

Basic principles of pseudo-random number generators

Jan Krhovják

Faculty of Informatics, Masaryk University

Outline

- True-randomness and pseudo-randomness
- Basic concepts of PRNGs
 - Linear feedback shift registers
 - Cryptographically secure PRNGs
 - PRNGs based on cryptographic functions
- Cryptanalytic attacks on PRNGs
- Yarrow family and others
 - Yarrow-160
 - Fortuna

True-randomness and pseudo-randomness

- True-randomness in general purpose computer systems
 - Hardware-based (almost any input)
 - Exact timing of keystrokes or exact movements of mouse
 - Microphone, video camera, or fluctuations in HDD access time
 - Built-in on-chip (Intel) or special add-on cards (Quantis)
 - Software-based
 - Process, network, or I/O completion statistics
 - Difficulty of collecting sufficient amount truly random data
=> the need of pseudo-random data
- Pseudo-randomness
 - PRNG is deterministic finite state machine =>
at any point of time it is in a certain internal state
 - PRNG state is secret (PRNG output must be unpredictable)
 - PRNG (whole) state is repeatedly updated (PRNG must produce different outputs)

True-randomness and pseudo-randomness

- True-randomness in general purpose computer systems
 - Hardware-based (almost any input)
 - Exact timing of keystrokes or exact movements of mouse
 - Microphone, video camera, or fluctuations in HDD access time
 - Built-in on-chip (Intel) or special add-on cards (Quantis)
 - Software-based
 - Process, network, or I/O completion statistics
 - Difficulty of collecting sufficient amount truly random data
=> the need of pseudo-random data
- Pseudo-randomness
 - PRNG is deterministic finite state machine =>
at any point of time it is in a certain internal state
 - PRNG state is secret (PRNG output must be unpredictable)
 - PRNG (whole) state is repeatedly updated (PRNG must produce different outputs)

Linear feedback shift register (LFSR)

- Initial value of LFSR is called seed
 - Input bit is a linear function of its previous state
 - Typically XOR of some bits from its state
 - Output is particular pseudorandom bit
- Finite number of possible states \Rightarrow repeating cycles
- Outputs of LFSRs are linear \Rightarrow easy cryptanalysis
- Improvement necessary for cryptographic purposes
 - Non-linear combination of several LFSRs
 - Using one (or several) LFSR to clock another (or combination of more) LFSR

Linear feedback shift register (LFSR)

- Initial value of LFSR is called seed
 - Input bit is a linear function of its previous state
 - Typically XOR of some bits from its state
 - Output is particular pseudorandom bit
- Finite number of possible states \Rightarrow repeating cycles
- Outputs of LFSRs are linear \Rightarrow easy cryptanalysis
- Improvement necessary for cryptographic purposes
 - Non-linear combination of several LFSRs
 - Using one (or several) LFSR to clock another (or combination of more) LFSR

Linear feedback shift register (LFSR)

- Initial value of LFSR is called seed
 - Input bit is a linear function of its previous state
 - Typically XOR of some bits from its state
 - Output is particular pseudorandom bit
- Finite number of possible states \Rightarrow repeating cycles
- Outputs of LFSRs are linear \Rightarrow easy cryptanalysis
- Improvement necessary for cryptographic purposes
 - Non-linear combination of several LFSRs
 - Using one (or several) LFSR to clock another (or combination of more) LFSR

Linear feedback shift register (LFSR)

- Initial value of LFSR is called seed
 - Input bit is a linear function of its previous state
 - Typically XOR of some bits from its state
 - Output is particular pseudorandom bit
- Finite number of possible states \Rightarrow repeating cycles
- Outputs of LFSRs are linear \Rightarrow easy cryptanalysis
- Improvement necessary for cryptographic purposes
 - Non-linear combination of several LFSRs
 - Using one (or several) LFSR to clock another (or combination of more) LFSR

LFSRs for cryptographic purposes

- Non-linear combination of several LFSRs \Rightarrow the need of well designed nonlinear function f
 - Geffe generator combines three LFSRs
 - Uses combining function $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$
 - Cryptographically weak – information about the states of LFSR 1 and LFSR 3 leaks into the output sequence
 - Summation generator
 - Integer addition (over \mathbb{Z}_2) is good nonlinear function
- Irregular clocking of LFSR
 - Alternating step generator
 - LFSR 1 is used to clock LFSR 2 and LFSR 3
 - Shrinking generator
 - LFSR 1 is used to control output of LFSR 2

LFSRs for cryptographic purposes

- Non-linear combination of several LFSRs \Rightarrow the need of well designed nonlinear function f
 - Geffe generator combines three LFSRs
 - Uses combining function $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$
 - Cryptographically weak – information about the states of LFSR 1 and LFSR 3 leaks into the output sequence
 - Summation generator
 - Integer addition (over \mathbb{Z}_2) is good nonlinear function
- Irregular clocking of LFSR
 - Alternating step generator
 - LFSR 1 is used to clock LFSR 2 and LFSR 3
 - Shrinking generator
 - LFSR 1 is used to control output of LFSR 2

Cryptographically secure RSA & BBS PRNG

- RSA PRNG is based on public-key cryptosystem RSA
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - Seed x_0 is selected from $[2, n - 2]$
 - For i from 1 to m do the following:
 - $x_i = x_{i-1}^e \pmod n$;
 - $z_i = \text{lsb}(x_i)$; i.e., z_i is least significant bit of x_i
 - The output sequence of length m is z_1, z_2, \dots, z_m
- Better efficiency
 - $e = 3$ imply faster exponentiation
 - Extracting the j (exactly given) least significant bits
 - Using Micali-Schnorr modification of this PRNG
- Blum Blum Shub PRNG is similar: the difference is in requirement that $\gcd(x_0, n) = 1$ and $x_i = x_{i-1}^2 \pmod n$

Cryptographically secure RSA & BBS PRNG

- RSA PRNG is based on public-key cryptosystem RSA
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - Seed x_0 is selected from $[2, n - 2]$
 - For i from 1 to m do the following:
 - $x_i = x_{i-1}^e \pmod n$;
 - $z_i = \text{lsb}(x_i)$; i.e., z_i is least significant bit of x_i
 - The output sequence of length m is z_1, z_2, \dots, z_m
- Better efficiency
 - $e = 3$ imply faster exponentiation
 - Extracting the j (exactly given) least significant bits
 - Using Micali-Schnorr modification of this PRNG
- Blum Blum Shub PRNG is similar: the difference is in requirement that $\gcd(x_0, n) = 1$ and $x_i = x_{i-1}^2 \pmod n$

Cryptographically secure RSA & BBS PRNG

- RSA PRNG is based on public-key cryptosystem RSA
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - Seed x_0 is selected from $[2, n - 2]$
 - For i from 1 to m do the following:
 - $x_i = x_{i-1}^e \pmod n$;
 - $z_i = \text{lsb}(x_i)$; i.e., z_i is least significant bit of x_i
 - The output sequence of length m is z_1, z_2, \dots, z_m
- Better efficiency
 - $e = 3$ imply faster exponentiation
 - Extracting the j (exactly given) least significant bits
 - Using Micali-Schnorr modification of this PRNG
- Blum Blum Shub PRNG is similar: the difference is in requirement that $\gcd(x_0, n) = 1$ and $x_i = x_{i-1}^2 \pmod n$

Cryptographically secure Micali-Schnorr PRNG

- Based on public-key RSA PRNG
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - $N = \lfloor \log_2 n \rfloor + 1$; $80e < N$; $k = \lfloor N(1 - \frac{1}{e}) \rfloor$; $r = N - k$
 - Bitlength of seed x_0 is r
 - For i from 1 to m do the following:
 - $y_i = x_{i-1}^e \pmod n$;
 - $x_i = \text{msb}_r(y_i)$; i.e., x_i is r most significant bits of y_i
 - $z_i = \text{lsb}_k(y_i)$; i.e., z_i is k least significant bits of y_i
 - The output sequence of length km is z_1, z_2, \dots, z_m
- The distribution $x^e \pmod n$ for random r -bit sequences x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in the interval $[0, n - 1]$
- Stronger than requiring that the RSA problem be intractable

Cryptographically secure Micali-Schnorr PRNG

- Based on public-key RSA PRNG
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - $N = \lfloor \log_2 n \rfloor + 1$; $80e < N$; $k = \lfloor N(1 - \frac{1}{e}) \rfloor$; $r = N - k$
 - Bitlength of seed x_0 is r
 - For i from 1 to m do the following:
 - $y_i = x_{i-1}^e \pmod n$;
 - $x_i = \text{msb}_r(y_i)$; i.e., x_i is r most significant bits of y_i
 - $z_i = \text{lsb}_k(y_i)$; i.e., z_i is k least significant bits of y_i
 - The output sequence of length km is z_1, z_2, \dots, z_m
- The distribution $x^e \pmod n$ for random r -bit sequences x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in the interval $[0, n - 1]$
- Stronger than requiring that the RSA problem be intractable

Cryptographically secure Micali-Schnorr PRNG

- Based on public-key RSA PRNG
 - p, q primes; $n = pq$; $\Phi(n) = (p - 1)(q - 1)$; $\gcd(e, \Phi(n)) = 1$
 - $N = \lfloor \log_2 n \rfloor + 1$; $80e < N$; $k = \lfloor N(1 - \frac{1}{e}) \rfloor$; $r = N - k$
 - Bitlength of seed x_0 is r
 - For i from 1 to m do the following:
 - $y_i = x_{i-1}^e \pmod n$;
 - $x_i = \text{msb}_r(y_i)$; i.e., x_i is r most significant bits of y_i
 - $z_i = \text{lsb}_k(y_i)$; i.e., z_i is k least significant bits of y_i
 - The output sequence of length km is z_1, z_2, \dots, z_m
- The distribution $x^e \pmod n$ for random r -bit sequences x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in the interval $[0, n - 1]$
- Stronger than requiring that the RSA problem be intractable

PRNG based on cryptographic functions 3DES/AES

- ANSI X9.17/X9.31 is based on 64-bit 3DES-3 or 128-bit AES
 - The key K is reserved only for the generator
 - The key (similarly as seed) can be re-generated
 - Seed is a 64/128-bit value V
 - DT is a 64/128-bit representation of the date and time
 - This DT is updated on each iteration
 - Some kind of randomly initialized counter can be also used
 - In each iteration is performed:
 - $I_i = E_K(DT)$
 - $R_i = E_K(I_i \oplus V_i)$
 - $V_{i+1} = E_K(R_i \oplus I_i)$
 - The output is pseudorandom string R_i
- One from many existing modifications
 - $I_i = E_K(I_{i-1} \oplus DT)$
 - This corresponds to encrypting DT in CBC mode (instead of in ECB)

PRNG based on cryptographic functions 3DES/AES

- ANSI X9.17/X9.31 is based on 64-bit 3DES-3 or 128-bit AES
 - The key K is reserved only for the generator
 - The key (similarly as seed) can be re-generated
 - Seed is a 64/128-bit value V
 - DT is a 64/128-bit representation of the date and time
 - This DT is updated on each iteration
 - Some kind of randomly initialized counter can be also used
 - In each iteration is performed:
 - $I_i = E_K(DT)$
 - $R_i = E_K(I_i \oplus V_i)$
 - $V_{i+1} = E_K(R_i \oplus I_i)$
 - The output is pseudorandom string R_i
- One from many existing modifications
 - $I_i = E_K(I_{i-1} \oplus DT)$
 - This corresponds to encrypting DT in CBC mode (instead of in ECB)

PRNG based on cryptographic functions DES/SHA-1

- Basic structure of DSS/FIPS-186 PRNG
 - Integer $160 \leq b \leq 512$ and predefined two 160-bit string(s) t
 - For G based on single DES must be b set to 160
 - One t for generating keys; second t for per-message secrets
 - Select b -bit seed s and optional user input y_i (default is 0)
 - For per-message secrets is no optional user input allowed
 - In iteration i is performed:
 - $z_i = (s_i + y_i) \bmod 2^b$
 - $a_i = G(t, z_i) \bmod q$
 - $s_{i+1} = (s_i + a_i + 1) \bmod 2^b$
 - The output is pseudorandom string a_i
- For general purpose PRNG can be $\bmod q$ omitted (necessary only for DSS where all arithmetic is done $\bmod q$)
- Notes about one-way function G based on single DES/SHA-1
 - SHA-1: Chaining variables H_i must be adapted to t
 - DES: New key is constructed for each call of G

PRNG based on cryptographic functions DES/SHA-1

- Basic structure of DSS/FIPS-186 PRNG
 - Integer $160 \leq b \leq 512$ and predefined two 160-bit string(s) t
 - For G based on single DES must be b set to 160
 - One t for generating keys; second t for per-message secrets
 - Select b -bit seed s and optional user input y_i (default is 0)
 - For per-message secrets is no optional user input allowed
 - In iteration i is performed:
 - $z_i = (s_i + y_i) \bmod 2^b$
 - $a_i = G(t, z_i) \bmod q$
 - $s_{i+1} = (s_i + a_i + 1) \bmod 2^b$
 - The output is pseudorandom string a_i
- For general purpose PRNG can be mod q omitted (necessary only for DSS where all arithmetic is done mod q)
- Notes about one-way function G based on single DES/SHA-1
 - SHA-1: Chaining variables H_i must be adapted to t
 - DES: New key is constructed for each call of G

PRNG based on cryptographic functions DES/SHA-1

- Basic structure of DSS/FIPS-186 PRNG
 - Integer $160 \leq b \leq 512$ and predefined two 160-bit string(s) t
 - For G based on single DES must be b set to 160
 - One t for generating keys; second t for per-message secrets
 - Select b -bit seed s and optional user input y_i (default is 0)
 - For per-message secrets is no optional user input allowed
 - In iteration i is performed:
 - $z_i = (s_i + y_i) \bmod 2^b$
 - $a_i = G(t, z_i) \bmod q$
 - $s_{i+1} = (s_i + a_i + 1) \bmod 2^b$
 - The output is pseudorandom string a_i
- For general purpose PRNG can be mod q omitted (necessary only for DSS where all arithmetic is done mod q)
- Notes about one-way function G based on single DES/SHA-1
 - SHA-1: Chaining variables H_i must be adapted to t
 - DES: New key is constructed for each call of G

Direct and input-based cryptanalytic attacks

- Direct cryptanalytic attacks
 - Attacker is able to distinguish between PRNG outputs and random outputs
 - Applicable only when the PRNG output can be observed
 - Cryptanalysis of underlying primitive is typically required
- Input-based attacks
 - Attacker is able to use the PRNG input(s) to distinguish between PRNG output and random values
 - Problem of user inputs in FIPS 186 PRNG
 - Attacker simply choose $y_i = y_{i-1} - a_{i-1} - 1 \pmod{2^b}$
 - $z_i = s_i + y_i = (s_{i-1} + a_{i-1} + 1) + (y_{i-1} - a_{i-1} - 1) = z_{i-1}$
 - Attack fails when user input is hashed before sending to PRNG

Direct and input-based cryptanalytic attacks

- Direct cryptanalytic attacks
 - Attacker is able to distinguish between PRNG outputs and random outputs
 - Applicable only when the PRNG output can be observed
 - Cryptanalysis of underlying primitive is typically required
- Input-based attacks
 - Attacker is able to use the PRNG input(s) to distinguish between PRNG output and random values
 - Problem of user inputs in FIPS 186 PRNG
 - Attacker simply choose $y_i = y_{i-1} - a_{i-1} - 1 \pmod{2^b}$
 - $z_i = s_i + y_i = (s_{i-1} + a_{i-1} + 1) + (y_{i-1} - a_{i-1} - 1) = z_{i-1}$
 - Attack fails when user input is hashed before sending to PRNG

State compromise cryptanalytic attacks

- State compromise extension attacks
 - Attacker misuse previously compromised state
 - The goal is recover unknown PRNG outputs (or distinguish those PRNG outputs from random values)
 - Almost all PRNGs described above will never recover from a state compromise (or they recover only after a very long time)
 - Consider e.g., ANSI PRNG with the compromised key
- Recovering is (typically) based on mixing small amounts of entropy to the secret state
 - Problem is limited amount of entropy between two requests for pseudorandom data
 - Attacker can make frequent requests and try all possibilities for the inputs to obtain secret state
 - Solution is pooling entropy to sufficient amount and then mix it to the secret state

State compromise cryptanalytic attacks

- State compromise extension attacks
 - Attacker misuse previously compromised state
 - The goal is recover unknown PRNG outputs (or distinguish those PRNG outputs from random values)
 - Almost all PRNGs described above will never recover from a state compromise (or they recover only after a very long time)
 - Consider e.g., ANSI PRNG with the compromised key
- Recovering is (typically) based on mixing small amounts of entropy to the secret state
 - Problem is limited amount of entropy between two requests for pseudorandom data
 - Attacker can make frequent requests and try all possibilities for the inputs to obtain secret state
 - Solution is pooling entropy to sufficient amount and then mix it to the secret state

Yarrow PRNG

- Generic Yarrow design
 - Entropy accumulator – collects entropy samples in two pools
 - Each (fast&slow) pool maintain on the fly the hashing context
 - Reseed mechanism – periodically reseeds the key from pools
 - Fast pool for frequent reseeds and slow pool for rare reseeds
 - Reseed control – determines when a reseed is to be performed
 - Based on entropy estimators and predefined thresholds
 - Generation mechanism – generates outputs based on key/seed
 - Based on block cipher in CRT mode
 - The seed is key K ; in iteration i is performed:
 - $C_i = (C_{i-1} + 1) \bmod 2^n$
 - $R_i = E_K(C_i)$, where R_i is the output
- Currently the most sophisticated PRNGs
 - Yarrow-160 and its successors Tiny and Fortuna

Yarrow PRNG

- Generic Yarrow design
 - Entropy accumulator – collects entropy samples in two pools
 - Each (fast&slow) pool maintain on the fly the hashing context
 - Reseed mechanism – periodically reseeds the key from pools
 - Fast pool for frequent reseeds and slow pool for rare reseeds
 - Reseed control – determines when a reseed is to be performed
 - Based on entropy estimators and predefined thresholds
 - Generation mechanism – generates outputs based on key/seed
 - Based on block cipher in CRT mode
 - The seed is key K ; in iteration i is performed:
 - $C_i = (C_{i-1} + 1) \bmod 2^n$
 - $R_i = E_K(C_i)$, where R_i is the output
- Currently the most sophisticated PRNGs
 - Yarrow-160 and its successors Tiny and Fortuna

Yarrow-160 & Fortuna PRNG

- Yarrow-160 uses 3DES-3 and SHA-1
 - SHA-1 pools imply maximum 160 bits
 - After every 10 outputs is the key changed (no new entropy)
 - In principle AES can be used together with SHA-256
- Fortuna uses AES/Serpent/Twofish and SHA-256
 - Deals with entropy in a much better way
 - There are 32 pools for collecting entropy (cyclically)
 - P_0 is used every reseed, P_1 is used every second reseed, P_2 is used every fourth reseed, etc.
 - There will be always pool with sufficient amount of entropy
 - Solves the problem of defining/constructing entropy estimators
 - Currently best designed PRNG

Yarrow-160 & Fortuna PRNG

- Yarrow-160 uses 3DES-3 and SHA-1
 - SHA-1 pools imply maximum 160 bits
 - After every 10 outputs is the key changed (no new entropy)
 - In principle AES can be used together with SHA-256
- Fortuna uses AES/Serpent/Twofish and SHA-256
 - Deals with entropy in a much better way
 - There are 32 pools for collecting entropy (cyclically)
 - P_0 is used every reseed, P_1 is used every second reseed, P_2 is used every fourth reseed, etc.
 - There will be always pool with sufficient amount of entropy
 - Solves the problem of defining/constructing entropy estimators
 - Currently best designed PRNG

Conclusion

- A lot of different PRNG designs exist, some of them are:
 - Extremely fast (e.g., LFSR based PRNG)
 - Extremely slow (e.g., cryptographically secure PRNG)
 - Intended for particular purpose (e.g., FIPS-186)
 - Designed for general purpose (e.g., Yarrow-160)
- A lot of different (and possibly unsecured) modifications of PRNGs are implemented in many applications
- When designing new PRNG
 - It is wise to expect that a secret state compromise may occur
 - Typical countermeasure is mix entropy to the secret state

Conclusion

- A lot of different PRNG designs exist, some of them are:
 - Extremely fast (e.g., LFSR based PRNG)
 - Extremely slow (e.g., cryptographically secure PRNG)
 - Intended for particular purpose (e.g., FIPS-186)
 - Designed for general purpose (e.g., Yarrow-160)
- A lot of different (and possibly unsecured) modifications of PRNGs are implemented in many applications
- When designing new PRNG
 - It is wise to expect that a secret state compromise may occur
 - Typical countermeasure is mix entropy to the secret state

Conclusion

- A lot of different PRNG designs exist, some of them are:
 - Extremely fast (e.g., LFSR based PRNG)
 - Extremely slow (e.g., cryptographically secure PRNG)
 - Intended for particular purpose (e.g., FIPS-186)
 - Designed for general purpose (e.g., Yarrow-160)
- A lot of different (and possibly unsecured) modifications of PRNGs are implemented in many applications
- When designing new PRNG
 - It is wise to expect that a secret state compromise may occur
 - Typical countermeasure is mix entropy to the secret state