Advanced Cryptography

（Provable Security）

Yi LIU

Shamir Secret Sharing

- $\mathcal{M} = \mathbb{Z}_p$, where p is a prime. $n < p$, $t \leq n$
- Share(m):
  - $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  - $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$
  - Let $s_i = (i, f(i)\%p)$ for $i = 1$ to n. Return $(s_1, \ldots, s_n)$.
- Reconstruct($\{ s_i \mid i \in U \}$):
  - $f(x) \coloneqq$ unique degree $-$ $(t-1)$ polynomial mod p passing through points $\{s_i \mid i \in U\}$
  - Return $f(0)$.

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e.,$\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

$\mathcal{L}_{\text{shamir-real}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\|U\| \geq t$: return err
    $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
    $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$
    For $i \in U$:
        $s_i \coloneqq (i, f(i) \% p)$
    return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-rand}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\|U\| \geq t$: return err
    For $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i \coloneqq (i, y_i)$
    return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e.,$\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

$\mathcal{L}_{\text{shamir-real}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\|U\| \geq t$: return err
    $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
    $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$
    For $i \in U$:
        $s_i \coloneqq (i, f(i)\%p)$
    return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-rand}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\|U\| \geq t$: return err
    For $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i \coloneqq (i, y_i)$
    return $\{s_i \mid i \in U\}$

Proof

Fix $m \in \mathbb{Z}_p$, fix set U with $|U| < t$.

For each $i \in U$, fix a value $y_i \in \mathbb{Z}_p$.

Consider the probability that **poly**(m, t, U) outputs $\{(i, y_i) \mid i \in U\}$ in each library.

- In $\mathcal{L}_{\text{shamir-real}}$, there are p^{t-1} such degree-$(t-1)$ polynomials (according to the previous corollary) such that $f(0) \equiv_p m$.
  - To be consistent with $(0, m) \cup \{(i, y_i) \mid i \in U\}$, there are $p^{t-(|U|+1)}$ such polynomials.
  - Happen with probability $\frac{p^{t-|U|-1}}{p^{t-1}} = p^{-|U|}$.

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e.,$\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

$\mathcal{L}_{\text{shamir-real}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\lvert U\rvert \geq t$: return err
    $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
    $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$
    For $i \in U$:
        $s_i \coloneqq (i, f(i)\%p)$
    return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-rand}}$

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\lvert U\rvert \geq t$: return err
    For $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i \coloneqq (i, y_i)$
    return $\{s_i \mid i \in U\}$

Proof

Fix $m \in \mathbb{Z}_p$, fix set U with $|U| < t$.

For each $i \in U$, fix a value $y_i \in \mathbb{Z}_p$.

Consider the probability that **poly**(m, t, U) outputs $\{(i, y_i) \mid i \in U\}$ in each library.

- In $\mathcal{L}_{\text{shamir-rand}}$, $|U|$ output values are chosen uniformly in $\mathbb{Z}_p$, $p^{|U|}$ ways to choose them, but only one cause **poly**(m, t, U) to output specific choice of $\{(i, y_i) \mid i \in U\}$. The probability of receiving this output is $p^{-|U|}$.

For all possible inputs to **poly**, both libraries assign the same probability to every possible output. Hence, the libraries are interchangeable.

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\mathcal{L}^{\mathcal{S}}_{\text{tsss-L}}$

$\textsc{share}(m_L, m_R, U)$:
- if $|U| \geqslant t$: return err
- $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
- $f(\boldsymbol{x}) := m_L + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
- for $i \in U$:
  - $s_i := (i, f(i) \,\%\, p)$
- return $\{s_i \mid i \in U\}$

$\textsc{share}(m_L, m_R, U)$:
- return $\textsc{poly}(m_L, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-real}}$

$\textsc{poly}(m, t, U)$:
- if $|U| \geqslant t$: return err
- $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
- $f(\boldsymbol{x}) := m + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
- for $i \in U$:
  - $s_i := (i, f(i) \,\%\, p)$
- return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\text{SHARE}(m_L, m_R, U)$:
  return $\text{POLY}(m_L, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-real}}$

$\text{POLY}(m, t, U)$:
  if $|U| \geqslant t$: return err
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(\boldsymbol{x}) := m + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
  for $i \in U$:
    $s_i := (i, f(i) \% p)$
  return $\{s_i \mid i \in U\}$

$\Rightarrow$

$\text{SHARE}(m_L, m_R, U)$:
  return $\text{POLY}(m_L, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

$\text{POLY}(m, t, U)$:
  if $|U| \geqslant t$: return err
  for $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
  return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\textsc{share}(m_L, m_R, U)$:
  return $\textsc{poly}(m_L, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

$\textsc{poly}(m, t, U)$:
  if $|U| \geqslant t$: return `err`
  for $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
  return $\{s_i \mid i \in U\}$

$\Rightarrow$

$\textsc{share}(m_L, m_R, U)$:
  return $\textsc{poly}(m_R, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

$\textsc{poly}(m, t, U)$:
  if $|U| \geqslant t$: return `err`
  for $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
  return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

```
SHARE(m_L, m_R, U):
  return POLY(m_R, t, U)
```

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

```
POLY(m, t, U):
  if |U| ⩾ t: return err
  for i ∈ U:
    y_i ← Z_p
    s_i := (i, y_i)
  return {s_i | i ∈ U}
```

$\Rightarrow$

```
SHARE(m_L, m_R, U):
  return POLY(m_R, t, U)
```

$\diamond$

$\mathcal{L}_{\text{shamir-real}}$

```
POLY(m, t, U):
  if |U| ⩾ t: return err
  f_1, ..., f_{t-1} ← Z_p
  f(x) := m + Σ_{j=1}^{t-1} f_j x^j
  for i ∈ U:
    s_i := (i, f(i) % p)
  return {s_i | i ∈ U}
```

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\textsc{share}(m_L, m_R, U)$:
  return $\textsc{poly}(m_R, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-real}}$

$\textsc{poly}(m, t, U)$:
  if $|U| \geqslant t$: return err
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(\boldsymbol{x}) := m + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
  for $i \in U$:
    $s_i := (i, f(i) \,\%\, p)$
  return $\{s_i \mid i \in U\}$

$\mathcal{L}^{S}_{\text{tsss-R}}$

$\textsc{share}(m_L, m_R, U)$:
  if $|U| \geqslant t$: return err
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(\boldsymbol{x}) := m_R + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
  for $i \in U$:
    $s_i := (i, f(i) \,\%\, p)$
  return $\{s_i \mid i \in U\}$

Basing Cryptography on Intractable Computations

What Qualifies as a "Computationally Infeasible" Attack?

- Intuition

  It doesn't really matter whether attacks are impossible, only whether attacks are computationally infeasible.

- For a scheme with λ-bit keys
  - The most direct computation procedure would be for the enemy to try all 2^{λ} possible keys, one by one. Obviously this is easily made impractical for the enemy by simply choosing λ large enough.
  - We call λ the security parameter of the scheme. A scheme described as having n-bit security if the best known attack requires 2^n steps.
    - A scheme with λ-bit keys may have attack that cost only $2^{\lambda/2}$.

What Qualifies as a “Computationally Infeasible” Attack?

clock cycles	*approx cost*	*reference*
2^{50}	*\$3.50*	*cup of coffee*
2^{55}	*\$100*	*decent tickets to a Portland Trailblazers game*
2^{65}	*\$130,000*	*median home price in Oshkosh, WI*
2^{75}	*\$130 million*	*budget of one of the Harry Potter movies*
2^{85}	*\$140 billion*	*GDP of Hungary*
2^{92}	*\$20 trillion*	*GDP of the United States*
2^{99}	*\$2 quadrillion*	*all of human economic activity since 300,000 BC*[4]
2^{128}	*really a lot*	*a billion human civilizations’ worth of effort*

Asymptotic Running Time

Definition A program runs in polynomial time if there exists a constant $c > 0$ such that for all sufficiently long input strings x, the program stops after no more than $O(|x|^c)$ steps.

- In crypto world, "polynomial-time" is a synonym for "efficient".
  - Polynomial time is not a perfect match to what we mean when we informally talk about "efficient" algorithms.
    - Algorithms with running time $\Theta(n^{1000})$ are technically polynomial-time, while those with running time $\Theta(n^{\log\log\log n})$ aren't.
- Closure property: repeating a polynomial-time process a polynomial number of times results in a polynomial-time process overall.

Potential Pitfall: Numerical Algorithms

- Remember that representing the number N on a computer requires only $\sim \log_2 N$ bits. This means that $\log_2 N$, rather than N, is our security parameter.
  - The difference between running time $O(\log N)$ and $O(N)$ is the difference between writing down a number and counting to the number.

Efficient algorithm known:	**No known efficient algorithm:**
Computing GCDs	Factoring integers
Arithmetic mod N	Computing $\phi(N)$ given N
Inverses mod N	Discrete logarithm
Exponentiation mod N	Square roots mod composite N

What qualifies as a "Negligible" Success Probability?

- We don't want to worry about attacks that are as expensive as a brute-force attack. ("Computationally Infeasible" Attack)
- We don't want to worry about attacks whose success probability is as low as a blind-guess attack. ("Negligible" Success Probability)

probability	*equivalent*
2^{-10}	*full house in 5-card poker* 满堂红（三张相同牌加对子）
2^{-20}	*royal flush in 5-card poker* 皇家同花顺
2^{-28}	*you win this week's Powerball jackpot*
2^{-40}	*royal flush in 2 consecutive poker games*
2^{-60}	*the next meteorite that hits Earth lands in this square* →

What qualifies as a “Negligible” Success Probability?

- For example, $1/2^{\lambda}$ approaches zero so fast that no polynomial can “rescue” it, i.e., $\lim_{\lambda \to \infty} \frac{p(\lambda)}{2^{\lambda}} = 0$ for polynomial p.
  - In other words, it approaches zero faster than 1 over any polynomial.

What qualifies as a “Negligible” Success Probability?

Definition A function f is negligible if, for every polynomial p, we have $\lim_{\lambda \to \infty} p(\lambda) f(\lambda) = 0$.

Claim If for every integer c, $\lim_{\lambda \to \infty} \lambda^c f(\lambda) = 0$, then f is negligible.

What qualifies as a “Negligible” Success Probability?

Claim If for every integer c, $\lim_{\lambda\to\infty} \lambda^c f(\lambda) = 0$, then f is negligible.

Proof

Suppose f has this property, and take arbitrary polynomial p, show that $\lim_{\lambda\to\infty} p(\lambda)f(\lambda) = 0$.

If d is the degree of p, then $\lim_{\lambda\to\infty} p(\lambda)/\lambda^{d+1} = 0$. Hence,

$$\lim_{\lambda\to\infty} p(\lambda)f(\lambda) = \lim_{\lambda\to\infty} [\frac{p(\lambda)}{\lambda^{d+1}}\left(\lambda^{d+1} \cdot f(\lambda)\right)]$$

$$= \left(\lim_{\lambda\to\infty} \frac{p(\lambda)}{\lambda^{d+1}}\right)\left(\lim_{\lambda\to\infty} \lambda^{d+1} \cdot f(\lambda)\right) = 0 \cdot 0 = 0$$

What qualifies as a "Negligible" Success Probability?

Definition If $f, g : \mathbb{N} \to \mathbb{R}$ are two functions, we write $f \approx g$ to mean that $|f(\lambda) - g(\lambda)|$ is a negligible function.

$$
\begin{aligned}
\Pr[X] \approx 0 \quad &\Leftrightarrow \quad \text{“event } X \text{ almost never happens”} \\
\Pr[Y] \approx 1 \quad &\Leftrightarrow \quad \text{“event } Y \text{ almost always happens”} \\
\Pr[A] \approx \Pr[B] \quad &\Leftrightarrow \quad \text{“events } A \text{ and } B \text{ happen with} \\
&\qquad\ \ \text{essentially the same probability”}
\end{aligned}
$$

Indistinguishability

Definition Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be two libraries with a common interface. We say that $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable, and write $\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$, if for all polynomial-time programs $\mathcal{A}$ that output a single bit, $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]$.

We call the quantity $|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]|$ the advantage or bias of $\mathcal{A}$ in distinguishing $\mathcal{L}_{\text{left}}$ from $\mathcal{L}_{\text{right}}$. Two libraries are therefore indistinguishable if all polynomial-time calling programs have negligible advantage in distinguishing them.

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$
$\underline{\text{PREDICT}(x)\text{:}}$
$s \leftarrow \{0,1\}^{\lambda}$
return $x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$
$\underline{\text{PREDICT}(x)\text{:}}$
return false

- The $\mathcal{L}_{\text{left}}$ library tells the calling program whether its prediction was correct.
- The $\mathcal{L}_{\text{right}}$ library doesn't even bother sampling a string, it just always says "*sorry, your prediction was wrong.*"

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$

```
PREDICT(x):
  s ← {0,1}^λ
  return x =? s
```

$\mathcal{L}_{\text{right}}$

```
PREDICT(x):
  return false
```

$\mathcal{A}_{obvious}$

```
do q times:
  if PREDICT(0^λ) = true
    return 1
return 0
```

$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = 0$

$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1]$
$= 1 - \Pr[\textit{all } q \textit{ independent calls to } \text{PREDICT } \textit{return } \text{false}]$
$= 1 - \left(1 - \frac{1}{2^\lambda}\right)^q$

Then $\mathcal{L}_{\text{left}} \not\equiv \mathcal{L}_{\text{right}}$. These two libraries are not interchangeable.

What about indistinguishability?

Compute an upper bound first

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$

PREDICT(x):

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

return $x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$

PREDICT(x):

return false

$$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = 0$$

$$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1]$$
$$\leq \Pr[first\ call\ to\ PREDICT\ returns\ true]$$
$$+ \Pr[second\ call\ to\ PREDICT\ returns\ true] + \cdots = q\frac{1}{2^\lambda}$$

$\mathcal{A}_{obvious}$

do q times:

if PREDICT(0^λ) = true

return 1

return 0

- This is an overestimate of some probabilities (e.g., if the first call to predict returns true, then the second call isn't made).
- $\mathcal{A}_{obvious}$ has advantage at most $q/2^\lambda$. Since $\mathcal{A}_{obvious}$ runs in polynomial time, it can only make a polynomial number q of queries to the library, so $q/2^\lambda$ is negligible.
- To show that the libraries are indistinguishable, we must show that every calling program's advantage is negligible. (prove later)

Other Properties

Lemma If $\mathcal{L}_1 \equiv \mathcal{L}_2$ then $\mathcal{L}_1 \approx \mathcal{L}_2$. If $\mathcal{L}_1 \approx \mathcal{L}_2 \approx \mathcal{L}_3$ then $\mathcal{L}_1 \approx \mathcal{L}_3$.

Lemma If $\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$ then $\mathcal{L}^* \diamond \mathcal{L}_{\text{left}} \approx \mathcal{L}^* \diamond \mathcal{L}_{\text{right}}$ for any polynomial-time library $\mathcal{L}^*$.

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named 'bad' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr\left[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1\right]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

Fix an arbitrary calling program $\mathcal{A}$. Define the following events

$\mathcal{B}_{\text{left}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{left}}$ sets bad to 1 at some point.

$\mathcal{B}_{\text{right}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{right}}$ sets bad to 1 at some point.

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named '*bad*' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

$\mathcal{B}_{\text{left}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{left}}$ sets bad to 1 at some point.

$\mathcal{B}_{\text{right}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{right}}$ sets bad to 1 at some point.

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}]\Pr[\mathcal{B}_{\text{left}}] + \Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\Pr[\overline{\mathcal{B}_{\text{left}}}]$$

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]\Pr[\mathcal{B}_{\text{right}}] + \Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{right}}}]\Pr[\overline{\mathcal{B}_{\text{right}}}]$$

We have $\Pr[\mathcal{B}_{\text{left}}]=\Pr[\mathcal{B}_{\text{right}}]$, because $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$. Let $p^* =_{\text{def}} \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{B}_{\text{right}}]$.

$advantage_{\mathcal{A}} = \left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| = |p^*(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}] + (1 - p^*)\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\right)|$

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named '*bad*' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

Let $p^* =_{\text{def}} \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{B}_{\text{right}}]$.

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

$$= |p^*(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]$$

$$+\ (1 - p^*)\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\right)|$$

We already have $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]$:

$$advantage_{\mathcal{A}} = p^*|\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]\right|$$

Hence, $advantage_{\mathcal{A}} \leq p^* = \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \text{ sets bad} = 1]$

Return to the Example

- $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable.

$\mathcal{L}_{\text{left}}$
$\underline{\text{PREDICT}(x):}$
$s \leftarrow \{0,1\}^{\lambda}$
return $x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$
$\underline{\text{PREDICT}(x):}$
return `false`

Return to the Example

- $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable.

$\mathcal{L}_{\text{left}}$		$\mathcal{L}_{\text{hyb-L}}$		$\mathcal{L}_{\text{hyb-R}}$		$\mathcal{L}_{\text{right}}$
		bad := 0		bad := 0		
PREDICT(x): $s \leftarrow \{0,1\}^\lambda$ return $x \stackrel{?}{=} s$	$\equiv$	PREDICT(x): $s \leftarrow \{0,1\}^\lambda$ if $x = s$: bad := 1 return true return false	$\approx$	PREDICT(x): $s \leftarrow \{0,1\}^\lambda$ if $x = s$: bad := 1 return false	$\equiv$	PREDICT(x): return false

$\mathcal{L}_{\text{hyb}-\text{L}} \approx \mathcal{L}_{\text{hyb}-\text{R}}$: $\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb}-\text{L}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb}-\text{R}} \Rightarrow 1]\right| \le \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb}-\text{L}}\ \textit{sets bad} = 1]$
$\mathcal{A} \diamond \mathcal{L}_{\text{hyb}-\text{L}}\ \textit{sets bad} = 1$ only if s is successfully predicted, which happens at most $q/2^\lambda$, which is negligible when $\mathcal{A}$ runs in polynomial time.

Birthday Probabilities

- Taking q independent, uniform samples from a set of N items. What is the probability that the same value gets chosen more than once? In other words, what is the probability that the following program outputs 1?

$\mathcal{B}(q, N)$

for $i := 1$ to q:
  $s_i \leftarrow \{1, \ldots, N\}$
  for $j := 1$ to $i - 1$:
    if $s_i = s_j$ then return 1
return 0

- $\text{BirthdayProb}(q, N) =_{\text{def}} \Pr[\mathcal{B}(q, N) \text{ outputs } 1]$

Birthday Probabilities

Lemma $\text{BirthdayProb}(q, N) = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})$.

Proof

We instead compute the probability that $\mathcal{B}$ outputs 0. In order for $\mathcal{B}$ to output 0, it must avoid the early termination conditions in each iteration.

$$\begin{aligned}&\Pr[\mathcal{B}(q, N) \text{ outputs } 0] \\ &= \Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i = 1] \times \cdots \\ &\times \Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i = q]\end{aligned}$$

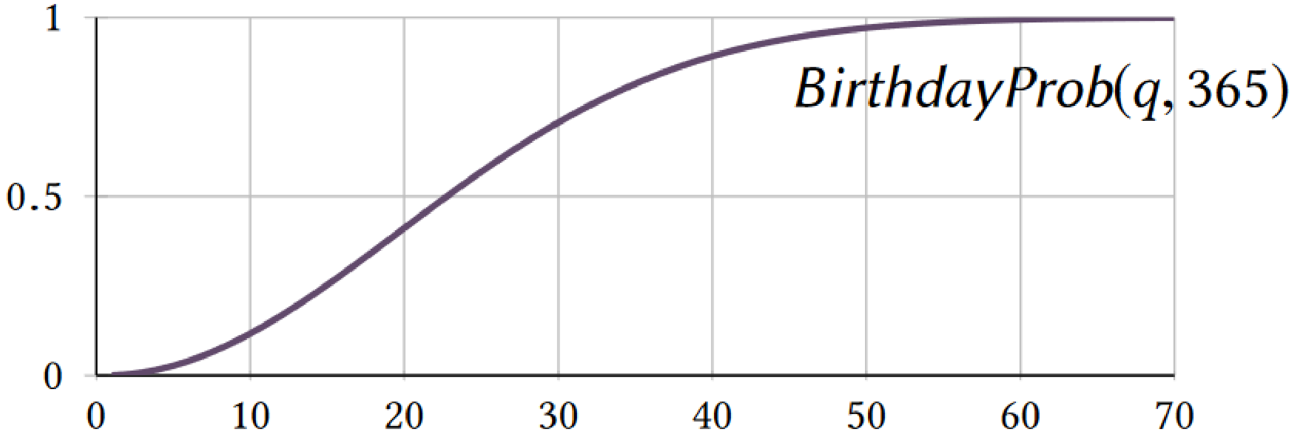
We have $\Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i] = 1 - \frac{i-1}{N}$.

$$\text{BirthdayProb}(q, N) = \Pr[\mathcal{B}(q, N) \text{ outputs } 1] = 1 - \Pr[\mathcal{B}(q, N) \text{ outputs } 0]$$

$$= 1 - \left(1 - \frac{1}{N}\right) \times \cdots \times \left(1 - \frac{q-1}{N}\right) = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})$$

Birthday Probabilities

- Plotting $\text{BirthdayProb}(q, 365)$



- With only $q = 23$ people, the probability already exceeds 50%.
- $q = 70$, the probability exceeds 99.9%.

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

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

Upper bound:

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

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

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

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

Asymptotic Bounds on the Birthday Probability

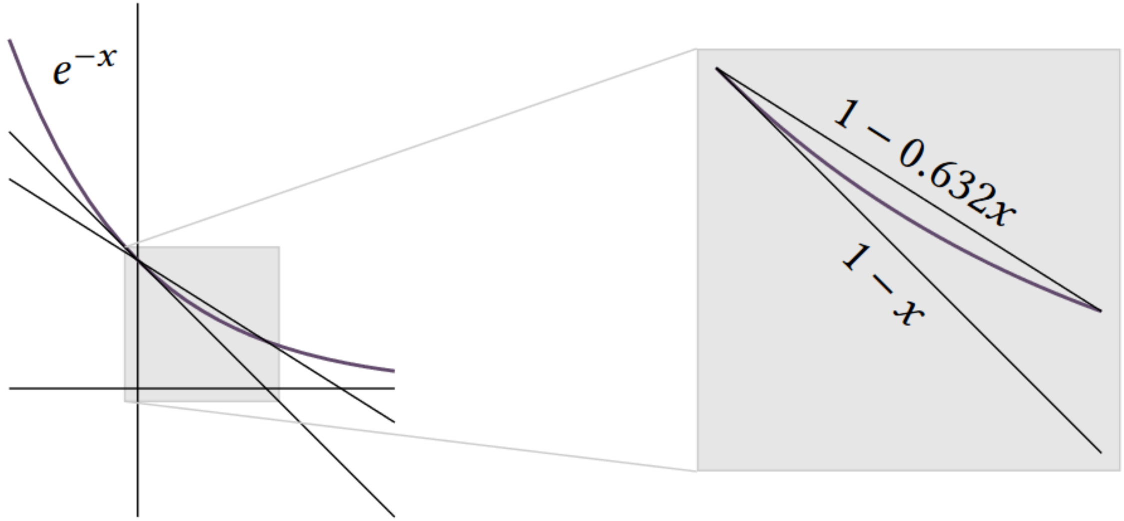
Lemma if $q \leq \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \leq \text{BirthdayProb}(q, N) \leq \frac{q(q-1)}{2N}$. This means that $\text{BirthdayProb}(q, N) = \Theta(\frac{q^2}{N})$

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 $1 - \frac{1}{e} = 0.632112 \ldots$



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 $0 \leq x \leq 1$,

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

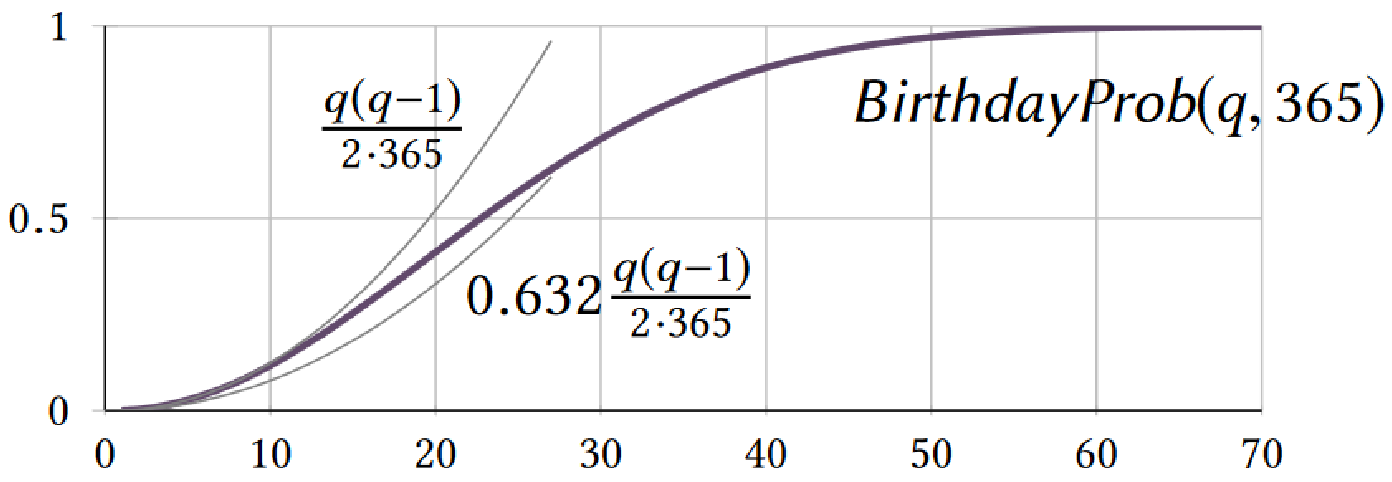
We have $\prod_{i=1}^{q-1}(1 - \frac{i}{N}) \leq \prod_{i=1}^{q-1} e^{-\frac{i}{N}} = e^{-\sum_{i=1}^{q-1}\frac{i}{N}} = e^{-\frac{q(q-1)}{2N}} \leq 1 - 0.632\frac{q(q-1)}{2N}$

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

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

Asymptotic Bounds on the Birthday Probability

Lemma if $q \le \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \le \text{BirthdayProb}(q,N) \le \frac{q(q-1)}{2N}$. 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**

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

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

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

```
R := ∅

SAMP():
  r ← {0,1}^λ \ R
  R := R ∪ {r}
  return r
```

- A natural way to distinguish them: call SAMP many times
  - If ever see a repeated output, then must be linked to $\mathcal{L}_{\text{samp−L}}$. 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 $\mathcal{L}_{\text{samp−R}}$.

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 $\text{BirthdayProb}(q, 2^\lambda)$.

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

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

proof

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

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

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

$R := \emptyset$
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

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

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

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

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

$\mathcal{L}_{\text{samp-L}}$	$\mathcal{L}_{\text{samp-R}}$
$\text{SAMP}(\mathcal{R} \subseteq \{0,1\}^\lambda)$:	$\text{SAMP}(\mathcal{R} \subseteq \{0,1\}^\lambda)$:
$r \leftarrow \{0,1\}^\lambda$	$r \leftarrow \{0,1\}^\lambda \setminus \mathcal{R}$
return 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: $k \leftarrow \{0,1\}^{\lambda}$
  - $\text{Enc}(k, m)$: $m \oplus k$
  - $\text{Dec}(k, c)$: $m \oplus c$
- To encrypt $m \in \{0,1\}^{2\lambda}$
- If we have a function $G: \{0,1\}^{\lambda} \rightarrow \{0,1\}^{2\lambda}$
  - $\text{Enc}(k, m) = G(k) \oplus m$
  - 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

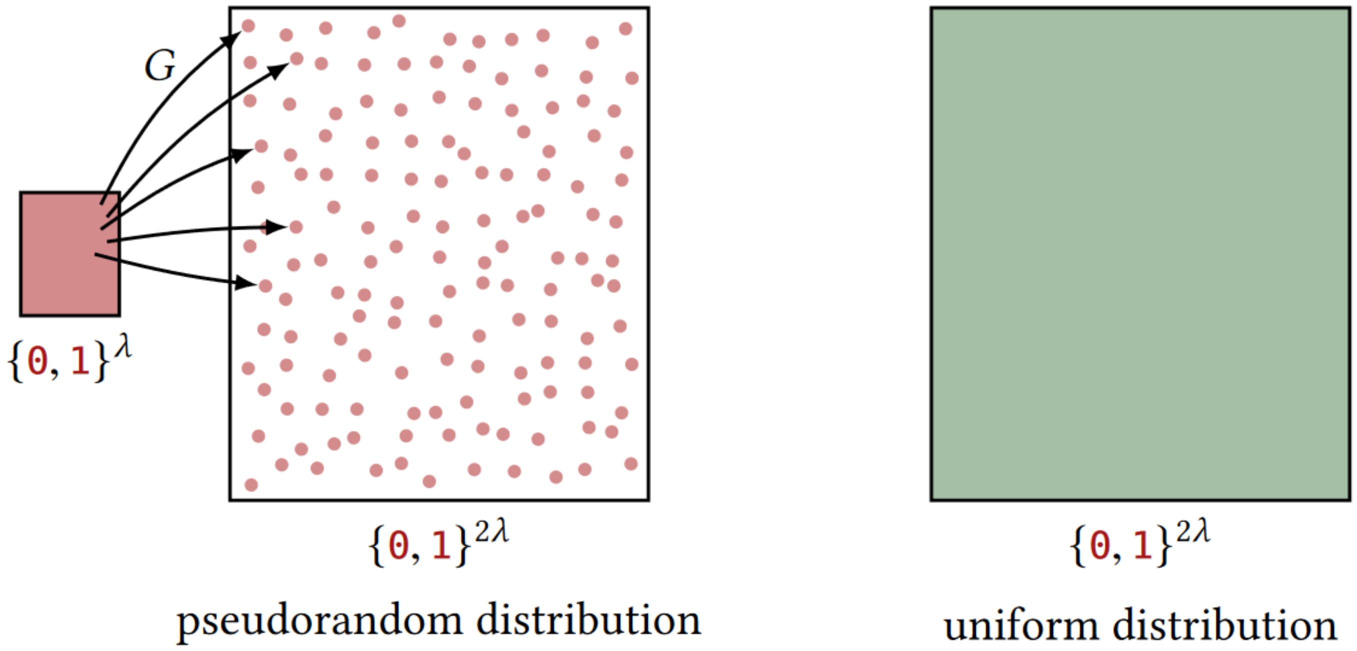
Definition Let $G\colon\{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be a deterministic function with $\ell > 0$. We say that G is a secure pseudorandom generator (PRG) if $\mathcal{L}^G_{\text{prg-real}} \approx \mathcal{L}^G_{\text{prg-rand}}$, where:

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

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

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

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

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

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

$$\underline{G(s):}$$
$$\quad \text{return } s\|s$$

- No!

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

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 $x \stackrel{?}{=} y$

$\diamond$

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

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

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

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 $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots}-\text{L}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots}-\text{R}} \Rightarrow 1]$.

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

EAVESDROP($m_L, m_R \in \Sigma.\mathcal{M}$):
- $k \leftarrow \Sigma.\mathsf{KeyGen}$
- $c \leftarrow \Sigma.\mathsf{Enc}(k, m_L)$
- return c

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

EAVESDROP($m_L, m_R \in \Sigma.\mathcal{M}$):
- $k \leftarrow \Sigma.\mathsf{KeyGen}$
- $c \leftarrow \Sigma.\mathsf{Enc}(k, m_R)$
- 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

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

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

$$
\begin{array}{|l|}
\hline
\underline{\textsc{eavesdrop}(m_L, m_R):} \\
\quad z \leftarrow \textsc{query}() \\
\quad c := z \oplus m_L \\
\quad \text{return } c \\
\hline
\end{array}
\diamond
\begin{array}{|l|}
\hline
\quad \mathcal{L}^{G}_{\text{prg-rand}} \\
\hline
\underline{\textsc{query}():} \\
\quad \boxed{r \leftarrow \{0,1\}^{\lambda+\ell}} \\
\quad \text{return } \boxed{r} \\
\hline
\end{array}
\quad \Rightarrow \quad
\begin{array}{|l|}
\hline
\quad \mathcal{L}^{\text{OTP}}_{\text{ots-L}} \\
\hline
\underline{\textsc{eavesdrop}(m_L, m_R):} \\
\quad \boxed{z \leftarrow \{0,1\}^{\lambda+\ell}} \\
\quad c := z \oplus m_L \\
\quad \text{return } c \\
\hline
\end{array}
$$

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

$\Rightarrow$

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

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

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

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

$\Rightarrow$

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

$\diamond$

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

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

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

$\diamond$

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

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

$\Rightarrow$

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

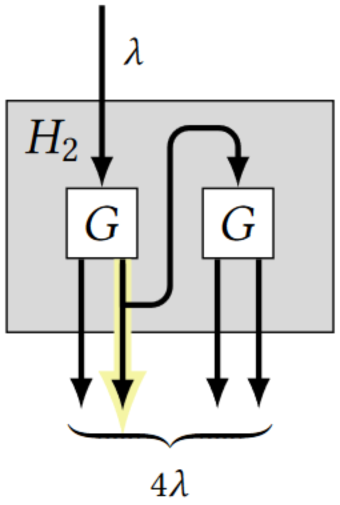
$\underline{\textsc{eavesdrop}(m_L, m_R):}$
$\quad k \leftarrow \{0,1\}^\lambda$
$\quad c := G(k) \oplus m_R$
$\quad$ return c

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 $x \| u \| v$