<u>One Time Pad</u>: Alice and Bob agree on a secret bit string pad $= b_1 b_2 \cdots b_n$, where $b_i \in_R \{0,1\}$ (pad is chosen in $\{0,1\}^n$ with uniform probability). This is the common secret key. To encrypt a message $m = m_1 m_2 \cdots m_n$ where $m_i \in \{0,1\}$, Alice computes $E(\text{pad}, m) = m \oplus \text{pad}$ (bitwise exclusive or). To decrypt ciphertext $c \in \{0,1\}^n$, Bob computes $D(\text{pad}, c) = \text{pad} \oplus c = \text{pad} \oplus (m \oplus \text{pad}) = m$.

We can verify this easily as follows:

$$x \oplus y = \bar{x} y + x \bar{y}$$

$$x \oplus (x \oplus y) = \bar{x}(\bar{x} y + x \bar{y}) + x(\overline{\bar{x} y + x \bar{y}})$$

$$= \bar{x} y + x(\overline{\bar{x} y})(\overline{x \bar{y}})$$

$$= \bar{x} y + x(x + \bar{y})(\bar{x} + y)$$

$$= \bar{x} y + x(xy + \bar{y}\bar{x})$$

$$= \bar{x} y + xy = (\bar{x} + x) y = y$$

It is easy to verify that $\Pr_{\text{pad}}[E(\text{pad}, m) = c] = \frac{1}{2^n}$, for all m and c.

For a single bit we have:

$$c_i = b_i \oplus m_i \Rightarrow b_i = m_i \oplus c_i$$

m_i	c_i	
0	0	$\Rightarrow b_i = 0 \oplus 0 = 0$
0	1	$\Rightarrow b_i = 0 \oplus 1 = 1$
1	0	$\Rightarrow b_i = 1 \oplus 0 = 1$
1	1	$\Rightarrow b_i = 1 \oplus 1 = 0$

$\Rightarrow \Pr_{b_i}[E(b_i, m_i) = c_i] = 1/2$

$$\Rightarrow \Pr_{\text{pad}}[E(\text{pad}, m) = c] = \frac{1}{2^n}$$

From this, it can be argued that seeing c gives no information about what has been sent. The adversary's a posteriori probability of predicting m given c is no better than her a priori probability of predicting m without being given c. Now suppose Alice wants to send Bob an additional message m'. If Alice were to simply send $c = E(pad, m')$, then the adversary can compute $E(pad, m) \oplus E(pad, m') = m \oplus m'$ which gives information about m and m' (e.g. can tell which bits of m and m' are equal and which are different). To fix this, the length of the pad agreed upon a-priori should be the sum total of the length of all messages ever to be exchanged over the insecure communication line.

The difficulty with one time pad encryption scheme is that the key length is same as the message length. This makes it impractical to exchange large messages. We can remove this difficulty by making use of <u>Pseudo-Random Bit Generators</u> (PSRG).

Pseudo-Random Bit Generators (PSRG):

A PSRG is a deterministic program used to generate long sequence of bits which looks like random sequence, given as input a short random sequence (the input seed). In one time pad encryption scheme, Alice and Bob need to agree on a short seed r, and exchange the message $G(r) \oplus m$.

Polynomial-Time Indistinguishability:

Let X_n, Y_n be probability distributions on $\{0,1\}^n$ (That is, by $t \in X_n$, we mean that $t \in \{0,1\}^n$ and it is selected according to the distribution X_n).
We say that $\{X_n\}$ is polynomial-time indistinguishable from $\{Y_n\}$ if for all probabilistic polynomial-time algorithms A, $\forall$ polynomial Q, $\exists n_0$ such that $\forall n > n_0$,

$$\left| \Pr_{t \in X_n}[A(t)=1] - \Pr_{t \in Y_n}[A(t)=1] \right| < \frac{1}{Q(n)}$$

That is, for sufficiently long strings, no probabilistic polynomial-time algorithm can tell whether the string was sampled according to X_n or according to Y_n. Intuitively, pseudo random distribution would be indistinguishable from the uniform distribution. We denote the uniform probability distribution on $\{0,1\}^n$ by U_n. That is, for every $\alpha \in \{0,1\}^n$, $\Pr_{x \in U_n}[x = \alpha] = \frac{1}{2^n}$.

<u>Pseudo Random Distribution</u>: We say that $\{X_n\}$ is pseudo random if it is polynomial-time indistinguishable from $\{U_n\}$. That is, for all probabilistic polynomial-time algorithm A, $\forall$ polynomial Q, $\exists\, n_0$, such that $\forall n > n_0$,

$$\left| \Pr_{t \in X_n}[A(t)=1] - \Pr_{t \in U_n}[A(t)=1] \right| < \frac{1}{Q(n)}$$

<u>Pseudo-Random Generator (PSRG)</u>:

A polynomial time deterministic program $G : \{0,1\}^k \longrightarrow \{0,1\}^{\hat{k}}$ is a PSRG if the following conditions are satisfied:

① $\hat{k} > k$

② $\{G_{\hat{k}}\}_{\hat{k}}$ is pseudo-random where $G_{\hat{k}}$ is the distribution on $\{0,1\}^{\hat{k}}$ obtained as follows:

to get $t \in G_{\hat{k}}$:

pick $x \in U_k$

set $t = G(x)$

That is, $\forall$ PPT algorithm A, $\forall$ polynomial Q, $\forall$ sufficiently large k,

$$\left| \Pr_{t \in G_{\hat{k}}}[A(t)=1] - \Pr_{t \in U_{\hat{k}}}[A(t)=1] \right| < \frac{1}{Q(\hat{k})}$$

Blum-Blum-Shub (BBS) PSRG :

Let p, q be two $(k/2)$-bit primes such that $p \equiv q \equiv 3 \bmod 4$, and define $n = pq$. Let $QR(n)$ denote the set of quadratic residues modulo n. A seed s_0 is any element of $QR(n)$. For $0 \le i \le l-1$, define $s_{i+1} = s_i^2 \bmod n$, and then define $f(s_0) = (z_1, z_2, \cdots, z_l)$, where $z_i \equiv s_i \bmod 2$, $1 \le i \le l$. Then f is a (k, l)-PSRG, called the BBS PSRG.

input bit-length   output bit-length

One way to choose an appropriate seed is to select an element $s_{-1} \in \mathbb{Z}_n^*$ and compute $s_0 \equiv s_{-1}^2 \bmod n$. This ensures that $s_0 \in QR(n)$.

Example : Suppose $n = 192649 = 383 \times 503$, and $s_0 = 101355^2 \bmod n = 20749$.

i	s_i	z_i
0	20749	[illegible]
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0