Who am I?

Arnis Paršovs

MSc in Cyber Security
Tallinn University of Technology, 2012

Computer Science PhD student at UT
Who are you?

- BSc − 2
- MSc (Computer Science) − 5
- MSc (Cyber Security) − 9
- MSc (Software Engineering) − 8
- PhD − 5
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: Wed 14:15 – 16:00
- Practice: Thu 12:15 – 14:00
  - Vote to swap the times: http://doodle.com/foxque6yyi44feria

6 ECTS – 10 hours weekly

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

• Homework every week

• Homeworks give 70% of final grade

• 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

Reason: focus on cryptography, not language

Test environment: Ubuntu 14.04, Python 2.7.x
Homework submission

• Create a private Bitbucket repository and grant me ‘read’ privileges:
  https://bitbucket.org/appcrypto/2015/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/2015/

• Homework deadline – beginning of next lecture
  • I might check your solution and give a feedback before deadline
  • Feedback will be given using Bitbucket’s code comment feature

• Do not look on other’s homework solutions!
  • 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 crypto?
  - For keys, nonces, etc.

- What is random sequence?
  - Sequence of numbers that do 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 can we get random numbers?
  - Can we flip a coin to get 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
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
<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;NULL&gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt;SOH&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;STX&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;ETX&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;EOT&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;ENQ&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;ACK&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;BEL&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&lt;BS&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;TAB&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;LF&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;VT&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;FF&gt;</td>
</tr>
<tr>
<td>13</td>
<td>&lt;CR&gt;</td>
</tr>
<tr>
<td>14</td>
<td>&lt;SO&gt;</td>
</tr>
<tr>
<td>15</td>
<td>&lt;SI&gt;</td>
</tr>
<tr>
<td>16</td>
<td>&lt;DLE&gt;</td>
</tr>
<tr>
<td>17</td>
<td>&lt;DC1&gt;</td>
</tr>
<tr>
<td>18</td>
<td>&lt;DC2&gt;</td>
</tr>
<tr>
<td>19</td>
<td>&lt;DC3&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;DC4&gt;</td>
</tr>
<tr>
<td>21</td>
<td>&lt;NAK&gt;</td>
</tr>
<tr>
<td>22</td>
<td>&lt;SYN&gt;</td>
</tr>
<tr>
<td>23</td>
<td>&lt;ETB&gt;</td>
</tr>
<tr>
<td>24</td>
<td>&lt;CAN&gt;</td>
</tr>
<tr>
<td>25</td>
<td>&lt;EM&gt;</td>
</tr>
<tr>
<td>26</td>
<td>&lt;SUB&gt;</td>
</tr>
<tr>
<td>27</td>
<td>&lt;ESC&gt;</td>
</tr>
<tr>
<td>28</td>
<td>&lt;FS&gt;</td>
</tr>
<tr>
<td>29</td>
<td>&lt;GS&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;RS&gt;</td>
</tr>
<tr>
<td>31</td>
<td>&lt;US&gt;</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 can represent 4 bits
- Two hex symbols needed to represent a byte

\[
2E = 0010\ 1110
\]

```
>>> 46
46
```

```
>>> 2e
SyntaxError: invalid token
```

```
>>> 0x2e
46
```

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

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

```python
>>> 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

bn+ITbj/TRwcSAwT8CZnFZN0me5/AGdFIKNLBPPo7Nc07T6XTpsTw0QxnM++9xJXKkEEcaEn2Vo9MiAVPVUR5PsFGKZbL7coPRdHD058RokCF4aizWv6+Dqg0lsXsmXliWusn0Q==

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

AND:
- extract partition of bit string

```
0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0 (bit mask) >>> 60 & 6
------------------
0 0 0 0 0 1 0 0 (AND)
```

OR:
- set specific bits

```
0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0 >>> 60 | 6
------------------
0 0 1 1 1 1 1 0 (OR)
```

XOR:
- flip specific bits

```
0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0 >>> 60 ^ 6
------------------
0 0 1 1 1 0 1 0 (XOR)
```

Shift:
- shift and pad with 0

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

```
Bitwise operations: AND

- Extract bits we are interested in

Example:

```
0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0 (bit mask)
---------------
0 0 0 0 0 1 0 0 (AND)
```

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 1 0 (OR)
```

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:

```python
>>> 60 ^ 6
58
```
Bitwise operations: Shift

- Shift (right or left) and pad with zeros

Example:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(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
- ...

### Endianness (byte order):

- Big Endian (network byte order)
  - MSB left-most byte
- Big Endian order used in most standards
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
Stream Cipher

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

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

plaintext1 ⊕ keystream = ciphertext1
plaintext2 ⊕ keystream = ciphertext2 ⊕ plaintext2 = keystream
keystream ⊕ ciphertext1 = plaintext1
Stream Cipher

Solution – on every encryption add unique nonce to the key:

- The same nonce must never be reused!
- How to generate nonce?
  - Counter value
  - Random value
  - Current time
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.plaintext

• Commit “otp.py” to your repository
Task: Template

#!/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! Functions `bin()`, `str()`, `int()` are banned!

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

```bash
$ echo -n -e "\x85\xce\xa2\xa2\x25" > 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.key file.enc fileorig.plain
$ hexdump -C fileorig.plain
00000000  00 00 61 62 43 00   |..abC.|
```
Please!

• 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 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

• Commit solution in the main branch with the filename required

Thank you!