Chapter 5

SECURITY SYSTEM FOR MIGRATING CRAWLERS

5.1 GENERAL

As discussed in chapter 2, owing to their inherent security problems, migrants have limited applications especially when they are more prone to misuse by some other applications, resulting in increase in the scale of threats. Nevertheless migrants are potential contributors for implementation of migrating crawlers because of their capability to move to the information resource itself. In this chapter, general security objectives for migrants are identified and corresponding mechanisms for facing the identified threats have been designed.

5.2 SECURITY THREATS

The main threat categories to migrating crawlers are given below:

1. Host to migrant attack
2. Migrant to host attack
3. Migrant to migrant attack
4. Third party attack.

A migrant created by a trustworthy system such as crawler manager is a safe system i.e. non malicious. Hence the migrant to host attack and migrant to migrant attack become apparently irrelevant as far as working of migrating crawler is concerned. Whereas in reality, a migrant being safe does not guarantee to be secure in the sense that a malicious host may attack a migrant and corrupt it during its itinerary. Having got corrupted the migrant starts behaving like a malicious one, so all these four threat categories including migrant to host attack and migrant to migrant attack become relevant.

In the following sections each of the above identified threat category and its proposed solutions are discussed.
5.2.1 HOST TO MIGRANT ATTACK

A migrant is susceptible to the following attacks during its itinerary on the Internet.

- **Masquerade:** In order to deceive, any of the visited host may hide its actual identity from a migrant and may masquerade to extract the sensitive information like its origin, resources held etc. from the migrant. This sensitive information becomes the source for future attacks by the masquerading host.

- **Denial of Service:** When a migrant arrives at a host, it expects the host to execute its requests faithfully and provide fair allocation of resources. However, a malicious host may ignore migrant’s service requests, introduce unacceptable delays for critical tasks or even terminate the migrant without notification. These types of attacks from a host are known as denial of service attacks.

- **Eavesdropping:** A host may intercept and monitor the secret communications among the migrants and between migrant and the mother system i.e. the crawler manager. Additionally, since the host has access to the migrant’s code, state, and data, the host can steal proprietary algorithms or other sensitive information from the migrant.

- **Alteration:** In the light of above threats, a host having been privy to the sensitive information of the migrant may alter its code, state, and data and may change the behavior of the migrant. Such migrants may produce wrong picture of the collection at search engine side.

5.2.2 MIGRANT TO HOST ATTACK

A host after stealing the information from a valid migrant may kidnap any visiting migrant to create its own migrant i.e. a clone. A kidnapped migrant may behave like a malicious one and may attack the next visiting host. Henceforth, such a kidnapped migrant will be referred to as malicious migrant.

Following are some of the possible attacks by a malicious migrant to a visiting host:
• **Masquerading**: A malicious migrant may masquerade on behalf of its foster host in order to gain access to the services and resources of a server resulting in its non-availability to its mother system i.e. crawler manager. A malicious action performed by the kidnapped migrant, may lead to damage of the trust established between a genuine migrant and the host.

• **Denial of Service**: The malicious migrant may carry harmful code designed to disrupt the services of other hosts by unnecessarily consuming their resources in such a way that they could not provide services to their legitimate users resulting in denial of service. After such an attack, hosts either responds slowly or completely shut down leading to the state of unavailability.

• **Unauthorized Access**: Nevertheless, the malicious migrant definitely accesses unauthorized information from servers on behalf of its foster host.

### 5.2.3 MIGRANT TO MIGRANT ATTACK

A corrupt migrant may even attack a legitimate migrant. The following are the examples of migrant to migrant attacks.

• **Masquerade**: Migrant-to-migrant communication can take place directly between two migrants or may require the participation of the underlying host. In either case, a malicious migrant may deceive the communicating migrant for the purpose of extracting information on behalf of its foster host.

• **Denial of Service**: In addition to launching denial of service attacks on host, a malicious migrant can also launch denial of service attacks against other migrants by repeatedly sending messages to the migrants. This bombardment of messages will result in unnecessary burden on the migrant causing loss in response time of that migrant, thereby reducing the performance of the migrating crawler.

• **Repudiation**: A malicious migrant participating in a transaction or communication may later claim that the transaction or communication never took place.
5.2.4 THIRD PARTY ATTACK

- **Alteration:** Generally, a migrant after locally collecting the information sends it to the crawler manager in its original form. Such unsecured transmission can be intercepted by a third party to change the contents of the information. This fabrication of information may lead to wrong indexing of information by search engine.

5.3 SECURITY SYSTEM FOR MIGRATING CRAWLER

In this work, mechanisms have been suggested that help the migrating crawlers to become more secure and robust to possible attacks. Before a discussion on the proposed mechanisms is done, let us have a look at the following expected characteristics that need to be considered [90, 97]:

- **Isolation:** User programs and migrants must be protected from each other, and the host should also be protected from malicious migrants.
- **Adaptability:** The migrant must be adaptable to the environment in which it runs i.e. it must be programmed to respond to the differing trust levels of the hosts that it visits and adapt its defenses accordingly.
- **Survivability:** The migrant should be capable of surviving the attacks i.e. its code should be intact. In the event of host to migrant attack, the migrant should replicate its code to remain unaffected by the attack.
- **Believability:** Mechanism may be developed to verify the authenticity of the information supplied by a migrant.
- **Confidentiality:** The information or documents carried by a migrant must remain confidential during the transportation.

In the following sections the proposals are made for protecting the migrant, host and data from the possible attacks.
5.4 MIGRANT PROTECTION

In this work, a novel mechanism to protect a migrant against malicious host attacks has been designed. This mechanism is based on analyzing the probable causes of migrant failure to perform its intended function. The attacks from a host can be categorized into two types: active and passive. In a passive attack, the host does not interfere with the migrant, but only attempts to extract useful information from it. In active attacks, the host can intercept and modify code and data of the migrant. The proposed mechanism takes care of both types of attack. The idea is to make an image of migrant in terms of its code and state before its migration to next host. If any suspicious activity is detected then the saved image is used to restore the migrant to its original form.

A unit for migrant security, called Migrant Security Module (MSM) consisting of two components namely monitor and restore, runs at penultimate host (previous to current) as shown in Fig. 5.1, has been designed.

![Fig. 5.1: Migrant Security Module (MSM)](image)

The *monitor* observes the behavior of migrant executing at current host through remote monitoring. *Restore* maintains an image of the migrant before migration. In case of an attack, monitor destroys the malicious migrant and sends a signal to the process called *restore*. Restore regenerates the migrant from its stored image and migrates it to an alternate host/server.
A discussion on both the components is given in following section.

**5.4.1 MONITOR**

Monitor continuously observes the behavior of the executing migrant. Based on the type of attack, from the current host, the victim migrant accordingly must show analogous symptoms. A list of possible symptoms is given in table 5.1.

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Symbolic Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long execution time</td>
<td>(LET)</td>
</tr>
<tr>
<td>Extraordinary time spent at a host</td>
<td>(EOTH)</td>
</tr>
<tr>
<td>Change in size of code of migrant</td>
<td>(CSC)</td>
</tr>
<tr>
<td>Repeated intermittent execution of migrant code</td>
<td>(RIE)</td>
</tr>
</tbody>
</table>

The monitor uses two counters called host timer (HT) and execution timer (ET) for tracking the extraordinary time spent by a migrant at a host and long execution time respectively. Monitor waits on a signal called migrated from restore component. After receiving the signal it observes the symptoms of executing migrant through remote monitoring. Now based upon the combination of symptoms (as given in Table 5.2), it assesses possible attack type.

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Possible Attack Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LET &amp; CSC &quot;or&quot; CSC &amp; EOTH</td>
<td>Masquerading</td>
</tr>
<tr>
<td>LET &amp; EOTH</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CSC &amp; RIE</td>
<td>Alteration or Eavesdropping</td>
</tr>
</tbody>
</table>

105
The combination of *long execution time* with *change in size of code of migrant* or *extraordinary time spent at a host* with *change in size of code of migrant* indicates the possibility of masquerading attack by a malicious host. Similarly, the combination of *long execution time* and *extraordinary time spent at a host* indicates the possibility of denial of service attack. The combination of *change in size of code of migrant* and *repeated intermittent execution of migrant code* indicates the possibility of alteration or eavesdropping attack. Having any of the above conditions encountered, the monitor stops execution of migrant. After a random time interval it resumes the execution of migrant and assesses it. If it detects the possible attacked condition three successive times, it destroys the migrant and sends a signal *restore_m* to the *restore* component to reinstate the migrant from its stored image. The algorithm for monitor component is given in Fig. 5.2.

```
Monitor ()
Step 1: wait (migrated);
2: initialize the counters ET and HT with worst case values;
3: Initialize LOOP =3 and FLAG=4;
4: while (LOOP>0 && FLAG>0)
   4.1 FLAG =0
   4.2: monitor the symptoms
      if ( (LET & CSC) “or” (CSC & EOTH))
         4.2.1: FLAG =1;
      if (LET & EOTH)
         4.2.2: FLAG =2;
      if (CSC & RIE)
         4.2.3: FLAG=3;
4.3: if (FLAG>0)
   4.3.1 stop execution of migrant
   4.3.2 LOOP=LOOP-1;
   4.3.3 if LOOP >0
      4.3.3.1 delay (1000);
      4.3.3.2 Resume migrant execution and go to setp 2
   else
      4.3.3.3 break;
5: if (FLAG > 0)
   5.1: destroy the migrant;
   5.2: signal (restore_m);
```

Fig. 5.2: Algorithm for Monitor Component

A discussion on restore component is given in following section.
5.4.2 RESTORE

The Restore component creates an image of the migrant by storing its code and state at a temporary storage before sending the migrant to the next host. The state of a migrant is defined as the data carried by a migrant, e.g. path history and downloaded documents if any, from its previously visited hosts. Thereafter it sends a signal called migrated to the monitor component and waits on the signal restore_m from the monitor. As soon as it receives the signal restore_m it re-creates the migrant from the stored image and sends it to the next host in the list for downloading the documents. It also informs the crawler manager about the malicious host by sending a message containing the host ID. Crawler manager uses this information to black list the host by setting its valid tag as ‘0’. The algorithm for the restore component is given in Fig. 5.3.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>create an image of migrant;</td>
</tr>
<tr>
<td>2.</td>
<td>signal (migrated);</td>
</tr>
<tr>
<td>3.</td>
<td>wait (restore_m);</td>
</tr>
<tr>
<td>4.</td>
<td>re-create the migrant from stored image</td>
</tr>
<tr>
<td>5.</td>
<td>send the migrant to the next host in the list</td>
</tr>
<tr>
<td>6.</td>
<td>send a message containing hostid of malicious host to crawler manager</td>
</tr>
</tbody>
</table>

Fig. 5.3: Algorithm for Restore Component

5.5 HOST PROTECTION

The Host Security Manager (HSM) secures a host from possible attacks by malicious migrant. It resides on the host as shown in Fig. 5.4.

Fig. 5.4: Host Protection using Security Manager
The host security manager employs two major components \textit{Migrant watch} (MW) and \textit{Validate visit} (VV) as shown in Fig 5.5.

![Host Security Module Diagram](image)

Fig. 5.5: Host Security Module

It is suggested that before entering a host, the migrant has to pass through host security module and only after getting the permission from a host security module, a migrant may get access to the host platform.

The host security manager waits for appropriate signals from \textit{Migrant watch} and \textit{Validate visit} components and allocates a separate address space for migrant execution. Separate address space mitigates the possibility of denial of service attack. The algorithm for host security manager is given in Fig. 5.6.

```
Host Security Manager ( )
Step 1: wait for migrant
2: call migrant watch module (pass code);
3: wait (migrant_authenticated);
4: call validate visit module (visit history);
5: wait (safe)
6: If (safe & migrant_authenticated)
   6.1: Allow migrant to enter in to the host
   6.2: Allocate it a separate address space for execution
Else
   6.3: Access to the host is denied
```

Fig. 5.6: Algorithm for Host Security Manager

108
A discussion on the *Migrant watch (MW)* and *validate visit (VV)* components are given in following section.

### 5.5.1 MIGRANT WATCH

A migrant is assigned a unique pass code generated by private key by the crawl manager. *Migrant watch* verifies the pass code of visiting migrant with the help of a public key supplied by the crawl manager. In fact, it uses asymmetric key principle [75] for verifying the authenticity of an incoming migrant. The arrangement for migrant authentication is shown in Fig. 5.7.

After authentication the *Migrant watch* sends a signals *migrant_authenticated* to the host security manager. The algorithm for *Migrant watch* component is shown in Fig. 5.8.

---

**Fig. 5.7: Authentication Process**

**Fig. 5.8: Algorithm for Migrant Watch Module**

---

109
5.5.2 VALIDATE VISIT

When a network is connected to another network then all the traffic from one network to another must pass through a bridge. Similarly, when a migrant travels from a sub network to another sub network then it must travel through some specified intermediate nodes. Consider the network arrangement given in Fig. 5.9.

![Network Diagram]

It may be observed that all traffic coming from “sub network A” to “sub network B” must pass through node 5 and node 6. Where, node 5 and 6 contain the subnet ID’s of “sub network A” and “sub network B” respectively. When a crawl manager receives the signal ‘malicious_hostid’ from restore module of migrant protection system, it registers the ‘host_id’ as dirty and broadcasts periodically this information to the different nodes lying on the network. Each host maintains the two lists: list of mandatory nodes and list of dirty nodes. Where mandatory nodes are the nodes through at least one of which, the migrant must reach the host. The dirty nodes is a list of black listed nodes. The migrant maintains a list called List_of_visited_nodes containing information about the nodes it has already visited before reaching the current host. The validate visit module checks the list_of_visited_nodes for presence of mandatory nodes and dirty nodes. If mandatory node is present and dirty nodes are not present then migrant is considered to be safe. If both the conditions are satisfied then visit validation module sends the signal safe. If an incoming migrant is not containing the mandatory nodes then it is taken to be fabricated one i.e. is unsafe. The algorithm for visit validation module is shown in Fig 5.10.
Validate Visit()

Step 1: Check the list_of_visited_nodes for presence of mandatory and dirty nodes.
2: if any of the mandatory nodes is present in the list_of_visited_nodes and none of the dirty node is present then
2.1 signal (safe);
else
2.2 block the migrant.

Fig. 5.10: Algorithm for Validate Visit Module

Having secured the migrant and its related attacks, the data/information carried or sent by the migrant needs to be protected during transportation. A method to encrypt the data has been developed. A brief discussion on the same is given in the next section.

5.6 A NOVEL MECHANISM FOR SECURING CRAWLER DATA

A migrant downloads text, symbols, images, sound and media files. During the transmission of the information, there is likelihood that a mischievous host may masquerade to steal the information. Thus, there is a need to encrypt the information at the migrant level. In this work, a novel method for encrypting the text information has been developed and implemented. The method utilizes a combination of an invertible function [76] and multiplicative inverse [77] for the required text encryption. An invertible function can be precisely defined as:

A function $F^{-1}$ is called an invertible function if it exactly reverses the transformation produced by a function $F$. For a given domain and co-domain $x$ and $y$, and $x \in X, y \in Y$, the relation can be defined as given below in equation 5.1:

$$\text{If } f(x) = y \text{ then } f^{-1}(y) = x.$$  \hspace{1cm} (5.1)

Similarly, a multiplicative inverse can be precisely defined as:
Two elements $U$ and $V$ are known to be the **multiplicative inverse** of each other over prime number $W$ if they hold the following property:

$$U \times V \pmod{W} = 1.$$  \hspace{1cm} (5.2)

### 5.6.1 THE MECHANISM

The mechanism developed for securing crawler data requires the following secret keys, essentially prime numbers. These keys are shared between crawler manager and the migrants:

- $P$
- $Q$
- $M$
- An invertible function $F(a)$

where: $P$, $Q$ & $M$ are any prime numbers.

The above keys are used to generate a pivot called ‘$N$’ as per the relation given below:

$$N = P \times Q$$

accordingly the value of key ‘$M$’ is chosen larger than ‘$N$’ i.e. $M > N$.

The above mentioned keys and the pivot are employed to encrypt the text as per the Flow chart shown in Fig. 5.11. It summarizes the complete encryption and decryption process. Where: $IK_1$, $IK_2$, and $IK_3$ are the intermediate keys.
5.6.1.1 Encryption process

Let us encrypt the string "INDIA IS GREAT" for which the following steps would be followed. Given

\[ P = 251, \quad Q = 17 \quad \text{and} \quad M = 9973 \]

For each character in the string repeat step 1 to 4.

**Step 1:** Apply the following relation on ASCII equivalent (i.e. ascEq) of the character.

\[
IK_1 = \text{ascEq} + P \times (R \% 100) \quad (5.3)
\]
Where: R is a random number, say 31288.
ascEq is ASCII equivalent of the character.
The first character in the string is “I” with ascEq = 73.
Puting ascEq =73 and R=31288, P=251 in relation (5.3), we get
IK₁ = 73 + 251*88
= 22161

**Step 2:** The intermediate key IK₁ obtained from above relation is employed to get the next level of intermediate key IK₂ by the following relation.

\[ IK₂ = IK₁ \mod N \] (5.4)

Putting Q=17 and IK₁ 22161 in relation (5.4), we get
IK₂ = 22161 mod 4267.
= 826

**Step 3:** The next level of encryption IK₃ is obtained by computing the multiplicative inverse of IK₂ i.e. (IK₂)' with respect to M as per relation given below. Putting the value of intermediate key IK₂ in the following relation:

\[ (IK₂ \cdot (IK₂)') \mod M = 1 \] (5.5)

where: (IK₂)' is the multiplicative inverse of IK₂ over M.

It may be noted that the multiplicative inverse is generated by extended euclidean algorithm, iterative method [79].

Putting the value of M = 9973, and IK₂ we get:

\[ (826 \cdot (IK₂)') \mod 9973 = 1. \]

\[ (IK₂)' = 6049. \text{ Because } (826 \cdot 6049) \mod 9973 = 1 \]

This (IK₂)' becomes the next encryption IK₃.

**Step 4:** Now to get the final encryption key (FEK) employ the value of IK₃ in the invertible function satisfying the following relation.

\[ f(IK₃) = FEK \text{ then } f^{-1}(FEK) = IK₃ \] (5.6)
Taking an invertible function (say Y=5X+3) and putting the value of IK₃ in place of X, we get:

\[ Y = 5 \times 6049 + 3 = 30248 \]

The resultant invertible function f(IK₃) is the final encryption of text “I” in the test string.

Similarly, applying step 1 to 4 for other characters of the string, we get

INDIA IS GREAT = \{30248, 11708, 19033, 36478, 523, 22733, 24078, 253, 23898, 27723, 13728, 42168, 11898, 13348\}

This encrypted form of information is transmitted by the migrant to the crawler manager.

At the crawler manager side the decryption process is applied to convert the encrypted text to the actual text. The decryption process takes the following steps:

5.6.1.2 Decryption Process

The decryption process is carried out by using the following steps. For each encrypted character apply step 1 to 3.

**Step 1:** By applying relation (5.6) on the received data we get IK₃ as follows:

If \( Y = 5X + 3 \) Then,
\[ X = \frac{(Y-3)}{5} \]
Here \( Y = 30248 \) then,
\[ X = \frac{(30248-3)}{5} = 6049 \] which is IK₃ for this case.

**Step 2:** Now putting this value of IK₃ i.e \((IK₂)’\) in relation (5.5), we get multiplicative inverse of IK₃ over M, i.e.
\[ IK₂ = 826 \] Because \((826 \times 6049) \text{ mode } 9973 = 1\]

**Step 3:** Now to get the final text, employ the value of IK₂ and P in the following relation:
we get,
\[ a = 826 \mod 251 = 73, \]
which is ASCII equivalent of text "I"

Similarly we get the complete string: "INDIA IS GREAT"

5.6.2 A TYPICAL EXAMPLE OF ENCRYPTION FOR ALL TYPES OF DATA

In the following example the encryption and decryption has been applied on a string "aba5#" that contains three characters, one digit and one special symbol. The process takes P, Q, M and an invertible function as shared keys among the migrant and the search engine.

Assume: P=251,
Q=17,
M=9973 and
Invertible function \( f(X) = 5X + 3 \)
and \( N = P \times Q \Rightarrow N = 251 \times 17 = 4267 \)

5.6.2.1 Encryption Process

Step 1: Process extracts one character from string, encrypts it until whole string is encrypted and send the encrypted string over channel.

Now by taking the ASCII value corresponding to each character we get:

'\( a \)' = 97
'\( b \)' = 98
'5' = 53
'\#' = 35

So,
\[ x_1 = 97, \]
\[ x_2 = 98, \]
\[ x_3 = 97, \]
\[ x_4 = 53 \& \]
\[ x_5 = 35 \]

Let the value of \( R \) is assumed for each case as \( R_1 = 64, R_2 = 76, R_3 = 56, R_4 = 10 \& R_5 = 94 \) respectively.

Therefore, the value of \( A_1, A_2, A_3, A_4 \) and \( A_5 \) is computed using equation number (5.3) we get:

\[ A_1 = 97 + 64 \times 251 = 16161, \]
\[ A_2 = 98 + 76 \times 251 = 19174, \]
\[ A_3 = 97 + 56 \times 251 = 14153, \]
\[ A_4 = 53 + 10 \times 251 = 2563 \text{ and} \]
\[ A_5 = 35 + 94 \times 251 = 23629. \]

Now first cipher is calculated using equation number (5.4) we get:

\[ E_1 (97) = 16161 \text{ mod } 4267 = 3360, \]
\[ E_2 (98) = 19174 \text{ mod } 4267 = 2106, \]
\[ E_3 (97) = 14153 \text{ mod } 4267 = 1352, \]
\[ E_4 (53) = 2563 \text{ mod } 4267 = 2563 \text{ and} \]
\[ E_5 (35) = 23629 \text{ mod } 4267 = 2294. \]

**Step 2:** Now take multiplicative inverse of \( E_1 (97), E_2 (98), E_3 (97), E_4 (53) \) & \( E_5 (35) \) using equation (5.5) over \( M \) (\( M = 9973 \)) for finding \( E'(x) \).

\[ E_1' (97) = 9507, \text{ Because: } (3360 \times 9507) \text{ mod } 9973 = 1 \]
\[ E_2' (98) = 8093, \text{ Because: } (2106 \times 8093) \text{ mod } 9973 = 1 \]
\[ E_3' (97) = 9346, \text{ Because: } (1352 \times 9346) \text{ mod } 9973 = 1 \]
\[ E_4' (53) = 2109, \text{ Because: } (2563 \times 2109) \text{ mod } 9973 = 1 \]
\[ E_5' (35) = 3804, \text{ Because: } (2294 \times 3804) \text{ mod } 9973 = 1 \]

**Step 3:** Now putting the values of \( E'(x) \) into invertible function \( f(X) = 5X + 3 \). we get \( f(E'(x)) \) as follows:
\begin{align*}
f(9507) &= 47538, \\
f(8093) &= 40468, \\
f(9346) &= 46733, \\
f(2109) &= 10548, \\
f(3804) &= 19023
\end{align*}

Therefore cipher text of character 'a', 'b', 'a', '5' & '#' would be 47538, 40468, 46733, 10548 & 19023 respectively. And these values of \( f(x) \) are sent over communication channel as final cipher text.

5.6.2.2 Decryption Process

**Step 1:** Put all received values in invertible function \( f(X) = 5X + 3 \) for retrieving the value of \( E'(x) \) from it. i.e.

\[
\begin{align*}
\text{if } f((E'(x)) - 5X + 3, \\
\text{then } X = (f((E'(x)) - 3) / 5.
\end{align*}
\]

After applying the inverse process we get original multiplicative inverse as follows:

\[
\begin{align*}
E_1'(97) &= 9507, & \text{because: } (47538-3)/5 = 9507 \\
E_2'(98) &= 8093, & \text{because: } (40468-3)/5 = 8093 \\
E_3'(97) &= 9346, & \text{because: } (46733-3)/5 = 9346 \\
E_4'(53) &= 2109, & \text{because: } (10548-3)/5 = 2109 \\
E_5'(35) &= 3804, & \text{because: } (19023-3)/5 = 3804
\end{align*}
\]

**Step 2:** Now by taking multiplicative inverse of \( E_1'(97), E_2'(98), E_3'(97), E_4'(53) \) & \( E_5'(35) \) using equation (5.5) over \( M = 9973 \) for finding \( E(x) \) we get:

\[
\begin{align*}
E_1(97) &= 3360, & \text{because: } (3360*9507) \ mod \ 9973 = 1 \\
E_2(98) &= 2106, & \text{because: } (2106*8093) \ mod \ 9973 = 1 \\
E_3(97) &= 1352, & \text{because: } (1352*9346) \ mod \ 9973 = 1 \\
E_4(53) &= 2563, & \text{because: } (2563*2109) \ mod \ 9973 = 1 \\
E_5(35) &= 2294, & \text{because: } (2294*3804) \ mod \ 9973 = 1
\end{align*}
\]
**Step 3:** Now to get the final text employ the value of IK₂ and P in the relation (5.7) we get:

\[ x_1 = 97, \text{ Because: } 3360 \mod 251 = 97, \text{ which is ASCII equivalent to ‘a’.} \]
\[ x_2 = 98, \text{ Because: } 2106 \mod 251 = 98, \text{ which is ASCII equivalent to ‘b’.} \]
\[ x_3 = 97, \text{ Because: } 1352 \mod 251 = 97, \text{ which is ASCII equivalent to ‘a’.} \]
\[ x_4 = 53, \text{ Because: } 2263 \mod 251 = 53, \text{ which is ASCII equivalent to ‘5’.} \]
\[ x_5 = 35, \text{ Because: } 2294 \mod 251 = 35, \text{ which is ASCII equivalent to ‘#’.} \]

A juxtaposition of above received character gives the following string:

“aba5#”

which is the actual transmitted text string.

It may be noted that though the text x₁ and x₃ both represent the same character ‘a’ but even then the intermediate keys and final encryption keys are different thereby reducing the possibility of guessing the contents by a malicious host by predicting the repeated words. Moreover, the unpredictability has been added by replacing the linear relationship between plain text & cipher text by random numbers.

In the next chapter a mechanism for improving the temporal relevance of the documents is discussed.