Pseudo-Random Generators

Casmali Lopez and Paisios Woodcock

Simulating Randomness with Binomials

July 17, 2022

PRG's

- We a random number.

PRG's

- We a random number.
- That's hard.

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers:

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers: deterministically generated (thus doable with code) and hard to predict (thus secure).

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers: deterministically generated (thus doable with code) and hard to predict (thus secure).
- A Pseudo-Randon Generator does this:

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers: deterministically generated (thus doable with code) and hard to predict (thus secure).
- A Pseudo-Randon Generator does this:
  Given n-bit input (seed), a PRG outputs a Q(n)-bit binary number.

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers: deterministically generated (thus doable with code) and hard to predict (thus secure).
- A Pseudo-Randon Generator does this:
  Given n-bit input (seed), a PRG outputs a Q(n)-bit binary number.
- I.e. given binary 'string' length n (seed), PRG's create length $Q(n) > n$ (expanded) pseudo-random binary sequences

PRG's

- We a random number.
- That's hard. Instead, we use Pseudo-random numbers: deterministically generated (thus doable with code) and hard to predict (thus secure).
- A Pseudo-Randon Generator does this:
  Given n-bit input (seed), a PRG outputs a Q(n)-bit binary number.
- I.e. given binary 'string' length n (seed), PRG's create length $Q(n) > n$ (expanded) pseudo-random binary sequences

Unpredictability

- A PRG $G(x_0)$, on input seed x_0, outputs $(y_1, \cdots, y_{Q(n)})$ such that:

Unpredictability

- A PRG $G(x_0)$, on input seed x_0, outputs $(y_1, \cdots, y_{Q(n)})$ such that: Given x_0, we can easily compute $(y_1, \cdots, y_{Q(n)})$, but it takes an adversary a long time to compute y_{i+1} given $y_1, \cdots, y_i$, but not x_0.

Unpredictability

- A PRG $G(x_0)$, on input seed x_0, outputs $(y_1, \cdots, y_{Q(n)})$ such that: Given x_0, we can easily compute $(y_1, \cdots, y_{Q(n)})$, but it takes an adversary a long time to compute y_{i+1} given $y_1, \cdots, y_i$, but not x_0.
- I.e. To predict with accuracy better than a half takes a long time.

Unpredictability

- A PRG $G(x_0)$, on input seed x_0, outputs $(y_1, \cdots, y_{Q(n)})$ such that: Given x_0, we can easily compute $(y_1, \cdots, y_{Q(n)})$, but it takes an adversary a long time to compute y_{i+1} given $y_1, \cdots, y_i$, but not x_0.
- I.e. To predict with accuracy better than a half takes a long time.
- Put one last way:

Unpredictability

- A PRG $G(x_0)$, on input seed x_0, outputs $(y_1, \cdots, y_{Q(n)})$ such that: Given x_0, we can easily compute $(y_1, \cdots, y_{Q(n)})$,
  but it takes an adversary a long time to compute y_{i+1} given $y_1, \cdots, y_i$, but not x_0.
- I.e. To predict with accuracy better than a half takes a long time.
- Put one last way: No algorithm running within a certain time limit can predict a next bit for a fraction much better than $\frac{1}{2}$ of all inputs (i.e. better than guessing).

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.
  I.e. For any polynomial $p(n)$,
  $\exists\ n_0$ such that $n > n_0 \implies f(n) > p(n)$.

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.
  I.e. For any polynomial $p(n)$,
  $\exists\ n_0$ such that $n > n_0 \implies f(n) > p(n)$.
- A $\Gamma\Upsilon$-PRG is a family of functions $G = \{G_n\}$,

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.
  I.e. For any polynomial $p(n)$,
  $\exists\, n_0$ such that $n > n_0 \implies f(n) > p(n)$.

- A $\Gamma\Upsilon$-PRG is a family of functions $G = \{G_n\}$,
  where $G_n : \{0,1\}^n \to \{0,1\}^{Q(n)}$ and $Q(n) > n$,

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.
  I.e. For any polynomial $p(n)$,
  $\exists\ n_0$ such that $n > n_0 \implies f(n) > p(n)$.
- A $\Gamma\Upsilon$-PRG is a family of functions $G = \{G_n\}$,
  where $G_n : \{0,1\}^n \to \{0,1\}^{Q(n)}$ and $Q(n) > n$,
  s.t. for any algorithm $A \in \Gamma$ and any $i \in \{1, \cdots, Q(n) - 1\}$,
  $|Prob_{x_0 \in \{0,1\}^n}(A[y_1, \cdots, y_i] = y_{i+1}) - \frac{1}{2}| \in \frac{1}{n^{\omega(1)}}$,

Details

- A function $f(n) \in n^{\omega(1)}$ iff $f(n)$ is asympototically bigger than any polynomial in n.
  I.e. For any polynomial $p(n)$,
  $\exists\ n_0$ such that $n > n_0 \implies f(n) > p(n)$.

- A $\Gamma\Upsilon$-PRG is a family of functions $G = \{G_n\}$,
  where $G_n : \{0,1\}^n \to \{0,1\}^{Q(n)}$ and $Q(n) > n$,
  s.t. for any algorithm $A \in \Gamma$ and any $i \in \{1, \cdots, Q(n) - 1\}$,
  $|Prob_{x_0 \in \{0,1\}^n}(A[y_1, \cdots, y_i] = y_{i+1}) - \frac{1}{2}| \in \frac{1}{n^{\omega(1)}}$,
  and G computes in time on $O(\Upsilon)$ (in n).

Significance

- Applications include Procedural Simulations of Nature

- Real Applications typically use (mathematically speaking) pretty horrible PRG's.
- Hackers can know your method of generating... just not the seed. Keeping the seed hidden is what matters most. Humans choose the seed.
- Symmetric Key Cryptography Applications (seed is key)

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique:

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

  Take $f(x), f(f(x)), \cdots, f^{(Q(n))}(x)$.

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

  Take $f(x), f(f(x)), \cdots, f^{(Q(n))}(x)$.

  Take $B(f(x)), B(f(f(x))), \cdots, B(f^{(Q(n))})$

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

  Take $f(x), f(f(x)), \cdots, f^{(Q(n))}(x)$.

  Take $B(f(x)), B(f(f(x))), \cdots, B(f^{(Q(n))})$, and REVERSE order!

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

  Take $f(x), f(f(x)), \cdots, f^{(Q(n))}(x)$.

  Take $B(f(x)), B(f(f(x))), \cdots, B(f^{(Q(n))})$, and REVERSE order!
- $G_n(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x

Blum-Micali PRG

- Friendly Function f (just a function) and Predicate Function B.
- Predicate: Maps numbers to a single bit,
- PRG Technique: For length n seed (n-bit binary number) x,

  Take $f(x), f(f(x)), \cdots, f^{(Q(n))}(x)$.

  Take $B(f(x)), B(f(f(x))), \cdots, B(f^{(Q(n))})$, and REVERSE order!
- $G_n(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x
- Treats binary expansion of x as n-bit sequence

Example

- Blum and Micali use $f(x) = g^x$, where g is a generator for $\mathbb{F}_p^*$.

Example

- Blum and Micali use $f(x) = g^x$, where g is a generator for $\mathbb{F}_p^*$. Choice of seed includes x and g.
- $B(x) = 1$ iff the smallest s satisfying $x = g^s$ has $s \geq \frac{p-1}{2}$.

Example

- Blum and Micali use $f(x) = g^x$, where g is a generator for $\mathbb{F}_p^*$. Choice of seed includes x and g.
- $B(x) = 1$ iff the smallest s satisfying $x = g^s$ has $s \geq \frac{p-1}{2}$. Else 0.

Example

- Blum and Micali use $f(x) = g^x$, where g is a generator for $\mathbb{F}_p^*$. Choice of seed includes x and g.
- $B(x) = 1$ iff the smallest s satisfying $x = g^s$ has $s \geq \frac{p-1}{2}$. Else 0.
- Equivalently, $B_{p,g}(y) = 1$ iff y is the principal square root of y^2 (mod p).

Example

- Blum and Micali use $f(x) = g^x$, where g is a generator for $\mathbb{F}_p^*$. Choice of seed includes x and g.
- $B(x) = 1$ iff the smallest s satisfying $x = g^s$ has $s \geq \frac{p-1}{2}$. Else 0.
- Equivalently, $B_{p,g}(y) = 1$ iff y is the principal square root of y^2 (mod p).
- So $G(x) = (B(g^{g^{\cdots^{g^x}}}), \cdots, B(g^{g^{g^x}}), B(g^{g^x}), B(g^x))$.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.
- Roughly, a PRG has two requirements:

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.
- Roughly, a PRG has two requirements:
  1) $x \to B(f(x))$ (computing $B(f(x))$ from x) is easy

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.
- Roughly, a PRG has two requirements:
  1) $x \to B(f(x))$ (computing $B(f(x))$ from x) is easy
  2) $x \to B(x)$ cannot be easily computed (Why?)

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.
- Roughly, a PRG has two requirements:
  1) $x \to B(f(x))$ (computing $B(f(x))$ from x) is easy
  2) $x \to B(x)$ cannot be easily computed (Why?)
- Just given a $y \in \mathbb{F}_p^*$, deciding whether y is the principal square root of y^2 is very hard.

Predicate

- For unpredictability, want half of domain to have $B(x) = 1$, and other half $B(x) = 0$. Otherwise, easy to predict with probability $> \frac{1}{2}$.
- Want $x \to B(f(x))$ easy to compute so we can generate the sequence.
- Indeed, $B(f(x)) = B(g^x)$ easy: simply ask whether $x \geq \frac{p-1}{2}$.
- Roughly, a PRG has two requirements:
  1) $x \to B(f(x))$ (computing $B(f(x))$ from x) is easy
  2) $x \to B(x)$ cannot be easily computed (Why?)
- Just given a $y \in \mathbb{F}_p^*$, deciding whether y is the principal square root of y^2 is very hard.

The Why

- Efficiently predicting the next bit of our sequence

The Why

- Efficiently predicting the next bit of our sequence is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$

The Why

- Efficiently predicting the next bit of our sequence is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$
- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.

The Why

- Efficiently predicting the next bit of our sequence is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$
- Suppose we can predict $(i+1)^{st}$ bits from the first i bits. Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since $x \to B(f(x))$ easy,

The Why

- Efficiently predicting the next bit of our sequence is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$
- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.
  Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since $x \to B(f(x))$ easy,
  and then we can predict $B(x)$ (by bullet one);

The Why

- Efficiently predicting the next bit of our sequence
  is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$
- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.
  Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since
  $x \to B(f(x))$ easy,
  and then we can predict $B(x)$ (by bullet one);
  Hence $x \to B(x)$ is efficiently computable!

The Why

- Efficiently predicting the next bit of our sequence is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$
- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.
  Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since $x \to B(f(x))$ easy,
  and then we can predict $B(x)$ (by bullet one);
  Hence $x \to B(x)$ is efficiently computable!
- That is:
  Next bit predictable $\implies$ $x \to B(x)$ easy.

The Why

- Efficiently predicting the next bit of our sequence
  is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$

- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.
  Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since
  $x \to B(f(x))$ easy,
  and then we can predict $B(x)$ (by bullet one);
  Hence $x \to B(x)$ is efficiently computable!

- That is:
  Next bit predictable $\implies x \to B(x)$ easy.

  So, by contrapositive:

The Why

- Efficiently predicting the next bit of our sequence
  is equivalent to finding $B(x)$ from $B(f^{(i)}(x)), \cdots, B(f(x))$

- Suppose we can predict $(i+1)^{st}$ bits from the first i bits.
  Then, given x, we can efficiently find $B(f(x)), \cdots, B(f^{(i)}(x))$, since $x \to B(f(x))$ easy,
  and then we can predict $B(x)$ (by bullet one);
  Hence $x \to B(x)$ is efficiently computable!

- That is:
  Next bit predictable $\implies x \to B(x)$ easy.

  So, by contrapositive:
  $x \to B(x)$ hard $\implies$ Our PRG Sequence is Unpredictable!

Efficient v. Hard

- By "easy" or "hard", we mean:

Efficient v. Hard

- By "easy" or "hard", we mean: Is it computable in time on the Order of a certain function?

Efficient v. Hard

- By "easy" or "hard", we mean: Is it computable in time on the Order of a certain function?
- $f(n) \in O(g(n))$ iff $\exists c \in \mathbb{R}, n_0 \in \mathbb{N}$ s.t.

Efficient v. Hard

- By "easy" or "hard", we mean: Is it computable in time on the Order of a certain function?
- $f(n) \in O(g(n))$ iff $\exists c \in \mathbb{R}, n_0 \in \mathbb{N}$ s.t. $n > n_0 \implies f(n) \leq cg(n)$

Efficient v. Hard

- By "easy" or "hard", we mean: Is it computable in time on the Order of a certain function?
- $f(n) \in O(g(n))$ iff $\exists c \in \mathbb{R}, n_0 \in \mathbb{N}$ s.t.

  $n > n_0 \implies f(n) \leq cg(n)$

  I.e. $f \in O(g)$ if f is eventually bounded above by some multiple of g.

Efficient v. Hard

- By "easy" or "hard", we mean: Is it computable in time on the Order of a certain function?
- $f(n) \in O(g(n))$ iff $\exists c \in \mathbb{R}, n_0 \in \mathbb{N}$ s.t.

  $n > n_0 \implies f(n) \leq cg(n)$

  I.e. $f \in O(g)$ if f is eventually bounded above by some multiple of g.
- Note: Γ is the set of algorithms computable on $O(F_i)$ for some function F_i in a family of functions F.

Binomials

- Let $f(x) = x^a + cx^b$

Binomials

- Let $f(x) = x^a + cx^b \in \mathbb{F}_p^*[x]$,
- Can we use this as a friendly function?

Binomials

- Let $f(x) = x^a + cx^b \in \mathbb{F}_p^*[x]$,
- Can we use this as a friendly function?
- That is: does there exist predicate B such that:

Binomials

- Let $f(x) = x^a + cx^b \in \mathbb{F}_p^*[x]$,
- Can we use this as a friendly function?
- That is: does there exist predicate B such that:

  $x \to B(x^a + cx^b)$ is easy, but

Binomials

- Let $f(x) = x^a + cx^b \in \mathbb{F}_p^*[x]$,
- Can we use this as a friendly function?
- That is: does there exist predicate B such that:

  $x \to B(x^a + cx^b)$ is easy, but
  $x \to B(x)$ is hard?
- If we can find a predicate meeting certain conditions (to be seen),

Binomials

- Let $f(x) = x^a + cx^b \in \mathbb{F}_p^*[x]$,
- Can we use this as a friendly function?
- That is: does there exist predicate B such that:

  $x \to B(x^a + cx^b)$ is easy, but
  $x \to B(x)$ is hard?
- If we can find a predicate meeting certain conditions (to be seen), then yes: this f can make a PRG.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?
- Goal: We want to use Hardness of Solving Trinomials to make this binomial a friendly function for a PRG.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?
- Goal: We want to use Hardness of Solving Trinomials to make this binomial a friendly function for a PRG.
- Method: For $f = x^a + cx^b$, find B such that:

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?
- Goal: We want to use Hardness of Solving Trinomials to make this binomial a friendly function for a PRG.
- Method: For $f = x^a + cx^b$, find B such that:

  If $x \to B(x)$ easy, then finding x from $f(x)$ is easy.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?
- Goal: We want to use Hardness of Solving Trinomials to make this binomial a friendly function for a PRG.
- Method: For $f = x^a + cx^b$, find B such that:

  If $x \to B(x)$ easy, then finding x from $f(x)$ is easy.
  Hence if solving Trinomials is Hard, (by contrapositive) the PRG condition "$x \to B(x)$ is hard" is satisfied.

Trinomial Hardness

- Now suppose we're given $y \in \mathbb{F}_p^*$.
  How hard is it to find x s.t. $x^a + cx^b = y$ (mod p)? $\mathbb{F}_p^*$.
- Or: How long does it take to solve $x^a + cx^b - y$ over $\mathbb{F}_p^*$?
- Goal: We want to use Hardness of Solving Trinomials to make this binomial a friendly function for a PRG.
- Method: For $f = x^a + cx^b$, find B such that:

  If $x \to B(x)$ easy, then finding x from $f(x)$ is easy.
  Hence if solving Trinomials is Hard, (by contrapositive) the PRG condition "$x \to B(x)$ is hard" is satisfied.
- Conditional Result.

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that
  knowing $y \to B(y)$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that
  knowing $y \to B(y)$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs
  (where P is some polynomial),

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that
  knowing $y \to B(y)$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs
  (where P is some polynomial),
  would allow us to efficiently predict the root of $x^a + cx^b - y$

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that
  knowing $y \to B(y)$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs
  (where P is some polynomial),
  would allow us to efficiently predict the root of $x^a + cx^b - y$
  for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs.

Goal

- By easy, we mean doable for a certain fraction better than half of all inputs.
- We want a predicate B such that knowing $y \to B(y)$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs (where P is some polynomial), would allow us to efficiently predict the root of $x^a + cx^b - y$ for a fraction $> \frac{1}{2} + \frac{1}{P(n)}$ of y inputs. (Where p is n-bit)

Note

- What is the fastest known algorithm for solving this trinomial for a root over $\mathbb{F}_p^*$
- $\sqrt{p}$-time. That is, $2^{\frac{n}{2}}$-time.

Inputs (Technicalities)

- $D_p \subseteq \mathbb{F}_p^*$ is the set of inputs to our friendly function f_p and predicate B_p (dependent on the prime p).
- Size-n inputs to Predicate and Friendly Function:

Inputs (Technicalities)

- $D_p \subseteq \mathbb{F}_p^*$ is the set of inputs to our friendly function f_p and predicate B_p (dependent on the prime p).
- Size-n inputs to Predicate and Friendly Function: $I_n = \{(p, x)|p$ is n-bit prime $x \in D_p\}$

Inputs (Technicalities)

- $D_p \subseteq \mathbb{F}_p^*$ is the set of inputs to our friendly function f_p and predicate B_p (dependent on the prime p).
- Size-n inputs to Predicate and Friendly Function:
  $I_n = \{(p, x) | p$ is n-bit prime $x \in D_p\}$
- Friendly function and predicate are sets of functions, dependent on input length n.

Terms

- General Approach: If $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$,

Terms

- General Approach: If $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$, We need $B(x)$ to be unpredictable when given x.

Terms

- General Approach: If $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$, We need $B(x)$ to be unpredictable when given x.
- If f is sufficiently 'random' under iteration, why not just use $G(x) = (f(x), \cdots, f^{(Q(n))}(x))$?

Terms

- General Approach: If $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$, We need $B(x)$ to be unpredictable when given x.
- If f is sufficiently 'random' under iteration, why not just use $G(x) = (f(x), \cdots, f^{(Q(n))}(x))$?
- Need binary output, so have to choose a digit from each.
- And! Hacker can see our function f! PRG's need to be that strong.

Terms

- General Approach: If $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$, We need $B(x)$ to be unpredictable when given x.
- If f is sufficiently 'random' under iteration, why not just use $G(x) = (f(x), \cdots, f^{(Q(n))}(x))$?
- Need binary output, so have to choose a digit from each.
- And! Hacker can see our function f! PRG's need to be that strong.
- Importance of seed in Symmetric Key Cryptography applications. How it's generated. Knowing seed is everything!

Accessibility

Def A predicate B is v-accessible if there is a probabilistic algorithm with expected run time $v(n)$ such that, on input an n-bit integer, the algorithm outputs some $(p, x) \in I_n$ with uniform probability among elements of I_n;

Accessibility

Def A predicate B is v-accessible if there is a probabilistic algorithm with expected run time $v(n)$ such that, on input an n-bit integer, the algorithm outputs some $(p, x) \in I_n$ with uniform probability among elements of I_n;
or, when it doesn't, it outputs nothing, but only with probability $\frac{1}{2^c}$ for some constant c.

- So: A predicate is v-Accessible if its n-bit inputs can be randomly, uniformly sampled from n-bit integers in time $v(n)$; but it allows possibility that there is a small chance your sampling algorithm doesn't work.

Accessibility

Def A predicate B is v-accessible if there is a probabilistic algorithm with expected run time $v(n)$ such that, on input an n-bit integer, the algorithm outputs some $(p, x) \in I_n$ with uniform probability among elements of I_n;
or, when it doesn't, it outputs nothing, but only with probability $\frac{1}{2^c}$ for some constant c.

- So: A predicate is v-Accessible if its n-bit inputs can be randomly, uniformly sampled from n-bit integers in time $v(n)$; but it allows possibility that there is a small chance your sampling algorithm doesn't work.
- Typically defined other way:
  v is however long it takes to sample inputs uniformly.

Accessability

- Why accessability?

Accessability

- Why accessability? Our PRG takes any n-bit input, but since f_p (our friendly function) has to be a permutation on the input (in order to make a PRG), we must restrict the input to some D_p.

Accessability

- Why accessability? Our PRG takes any n-bit input, but since f_p (our friendly function) has to be a permutation on the input (in order to make a PRG), we must restrict the input to some D_p.
- So, given random n-bit integer, we need to quickly get a 'random' n-bit input to our friendly function in order to calculate the PRG.

Unapproximability

Def A predicate B is Γ-unapproximable if no algorithm in Γ can correctly compute $B(x)$ from x for more than a fraction $\frac{1}{2} + \frac{1}{P(n)}$ of all n-bit inputs (p, x), for any polynomial P.

Unapproximability

Def A predicate B is Γ-unapproximable if no algorithm in Γ can correctly compute $B(x)$ from x for more than a fraction $\frac{1}{2} + \frac{1}{P(n)}$ of all n-bit inputs (p, x), for any polynomial P.

- Basically, output of predicate is "unpredictable" (accuracy better than guessing requires enormous computation)

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes
- $f : (p, x) \to D_p$ calculates in time on the order of some function from a family of functions Υ ("efficiently computable")

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes
- $f : (p, x) \to D_p$ calculates in time on the order of some function from a family of functions Υ ("efficiently computable")
- $h : (p, x) \in I \to B_p(f_p(x))$ also in Υ ("$x \to B_p(f(x))$ easy")

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes
- $f : (p, x) \to D_p$ calculates in time on the order of some function from a family of functions Υ ("efficiently computable")
- $h : (p, x) \in I \to B_p(f_p(x))$ also in Υ ("$x \to B_p(f(x))$ easy")
- B is υ-accessible, where $\upsilon \in O(\Upsilon)$

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes
- $f : (p, x) \to D_p$ calculates in time on the order of some function from a family of functions Υ ("efficiently computable")
- $h : (p, x) \in I \to B_p(f_p(x))$ also in Υ ("$x \to B_p(f(x))$ easy")
- B is υ-accessible, where $\upsilon \in O(\Upsilon)$
- B is Γ-unapproximable. ("$x \to B_p(x)$ hard")

Generalized Sufficient Conditions

[Theorem] Sufficient conditions to form a PRG are:

- $f_p \equiv f(p, \cdot) : D_p \to D_p$ be a permutation for all n-bit primes
- $f : (p, x) \to D_p$ calculates in time on the order of some function from a family of functions Υ ("efficiently computable")
- $h : (p, x) \in I \to B_p(f_p(x))$ also in Υ ("$x \to B_p(f(x))$ easy")
- B is υ-accessible, where $\upsilon \in O(\Upsilon)$
- B is Γ-unapproximable. ("$x \to B_p(x)$ hard")
- $\Gamma \supseteq \Upsilon$ (otherwise it may be more easily broken than computed.)

Words

In words: if you want a PRG made this way...

- f_p depends on p, so to efficiently compute, need to find f_p quick and calculate $B(f_p(x))$ quick.

Words

In words: if you want a PRG made this way...

- f_p depends on p, so to efficiently compute, need to find f_p quick and calculate $B(f_p(x))$ quick.
- Need f_p to be a permutation on D_p.

Words

In words: if you want a PRG made this way...

- f_p depends on p, so to efficiently compute, need to find f_p quick and calculate $B(f_p(x))$ quick.
- Need f_p to be a permutation on D_p.
- Need D_p (input to f_p) efficiently randomly sample-able

Words

In words: if you want a PRG made this way...

- f_p depends on p, so to efficiently compute, need to find f_p quick and calculate $B(f_p(x))$ quick.
- Need f_p to be a permutation on D_p.
- Need D_p (input to f_p) efficiently randomly sample-able
- The time it takes to compute $f_p(x)$ and $B_p(f_p(x))$ are both on the order of the time it takes to 'access' D_p.

Proof of Sufficient Conditions

- $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x
- Need to prove G can be generated in time on $O(\Upsilon)$ and resists next-bit prediction by algorithms on $O(\Gamma)$.

Proof of Sufficient Conditions

- $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x
- Need to prove G can be generated in time on $O(\Upsilon)$ and resists next-bit prediction by algorithms on $O(\Gamma)$.

Brief Proof Outline:

Proof of Sufficient Conditions

- $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x
- Need to prove G can be generated in time on $O(\Upsilon)$ and resists next-bit prediction by algorithms on $O(\Gamma)$.

Brief Proof Outline:

- Generating $G(x)$ requires $Q(n)$ calculations of $B_p(f_p(x))$ and $f(p, \cdot)$, i.e. calculating f and $h_p \in \Upsilon$.
  Thus generating $G(x)$ takes time in Υ.

Proof of Sufficient Conditions

- $G(x) = (B(f^{(Q(n))}(x)), \cdots, B(f(x)))$ for n-bit seed x
- Need to prove G can be generated in time on $O(\Upsilon)$ and resists next-bit prediction by algorithms on $O(\Gamma)$.

Brief Proof Outline:

- Generating $G(x)$ requires $Q(n)$ calculations of $B_p(f_p(x))$ and $f(p, \cdot)$, i.e. calculating f and $h_p \in \Upsilon$.
  Thus generating $G(x)$ takes time in Υ.
- If there exists next-bit prediction algorithm in Γ, then use this algorithm to predict $B(f^{(i+1)}(x))$ from $B(f^{(i)}(x))$: i.e. predict $B(x)$ from x. This contradicts unapproximability (unpredictable output)!

Pause

- What are we doing?

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.
- This means algorithms running in certain times can't predict with certain accuracy.

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.
- This means algorithms running in certain times can't predict with certain accuracy.
- Γ is our measure of "certain times", i.e. the strength of algorithms that cannot predict our sequence.

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.
- This means algorithms running in certain times can't predict with certain accuracy.
- Γ is our measure of "certain times", i.e. the strength of algorithms that cannot predict our sequence.
  Υ, the time it takes to generate the PRG, sets bounds on this Γ,

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.
- This means algorithms running in certain times can't predict with certain accuracy.
- Γ is our measure of "certain times", i.e. the strength of algorithms that cannot predict our sequence.
  Υ, the time it takes to generate the PRG, sets bounds on this Γ, because $\Upsilon \subseteq \Gamma$.
  The a, b, c, D_p (with $f_p(x) = x^a + cx^b$ being permutation on D_p) determine Υ

Pause

- What are we doing?
- Reminder: we want to generate unpredictable binary strings.
- This means algorithms running in certain times can't predict with certain accuracy.
- Γ is our measure of "certain times", i.e. the strength of algorithms that cannot predict our sequence.
  Υ, the time it takes to generate the PRG, sets bounds on this Γ, because $\Upsilon \subseteq \Gamma$.
  The a, b, c, D_p (with $f_p(x) = x^a + cx^b$ being permutation on D_p) determine Υ
- Studying what choice of Γ and Υ will work shows how good (if possible) our PRG is.

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.
- Let $\delta(n)$ be the fastest time it takes to compute a root of $x^a + cx^b$ in $\mathbb{F}_p^*$, where p is n-bits.

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.
- Let $\delta(n)$ be the fastest time it takes to compute a root of $x^a + cx^b$ in $\mathbb{F}_p^*$, where p is n-bits.
- Let Δ be all functions on $O(\delta(n))$.

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.
- Let $\delta(n)$ be the fastest time it takes to compute a root of $x^a + cx^b$ in $\mathbb{F}_p^*$, where p is n-bits.
- Let Δ be all functions on $O(\delta(n))$.
- Recall the definition:

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.
- Let $\delta(n)$ be the fastest time it takes to compute a root of $x^a + cx^b$ in $\mathbb{F}_p^*$, where p is n-bits.
- Let Δ be all functions on $O(\delta(n))$.
- Recall the definition:

  G is a $\Gamma\Upsilon$-PRG if no algorithm in Γ can predict with accuracy $\frac{1}{2} + \frac{1}{P(n)}$ for any polynomial P, and G runs in time on $O(\Upsilon)$.

Neccessary Conditions on PRG's

- Another thing that limits our choice of Γ is what we call Δ.
- Let $\delta(n)$ be the fastest time it takes to compute a root of $x^a + cx^b$ in $\mathbb{F}_p^*$, where p is n-bits.
- Let Δ be all functions on $O(\delta(n))$.
- Recall the definition:

  G is a $\Gamma\Upsilon$-PRG if no algorithm in Γ can predict with accuracy $\frac{1}{2} + \frac{1}{P(n)}$ for any polynomial P, and G runs in time on $O(\Upsilon)$.

Delta

- Say $\Delta \subseteq \Gamma$.

Delta

- Say $\Delta \subseteq \Gamma$.
  Then $\delta \in\ O(\Gamma)$, so we cannot predict $B(y)$ give y (better than guessing).

Delta

- Say $\Delta \subseteq \Gamma$.
  Then $\delta \in \ O(\Gamma)$, so we cannot predict $B(y)$ give y (better than guessing).
  But given y, find x s.t. $f(x) = y$, then $B(f(x)) = B(y)$.

Delta

- Say $\Delta \subseteq \Gamma$.
  Then $\delta \in O(\Gamma)$, so we cannot predict $B(y)$ give y (better than guessing).
  But given y, find x s.t. $f(x) = y$, then $B(f(x)) = B(y)$.
  Doable in time $O(\delta(n)) + O(\Upsilon)$, so on $O(\Upsilon)$.

Delta

- Say $\Delta \subseteq \Gamma$.
  Then $\delta \in\ O(\Gamma)$, so we cannot predict $B(y)$ give y (better than guessing).
  But given y, find x s.t. $f(x) = y$, then $B(f(x)) = B(y)$.
  Doable in time $O(\delta(n)) + O(\Upsilon)$, so on $O(\Upsilon)$.
  Contradiction!
- Thus we must have $\Gamma \subsetneq \Delta$.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2\log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p?

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p? Is this easy?
- Good questions.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p? Is this easy?
- Good questions. That's where the bulk of computation time goes.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p? Is this easy?
- Good questions. That's where the bulk of computation time goes.
- The thing which may keep this binomial from making a PRG is it being "too expensive" to systematically find the a, b, c, D_p such that f_p is a permutation on D_p.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p? Is this easy?
- Good questions. That's where the bulk of computation time goes.
- The thing which may keep this binomial from making a PRG is it being "too expensive" to systematically find the a, b, c, D_p such that f_p is a permutation on D_p.
- Especially because $|D_p| \geq Q(n)$.

Current Bound

- Current Bounds: Calculating $f_p(x)$ and $h_p(x)$ take time on $O(n^2 \log(n))$.
- How do we choose the a, b, c, D_p to make $f_p(x) = x^a + cx^b$ a permutation on D_p? Is this easy?
- Good questions. That's where the bulk of computation time goes.
- The thing which may keep this binomial from making a PRG is it being "too expensive" to systematically find the a, b, c, D_p such that f_p is a permutation on D_p.
- Especially because $|D_p| \geq Q(n)$.
- Can't begin being periodic too quickly, so must have bigger range of outputs of $f(x)$ than elements in the outputted sequence.

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.
- If calculating f_p and finding D_p take too long, Γ skyrockets,

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.
- If calculating f_p and finding D_p take too long, Γ skyrockets, then we would need to show: No algorithm on this huge Γ runtime can predict with good accuracy.

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.
- If calculating f_p and finding D_p take too long, Γ skyrockets, then we would need to show: No algorithm on this huge Γ runtime can predict with good accuracy.
- As Γ increases, this becomes a stronger and stronger statement.

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.
- If calculating f_p and finding D_p take too long, Γ skyrockets, then we would need to show: No algorithm on this huge Γ runtime can predict with good accuracy.
- As Γ increases, this becomes a stronger and stronger statement.
- At very least, need f_p and D_p computable in time on smaller order than δ (generate faster than break).

Too Expensive?

- We need $\Upsilon \subseteq \Gamma \subsetneq \Delta$.
- If calculating f_p and finding D_p take too long, Γ skyrockets, then we would need to show: No algorithm on this huge Γ runtime can predict with good accuracy.
- As Γ increases, this becomes a stronger and stronger statement.
- At very least, need f_p and D_p computable in time on smaller order than δ (generate faster than break).
- A lower bound on Υ is $n^2 \log(n)$ (time to calculate each $f_p(x)$ when a, b, c, D_p are known).

Sanity Check

- Reminder: We're assessing Γ and Υ to see whether binomials can generate PRG's. And D_p determines Υ, which determines whether there is a Γ to work.

Bounds

- Suppose finding a root of $f(x) = x^a + cx^b$ is doable in time on $O(n^2 log(n))$
  ; that is, trinomials are solvable in time on $O(\log^2(p)\log(\log(p)))$.

Bounds

- Suppose finding a root of $f(x) = x^a + cx^b$ is doable in time on $O(n^2 log(n))$
  ; that is, trinomials are solvable in time on $O(\log^2(p)\log(\log(p)))$.
- Then f cannot be used to create a PRG (for any Γ, Φ, Υ, or B)!

Bounds

- Suppose finding a root of $f(x) = x^a + cx^b$ is doable in time on $O(n^2 log(n))$
  ; that is, trinomials are solvable in time on $O(\log^2(p)\log(\log(p)))$.
- Then f cannot be used to create a PRG (for any Γ, Φ, Υ, or B)!
- In fact, if finding the root of a d-degree t-nomial f over $\mathbb{F}_p^*$ is doable in time on $O(t\log^2(p)\log(\log(p)))$,

Bounds

- Suppose finding a root of $f(x) = x^a + cx^b$ is doable in time on $O(n^2 log(n))$
  ; that is, trinomials are solvable in time on $O(\log^2(p)\log(\log(p)))$.
- Then f cannot be used to create a PRG (for any Γ, Φ, Υ, or B)!
- In fact, if finding the root of a d-degree t-nomial f over $\mathbb{F}_p^*$ is doable in time on $O(t\log^2(p)\log(\log(p))$, then f cannot be used as a friendly function (ever).

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:
- In time on $O(p \log^2(p) \log(\log(p)))$, we need to systematically choose a, b, c, and $D_p \subseteq \mathbb{F}_p^*$ such that $f_p(x) = x^a + cx^b$ is a permutation on D_p.

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:
- In time on $O(p \log^2(p) \log(\log(p)))$, we need to systematically choose a, b, c, and $D_p \subseteq \mathbb{F}_p^*$ such that $f_p(x) = x^a + cx^b$ is a permutation on D_p.
- What algorithm works?

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:
- In time on $O(p \log^2(p) \log(\log(p)))$, we need to systematically choose a, b, c, and $D_p \subseteq \mathbb{F}_p^*$ such that $f_p(x) = x^a + cx^b$ is a permutation on D_p.
- What algorithm works? We don't know any.

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:
- In time on $O(p\log^2(p)\log(\log(p)))$, we need to systematically choose a, b, c, and $D_p \subseteq \mathbb{F}_p^*$ such that $f_p(x) = x^a + cx^b$ is a permutation on D_p.
- What algorithm works? We don't know any.But statistically speaking, "good" choices are hard to come by.

Importance of D_p

- D_p is the restriction of $\mathbb{F}_p^*$ such that $x^a + cx^b$ is a permutation on D_p.
- To decide whether this binomial can be used for PRG's, one prerequisite is thus:
- In time on $O(p \log^2(p) \log(\log(p)))$, we need to systematically choose a, b, c, and $D_p \subseteq \mathbb{F}_p^*$ such that $f_p(x) = x^a + cx^b$ is a permutation on D_p.
- What algorithm works? We don't know any.But statistically speaking, "good" choices are hard to come by.
- D_p will be a subset of $\mathbb{F}_p^*$ that forms a cycle under f_p, so this boils down to studying cycle lengths and frequencies of $x^a + cx^b \in \mathbb{F}_p^*[x]$.

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds,
  and to choose D_p, we need to actually iterate through f_p until reaching a repeat.

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds,
  and to choose D_p, we need to actually iterate through f_p until reaching a repeat.
- Need cycle length at least $Q(n)$,

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds,
  and to choose D_p, we need to actually iterate through f_p until reaching a repeat.
- Need cycle length at least $Q(n)$, but Cycle Length + Pre-Period Length less than $O(p)$

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds,
  and to choose D_p, we need to actually iterate through f_p until reaching a repeat.
- Need cycle length at least $Q(n)$, but Cycle Length + Pre-Period Length less than $O(p)$, lest calculating D_p takes time on $O(\Delta)$ and the whole PRG is useless.

D_p constraints

- By technical definitions, we only need a PRG to be unpredictable for "almost all" input seeds,
  and to choose D_p, we need to actually iterate through f_p until reaching a repeat.
- Need cycle length at least $Q(n)$, but Cycle Length + Pre-Period Length less than $O(p)$, lest calculating D_p takes time on $O(\Delta)$ and the whole PRG is useless.
- Study pre-period and closest-cycle lengths for elements on $\mathbb{F}_p^*$

Frequency of Good a, b, c Choices

- So.... What do know about these cycles in F_p?
- The following slides represent some experimental results for various $f(x)$
- $f(x)$ over the Field
- Example of iterating $f(x)$
- Discrete Fourier Analysis(discrepancy) of iteration
- Functional Graph of $f(x)$

Graphs DLP

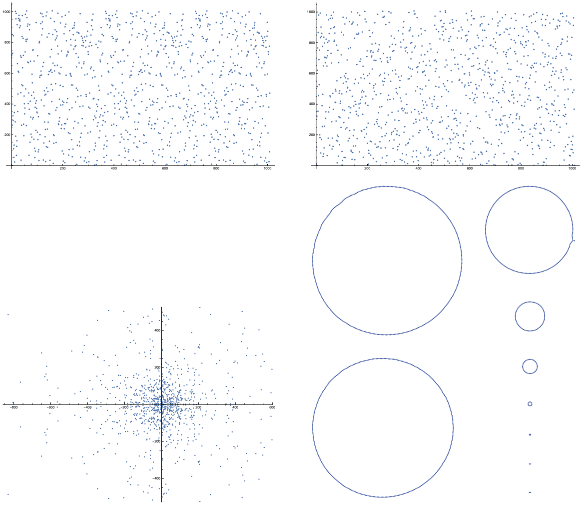

Figure: $f(x) = 11^x \mod 1009$, $p = 1009$, Itervalue: 582(top left), Number of Components: 10

Graphs Binomial

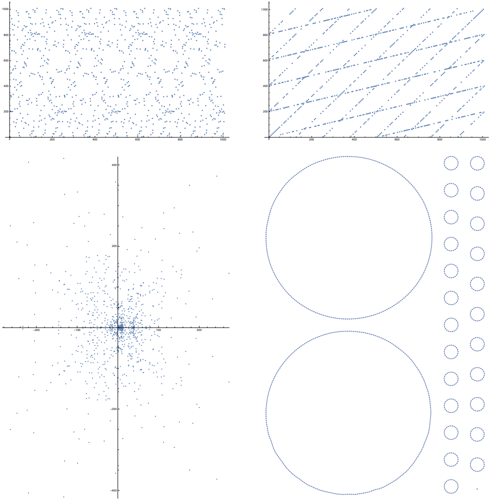

Figure: $f(x) = x + cx^{(p+1)/2}$, $p = 1009$, Itervalue: 706(top left), $c = 606$ satisfies $1 - c^2 = d^2$ where $d \in F_p$, Number of Components: 27

Graphs Trinomals

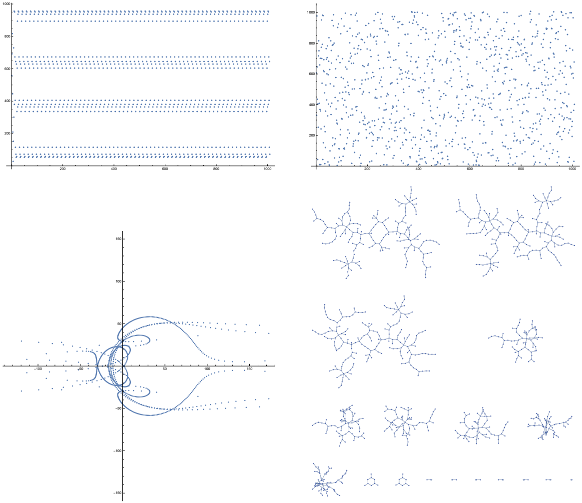

Figure: $f(x) = x^7 + 606x^{505}$, $p = 1009$, Itervalue: 756(top left), Number of Components: 936

Graphs Trinomials

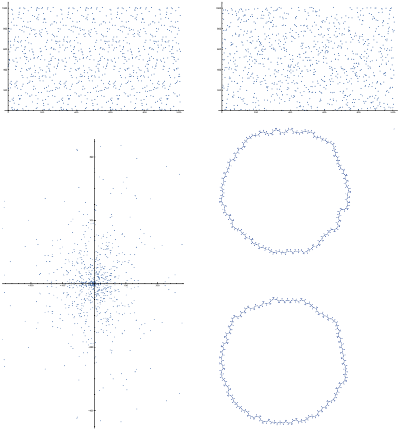

Figure: $f(x) = x^7 + 144x^{151}$, $p = 1009$, Itervalue: 82(top left), $\gcd(7, 1008) > 2$ and $\gcd(144, 1008) > 2$, Number of Components: 435

Cycle Close Up

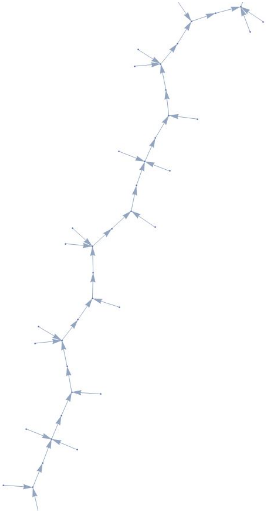

Figure: Closeup of Section of a Cycle in a Functional Graph

Exponential Decay

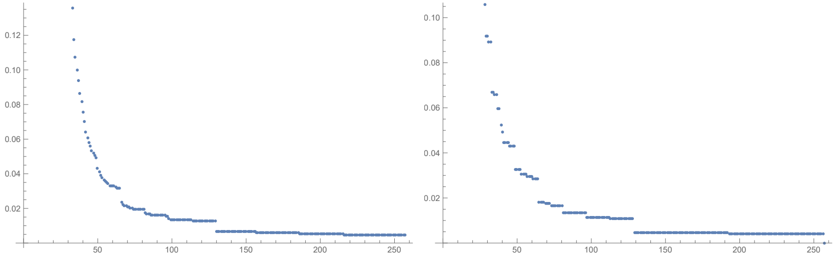

Figure: Fraction of c, d, x(Y Axis) for $f(x) = x + cx^d \mod p$ on F_p with $p = 257$ with Pre-Cycle plus Cycle Satisfying Certain Length (X Axis)(Left), and only Cycle Satisfying Certain Length(X Axis)(Right)

Exponential Decay

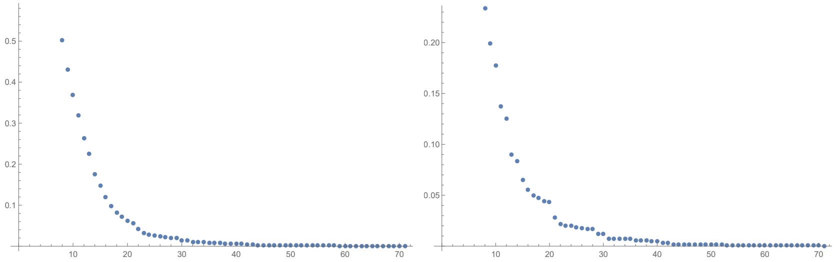

Figure: Fraction of a, c, b, x(Y Axis) for $f(x) = x^a + cx^b \mod p$ on F_p with $p = 71$ with Pre-Cycle plus Cycle Satisfying Certain Length (X Axis)(Left), and only Cycle Satisfying Certain Length(X Axis)(Right)

Side Results

[Theorem] If f is a friendly function for a $\Gamma\Upsilon$-PRG, f^{-1} cannot be a friendly function for a $\Gamma\Upsilon$-PRG.

[Conjecture] For a suitable friendly function f to form a PRG, it suffices to have a large complexity difference between f and f^{-1}, where f is on $O(\Upsilon)$ and f^{-1} is on $O(\Gamma)$.

Recap

- We want pseudo-random generator.

Recap

- We want pseudo-random generator.
- We generate it the way Blum and Micali do.

Recap

- We want pseudo-random generator.
- We generate it the way Blum and Micali do.
- We want to use Binomials (instead of DLP),

Recap

- We want pseudo-random generator.
- We generate it the way Blum and Micali do.
- We want to use Binomials (instead of DLP),

  under the assumption that solving Trinomials is hard.
- There are a couple interesting avenues we wish we had time to look into here

Recap

- We want pseudo-random generator.
- We generate it the way Blum and Micali do.
- We want to use Binomials (instead of DLP),

  under the assumption that solving Trinomials is hard.
- There are a couple interesting avenues we wish we had time to look into here
- This requires systematically finding a, b, c, D_p (restriction of $\mathbb{F}_p^*$ on which f_p is a permutation).
- However...

Recap

- We want pseudo-random generator.
- We generate it the way Blum and Micali do.
- We want to use Binomials (instead of DLP),

  under the assumption that solving Trinomials is hard.
- There are a couple interesting avenues we wish we had time to look into here
- This requires systematically finding a, b, c, D_p (restriction of $\mathbb{F}_p^*$ on which f_p is a permutation).
- However... such choices of a, b, c, D_p are exceedingly rare.

fin