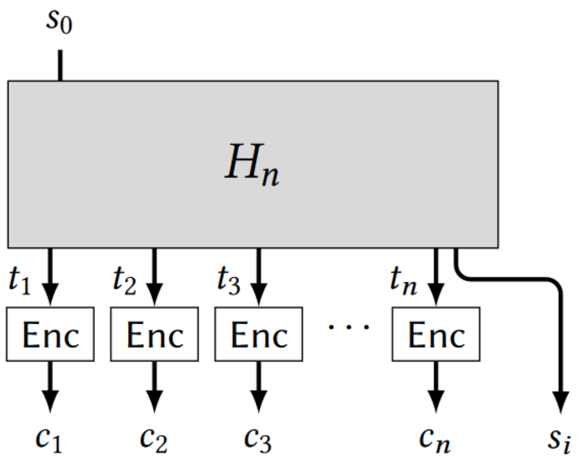
Claim If the symmetric ratchet is used with a secure PRG G and an encryption scheme Σ that has uniform ciphertexts (and $\Sigma.\mathcal{K} = \{0,1\}^\lambda$), then the first n ciphertexts are pseudorandom, even to an eavesdropper who compromises the key s_n.

Proof

The situation is captured by the following library

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

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



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

for $i = 1$ to n:
  $t_i \leftarrow \{0,1\}^\lambda$
$s_n \leftarrow \{0,1\}^\lambda$
for $i = 1$ to n:
  $c_i \leftarrow \Sigma.\mathsf{Enc}(t_i, m_i)$
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

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

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

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

for $i = 1$ to n:

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

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

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

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

$\textsc{attack}(m_1, \ldots, m_n)$:
  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 $(c_1, \ldots, c_n, s_n)$

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

$\diamond$

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

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

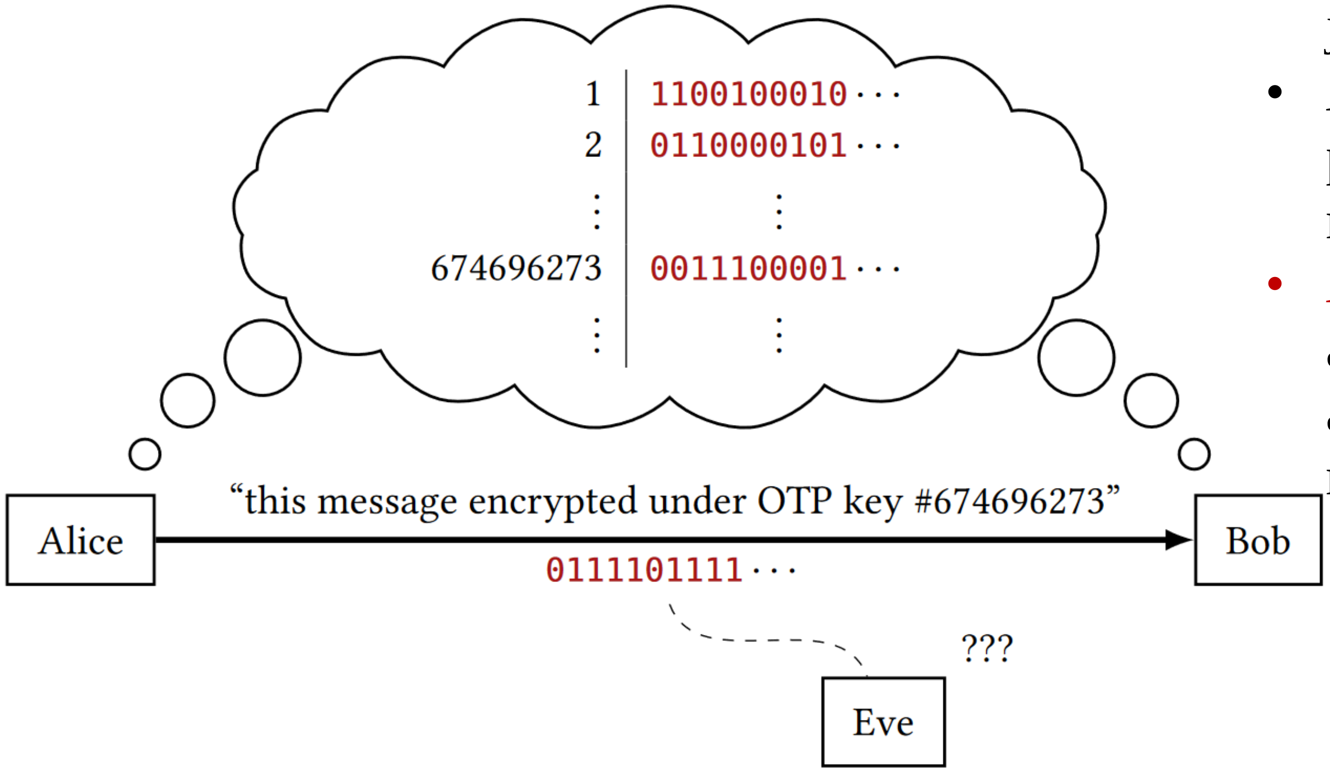
$\text{ATTACK}(m_1, \ldots, m_n)$:	$\diamond$	$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$		$\text{ATTACK}(m_1, \ldots, m_n)$:
for $i = 1$ to n:		$\text{CTXT}(m \in \Sigma.\mathcal{M})$:		for $i = 1$ to n:
$c_i \leftarrow \text{CTXT}(m_i)$		$k \leftarrow \Sigma.\text{KeyGen}$	$\Rightarrow$	$c_i \leftarrow \Sigma.\mathcal{C}$
$s_n \leftarrow \{0,1\}^\lambda$		$c \leftarrow \Sigma.\text{Enc}(k, m)$		$s_n \leftarrow \{0,1\}^\lambda$
return $(c_1, \ldots, c_n, s_n)$		return c		return $(c_1, \ldots, c_n, s_n)$

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

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

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

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

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

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

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

  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 $T[x]$

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

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

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

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

  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 $T[x]$

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?

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

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:

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

$$\begin{array}{|l|}\hline \mathcal{A} \\ \hline \text{pick } x_1 \neq x_2 \in \{0,1\}^{2\lambda} \\ z_1 := \textsc{lookup}(x_1) \\ z_2 := \textsc{lookup}(x_2) \\ \text{return } z_1 \oplus z_2 \stackrel{?}{=} x_1 \oplus x_2 \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \mathcal{L}^F_{\text{prf-real}} \\ \hline k \leftarrow \{0,1\}^\lambda \\ \\ \underline{\textsc{lookup}(x):} \\ \quad \text{return } G(k) \oplus x \text{ // } F(k,x) \\ \hline \end{array}$$

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

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:

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

$\mathcal{A}$ 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 $z_1 \oplus z_2 \overset{?}{=} x_1 \oplus x_2$	$\diamond$	$T :=$ 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 $F\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ (i.e., $in = out = \lambda$). We can build a length-doubling PRG in the following way (Counter PRG):

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

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:

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

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

$\underline{\textsc{lookup}(x \in \{0,1\}^{in}):}$
  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 $T[x]$

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:

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

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

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

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

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

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

Constructing a PRG from a PRF

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

proof

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

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

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

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

$\underline{\textsc{query}():}$

$s \leftarrow \{\mathtt{0},\mathtt{1}\}^\lambda$

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

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

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

$\underline{\textsc{lookup}(x \in \{\mathtt{0},\mathtt{1}\}^{in}):}$

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.

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

$count := 0$
 	
$\underline{\text{QUERY}()}$:
  $count := count + 1$
  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 $x \| y$

- 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

QUERY():
  count := count + 1
  if count ≤ i :
    r ← {0,1}^{2λ}
    return r
  else:
    s ← {0,1}^λ
    x := F(s, 0···00)
    y := F(s, 0···01)
    return x||y
```

```
count := 0

QUERY():
  count := count + 1
  if count < i :
    r ← {0,1}^{2λ}
    return r
  elsif count = i :
    s* ← {0,1}^λ
    x := F(s*, 0···00)
    y := F(s*, 0···01)
    return x||y
  else:
    s ← {0,1}^λ
    x := F(s, 0···00)
    y := F(s, 0···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.

$$
\begin{array}{|l|}
\hline
\textit{count} := 0 \\
\\
\underline{\text{QUERY}():} \\
\quad \textit{count} := \textit{count} + 1 \\
\quad \text{if } \textit{count} < i : \\
\qquad r \leftarrow \{\texttt{0},\texttt{1}\}^{2\lambda} \\
\qquad \text{return } r \\
\quad \text{elsif } \textit{count} = i : \\
\qquad s^* \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda} \\
\qquad x := F(s^*, \texttt{0}\cdots\texttt{00}) \\
\qquad y := F(s^*, \texttt{0}\cdots\texttt{01}) \\
\qquad \text{return } x \| y \\
\quad \text{else:} \\
\qquad s \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda} \\
\qquad x := F(s, \texttt{0}\cdots\texttt{00}) \\
\qquad y := F(s, \texttt{0}\cdots\texttt{01}) \\
\qquad \text{return } x \| y \\
\hline
\end{array}
\quad \Rightarrow \quad
\begin{array}{|l|}
\hline
\textit{count} := 0 \\
\\
\underline{\text{QUERY}():} \\
\quad \textit{count} := \textit{count} + 1 \\
\quad \text{if } \textit{count} < i : \\
\qquad r \leftarrow \{\texttt{0},\texttt{1}\}^{2\lambda} \\
\qquad \text{return } r \\
\quad \text{elsif } \textit{count} = i : \\
\qquad x := \text{LOOKUP}(\texttt{0}\cdots\texttt{00}) \\
\qquad y := \text{LOOKUP}(\texttt{0}\cdots\texttt{01}) \\
\qquad \text{return } x \| y \\
\quad \text{else:} \\
\qquad s \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda} \\
\qquad x := F(s, \texttt{0}\cdots\texttt{00}) \\
\qquad y := F(s, \texttt{0}\cdots\texttt{01}) \\
\qquad \text{return } x \| y \\
\hline
\end{array}
\diamond
\begin{array}{|l|}
\hline
\quad \mathcal{L}^{F}_{\text{prf-real}} \\
\hline
k \leftarrow \{\texttt{0},\texttt{1}\}^{\lambda} \\
\\
\underline{\text{LOOKUP}(x):} \\
\quad \text{return } F(k, x) \\
\hline
\end{array}
$$

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.

$count := 0$

$\underline{\text{QUERY}()}$:
  $count := count + 1$
  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 $x \| y$

$\diamond$

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

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

$\underline{\text{LOOKUP}(x)}$:
  return $F(k, x)$

$count := 0$

$\underline{\text{QUERY}()}$:
  $count := count + 1$
  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

$\underline{\text{LOOKUP}(x)}$:
  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 $count < \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = \boxed{i}$:
  - $x := \text{LOOKUP}(0\cdots00)$
  - $y := \text{LOOKUP}(0\cdots01)$
  - 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 $count < \boxed{i}$:
  - $r \leftarrow \{0,1\}^{2\lambda}$
  - return r
- elsif $count = \boxed{i}$:
  - $x := \text{LOOKUP}(0\cdots00)$
  - $y := \text{LOOKUP}(0\cdots01)$
  - 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.

$count := 0$

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

$\diamond$

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

$\Rightarrow$

$count := 0$

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

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.

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

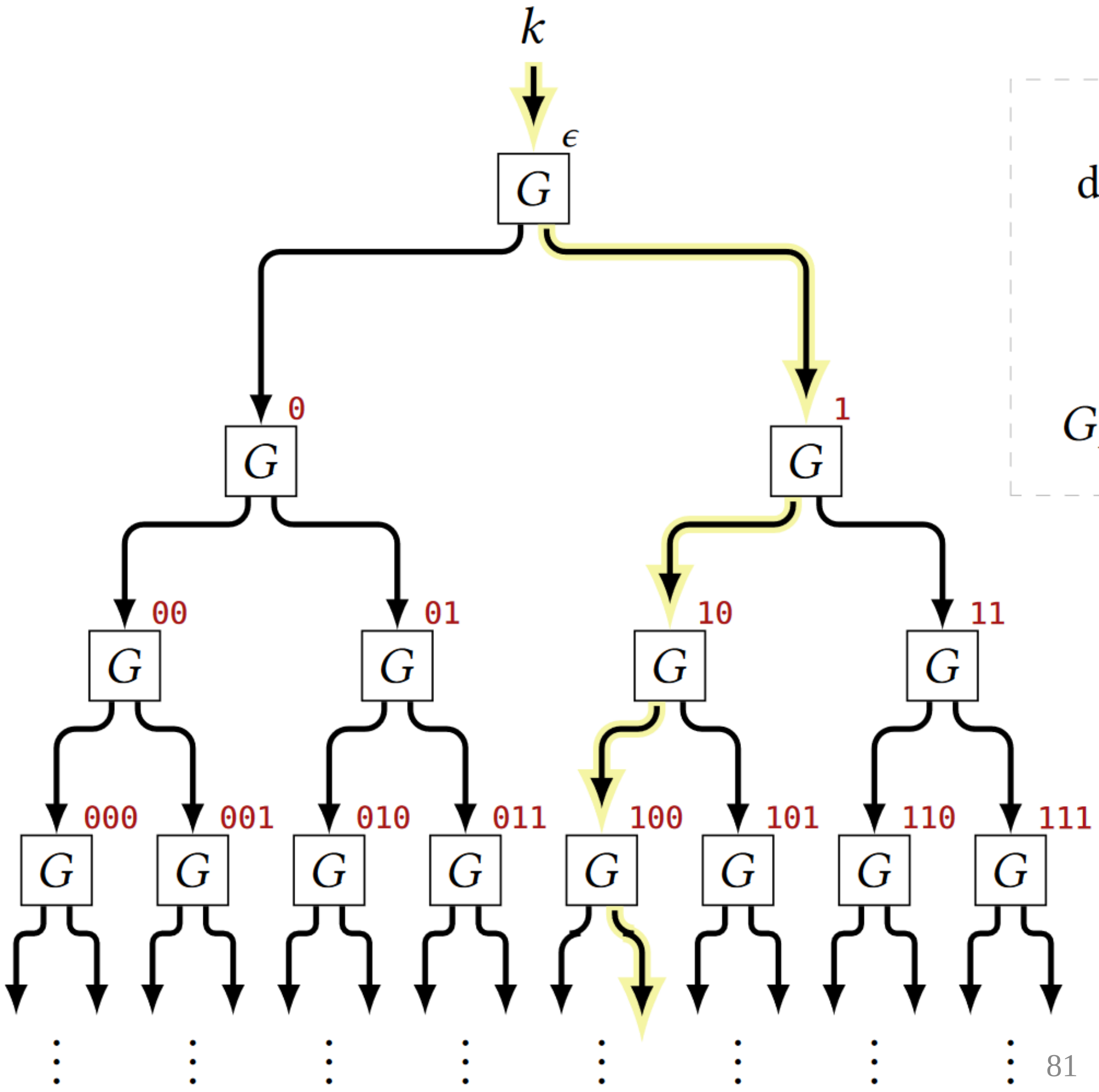
```
count := 0

QUERY():
  count := count + 1
  if count < i :
    r ← {0,1}^{2λ}
    return r
  elsif count = i :
    x ← {0,1}^λ
    y ← {0,1}^λ
    return x||y
  else:
    s ← {0,1}^λ
    x := F(s, 0···00)
    y := F(s, 0···01)
    return x||y
```

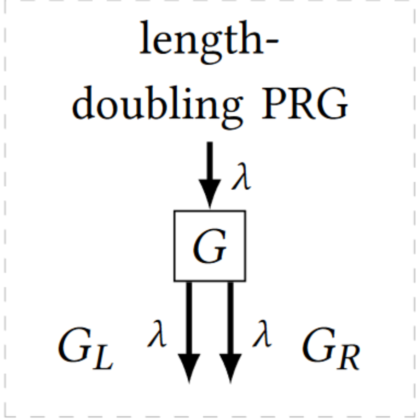
Constructing a PRF from a PRG

- 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.

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

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

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* $T[\epsilon]$
$\underline{\text{LOOKUP}(x)}$:	
$p :=$ first $\boxed{0}$ 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 $i = \boxed{1}$ 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	
LOOKUP(x):	
$p :=$ 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

LOOKUP(x):
  p := first d − 1 bits of x
  if T[p] undefined:
    T[p] ← {0,1}^λ
    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
```

➡

```
T := empty assoc. array

LOOKUP(x):
  p := first d − 1 bits of x
  if T[p] undefined:
    T[p‖0] ‖ T[p‖1] := 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
```

$\diamond$

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

```
QUERY():
  s ← {0,1}^λ
  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.

$$
\begin{array}{l}
T := \text{empty assoc. array} \\
\\
\underline{\text{LOOKUP}(x):} \\
\quad p := \text{first } d-1 \text{ bits of } x \\
\quad \text{if } T[p] \text{ undefined:} \\
\qquad T[p\|0] \| T[p\|1] := \text{QUERY}() \\
\quad p' := \text{first } d \text{ bits of } x \\
\quad v := T[p'] \\
\quad \text{for } i = d+1 \text{ to } in: \\
\qquad \text{if } x_i = 0 \text{ then } v := G_L(v) \\
\qquad \text{if } x_i = 1 \text{ then } v := G_R(v) \\
\quad \text{return } v
\end{array}
\diamond
\begin{array}{|l|}
\hline
\mathcal{L}^G_{\text{prg-real}} \\
\hline
\underline{\text{QUERY}():} \\
\quad s \leftarrow \{0,1\}^\lambda \\
\quad \text{return } G(s) \\
\hline
\end{array}
$$

$\Rightarrow$

$$
\begin{array}{l}
T := \text{empty assoc. array} \\
\\
\underline{\text{LOOKUP}(x):} \\
\quad p := \text{first } d-1 \text{ bits of } x \\
\quad \text{if } T[p] \text{ undefined:} \\
\qquad T[p\|0] \| T[p\|1] := \text{QUERY}() \\
\quad p' := \text{first } d \text{ bits of } x \\
\quad v := T[p'] \\
\quad \text{for } i = d+1 \text{ to } in: \\
\qquad \text{if } x_i = 0 \text{ then } v := G_L(v) \\
\qquad \text{if } x_i = 1 \text{ then } v := G_R(v) \\
\quad \text{return } v
\end{array}
\diamond
\begin{array}{|l|}
\hline
\mathcal{L}^G_{\text{prg-rand}} \\
\hline
\underline{\text{QUERY}():} \\
\quad r \leftarrow \{0,1\}^{2\lambda} \\
\quad \text{return } r \\
\hline
\end{array}
$$

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] ‖ T[p‖1] := 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
```

$\diamond$

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

```
QUERY():
  r ← {0,1}^{2λ}
  return r
```

$\Rightarrow$

```
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
```

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
```