Analysis, demands, and properties of pseudorandom number generators

Jan Krhovják

Department of Computer Systems and Communications
Faculty of Informatics, Masaryk University
Brno, Czech Republic

Outline

- Random and pseudorandom data in cryptography
- Review of demands of common cryptographic schemes on pseudorandom data
  - Cryptographic keys and initialization vectors
  - Padding schemes and salting
  - Cryptographic protocols
- The analysis of properties used in PRNGs
  - Generating pseudorandom data in computer systems
  - Basic categories and principles of PRNGs
- Conclusion & future research

Random and pseudorandom data in cryptography

- Random data in cryptography
  - Cryptographic keys, padding values, nonces, etc.
  - Quality and unpredictability is critical
- Generating truly random data
  - Based on nondeterministic physical phenomena
    - Radioactive decay, thermal noise, etc.
  - In deterministic environments extremely hard and slow
    - Only a small amount of random data in a reasonable time
- Generating pseudorandom data
  - Typically (in many computational environments) faster
  - Generated by deterministic algorithm
    - Short input (often called seed) – truly random data
    - Output – computationally indistinguishable from truly random data

Random and pseudorandom data in cryptography

- Random data in cryptography
  - Cryptographic keys, padding values, nonces, etc.
  - Quality and unpredictability is critical
- Generating truly random data
  - Based on nondeterministic physical phenomena
    - ★ Radioactive decay, thermal noise, etc.
  - In deterministic environments extremely hard and slow
    - ★ Only a small amount of random data in a reasonable time
- Generating pseudorandom data
  - Typically (in many computational environments) faster
  - Generated by deterministic algorithm
    - ★ Short input (often called seed) – truly random data
    - ★ Output – computationally indistinguishable from truly random data

Random and pseudorandom data in cryptography

- Random data in cryptography
  - Cryptographic keys, padding values, nonces, etc.
  - Quality and unpredictability is critical
- Generating truly random data
  - Based on nondeterministic physical phenomena
    - ★ Radioactive decay, thermal noise, etc.
  - In deterministic environments extremely hard and slow
    - ★ Only a small amount of random data in a reasonable time
- Generating pseudorandom data
  - Typically (in many computational environments) faster
  - Generated by deterministic algorithm
    - ★ Short input (often called seed) – truly random data
    - ★ Output – computationally indistinguishable from truly random data

Random and pseudorandom data in cryptography

- Random data in cryptography
  - Cryptographic keys, padding values, nonces, etc.
  - Quality and unpredictability is critical
- Generating truly random data
  - Based on nondeterministic physical phenomena
    - Radioactive decay, thermal noise, etc.
  - In deterministic environments extremely hard and slow
    - Only a small amount of random data in a reasonable time
- Generating pseudorandom data
  - Typically (in many computational environments) faster
  - Generated by deterministic algorithm
    - Short input (often called seed) – truly random data
    - Output – computationally indistinguishable from truly random data

Cryptographic keys and initialization vectors I

- Symmetric cryptosystems (block & stream ciphers)
  - Supported length of keys and initialization vectors is hardwired & their potential modification imply:
    - ★ Change of usage model (e.g., from DES to 3DES-2/3)
    - ★ Change of cipher itself (e.g., from CAST-128 to CAST-256)
- Block ciphers (requirements)
  - Keys: mostly between 112 and 256 bits (e.g., 3DES-2, AES, Serpent)
    - ★ <80 bits (DES); 256–448 (Blowfish, MARS); 448< (RC5, RC6)
  - Initialization vectors: same as blocksize (i.e., 64, 128, or 256 bits)
- Stream ciphers (very similar requirements)
  - Keys: typically do not go beyond 256 bits (e.g., HC-256, Dragon-256)
  - Initialization vectors: mostly comparable to the length of the used key
- Initialization vectors require only freshness (not secrecy)

Cryptographic keys and initialization vectors I

- Symmetric cryptosystems (block & stream ciphers)
  - Supported length of keys and initialization vectors is hardwired & their potential modification imply:
    - Change of usage model (e.g., from DES to 3DES-2/3)
    - Change of cipher itself (e.g., from CAST-128 to CAST-256)
- Block ciphers (requirements)
  - Keys: mostly between 112 and 256 bits (e.g., 3DES-2, AES, Serpent)
    - $<$80 bits (DES); 256–448 (Blowfish, MARS); 448$<$ (RC5, RC6)
  - Initialization vectors: same as blocksize (i.e., 64, 128, or 256 bits)
- Stream ciphers (very similar requirements)
  - Keys: typically do not go beyond 256 bits (e.g., HC-256, Dragon-256)
  - Initialization vectors: mostly comparable to the length of the used key
- Initialization vectors require only freshness (not secrecy)

Cryptographic keys and initialization vectors I

- Symmetric cryptosystems (block & stream ciphers)
  - Supported length of keys and initialization vectors is hardwired & their potential modification imply:
    - Change of usage model (e.g., from DES to 3DES-2/3)
    - Change of cipher itself (e.g., from CAST-128 to CAST-256)
- Block ciphers (requirements)
  - Keys: mostly between 112 and 256 bits (e.g., 3DES-2, AES, Serpent)
    - $<$80 bits (DES); 256–448 (Blowfish, MARS); 448$<$ (RC5, RC6)
  - Initialization vectors: same as blocksize (i.e., 64, 128, or 256 bits)
- Stream ciphers (very similar requirements)
  - Keys: typically do not go beyond 256 bits (e.g., HC-256, Dragon-256)
  - Initialization vectors: mostly comparable to the length of the used key
- Initialization vectors require only freshness (not secrecy)

Cryptographic keys and initialization vectors I

- Symmetric cryptosystems (block & stream ciphers)
  - Supported length of keys and initialization vectors is hardwired & their potential modification imply:
    - ★ Change of usage model (e.g., from DES to 3DES-2/3)
    - ★ Change of cipher itself (e.g., from CAST-128 to CAST-256)
- Block ciphers (requirements)
  - Keys: mostly between 112 and 256 bits (e.g., 3DES-2, AES, Serpent)
    - ★ $<$80 bits (DES); 256–448 (Blowfish, MARS); 448$<$ (RC5, RC6)
  - Initialization vectors: same as blocksize (i.e., 64, 128, or 256 bits)
- Stream ciphers (very similar requirements)
  - Keys: typically do not go beyond 256 bits (e.g., HC-256, Dragon-256)
  - Initialization vectors: mostly comparable to the length of the used key
- Initialization vectors require only freshness (not secrecy)

Cryptographic keys and initialization vectors II

- Asymmetric cryptosystems (in comparison with symmetric)
  - Depend on the intractability of certain mathematical problems
    - Their solution is not so time consuming as an exhaustive search of the key space => the need of several times larger keys
    - Typically between 1024 and 8192 bits (or 160 and 512 bits for ECC)
  - Easily parameterizable
    - Key-length is restricted only by implementation
- Common asymmetric cryptosystems
  - RSA: size of key is bit-length of its modulus $n = pq$
  - DSA: size of key is bit-length of its prime modulus p
  - ECDSA: size of key is bit-length of order n of the base point G (of the chosen elliptic curve E)

Cryptographic keys and initialization vectors II

- Asymmetric cryptosystems (in comparison with symmetric)
  - Depend on the intractability of certain mathematical problems
    - Their solution is not so time consuming as an exhaustive search of the key space => the need of several times larger keys
    - Typically between 1024 and 8192 bits (or 160 and 512 bits for ECC)
  - Easily parameterizable
    - Key-length is restricted only by implementation
- Common asymmetric cryptosystems
  - RSA: size of key is bit-length of its modulus $n = pq$
  - DSA: size of key is bit-length of its prime modulus p
  - ECDSA: size of key is bit-length of order n of the base point G (of the chosen elliptic curve E)

Padding schemes and salting

- Padding used to extend messages to required length of block (or integer multiple of block)
  - Padding typically not required by stream ciphers
  - Deterministic padding required by block ciphers (in the CBC mode) and cryptographic hash functions
  - Randomized padding required by deterministic asymmetric cryptosystems (e.g., RSA)
- Padding schemes adapted for algorithm RSA (see PKCS #1)
  - Encryption schemes: RSAES-OAEP and RSAES-PKCS1-v1_5
    - *hLen* bytes and $k - mLen - 3$ bytes of random data
  - Signature scheme: RSASSA-PSS (and RSASSA-PKCS1-v1_5)
    - *hLen* bytes (and 0 bytes) of random data
- Salting – used commonly in the password-based cryptography
  - Key derivation functions as PBKDF1/PBKDF2 (see PKCS #5)
    - PBKDF1 requires 64 bits of salt; PBDF2 requires 8 bits of salt
  - UNIX function `crypt` also uses up to 128 bits of salt

Padding schemes and salting

- Padding used to extend messages to required length of block (or integer multiple of block)
  - Padding typically not required by stream ciphers
  - Deterministic padding required by block ciphers (in the CBC mode) and cryptographic hash functions
  - Randomized padding required by deterministic asymmetric cryptosystems (e.g., RSA)
- Padding schemes adapted for algorithm RSA (see PKCS #1)
  - Encryption schemes: RSAES-OAEP and RSAES-PKCS1-v1_5
    - ★ $hLen$ bytes and $k - mLen - 3$ bytes of random data
  - Signature scheme: RSASSA-PSS (and RSASSA-PKCS1-v1_5)
    - ★ $hLen$ bytes (and 0 bytes) of random data
- Salting – used commonly in the password-based cryptography
  - Key derivation functions as PBKDF1/PBKDF2 (see PKCS #5)
    - ★ PBKDF1 requires 64 bits of salt; PBDF2 requires 8 bits of salt
  - UNIX function `crypt` also uses up to 128 bits of salt

Padding schemes and salting

- Padding used to extend messages to required length of block (or integer multiple of block)
  - Padding typically not required by stream ciphers
  - Deterministic padding required by block ciphers (in the CBC mode) and cryptographic hash functions
  - Randomized padding required by deterministic asymmetric cryptosystems (e.g., RSA)
- Padding schemes adapted for algorithm RSA (see PKCS #1)
  - Encryption schemes: RSAES-OAEP and RSAES-PKCS1-v1_5
    - *hLen* bytes and $k - mLen - 3$ bytes of random data
  - Signature scheme: RSASSA-PSS (and RSASSA-PKCS1-v1_5)
    - *hLen* bytes (and 0 bytes) of random data
- Salting – used commonly in the password-based cryptography
  - Key derivation functions as PBKDF1/PBKDF2 (see PKCS #5)
    - PBKDF1 requires 64 bits of salt; PBDF2 requires 8 bits of salt
  - UNIX function `crypt` also uses up to 128 bits of salt

Cryptographic protocols

- Focused on challenge-response protocols
  - Authentication protocols: random data for random challenges (nonces)
  - Key establishment protocols: random data for generating shared keys
  - Authenticated key establishment protocols: their combination
- Common authentication protocols (e.g., ISO/IEC 9798)
  - Random challenges are typically 64 (or better 128) bits long
  - Some protocols use timestamps or sequence numbers instead of nonces
- Key establishment (key distribution & key agreement) protocols
  - One party is involved in key generation process
    - ⋆ Requirements are dependent on the type of generated key
  - Both (or more) parties is involved in key generation process
    - ⋆ Typically based on Diffie-Hellman exponential key exchange
    - ⋆ Modulus at least 1024 bits; Key(s) at least 160 bits

Cryptographic protocols

- Focused on challenge-response protocols
  - Authentication protocols: random data for random challenges (nonces)
  - Key establishment protocols: random data for generating shared keys
  - Authenticated key establishment protocols: their combination
- Common authentication protocols (e.g., ISO/IEC 9798)
  - Random challenges are typically 64 (or better 128) bits long
  - Some protocols use timestamps or sequence numbers instead of nonces
- Key establishment (key distribution & key agreement) protocols
  - One party is involved in key generation process
    - ⋆ Requirements are dependent on the type of generated key
  - Both (or more) parties is involved in key generation process
    - ⋆ Typically based on Diffie-Hellman exponential key exchange
    - ⋆ Modulus at least 1024 bits; Key(s) at least 160 bits

Cryptographic protocols

- Focused on challenge-response protocols
  - Authentication protocols: random data for random challenges (nonces)
  - Key establishment protocols: random data for generating shared keys
  - Authenticated key establishment protocols: their combination
- Common authentication protocols (e.g., ISO/IEC 9798)
  - Random challenges are typically 64 (or better 128) bits long
  - Some protocols use timestamps or sequence numbers instead of nonces
- Key establishment (key distribution & key agreement) protocols
  - One party is involved in key generation process
    - Requirements are dependent on the type of generated key
  - Both (or more) parties is involved in key generation process
    - Typically based on Diffie-Hellman exponential key exchange
    - Modulus at least 1024 bits; Key(s) at least 160 bits

Generating pseudorandom data in computer systems

- 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)
- Secret state compromise may occur – recovering is difficult
  - Mixing data with small amounts of entropy to the secret state
  - Problem is limited amount of entropy between two requests for pseudorandom data (solution is pooling)
    - ⋆ Frequent requests & brute force => new secret state
    - ⋆ Solution is pooling of incoming entropy to sufficient amount, and then to mix it to the secret state
- Basic types of PRNGs utilize
  - Linear feedback shift register (LFSR), hard problems of number and complexity theory, typical cryptographic functions/primitives

Generating pseudorandom data in computer systems

- 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)
- Secret state compromise may occur – recovering is difficult
  - Mixing data with small amounts of entropy to the secret state
  - Problem is limited amount of entropy between two requests for pseudorandom data (solution is pooling)
    - Frequent requests & brute force => new secret state
    - Solution is pooling of incoming entropy to sufficient amount, and then to mix it to the secret state
- Basic types of PRNGs utilize
  - Linear feedback shift register (LFSR), hard problems of number and complexity theory, typical cryptographic functions/primitives

Generating pseudorandom data in computer systems

- 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)
- Secret state compromise may occur – recovering is difficult
  - Mixing data with small amounts of entropy to the secret state
  - Problem is limited amount of entropy between two requests for pseudorandom data (solution is pooling)
    - Frequent requests & brute force => new secret state
    - Solution is pooling of incoming entropy to sufficient amount, and then to mix it to the secret state
- Basic types of PRNGs utilize
  - Linear feedback shift register (LFSR), hard problems of number and complexity theory, typical cryptographic functions/primitives

Linear feedback shift register (LFSR)

- Finite number of possible states => repeating cycles
- Outputs of LFSRs are linear => easy cryptanalysis
- Improvement necessary for cryptographic purposes
  - Non-linear combination of several LFSRs imply the need of well designed nonlinear function f
    - Geffe generator: function $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$
    - Summation generator: integer addition (over $\mathbb{Z}_2$)
  - Using one (or several) LFSR to clock another (or combination of more) LFSR
    - Alternating step generator: LFSR 1 used to clock LFSR 2 and LFSR 3
    - Shrinking generator: LFSR 1 used to control output of LFSR 2

Linear feedback shift register (LFSR)

- Finite number of possible states => repeating cycles
- Outputs of LFSRs are linear => easy cryptanalysis
- Improvement necessary for cryptographic purposes
  - Non-linear combination of several LFSRs imply the need of well designed nonlinear function f
    - Geffe generator: function $f(x_1, x_2, x_3) = x_1 x_2 \oplus x_2 x_3 \oplus x_3$
    - Summation generator: integer addition (over $\mathbb{Z}_2$)
  - Using one (or several) LFSR to clock another (or combination of more) LFSR
    - Alternating step generator: LFSR 1 used to clock LFSR 2 and LFSR 3
    - Shrinking generator: LFSR 1 used to control output of LFSR 2

Hard problems of number and complexity theory

- 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:
    - ★ $x_i = x_{i-1}^e \mod n$;
    - ★ $z_i = lsb(x_i)$; i.e., z_i is least significant bit of x_i
  - The output sequence of length m is $z_1, z_2, \ldots z_m$
  - Security based on the intractability of RSA problem
- Blum Blum Shub PRNG on modular squaring
  - Difference is that is used $x_i = x_{i-1}^2 \mod n$
  - Security based on the intractability of quadratic residuosity problem
- Generators based on discrete logarithm problem or Diffie-Hellman problem (with stronger DDH assumption)

Hard problems of number and complexity theory

- 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:
    - ★ $x_i = x_{i-1}^e \mod n$;
    - ★ $z_i = lsb(x_i)$; i.e., z_i is least significant bit of x_i
  - The output sequence of length m is $z_1, z_2, \ldots z_m$
  - Security based on the intractability of RSA problem
- Blum Blum Shub PRNG on modular squaring
  - Difference is that is used $x_i = x_{i-1}^2 \mod n$
  - Security based on the intractability of quadratic residuosity problem
- Generators based on discrete logarithm problem or Diffie-Hellman problem (with stronger DDH assumption)

Hard problems of number and complexity theory

- 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:
    - $x_i = x_{i-1}^e \mod n$;
    - $z_i = lsb(x_i)$; i.e., z_i is least significant bit of x_i
  - The output sequence of length m is $z_1, z_2, \ldots z_m$
  - Security based on the intractability of RSA problem
- Blum Blum Shub PRNG on modular squaring
  - Difference is that is used $x_i = x_{i-1}^2 \mod n$
  - Security based on the intractability of quadratic residuosity problem
- Generators based on discrete logarithm problem or Diffie-Hellman problem (with stronger DDH assumption)

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
  - Seed is a 64/128-bit value V
  - DT is a 64/128-bit representation of the date and time
  - 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
  - Seed is a 64/128-bit value V
  - DT is a 64/128-bit representation of the date and time
  - 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)

Conclusion

- We described demands of common cryptographic schemes (between hundreds and thousands of bits)
  - Smaller amounts: symmetric cryptography
  - Larger amounts: asymmetric cryptography
- Analysis of properties of common pseudorandom number generators, some of them are:
  - Extremely fast (e.g., LFSR based PRNG)
  - Extremely slow (e.g., cryptographically secure PRNG)
  - Intended for particular purpose (e.g., ANSI X9.17/X9.31, FIPS-186)
  - Designed for general purpose (e.g., Yarrow-160, Fortuna)
- Future research
  - PRNGs in mobile computing environments (limited resources as CPU speed, memory, or energy)

Conclusion

- We described demands of common cryptographic schemes (between hundreds and thousands of bits)
  - Smaller amounts: symmetric cryptography
  - Larger amounts: asymmetric cryptography
- Analysis of properties of common pseudorandom number generators, some of them are:
  - Extremely fast (e.g., LFSR based PRNG)
  - Extremely slow (e.g., cryptographically secure PRNG)
  - Intended for particular purpose (e.g., ANSI X9.17/X9.31, FIPS-186)
  - Designed for general purpose (e.g., Yarrow-160, Fortuna)
- Future research
  - PRNGs in mobile computing environments (limited resources as CPU speed, memory, or energy)

Conclusion

- We described demands of common cryptographic schemes (between hundreds and thousands of bits)
  - Smaller amounts: symmetric cryptography
  - Larger amounts: asymmetric cryptography
- Analysis of properties of common pseudorandom number generators, some of them are:
  - Extremely fast (e.g., LFSR based PRNG)
  - Extremely slow (e.g., cryptographically secure PRNG)
  - Intended for particular purpose (e.g., ANSI X9.17/X9.31, FIPS-186)
  - Designed for general purpose (e.g., Yarrow-160, Fortuna)
- Future research
  - PRNGs in mobile computing environments (limited resources as CPU speed, memory, or energy)