Who am I?

Arnis Paršovs

MSc in Cyber Security
Tallinn University of Technology, 2012

Computer Science PhD student at UT
Who are you?
Crypto courses in UT

- Cryptology I
- Cryptographic protocols
- Cryptology II
- Research seminar in cryptography
- Quantum cryptography
- Topics of mathematics in cryptology
- Applied cryptography
This course

- Practical course
- Learn by implementing
- No proofs – just intuition
Course timeline

16 weeks

- Lecture: Fri 12:15 – 14:00 (room 203)
- Practice: Fri 14:15 – 16:00 (room 402)

6 ECTS – 10 hours weekly

- 2 hours in class
- 8 hours on homework (may vary)
Grading

- Homework every week
- Homeworks give maximum 70% of final grade
  - Bonus points help to achieve the maximum
- Deadlines are strict!
  - May be extended if you have a good excuse
  - Late submission gets 50% of grade
  - Homeworks submitted later than 1 week after the deadline are not accepted
- Exam gives another 30% of final grade
  - Should be easy if you follow the lectures
Programming Environment

Course history:

- 2010: C, C++, Java
- 2011: Java
- 2012: Java
- 2013: Python
- 2014: Python
- 2015: Python
- 2016: Python
- 2017: Python

Reason: focus on cryptography, not language

Test environment: Ubuntu 16.04, Python 2.7.x
Homework submission

• Create a private Bitbucket repository and grant me ‘read’ privileges:
  https://bitbucket.org/appcrypto/2017/src/master/setup/
  • Send me an e-mail if your identity is not obvious from your Bitbucket account name.

• Homework templates will be published at course repository: https://bitbucket.org/appcrypto/2017/

• Homework deadline – beginning of next lecture
  • Feedback will be given using Bitbucket’s code comment feature

• Do not look on homework solutions of others!
  • Plagiarism cases will be handled in accordance with UT Plagiarism Policy
Python Cheat Sheet


```python
>>> "abraca" + 'dabra'
'abracadabra'
>>> "abraca" + "\x31\x05\x77"
'abraca1\x05w'
>>> for character in "foo":
...    print "char=%s" % (character)
char=f
char=o
char=o
>>> "abraca"[2:5]
'rac'
>>> "abracadabra".encode('hex')
'6162726163616461627261'
>>> "abracadabra".encode('base64')
'YWJyYWNhZGFicmE=
>>> "abracadabra".encode('base64').decode('base64')
'abracadabra'
```
Randomness

- Why do we need randomness in real life?
- Why do we need randomness in crypto?
  - For keys, nonces, etc.
- What is random sequence?
  - Sequence of numbers that does not follow any deterministic pattern
  - None of the numbers can be predicted based on previous numbers
  - Has no description shorter than itself
  - Sequence of bits that cannot be compressed
- Where we can get random numbers?
  - Can we flip a coin to get a random number?
  - Can a computer program generate random numbers?
  - Thermal noise, photoelectric effect, quantum phenomena
Pseudo-Random Number Generator (PRNG)

Deterministic algorithm that produces endless stream of numbers which are indistinguishable from truly random. PRNG is initialized using (hopefully) random 'seed' value.

Linux /dev/urandom implementation:

In Python: `os.urandom(n)`
Randomness

- Can we tell whether the sequence is random?
  
  \[\ldots 41592653589\ldots\]
  
  \[3.141592653589793\ldots\]
  
  \[\ldots 000000\ldots\]

- Randomness tests
Bits and Bytes

Bit string:
100010000011

$2^{11} + 2^7 + 2^1 + 2^0$

Most significant bit (msb) – left-most bit

Bytes - 8-bit collections (0-255)

Byte - basic addressable element
<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;NULL&gt;</td>
<td>32</td>
<td>&lt;SPACE&gt;</td>
<td>64</td>
<td>@</td>
<td>96</td>
<td>`</td>
<td>128</td>
<td>Å</td>
<td>160</td>
<td>†</td>
</tr>
<tr>
<td>1</td>
<td>&lt;SOH&gt;</td>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>97</td>
<td>a</td>
<td>129</td>
<td>Å</td>
<td>161</td>
<td>°</td>
</tr>
<tr>
<td>2</td>
<td>&lt;STX&gt;</td>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>98</td>
<td>b</td>
<td>130</td>
<td>Ç</td>
<td>162</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>&lt;ETX&gt;</td>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>99</td>
<td>c</td>
<td>131</td>
<td>É</td>
<td>163</td>
<td>£</td>
</tr>
<tr>
<td>4</td>
<td>&lt;EOT&gt;</td>
<td>36</td>
<td>$</td>
<td>68</td>
<td>D</td>
<td>100</td>
<td>d</td>
<td>132</td>
<td>Ñ</td>
<td>164</td>
<td>§</td>
</tr>
<tr>
<td>5</td>
<td>&lt;ENQ&gt;</td>
<td>37</td>
<td>%</td>
<td>69</td>
<td>E</td>
<td>101</td>
<td>e</td>
<td>133</td>
<td>Ø</td>
<td>165</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>&lt;ACK&gt;</td>
<td>38</td>
<td>&amp;</td>
<td>70</td>
<td>F</td>
<td>102</td>
<td>f</td>
<td>134</td>
<td>Ü</td>
<td>166</td>
<td>¶</td>
</tr>
<tr>
<td>7</td>
<td>&lt;BEL&gt;</td>
<td>39</td>
<td>'</td>
<td>71</td>
<td>G</td>
<td>103</td>
<td>g</td>
<td>135</td>
<td>à</td>
<td>167</td>
<td>ß</td>
</tr>
<tr>
<td>8</td>
<td>&lt;BS&gt;</td>
<td>40</td>
<td>(</td>
<td>72</td>
<td>H</td>
<td>104</td>
<td>h</td>
<td>136</td>
<td>â</td>
<td>168</td>
<td>®</td>
</tr>
<tr>
<td>9</td>
<td>&lt;TAB&gt;</td>
<td>41</td>
<td>)</td>
<td>73</td>
<td>I</td>
<td>105</td>
<td>i</td>
<td>137</td>
<td>ä</td>
<td>169</td>
<td>©</td>
</tr>
<tr>
<td>10</td>
<td>&lt;LF&gt;</td>
<td>42</td>
<td>*</td>
<td>74</td>
<td>J</td>
<td>106</td>
<td>j</td>
<td>138</td>
<td>å</td>
<td>170</td>
<td>™</td>
</tr>
<tr>
<td>11</td>
<td>&lt;VT&gt;</td>
<td>43</td>
<td>+</td>
<td>75</td>
<td>K</td>
<td>107</td>
<td>k</td>
<td>139</td>
<td>ä</td>
<td>171</td>
<td>'</td>
</tr>
<tr>
<td>12</td>
<td>&lt;FF&gt;</td>
<td>44</td>
<td>,</td>
<td>76</td>
<td>L</td>
<td>108</td>
<td>l</td>
<td>140</td>
<td>å</td>
<td>172</td>
<td>”</td>
</tr>
<tr>
<td>13</td>
<td>&lt;CR&gt;</td>
<td>45</td>
<td>-</td>
<td>77</td>
<td>M</td>
<td>109</td>
<td>m</td>
<td>141</td>
<td>ç</td>
<td>173</td>
<td>±</td>
</tr>
<tr>
<td>14</td>
<td>&lt;SO&gt;</td>
<td>46</td>
<td>.</td>
<td>78</td>
<td>N</td>
<td>110</td>
<td>n</td>
<td>142</td>
<td>é</td>
<td>174</td>
<td>Æ</td>
</tr>
<tr>
<td>15</td>
<td>&lt;SI&gt;</td>
<td>47</td>
<td>/</td>
<td>79</td>
<td>O</td>
<td>111</td>
<td>o</td>
<td>143</td>
<td>è</td>
<td>175</td>
<td>Ø</td>
</tr>
<tr>
<td>16</td>
<td>&lt;DLE&gt;</td>
<td>48</td>
<td>0</td>
<td>80</td>
<td>P</td>
<td>112</td>
<td>p</td>
<td>144</td>
<td>é</td>
<td>176</td>
<td>∞</td>
</tr>
<tr>
<td>17</td>
<td>&lt;DC1&gt;</td>
<td>49</td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td>113</td>
<td>q</td>
<td>145</td>
<td>è</td>
<td>177</td>
<td>±</td>
</tr>
<tr>
<td>18</td>
<td>&lt;DC2&gt;</td>
<td>50</td>
<td>2</td>
<td>82</td>
<td>R</td>
<td>114</td>
<td>r</td>
<td>146</td>
<td>í</td>
<td>178</td>
<td>±</td>
</tr>
<tr>
<td>19</td>
<td>&lt;DC3&gt;</td>
<td>51</td>
<td>3</td>
<td>83</td>
<td>S</td>
<td>115</td>
<td>s</td>
<td>147</td>
<td>í</td>
<td>179</td>
<td>±</td>
</tr>
<tr>
<td>20</td>
<td>&lt;DC4&gt;</td>
<td>52</td>
<td>4</td>
<td>84</td>
<td>T</td>
<td>116</td>
<td>t</td>
<td>148</td>
<td>î</td>
<td>180</td>
<td>ß</td>
</tr>
<tr>
<td>21</td>
<td>&lt;NAK&gt;</td>
<td>53</td>
<td>5</td>
<td>85</td>
<td>U</td>
<td>117</td>
<td>u</td>
<td>149</td>
<td>î</td>
<td>181</td>
<td>µ</td>
</tr>
<tr>
<td>22</td>
<td>&lt;SYN&gt;</td>
<td>54</td>
<td>6</td>
<td>86</td>
<td>V</td>
<td>118</td>
<td>v</td>
<td>150</td>
<td>ŋ</td>
<td>182</td>
<td>ă</td>
</tr>
<tr>
<td>23</td>
<td>&lt;ETB&gt;</td>
<td>55</td>
<td>7</td>
<td>87</td>
<td>W</td>
<td>119</td>
<td>w</td>
<td>151</td>
<td>õ</td>
<td>183</td>
<td>ß</td>
</tr>
<tr>
<td>24</td>
<td>&lt;CAN&gt;</td>
<td>56</td>
<td>8</td>
<td>88</td>
<td>X</td>
<td>120</td>
<td>x</td>
<td>152</td>
<td>õ</td>
<td>184</td>
<td>ß</td>
</tr>
<tr>
<td>25</td>
<td>&lt;EM&gt;</td>
<td>57</td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td>121</td>
<td>y</td>
<td>153</td>
<td>Œ</td>
<td>185</td>
<td>£</td>
</tr>
<tr>
<td>26</td>
<td>&lt;SUB&gt;</td>
<td>58</td>
<td>:</td>
<td>90</td>
<td>Z</td>
<td>122</td>
<td>z</td>
<td>154</td>
<td>Ö</td>
<td>186</td>
<td>¶</td>
</tr>
<tr>
<td>27</td>
<td>&lt;ESC&gt;</td>
<td>59</td>
<td>;</td>
<td>91</td>
<td>[</td>
<td>123</td>
<td>{</td>
<td>155</td>
<td>Ð</td>
<td>187</td>
<td>£</td>
</tr>
<tr>
<td>28</td>
<td>&lt;FS&gt;</td>
<td>60</td>
<td>&lt;</td>
<td>92</td>
<td>\</td>
<td>124</td>
<td></td>
<td></td>
<td>156</td>
<td>ð</td>
<td>188</td>
</tr>
<tr>
<td>29</td>
<td>&lt;GS&gt;</td>
<td>61</td>
<td>=</td>
<td>93</td>
<td>]</td>
<td>125</td>
<td>}</td>
<td>157</td>
<td>Ù</td>
<td>189</td>
<td>Ω</td>
</tr>
<tr>
<td>30</td>
<td>&lt;RS&gt;</td>
<td>62</td>
<td>&gt;</td>
<td>94</td>
<td>^</td>
<td>126</td>
<td>~</td>
<td>158</td>
<td>Ù</td>
<td>190</td>
<td>Æ</td>
</tr>
<tr>
<td>31</td>
<td>&lt;US&gt;</td>
<td>63</td>
<td>?</td>
<td>95</td>
<td>_</td>
<td>127</td>
<td>&lt;DEL&gt;</td>
<td>159</td>
<td>Ù</td>
<td>191</td>
<td>Ø</td>
</tr>
</tbody>
</table>

http://www.asciitable.com/
## Hexadecimal (Base16) Encoding

<table>
<thead>
<tr>
<th>Hex</th>
<th>Value</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0'</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>'1'</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>'2'</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>'3'</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>'4'</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>'5'</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>'6'</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>'7'</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>'8'</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>'9'</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>'A'</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>'B'</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>'C'</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>'D'</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>'E'</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>'F'</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

- One hex symbol represents 4 bits
- Two hex symbols needed to represent a byte

\[ 2E = 0010 \ 1110 \]

```
>>> 46
46
>>> 0x2e
46
>>> 0x2e + 1
47
```
Python

Python’s `str` data type can store any byte:

```
>>> s = 'Abc\x00\x61'
>>> len(s)
5
>>> s[0], s[1], s[2], s[3], s[4]
('A', 'b', 'c', '\x00', 'a')
>>> ord(s[0])
65
>>> chr(65)
'A'
```

- `ord()` can be used to convert byte to integer
- `chr()` can be used to convert integer to byte
Base64 encoding

- Can represent binary data using printable characters
- Base64 encoded data approximately 33% larger
Bitwise operations

AND:
- extract partion of bit string
  -操作前的位串：0 0 1 1 1 1 0 0
  -位掩码：0 0 0 0 0 1 1 0
  -按位与运算：0 0 0 0 0 1 0 0 (AND)

OR:
- set specific bits
  -操作前的位串：0 0 1 1 1 1 0 0
  -按位或运算：0 0 1 1 1 1 1 0 (OR)

XOR:
- flip specific bits
  -操作前的位串：0 0 1 1 1 1 0 0
  -按位异或运算：0 0 1 1 1 0 1 0 (XOR)

Shift:
- shift and pad with 0
  -操作前的位串：0 0 1 1 1 1 0 0
  -右移两位：0 0 0 0 1 1 1 1 (right shift by two)
Bitwise operations: AND

- Extract bits we are interested in

Example:

\[
\begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
\ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H \\
\ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H & \ H \\
\end{array}
\]

---

\[
\begin{array}{ccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Python:

```python
>>> 60 & 6
4
```
Bitwise operations: OR

- Set specific bits

Example:

0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0
---------------
0 0 1 1 1 1 1 0 (OR)

Python:

```python
>>> 60 | 6
62
```
Bitwise operations: XOR

- Flip specific bits

Example:

0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0
---------------
0 0 1 1 1 0 1 0 (XOR)

Python:

>>> 60 ^ 6
58
Bitwise operations: Shift

- Shift (right or left) and pad with zeros

Example:

0 0 1 1 1 1 0 0
---------------
0 0 0 0 1 1 1 1 (right shift by two)

Python:

```python
>>> 60 >> 2
15
>>> 15 << 1
30
```

- Fast multiplication and division by 2
Python: Bytes to Integer

```python
>>> s = 'abC'
>>> i = ord(s[0])
>>> i
97
>>> bin(i)
'0b1100001'
>>> i = i << 8
>>> bin(i)
'0b1100001000000000'
>>> i = i | ord(s[1])
>>> bin(i)
'0b110000101100010'
>>> i = i << 8
>>> bin(i)
'0b110000101100010000000000'
>>> i = i | ord(s[2])
>>> bin(i)
'0b11000010110001001000011'
>>> i
6382147
```

- Convert first byte to integer
- Left-shift integer 8 times
- Convert second byte to integer
- Load second integer in first 8 bits
- ...
One-Time Pad (OTP)

- Key generation: key (one-time pad) is random sequence the same length as plaintext
- Encryption operation: XOR ($\oplus$) the key with plaintext
- Decryption operation: XOR ($\oplus$) the key with ciphertext
One-Time Pad (OTP)

Information-theoretically secure, if:

- Key (one-time pad) is truly random
- Key has the same length as plaintext
- Key is never reused

- Not used in practice
Stream Cipher

- Key generation: key is a 'seed' for PRNG
- Encryption operation: XOR ($\oplus$) the key with plaintext
- Decryption operation: XOR ($\oplus$) the key with ciphertext

Why is it less secure than one-time pad?
- Malleability attacks
- The same keystream must never be reused!

plaintext_1 \oplus \text{keystream} = \text{ciphertext}_1
plaintext_2 \oplus \text{keystream} = \text{ciphertext}_2 \oplus \text{plaintext}_2 = \text{keystream}
\text{keystream} \oplus \text{ciphertext}_1 = \text{plaintext}_1
Stream Cipher

Solution – on every encryption add unique \textit{nonce} to the key:

- The same \textit{nonce} must never be reused!
- How to generate \textit{nonce}?
  - Counter value
  - Random value
  - Current time
Questions

- Where we can get (true) random numbers?
- Why pseudo-random number is not as good as random number?
- What are the properties of random sequence?
- Can we tell whether the provided sequence is random?
- What happens to data if we XOR it with random data?
- Why brute-force attacks are ineffective in breaking one-time pad?
- Why unbreakable one-time pad is not used in enterprise products?
- How is stream cipher different from one-time pad?
Task: One-Time Pad (OTP)

Implement One-Time Pad cryptosystem.

Encryption should produce a random key file and encrypted output file:

$ chmod +x otp.py
$ ./otp.py encrypt datafile datafile.key datafile.encrypted

Decryption should use the key file and produce decrypted original plaintext file:

$ ./otp.py decrypt datafile.encrypted datafile.key datafile.plain

- Commit “otp.py” to your repository
#!/usr/bin/env python
import os, sys  # do not use any other imports/libraries
# specify here how much time your solution required

def encrypt(pfile, kfile, cfile):
    # your implementation here
    pass

def decrypt(cfile, kfile, pfile):
    # your implementation here
    pass

def usage():
    print "Usage:"
    print "encrypt <plaintext file> <output key file> <ciphertext output file>"
    print "decrypt <ciphertext file> <key file> <plaintext output file>"
    sys.exit(1)

if len(sys.argv) != 5:
    usage()
elif sys.argv[1] == 'encrypt':
    encrypt(sys.argv[2], sys.argv[3], sys.argv[4])
elif sys.argv[1] == 'decrypt':
    decrypt(sys.argv[2], sys.argv[3], sys.argv[4])
else:
    usage()
Task: One-Time Pad (OTP)

- **Encrypter:**
  - Read the file contents into byte string (e.g., `s = open('file.txt').read()`)
  - Convert plaintext byte string to one big integer
  - Obtain random byte string the same length as plaintext (use `os.urandom()`)
  - Convert random byte string to one big integer
  - XOR plaintext integer and key integer (**please, use this approach**)
  - Save the key (one-time pad) and XOR’ed result (ciphertext) to file:
    - Convert key integer to byte string:
      - Use bit masking and left shift
    - Once more: use bitwise operations!
      - Banned functions: `bin()`, `str()`, `int()`, `bytearray()` and `operator **`!

- **Decrypter:**
  - Perform the operations in reverse order
Task: Test Case

$ echo -n -e "\x85\xce\xa2\xa2" > file.enc
$ hexdump -C file.enc
00000000  85  ce  a2  25   |...%|
$ echo -n -e "\xe4\xac\xe1\x2f" > file.key
$ hexdump -C file.key
00000000  e4  ac  e1  2f   |.../|
$ ./otp.py decrypt file.enc file.key file.plain
$ hexdump -C file.plain
00000000  61  62  43  0a   |abC.|

$ echo -n -e "\x00\x00\x61\x62\x43\x00" > file.plain
$ hexdump -C file.plain
00000000  00  00  61  62  43  00   |..abC.|
$ ./otp.py encrypt file.plain file.key file.enc
$ ./otp.py decrypt file.enc file.key fileorig.plain
$ hexdump -C fileorig.plain
00000000  00  00  61  62  43  00   |..abC.|
Please!

• Do not waste your time on input validation
• Do not use imports/libraries that are not explicitly allowed
• Include information of how much time the tasks took (as a comment at the top of your source code)
• Give a feedback about parts that were hard to grasp or implement
• Make a note if something was wrong on slides or you have an idea for improvement
• Do not leave any unrequired debugging output in your solution
• Commit the solution to the main branch of your repository with the filename required

Thank you!