BILL RECORD LECTURE!!!!
The One-Time Pad
Trying to Fake the OTP
Failing To Do So
The One-Time Pad
One-Time Pad

Let $M = \{0, 1\}^n$, the set of all messages.

- $Gen$: choose a uniform key $k \in \{0, 1\}^n$.
- $Enc_k(m) = k \oplus m$.
- $Dec_k(c) = k \oplus c$.

Correctness: $Dec_k(Enc_k(m)) = k \oplus (k \oplus m) = k \oplus k \oplus m = m$. 

One-Time Pad

- Let $\mathcal{M} = \{0, 1\}^n$, the set of all messages.
One-Time Pad

- Let $\mathcal{M} = \{0, 1\}^n$, the set of all messages.

- $Gen$: choose a uniform key $k \in \{0, 1\}^n$. 
One-Time Pad

- Let $\mathcal{M} = \{0, 1\}^n$, the set of all messages.
- $Gen$: choose a uniform key $k \in \{0, 1\}^n$.
- $Enc_k(m) = k \oplus m$. 

Correctness:

$Dec_k(Enc_k(m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = m$
One-Time Pad

- Let $\mathcal{M} = \{0, 1\}^n$, the set of all messages.

- $Gen$: choose a uniform key $k \in \{0, 1\}^n$.

- $Enc_k(m) = k \oplus m$.

- $Dec_k(c) = k \oplus c$. 

- Correctness: $Dec_k(Enc_k(m)) = k \oplus (k \oplus m) = m$. 
One-Time Pad

- Let $\mathcal{M} = \{0, 1\}^n$, the set of all messages.

- $\text{Gen}$: choose a uniform key $k \in \{0, 1\}^n$.

- $\text{Enc}_k(m) = k \oplus m$.

- $\text{Dec}_k(c) = k \oplus c$.

- Correctness:

  $$\text{Dec}_k(\text{Enc}_k(m)) = k \oplus (k \oplus m)$$
  $$= (k \oplus k) \oplus m$$
  $$= m$$
Example Of One-Time Pad

Key is 10001010001000111110111100

1. PRO $\oplus$ is FAST!

2. CON If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable: VOTE: Yes, No, or Other. Yes. Really! Caveat: Generating truly random bits is hard.
Example Of One-Time Pad

Key is 10001010001000111110111100
Alice wants to send Bob 1110.
Example Of One-Time Pad

Key is 100010100010001111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.

1. \text{PRO} $\oplus$ is FAST!
2. \text{CON} If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable: VOTE: Yes, No, or Other.
Yes. Really!
Caveat: Generating truly random bits is hard.
Example Of One-Time Pad

Key is 1000101000100011111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
Example Of One-Time Pad

Key is 10001010001000111110111110
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$. 

1. $\oplus$ is FAST!
2. If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable:

VOTE: Yes, No, or Other.

Yes. Really!
Caveat: Generating truly random bits is hard.
Example Of One-Time Pad

Key is 100010100010001111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$.

1. **PRO** $\oplus$ is FAST!

2. **CON** If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable: VOTE: Yes, No, or Other.

Yes. Really!
Caveat: Generating truly random bits is hard.
Example Of One-Time Pad

Key is 10001010001000111110111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$.

1. **PRO** $\oplus$ is FAST!
2. **CON** If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable:

VOTE: Yes, No, or Other.

Yes. Really!

Caveat:
Generating truly random bits is hard.
Example Of One-Time Pad

Key is 100010100010001111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$.

1. **PRO** $\oplus$ is FAST!
2. **CON** If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable:
**VOTE:** Yes, No, or Other.
Example Of One-Time Pad

Key is 1000101000100011111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$.

1. **PRO** $\oplus$ is FAST!

2. **CON** If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable:

**VOTE:** Yes, No, or Other.
Yes. Really!
Example Of One-Time Pad

Key is 100010100010001111101111100
Alice wants to send Bob 1110.
She sends $1110 \oplus 1000 = 0110$.
Then Bob wants to send Alice 00111.
He sends $00111 \oplus 10100 = 10011$.

1. **PRO** $\oplus$ is FAST!

2. **CON** If Key is $N$ bits long can only send $N$ bits.

Is the one-time pad uncrackable:

**VOTE:** Yes, No, or Other.
Yes. Really!

**Caveat:** Generating truly random bits is hard.
One-time pad
One-time pad (OTP)

The OTP was patented in 1917 by Vernam. Historical research indicates the OTP was invented at least 35 years earlier. The OTP was proven info-theoretic secure by Shannon in 1949.
One-time pad (OTP)

The OTP was patented in 1917 by Vernam.
One-time pad (OTP)

- The OTP was patented in 1917 by Vernam.
- Historical research indicates the OTP was invented at least 35 years earlier.
One-time pad (OTP)

- The OTP was patented in 1917 by Vernam.
- Historical research indicates the OTP was invented at least 35 years earlier.
- The OTP was proven info-theoretic secure by Shannon in 1949.
Linear Cong. Generators
How Hard is it to Generate Truly Random Bits?

Paraphrase of a Recent Piazza conversation

Student  You said that generating Random Bits is hard. Why?
Paraphrase of a Recent Piazza conversation

**Student**  You said that generating Random Bits is hard. Why?

**Bill**  *Truly*  Rand Bits are hard. How would you do it?
Paraphrase of a Recent Piazza conversation

**Student**  You said that generating Random Bits is hard. Why?

**Bill**  *Truly*  Rand Bits are hard. How would you do it?

**Student**  Just use the Random function in Java!
Paraphrase of a Recent Piazza conversation

**Student** You said that generating Random Bits is hard. Why?

**Bill** *Truly* Rand Bits are hard. How would you do it?

**Student** Just use the Random function in Java!

**Bill** Okay. How does Java do it? Is it *Truly* Random?
Paraphrase of a Recent Piazza conversation

**Student** You said that generating Random Bits is hard. Why?

**Bill** *Truly* Rand Bits are hard. How would you do it?

**Student** Just use the Random function in Java!

**Bill** Okay. How does Java do it? Is it *Truly* Random?

**Student** Oh.
How Hard is it to Generate Truly Random Bits?

Paraphrase of a Recent Piazza conversation

**Student**  You said that generating Random Bits is hard. Why?

**Bill**  *Truly*  Rand Bits are hard. How would you do it?

**Student**  Just use the Random function in Java!

**Bill**  Okay. How does Java do it? Is it *Truly* Random?

**Student**  Oh. Okay, you tell me— how does Java do it?
Paraphrase of a Recent Piazza conversation

**Student**  You said that generating Random Bits is hard. Why?

**Bill**  *Truly*  Rand Bits are hard. How would you do it?

**Student**  Just use the Random function in Java!

**Bill**  Okay. How does Java do it? Is it *Truly* Random?

**Student**  Oh. Okay, you tell me— how does Java do it?

**Bill**  I will show what Java does and why it bytes.
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**. When the computer is turned on (and once a month after that):

1. Pick \( M \) large. A power of 2 makes life easier for Alice and Bob, but might not want to do that—we’ll see why later.
2. \( A, B, x_0 \) are random-looking. E.g. the number of nanoseconds since last time reboot.
3. The computer has the recurrence \( x_{i+1} = Ax_i + B \mod M \)
4. The \( i \)th time a random number is chosen, use \( x_i \).
5. Computer need only keep \( x_i, A, B, M \) in memory. Depending on \( A, B, x_0 \) this can look random... or not.
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**. When the computer is turned on (and once a month after that):

1. Pick \( M \) large. A power of 2 makes life easier for Alice and Bob, but might not want to do that— we’ll see why later.

2. \( A, B, x_0 \) are random-looking. E.g. the number of nanoseconds mod \( M \) since last time reboot.

3. The computer has the recurrence:
   \[
   x_{i+1} = Ax_i + B \pmod{M}
   \]

4. The \( i \)th time a random number is chosen, use \( x_i \).

5. Computer need only keep \( x_i, A, B, M \) in memory. Depending on \( A, B, x_0 \) this can look random... or not.
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**.
When the computer is turned on (and once a month after that):

1. Pick $M$ large. A power of 2 makes life easier for Alice and Bob, but might not want to do that— we’ll see why later.

2. $A, B, x_0$ are random-looking. E.g. the number of nanoseconds mod $M$ since last time reboot.
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**. When the computer is turned on (and once a month after that):

1. Pick $M$ large. A power of 2 makes life easier for Alice and Bob, but might not want to do that— we’ll see why later.

2. $A, B, x_0$ are random-looking. E.g. the number of nanoseconds mod $M$ since last time reboot.

3. The computer has the recurrence

\[ x_{i+1} = Ax_i + B \pmod{M} \]
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**. When the computer is turned on (and once a month after that):

1. Pick $M$ large. A power of 2 makes life easier for Alice and Bob, but might not want to do that— we’ll see why later.
2. $A, B, x_0$ are random-looking. E.g. the number of nanoseconds mod $M$ since last time reboot.
3. The computer has the recurrence

   \[ x_{i+1} = Ax_i + B \pmod{M} \]

4. The $i$th time a random number is chosen, use $x_i$. 
How Does Java Produce Random Numbers

Java (and many old langs) uses a Linear Cong. Generator. When the computer is turned on (and once a month after that):

1. Pick $M$ large. A power of 2 makes life easier for Alice and Bob, but might not want to do that—we’ll see why later.

2. $A$, $B$, $x_0$ are random-looking. E.g. the number of nanoseconds mod $M$ since last time reboot.

3. The computer has the recurrence

   \[
   x_{i+1} = Ax_i + B \pmod{M}
   \]

4. The $i$th time a random number is chosen, use $x_i$.

5. Computer need only keep $x_i$, $A$, $B$, $M$ in memory.
How Does Java Produce Random Numbers

Java (and many old langs) uses a **Linear Cong. Generator**. When the computer is turned on (and once a month after that):

1. Pick $M$ large. A power of 2 makes life easier for Alice and Bob, but might not want to do that— we’ll see why later.
2. $A, B, x_0$ are random-looking. E.g. the number of nanoseconds mod $M$ since last time reboot.
3. The computer has the recurrence

   $$x_{i+1} = Ax_i + B \pmod{M}$$

4. The $i$th time a random number is chosen, use $x_i$.
5. Computer need only keep $x_i, A, B, M$ in memory. Depending on $A, B, x_0$ this can look random... or not.
Restrictions on $A, B, M$

What if $M$ and $A$ share a factor?
Restrictions on $A$, $B$, $M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$
Restrictions on $A, B, M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$

$x_1 = 2 \times 5 + 5 = 15 \equiv 7$
Restrictions on $A$, $B$, $M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$

$x_1 = 2 \times 5 + 5 = 15 \equiv 7$

$x_2 = 2 \times 7 + 5 = 19 \equiv 3$
Restrictions on $A, B, M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$

$x_1 = 2 \times 5 + 5 = 15 \equiv 7$

$x_2 = 2 \times 7 + 5 = 19 \equiv 3$

$x_3 = 2 \times 3 + 5 = 11 \equiv 3$
Restrictions on $A, B, M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$

$x_1 = 2 \times 5 + 5 = 15 \equiv 7$

$x_2 = 2 \times 7 + 5 = 19 \equiv 3$

$x_3 = 2 \times 3 + 5 = 11 \equiv 3$

$(\forall i \geq 2)[x_i = 3]$. 
Restrictions on $A$, $B$, $M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$

$x_{n+1} \equiv 2x_n + 5 \pmod{8}$

$x_1 = 2 \times 5 + 5 = 15 \equiv 7$
$x_2 = 2 \times 7 + 5 = 19 \equiv 3$
$x_3 = 2 \times 3 + 5 = 11 \equiv 3$

$(\forall i \geq 2)[x_i = 3]$.

This is typical. If $A$ is not rel prime to $M$ then the numbers obtained will be only a small part of $\{0, \ldots, M - 1\}$. 

Eve will assume that $A$ and $M$ are rel prime.
Restrictions on $A, B, M$

What if $M$ and $A$ share a factor?

**Example**

$x_0 = 5$
$x_{n+1} \equiv 2x_n + 5 \pmod{8}$
$x_1 = 2 \times 5 + 5 = 15 \equiv 7$
$x_2 = 2 \times 7 + 5 = 19 \equiv 3$
$x_3 = 2 \times 3 + 5 = 11 \equiv 3$

$(\forall i \geq 2)[x_i = 3]$.

This is typical. If $A$ is not rel prime to $M$ then the numbers obtained will be only a small part of $\{0, \ldots, M - 1\}$.

Eve will assume that $A$ and $M$ are rel prime.
Example of Linear Cong. Gen

\[ x_0 = 21, \ A = 19, \ B = 30, \ M = 91 \]
\[ x_0 = 21 \]
\[ x_1 = 19 \times 21 + 30 \pmod{91} = 65 \]
\[ x_2 = 19 \times 65 + 30 \pmod{91} = 82 \]
\[ x_3 = 19 \times 82 + 30 \pmod{91} = 41 \]
\[ x_4 = 19 \times 41 + 30 \pmod{91} = 81 \]
\[ x_5 = 19 \times 81 + 30 \pmod{91} = 22 \]
\[ x_6 = 19 \times 22 + 30 \pmod{91} = 84 \]
\[ x_7 = 19 \times 84 + 30 \pmod{91} = 79 \]
\[ x_8 = 19 \times 79 + 30 \pmod{91} = 75 \]

Does this sequence look random?
Hard to say.
Example of Linear Cong. Gen

\[ x_0 = 21, \ A = 19, \ B = 30, \ M = 91 \]
\[ x_0 = 21 \]
\[ x_1 = 19 \times 21 + 30 \pmod{91} = 65 \]
\[ x_2 = 19 \times 65 + 30 \pmod{91} = 82 \]
\[ x_3 = 19 \times 82 + 30 \pmod{91} = 41 \]
\[ x_4 = 19 \times 41 + 30 \pmod{91} = 81 \]
\[ x_5 = 19 \times 81 + 30 \pmod{91} = 22 \]
\[ x_6 = 19 \times 22 + 30 \pmod{91} = 84 \]
\[ x_7 = 19 \times 84 + 30 \pmod{91} = 79 \]
\[ x_8 = 19 \times 79 + 30 \pmod{91} = 75 \]
Does this sequence look random?
Example of Linear Cong. Gen

\[ x_0 = 21, \ A = 19, \ B = 30, \ M = 91 \]
\[ x_0 = 21 \]
\[ x_1 = 19 \cdot 21 + 30 \pmod{91} = 65 \]
\[ x_2 = 19 \cdot 65 + 30 \pmod{91} = 82 \]
\[ x_3 = 19 \cdot 82 + 30 \pmod{91} = 41 \]
\[ x_4 = 19 \cdot 41 + 30 \pmod{91} = 81 \]
\[ x_5 = 19 \cdot 81 + 30 \pmod{91} = 22 \]
\[ x_6 = 19 \cdot 22 + 30 \pmod{91} = 84 \]
\[ x_7 = 19 \cdot 84 + 30 \pmod{91} = 79 \]
\[ x_8 = 19 \cdot 79 + 30 \pmod{91} = 75 \]

Does this sequence look random? Hard to say.
Our Running Example

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]

\[ x_0 = 2134 \text{ view as } 21, 34 \]
\[ x_{n+1} = 4381x_n + 7364 \pmod{8397} \]
Our Running Example

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]

\[
\begin{align*}
  x_0 &= 2134 \text{ view as 21, 34} \\
  x_{n+1} &= 4381x_n + 7364 \pmod{8397}
\end{align*}
\]

We use this to gen rand-looking bits, so 1-time-pad with psuedo-random bits.
Our Running Example

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]

\[ x_0 \ = 2134 \text{ view as } 21, 34 \]
\[ x_{n+1} \ = 4381x_n + 7364 \pmod{8397} \]

We use this to gen rand-looking bits, so 1-time-pad with psuedo-random bits.

We will then crack it.
Our Running Example

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]

\[
\begin{align*}
x_0 &= 2134 \text{ view as } 21, 34 \\
x_{n+1} &= 4381x_n + 7364 \pmod{8397}
\end{align*}
\]

We use this to gen rand-looking bits, so 1-time-pad with psuedo-random bits.

We will then crack it.

We will assume Eve knows that the random numbers are gen by a recurrence of the form

\[ x_{i+1} = Ax_i + B \pmod{M} \]

but that Eve do not know \( x_0, A, B, M. \) Does know \( A, B \) rel prime.
Psuedo One-Time Pad

\[ A = 01, \ B = 02, \cdots \ Z = 26 \ (\textbf{Not our usual since } A = 01. ) \]

View each letter as a two-digit number mod 26.
Psuedo One-Time Pad

$A = 01, B = 02, \cdots Z = 26$ (\textbf{Not our usual since $A = 01$.})

View each letter as a two-digit number mod 26.
Want a LONG sequence of 2-digit numbers $k_1, k_2, \ldots$
Psuedo One-Time Pad

$A = 01, \quad B = 02, \quad \cdots \quad Z = 26 \ (\text{Not our usual since } A = 01.)$

View each letter as a two-digit number mod 26.

Want a LONG sequence of 2-digit numbers $k_1, k_2, \ldots$

1. Will code $m_1, m_2, \ldots$ by, by adding mod 10 to each digit

   **Example** If key is 12 38 and message is 29 23 then send

   $\begin{array}{c}
   12 \\
   38 \\
   29 \\
   23 \\
   \hline
   31 \\
   51 \\
   \end{array}$

   So send 31 51 (these do not correspond to letters, thats fine).
Psuedo One-Time Pad

\[ A = 01, \quad B = 02, \quad \cdots \quad Z = 26 \quad (\text{Not our usual since } A = 01. \quad ) \]

View each letter as a two-digit number mod 26.
Want a LONG sequence of 2-digit numbers \( k_1, k_2, \ldots \)

1. Will code \( m_1, m_2, \ldots \) by, by adding mod 10 to each digit

   **Example**  If key is 12 38 and message is 29 23 then send

   \[
   \begin{array}{cc}
   12 & 38 \\
   29 & 23 \\
   \hline
   31 & 51
   \end{array}
   \]

   So send 31 51 (these do not correspond to letters, thats fine).

2. View as One-time pad with psuedo-random sequence.
Psuedo One-Time Pad

\[ A = 01, \, B = 02, \, \cdots \, Z = 26 \] (Not our usual since \( A = 01 \).)

View each letter as a two-digit number mod 26.

Want a LONG sequence of 2-digit numbers \( k_1, k_2, \ldots \)

1. Will code \( m_1, m_2, \ldots \) by, by adding mod 10 to each digit

   **Example**  If key is 12 38 and message is 29 23 then send

   \[
   \begin{array}{cccc}
   12 & 38 \\
   29 & 23 \\
   \hline
   31 & 51 
   \end{array}
   \]

   So send 31 51 (these do not correspond to letters, thats fine).

2. View as One-time pad with psuedo-random sequence.

How to get a long random (looking?) sequence? Next slide.
Use Rec. $x_0, A, B, M$ is Short Private Key

Example from **Cracking a Random Number Generator** by James Reed. Paper on Course Website.)
Example from **Cracking a Random Number Generator** by James Reed. Paper on Course Website.)

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]
Use Rec. $x_0, A, B, M$ is Short Private Key

Example from *Cracking a Random Number Generator* by James Reed. Paper on Course Website.)

\[ x_0 = 2134, \ A = 4381, \ B = 7364, \ M = 8397. \]

\[
\begin{align*}
  x_0 &= 2134 \text{ view as } 21, 34 \\
  x_{n+1} &= 4381x_n + 7364 \pmod{8397}
\end{align*}
\]
Use Rec. \(x_0, A, B, M\) is Short Private Key

Example from *Cracking a Random Number Generator* by James Reed. Paper on Course Website.)

\[ x_0 = 2134, \; A = 4381, \; B = 7364, \; M = 8397. \]

\[ x_0 = 2134 \text{ view as } 21, 34 \]
\[ x_{n+1} = 4381x_n + 7364 \pmod{8397} \]

We show that this random-looking sequence is NOT that random and, if used for a psuedo-one-time-pad, can be cracked.
Example 1

They start with $x_1$. If the document began with the word secret then encode by adding columns base 10:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>S</th>
<th>E</th>
<th>C</th>
<th>R</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>19</td>
<td>05</td>
<td>03</td>
<td>18</td>
<td>05</td>
<td>20</td>
</tr>
<tr>
<td>Key-Digits</td>
<td>21</td>
<td>60</td>
<td>69</td>
<td>05</td>
<td>37</td>
<td>78</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>30</td>
<td>65</td>
<td>62</td>
<td>13</td>
<td>32</td>
<td>98</td>
</tr>
</tbody>
</table>

Note E is coded as 65 and then later as 32. Recall that the whole point of OTP is that a letter won't always be coded the same way.
Example 1

\[ x_0 = 2134 \]
\[ x_1 = 2160 \]
\[ x_2 = 6905 \]
\[ x_3 = 3778 \]

They start with \( x_1 \).
Example 1

\[x_0 = 2134\]
\[x_1 = 2160\]
\[x_2 = 6905\]
\[x_3 = 3778\]

They start with \(x_1\).

If the document began with the word *secret* then encode by adding columns base 10:
Example 1

$x_0 = 2134$
$x_1 = 2160$
$x_2 = 6905$
$x_3 = 3778$

They start with $x_1$.
If the document began with the word secret then encode by adding columns base 10:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>S</th>
<th>E</th>
<th>C</th>
<th>R</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>19</td>
<td>05</td>
<td>03</td>
<td>18</td>
<td>05</td>
<td>20</td>
</tr>
<tr>
<td>Key-Digits</td>
<td>21</td>
<td>60</td>
<td>69</td>
<td>05</td>
<td>37</td>
<td>78</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>30</td>
<td>65</td>
<td>62</td>
<td>13</td>
<td>32</td>
<td>98</td>
</tr>
</tbody>
</table>

Note  E is coded as 65 and then later as 32. Recall that the whole point of OTP is that a letter won’t always be coded the same way.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time. Eve knows rec of form $x_{n+1} = Ax_n + B \pmod{M}$.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time. Eve knows rec of form $x_{n+1} = Ax_n + B \pmod{M}$. Eve knows that $A, B, M$ are all 4-digits. If she fails she may try again with 6-digits.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time. Eve knows rec of form $x_{n+1} = Ax_n + B \pmod{M}$. Eve knows that $A, B, M$ are all 4-digits. If she fails she may try again with 6-digits. Eve knows that the document is about India and Pakistan.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time. Eve knows rec of form $x_{n+1} = Ax_n + B \pmod{M}$.
Eve knows that $A, B, M$ are all 4-digits. If she fails she may try again with 6-digits.
Eve knows that the document is about India and Pakistan.
Eve thinks Pakistan will be in the document.
Eve thinks $M$ is 4-digits.
Example 2

Alice sends Bob a document using the $x_i$ as a two chars at a time.

Eve knows rec of form $x_{n+1} = A x_n + B \pmod{M}$.

Eve knows that $A, B, M$ are all 4-digits. If she fails she may try again with 6-digits.

Eve knows that the document is about **India** and **Pakistan**.

Eve thinks **Pakistan** will be in the document.

Eve thinks $M$ is 4-digits.

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
</tr>
</tbody>
</table>
Eve sees

| Ciphertext | 24 66 87 47 17 45 26 96 |

And thinks it is PAKISTAN.
Eve sees

| Ciphertext | 24 66 87 47 17 45 26 96 |

And thinks it is PAKISTAN.

So Eve thinks the following:
Eve sees

| Ciphertext | 24  66  87  47  17  45  26  96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

| Text-Letter | P   | A   | K   | I   | S   | T   | A |
| Text-Digits | 16  | 01  | 11  | 09  | 19  | 20  | 01 |
| Key-Digits  | $k_{11}k_{12}$ | $k_{21}k_{22}$ | $k_{31}k_{32}$ | $k_{41}k_{42}$ | $k_{51}k_{52}$ | $k_{61}k_{62}$ | $k_{71}k_{72}$ |
| Ciphertext | 24  | 66  | 87  | 47  | 17  | 45  | 26 |

Can Eve find the Key-Digits?

Yes!

All $\equiv$ mod 10.

1 + $k_{11}$ $\equiv$ 2 so $k_{11}$ $\equiv$ 2 - 1 $\equiv$ 1.

6 + $k_{12}$ $\equiv$ 4 so $k_{12}$ $\equiv$ 4 - 6 $\equiv$ -2 $\equiv$ 8.

Etc.

Next slide gives complete answer.
Eve sees

| Ciphertext | 24  | 66  | 87  | 47  | 17  | 45  | 26  | 96  |

And thinks it is PAKISTAN.

So Eve thinks the following:

| Text-Letter |  P  | A  | K  | I  | S  | T  | A  |
| Text-Digits | 16  | 01 | 11 | 09 | 19 | 20 | 01 |
| Key-Digits  | $k_{11}k_{12}$ | $k_{21k_{22}}$ | $k_{31}k_{32}$ | $k_{41}k_{42}$ | $k_{51}k_{52}$ | $k_{61}k_{62}$ | $k_{71}k_{72}$ |
| Ciphertext  | 24  | 66 | 87 | 47 | 17 | 45 | 26 |

Can Eve find the Key-Digits?
Eve sees

| Ciphertext | 24  66  87  47  17  45  26  96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

| Text-Letter | P   | A   | K   | I   | S   | T   | A   |
| Text-Digits | 16  | 01  | 11  | 09  | 19  | 20  | 01  |
| Key-Digits  | $k_{11}$ $k_{12}$ $k_{21}$ $k_{22}$ $k_{31}$ $k_{32}$ $k_{41}$ $k_{42}$ $k_{51}$ $k_{52}$ $k_{61}$ $k_{62}$ $k_{71}$ $k_{72}$ |
| Ciphertext | 24  | 66  | 87  | 47  | 17  | 45  | 26  |

Can Eve find the Key-Digits? Yes!
Eve sees

| Ciphertext | 24  66  87  47  17  45  26  96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

| Text-Letter | P   | A   | K   | I   | S   | T   | A   |
| Text-Digits | 16  | 01  | 11  | 09  | 19  | 20  | 01  |
| Key-Digits  | $k_{11}k_{12}$ | $k_{21}k_{22}$ | $k_{31}k_{32}$ | $k_{41}k_{42}$ | $k_{51}k_{52}$ | $k_{61}k_{62}$ | $k_{71}k_{72}$ |
| Ciphertext | 24  | 66  | 87  | 47  | 17  | 45  | 26  |

Can Eve find the Key-Digits? Yes! all $\equiv$ are mod 10.
Though Experiment

Eve sees

| Ciphertext | 24 66 87 47 17 45 26 96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P  A  K  I  S  T  A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16 01 11 09 19 20 01</td>
</tr>
<tr>
<td>Key-Digits</td>
<td>$k_{11} k_{12}$ $k_{21} k_{22}$ $k_{31} k_{32}$ $k_{41} k_{42}$ $k_{51} k_{52}$ $k_{61} k_{62}$ $k_{71} k_{72}$</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>24 66 87 47 17 45 26</td>
</tr>
</tbody>
</table>

Can Eve find the Key-Digits? Yes! all $\equiv$ are mod 10.

$1 + k_{11} \equiv 2$ so $k_{11} \equiv 2 - 1 \equiv 1.$
Eve sees

| Ciphertext | 24 66 87 47 17 45 26 96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
</tr>
<tr>
<td>Key-Digits</td>
<td>$k_{11}k_{12}$</td>
<td>$k_{21}k_{22}$</td>
<td>$k_{31}k_{32}$</td>
<td>$k_{41}k_{42}$</td>
<td>$k_{51}k_{52}$</td>
<td>$k_{61}k_{62}$</td>
<td>$k_{71}k_{72}$</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
</tr>
</tbody>
</table>

Can Eve find the Key-Digits? Yes! all $\equiv$ are mod 10.

$1 + k_{11} \equiv 2$ so $k_{11} \equiv 2 - 1 \equiv 1$.

$6 + k_{12} \equiv 4$ so $k_{12} \equiv 4 - 6 \equiv -2 \equiv 8$.

Etc.
Though Experiment

Eve sees

| Ciphertext | 24  66  87  47  17  45  26  96 |

And thinks it is PAKISTAN.

So Eve thinks the following:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
</tr>
<tr>
<td>Key-Digits</td>
<td>$k_{11}k_{12}$</td>
<td>$k_{21}k_{22}$</td>
<td>$k_{31}k_{32}$</td>
<td>$k_{41}k_{42}$</td>
<td>$k_{51}k_{52}$</td>
<td>$k_{61}k_{62}$</td>
<td>$k_{71}k_{72}$</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
</tr>
</tbody>
</table>

Can Eve find the Key-Digits? Yes! all $\equiv$ are mod 10.

$1 + k_{11} \equiv 2$ so $k_{11} \equiv 2 - 1 \equiv 1$.

$6 + k_{12} \equiv 4$ so $k_{12} \equiv 4 - 6 \equiv -2 \equiv 8$.

Etc.

Next slide gives complete answer.
Eve Thinks:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
<td>96</td>
</tr>
</tbody>
</table>
Through Experiment Continued

Eve Thinks:

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
<td>96</td>
</tr>
</tbody>
</table>

If Eve is correct then:

| Key–Digits | 18| 65| 76| 48| 08| 25| 25| 82|
Through Experiment Continued: Eve gets Equations

If Eve is correct then:

<table>
<thead>
<tr>
<th>Key–Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since

\[ x^n + 1 \equiv A x^n + B \pmod{M} \]

\[ 7648 \equiv 1865 A + B \pmod{M} \]

\[ 825 \equiv 7648 A + B \pmod{M} \]

\[ 2582 \equiv 825 A + B \pmod{M} \]

Can we solve these?

Yes!
If Eve is correct then:

<table>
<thead>
<tr>
<th>Key–Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since $x_{n+1} \equiv Ax_n + B \pmod{M}$
If Eve is correct then:

| Key-Digits | 18 | 65 | 76 | 48 | 08 | 25 | 25 | 82 |

Since $x_{n+1} \equiv Ax_n + B \pmod{M}$

$7648 \equiv 1865A + B \pmod{M}$
If Eve is correct then:

<table>
<thead>
<tr>
<th>Key-Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since \( x_{n+1} \equiv A x_n + B \pmod{M} \)

7648 \( \equiv 1865A + B \pmod{M} \)

825 \( \equiv 7648A + B \pmod{M} \)
If Eve is correct then:

<table>
<thead>
<tr>
<th>Key–Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since $x_{n+1} \equiv Ax_n + B \pmod{M}$

$7648 \equiv 1865A + B \pmod{M}$

$825 \equiv 7648A + B \pmod{M}$

$2582 \equiv 825A + B \pmod{M}$
If Eve is correct then:

<table>
<thead>
<tr>
<th>Key–Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since \( x_{n+1} \equiv Ax_n + B \pmod{M} \)

\[
7648 \equiv 1865A + B \pmod{M}
\]

\[
825 \equiv 7648A + B \pmod{M}
\]

\[
2582 \equiv 825A + B \pmod{M}
\]

Can we solve these?
If Eve is correct then:

<table>
<thead>
<tr>
<th>Key–Digits</th>
<th>18</th>
<th>65</th>
<th>76</th>
<th>48</th>
<th>08</th>
<th>25</th>
<th>25</th>
<th>82</th>
</tr>
</thead>
</table>

Since $x_{n+1} \equiv A x_n + B \pmod{M}$

$7648 \equiv 1865A + B \pmod{M}$

$825 \equiv 7648A + B \pmod{M}$

$2582 \equiv 825A + B \pmod{M}$

Can we solve these? Yes!
Thought Exp: Eve Can Finding $M$ (I)

EQ1: $7648 \equiv 1865A + B \pmod{M}$
EQ2: $825 \equiv 7648A + B \pmod{M}$
EQ3: $2582 \equiv 825A + B \pmod{M}$
Thought Exp: Eve Can Finding $M$ (I)

EQ1: $7648 \equiv 1865A + B \pmod{M}$
EQ2: $825 \equiv 7648A + B \pmod{M}$
EQ3: $2582 \equiv 825A + B \pmod{M}$

By looking at EQ2–EQ1 and EQ3–EQ1 get 2 equations and no $B$
Thought Exp: Eve Can Finding $M$ (I)

EQ1: $7648 \equiv 1865A + B \pmod{M}$
EQ2: $825 \equiv 7648A + B \pmod{M}$
EQ3: $2582 \equiv 825A + B \pmod{M}$

By looking at EQ2−EQ1 and EQ3−EQ1 get 2 equations and no $B$

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$
Thought Exp: Eve can Find $M$ (II)

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$
Thought Exp: Eve can Find $M$ (II)

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$

Mult EQ4 by 1040 and EQ5 by 5783 to get:

EQ4': $-6823 \times 1040 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5': $-5066 \times 5783 \equiv -1040 \times 5783 \times A \pmod{M}$
Thought Exp: Eve can Find $M$ (II)

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$

Mult EQ4 by 1040 and EQ5 by 5783 to get:

EQ4': $-6823 \times 1040 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5': $-5066 \times 5783 \equiv -1040 \times 5783 \times A \pmod{M}$

We rewrite a bit:
Thought Exp: Eve can Find $M$ (II)

EQ4: $-6823 \equiv 5783A \, (\text{mod } M)$
EQ5: $-5066 \equiv -1040A \, (\text{mod } M)$

Mult EQ4 by 1040 and EQ5 by 5783 to get:

EQ4': $-6823 \times 1040 \equiv 5783 \times 1040 \times A \, (\text{mod } M)$
EQ5': $-5066 \times 5783 \equiv -1040 \times 5783 \times A \, (\text{mod } M)$

We rewrite a bit:

EQ4': $-7095920 \equiv 5783 \times 1040 \times A \, (\text{mod } M)$
EQ5': $-29296678 \equiv -5783 \times 1040 \times A \, (\text{mod } M)$

Can we use this?
Yes We Can!
Thought Exp: Eve can Find $M$ (II)

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$

Mult EQ4 by 1040 and EQ5 by 5783 to get:

EQ4’: $-6823 \times 1040 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5’: $-5066 \times 5783 \equiv -1040 \times 5783 \times A \pmod{M}$

We rewrite a bit:

EQ4’: $-7095920 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5’: $-29296678 \equiv -5783 \times 1040 \times A \pmod{M}$

Add EQ4’ and EQ5’ to get:

$$-36392598 \equiv 0 \pmod{M}$$

Can we use this?
**Thought Exp: Eve can Find $M$ (II)**

EQ4: $-6823 \equiv 5783A \pmod{M}$
EQ5: $-5066 \equiv -1040A \pmod{M}$

Mult EQ4 by 1040 and EQ5 by 5783 to get:

EQ4': $-6823 \times 1040 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5': $-5066 \times 5783 \equiv -1040 \times 5783 \times A \pmod{M}$

We rewrite a bit:

EQ4': $-7095920 \equiv 5783 \times 1040 \times A \pmod{M}$
EQ5': $-29296678 \equiv -5783 \times 1040 \times A \pmod{M}$

Add EQ4' and EQ5' to get:

$$-36392598 \equiv 0 \pmod{M}$$

Can we use this? Yes We Can!
Thought Exp: Eve Finds $M$ (III)

\[36392598 \equiv 0 \pmod{M}\]
Thought Exp: Eve Finds $M$ (III)

\[ 36392598 \equiv 0 \pmod{M} \]

1. $M$ divides 36392598.
Thought Exp: Eve Finds $M$ (III)

\[ 36392598 \equiv 0 \pmod{M} \]

1. $M$ divides 36392598.
2. $M$ is 4 digits long.

Hence a SMALL number of possibilities for $M$.

Two ways to find possibilities for $M$ on next few slides.
Thought Exp: Eve Finds $M$ (III)

\[ 36392598 \equiv 0 \pmod{M} \]

1. $M$ divides 36392598.
2. $M$ is 4 digits long.
3. The cipher used 7648, so $M > 7648$, hence $7649 \leq M \leq 9999$.

Hence a SMALL number of possibilities for $M$. 
Thought Exp: Eve Finds $M$ (III)

$$36392598 \equiv 0 \pmod{M}$$

1. $M$ divides 36392598.
2. $M$ is 4 digits long.
3. The cipher used 7648, so $M > 7648$, hence $7649 \leq M \leq 9999$.

Hence a SMALL number of possibilities for $M$.
Two ways to find possibilities for $M$ on next few slides.
Eve Factors to Find $M$

Eve factors 36392598.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$
Eve Factors to Find \( M \)

Eve factors 36392598.

\[
36392598 = 2 \times 3^3 \times 11 \times 197 \times 311
\]

Factoring? Really? Eve has to Factor?
Eve Factors to Find $M$

Eve factors 36392598.

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

Factoring? Really? Eve has to Factor? 

(Sarcastic) does she have a quantum computer?
Eve Factors to Find $M$

Eve factors 36392598.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

Factoring? Really? Eve has to Factor? (Sarcastic) does she have a quantum computer?

We will address this point later.
Eve factors 36392598.

\[ 36392598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

Factoring? Really? Eve has to Factor?  
(Sarcastic) does she have a quantum computer?  
We will address this point later.

1. \( M \) is a divisor of 36392598.
Eve factors 36392598.

\[ 36392598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

Factoring? Really? Eve has to Factor? *(Sarcastic)* does she have a quantum computer?

We will address this point later.

1. \( M \) is a divisor of 36392598.
2. \( M \) is 4 digits long.
Eve factors 36392598.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

Factoring? Really? Eve has to Factor? (Sarcastic) does she have a quantum computer?

We will address this point later.

1. $M$ is a divisor of 36392598.
2. $M$ is 4 digits long.
3. The cipher used 7648, so $M > 7648$. 
Eve Can Crack It!—Finding $M$

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?
36392598 = 2 \times 3^3 \times 11 \times 197 \times 311

M is a factor of 36392598 such that 7648 \leq M \leq 9999.

How many factors of 2 \times 3^3 \times 11 \times 197 \times 311?

2 \times 4 \times 2 \times 2 \times 2 = 64.
Eve Can Crack It!–Finding $M$

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of $36392598$ such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?

$2 \times 4 \times 2 \times 2 \times 2 = 64$.

1. Can’t use 197 AND 311: $197 \times 311 = 61267 > 9999$. 

The original article did do it by hand. It was written in 1977. The next slide shows how to do it by hand. We won’t go over it, but you can if you want.
36392598 = 2 \times 3^3 \times 11 \times 197 \times 311

M is a factor of 36392598 such that 7648 \leq M \leq 9999.

How many factors of 2 \times 3^3 \times 11 \times 197 \times 311?

2 \times 4 \times 2 \times 2 \times 2 = 64.

1. Can’t use 197 AND 311: 197 \times 311 = 61267 > 9999.

2. Could continue to do this by hand.
Eve Can Crack It!—Finding $M$

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?

$2 \times 4 \times 2 \times 2 \times 2 = 64$.

1. Can’t use 197 AND 311: $197 \times 311 = 61267 > 9999$.
2. Could continue to do this by hand.

   We won’t—we are busy people and we have computers to do it for us.
Eve Can Crack It!—Finding $M$

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?

$2 \times 4 \times 2 \times 2 \times 2 = 64$.

1. Can’t use 197 AND 311: $197 \times 311 = 61267 > 9999$.
2. Could continue to do this by hand.

We won’t—**we are busy people** and we have computers to do it for us.

The original article did do it by hand. It was written in 1977.
Eve Can Crack It!—Finding $M$

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?

$2 \times 4 \times 2 \times 2 \times 2 = 64$.

1. Can’t use 197 AND 311: $197 \times 311 = 61267 > 9999$.

2. Could continue to do this by hand.

We won’t—we are busy people and we have computers to do it for us.

The original article did do it by hand. It was written in 1977.

The next slide shows how to do it by hand. We won’t go over it, but you can if you want.
Eve Can Crack It!—Finding $M$ OLD WAY

**THIS SLIDE IS OPTIONAL**

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?
Eve Can Crack It!—Finding $M$ OLD WAY

THIS SLIDE IS OPTIONAL

36392598 = $2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

How many factors of \(2 \times 3^3 \times 11 \times 197 \times 311\)?

\[2 \times 4 \times 2 \times 2 \times 2 = 64.\]
Eve Can Crack It!—Finding $M$ OLD WAY

THIS SLIDE IS OPTIONAL

$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$

$M$ is a factor of $36392598$ such that $7648 \leq M \leq 9999$.

How many factors of $2 \times 3^3 \times 11 \times 197 \times 311$?

$2 \times 4 \times 2 \times 2 \times 2 = 64$.

1. Can’t use 197 AND 311: $197 \times 311 = 61267 > 9999$.
2. If use 311 then need a 3: $2 \times 11 \times 311 = 6842 < 7648$.
3. If use 311 and exactly one 3 does not work:
   (a) Use 2 but not 11: $311 \times 3 \times 2 = 1866 < 7648$
   (b) Use 11: $\geq 311 \times 3 \times 11 = 10263 > 9999$.
4. If use 311, at least two 3’s, and 11:
   $311 \times 11 \times 9 = 30789 > 9999$.
5. If use 311 and 9 does not work: $311 \times 2 \times 9 = 5598 < 7648$.
6. If use 311 and 27: $311 \times 27 = 8397$. WORKS!
7. Leave it to you to show that using 197 does not work.
8. So $M = 8397$. 
How to do it in 2021

Recall

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$. 
Recall

$M$ is a factor of $36392598$ such that $7648 \leq M \leq 9999$.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$
Recall

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

36392598 has $2 \times 4 \times 2 \times 2 \times 2 = 64$ factors.

Two ways to find possibilities for $M$

1. Look at all 64 factors and see which ones are in $[7648, 9999]$. 

How to do it in 2021

Recall

$M$ is a factor of 36392598 such that $7648 \leq M \leq 9999$.

$$36392598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

36392598 has $2 \times 4 \times 2 \times 2 \times 2 = 64$ factors.

Two ways to find possibilities for $M$

1. Look at all 64 factors and see which ones are in $[7648, 9999]$.

2. Even less clever: Look at ALL numbers in $[7648, 9999]$ and see which ones are factors of $M$. 
Reflect

If we do this we find that the only candidate that works is $M = 8397$. 
Reflect

If we do this we find that the only candidate that works is $M = 8397$.

We might have found no $M$ works. So Eve was wrong.
Reflect

If we do this we find that the only candidate that works is \( M = 8397 \).

We might have found no \( M \) works. So Eve was wrong.

We might have found several \( M \) works. In that case, do what is on the next few slides with each one.
Eve Determines Which $M$ Is Correct, If Any

EQ4: $-6823 \equiv 5783A \pmod{M}$

By either brute force of cleverness we found that

If Even’s Guess Is Correct then $M = 8397$.

EQ4: $-6823 \equiv 5783A \pmod{8397}$

Use Euclid algorithm to find that $5783 - 1 \equiv 1982 \pmod{8397}$.

Reflect: It is possible the inverse does not exist. Then Eve is wrong. In the case at hand, the inverse exists.

Multiply both sides of EQ4 by 1982 to get:

$-6823 \times 1982 \equiv A \pmod{8397}$

$A \equiv -6823 \times 1982 \equiv 4381 \pmod{8397}$
Eve Determines Which $M$ Is Correct, If Any

EQ4: $-6823 \equiv 5783A \pmod{M}$
By either brute force of cleverness we found that
If Even’s Guess Is Correct then $M \equiv 8397$.

EQ4: $-6823 \equiv 5783A \pmod{8397}$
Use Euclid algorithm to find that $5783^{-1} \pmod{8397} \equiv 1982$. 
Eve Determines Which $M$ Is Correct, If Any

EQ4: $-6823 \equiv 5783A \pmod{M}$
By either brute force of cleverness we found that
If Even’s Guess Is Correct then $M = 8397$.

EQ4: $-6823 \equiv 5783A \pmod{8397}$
Use Euclid algorithm to find that $5783^{-1} \pmod{8397} \equiv 1982$.
Reflect It is possible the inverse does not exist. Then Eve is wrong. In the case at hand, the inverse exists.
Eve Determines Which $M$ Is Correct, If Any

EQ4: $-6823 \equiv 5783A \pmod{M}$
By either brute force of cleverness we found that
If Even’s Guess Is Correct then $M = 8397$.

EQ4: $-6823 \equiv 5783A \pmod{8397}$
Use Euclid algorithm to find that $5783^{-1} \pmod{8397} \equiv 1982$.
Reflect It is possible the inverse does not exist. Then Eve is wrong. In the case at hand, the inverse exists.
Multiply both sides of EQ4 by 1982 to get:

$$-6823 \times 1982 \equiv A \pmod{8397}$$

$$A \equiv -6823 \times 1982 \equiv 4381 \pmod{8397}$$
Now want to find $B$. Recall:

$$
\text{EQ1: } 7648 \equiv 1865 \mod M \\
\text{By plugging in } M = 8397 \text{ and } A = 4381 \text{ we get } \\
7648 \equiv 1865 \cdot 4381 + B \mod 8397 \\
B \equiv 7648 - 1865 \cdot 4381 \equiv 7364 \mod 8397
$$
Now want to find $B$. Recall:

EQ1: $7648 \equiv 1865A + B \ (mod \ M)$
Eve Checks $M$

Now want to find $B$. Recall:

EQ1: $7648 \equiv 1865A + B \pmod{M}$

By plugging in $M = 8397$ and $A = 4381$ we get

$$7648 \equiv 1865 \times 4381 + B \pmod{8397}$$
Eve Checks \( M \)

Now want to find \( B \). Recall:

EQ1: \( 7648 \equiv 1865A + B \pmod{M} \)

By plugging in \( M = 8397 \) and \( A = 4381 \) we get

\[
7648 \equiv 1865 \times 4381 + B \pmod{8397}
\]

\[
B \equiv 7648 - 1865 \times 4381 \equiv 7364 \pmod{8397}
\]
Eve Checks \( M \)

Now want to find \( B \). Recall:

\[
\text{EQ1: } 7648 \equiv 1865A + B \pmod{M}
\]

By plugging in \( M = 8397 \) and \( A = 4381 \) we get

\[
7648 \equiv 1865 \times 4381 + B \pmod{8397}
\]

\[
B \equiv 7648 - 1865 \times 4381 \equiv 7364 \pmod{8397}
\]

**Upshot**  If Eve’s Guess Is Correct Then  \( A = 4381, B = 7364, M = 8397 \).
Eve Can Find $x_0$

Eve wants to test $A = 4381, B = 7634, M = 8307$. 
Eve Can Find $x_0$

Eve wants to test $A = 4381$, $B = 7634$, $M = 8307$.

$$x_{n+1} \equiv 4381x_n + 7364 \pmod{8397}$$
Eve Can Find $x_0$

Eve wants to test $A = 4381, B = 7634, M = 8307$.

$$x_{n+1} \equiv 4381x_n + 7364 \pmod{8397}$$

Need $x_0$. 

4381 is rel prime to 8397 so $(4381)^{-1}$ exists. 

It is 8374. Mult equation by 8374.

$$8374x_n + 1 \equiv 8374 \times 4381x_n + 8374 \times 7364 \pmod{8397}$$

$$8374x_n + 1 \equiv x_n + 6965 \pmod{8397}$$

$$x_n \equiv 8374x_n + 1 - 6965 \equiv 8374x_n + 1 + 1432$$
Eve Can Find $x_0$

Eve wants to test $A = 4381, B = 7634, M = 8307$.

$$x_{n+1} \equiv 4381x_n + 7364 \pmod{8397}$$

Need $x_0$.

4381 is rel prime to 8397 so $(4381)^{-1} \pmod{8397}$ exists.
It is 8374. Mult equation by 8374.
Eve wants to test $A = 4381, B = 7634, M = 8307$.

\[ x_{n+1} \equiv 4381x_n + 7364 \pmod{8397} \]

Need $x_0$.

4381 is rel prime to 8397 so $(4381)^{-1} \pmod{8397}$ exists. It is 8374. Mult equation by 8374.

\[ 8374x_{n+1} \equiv 8374 \times 4381x_n + 8374 \times 7364 \pmod{8397} \]
Eve Can Find $x_0$

Eve wants to test $A = 4381, B = 7634, M = 8307$.

$$x_{n+1} \equiv 4381x_n + 7364 \pmod{8397}$$

Need $x_0$.

4381 is rel prime to 8397 so $(4381)^{-1} \pmod{8397}$ exists.

It is 8374. Mult equation by 8374.

$$8374x_{n+1} \equiv 8374 \cdot 4381x_n + 8374 \cdot 7364 \pmod{8397}$$

$$8374x_{n+1} \equiv x_n + 6965 \pmod{8397}$$
Eve Can Find $x_0$

Eve wants to test $A = 4381, B = 7634, M = 8307$.

$$x_{n+1} \equiv 4381x_n + 7364 \pmod{8397}$$

Need $x_0$.

4381 is rel prime to 8397 so $(4381)^{-1} \pmod{8397}$ exists.

It is 8374. Mult equation by 8374.

$$8374x_{n+1} \equiv 8374 \times 4381x_n + 8374 \times 7364 \pmod{8397}$$

$$8374x_{n+1} \equiv x_n + 6965 \pmod{8397}$$

$$x_n \equiv 8374x_{n+1} - 6965 \equiv 8374x_{n+1} + 1432$$

How will this help us?
Eve Finds $x_0$ (cont)

\[ x_n \equiv 8374x_{n+1} + 1432 \]
Eve Finds $x_0$ (cont)

\[ x_n \equiv 8374x_{n+1} + 1432 \]

PAKISTAN had the $P$ on the (say) 191st spot. We know the key at 191 spot. Hence can use recurrence above to get key at 190th, 189th, \ldots, 0th spot.
Eve Finds $x_0$ (cont)

$$x_n \equiv 8374x_{n+1} + 1432$$

PAKISTAN had the $P$ on the (say) 191st spot. We know the key at 191 spot. Hence can use recurrence above to get key at 190th, 189th, \ldots, 0th spot.

So can get $x_0$. 
Eve Finds $x_0$ (cont)

$x_n \equiv 8374x_{n+1} + 1432$

PAKISTAN had the $P$ on the (say) 191st spot. We know the key at 191 spot. Hence can use recurrence above to get key at 190th, 189th, ..., 0th spot.

So can get $x_0$.

Are we done yet? No.
Eve Uses Is-English

Eve has $x_0$, $A$, $B$, $M$ so Eve can generate the entire key.
Eve Uses Is-English

Eve has $x_0, A, B, M$ so Eve can generate the **entire** key. She uses it to recover the **entire** plaintext.
Eve Uses Is-English

Eve has $x_0, A, B, M$ so Eve can generate the *entire* key. She uses it to recover the *entire* plaintext. Use IS-ENGLISH.
Eve Uses Is-English

Eve has $x_0, A, B, M$ so Eve can generate the entire key. She uses it to recover the entire plaintext. Use IS-ENGLISH.

If Eve’s Guess Is Correct then it will return YES-IS ENGLISH. So Eve is done!
Eve Uses Is-English

Eve has $x_0, A, B, M$ so Eve can generate the entire key. She uses it to recover the entire plaintext. Use IS-ENGLISH.

If Eve’s Guess Is Correct then it will return YES-IS ENGLISH. So Eve is done!

If Eve’s Guess Is Not Correct then either the procedure would have failed long before this point OR we find ISNOT-English.
But This Was All Predicated on Eve’s Guess

We just showed that IF Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.
2. If the guess is correct then she can find $A$, $B$, $M$, $x_0$ and decode the message.

How can Eve use this to break the cipher? For every 8-letter sequence Eve guesses that it is PAKISTAN and does out the procedure above. Most of the time she will be wrong. But the one time she is right, she will decode the message.
But This Was All Predicated on Eve’s Guess

We just showed that **IF** Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.

How can Eve use this to break the cipher?

For every 8-letter sequence Eve guesses that it is PAKISTAN and does out the procedure above. Most of the time she will be wrong. But the one time she is right, she will decoded the message.
But This Was All Predicated on Eve’s Guess

We just showed that **IF** Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.
2. If the guess is correct then she can find $A, B, M, x_0$ and decode the message.
But This Was All Predicated on Eve’s Guess

We just showed that **IF** Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.
2. If the guess is correct then she can find $A, B, M, x_0$ and decode the message.

How can Eve use this to break the cipher?
But This Was All Predicated on Eve’s Guess

We just showed that **IF** Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.
2. If the guess is correct then she can find $A, B, M, x_0$ and decode the message

How can Eve use this to break the cipher? For **every** 8-letter sequence Eve **guess’s** that it is PAKISTAN and does out the procedure above.
But This Was All Predicated on Eve’s Guess

We just showed that **IF** Eve thinks that PAKISTAN occurred in (say) spaces 190 to 197 then:

1. She can test if the guess is correct.
2. If the guess is correct then she can find $A, B, M, x_0$ and decode the message.

How can Eve use this to break the cipher?

For **every** 8-letter sequence Eve **guesses** that it is PAKISTAN and does out the procedure above.

Most of the time she will be wrong. But the one time she is right, she will decoded the message.
Putting it All Together

1. Input is long ciphertext T that Eve knows was coded with recurrence. Eve knows a word w that she is sure appears in the text and is L letters.

2. For EVERY L-letter seq Eve does the following:
   2.1 Assuming L-letter seq is w form equations and try to solve them. If can't then goto next L-letter seq.
   2.2 Use A, B, M, x0 to generate entire key. Decode entire text. If IS-ENGLISH=YES, DONE! Else goto next L-letter seq.
1. Input is long ciphertext $T$ that Eve knows was coded with recurrence. Eve knows a word $w$ that she is sure appears in the text and is $L$ letters.
Putting it All Together

1. Input is long ciphertext $T$ that Eve knows was coded with recurrence. Eve knows a word $w$ that she is sure appears in the text and is $L$ letters.

2. For EVERY $L$-letter seq Eve does the following:

   2.1 Assuming $L$-letter seq is $w$ form equations and try to solve them. If can't then goto next $L$-letter seq.

   2.2 Use $A$, $B$, $M$, $x_0$ to generate entire key. Decode entire text. If IS-ENGLISH=YES, DONE! Else goto next $L$-let-seq.
Putting it All Together

1. Input is long ciphertext $T$ that Eve knows was coded with recurrence. Eve knows a word $w$ that she is sure appears in the text and is $L$ letters.

2. For EVERY $L$-letter seq Eve does the following:
   2.1 Assuming $L$-letter seq is $w$ form equations and try to solve them. If can’t then goto next $L$-letter seq.
Putting it All Together

1. Input is long ciphertext $T$ that Eve knows was coded with recurrence. Eve knows a word $w$ that she is sure appears in the text and is $L$ letters.
2. For EVERY $L$-letter seq Eve does the following:
   2.1 Assuming $L$-letter seq is $w$ form equations and try to solve them. If can’t then goto next $L$-letter seq.
   2.2 Use $A, B, M, x_0$ to generate entire key. Decode entire text. If IS-ENGLISH=YES, DONE! Else goto next $L$-let-seq.
Eve had to factor:

\[ 36, 392, 598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]
Eve had to factor:

\[ 36, 392, 598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

We usually say

Factoring is Hard
About Eve Factoring Fast

Eve had to factor:

\[ 36, 392, 598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

We usually say

**Factoring is Hard**

But what do we mean by **Factoring is Hard**?
Eve had to factor:

\[ 36, 392, 598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

We usually say

**Factoring is Hard**

But what do we mean by **Factoring is Hard**?

1. If *Alice* picks two primes \( p, q \) of length \( n \) and picks \( N = pq \) then factoring \( N \) is hard.
Eve had to factor:

\[ 36,392,598 = 2 \times 3^3 \times 11 \times 197 \times 311 \]

We usually say

**Factoring is Hard**

But what do we mean by **Factoring is Hard**?

1. If *Alice* picks two *primes* \( p, q \) of length \( n \) and picks \( N = pq \) then factoring \( N \) is hard.

2. If a **random** number is given then half the time it’s even. A third of the time is divided by 3. Not so hard to factor.
Eve had to factor:

$$36,392,598 = 2 \times 3^3 \times 11 \times 197 \times 311$$

We usually say

Factoring is Hard

But what do we mean by Factoring is Hard?

1. If Alice picks two primes $p, q$ of length $n$ and picks $N = pq$ then factoring $N$ is hard.

2. If a random number is given then half the time it’s even. A third of the time is divided by 3. Not so hard to factor.

Our scenario is closer to random than to Alice.
Some Real World Notes

1. Java and other langs use an LCG with some $A$, $B$, $M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern langs use the Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)

3. Why do Java and Python and other langs have such bad random number generators?
   3.1 They are bad for crypto.
   3.2 They are fine for randomized algorithms (like quicksort).
1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.
1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern language use The Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)
Some Real World Notes

1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern language use The Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)

3. Why do Java and Python and other langs have such bad random number generators?
Some Real World Notes

1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern language use The Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)

3. Why do Java and Python and other langs have such bad random number generators?
1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern language use The Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)

3. Why do Jave and Python and other langs have such bad random number generators?
   3.1 They are bad for crypto.
Some Real World Notes

1. Java and other langs use an LCG with some $A, B, M$. Actually the $M$ is always $2^{32}$ or $2^{64}$. This makes the LCG even easier to crack.

2. Python and other modern language use The Mersenne Twister to generate random numbers. It is also not secure. (I will discuss it very soon.)

3. Why do Java and Python and other langs have such bad random number generators?
   3.1 They are bad for crypto.
   3.2 They are fine for randomized algorithms (like quicksort).
Mersenne Twister

We do a very small example with a smaller word size than is used. The Mersenne Twister generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \{0, \ldots, 26\}).
Mersenne Twister

We do a very small example with a smaller word size than is used. The **Mersenne Twister** generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \{0, \ldots, 26\}).

We give an example:
Params: 7, 5, 3, 5, 3, \(x_0, \ldots, x_6\), unknown to Eve.
Mersenne Twister

We do a very small example with a smaller word size than is used. The **Mersenne Twister** generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \( \{0, \ldots, 26\} \)).

We give an example:

Params: \( 7, 5, 3, 5, 3, x_0, \ldots, x_6 \), unknown to Eve.

\[
x_{n+7} = x_{n+5} \oplus f(x_n^{\text{first 3 bits}}, x_{n+1}^{\text{last 5 bits}})
\]

\( f \) shifts bits 3 to the left (its more complicated).
Mersenne Twister

We do a very small example with a smaller word size than is used. The Mersenne Twister generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \{0, \ldots, 26\}).

We give an example:

Params: 7, 5, 3, 5, 3, \(x_0, \ldots, x_6\), unknown to Eve.

\[ x_{n+7} = x_{n+5} \oplus f(x_n^{\text{first 3 bits}}, x_n^{\text{last 5 bits}}) \]

\(f\) shifts bits 3 to the left (it's more complicated).

1. Very fast since \(\oplus\) and concat and shift are fast.
**Mersenne Twister**

We do a very small example with a smaller word size than is used. The **Mersenne Twister** generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \( \{0, \ldots, 26\} \)). We give an example:

**Params:** 7, 5, 3, 5, 3, \( x_0, \ldots, x_6 \), unknown to Eve.

\[
x_{n+7} = x_{n+5} \oplus f(x_n^{\text{first 3 bits}}, x_{n+1}^{\text{last 5 bits}})
\]

\( f \) shifts bits 3 to the left (it's more complicated).

1. Very fast since \( \oplus \) and concat and shift are fast.
2. Has same problem for crypto that LCG does: it's a recurrence. Can guess that a word or phrase is in the text.
Mersenne Twister

We do a very small example with a smaller word size than is used. The **Mersenne Twister** generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \{0, \ldots, 26\}).

We give an example:

Params: 7, 5, 3, 5, 3, \(x_0, \ldots, x_6\), unknown to Eve.

\[
x_{n+7} = x_{n+5} \oplus f(x_n^{\text{first 3 bits}}, x_{n+1}^{\text{last 5 bits}})
\]

\(f\) shifts bits 3 to the left (it's more complicated).

1. Very fast since \(\oplus\) and concat and shift are fast.
2. Has same problem for crypto that LCG does: it's a recurrence. Can guess that a word or phrase is in the text.
3. Would need to be a very long phrase so that the recurrence produces equations.
Mersenne Twister

We do a very small example with a smaller word size than is used. The **Mersenne Twister** generates a sequence of 10-bit numbers (two 5-bit numbers, so for us 2 numbers in \{0, \ldots, 26\}).

We give an example:

**Params:** 7, 5, 3, 5, 3, \(x_0, \ldots, x_6\), unknown to Eve.

\[x_{n+7} = x_{n+5} \oplus f(x_n^{\text{first 3 bits}}, x_{n+1}^{\text{last 5 bits}})\]

\(f\) shifts bits 3 to the left (its more complicated).

1. Very fast since \(\oplus\) and concat and shift are fast.
2. Has same problem for crypto that LCG does: its a recurrence. Can guess that a word or phrase is in the text.
3. Would need to be a very long phrase so that the recurrence produces equations.
4. The larger the parameter which we have as 7, the longer the phrase has to be.
**Mersenne Twister Example with Digits**

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
<th>B</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
<td>15</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
<td>96</td>
<td>06</td>
<td>11</td>
</tr>
<tr>
<td>Key</td>
<td>18</td>
<td>65</td>
<td>76</td>
<td>48</td>
<td>08</td>
<td>25</td>
<td>25</td>
<td>82</td>
<td>04</td>
<td>04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>R</th>
<th>D</th>
<th>E</th>
<th>R</th>
<th>S</th>
<th>I</th>
<th>N</th>
<th>D</th>
<th>I</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>18</td>
<td>04</td>
<td>05</td>
<td>18</td>
<td>19</td>
<td>09</td>
<td>14</td>
<td>04</td>
<td>09</td>
<td>01</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>23</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
</tr>
<tr>
<td>Key</td>
<td>95</td>
<td>12</td>
<td>04</td>
<td>03</td>
<td>90</td>
<td>10</td>
<td>16</td>
<td>07</td>
<td>15</td>
<td>09</td>
</tr>
</tbody>
</table>

Eve will guess the 7 and 5, does not know \( f, a, b \)

\[
x_{n+7} = x_{n+5} \oplus f(x_n \text{ first } a \text{ digs } x_{n+1} \text{ last } b \text{ digs })
\]
### Mersenne Twister Example with Digits

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
<th>B</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
<td>15</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
<td>96</td>
<td>06</td>
<td>11</td>
</tr>
<tr>
<td>Key</td>
<td>18</td>
<td>65</td>
<td>76</td>
<td>48</td>
<td>08</td>
<td>25</td>
<td>25</td>
<td>82</td>
<td>04</td>
<td>04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text-Letter</th>
<th>R</th>
<th>D</th>
<th>E</th>
<th>R</th>
<th>S</th>
<th>I</th>
<th>N</th>
<th>D</th>
<th>I</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Digits</td>
<td>18</td>
<td>04</td>
<td>05</td>
<td>18</td>
<td>19</td>
<td>09</td>
<td>14</td>
<td>04</td>
<td>09</td>
<td>01</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>23</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
</tr>
<tr>
<td>Key</td>
<td>95</td>
<td>12</td>
<td>04</td>
<td>03</td>
<td>90</td>
<td>10</td>
<td>16</td>
<td>07</td>
<td>15</td>
<td>09</td>
</tr>
</tbody>
</table>

Eve will guess the 7 and 5, does not know \( f, a, b \)

\[
x_{n+7} = x_{n+5} \oplus f(x_n, x_{n+1})
\]

\[
1509 = 9010 \oplus f(0825, 2528)
\]
Eve will guess the 7 and 5, does not know $f, a, b$

\[ x_{n+7} = x_{n+5} \oplus f(x_n \text{ first a digs } x_{n+1} \text{ last b digs}) \]

\[
1509 = 9010 \oplus f(0825 \text{ first a digs } 2528 \text{ last b digs})
\]

\[
1607 = 0403 \oplus f(7648 \text{ first a digs } 4808 \text{ last b digs})
\]
Mersenne Twister Example with Digits

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>A</th>
<th>K</th>
<th>I</th>
<th>S</th>
<th>T</th>
<th>A</th>
<th>N</th>
<th>B</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Letter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-Digits</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
<td>15</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>24</td>
<td>66</td>
<td>87</td>
<td>47</td>
<td>17</td>
<td>45</td>
<td>26</td>
<td>96</td>
<td>06</td>
<td>11</td>
</tr>
<tr>
<td>Key</td>
<td>18</td>
<td>65</td>
<td>76</td>
<td>48</td>
<td>08</td>
<td>25</td>
<td>25</td>
<td>82</td>
<td>04</td>
<td>04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>D</th>
<th>E</th>
<th>R</th>
<th>S</th>
<th>I</th>
<th>N</th>
<th>D</th>
<th>I</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-Letter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-Digits</td>
<td>18</td>
<td>04</td>
<td>05</td>
<td>18</td>
<td>19</td>
<td>09</td>
<td>14</td>
<td>04</td>
<td>09</td>
<td>01</td>
</tr>
<tr>
<td>Cipher-text</td>
<td>23</td>
<td>16</td>
<td>01</td>
<td>11</td>
<td>09</td>
<td>19</td>
<td>20</td>
<td>01</td>
<td>14</td>
<td>02</td>
</tr>
<tr>
<td>Key</td>
<td>95</td>
<td>12</td>
<td>04</td>
<td>03</td>
<td>90</td>
<td>10</td>
<td>16</td>
<td>07</td>
<td>15</td>
<td>09</td>
</tr>
</tbody>
</table>

Eve will guess the 7 and 5, does not know \( f, a, b \)

\[
x_{n+7} = x_{n+5} \oplus f(x_n \text{ first a digs}, x_{n+1} \text{ last b digs})
\]

\[
1509 = 9010 \oplus f(0825 \text{ first a digs}, 2528 \text{ last b digs})
\]

\[
1607 = 0403 \oplus f(7648 \text{ first a digs}, 4808 \text{ last b digs})
\]

\[
9010 = 9512 \oplus f(1865 \text{ first a digs}, 6576 \text{ last b digs})
\]
Eve will guess the 7 and 5, does not know \(f, a, b\).

\[
x_{n+7} = x_{n+5} \oplus f(x_n \text{ first } a \text{ digs}, x_{n+1} \text{ last } b \text{ digs})
\]

\[
1509 = 9010 \oplus f(0825 \text{ first } a \text{ digs}, 2528 \text{ last } b \text{ digs})
\]

\[
1607 = 0403 \oplus f(7648 \text{ first } a \text{ digs}, 4808 \text{ last } b \text{ digs})
\]

\[
9010 = 9512 \oplus f(1865 \text{ first } a \text{ digs}, 6576 \text{ last } b \text{ digs})
\]

Can use recurrences to find \(f, a, b\).
Eve will guess the 7 and 5, does not know $f$, $a$, $b$

$$x_{n+7} = x_{n+5} \oplus f(x_n \text{ first a digs}, x_{n+1} \text{ last b digs})$$

$$1509 = 9010 \oplus f(0825 \text{ first a digs}, 2528 \text{ last b digs})$$
$$1607 = 0403 \oplus f(7648 \text{ first a digs}, 4808 \text{ last b digs})$$
$$9010 = 9512 \oplus f(1865 \text{ first a digs}, 6576 \text{ last b digs})$$

Can use recurrences to find $f$, $a$, $b$. Will need more equations and some guesswork, but crackable!
Any pseudo-random generator that is based on recurrences is crackable.