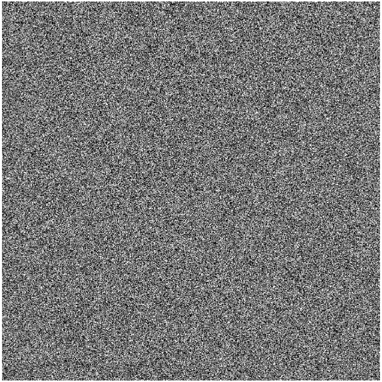
Number-Theoretic DRNGs



Number-Theoretic DRNGs

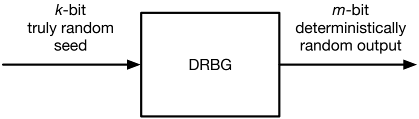
- Number-theoretic DRNGs are also called as Deterministically Random Bit Generators: DRBGs
- Their security is based on difficulty assumptions of certain number theoretic problems, such as factoring and discrete logarithm
- Examples: RSA, Blum-Blum-Shub, Naor-Reingold bit generators
- Elliptic curve DRNGs arealso in this category however they are based on the difficulty of the ECDLP

Number-Theoretic DRNGs

- Certain security properties, such as next-bit security, are proved on the basis these intractability assumptions
- Usually only asymptotic security properties can be proved, for example, with increasing RSA modulus
- However, these algorithms are not very practical due to their low output rate
- They may still be useful for PKC platforms which are already equipped with efficient hardware

Cryptographically Secure DRBGs

- The number-theoretic DRNGs are also called deterministic random bit generators (DRBGs)
- The general model:



- The seed length k should be sufficiently large so that exhaustive searching over 2^k seeds is practically infeasible
- An adversary must not efficiently distinguish between the output sequences of the DRBG and truly random bit sequences

Cryptographically Secure DRBGs

- We say that the DRBG passes all **polynomial-time statistical tests** if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than $\frac{1}{2}$
- We say that the DRBG passes the **next-bit test** if there is no polynomial-time algorithm which, on input of the first s-bits of an output sequence R can predict the $(s+1)$st bit of R with probability significantly greater than $\frac{1}{2}$
- A DRBG passes the next-bit test if and only if it passes all polynomial-time statistical tests.
- A DRBG that passes these tests are **cryptographically secure**
- These definitions are meaningful for asymptotically large inputs only

RSA DRBG

- The output of the RSA DRBG is the sequence of the deterministically random bits $R_1, R_2, \ldots, R_m$ for a given m
- **Setup:** Generate two secret primes p and q
  Compute $n = p \cdot q$ and $\phi = (p-1)\cdot(q-1)$
  Select a random integer $e \in [2, \phi-1]$ with $\gcd(e, \phi) = 1$
- **Seed:** Select a random integer $r_0 \in [1, n-1]$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $r_i \leftarrow r_{i-1}^e \pmod{n}$
  - $R_i \leftarrow$ the LSB of r_i
- **Output:** The sequence of bits $R_1, R_2, \ldots, R_m$
- The RSA DRBG produces m bits in m steps

RSA DRBG

- The RSA DRBG is secure under the assumption that the factoring of the modulus is difficult for the given size
- The generation of a single bit R_i requires a modular exponentiation operation with k-bit modulus, where $k = \log_2(n)$
- Typically, a 1024-bit modular exponentiation takes 1ms, thus, the RSA DRBG will work at the speed 1 Kbit/sec
- Sometimes $e = 3$ is chosen so that a single exponentiation takes 1 modular multiplication and 1 modular squaring
- This will bring the speed near 1 Mbit/sec

Modified Versions of RSA DRBG

- The efficiency can further be improved by extracting L bits per exponentiation where $L = c \cdot \log\log(n)$ for a constant c
- Provided that n is sufficiently large, this modified generator is also cryptographically secure
- However, an explicit range of values of c for which the generator remains secure is not known

Micali-Schnorr DRBG

- The Micali-Schnorr also improves the efficiency of RSA DRBG
- **Setup:** Generate two secret primes p and q
  Compute $n = p \cdot q$ and $\phi = (p-1) \cdot (q-1)$ with $k = \log_2(n)$
  Select e with $80e \leq k$ and $\gcd(e, \phi) = 1$
  The word size $s = k(1 - 2/e)$
- **Seed:** Select a random integer $r_0 \in [1, n-1]$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $u_i \leftarrow r_{i-1}^e \pmod{n}$
  - $r_i \leftarrow$ the $k - s$ most significant bits of u_i
  - $R_i \leftarrow$ the s least significant bits of u_i
- **Output:** The sequence of s-bit numbers $R_1, R_2, \ldots, R_m$

- The Micali-Schnorr DRBG produces sm bits in m steps

Micali-Schnorr DRBG

- The Micali-Schnorr DRBG is more efficient than RSA DRB
- At each step $s = k(1 - 2/e)$ bits are generated
- For example, when $e = 3$ and $k = 1024$ (satisfying $80e \leq k$), then the Micali-Schnorr DRBG generates s bits in every exponentiation step such that

$$s = k(1 - 2/e) = 1024(1 - 2/3) = 341$$

- Therefore, it generates $341m$ bits in m steps
- Moreover, by selecting $e = 3$, the computation of $u_i = r_{i-1}^3 \pmod{n}$ requires one modular squaring with a $(k - s)$-bit number, and one modular multiplication with two k-bit numbers

Micali-Schnorr DRBG

- The Micali-Schnorr DRBG is secure under the assumption that the distribution of r^e mod n for random k-bit sequences r are indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in the interval $[0, n-1]$
- This assumption is stronger the requiring that RSA problem be intractable
- Micali and Schnorr also describe a method that transforms any cryptographically secure DRBG into into one that can be accelerated by parallel evaluation
- The method of parallelization is perfect: P parallel processors speed the generation of deterministically random bits by a factor of P

Blum-Blum-Shub DRBG

- The Blum-Blum-Shub DRBG is also known as the BBS generator
- It s secure if the integer factorization is intractable
- **Setup:** Generate two secret primes p and q, with the property that they are equal to 3 mod 4, and compute $n = p \cdot q$
- **Seed:** Select a random $r_0 \in [1, n-1]$ such that $\gcd(r_0, n) = 1$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $r_i \leftarrow r_{i-1}^2 \pmod{n}$
  - $R_i \leftarrow$ the LSB of r_i
- **Output:** The sequence of bits $R_1, R_2, \ldots, R_m$

- The BBS DRBG produces m bits in m steps

Blum-Blum-Shub DRBG

- Generating each deterministically random bit R_i requires one modular squaring with k-bit numbers, with $k = \log_2(n)$
- The efficiency can further be improved by extracting L bits per exponentiation where $L = c \cdot \log\log(n)$ for a constant c
- Provided that n is sufficiently large, this modified generator is also cryptographically secure
- However, an explicit range of values of c for which the generator remains secure is not known

Modified Rabin DRBG

- The modified Rabin DRBG differs slightly from the BBS DRBG
- **Setup:** Generate two secret primes p and q, with the property that they are equal to 3 mod 4, and compute $n = p \cdot q$
- **Seed:** Select a random $r_0 \in [1, n-1]$ such that $\gcd(r_0, n) = 1$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $r_i' \leftarrow r_{i-1}^2 \pmod{n}$
  - If $r_i' < n/2$, then $r_i = r_i'$; otherwise, $r_i = n - r_i'$
  - $R_i \leftarrow$ the LSB of r_i
- **Output:** The sequence of bits $R_1, R_2, \ldots, R_m$

- The modified Rabin DRBG produces m bits in m steps

Power Generator

- The security of the Power Generator is based on the DLP in $\mathcal{Z}_p$
- **Setup:** Generate the prime p and the primitive element g mod p, such that $k = \log_2(p)$
- **Seed:** Select a random $r_0 \in [1, p-1]$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $r_i \leftarrow g^{r_{i-1}} \pmod{p}$
  - $R_i \leftarrow$ the LSB of r_i
- **Output:** The sequence of bits $R_1, R_2, \ldots, R_m$

- The Power Generator produces m bits in m steps

Modified Power Generator

- For efficiency purposes, the exponent r is sometimes restricted to 128 or 160 bits, since the exponentiation requires few multiplications
- However, we need to make sure that the difficulty of the DLP is not jeopardized when short exponents are used
- Patel and Sundaram showed that when $p = 2q + 1$ and q is a prime, any information about the $k - \Omega(\log k)$ bits can be used to compute the discrete logarithm of $g^r \pmod{p}$, where r has $\Omega(\log k)$ bits
- This gives a secure and efficient algorithm which generates $k - s - 1$ bits per iteration, where $s = \Omega(\log k)$
- For example, when $k = 1024$ and $s = 128$, the modified Power Generator produces $1024 - 128 - 1 = 895$ bits per iteration

Modified Power Generator

- The modified Power Generator is due to Patel and Sundaram
- **Setup:** Generate the prime p and the primitive element $g \bmod p$, furthermore, $p = 2q + 1$ such that q is also a prime
  For $k = \log_2(p)$, we also have $s = \Omega(\log k)$
- **Seed:** Select a random $r_0 \in [1, p - 1]$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $r_i \leftarrow g^{r_{i-1}} \pmod{p}$
  - $R_i \leftarrow$ the least significant $k - s$ bits of r_i, except the LSB
- **Output:** The sequence of $(k - s - 1)$ bits $R_1, R_2, \ldots, R_m$
- The modified Power Generator produces $m(k - s - 1)$ bits in m steps

Naor-Reingold DRBG

- Similar to the BBS DRBG, the security of the Naor-Reingold DRBG depends on the difficulty of factoring integers
- Given two k-bit primes p and q, the $2k$-bit modulus is $n = p \cdot q$
- Also take g which is a square mod n, that is $g = x^2 \pmod{n}$ for some $x \in [1, n-1]$
- The Naor-Reingold DRBG needs three constructions:
  - Binary vector representations $\text{bin}_k(u)$ and $\text{bin}_{2k}(u)$
  - The mod 2 inner-product $\odot$
  - Integer-valued vector function $f(A, b)$

Binary Vector Representation

- $\text{bin}_k(u)$ represents the k-dimension binary vector representation of the integer u, for example, if $k = 3$ and $u = 5 = (101)$, then $\text{bin}_3(5) = (1, 0, 1)$
- $\text{bin}_{2k}(u)$ represents the $2k$-dimension binary vector representation of the integer u, for example, if $k = 3$ and $u = 13 = (1101)$, then $\text{bin}_6(13) = (0, 0, 1, 1, 0, 1)$

The mod 2 Inner-Product $\odot$

- The mod 2 inner-product of two binary vectors $v = (v_1, v_2, \ldots, v_k)$ and $w = (w_1, w_2, \ldots, w_k)$ is defined as

$$v \odot w = \sum_{i=1}^{k} v_i \cdot w_i \pmod 2$$

- For example, assume $v = (1, 1, 1)$ and $w = (0, 1, 1)$
- We compute $v \odot w$ as

$$\begin{aligned}(1,1,1) \odot (0,1,1) &= 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 \pmod 2 \\ &= 2 \pmod 2 \\ &= 0 \pmod 2\end{aligned}$$

Integer-Valued Vector Function $f(A, b)$

- Assume, an integer vector A of dimension $2k$ is given

$$A = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{k,0}, a_{k,1})$$

- Also, assume a binary vector $b = (b_1, b_2, \ldots, b_k)$ is given
- We define the integer valued function $f(A, b)$ as

$$f(A, b) = \sum_{i=1}^{k} a_{i,b_i}$$

- For example, $A = (19, 5, 23, 16, 11, 20)$ and $b = (1, 1, 0)$ imply

$$f(A, b) = a_{1,1} + a_{2,1} + a_{3,0} = 5 + 16 + 11 = 32$$

Naor-Reingold DRBG

- **Setup:** Generate two k-bit secret primes p and q, and the $2k$-bit modulus $n = p \cdot q$, and select a random g which is a square mod n Select a random vector of integers $A = (a_{1,0}, a_{1,1}, \ldots, a_{k,0}, a_{k,1})$
- **Seed:** Select a random binary vector $r = (r_1, r_2, \ldots, r_{2k})$
- **Random Bit Generation:** For $i = 1, 2, \ldots, m$
  - $b \leftarrow \text{bin}_k(i)$
  - $u \leftarrow f(A, b)$
  - $v \leftarrow g^u \pmod{n}$
  - $R_i \leftarrow r \odot \text{bin}_{2k}(v)$
- **Output:** The sequence of bits $R_1, R_2, \ldots, R_m$

- The Naor-Reingold (NR) DRBG produces m bits in m steps
- If factoring the modulus n is infeasible, the output of the NR DRBG is indistinguishable from a random sequence of bits