Byte-wise shift cipher

- Instead of $a, b, c, d, \ldots, z$ have (for example) 0000, 0001, \ldots, 1111.
- Works for an alphabet of bytes rather than (English, lowercase) letters
  - Data in a computer is stored this way anyway. So works natively for arbitrary data!

- Use XOR instead of modular addition. Fast!
- Decode and Encode are both XOR.
  - Essential properties still hold
# Hexadecimal (base 16)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bits (“nibble”)</th>
<th>Decimal</th>
<th>Hex</th>
<th>Bits (“nibble”)</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
<td>A</td>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
<td>B</td>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
<td>C</td>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
<td>D</td>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
<td>E</td>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
<td>F</td>
<td>1111</td>
<td>15</td>
</tr>
</tbody>
</table>
Hexadecimal (base 16)

Notation: 0x before a string of \{0, 1, \ldots, 9, A, B, C, D, E, F\} means that the string will be base 16.

- 0x10
  - 0x10 = 16*1 + 0 = 16
  - 0x10 = 0001 0000

- 0xAF
  - 0xAF = 16*A + F = 16*10 + 15 = 175
  - 0xAF = 1010 1111
ASCII

- Characters (often) represented in ASCII
  - 1 byte/char = 2 hex digits/char
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Char</th>
<th>Hex</th>
<th>Dec</th>
<th>Char</th>
<th>Hex</th>
<th>Dec</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
<td>NULL</td>
<td>0x20</td>
<td>32</td>
<td>Space</td>
<td>0x40</td>
<td>64</td>
<td>@</td>
</tr>
<tr>
<td>0x01</td>
<td>1</td>
<td>SOH</td>
<td>0x21</td>
<td>33</td>
<td>!</td>
<td>0x41</td>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>0x02</td>
<td>2</td>
<td>STX</td>
<td>0x22</td>
<td>34</td>
<td>&quot;</td>
<td>0x42</td>
<td>66</td>
<td>B</td>
</tr>
<tr>
<td>0x03</td>
<td>3</td>
<td>ETX</td>
<td>0x23</td>
<td>35</td>
<td>#</td>
<td>0x43</td>
<td>67</td>
<td>C</td>
</tr>
<tr>
<td>0x04</td>
<td>4</td>
<td>EOT</td>
<td>0x24</td>
<td>36</td>
<td>$</td>
<td>0x44</td>
<td>68</td>
<td>D</td>
</tr>
<tr>
<td>0x05</td>
<td>5</td>
<td>ENQ</td>
<td>0x25</td>
<td>37</td>
<td>%</td>
<td>0x45</td>
<td>69</td>
<td>E</td>
</tr>
<tr>
<td>0x06</td>
<td>6</td>
<td>ACK</td>
<td>0x26</td>
<td>38</td>
<td>&amp;</td>
<td>0x46</td>
<td>70</td>
<td>F</td>
</tr>
<tr>
<td>0x07</td>
<td>7</td>
<td>BELL</td>
<td>0x27</td>
<td>39</td>
<td>'</td>
<td>0x47</td>
<td>71</td>
<td>G</td>
</tr>
<tr>
<td>0x08</td>
<td>8</td>
<td>BS</td>
<td>0x28</td>
<td>40</td>
<td>(</td>
<td>0x48</td>
<td>72</td>
<td>H</td>
</tr>
<tr>
<td>0x09</td>
<td>9</td>
<td>TAB</td>
<td>0x29</td>
<td>41</td>
<td>)</td>
<td>0x49</td>
<td>73</td>
<td>I</td>
</tr>
<tr>
<td>0x0A</td>
<td>10</td>
<td>LF</td>
<td>0x2A</td>
<td>42</td>
<td>*</td>
<td>0x4A</td>
<td>74</td>
<td>J</td>
</tr>
<tr>
<td>0x0B</td>
<td>11</td>
<td>VT</td>
<td>0x2B</td>
<td>43</td>
<td>+</td>
<td>0x4B</td>
<td>75</td>
<td>K</td>
</tr>
<tr>
<td>0x0C</td>
<td>12</td>
<td>FF</td>
<td>0x2C</td>
<td>44</td>
<td>,</td>
<td>0x4C</td>
<td>76</td>
<td>L</td>
</tr>
<tr>
<td>0x0D</td>
<td>13</td>
<td>CR</td>
<td>0x2D</td>
<td>45</td>
<td>-</td>
<td>0x4D</td>
<td>77</td>
<td>M</td>
</tr>
<tr>
<td>0x0E</td>
<td>14</td>
<td>SO</td>
<td>0x2E</td>
<td>46</td>
<td>.</td>
<td>0x4E</td>
<td>78</td>
<td>N</td>
</tr>
<tr>
<td>0x0F</td>
<td>15</td>
<td>SI</td>
<td>0x2F</td>
<td>47</td>
<td>/</td>
<td>0x4F</td>
<td>79</td>
<td>O</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>DLE</td>
<td>0x30</td>
<td>48</td>
<td>0</td>
<td>0x50</td>
<td>80</td>
<td>P</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>DC1</td>
<td>0x31</td>
<td>49</td>
<td>1</td>
<td>0x51</td>
<td>81</td>
<td>Q</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>DC2</td>
<td>0x32</td>
<td>50</td>
<td>2</td>
<td>0x52</td>
<td>82</td>
<td>R</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>DC3</td>
<td>0x33</td>
<td>51</td>
<td>3</td>
<td>0x53</td>
<td>83</td>
<td>S</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>DC4</td>
<td>0x34</td>
<td>52</td>
<td>4</td>
<td>0x54</td>
<td>84</td>
<td>T</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>NAK</td>
<td>0x35</td>
<td>53</td>
<td>5</td>
<td>0x55</td>
<td>85</td>
<td>U</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>SYN</td>
<td>0x36</td>
<td>54</td>
<td>6</td>
<td>0x56</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>ETB</td>
<td>0x37</td>
<td>55</td>
<td>7</td>
<td>0x57</td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>CAN</td>
<td>0x38</td>
<td>56</td>
<td>8</td>
<td>0x58</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>EM</td>
<td>0x39</td>
<td>57</td>
<td>9</td>
<td>0x59</td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>SUB</td>
<td>0x3A</td>
<td>58</td>
<td>:</td>
<td>0x5A</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>FSC</td>
<td>0x3B</td>
<td>59</td>
<td>;</td>
<td>0x5B</td>
<td>91</td>
<td>[</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>FS</td>
<td>0x3C</td>
<td>60</td>
<td>&lt;</td>
<td>0x5C</td>
<td>92</td>
<td>\</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>GS</td>
<td>0x3D</td>
<td>61</td>
<td>=</td>
<td>0x5D</td>
<td>93</td>
<td>]</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>RS</td>
<td>0x3E</td>
<td>62</td>
<td>&gt;</td>
<td>0x5E</td>
<td>94</td>
<td>^</td>
</tr>
<tr>
<td>0x1F</td>
<td>31</td>
<td>US</td>
<td>0x3F</td>
<td>63</td>
<td>?</td>
<td>0x5F</td>
<td>95</td>
<td>_</td>
</tr>
</tbody>
</table>

ASCII

- ‘1’ = 0x31 = 0011 0001

- ‘F’ = 0x46 = 0100 0110

- Note that writing 0x00 to a file is different from writing “0x00” to a file
  - 0x00 = 0000 0000 (1 byte)
  - “0x00” = 0x30 78 30 30
    = 0011 0000 0111 1000... (4 bytes)
Useful observations

- Only 128 valid ASCII chars (128 bytes invalid)
- 0x20-0x7E printable
- 0x41-0x7A includes upper/lowercase letters
  - Uppercase letters begin with 0x4 or 0x5
  - Lowercase letters begin with 0x6 or 0x7
Byte-wise shift cipher

- $\mathcal{M} = \{\text{strings of bytes}\}$

- $Gen$: choose uniform byte $k \in \mathcal{K} = \{0, \ldots, 255\}$

- $Enc_k(m_1 \ldots m_t)$: output $c_1 \ldots c_t$, where $c_i := m_i \oplus k$

- $Dec_k(c_1 \ldots c_t)$: output $m_1 \ldots m_t$, where $m_i := c_i \oplus k$

- Verify that correctness holds...
Is this cipher secure?

- No – only 256 possible keys!
  - Given a ciphertext, try decrypting with every possible key
  - If ciphertext is long enough, only one plaintext will “make sense”

- Can further optimize
  - First nibble of plaintext likely 0x4, 0x5, 0x6, 0x7 (assuming letters only)
  - Can reduce exhaustive search to 26 keys (how?)
  - Talk to your friends or blood enemies about this.
The key space must be large enough to make exhaustive-search attacks impractical.

How large do you think that is?

Note: this makes some assumptions...

- English-language plaintext
- Ciphertext sufficiently long so only one valid plaintext
The Vigenère cipher

- Shift the key was \( k \in \{a, \ldots, z\} = \{0, \ldots, 25\} \).
  Vig the key is \( k \in \{a, \ldots, z\}^* = \{0, \ldots, 25\}^* \)
When used \( k \) was a phrase like
\( Jacob \ Prinz \ is \ a \ Physics \ Major, \)
easy to remember and transmit. All arithmetic is mod 26.

- Let \( k = (k_1, k_2, \ldots, k_n) \).
  To encrypt \( \text{Enc}(m_1, m_2, \ldots, m_N) = \)
  \[
  m_1 + k_1, m_2 + k_2, \ldots, m_n + k_n,
  \]
  \[
  m_{n+1} + k_1, m_{n+2} + k_2, \ldots, m_{n+n} + k_n,
  \]
  \[
  \vdots
  \]
- Decryption just reverse the process
The Vigenère cipher

- **Size of key space?**
  - If keys are 14-character strings over the English alphabet, then key space has size $26^{14} \approx 2^{66}$
  - If variable length keys, even more.
  - If only 14-letter phrases are used, then less.
  - Brute-force search infeasible

- **Is the Vigenère cipher secure?**
  - Believed secure for many years...
  - Might not have even been secure then...
Attacking the Vigenère cipher

- (Assume a 14-character key)

- Observation: every 14\textsuperscript{th} character is “encrypted” using the same shift:
  veqpj iredo zxoeu alpcm sdjq uiqn dnoss oscdc usoak jqmxp qrhyy cjquoq qodhj cciow ieii
Using plaintext letter frequencies
Attacking the Vigenère cipher

- Look at every 14\textsuperscript{th} character of the ciphertext, starting with the first
  - Call this a “stream”
- Let $\alpha$ be the most common character appearing in this stream
- Most likely, this character corresponds to the most common plaintext character (‘e’)
  - Guess that the first character of the key is $\alpha$ - ‘e’
- Repeat for all other positions
- This is somewhat haphazard
A better attack

- Let $p_i$ ($0 \leq i \leq 25$) denote the frequency of the $i^{th}$ English letter in general text
  - One can compute that $\sum_i p_i^2 \approx 0.065$

- Let $q_i$ denote the observed frequency of the $i^{th}$ letter in a given stream of the ciphertext

- If the shift for a stream is $j$, expect $q_{i+j} = p_i \forall i$
  - So expect $\sum_i p_i q_{i+j} \approx 0.065$

- Test for every value of $j$ to find the right one
  - Repeat for each stream
Finding the key length

- When using the correct key length, the ciphertext frequencies \( \{q_i\} \) of a stream will be shifted versions of the \( \{p_i\} \)
  - So \( \sum q_i^2 = \sum (\frac{1}{26})^2 = \frac{1}{26} \approx 0.065 \)

- When using an incorrect key length, expect (heuristically) that the \( \{q_i\} \) are equal
  - So \( \sum q_i^2 = \sum (\frac{1}{26})^2 = 0.038 \)

- In face, good enough to find the key length \( N \) that maximizes \( \sum q_i^2 \)
  - Can check with other streams . . .
 Byte-wise Vigenère cipher

- The key is a string of bytes
- The plaintext is a string of bytes
- To encrypt, XOR each character in the plaintext with the next character of the key
  - Wrap around in the key as needed
- Decryption just reverses the process
Example

- Say plaintext is “Hello!” and key is 0xA1 2F
- “Hello!” = 0x48 65 6C 6C 6F 21
- XOR with 0xA1 2F A1 2F A1 2F
- 0x48 ⊕ 0xA1
  - 0100 1000 ⊕ 1110 1001 = 0xE9
- Ciphertext: 0xE9 4A CD 43 CE 0E
Attacking the (variant) Vigenère cipher

- Two steps
  - Determine the key length
  - Determine each byte of the key
- Same principles as before . . .
Determining the key length

- Let $p_i$ (for $0 \leq i \leq 255$) be the frequency of byte $i$ in general English text
  - i.e. $p_i = 0$ for $i < 32$ or $i > 127$
  - i.e. $p_{97} = \text{frequency of ‘a’}$
  - the distribution is far from uniform

- If the key length is $N$, then every $N^{th}$ character of the plaintext is encrypted using the same “shift”
  - If we take every $N^{th}$ character and calculate frequencies, we should get the $p_i$’s in permuted order
  - If we take every $M^{th}$ character ($M$ not a multiple of $N$) and calculate frequencies, we should get something close to uniform
Determining the key length

- How to distinguish these two?

- For some candidate key length, tabulate $q_0, \ldots, q_{255}$ and compute $\sum q_i^2$
  - If close to uniform, $\sum q_i^2 \approx 256 \left( \frac{1}{256} \right)^2 = \frac{1}{256}$
  - If a permutation of $p_i$, then $\sum q_i^2 \approx \sum p_i^2$
    - Could compute $\sum p_i^2$ (but somewhat difficult)
    - Key point: it will be much larger than $\frac{1}{256}$

- Compute $\sum q_i^2$ for each possible key length, and look for maximum value
  - Correct key length should yield a large value for every stream
Determining the \( i^{th} \) byte of the key

- Assume the key length \( N \) is known

- Look at every \( N^{th} \) character of the ciphertext, starting with the \( i^{th} \) character
  - Call this the \( i^{th} \) ciphertext ”stream”
  - Note that all bytes in this stream were generated by XORing plaintext with the same byte of the key

- Try decrypting the stream using every possible byte value \( B \)
  - Get a candidate plaintext stream for each value
Determining the $i^{th}$ byte of the key

- Could use $\{p_i\}$ as before, but not as easy to find

- When the guess $B$ is correct:
  - All bytes in the plaintext stream will be between 32 and 127
  - Frequencies of lowercase letters (as a fraction of all lowercase letters) should be close to known English-letter frequencies
    - Tabulate observed letter frequencies $q'_0, \ldots, q'_{25}$ (as fraction of all lowercase letters)
    - Should find $\sum q'_ip'_i \approx \sum (p'_i)^2$, where $p'_i$ corresponds to English-letter frequencies
    - In practice, take $B$ that maximizes $\sum q'_ip'_i$, subject to caveat above (and possibly others)
Attack time?

- Say the key length is between 1 and $L$
- Determining the key length: $\approx 256L$
- Determining all bytes of the key: $< 256^2L$
- Brute-force key search: $\approx 256^L$
The attack in practice

- Attack is more reliable as the ciphertext length grows larger.

- Attack still works for short(er) ciphertexts, but more “tweaking” and manual involvement can be needed.
First programming assignment

- Decrypt ciphertext (provided online) that was generated using the Vigenère cipher